Controlled Design of Human-Like Agents with Context-Guided Learning for Automated Video Game Playing

Dominik Gotojuch

Submitted in fulfilment of the requirements for the
Degree of Doctor of Philosophy

School of Computing Science
College of Science and Engineering
University of Glasgow

September 2025

Dedicated to Stefan Gotojuch.

Thanks for everything, Dad.

Abstract

The video game industry state-of-the-practice ad hoc behaviour authoring techniques produce transparent and highly controllable autonomous agent artificial intelligence (AI) representations, but show limitations in adaptive and human-like behaviour design. Machine learning (ML) methods can cope with such constraints, but the black-box nature, high training costs in terms of data volume and time, as well as incompatibility with iterative workflows, make ML models unsuitable for commercial game development. To address the shortcomings of both these approaches, we investigate a non-disruptive, modular design approach to integrating small-scale learning models featuring performance and execution guarantees, as well as embedded human designer intent, into behaviour tree (BT) architecture for autonomous video game agents.

We deployed the proposed design in the environment of a published, commercial video game *60 Seconds!*, which we instrumented for agent training and evaluation using an off-the-shelf game engine, BT and a learning library. After quantitative analysis of the mass-scale gameplay telemetry dataset of 8,244,111 trajectories from real game users, we clustered the player population with respect to estimated play skill, using a gameplay context-based score metric. Output agent models were then developed and trained in the game's environment by applying the design, guided by game context-relevant segmentation of logic and behaviour of the top play skill persona model, derived from the trajectory data of the 7% top-scoring player cluster.

The output agent's gameplay performance was benchmarked against that of a reference agent, and experimentally evaluated in a normalised game scenario against 18,947 human players. It was found to be valid in the context of the game environment, functional, and capable of pursuing gameplay objectives in unseen scenarios with competency. However, it was unable to outperform human players due to the suboptimal performance of its trained learning models. We determined that software stability issues of the learning library used, limited observation space, and egocentric data adversely affected agent training. While further work to improve the training process is necessary, the successful application of the context-guided agent design in a commercial video game environment confirmed its potential for industrial applications. By contributing the design, the mass-scale dataset, and the tools used in our research, we enable the context-guided agents to be deployed in alternative contexts.

Declaration

I hereby declare that all the work presented in this thesis is original and was carried out by the author, unless attributed in the Acknowledgements or Collaborations or otherwise explicitly stated.

Acknowledgements

I want to thank my supervisor, Roderick Murray-Smith, for his never-ending patience, support, and encouragement, which motivated me to see my research through. I am also very grateful for inspiring discussions and feedback on my thesis, courtesy of my second supervisor, John Williamson, and my examiners: Nicolas Pugeault, Leif Azzopardi, and Gethin Norman.

My research would not have been possible without the backing of Robot Gentleman. I want to thank Juliusz Zenkner for being on board with this project and all my far-fetched ideas. I would also like to thank John Catlow for supporting me on my academic journey from day one.

Finally, I cannot overstate how much I appreciate all the support I received from friends and loved ones, especially from Janka and Ernest, who gave me the push I needed to complete my thesis.

Collaborations

Research work documented in this thesis was carried out in collaboration with Robot Gentleman sp. z o.o. Robot Gentleman provided access to their resources, including codebase and executables of their game *60 Seconds!*, its data collection pipeline, gameplay telemetry dataset, data storage facilities, and hardware. They also permitted conducting experimental evaluation trials in the commercial version of their game.

Contents

List of Tables

List of Figures

Abbreviations

2D	two-dimensional
3D	three-dimensional
AI	artificial intelligence
ANN	Artificial Neural Network
AGI	Artificial General Intelligence
BT	behaviour tree
CGFs	Computer Generated Forces
CPU	Central processing unit
DBI	Davies-Bouldin index
DL	Deep Learning
DLC	downloadable content
DRL	Deep Reinforcement Learning
EA	Electronic Arts
EMD	Earth Mover's Distance
EULA	End User License Agreement
FSM	finite-state machine
GAIL	Generative Adversarial Imitation Learning
GaaS	Games-as-a-Service
GGP	General Game Playing
GP	genetic programming
GPU	graphics processing unit
GUI	graphical user interface
HFSM	hierarchical finite-state machine
IL	Imitation Learning
IRL	Inverse Reinforcement Learning
IQR	interquartile range
LfD	learning from demonstration
LfO	Learning from observation
LTS	Long Term Support
MDP	Markov Decision Process

ML - machine learning

MCTS - Monte Carlo Tree Search

MOBA - multiplayer online battle arena

MSE - mean squared error

ONNX - Open Neural Network Exchange

PbD - program by demonstration

PCG - procedural content generation

POMDP - Partially Observable Markov Decision Process

PPO - Proximal Policy Optimisation

QA - quality assurance

QQ - Quantile-quantile

RAM - Random Access Memory

RL - Reinforcement Learning

R&D - Research and development

RTS - Real Time Strategy

SEED - Search for Extraordinary Experiences Division

SNSB - Select Next Scavenge Behaviour

SNTI - Select Next Target Item

SNTA - Select Next Target Area

STRIPS - Stanford Research Institute Problem Solver

SSE - sum of squares error

TD - Temporal-Difference

Chapter 1

Introduction

Summary. In this chapter, we present the premise and goals of our research work. First, we outline the state of video game AI and the reasons for the limited use of learning models in industrial game agent AI. The design proposal for integrating the learning model into the ad hoc behaviour architecture, adhering to the reality of industrial development pipelines, follows. Finally, we elaborate on the contributions of our research and detail the structure of this dissertation.

1.1 Premise

1.1.1 AI and Games

Humans have been playing games since the dawn of civilisation [109]. It was only natural that the invention of digital computers was soon followed by the development of the first computer games[1]. Automated game playing was a source of human fascination long before the digital age, with contraptions such as the hoax chess automaton Mechanical Turk [106] and early 20th century electro-mechanical carnival arcades [174] capturing the imagination of crowds worldwide. When Alex Bernstein implemented a fully functional chess-playing program on an IBM 704 mainframe in 1957 [155], building on prior work of Alan Turing [247], human players were finally able to compete against artificial opponents. For the next half a century, chess would come to serve as a benchmark for AI development, driving the ambition to create a program that could match, and ultimately surpass, human play skill. The milestone moment arrived in 1997 when IBM's purpose-built Deep Blue supercomputer defeated the reigning world chess champion, Garri Kasparov [106].

By that time, the video game industry's commercial development and publishing pipeline was

[1]Since the 1970s, also called video games, due to the emergence of game consoles connected to television sets [3]. We will be using both these names interchangeably.

already 30 years in the making [273]. AI in computer games evolved from simple sequential logic behaviours of the early arcade titles to sophisticated multi-agent decision-making systems for large state spaces found in modern strategy games. However, goals and methods of game AI deviated from those of academic AI research. The industry has been focussed on refining player experience and enjoyment, taking advantage of manual authoring and rule-based solutions, as well as tricks and optimisations [145, 261] to create approximated illusions of intelligence, rather than complete models of human-like behaviour [59, 196, 247]. This resulted in the so-called "gap" between the industry and the academia [189, 270]. It was not until the early 2000s that the academic community started to recognise the potential of computer games for use in research towards achieving human-level AI [141]. Classic board games such as chess, originally considered the "drosophila for AI research" [154], were no longer a challenge for contemporary computing. Video games became prime candidates to serve as the next universal AI benchmark [246, 270]. Research interest in game AI intensified in the 2010s, as ML gained traction and AI for playing video games started to achieve impressive results in complex game environments.

1.1.2 Towards Learning Agents in Commercial Video Games

Despite the recent research achievements in developing autonomous learning agents using video games, there has been limited adoption of relevant learning techniques for producing agents in modern video game development. The game industry's AI community has been more concerned with iterating on existing technology to address practical problems of contemporary game projects [191, 197]. The safety of established, proven AI methods and business considerations make the industry hesitant to adopt a "high risk-high gain" model and incorporate new AI techniques into runtime environments of commercial games [270]. Limited domain knowledge transfer has been hindering the potential to generalise and scale research findings for practical applications in video game development [247]. Although the last decade brought the industry and academic experts closer, constructive dialogue to rectify the "gap" is still in its infancy [267]. AI solutions targeting video games remain focussed on the context of video game development, or even specific gameplay problems. On the other hand, academic researchers are more invested in universally applicable General Game Playing (GGP) methods [246], with no requirements for prior knowledge of an underlying state data model. However, such solutions suffer from high costs and a limited flexibility of training output models. Even research achievements that might appear relevant for the game industry, such as *AlphaStar* winning in *Starcraft II* [256], are prohibitively expensive to produce in terms of infrastructure, training time, and requirements for pre-existing, massive training datasets [202]. Similar to IBM's Deep Blue [215], they are purpose-built, and their generalisation to address commercial video game production challenges within a reasonable time horizon is unlikely.

Although few commercial games have used data-driven models for runtime agent behaviours, learning has found successful applications in game production workflows. ML-driven game analytics and player modelling [64,102], as well as asset generation [221,270], are now an industry standard. The common denominator of these scenarios is the capacity for human expert curation and co-authoring of the data-driven model, and its outputs, at production time [148, 226]. In runtime environments, uncontrollable, non-deterministic, and potentially erroneous model outputs introduce the likelihood of undesired AI behaviours, violating the context of the game environment, and degrading player experience [143, 196]. Such risks are unacceptable in commercial game titles, and discourage developers from choosing data-driven, black-box solutions over manual authoring of transparent AI model representations [194], even if applied learning could result in better models of human-like behaviours [270]. From an industrial perspective, maximising control over training and providing guarantees of reliable execution is required to make data-driven models a viable option for scenarios where applying autonomous learning agents in runtime game environments would provide an added value. Documented examples of such scenarios include agents for game playing and automated playtesting [23, 184], which could be trained with human-generated data to effectively model human-like AI behaviours. This would allow for the greatly automated and enhanced functional, exploratory, and even experience game testing with artificial players. Combined with targeted player modelling, such agents could be used in AI-assisted design, enabling developers to observe and analyse specific in-game behaviours, and to adjust game environments to accommodate playstyles of different types of players [12, 85]. Such developments could contribute not only to tailoring production time content, but also to a streamlined production of data-driven, dynamic game experience balancing solutions in live, commercial games [177].

Research on developing autonomous game agents with controllable learning is ongoing in both the academia and the industry. However, academic research has been constrained by limited technical domain knowledge of game development and access to commercial game environments [85, 270], while industrial studies remain sparse. A promising solution to this challenge is to integrate learning logic into state-of-the-practice ad hoc AI representations, such as BTs [29, 72, 211, 265, 285]. This approach is potentially applicable in the game industry, as it combines and leverages the advantages of both techniques, and incorporates human expert authorial control and design intent embedding, in compatibility with established game development AI workflows. To our knowledge, at this point in time, industry viable case studies, involving commercial development workflows and environments that would demonstrate an integration of learning logic into ad hoc AI representations with execution and performance guarantees, are not available.

1.2 Thesis Overview

1.2.1 Statement

Commercial video game playing AI relies on ad hoc behaviour authoring solutions, which do not model human-like behaviours as well as ML, but provide greater design control, transparency and reliability in terms of performance and execution, in comparison to data-driven models. We assert that expert-curated learning models, trained with real player data, can be integrated into the industry state-of-the-practice ad hoc AI architecture with performance and execution guarantees, to achieve approximately similar or better gameplay performance than human players in unseen game environments.

1.2.2 Proposal

We postulate that to adhere to the established game industry development practices and workflows, striving for determinism and authoring control, non-automated human involvement, and maximised manual modification capabilities [59, 194, 211, 270, 285] must remain part of the game-playing AI design process, even when learning techniques are involved. We apply these assumptions and contribute to the work towards practical applications of learning AI in video games by proposing and investigating a design approach for video game playing, autonomous agent AI, featuring optimal design intent embedding and safe behaviour flow execution based on reliable neural network controllers with performance and execution guarantees. Our proposal constitutes a data-driven, industry-applicable solution which addresses domain requirements and builds upon prior research work in the field. The designed behaviour flow is structured using hierarchical ad hoc authoring, based on the principles of subsumption architecture [27,28], generalised by behaviour trees (BT) [42, 44]. Layering behaviour flow through informed context guidance and decomposition provides measures for modular integration of learning logic instances into the ad hoc flow. Performance and execution guarantees of learning model logic are established on the basis of safety control, embedded through ad hoc-driven redundancies for neural network components [30,234]. Human-like behaviour of designed agents is achieved by employing a combination of Reinforcement Learning (RL) and Imitation Learning (IL) algorithms in integrated learning models, trained using a game environment, and gameplay trajectory data sourced from real players of the game.

The ambition of the proposed design, designated as context-guided design, is to empower game designers in taking advantage of the adaptive properties of learning AI, while maintaining high-level control of the designed flow and execution of agent behaviours. From a design perspective it is intended to offer a familiar, human readable representation of an AI model, while supporting context-based segmentation of learning, and flexible ad hoc authoring for iterative AI development and deployment. The proposed design is original and was informed by the industry requirements and prior research work in the field. It features an original contribution to the structure of a BT learning node, which expands the concept with industry-applicable safety redundancies.

1.2.3 Objectives

The primary objectives of this thesis were to present the context-guided design proposal, and to investigate its application in the real world conditions of a commercial game environment:

- Formulate an industry-applicable context-guided design, and its deployment workflow.
- Deploy a context-guided agent in a commercial game environment.
- Evaluate the deployed agent in new scenarios of the commercial game environment.
- Analyse the training and operations of the deployed agent instance.

To support the work towards primary objectives of this thesis, and facilitate the anticipated scope of research and development, we identified a set of secondary goals to attain:

- Interface with and set up the game environment for agent simulation and evaluation.
- Acquire gameplay telemetry dataset from the game environment.
- Establish an evaluation metric of gameplay performance in the game environment.
- Extract relevant data from the gameplay telemetry dataset to use in imitative training of context-guided agents.
- Deploy evaluation scenarios in the game environment.

1.2.4 Questions

On the basis of the premise, the proposal, and primary objectives outlined for our research, we set out to address the following questions in this thesis:

- **RQ1**: can models with execution and performance guarantees of learning logic be integrated into the game industry, state-of-the-practice, ad hoc behaviour AI architecture for applied use in video game playing AI?
- **RQ2**: how well can a trained context-guided learning agent perform in unseen game environment scenarios, in comparison to human players, in approximately similar gameplay conditions?

1.2.5 Plan

We chose to explore our proposal in the format of an industrial case study, using the environment of a commercial video game *60 Seconds!* (Robot Gentleman, 2015), industry state of the practice workflows, and off-the-shelf solutions. Plan for our research work was aligned with the identified objectives, and the conditions and features of the selected game environment. This afforded us opportunities to access to the game's mass-scale, gameplay telemetry dataset, generated by the game's large player base, and adopt a data-driven and quantitative approach in our work. The author's extensive, over 15-year background in commercial video game development supported the practical perspective of this investigation. This, combined with the identified objectives, informed the plan for our research work:

- Formulate an industry-applicable context-guided design and its deployment workflow.
- Instrument the game environment to support the simulation of autonomous agents and conduct evaluations with real and artificial players.
- Collect and process gameplay telemetry dataset generated by players of *60 Seconds!*.
- Establish a game score metric to evaluate gameplay performance in the game.
- Conduct an analysis of game scores recorded by players of the game.
- Extract a top-skill persona trajectory dataset from the gameplay telemetry dataset.
- Design and deploy a context-guided agent instance, trained with the top-skill persona data, in the instrumented game environment.
- Deploy new game environment scenarios for experimental evaluation with context-guided agents and human players.
- Evaluate the deployed context-guided agent in experimental conditions.
- Analyse the training process and the evaluation results of the deployed context-guided agent instance.

1.2.6 Structure

This thesis is structured into the following chapters and appendices, which present the complete line of inquiry and content of our research work:

- **Chapter 1: Introduction**: introduces readers to the general premise of our work of integrating learning into video game agent AI ad hoc architecture, and presents the goals of our research.

- **Chapter 2: Game Environment**: presents the environment of the commercial video game *60 Seconds!* used in our research work.
- **Chapter 3: Background**: explores the background of our research through a literature review and discussion of relevant work in the context of video game design, computer game AI, and machine learning.
- **Chapter 4: Context-Guided Agents**: establishes the foundations of our research by introducing our methodological approach, procedure applied, and evaluation planned, as well as presenting the learning agent design proposal, discussing the game environment instrumentation, and documenting gameplay telemetry dataset collection and processing. Chapter 4 addresses research question **RQ1** from a theoretical perspective.
- **Chapter 5: Game Score Study**: proposes a game context-derived game score metric and uses it to conduct statistical analysis of the gameplay telemetry dataset, in order to quantitatively interpret and extract relevant data to produce a top-skill play persona dataset.
- **Chapter 6: Agent Study**: details and conducts analysis of the process and results of designing, training, and experimentally evaluating the deployed context-guided agent model. Chapter 6 addresses research question **RQ2**, and provides an empirical response to research question **RQ1**.
- **Chapter 7: Conclusions**: summarises the process and the outputs of our research work, and discusses its challenges, contributions, and perspectives for follow-up investigations.
- **Appendix A: Data**: provides supplementary information about the data used in our research work.
- **Appendix B: Evaluation**: features additional documentation about the experimental evaluation conducted in the commercial game environment.
- **Appendix C: Software**: provides information about the setup, requirements, and operations of the software developed for our research work.
- **Appendix D: Additional Results**: presents supplementary results in support of our research work, including calculations, tables, and figures that were not included in the main body of the thesis.

1.2.7 Research Contributions

Contributions made in the course of our work include:

Chapter 4: Context-Guided Agents

- **C1**: industry-applicable context-guided learning agent design and deployment workflow proposal.
- **C2**: extension of the BT learning node concept, featuring additional safety redundancies.
- **C3**: a mass-scale, processed gameplay telemetry dataset from the game *60 Seconds!* that was used in our research, and later shared with the research community.

Chapter 5: Game Score Study

- **C4**: game score metric for quantitatively measuring play skill in terms of gameplay performance in the *scavenge* segment of the game *60 Seconds!*.
- **C5**: analysis of game scores measured for the gameplay telemetry dataset from the *scavenge* segment of the game *60 Seconds!*.

Chapter 6: Agent Study

- **C6**: analysis of the training process of a context-guided learning agent, capable of playing the *scavenge* segment of the game *60 Seconds!*, developed on the basis of the context-guided agent design.

1.2.8 Research Outputs

Outputs produced in the course of our work include:

- **O1**: implementation of a BT learning node, integrating *Unity Machine Learning Agents* learning capacity into PadaOne Games BT library *Behavior Bricks*.
- **O2**: instrumentation of the *scavenge* segment of the commercial video game *60 Seconds!* for enhanced gameplay telemetry data acquisition, simulating autonomous agent operations, and experimental evaluations with human players.
- **O3**: standalone simulator software for simulating autonomous agents, based on the *scavenge* segment of the *60 Seconds!* gameplay environment.
- **O4**: k-nearest neighbours classifier of play skill in the *scavenge* segment of the game *60 Seconds!*.
- **O5**: a trained context-guided learning agent, capable of playing the *scavenge* segment of the game *60 Seconds!*, developed on the basis of the context-guided agent design, using game industry off-the-shelf solutions.

- **O6**: a published game AI book series *Game AI Uncovered* chapter presenting the context-guided learning agent design and deployment workflow [88].

1.2.9 Reproducible Results

This thesis was written to facilitate as complete reproduction of procedures and results presented, as possible. LaTeX, Python, and Jupyter Notebook source files, as well as supplementary data files, used in our work, are part of the digital supplement to the dissertation. Most of the C# code written to extend the environment of the game *60 Seconds!*, and the binary version of the agent simulator environment developed on the basis of the game, are also provided. However, the complete set of project files for the agent simulator environment, as well as the game's original C# source files, including some that were extended for the purposes of our research, could not be shared publicly and are not included in the digital supplement to the dissertation. Developers of the game have agreed to make them available upon specific requests, for research purposes only.

The processed telemetry dataset used in our research will be shared online. Developers of the game have consented for the academic community to reuse the data contained in the dataset in other research projects. Information about accessing the processed gameplay telemetry dataset is provided in Appendix A: Data.

Chapter 2

Game Environment

Summary. This chapter presents the environment of the commercial video game that we chose to use in our research. We first discuss the structure of the game, and then examine the *scavenge* portion of the game, whose foraging gameplay was the focus of our investigation. Finally, additional elements of the game are discussed.

2.1 Overview

2.1.1 Goals

The objective of this chapter is to present the environment of the video game *60 Seconds!* and discuss its features that were of interest in our research. After reading this chapter, readers will understand the structure and the gameplay context of the game environment.

2.1.2 Structure

The contents of this chapter are divided into the following sections:

- **The Game**: outlines the premise of the game, technology used and its commercial availability.
- **Structure**: discusses the flow of the game, based on the *scavenge* and *survival* segments.
- **Scavenge**: examines features of the *scavenge* portion of the game, including its gameplay context, rules of play, setup parameters, and presentation.
- **Survival**: provides a brief outline of the gameplay context of the *survival* portion of the game.
- **Game Types**: details different types of gameplay available in the game.

2.2 The Game

60 Seconds! is an action-adventure video game, challenging players to survive a nuclear apocalypse. The game was originally developed using the Unity game engine[1], and has been commercially available for Windows and OSX desktop users of the video game digital distribution platform Steam[2] since May 2015. The game is a premium title purchased with a single payment of £5.99[3]. Between May 2015 and July 2019 it was priced at £6.99. Since its original release, the game has been discounted several times for limited time periods.

As of September 2025, the desktop version of the game was purchased by over a million players on Steam. The game's audience is consistent with Steam's user demographic, which includes players of various ages[4], sex, ethnic backgrounds and nationalities. Age ratings for *60 Seconds!* are PEGI 12 and ESRB T. The game is also available for mobile and console platforms, and an enhanced edition of the game, *60 Seconds! Reatomized* (Robot Gentleman, 2019) is available for purchase on digital PC, console and mobile storefronts. Total game sales across all platforms exceed 5 million units as of 2025. Our research only uses data collected from users of the Steam PC and OSX 2015 version of the game.

2.3 Structure

A single playthrough of the game is divided into two gameplay sections linked by the common premise of surviving the nuclear apocalypse: *scavenge* and *survival*. Each of them features a distinct game loop, involving different player interactions and objectives. *Scavenge* is an action sequence, which tasks players with collecting and depositing collectable items in the safety of a fallout shelter, within the titular 60 seconds. If completed successfully, the game progresses to the turn-based *survival* gameplay segment. *Survival* takes place in the fallout shelter stocked with supplies collected by the player during *scavenge*. Each turn in the *survival* gameplay, equivalent to a single day in the game, challenges players with catering to the needs of surviving characters, resolving daily situations by optimal decision-making, and making the best use of available supplies. A single playthrough concludes if all adult characters perish or if one of multiple story endings is reached. Regular playthrough always begins with *scavenge*, which is then followed by *survival*. Alternative game types allow players to limit their gameplay to either *scavenge* or *survival* section of the game.

[1]A generalist game development engine created by Unity Technologies, extensively used by the game industry.

[2]Valve's Steam service has a community of over 132 million monthly active users, as of January 2023, and is available worldwide, except for countries excluded from the platform by United States imposed embargoes. Current data available at: https://partner.steamgames.com/.

[3]Or equivalent in local currencies, including \$ 8.99 and €8.99.

[4]But not younger than 13, as restricted by the Steam user agreement.

2.4 Scavenge

2.4.1 Gameplay

In *scavenge* players navigate a humanoid, bipedal avatar in a randomised, three-dimensional (3D) environment e, foraging *items* and attempting to reach an exit area, before the time $t_{scav} = 60$ s runs out. The fail state s_f is triggered if the avatar is not present in the exit area at the end of the t_{scav} collection time, no matter how many *items* were collected. Conversely, the success state s_w is triggered when t_{scav} runs out, and the avatar is located within the exit area. Since players have a limited carrying capacity, they need to regularly approach the exit and deposit their current inventory load. They are not able to collect more *items* until they do so. Collectable *items* have different weights, which directly impacts player carrying capacity. Depending on the difficulty level of the game, before a player can start collecting *items*, they get an opportunity to explore the environment in a short time period t_{exp}, preceding the actual collection t_{scav}. The game requires strategising survival and foraging in a semi-unknown environment. Because of that, the exploration factor plays an important role in *scavenge*. Players are challenged to maximise time spent on the move, balance limited inventory load, and adaptively min-max exploration (for discovery of new targets) and foraging. Reaching the exit area in time but with none or few *items* could be considered a poor result as it will negatively impact the player's starting conditions in the subsequent, *survival*, stage of the game.

2.4.2 Mechanics

Navigation in *scavenge* is restricted to a flat, two-dimensional (2D) surface, even though the game's presentation is 3D. Players control their avatar via virtual navigation mapped to one of the supported input device combinations: mouse, keyboard, keyboard with mouse or a gamepad. The mappings cover the player permitted navigation action space, including vertical axis movement (forward and backwards), horizontal axis movement (left and right strafe) and rotation around the avatar's yaw axis. By default, the horizontal and vertical velocity of the avatar is $v_{start} = 0\ \frac{u}{s}$. By continuously moving on a specific axis, the velocity for the avatar builds up with an acceleration of $a = 5\ \frac{u}{s^2}$, until reaching the velocity limit for a specific axis. The avatar's maximum velocity for horizontal axis movement and backwards vertical axis movement is uniform at $v_{bmax} = 4\ \frac{u}{s}$, while the maximum forward velocity is $v_{fmax} = 7\ \frac{u}{s}$. Avatar's movement is continuously illustrated by a looping 3D run animation, which plays for as long as the velocity of the avatar is greater than zero. The velocity of the avatar is reduced to zero, and its movement in the environment and running animation are stopped if no navigation input is provided or if the player inputs a movement blocking avatar interaction. Collisions with environment elements are possible (with walls and props) and result in slowing down the avatar, relative to the scale of the physics reaction triggered by the collision. Bigger and more sturdy obstacles can effectively stop the avatar, while smaller ones can be pushed out of the way by the avatar's body. Designers of the game defined individual physics parameters of each prop present in the environment.

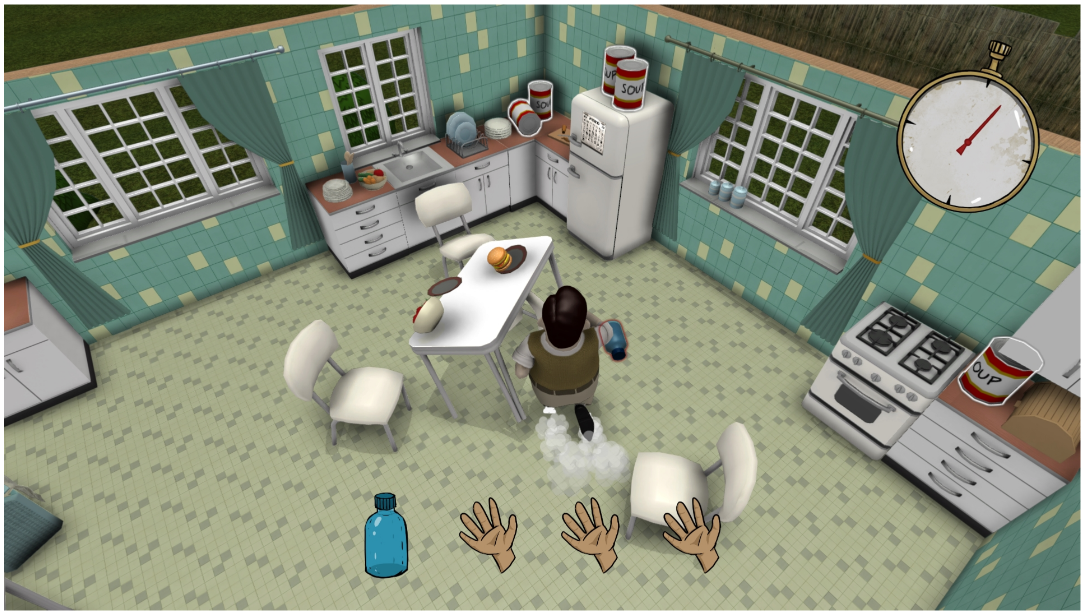


Figure 2.1: Scavenge gameplay screen.

Player can interact with the game space via context-sensitive interaction functionality. If triggered, it can invoke one of two available interactions: collecting an *item* or depositing all carried *items*. *Scavenge* environment *E* is populated with a set of *items T* to be collected. To collect an *item*, the player must position their avatar facing an *item*, within its individually defined interaction range. Player's avatar has a $i = 4$ slot carry capacity for *items*, defined as *inventory* by the game's designers. It constitutes a fixed maximum size set of *items I*. Each *item* found in the game space has a weight *w* associated with it, which translates into how many *inventory* slots it occupies. Successfully collecting an *item* will play the appropriate 3D animation, update the game space state and insert the collected *item* into the *inventory* set by blocking a number of slots corresponding to the *item's* weight. Attempting to pick up an *item* that does not fit into the *inventory* will fail and produce an audio-visual cue for the player. There are 21 unique types of *items* available in the game. They occupy a single *inventory* slot (13 *item* types), two *inventory* slots (5 *item* types) or three *inventory* slots (3 *item* types). Most *items* only have one collectable instance present in an environment, but two special *items* (*soup* and *water*) can appear more than once in a single game session. Three family members to be rescued are also classified as collectable *items* in terms of game design and follow the same collection rules, as any other *item*.

Once one or more *items* are placed in the *inventory*, its slots can only be freed up by depositing collected *items* in the exit area deposit, designated as collected *items* set *C*. To deposit current *inventory*, the player must position their avatar inside the exit area's interaction range and frontfacing the exit area visual object. Successful depositing will empty the avatar's *inventory* set *I*, making room for new *items*, and transfer all its contents to the collected *items* set *C*. *Item* collection and depositing are only possible in the 60-second gameplay time frame, while the exit area is available. Completing a *scavenge* game requires the player's avatar to enter the exit area before the time runs out. Meeting this condition triggers the success state s_w, illustrated by the success cutscene[5]. The game then progresses into the *survival* gameplay section. Failing to make it to the exit area triggers the failure state s_f, illustrated by the failure cutscene. The game then terminates the ongoing game session.

2.4.3 Setup

A *scavenge* game can be parametrised with a selection of game setup parameters, including:

The game's setup parameters include:

- **Game type**: *scavenge*, *full*, *scavenge challenge*. Game type choice is the first player-made decision for configuring new gameplay. It influences available level selection, as well as *item* collection rules.

- **Difficulty level**: *easy*, *normal*, *hard*. Difficulty choice in *scavenge* translates into the amount of exploration time t_{exp} the player gets to inspect the environment (for easy t_{exp} = 20 seconds, for normal t_{exp} = 10 seconds, for hard t_{exp} = 0 seconds), before the actual time of t = 60 seconds of *scavenge item* collection begins. During the exploration period, the exit area remains locked. At that time, the player can navigate the avatar at reduced velocity of $v_{bmax} = 4\ \frac{u}{s}$, they are unable to interact with collectable *items* in the environment. They are, however, able to discover the layout and placement of *items*, as well as position their avatar favourably for when the actual *item* collection begins.

- **Character selection**: *Ted* or *Dolores*. While the two available avatar characters are different presentation-wise, they do not operate differently in terms of gameplay behaviour. However, when playing as *Ted* players get to collect *Dolores*, but not *Ted*, and vice versa. This introduces a minor change to *item* collection, as *Ted's* inventory weight (w_t = 3) is different from *Dolores's* (w_d = 2). This affects the set of *item*s *T* to be collected and makes scavenging with *Dolores* potentially more challenging.

- **Extended *item* set**: true or false. This parameter is beyond player control and is automatically true for any game played after August 2018, when the developers introduced a new content update to the game, expanding the original *item* set. Featuring an additional *item* does affect the set of *items* *T* and the total weight that can be collected.

[5]Cutscene is either a film sequence embedded within a game or a scripted, animation sequence implemented in the game engine, whose goal is to enrich the game's narrative with a presentation segment that does not involve player's input and control.

- **Level**: scavenge level range $L = [0, 19]$ in *scavenge* and *full* type games, custom levels in *scavenge challenge* type games. Selected with no player input, either pseud-randomly for the available level range in *scavenge* and *full* type games, or specifically in *scavenge challenge* type games.

2.4.4 Environment

Scavenge gameplay takes place in a stylised 3D environment structured as a simple, closed-circuit maze. *Scavenge* environments are inspired by a quasi-realistic layout of a suburban, American house. There are $n_e = 20$ base variations of the *scavenge* environment; however, its general architecture remains uniform with 8 room types, 7 of which are fully traversable and may contain collectable *items*. An instance of the environment, with a specific combination of design elements and variants, established manually or procedurally, is identified as a game level.

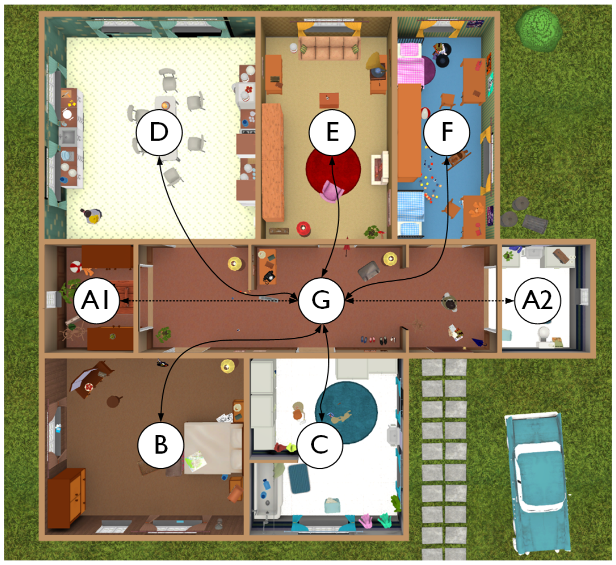


Figure 2.2: Game environment architecture and traversal graph; A1 and A2 are conditionally connected with G, depending on which room placement is occupied by the exit.

Architecture has 8 room placements defined, each with a specific identifier. Certain room types can only be inserted into specific placements, while others can be put in more than one. Some rooms have versions that differ in terms of obstacle, decoration and *item* positioning. Each room type can only appear once during a single playthrough. Possible room designs are outlined in table 2.1 on page 16.

For each environment variation, room placements and versions may be different. The player's starting position is fixed in all environment variations and is found in the G placement, which serves as a nexus, providing access to all other rooms. Other rooms are not directly connected to

Room type	Allowed room placement ids	Number of versions	Function
Shelter	A1, A2	3	Exit
Hall	G	3	Start, traversable
Bathroom	B, C, D, E, F	1	Traversable
Bedroom	B, C, D, E, F	1	Traversable
Kids' room	B, C, D, E, F	1	Traversable
Kitchen	B, C, D, E, F	1	Traversable
Living room	B, C, D, E, F	1	Traversable
Toilet	A1, A2	3	Traversable

Table 2.1: Rooms available in the game's environment architecture.

each other. The exit is always located in the shelter room type, placed in node A1 or A2. The positions of collectable *items* appearing in the game are pseudo randomised[6] for each playthrough, utilising a large set of possible valid position markers, manually defined, and placed by game designers for each room version. Unless a specific environment variation is forced (in game modes, such as *scavenge* challenge), the *scavenge* environment variation is chosen pseudo-randomly from the available designs for each newly started *scavenge* playthrough.

While traversing the room-based environment, the player is continuously challenged by physical obstacles that may impede their progress. Some objects can be knocked out of the way, but will slow players down, while others will block the avatar's movement completely. The environment resembles a simple, rectangular maze. However, the necessity of travelling multiple times between the exit and foraging points of interest makes it challenging from an abstract route planning perspective, rather than finding a way out, as is the case with traditional mazes.

The representation and dynamics of 3D game space are abstracted in terms of the Unity Engine's implementation of the Cartesian coordinate system and its approximated model of Newtonian physics. Distances and positions are expressed in Unity units u, whose relevant value range is relative to the parameters chosen for a specific game project and sizing of imported 3D graphical assets. Unity recommends defining $u = 1$ m. In *60 Seconds!*, the u is fixed across all of its 3D environments with $u = 5.1282$ m, due to variance in 3D assets used.

2.4.5 Presentation

The viewport camera, providing the game's presentation, is anchored to the player avatar, continuously following it from behind at a fixed distance and observing a 45°downwards angle with respect to the environment's flat, ground surface. The camera rotation is synchronised and continuously updated to match the avatar's rotation. The rotation speed is dependent on player input sensitivity settings and can effectively be anywhere in the range $[0.0, \infty]\ \frac{o}{s}$. The camera is positioned in a way that allows the player to clearly see the immediate surroundings of their avatar

[6]Using Unity's built-in random implementation, based on the *Xorshift 128* algorithm, which is used for all pseudo randomised elements in the game.

and to get a partial view of what lies further ahead, but with walls and other tall obstacles effectively obscuring full view. Collectable *items* are displayed with highlighted silhouettes, which are visible even through solid obstacles, offering additional guidance. The exit area is marked with special, visual guidance cues. The player's view contains screen space user interface elements that provide information about the game's progress, including the regularly refreshed countdown timer and the visualisation of the current state of the avatar's inventory. Additional notifications appear as the player interacts with the game space and as time flows. Player is made aware of reaching the midpoint of the *scavenge* playthrough by an increase in music intensity and alarming visual cues.

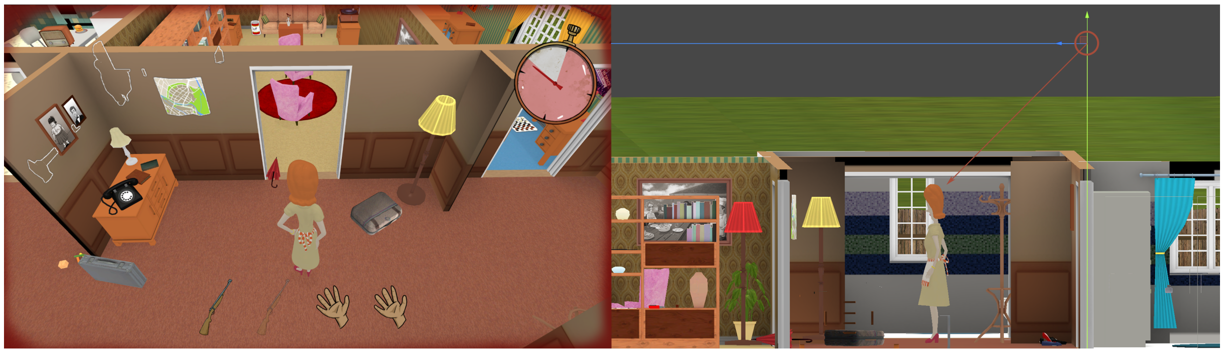


Figure 2.3: Scavenge gameplay camera positioning.

2.5 Survival

2.5.1 Gameplay

In *survival* players take on a custodian role, trying to keep their avatar character, as well as all the characters they rescued during *scavenge*, alive and healthy. Gameplay progresses day by day in a turn-based fashion, where each day represents a single turn. Every day tasks the player with rationing consumable supplies to the surviving characters (food, water, and medicine), optionally planning an expedition to scavenge supplies in the outside world, and making decisions in response to *events* that unfold. The latter often necessitates the use of previously scavenged supplies. *Event* resolution is always shown on the very next day, and depending on the context and the decision made by the player, it can lead to negative or positive effects that impact the game state. The game failure state is triggered if no control characters (adults - *Ted* and *Dolores*) remain alive, or if a story scenario concluding with a fail state is reached. The success state is achieved by successfully completing one of the storylines, conveyed through a string of specific *events*. *Survival* game space is a claustrophobic fallout shelter, presented in the form of a 2D screen, which changes to reflect a variety of developments that take place during gameplay. Players interact with the world and their characters via a multi-layer graphical user interface (GUI) abstraction of a journal, which presents the *events* of the game.

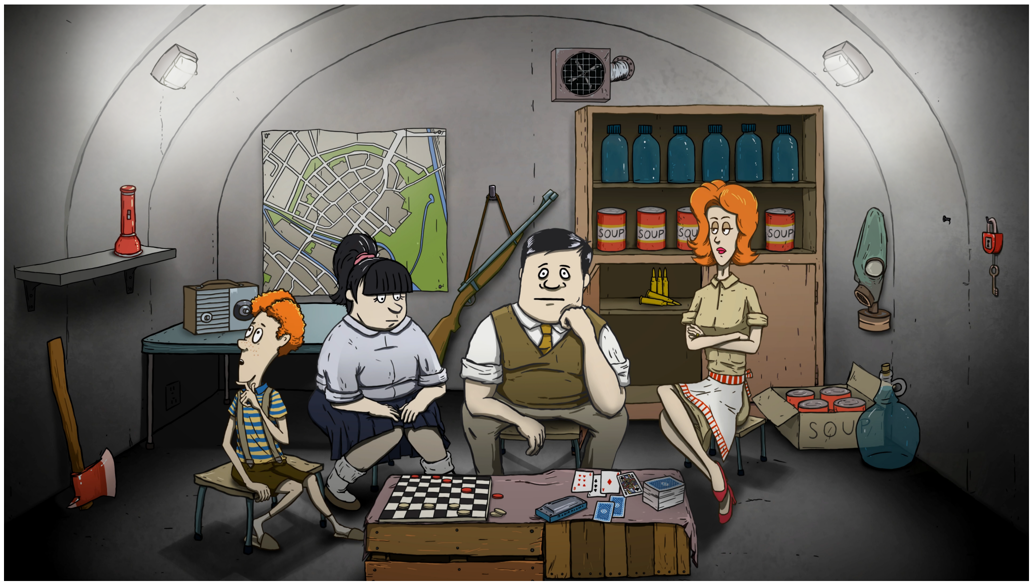


Figure 2.4: Survival gameplay screen.

2.5.2 Mechanics

The starting *survival* conditions are directly dependent on the contents of the collected *item* set C. Whatever and whomever was collected by the player in the *scavenge* section of the game will be included in the *survival* segment. While more supplies can be obtained from *survival* events, large and varied C is key to increasing chances of succeeding in the game. Players progress through *survival* day by day. A day is equivalent to a turn. There are no time limits, so players can take as long as they need to finish a single day, and an entire *survival* session. Developers of the game stated that a single *survival* game session is expected to last approximately 30-60 minutes. The game saves the state of gameplay automatically upon the start of each new day, allowing players to exit their current session and come back to it later. There is only one save file available, so players are unable to manually reload an earlier segment of their playthrough or to have more than one active game session in progress. This enforces a commitment to the consequences of decisions made earlier, in the course of the gameplay session. Completing the *survival* game mode with a success or fail state concludes a single gameplay session and provides players with an end summary of their progress.

2.6 Game Types

By design, a full game playthrough of *60 Seconds!* includes both *scavenge* and *survival* segments. However, developers provided other ways to play the game:

- ***Tutorial***: a guided tour of *scavenge* and *survival* gameplay features for new players.
- ***Scavenge***: a game of *scavenge*, with no *survival* follow-up.

- ***Survival***: a game of *survival*, with no *scavenge* segment. Starting conditions are determined based on pseudo-randomisation from the game design's predefined values to simulate results of a *scavenge* run.

- ***Scavenge Challenge***: a game of *scavenge*, with no *survival* segment, using a predefined collection goal, and optionally a custom environment and altered gameplay rules. Players are required to collect a specific set of *items* in the shortest time possible. Completing a *scavenge* challenge awards players special visual awards for use in the *survival* segment.

2.7 Gameplay Telemetry Data Collection

In an effort to conduct data-driven investigations of player gameplay, the developers of *60 Seconds!* integrated a data collection mechanism into the *scavenge* segment of the game. This enabled the development studio to remotely crowdsource gameplay telemetry data from the game by recording *scavenge* trajectories when the Steam PC version of the game was played by its users. Developers did not implement a data collection mechanism for the survival segment of the game. Full discussion of the data collection procedure, acquired datasets, and the ethics involved follows in the Gameplay Telemetry Dataset section of Chapter 4.

2.8 Summary

The gameplay of the *scavenge* section of the commercial video game *60 Seconds!* can be described as a virtual foraging task in a simple, maze-like environment with dynamically altered architecture, procedurally populated with collectable items, as well as obstacles impeding the progress of the player avatar. It features a large enough state space and play skill requirements to generate interesting challenges for the game's players, and potentially for autonomous game-playing agents. The commercial nature of the game, its use of the state-of-the-practice development workflows, and a pre-existing data collection mechanism in the *scavenge* segment of the game made it a valid target environment for a game industry-relevant case study.

Chapter 3

Background

Summary. In this chapter, we establish the background of our research by conducting a literature and subject matter review. We first investigate the context of video game design and game AI, and then explore the capacity of learning models and their integration into autonomous agent AI in video games. Finally, we present a summary of the most relevant points, leading up to our research work.

3.1 Overview

3.1.1 Goals

The objective of this chapter is to explore the background context of our research and present relevant past work. After familiarising themselves with this chapter, readers will be equipped to follow the line of inquiry presented in this thesis.

3.1.2 Structure

The contents of this chapter are divided into the following sections:

- **Video Game Design**: discusses the basic premise and goals of video game design and outlines the role that AI plays in it.
- **Video Game AI**: describes selected areas of video game AI, relevant to our research, including game-playing AI, player modelling and automated playtesting.
- **Behaviour Trees**: documents the structure and functioning of behaviour trees, the state of the practice game industry ad hoc authoring technique.
- **Learning for Video Game Playing**: introduces the methods of ML, and then explores efforts to incorporate learning models into video games.

- **Digest**: presents a summary of the discussed background topics to provide a foundation for understanding the themes explored in our research work.

3.2 Video Game Design

3.2.1 Designing for Fun

Although video games can be considered software projects, the process and goals of their development are different from those of other computer programs [169]. Video games are designed to provide a fun, enjoyable, and playful interactive experience for their players cite Koster2013, Sicart2014. Some games are rigorously modelled as formal systems and can be effectively quantified [123], but for others it is hard to classify player enjoyment and engagement in terms of functional requirements, as reception of a game's content is subjective [4, 102]. One way to measure how captivating a game experience may be is to observe if the players find themselves in a state of flow, understood as a full immersion in the game's virtual environment and gameplay [49]. Triggering such an attention-absorbing state requires optimal balancing of challenges, achievements and failures experienced by the player [10, 18]. The challenge element of video game experience is often cited as the key trigger of intrinsic player motivation to solve puzzles embedded in a game's environment [151, 239] and to "succeed" in the context of the game's rules [109]. Creating satisfactory game challenges aimed at capturing specific, directed experiences in controlled environments requires a playful and systematic approach from game designers [78]. The complexity of game development makes crafting games a complicated process, which often requires an individual approach to authoring specific game mechanics, systems, and features [19, 124]. To support the manual labour of multidisciplinary domain experts in this process, and to decrease the risks involved in game production, data-driven solutions to many problems faced by game designers have been proposed, developed, and integrated into industry workflows [64, 102]. However, overreliance on procedural solutions, especially in scenarios where their output compromises authored experiences, has been known to be detrimental to the quality of games and their gameplay [226].

3.2.2 AI in Video Games

What is AI?

The birth of AI as a field in the 1950s carried an ambition to make computers capable of intelligent decision-making, comparable to that of humans and animals [160]. The original perspective on how to achieve this was a model-based, symbolic approach of embedding the problem to be solved in a state space, which was to be traversed or searched [270]. This way of implementing AI enjoyed focus for decades in a variety of forms, but always required and assumed human expert participation to drive the process of establishing knowledge representation of a problem and a framework to solve it. This, and other limitations in scaling symbolic approaches, as well as difficulty in expressing biologically plausible intelligence, introduced frustrations, which motivated researchers to explore alternative techniques, inspired by biology and other natural systems [160]. One of the investigated techniques was the Artificial Neural Network (ANN) - a non-linear data structure evoking the principles of neurology [156]. It was eventually combined with ML, whose methods generate a solution to a problem by processing data without explicit instructions, in a supervised or unsupervised manner. As early as 1959, Arthur Samuel presented the first instance of ML in the form of RL-based self-learning AI, which taught itself to play checkers by iteratively improving its results via perceived error reduction [214]. With the increase in computer processing power, large-scale data collection and optimising the handling of underlying ANNs in the last two decades, ML has produced impressive results, and has enjoyed a reinvigorated interest in research, engineering, and business.

What is Game AI?

Game AI originated as a specialisation of symbolic AI, but it was never aimed at producing actual intelligence, but rather a convincing illusion of intelligence [160]. Ever since the first computer games were created, AI has been an integral part of the video game experience, driving the behaviours of non-player characters and player-like, artificial opponents. Be it in adversarial, zero-sum games, such as digital adaptations of chess, or in arcade games that throw waves of enemies at the player. Fifty years of game development saw an evolution of complexity and believability of video game AI and environments in which it operates [160, 164]. Despite these advancements, the purpose of computer game AI remains the same: to support and refine the experience and entertainment value a game offers to its players [59]. AI solutions in video games achieve this by producing approximated illusions, rather than complete models, of intelligence in the service of gameplay [196]. Such illusions are crafted by combining algorithmic solutions, data from fully or partially observable game environments, target scenario context-specific optimisations, and optimal asset creation [270].

Game AI Development

Game developers create virtual game environments and populate them with challenges for players, many of which involve the use of AI techniques [197, 247]. As the complexity, costs and risks associated with commercial video game production and the development of game environments continue to rise [1,16,23,194,218], the business perspective dictates a restrained approach towards experimentation in game development. This has not stopped some in the game industry from pursuing cutting-edge AI concepts, which have resulted in notable breakthroughs. Highly praised for original interweaving of AI and gameplay, these milestone titles influenced technology advancement trends in the entire game industry [267]. Now a state of practice, BTs were once a novel technique popularised by the first two instalments of the *Halo* series (Bungie, 2001-2004). Sufficiently believable human-like agent simulation based on utility AI introduced in the *Sims* (Maxis, 2000) stood out [96], before more complex societal and building simulation games like *Dwarf Fortress* (Bay 12 Games, 2006-2022) emerged as a genre. Experiments in sensor driven logic of opponents in *Thief: The Dark Project* (Looking Glass Studios, 1998), goal-oriented planning based on the simplified Stanford Research Institute Problem Solver (STRIPS) algorithm for enemy teams in *F.E.A.R.* (Monolith Productions, 2005), smart companion AI of Elizabeth in *BioShock Infinite* (Irrational Games, 2013) and the macro and micro level behaviour direction of the titular antagonist in the *Alien: Isolation* (Creative Assembly, 2014) are widely considered hallmarks of advanced character control AI [160, 270]. Higher level use of AI for player experience management via gameplay balancing and pacing has been explored with the AI director architecture in *Left 4 Dead* (Valve Corporation & Turtle Rock Studios, 2008) and other meta AI solutions [272].

Recent developments of technology and pipelines used in the game industry have expanded the range of problems addressed by AI in modern video games. AI solutions are now not only used for game playing but are also commonly applied in game production workflows for generating content both in design-time and runtime [221,226], modelling players [64,69] and supporting the game design process in a myriad of ways through AI-assisted design [93, 218]. More granular taxonomies of game AI specialisations have been proposed, but ultimately, two main types of AI used in game development are AI for player experience and AI for game production [270].

3.3 Selected Fields of Video Game AI

While video game AI has many applications in modern games, our research is concerned with the following topics: game-playing AI, player modelling, and automated playtesting. We outline each of them below.

3.3.1 Game Playing AI

Game-playing AI can be defined as the decision-making logic of artificial participants of gameplay, taking place in a game environment. Game-playing logic can drive autonomous agents, strategic opponent bots imitating human gameplay, and systems simulating the logic of the game's world. When controlling NPC behaviours, it is commonly identified as the "AI" in games by the player audience and considered "character AI" by some game developers [164]. Developing game-playing AI requires a multi-layered approach, achieved by combining specialised AI areas of decision-making, navigation (spatial actions), animation handling (visual actions) and meta AI (player experience management and interweaving of gameplay systems with AI) [160, 164].

The objective of game-playing AI is to generate sequences of decisions and output agent actions which players will see as valid in the context of the game's virtual environment. Ideally, players should be able to interpret these sequences as believable behaviours, which enrich their experience and consistently reinforce the context of the game. Not all game agents are expected to exhibit human-like behaviours, but when they do, their believability is usually preferred over maximising agent operational efficiency [100, 180]. This is a challenge, as achieving valid and efficient behaviours is often easier than producing believable behaviours in multi-dimensional and complex virtual environments [189]. While game environments are abstractions, the approximated models of reality they offer correspond to the real world [191] and virtual agent behaviours are expected to approximate what one may expect to see in the real world [56,158,283]. At the same time, attempts at enforcing strictly realistic human-likeness are not optimal if the target project is not designed as a serious game or real-world simulation. Such efforts can potentially compromise the player's gameplay experience, which does not necessarily benefit from increased realism, and should always be a priority for game AI [59,196]. These factors make the development of agent behaviour AI a non-trivial endeavour. The deployment of agent behaviours capable of adapting to changing conditions warrants a generic approach to both valid decision-making under uncertainty and the execution of output actions. Accommodating imitation of subjective decision-making and potential human biases is also desired, as perfectly performing or repetitive AI behaviours are detrimental to the gameplay experience. Embedding of human distortion in the process is relevant if the behaviour approximation is to be consistent with the psychology of how people judge their circumstances and make their choices [86, 125, 126, 133]. Objective evaluation of the believability of artificial behaviours is yet another challenge, which has been explored with the use of external human observers, Turing Tests and data-driven models of believability [245].

Industry standard game playing AI is implemented using a variety of techniques, predominantly symbolic based, including rule-based systems, sequential scripting, search algorithms, utility methods, action planning systems, and state graph-based solutions, such as finite-state machines (FSMs) and BTs [25, 203, 270]. All these approaches support the embedding of design intent through expert-curated logic deployment, often combined with optimal structuring of environment and asset placement. Some solutions, such as BTs, are frequently fully controlled by human designers. Others involve designer-guided automation. One such example is the A* search algorithm commonly used for navigation pathfinding in game environments, which requires a navigation mesh asset that is automatically generated on the basis of parameters provided by human designers [198]. To address the increasing complexity and scale of video game playing, AI solutions have seen forays into evolutionary, automated and learning solutions, but few of these can be considered more than an avant-garde, at this time [241]. Game developers continue to rely on well-established, human-defined forward model-based methods, capable of servicing the majority of agent behaviour challenges encountered in contemporary game design. What those solutions lack in terms of algorithmic flexibility is balanced out by ensemble coupling of different types of algorithms and optimal content creation, with an understanding of the gameplay context and the boundaries of technology used [59, 270].

3.3.2 Player Modelling

Player modelling is a specialised subset of user modelling [61, 268], focussed on producing computational models of video game players, based on player characteristics and gameplay telemetry data generated from player interactions with a game environment [64]. Player characteristics, manifested through cognitive, affective, and behavioural patterns, undergo a process of detection, monitoring and interpretation [7, 63, 173]. By mathematically function mapping them against gameplay interactions, player models enable dynamic prediction of future player states and actions from previously unseen data [69]. Models can be generated for an individual player, as well as archetypes known as play personas, representing a cluster of players manifesting common characteristics [33, 249]. Player modelling plays a vital role in the interdisciplinary fields of games user research and game analytics, derived from data science [64, 68, 236]. AI techniques have enhanced player modelling, supporting processing of high volume and multi-layered telemetry datasets with supervised and unsupervised ML approaches [190, 270]. Models generated by applying unsupervised learning to quantitative datasets can provide an insightful representation of the players of the game. They might even contain information that would not be identifiable in top-down, model-based approaches commonly guided by the intuition of a human analyst [69].

Inference capabilities and the form of a standalone player representation make computational models a useful tool for data-driven enhancement of the game production process pre- and post-launch, as well as dynamic game balancing in runtime. They have already found applications in predicting player experience [91, 220], behaviour [61, 62, 150] and gameplay performance [204]. A common use case has also been personalising and creating new game content through Procedural Content Generation (PCG) [188, 221], and adjusting gameplay flow or game difficulty to accommodate distinct types of players [162, 167, 281]. Player modelling has found extensive use in online multiplayer games. Microsoft has been using the *TrueSkill* solution in games created by their internal studios to facilitate data-driven, skill-based matchmaking [98, 161]. Player behaviour monitoring to detect cheating or toxic interactions on the basis of supervised learning models was applied in *For Honor* (Ubisoft Montreal, 2017) [34]. Proliferation of Games-as-a-Service (GaaS) business model[1] has encouraged game studios to actively pursue data-driven

[1] GaaS model considers game release as an intermediate stage, rather than a closing act of the game's production.

solutions in their development pipelines, based on big data mining and analytics, leading to player profiling and modelling, and subsequent fine-tuning of games and their design towards identified user behaviours and habits [64, 246].

3.3.3 Automated Playtesting

Automated playtesting is a variant of game-playing AI, commonly considered a part of AI-assisted design [235, 236, 277]. Conventional game playtesting is a time-consuming and expensive process of quality assurance (QA) executed manually by human testers. It involves functional, exploratory, and experience testing in order to validate the game and detect any outstanding technical or quality issues. The increasing complexity and scale of modern game environments have rendered such a traditional approach insufficient [1]. Game-playing AI for automated playtesting is intended to procedurally imitate human players playing the game in a flexible, scalable, and cost-effective manner [270]. This translates well into a process of training agent-based AI to execute human-like gameplay [23].

Automation of playtesting can be as simple as making an agent, or multiple agents [93, 152], follow a fixed sequence of actions, or as sophisticated as deploying a curiosity-driven algorithm [219] to control an adaptive agent's exploration of the game's environment [229]. While in many cases simpler, scripting or rule-based approaches are applied [237] more intricate automation of playtesting can be traced as far as 1999, when AI players were configured to play against each other in the strategy game *Sid Meier's Alpha Centauri* (SMAC, 1999) [25]. Despite its advantages and potential to scale up testing efforts [218], integration of automated playtesting into industry workflows has only picked up speed over the last several years [233]. Ad hoc, scripted approaches have been augmented with search-based methods driven by such algorithms as A* [229, 277] and MCTS [104, 168]. Recent embrace of data-driven techniques in game development [64,69] have encouraged adopting data-based approaches to automated playtesting, incorporating player modelling, and even learning [12, 85].

3.4 Behaviour Trees

The state of the practice ad hoc behaviour modelling solution for video game AI is the BT. BT is a stateless data structure, based on a directed, rooted tree flow processing. It updates the relevant traversal path through a tree during a single compute frame, at a regular time interval and invokes logic embedded within traversed tree nodes [45]. The transparent structure of BTs effectively generalises the concepts of subsumption architecture, sequential composition, and decision trees [27,28,244]. A BT can be defined as $T =< N, E, \tau >$, where N is the set of all the

Further monetization and development efforts are undertaken by updating the game with free or paid downloadable content (DLC) updates and offering varied subscription services.

tree nodes, E is the set of tree node connecting edges, and $\tau \in N$ is the tree's root node [275]. The root has no parents and only one child node. Other nodes always have a parent node and may contain child nodes. The update "tick" signal is propagated by parent nodes to their child nodes, according to a specific node type propagation logic. Every node has a behaviour associated with it, which represents the logic executed when the node is updated. Node behaviour execution yields a success, failure or a still-running response, reported to the node's parent. The running result indicates that the node's behaviour logic requires additional computation frames to complete its execution. Nodes with children, called composites, embed composite behaviours, which affect the tree traversal. Commonly used composites include *sequences* (executes all children nodes, until one fails) and *selectors*[2] (executes all children nodes, until one succeeds or continues running). Childless, tree-terminating nodes, called *leaf* nodes, carry action behaviours, which constitute low-level logic that affects the game environment. BTs also feature *condition* nodes for testing specific conditions to affect the tree traversal, and *decorator* nodes, which combined with other types of nodes inject additional functionality, on top of the base node's behaviour (a common *decorator* example is an *inverter*, which negates the result returned by the base node). Advanced BT features also include *parallel* nodes for splitting the update signal to support a multi-path tree traversal, nesting subtrees within trees, dynamic runtime tree linking and data sharing resources, across nodes in a tree [36].

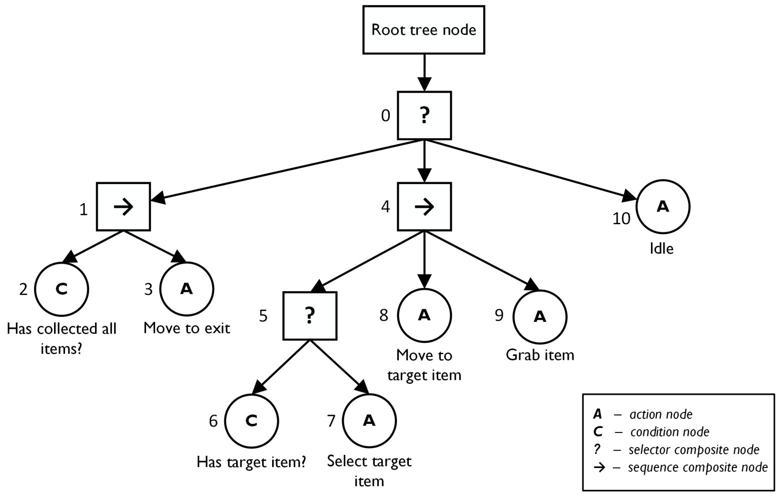


Figure 3.1: An example of a game-playing AI behaviour tree. BT instructs the agent to collect all items in the environment. When no items are left, the agent will move to the exit. The order of node execution is indicated by node numbering.

Since their first documented use for agent AI in the early instalments of the *Halo* series (Bungie, 2001-2004), BTs have become the de facto standard for implementing agent behaviours in video games, superseding FSMs as the ad hoc method of choice for many game developers [44]. Characterised by their modularity, reactivity, scalability, and readability, BTs also found applications

[2]Also known as *fallback* nodes.

in robotics, simulations, and dialogue tree implementations [15, 45, 110, 111]. Their robustness and safety were observed through years of the game industry's research and development work [36, 197] as well as their formal investigation by academics [42, 45, 234]. These features and their capacity to function as a rule-based system for knowledge representation [15] have encouraged experiments with dynamic restructuring and synthesis of BTs. This line of inquiry was carried out by both game industry AI experts and the academia [146, 179, 270]. Academic researchers focussed on automation and using evolutionary techniques, such as genetic programming (GP), to automatically or semi-automatically generate and select improved versions of BTs for robot manipulation tasks, filling in the gaps resulting from the limits and risks of manual behaviour authoring [15, 110]. The modern embrace of ML shifted automated BT creation towards learning from demonstration (LfD) and learning from observation (LfO) using traces from human experts to generate new BT structures [212], improve existing trees with follow-up user demonstrations [110, 111], or to identify optimal tree execution paths [57]. In these approaches, the involvement of human designers has been restricted to producing reliable data for BT generation and providing critical assessment, as well as potential editing, of the generated structures. Previous attempts at automatic generation found limited success in practice [275], partially due to the fact that developers versed in manual labour and a high level of control over their output are likely to be sceptical of automation and modelling approaches which obfuscate design transparency [194, 270].

3.5 Learning for Video Game Playing

3.5.1 What is Learning?

An alternative, procedural approach to forward model-based, symbolic solutions widely used in games AI is to generate a model without explicitly defining how to do it. This can be achieved with ML. Tom Mitchell proposed a definition for learning, which states that "(...) a computer program is said to learn from experience *E* with respect to some class of tasks *T* and performance measure *P*, if its performance on *T*, as measured by *P*, improves with experience *E*" [163]. Thus, more experience leads to improved problem solving in ML. Experience accumulation varies for different types of ML tasks. Supervised learning uses labelled training data to create a mathematical function mapping between input values and output variables to solve problems such as classification and regression. Unsupervised learning operates on unlabeled data to establish a structure of information contained within the data, tackling issues such as clustering and dimensionality reduction. Combinations of the two exist to execute semi-supervised learning, which is capable of handling partially labelled data [82]. Another approach is learning based on experience, known as RL, where a software agent explores an environment using permitted actions in a trial-and-error fashion, receives feedback on its performance from a reward function, and learns to maximise its future reward [17]. Learning can be conducted online to adapt to the flow of new data from the environment incrementally and dynamically, or offline to train models using previously collected datasets, with more flexibility in terms of time and resources involved [160]. There are many algorithmic approaches to ML, applicable in different contexts. In the last decade, statistical methods have been overtaken by the neurology-inspired ANNs, whose scaling with the use of modern hardware has allowed generating complex structures with hidden layers, providing a significant increase in power and processing capabilities. This advancement introduced a new learning paradigm called Deep Learning (DL), where a neural network's depth is determined by the number of hidden layers it contains [87]. It has been key to making ML a valuable tool for a wide range of research, engineering, and business projects.

3.5.2 Learning and Autonomous Agents

Agent representation was originally envisioned in the form of the Von Neumann machine, a theoretical machine capable of pursuing the goal of self-reproduction [257]. The concept was later built upon by John Conway to develop the cellular automaton known as the *Game of Life* [80]. It simulates individual cells pursuing survival, which are interacting with the environment in which they exist. This metaphor of embedding intelligence within an individual object residing in an environment, which can exhibit agency and reactivity, gained popularity across different fields, including games. The objective of an agent-based AI is to simulate a system, present in an environment and capable of collecting data from it, which is then able to determine what actions to perform to fulfil its goals, given the environment's state [149]. In contrast to top-down, systemic AI, which attempts to simulate larger scale phenomena, agent AI is focussed on a bottom-up approach of representing a single instance of an object with agency [160]. While agent AI in commercial games commonly serves the gameplay and is meant to enrich the experience provided by the game, research into AI for games has been more concerned with producing agents capable of playing games, in lieu of a human player [270]. Researchers have been experimenting with integrating learning solutions to support that goal in many different ways, but perhaps the most relevant approaches for developing agent-based AI are RL and IL.

Reinforcement Learning

The original RL work by Arthur Samuel investigated a self-learning system, which was effectively an agent playing against another version of itself to improve and win in a game of checkers [214]. The nature of RL, and the string of recent successes of RL-based game-playing agent AI [166, 256, 264] made it a go-to technique for simulating agents, operating in an uncertain environment. The basic premise of RL is to maximise the agent's accumulated reward, consistent with the agent's goals, by exploring an environment through interaction, and balancing it with exploitation of actions the agent had successfully tried before. Learning from interaction

to achieve a goal can be modelled as a Markov Decision Process (MDP), featuring an agent interacting with an environment in a sequence of discrete time steps t. At each time step, the agent perceives an environment state S_t and then selects an action A from the set of available actions A_{S_t} to perform it. In the next time step, a quantified, positive or negative, reward R_{t+1} is given to the agent by the *reward function R(S,A)*, and an updated state of the environment S_{t+1} is observed. The agent's behaviour is defined as *policy* π, which maps observed states of the environment to the agent's actions and the probability of their execution. The policy is updated on the basis of reward signals, which can encourage or discourage specific actions. Reward signals are immediate, while a reward function predicts potential rewards in the long run, given the state the agent finds itself in at a given time [238]. It is, in fact, the long-term cumulative reward that an optimal policy, learned via RL, is expected to maximise. A policy can be trained *online* by directly interacting with the target environment and acquiring experience for immediate consumption, or *offline* to utilise a previously collected batch of data.
The basic premise of RL has evolved over many years of active research. One of the key developments introduced was the *temporal-difference* (TD) learning, which combined a Monte Carlo approach and dynamic programming to learn from experience in solving prediction problems. TD methods are capable of updating their value estimations based on other learned estimates, with no prior environment model [238]. Using TD together with neural networks has led to the exploration of Deep Reinforcement Learning (DRL) [165], which has delivered unprecedented results in many scenarios, including complex game environments [15, 77, 97, 264] with multi-agent setups [192, 256]. However, for effective operations, both RL and DRL require a substantial amount of training time and data [99], establishing an optimal reward function, as well as addressing the problems of long-horizon decision-making and large decision spaces with sparse rewards [122]. Despite increasing compute budgets and attempts at optimising RL with a variety of approaches, such as reward function adjustment [121, 129, 217], and task complexity decomposition via Hierarchical Reinforcement Learning (HRL) [95, 183, 185], these issues remain an ongoing challenge [119].

Imitation Learning

Some RL tasks, especially those with sparse rewards and long-time horizons, require manual adjustment of the reward function to achieve the desired behaviour execution by a RL agent. This can be complicated and difficult. However, if the end behaviour can be demonstrated by a human expert, or an alternative agent model [94], the agent can then learn an optimal policy by imitating the expert's steps, with consideration of all steps taken, instead of just individual actions [243]. This approach, known as IL, was inspired by the capacity of animals to imitate each other's behaviours [79] and the activity of mirror neurons, which are involved both during the learning of a task, as well as its execution [135]. The concept of imitation was effectively applied in robotics using the notion of high-level representations known as movement primitives to encode temporal behaviour [74]. Just like in RL, the environment in IL is modelled as an MDP. Instead of operating with an explicit reward function, an imitating agent learns from a set of expert demonstrations $\tau = (s_0, a_0, s_1, a_1, ..., s_n, a_n)$ recorded as trajectories representing the optimal policy *pi**. This can either be achieved by direct imitation of the expert policy, or by learning its reward function to indirectly imitate the expert behaviour [117].

Direct learning is commonly facilitated using Behavioural Cloning (BC), which is based on training the model in a supervised manner to maximise distribution matching of the expert trajectory data. It achieves an optimal policy by mapping state-action pairs from the expert demonstrations to the control input [181]. The simplicity and effectiveness of BC have made it a popular method for IL tasks, despite the fact that it suffers from the compounding errors problem, originating in sequenced state deviations increasing over time. This may eventually lead to an agent's mistake, which could put it in a state that was not covered by the expert data, and result in undefined behaviour [266]. This is especially true for large spaces that require long-term planning. Interactive demonstrator algorithms, such as the Dataset Aggregation (DAgger), incorporate additional expert feedback for each state to produce coping strategies in error scenarios [205]. Directly learned policy is valid for as long as the learned action-state mappings remain valid in the context of the environment. [181].

Indirect learning extracts the reward function from the expert demonstrations via Inverse Reinforcement Learning (IRL) under the assumption that behaviour trajectories examined are approximately optimal with respect to the targeted reward function [175]. For a set of expert demonstrations $D = (\tau_1, \tau_2, ..., \tau_n)$ and the reward function $R(S_t, A_t)$ policy π_n is learned via RL in each step of an iterative process of fine-tuning reward function parameters, until π_n approximates the expert policy $\pi*$ [108]. Well trained IRL models have the potential to generalise well to unseen scenarios and may even outperform the expert in large state spaces at the cost of increased complexity and computation load [181]. More recently, Ho & Ermon proposed Generative Adversarial Imitation Learning (GAIL), which combines the properties of Generative Adversarial Networks (GANs) with an IRL like approach to constrain the behaviour of an agent to be approximately optimal with respect to an unknown reward function, without recovering that function. GAIL applies a discriminator D to differentiate between state-action pairs originating from the expert trajectories, and a generator G to imitate the expert policy by maximising reward signals from the discriminator. The procedure follows a minimax optimisation approach, which seeks to achieve an optimum $D*$ that would minimise the state-action distribution discrepancy between the expert policy $\pi*$ and generated policy π, in terms of the Jensen-Shannon divergence [101]. It is potentially able to service large, high-dimensional environments better than BC, which has a quadratic dependency on the planning horizon, whereas GAIL's dependency is linear. It also performs well with limited or incomplete trajectories but can be considered mode-seeking and thus difficult to train in certain scenarios [259, 266].

3.5.3 Policy Context Decomposition

Humans intuitively approach planning for task execution as a process of breaking down tasks into actionable subgoals, often extending far into the future [103]. Autonomous learning agents are expected to do the same, to be able to effectively operate in complex environments and solve challenges that warrant executing long sequences of actions [95]. Despite an increase in available compute budgets, currently used solutions have a limited task horizon and generally do not scale well to large state and action spaces. This results in suboptimal performance of standard algorithms in such scenarios unless additional policy context decomposition techniques are involved [185].

Attempts to address this issue through automated decomposition of long-horizon tasks into a hierarchy of subproblems to be solved in shorter time horizons have been made with HRL [200]. The hierarchical structure generated by HRL exhibits a temporal abstraction property that manifests in higher-level policy execution via higher-level actions, which persist for longer periods of time. Lower-level policy, on the other hand, is relegated to a shorter horizon, lower-level actions [185]. Thus, HRL solves a complex problem by decomposing it into simpler subproblems to be solved. This approach has been proven to outperform standard RL in a variety of scenarios, due to achieving more efficient exploration through subtask granularity [51, 171]. Further HRL improvements have been investigated, such as options in HRL and feudal learning. The former works to optimise the HRL hierarchy by generalising tasks into reusable, functional archetypes that other actions would be able to execute [285]. The latter utilises the concept of decoupling end-to-end learning and taking advantage of a manager-submanager relationship, where submanagers receive commands from managers and service them by maximising their learning in the context of the given command [54,254]. Alternative approaches to leveraging a macro-micro hierarchical structuring have also been investigated [48, 142], indicating a consistent interest in an intuitive task decomposition, provided by context-relevant, hierarchical structuring.

While in many cases generalist, large-scale ML models are preferred [172], smaller-scale, specialised models, trained or fine-tuned to local context, may service scenarios with specific runtime and handling requirements better [47]. The decomposition strategy for such cases is often based on deploying an ensemble of small-scale models [60,170,172]. Their strategically limited scope enables the shrinking of the action space to afford a more efficient learning [129].

3.5.4 General Game Playing

In the course of the Dagstuhl 2012 Artificial and Computational Intelligence in Games seminar participating game AI researchers identified general game AI, or GGP, as one of the key research directions for game AI [269]. GGP operates under the assumption that games share common characteristics, which can be discovered with the use of generalist learning algorithms. As such, GGP is not intended to target the local context of any particular game. It learns gameplay policy from high-level inputs, such as screen scraping and general interaction with the virtual environment through external interfaces, rather than taking advantage of internal Application Programming Interfaces (API), and the full observability of a game state [247, 270, 277]. In a number of cases, RL-based learning in GGP was reinforced with IL methods to bootstrap policy learning and provide human styling to gameplay behaviour [15, 256].

Many highly publicised, recent successes of game-playing AI were the product of GGP. Experiments with universally applicable GGP solutions in real game environments were made possible with AI testbeds, such as the *VizDoom* framework for *Doom* (id Software, 1993) [132], *Pogamut* software for *Unreal Tournament 2004* (Epic Games, 2004) [81], which spawned the *2K BotPrize* competition, *Project Malmo* for *Minecraft* [115] and the release of the OpenAI *Gym* framework [26]. Significant milestones in the field included two AI controlled bots passing a game-based Turing test[3] in *Unreal Tournament 2004* in 2012 [216, 270], DeepMind's model-free DRL algorithms mastering a suite of classic Atari 2600 console games solely from raw pixel input in 2013 [165, 166, 215], DeepMind's *AlphaGo* beating a Go world champion in 2016 [231, 232][4] and DeepMind's *AlphaStar* reaching grandmaster level in a specific setup of the highly complex real time strategy game *StarCraft II* (Blizzard Entertainment, 2010) [256]. Similar architectures allowed OpenAI *Five* to excel at a ten-player multiplayer online battle arena (MOBA) game *Dota 2* (Valve Corporation, 2013) in 2019 [15].

Although GGP is a promising research direction with valuable output that could support further work towards Artificial General Intelligence (AGI) [246], it is unlikely to be applicable in commercial game runtime environments within a reasonable time horizon. Reasons for this include high deployment costs and incompatibilities with established industry workflows [194, 202, 277].

3.5.5 State of Practice

Reluctance

Design and production of video games is not an easy undertaking. Modern game development cycles can take anywhere from a few months to many years, depending on project scope. Increasing complexity and budgets create constraints in the adoption of new, untested solutions in game projects, and large multidisciplinary teams involved make human factor a key consideration when planning long-term development [194, 211]. Industry standard behaviour authoring methods generate human-readable and easily modifiable AI representations, allowing for rapid change and iterative, incremental development [111]. Despite a wish for "emergent" behaviours and adaptive properties of behaviour AI [25, 146], game designers are invested in retaining au-

[3] A variant of the classical Turing test, in which game playing behaviour is observed by a panel of individuals, who must correctly identify it as generated by a human or an AI.

[4] *AlphaGo* combined RL, DL and Monte Carlo Tree Search (MCTS)

thorial control over behaviour logic and achieving stable, deterministic output to facilitate testing and debugging of game environments [122]. Game developers are not interested in AI that is left to its own devices, but rather AI whose operations they can understand and whose processing they can guide in a desired direction [270]. These requirements are currently not directly addressed by standard ML models, whose monolithic representations, as well as high training data and time costs, are not compatible with the iterative development workflows of game AI [8, 241]. Although video games are not critical systems, and the severity of bugs they exhibit might seem trivial, there is a high degree of scrutiny of technical issues from the audiences and critics. Problematic AI immediately stands out and can easily break the player's sense of immersion, negatively impacting their play experience [143]. Automating AI development for users, who are used to manual labour and deterministic output with well-established toolsets, is another factor that provokes suspicion and scepticism in embracing non-transparent model representations [194].

Learning for Runtime Logic

Despite these issues, adaptive learning solutions were applied for runtime behaviour AI in a limited number of commercial games. Notable and documented cases included training of agent behaviour models through players' feedback in the *Creatures* (Millennium Interactive, 1996) [89], and later in *Black & White* (Lionhead Studios, 2001) [279]. Strategy games have experimented with ANNs in a variety of ways, such as the use of multi-layer perceptrons for the unit reaction system in *Supreme Commander 2* (Gas Powered Games, 2010) [279]. The longest-running learning solution in a commercial title is the *Drivatar* system, which has been deployed in the *Forza Motorsport* series (Microsoft Game Studios, 2005-2024) to imitate driving behaviours of human players through generating their artificial personas [105]. More recently, the developers of *Gran Turismo Sport* (Polyphony Digital, 2017) have succeeded in developing an AI capable of outperforming champion-level human players using off-policy, model-free DRL trained with full observability of the game state [264]. AI behaviours in *Age of Empires IV* (Relic Entertainment, 2021) and *Halo Infinite* (343 Industries, 2021) were also trained with the use of ML methods [241], and *Little Learning Machines* (Transitional Forms, 2023) allows players to personally oversee the training of their RL-driven virtual companions [32]. While a few other examples of games using learning in some capacity exist [279], thousands of new games are released every year, and only a handful of commercial titles have featured applied learning for runtime logic in the last 30 years. Contemporary industry work towards learning integrations has been mostly concerned with expanding AI for game production toolsets [270]. Some interest in ML agent development stems from the latter, as it covers automated game playtesting, which has already proven to improve the quality and speed of game production [23, 97, 112, 194, 229]. Further experimentation in that area may intensify due to the recent integration of RL agent interfaces in widely used game engines, such as Unity [55, 118, 140, 144, 271, 278] and Godot [11, 134, 207].

Learning for Production Workflows

While runtime behaviours generated with learning models are not common in commercial games, industry production workflows have been extensively using ML-based methods for many years. Game analytics and player modelling using large-scale datasets and ML-based processing are now a common feature of production toolsets for games [64, 102]. This is especially the case in online multiplayer games where data-driven solutions have provided scalability to skill-based matchmaking via systems like the *TrueSkill* [98, 161], as well as anomaly detection in player behaviours in competitive games, such as *For Honor* [34] and *Counter-Strike Global Offensive* (Valve, 2012) [241], to identify instances of toxic interactions and cheating. Visual improvements in games, ranging from automated upscaling of low resolution graphical assets, which assisted the development team of *Dead Space* [85], sophisticated animation blending and controlling solutions found in AAA games[5] of the likes of *Hitman* (IO Interactive, 2016), *For Honor*, and *FIFA 22* (EA Vancouver, 2021), to real time upscaling of display using NVIDIA's Deep Learning Super Sampling can already be considered industry standards [241]. The same can be said about offline methods for player profiling and modelling, and subsequent fine-tuning of games and their design towards user behaviours and habits [64, 131, 150, 225, 246]. Procedural approach to expanding game development toolsets with AI assisted design [22, 277] and algorithmically generated content [221, 226] has also enjoyed more focus and integration interest, in response to a string of commercially successful titles built around PCG, such as *Spelunky* (Mossmouth, 2008), *Minecraft* (Mojang, 2011), and *No Man's Sky* (Hello Games, 2016) [269]. Combined with offline learning, it offers an industry-relevant scenario where output content can be validated with high confidence, at production time, to ensure quality guarantees of the in-game content.

The widespread adoption of design time learning solutions, and their integration into existing game development workflows is a testament to the capacity of the game industry to swiftly invest and deploy technologies that provide a concrete benefit to the game production process [221, 270].

Learning for Automated Playtesting

Automated playtesting is a field that could benefit from applied learning [184]. While it may operate in runtime conditions of a game, it is technically part of the production workflow and is not necessarily constrained by the same factors as the game's environment, targeted towards user-grade hardware. While there have been a number of interesting, academic projects in the

[5]AAA or triple-A is a nomenclature derived from the credit industry's bond ratings [16] denoting large video game publishers or studios producing high budget games, often comparable to those of Hollywood blockbuster film budgets. AAA video games tend to have a high level of polish and a large amount of assets, due to big development teams involved (hundreds or even thousands of developers). Independent or indie studios are lean development teams, sometimes individuals, who create small to medium sized video games with focus on experimentation and innovation. In many ways their approach is comparable to that of independent filmmakers and musicians.

area [6,104,282], they have had a limited influence on the state of the practice methods applied in the industry. The primary reason is that most of them utilise simple scenarios, which do not correspond to the complexity of environments found in commercial games [85]. However, the industry has also been active in the area. Research and development (R&D) groups in larger video game companies have been using their accumulated, massive user gameplay telemetry and behaviour datasets for automated playtesting projects using learning solutions, but formal documentation of their efforts is relatively sparse. Only recently, some of these R&D teams started sharing their results more openly. Electronic Arts (EA) has been especially active in presenting their work towards automated game playing and playtesting agents trained with the use of ML. Some of the games they investigated included the *FIFA* series (EA, 1993-2022) [71], *The Sims Mobile* (Maxis, 2017-2019) [25,229,277], *Battlefield 1* (DICE, 2016) [97,210], *Battlefield 2042* (DICE, 2021) and *Dead Space* (Motive Studio, 2023) [85], *Plants vs. Zombies 2* (PopCap Games, 2013) [13], as well as unnamed open-world games [21,22,24,218,219]. Other game industry entities which have published their research include Ubisoft [134,207,250], Sony [147, 264], Ninja Theory [56, 283], Microsoft [1, 158, 186] Tencent [107, 260], and King [92]. It is highly likely that other parties in the game industry are working towards developing automated playtesting infrastructures, but the commercial nature of these projects prevents those involved from sharing their results prematurely.

Industry work in the field of learning agents for playtesting has explored a variety of techniques in production environments, targeting both RL and IL methods. RL is usually applied in the form of DRL to maximise the capacity of a trained agent [12,134,264,277], but IL is preferred when human-like styling is desired, or large demonstration datasets are available [21,22,71,107,219]. Method selection is also based on the agent's target mode of operations, as an agent can exhibit play-to-win behaviours, human-like gameplay behaviours, or maximised environment exploration and bug searching behaviours [21,277].

RL solutions have been proven to be effective and scalable in automated playtesting scenarios for game environments [85]. However, their training carries a high compute and time cost, especially in large-scale DRL, and should ideally be conducted offline to respect realistic budget, infrastructure, and workflow constraints [134]. Further optimisations such as model decomposition to streamline training [147] and multi-task training approaches to improve model transferability and reuse [260] were found to provide additional advantages for industry-grade RL solutions. Multiple projects took advantage of the on-policy Proximal Policy Optimisation (PPO) algorithm, which provides stability [12, 13, 260] and steady improvement on IL bootstrapped policy [84].

IL methods were found to be suitable for solving complex problems in game environments, in a human-like manner, but standard algorithms such as BC have shown their limitations in large state spaces [71]. IL can also experience issues with behaviour generalisation, resulting from limited or biased input data [22]. This can be mitigated by using a more sophisticated data-driven approach, involving larger demonstration datasets and an advanced algorithm, such as GAIL [218]. Combining IL and RL can lead to even more promising outputs, especially if demonstrations are used to bootstrap reinforcement training. This results in faster and more accurate RL training, introduces styling and maximises agent exploration behaviours [21,24,97].

While the majority of the industry work examined takes advantage of direct access to game state data, some teams investigated generalist approaches using high-level pixel inputs [1, 186]. This is unlike research conducted outside the industry, which is more inclined to pursue GGP methods [15, 256]. Since the discussed research was conducted in game production environments, industry workflow and practical considerations were of utmost importance [134]. This led to a determination that learning is not suitable to address all gameplay testing scenarios [12]. In fact, learning techniques were found to achieve better results when combined with heuristic or otherwise script-based AI [13, 24], especially when an existing solution was strategically augmented with learning capabilities [85].

3.5.6 Combining Learning and Ad Hoc Techniques

Output learning models are black boxes, and iteratively evolving their underlying logic often requires full model retraining [194, 274]. This severely limits their immediate applicability in fields where flexible control, modularity and transparency are preferred, as is the case with video game agent AI. Work on hierarchical model structuring, such as HRL, has led to partially optimising output behaviour policy complexity by decoupling long-term decision-making and action learning [138, 185], but it has not resulted in a solution to the model transparency and control issues that are relevant for game developers. However, games and robotics-focused research have shown that these issues could be alleviated by embedding learning within manually authored ad hoc control structures, such as BTs [15, 212, 274]. This integration approach was initially investigated from a holistic perspective of learning complete machine segments of Hierarchical Finite State Machines (HFSM) [265], and BT subtrees [43, 146], as well as optimising existing BTs using predefined constraints [176]. However, a more direct and effective solution to embedding a specific learning capacity within an ad hoc structure was established in the form of a neural network controller, represented by an ad hoc compatible learning component. In BTs such a structure can be expressed as a tree *learning node*, as proposed by de Pontes Pereira & Engel [285]. It allows the inherent robustness and modularity of BTs to be partially transferred to the contained learning models [234]. First investigations towards such embeddings originated in robotics [45, 244]. Experimentation in game environments followed soon after and involved investigations into replacing BT composite nodes with their RL policy driven equivalents for learned execution path selection [57, 76], as well as using leaf nodes containing RL neural network controllers to facilitate specific low-level actions [285].

Follow-up investigations took advantage of the capacity of the learning node to accommodate different policy learning techniques and mix them according to local context requirements. Zhang *et al.* proposed the use of a hybrid method of LfO and experiential learning to generate agents, who were able to learn tactical skills from both expert trajectories and trial and error experimentation, to be used in the domain of Computer Generated Forces (CGFs) [274]. *Trained-query nodes* were discussed by Sagredo-Olivenza *et al.* for robust incorporation of models learned from human expert trajectories into BTs [211, 212]. Similarly, a data-driven framework for IL-based model integration was deployed by Buche *et al.* with the intent to be used in practice [29,30]. Li *et al.* explored a DRL approach using the options framework to train an agent to operate in a Unity 3D environment [144]. A thorough, practical use case involving macro, strategic action extraction from expert demonstrations to reduce action space. Later, embedding micro, tactical actions trained using RL to competently play *Starcraft 2* was presented by Pang *et al.* [183].
While the learning node has proven to be an effective way of embedding learning in ad hoc structures, its recurring implementation suffers from two issues: it offers limited control over the learning node's execution and provides no controllable resolution should the node fail to execute according to designer's expectations. Failure of execution may be defined as producing invalid output, such as undesired behaviour, or violating execution safety, for instance, because of performance issues, potentially compromising the execution flow of the entire agent behaviour logic. Sprague & Ögren [234] presented and formally documented a safe structural approach to tackle both problems by embedding learning logic in a subtree, featuring additional ad hoc redundancy to safeguard learning logic execution and performance. Such fallback structuring is necessary in the context of game industry workflow requirements, as it provides a deterministic, worst-case behaviour execution scenario, which can be fully controlled by game designers. To our knowledge, this model of learning integration has not been applied and tested in industry workflows.

3.6 Digest

Game developers and AI researchers have pursued human-like and adaptive agent AI behaviours alike in an effort to enhance player experience and solve complex challenges presented by virtual environments [100]. Ad hoc authoring methods used in commercial video game development are able to address these challenges within certain parameters, but their shortcomings become apparent with scale. This warrants specific optimisations to maintain an approximated illusion of agent intelligence with respect to the context of desired gameplay experience [197, 247], as well as technical constraints of consumer-grade hardware that games are expected to run on [134]. While recent advancements in application of learning techniques in game playing AI [15, 231, 256, 264] have demonstrated a potential to create learning models capable of generating competent behaviours in complex video game environments, the high cost of their training and incompatibility with the domain requirements and industry workflows make them unsuitable for professional game development at this time [202,247]. This is not likely to change in the near future, as research motivations and priorities in learning model development differ from those of the industry, as indicated by the AI research "gap" [189, 270]. Non-industry research into game-playing AI has been mostly concerned with operational efficiency and problem-solving in an attempt to competently tackle complex puzzles embodied by different game environments [165, 191, 270]. Although research into the believability of video game-playing AI is gaining traction, it can still be considered niche [230] in comparison to the popularity of projects with an algorithmic perspective on learning [191]. The incompatibility of high-profile developments in learning with game industry pipelines can also be attributed to a limited transfer of the business and technical domain knowledge of game development to the research community [194, 270]. The experience-driven nature of video games, combined with a high level of player agency and game environment complexity, requires servicing both highly deterministic and controlled behaviours, as well as adaptive, reactive intelligence of agents responding to the continuously changing state of the game [59]. Unpredictable or erroneous AI behaviour execution is undesirable, as it may result in violating the context of the target game environment. This makes industry state-of-the-practice expert curation and authoring an important part of the process of delivering convincing and enticing video game agent behaviour [59,194,211,285]. Compromising execution and context safety guarantees can disrupt the immersion of a game and, in consequence, the player experience, failing to deliver on a promise and an illusion of a believable game world [196]. If learning techniques are to become applicable in runtime video game playing agent AI, then maximising control over model execution and development is required to ensure these guarantees can be delivered, and unpredictable model outputs minimised. However, additional consideration in terms of architecting the context flow of agent AI behaviours is also relevant, if the cost and flexibility of iterating on learning logic are to be optimised to a level that would be acceptable in commercial video game development. Recent advancements in learning game-playing AI suggest that ensemble approaches, combining different, complementary techniques, produce higher quality outputs [111,222,270]. This direction enabled solutions such as *AlphaGo* (RL, Deep Learning and MCTS) [231], *AlphaStar* (supervised learning and multi-agent RL) [256], and *Gran Turismo Sophy* (DRL and mixed-scenario training) [264] to achieve high performance in their target game environments. In all these cases, human involvement was an important part of the training process and was key to achieving satisfactory and context-relevant outputs. This is consistent with the growing research and practical interest in investigating the semantic and responsibility gap resulting from the increasing human actor detachment from learning system control [31, 73, 153, 228, 258]. Applied use of ML in PCG and other fields of AI for game production confirms that the industry is ready to accommodate learning in games, provided that its output is controllable and can be scrutinised by a human expert at the time of game development [148,221,226,270]. Some attempts to achieve this in compatibility with the game industry requirements for video game playing AI have already been made. Evolutionary solutions to generating optimal AI model representations, such as BTs based on functional goals [43, 57, 275] and human demonstrations [75, 128, 212, 274], relegate human creators to editing procedurally generated content or providing expert-level training data. An alternative approach is to integrate dedicated learning segments into ad hoc AI representations to take advantage of their well-researched and robust features, along with the domain expertise of their effective use [29,72,211,265,285]. The latter is potentially more applicable in the realities of contemporary game development, as it retains human expert design intent embedding and authorial control, with respect to the established game development AI workflows. However, viable industry case studies are required to demonstrate the actual robustness of any proposed learning integrations [85]. While recent research projects have been investigating this approach, few of them have presented scenarios that would be relevant to the game industry. However, synthesising their findings and expanding the learning node safety redundancies proposed by Sprague & Ögren [234] in a commercial game environment is likely to produce an industry viable solution to safe and robust learning integration in ad hoc behaviour representations.

Chapter 4

Context-Guided Agents

Summary. This chapter establishes the foundations of our research work. It documents the chosen methodological approach, the research procedure applied, and the planned evaluation, followed by the design proposal for integrating learning into video game agent AI. It also discusses the development work to instrument the game environment, and data work involving the collection and processing of the mass-scale gameplay telemetry dataset used in our research work.

4.1 Overview

4.1.1 Goals

In this chapter, we aim to introduce the reader to the principles of our research work and to present its primary output: the industry-applicable context-guided learning agent design and deployment workflow proposal. In-depth discussion of the design and its features highlights original contributions C1 and C2, and provides a response to research question RQ1 from a theoretical perspective:

- **RQ1**: can models with execution and performance guarantees of learning logic be integrated into the game industry state-of-the-practice ad hoc behaviour AI architecture for applied use in video game playing AI?

The application of the proposed design in a commercial game environment is the focus of the remainder of the thesis. To prepare the game environment of *60 Seconds!*, presented in Chapter 2, for our research work, we have instrumented it to support autonomous AI agent operations, improved data collection, and experimental evaluations with human players. We present the scope of the instrumentation in this chapter and outline relevant research outputs produced in the course of our development work, including O1, O2, and O3.

Finally, we discuss the mass-scale gameplay telemetry dataset from the game and document the extent of the work that went into collecting and processing its data samples. Although the dataset was prepared for use in training an instance of a context-guided agent, it was packaged and shared with the research community as part of research contribution C3.

Research contributions and outputs reported in this chapter include:

- **C1**: industry-applicable context-guided learning agent design and deployment workflow proposal.
- **C2**: extension of the BT learning node concept, featuring additional safety redundancies.
- **C3**: a mass-scale, processed gameplay telemetry dataset from the game *60 Seconds!* that was used in our research and later shared with the research community.
- **O1**: implementation of a BT learning node, integrating *Unity Machine Learning Agents* learning capacity into PadaOne Games BT library *Behavior Bricks*.
- **O2**: instrumentation of the *scavenge* segment of the commercial video game *60 Seconds!* for enhanced gameplay telemetry data acquisition, simulating autonomous agent operations, and experimental evaluations with human players.
- **O3**: standalone simulator software for simulating autonomous agents, based on the *scavenge* segment of the *60 Seconds!* gameplay environment.
- **O6**: a published game AI book series *Game AI Uncovered* chapter presenting the context-guided learning agent design and deployment workflow [88].

4.1.2 Structure

The contents of this chapter are divided into the following sections:

- **Assumptions**: outlines the intentions, methodological approach, procedure applied, and the anticipated outcome of our research work.
- **Context-Guided Agent Design**: presents the proposed learning agent design and deployment workflow, providing a theoretical and practical insight into the design proposal.
- **Game Environment Instrumentation**: discusses the scope and challenges of the instrumentation of the game environment introduced in the course of our research work for the purposes of enhanced data acquisition, agent simulation, and experimental evaluations.

- **Gameplay Telemetry Dataset**: details the structure of the mass-scale gameplay telemetry dataset acquired for our research and highlights the data acquisition and processing procedures applied.

- **Conclusions**: summarises the design proposal and the work with the game environment and datasets conducted in the course of this chapter. Finally, it outlines the contributions documented in the chapter and highlights how its output supports the follow-up work in consecutive chapters of the thesis.

4.2 Assumptions

4.2.1 Motivation

Applied learning has been gaining traction across different industrial fields, but the game industry has seen limited adoption of learning. Few commercial games have used learning for agent AI runtime logic, thus far [279]. There are a number of valid reasons for the lack of use of learning models in game agent AI, including: risk averse business perspective of game development studios [270], proven reliability and efficiency of state of the practice ad hoc AI behaviour authoring techniques [111], and reluctance of game designers to embrace non-transparent learning models, which are not compatible with the industry's iterative workflows [194]. However, there is a growing interest in changing this, and research into viable learning integrations has intensified over the last few years. Promising use cases, such as automated game playtesting, are already investigated by academics and the industry [23, 112, 229]. Game engines used in the industry, such as Unity and Godot, have been equipped with ML libraries for conducting RL and IL training for some time [11, 118]. Despite all these conducive factors, there are not enough industry case studies involving real, commercial game environments to promote ML models as an appealing and realistic replacement for ad hoc AI representations. This requires more engagement from the industry. But such commitment is unlikely, unless concerns of game developers towards learning models are addressed in a convincing fashion.

The focus on human authoring control [59] and observing game development workflow requirements guided our choice to take advantage of the state of the practice BTs for video game AI authoring BTs [36, 44, 45]. The use of this established technique guarantees compatibility with industry workflows, and pre-existing expertise of video game AI developers [194, 211, 270, 285]. Its tree structure offers an intuitive solution to integrating learning by encapsulating the learning logic in a dedicated tree node, as confirmed by different research endeavours [30, 57, 72, 211, 234, 274]. To our knowledge, none of these investigations have found applications that would address the requirements of game industry workflows. This can be either attributed to their research-focused objectives or operating within controlled environments, which did not represent standard game industry workflows [85]. However, we think that many of the ideas explored by prior research, especially the concept of safe neural network controllers integrated into BTs with performance and execution guarantees, as demonstrated by Sprague & Ögren [234], are relevant to achieving an industry viable approach to integrating learning into runtime video game agent AI. This motivated us to contribute an original extension to the structure of a BT learning node, incorporating industry-applicable safety redundancies. It became the key element of the proposed design.

The author's knowledge of the domain and its requirements presented an opportunity to expand on prior work in the field in a meaningful and practical way. This translated into an original design proposal, which could be readily introduced into existing industry workflows, while preserving and taking advantage of the central role and domain experience of a human expert in the creation process of video game agent AI. This approach is of interest to the industry, as confirmed by our design proposal being featured in the recently published game AI book series *Game AI Uncovered* [88], which constitutes research output **O6** of this thesis. It was also imperative for the proposed solution to be evaluated in a data-driven manner, in the format of an industry case study. We assumed it would be best achieved in an environment of a commercial game, developed using state-of-the-practice tools and workflows. This informed the decision to deploy, evaluate, and demonstrate the applicability of the proposed design in a published, commercial game *60 Seconds!*. The selected game environment was instrumented to support inference and training procedures for AI-driven game-playing autonomous agents, developed using gameplay telemetry data-driven player modelling, and off-the-shelf solutions. It was made possible by the game's continued popularity, which contributed to long-term data collection, resulting in a user-generated gameplay telemetry dataset of millions of samples that could be used in our research.

We consider our research to be of potential use to both the industry and academia in future investigations and the development of learning integrations in other game environments. However, our outputs are also likely to have an immediate use value for the target game environment of the game *60 Seconds!*. The context-guided agent trained during our research could be used to automatically playtest newly designed environment scenarios. By observing completion ratio, collection scores, and other values, designers would be able to quantitatively evaluate their work and adjust the target scenario accordingly. Training a suite of agents with behaviour logic based on different player personas would allow for a wider scope of testing with respect to different types of playstyles. Additional development work could lead to integrating player models for supporting AI-assisted design, or even dynamic tailoring of environments to balance them against a particular player's play skill.

4.2.2 Design Principles

In this chapter, we propose a non-disruptive, robust design approach to safe and context-guided integration of small-scale, context-segmented learning models into BT-based behaviour AI architecture of autonomous video game agents.

Key assumptions of the proposed design include:

- **Non-disruptive, robust design**: design that is compatible with the game industry state of the practice ad hoc workflows, without disrupting a game's pre-existing development and runtime environments.
- **Context-guidance**: context is understood here as the context of agent AI behaviour flow; guidance implies human curated authoring, rather than automated generation of the behaviour flow context.
- **Safe integration of learning**: safe integration of learning assumes that it must provide performance and execution guarantees, protecting the underlying architecture from failures that may be caused by an attempted execution of the learning logic. Learning logic is assumed to be modelled in a format that is easy to incorporate into standardised industry workflows, such as learning models expressed as neural networks, accessible in the context of an ad hoc behaviour flow structure.
- **Small-scale, context-segmented learning models**: the scope of learning logic in the design is assumed to be contained through domain expert-driven task decomposition, with respect to the behaviour context, resulting in a segmentation of learning into multiple, smaller models, strategically embedded in the BT.

4.2.3 Methodology

We employed a practical and data-driven perspective, based on assumptions derived from the current state of the field, discussed in Chapter 3: Background, and the conditions and requirements of the game industry. This significantly influenced the selection of our research and development methods, as detailed in the following sections.

Industry Case Study

To address the tangible needs of the game industry, we decided to pursue key deliverables of our research work, with respect to a specific, commercial game environment and its workflow. We chose to demonstrate the deployment of the proposed design in the format of an industry-relevant case study, informed by the real-world domain requirements. The concrete, but abstract, work-oriented and user-centred nature of the proposal for our design and its deployment was consistent with the practical vector of our research and scenario-based design approaches [35,206]. It was also expected to fulfil the requirements of being applicable, replicable and generalisable. The practical approach of employing an industry-relevant case study was reinforced by the author's game industry experience in developing video games and their AI. The perspective of a professional game developer allowed us to approach our work in accordance with industry workflows and with respect to real industry requirements, resolving the common research issue of limited domain expertise encountered in many other academic projects [191].

Not General Game Playing

While GGP is an exciting research field, it is unlikely to lead to advances in commercial video game agent AI in the near future, due to high deployment costs and incompatibility with established industry workflows [194,202,247]. The practical focus of our research was to solve specific challenges in the examined video game environment, while generalising it to be relevant for similar scenarios of the contemporary and near-future game industry. In this instance, a GGP approach would have likely generated additional distractions and failed to produce results applicable in commercial video game production. Instead, we chose to take full advantage of our access to the underlying, discrete world and flow representations of the investigated video game environment. Such an approach may be constrained to the context of the target game environment [246,270], but we mitigated this risk by conducting a requirements analysis with respect to standard industry workflows. Furthermore, the applicable, replicable, and generalisable format of our design ensures the scope and focus of our work does not suffer from limiting our design deployment to a single game environment.

Quantitative Approach

We have chosen to structure our research around quantitative, data-driven measures, supporting its intended, practical perspective:

- Mass-scale, remote crowdsourcing of numerical, gameplay telemetry data.
- Quantifying of gameplay performance, expressed as play skill, observed in the sourced data samples.
- Applying statistical analysis to investigate the sourced dataset.
- Developing and training data-driven learning models using sourced data.

We were invested in exploring a numerical representation of gameplay data acquired directly from the game environment for objective analysis and modelling, as is the case in game analytics [64,102]. Remotely crowdsourcing data from users enabled them to contribute their trajectories by interacting with a familiar game environment in a natural way, in the comfort of their own play setups. Measuring the subjective human experience of users was beyond the scope of our investigation, and so we chose not to acquire additional qualitative data from them [7,64,102,150], due to the risk of it compromising the natural flow of the game environment and adversely affecting the quantitative data collection process. Additionally, it would have likely introduced player self-reporting bias to the sourced data [4, 248], as first-person reporting is susceptible to self-deception and memory limitations [269] resulting from the memory-experience gap [127].

We decided against trimming statistically detectable outliers to counter data non-normality, in order to avoid excluding user-generated scores, which were valid in the context of the game. To avoid inflating the risk of an increased Type I error rate, we decided against variance testing for normality test selection [280]. When normality was desired, but samples violated normality assumptions, we used data extrapolation via bootstrapping m by n resampling [58,66,193], with appropriate sample size estimation for m, and $n = 1499$, as suggested for a 1% margin of error by Davidson & MacKinnon [52]. We also employed bootstrap hypothesis testing. Confidence intervals were established using the percentile method, due to its computing efficiency and "(...) wide applicability combined with near-exact accuracy." [67], in comparison to alternative methods, such as the bias-corrected interval calculation [66, 120].

For value reporting, we used confidence intervals, where possible. Our work also made use of standardised effect size estimation [40], and normalised sample sizing [9,39]. For score distribution distance measurements, we chose to apply the central tendency measure differences. Where normalised measuring of distances between different score distributions was required, we opted to use the Kantorovich distance[1] symmetric method [208,255], which is more generalisable than Jensen-Shannon divergence [136], satisfies the triangle inequality, and can be considered a valid, objective metric, whose values do not suffer from variance, shape or modality changes [182]. It has been used in a variety of fields, including AI research [187, 224].

Approximated Models

The final output of our research work was planned to be a context-guided agent model, based on a player behaviour persona [102] modelling gameplay behaviours sampled from a set of gameplay telemetry samples, originating from a selected group of players.
Virtual game environments provide abstracted depictions of the real world [137, 223]. Even serious games that attempt to accurately simulate it are at best approximated models of reality [2]. This implies that gameplay telemetry data generated in games and any output that is generated on the basis of such data are also approximated. Thus, the choice to use a game environment in our research committed our work and any results it would produce to embrace the approximation of the world as delivered by the game. Additional approximations were likely to be introduced

[1]Also known as the Earth Mover's Distance or the Wasserstein distance

in the course of the development work on the game instrumentation, as the sensory interaction of agents with the game environment was bound to differ from that of a human player. We did not consider embracing approximations as a constraint, as this approach supports consistency between the output generated in the course of our research and the game environment, in which it can be competently evaluated. It is also consistent with similar work done in the field [33,249]. Additionally, we did not intend to target critical applications in our research, making the focus on approximated models reasonable and acceptable in the context of our work. However, if our outputs were to be used outside of the context of the investigated video game, their approximated nature would have to be appropriately accounted for.

Experiments in the Game Environment

For the evaluation of our trained agents, we chose to conduct experimental evaluations in the game environment. This enabled us to take advantage of the instrumented game environment for the purposes of simulating AI-driven agent operations in evaluation scenarios, as well as the game's data collection pipeline for recording evaluation gameplay trajectories. Additionally, this created an opportunity to conduct online evaluations with real, human players. By deploying evaluation scenarios in the live version of the game and remotely sourcing trajectories from participating users, the experiment was contained in the familiar context of the game's environment. Users were able to participate in the evaluation in the comfort of their own play setups, without being distracted by the elements of a controlled experiment setup or the presence of an evaluator.

4.2.4 Research Procedure

Overview

The line of inquiry of the planned research was divided into a sequence of steps, incorporating practical, analysis and evaluation work, following the outlined methodological assumptions and incremental progress towards reaching our research goals:

- Context-guided agent design
- Game environment instrumentation
- Gameplay data collection and processing
- Game score study
- Agent study

Context-Guided Agent Design

We first defined the scope of the proposed design and outlined its features to establish the theoretical foundation and a starting point for our work, which informed the structure and procedures involved in each of the consecutive research steps. The design proposal was conceived with respect to prior work in the field, explored in Chapter 3: Background, and to achieve the envisioned research and industrial objectives. The procedure for generating the design involved:

- Defining the requirements for the design, derived from our research assumptions and the domain context.
- Architecting the structure and flow of the design proposal, based on the integration of learning models into subsumption architecture, generalised by BTs.
- Establishing the theoretical grounding of the key design characteristics: context-guidance, small-scale learning, safe learning integration and agent perception.
- Documenting a deployment workflow for the proposed design, compatible with the state of the practice industry methods, the design itself, and our research goals.

Game Environment Instrumentation

We then moved to the preliminary, practical stage of preparing the environment for our research work. We examined the game environment targeted for the deployment of the proposed design and instrumented it to fulfil the requirements of our research:

- Refining the game's data collection pipeline scope and functionality, based on the conclusions drawn from the review and work with the pre-existing data collection pipeline.
- Instrumenting the game environment to support training and simulation of autonomous agents, controlled by AI through BTs and learning models.
- Implementing experimental evaluation flow for online evaluations with human users, and offline evaluations with agents.

Data Collection and Processing

In parallel to the instrumentation of the game environment, we had begun our practical work on the data pipeline. This involved both refining pre-existing functionality and developing a data processing solution to make acquired data usable in our research:

- Inspection of the existing gameplay telemetry dataset and its sample format.

- Identifying raw data collection problems and resolving them for the benefit of our research, and continued developer data sourcing.
- Implementing a processing pipeline to transform raw data into a usable research dataset in a multi-step procedure of reformatting, clean-up, normalisation, and inference.
- Conducting data collection and processing procedures with the established data pipeline to accumulate a dataset to be used in our research.

Game Score Study

Game score study is an analytical investigation of the population dataset, using the proposed game score metric for evaluating samples from the gameplay telemetry dataset. Statistical analysis of emergent distributions and clustering of a selectively sampled population to establish a top play skill player persona, informed further work towards generating learning models that would be applicable in the context of the game's environment:

- Proposal of a quantifiable measure of game score exhibited in gameplay telemetry dataset samples, derived from the game environment and its design context.
- Sampling of the population dataset for use in further analysis and practical work, with respect to parameters observed in Chapter 4: Context-Guided Agents.
- Statistical analysis of population-wide game score distributions and investigation into properties of the game score metric.
- Establishing the concept of play skill and applying it to cluster the game sample population dataset.
- Extracting the top-skill persona dataset to use it in training our agent models.

Agent Study

Agent study covers the practical work stages of training and benchmarking the context-guided agent with experimental evaluation to measure its play skill. It concludes with an analysis of the results obtained to address relevant research questions and review the context-guided agent's performance, design, and deployment.

The training portion of the Agent Study chapter includes:

- Design and development of the context-guided agent model, combining BTs and context-segmented small-scale learning models, trained to model a top-skill player persona.

- Design and development of a reference agent, to provide a comparison case for the context-guided agent model.
- Benchmarking of the context-guided and reference agents.

The evaluation portion of the Agent Study chapter includes:

- Online experimental evaluation, deployed in the instrumented, commercial version of the game for the human player live audience to play new, unseen game scenarios.
- Offline experimental evaluation deployed in the agent simulator environment to simulate the context-guided agent playing the same, unseen scenarios as human players.

The analysis portion of the Agent Study chapter includes:

- Analysis of gameplay performance of the context-guided and reference agents during benchmarking.
- Analysis of gameplay performance of the context-guided agent and human players during experimental evaluation.
- Review of the context-guided agent's design, deployment, and learning.

Evaluation

The planned evaluation of the trained context-guided agent involved conducting offline and online experiments, as detailed in the Experiments in the Game Environment section. In both cases, evaluation participants were expected to play the same set of new, unseen scenarios of the instrumented version of the *60 Seconds!* game environment. Agents were evaluated in the offline experiment, conducted in the agent simulator environment. Human players were evaluated in the online experiment, conducted in the instrumented, commercial version of the game, distributed through the developer's Steam update pipeline. This allowed us to conduct a normalised, comparative analysis of the gameplay performance, measured as play skill, for both agents and humans, in the same scenarios, and approximately similar gameplay conditions. In an effort to integrate the online evaluation into the natural flow of *60 Seconds!*, the experiment was presented as part of one of the game modes, in the default interface of the game, as detailed in the Experimental Evaluations section. The ethics of data collection in the game environment, including the special case of sourcing data in the course of the online, experimental evaluation, is discussed in the Ethics section.
Our proposal to use the commercial game environment for online, experimental evaluations with a real player audience was approved by the University of Glasgow College of Science and Engineering Ethics Committee in November 2016, under ethics application number 300150079.

The approved application covered three online experiments to collect data that would be used for player gameplay performance estimation, agent development, and dynamic environment adjustment. Ultimately, we only planned for the first of these experiments to be conducted as part of the evaluation in this thesis.

4.3 Context-Guided Agent Design

4.3.1 Requirements

Assumptions

The requirements for our design approach were derived from the state of the practice game development workflows, to ensure a realistic and practical approach to the formulated proposal. We were also invested in preventing any game environment-specific assumptions from contaminating the design, which could have adversely impacted its capacity for potential generalisation. To that end, the identified requirements were an expression of the needs of the industry and its experts, by making the proposed design applicable, replicable, and generalisable.

Applicable

Design must be applicable in an actual video game environment, providing a real solution to a real problem. Its integration cannot destructively disrupt the standardised production workflow and runtime architecture of the game. If any augmentation of the targeted game environment is necessary for the purposes of the design's integration, it should be carried out as an increment to the pre-existing architecture, without compromising its original functionality.

Replicable

Design must be easily replicable, leading to reliable, repeatable results when applying the scenario in both design time and gameplay runtime execution. This should also allow for conducting iterative development and flexible revisioning of the output produced, within the target game environment.

Generalisable

Design must be specific enough to address domain problems but abstracted to a point which would allow it to be adaptable to a variety of scenarios and game environments, without compromising the standards of its integration or the target environment.

4.3.2 Architecture

To address the outlined requirements for robust integration of learning into standardised game industry development workflows for game agent AI, we proposed to structure the design around the following key concepts:

Context-Guidance

Human-curated control over behaviour flows through ad hoc authoring with a layered, hierarchical approach to separating the context of decision-making and action execution with an intermediate proxy layer between the two. The context-guided structuring of the proposed design is visualised in figure 4.1 on page 54.

Safe Learning Integration

Direct integration of learning logic into BTs as learning subtrees, featuring a dedicated learning node, supported by auxiliary ad hoc nodes. Learning logic is expected to be modelled in the format of an artificial neural network, which can be developed and trained according to specific domain and design requirements. The template structure of dedicated learning nodes includes ad hoc nodes to provide control and guarantees over their performance and execution. In addition, a fallback, ad hoc logic node is present to take over the behaviour flow in case the learning logic fails to execute within safety constraints. The fallback logic provides a guarantee of a worst-case scenario execution, while also serving as a potential starting point for an incremental implementation of a learning subtree before the actual learning logic is fully trained.

Small-Scale Learning

Isolation of the scope of learning logic to the relevant context, equivalent to task decomposition, in an effort to optimise training time and data requirements, while increasing the accuracy of the learning model output. This results in multiple, smaller, and specialised learning models, instead of a single, monolithic learning model. This approach allows utilising a variety of learning techniques or model parametrisations in different contexts of the same behaviour logic, while encouraging modularity, increasing design control, and potentially decreasing training costs.

4.3.3 Context-Guidance

The hierarchical, subsumption architecture and decision tree properties of a BT organically create assumptions about directing the flow of the behaviour logic. The highest level of abstraction is occupied by flow directing decision-making, while the leaf nodes found on the lowest levels of the tree correspond to actual interactions in the game's environment [43,45,179]. By combining these inherent properties of BTs with a structuring optimisation strategy of context separation,

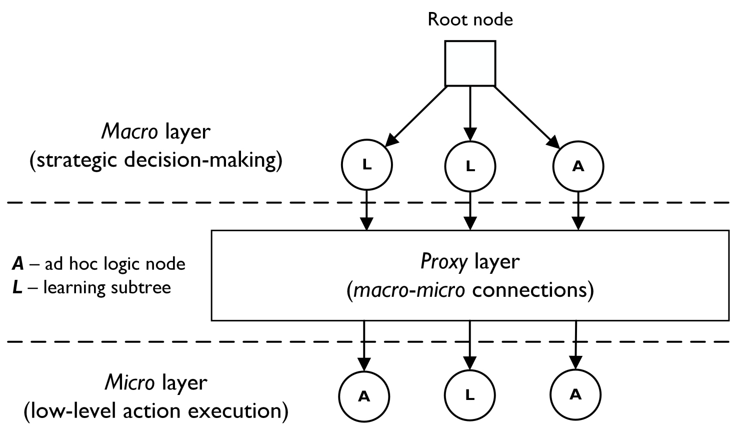

Root node
Macro layer
(strategic decision-making)
L
L
A
A – ad hoc logic node
L – learning subtree
Proxy layer
(*macro-micro* connections)
Micro layer
(low-level action execution)
A
L
A

Figure 4.1: Context-guided architecture with a presentation of the default context layering of *macro*, *micro* and *proxy* layers.

inspired by the HRL task decomposition [185, 200] and an intuitive human drive towards segmenting complexity [75, 103] we observe three context layers in a context-guided BT: *macro*, *proxy* and *micro*. The *macro* layer features strategic and more abstract decision-making logic and is found just below the tree's root node. The *micro* layer represents lower-level actions to be executed by the agent in the environment and is located at the bottom of the tree. The optional *proxy* layer facilitates authoring control over connections between the *macro* and *micro* layers. The separation of task layers and observing a decision-making *macro* layer has been explored in the field, as the issue of combining learning reactive actions and temporally modelled decisions, which suffer from sparse rewards, remains an ongoing challenge [200, 263, 276]. As such, the role of the proposed layers is to provide a disciplined, context-guided framework to support flow planning and positioning of learning subtrees by human designers, who are expected to provide a valid contextual separation of the agent's behaviour flow. Layered optimisations of the kind are usually efficiently embedded in the learning models themselves, as was the case with HRL and IL in multiple projects [108, 138, 142, 185], at the price of transparency and iterative control over their logic. Our assumption that video game AI domain experts should remain at the centre of the AI creation process and have as much authoring control over the agent decision-making logic model, as possible [59, 194, 211] influenced our decision to delegate the context flow separation to a human designer. Similar sentiment was expressed by de Pontes Pereira & Engel, who argued that a BT can be modelled as a specialisation of an option-based HRL, but instead of treating the manual division of behaviours as an issue to be solved through automation, one should observe it as an opportunity to take advantage of prior and expert knowledge of the problem [285].

This approach also tasks the designers with deciding which behaviours are to be modelled using learning logic. Ultimately, the goal of context-guidance is to maximise the utilisation and potential of the learning subtrees, strategically incorporated in the BT's flow structure. Context-relevant positioning of learning subtrees in conjunction with their small-scale learning targets will allow for the feature of adaptive behaviours where they are most required. While *macro* and *micro* layers can contain both ad hoc and learning nodes, the *proxy* layer should only use ad hoc logic to guarantee fully deterministic and controllable connections between the context of what the agent decides and what the agent does. Our design does not enforce a specific volume of learning, instead leaving the exact balancing of the presence of learning subtrees in the architecture at the discretion of a human designer. Greater utilisation of learning may produce a more adaptive agent, but at the price of decreased design flexibility and flow context control. The modularity of the outlined design is intended to make it possible to scale up the presence of learning logic easily and organically. This involves designers starting out with a simple, ad hoc implementation of the intended agent behaviour logic, before replacing it with a learning model, but retaining the original ad hoc implementation as a safety fallback in the event that the corresponding learning subtree fails to execute.

4.3.4 Safe Learning Integration

The proposed design assumes that the execution of learning logic is potentially unsafe and unreliable [234]. In the best-case scenario, learning logic will produce output that is approximately accurate, with respect to design expectations. In the worst-case scenario, learning logic fails to execute and may compromise the execution of the entire agent behaviour logic. This risk and lack of execution determinism are key issues, which make learning models too unstable to be confidently used in professional game development [194, 270]. Ad hoc safeguards and fallback logic are necessary to create performance and execution guarantees for integrated learning.

Such an approach can be achieved by packaging learning logic within a BT subtree, featuring a dedicated learning BT node and ad hoc safety control logic. This was formally proven to provide the required safety by Sprague & Ögren [234], making it an optimal solution to address the issues with integrated learning. The execution of a learning tree follows the regular BT structuring flow. First, the execution safety controller evaluates conditions, defined by designers, to determine if the subtree should proceed with execution. If not, it is aborted. Otherwise, performance safety controller tests key processing requirements, also defined by designers. Failed performance safety testing leads to the execution of ad hoc fallback logic, instead of the primary learning logic. If both execution and performance safety tests are passed, the dedicated learning node can run.

To accommodate a broader range of game development-relevant scenarios, we modified the original design of Sprague & Ögren [234] and addressed its two shortcomings: a limited scope of performance safety control and a lack of ad hoc fallback logic execution guarantees. Our version of the learning subtree is presented in figure 4.2 on page 56.

The original design limited performance safety control to a single time response control test. Inspired by the Lyapunov design methods for safety [14, 38], we assumed that performance testing should be structured to potentially support multiple stability and safety tests, compatible with the target domain. In our interpretation, any condition which is expected to invoke ad hoc logic, instead of the learning logic, should be part of the performance safety controller. A practical example of such a condition is hardware testing to avoid learning performance issues on slower machines and diverting execution to lightweight, fallback ad hoc logic, instead.
Our design also features a revised composite flow of the learning subtree, which supports ad hoc fallback logic execution in the event of learning logic failure, which was not originally the case. This was a critical change, as the output expectation must always be met, even if only the worst-case version of the output can be produced.

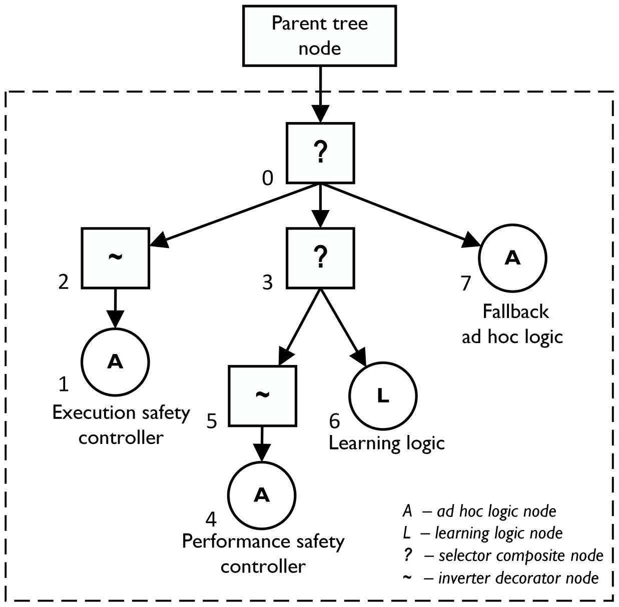


Figure 4.2: Learning subtree design, derived from the formalised definition of safe BT neural network controller by Sprague and Ögren [234]. Order of node execution is indicated by node numbering.

The proposed and documented extension of the learning node constitutes the research contribution **C2**.

4.3.5 Small-Scale Learning

Each learning subtree embedded in the BT architecture may operate independently of others, representing an individual instance of learning logic, contained within the learning node. It is the responsibility of a human expert to conduct manual task decomposition, according to the intended and pursued behaviour context of the agent and delegate the appropriate learning context to each subtree. This practice is to be supported by context-guidance-driven structuring of the BT architecture. While the design is concerned with manual context decomposition, the process is inspired by research into automated task decomposition within learning models [55, 60, 122, 209]. Isolation of the scope of learning in each subtree to the relevant context is motivated by the potential reduction in training time and data requirements. It is also likely to result in increasing the accuracy of the learning model output, as is the case with the accuracy of individual task learning in automated multi-task learning [5,37]. The small-scale learning output will utilise multiple, smaller and specialised learning models, instead of a single, monolithic learning model, supporting modularity and authoring control. This approach will also allow mixing learning techniques applied in different learning subtrees, instead of enforcing a single strategy for policy learning.

4.3.6 Agent Perception

We assume an agent to be an object with capacity for autonomous operations in a video game's virtual environment e. Agent's interactions with the environment follow a specified policy π_a, expressed as a finite set of actions A_a sequenced with an underlying stochastic or deterministic logic model, executed with respect to the state of the environment s and discrete timestep $t \in \{t_0, t_1, ..., t_n\}$. The game's environment can be formally modelled as an MDP [195], which is a common practice in games' research [122, 238, 270]. In such a case, the MDP is described by the tuple $< S, A, P, R >$ where:

- S - a finite set of states $\{s_0, s_1, ..., s_n\}$, considered state space or information about the environment's conditions.
- A - a finite set of actions $\{a_0, a_1, ..., a_n\}$, considered action space or ways in which the agent can interact with the environment.
- $P(s, s', a)$ - the state transition function expressed as the probability of transitioning from state s to s' upon executing action a. It fulfils the Markov property, which states that future states of the process depend solely on the present state, rather than the sequence of states that came before. This implies that the current state s provides complete and valid information about the environment.
- $R(s, s', a, r)$ - the reward function given transition from state s to s' through execution of action a. It specifies the immediate reward r, or otherwise a signal, provided to the agent when a transition from state s to s' occurs through action a.

Assuming the game environment e is fully observable, and its world model defined by P and R is known, estimation of transition probabilities P and reward r calculation are directly available. This allows to maximise the reward function R for achieving an optimal and fault-proof policy for agent interactions A_a using model-based approaches. However, given an optimised reward function R, this would result in a maximally efficient agent operating with a machine-like precision, rather than exhibiting human-like gameplay behaviour. This is desired for certain classes of agents, which play for performance [270]. In our research, we were interested in agents playing for experience, which do not necessarily strive to maximise their reward function yield. Instead, they are prepared to imitate human gameplay behaviour style, which requires learning previously demonstrated or identified stylistic elements. In such a case, it is intuitive to assume that agent perception should also imitate that of a human, and as it would not permit full observability of the environment, despite the fact that full state information might be available. Hence, we had chosen to model the agent's behaviour flow as a generalisation of an MDP in the form of a Partially Observable Markov Decision Process (POMDP) [24]. While we still consider the game environment to be a fully observable MDP, interfacing with it from a perspective of a POMDP effectively emulates human-like sensory perception of the surrounding world. It is also computationally beneficial, as an agent interacting with a completely observable MDP and receiving full information about the current state s may lead to run-away observational complexity, excess of data volume, and high costs of collecting observations [21, 23, 277]. As the agent does not directly observe the environment's state s, its decisions are made under uncertainty, on the basis of a partial environment state s information derived from a set of observations Ω received by the agent at a regular timestep t. A POMDP is described by the tuple $< S$, A, P, R, ω, o, $\gamma >$ where:

- S - state space, similar to MDP.
- A - action space, similar to MDP.
- $P(s, s', a)$ - state transition function, similar to MDP.
- $R(s, s', a, r)$ - reward function, similar to MDP.
- Ω - a finite set of observations $o \in \Omega$, received upon transition to new state s', as a result of action a.
- O - set of conditional observations probabilities for receiving observations Ω, expressed as the probability $O(o, s', a)$ of receiving observation $o \in \Omega$ upon transition to new state s', as a result of action a.
- γ - discount factor $\gamma \in [0, 1]$, where $\gamma = 0$ promotes immediate rewards over more distant rewards. The greater the γ, the more valued maximising the expected sum of future rewards is.

A POMDP modelled decision process begins with an initial belief b_0 about the true state of the environment, which may be updated after the agent takes action a and observes o. In accordance with the Markovian property, formulating the next belief state b' requires only information about the previous belief state b. As a result, training the agent's environment interaction policy π_a on such predictions, rather than direct state data, results in a more natural response to the changes in the environment, instead of machine-like efficiency. This approach can further be supported by choosing to model the perception of the state of the environment using low-dimensional observations.

4.3.7 Deployment Workflow

Assumptions

The deployment of a context-guided agent model instance, architected on the basis of a design targeting a specific game environment, is structured as an iterative workflow, repeating particular stages of the process in development-evaluation cycles. This is consistent with the game industry development workflows and supports continuous evaluation and fine-tuning of the output model until a satisfactory level of quality is achieved. The flow of the deployment workflow involves the following steps:

- **Step 1**: Design behaviour context.
- **Step 2**: Establish learning subtrees.
- **Step 3**: Implement context architecture.
- **Step 4**: Implement learning logic.
- **Step 5**: Acquire data.
- **Step 6**: Train learning models.

The sequential flow of these stages is presented in figure 4.3 on page 60. We discuss each of the stages in the sections below.

Step 1: Design Behaviour Context

The deployment workflow of a context-guided agent begins with identifying the context of the intended behaviour flow, to inform the development of the agent's behaviour context architecture. This process involves conducting a requirements analysis of the target environment by:

- Specifying goals that an agent is expected to pursue and achieve in the environment.
- Reporting interactions that an agent may perform in the environment.

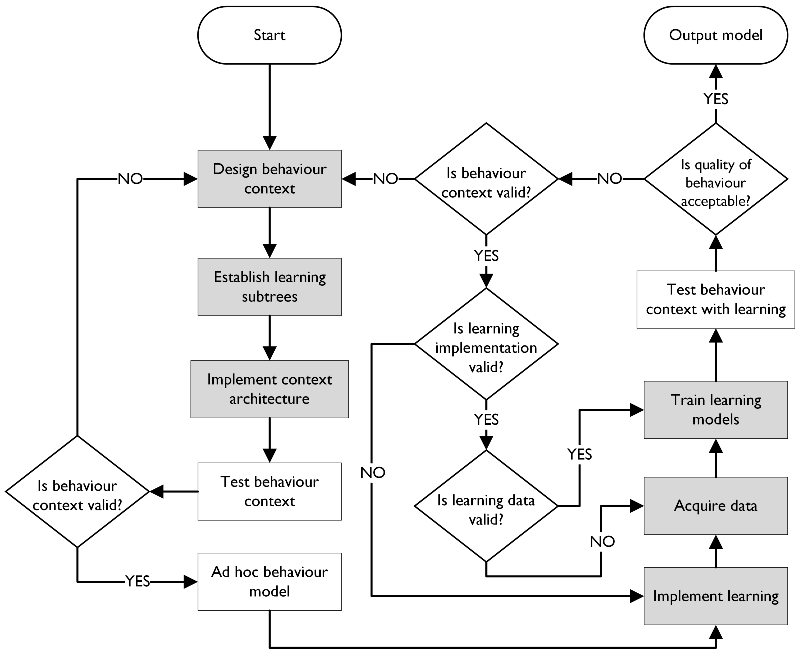

Start
Output model
YES
Design behaviour context
NO
Is behaviour context valid?
NO
Is quality of behaviour acceptable?
YES
Establish learning subtrees
Test behaviour context with learning
Is learning implementation valid?
Implement context architecture
YES
Train learning models
NO
YES
Is behaviour context valid?
Test behaviour context
Is learning data valid?
Acquire data
NO
YES
Ad hoc behaviour model
Implement learning

Figure 4.3: Context-guided design deployment workflow.

Mapping the identified behaviour objectives to the *macro* layer, the available interactions to the *micro* layer, and proposing how they connect via the *proxy* layer, follows. The designed mapping will only be validated after an initial model is deployed. Hence, competent identification of the behaviour context will greatly benefit from an expert-level understanding of the game environment. However, this may not be possible due to the game's production being at an early stage, where complete knowledge of the environment and its representation is still unavailable. If that is the case, up-to-date information about the environment should provide data necessary to proceed with the workflow, while ongoing exploration and discovery, organically taking place during development, is expected to provide revisions of knowledge and be iteratively integrated into the design of the behaviour context.

Step 2: Establish Learning Subtrees

While the designed behaviour context is a blueprint to fully implement the context architecture using ad hoc authoring, it does not yet provide information about integrating learning into the behaviour flow. The next step is to identify and describe behaviour context flow segments, which would benefit from being modelled with learning logic. This is achieved in three steps:

- Identifying flow segments that should feature learning.
- Selecting learning policy strategies and techniques for the identified learning subtrees.

- Determining data requirements for the identified learning subtrees, and their chosen learning policy approaches.

To identify flow segments of interest, each of the mappings in the *macro* and *micro* layers should be assessed with respect to their intended functionality, and the potential advantages of achieving that functionality with learning. Likely candidates for learning integration include instances of flow that:

- Need to be highly adaptive.
- Cannot be effectively represented by ad hoc logic, either due to constraints on ad hoc logic flow, or because of the prohibitive explosion of complexity that would result from scaling it up.
- Are expected to produce imitative output, on the basis of demonstration data.

If any of these conditions are met, a human designer needs to decide if the examined case is convincing enough to integrate learning. While not all flow logic should be modelled with learning, it is important to remember that the risks of including too much learning are limited. The worst-case deployment scenarios of failing to produce an effective enough learning subtree or discovering that learning was not necessary for a particular piece of flow logic have a low impact. In both these cases, an ad hoc fallback logic should already be in place, providing an optimal ad hoc solution to the problem, or at the very least a functional one, which can be used in further development. Confirmed learning subtrees can employ different learning policy generation approaches to achieve their goals. Acceptable techniques can encompass anything that is technically supported by the targeted game development and runtime environment. These choices need not be limited to a single method. Combining different techniques to produce a higher quality of output has been found to produce promising results [111, 222, 270], and the natural decomposition of the context of tasks and their associated learning logic found in the context-guided design supports this methodology. It is important to make informed decisions about the policy generation, since they determine data requirements for the learning models involved, and the scope of the technical work required to implement the context architecture.

Data requirements for each of the learning subtrees can then be inferred from the context of the desired behaviour, and on the basis of the selected policy generation strategy. The pinpointed requirements should provide enough information to describe the necessary volume and sourcing of the data, and structure data collection procedures to be conducted in Step 4 of the deployment workflow.

Step 3: Implement Context Architecture

With design foundations laid down, the technical part of the deployment can commence. Establishing a behaviour context architecture encompasses BT flow structuring and implementing game environment-specific solutions to facilitate that flow. While the technical execution of these two tasks will vary across different gameplay solutions and tools used, the implementation process will remain the same and involve:

- Implementing functionality to interface with the game environment through BTs.
- Implementing ad hoc BT flow (*macro*, *proxy*, and *micro* layers).
- Implementing stubs of learning subtrees in the ad hoc BT flow.
- Implementing ad hoc fallback logic for each learning subtree.

Once the implementation reaches a stage of maturity where testing becomes possible, an iterative cycle of testing and fine-tuning can begin. The implemented context architecture should already support the pursuit of the full scope of the identified behaviour goals for the agent, representing a valid behaviour context. These assumptions need be evaluated in the game's environment, with respect to the requirements of the target environment, identified in step 1. If confirmed to be valid, the output of the context architecture implementation constitutes a stable, ad hoc version of the intended behaviour model. Detected behaviour context violations may warrant an architecture revision by repeating one or all the steps 1-3, until all issues are resolved.

Step 4: Implement Learning

After achieving a stable behaviour context, expressed as an ad hoc flow with learning subtree stubs in place, the intended learning logic can be integrated into the architecture. For each learning subtree, a dedicated learning node should be implemented, with respect to the local set of technical factors, such as the project architecture, tools used, and the specifics of the target game environment. A single-node implementation might suffice if the same learning technique is used across many learning subtrees. For varied scenarios, a range of learning node specialisations, which could address local training and inference requirements, may be required.

Additional technical work may be required beforehand, if the target environment is not ready to accommodate learning processing:

- Implementing relevant training and inference logic in code, to facilitate learning logic execution.
- Connecting the output of a context-guided agent model to an actual agent object, operating in the game's environment.

Step 5: Acquire Data

Different learning methods have varied data requirements. Depending on what methods were chosen for dedicated learning nodes, featured in the established behaviour context, appropriate data sourcing must be applied. In some cases, especially involving RL techniques, data will be collected from training-time observations of the agent, and as such, the data collection process would have already been part of the learning implementation in Step 4. In such cases, deployment can forego Step 5. Otherwise, appropriate data acquisition and transformation procedures must be applied to generate valid datasets for use in target learning models. It is beyond the scope of this workflow to explore this process, but it can be as simple or as complex as required. The only requirement for the data is to be usable in the targeted game environment and to be compatible with the learning subtrees' policy generation strategy that is supposed to take advantage of the collected data. The output of this step is expected to be a dataset, or datasets, of valid training data for all the learning subtrees found in the context architecture.

Step 6: Train Learning Models

The final, and potentially the most labour-intensive step of the workflow, tasks the designer with iteratively training learning models, embedded within the learning subtrees of the implemented context architecture. Each iteration of training generates an output agent model, which can then be evaluated in the context of the game environment. The evaluation is aimed at validating whether the agent's behaviours can be considered competent, operational, valid and of high enough output quality, in terms of the intended behaviour context. Multiple training passes are likely to be required, since varied parametrisation and timing of the training process impact policy generation. The time necessary for training each of the learning subtrees may be different, and the training process might involve local partitioning to validate learning models separately. However, a holistic perspective should be adopted to consider all the relevant factors of the intended behaviour context.

Validation of the behaviour context in Step 6 mirrors the testing procedures used in Step 3. Any issues with the underlying behaviour context detected at this point would require backtracking the deployment process to repeat Steps 1-3 to revise the design. Problems resulting from the training process itself would simply prompt another iteration of the learning training-testing process. A data-driven approach should be applied to quantitatively compare different agent output model revisions with respect to the output fitness. This enables automating the iterative training-testing cycles, and potentially even procedural, incremental improvement of the output models using evolutionary algorithms. Quantitative comparisons may focus exclusively on output fitness, but ultimately, the main focus should be on the game context-derived measures of success, and other factors that root the agent behaviour model in the context and reality of the game environment. The evaluation itself should take place in the game environment to fully align with the investigated context. A truly holistic evaluation of the quality of the intended behaviour can be considered a subjective endeavour, and as such requires involvement from a human expert. Especially that repeating quality concerns might need an expert perspective to target the issue, and fix the underlying cause, leading to repeating all or selected deployment learning steps. After the trained gameplay policy reaches the desired level of quality, the learning training work is complete, and the final output model is ready.

Output Model

The final output of the deployment workflow is a context-guided agent behaviour model, architected with an ad hoc established behaviour context, including integrated learning logic, trained with respect to the valid input data. The produced behaviour model is expected to be applicable in the game environment it was developed and trained for, and fulfil the objectives of the behaviour context, as outlined by human experts at early stages of the deployment workflow.

4.3.8 Summary

We have presented the context-guided agent design proposal, which combines the concepts of ad hoc authored context-guidance, safe learning integration into BTs, and small-scale learning models. The documented design is based on the established and reliable BTs, a formally proven safe approach to integrating learning into BTs and addressing the practical requirements of the state of the practice workflows. The presented design proposal constitutes a theoretical response to the research question **RQ1** and constitutes the research contribution **C1**. An empirical response to **RQ1** will follow in Chapter 6: Agent Study.

4.4 Game Environment Instrumentation

4.4.1 Overview

Developers of *60 Seconds!* provided us with full access to the game's project assets, as well as its codebase and data collection pipeline. This also included previously collected gameplay telemetry datasets. This direct access allowed us to instrument the game for the purposes of our research. Instrumentation work included the improvement of the data acquisition flow, deployment of experimental evaluations in the live version of the game and implementing support for autonomous agent training and simulation. Any code developed and deployed in the course of our work was intended to seamlessly and non-disruptively integrate into the game environment. This was the case for both the code we developed ourselves and any external libraries integrated by us. In the latter case, we took advantage of the off-the-shelf nature of the game's engine and only used solutions that were compatible with the game project and were available free of charge.

4.4.2 Data Collection

60 Seconds! was originally equipped for *scavenge* gameplay telemetry data logging by its developers. A functional data collection pipeline for automatic crowdsourcing of data samples from the players of the game was already in place. However, in the course of our investigation of the pre-existing gameplay telemetry dataset, we identified a number of issues with the implementation of the data collection in the game's executable. We assisted the developers in resolving the problems discovered, without breaking or changing the underlying data collection mechanism. The revised data collection implementation was introduced in the commercial version of the game and released to its players as a free, automatic, online update. This resulted in players generating corrected gameplay trajectories, which would later be acquired using the original data collection pipeline and used in our work. We discuss the data collection issues and how they were addressed in the course of our game instrumentation work in the Gameplay Telemetry Dataset section.

The *survival* section of the game was not outfitted with a similar data collection mechanism, and the developer had no plans to implement it. Since that was the case, we were not at liberty to modify the *survival* segment of the game. The lack of access to *survival* data informed our decision to limit our research focus to data from the *scavenge* segment of the game.

4.4.3 Agent Training and Simulation

Overview

60 Seconds! did not support AI-driven, autonomous agents in any capacity. It was originally designed as a single player experience, controlled solely by the player's input. Learning was also not an original feature of the game's development environment. To be able to carry out our research investigation, we had to integrate off-the-shelf libraries for AI behaviour ad hoc authoring with BTs and ML model training and inference into *60 Seconds!*. We then modified the game environment to support the operations of autonomous agents. Finally, we deployed a standalone agent simulator for training and inference of agent AI behaviours, based on the game environment of *60 Seconds!*.

Learning Models

Support for native learning integration into Unity game engine projects was developed by Unity Technologies in the form of a free-of-charge *Unity Machine Learning Agents (ML-Agents)* toolkit version 0.30.0 [118][2]. Since it was authored by the makers of the engine itself and effectively leveraged the internal implementation of Unity, we opted for using this solution in our research. It is still in active development, but already contains an array of stable, state-of-the-art learning features. Training is facilitated via the open-source Python library *PyTorch*, which is a standard tool for ML practitioners from different fields of research [177]. Model inference in the engine environment is facilitated using the *Barracuda* inference engine. *ML-Agents* supports training learning models via Deep RL (PPO, Soft Actor Critic, Multi Agent Posthumous Credit Assignment, self-play) and IL (BC and GAIL). Models generated by training are stored in the Open Neural Network Exchange (ONNX) format. The library was previously used in other research projects [55, 140, 144, 271, 278]. To take advantage of *ML-Agents* we have developed agent representations that were compatible with the library's API. We then interfaced these custom agent implementations with our dedicated BT learning node implementation. This allowed us to control the flow of training and inference signals between *ML-Agents* and a BT driving the logic of an autonomous agent AI.

BT Authoring

We have chosen the *Behavior Bricks* BT implementation by PadaOne Games[3], associated with the Group of Artificial Intelligence Applications at the University of Madrid[4], to integrate BT support into the game. Our decision was motivated by the fact that this library is a native and a stable Unity solution, available free of charge, which features functionality compatible with the game industry state of the practice BT requirements. Apart from the basic BT flow logic, this includes a responsive visual editing tool for trees with support for real-time debugging. Initially, it was also our understanding that *Behaviour Bricks* included an implementation of a BT learning node. Having discovered its support was discontinued, we developed an *ML-Agents* compatible implementation of a dedicated learning node on the basis of the *Behaviour Bricks* codebase and the assumptions of safe learning integration outlined in our design proposal. We intend to share this implementation with the developers of *Behaviour Bricks*, as well as the development community at large. This part of our work constitutes research output **O1**.

Autonomous Agent Support

We anticipated that introducing autonomous agent functionality into *60 Seconds!* would be a relatively straightforward process, based on selective mapping of interaction signals from player input to AI model decision-making logic. Unfortunately, the game's architecture was based on an assumption that player input is the sole driver of the player avatar operating in the game environment. This resulted in a tight coupling of avatar interactions and player input logic code

[2]https://github.com/Unity-Technologies/ml-agents
[3]https://gaia.fdi.ucm.es/research/bb/
[4]https://gaia.fdi.ucm.es/

through continuous processing of the input-logic loop. A forward model of the game's simulation was not present. Decoupling these features and introducing a partial, forward simulation model required a significant intervention in the game environment codebase. We took care to ensure these modifications were non-disruptive, as it was an important assumption of our design work. This had proven more labour-intensive than expected. In the end, we were able to achieve our goals without disrupting the functionality of any of the game's pre-existing systems. However, due to the way that player input was coupled with the camera control in the original code, it was necessary to introduce an alternative solution to handle viewport control and interactions separately for artificial agents. The camera model implemented for autonomous agents approximately replicated the functionality and behaviour of the game's original camera. Additionally, it was expanded to act as a vision sensor for autonomous agents that would imitate human-like observation of points of interest in the game environment. Specifically: *items* to be collected. Observed *items* could then be registered to be used in AI decision-making logic. For the purposes of simulation preview, we also introduced an alternative, top-down camera option to provide an elevated overview of the entire game environment.

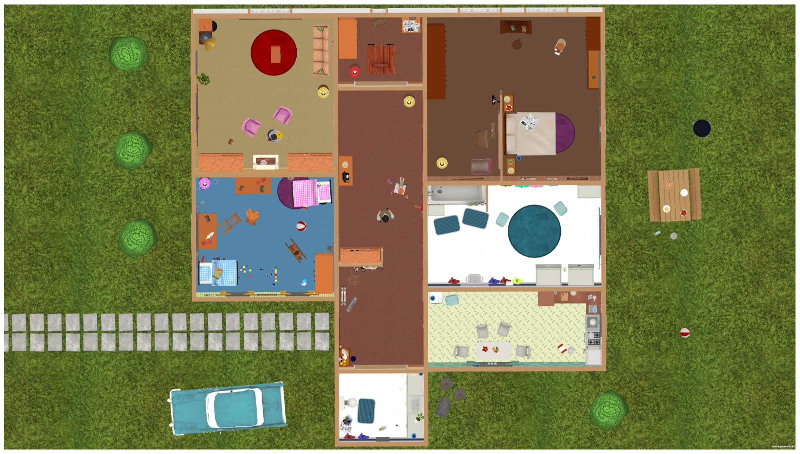


Figure 4.4: Top-down camera implemented in the agent simulator for convenient simulation preview.

Avatar interactions in the game were investigated with respect to object interactions and navigation. Object interactions, including collecting and depositing *items*, were not as coupled with other systems as navigation. This made it possible to facilitate forward model-based object interaction requests and querying with minimal facade code. This allowed us to directly use the pre-existing avatar interaction model, designed for human player input. Navigation reimplementation had proven more challenging. Aside from system decoupling, we had to consider the pre-existing assumptions of the game's physics system, as well as servicing two types of navigation input: absolute position vector values from gameplay trajectories, and AI decision-making. Navigation step recording frequency for gameplay trajectory data samples was set to $f_D = 10$ Hz (data recorded every 100 ms), which was not as precise as the player input capture, running at a frequency of $f_P \approx 60$ Hz (data recorded approximately every 16.66 ms). Hence, for gameplay trajectory replays, we introduced discrete navigation steps that agents would move between, masking lower frequency recordings using interpolation. For AI decision-making we wanted to approximate the human player input frequency and allow for a more reactive navigation. We decided against direct simulation of raw player inputs, as this would require a non-trivial development of the avatar navigation and steering system in the pre-existing game environment. Furthermore, we predicted that such an approach would later force us to spend a substantial amount of time, effort and compute on training learning models that could cope with navigation in relatively complex environments. So instead, we opted to approximate the human navigation dynamics by generating a navigation mesh for the investigated game levels and finding optimal navigation points on it using Unity's implementation of the A* pathfinding algorithm[5]. While this approach decreased the number of navigation blunders and collisions that a human player was likely to cause, we were able to balance the implementation to make it accurate enough, but not perfect. By removing the navigation as a dimension in learning model training, we also reduced the complexity of planned learning tasks. This allowed our learning investigation to be focussed on decision-making, rather than having an agent learn environment operations that could be solved with more reliable and less computing-intensive methods.

Agent Simulator

With the support for autonomous agents in place, we moved to develop a configurable simulation environment to execute agent training and inference scenarios. The simulator was implemented in the form of a partial game environment extracted from *60 Seconds!*, restricted to the selected *scavenge* game type functionality. The simulator was designed to be configurable, making it possible to parametrise executed scenarios with respect to a range of input variables. The main features of the simulation environment are:

- Human-controlled gameplay of *scavenge* games to play the game.
- Simulating agent behaviours in inference mode, on the basis of previously trained learning models or BT assets, with respect to the parameters provided.
- Generating new gameplay trajectories in AI inference or human play mode.
- Replaying previously recorded *scavenge* gameplay trajectories.
- Generating Unity compatible demonstrations, used for IL in *ML-Agents*, from previously recorded *scavenge* gameplay trajectories.
- Training agent ML models in training mode, on the basis of parameters supplied, to generate new learning models using *ML-Agents*.

[5]Using Unity's NavMesh Agent component.

The simulator was deployed using Unity engine version 2023.2.20f1 as a standalone, Microsoft Windows 64-bit based executable. It can be executed without ownership of the original version of the game *60 Seconds!*. Please be aware that assets contained within the agent simulator environment are covered by copyright, and as of 2025 are the property of Robot Gentleman sp. z o.o.
The simulation environment scenarios are configured using parameters provided in JavaScript Object Notation (JSON) text files, featuring relevant settings for the simulator environment, the procedure, and the simulated agent. Additional configuration is possible on the executable's level to service non-display batch mode via Unity's headless mode execution. Running multiple instances of the simulator in parallel is supported in both visual and batch mode. The execution speed of the simulation environment can be configured in the range [0.0, 10.0] to compress inference time, at no loss of processing steps. Training learning agents via *ML-Agents* is possible in combination with the *PyTorch* based control. For ML training, hyperparameters are configured using YAML setup files, required by *ML-Agents*.

Although we were able to apply Unity, the game environment developed in it, as well as *ML-Agents* for our research, we encountered a number of issues during our work on the simulator. The constraints of Unity's physics system, extensively used in the gameplay logic of *60 Seconds!* for collision detection and navigation, limited the simulator. We originally planned to allow the simulator execution speed to be increased without restrictions, but found Unity's physics system to become unstable and produce unreliable outputs if the execution speed was increased by more than a factor of 10.0. In such a case, agents would no longer properly detect collisions with the environment and other objects in it, miss interaction opportunities, and as a result generate invalid behaviours or violate the consistency of the game's environment. This slowed down agent training. We partially addressed this problem by employing an execution setup that combined multiple instances of the simulator on a single machine, in a non-display batch mode. However, our work with learning agent training was further hindered by central processing unit (CPU) bottlenecks caused by the implementation of *ML-Agents* [55], and its parametrisation overriding execution speed defined by the simulator's configuration files. Since that was the case, when training learning models, the simulation execution speed must be configured via *ML-Agents* parametrisation. The structuring of the context agent design also implied that more than one ML model would be used at the same time. While this caused no issues for inference scenarios, training multiple models in a single environment was not possible. This informed the training pipeline for context agents to execute individual training for each of the models, while other models operated in inference mode, or were absent, and their logic was facilitated by ad hoc fallback representations.

The presented simulation environment, developed on the basis of the commercial game *60 Sec-*

onds!, constitutes research output **O2**. The use and configuration of the simulator are documented in Appendix C.

4.4.4 Experimental Evaluations

To run experimental evaluations with human players, we augmented the *scavenge challenge* game mode in the commercial version of *60 Seconds!* to make it a deployment driver for the planned evaluation trials. This allowed us to take advantage of the original, unaltered setup of the game, which was familiar to its users, and reach the game's large userbase, capable of generating valid gameplay telemetry data in their natural play environment. Deploying experimental evaluations as *scavenge* challenges made it intuitive to extract and isolate sample datasets from individual experimental trials. To frame research challenges as part of the ongoing update publishing pipeline of the game, developers agreed to release them as instalments of the *Rocket Science* DLC. *Rocket Science* was an open-ended update package, which delivered new, free-of-charge *scavenge* challenges to the players of *60 Seconds!* on a semi-regular basis. Completing each challenge unlocked a cosmetic reward.

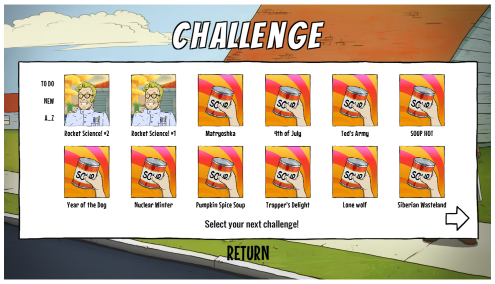
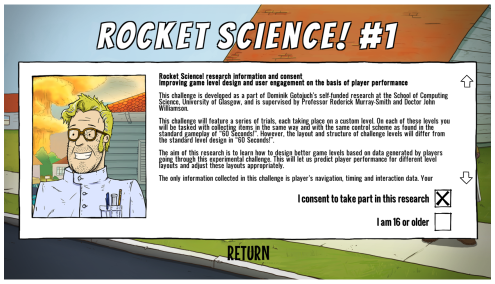


Figure 4.5: Scavenge challenge selection, including experimental evaluations.

The chosen approach made it possible to utilise the natural flow of the game, and get users involved with our research, without special recruitment or training. Research challenges were available in the context of the game's regular *scavenge* challenges. They were presented using the same GUI, making them easily reachable to potential participants. Presentation of research challenges differed from the standard challenge presentation format at the point of providing challenge details. For regular *scavenge challenges*, only their goals were outlined, while experimental evaluations featured complete information about their research context. The scope of the information presented was documented in Appendix B. Additionally, the research challenge user interface included informed participation consent and age verification check boxes, which had to be checked to start any given experiment. Those controls were included to ensure that only willing players over 16 years of age took part and contributed their gameplay data in research challenges. Participants were also free to opt out of contributing to the data collection outside of the research challenge interface. The opt-out was accessible from the game's main menu user interface setting section.



Rocket Science! research information and consent
Improving game level design and user engagement on the basis of player performance

This challenge is developed as a part of Dominik Gotojuch's self-funded research at the School of Computing Science, University of Glasgow, and is supervised by Professor Roderick Murray-Smith and Doctor John Williamson.

This challenge will feature a series of trials, each taking place on a custom level. On each of these levels you will be tasked with collecting items in the same way and with the same control scheme as found in the standard gameplay of "60 Seconds!". However, the layout and structure of challenge levels will differ from the standard level design in "60 Seconds!".

The aim of this research is to learn how to design better game levels based on data generated by players going through this experimental challenge. This will let us predict player performance for different level layouts and adjust these layouts appropriately.

The only information collected in this challenge is player's navigation, timing and interaction data. Your

I consent to take part in this research ☒

I am 16 or older ☐

Figure 4.6: Experimental evaluation consent and information form.

Research challenges replicated the flow and setup of regular *scavenge* challenges. Players were placed in a *scavenge* environment, designed by the author, and given a collection goal to fulfil within the 60-second timeframe. Collection goals were either defined as a list of *items* to collect or as a general objective of acquiring as many *items* as possible, within the time limit. In the former case, an experiment was completed as soon as the player had collected all the *items* listed. Otherwise, the challenge continued until the timer had run out. Players were given an exploration time of $t_{exp} = 5$ s, before the $t_{scav} = 60$ s collection period.

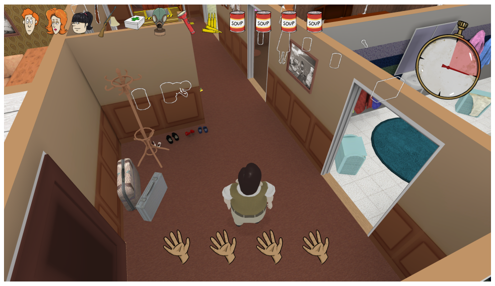


Figure 4.7: Scavenge challenge game session.

To allow for designing independent, as well as repeated measures evaluation trials, we introduced additional setup functionality for deploying *scavenge* challenges with a sequence of environments. It made it possible for experiments to involve any number of environments or environment variations, chosen in a pre-designed manner or at random. In such cases, only after playing all the environments was the experiment reported as complete. *Scavenge* challenges yielded a visual reward, in the form of a wearable *survival* character hat, which was issued to a player after they completed the challenge. Research challenges took advantage of this feature and provided users with a unique visual reward for each completed experiment. No other gratification was provided to the participants.

4.4.5 Game Environment Changes

While we attempted to minimise the number of modifications introduced to the game environment in the course of our instrumentation work, some changes were necessary. Most notably, the use of *ML-Agents* required us to upgrade the Unity engine version used by the game from version 5.6.6f2 (released in 2018) to 2023.2.20f1 LTS (released in 2024). This was a significant change, covering an upgrade of several key systems and warranted a thorough review of the game's functionality. It included technical adjustments and code changes in different parts of the game's codebase, necessary to remove deprecated API calls and adhere to the changed development environment setup. Before proceeding with our research work, we confirmed that the original functionality of the game remained unchanged. Another significant intervention into the game environment was a result of introducing the autonomous agent support, discussed in the Autonomous Agent Support section, combined with the support for learning model training and inference. Introducing a navigation mesh to facilitate agent navigation required generating navigation mesh assets for each of the game levels used by agents. Revisions of the data collection code of the game, referenced in the Data Collection section, also contributed to the programming work involved in instrumentation of the game environment for the purposes of our research.

4.4.6 Summary

The instrumentation work required to conduct our research constituted a significant programming effort, but despite complications, it was completed successfully and without disruptions to the game's functionality. Additional information about the software used, solutions implemented, as well as the agent simulator created for the purposes of our research can be found in Appendix C. Research outputs **O1**, **O2**, and **O3** were documented as part of the instrumentation of the game environment of the commercial game *60 Seconds!*.

4.5 Gameplay Telemetry Dataset

4.5.1 Overview

The mass-scale gameplay telemetry dataset used in our research was crowdsourced from the user population playing the *scavenge* segment of the game *60 Seconds!*. In May 2016, the developers of the game deployed a data collection pipeline to remotely source gameplay data from the *scavenge* segment of the game. They intended to use the collected data to analyse gameplay trajectories of their players, and to potentially use the results of their investigation in future design work for the game. The developers' consent to use the data collection pipeline and output in our work afforded us the opportunity to conduct our research and to assist them in improving their data collection mechanism. As referenced in the Data Collection section of the Game Environment Instrumentation discussion, collected data was limited to gameplay trajectories from the *scavenge* segment of the game.

4.5.2 Raw Data

Overview

We designated the samples collected from users as raw data. We found the raw data sample format that had been originally implemented by the developers to be ill-structured for analysis. To make it usable in our research, we deployed and applied a multi-step transformation and clean-up procedure to the collected raw game samples. This procedure produced a dataset of processed game samples that we were able to analyse in a normalised manner.

Format

The game sample data was recorded in a plain, textual ANSI format for human readability. Data points collected for game samples were aggregated in a continuous string of relevant data groups and data points, separated with appropriate tags and a common separator symbol. Data groups contained in a game sample included:

- **Game setup and progress**: game parametrisation and progress information, including such parameters as game type and difficulty, important timestamps and the data on the end state reached.
- **Room setup**: environment's architecture, randomisation information of room positional and type data.
- **Item setup**: positional and type data for all *items* placed and available for collection in the game's environment.
- **Navigation trace**: positional data records for player avatar, sampled at f = 100 ms, time stamped.
- **Collections log**: all recorded instances of *items* picked up by the player avatar, time stamped.
- **Deposits log**: all recorded instances of player depositing carried *items*, time stamped.
- **Collisions log**: all recorded instances of player avatar colliding with a collidable object in the game's environment, time-stamped.

The structure of the developer-deployed data sample format, referred to as the raw data sample format in this thesis, is documented in Appendix A.

Collection

Developers of the game implemented data sample collection as a decentralised, client-side-driven process. The instance of the game running on the user's desktop computer was responsible for recording the sampled data, serialising it into a textual file and then transmitting that file to the developer's secure, online storage via a dedicated web service. If the transmission had failed, the sample file remained on the user's computer's local storage until the next time the game was run, and another attempt at file transfer could be made.

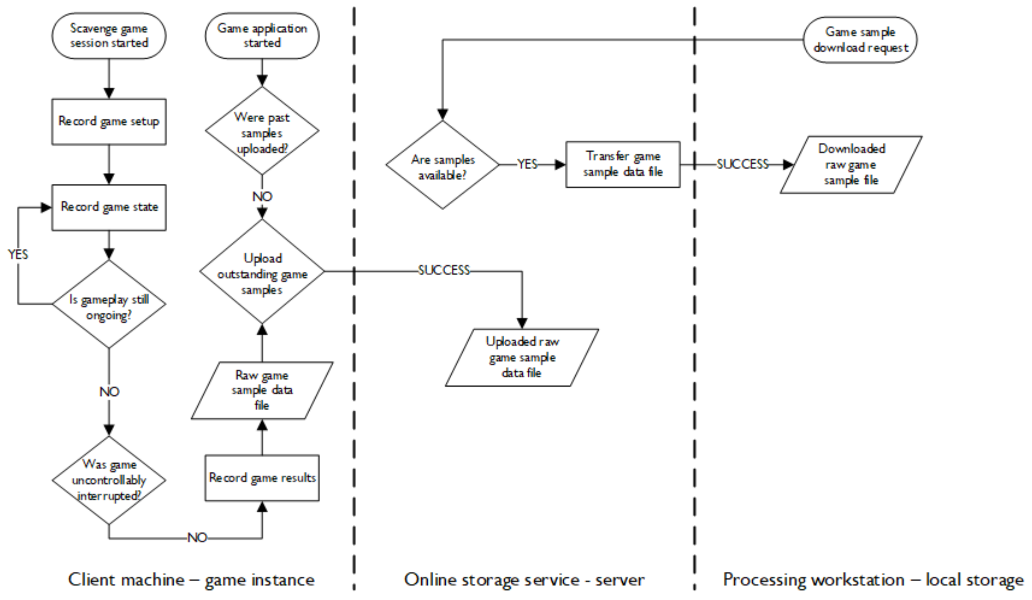

Scavenge game session started
Record game setup
Record game state
YES
Is gameplay still ongoing?
NO
Was game uncontrollably interrupted?
NO
Game application started
Were past samples uploaded?
NO
Upload outstanding game samples
Raw game sample data file
Record game results
SUCCESS
Uploaded raw game sample data file
Are samples available?
YES
Transfer game sample data file
SUCCESS
Downloaded raw game sample file
Game sample download request
Client machine – game instance
Online storage service - server
Processing workstation – local storage

Figure 4.8: *Scavenge* gameplay telemetry data collection pipeline flow chart.

Data logging was triggered at the start of the *scavenge* gameplay and continued until the game was over. Readily available game setup data was immediately captured, while emerging information was recorded continuously. Interaction events invoked by the player avatar were logged point to point, and the navigation trace was sampled at a frequency of $f = 100$ ms. Playthrough completion information was added at the conclusion of gameplay. At this point, the full game sample was serialised into a file and stored offline, on the user's hard drive. An attempt to transmit the generated file to the developer server was made as soon as the local file input-output operations were completed.

The data collection procedure was automatically invoked for any started *scavenge* game, in any game mode. Abandoning or restarting a playthrough was possible through an in-game on-screen user interface, accessible when the game was paused. In such cases, the game sample was still recorded but appropriately tagged as aborted by the user. As a result of such interruption, certain pieces of data, most importantly the full navigation trace, would not have been recorded in comparison to a finished game. External interruptions, such as system or hardware-level shutdown of the game application, were not handled by the game data logging, effectively resulting in a complete loss of the sample and any data collected for it.

Revisions

In the course of our research work, we assisted in improving the data collection mechanism and revising the format of the raw game sample data. This was necessary to eliminate issues with the data collection implementation, as well as the raw data sample format. Due to problems encountered, only samples recorded in the final format version, available since 2017, were considered valid for use in our research and were used in our work. We detailed the extent of updates that had been introduced to the data collection pipeline in the Data Collection Errors section.

4.5.3 Processed Data

Format

The processed game sample data was a product of extracting the original, raw sample data and processing it in a series of steps to generate a structured JSON file. JSON is a widely adopted open standard file and data interchange format. It was chosen as the target format to simplify the process of handling the data, interfacing it with third party analytics libraries and code, and to later simplify sharing our dataset with the research community. In comparison to raw data, processed data has an enforced data group structure. All data points contained within a data group are individually tagged, according to the JSON format standard. No custom separators or non-standard aggregations are necessary. While all relevant data content from raw samples is replicated in the processed samples, some data elements were found to be redundant and were removed.

The structure of processed game samples is documented in Appendix A.

Game Design Data

To introduce normalised game context values into our data processing pipeline, and later analysis work, we extracted all relevant game design values from the game *60 Seconds!*. The values were sourced from the game's implementation and design documentation, provided by the developer. All extracted design values were numerically indexed to normalise their representation and potential references. Extracted game design values were serialised using the JSON file format and each game design value category was stored in a dedicated file. Stored value collections included:

- **Difficulty data**: identification and basic setup of each difficulty level that can be selected for a specific game.

- **Game type data**: identification of all game types that can be played and influence the conditions of a specific game.
- **Item data**: details of each *item* type available to be collected in the game.
- **Environment data**: layout details for all environments found in the game.

The structure and contents of game design data files are documented in Appendix A.

Processing

Initial investigation of the raw dataset shared by the developer, as well as the format of data samples recorded, revealed that we would not be able to use them directly without additional intervention. In order to make the acquired raw data usable in our research work, raw game samples were subject to a multi-step transformation process involving the following stages:

- Reformatting
- Clean-up
- Normalisation
- Inference

To optimise processing, the transformation stages were not called sequentially but rather invoked for specific portions of the process. This allowed us to rapidly recognise any issues manifested at each step of the process. If the process was completed with no issues, a processed JSON data file was generated. Any error detected immediately halted the transformation process. In such a case, no output was generated for the processed game sample, and the discovered problem was reported in the processing log.

The data processing pipeline was fully implemented using Python. Our implementation extensively used the Anaconda scientific computing package for configuring the operating environment, combined with custom functionality to support data extraction and processing.

Reformatting

When working with the raw dataset, we quickly encountered problems with sampling and operating on subsets of data. This motivated us to introduce an alternative serialisation of game sample data into a well-structured and serviced storage format. Raw game samples were first read into memory and parsed into textual tokens. Following the iteration of the known structure of raw data files, we then encoded each data element into a JSON file stream.

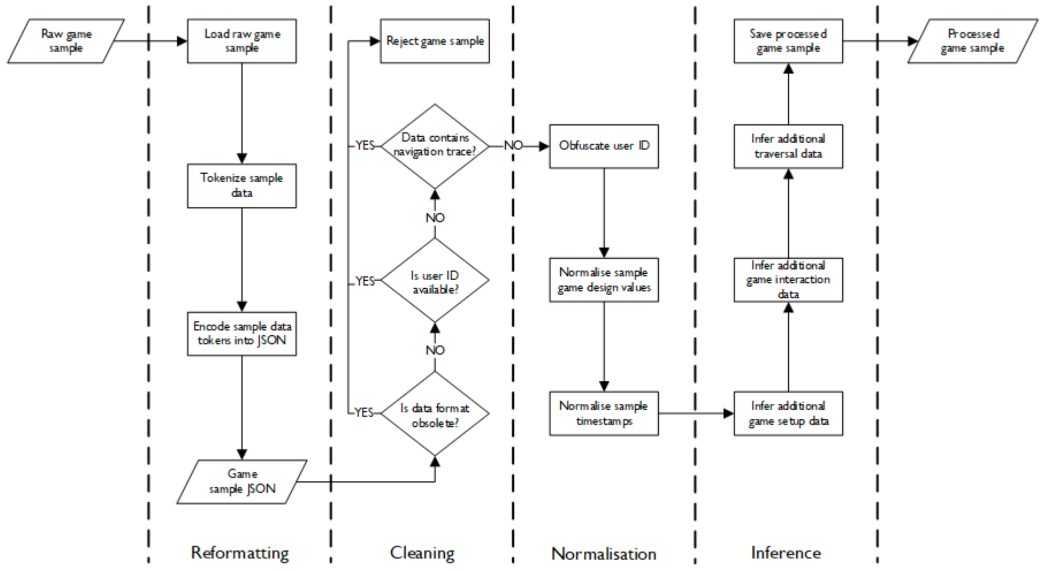

Raw game sample
Load raw game sample
Tokenize sample data
Encode sample data tokens into JSON
Game sample JSON
Reject game sample
YES
Data contains navigation trace?
NO
Is user ID available?
Is data format obsolete?
Obfuscate user ID
Normalise sample game design values
Normalise sample timestamps
Infer additional game setup data
Infer additional game interaction data
Infer additional traversal data
Save processed game sample
Processed game sample
Reformatting
Cleaning
Normalisation
Inference

Figure 4.9: Gameplay telemetry data processing pipeline flow chart.

Normalisation

Only a limited portion of each raw data sample was altered during the reformatting stage, but in some cases, specific data elements were subject to normalisation procedures to produce revised data representation. This included:

- **User identifier**: Steam user identifier was obfuscated using a hashing algorithm with a secret key input to anonymise users in research datasets, while preserving their unique reference identifiers.
- **Game design value inference**: some data elements, rooted in the game's design context, were compressed to a numerical format. This allowed for the removal of redundant data, while still referencing relevant game design data values in a globally normalised fashion. Referenced values included identifiers for in-game objects, environments and game setup data.
- **Raw timestamps**: raw data timestamps recorded directly from the game were expressed in terms of internal game time, measured from the launch of the game application to its closure. In order to operate on a comparable time frame for navigation and interaction traces, we normalised their timestamps to a clamped time range $[0, t_{end}]$, where t_{end} represents the normalised conclusion time of a specific game sample.

Clean-up

Reformatting and normalisation stages resolved a range of issues, significantly reducing the number of raw data sample processing errors. However, we were not able to address all the issues present. This included:

- Raw data sample serialisation with an obsolete file format.

- Data samples with no player avatar navigation trace.
- Data samples originating from non-Steam versions of the game, likely pirated.

The first two issues resulted in key portions of the sample data being missing, rendering such samples unusable. In the last case, we were unable to identify the user who generated the game sample. Considering the scale of the raw dataset, we decided that it was optimal to maintain the policy of full sample rejection if any of these issues were found in a data sample.

Inference

To complement the data found in raw samples, the processing routine was concluded with an inference stage, covering the derivation of complementary game setup, interaction and navigation information. While some of the data we inferred could have been captured at the time of in-game data logging, we decided against extending the data collection to accommodate that. We assumed it would have altered the data collection functionality in a way that could have affected non-experimental samples. It would have also resulted in increased sizes of raw data files, significantly increasing an already large dataset. All inferred data was included in the output, processed file, and was used in our analysis work.

The first part of the inference focused on filling in missing description data of the game setup. The second part extended game samples with additional data, such as traversal and timing information inferred from the analysis of navigation log sequences. We also inferred a collection of player interaction and navigational behaviour data for specific gameplay stages of *scavenge*, derived from the game's design and game's progression cues, provided to the player:

- **Preparation**: exploration time before collection begins. Allotted time based on the difficulty chosen - $t_{exp} = 0$ s, $t_{exp} = 5$ s or $t_{exp} = 10$ s.
- **Early game**: game flow from the beginning of the collection period at $t = 0$ s, until the game's mid-point, where $t = 30$ s.
- **Late game**: game flow from the game's mid-point $t_{scav} = 30$ s, until the conclusion of *scavenge* gameplay at $t = 60$ s.

Not all valid game samples underwent full inference processing. In some cases, inference had to be cancelled, because it was not possible to normalise the sample data:

- **Tutorial mode games**: tutorial games did not follow the game's regular *scavenge* ruleset, such as enforcing the time limit and could not be compared against games abiding by that time limit.

- **Paused games**: pausing a game violated the natural flow of gameplay and may have indicated cheating behaviours. It also resulted in a loss of objective mapping of time spent in the gameplay environment, introducing discrepancies when comparing paused games with non-paused games.

Samples with cancelled inference were not used in our analysis work. However, they remained in the dataset for potential future use in an alternative research context.

4.5.4 Data Handling Challenges

Data Volume

Due to the large volume of data contained in the mass-scale gameplay telemetry dataset, processing and exploration of relevant data had proven to be time-consuming. While operating on sampled subsets of data was feasible, any actions applied to the full game sample population dataset, or its larger portions, involved substantial amounts of time and compute. This constraint affected the amount of time we spent on data handling in the course of our research work.

Data Collection Errors

The data collection functionality, originally deployed in 2016, had suffered from several issues, which compromised collected data samples. These problems were only identified in the course of our data collection and processing work. While this allowed us to assist developers in refining the functionality and data formats, the work involved consumed a significant amount of time. Our initial efforts to salvage the existing dataset were not successful. It was not until the fourth iteration of the data collection mechanism, deployed in January 2017, that it had proven reliable.

We advised and assisted developers on resolving the following problems:

- **Server-side dating**: introducing server-side, reliable game sample file dating. Originally, developers trusted the player's system to provide a valid date and time of sampling. This had proven unreliable, as many players manually changed their system dates, most likely to circumvent application license protection.
- **Separator issues**: fixing missing separators between data elements, which corrupted data file formatting.
- **Data group tagging**: improving sample data file structuring by including identification tags for aggregated data groups. Prior lack of tags resulted in mixing of data groups in some edge cases.
- **Environment state**: expanding the scope of player interactions recorded in data samples, originally only limited to those invoked by the player avatar.

- **Unfinished games**: adding support for recording unfinished games, aborted by players before the gameplay was concluded in accordance with the game's designed flow and a fail or a win state.

Data Collection Gaps

Due to a number of server-side outages of the developer-hosted data transmission web service, the raw sample data collection suffered from several interruptions. As a result, some of the game samples generated by users were not uploaded to the online storage and were not included in the final sample dataset. We were unable to estimate the exact number of samples that were lost to these outages.

Withheld Data Files

It is likely that more raw game samples were not acquired due to user-side connectivity issues. Players without an Internet connection or playing offline may have never had their game traces uploaded. However, since Steam users are required to stay online to download and update their game library, we believe the number of such cases is limited. Nevertheless, this problem may have contributed to increasing data collection gaps, as well as the significance of the unknown player experience issue.

Unknown Player Experience

Since the raw sample data collection was not running at the time of the game's original release, it is likely that the final dataset was missing full game histories for an unknown number of recorded users. Additional reasons for missing games included data collection gaps and withheld data files. This could have been a significant issue if we were to analyse and compare individual player track records. However, since we were interested in analysing the population and its subsets, we assumed that the scale of the dataset and approximations, derived from trends exhibited by the user population, mitigated the impact of missing samples.

Hardware Issues

Due to the variance of the Steam user population, the range of hardware configurations used by players was most likely very wide. Successful recording of a game sample indicated that the computer used to play the game met the minimum hardware requirements of *60 Seconds!*, as outlined by the developer. However, we cannot assume that all the game samples were recorded with the game running optimally. *60 Seconds!* allows users to manipulate their in-game technical settings. A wrong technical configuration may result in subpar performance of the game. This could cause technical issues, such as frame rate drops, impacting the user experience and potentially user performance. We were not able to identify games suffering from such problems due to the lack of hardware information tracking. However, this had no impact on the data recording itself, since *scavenge* sampling frequency is bound to the game's time flow and other data points were recorded point to point.

Uncontrolled Data Collection Environment

Since players had their games recorded in their natural play environments, we were unable to exercise any control over their real-world play conditions. This made our experimental evaluation trials with real users inherently less controlled and vulnerable to interruption. However, the lack of control over the play environment was consistent with our intentions, as we assumed that users interacting with the game in a natural way generate realistic data. Minimal environment control, the absence of an observing and controlling party and no direct contact between the researcher and the player limited the impact of external factors on generated data. We also believe player motivations to play the game were natural, as they operated in an environment and the context of the game they knew, of their own volition. We assumed this approach to be valid for our research since the data collected was neither sensitive nor mission-critical, and our research work was focused on approximated data models.

Account Swapping

One of the issues resulting from the lack of full control over the data collection environment was that some users could have shared their Steam account with other users. This manifested in abnormally varied player skill and a high number of game sessions, originating from a single user account. To avoid contaminating the dataset with such unreliable data, all user accounts suspected of serving more than one player were classified as outliers and excluded from the dataset.

Lack of Qualitative User Data

We decided to focus on quantitative, data-driven methodology for our research and did not seek to acquire additional qualitative data from users. This stopped us from collecting user data that might have provided alternative insight into user behaviours, but it also reinforced our approach of not introducing additional bias to the users involved and the data acquired. Since that was the case, we did not consider it a limitation from the perspective of our research work.

Explicit User Consent and Age Verification

Conducting experimental evaluations using modified versions of the game's environment required explicit consent from the users involved. To address this, we integrated consent forms into the game's on-screen user interface. These forms asked players to confirm their intention to participate in an experiment and to verify that their age is above 16. These forms also provided information about the nature of our research, the goals and flow of the experiment and the researcher's contact details. Without providing explicit consent and age confirmation, players were not allowed to participate in any of our experimental trials. This was the only point of deviation from the natural flow of the game. Documentation of evaluation consent and information forms can be found in Appendix B.

User Data Collection Concerns

Despite the limited impact of research instrumentation and observation, we assumed some of the users participating in our experimental trials may have had concerns about being rated or compared to other players. Since our intention was to evaluate play skill exhibited in game samples, and to use it in further analysis, this was a valid concern we had to address. Experimental trial forms integrated into the game's user interface included information that individual data samples were to be anonymised and processed into a large dataset. We indicated there was no way for us, or third parties, to link individual game samples to specific users.

Data Processing Errors

A minor number of data processing errors were traced to user-side technical problems. This was only discovered during our data processing and analysis work. Local file input-output and transmission errors corrupted game sample files to a point, where we were not able to salvage their contents. As discussed before, we decided that the most optimal approach was to reject any data files that exhibited non-fixable errors. Since the number of excluded samples was relatively small for the size of our dataset, it was deemed acceptable. We incorporated selected sample rejection as part of the data cleaning procedure.

Data Confidentiality

Keeping user data confidential was a priority for the research and development team. It was achieved through a strict protocol of data handling. All data collected in the game was uploaded to the developer's secure, online storage. When downloaded for offline data storage and processing, the data resided exclusively on dedicated servers and workstations, physically located in the developer's offices. The game sample dataset was not distributed outside of the developer's network, and access to it was restricted to selected personnel. No user personal information was stored in the dataset. It was not possible to identify users from processed data, as samples were anonymised by identifier obfuscation through a one-way SHA-256 hashing function. As a result, the processed dataset contains no real identifiers or quasi-identifiers. We decided against applying more advanced privacy measures, such as k-anonymity [213], as additional suppression or generalisation of data samples would have limited our analysis capabilities. At the same time, the risk of identifying specific users from their recorded gameplay telemetry was deemed minimal.

Data Variance

With a large volume of collected data samples in uncontrolled environments, it was possible for the dataset to exhibit unexpected variance. We assumed this to be a risk and conducted our data analysis employing regular dataset sampling and testing methods to identify and process potential data skew.

4.5.5 Ethics

The Steam version of the game requires players to agree to the custom End User License Agreement (EULA) before launching *60 Seconds!* for the first time. The EULA, presented in Appendix B, contains a permission for collecting gameplay telemetry data for the purposes of analytics and research. This allowed developers to deploy the original data collection functionality and start accumulating game samples for internal use.

Utilising the developer's gameplay telemetry dataset in our research was deemed ethically appropriate, since it was sampled from an unaltered game, covered by the game's EULA. However, conducting experimental evaluation trials in a modified game environment necessitated additional scrutiny. As referenced in the Evaluation section, it was approved by our college's ethics committee. This led to an appropriate framing and presentation of the experimental procedure within the game environment, as well as integrating relevant consent forms. As documented in the Experimental Evaluations section, we had included consent and age verification checks, as well as information about the nature of our experiments, into the game's environment.

Data used for our research was subject to a data handling protocol agreed upon by the developers and the author. Raw datasets used for research work were never exposed to third parties and were stored securely on server storage controlled by the developers. Processed datasets were anonymised and stored on server storage, operated by the developer of the game.

4.5.6 Summary

In the course of our data collection and processing work, we have revised the data collection functionality in the game *60 Seconds!* and developed a processing procedure to transform acquired data samples into gameplay trajectories that would be usable in our research. The established pipeline allowed us to collect a raw game sample dataset of 11,925,961 samples from the live audience of the commercial version of the game *60 Seconds!*. Data was collected between January 2017 and May 2022. The accumulated raw game sample dataset was then transformed using our processing pipeline in May 2022. The output processed dataset contained 8,244,111 valid game samples for 808,659 unique users. Approximately 30.87% of raw trajectories were rejected, due to samples originating from non-Steam users (19.23% of collected samples), obsolete file format (11.18% of collected samples) and other processing errors (0.46% of collected samples).

The processed gameplay telemetry dataset, generated in the course of our collection and processing work, constitutes the research contribution **C3**, and one of the primary outputs of our thesis. More information about the dataset is provided in Appendix A.

4.6 Conclusions

This chapter established the foundations of our research and initiated the preparatory work required for the empirical, analysis and evaluation efforts carried out in Chapter 5: Game Score Study and Chapter 6: Agent Study. Content documented in this chapter includes:

- Methodology, procedures, and requirements of the planned research work.
- Evaluation of the context-guided agent model, taking advantage of the instrumentation of the game environment.
- Context-guided agent design and deployment workflow proposal, to be used for developing the target agent model in Chapter 6: Agent Study.
- Instrumentation of a commercial game environment of the game *60 Seconds!* to conduct data collection and processing, experimental evaluations, and training of agent AI models.
- Processed gameplay telemetry dataset to be investigated in Chapter 5: Game Score Study and later used to train agent models in Chapter 6: Agent Study.

This chapter has addressed research question **RQ1** from a theoretical perspective, by demonstrating a design approach for integrating learning into the game industry state of the practice ad hoc behaviour AI architecture, while providing industry viable execution and performance guarantees of integrated learning.

In the course of this chapter, we have documented the following research contributions:

- **C1**: industry applicable context-guided learning agent design and deployment workflow proposal; presented in the Context-Guided Agent Design section.
- **C2**: extension of the BT learning node concept, featuring additional safety redundancies; presented in the Safe Learning Integration section.

- **C3**: a mass-scale, processed gameplay telemetry dataset from the game *60 Seconds!* that was used in our research and later shared with the research community; presented in the Gameplay Telemetry Dataset section.

Chapter 5

Game Score Study

Summary. This chapter documents the investigation of the gameplay telemetry dataset, with respect to the game context-derived score metric. The proposed metric was used to quantify play skill exhibited in the recorded samples and to conduct a statistical analysis of the game score distributions to extract a baseline dataset. Finally, we clustered gameplay trajectories from the baseline dataset to establish a top play skill persona that could be used in the training of context-guided agents.

5.1 Overview

5.1.1 Goals

The objective of this chapter is to conduct a statistical analysis of the gameplay telemetry dataset in order to quantitatively investigate the play skill exhibited by the game's players and use it to establish the ground truth for modelling the game's top play skill persona. We achieve this by proposing a game score metric to objectively, quantitatively measure gameplay performance in individual play sessions, recorded in the gameplay trajectory dataset. We expressed the subgoal of observing play skill variations based on gameplay performance aligned score metric as a local research question RQG, to be addressed in this chapter:

- **RQG**: is there an observable, quantifiable difference between the gameplay performance-based play skill exhibited in gameplay trajectory samples from the *scavenge* segment of the game *60 Seconds!*?

The score metric constitutes research contribution C4. By applying it in the investigation of the gameplay telemetry dataset, we aim to derive relevant observations about game score distributions that would lead to clustering a normalised, baseline subset of the population dataset, with respect to play skill identified. Producing a play skill classifier constitutes research output O4 and will allow us to produce the top-skill play persona dataset. The analysis conducted in this chapter constitutes the research contribution C5.

Research contributions and outputs reported in this chapter include:

- **C4**: game score metric for quantitatively measuring play skill in terms of gameplay performance in the *scavenge* segment of the game *60 Seconds!*.
- **C5**: analysis of game scores measured for the gameplay telemetry dataset from the *scavenge* segment of the game *60 Seconds!*.
- **O4**: k-nearest neighbours classifier of play skill in the *scavenge* segment of the game *60 Seconds!*.

5.1.2 Structure

The chapter is divided into the following sections, detailing the investigation of the gameplay trajectory samples, with respect to the game scores recorded:

- **Game Score**: defines a game score metric for quantification of play skill in each recorded gameplay telemetry trace and establishing the basis for numerical analysis in our research.
- **Game Score Exploration**: explores the gameplay telemetry sample population with respect to game score distributions observed in the dataset.
- **Baseline Dataset**: extracts a normalised, baseline dataset from the gameplay telemetry sample population to investigate the game score distributions further.
- **Play Skill**: investigates the concept of play skill in the game and conducts clustering on the baseline dataset to identify different levels of play skill.
- **Top-skill Persona**: extracts a persona gameplay trajectory dataset, representative of the ground truth for generating a top play skill persona game-playing agent model.

5.2 Game Score Metric

5.2.1 Play Skill

The element of tension and solution governs game playing. Tension is uncertainty, and players strive to "succeed" in order to resolve it [109]. Player proficiency in achieving success should be estimated with respect to the context of the game by defining a task success metric, derived from the context of the user's interactions [248] or aligned with the game's gameplay goals [64]. Such an estimation of a player's gameplay performance can be considered indicative of their play skill in the game environment. In multiplayer games, establishing user play skill can be based on comparing them against other players, as is the case with the Elo rating system for Chess [70]. In single-player games with custom game environments, it is usually necessary to derive a local play skill metric based on the game context, unless an alternative, data-driven approach can be applied.

Sources of uncertainty in games are varied, ranging from incomplete information to chance factors. Skill-based games predominantly rely on play skill, and chance has limited or no influence on their outcomes [65]. Use of chance elements in video games to maintain uncertainty across multiple playthroughs and promote replayability is standard practice in the game industry [78]. It is most commonly achieved through randomisation of the state of the game environment. Impact of chance differs from game to game, but the mere presence of randomness does not imply a game is not skill-based [46].

5.2.2 Play Skill in *60 Seconds!*

Our investigation into play skill observed in gameplay trajectories from *60 Seconds!* was limited to the *scavenge* segment of the game, since only the data from that portion of the game was available to us. As stated in the Data Collection and the Gameplay Telemetry Dataset sections in Chapter 4: Context-Guided Agents, developers of the game had not implemented a data collection mechanism for the *survival* segment of the game. As we were not allowed to instrument *survival*, it was beyond the scope of our research to obtain and use data from that portion of the game.

The majority of uncertainty the player is faced with in each playthrough of *scavenge* stems from the imposed time pressure, while trying to balance out environment exploration, strategising item collections and deposits, and ensuring the safety of the avatar before the time runs out. Player interactions are in no way influenced by chance. The starting state of the environment is a result of random level selection and randomised object placement in the level. This is a common feature of many skill-based games, which vary their start state to enhance replayability [46]. It may be perceived as unpredictable by inexperienced players. However, since there is a fixed number of levels and object placement locations for each of the levels, there is a finite number of environment randomisation combinations. The player will eventually be able to observe repeating environment combinations, reducing uncertainty generated by randomness. Thus, we considered the *scavenge* segment of *60 Seconds!* to be pre-dominantly skill-based. And so, we assumed the *scavenge* play skill had the potential to be a relevant measure for investigating the game and its dynamics.

5.2.3 Metric Proposal

Scavenge gameplay performance in *60 Seconds!* was not directly quantified by the game itself. To quantify and measure it in terms of play skill, we had to define a custom measure of play skill for the game. We proposed to derive a quantifiable and normalised custom measure to score player-generated gameplay trajectories with respect to the game's design context and values [102], which were available to us from the game data assets. The devised metric was based on measuring the player's success using completion score s_{com} and collection score s_{col}, and then combining them into a full score metric s for each game trajectory recorded. Full game score, and its completion and collection components, were all normalised to a value range of (0.0, 1.0).

5.2.4 Completion Score

The completion objective of the game's *scavenge* section is measured as a binary value, which provides an inherently objective identification of the outcome as a success, or failure [248], which corresponds to a game session ending with a failure or a success state. There is no additional qualification for the degree of failure or success. A success is objectively and intrinsically desirable as a manifestation of the player's competence in the game. It also provides extrinsic motivation in the form of rewarding the player with progression into the *survival* section of *60 Seconds!* [102]. A failure concludes the game with a fail state. Based on that, we quantified them as minimum and maximum values on the normalised value range, assigning the completion score of $s_{com} = 0.0$ to a game that was not completed, and $s_{com} = 1.0$ to a completed game.

5.2.5 Collection Score

Scavenge gameplay sees players foraging and depositing actions as many *items*, as possible, within the time limit t_{scav}. Each type of *item* has a weight w assigned to it, which consumes the player's limited inventory space $i = 4$. While game progression is not possible without completion, on its own, it does not represent the full extent of play skill exhibited. Hence, we required a complementary success metric [248]. We could not directly convert the number of *items* collected into an objective score value, as varying weights make some *items* potentially more valuable than others. However, since these weight values are rooted in the game's context and gameplay mechanics, we decided to use them to establish a valid collection metric.

We initially investigated an alternative approach that would have combined the cost of *item* collection in *scavenge* based on weight, and its utility value in *survival*. Although we had access to *survival* design data, it was not possible to objectively estimate *item* utility, beyond the number of times it appeared as an option in different *events*. However, for many *events* the value of *item* use was not constant, as its effects could vary based on the game state, affected by the player's decisions and randomness. Another way to define a custom metric was to derive it in a data-driven fashion from the gameplay trajectory dataset. While this would have been the preferred option, we decided against it due to the limitations of the dataset. Since we did not have access to any *survival* gameplay trajectories, it was impossible to determine how specific *item* sets C collected in *scavenge* would have impacted player gameplay performance in *survival*. Without such a grounding, we would have had to estimate values of *items* based on their collection counts recorded in the dataset. This metric would have suffered from a potential bias of how players perceive the value of an *item*, versus how valuable it truly is from the perspective of gameplay. With the information at our disposal, we were unable to estimate or even confirm whether such a bias was present in individual trajectories. Thus, we decided to use a relatively simple metric, directly rooted in the game's design context.

To calculate the collection score, we had to consider the set of *items* collected C by the player from the set of all *items* available to be collected T in a *scavenge* environment e. To arrive at a normalised collection score, we divided the sum of weights of *items* collected C by the sum of weights of all *items* to be collected T:

$$s_{col} = \frac{\sum_{n=1}^{n} w(C_n)}{\sum_{m=1}^{m} w(T_m)}.$$

This base formula is applicable to *scavenge* and *full* type games, as long as they follow a uniform setup routine. That was the case in *60 Seconds!*. The default set of collectable *items* available T_d was always weighted the same, since the environment was populated with the same number of *items* for every *scavenge* and *full* game session. The only discrepancies originated due to specific game setup parameters:

- **Extended *item* set**: an additional *item*, the harmonica, was included in T, increasing the sum of weights of all *items* to be collected by $w = 1$.
- **Character selection**: depending on the avatar character selection, a different character, with a different weight w, is added to T. If *Ted* is selected, *Dolores* becomes a collectable *item* with a $w = 2$. If *Dolores* is selected, *Ted* becomes a collectible *item*, with a $w = 3$.

However, alternative setups were possible. Changing the number of *items* to be collected could have effectively skewed comparisons of the collection score between two samples with different values of T. To normalise the collection score for such comparisons, we introduced the normalisation factor s_n by dividing the sum of weights of all *items* to be collected T from a given sample by the default *item* weight sum T_d:

$$s_n = \frac{T}{T_d}.$$

To calculate a normalised collection score for a sample with a $T \neq T_d$, the revised s_{col} calculation formula was proposed:

$$s_{coln} = s_{col} * s_n.$$

While the collection score could technically reach the maximum value of 1.0, it is unlikely for players to score that high. Most *scavenge* environments in the game were designed so that it would not be possible for players to collect all the *items* from T in the available *scavenge* time t_{scav}.

5.2.6 Full Score

The full score s was calculated as the arithmetic mean of collection score s_{col} and completion score s_{com}:

$$s = \frac{s_{com} + s_{col}}{2}.$$

We initially considered utilising a weighted average for the full score, prioritising collection over completion score. However, the game's context offered no objective indication of how to weigh them against each other. Because all our assumptions were either based on numerically objective measures or game design data, we decided to quantify them equally. Full score, as well as its completion and collection components, were recorded for each scored game sample.

5.2.7 Scoring Game Samples

All valid game samples from the processed gameplay telemetry dataset were scored, as long as it was possible to infer additional, relevant information from their original trajectories. The game sample scoring procedure followed the score calculation methods outlined above. We integrated the scoring procedure into the processing pipeline, incorporating the score calculation as the final inference step. Instead of being deployed individually, scoring was included in the pipeline to reduce processing time and handling overhead. Scores calculated for each game sample were recorded as part of that sample, becoming an integral part of the processed gameplay telemetry dataset.

5.2.8 Dataset Sampling

The processed gameplay telemetry dataset augmented with game scores was then prepared for selective sampling with respect to the requirements of different stages of our analysis work. The implemented sampling mechanism enabled us to extract data subsets with respect to multiple parameters provided.

We identified a list of global parameters to be enforced for all valid subset sampling, in order to exclude outliers observed in the course of our dataset processing work. Identified global parameters include:

- **Game not paused**: only games that were not paused could be used for analysis, due to sample issues discussed before, and the resulting lack of full inference and score data.
- **Game finished**: only games featuring a valid conclusion could be used for analysis, since games that were not finished did not contain full inference and score data.
- **Movement recorded**: only games with a navigation trace that recorded a non-zero player avatar movement could be considered, as the opposite indicated a complete lack of player's gameplay engagement, either due to a software problem or a focus interruption on the user's side.

Data subsets used for analysis were stored individually and are part of the digital supplement to the dissertation. They are documented in Appendix A.

5.2.9 Summary

We have proposed a game score metric, derived from the design context of the *scavenge* segment of the game *60 Seconds!*, to quantitatively measure play skill exhibited in individual game sessions, including those recorded in gameplay trajectory data samples. While the proposed metric is a custom measure, it is objectively rooted in the design and values established by the developers of the game. Beyond these pre-existing assumptions, no additional qualification of the play skill context was introduced to prevent the inclusion of unfounded assumptions and ensure the validity of the metric. As a simple point value, the measure can be numerically handled in a straightforward manner, facilitating further analysis and supporting the data-driven, quantitative nature of our research. For insight into a specific component of the score, we can refer to either the completion or collection score values, rather than the full score. The proposed game score metric constitutes the research contribution **C4**.

5.3 Game Score Exploration

5.3.1 Assumptions

The first goal of our work with the processed gameplay telemetry dataset was to conduct a top-down exploration of the game score data sample population. This initial inquiry was expected to direct our follow-up investigation, allowing for an informed utilisation of game scores as an analysis and evaluation measure. We began by defining inclusion criteria for what was considered a valid dataset, representative of the population. Selected parameters were to be used to sample this population dataset from the origin gameplay telemetry dataset, and to allow us to analyse the distributions of full, collection and completion game scores. Conclusions drawn from this initial exploration were expected to shape our inquiry into the nature of completed and not completed game distributions.

5.3.2 Sampling

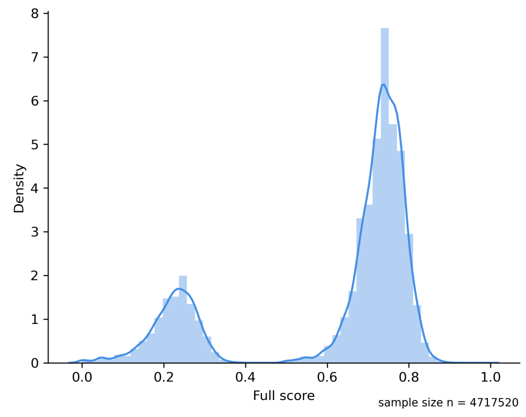
We initiated the exploration of processed data by extracting game samples that would be used for the analysis, from the complete origin dataset ORIG, to create the population dataset POP (see table 5.1 on page 93). Although invalid data was rejected during processing, the dataset ORIG still contained data points that would not have been useful in analysis, due to their lack of inference data. This included paused, unfinished, or unrecorded movement game samples. To make the newly produced dataset representative of the entire, relevant sample population, we decided against prematurely enforcing strict sampling parametrisation. Sampling was only parametrised with respect to game type, in an effort to extract reasonably normalised sample collection. Dataset POP targeted *full* and *scavenge* games, since these two game types follow the same gameplay ruleset, making their game samples comparable. They also happened to compose the most numerous game sample subset in dataset ORIG, totalling 4,717,520 samples (57.22% of dataset ORIG). Game samples of game types *tutorial* and *scavenge challenge* were excluded in the process. The former had no inference data to be analysed, due to a completely different gameplay flow. The latter uses an alternative ruleset and offers a different gameplay experience, which is not directly comparable with *full* and *scavenge* games.

Dataset created	POP
Sampled dataset	ORIG
Sample size	4,717,520
Values	s
Sampling parameter	**Value**
Game type	*Full* or *scavenge*
Paused	False
Finished	True
Game move distance	$d > 0$
Measurement	**Value**
Min	datasetPOPmin
Max	datasetPOPmax
Mean	datasetPOPmean
Median	datasetPOPmedian
Mode	datasetPOPmode
Standard deviation	datasetPOPstdv

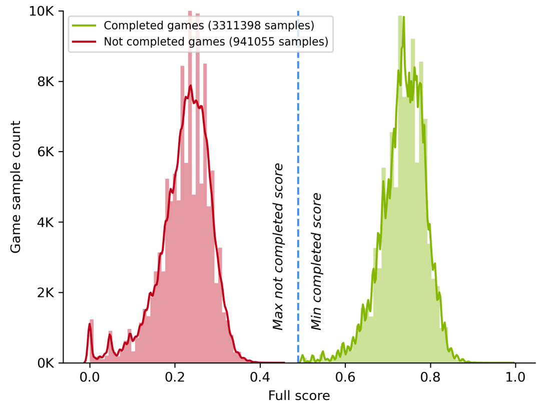
Table 5.1: Sampling parameters and description of full score dataset POP.

5.3.3 Full Score

When we first proposed the game score metric, we assumed that its full score s component would serve as the primary quantifiable measure for analysis. However, preliminary, visual observations (see figure 5.1 on page 94) of the full score distribution of dataset POP samples revealed a bimodality of the distribution. Two distinct peaks were observed, one for completed and one for uncompleted games. The disconnect and the resulting shift were introduced due to the binary, discrete nature of the completion score and how it was weighed against the collection score. In terms of full score, a game that was not completed would have never scored higher than $s = 0.5$, and a completed game would have never scored lower than $s = 0.5$. We considered a single value to be a valid choice to reduce the dimensionality of score measurement. However, the presence of a bimodal distribution, which disrupts a direct interpretation of emerging distributions, prompted us to reconsider and analyse score data with respect to the collection score instead. We were still able to follow through with our full score handling assumptions, since full score and collection score were expressed in a uniform, normalised value range. The departure from the intended reduction of dimensionality resulted in observing two game sample-dependent variables: completion score and collection score.



(a) Population full score distribution.



(b) Full score distributions of completed and not completed games.

Figure 5.1: Dataset POP population full score kernel density distribution estimation visualisations.

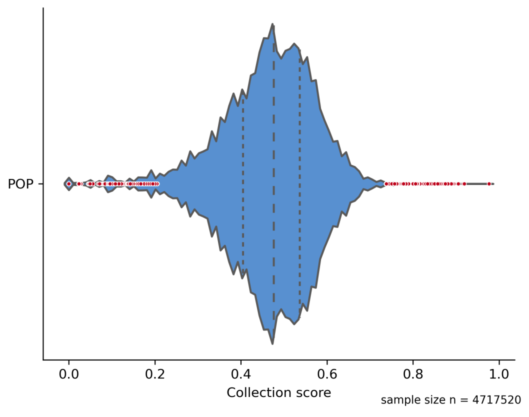
5.3.4 Collection Score

The inspection of the dataset POP collection scores distribution (see table 5.2 on page 95) revealed that the range of scores recorded $s_{col} = [0.0, 0.98]$ approximately covered the permitted game collection score range. Despite that, an observable number of data points approached central tendency with 50% of sample scores located in the range $s_{col} = [0.4, 0.54]$. Visualisations of

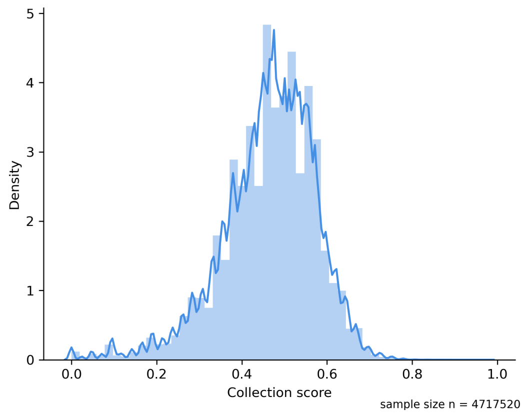
Dataset	POP
Sample size	4,717,520
Values	s_{col}
Measurement	**Value**
Min	0.0
Max	0.98
Mean	0.46
Median	0.48
Mode	[0.5]
Standard deviation	0.11

Table 5.2: Sampling parameters and description of the collection score dataset POP.

the underlying data distribution (see figure 5.2 on page 95) indicated the existence of a negative, left skew, caused by a concentration of lower scores. While the general shape of the distribution seemed to approach normality, the observed skew and the volume of outliers detected (2.64% of dataset POP data points) had the potential to affect the character of the distribution. These factors and the apparent lack of equality between mean (m_{POPcol} = 0.46) and median (M_{POPcol} = 0.48) central tendency measures of the dataset suggested that the collection score distribution POP was non-normal, non-symmetric, and skewed. To determine if the observed outliers had any significant impact on the POP collection scores, we continued the investigation of the nature of the distribution. Based on our research assumptions, we did not trim detected outliers, as they appeared to be valid data points. Left-skewed concentration of lower scores was representative of beginner-level players, who were still learning to play the game, and in the process achieved low scores. This was consistent with a learning curve dynamic.



(a) Distribution with data outliers.



(b) Kernel density distribution estimation.

Figure 5.2: Dataset POP population collection score distribution visualisations.

We first reviewed whether the distribution was normal, to inform further analysis method selection. We confirmed its non-normality visually with a quantile-quantile (QQ) plot (see figure D.1

on page 209), and quantitatively using the Shapiro-Wilk test [83,201] (α = 0.01, n = 4,717,520, p = .001, s = 0.98). We compared the distribution against 80 distributions in the *SciPy* library and found no match[1]. To investigate the potential non-symmetry of POP we calculated the Fisher-Pearson coefficient of skewness [284], arriving at a non-zero, negative value S_{POP} = -0.81. We conducted a two-sided skewness and kurtosis statistics-based test of distribution normality assumptions [50] and confirmed under conditions examined (α = 0.01, s = 4,717,520, p = .001, s = -632.97) the presence of a non-normal distribution skew.

Dataset created	POP-NO
Sampled dataset	POP
Sample size	4,592,880
Values	s_{col}
Sampling parameter	**Value**
s_{col}	s_{col} > OF_{POP-1} & s_{col} < OF_{POP-2}
Measurement	**Value**
Min	0.21
Max	0.73
Mean	0.47
Median	0.48
Mode	[0.5]
Standard deviation	0.09

Table 5.3: Sampling parameters for dataset POP-NO.

To establish if the source of the skewness was caused by the presence of outliers we repeated POP sampling but excluded outlier values beyond upper outer fence OF_{POP-2} and lower outer fence OF_{POP-1} of POP (see table 5.3 on page 96). We assumed the means of POP m_{POP} and POP-NO m_{POPNO} to be equal, which would have indicated the outliers had no significant effect on the collection score distribution POP. In such a case, extreme collection scores would not bias the distribution, rendering its mean less reliable for evaluating the central tendency of collection scoring. Since POP was in violation of normality assumptions, and POP-NO was sampled from it, we decided to extrapolate both these datasets via bootstrap m by n resampling with replacement (see table 5.4 on page 97) with m = 1499 resamples at Cochran calculated subsample sizes of n_{POPB} = 16,530 for POP and n_{POPBNO} = 16,528 for POP-NO. The resampled data samples were then used to calculate the standardised mean difference, using the Cohen d standardised effect size measure. The outputted values formed distribution POP-B, which approached normality per the Central Limit Theorem. This allowed us to conduct an assumption-free analysis on the source data and test if outliers had an impact on the collection score distribution mean, expressed as Cohen d d_{POPB} between POP and POP-NO.

[1]Using distribution fitter code authored by Thomas Cokelaer [41].

Source data		Bootstrapped distribution POP-B							99% CI$_{POPNOB}$	
Data	***n***	**Statistic**	***m***	**Mean**	**Median**	**SD**	**Min**	**Max**	**Lower Bound**	**Upper Bound**
POP	16,530	Cohen *d*	1499	-0.08	-0.08	0.01	-0.12	-0.04	-0.12	-0.08
POP-NO	16,528									

Table 5.4: *m* by *n* bootstrapping for testing the standardised difference of means between POP and POP-NO.

We tested the alternative hypothesis of inequality of means between POP and POP-NO using a $\alpha = 0.01$ significance level two-tailed test with resample size $m = 1499$. Bootstrap hypothesis testing was applied to establish and compare confidence intervals of the resulting Cohen d value distribution POP-B using the percentile method. Confidence intervals were found to be CI$_{POPNOB}$ = [-0.12, -0.08] for the POP-B value distribution, and CI$_{POPNON}$ = [-0.04, -0.0] for the distribution under null hypothesis. Under conditions examined, the test results (p = .001) rejected the null hypothesis, indicating that the outlier influence on the mean collection score in POP was statistically significant. However, the values in the interval CI$_{POPNOB}$ represented an effect size below what was considered a small effect (d = 0.2), by Cohen [40].

Source data		Bootstrapped distribution POP-BMSE							99% CI$_{POPBMSE}$	
Data	***n***	**Statistic**	***m***	**Mean**	**Median**	**SD**	**Min**	**Max**	**Lower Bound**	**Upper Bound**
POP	16,530	MSE	1499	0.02	0.02	0.0	0.02	0.02	0.02	0.02
POP-NO	16,530									

Table 5.5: *m* by *n* bootstrapping for MSE calculation of POP and POP-NO.

To evaluate the bias of the outliers, we estimated the mean squared error (MSE) confidence interval between datasets POP and POP-NO. Once again, we utilised bootstrap *m* by *n* resampling with replacement (see table 5.5 on page 97) with m = 1499 resamples at a uniform, Cochran calculated subsample size of n = 16,530 to maintain consistency for MSE calculations. The resampled data produced distribution POP-BMSE, with confidence interval $CI_{POPBMSE}$ = [0.02, 0.02], which featured a small MSE in relation to the collection score range (see table 5.5 on page 97).

In the investigation of the POP collection score distribution, we observed a left skew, caused by a limited number of valid data points that manifested as low collection score outliers. We asserted they were generated by beginner-level players, who were at the bottom of the learning curve of the game. The outliers were found to have a significant effect on the distribution mean, but with an effect size below what is considered small. It was also confirmed that the bias of these outliers was minimal. This supported the notion that we would be able to use the distribution in our further analysis without additional normalisation to compensate for the abnormal skew.

5.3.5 Completion Score

The binary nature of the completion score divided the data samples into completed and uncompleted games. We decided that the most transparent and intuitive way of expressing the dynamics of completion was to use the probability of completing the game *P(com)*. Since we were operating on high-volume datasets, representative of the general population, we observed the frequency of completed games in the sample dataset D as a valid estimate of *P(com)* in D. Leveraging this assumption we estimated the 99% confidence interval of global probability of completion in dataset POP by extrapolating the data with m by n bootstrapping, producing dataset CI_{POPBP} with m = 1499 resamples at n = 16,530 sample size (see table 5.6 on page 98). We then calculated its confidence interval CI_{POPBP} = [0.76, 0.77] using the percentile method. The decision to establish confidence intervals using bootstrapping for *P(com)* was consistent with our research assumptions.

Source data		**Bootstrapped distribution POP-BP**							**99% CI$_{POPBP}$**	
Data	***n***	**Statistic**	***m***	**Mean**	**Median**	**SD**	**Min**	**Max**	**Lower Bound**	**Upper Bound**
POP	16,530	*P(com)*	1499	0.77	0.77	0.0	0.76	0.79	0.76	0.77

Table 5.6: m by n bootstrapping for establishing the 99% confidence interval CI$_{POPBP}$ of *P(com)* for dataset POP.

To complement these results, we were interested in observing the breakdown of the completion probability values across discrete regions of the dataset POP collection score values to see how the probability of completion matched against the collection scores achieved. We assumed that achieving higher collection scores would be correlated with higher completion scores. To that effect, we isolated collection score bins, totalling n_{cbPOP} = 312 across dataset POP, and calculated their individual *P(com)* values. The output distribution POP-CB of completion probabilities of POP collection score bins (see table 5.7 on page 99) had half its data points estimated to be greater than M_{POPCB} = 0.76, reinforcing the notion of consistency in high probabilities of completion. Additionally, visualisations of POP-CB (see figure 5.3 on page 99) presented with a rising trend in probability value, which increased with each subsequent bin of higher collection score. A noticeable anomaly presented itself in the 12 bins in the low collection score range of s_{col} = [0.03, 0.1] reporting higher completion scores (s_{com} >= 0.5) than expected. Closer examination revealed these to be games where players minimally engaged in collection, and in most cases quickly proceeded to the exit. It is indicative of a fraction of beginner-level players (0% of POP dataset samples) who were likely still learning the rules of the game and figuring out how to strategise their interactions in the limited time provided.

Due to the non-Gaussian properties of the dataset POP, we applied the non-parametric Spearman's correlation coefficient to investigate a potential correlation between the completion probability and collection score values. The calculated value $\rho = 0.87$ indicated a strong, positive, monotonic correlation between the rising completion probability and mean values of specific collection score bins in dataset POP, supporting our assumption of a meaningful correlation of the sampled data.

Dataset created	POP-CB
Sampled dataset	POP
Sample size	312
Values	*P(com)*
Measurement	**Value**
Min	0.32
Max	1.0
Mean	0.73
Median	0.76
Mode	[1.]
Standard deviation	0.14

Table 5.7: Sampling parameters and description of *P(com)* values distribution for the population's collection score bins of dataset POP.

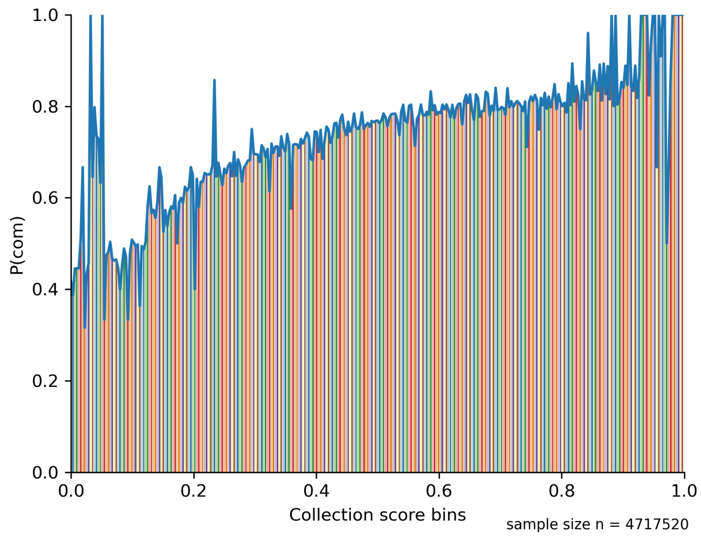


Figure 5.3: Visualisation of dataset POP *P(com)* values for the population's collection score bins.

We examined the completion score distribution of the POP dataset and committed to the use of an intuitive value of the probability of completing the game *P(com)* to represent completion. *P(com)* exhibited a rising trend, and its values strongly, positively correlated with rising mean collection scores, supporting the assumption of higher play skill manifesting jointly in terms of both completion and collection scores.

5.3.6 Completed and Not Completed Games

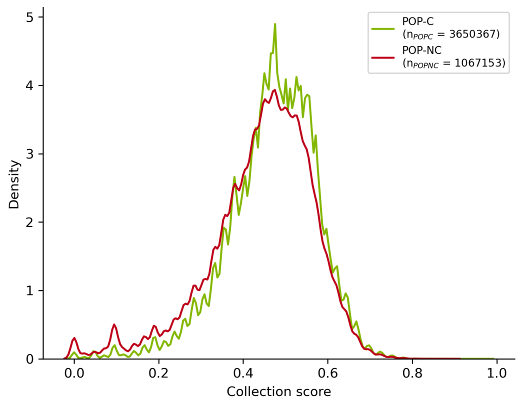
Sampling and Statistical Comparison

Dataset created	POP-C
Sampled dataset	POP
Sample size	3,650,367
Values	s_{col}
Sampling parameter	**Value**
s_{com}	1.0
Measurement	**Value**
Min	0.0
Max	0.98
Mean	0.47
Median	0.48
Mode	[0.5]
Standard deviation	0.1

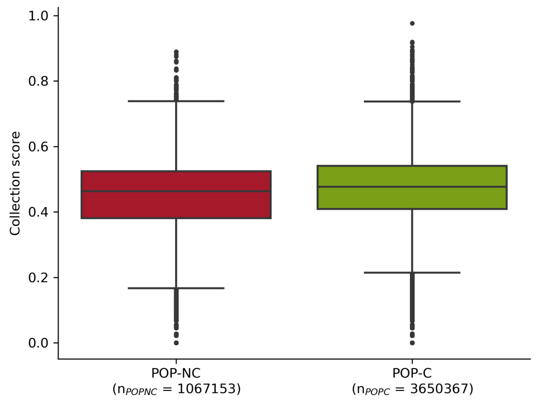
Dataset created	POP-NC
Sampled dataset	POP
Sample size	1,067,153
Values	s_{col}
Sampling parameter	**Value**
s_{com}	0.0
Measurement	**Value**
Min	0.0
Max	0.89
Mean	0.44
Median	0.46
Mode	[0.5]
Standard deviation	0.12

Table 5.8: Sampling parameters and description of datasets POP-C and POP-NC.

To individually examine, and then compare, completed and not completed game sample data, we selectively sampled the population dataset POP into completed games dataset POP-C and not completed games dataset POP-NC (see table 5.8 on page 100). Although the sample count for completed games was significantly higher (3.42 times more completed than not completed samples), the general shape of distributions of POP-C and POP-NC appeared similar (see figure 5.4 on page 100). Both approximately approached a left-skewed normal distribution, and their shapes resembled the shape of the POP collection score distribution.



(a) Kernel density estimations of distributions.



(b) Distribution boxplots.

Figure 5.4: Selectively sampled collection score distributions for completed POP-C and not completed POP-NC game samples.

To repeat our analysis procedure used for the POP collection scores distribution, and to inform our method selection, we visually observed violations of normality assumptions in QQ plots of both POP-C and POP-NC (see figure D.2 on page 210), which were also confirmed quantitatively with the Shapiro-Wilk test for POP-C (α = 0.01, sample size n = 3,650,367, p = .001, s = 0.98) and POP-NC (α = 0.01, sample size n = 1,067,153, p = .001, s = 0.96) under conditions examined. Both POP-C and POP-NC were observed to have approximately symmetrical data distributions, which did not conform to the notion of parametric skewness. To numerically interpret the symmetry of POP-C and POP-NC we calculated the Fisher-Pearson coefficient of skewness for both of them, arriving at negative, non-zero values for POP-C S_{POPW} = -0.72 and POP-NC S_{POPL} = -0.88. These results, indicating a left skew, were quantitatively confirmed with two-sided skewness and kurtosis statistics based tests for POP-C (α = 0.01, sample size n = 3,650,367, p = .001, s = -508.72) and POP-NC (α = 0.01, sample size n = 1,067,153, p = .001, s = -321.72), which indicated a non-normal distribution skew in both investigated datasets, under conditions examined. This contradicted the assumptions of symmetry in POP-C and POP-NC.

One observable difference between the two distributions was an upwards value shift of the POP-C values, discernible from the statistical description of central tendency measures of the examined distributions ($m_{POPW} > m_{POPL}$ and $M_{POPW} > M_{POPL}$), as well as their boxplot visualisations. Despite this shift, both POP-C and POP-NC exhibited their primary density peaks at approximately similar collection score value points. Standard deviation values and visual spread of both distributions suggested high dispersion of data, with the range of scores recorded encompassing nearly the entire collection score range. However, the coefficients of variation for POP-C CV_{POPW} = 0.22 and POP-NC CV_{POPL} = 0.27 indicated limited data spread, relatively similar in both cases, caused by a high concentration of average scores around central tendency. The two distributions also contained observable amounts of outliers (2.11% of data points in POP-C, and 3.3% of data points in POP-NC), just like their source distribution POP. To determine the relevance of these outliers in these selectively sampled, non-normal datasets, we repeated the procedure of outlier investigation, previously applied to distribution POP, involving bootstrap resampling and confidence interval testing. First, POP-C and POP-NC distributions were sampled with their outliers excluded (see table 5.9 on page 102).

For the purposes of validating the assumption of outliers present in the distributions significantly influencing their value distributions, we assumed the absence of a significant effect would translate into the equality of means between the mean of POP-C m_{POPW} and the mean of POP-C-NO m_{POPWNO}, as well as the mean of POP-NC m_{POPL} and the mean of POP-C-NO m_{POPLNO}. To investigate this notion, we extrapolated them via bootstrap m by n resampling with replacement using m = 1499 resamples, to compensate for the normality violations present in the two distributions. The resampled data samples were then used to calculate the standardised mean difference, using Cohen d standardised effect size measure. Output values formed distributions POP-C-B (see table 5.10 on page 102) and POP-NC-B (see table 5.11 on page 102), both of

Dataset created	POP-C-NO
Sampled dataset	POP-C
Sample size n$_{POPWNO}$	3,573,245
Values	s_{col}
Sampling parameter	**Value**
s_{col}	$s_{col} > OF_{1-POP-W}$ & $s_{col} < OF_{2-POP-W}$
Measurement	**Value**
Min	0.21
Max	0.74
Mean	0.48
Median	0.48
Mode	[0.5]
Standard deviation	0.09

Dataset created	POP-NC-NO
Sampled dataset	POP-NC
Sample size n$_{POPLNO}$	1,031,983
Values	s_{col}
Sampling parameter	**Value**
s_{col}	$s_{col} > OF_{1-POP-N}$ & $s_{col} < OF_{2-POP-N}$
Measurement	**Value**
Min	0.17
Max	0.74
Mean	0.46
Median	0.47
Mode	[0.5]
Standard deviation	0.1

Table 5.9: Sampling parameters and description of datasets POP-C-NO and POP-NC-NO, derived from POP-C and POP-NC, respectively, with their outliers excluded.

which approached normality per the Central Limit Theorem.

Source data		**Bootstrapped distribution POP-C-B**							**99% CI$_{POPWB}$**	
Data	***n***	**Statistic**	***m***	**Mean**	**Median**	**SD**	**Min**	**Max**	**Lower Bound**	**Upper Bound**
POP-C	16,513	Cohen *d*	1499	-0.06	-0.06	0.01	-0.1	-0.03	-0.1	-0.06
POP-C-NO	16,511									

Table 5.10: *m* by *n* bootstrapping for testing the standardised difference of means between POP-C and POP-C-NO.

Source data		**Bootstrapped distribution POP-NC-B**							**99% CI$_{POPLB}$**	
Data	***n***	**Statistic**	***m***	**Mean**	**Median**	**SD**	**Min**	**Max**	**Lower Bound**	**Upper Bound**
POP-NC	16,334	Cohen *d*	1499	-0.1	-0.1	0.01	-0.14	-0.07	-0.14	-0.1
POP-NC-NO	16,325									

Table 5.11: *m* by *n* bootstrapping for testing the standardised difference of means between POP-NC and POP-NC-NO.

We analysed them under the alternative hypothesis of outliers impacting the collection score distribution mean, expressed as Cohen d d_{POPWB} between POP-C and POP-C-NO not being equal to 0, and Cohen d d_{POPLB} between POP-NC and POP-NC-NO not being equal to 0. The resulting Cohen d value distribution POP-C-B (see figure D.3 on page 210) did not appear to have their standardised mean difference values converging around the measured distance of $d_i = 0$. We formally tested the hypothesis with a significance level two-tailed test ($\alpha = 0.01$, sample size $n = 1499$), using the bootstrap hypothesis testing method. The confidence interval of the output distribution POP-C-B $CI_{POPWNOB}$ = [-0.1 , -0.06], and the confidence interval of the distribution under the null hypothesis $CI_{POPWNON}$ = [-0.03, 0.0] were established. The null hypothesis was rejected (p = .001) under the conditions examined, indicating the outlier influence on the mean collection score in POP-C was significant. Yet, the full range of $CI_{POPWNON}$ values was classified below the small effect size ($d = 0.2$), according to Cohen [40].

The visualisation of POP-NC Cohen d value distribution POP-NC-B (see figure D.3 on page 210) indicated that it was not contained within the confidence interval $CI_{POPLNON}$ = [-0.04, 0.0] under the null hypothesis. We conducted a two-tailed bootstrap hypothesis test (α = 0.01, sample size n = 1499). Under conditions examined, the null hypothesis was rejected (p = .001), indicating the outlier influence on the mean collection score in POP-NC was significant, as was the case with POP-C. Again, the value range of $CI_{POPLNON}$ was found to be in the effect size range classified below what is considered a small effect size (d = 0.2) by Cohen [40].

Source data		Bootstrapped distribution POP-WBMSE							99% CI$_{POPWBMSE}$	
Data	***n***	**Statistic**	***m***	**Mean**	**Median**	**SD**	**Min**	**Max**	**Lower Bound**	**Upper Bound**
POP-C	16,513	MSE	1499	0.02	0.02	0.0	0.02	0.02	0.02	0.02
POP-C-NO	16,513									

Table 5.12: m by n bootstrapping for MSE calculation of POP-C and POP-C-NO.

Source data		Bootstrapped distribution POP-LBMSE							99% CI$_{POPLBMSE}$	
Data	***n***	**Statistic**	***m***	**Mean**	**Median**	**SD**	**Min**	**Max**	**Lower Bound**	**Upper Bound**
POP-NC	16,334	MSE	1499	0.03	0.03	0.0	0.02	0.03	0.02	0.03
POP-NC-NO	16,334									

Table 5.13: m by n bootstrapping for MSE calculation of POP-NC and POP-NC-NO.

To determine the extent of the bias of outliers in the completed and not completed collection score distributions, we calculated the MSE confidence intervals between the base distributions, including outliers, POP-C and POP-NC, and their respective samples excluding outliers, POP-C-NO and POP-NC-NO. The bootstrap m by n resampling with replacement was used in both cases, with m = 1499 resamples at a uniform, Cochran calculated subsample size for each bootstrapping to maintain consistency for MSE calculations. The resampled data produced completed games derived distribution POP-WBMSE (see table 5.12 on page 103), with confidence interval $CI_{POPWBMSE}$ = [0.02, 0.02], and not completed games derived distribution POP-LBMSE (see table 5.13 on page 103), with confidence interval $CI_{POPLBMSE}$ = [0.02, 0.03], both of which indicated a small MSE in relation to the collection score range.

Exploration of the completed POP-C and not completed POP-NC game score distributions revealed they were non-symmetrical and featured non-normal left skews. Just like in their source dataset POP, these skews were caused by low-score outliers, contributed by players with lower play skill. They were found to significantly impact the mean collection score, but their effect size was below small, and bias was found to be minimal. This confirmed that we would be able to use these distributions in our further analysis without additional normalisation to compensate for an abnormal distribution skew. It was observed that the examined distributions were both characterised by a high dispersion of data, accounting for a wide range of scores achieved. This suggested that for both completed and not completed games, players' scores varied, and a high collection score did not always imply completion would follow. We also observed a limited data spread, which was consistent with a high concentration of data points around the central tendency. This supported the notion that the majority of players eventually advancing on the learning curve beyond beginner play skill, and reaching an approximately similar, average play skill level. Future investigation focussed on tracking user progression in terms of play skill could analyse how many of the low-score data points in the dataset were representative of early games of users, who later advanced on the learning curve. However, this would require verifying which users had a continuous and preferably complete set of playthroughs recorded in the dataset, which was beyond the scope of our research.

Collection Score Difference

The visual presentation of completed and not completed collection score distributions POP-C and POP-NC (see figure 5.4 on page 100) suggested a similarity between the two. On the contrary, statistical descriptions (see table 5.8 on page 100) revealed differences between central tendency measures of these distributions, potentially indicating the existence of a significant difference between them, with POP-C collection scores being higher. To validate that assumption numerically and formally, we moved to investigate the equality of means of POP-C (m_{POPW}) and POP-NC (m_{POPL}) collection score datasets. Previously detected normality violations of both distributions prompted us to use bootstrap hypothesis testing for the task, based on bootstrap m by n resampling with replacement of relevant data with m = 1499 resamples. Standardised mean difference, expressed as Cohen d standardised effect size measure, was calculated for each resampled dataset. The output values formed distribution POP-COL-B (see table 5.14 on page 104), approaching normality per the Central Limit Theorem. We analysed it under the assumption that POP-C distribution exhibited higher collection scores than those of POP-NC. We expressed this claim as the alternative hypothesis of Cohen d d_{POPCB} standardised collection score mean difference between POP-C and POP-NC being greater than 0.

Source data		**Bootstrapped distribution POP-COL-B**							**99% CI_{POPCB}**	
Data	***n***	**Statistic**	***m***	**Mean**	**Median**	**SD**	**Min**	**Max**	**Lower Bound**	**Upper Bound**
POP-C	16,513	Cohen d	1499	0.23	0.23	0.01	0.19	0.26	0.19	0.23
POP-NC	16,511									

Table 5.14: m by n bootstrapping for Cohen d calculation between POP-C and POP-NC.

We formally tested the hypothesis with a right-tailed test, using the bootstrap hypothesis testing method (α = 0.01, resample size m = 1499). The Cohen d distribution POP-COL-B (see figure D.4 on page 211) standardised mean difference values, contained within the confidence interval CI_{POPCB} = [0.19, 0.23] were far removed from the confidence interval under null hypothesis CI_{POPCBN} = [-0.03, 0.0]. Test results (p = .001) rejected the null hypothesis under conditions examined, indicating the collection score distribution of POP-C scored higher than that of POP-NC. The effect size of CI_{POPCB} was classified above small effect size (d = 0.2), according to Cohen [40].

As intuitively assumed, collection scores achieved in completed games were found to be significantly higher under conditions examined, which supported the notion of the established, positive correlation between rising collection and completion scores. This suggested that at higher play skill levels, players could be expected to score higher in terms of both completion and collection, in comparison to lower play skill levels.

5.3.7 Summary

We examined the origin dataset ORIG and selectively sampled it to produce an actionable dataset POP. In our exploration of dataset POP we detected a bimodality of the full score s distribution caused by the binary nature of the completion score. This informed our decision to investigate the collection scores of completed and not completed games' distributions, rather than use the full score in our further analysis. We observed learning curve dynamics in both the collection scores and probability of completion distributions. Collection score distributions' left skew, caused by low scores from beginner-level players, and a high concentration of scores around the central tendency, which was positioned approximately in the middle of the permitted score range, suggested the game's balance enabled a progression from beginner to a higher play skill level. Most players were able to achieve scores that were approximately average, with some highly skilled users advancing beyond the average. Completion was not an issue for players, as the majority of recorded games were completed. This was consistent with the simplicity of achieving completion in the game, which only required the player to proceed to the exit area. However, few players attempted completion without collection, and it was more likely for a completed game to be scored higher in terms of collection. We found a strong correlation between the rising completion probability and mean collection scores, further supporting the learning curve dynamic and an observable play skill improvement.

5.4 Baseline Dataset

5.4.1 Assumptions

Each game session of *60 Seconds!* can be configured with respect to several game setup parameters. Some, or all, of them could be considered independent variables potentially impacting the conditions of play, and in consequence, game scores. To establish a normalised baseline for further investigations, we set out to identify a default set of parameter values used for game setup. The default value selection was expected to approximate the default game setup flow. A secondary consideration was the volume of samples with specific parametrisation found in the processed sample dataset. We assumed the character of the baseline score data distribution might be different from that of the population score dataset POP, due to selective sampling. However, we anticipated that the baseline dataset would exhibit similar characteristics to those observed in the population POP dataset.

5.4.2 Sampling

We examined each of the game setup parameters individually, with respect to the context of the game and the contents of the processed gameplay telemetry dataset, narrowing down the inclusion criteria for the baseline dataset:

- **Game type**: only *scavenge* and *full* game types were considered, as they follow the standard ruleset and conditions of gameplay. This makes them preferable from the perspective of the proposed game score metric. Since most samples were generated in *full* games (n = 4,245,215, 89% of dataset POP samples), this game type was chosen as the default.

- **Difficulty**: when players open the game launch screen, they are presented with a difficulty level set by default to the *normal* difficulty. This likely contributed to the fact that most *full* game samples from dataset POP (n = 2,024,363, 47% of samples) were played in *normal* difficulty. This positioned the *normal* to be considered the default difficulty.

- **Character selection**: as with difficulty, the character choice is pre-selected to *Ted*, when presented. While the player can manually change this parameter, the proposed selection likely contributed to the fact that the majority of *full*, *normal* game samples from dataset POP were played with that character selected (n = 1,010,060, 89% of samples). This informed our choice to designate the pre-selected value of *Ted* to be the default value for this parameter. The choice also influenced the normalisation of T_d. We did not alter this parameter in the live version of the game to be randomised, as it was beyond the scope of modifications agreed upon with the developer. However, the remastered version of the game, which was released later, did in fact feature a random selection of the playable character to avoid defaulting to only one of them.

- **Extended *item* set**: the majority of samples from dataset POP, parametrised as *full*, *normal* and *Ted* character games, were played with the extended *item* set (n = 1,130,503, 55% of samples). The sample volume, automatic enforcement, as well as the preference to normalise T_d made an active extended *item* set the default value for this parameter.

- **Level**: all game levels for the *full* game type have an equal chance of being pseudo-randomly chosen to be played, and players have no influence on that selection. All *scavenge* levels had an approximately similar sample count parametrised for *full*, *normal*, *Ted* character and extended item set games from dataset POP (see figure 5.5 on page 107).

Since that was the case, we decided to use samples from the full level range for the purposes of our research. for each level in the game, there is a fixed number of *item* placement points, where *items* can be pseudo-randomly positioned. There are more placement points available than there are *items* to position. The number of placements is approximately similar across all levels, but placement positions and the amount of specific *item* placements vary, due to differences in environment staging. For every game, the same number of *items* of a specific type are placed. While there is a finite number of *item* placement combinations for each level, differences between levels make accounting for them a complex endeavour. Since the placement points are specific to each of the levels, based on our assumptions of approximated modelling, we assumed the impact of pseudo-random *item* placement, influencing the starting state of the environment, would have been embedded within and represented by the level itself.

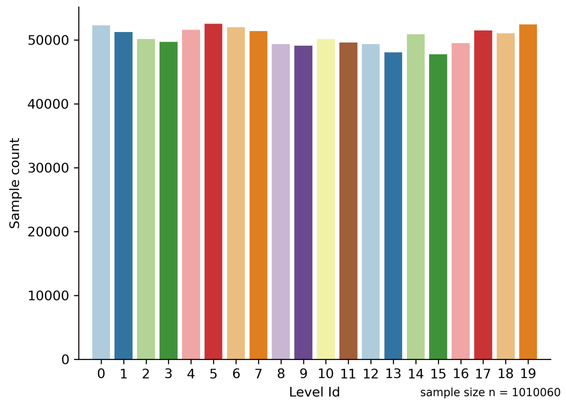


Figure 5.5: Sample counts for game levels parametrised for *full*, *normal*, *Ted* character and extended *item* set games from dataset POP.

Using the identified set of default parameter values, we then sampled dataset POP to produce the baseline dataset BAS (see table 5.15 on page 107)

Dataset created	BAS
Sampled dataset	POP
Sample size	1,010,060
Sampling parameter	**Value**
Game type	1 (*full*)
Difficulty id	1 (*normal*)
Extended *item* set	True
Default character	True (*Ted*)
Level id	[1, 20]

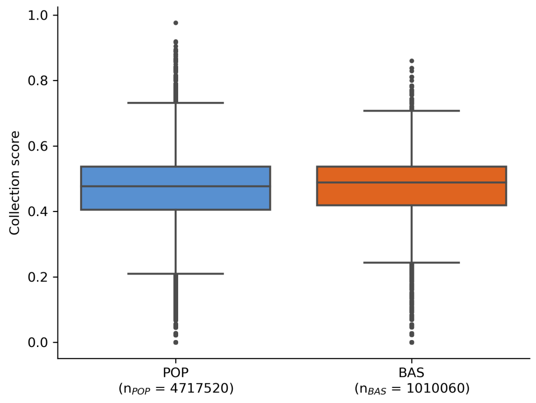
Sampled dataset	BAS
Sample size n_{BAS}	1,010,060
Values	s_{col}
Measurement	**Value**
Min	0.0
Max	0.86
Mean	0.47
Median	0.49
Mode	[0.46511628]
Standard deviation	0.1

Sampled dataset	BAS
Sample size	1010060
Values	s_{com}
Measurement	**Value**
Min	0.0
Max	1.0
Mean	0.78
Median	1.0
Mode	[1.]
Standard deviation	0.42

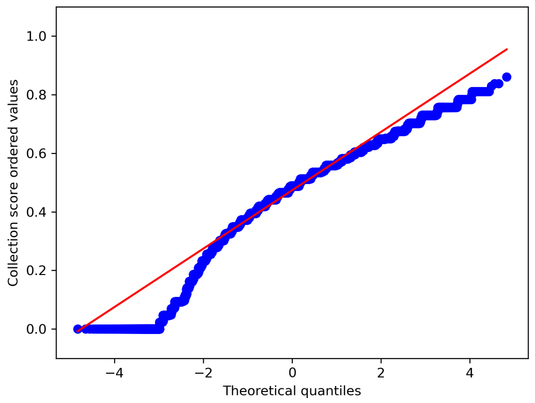
Table 5.15: Default game setup variable sampling parameters for dataset BAS and statistical description of its s_{col} and s_{com} values' distributions.

5.4.3 Collection Score

We assumed the baseline dataset collection score distribution would exhibit similar properties to that of the general population. To inform further analysis method selection, and to compare the baseline distributions to those of the POP dataset, we investigated normality of BAS collection score distribution visually using the QQ plot (see figure 5.6 on page 108) and quantitatively with the Shapiro-Wilk test (α = 0.01, sample size n = 1,010,060, p = .001, s = 0.96) under the conditions examined.



(a) Comparison of collection score distributions using boxplots for datasets POP and BAS.



(b) Dataset BAS collection score distribution QQ plot.

Figure 5.6: Dataset BAS collection score distribution visualisations.

Visual inspection of the collection score value distributions of the population dataset POP and baseline dataset BAS suggested that collection scores achieved in dataset BAS were higher than those recorded in dataset POP. To compare their unevenly sized and non-normal collection score data, we extrapolated both datasets using bootstrap m by n resampling with replacement (see table 5.16 on page 109) with m = 1499 resamples at Cochran calculated subsample sizes of n_{POP} = 16,530 for dataset POP, and n_{BAS} = 16,320 for dataset BAS. The resampled data samples were then used to calculate the standardised mean difference, using the Cohen d standardised effect size measure. The output values formed the distribution BAS-COL-B, which approached normality per the Central Limit Theorem. This allowed us to conduct an assumption-free analysis of the source data, under the alternative hypothesis of the higher value of the mean of the collection score distribution BAS, over the collection score distribution POP, expressed as Cohen d d_{BASB}.

The confidence intervals of the resulting Cohen d values distribution BAS-COL-B (see figure D.5 on page 211) CI_{BASB} = [0.06, 0.09], and the distribution under null hypothesis CI_{BASBN} = [-0.03, -0.0] were established. The full data range of distribution BAS-COL-B was found to be outside the boundaries of the confidence interval CI_{BASBN}. Bootstrap hypothesis right-tailed test (α = 0.01, resample size m = 1499) rejected the null hypothesis, under conditions examined

Source data		Bootstrapped distribution BAS-COL-B							$99\%\ CI_{BASB}$	
Data	***n***	**Statistic**	***m***	**Mean**	**Median**	**SD**	**Min**	**Max**	**Lower Bound**	**Upper Bound**
POP	16,530	Cohen *d*	1499	0.09	0.09	0.01	0.06	0.13	0.06	0.09
BAS	16,320									

Table 5.16: m by n bootstrapping for Cohen d between collection score distributions POP and BAS.

(p = .001). This suggested that the collection score distribution of the baseline distribution BAS was significantly higher than the collection score distribution of the population dataset POP. The range of standardised mean difference values in CI_{BASB} all represented effect size below small significance (d = 0.2), according to Cohen [40].

5.4.4 Completion Score

The completion probability *P(com)* for the population dataset POP was estimated to be within a 99% confidence interval CI_{POPBP} = [0.76, 0.77]. By bootstrapping the *P(com)* data for baseline dataset BAS with m = 1499 resamples by n = 16,320, we produced the distribution BAS-BP. Using the percentile method, we established the completion probability of dataset BAS to be within the 99% confidence interval CI_{BASBP} = [0.77, 0.78] (see table 5.17 on page 109).

Source data		Bootstrapped distribution BAS-BP							$99\%\ CI_{BASBP}$	
Data	***n***	**Statistic**	***m***	**Mean**	**Median**	**SD**	**Min**	**Max**	**Lower Bound**	**Upper Bound**
BAS	16,320	*P(com)*	1499	0.78	0.78	0.0	0.77	0.79	0.77	0.78

Table 5.17: m by n bootstrapping for establishing the 99% confidence interval CI_{BASBP} of *P(com)* for dataset BAS.

The completion probability confidence interval CI_{BASBP} for baseline dataset BAS appeared to be approximately similar to CI_{POPBP} for population dataset POP. To determine if this was the case, we proposed the alternative hypothesis of a difference between the completion proportions of dataset BAS and dataset POP being non-zero. To compare the completion scores from the unevenly sized datasets POP and BAS, we had conducted a bootstrapped, two-tailed, proportion mean difference hypothesis test (α = 0.01), involving m = 1499 for Cochran calculated sample sizes for the datasets involved (see table 5.18 on page 110, and figure D.6 on page 212). The null hypothesis was not rejected under conditions examined (p = .951), indicating that completion probabilities in datasets POP and BAS were statistically similar.

To compare the breakdown of the completion probability values in dataset BAS collection scores, we repeated our steps from the Game Score Exploration work with dataset POP and established collection score bins in dataset BAS, totalling n_{cbBAS} = 201. We then calculated their individual *P(com)* values (see table 5.19 on page 110) generating output distribution BAS-CB (see figure 5.7 on page 111). Similarly to POP, we observed anomalous fluctuations of data,

Source data		Bootstrapped distribution BAS-COM-B								99% $CI_{BASCOMB}$	
Data	***n***	**Statistic**	***m***	**Mean**	**Median**	**SD**	**Min**	**Max**	**Lower Bound**	**Upper Bound**
POP	16,530	Proportion mean difference	1499	0.67	0.67	0.98	-2.36	3.59	-2.49	2.29
BAS	16,320									

Table 5.18: *m* by *n* bootstrapping for proportion difference calculation between completion score distributions POP for the population and BAS for the baseline.

which were more scattered than those in POP. Some of them could have been attributed to beginner players, as was the case in POP, but additional spikes were visible for higher values of s_{col}. Despite these fluctuations of data, BAS-CB appeared to exhibit a positive correlation between high collection score and completion probability for the given score bin, similar to what we observed in dataset POP. To quantitatively confirm this observation, we calculated the Spearman's correlation coefficient $\rho_{BAS} = 0.61$. The result suggested a strong, positive correlation between the collection score value associated with a given score bin and the rising probability of completion.

Dataset created	BAS-CB
Sampled dataset	BAS
Sample size	312
Values	*P(com)*
Measurement	**Value**
Min	0.32
Max	1.0
Mean	0.73
Median	0.76
Mode	[1.]
Standard deviation	0.14

Table 5.19: *P(com)* values for dataset BAS collection score bins.

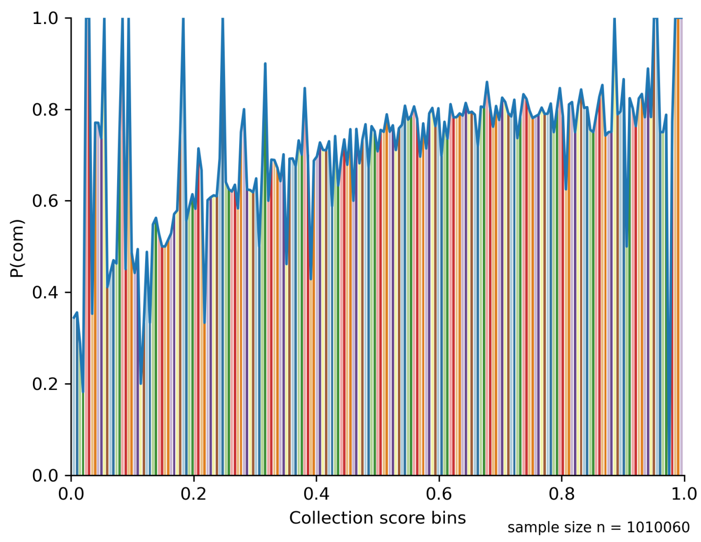


Figure 5.7: Visualisation of dataset BAS *P(com)* values for the baseline collection score bins.

To formally determine whether the observed correlation was similar to the trend exhibited in dataset POP we formulated a hypothesis of the lack of equivalence of the correlation coefficients of the two datasets. The alternative hypothesis was expressed as $\rho_{POP} \neq \rho_{BAS}$. Using Fisher's r to z two-tailed transformation test (α = 0.01) we compared the values of ρ_{POP} and ρ_{BAS} and found that under the conditions examined, the null hypothesis was rejected (p = .001). Despite both datasets exhibiting a strong positive correlation in terms of score bin collection score value and the completion probability associated with that score bin, their respective correlations were statistically different, under conditions examined.

5.4.5 Summary

We successfully sampled the baseline dataset BAS from dataset POP, using an informed selection of inclusion criteria, based on optimal game setup parameter values, derived from the game's context. While we could have furthered this optimisation by narrowing the inclusion criteria even more and increasing control over identified variables, it would have required additional investigation, which was beyond the scope of this thesis. The emergent baseline dataset BAS was considered representative of the population and suitable for completing our research objectives. While its collection score distribution appeared similar to that of POP, we found the collection scores of BAS to be higher. We had controlled for the level selection and included data from all possible level environments in BAS, which meant randomness was not a factor in this difference. However, limiting the difficulty level to *normal* likely influenced the scores, potentially removing lower scores from beginner players on *easy* difficulty and lower scores from failed attempts by more advanced players challenging themselves on *hard* difficulty. Restricting the game type to *full* might have also had an impact, though limited, since fewer samples were excluded compared to the difficulty sampling. Additional investigation would be required to accurately identify the degree of influence of each of these factors. However, since the effect size of higher collection scores in BAS was below small, and the numerical difference was minimal, we considered the baseline dataset to be a satisfactory representation of the investigated population. The significant similarity of completion probabilities in BAS and POP further confirmed this. Like POP, BAS also featured a strong, positive, monotonic correlation between the rising collection score and the probability of completing a game, indicating learning curve dynamics. However, in the case of BAS more completion probability fluctuations were present, which caused it to be quantitatively dissimilar to POP. While we attributed such anomalies to beginners in POP, BAS fluctuations appeared in higher collection score bins. This suggested that perhaps the beginner play skill level covers a broader range of collection scores than what was observed for the POP completion probability fluctuations.

5.5 Play Skill

5.5.1 Assumptions

The game's design does not directly indicate how to measure gameplay performance, expressed as play skill. While it is in the player's best interest to maximise collection score, the only objective requirement the game enforces is to complete the *scavenge* phase, in order to progress to the *survival* portion of the game. Reaching the failure state nullifies all the collection progress accumulated by the player. In terms of completion, good skill is hence equivalent to reaching the success state. For collection, the definition of good skill had to be established with respect to the collected data and formulating definitions of high and low scores, based on the proposed score metric. Having measured the difference between the sampled datasets of completed and not completed games, we were able to conclude that quantitatively, the distribution of completed games POP-C exhibits better gameplay skill in terms of the collection score than not completed games from POP-NC. While this has confirmed that differences between score distributions are measurable, it did not inform how to classify games with respect to collection play skill. Further data-driven investigation was necessary to derive a practical method of score result interpretation.

To identify play skill mapped score groups, we decided to conduct clustering on the normalised baseline dataset BAS, which we considered representative of the general population. Our assumptions about clustering collection score data were:

- Number of clusters was to be limited, to enable manual labelling them with relevant gameplay skill context descriptions (high score, low score, etc.).

- Since we expected a few clusters, we were interested in applying unsupervised methods that could be parametrised with predefined cluster count configurations.
- Final decision on cluster count and positioning was to be derived from quantitative and qualitative evaluation of the generated cluster data.

5.5.2 Sampling

Both completed and not completed games had to be considered for score clustering, as we were interested in establishing a global mapping of gameplay skill against score groups. For the purposes of the play skill investigation, we used the previously extracted baseline dataset BAS.

Dataset created	BAS
Sampled dataset	POP
Sample size	1,010,060

Sampling parameter	**Value**
Confidence interval	99%
Margin of error	1%

(a) Sampling parameters for dataset BAS

	Value
Min	0.0
Q1	0.42
Median	0.49
Q3	0.54
Max	0.86

(b) Quartile description for dataset BAS.

Table 5.20: Dataset BAS sampling and description details.

5.5.3 Play Skill Clustering

Clustering Approach

In our research, the focus was on clustering the collection score from the dataset BAS. The most computationally economical and optimal way to address this was to approach the data as a one-dimensional set of collection score values. The assumption that few clusters would be generated, allowed us to investigate simpler clustering methods for the task. Initial examination of the data was performed using quartile statistics to provide an overview of the dataset's clustering potential. For the primary clustering method, we considered two unsupervised learning algorithms: Jenks natural breaks optimisation [113] and k-means [178]. While the former is very applicable to one-dimensional data, the scale of the dataset BAS (n = 1,010,060) had proven the algorithm to be computationally uneconomical. Standard *SciPy* implementation of the k-means algorithm did not fare much better, as it is not commonly used for single-dimensional data. However, an optimised version of the algorithm proposed by Gronlund *et al.* can solve k-means clustering for single-dimensional data in polynomial time [90]. We chose to apply this solution in our work.

Since cluster counts were selected manually, we incorporated quantitative evaluation using two numerical metrics. First, the elbow method heuristic [242] using the sum of squares errors with

Euclidean distance calculation was included for preliminary numerical and visual determination of cluster quality. Second, for cluster similarity assessment, we applied the Davies-Bouldin index (DBI) to cover within and between cluster distance evaluation [53]. These evaluation methods can be applied to any clustering process, as they do not require a priori knowledge of the ground-truth labels and the necessary input is limited to the clustered data itself.

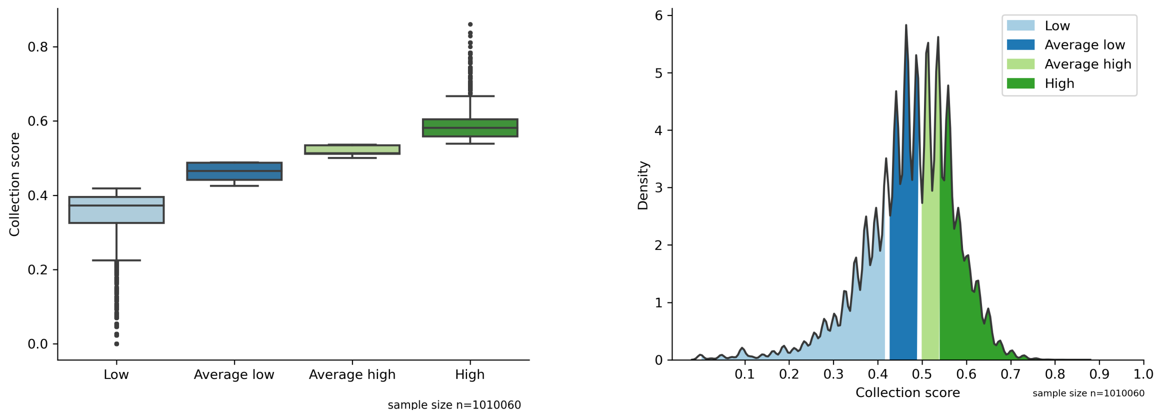
While our decision to take advantage of one-dimensional data and simpler clustering methods suited the purposes of our research and agent training, it also limited the amount of interesting and potentially useful observations we could have derived from a more sophisticated clustering approach. We assumed that the complexity of useful observation mapping would be executed as part of the learning model training in the **Agent Study**. Clustering was a means to an end, paving the way to classify play skill and use it to identify gameplay trajectories in the dataset that would be relevant for agent training. We initially considered but decided against clustering play skill based on both collection and completion scores. Instead, we focussed solely on the latter and reviewed completion probabilities for identified clusters. This was motivated by the inherent limitation of the completion score, caused by its binary value representation. However, if the investigation into the dataset had a wider scope and was more focussed on players, rather than individual game trajectories, it would have been preferable to cluster multidimensional data. We still would have defaulted to unsupervised methods to operate with minimal assumptions about the underlying model. However, more advanced clustering methods could have been involved, supporting the discovery of behavioural patterns in longitudinal studies of the user population. Such a follow-up investigation could result in producing more refined player personas to use in agent training, incorporating not only estimated gameplay performance data but also long-term behavioural aspects of user operations.

Quartile Statistics

For the quartile statistical approach, we chose to observe the quartiles of the dataset BAS, naturally clustering the dataset into four groups. Median was selected as a reference point of division over the mean, due to negative skew and outlier bias present in the distribution. Since there was not enough data to identify what constitutes a high score in the context of game design, it was reasonable to assume the higher-scoring 50% of the population would provide an approximated representation of better gameplay skill, equivalent to higher collection scores. Following that logic, we assigned the following labels to the clustering categories that emerged:

- ***Low*** - bottom 25% (Min - Q1 value range of dataset BAS).
- ***Average low*** - bottom, middle 25% (Q1 - Q2 value range of dataset BAS).
- ***Average high*** - upper, middle 25% (Q2 - Q3 value range of dataset BAS).

- ***High*** - top 25% (Q3 - Max value range of dataset BAS).



(a) Clustered collection score distribution boxplot breakdown.

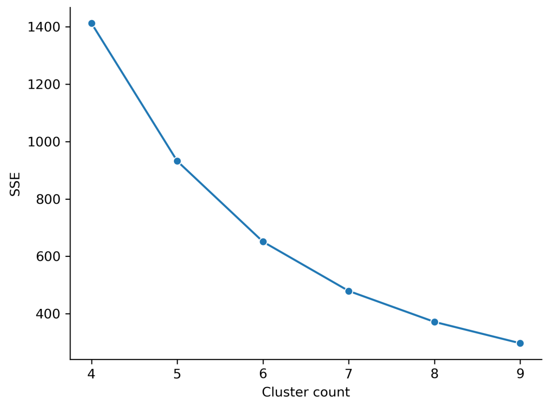
(b) Clustered collection score distribution according to cluster derived boundaries.

Figure 5.8: Distribution of manually assigned collection score categories in dataset BAS.

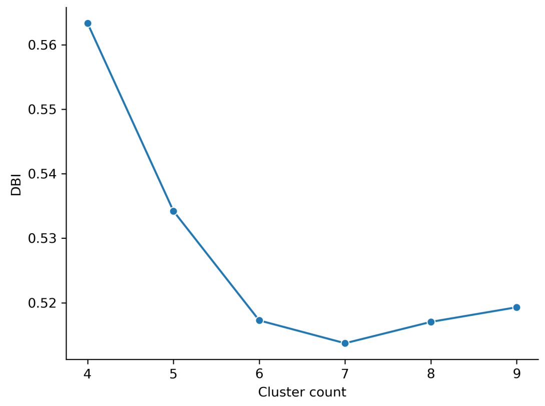
Visual examination of the plotted quartile ranges indicated a limited spread of the middle 50% of data points. Despite enjoying a high concentration of samples, it spanned a considerably smaller value range than *low* and *high* scores. For *low* scores, an upwards tendency was visible, with the median positioned in the upper part of the low score range. Bottom half of the *low* score range featured a discernible number of outliers. On the other hand, the *high* score range had a visible outlier concentration at the top of its value range. These observations make sense if considered in the context of the game's difficulty curve. The extended range of *low* scores could be associated with new players learning the rules of gameplay. High concentration of data points in the central range of the dataset was representative of an average gameplay skill, which most players seem to achieve. Finally, *high* score outlier trail could have been attributed to a few expert players, who pushed the boundaries of collection gameplay. The median of the dataset BAS collection score was found to be $M_{col} = 0.49$, and the maximum score achieved was established to be $s_{col} = 0.86$. As discussed earlier, the game's design makes it impossible to collect all *items* present in an environment, making it unrealistic for the collection score value of $s_{col} = 1.0$ ever to be reached. This made the maximum score recorded from the population a reference for the highest score attainable. It appeared that the game was balanced in a way to make it easy enough for most game samples to end with an approximately average score, but challenging enough to demand high gameplay skill to earn higher scores.

Unsupervised Clustering

Quartile-based, manual clustering into four score groups provided initial information about the dynamics of gameplay skill in collected samples, while also opening space for additional inquiry. More detailed clustering of the *average low* and *average high* score ranges appeared to be desirable, due to a large concentration of data points. Since clusters identified through quartile clustering appeared reasonable in the context of the game, we decided to use them to inform further clustering work. This motivated setting the minimal cluster count for unsupervised clustering to $k_{min} = 4$, on par with quartile-based clustering. At the same time, with respect to our clustering goals, we wanted to limit the maximum cluster count to a number that would be rational from a discretisation perspective. We anticipated that clustering into more groups would provide a better data fit. Still, we wanted to both avoid overfitting and stop data granularisation from reaching a point where descriptive labelling of clusters would no longer be possible. The worst-case scenario was producing so many clusters that their number would be comparable to that of discrete game score bins ($n = 312$ score bins in dataset POP). The best case was to have just enough clusters so they could be labelled in terms of gameplay skill context and easily understood and remembered by humans. Because of this we decided to tie our optimal cluster count range to "the magical number seven, plus or minus two", well established in psychology as the average information processing capacity limit for humans [159]. In accordance with this premise, the maximum cluster count was set to $k_{max} = 9$.



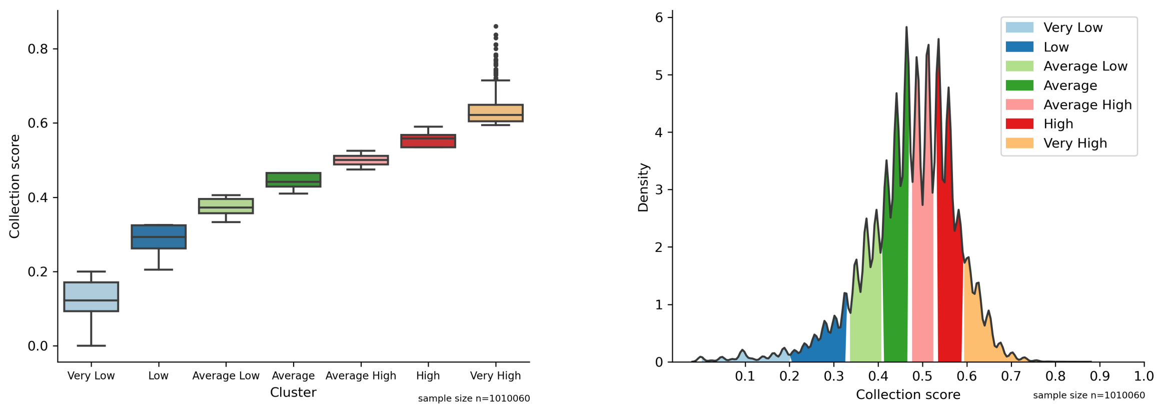
(a) k-means dataset BAS clustering elbow method determination.



(b) k-means dataset BAS cluster DBI visualisation.

Figure 5.9: Numerical evaluation of k-means generated cluster counts in dataset BAS.

We then moved to execute the unsupervised clustering process with the use of a single dimension k-means algorithm for each cluster count in the range $k = [k_{min}, k_{max}]$. The cluster count k was the only parameter to be configured for the clustering process. Each considered cluster count had a dataset BAS based model fitted via the method applied. The generated labels and centroids were stored for further analysis, along with the calculated evaluation values of the DBI, and the sum of square errors (SSE). Visual inspection of the clustering evaluation value plots (figure 5.9 on page 116) revealed that $k = 7$ presented the most optimal elbow and numerical result, indicating the best cluster separation.



(a) Clustered collection score distribution boxplot breakdown.

(b) Clustered collection score distribution according to cluster derived boundaries.

Figure 5.10: Dataset BAS collection scores clustered with one dimensional k-means algorithm using $k = 7$ clusters.

Having committed to the value of $k = 7$, we then visually plotted the segmentation of the dataset distribution into clusters identified (figure 5.10 on page 117). The clustering process produced well-distributed clusters. With minimised intra-cluster and maximised inter-cluster variances, they provided an optimal and granular representation of play skill. The middle range of the dataset BAS was partitioned into four clusters. The bottom and top collection score clusters were approximately similar to their quartile-based counterparts. To assign interpretation in the context of play skill, we organised the identified clusters into three groups associated with perceived macro gameplay skill level: low, average and high. In continuation of our approach from quartile clustering, each of the clusters produced via k-means was individually labelled to denote the micro play skill level (see table 5.21 on page 118).

5.5.4 Probability of Completion

We assumed the frequency of completion derived from the large population of dataset POP f_{com} = 0.77 could serve as a valid estimate of global completion probability *P(com)*. To compare it against the completion performance of players belonging to specific play skill clusters we calculated the probability of achieving completion for specific play skill clusters. The rising value of *P(com)* in each consecutive skill cluster suggested a trend of the probability of completion increasing with play skill improvement (see table 5.21 on page 118). Due to the non-Gaussian properties of the score datasets, we used the non-parametric Spearman's correlation coefficient to investigate a potential correlation between the two. The calculated value ρ = 0.96 indicated a strong, positive, monotonic correlation between completion probability and mean collection score of specific clusters in dataset BAS. This supported an intuitive notion that lower-scoring players were less likely to complete a game, in comparison to more skilled players.

Collection skill level	**Cluster index *k***	**Description**	***P(com)***
Low	1	*Very low*	0.54
	2	*Low*	0.69
Average	3	*Average low*	0.75
	4	*Average*	0.78
	5	*Average high*	0.79
High	6	*High*	0.81
	7	*Very high*	0.8

Table 5.21: Annotations for clusters identified in dataset BAS.

5.5.5 Play Skill Classification

Based on our clustering work with the k-means algorithm, we extrapolated centroid and labelling data for identified clusters. We were able to use them to develop a k-nearest neighbours collection play skill classifier in Python, to be applied to other collection game score datasets, originating from the game *60 Seconds!*. The collection play skill classifier, included in the digital supplement to the dissertation, constitutes research output **O4**. Play skill classes, derived from the previously established play skill clusters, were used in our further work on establishing a top-skill persona model.

5.5.6 Summary

By applying the selected sampling criteria for producing dataset BAS, outlined in section Sampling, we committed to the default collection total weight available $T_d = 43$. This was the result of normalising the collection score s_{col} calculation with respect to the chosen values of the character selection, and extended *item* set game setup parameters. Using such a normalised dataset, we were able to cluster s_{col} collection scores and identify $k = 7$ clusters, or levels, of collection play skill exhibited in gameplay trajectories. This enabled us to develop a collection play skill classifier, required for our further work. Our investigation into play skill identification has addressed the local research question **RQG**. Differences between the play skill exhibited in different gameplay trajectory samples from the game *60 Seconds!* were found to be observable and quantifiable, and it was possible to classify them with respect to the collection score clusters. Thus, we were able to classify collection gameplay performance with respect to varied collection scores observed in the data samples of the normalised, baseline dataset BAS. The delivery of a working classifier constitutes research output **O4**. However, as mentioned in the Clustering Approach section, our clustering approach was economical, and only used single-dimensional collection score data. While we consider it valid and sufficient for the purposes of our research, applying a more complex clustering approach to multidimensional data would have likely provided us with additional, interesting information about the ways users play the game.

5.6 Top-Skill Persona

5.6.1 Assumptions

To maximise the performance potential of our trained agent models we considered extracting high play skill gameplay trajectories that could be used in the training procedure a priority. We assumed the following was of importance to produce a performing agent model:

- High collection play skill exhibited in gameplay trajectories.
- High rate of game completion, exhibited in gameplay trajectories.
- A selection of gameplay trajectories that would benefit an IL based training procedure.

5.6.2 Sampling

Based on our assumptions, we decided to extract a dataset with gameplay trajectories classified as samples of the *very high* play skill, which were also successfully completed. In some cases, this could have introduced an unwanted bias to the data. However, we were operating under the assumption that our decentralised, segmented learning model design and its deployment conditions would have prevented such an issue. The chosen sampling approach also increased the likelihood that gameplay trajectories extracted would exhibit constructive gameplay behaviours that could contribute to the emergence of skilful gameplay of trained agents. To sample the top-skill persona dataset TOP we applied our regular sampling approach to dataset BAS, combined with the use of inclusion criteria based on classification, utilising the classifier developed in our prior work (see table 5.21 on page 118). The median of s_{col} values in dataset TOP $M_{TOP} = 0.62$ was noticeably higher than the corresponding value of baseline dataset BAS s_{col} $M_{BAS} = 0.49$, which was expected of gameplay trajectories with the highest collection scores recorded.

Dataset created	TOP
Sampled dataset	BAS
Sample size	76,406
Values	s_{col}
Sampling parameter	**Value**
Play skill	*Very high*
s_{com}	1.0
Measurement	**Value**
Min	0.59
Max	0.86
Mean	0.63
Median	0.62
Mode	[0.60465116]
Standard deviation	0.03

Table 5.22: Sampling parameters and description of the collection score data from dataset TOP.

5.6.3 Summary

By selectively sampling dataset TOP from the baseline dataset BAS we were able to extract a collection of gameplay trajectories, which were representative of the top-skill persona, in the game environment conditions investigated. Sampling incorporated additional data produced by the play skill classifier, which identified samples exhibiting high play skill. These highest scoring game samples of the dataset BAS constituted 7% of the entire dataset. They were deemed suitable to be used for training context-guided agent models in our further work.

5.7 Conclusions

In this chapter we have conducted a quantitative analysis of the gameplay telemetry dataset with respect to the proposed game score metric. Content documented in this chapter included:

- Game score metric proposal for quantitatively measuring gameplay performance in the *scavenge* segment of the game *60 Seconds!*.
- Investigation of game score distributions and exploring their characteristics.
- Normalised baseline dataset sampling and analysis.
- Investigation of the concept of play skill and developing a method of play skill classification.
- Establishing a top-skill persona on the basis of selective sampling of gameplay trajectories with classification, to be used in agent model training in Chapter 6: Agent Study.

This chapter has addressed the local research question **RQG** through the analysis of the gameplay trajectory sample population, identifying observable differences in game score variations between these samples, and eventually proposing a solution to the classification of different levels of play skill, exhibited in the recorded trajectories.

In the course of this chapter, we have documented the following research contributions and outputs:

- **C4**: game score metric for quantitatively measuring play skill in terms of gameplay performance in the *scavenge* segment of the game *60 Seconds!*; presented in the Game Score section.
- **C5**: analysis of game scores measured for the gameplay telemetry dataset from the *scavenge* segment of the game *60 Seconds!*; presented in Chapter 5: Game Score Study.
- **O4**: k-nearest neighbours classifier of play skill in the *scavenge* segment of the game *60 Seconds!*; presented in the Play Skill Classification section.

Chapter 6

Agent Study

Summary. This chapter documents the process of designing, training, and evaluating the context-guided agent model. We first outline the plan for conducting the study and then review the process of designing, training, and benchmarking the context-guided agent model. Experimental evaluation of the gameplay performance of context-guided agents and human players follows. In the course of the chapter, we analyse and discuss relevant results.

6.1 Overview

6.1.1 Goals

The objective of this chapter is to document and discuss the deployment and evaluation of a context-guided agent model in a real game environment. The development of the agent model was based on the design proposed and modelled after the top-skill persona dataset, extracted in Chapter 5: Game Score Study. The chapter will address the following research questions:

- **RQ1**: can models with execution and performance guarantees of learning logic be integrated into the game industry, state of the practice, ad hoc behaviour AI architecture for applied use in video game playing AI?
- **RQ2**: how well can a trained context-guided learning agent perform in unseen game environment scenarios, in comparison to human players, in approximately similar gameplay conditions?

This chapter also documents the following research contributions and outputs:

- **C6**: analysis of the training process of a context-guided learning agent, capable of playing the *scavenge* segment of the game *60 Seconds!*, developed on the basis of the context-guided agent design.

- **O5**: a trained context-guided learning agent, capable of playing the *scavenge* segment of the game *60 Seconds!*, developed on the basis of the context-guided agent design, using game industry off-the-shelf solutions.

6.1.2 Structure

The Agent Study chapter is divided into the following sections, detailing the agent development, evaluation and analysis work:

- **Study Plan**: outlines the plan for conducting the Agent Study.
- **Agent Training**: documents the design, deployment and benchmarking of the context-guided learning agent and reference agent models.
- **Agent Evaluation**: presents the experimental evaluation conducted online with human players, and offline with the context-guided agent.
- **Agent Review**: investigates the results of agent training and evaluation to examine the performance of the deployed context-guided agent model, and the relevance of the proposed context-guided learning agent design.
- **Conclusions**: summarises the training, evaluation and analysis work conducted in this chapter, and then highlights the key takeaways and contributions delivered.

6.2 Study Plan

6.2.1 Overview

In the course of the Agent Study we applied the proposed context-guided agent design in practice and conducted a formal investigation of its viability through experimental evaluation and analysis of the results produced. This approach segmented the study into a three-stage structure, supporting tangible, practical objectives of producing a working agent model and analysing its output: Agent Training, Agent Evaluation, and Agent Review.

6.2.2 Agent Training

The Agent Training stage of the Agent Study details the deployment and benchmarking of the context-guided agent model. The structure of the deployment was directly derived from the deployment workflow proposed for the context-guided agent design in the Deployment Workflow section of Chapter 4: Context-Guided Agents and covers the following points:

- **Step 1**: Design behaviour context.

- **Step 2**: Establish learning subtrees.
- **Step 3**: Implement context architecture.
- **Step 4**: Implement learning logic.
- **Step 5**: Acquire data.
- **Step 6**: Train learning models.

Benchmarking of the context-guided agent model was to be done after its deployment. In addition to benchmarking context-guided agents, we required a reference agent to compare them against. The selection of the reference model and its individual benchmarking was also a part of the work done in the Agent Training stage. Benchmarking is discussed in the following sections:

- **Benchmarking Procedure**: outlines the benchmarking procedure used for the reference and context-guided agents.
- **Reference Model Selection**: discusses the selection of the reference agent, to be compared with the context-guided agent.
- **Reference Agent Benchmarking**: documents the execution and results of the reference agent benchmarking.
- **Context-Guided Agent Benchmarking**: discusses the selection of model revisions for use in the context-guided agent and compares the results of the context-guided agent's benchmark against those of the reference agent.
- **Agent Versus Agent**: discusses the benchmarking procedures of the context-guided agent and its output gameplay performance, with respect to the reference agent.

6.2.3 Agent Evaluation

The Agent Evaluation stage of the study covers the experimental evaluation of the context-guided agent model trained and deployed during the Agent Training stage. The evaluation of the trained context-agent model was intended to take advantage of the extensive instrumentation of the game environment, discussed in the Experimental Evaluations section in Chapter 4: Context-Guided Agents, and be conducted both offline and online. The offline part of the experiment was targeted at measuring the play skill exhibited by simulated agent models in the agent simulator environment. In contrast, the online part of the experiment was to crowdsource play skill measurements from the large audience of the game's human players.

The Agent Evaluation stage was structured to follow formal experiment reporting, and features the following sections:

- **Goals**: defines the objectives of the evaluation.
- **Setup**: discusses the setup of both the online and offline parts of the experiment and highlights their differences.
- **Procedure**: provides a walkthrough of the procedure for both the online and offline parts of the experiment.
- **Results**: reports how the evaluation was conducted, and presents the results obtained in the course of the offline and online parts of the experiment.
- **Agent Versus Humans**: investigates the gameplay performance of human players in the crowdsourced, online evaluation, and how it compares against that of the context-guided agent.

6.2.4 Agent Review

The Agent Review brings together relevant data collected and developments made in the course of the Agent Training and Agent Evaluation. It reviews the deployed context-guided agent, discusses conditions and challenges encountered, decisions made, and how they shaped the output of the agent training process.

6.3 Agent Training

6.3.1 Assumptions

To deploy a context-guided agent, we applied the set of design requirements specified in the context-guided agent design Requirements section of Chapter 4: Context-Guided Agents. The trained, output agent models were expected to be functional and valid in the context of the game environment of *60 Seconds!*.

Hardware

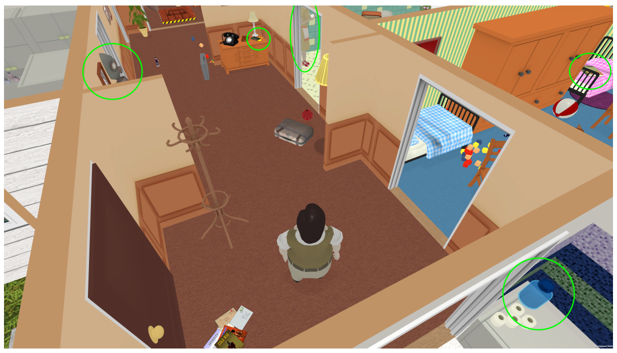
We intended to conduct all our training, learning and inference procedures on a single, off-the-shelf machine that could realistically be used in a contemporary game development studio. The hardware chosen for all offline computations was a single Microsoft Windows 11 Pro workstation, equipped with an AMD Ryzen 9 3900x 12-core central processing unit (CPU) clocked at 3792 MHz, 64 gigabytes of Random Access Memory (RAM), and NVIDIA GeForce RTX 3080 Ti graphics processing unit (GPU).

Approximations in Agent Training

As discussed in the Approximated Models section of Chapter 4: Context-Guided Agents we assumed we would be working with approximated model representations, resulting from both the characteristics and dynamics of game environments, as well as the nature of our research. In that regard, additional considerations were taken into account for observability of the game environment, agent navigation abstraction, and the simulator environment limitations, during agent training.

Observability of the Game Environment

In the Agent Perception section in Chapter 4, we stated that agents were not supposed to be omniscient and were only provided with partial observability of the environment, to simulate human-like sensory perception. The observability capacity of each of the agent models was modelled after the environment visibility afforded by the player-controlled avatar's camera viewport. This enabled agents to detect points of interest, which had been registered within the camera's viewport, as the agent navigates the game level (see figure 6.1 on page 125). At the start of each gameplay session, the agent was also made aware of the location of the *exit*. To make long-term use of registered points of interest and other game session data in behaviour logic, we equipped agents with a memory representation. It was implemented to extend the agent's model representation in the game environment. Points of interest observable by agents included: collectable *items*, the level *exit*, and room *areas*.


(a) Agent game camera view.

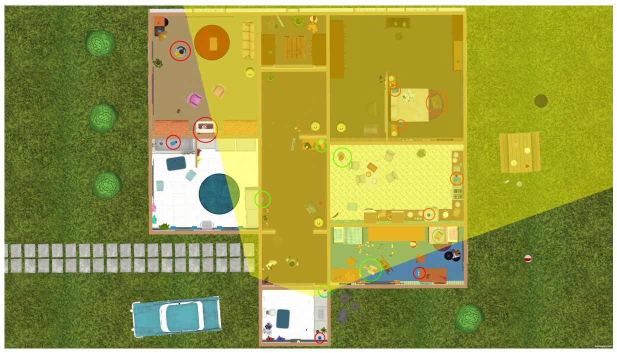

(b) Agent top-down camera view with the approximate field of view in the level of the agent's game camera.

Figure 6.1: Agent's perception of *items* in the environment. *Items* visible within the agent's camera's viewport are considered spotted (marked green). Those outside the viewport are not (marked red) until the agent faces them and they enter the camera's viewport.

To further support the agent's perception of the environment in terms of time, we introduced the concept of game flow stages, tracked with respect to the time elapsed in a game session. The developers had already used the game's UI and audio to signal the player about the passage of time using thresholds defined for different stages of the game sessions. We mapped their time ranges to proposed flow stages: *exploration*, *early game*, *mid game*, *late game*, and *very late game*. This discretisation of the game time would later be used in the logic of BT nodes and learning models.

Agent Model

As discussed in the Agent Perception section in Chapter 4, the game environment is modelled as an MDP, while the behaviour flow of agents operating in that environment is modelled as a POMDP. While technically we could afford full access to the underlying game environment's state information, agents are only allowed to acquire data that can be perceived by them in the context of the environment, thereby simulating an approximated human-like perception. The extent of their observability of the game environment was discussed in the Observability of the Game Environment section.

Agent Q executes its interaction policy π_a in a game environment e, characterised by the environment's current state s_e, which belongs to the environment's finite state space S_e. The state of the environment contains information about game state, game time, physical space, physical objects simulated in the environment, *item* placement, *exit* placement, and the agent's full, real state s_Q. The environment's state can be altered by a state transition function $P_e(s_e, s'_e, a_e)$, triggered by invoking an action a_e from its permitted action space A_e. Action space A_e contains two subsets of actions: those that can be invoked by an agent A_{ea}, and those which are only triggered by the environment control logic A_{ex}. The control logic of *60 Seconds!* invokes actions from A_{ex} independently of the agent's operations, by means of a sequenced game loop policy π_e. They include placing collectable *items* in the environment, starting the game, unlocking the ability to collect items (after the exploration period), ending the game, controlling time progression, and simulating the physical state of objects in the environment. Agents cannot trigger any actions from A_{ex}, but can invoke all the actions from the action subset A_{ea}: repositioning the avatar, collecting an item, depositing items, colliding with an object in the environment.

Agent Q operates with respect to two states: its real, complete state s_Q, which is part of the state space S_Q, and its belief state about the environment b_Q. The former features complete information about the agent, and is technically part of the environment state s_e. It is fully accessible to the environment for the purposes of servicing the gameplay of the game. Meanwhile, the agent relies on its formulated belief state b_Q, which is an agent's prediction of s_e. It is based on partial information collected by the agent from observations Ω_Q of the environment e, received as a result of a state transition, triggered by an action a_Q from the agent's individual action space A_Q. Finite set of observations Ω_Q simulates what a human player would be able to register by interfacing with the game visually and gameplay-wise: data about *items* spotted in the environment, *exit* information, time left in the game, *inventory* state, information about previous *deposits*, and current behaviour context state. A_Q is defined with respect to the agent-specific implementation of features and behaviour logic. Minimal action space of an agent facilitates calling into the environment action subset A_{ea}, and passively observing the environment's space, based on the agent's position and orientation in the environment.

Incorporating additional behaviour logic, including data-driven models, into the agent's model augments the outlined model by expanding the agent's observation space Ω_Q, action space A_Q, and potentially its state space S_Q. This is not mutually exclusive with modelling learning logic as a POMDP individually, which might make it easier to express the local properties of a data-driven model.

Navigation Abstraction

For our agent models, we decided to embrace an abstracted modelling of the agent's navigation in the form of A* pathfinding on a navigation mesh to approximate human player navigation, as outlined in the Autonomous Agent Support section of Chapter 4: Context-Guided Agents. As a game industry standard solution, it was applicable in the context of our work and solved the issue of moving the agent around the game environment. Modelling human-like navigation through learning is a challenging research direction, which has been pursued by other teams working on learning in games [56, 158, 283]. One of the reasons for this is that humans are often able to easily distinguish bots playing a game, rather than humans, solely based on unnatural navigation patterns exhibited by agents. However, given a limited navigation space, an expert trajectory dataset for navigation styling, and a big enough compute budget, this is a solvable problem when combining RL and IL [243]. To avoid expending our resources on perfecting a learning-based navigation model, we chose to focus our learning efforts on the agent's decision-making. Agents were only allowed to attempt navigation to previously discovered points of interest. This included room areas, which were introduced as representations of individual room spaces within the game's environment. Targeting and visiting areas supports the agent's exploration process, which increases the likelihood of spotting additional *items* to be targeted for collection.

Semi-Automated Training Pipeline

We considered it beyond the scope of this thesis to develop a training pipeline that would support full automation of the learning-evaluation-revision process. While such an approach, based on sequenced logic, genetic algorithms, or other methods, would have benefitted learning model training, its engineering cost was deemed too high. We expected the development of a fault and interruption-proof solution for an end-to-end learning model deployment to involve too many factors outside of our control. Especially, the pipeline we were working towards involved different, interconnected off-the-shelf solutions and libraries, whose flow had yet to be tested. Thus, we assumed we would only produce a limited number of functional learning models during agent training, using a semi-automated pipeline and a conservative compute budget. This decision was also motivated by the assumptions of the context-guided agent design and industrial requirements. Training of small-scale models for context-guided agents was expected to generate a lower training cost than training monolithic, holistic learning models. Shorter training times involve smaller compute budgets and encourage rapid iteration in the form of model revisioning and retraining. Such an approach would be empowering for teams, where individuals or groups could focus their work on selected segments of the learning logic within an agent's behaviour architecture.

6.3.2 Agent Deployment

Overview

We followed the deployment plan of the context-guided agent design proposal Deployment Workflow in Chapter 4: Context-Guided Agents to produce a working context-guided agent model. The following sections detail the work process and deliverables of each individual work step executed.

Design Behaviour Context

On the basis of the game environment and gameplay specifications, presented in the Game Environment chapter, we outlined objective-based behaviours that a player, and by extension an AI agent, playing in the *scavenge* game environment of *60 Seconds!* would be expected to achieve:

- **Navigate**: relocate to points of interest in the navigable game level space, with as little interruption of the movement as possible.
- **Explore**: discover points of interest, such as collectable *item*s, in the game environment, to be targeted later.
- **Collect**: target and collect relevant *items* in the environment, with respect to the weight of the avatar's inventory.
- **Deposit**: deposit collected *items* at the *exit* location, but only one or more *items* have been collected.
- **Evacuate**: complete the game successfully by navigating to the *exit* location, before the time runs out.

By implementing the navigation mesh-based avatar movement as part of the game environment instrumentation, we ensured the agent's capacity to handle the navigation aspect of playing the game and achieving other gameplay objectives required to pursue targeted context architecture and learning implementations, according to the context-guided agent design. We assumed that an agent should always operate in a single behaviour context, matching one of the proposed objective-based behaviours. This introduced some constraints, as not all behaviours were appropriate for the avatar to execute at any moment in the game:

- **Explore**: only valid during the *exploration* stage of a *scavenge* game, before *item* collection becomes possible, or if the agent has no potential target *items* spotted.
- **Collect**: only valid after the *exploration* stage, when *item* collection is possible, and when the avatar's *inventory* is not full.
- **Deposit** only valid after the *exploration* stage, when the avatar's *inventory* contains at least one *item*.
- **Evacuate**: valid at all times, but only reasonable in the later stages of the game, to leave enough time for *item* collection.

We also assumed a fallback *idle* behaviour should be available to enable the agent to await the end of the game, if they had evacuated with time to spare, and were forced to remain stationary at the *exit* location for the remainder of the game session. It was designated as the default, worst-case behaviour choice, since the player should strive to remain in constant motion, to make the most of the gameplay time available.

The adopted behaviour context design implied the following gameplay logic had to be implemented for an AI agent to operate autonomously in the game's environment:

- Selecting the next *scavenge* behaviour (SNSB) to pursue.
- Selecting the next target *item* (SNTI) to be collected.
- Selecting the next room *area* (SNTA) to be explored.
- Sequence of actions taken to explore the environment.
- Sequence of actions taken to collect an *item* in the environment.
- Sequence of actions taken to deposit *items* from the avatar's *inventory*.
- Sequence of actions taken to evacuate to the *exit* to complete a game session.

This separation of logic informed an intuitive segmentation of the architecture into *macro*, *proxy*, and *micro* layers. Strategic, *macro* decision-making was embodied by the behaviour selection logic, while the lower-level action sequences for each of the behaviours occupied the *micro* layer. Connections between the two, facilitated by additional ad hoc scaffolding or conditions, emerged as the *proxy* layer.

Establish Learning Subtrees

The premise of the context-guided agent design encourages positioning learning logic where it can deliver more value than ad hoc solutions. To facilitate that, we investigated the gameplay logic required for our AI agent using the previously established game behaviour context. In the process, we identified several action sequences for each of the gameplay behaviours defined. The presence of sequential execution indicated that causality and ordering of individual tasks was relevant. This presented a potential learning challenge, especially in a more complex environment with sparse rewards. Thus, we considered all action sequences to be candidates for ad hoc authoring; however, in the cases of *scavenge* behaviour and next target *item* selection logic, their ad hoc implementations would have likely warranted an intuition-driven, heuristics-based approach. A data-driven logic model was expected to be a preferable alternative, potentially offering a more optimal solution to these problems. And so, we decided to implement the SNSB and SNTI logic as learning subtrees in our context-guided agent architecture. SNTA was also considered, but due to its limited impact on the completion and collection objectives, we decided against implementing it as the third learning subtree. The behaviour of the context-agent model was planned to be styled using data from the top-skill persona gameplay trajectories. This suggested IL as a policy learning technique to be used for both SNSB and SNTI. With the amount of expert data we had at our disposal, we assumed that, in combination with RL, we could effectively bootstrap learning, providing both behaviour styling and leaving room for more open-ended training. Commonly used and tested learning algorithms supported by *ML-Agents* considered for the task included PPO for RL, and either BC or GAIL, or the combination of the two, for IL.

Implement Context Architecture

To facilitate the implementation of the outlined context architecture, we created a BT asset using the *Behavior Bricks* BT library, discussed in the BT Authoring section of Chapter 4: Context-Guided Agents, which mapped the identified gameplay behaviours and their associated gameplay logic onto a BT structure. We implemented additional BT nodes and conditions in the game's code to be able to express that logic fully in terms of a BT. This included action nodes that would later be used as an ad hoc fallback for the learning logic, embedded in the context-guided agent's BT:

- **SNSB**: next behaviour selection logic was implemented using a heuristic-based approach, derived from the constraints identified in the Design Behaviour Context section.
- **SNTI**: next target *item* selection was implemented by randomly selecting one of the *items* that the agent spotted in the environment.

The context architecture was developed to be relatively lightweight, simplify testing and debugging. It relied on the behaviour selection logic deciding which behaviour should be executed next, and then entering the tree branch that represented an action sequence associated with the selected behaviour. Until that behaviour was concluded, processing was contained to the behaviour's branch. The implementation process of the context architecture using ad hoc authoring was both an engineering and design task, comparable to typical implementation procedures in commercial game production.

Ad Hoc Behaviour Model

The implementation of the context architecture, presented in the Design Behaviour Context and Implement Context Architecture sections, was concluded with a fully functional, ad hoc authored agent behaviour model, based on a BT. To investigate the validity and competence of the ad hoc behaviour model, we conducted a series of test sessions in the environment of the Unity editor. In the process, we identified a series of issues with the underlying implementation, most of which originated in the setup of Unity's navigation system. All problems detected were resolved, enabling the ad hoc agent to play the game. The BT we used for the ad hoc behaviour model is presented in figure 6.2 on page 131.

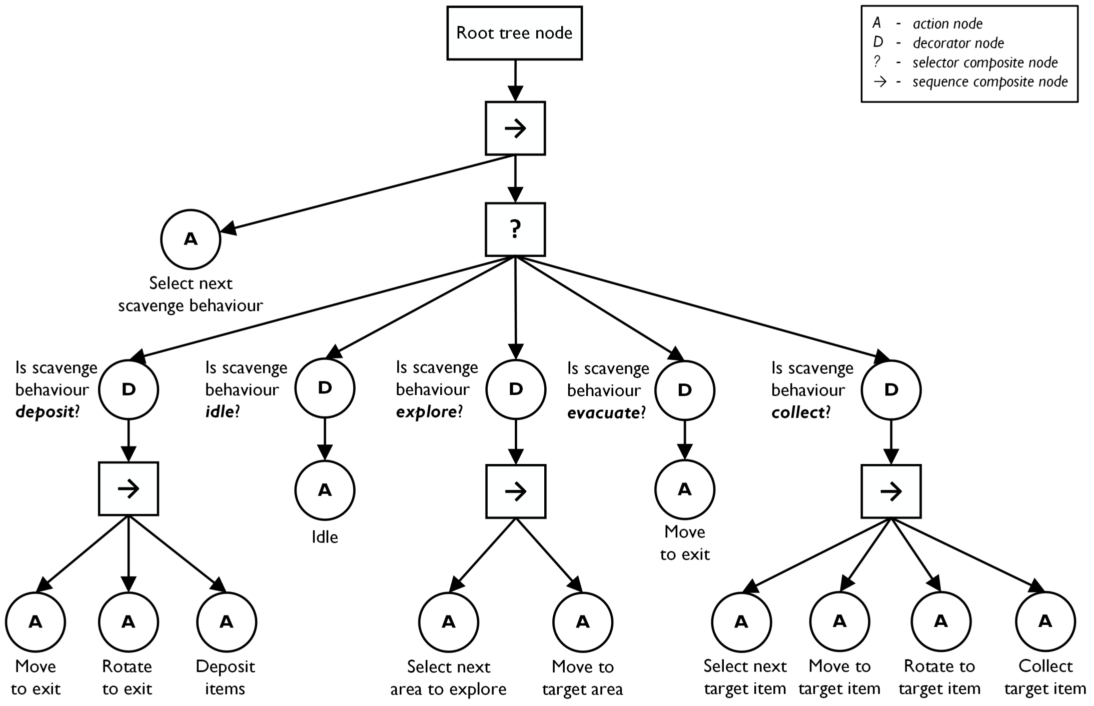

Root tree node
A - action node
D - decorator node
? - selector composite node
→ - sequence composite node
Select next scavenge behaviour
Is scavenge behaviour deposit?
Is scavenge behaviour idle?
Is scavenge behaviour explore?
Is scavenge behaviour evacuate?
Is scavenge behaviour collect?
Idle
Move to exit
Move to exit
Rotate to exit
Deposit items
Select next area to explore
Move to target area
Select next target item
Move to target item
Rotate to target item
Collect target item

Figure 6.2: Ad hoc behaviour agent model expressed as a BT, featuring the implement behaviour context.

Implement Learning: Overview

After validating the ad hoc behaviour model, we were able to use it as a basis for the context-guided agent model, which would incorporate learning logic. We first took the ad hoc model BT and replaced the ad hoc logic for SNSB and SNTI with learning nodes. Each of the replaced ad hoc logic action nodes was placed inside the relevant learning subtree to provide fallback functionality in the event of learning failure. The learning process was conducted when a learning node in the BT was ticked, according to the flow of the tree logic. Upon tick, a learning node update requested a decision to be made by the associated learning model. This triggered the model's observation collection, followed by action generation, which was then processed by the gameplay code. The architecture and the general flow of the agent's logic remained intact, as pictured in figure 6.3 on page 132. To be able to facilitate the learning process using *ML-Agents*, we had to expand our original instrumentation of the game environment to accommodate cases such as ours, where two learning models could be active in inference mode at the same time. We were unable to achieve the same in training mode due to the lack of Unity's support for training more than one model at a time. This additional effort had an advantageous side-effect, as it made the code-side implementation of reward signals and observation acquisition more centralised. We also made the values of reward signals configurable outside of code, in the agent settings data file, to enable a more flexible manipulation of their values during training. Learning episodes for both the SNSB and SNTI models were set to match the length of a single *scavenge* gameplay session. This resulted in both models having to handle sparse, long-term rewards. Sufficient short-term rewards were also necessary to encourage the model's learning progress.

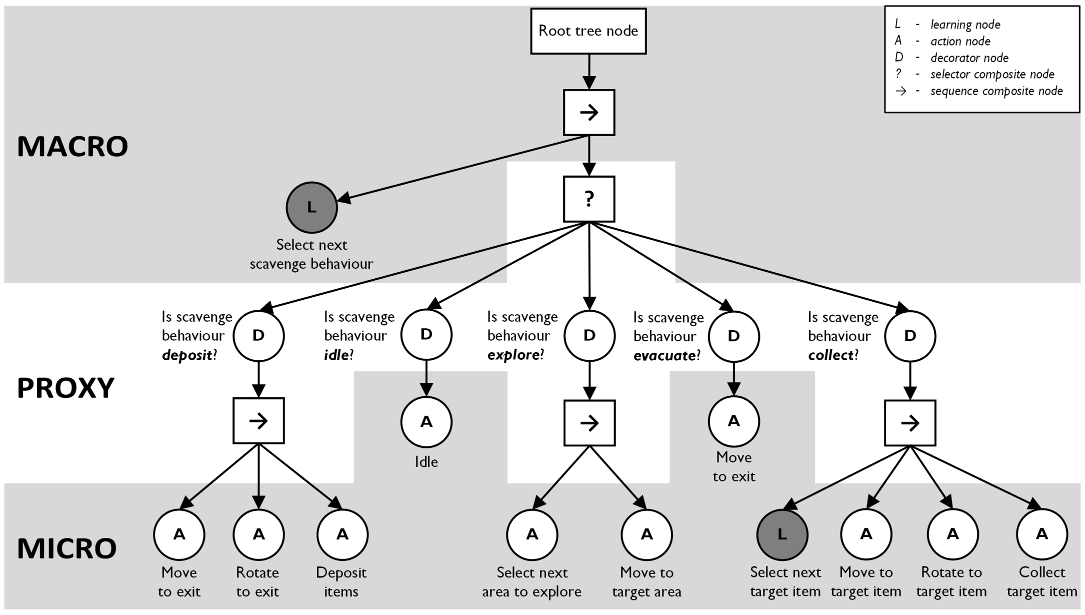

Root tree node
L - learning node
A - action node
D - decorator node
? - selector composite node
→ - sequence composite node
MACRO
PROXY
MICRO
Select next scavenge behaviour
Is scavenge behaviour deposit?
Is scavenge behaviour idle?
Is scavenge behaviour explore?
Is scavenge behaviour evacuate?
Is scavenge behaviour collect?
Idle
Move to exit
Move to exit
Rotate to exit
Deposit items
Select next area to explore
Move to target area
Select next target item
Move to target item
Rotate to target item
Collect target item

Figure 6.3: Context-guided behaviour agent model expressed as a BT incorporating learning subtrees. The presented ad hoc BT is representative of the context-guided architecture designed for the behaviour context of the game. SNSB and SNTI learning nodes have replaced ad hoc action nodes and facilitate learning. An example of layering architecture with *macro*, *micro*, and *proxy* layers is provided.

Implement Learning: SNSB

The SNSB learning model was designed with an action space A_{SNSB} featuring a single discrete action branch, with the set of $n = 5$ supported actions representing selections of different *scavenge* behaviours that the context-guided agent model was expected to service: *idle*, *explore*, *collect*, *deposit*, and *evacuate*. Since these actions were mutually exclusive, it was possible to express them within a single discrete action branch, with each of the actions mapped to a specific integer value. The integer values used corresponded to the behaviour type enumerator value, defined for the agent in code. Agent's action selection triggered the next *scavenge* behaviour to be executed by the agent.

SNSB was a decision-making model, guiding the agent's operations from the *macro* level. This informed the selection of observation values that were expected to generate a holistic perspective of the agent's situation in the context of the gameplay with respect to the state of the agent s_Q, and its belief state about the environment b_Q. SNSB's observation space Ω_{SNSB} featured $n = 12$ vectors:

- Current *scavenge* behaviour (integer, derived from the behaviour type enumerator value)
- Previous *scavenge* behaviour (integer, derived from the behaviour type enumerator value)
- Target *item* type (integer, derived from the game's *item* type enumerator value)
- Target *item* distance (float)
- Target *item* weight (float)
- Agent's current *inventory* weight (float)
- Agent's *inventory* maximum weight (float)
- Time left in the game (integer)
- Current flow stage of the game (integer, derived from the flow stage enumerator value)
- Agent's avatar distance to *exit* (float)
- Number of *areas* left to visit (integer)
- Spotted *item* count (integer)

SNSB's reward function R_{SNSB} combined local objective, global objective, and behaviour validity reward signals. The majority of SNSB reward signals were mapped to local objectives of specific behaviours, which emitted reward signals in the course of a learning episode, following an action-observation mapping. Global objectives (collection and completion) were rewarded accordingly at an end of a learning episode, based on the criteria defined within R_{SNSB}. Additionally, since the SNSB model had the capacity to select an invalid behaviour, rewards promoting valid and discouraging invalid choices of behaviours in the context of the environment state s_e were also issued. Reward values were configured in the model's input configuration file, which is presented in table D.1 on page 212. Pseudocode for the SNSB's reward function R_{SNSB} can be reviewed in table D.4 on page 218.

Implement Learning: SNTI

The SNTI learning model was designed with an action space A_{SNTI} that featured a single discrete action branch. It supported a set of $n = 21$ actions, each corresponding to a collectable *item* type, whose integer value was derived from the game's *item* type enumerator value. The agent's action choice was translated into the type of *item* they wanted to target. Since it was assumed that the agent could only observe a single target at any given time, a discrete branch representation was valid here. The actual target *item* was then retrieved from the collection of spotted *items*, maintained in the model's gameplay logic, on the basis of prior observations o_{SNTI}.

SNTI's goal was to make a local target selection decision, which was considered part of the *micro* layer of the behaviour context. This informed the decision to focus its observation space on *item* related data, and forego more general information, which was observed for SNSB. Still, SNTIs observation space Ω_{SNTI} featured as many as $n = 89$ vectors, which included:

- Current target *item* type (integer, derived from the game's *item* type enumerator value)
- Time left in the game (integer)
- Agent's distance to *exit* (float)
- Agent's current *inventory* weight (float)
- Agent's maximum *inventory* weight (float)
- Specific *item* type observations for each of the 21 *item* types present in the game.: *item* type (integer), distance to the closest instance of the particular *item* type (float), *item's* weight (float), and the number of previously collected instances of the *item* (integer).

Primary reward signals of the SNTI's reward function R_{SNTI} were concerned with effective target selection. Similarly to SNSB, SNTI was able to select an invalid target *item*. Thus, successful target selection for an *item* that would fit in the agent's *inventory* was rewarded, whereas failed target selection was penalised. Additional rewards were provided for the agent performing a successful *item* collection, as well as collecting the first instance of a specific *item* type. The collection global objective was also promoted by a reward signal at the end of each session, with value tied to the actual collection performance recorded. Reward values were configured in the model's input configuration file, which is presented in table D.1 on page 212. Pseudocode for R_{SNTI} can be reviewed in table D.5 on page 218.

Acquire Data

Although we had already aggregated the gameplay trajectory dataset to be used for IL, it was still necessary to generate a Unity-compatible demonstration file for each of the trajectories, to be used as the input data for the IL process via *ML-Agents*. Demonstration files had to be generated individually for SNSB and SNTI, as these two models had a distinct set of observations, rewards, and actions. The process was conducted using the gameplay trajectory replay feature of the agent simulator environment, configured to output demonstration files. Initially, we attempted to encode all our trajectories into a single file: one for SNSB, and one for SNTI. In the process of training our learning models, we found out that trajectories encoded into Unity's proprietary demonstration format were unusable. Through trial and error, we were able to deduce that the root cause of the problem was the number of trajectories encoded into a single demonstration file. The issue manifested sooner with SNTI training, as the demonstration files for SNTI were larger than those generated for SNSB. It was likely a result of SNTI's larger observation space. We established that we could encode no more than 100 trajectories into a single demonstration file before the software stability of the learning library became compromised. This resulted in repeating our data acquisition step and generating 765 demonstration files for each of the learning models to be trained, each with only 100 episodes encoded. However, as we would find out during subsequent model training, we were also forced to limit the number of demonstration files used in training our models, due to the software stability problems with *ML-Agents*.

Train Learning Models

Due to the constraints of *ML-Agents*, which required individual training of each of the learning models involved, the training of SNSB and SNTI was planned accordingly. Each of the models was trained separately, with its context-guided agent BT learning node active and set to training mode, participating in the learning process. The other, inactive model still had its learning node tick in the BT, but due to it being offline, it would automatically fail, and use its ad hoc, fallback logic instead. This way, training was normalised with respect to what we assumed would be the worst-case scenario provided by the ad hoc, fallback logic. To further normalise the training conditions of both models, we configured both training scenarios with game setup parameters equivalent to those used to sample the baseline dataset BAS in the Baseline section of Chapter 5: Game Score Study:

- **Game type**: *scavenge* and *full* game type ruleset.
- **Difficulty**: *normal*.

- **Extended *item* set**: true, per default setup of levels used.
- **Character selection**: *Ted*
- **Level**: full range $L = [1, 20]$ of *scavenge* levels available.

We then configured rewards for each of the models. We attempted to encourage learning through consistent and positive, but granular, feedback. Penalties were only applied when invalid actions or actions that did not make sense in the context of the game environment were taken. The end of each episode rewarded achievement of completion and collection of objectives. Extreme, negative states, where not a single *item* was collected, or completion was not achieved, were penalised. Reward configuration for SNSB and SNTI can be reviewed in table D.1 on page 212. The hyperparameter configuration for SNSB and SNTI covered the setup of PPO, BC, and GAIL. For initial hyperparameter value setup in each of the scenarios, we referred to the official Unity *ML-Agents* documentation, and applied the suggested, optimal values provided for each of the algorithms [251]. However, due to the specifics of our learning scenarios, we introduced some modifications:

- **Batch size**: since the action spaces of both SNSB and SNTI were discrete, we decreased the batch size to be smaller, from 1024 to 128.
- **Maximum steps**: the number of learning steps was doubled to 1,000,000, following initial RL only tests, which suggested a larger number of steps might be required to generate better models.
- **BC and GAIL strength**: since we intended the applied IL algorithms to have a significant influence on the learning process, we increased the strength of BC and GAIL to 0.1, from the default value of 0.01.

The first model version, and the hyperparameter configuration associated with it, was labelled **A000** (see hyperparameter values D.2 on page 215). In the course of our training work, we produced two more revisions of both the SNSB and SNTI models, labelled **A100** (see hyperparameter values D.3 on page 217) and **A200** (see hyperparameter values D.6 on page 221). The agent simulator environment was configured to conduct semi-automated training for the individual scenarios of SNSB and SNTI using 10 CPU-bound simulator environments running in parallel, jointly contributing to the learning process.

The first attempt at training A000 failed on startup, due to errors generated by *ML-Agents*, pertaining to the BC configuration. We were unable to track the exact reason for this, but we were able to deduce that the problem originated in the encoding of our demonstration files. Neither Unity's official nor unofficial community channels provided an explanation. Unity's official channels refrained from assisting users who had applied *ML-Agents* in custom environments [253]. Because of this development, we had to reluctantly remove BC as one of our IL algorithms and only apply a combination of PPO and GAIL. Additional problems related to IL appeared in our second attempt at training A000, despite removing BC from our learning configurations. No instances of the agent simulator environment were able to complete startup to begin training. All of them eventually crashed, but not before spending several hours on what appeared to be loading input demonstration files. No errors were produced, but by trial and error, we were able to conclude that the issue originated with the size and number of demonstration files. We repeated the data acquisition step and, as discussed in the Acquire Data section, found out that we had to limit the number of gameplay trajectories from the top-skill persona dataset used for training. The number of demonstrations was limited to 180 for A000 SNSB, and 100 for A000 SNTI, to ensure uninterrupted training.

	A000	**A100**	**A200**
Buffer size	10,240	5120	5120
Extrinsic strength	1.0	0.1	1.0
GAIL strength	0.1	0.9	0.5
Hidden units (network)	128	64	64
Hidden units (extrinsic)	128	64	64
Hidden units (curiosity)	128	64	64
Encoding size (curiosity)	256	128	128

Table 6.1: Hyperparameter configuration changes between iterations of SNSB and SNTI models A000, A100, and A200.

The first iteration A000 produced a model that appeared to be able to play the game. Reward values achieved during training, as well as collection and completion scores, looked promising, but we assumed there was room for improvement. Especially, the computational cost incurred due to a long training time, which exceeded 36 hours, was prohibitive. This informed the hyperparameter configuration of the second iteration, A100. To decrease the complexity of the trained network and, consequently, its training time, we halved a selection of hyperparameter values, including the buffer size, hidden unit count, and curiosity encoding size. We also decided to experiment with radically increasing the GAIL signal strength, while minimising the strength of extrinsic rewards (see table 6.1 on page 137). Other hyperparameters were copied from A000. While A100 was trained in less than 7 hours, its performance was underwhelming. The A100 SNSB model became fixated on completion, ignoring collection objectives. The A100 SNTI model scored negatively in terms of the cumulative and extrinsic rewards. To cope with this drop in performance, while still attempting to keep the training time lower, we increased the extrinsic signal strength and decreased the strength of GAIL, producing configuration A200. Other hyperparameters remained unchanged from A100. The improvement was minimal, while the length of the training increased to more than 17 hours. Due to the iteration limits involved, we concluded our model training at that point.

The general results and training times of models A000, A100, and A200 are presented in table D.7 on page 221. Visual presentation of training progression in terms of key measures is shown in figure D.7 on page 222 for the SNSB models, and in figure D.8 on page 222 for the SNTI models.

Test Behaviour Context with Learning

Having trained SNSB and SNTI learning models A000, A100, and A200, we expanded the agent simulator environment to incorporate these models as embedded assets. This made it possible to generate an instance of the context-guided agent with functional learning logic of SNSB and SNTI operating in the environment's executable. We then simulated the context-guided agent's operations in inference mode in the game environment and observed its behaviour. The agent's behaviour appeared to be valid in terms of the game's context and was concerned with pursuing the game's objectives with a varying degree of success. Quality checking of the behaviour context was left to be conducted later, during agent benchmarking.

6.3.3 Agent Benchmarking

Procedure

Benchmarking was conducted as follows:

- Selecting the reference model to be used for comparison with the context-guided agent model.
- Benchmarking the reference model to determine its gameplay performance.
- Benchmarking of the trained context-guided agent models to compare their gameplay performance and select the best performing one to be used in further benchmarking and in the experimental evaluation.
- Selection of the context-guided agent model to be evaluated and comparing its gameplay performance to that of the reference agent.

Benchmarking of all agent models was conducted in normalised conditions, based on the game setup parameters used to sample baseline dataset BAS:

- **Game type**: *scavenge* and *full* game type ruleset.
- **Difficulty**: *normal*.
- **Extended *item* set**: true, per default setup of levels used.

- **Character selection**: *Ted*
- **Level**: full range $L = [1, 20]$ of *scavenge* levels available.

All agent models were simulated in the inference mode, in the agent simulator environment. They were all tasked with playing every game level from the configured level range 50 times with the execution speed of the simulator multiplied by 10. Each agent model generated 1000 gameplay trajectories, which were then processed, using our data processing pipeline, to be analysed with respect to the collection and completion scores achieved.

Reference Model Selection

While we planned for an experimental evaluation with human players, whose gameplay we were ultimately trying to model and simulate with the context-guided agent model, it was desirable to compare its performance with that of an alternative agent implementation before conducting the experiment. Since one of the key steps in developing a context-guided agent, according to the proposed design, was the deployment of an ad hoc authored behaviour model, we chose to use it as our baseline model. Intuitively, the context-guided agent model was expected to surpass its ad hoc, fallback base. If it failed, that would have implied the output of the context-guided agent's learning model was no better than that produced by a manually authored solution, at a much lower training cost. Thus, the agent model we chose to be our reference model was the ad hoc behaviour model developed in the course of the context-guided agent training, which was discussed in the Ad Hoc Behaviour Model section.

Reference Agent Benchmarking

Data processing of the gameplay trajectories produced by the reference model benchmarking resulted in the creation of dataset REF (see table 6.2 on page 139). Statistical description of dataset REF revealed its median collection score to be $M = 0.33$. The collection play skill of the model was classified as *Low*. Completion probability of REF was found to be *P(com)* = 0.57.

Dataset	REF
Sample size	1000
Values	s_{col}
Measurement	**Value**
Min	0.0
Max	0.54
Mean	0.32
Median	0.33
Mode	[0.27906977]
Standard deviation	0.06

Dataset	REF
Sample size	1000
Values	s_{com}
Measurement	**Value**
Min	0.0
Max	1.0
Mean	0.57
Median	1.0
Mode	[1.]
Standard deviation	0.5

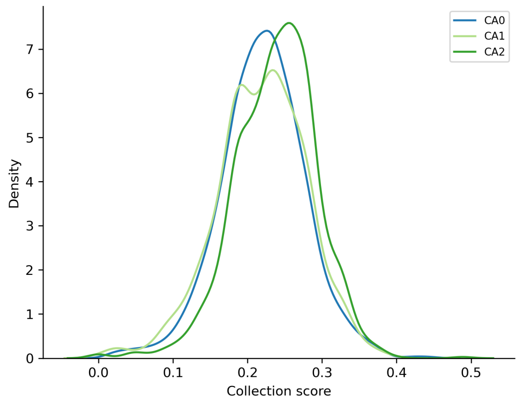
Table 6.2: Statistical description of the s_{col} and s_{com} value distributions of the reference model benchmarking results aggregated in dataset REF.

Context-Guided Agent Benchmarking

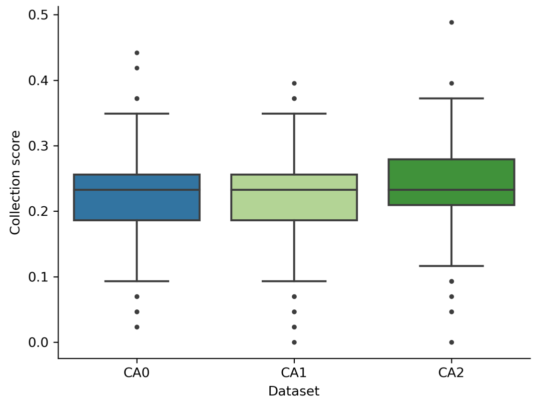
Data processing of the gameplay trajectories generated by benchmarking the trained context-guided agent models resulted in the creation of datasets CA0, CA1, and CA2 for model iterations A000, A100, and A200, respectively. Statistical description of the datasets (see table D.8 on page 224) revealed the collection results to be consistently lower for all model iterations than those achieved by the reference model (see figure 6.5 on page 141). Even worse results were recorded for their completion probability *P(com)* (see table 6.3 on page 140), which did not exceed the value of *P(com)* = 0.15 for any of the models tested.

SNSB	**SNTI**	**Dataset**	M_{col}	M_s	***P(com)***
A000	A000	CA0	0.23	0.12	0.13
A100	A100	CA1	0.23	0.12	0.15
A200	A200	CA2	0.23	0.13	0.12

Table 6.3: General benchmarking results for context-guided agent model iterations A000, A100, and A200 aggregated in datasets CA0, CA1, and CA2 respectively. Presented values include collection score median M_{col}, probability of completion *P(com)* and full score median M_s.



(a) Collection score distribution density for A000, A100, and A200.



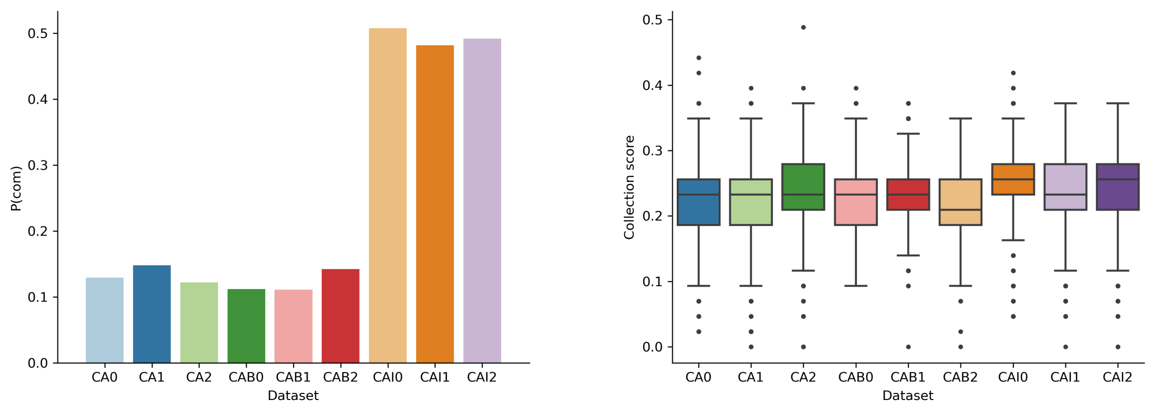
(b) Collection score boxplots for A000, A100, and A200.

Figure 6.4: Visual comparison of collection score distributions for benchmarked learning model iterations A000, A100, and A200, recorded in datasets CA0, CA1, and CA2, respectively.

To establish whether it was only one of the models in each iteration pair affecting the scores, we decided to benchmark all SNSB and SNTI models individually. Each of them was paired with fallback logic in place of the other learning model. First, using the same benchmark procedure, we simulated SNSB models A000, A100, and A200, with the SNTI model deactivated and instead calling SNTI fallback and ad hoc logic. Then, we benchmarked SNTI models A000, A100, and A200, with the SNSB model deactivated. Visual comparison of the results of all these benchmarks, presented in figure 6.5 on page 141 indicated that SNSB models were underperforming and compromising both the completion and collection objectives of the game. SNTI only benchmarks delivered better completion and collection scores. However, their collection scores were still lower than those of the reference agent. This suggested that all the trained SNTI models selected their targets in a less optimal manner than random logic. SNTI A000 paired with fallback SNSB produced the highest completion probability *P(com)* = 0.51, median collection score M_{col} = 0.26, and median full score M_s = 0.53 from the model combinations benchmarked (see table 6.4 on page 141).

SNSB	**SNTI**	**Dataset**	M_{col}	***P(com)***	M_s
A000	Fallback	CAB0	0.23	0.11	0.12
A100	Fallback	CAB1	0.23	0.11	0.12
A200	Fallback	CAB2	0.21	0.14	0.12
Fallback	A000	CAI0	0.26	0.51	0.53
Fallback	A100	CAI1	0.23	0.48	0.16
Fallback	A200	CAI2	0.26	0.49	0.17

Table 6.4: Benchmarking results for individual SNSB models paired with fallback SNTI logic, and vice versa. Presented values include collection score medians M_{col}, probabilities of completion *P(com)* and full score medians M_s.



(a) *P(com)* for benchmarked models.

(b) Collection score value distributions for benchmarked models

Figure 6.5: Visual comparison of collection score distributions and probability of completion recorded for benchmarked SNSB and SNTI models in datasets CA0-CAI2.

Context-Guided Agent Selection

Benchmarking results confirmed that all investigated combinations of SNSB and SNTI models were capable of playing the game by achieving non-zero collection and completion scores. However, their competency varied and was consistently below that of the reference model. The deployed SNSB models, representative of high-level pursuit of both collection and completion objectives, were unable to direct the agent effectively. They scored lower on both completion and collection, in comparison to SNTI models with SNSB fallback. We concluded that SNSB learning was insufficient, and further evaluation of an SNSB model would not provide additional information in our research. However, since the context-guided agent design does not enforce the number of learning models deployed in an agent, we decided to continue our work with a single learning model: the highest-scoring SNTI model, A000. We paired it with SNSB ad hoc fallback, as we did for benchmarking.

Summary

While the trained context-guided agent model did not appear to have surpassed the reference agent in terms of gameplay performance, it had proven to be functional. We successfully applied the context-guided agent design in practice, in a commercial game environment, and successfully deployed an agent valid in the context of the game's environment. As such, we considered the trained and operational context-guided agent model to be a practical response to research question **RQ1**. It also constituted research output **O5**.

6.3.4 Agent Versus Agent

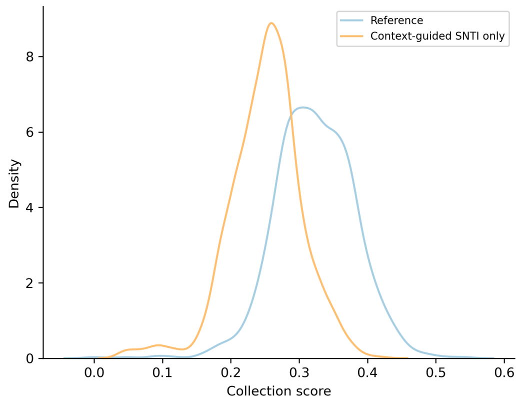
Assumptions

We assumed the context-guided agent would be able to achieve higher or similar collection and completion game scores (s_{colC} and P_{comC}, respectively) to the reference agent (s_{colR} and P_{comR}, respectively). This led us to formulate hypothesis H_{1-REF} of the context-guided agent achieving higher collection scores, and a higher probability of completion, than the reference agent:

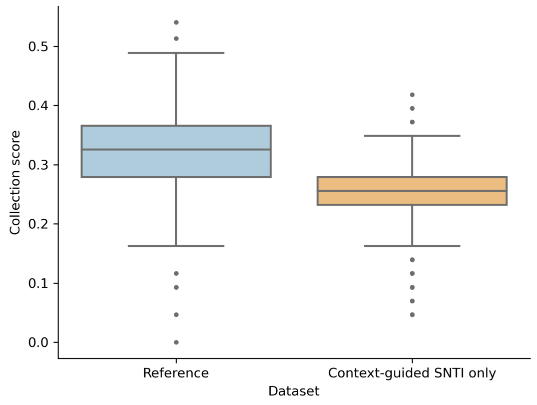
H$_{0-REF}$: $s_{colC} < s_{colR}$ or $P_{comC} < P_{comR}$
H$_{1-REF}$: $s_{colC} >= s_{colR}$ and $P_{comC} >= P_{comR}$

Comparing Agent Models

To test hypothesis H_{1-REF}, we established the context-guided agent's collection and completion scores' value distributions to be equivalent to those exhibited in dataset CAI0. Reference agent's collection and completion scores' value distributions were equivalent to those exhibited in dataset REF.



(a) Collection score distribution density.



(b) Collection score boxplots.

Figure 6.6: Visual comparison of collection score distributions for the context-guided agent using the SNTI learning model A000, and the reference, BT agent.

We moved to compare these collection score distributions visually and quantitatively. A difference between the two was observable in visualisation, with the reference model scoring higher (see figure 6.6 on page 143). Since both datasets were generated in the agent simulator, we assumed their value distributions to be normal. Review of their QQ plots confirmed this (see figure D.9 on page 223). Based on the visual observation of the score difference, and the collection score component of hypothesis H_{1-REF}, we put forward local hypothesis $H_{1-REF-C}$, stating that the mean of collection scores in the reference agent generated dataset REF m_{REFcol} was higher than the mean of collection scores $m_{CAI0col}$ generated by the context-guided agent in dataset CAI0:

$\mathbf{H}_{0-REF-C}$: $m_{REFcol} <= m_{CAI0col}$
$\mathbf{H}_{1-REF-C}$: $m_{REFcol} > m_{CAI0col}$

Different sample sizes for normal datasets REF and CAI0 informed the decision to test the hypothesis using one-sided, Welch's t-test ($\alpha = 0.01$, sample sizes $n_{REF} = 1000$ and $n_{CAI0} = 10{,}00$). Test results ($p = .001$, $s = 29.2$) rejected the null hypothesis $H_{0-REF-C}$, under conditions examined. In the investigated scenario, the reference agent was found to significantly outperform the context-guided agent in collection scores achieved.

We then compared the completion scores of the reference and the context-guided agents. The completion probability of the reference agent $P(com_{REF}) = 0.57$ appeared to be higher than the completion probability of the context-guided agent $P(com_{CAI0}) = 0.51$. Based on this observation, and the completion score component of hypothesis H_{1-REF}, we put forward local hypothesis $H_{1-REF-W}$, stating that the probability of completion of the reference agent $P(com_{REF})$ was higher than the probability of completion of the context-guided agent P(com_{CAI0}):

H$_{0-REF-W}$: $P(com_{REF}) <= P(com_{CAI0})$
H$_{1-REF-W}$: $P(com_{REF}) > P(com_{CAI0})$

To test the hypothesis we conducted a one-sided, proportion z-test (α = 0.01, sample sizes n_{REF} = 1000 and n_{CAI0} = 1000). Test results (p = .004, z = 2.65) rejected the null hypothesis H$_{0-CAI0-W}$, under conditions examined. In the investigated scenario the reference agent was found to significantly outperform the context-guided agent in completion scores achieved.

Summary

Since both hypotheses H$_{1-REF-C}$ and H$_{1-REF-W}$ were not rejected, we were able to conclude that hypothesis H$_{1-REF}$ of the context-guided agent achieving higher collection and completion scores than the reference agent was rejected. Thus, context-guided learning agents did not achieve approximately similar or better game scores than reference agents, in the same game environment conditions.

6.4 Agent Evaluation

6.4.1 Overview

Goals

The goal of the context-guided agent model evaluation was to experimentally evaluate the gameplay performance of the context-guided agent model, in order to determine whether it could cope with unseen game environments in the game *60 Seconds!*, and address research question **RQ2**. The training of the evaluated context-guided agent was presented in the Agent Training section. The evaluation was conducted with the use of the quantifiable game score metric, defined in the Game Score section of Chapter 5: Game Score Study. Evaluation was divided into an offline trial, which evaluated the agent model, and an online trial, which evaluated human players. Both parts of the experiment were conducted in normalised conditions of the game environment of *60 Seconds!*, using a set of new game levels designed for the purposes of the evaluation.

Report

All relevant information about the setup, procedure, and results of the experimental evaluation is reported in the sections below. Supplementary documentation for the online evaluation is provided in Appendix B.

6.4.2 Setup

Design

Gameplay performance was measured in an experimental evaluation trial, challenging players to face previously unseen game environments in the form of new game levels. A single evaluation session in the trial required players to play through the presented levels without interruption. Five levels, each with a different and unique layout, were created to be played in the course of the evaluation. New levels were based on the standard, shared environment architecture used in the original game levels. All of them were constructed from pre-designed room prefabricates, created by the game's developers. It was done to root new environments in a familiar, spatial and gameplay context, preventing any potential overhead that could have been caused by departing from the game's familiar setup. In contrast to regular game levels, which are pseudo-randomly populated with *items* to be collected, every evaluation level had its own, fixed *item* type and positioning setup, which was repeated for all playthroughs of a level. On each level, the player was given the standard $t_{scav} = 60$ s collection time, and $t_{exp} = 5$ s exploration time, similar to the game's *scavenge challenges*. The total weight of items to be collected in each evaluation level was set to $T = 41$. Playthrough of each level in the trial generated a gameplay telemetry trajectory data sample, which could later be used to analyse the gameplay performance of the player.

The same evaluation trial design was deployed in two, approximately similar setups:

- **Online**: in the instrumented, commercial game environment of the game *60 Seconds!* for human players.
- **Offline**: in the agent simulator version of the game environment for AI agents.

Online: Human User Evaluation

To benchmark the context-guided agent against that of "human experts", it was trained to imitate [248], we had prepared an experimental evaluation trial deployment for the live audience of the commercial version of the game *60 Seconds!*. This was made possible by the instrumentation of the game environment for the purposes of conducting online experimental evaluations, documented in the Experimental Evaluations section in Chapter 4: Context-Guided Agents. The experiment was approved by the University of Glasgow College of Science and Engineering Ethics Committee, as detailed in the Evaluation section in Chapter 4: Context-Guided Agents.

New game levels designed for the evaluation were presented to users in the familiar interface and format of *scavenge challenges*, taking advantage of their intrinsic play motivation and familiarity with the game. The gameplay in the trial was identical to the one experienced and known to players from the standard game sessions, and did not alter the rules of *scavenge*, or the control over the player's avatar. The only features manipulated for the purposes of research-driven evaluation were the new level designs used for evaluation, and the delivery method of presenting these levels in a sequence, rather than individually. The online evaluation was designed as a randomised controlled experiment. Every player taking part in the experiment was randomly assigned to one of three groups:

- **Group 1**: participant played a sequence of five evaluation levels in a predetermined, ascending order 1-5.
- **Group 2**: participants played a sequence of five evaluation levels in a predetermined, descending order 5-1.
- **Group 3**: for each participant, the order of the sequence of five evaluation levels was individually randomised.

Randomised control was introduced to provide enough variance to minimise any population bias resulting from playing evaluation levels in a specific order. We also assumed the first level allocated to a participant would be considered a calibration scenario. The number of users that would be involved in the trial was purposefully undetermined at the time of evaluation and proposal. Since the experiment delivery was planned to use a live game, the experiment was open-ended, and we intended to acquire as many samples from the players willing to participate as possible.

Offline: Agent Evaluation

An offline version of the experimental, evaluation trial was deployed in the agent simulator, presented in the Agent Simulator section of Chapter 4: Context-Guided Agents, to evaluate the play skill of the context-guided agent model. No changes to the environment conditions designed for the evaluation were introduced, with the exception of those discussed in the Game Environment Changes section of Chapter 4: Context-Guided Agents.

The offline evaluation was planned to be executed after the online evaluation had concluded. The number of valid, online evaluation sessions recorded was expected to inform the number of agent evaluation sessions to be simulated. Since we understood that AI agents would not exhibit any bias, resulting from the specific ordering of the sequence of levels played, all offline evaluation sessions were planned to be simulated in a predetermined, ascending order, 1-5.

6.4.3 Procedure

Online: Human User Evaluation

All active players of the game were considered as potential participants of the evaluation and were able to learn about it by following the standard communication channels of the game's developer. No special equipment or setup was required to take part. Every user interested in taking part was able to easily access the experimental evaluation through the *scavenge challenge* mode of play, present in the game. Upon first access to the evaluation, they were greeted with the consent and information page, where they were introduced to the procedure for the evaluation trial. If they had consented to take part in our evaluation and confirmed their age to be 16 or above, they were allowed to participate in the trial. Players were able to opt out of their consent at a later date through the appropriate option, present in the game's settings menu. Failure to provide consent or verify their age prevented players from accessing the evaluation trail. They would also not receive digital, in-game rewards that were only given only to players who had participated in the evaluation trial.

Before the trial began, each player was presented with an information screen explaining the differences between the evaluation and original gameplay, as well as the goal of the evaluation. The screen also featured consent and age verification check box controls. Information was provided in textual form, embedded in the game's user interface, localised for the active language version of the game, including English, Polish, French, German, Italian, Spanish, Portuguese Brazilian, Japanese, Simplified Chinese, Korean, and Russian. All other trial-related textual content was also translated accordingly. Leaving the information screen initiated the evaluation gameplay and began logging research data. At the end of the evaluation trial, data logging was deactivated, and the participant was presented with a debriefing screen.

The online data collection pipeline was used to crowdsource the gameplay telemetry data from users, who chose to participate in the evaluation. Upon completing the evaluation, gameplay trajectories logged for each of the levels they played were uploaded to the developer's server storage.

Offline: Agent Evaluation

The offline evaluation was executed after the conclusion of the online evaluation. All gameplay trajectory data samples were generated in the agent simulator environment, running the context-agent model in inference mode at 10 times the regular execution speed. Simulation was executed on local hardware, described in the Hardware section. Since agent simulations were executed offline, there was no need to upload data samples through the game's online data collection pipeline, as was the case with the online part of the experiment.

6.4.4 Results

Online: Human User Evaluation

The online part of the experimental evaluation was initiated on the 21st of May 2019, when the game's update containing the evaluation challenge was uploaded by the developers of the game (update 1.403 *Rocket Science!*). The online evaluation was concluded on the 29th of April 2022, when the developers of the game deactivated research data logging. There were 94,735 valid gameplay trajectories generated for all evaluation levels by 18,947 unique users, who played through the evaluation without interruption. Since evaluation levels had a total collectable item weight set to $T = 41$, which was different from the established $T_d = 43$ (see Play Skill Classification section of Chapter 5: Game Score Study), scores recorded during evaluation were normalised accordingly. We transformed the data collected during the online experiment using the data collection and processing pipeline and aggregated valid player samples in the following datasets:

- **EHA**: player scores from all evaluation samples recorded.
- **EHS1 - EHS5**: player scores achieved with respect to the sequence playthrough order. First playthroughs were placed in dataset EHS1, second playthroughs in EHS2, and so on.
- **EHL1-EHL5**: player scores achieved in specific level environments.
- **EH**: player scores from all playthroughs, but with the first, calibration playthrough excluded for each participant (see table 6.5 on page 151).

Offline: Agent Evaluation

Upon the completion of the online part of the experimental evaluation, we recorded the number of users who contributed valid gameplay trajectories. This informed the number of evaluation sessions to be simulated in the agent simulator environment for the offline part of the experiment. A total of 94,735 gameplay trajectories were generated for all evaluation levels by 18,947 virtual users. While each virtual user was technically an instance of the context-guided agent, individual recordings streamlined our organisation of the collected data for further analysis. We repeated the score normalisation routine, applied earlier to the scores from the online evaluation, for offline evaluation scores. Data from the offline experiment was transformed using the processing pipeline and aggregated into the following datasets:

- **EAA**: agent scores from all evaluation samples recorded.
- **EAS1 - EAS5**: agent scores achieved with respect to the sequence playthrough order. First playthroughs were placed in dataset EAS1, second playthroughs in EAS2, and so on.

- **EAL1-EAL5**: agent scores achieved in specific level environments.
- **EA**: agent scores from all playthroughs, but with the first, calibration playthrough excluded for each virtual user (see table 6.6 on page 151).

6.4.5 Agent Versus Humans

Assumptions

We assumed the evaluated context-guided agent would achieve higher or similar collection and completion game scores (s_{colC} and P_{comC}, respectively) to human players (s_{colH} and P_{comH}, respectively) participating in the evaluation. This led us to formulate hypothesis H_{1-RQ2}, and null hypothesis H_{0-RQ2}:

H_{0-RQ2}: $s_{colC} < s_{colH}$ or $P_{comH} < P_{comH}$
H_{1-RQ2}: $s_{colC} >= s_{colH}$ and $P_{comC} >= P_{comH}$

Description of Human Player Evaluation Scores

We assumed that human players participating in the online evaluation would score less in the first calibration playthrough of their evaluation session. Due to the way datasets EHA and EH were aggregated, we were able to express that assumption in the form of the hypothesis H_{1-CAL} of the collection scores in EHA being lower than those found in EH in terms of central tendency:

H_{0-CAL}: collection scores in EHA are higher than or equal to those in EH
H_{1-CAL}: collection scores in EHA are lower than those in EH

Visual examination of differences between collection score value distributions of the five game playthroughs played in order by the participants of the online experiment in each evaluation session (see figure 6.7 on page 150) supported the notion that the first game in the sequence exhibited lower collection scores.

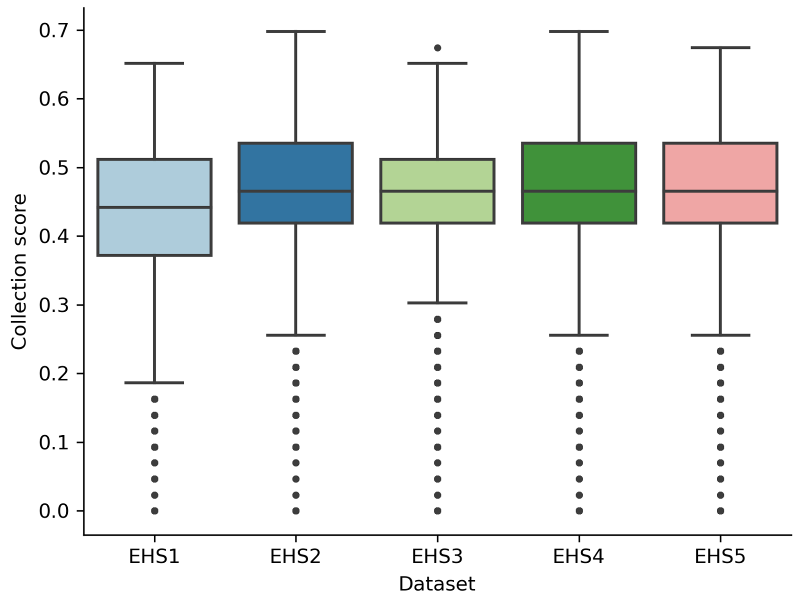


Figure 6.7: Collection score value distributions for each of the five game playthroughs during each online evaluation session, aggregated in datasets EHS1-EHS5.

To determine if that was the case, we compared the collection score distributions in datasets EHA and EH. To inform our method selection for the comparison, we visually inspected QQ plots for both distributions (see figure D.10 on page 223), which suggested normality violations in both datasets. We then quantitatively tested their normality using Shapiro-Wilk tests for EHA (α = 0.01, n = 94,735) and EH (α = 0.01, n = 75,788). Both EHA (p = .001, s = 0.93) and EH (p = .001, s = 0.92) violated normality assumptions under conditions examined, confirming our visual observations. Normality violations in the collection score distributions of EHA and EH informed our decision to use a one-tailed, non-parametric Mann-Whitney U test to compare these distributions. Test results (α = 0.01, n_{EHA} = 94,735, n_{EH} = 75,788, p = .001, s = 3,485,101,330.5) rejected the null hypothesis, under conditions examined. The collection score distribution of EHA was found to exhibit significantly smaller values than the collection score distribution of EH. This indicated that calibration playthroughs suffered from lower gameplay performance, likely caused by the lack of immersion and focus, which only improved for the subsequent sessions. Hence, calibration playthroughs were excluded from further analysis. EH was used as dataset representative of the online evaluation samples (see table 6.5 on page 151).

Dataset	EH
Sample size	75,788
Values	s_{col}
Measurement	**Value**
Min	0.0
Max	0.7
Mean	0.46
Median	0.47
Mode	[0.46511628]
Standard deviation	0.1

Dataset	EH
Sample size	75,788
Values	s_{com}
Measurement	**Value**
Min	0.0
Max	1.0
Mean	0.89
Median	1.0
Mode	[1.]
Standard deviation	0.31

Table 6.5: Statistical description of the s_{col} and s_{com} value distributions generated by the online evaluation playthroughs, with the calibration playthrough excluded, aggregated in dataset EH.

Description of Agent Evaluation Scores

Since we excluded calibration playthroughs from the online evaluation results for human participants, we applied the same approach to the agent score data from the offline evaluation. This resulted in dataset EA being representative of the offline evaluation samples that were deemed relevant for further analysis (see table 6.6 on page 151).

Dataset	EA
Sample size	75,788
Values	s_{col}
Measurement	**Value**
Min	0.0
Max	0.51
Mean	0.3
Median	0.31
Mode	[0.310,438,07]
Standard deviation	0.05

Dataset	EA
Sample size	75,788
Values	s_{com}
Measurement	**Value**
Min	0.0
Max	1.0
Mean	0.59
Median	1.0
Mode	[1.]
Standard deviation	0.49

Table 6.6: Statistical description of the s_{col} and s_{com} value distributions generated by the offline evaluation playthroughs, with the calibration playthrough excluded, aggregated in dataset EA.

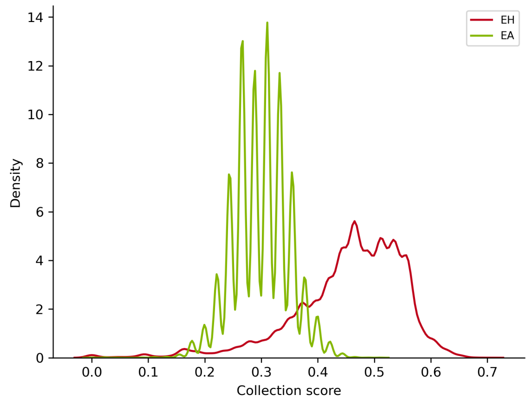
In preparation for method selection for comparing agent scores with those of human participants, we reviewed EA's QQ plot, which suggested the distribution was normal (see figure D.11 on page 223). We conducted a Shapiro-Wilk test to quantitatively confirm it ($\alpha = 0.01$, sample size $n = 75{,}788$, $p = .001$, $s = 0.98$), but the test results revealed it to be non-normal, under conditions examined.

Comparison of Agents and Humans

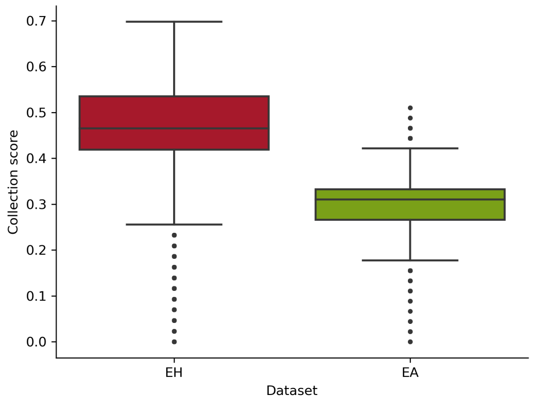
Results from the online and offline parts of the experimental evaluation supplied us with data to conduct a normalised comparison of the measured gameplay performance of the context-guided agent and human players. This was required to provide a formal response to RQ2, to determine how well trained context-guided agents operate in unseen environment scenarios, in normalised conditions, in comparison to human players. Examination of the median collection score M_{col} and completion probability *P(com)* values in EH and EA suggested that human players performed better in collection and completion, in the normalised conditions of the experiment (see table 6.7 on page 152). Visual inspection of the collection score value distributions in EH and EA also supported this notion (see figure 6.8 on page 152).

Experiment	Dataset	M_{col}	Collection play skill	*P(com)*
Online - humans	EH	0.47	Average	0.89
Offline - agents	EA	0.31	Low	0.59

Table 6.7: Results from the online and offline parts of the experimental evaluation, aggregated in datasets EH and EA. Presented values include collection score medians M_{col}, probabilities of completion *P(com)* and full score medians M_s.



(a) Collection score distribution density.



(b) Collection score boxplots.

Figure 6.8: Visual comparison of collection score distributions from results of the online and offline experimental evaluation, aggregated in datasets EH and EA, respectively.

Based on the visual and quantitative observations of the collection score difference, as well as the collection score component of hypothesis H_{1-RQ2}, we put forward local hypothesis $H_{1-RQ2-C}$, stating that the median of collection scores of human participants in dataset EH M_{EHcol} was higher than the median of collection scores of the agent in dataset EA M_{EAcol}:

$\mathbf{H}_{0-RQ2-C}$: $M_{EHcol} <= M_{EAcol}$
$\mathbf{H}_{1-RQ2-C}$: $M_{EHcol} > M_{EAcol}$

Since collection score value distributions in datasets EH and EA were found to be non-normal, we chose a one-tailed, non-parametric Mann-Whitney U test (α = 0.01, sample sizes n_{EH} = 75,788, n_{EH} = 757,88) to compare the two distributions. Test results (p = .001, s = 5279830675.5) rejected the null hypothesis $H_{0-RQ2-C}$, under conditions examined. This indicated that human players significantly outperformed the context-guided agent in collection scores achieved during the experimental evaluation.

We then compared the completion scores of humans who participated in the evaluation, and the evaluated context-guided agent model. The completion probability of human players $P(com_{EH})$ = 0.89 appeared higher than the completion probability of the context-guided agent $P(com_{EA})$ = 0.59. Based on this observation, and the completion score component of hypothesis H_{1-RQ2}, we proposed a local hypothesis $H_{1-RQ2-W}$, stating that the probability of completion of evaluated human players was higher than the probability of completion of the evaluated context-guided agent:

$\mathbf{H}_{0-RQ2-W}$: $P(com_{EH}) <= P(com_{EA})$
$\mathbf{H}_{1-RQ2-W}$: $P(com_{EH}) > P(com_{EA})$

To test the hypothesis we conducted a one-sided, proportion z-test (α = 0.01, sample sizes n_{EH} = 75,788 and n_{EA} = 75,788). Test results (p = .001, z = 131.82) rejected the null hypothesis $H_{0-RQ2-W}$, under conditions examined. In the investigated scenario, evaluated human players significantly outperformed the context-guided agent in terms of completion scores achieved.

Summary

Since both local hypotheses, $H1-RQ2-C$ and $H1-RQ2-W$, were not rejected, we were able to conclude that hypothesis $H1-RQ2$ of the context-guided agent achieving higher collection and completion scores than the human players was rejected. Our findings indicated that context-guided learning agents did not achieve approximately similar or better game scores than those of human players, in approximately similar game environment conditions of unseen game scenarios. The recorded game score data showed that while the agents were able to achieve non-zero scores, their collection and completion scores were on average datasetEHEAcolcomdiffpercent% lower than those reached by human players. This translated into *Low* play skill, in contrast to human players performing *Average*.

6.5 Agent Review

6.5.1 Overview

In the course of the study, we demonstrated that the proposed context-guided agent design and deployment workflow can be successfully applied in the context of a commercial game environment. By following the workflow we architected, implemented, and trained a context-guided agent model, which was capable of valid operations in the context of the game's environment and pursuing the game's gameplay objectives. However, quantitative analysis of the data collected during experimental evaluation and benchmarking of the trained context-guided agent revealed it to exhibit limited competence in playing the game. It scored lower than an ad hoc agent playing standard game levels, as well as human players playing an experimental set of levels in controlled conditions. Its collection play skill was classified as *Low* in both benchmark and evaluation, which could be considered on par with a beginner human player. The evaluated agent featured only one of the two learning models we set out to train. Due to quality issues detected with all trained iterations of the SNSB learning model, we decided to replace its learning logic with ad hoc fallback.

We set out to review and discuss the results of the study with respect to the output context-guided agent: its design, training process and the evaluated output quality. We based the review on areas that could have had an influence on the agent's performance. For each of them, we identified potential issues, phrased them as questions, and addressed them as part of the review. Examined areas included:

- Agent design and implementation
- Learning logic design and implementation
- Learning data quality
- Training learning models

6.5.2 Agent Design and Implementation

We conducted the study under the assumption that the proposed context-guided agent design and deployment workflow would work in practice. One of the study's goals was to test this assumption and execute on the proposed theory to empirically address research question **RQ1**. Any underlying problems with the design proposal, applying the design, or relevant engineering work would have prevented the context-guided agent instance from operating, as planned.

Is the proposed context-guided agent design applicable in practice?

As demonstrated in the Agent Training section, by following the context-guided agent design proposal and its deployment workflow, described in the Deployment Workflow section of Chapter 4: Context-Guided Agents, we were able to deploy a context-guided agent instance, which integrated learning with ad hoc logic. It was found to be functional and valid in the context of the target game's environment. This constitutes a validation of the context-guided agent design proposal.

Were any mistakes made when designing the behaviour context of the context-guided agent in the study?

While designing the behaviour context of the context-guided agent can be considered a subjective task, left at the discretion of the human expert involved, it can be validated against the target gameplay environment. Both the ad hoc reference agent, as well as the output context-guided agent produced in the study, were able to play the game, and pursue its gameplay objectives, achieving non-zero scores. Thus, the designed behaviour context and the resulting agent's design could be considered valid.

Were there any underlying engineering problems in the agent behaviour logic implementation that could have affected the output context-guided agent?

Although we had encountered several technical issues during the development of the context-guided agent logic, we resolved all of them before agents involved in the study were benchmarked and evaluated. If any engineering problems had persisted, we would have observed failures or inconsistencies with the behaviour of all agents deployed, including the ad hoc reference agent. Since that was not the case, it is unlikely that the context-guided agent's performance was affected by behaviour logic-related engineering issues.

6.5.3 Learning Logic Design and Implementation

Incorporating learning logic in the developed agent was a key part of applying the proposed design. Invalid design decisions concerning the positioning of the learning logic in the context of the agent's behaviour were likely to diminish the quality of the agent's output. Even with the right design decisions, using unsuitable methods for training learning logic could have introduced risks of producing suboptimal models. Additionally, although learning was successfully integrated into BT architecture in our implementation of learning nodes, its software stability was untested, due to reliance on the *ML-Agents* external library.

Was learning suitable in the behaviour context designed for the context-guided agent?

As part of the context-guided design proposal in the Establish Learning Subtrees section of Chapter 4: Context-Guided Agents, we established that logic, which is required to be adaptive, data-driven, and could not be represented well with ad hoc authoring, would benefit from a learning-based representation. Although we deployed a working ad hoc logic-based reference agent, its decision-making was implemented using randomisation and intuition-driven heuristics, as discussed in the Implement Context Architecture section. Further sequential logic-based improvements to the ad hoc logic model would have likely involved more intuition-based trial and error to refine the heuristics and replace randomisation. A data-driven approach was preferable to capture the multidimensionality of the choices made by the agent in the context of the game's environment. With an expert gameplay trajectory dataset available for imitation, learning was considered a viable and suitable option for the context-guided agent. This approach was also supported by the length of learning episodes, which matched the length of an entire game session, and featured sparse rewards, issued in the context of multiple global and local gameplay objectives.

Were learning subtrees positioned properly in the context of the agent's behaviour logic?

While we found learning suitable in the behaviour context of the designed context-guided agent, its applicability was only considered for the agent's high-level decision-making logic. We wanted to avoid training a navigation model, as discussed in the Navigation Abstraction section. The navigation approximation provided by the navigation mesh and A* pathfinding, combined with ad hoc steering logic, was considered sufficient for the purposes of the agent simulation in our research. The *proxy* layer ad hoc connections between the *macro* and *micro* layers of the ad hoc representation were better serviced using BT nodes. This only left three pieces of functionality to be potentially outfitted with learning: SNSB, SNTI, and SNTA. As described in the Establish Learning Subtrees section, SNTA was deemed the least impactful in terms of the collection and completion objectives, of the three. SNSB had a direct impact on both these objectives, while SNTI was key to maximising informed collection. As such, we considered the decision to model SNSB and SNTI using learning as the most optimal approach in the explored scenario. Multiple learning models used in an agent were consistent with the context-guided design proposal.

Were the learning algorithms used for training learning models suitable for the task?

As discussed in the Establish Learning Subtrees section, algorithms considered for both learning models included PPO for RL, together with BC and GAIL for IL. Such a combination was found to produce good results in prior research conducted in game environments [24, 84, 271] and appeared to be suitable for the requirements of the context-guided agent. We originally wanted to avoid excessive bias from the demonstration gameplay trajectories through optimal balancing of BC and GAIL for IL [130, 139], while maximising bootstrapping of RL through applying BC to shorten training time considerably. However, even the worst-case scenario of BC overfitting training data was considered acceptable, as SNSB and SNTI were not holistic models. Thus, overfitting would have only occurred in an isolated domain of strategic decision-making, rather than in the full context of the environment. However, as described in the Train Learning Models section, we discovered that the *ML-Agents* software stability became unreliable when BC was enabled and consistently crashed the training process. We were forced to exclude BC and rely solely on PPO and GAIL, instead. This resulted in prohibitive training times to produce higher quality output (see table D.7 on page 221), as was the case with the highest scoring SNTI model from iteration A000, which took 24 hours 18 minutes 47 seconds to train. Models with decreased complexity of the network via relevant hyperparameter adjustment (see table 6.1 on page 137) incurred lower training cost but also delivered substantially lower quality of output. Thus, PPO and GAIL had shown that, given more time, they were capable of delivering better models. Functionally, they were suitable for the task, but from an industrial perspective, they were far from optimal. If it were not for the involuntary exclusion of BC from the suite of learning algorithms used, policies learned would have likely benefited from demonstration-based pretraining. This would have led to shorter training times and potentially higher-quality policy outputs.

Was the learning integration software stability sufficient to train learning models?

We were able to successfully integrate the learning functionality serviced by *ML-Agents* and *PyTorch* into the game environment, and the BT ad hoc architecture used to model behaviours of designed agents. For as long as we were operating in the realm of Unity's simple sample environments and scenarios, *ML-Agents* learning operated as expected. However, using IL in the game environment introduced software stability issues that substantially impacted our work. Previously discussed problems with BC resulted in the exclusion of an algorithm we considered essential for achieving optimal results. Moreso, in the process of training our GAIL-based models, discussed in the Training Learning Models section, we found out that demonstrations encoded in Unity's proprietary format, enforced for IL training data in *ML-Agents*, were highly unreliable, and potentially unusable. In some cases, they took extremely long to be loaded upon training startup, in others, they would crash the training. This forced us to limit the number and size of demonstrations used, as outlined in the Acquire Data section. Furthermore, we were unable to train more than one model simultaneously, which further increased the training time. These problems consumed a substantial amount of computing and engineering time, introducing limitations that directly affected the quality of learning model training. Unity did not provide support beyond their sample scenarios, as indicated in the Training Learning Models section. While we were able to train functional learning models in a commercial game environment using the library, the process was expensive, and we found its output to be suboptimal. It is worth noting that all prior research work with *ML-Agents* we consulted was conducted in controlled, research-specific environments, rather than commercial game environments [55, 140, 144, 271, 278]. We cannot categorically state there were no issues related to our own integration work that could have affected the flow of the training using *ML-Agents*. However, the problems we encountered, the volume of issues experienced by other users, and the lack of complex learning scenarios trained using the library appear to be indicative of a problem with *ML-Agents*, itself.

6.5.4 Learning Data Quality

The learning model is as good as the data it was trained on. We understood that even with the most reliable and effective learning pipeline, suboptimal provision of training data could have had a significant negative impact on the trained policy. For the most part, input data was under our control. The action, observation and reward spaces defined for the SNSB and SNTI models had to be examined. Additionally, a review of both the quality and quantity of the demonstration data used with IL was necessary.

Was the quality of demonstration data used for SNSB and SNTI optimal?

The design of the context-guided agent assumed its training would involve imitation of a top-skill persona, established through game score exploration and clustering in the Top-skill Persona section of Chapter 5: Game Score Study. We accumulated the training dataset by encoding gameplay trajectories of the top-skill persona, which constituted the top-scoring 7% of all games recorded in our research, into Unity's proprietary demonstration format. This approach was consistent with the notion that the quality of the training dataset is more important than its volume, and that approximately the top performing 5% of the demonstrations should be used as training input [130]. To ensure high quality of the data, our demonstrations were sourced from actual human experts, rather than simulated users. The latter is a common practice in IL research focussed on autonomous agents [94]. Thus, we considered our expert demonstration data to be of the highest possible quality and representative of the top play skill in the context of the game.

Was the quantity of demonstration data used for SNSB and SNTI optimal?

While the quality of the training input data was considered more important than quantity, we expected to use all of the 765 demonstrations featuring 76,406 encoded episodes, for model training. Availability of high-quality data, as well as the requirements of BC [139] informed our motivation to do so. Unfortunately, as discussed in the Training Learning Models section, issues with *ML-Agents* not accepting more than 180 demonstrations for SNSB, and 100 demonstrations for SNTI, resulted in using only a part of the demonstration data prepared. Due to a limited number of model training iterations, we were unable to assess if this constraint influenced the quality of the end policy. However, prior industry research found that for training agents with human play styling using human expert demonstrations, fewer demos might be sufficient [24]. This is also consistent with the characteristics of GAIL-driven IL [101].

Was the action space optimally defined for SNSB and SNTI?

Due to different objectives and the SNSB and SNTI logic, we defined individual, discrete action spaces for each of them. Action spaces for both models were directly mapped to the gameplay objectives and possible decision states outlined in the Implement Learning section. By restricting the defined action spaces to specific decisions that these models were capable of making, we reduced the potential of the policy to make continuous value-based choices that would have been invalid in the context of the game environment. Due to the structure of the game, there were still decision states that were valid, but would not make sense in specific game states. Such edge cases were addressed by guiding the model away from nonsensical decision states using appropriate negative reward signals. This approach shifted the risk of action space problems to reward signal balancing. We discussed the specifics of the action space of each of the models in the Implement Learning: SNSB and Implement Learning: SNTI sections. We chose to represent available actions in the simplest possible way, which adhered to gameplay objectives, as well as the action input of the ANN. This was considered the most optimal approach in the investigated scenario.

Was the observation space optimally defined for SNSB and SNTI?

We were operating under the assumption that an egocentric representation of the environment features would provide sufficient observation space for the agent's operations. This notion was compatible with our assumption of the agent's partial observability of the environment, discussed in the Observability of the Game Environment section, and in the Agent Perception section of Chapter 4: Context-Guided Agents. Such an approach approximated human-like sensory perception. For SNSB the observation space was defined using high-level information about the game and agent state, totalling 12 vectors. For SNTI the observation space was defined with respect to all types of collectable *items*, totalling 89 vectors. Despite the fact that the SNSB network was updated more often than that of SNTI, the demonstration file size for the latter was substantially larger (approximately 4.34 times larger). Since the action-observation mapping was only recorded at the time of the update, we concluded that the larger observation space size was the factor that contributed to a larger volume of data flowing through the ANN, which manifested through greater sizes of the SNTI demonstration files. As demonstrated in the Context-Guided Agent Benchmarking section, we were able to train a SNTI model iteration A000, whose output was approaching acceptable quality. That was not the case for the SNSB model. We concluded that the observation space for the SNSB model was too small and likely contributed to the limited performance of the trained SNSB policy. To improve it, the observation space would need to be increased, potentially incorporating observation data of SNTI, and more. Such an improvement would result in greater demonstration file sizes, which were already problematic in terms of the *ML-Agents* software stability. Another strategy would be to combine allocentric and egocentric information to enhance the action-observation mappings made by the ANN [84]. Since SNTI models also delivered suboptimal performance, restructuring of its observation space would also be warranted. An alternative approach to SNTI could feature recording observations of all *items* in the environment, not just the closest spotted item, of the given type. When designing the observation spaces for SNTI and SNSB we decided to encode all positioning data as egocentric distance values. Each of them represented distances between the position of a given target and the position of the avatar. This decreased the size of the observation space, as only one floating point value was required, instead of three to represent coordinates in 3D space. However, it is possible that the latter would inform the mappings of the learned policy better, for both models.

Were the reward signals for SNSB and SNTI optimal?

The structuring chosen for the action space, derived from its mapping to gameplay objectives, resulted in introducing both positive and negative reward signals for the agents. Negative rewards were used to discourage the policy from selecting decision states that were nonsensical in the context of the gameplay, as well as to motivate agent progression through minor penalisation. Rewards issued included short-term, local objective-based rewards, and long-term, global objective rewards given at the end of an episode. Reward configuration for SNSB and SNTI can be reviewed in table D.1 on page 212. Despite this division, all rewards were sparse. Models were only updated, prompting an observation-action processing step, when their learning nodes in the BT were ticked. On average, SNSB made approximately 1892.51 steps per episode, while SNTI made 1878.45 steps in a single episode. Total episode time was t_a = 70 s during agent benchmarking, and t_a = 65 s during agent evaluation. In comparison, Unity's default RL setup for *ML-Agents* triggers the observation-action processing step 10 times a second [252]. We intentionally triggered processing steps only when the behaviour context invoked the relevant learning model to avoid recording intermediate states. This was expected to shield the learned policy from absorbing invalid or nonsensical state mappings, originating from intermediate states. Based on the results of the deployed context-guided agent, we concluded that our overcautious approach may have deprived networks in training of a richer data flow per episode. At the same time, we found that the chosen reward values and their balancing in terms of gameplay provided a consistent increase in cumulative reward values for model iterations, which performed better in terms of pursuing completion and collection objectives (see table D.7 on page 221).

6.5.5 Training Learning Models

The quality of the process of training a learning model has the capacity to impact the quality of the output policy. This prompted us to review the way we trained our models, and to investigate the model quality produced, and the costs it generated.

Was the process of training learning models optimal?

The pipeline, which facilitated the learning model training in our research work, was deployed on the basis of engineering and data work, informed by the requirements of Unity, *ML-Agents*, the game environment, and the agent simulator. It enabled us to semi-automatically train models by connecting training data, crowdsourced from millions of users and then processed for use in the game, with the agent simulator environment and the learning libraries involved. This was a multi-step process with several potential points of failure. We were able to apply it in practice to produce functional learning models in a commercial game environment, which can be considered a practical validation scenario of the pipeline and the process of conducting learning with it. However, the lack of full automation prevented us from hands-off generation of new iterations of models by revising their hyperparameters, reward values, as well as action and observation spaces. As described in the Semi-Automated Training Pipeline section, deployment of a more advanced pipeline was deemed too expensive and beyond the scope of this thesis. Multiple problems encountered during our work with off-the-shelf learning solutions and Unity-based environments supported the high-cost argument. Long training times, incurred by multiple models trained, contradicted our perspective of cheap and quick training of small-scale learning models, in the investigated configurations. Software stability issues and troubled IL demonstration generation, discussed in the Train Learning Models section, would have likely disrupted an automated solution and necessitated manual interventions. An automated approach would also not have resolved the data issues identified in earlier review sections. Instead, it could have potentially propagated them further. Considering these challenges and the additional engineering cost required, we chose to use a semi-automated solution, optimal for the purposes of the investigated scenario. However, this approach resulted in the production of a limited number of model iterations.

Was the performance of trained learning models optimal?

The results of the agent benchmarking, analysed in the Agent Versus Agent section, and the experimental evaluation, analysed in the Agent Versus Humans section, indicated that although they were found functional, trained learning models used in the context-guided agent did not achieve optimal performance. Unless the rewards for learning are accurately defined, the cumulative reward is not necessarily representative of a policy's good performance in a learned task [116]. And so, during model training, we also tracked the policy performance with respect to custom metrics of collection, completion, and full game scores. This enabled us to observe differences between scores achieved in the course of training and benchmarking of individual models. Since the training data supplied was uniform for all iterations of a specific type of model, differences in the performance of output models were solely the result of varied hyperparameter configurations.

The set of demonstrations used to train the SNSB model generated mean cumulative reward in the range R_{SNSB} = [37.74, 42.24], while the SNTI training demonstration set recorded mean cumulative reward in the range R_{SNTI} = [32.44, 35.14]. Given that these reward value ranges were generated by top-skill player trajectories, we assumed they were representative of the upper bound of cumulative reward values achievable in training. None of the trained model iterations of SNSB and SNTI produced a cumulative reward that would be contained within these ranges. Models that with the highest cumulative reward values, including SNSB A000 (r = 20.05), SNSB A200 (r = 19.72) and SNTI A000 (r = 15.39), also produced the highest collection scores. Examination of the training results revealed that SNSB and SNTI model iterations A000 and A200 achieved approximately similar values in terms of extrinsic and cumulative rewards (see table D.7 on page 221). GAIL rewards appeared to have had a negligible impact on cumulative rewards achieved by these models. Only iteration A100, whose hyperparameter configuration featured maximised GAIL strength and minimised extrinsic strength (see table 6.1 on page 137), exhibited an elevated GAIL reward value for both SNSB and SNTI. However, this did not boost its performance. Instead, policies of both A100 models became fixated on pursuing the completion objective. This led to achieving high completion scores at the price of zero or near-zero collection scores. GAIL reward collapsed in iteration A000 and fluctuated in all other iterations for both SNSB and SNTI (see figure D.7 on page 222 and figure D.8 on page 222).

SNSB and SNTI models from each iteration were benchmarked in tandem, as described in the Context-Guided Agent Benchmarking section. In all cases they produced approximately similar results, characterised by low completion probability and collection score classified as *Low* (see table 6.3 on page 140). Benchmarking of individual SNSB and SNTI models from each iteration, paired with an ad hoc fallback in place of the other model, revealed that the SNTI A000 model achieved the best results in terms of collection score median, completion probability, and full score median. Other models delivered varied results, but all of them suffered from low full scores, indicating that they regularly scored better in collection or completion, but not both. On the basis of individual model benchmarking results, we replaced the SNSB learning model of the context-guided agent with its ad hoc fallback, also used by the reference agent. We expected the completion probability *P(com)* of the reference and context-guided agents to be comparable in such a configuration. That was not the case, with the ad hoc agent achieving a higher completion probability. This indicated that the SNTI model, believed to primarily influence the collection objective, also impacted the completion objective. The way the behaviour context was architected for deployed agents, as described in the Implement Context Architecture section, the next scavenge behaviour was not selected until the previous one had been concluded. Some behaviours took longer to complete than others. The further away from the avatar the target *item* selected by SNTI was, the longer it took for the BT to process the collection behaviour, and for the SNSB to be invoked. Suboptimal, ill-timed selection of the next target by the SNTI was capable of sending the agent far away from the *exit*, minimising its chances of achieving completion. Since interrupting running behaviours was not supported by the implementation of the context architecture, suboptimal target choices resulted in lower collection and completion scores. While this can be considered a constraint of the way the context architecture was implemented as a BT, it would not have been an issue if SNTI had provided optimal targeting. We believe that in the investigated scenario model, learning failed to capture and leverage the relationship between the placement of collectable *items*, the *exit*, and the avatar. For SNTI, including allocentric data about *items* could have potentially supported a better mapping of such relationships.

We concluded that all trained models delivered suboptimal output. Additional model iterations, with a greater scope of hyperparameter revisions and a potential observation space restructuring, would be required to produce potentially more optimal policies. The best-performing models observed during training were the ones driven by extrinsic signals from RL. This implied that IL played a limited role in boosting their performance. This was also supported by low GAIL rewards recorded. While increasing the complexity of the trained network was beneficial in terms of model performance, as shown by SNTI A000, the likely source of our training issues was the configuration affecting GAIL. Based on our prior issues with IL in *ML-Agents*, we were, however, unable to rule out additional, undetected problems with the library affecting the learning process, or encoding demonstrations from our trajectories.

Was the cost of training learning models optimal?

We set out to create an agent model that could be considered industry viable. Thus, the environment, design and deployment workflow, as well as the intended goals of the agent, were defined in line with industrial requirements. The process of the context-guided agent's development was considered similar to that of a typical game agent. The differences in the process originated in learning, training, and deployment. Ad hoc solutions to game AI are controllable and transparent, which reduces the risks of agent behaviours generating low-quality output. The agent's low-quality output is a problem, as well as a factor that extends the development process, in order to refine the behaviour model and its output. Learning's opaque nature increases the risks associated with a prolonged refinement process of the learning model's output. We attempted to reduce these risks using learning model decomposition through context segmentation in the context-guided architecture, which was also expected to service the agent's individual task learning more effectively and efficiently [47]. However, the cost of such risk reduction during context-guided agent development incurred additional costs in terms of data preparation, model training, and model output analysis. Problems and constraints we faced rendered learning data preparation a suboptimal process, which consumed more time and effort than expected.

Deploying any data-driven model involves data preparation, but the unexpected amount of work it generated during our research made it more effort-consuming than anticipated. *ML-Agents* necessitated transforming our gameplay trajectories into binary demonstration assets by replaying each trajectory in the agent simulator environment and recording demonstration data in the process. The speed of this transformation was bound by the maximum, reliable simulation execution speed of the agent simulator $s = 10$. We were unable to sidestep this process and directly extract and use relevant data from the processed trajectories to generate demonstrations, as it was not supported by Unity at the time of our work. This prevented us from creating demonstrations in a more controlled and efficient manner. It also locked us in a workflow that caused multiple and time-consuming problems with demonstration encoding during both the data acquisition stage, described in the Acquire Data section, and the model training stage, discussed in the Train Learning Models section. What we encountered were likely software issues originating in *ML-Agents*, rather than problems specifically related to learning. Had we used an alternative environment, such as the *Godot Reinforcement Learning Agents* [11], which supports transparent encoding of demonstrations using JSON files, we would have been able to generate demonstrations without the constraint of invoking a proxy environment built on top of a game engine. Unfortunately, due to the fact that *60 Seconds!* was originally made with Unity, using an alternative learning library was beyond the scope of our research work.

For the most part, our learning model training was automated and did not require manual intervention, provided no issues were encountered. However, long training time caused by issues with IL and over-reliance on RL, combined with software stability problems of the learning library, introduced additional costs in terms of time, effort, and compute. In our research, the most problematic of the three was time. As Crankshaw *et al.* stated: "(...) a trained model is useless, unless it can be evaluated quickly and reliably" [47]. Long training times delayed the evaluation of trained models. Due to Unity's constraints in loading ONNX model files, the agent simulator environment was unable to consume newly generated model assets and use them for implemented agents. This forced us to rebuild the agent simulator environment with new model assets embedded every time training was completed. The only alternative was to test them in the Unity editor, which supported direct and automatic insertion of newly trained models. However, they could only be evaluated in a single environment instance at a time. This needlessly complicated automation and introduced constraints in the training pipeline, making it less optimal than originally envisioned.

Output analysis for models that had been trained and benchmarked was complicated by software issues unrelated to learning and the context-guided architecture. These issues generated distractions from identification of relevant problems, as unreliable software had to be considered and excluded as a potential culprit. Thus, the output inspection required a human expert with an understanding of the context of the game, the learning process and its input data, the architecture of the agent, as well as the learning integration. In the case of our research, a single person with full knowledge of engineering and design was involved. While we were eventually able to identify actual problems with the context-guided agent itself, such as the limited observation space of SNSB, it took more time and effort than anticipated. In the real-world conditions of the industry, this would present additional issues, as the engineering and design roles would likely be delegated to two or more developers, with different areas of expertise. The resulting communication and coordination overhead would have negatively impacted the process of analysis and potential rectification of problems identified.

6.5.6 Summary

In the review of the deployed context-guided agent, we found the context-guided design to be applicable in practice. We concluded that the deployed context-guided agent's behaviour context was valid and did not suffer from engineering issues. Learning was determined to be suitable for the designed behaviour context of the agent, positioned optimally, and chosen learning algorithms were deemed appropriate for the tasks for which they were selected. However, software stability issues of the *ML-Agents* learning library negatively affected the agent's learning model training process and forced us to exclude BC as one of the used algorithms, resulting in longer training times. The quality of the supplied demonstration data was high and representative of the skill of the best-performing human players of the game. The quantity of demonstrations had to be limited due to the learning library software stability issues, but it was deemed sufficient due to the lower input data requirements of GAIL. While the action space for trained models was considered optimal, the observation space appeared to be too small and likely suffered from the lack of an allocentric data representation. We declared the choice of rewards to be suitable, but their sparse nature, combined with few observation-decision steps during each learning episode, was potentially a factor that limited the flow of data through the neural network, and, in consequence, learning. The policies of trained models were found to deliver suboptimal output. Further revisions and restructuring of the model observation space were suggested as a remedy. Still, the semi-automated nature of the learning model training pipeline limited the number of model iterations we were able to produce. While sufficient for the investigated scenario, in combination with encountered software stability issues of *ML-Agents* it contributed to the higher than expected learning model training cost. We determined such a cost level to be problematic in industrial scenarios.

The analysis conducted in this section constitutes research contribution **C6**.

6.6 Conclusions

In this chapter, we have applied the proposed, context-guided design in a practical setting of a commercial game *60 Seconds!* and experimentally evaluated it in the context of the game's environment. By deploying a functional and valid, context-guided agent model, we addressed research question **RQ1** from an empirical perspective. We then performed quantitative and qualitative analysis of the results obtained throughout agent training and evaluation and delivered an answer to research question **RQ2**. We found that context-guided agent training and deployment are possible in a commercial game environment, using the proposed design. The agent was capable of operating with respect to the gameplay objectives of the game in both seen and unseen environments. However, it was unable to outperform human players in controlled, experimental conditions, in terms of gameplay performance, under the conditions investigated.

In the course of the chapter, we have documented the following research contributions:

- **C6**: analysis of the training process of a context-guided learning agent, capable of playing the *scavenge* segment of the game *60 Seconds!*, developed on the basis of the context-guided agent design; presented in the Agent Review section.
- **O5**: a trained context-guided learning agent, capable of playing the *scavenge* segment of the game *60 Seconds!*, developed on the basis of the context-guided agent design, using game industry off-the-shelf solutions; presented in the Context-Guided Agent Benchmarking section.

Chapter 7

Conclusions

Summary. This chapter discusses the output of our research and its key conclusions. We first summarise the research documented in the thesis and then report the outcomes of our work. A presentation of contributions delivered, challenges encountered, and identified limitations of our research follows. Finally, we discuss the potential for future work in this area and the application of the proposed context-guided agent design in industrial scenarios.

7.1 Overview

7.1.1 Goals

In this thesis, we set out to investigate whether expert-curated learning models could be integrated into the industry state-of-the-practice ad hoc AI architecture with learning model performance and execution guarantees. We expressed this assertion in the form of the context-guided design proposal for video game playing, autonomous agent AI, featuring optimal design intent embedding and safe behaviour flow execution involving reliable neural network controllers. To approach the investigation from a practical and data-driven perspective of an industry case study, an agent instance based on the proposed agent design was deployed in the environment of the commercial video game *60 Seconds!*. It was trained using a mass-scale gameplay telemetry dataset crowdsourced from real users of the game, and game industry state-of-the-practice workflows and off-the-shelf solutions.

Our research goals were expressed through the following research questions:

- **RQ1**: can models with execution and performance guarantees of learning logic be integrated into the game industry state-of-the-practice ad hoc behaviour AI architecture for applied use in video game playing AI?
- **RQ2**: how well can a trained context-guided learning agent perform in unseen game en-

vironment scenarios, in comparison to human players, in approximately similar gameplay conditions?

7.1.2 Procedure

Our work commenced with an examination of the environment of the target, commercial game *60 Seconds!*, and a review of prior research in fields relevant to our investigation. We then formulated the context-guided agent design proposal, featuring BT based ad hoc architecture and context-segmented, small-scale learning models, with respect to the domain requirements of the state of the practice game industry workflows. The next step involved instrumenting the target game environment to support autonomous agent AI training and simulation, training learning models, conducting experimental evaluations with the live audience of the game, and collecting and processing data for the purposes of our research.

Using the game's instrumentation, we acquired and processed gameplay trajectory data for our research from the mass-scale, gameplay telemetry dataset, crowdsourced from over 800 thousand unique users of the game. To quantify gameplay skill exhibited in the collected gameplay trajectories, we proposed a custom game score metric, derived from the game's design context. This enabled us to conduct an exploratory, statistical analysis of the gameplay trajectory sample population. By selective sampling of the population dataset, we extracted the baseline dataset, featuring trajectories characterised by normalised gameplay conditions. The baseline dataset was then used to cluster game scores, identify play skill classes, and create a scavenge collection play skill classifier. The classifier was applied to extract the gameplay trajectory dataset, representative of a top play skill persona from the game.

We then moved to design, train, and deploy the context-guided agent, on the basis of the proposed design, deployment workflow, and collected data. First, the context-guided behaviour context was designed with respect to the context of the game environment. It informed the development of the base BT architecture. Second, the learning subtrees were positioned in the context-guided agent's architecture, and the training of multiple learning models was conducted using the top play skill persona trajectories. On the basis of our conclusions from the training process and benchmarking the gameplay performance of the trained agent models, we selected one instance of a context-guided agent to be evaluated. As part of the experimental evaluation, human players played a sequence of unseen game levels, designed for the purposes of the evaluation, in an instrumented, commercial version of the game. The context-guided agent was challenged to play the same levels, in approximately similar conditions, in the agent simulator environment. The results of agent benchmarking and evaluation with human participants were used to analyse the gameplay performance of the context-guided agent. We concluded the analysis with a review of the agent's design, deployment, and training.

7.1.3 Results

In this thesis, we proposed and demonstrated the practical application of the context-guided agent design, which integrates learning logic, with performance and execution guarantees, into the game industry state of the practice, the BT architecture for video game agent AI. The context-guided agent instance was successfully deployed in the instrumented environment of the commercial video game *60 Seconds!*, incorporating learning models trained using top play skill persona data crowdsourced from the mass-scale dataset of real user gameplay trajectories. Gameplay performance of the agent was measured using a quantitative game score metric, derived from the game's context. The capacity of the context-guided agent to competently play the game was evaluated during its benchmarking in standard game levels and experimental evaluation in unseen game levels. On the basis of their results, we determined that the context-guided agent was able to play the game, pursuing gameplay objectives, and achieving non-zero game scores. It was not able to outperform human players in the conditions examined. We concluded that despite suboptimal output of the deployed agent's learning models, the context-guided agent design had proven valid and applicable in the investigated game environment. In the course of the analysis and review of the agent training, we determined that trained learning models were compromised by suboptimal observation space design, sparse observation-action step recording, a limited number of model iterations, and learning library software stability issues.

The deployed context-guided agent was able to competently operate in unseen scenarios of the game environment, which fulfilled the assumptions and objectives of its design. Moreover, the result of the suboptimal agent training best illustrated the value of the context-guided agent design. First, the worst-case output guaranteed by the context-guided architecture, achieved through ad hoc fallback logic, still delivered a functional and valid game agent AI. In the case of the deployed context-guided agent, it delivered higher quality output than any combination of the trained learning models. Second, the benefits of learning context-segmentation isolating points of failure were demonstrated when we decided not to use any of the SNSB learning models in the deployed agent. Since all trained SNSB policies severely reduced the agent's collection and completion scores, ad hoc fallback logic was a better alternative. Selective removal of learning would not have been possible had we trained a single monolithic learning model to facilitate the agent's entire AI logic. Third, the process of further, iterative improvement of the deployed agent's behaviour AI can be learning or human intent-driven, making it adaptable to different workflow approaches used in the industry. Commercial game developers might be more willing to experiment with learning, knowing that a potential failure to deliver a high-quality learning policy would not result in a non-functional model. This makes the context-guided agent design potentially applicable in viable scenarios for the game industry, as well as in other contexts.

7.1.4 Outcome

In the course of our research work, we addressed the following research questions, formulated on the basis of our thesis statement and research objectives outlined in Chapter 1: Introduction:

RQ1: can models with execution and performance guarantees of learning logic be integrated into the game industry state of the practice ad hoc behaviour AI architecture for applied use in video game playing AI?

We confirmed it to be possible from theoretical and empirical perspectives by documenting, applying in practice, and then successfully experimentally evaluating an industry-applicable context-guided learning agent design and deployment workflow proposal.

The design proposal was presented in the Context-Guided Agent Design section of Chapter 4: Context-Guided Agents and was based on the researcher's extensive game industry experience, supported by theoretical grounding through literature review conducted in Chapter 3: Background, which covered both academic and industrial publications. This original design for safe integration of data-driven models into BTs was established on the basis of prior research into the reliability of BTs [42], formal proofs of performance and execution guarantees provided by a dedicated learning BT node [234], and game industry requirements. The safety of learning integration was ensured by an original extension of the BT learning node concept, featuring additional safety redundancies to facilitate multiple safety tests and runtime fallback to ad hoc logic. The node extension was detailed in the Safe Learning Integration section in Chapter 4: Context-Guided Agents. The holistic design approach and synthesis of research and industry perspectives enabled us to evaluate the design in a practical setting, specifically within the commercial game environment of *60 Seconds!*, which was instrumented for autonomous agent operations and experimental evaluations involving AI agents and human players. Agents were trained using RL and IL, with imitation training data sourced from the mass-scale gameplay telemetry dataset generated by the real users of the game. In the course of preparing data for use in our research, documented in the Gameplay Telemetry Dataset section in Chapter 4: Context-Guided Agents, we produced a processed trajectory dataset, which constituted another original contribution of this thesis, and will be shared with the research community.

The practical application of the design in the commercial game environment was demonstrated and experimentally evaluated in Chapter 6: Agent Study. It confirmed that trained context-guided agents were able to operate competently in the target game environment, with respect to the gameplay objectives and rules of the game in both seen and unseen environment scenarios. Analysis of the design, training, and evaluation process in the Agent Review section in Chapter 6: Agent Study confirmed the applicability of the proposed design for video game playing AI, supporting the empirical evidence accumulated in the course of work conducted in the Agent

Study.

RQ2: how well can a trained context-guided learning agent perform in unseen game environment scenarios, in comparison to human players, in approximately similar gameplay conditions?

We experimentally evaluated the gameplay performance of trained context-guided learning agents and human players in approximately similar gameplay conditions, in unseen game environment scenarios. The agents were found to play the game competently, achieving non-zero scores in pursuit of game objectives. This translated into *Low* play skill, representative of beginner players. However, the normalised, quantitative comparison of game scores revealed that human players significantly outperformed agents in terms of collection and completion scores. On average, they achieved datasetEHEAcolcomdiffpercent% higher collection and completion scores.

We conducted the experimental evaluation in the instrumented game environment of the commercial game *60 Seconds!*. Context-guided agent instances played the game in the context of the agent simulator environment, whereas human participants took part in the experiment playing in the regular environment of the commercial game, on their own computers. Although the evaluated context-guided agent featured the best-performing learning model trained, the analysis of its operations found it to deliver suboptimal output. Despite this, it was still able to achieve non-trivial results. We expect that further improvements to the learning model training would likely provide additional gains in terms of the agent's performance. Experimental evaluation was detailed in the Agent Evaluation section, and the analysis of its results was documented in the Agent Versus Humans section, both in Chapter 6: Agent Study.

7.1.5 Takeaway

We set out to conduct our research in a format that would connect research and industrial perspectives. It was made possible by the author's background in game development, our unprecedented access to a commercial game environment, and its mass-scale gameplay telemetry dataset for the purposes of our research. This allowed us to incorporate industrial requirements, workflows, and real user data in our investigation. Additional motivation stemmed from the chance to train agents that could be readily applicable in the context of a commercial game environment, in order to support its continued development. This would not be the case with a custom environment, created only for the purposes of our work. Such an opportunity allowed us to face real-world challenges and problems encountered in game development, and to determine how learning aligns with it. We were able to successfully demonstrate that integrating learning models for runtime operations is possible in a commercial game environment. The resulting, trained game-playing agent instance can be incorporated to service automated playtesting scenarios in the game *60 Seconds!*.

The focus on a single environment made the deployed agent's applicability non-universal, but this was offset by the generalisable nature of the proposed design. It has already garnered interest from the industry and was shared with the research and industry AI community as a publication [88]. Furthermore, the safety redundancies proposed for the learning node concept can be reused in alternative design approaches to learning integrations into ad hoc AI representations. We were able to achieve our research objectives while creating added value for the industry due to the industry case study format of this investigation and the potential continuity that our research contributions and outputs afford. At the same time, this generated constraints and resulted in limitations. One of our primary motivations was to keep a human expert at the centre of the design process for AI behaviours, even when learning was involved. This is at odds with the nature of data-driven model training, which does not promote transparency and is not easily incorporated into iterative workflows that are standard in the industry.

In the process of integrating learning into the target game environment, we experienced many similar issues that others had faced before. There is a reason why there are so few games with runtime, data-driven models: it is usually much easier and more economically viable to address development challenges with simpler and more reliable solutions. It is also necessary to plan learning integration early, during game production, in order to architect the game appropriately and establish a forward model for streamlined simulation of the environment. That was not the case with *60 Seconds!*, which resulted in extensive work to circumvent the tight coupling of the game loop, presentation, and input logic, which is not uncommon in game development. Hence, developers have valid reasons to be sceptical of learning. Especially that integrating and servicing learning models requires large amounts of data, long training times, and expert knowledge. Despite the fact that we had data and know-how, the output of our learning models was suboptimal due to problems encountered during their training. Had they been based on monolithic, black box models, we would not have been able to roll the agent out to operate competently enough in the game environment. However, by combining algorithmically simpler BT structures and learning, we produced an operational agent whose underlying logic can now be further iterated upon, due to the inherent transparency of BTs. We found that rooting learning in the established and reliable architecture of BTs reduced the risks involved. This was further supported by the behaviour context segmentation into small-scale learning models. As proven by our issues with the trained learning models, the ad hoc logic fallback embedded in our design allowed the agent to operate competently, making the potentially worst-case quality scenario acceptable. It also supported an iterative approach to developing AI behaviours, which is not the case with monolithic, black-box learning models. And yet, further work towards improving iterative revisioning and training pipeline automation is required to make such solutions viable in the long term, and for different production scales.

Learning is unlikely to become a universal solution in game development. Due to the experience-driven nature of video games, many challenges in game development will potentially always be solved with simpler means. However, for those problems that call for and could substantially benefit from the specific properties of learning, ways to realistically embed learning into games are desired. While we were able to resolve some issues encountered, our case study was just the first step in demonstrating learning compatibility with industrial workflows.

7.2 Highlights

7.2.1 Contributions

In the course of our work, we delivered several research contributions. Each of them was instrumental in furthering the progress of our research and providing additional value, which we intend to share with the research community:

C1: industry applicable context-guided learning agent design and deployment workflow proposal

We formulated and documented a proposal for designing and deploying context-guided learning agents, with respect to the requirements of the game industry and prior research work in the field. The proposed design was later applied in the practical scenario of a commercial game environment. The design proposal is presented in the Design Proposal section in Chapter 4: Context-Guided Agents.

C2: extension of the BT learning node concept, featuring additional safety redundancies

To provide industry-applicable performance and execution guarantees to learning logic embedded within ad hoc AI BT architectures, we proposed an extension of the BT learning node concept by Sprague & Ögren [234], incorporating additional safety redundancies. The contribution is presented in the Safe Learning Integration section in Chapter 4: Context-Guided Agents.

C3: a mass-scale, processed gameplay telemetry dataset from the game *60 Seconds!* to be used in our research, and later shared with the research community

The data-driven perspective of our work was supported by using the mass-scale gameplay telemetry dataset, generated from gameplay trajectories crowdsourced from real players of the game. To be able to use the collected data in our research, we developed a processing pipeline, which transformed raw trajectories into usable, well-structured data samples. Our work with the gameplay telemetry dataset is explored in the Gameplay Telemetry Dataset section of Chapter 4: Context-Guided Agents. Additional documentation of the dataset is provided in Appendix A. A subset of the gameplay telemetry dataset used in our work is part of the digital supplement to the dissertation. The entire processed gameplay telemetry dataset will be shared online with the research community.

C4: game score metric for measuring play skill in terms of gameplay performance in the *scavenge* segment of the game *60 Seconds!*

The game *60 Seconds!* did not originally feature any quantitative metric for measuring play skill exhibited by players during gameplay. To be able to numerically model and analyse data from the gameplay telemetry dataset, we derived a set of formulas, rooted in the game's context and design values, to structure a metric for normalised quantification of play skill. The proposed game score metric is outlined in the Game Score section of Chapter 4: Context-Guided.

C5: analysis of game scores measured for the gameplay telemetry dataset from the *scavenge* segment of the game *60 Seconds!*

In order to be able to extract useful data and apply it in our work, we conducted a study to identify relevant properties of the game sample population dataset. The study was structured with respect to the proposed game score metric and factors derived from the game's context, culminating in the analysis of the play skill observed in the normalised baseline dataset, selectively sampled from the population dataset. The investigation of game scores found in the gameplay telemetry dataset is presented in Chapter 5: Game Score Study.

C6: analysis of the training process of a context-guided learning agent, capable of playing the *scavenge* segment of the game *60 Seconds!*, developed on the basis of the context-guided agent design

Following the design, deployment, and evaluation of the context-guided agent in Chapter 6: Agent Study we conducted an extensive analysis of the process of the agent's training, and its outputs. The study provides a review of the practicalities of deploying a context-guided agent, as well as the procedures, challenges, and opportunities afforded by attempts to integrate learning models into industrial workflows. The study can be found in the Agent Review section in Chapter 6: Agent Study.

7.2.2 Outputs

O1: implementation of a BT learning node, integrating *Unity Machine Learning Agents* learning capacity into PadaOne Games BT library *Behavior Bricks*

To integrate learning into the ad hoc structure of an off-the-shelf BT solution, we had to implement a learning node that was compatible with the *ML-Agents* learning library, *Behavior Bricks* BT library, and our autonomous agent API. Implementation of the learning node is discussed in the BT Authoring section of Chapter 4: Context-Guided Agents. The learning node implementation is a part of the digital supplement to the dissertation and will also be shared online with the research and game development communities.

O2: instrumentation of the *scavenge* segment of the commercial video game 60 Seconds! for enhanced gameplay telemetry data acquisition, simulating autonomous agent operations, and experimental evaluations with human players.

To take advantage of an environment and gameplay telemetry data from a commercial game, we instrumented the game *60 Seconds!* to support online, experimental evaluations with the game's live audience, delivered seamlessly through the familiar interface of the game. Since the game *60 Seconds!* did not implement any form of agent AI, the game's environment was upgraded to support game playing, autonomous agent AI, using off-the-shelf BT and learning libraries. We also augmented the game's data collection pipeline to resolve pre-existing data acquisition issues and deployed additional measures for enhancing the data collection process. The game instrumentation work is documented in the Game Environment Instrumentation section of Chapter 4: Context-Guided Agents, and selected parts of its codebase are included in the digital supplement to the thesis.

O3: standalone simulator software for simulating autonomous agents, based on the *scavenge* segment of the *60 Seconds!* gameplay environment.

We isolated the relevant portion of the game to create a dedicated agent simulator environment and equipped it with a configuration pipeline to enable training and simulating agents for research purposes. The simulator was used to fulfil our research objectives and will later be shared with the academic community. The instrumentation of the game environment for agent training and simulation is discussed in section Agent Training and Simulation section of Chapter 4: Context-Guided Agents. Additional documentation of the simulator can be found in Appendix C. The agent simulator environment is a part of the digital supplement to the dissertation and will also be shared online with the research community.

O4: k-nearest neighbours classifier of play skill in the *scavenge* segment of the game *60 Seconds!*

Following unsupervised play skill-based game score k-means clustering of gameplay trajectory dataset, we created a k-nearest neighbours classifier to identify the scavenge collection play skill exhibited in a given game data sample. The classification of play skill was established in the Play Skill Classification section of Chapter 5: Game Score Study.

O5: a trained context-guided learning agent, capable of playing the *scavnege* segment of the game *60 Seconds!*, developed on the basis of the context-guided agent design, using game industry off-the-shelf solutions

To put assumptions of the proposed design of the context-guided agent model to test, we implemented an instance of the agent in the instrumented game environment of *60 Seconds!*, using the Unity game engine, *ML-Agents* learning library, and the *Behavior Bricks* BT library. The agent's architecture was based on a BT flow, which incorporated small-scale learning models embedded in a learning node structure, as proposed in our design. Learning models were trained using PPO and GAIL, with demonstrations generated from real user gameplay trajectories. The design, deployment and benchmarking of the context-guided agent were presented in the Agent Training section of Chapter 6: Agent Study. The agent was capable of playing the game and pursuing collection and completion gameplay objectives. The deployed context-guided agent, and its variants, can be simulated in the agent simulator environment, included in the digital supplement to the thesis.

O6: a published game AI book series *Game AI Uncovered* chapter presenting the context-guided learning agent design and deployment workflow

The design proposal has already been shared publicly with the research community and the industry in the format of a chapter in a recently published game AI book series *Game AI Uncovered* [88].

7.2.3 Challenges

Data Handling

The opportunity to use a crowdsourced, mass-scale gameplay telemetry dataset in our research provided us with a large quantity of data but also presented multiple challenges in order to make it usable in our work. Some of the challenges encountered during our data work included:

- **Data volume**: processing and exploration of the large volume of data, contained in the mass-scale gameplay telemetry dataset, was time-consuming and computationally intensive. This was the result of the quantity and quality of raw data, as well as the multi-step data transformation process involved.

- **Collection and processing pipeline**: the original data collection pipeline, implemented by the developers of the game, was functional, but limited. Some of its problems were not discovered until we carried out preliminary processing and analysis of the collected data. The trial-and-error procedure for improving the collection pipeline took a substantial amount of time and effort before raw data samples could be considered valid for further processing. Despite improvements introduced to the data collection pipeline, raw gameplay trajectories still required a substantial amount of processing, to be transformed into usable data samples for our research work. The prolonged period of implementation and validation of the collection and processing pipeline delayed our progress towards the development and analysis stages of our work.

- **Data crowdsourcing**: crowdsourcing of gameplay trajectory data samples from the users in their natural play setting and within the known environment of the game, reduced controllable environment and researcher intervention bias, but introduced uncontrollable factors on the user side. We were not able to detect if more than one person played the game on a single account, potentially contaminating individual user trajectory datasets. While some gameplay violation patterns, such as cheating, were observable within the trajectories themselves, it was not possible to detect all issues of this kind. However, we assumed such undetected instances to be minimal in comparison to the volume of the collected dataset, and as such, we did not expect them to skew the results of our work. At the same time, server-side infrastructure failures, client-side hardware problems, and client-side connectivity issues resulted in the loss of an unknown amount of crowdsourced data. We were unable to technically resolve such issues, as they were beyond our control. It was also not possible to accurately estimate the exact number of data samples lost and the impact their absence had on the collected dataset. Thus, we assumed that due to the large volume of the collected data and the focus of our research on approximated model development, any such data gaps would not skew our results.

- **Ethical considerations**: since our data was crowdsourced using a commercial game, available to a worldwide, live audience on a third-party distribution service, ethical considerations were of utmost importance. This included the issue of safe data storage, which was provided by the developers of the game. Collected data confidentiality was ensured through anonymisation of user identifiers. Additionally, a data collection opt-in mechanism had to be implemented in the live version of the game, and research project information had to be presented for experimental evaluations involving real users of the game. This resulted in an increased workload in terms of engineering and testing of the instrumented game environment.

The full discussion of challenges encountered during the data collection stage of our work is presented in the Data Handling Challenges section of Chapter 4: Context-Guided Agents.

Instrumenting the Game Environment

Since the game *60 Seconds!* was not equipped for autonomous agent simulation and servicing learning models, relevant work had to be carried out as part of our research. It required a non-disruptive development and integration approach to ensure the pre-existing functionality of the game environment would remain intact. There were several challenges we faced during this process:

- **Lack of a forward model**: the way the game's control loop was implemented introduced challenges in our work towards augmenting the environment to support autonomous agent operations. The lack of a forward model and the tight coupling of the player input model with gameplay features prevented us from directly mapping agent signals to pre-existing simulation logic. We had to create control abstractions and approximated models of human player navigation and perception to facilitate player-like agent gameplay. Development of these features introduced a significant number of changes to the game's codebase. We had to ensure all these modifications would not disrupt the game's original functionality. Our work also entailed generating additional data assets, such as navigation meshes for each of the levels present in the game. Engineering and testing efforts involved, discussed in the Autonomous Agent Support section of Chapter 4: Context-Guided Agents, were carried out throughout the course of our entire research work.

- **Game engine and environment upgrades**: to be able to integrate and use the *ML-Agents* learning library in the agent simulator environment, we were forced to upgrade the engine used by the game. This caused a number of issues with the underlying implementation of the game environment, which had to be resolved and thoroughly tested. Possibly the most problematic of them was the modification of the engine's physics system, which altered the original game's physics configuration and flow. We discussed this portion of our work in the Game Environment Changes section of Chapter 4: Context-Guided Agents.

- **Game engine limitations**: during the development of the agent simulator environment, we discovered that Unity's physics simulation, used extensively in *60 Seconds!* to facilitate gameplay, could become unstable and produce unreliable outputs. The issue manifested consistently if the execution speed of the simulation was increased by more than a factor of 10.0. This forced us to restrict the maximum permitted execution speed of the agent simulator environment for both inference and training of agents. Since we simulated agent gameplay hundreds of thousands of times during our research, agent inference and training timeframes were significantly affected. The issue persisted in both display and batch mode execution. We detail this problem in the Agent Simulator section of Chapter 4: Context-Guided Agents.

Agent Training

Design and deployment of a context-guided agent instance was a key process in our research, which depended on our prior data and development work, as well as the *ML-Agents* learning library. It was also relatively fragile, with multiple potential points of failure. We had to face the following challenges during agent training:

- **Constrained model revisioning**: our decision to conduct agent training using a semi-automated training pipeline was justified with respect to the assumptions of our research and the potential cost of implementing a fully automated solution. If the learning software used was stable and reliable, a semi-automated pipeline would have likely been sufficient for the purposes of the scenario examined. However, unexpected problems experienced when using *ML-Agents* substantially impacted our capacity for flexible and efficient learning model training. Recurring stability issues and the lack of support for BC resulted in extended training times, which could have been considered prohibitive. For instance, if we were to revise observation space or reward values for any of the trained models, it would have entailed an extended process of encoding all demonstrations, followed by a long learning process. Based on our results, in the worst-case scenario, it would take days for a trained model to be ready for evaluation in the game environment. These factors contributed to a more expensive model revisioning, which prevented us from iterating further on SNSB and SNTI models. We outlined the semi-automated training pipeline assumptions in the Semi-Automated Training Pipeline section and discussed its performance in the Agent Review section, both found in Chapter 6: Agent Study.

- **Unreliable learning software**: since the game environment was implemented using the Unity game engine, we made an intuitive decision to use Unity's *ML-Agents* learning library in our research work. It was compatible with the game environment, included state-of-the-art learning features, and appeared to be well supported. Unfortunately, we found that using it in custom environments, such as ours, resulted in a range of unexpected, critical issues, which affected the quality of agent training. Context-guided agent design assumes multiple learning models may be included in the architecture, but only one model can be trained at a time in *ML-Agents*. It was necessary to structure our training workflow around this constraint. Human-like play styling was expected to be derived from gameplay trajectories using BC and GAIL. However, learning crashed for every attempted configuration of BC, which forced us to exclude it from agent training altogether. Encoding gameplay trajectories into demonstration binary files was a single-threaded process, which required replaying each of the available trajectories in the agent simulator environment. This was necessary, as *ML-Agents* generates demonstrations with a heuristic approach, which typically requires a human to play each learning episode in the Unity editor. We were able to successfully automate the process and encode all available trajectories, but the learning library crashed when demonstration files were too large or there were too many of them. This significantly constrained the number of gameplay trajectories used for training. Since developer assistance for custom environments was limited, we were unable to resolve software issues encountered in the course of our work. While *ML-Agents* is currently at version 0.30.0 and is still in development, the problems we encountered suggest that, in its current state, it cannot be considered a reliable solution to facilitate learning. We documented and discussed the relevance of issues experienced with *ML-Agents* in the Train Learning Models and Agent Review sections of Chapter 6: Agent Study.

7.2.4 Limitations

Although we were able to accomplish our research goals, we identified a number of limitations of our research, which affected our work and its outputs.

Suboptimal Agent Performance

We were hoping that in the course of our work, we would be able to deploy a context-guided learning agent with the capacity to surpass human players in terms of gameplay performance. This was not the case. The trained context-guided agent achieved significantly lower collection and completion scores than human players in the experimental evaluation. It was also not able to outperform the ad hoc reference agent, which employed heuristics and randomisation-based logic. In the analysis of the agent's training in the Agent Review section of Chapter 6: Agent Study we found these shortcomings were the result of the suboptimal output of the trained learning models. As discussed in the Agent Training section, issues with the *ML-Agents* learning library adversely affected agent training and resulted in lower quality and higher training cost of the models produced. Potentially relevant problems were also discovered in the hyperparameter configuration and observation space design of our learning models. However, the fact that the context-guided agent was functional, capable of pursuing gameplay objectives, and achieved non-zero game scores in the game environment confirmed that its design was valid. The problem of learning model quality was not caused by the agent's deployment but rather by factors that could be addressed externally by an alternative learning library and internally by deploying an automated and rapid learning model revisioning infrastructure. A follow-up investigation would be required to pursue this line of inquiry at scale.

Training of Multiple Models

The context-guided agent design assumes that more than one learning model can be incorporated into the agent's BT architecture. The agent developed in the course of our work was equipped with two models. While using multiple, smaller-scale learning models provides context segmentation advantages, it was also found to introduce a degree of redundancy. For each model involved, we had to individually design action and observation spaces, define reward signals, prepare training data, and iterate on its learning configuration. If our small-scale, context-segmented models were quick and cheap to train, this redundancy would have been acceptable, and perhaps even negligible. However, due to problems with the *ML-Agents* learning library, which resulted in longer learning model training times, the cost of handling multiple models became an issue. Additionally, *ML-Agents* only supported training one learning model at a time. This generated additional constraints in the process of training and benchmarking our agent models and the potential automation of the agent deployment workflow. While in the conditions of our investigation, the presence of multiple learning models manifested as an issue, we believe it to be a critical feature of the context-guided design. A follow-up investigation would be required to present a scenario in which this feature is not impacted by technical factors that are not linked to the design.

Play Skill Metric

While we found the proposed play skill metric to be sufficient for the purposes of our investigation, an alternative approach could have potentially served us better. In the course of the game score analysis, we discovered that our metric suffered from bimodality, caused by the binary nature of completion scores. This limited the applicability of the full score in our work and resulted in shifting our focus to individual analysis of collection and completion scores. At the same time, while the mapping of collection scores with respect to *item* weight values was considered quantitatively objective, it was not necessarily an accurate valuation for *items*. If that were the case, it would translate into an unreliable measure of success. An intuitive alternative could be established using a data-driven approach of calculating specific *item* value from data found in user-generated gameplay trajectories based on *item* collection popularity across the entire population dataset, or a selected baseline. It would have been consistent with the quantitative approach of our research. Additional insights about *item* relevance to continued gameplay could have been derived from *survival* trajectories, had we had access to those. However, that was not the case, and since we operated on the assumption that the game's design data was a reliable source of information, we decided to limit the scope of our investigation to a game context-derived metric.

Approximated Play Data

Our research assumptions involved working with approximated representations of reality. This provided us with a degree of freedom in terms of how we conducted our modelling work, but in some cases, it presented the risk of introducing inaccuracies. Combined with a potentially imprecise play skill metric, this could have resulted in unreliable data analysis. Although we accounted for different factors and strived for consistent investigation of the data in Chapter 5: Game Score Study, we did not incorporate random effects in our analysis. We also decided to use a simple clustering approach, targeting a one-dimensional dataset, in order to produce *scavenge* play skill clusters. Had the scope of our investigation been wider, and covered user data patterns in longitudinal studies, our approach would not have sufficed. It also prevented us from observing more nuanced data trends, which could have potentially benefited our research in producing more refined player personas. However, in line with our approximation assumptions, we expected that behavioural data patterns relevant to our work would emerge in the course of using gameplay trajectories as training data for learning models in Chapter 6: Agent Study. Because trained models delivered suboptimal output due to several potential factors, we were unable to objectively evaluate whether the expected behavioural patterns materialised during agents' operations.

Domain Expert Requirement

By integrating learning into BT architecture, we made our design proposal relevant to the state of the practice game industry workflows. However, proper deployment of a context-guided agent requires an expert versed in BT AI design, learning model development, and the game's context. While intended and consistent with the assumption of maximised human authoring control, this constraint could limit the design's applicability in certain circumstances. Even if an expert is available, the context of the desired agent gameplay behaviours may not be fully understood or even known. The design was structured to service exactly such a scenario. Implementing BT fallback logic, before the actual learning logic is deployed, supports continued iterative development of the agent's AI, until the targeted context is fully understood. Even if the expert involved is no longer available, the latest ad hoc version of the logic is expected to deliver acceptable output.

Single Game Context

Context-guided learning agents design, and deployment workflow proposal was intended to be generalisable, replicable and applicable to different game environments. In the course of our research work, we had only deployed and tested it in the game environment of *60 Seconds!*, as targeting other game contexts was considered to be beyond the scope of our investigation. We consider the context-guided agent design to be generalisable and applicable in any game environment, as long as it supports ad hoc-based behaviour architectures and learning. At the same time, the fact that we deployed the agent for a specific game environment allowed it to provide tangible industrial value, as it can now be used by the developers of the game, and provide a demonstration of the effectiveness of the context-guided design. However, since it was not tested in other game environments, its potential to be readily applicable in alternative contexts should be validated in follow-up investigations.

Limited Game Data Access

Due to the fact that developers of the game *60 Seconds!* only instrumented it to support data collection in the *scavenge* segment of the game, it prevented us from taking advantage of data that could have been recorded in the second half of the game: *survival*. Since the developers had no plans to outfit *survival* with data collection, we did not include it in the scope of our investigation. This created a potential gap, as *60 Seconds!* can be considered a holistic experience. The player first scavenges their house for supplies and then attempts to survive with them in their fallout shelter. The starting state of the *survival* segment of the game is directly impacted by the player's performance during *scavenge* gameplay. Had we had access to data about *survival* performance, following specific *scavenge* playthroughs, we would have been able to create more sophisticated, data-driven models of player behaviours and strategies for both segments of the game. At the same time, this would have increased the complexity of our data work, as player retention is easier to achieve in the relatively short, 60-second-long *scavenge* gameplay. For *survival*, which can last up to an hour, data collection of full trajectories would introduce additional sourcing, storage, and processing issues. Even partial *survival* trajectory data, indicating whether a player was able to succeed or perform better with a specific set of *items* could provide valuable insights towards modelling what constitutes better gameplay performance in *scavenge*, as well as valuation of specific *items*.

Quantitative Focus

The data-driven perspective of our research and the remote crowdsourcing of data from users informed the decision to forego qualitative data sourcing and analysis in our work. We believe this choice was valid in the conditions of our investigation, as it prevented qualitative data collection from adversely affecting the quantitative data collection process. Quantitative gameplay telemetry data was recorded in the background, with no visible change or disruption to the familiar flow of the game. Qualitative data collection would have had to be overt, making it potentially distracting to the players of the game. However, extending the scope of our work with qualitative data could have provided additional insights about player behaviours, which cannot necessarily be inferred from gameplay telemetry. Incorporating qualitative responses from game industry experts on the perceived quality of the proposed context-guided agent design could also provide an outside, non-functional perspective on the advantages and disadvantages of the proposed design.

Post-Launch Data Paradox

In our research work, we used a mass-scale gameplay telemetry dataset, which was collected after the launch of the game, and would not typically be available to developers at the time of the game's production. Even if extensive playtesting sessions were organised before the game's launch, it would be unlikely for them to involve hundreds of thousands of users and generate millions of data samples. This informed our motivation to reduce data requirements for agent model training using context segmentation. However, this does not solve the underlying issues of limited data availability before the game's launch. To address the issue, gameplay telemetry could be sourced from a limited group of players and used to train simple agent models. These trained agents would then generate more trajectories by playing the game. However, such an approach creates its own set of issues that were beyond the scope of our investigation.

7.3 Future Work

Expanded Player Modelling

The scope of our research work was limited to modelling a single, top-play skill persona. While this was sufficient for the purposes of our research, the gameplay telemetry dataset used in our work contains enough data to expand persona and user modelling to a full spectrum of user play skill models and potentially play style models. Generating context-guided agent behaviour AI for each of these models would allow us to investigate the potential of the design to scale up, in the context of the same game environment. Additional information about user decision-making and behaviours could also be acquired by employing qualitative methods to complement the available gameplay telemetry. This would allow for an increase in the dimensionality of modelling behaviour templates. Deploying more sophisticated modelling techniques, such as mixed effect models, to control for randomisation found in the gameplay of *60 Seconds!* could support a more accurate representation of player behaviours. It would also alleviate some of the limitations of our research, identified earlier. An improvement to the clustering methods used would also provide a benefit over the techniques used in our work. We settled on clustering the game score with respect to one dimension. By investigating the multi-dimensional representation of the population dataset, we would potentially arrive at more in-depth observations about the players and their gameplay. This could directly benefit the developers of the game, as they would be able to produce agent models trained on player personas incorporating different aspects of behavioural modelling. Such an approach could result in artificial players who operate not only with respect to varied play skills, but also different operational strategies, persistence, drive to exploration, and others.

Play Skill Estimation

Further improvements to data modelling could be achieved by exploring alternative approaches to play skill measuring, discussed earlier. Attempts at data-driven collection score calculations could be made with the already available data and infrastructure. If an opportunity to incorporate at least partial data from *survival* gameplay trajectories would emerge, it could expand the scope of player modelling for *60 Seconds!* that would be useful for its developers. Exploring and comparing such a different method against the used score metric could provide a scenario for structured improvement of the adopted modelling strategy and potentially arriving at player models that would better align with real player gameplay behaviours and performance.

Improved Learning Model Training Pipeline

Further development of the learning model training pipeline used in our research work is required to improve its reliability and effectiveness. Foremost, alternative learning libraries should be considered to replace *ML-Agents*. A preferred replacement would be game engine agnostic and not require the game environment to be instanced for training. Instead, it should allow the creation of abstract representations of problem spaces for the lightweight training of individual, small-scale models. The model training and deployment process would also benefit from further automation to enable hands-off revisioning of the learning configuration and its evolution using genetic algorithms. Infrastructural improvements of the learning model workflow would also require localised streamlining of the model revisioning to address issues such as those encountered in our work. The ability for rapid, iterative revisioning of the trained model is essential to decrease the risks associated with training costs and uncertainty. This also applies to revisioning of observation and action spaces, as well as balancing reward functions. Otherwise, constraints on the speed and volume of models trained will continue to limit attempts at applied learning.

Applied Use of Trained Agents

Ultimately, trained agent models are expected to be put to use by the developers of the game: either as bots to play their game in a specific setup, or as AI components that would assist in further development of the game. The former is more likely for multiplayer titles or single-player games involving symmetric opponents. For *60 Seconds!*, likely use scenarios would involve automated playtesting of new game levels by agents trained with different player personas. They need not be limited to play skill personas, but with expanded player modelling, could feature personas trained on multidimensional behavioural data. This could be achieved by either producing a suite of behaviourally varied agents or a single, adaptive agent featuring dynamically swapping behavioural models. Such agents could be immediately deployed to playtest any new game environment scenarios, authored manually or generated procedurally. With additional infrastructural work, these agents could be evolved to provide AI-driven design assistance in generating entirely new scenarios for the game. Models established for these agents could also be extracted to serve as live estimators for the performance or behaviours of human players of the game. This could ultimately lead to developing systems for dynamic balancing of the player experience via model-informed manipulation of the game's environment, such as changing its layout or contents.

Alternative Video Game Environments

Validating the context-guided design in alternative game environment contexts is a desired next step if the proposed design is to be considered useful for industrial applications. Further game industry case studies, ideally investigating games with different gameplay than that of *60 Seconds!*, could provide useful insight into a wider applicability of context-guided learning agent design. However, transfer learning scenarios could also be explored to determine if once trained agent models might be of use in other games. A prime candidate for such a follow-up is Robot Gentleman's second game, *60 Parsecs!* (Robot Gentleman, 2018), which has approximately similar gameplay objectives and environment to *60 Seconds!* but features different content and presentation.

Industrial Applicability

Experiments with the deployment of the design in alternative game industry contexts should ultimately lead to applied use in commercially released video games. Even with enough interest for this to happen, additional development work would likely be required in order to enhance the automation of the training and deployment workflow and align it with pre-existing pipelines and the interests of game developers. However, the direct applicability of the design and the already manifested interest of the industry are promising for future applications of the design.

Applications of the Gameplay Telemetry Dataset

Data from the gameplay telemetry dataset used for our research was directly applicable to the context of the game environment it was sourced from. However, many other applications of that data are possible. By sharing the dataset and its documentation with the research community we are hoping to see other researchers use it for both video game related investigations, as well as other projects.

7.4 Closing Statement

Cautious business perspective of the game industry decision makers, and the scepticism of game AI developers towards the feasibility of learning in game environments, has been delaying a wider pursuit of learning agent AI in video games. After what we experienced first-hand with the current state of learning support in the mainstream, game industry solutions, we cannot say we do not agree with their reservations. There is still a long way to go before learning agents become commonplace in games, but at a time when AI research and development are gaining traction on an unprecedented scale, it is a direction worth exploring. Video games have the potential to play an important role in the continued development of AI. Their varied and complex virtual worlds are an excellent proving ground for both entertainment and real-world applications of learning agents. With enough convincing case studies of learning model applicability and non-disruptive integration into industry workflow pipelines, game developers will be able to creatively use and push the boundaries of learning. Our context-guided learning agent design proposal, and its successful application in a commercial game environment is one such case study. We are hoping that more will follow.

Appendix A

Data

Summary. The Data appendix provides supplementary information about the data used in our research work.

A.1 Dataset and Digital Supplement Access

Up-to-date information about accessing the processed gameplay telemetry dataset used in our research work, the digital supplement to the thesis, and any additional relevant resources will be provided at the following online sources:

- https://gotojuch.com/contextguidedagents/
- https://github.com/viadomx/ContextGuidedAgents

The shared dataset contains processed gameplay telemetry data files, organised in directories indexed by individual user identifiers. All files in a user directory are trajectories generated by that particular user. Additional instructions on how to unpack and access the dataset may be provided.

A.2 Design Data

To support the data-driven processing in our research work, design values from *60 Seconds!* were transcribed to dedicated JSON files. It allowed us to load these files during processing and access specific values of interest. As a part of the processing pipeline, we also used design values to simplify data representation by referencing indexed design objects, stored in the design data JSON files. Design data files include:

- **Difficulty data**: identification and basic setup of each difficulty level that can be selected for a specific game.

- **Game type data**: identification of all game types that can be played and influence the conditions of a specific game.
- **Item data**: details of each *item* type available to be collected in the game.
- **Environment data**: layout details for all environments found in the game.

Design data files can be found at: */data/design*. Programmatic access to design data values is possible from Python code (*/source/processing/game_design_data.py*), as long as design data functionality dependencies are imported, and data files are accessible.

A.3 Raw Data Sample Structure

Raw gameplay telemetry data samples are stored in textual files, which contain a sequence of game session values separated by semicolons. Values can be numbers (integer or floating point), booleans (recorded as textual strings "True" or "False"), textual strings, 3D position vectors (recorded as three floating point values, separated with commas, enclosed in brackets), and four-dimensional vectors to represent quaternion-based rotation (recorded as four floating point values, separated with commas, enclosed in brackets). The structure of a raw gameplay telemetry data file is enforced by data sections, identified with textual tags, which are followed by a specific type of data. Tags used in raw data files include:

- **No tag**: game session information, always found at the start of the data file. Format: file version identifier (string, values in range "0001"-"0004"), recording date (string, YYYYMMDD format), user identifier (string), game level identifier (string), game was finished flag (boolean, true for properly finished game, false for game that was prematurely aborted), game session run time (float), fixed layout (integer, 0 for not fixed, 1 for fixed), exploration time (integer), scavenge time (integer), game session end time (float), exit position (3D vector), recording end time (float, only recorded for human controlled game sessions).
- **ROOMS**: spawned room archetypes, legacy data from when *60 Seconds!* supported randomised level layout. Format: number of rooms (integer), recording timestamp (float), list of rooms. Each room is represented by: position (3D vector), rotation (quaternion), room variant identifier (string, for example: "AShelter2").
- **ITEMS**: *items* spawned in the game level. Format: recording timestamp (float), number of *items* (integer), list of navigation steps. Each *item* is represented by: position (3D vector), rotation (quaternion), *item* type identifier (string, for example: "suitcase").
- **MOVEMENT**: navigation steps of the avatar in the environment. Format: number of steps (integer), start of recording timestamp (float), list of navigation steps. Each nav-

igation step is represented by: timestamp (float), avatar's position (3D vector), avatar's rotation (quaternion).

- **COLLECTED**: *items* collected during the game session. Format: number of *items* (integer), recording timestamp (float), list of *items*. Each collected *item* is represented by: collection timestamp (float), *item's* position (3D vector), *item's* rotation (quaternion), *item* type identifier (string, for example: "suitcase").
- **DROPS**: deposits made by the avatar. Format: number of deposits (integer), list of deposit timestamps (float).
- **COLLISIONS** : collisions with environment objects caused by the avatar during the game session. Please note that collisions are only recorded for human-controlled game sessions. Format: number of collisions (integer), list of collisions. Each collision is represented by: collision timestamp (float), position of the collided object (3D vector), rotation of the collided object (quaternion), identifier of the collision object (string, for example: "chair").

While the raw gameplay telemetry dataset is not shared, new raw trajectories may be generated with the use of the agent simulator. Processing raw trajectories is facilitated with the */source/processing/game_processing_.py* script.

A.4 Processed Data Sample Structure

Processed gameplay telemetry data samples are stored in JSON files, which contain normalised and structured game session data, as well as additional inference and scoring data. Some values are compressed during the processing of raw data samples to decrease file size, without losing relevant data. For example, 3D vectors are compressed into 2D vectors. The structure of a processed data sample is:

- **game**: game session information.
- **items**: *items* spawned in the game level.
- **collected**: *items* collected during the game session.
- **deposits**: deposits made by the avatar.
- **nav**: navigation steps of the avatar in the environment.
- **collisions**: collisions with environment objects caused by the avatar during the game session.

Game session information contains both the general game session data, as well as inferred and calculated values. They include:

- **game_id**: game identifier(string).
- **user_id**: user identifier (string).
- **date**: recording date (string, YYYY-MM-DD format).
- **time**: recording time (string, HHMMSS).
- **game_time**: actual execution time of the game session (float).
- **level_id**: level identifier(integer), referencing the relevant level design data.
- **finished**: was game finished flag (integer, 1 for properly finished game, 0 for game that was prematurely aborted).
- **won**: was the game session completed successfully (boolean).
- **paused**: was the game paused (boolean).
- **game_type_id**: game type identifier(integer), referencing the relevant game type design data (*/data/design/game_type_data.json*).
- **prep_time**: exploration stage time (integer).
- **run_time**: collection stage time (integer).
- **total_time**: game session total time, including exploration and collection times (integer).
- **elapsed_time**: actual game session total time, which includes startup and conclusion margins (float).
- **start_position**: the starting position of the avatar (2D vector).
- **exit_position**: the position of the exit (2D vector).
- **last_second_deposit**: was an automatic deposit made at the conclusion of a successful game session because the avatar's inventory was not empty(boolean).
- **difficulty**: identifier of the game session's difficulty level (integer), referencing the relevant difficulty design data (*/data/design/difficulty_data.json*).
- **default_character**: was the default character (*Ted*) used in the game session (boolean).
- **extended_item_set**: was the extended *item* set used in the game session (boolean).
- **items_count**: number of *items* collected during the entire game session (integer).
- **items_count_early_game**: number of *items* collected during early game (integer).

- **items_count_late_game**: number of *items* collected during late game (integer).
- **items_value**: total weight of *items* collected during the game session (integer).
- **items_value_early_game**: total weight of *items* collected during early game (integer).
- **items_value_late_game**: total weight of *items* collected during late game (integer).
- **items_distribution**: for each real time second of the collection stage of the game session, a flag denoting if a collection interaction was performed at a given time, or not (60 character string, each character is either 0 or 1, with the former denoting no collection, and the latter meaning that a collection was made).
- **items_order**: *items* collected during the game session, listed in the order of collection (string, each character represents the type of *item* collected, value range: "A"-"W")
- **items_collection_ratio**: ratio of collected *items* count to total number of spawned *items* (float).
- **items_spawned_count**: number of *items* spawned for the game session (integer).
- **items_spawned_value**: total weight of *items* spawned in the game session (float).
- **deposits_count**: total number of deposits during the game session (integer).
- **deposits_early_game**: number of deposits during early game (integer).
- **deposits_late_game**: number of deposits during late game (integer).
- **deposits_avg_load**: average deposit weight in the game session (float).
- **deposits_distribution**: for each real time second of the collection stage of the game session, a flag denoting if a deposit interaction was performed at a given time, or not (60 character string, each character is either 0 or 1, with the former denoting no deposit, and the latter meaning that a deposit was made).
- **interactions_count**: interaction count during the entire game session (integer).
- **interactions_early_game**: interaction count during early game (integer).
- **interactions_late_game**: interaction count during late game (integer).
- **collisions_count_game**: number of collisions during late game (integer).
- **collisions_early_game**: number of collisions during early game (integer).
- **collisions_late_game**: number of collisions during late game (integer).

- **collisions_early_game_ratio**: ratio of collision count during early game to total collision count (float).
- **collisions_late_game_ratio**: ratio of collision count during late game to total collision count (float).
- **move_time_prep**: move time during the exploration stage (float).
- **move_time_game**: move time during the entire game (float).
- **move_time_early_game**: move time during early game (float).
- **move_time_late_game**: move time during late game (float).
- **move_time_ratio_prep**: ratio of movement time during exploration stage to total exploration time (float).
- **move_time_ratio_early_game**: ratio of movement time during early game to total movement time during the game (float).
- **move_time_ratio_late_game**: ratio of movement time during late game to total movement time during the game (float).
- **move_time_ratio_game**: ratio of time spent on moving during the game to total game time (float).
- **pause_game_time**: length of game being paused (float).
- **pause_game_ratio**: ratio of time the game was paused to total game time (float).
- **interaction_game_time**: time spent on interactions during the game (float).
- **interaction_game_ratio**: ratio of time spent on interactions during the game to total game time (float).
- **move_distance_prep**: move distance during the exploration stage (float).
- **move_distance_game**: move distance during the collection stage (float).
- **move_distance_early_game**: move distance during early game (float).
- **move_distance_late_game**: move distance during late game (float).
- **move_distance_ratio_early_game**: ratio of movement distance during early game to total movement distance (float).
- **move_distance_ratio_late_game**: ratio of movement distance during late game to total movement distance (float).

- **traversal_prep**: sequence of rooms visited by the during the exploration stage (string, each character denotes the type of room, visited, value range: "A"-"F").
- **traversal_game**: sequence of rooms visited by the player during the collection stage (string, each character denotes the type of room visited, value range: "A"-"F").
- **traversal_game_distribution**: for each real time second of the collection stage of the game session, the room in which the avatar was at the time (60 character string, each character denotes the type of room, in which the was at a given second, value range: "A"-"F").
- **traversal_exploration_ratio**: ratio of movement time during the exploration stage to total exploration time (float).
- **evaluation_completion**: completion score achieved in the game session (float).
- **evaluation_collection**: collection score achieved in the game session (float).
- **evaluation**: full score achieved in the game session (float).

Items data contains a list of *items* spawned in the game level. Each entry contains the following data:

- **local_index**: index based on the entry's position in the list (integer).
- **item_id**: *item* identifier(integer), referencing the relevant *item* design data.
- **position**: position of the *item* (2D vector).
- **rotation**: rotation of the *item* (quaternion).

Collected data contains a list of *items* collected during a game session. Each entry contains the following data:

- **local_index**: index based on the order of collection (integer).
- **item_id**: *item* identifier(integer), referencing the relevant *item* design data.
- **position**: position of the collected *item* (2D vector).

Deposits data contains a list of deposits made during a game session. Each entry contains the following data:

- **local_index**: index based on the ordering of deposits (integer).
- **timestamp**: deposit interaction timestamp (float).

Nav data contains a list of an avatar's navigation steps during a game session. Each entry contains the following data:

- **local_index**: index based on the order of collection (integer).
- **timestamp**: navigation step timestamp (float).
- **position**: position of the avatar during the navigation step (2D vector).
- **rotation**: rotation of the avatar during the navigation step (quaternion).

Collisions data contains a list of collisions caused by the avatar during a game session. Each entry contains the following data:

- **local_index**: index based on the order of collisions (integer).
- **timestamp**: collision timestamp (float).
- **position**: position of the collided object (2D vector).
- **rotation**: rotation of the collided object (quaternion).

A.5 Sampled and Generated Datasets

In the course of our work in Chapter 5: Game Study and Chapter 6: Agent Study we sampled the processed gameplay telemetry dataset to produce context relevant game score subsets. Additional gameplay telemetry data samples generated during benchmarking and experimental evaluation were also sampled. All game score datasets produced for our analysis work were stored as JSON files and are part of the digital supplement to the thesis. They can be found at: */data/datasets*. The structure of game score datasets is uniform and is based on single or multi-dimensional subsets of data, which hold consistently ordered sample values. Each of the datasets listed contains the following data columns, unless stated otherwise:

- "evaluation": full game score value for each sampled gameplay trajectory.
- "evaluation_collection": collection game score value for each sampled gameplay trajectory.
- "evaluation_completion": completion game score value for each sampled gameplay trajectory.

Stored game score datasets include:

- **Dataset POP**: relevant population dataset used in our analysis.

- **Dataset BAS**: baseline dataset, normalised with respect to game setup parameters. Surplus data columns for each sampled gameplay trajectory: "game_type_id", "extended_item_set", "default_character", "level_id" and "difficulty".

- **Dataset TOP**: top-skill persona dataset. Surplus data columns for each sampled gameplay trajectory: "game_type_id", "extended_item_set", "default_character", "level_id" and "difficulty".

- **Dataset REF**: reference agent model benchmarking dataset.

- **Dataset CA0**: context-guided agent model iteration A000 benchmarking dataset.

- **Dataset CA1**: context-guided agent model iteration A100 benchmarking dataset.

- **Dataset CA2**: context-guided agent model iteration A200 benchmarking dataset.

- **Dataset CAB0**: context-guided agent model SNSB A000 and SNTI fallback benchmarking dataset.

- **Dataset CAB1**: context-guided agent model SNSB A100 and SNTI fallback benchmarking dataset.

- **Dataset CAB2**: context-guided agent model SNSB A200 and SNTI fallback benchmarking dataset.

- **Dataset CAI0**: context-guided agent model SNSB fallback and SNTI A000 benchmarking dataset.

- **Dataset CAI1**: context-guided agent model SNSB fallback and SNTI A100 benchmarking dataset.

- **Dataset CAI2**: context-guided agent model SNSB fallback and SNTI A200 benchmarking dataset.

- **Dataset EHA**: human player scores from all online evaluation samples recorded.

- **Datasets EHS1 - EHS5**: human player scores achieved with respect to the sequence playthrough order in the online evaluation. First playthroughs were placed in dataset EHS1, second playthroughs in EHS2, and so on.

- **Datasets EHL1-EHL5**: player scores achieved in a specific level environment of the online evaluation. Scores from the first-level environment were placed in dataset EHL1, second in EHL2, and so on.

- **Dataset EH**: player scores from all online evaluation playthroughs, but with the first calibration playthrough excluded for each participant.

- **Dataset EAA**: agent scores from all offline evaluation samples recorded.
- **Datasets EAS1 - EAS5**: agent scores achieved with respect to the sequence playthrough order in the offline evaluation. First playthroughs were placed in dataset EAS1, second playthroughs in EAS2, and so on.
- **Datasets EAL1-EAL5**: agent scores achieved in specific level environments of the offline evaluation. Scores from the first-level environment were placed in dataset EAL1, second in EAL2, and so on.
- **Dataset EA**: agent scores from all offline evaluation playthroughs, but with the first calibration playthrough excluded for each virtual user.

Additional datasets can be extracted from the processed gameplay telemetry dataset using the script located at */source/processing/game_data_retrieval.py*. Examples of using the script to source datasets */data/datasets/sourced/game_type.py* and */data/datasets/sourced/game_full_scav_normalised.py* are provided below.

Game Type Dataset Retrieval

```
python game_data_retrieval.py True True DesignPath ProcessedDataPath TargetPath True 3 evaluation evaluation_collection
evaluation_completion True game_type 2 paused 0 finished 1 # game_type_full_won game_type_full 2 game_type_id 1 won 1 #
game_type_full_lost game_type_full 2 game_type_id 1 won 0 # game_type_scavenge_won game_type_scavenge 2 game_type_id 2
won 1 # game_type_scavenge_lost game_type_scavenge 2 game_type_id 2 won 0 # game_type_scavenge_challenge_won
game_type_scavenge_challenge 2 game_type_id 4 won 1 # game_type_scavenge_challenge_lost game_type_scavenge_challenge 2
game_type_id 4 won 0
```

Game Full Scav Normalised Retrieval

```
python game_data_retrieval.py True True DesignPath ProcessedDataPath TargetPath True 8 evaluation evaluation_collection
evaluation_completion game_type_id extended_item_set default_character level_id difficulty True game_full_scav_normalised
5 game_type_id 1 game_type_id 2 paused 0 finished 1 move_distance_game GREATER 0 # game_full_scav_normalised_won
game_full_scav_normalised 1 won 1 # game_full_scav_normalised_lost game_full_scav_normalised 1 won 0
```

Appendix B

Evaluation

Summary. The Evaluation appendix features additional documentation about the experimental evaluation conducted in the game environment of *60 Seconds!*.

B.1 *60 Seconds!* EULA

60 Seconds!
Copyright (c) 2015 Robot Gentleman

*** END USER LICENSE AGREEMENT ***

IMPORTANT: PLEASE READ THIS LICENSE CAREFULLY BEFORE INSTALLING AND USING THIS SOFTWARE.

1. LICENSE

By receiving, installing or using 60 Seconds! ("Software") produced by Robot Gentleman sp. z o.o. ("Developer"), from a digital or physical source, you agree to this End User User License Agreement ("Agreement") and confirm that it is a legally binding and valid contract for you. You agree to abide by the intellectual property laws and all the terms and conditions of this Agreement.

Unless provided with a different license agreement signed by the Developer, your use of the Software signifies the acceptance of this license agreement and warranty.

Subject to the terms of this Agreement, the Developer grants you a limited, non-exclusive, non-transferable license, without right to sub-license, to use the Software in accordance with this Agreement and any other written agreement with the Developer.

If you disagree with any of the sections of this Agreement, please do not install the Software on your computer.

2. DISTRIBUTION

The Software and the license provided must not be copied, shared, distributed, re-sold, offered for re-sale, transferred or sub-licensed in whole or in part, save for a single copy that may be kept by you for archiving or backup purposes. For information about redistribution of the Software please contact the Developer.

3. USER AGREEMENT

3.1 Usage Restrictions

You shall use the Software in compliance with all applicable laws and not for any unlawful purpose. Displaying, distributing and otherwise using Software in combination with material that is pornographic, racist, vulgar, obscene, defamatory, libellous, abusive, promoting hatred or discrimination or displaying prejudice based on religion, ethnic heritage, race, sexual orientation or age is strictly prohibited.

3.2 Copyright Restriction

The Developer retains sole and exclusive ownership of all rights, title and interest in and to the Software, as well as Intellectual Property rights associated with the Software and this Agreement. Copyright law and international copyright treaty provisions protect the Software both as a whole, as well as its individual elements and products or services that it may provide. All rights not expressly granted in this Agreement are considered to be reserved for the Developer.

3.3 Limitation of Responsibility

You will indemnify and if need be defend the Developer, its employees, agents and distributors against any and all claims, proceedings, demands and costs resulting from or in any way connected with your use of the Software.

In no event or circumstances (including, without limitation, negligence) will the Developer, its employees, agents or distributors be liable for any consequential, incidental, indirect, special or punitive damages whatsoever (including, without limitation, damages for loss of profits, loss of use, business interruption, loss of information or data, or pecuniary loss), in connection with or resulting out of or related to this Agreement, use of the Software or inability to use the Software or any other circumstances whether based upon contract, tort or any other theory including negligence.

The Developer's full liability, without exception, is limited to the customers' reimbursement of the purchase price of the Software, proven by a sales confirmation or receipt, in an event of a justified refund request, when the Software was purchased directly from the Developer and not a distributor, third party service or a different vendor. If the refund is granted, you are obliged to return the refunded copy of the Software and all accompanying materials, products and other elements to the Developer.

3.4. Data collection

The Software may collect anonymous, numerical data for analytics, research and other applications and transfer them back to the Developer. By accepting this Agreement, you allow the Developer to use the Software for data collection purposes.

3.5 Warranties

Except if expressly stated in writing and signed by the Developer, the Developer makes no promises or warranties in respect to this Software and expressly excludes all other warranties, expressed or implied, oral or written, including, without limitation, any implied warranties of merchantable quality or fitness for a particular purpose.

3.6 Governing Law

This Agreement shall be governed by the law of Poland. You hereby irrevocably attorn and submit to the non-exclusive jurisdiction of the courts of Poland therefrom. If any provision shall be considered unlawful, void or otherwise unenforceable, then that provision shall be deemed severable from this License and not affect the validity and enforceability of any other provisions.

3.7 Termination

Failure to comply with the terms and conditions of this Agreement will result in automatic and immediate termination of this license. Upon termination of this license for whatever reason, you agree to immediately cease your use of the Software and destroy all copies of the Software or send them back to the Developer, under this Agreement. Any financial obligations incurred by you in obtaining or using the Software shall not be reverted or returned with the expiration or termination of this license.

4. DISCLAIMER OF WARRANTY

THIS SOFTWARE AND THE ACCOMPANYING FILES ARE SOLD "AS IS" AND WITHOUT WARRANTIES AS TO PERFORMANCE OR MERCHANTABILITY OR ANY OTHER WARRANTIES WHETHER EXPRESSED OR IMPLIED. THIS DISCLAIMER ALSO CONCERNS ALL FILES GENERATED AND EDITED BY THE Software.

B.2 Online Evaluation Information

Rocket Science! research information and consent

This challenge is developed as a part of Dominik Gotojuch's self-funded research at the School of Computing Science, University of Glasgow, and is supervised by Professor Roderick Murray-Smith and Doctor John Williamson.

This challenge will feature a series of trials, each taking place on a custom level. On each of these levels you will be tasked with collecting items in the same way and with the same control scheme as found in the standard gameplay of "60 Seconds!". However, the layout and structure of challenge levels will differ from the standard level design in "60 Seconds!"

The aim of this research is to learn how to design better game levels based on data generated by players going through this experimental challenge. This will let us predict player performance for different level layouts and adjust these layouts appropriately.

The only information collected in this challenge is the player's navigation, timing and interaction data. Your anonymity will be preserved at all times. Acquired data will be processed and analysed with a focus on behavioural patterns, performance measuring and establishing player experience profiles.

Trials for this research will be conducted until the project is concluded in 2022. Each trial will be explained to you, using the standard interface of "60 Seconds!" before the actual gameplay begins. You will be rewarded for your participation, upon completion of individual research trials, with in-game digital content. These rewards will only be exclusively available to those who choose to participate in research trials.

Taking part in this research is entirely voluntary and you may withdraw at any time. Withdrawal will be possible from the Other tab in the Settings menu. Please note you have to be 16 or older to participate. For questions or comments please contact the researcher Dominik Gotojuch (d.gotojuch.1research.gla.ac.uk) or the project supervisor Professor Roderick Murray-Smith (Roderick.Murray-Smithglasgow.ac.uk).

I consent to take part in this research
I am 16 or older

Complete a chain of challenge levels. Scavenge as many supplies as possible and get to the shelter in time in each one!

Appendix C

Software

Summary. The Software appendix explains the setup, requirements and operations of the software developed for our research work.

C.1 Setup

The conditions of our processing work can be recreated on a Windows-based machine by installing and configuring the following:

- Anaconda - install the application version 2023.09 or higher [1] and import Anaconda configuration files located at */software/anaconda*.
- Miktex - install the application version 24.1 [2].

C.2 Agent Simulator Environment

C.2.1 Running

The agent simulator environment is a standalone, Windows-based application made in Unity 2023.2.20f1 game engine, based on the game environment of the game *60 Seconds!*. The simulator was written in C#, and features new code, as well as the extended codebase and assets from *60 Seconds!*. Shared codebase is located at */source/simulator*. The binary version of the agent simulator is available in the digital supplement to the dissertation at: */software/simulator*. The simulator requires the following JSON configuration files to run: simulator settings, procedure settings, and agent settings. By default, the simulator will attempt to load all three files from the */software/simulator/Context Agent Simulator_Data/StreamingAssets/simulator* subdirectory. If any of these files are missing, the simulator will use default values for the simulation

[1] Available at: https://anaconda.org/anaconda/anaconda/files?version=2023.09
[2] Available at: https://miktex.org/download

configuration. If any of these files are invalid, the simulator will either fail to start or start and display relevant errors in the game environment's development console. Development console messages are only available when the simulator is running in display mode. To run the simulator in headless batch mode, the *-batchmode* argument must be used with the executable. Each run of the agent simulator generates a log file, which is stored in the */software/simulator/Context Agent Simulator_Data/StreamingAssets/simulator/logs* subdirectory.

C.2.2 Simulator Settings

Simulator settings include:

- **ProcedureSettingsFilepath**: configuration data file containing research procedure setup to be used.
- **AgentSettingsFilepath**: configuration data file containing the agent parameters to be used.
- **LevelsDataFilepath**: data file containing game design-based item definitions.
- **ItemsDataFilepath**: data file containing game design-based level definitions.
- **GameTypeDataFilepath**: data file containing game design-based game type definitions.
- **DifficultyDataFilepath**: data file containing game design-based game difficulty definitions.

C.2.3 Procedure Settings

Procedure settings include:

- **SimulationMode**: simulation processing mode. Options include USER (regular, human-controlled gameplay), TRAINING (learning model training), INFERENCE (BT or context-guided agent inference), TRAJECTORY_REPLAY (replaying of processed gameplay trajectories)
- **UserId**: custom user id to be stored in generated trajectories.
- **SeriesCount**: number of simulation series to be run in the course of a single simulation run. If set to 0, the simulation will continue forever until it is terminated.
- **SeriesStartIndex**: series starting index. Must be smaller than SeriesCount.
- **Levels**: list of names of levels to be played in each series. The number of levels is equivalent to the number of sessions in a series. Available levels include level_scavenge_1 - level_scavenge_20, level_space1_1 - level_space1_5.

- **Difficulty**: game difficulty selection for games in the simulation run. Values include EASY, NORMAL, and HARD.
- **GenerateDemos**: flag to allow generation of Unity demonstration files used for IL. Can only be used in TRAINING simulation mode. Available values: True, False.
- **DemoPath**: absolute path to the directory where newly generated demonstration files will be stored.
- **GenerateTrajectories**: flag to allow generation of raw gameplay trajectories from each simulation session. Available values: True, False.
- **TrajectoriesPath**: absolute path to the directory where newly generated raw trajectory files will be stored.

C.2.4 Agent Settings

Agent configuration is only relevant if the simulation mode's value is not USER. Agent configuration includes:

- **AgentType**: type of the simulated agent. Available agent types include TRAJECTORY (trajectory replaying agent), BT (BT-driven agent), and CONTEXT (context-guided architecture agent).
- **AgentVariant**: agent model variant used for simulation. Available agent variants include SNSB_A000_SNTI_A000, SNSB_A100_SNTI_A100, SNSB_A200_SNTI_A200.
- **Character**: character selection for games in the simulation run. Available values: TED, DOLORES.
- **RunSpeed**: execution speed of the simulation. Permitted value range: [0.001, 10.0].
- **GameGUIActive**: flag to display or hide in-game GUI in display mode. Available values: True, False.
- **AnimateNavigation**: flag to animate the avatar's navigation in display mode. Available values: True, False.
- **ActiveLearningBehaviours**: list of activated learning behaviours. Applicable to CONTEXT agents. If a learning behaviour is not listed, it will be inactive and fallback ad hoc logic will be used in its place. Available values: SelectNextTargetItem, SelectNextScavengeBehaviour.
- **SNTIRewardOnItemCollected**: SNTI reward signal value. Available values: float.

- **SNTIRewardOnEndSessionCollectionCount**: SNTI reward signal value. Available values: float.
- **SNTIRewardOnEndSessionNoCollection**: SNTI reward signal value. Available values: float.
- **SNTIRewardOnSelectedItemWillFitInInventory**: SNTI reward signal value. Available values: float.
- **SNTIRewardOnSelectedItemWillNotFitInInventory**: SNTI reward signal value. Available values: float.
- **SNTIRewardOnSelectedItemIsFirstOfItsType**: SNTI reward signal value. Available values: float.
- **SNTIRewardOnSuccesfulTargetSelection**: SNTI reward signal value. Available values: float.
- **SNTIRewardOnFailedTargetSelection**: SNTI reward signal value. Available values: float.
- **SNTIRewardOnChangingTargetSelection**: SNTI reward signal value. Available values: float.
- **SNSBRewardOnAreaVisited**: SNSB reward signal value. Available values: float.
- **SNSBRewardOnItemSpotted**: SNSB reward signal value. Available values: float.
- **SNSBRewardOnItemsDroppedIntoShelter**: SNSB reward signal value. Available values: float.
- **SNSBRewardOnItemsDroppedIntoShelterWeight**: SNSB reward signal value. Available values: float.
- **SNSBRewardOnItemCollected**: SNSB reward signal value. Available values: float.
- **SNSBRewardOnEndSessionEvacuated**: SNSB reward signal value. Available values: float.
- **SNSBRewardOnEndSessionNotEvacuated**: SNSB reward signal value. Available values: float.
- **SNSBRewardOnEndSessionCollectionCount**: SNSB reward signal value. Available values: float.

- **SNSBRewardOnEndSessionNoCollection**: SNSB reward signal value. Available values: float.
- **SNSBRewardOnEvacuateDuringVeryLateGame**: SNSB reward signal value. Available values: float.
- **SNSBRewardOnValidBehaviour**: SNSB reward signal value. Available values: float.
- **SNSBRewardOnInvalidBehaviour**: SNSB reward signal value. Available values: float.

C.2.5 System Requirements

Agent Simulator minimum requirements for Windows machines, based on the minimal requirements for Unity standalone player, version 2023 LTS:

- **Operating system version**: Windows 10 version 21H1 (build 19043) or newer.
- **CPU**: x86, x64 architecture with SSE2 instruction set support.
- **Graphics API**: DirectX10, DirectX11, and DirectX12 capable GPUs.
- **Additional requirements**: Hardware vendor officially supported drivers
- **For development**: IL2CPP scripting backend requires Visual Studio 2019 with C++ Tools component or later and Windows 10+ SDK.

C.2.6 Learning Model Trainig

Training of learning models using the agent simulator environment was conducted in the Anaconda environment. The Anaconda environment configuration file used in the process is included in the digital supplement to the dissertation and can be found at: */software/anaconda/AnacondaML*
To start a learning session using the agent simulator environment and *ML-Agents*, a learning script is required. Alternatively, a custom *mlagents-learn* command, referencing the agent simulator environment and a hyperparameter configuration file, may be used. Hyperparameter configuration files used in our research work, as well as models produced using these configurations, are available at */data/models/*.

Please note that while training new learning models using the agent simulator environment is supported, it is not possible to use these models for inference in the agent simulator environment binary. Unity does not support streaming of ONNX files. Embedding of such files in the simulator would require full access to the simulator project files in the Unity Editor.

C.2.7 Learning Node

The learning node implemented in the course of our research work for use with *ML-Agents* is located at */software/learning_node/*, along with the relevant *Behavior Bricks* package.

C.3 Data Processing Pipeline

The data processing pipeline used in our research work was implemented in Python and executed in the Anaconda environment. The Anaconda environment configuration file for data processing can be found at: */software/anaconda/AnacondaThesisEnvironmentConfig.yaml*. Python source files can be found at: */source/processing/*. While the processed gameplay telemetry dataset only contains processed data samples, the agent simulator environment can be used to generate new, raw samples from human or AI-controlled playthroughs of the game. These samples can then be processed using the data processing pipeline. Processing batch scripts used in our research work are available at: */source/processing*.

C.4 Jupyter Notebooks

Analysis and data work presented in Chapter 5: Game Score Study and Chapter 6: Agent Study were carried out in Jupyter Notebooks, with Python source file dependencies. Chapter 5: Game Score Study Jupyter Notebook can be found at: */notebooks/game_score_study.ipynb*. Chapter 6: Agent Study Jupyter Notebook can be found at: */notebooks/agent_study.ipynb*. Jupyter Notebooks were executed in the same Anaconda environment configuration as the data processing pipeline. Batch script to run our notebook environment is available at:
/source/thesis/run_jupyter_notebooks.bat.

C.5 Thesis Generation

Using the resources contained within the digital supplement to the dissertation, it is possible to recreate the flow of analysis conducted in the course of our research work. This can be achieved by running the batch script to initiate the notebook environment, launching the flow notebook at */notebooks/thesis_flow.ipynb*, and following instructions contained within it.
Execution of the notebook logic will generate results, stored at */data/results.json*, and the majority of figures included in our thesis. Note that some figures were originally created as static images, and as such cannot be procedurally recreated. The thesis generation batch script */source/thesis/run_thesis_pipeline.bat* will call the thesis pipeline, which will feed generated data to thesis source files (located at: */text*), and compile them using LaTeX.

Appendix D

Additional Results

Summary. The Additional Results appendix presents supplementary results in support of our research work, including calculations, tables and figures that were not included in the main body of the thesis.

D.1 Chapter 5: Game Score Study

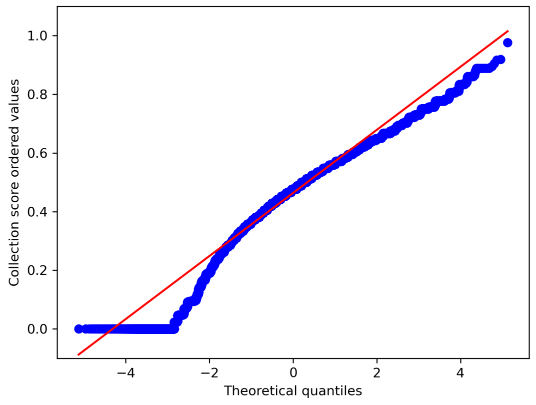
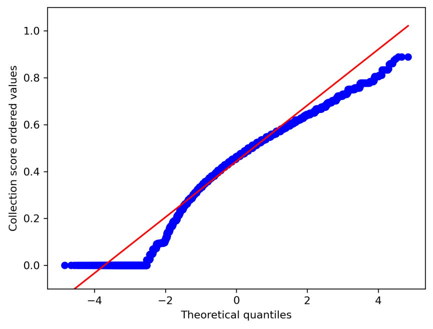
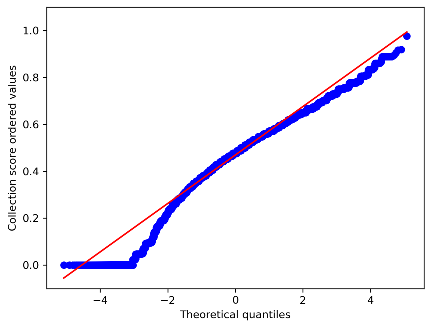


Figure D.1: Dataset POP population collection score distribution QQ plot.

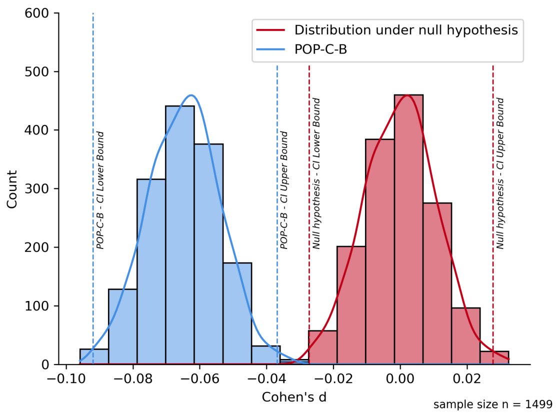


(a) POP-NC QQ plot

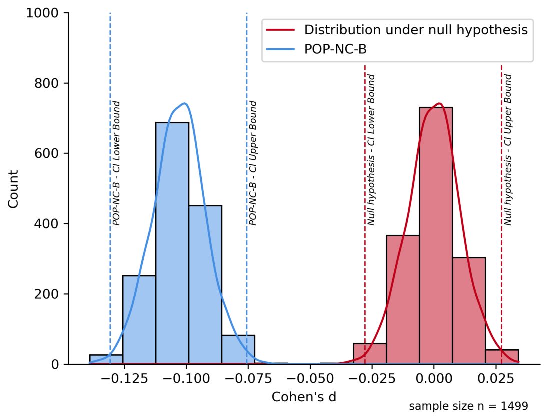


(b) POP-C QQ plot

Figure D.2: QQ plots for POP-C and POP-NC distributions, revealing normality violations in both distributions.



(a) Bootstrapped distribution POP-C-B.



(b) Bootstrapped distribution POP-NC-B.

Figure D.3: Standardised mean difference bootstrapped distribution histograms of POP-C-B and POP-NC-B, with confidence intervals for the bootstrapped distribution and distribution under null hypothesis marked.

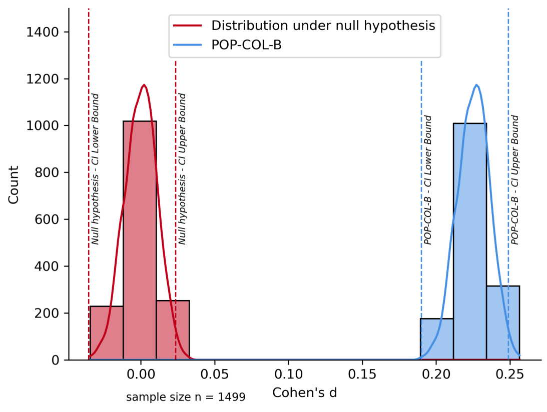


Figure D.4: Standardised mean difference bootstrapped distribution histogram of POP-COL-B, with confidence intervals for the bootstrapped distribution and distribution under null hypothesis marked.

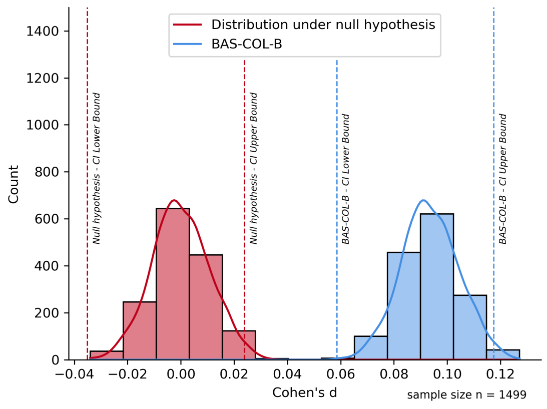


Figure D.5: Standardised mean difference bootstrapped distribution histogram of BAS-COL-B, with confidence intervals for the bootstrapped distribution and distribution under null hypothesis marked.

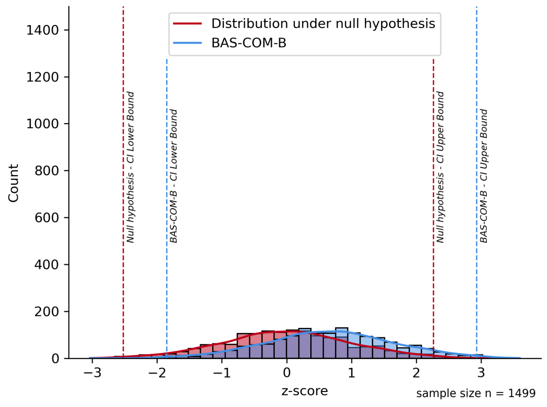


Figure D.6: Completion proportion mean difference bootstrapped distribution histogram of BAS-COM-B, with confidence intervals for the bootstrapped distribution and distribution under null hypothesis marked.

D.2 Chapter 6: Agent Study

```
"AgentType": "CONTEXT",
"AgentVariant" : "",
"Character": "DAD",
"RunSpeed" : 10.0,
"CameraMode" : "GAME",
"GameGUIActive" : "False",
"AnimateNavigation" : "False",
"ActiveLearningBehaviours":["SelectNextTargetItem",
"SelectNextScavengeBehaviour"],
"SNTIRewardOnItemCollected" : 1.0,
"SNTIRewardOnEndSessionCollectionCount" : 1.0,
"SNTIRewardOnEndSessionNoCollection" : -5.0,
"SNTIRewardOnSelectedItemWillFitInInventory" : 0.1,
"SNTIRewardOnSelectedItemWillNotFitInInventory" : -0.1,
"SNTIRewardOnSelectedItemIsFirstOfItsType" : 0.1,
"SNTIRewardOnSuccesfulTargetSelection" : 0.1,
"SNTIRewardOnFailedTargetSelection" : -0.1,
"SNTIRewardOnChangingTargetSelection" : -0.1,
"SNSBRewardOnAreaVisited" : 1.0,
"SNSBRewardOnItemSpotted" : 0.25,
"SNSBRewardOnItemsDroppedIntoShelter" : 1.0,
"SNSBRewardOnItemsDroppedIntoShelterWeight" : 0.25,
"SNSBRewardOnItemCollected" : 1.0,
"SNSBRewardOnEndSessionEvacuated" : 1.0,
"SNSBRewardOnEndSessionNotEvacuated" : -5.0,
"SNSBRewardOnEndSessionCollectionCount" : 1.0,
"SNSBRewardOnEndSessionNoCollection" : -10.0,
"SNSBRewardOnEvacuateDuringVeryLateGame" : 1.0,
"SNSBRewardOnValidBehaviour" : 0.1,
"SNSBRewardOnInvalidBehaviour" : -2.0
```

Table D.1: Agent configuration file with reward values used for training and inference of all SNSB and SNTI models.

```
default_settings: null
behaviors:
  SelectNextTargetItem:
    trainer_type: ppo
    hyperparameters:
      batch_size: 128
      buffer_size: 10240
      learning_rate: 0.0003
      beta: 0.005
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
      shared_critic: false
      learning_rate_schedule: linear
      beta_schedule: constant
      epsilon_schedule: linear
    network_settings:
      normalize: false
      hidden_units: 128
      num_layers: 2
      vis_encode_type: simple
      memory: null
      goal_conditioning_type: hyper
      deterministic: false
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
        network_settings:
          normalize: false
          hidden_units: 128
          num_layers: 2
          vis_encode_type: simple
          memory: null
          goal_conditioning_type: hyper
          deterministic: false
      curiosity:
        gamma: 0.99
        strength: 0.02
        network_settings:
          normalize: false
          hidden_units: 256
          num_layers: 2
          vis_encode_type: simple
          memory: null
          goal_conditioning_type: hyper
          deterministic: false
        learning_rate: 0.0003
        encoding_size: 256
      gail:
        gamma: 0.99
        strength: 0.1
        network_settings:
          normalize: false
          hidden_units: 128
          num_layers: 2
          vis_encode_type: simple
          memory: null
          goal_conditioning_type: hyper
          deterministic: false
        learning_rate: 0.0003
        encoding_size: 128
        use_actions: false
        use_vail: false
        demo_path: /demos/SNTI/
    init_path: null
    keep_checkpoints: 10
    checkpoint_interval: 500000
    max_steps: 1000000
    time_horizon: 64
    summary_freq: 10000
    threaded: true
    self_play: null
    behavioral_cloning: null
  SelectNextScavengeBehaviour:
    trainer_type: ppo
    hyperparameters:
      batch_size: 128
      buffer_size: 10240
      learning_rate: 0.0003
      beta: 0.005
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
      shared_critic: false
      learning_rate_schedule: linear
      beta_schedule: constant
      epsilon_schedule: linear
    network_settings:
      normalize: false
      hidden_units: 128
      num_layers: 2
      vis_encode_type: simple
      memory: null
      goal_conditioning_type: hyper
      deterministic: false
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
        network_settings:
          normalize: false
          hidden_units: 128
          num_layers: 2
          vis_encode_type: simple
          memory: null
          goal_conditioning_type: hyper
          deterministic: false
      curiosity:
        gamma: 0.99
        strength: 0.02
        network_settings:
          normalize: false
          hidden_units: 256
          num_layers: 2
          vis_encode_type: simple
          memory: null
          goal_conditioning_type: hyper
          deterministic: false
        learning_rate: 0.0003
        encoding_size: 256
      gail:
        gamma: 0.99
        strength: 0.1
        network_settings:
          normalize: false
          hidden_units: 128
          num_layers: 2
          vis_encode_type: simple
          memory: null
          goal_conditioning_type: hyper
          deterministic: false
        learning_rate: 0.0003
        encoding_size: 128
        use_actions: false
        use_vail: false
        demo_path: /demos/SNSB/
    init_path: null
    keep_checkpoints: 10
    checkpoint_interval: 50000
    max_steps: 1000000
    time_horizon: 64
    summary_freq: 10000
    threaded: true
    self_play: null
    behavioral_cloning: null
env_settings:
  env_path: /environments/ContextAgentSimulator
  env_args: null
  base_port: 5005
  num_envs: 10
  num_areas: 1
  seed: -1
  max_lifetime_restarts: 10
  restarts_rate_limit_n: 1
  restarts_rate_limit_period_s: 60
engine_settings:
  width: 84
  height: 84
  quality_level: 5
  time_scale: 10.0
  target_frame_rate: -1
  capture_frame_rate: 0
  no_graphics: true
environment_parameters: null
```

```
checkpoint_settings:
  run_id: Context_A100
  initialize_from: null
  load_model: false
  resume: false
  force: true
  train_model: false
  inference: false
  results_dir: results
torch_settings:
  device: null
debug: false
```

Table D.2: Hyperparameter configuration for the training of SNSB and SNTI models in iteration A000.

```
default_settings: null
behaviors:
  SelectNextTargetItem:
    trainer_type: ppo
    hyperparameters:
      batch_size: 128
      buffer_size: 5120
      learning_rate: 0.0003
      beta: 0.005
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
      shared_critic: false
      learning_rate_schedule: linear
      beta_schedule: constant
      epsilon_schedule: linear
    network_settings:
      normalize: false
      hidden_units: 64
      num_layers: 2
      vis_encode_type: simple
      memory: null
      goal_conditioning_type: hyper
      deterministic: false
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 0.1
        network_settings:
          normalize: false
          hidden_units: 128
          num_layers: 2
          vis_encode_type: simple
          memory: null
          goal_conditioning_type: hyper
          deterministic: false
      curiosity:
        gamma: 0.99
        strength: 0.02
        network_settings:
          normalize: false
          hidden_units: 128
          num_layers: 2
          vis_encode_type: simple
          memory: null
          goal_conditioning_type: hyper
          deterministic: false
        learning_rate: 0.0003
        encoding_size: 128
      gail:
        gamma: 0.99
        strength: 0.9
        network_settings:
          normalize: false
          hidden_units: 128
          num_layers: 2
          vis_encode_type: simple
          memory: null
          goal_conditioning_type: hyper
          deterministic: false
        learning_rate: 0.0003
        encoding_size: 128
        use_actions: false
        use_vail: false
        demo_path: /demos/SNTI/
    init_path: null
    keep_checkpoints: 10
    checkpoint_interval: 500000
    max_steps: 1000000
    time_horizon: 32
    summary_freq: 10000
    threaded: true
    self_play: null
    behavioral_cloning: null
  SelectNextScavengeBehaviour:
    trainer_type: ppo
    hyperparameters:
      batch_size: 128
      buffer_size: 5120
      learning_rate: 0.0003
      beta: 0.005
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
      shared_critic: false
      learning_rate_schedule: linear
      beta_schedule: constant
      epsilon_schedule: linear
    network_settings:
      normalize: false
      hidden_units: 64
      num_layers: 2
      vis_encode_type: simple
      memory: null
      goal_conditioning_type: hyper
      deterministic: false
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 0.1
        network_settings:
          normalize: false
          hidden_units: 128
          num_layers: 2
          vis_encode_type: simple
          memory: null
          goal_conditioning_type: hyper
          deterministic: false
      curiosity:
        gamma: 0.99
        strength: 0.02
        network_settings:
          normalize: false
          hidden_units: 128
          num_layers: 2
          vis_encode_type: simple
          memory: null
          goal_conditioning_type: hyper
          deterministic: false
        learning_rate: 0.0003
        encoding_size: 128
      gail:
        gamma: 0.99
        strength: 0.9
        network_settings:
          normalize: false
          hidden_units: 128
          num_layers: 2
          vis_encode_type: simple
          memory: null
          goal_conditioning_type: hyper
          deterministic: false
        learning_rate: 0.0003
        encoding_size: 128
        use_actions: false
        use_vail: false
        demo_path: /demos/SNSB/
    init_path: null
    keep_checkpoints: 10
    checkpoint_interval: 50000
    max_steps: 1000000
    time_horizon: 32
    summary_freq: 10000
    threaded: true
    self_play: null
    behavioral_cloning: null
env_settings:
  env_path: /environments/ContextAgentSimulator
  env_args: null
  base_port: 5005
  num_envs: 10
  num_areas: 1
  seed: -1
  max_lifetime_restarts: 10
  restarts_rate_limit_n: 1
  restarts_rate_limit_period_s: 60
engine_settings:
  width: 84
  height: 84
  quality_level: 5
  time_scale: 10.0
  target_frame_rate: -1
  capture_frame_rate: 0
  no_graphics: true
environment_parameters: null
checkpoint_settings:
  run_id: Context_A100
  initialize_from: null
  load_model: false
  resume: false
  force: true
  train_model: false
  inference: false
  results_dir: results
torch_settings:
  device: null
debug: false
```

Table D.3: Hyperparameter configuration for the training of SNSB and SNTI models in iteration A100.

```
function SelectNextAction(nextBehaviour, previousBehaviour):

  if nextBehaviour == idle:
    if previousBehaviour == evacuate:
      add reward for valid behaviour
    else:
      add reward for invalid behaviour
  else if nextBehaviour == deposit:
    if is inventory empty:
      add reward for invalid behaviour
    else:
      add reward for valid behaviour
  else if nextBehaviour == evacuate:
    if flow stage == very late game:
      add reward for evacuated during very late game
      add reward for valid behaviour
    else if flow stage == late game:
      add reward for valid behaviour
    else:
      add reward for invalid behaviour
  else if nextBehaviour == explore
    if flow stage == explore:
      add reward for valid behaviour
    else:
      if spotted items count == 0:
        add reward for valid behaviour
      else:
        add reward for invalid behaviour
  else if nextBehaviour == collect:
    if inventory full or if flow stage == explore:
      add reward for invalid behaviour
    else:
      add reward for valid behaviour

function RewardLocalObjectives():

  if new area visited:
    reward += new area reward

  if new item spotted:
    reward += new item spotted reward

  if deposit made:
    reward += deposit reward
    reward += (weight deposited * deposit weight reward)

  if item collected:
    reward += item collected reward

function RewardGlobalObjectives():

  if game completed:

    if game completed successfully:
      reward += game completed reward
    else:
      reward += game not completed reward

    if any items deposited:
      reward += (deposited items count * deposit reward)
    else:
      reward += no deposits reward
```

Table D.4: SNSB reward function pseudocode.

```
function SelectNextAction(nextTargetItemType, previousTargetItemType):

  if nextTargetItemType was spotted and nextTargetItemType != previousTargetItemType:
    add reward for valid target selection

    if will nextTargetItemType fit into inventory:
      add reward for item that will fit not fit into inventory

      if nextTargetItemType was not collected before:
        add reward for collecting first item of its type
    else:
      add reward for item that will not fit into inventory
  else:
    add reward for invalid target selection

function RewardLocalObjectives():

  if item collected:
    reward += item collected reward

function RewardGlobalObjectives():

    if any items deposited:
      reward += (deposited items count * deposit reward)
    else:
      reward += no deposits reward
```

Table D.5: SNTI reward function pseudocode.

```
default_settings: null
behaviors:
  SelectNextTargetItem:
    trainer_type: ppo
    hyperparameters:
      batch_size: 128
      buffer_size: 5120
      learning_rate: 0.0003
      beta: 0.005
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
      shared_critic: false
      learning_rate_schedule: linear
      beta_schedule: constant
      epsilon_schedule: linear
    network_settings:
      normalize: false
      hidden_units: 64
      num_layers: 2
      vis_encode_type: simple
      memory: null
      goal_conditioning_type: hyper
      deterministic: false
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
        network_settings:
          normalize: false
          hidden_units: 128
          num_layers: 2
          vis_encode_type: simple
          memory: null
          goal_conditioning_type: hyper
          deterministic: false
      curiosity:
        gamma: 0.99
        strength: 0.02
        network_settings:
          normalize: false
          hidden_units: 128
          num_layers: 2
          vis_encode_type: simple
          memory: null
          goal_conditioning_type: hyper
          deterministic: false
        learning_rate: 0.0003
        encoding_size: 128
      gail:
        gamma: 0.99
        strength: 0.5
        network_settings:
          normalize: false
          hidden_units: 128
          num_layers: 2
          vis_encode_type: simple
          memory: null
          goal_conditioning_type: hyper
          deterministic: false
        learning_rate: 0.0003
        encoding_size: 128
        use_actions: false
        use_vail: false
        demo_path: /demos/SNTI/
    init_path: null
    keep_checkpoints: 10
    checkpoint_interval: 500000
    max_steps: 1000000
    time_horizon: 32
    summary_freq: 10000
    threaded: true
    self_play: null
    behavioral_cloning: null
  SelectNextScavengeBehaviour:
    trainer_type: ppo
    hyperparameters:
      batch_size: 128
      buffer_size: 5120
      learning_rate: 0.0003
      beta: 0.005
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
      shared_critic: false
      learning_rate_schedule: linear
      beta_schedule: constant
      epsilon_schedule: linear
    network_settings:
      normalize: false
      hidden_units: 64
      num_layers: 2
      vis_encode_type: simple
      memory: null
      goal_conditioning_type: hyper
      deterministic: false
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
        network_settings:
          normalize: false
          hidden_units: 128
          num_layers: 2
          vis_encode_type: simple
          memory: null
          goal_conditioning_type: hyper
          deterministic: false
      curiosity:
        gamma: 0.99
        strength: 0.02
        network_settings:
          normalize: false
          hidden_units: 128
          num_layers: 2
          vis_encode_type: simple
          memory: null
          goal_conditioning_type: hyper
          deterministic: false
        learning_rate: 0.0003
        encoding_size: 128
      gail:
        gamma: 0.99
        strength: 0.5
        network_settings:
          normalize: false
          hidden_units: 128
          num_layers: 2
          vis_encode_type: simple
          memory: null
          goal_conditioning_type: hyper
          deterministic: false
        learning_rate: 0.0003
        encoding_size: 128
        use_actions: false
        use_vail: false
        demo_path: /demos/SNSB/
    init_path: null
    keep_checkpoints: 10
    checkpoint_interval: 50000
    max_steps: 1000000
    time_horizon: 32
    summary_freq: 10000
    threaded: true
    self_play: null
    behavioral_cloning: null
env_settings:
  env_path: /environments/ContextAgentSimulator
  env_args: null
  base_port: 5005
  num_envs: 10
  num_areas: 1
  seed: -1
  max_lifetime_restarts: 10
  restarts_rate_limit_n: 1
  restarts_rate_limit_period_s: 60
engine_settings:
  width: 84
  height: 84
  quality_level: 5
  time_scale: 10.0
  target_frame_rate: -1
  capture_frame_rate: 0
  no_graphics: true
environment_parameters: null
checkpoint_settings:
  run_id: Context_A200
  initialize_from: null
  load_model: false
  resume: false
  force: true
  train_model: false
  inference: false
  results_dir: results
torch_settings:
  device: null
debug: false
```

Table D.6: Hyperparameter configuration for the training of SNSB and SNTI models in iteration A200.

Model	**Training time**	s_{col}	s_{com}	s	**Extrinsic reward**	**GAIL reward**	**Cumulative reward**
SNSB A000	12 hours 58 minutes 54 seconds	0.19	0.14	0.17	20.5	0.25	20.05
SNSB A100	2 hours 27 minutes 6 seconds	0	1	0.5	-21.94	12.41	-219.4
SNSB A200	13 hours 15 minutes 58 seconds	0.19	0.13	0.16	19.72	0.9	19.72
SNTI A000	24 hours 18 minutes 47 seconds	0.23	0.53	0.38	15.39	0.02	15.39
SNTI A100	4 hours 14 minutes 37 seconds	0.04	0.87	0.46	-0.35	0.26	-3.49
SNTI A200	4 hours 15 minutes 43 seconds	0.09	0.91	0.5	-3.05	0.16	-3.05

Table D.7: Results of training SNSB and SNTI models A000, A100, and A200, expressed in mean collection scores, mean completion scores, mean full scores, mean extrinsic reward, mean GAIL reward, and mean cumulative reward achieved after training for one million steps.

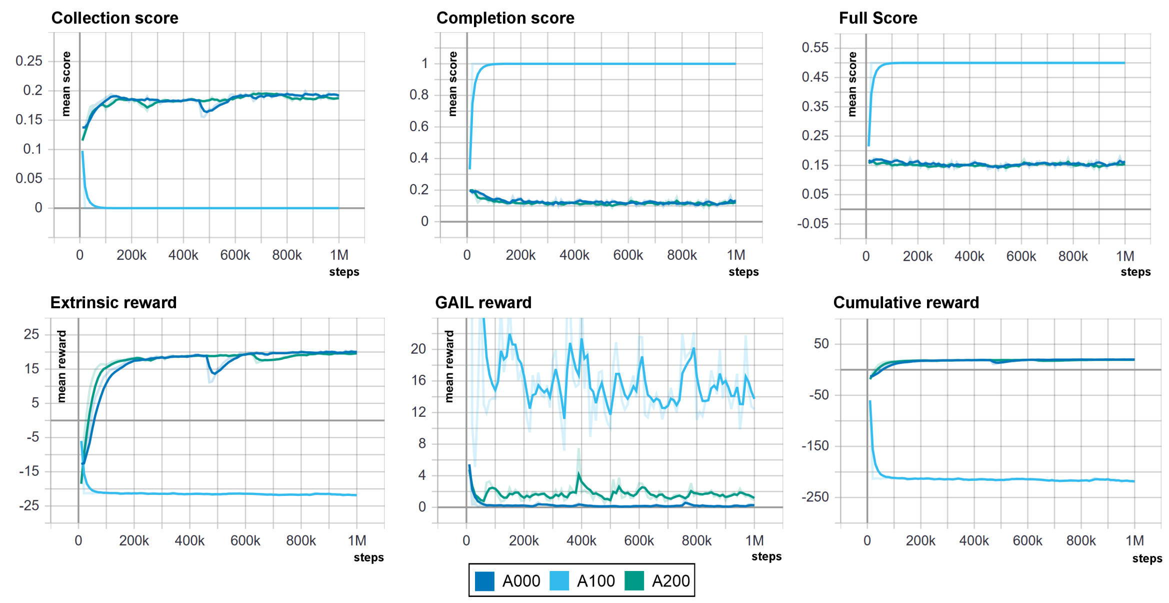


Figure D.7: Visual comparison of the mean values achieved by SNSB learning models A000, A100, and A200 during training. Graphs were generated in Tensorboard.

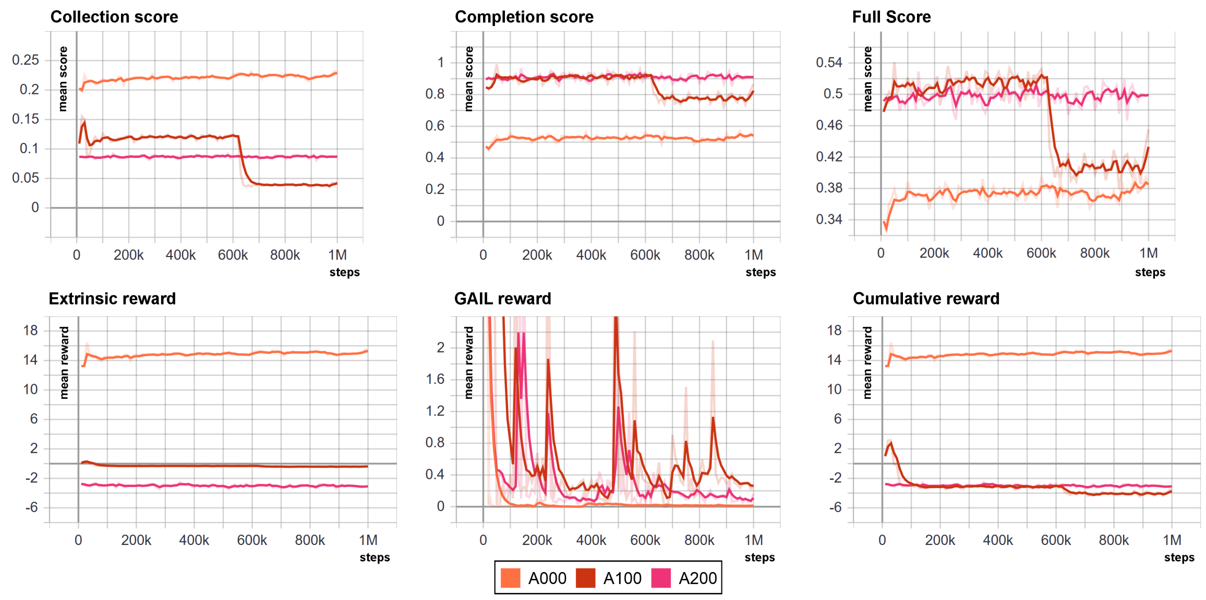
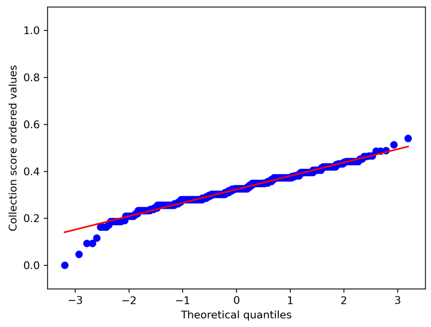
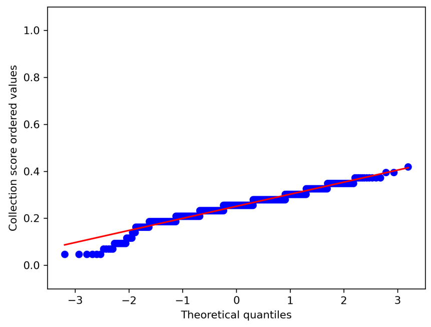


Figure D.8: Visual comparison of the mean values achieved by SNTI learning models A000, A100, and A200 during training. Graphs were generated in Tensorboard.

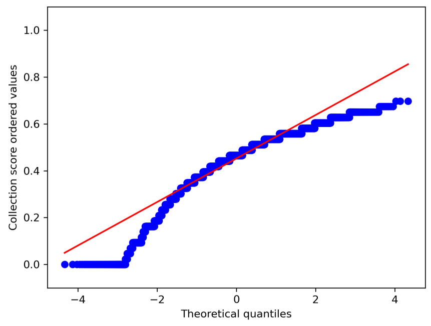


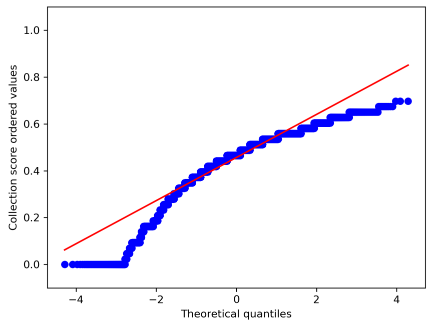


(a) Reference agent model.

(b) Context-guided agent model.

Figure D.9: QQ plots for collection score value distributions, produced by the reference and context-guided agent models during benchmarking.





(a) Dataset EHA QQ plot.

(b) Dataset EH QQ plot.

Figure D.10: QQ plots for collection score value distributions in dataset EHA, featuring all online evaluation samples, and dataset EH, containing samples from user playthroughs 2-5.

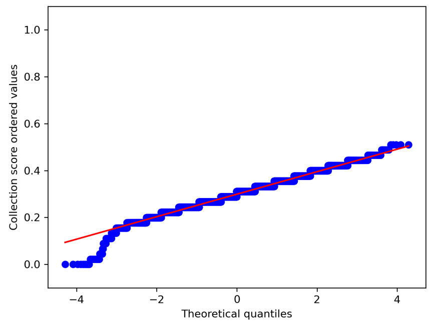


Figure D.11: QQ plot for collection score value distribution in dataset EA.

Dataset	CA0
Sample size	1000
Values	s_{col}
Measurement	**Value**
Min	0.02
Max	0.44
Mean	0.22
Median	0.23
Mode	[0.23255814]
Standard deviation	0.06

Dataset	CA0
Sample size	1000
Values	s_{com}
Measurement	**Value**
Min	0.0
Max	1.0
Mean	0.13
Median	0.0
Mode	[0.]
Standard deviation	0.34

Dataset	CA1
Sample size	1000
Values	s_{col}
Measurement	**Value**
Min	0.0
Max	0.4
Mean	0.22
Median	0.23
Mode	[0.23255814]
Standard deviation	0.06

Dataset	CA1
Sample size	1000
Values	s_{com}
Measurement	**Value**
Min	0.0
Max	1.0
Mean	0.15
Median	0.0
Mode	[0.]
Standard deviation	0.36

Dataset	CA2
Sample size	1000
Values	s_{col}
Measurement	**Value**
Min	0.0
Max	0.49
Mean	0.24
Median	0.23
Mode	[0.25581395]
Standard deviation	0.05

Dataset	CA2
Sample size	1000
Values	s_{com}
Measurement	**Value**
Min	0.0
Max	1.0
Mean	0.12
Median	0.0
Mode	[0.]
Standard deviation	0.33

Table D.8: Statistical description of the s_{col} and s_{com} value distributions of benchmarking context-guided agent models A000, A100, and A200.

Bibliography

[1] S. Abdelfattah, A. Brown, and P. Zhang, "Preference-conditioned pixelbased AI agent for game testing," in Conference on Games. IEEE, 2023.

[2] E. Adams, and J. Dormans, "Game Mechanics: Advanced Game Design," Pearson Education, Kindle Edition, 2012.

[3] E. Adlum, "The Replay Years: Reflections from Eddie Adlum," RePlay. Vol. 11. No. 2, pp. 134-175 (152). November 1985.

[4] T. Allart, G. Levieux, M. Pierfitte, A. Guilloux, and S. Natkin, "Difficulty influence on motivation over time in video games using survival analysis," Proceedings of Foundation of Digital Games, Cap Cod, MA, USA, 2017.

[5] D. Antotsiou, C. Ciliberto, and T. Kim. "Modular Adaptive Policy Selection for Multi-Task Imitation Learning through Task Division," 2022 international Conference on Robotics and Automation (ICRA), pp. 2459-2465, 2022.

[6] S. Ariyurek, A. Betin-Can, and E. Surer, "Automated Video Game Testing Using Synthetic and Human-Like Agents," IEEE Transactions on Games, Vol. 13, Issue 1, pp. 50-67, March 2021.

[7] S. C. J. Bakkes, P. H. M. Spronck, and G. van Lankveld, "Player behavioural modelling for video games," Entertainment Computing, Vol. 3, Issue 3, pp. 71-79, 2012.

[8] S. O. Barriales, "Building a Risk-Free Environment to Enhance Prototyping," in S. Rabin (ed.), *Game AI Pro 2*, CRC Press, Chapter 10, pp. 69-87, 2015, available online: http://www.gameaipro.com/GameAIPro2/GameAIPro2_Chapter10_Building_a_Risk-Free_Environment_to_Enhance_Prototyping.pdf.

[9] J. E. Bartlett II, J. W. Kotrlik, and C. C. Higgins, "Organizational Research: Determining Appropriate Sample Size in Survey Research," Information Technology, Learning, and Performance Journal, Vol. 19, No. 1, Spring 2001.

[10] C. Bateman, "Beyond Game Design: Nine Steps Toward Creating Better Videogames," Course Technology, Charles River Media/Cengage Technology, 2009.

[11] E. Beeching, J. Debangoye, O. Simonin, and C. Wolf, “Godot Reinforcement Learning Agents,” arXiv preprint arXiv:2112.03636, 2021.

[12] J. Bergdahl, C. Gordillo, K. Tollmar, and L. Gisslén, “Augmenting automated game testing with deep reinforcement learning,” Proceedings of 2020 IEEE Conference on Games (CoG). IEEE, pp. 600-603, 2020.

[13] J. Bergdahl, A. Sestini, L. Gisslén,, “Reinforcement Learning for High-Level Strategic Control in Tower Defense Games,” arXiv preprint arXiv:2406.07980, 2024.

[14] F. Berkenkamp, M. Turchetta, A. Schoellig, and A. Krause, “Safe model-based reinforcement learning with stability guarantees,” in Advances in Neural information Processing Systems, pp. 908-918, 2017.

[15] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. d.O. Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang, “Dota 2 with large scale deep reinforcement learning,” arXiv preprint arXiv:1912.06680, 2019.

[16] A. Gekker. and A. Bernevega, “The industry of Landlords: Exploring the Assetization of the Triple-A Game,” Games and Culture, Vol. 17, pp. 47-69, April 2021.

[17] C. M. Bishop, “Pattern Recognition and Machine Learning,” Springer, 2006.

[18] S. Bjor, and J. Holopainen, “Patterns in Game Design,” Charles River Media, 2005.

[19] J. Blow, “Game Development: Harder Than You Think: Ten or twenty years ago it was all fun and games. Now it’s blood, sweat, and code,” Queue, Vol. 1, Issue 10, pp. 28-37, 2004.

[20] R. Bono, J. Arnau, M. J. Blanca, and R. Alarcón, “Sphericity estimation bias for repeated measures designs in simulation studies,” Behav. Res. Methods 48, pp. 1621-1630, 2016.

[21] I. Borovikov, “Imitation Learning via Bootstrapped Demonstrations in an Open-World Video Game,” Workshop on Reinforcement Learning under Partial Observability, NeurIPS, 2018.

[22] I. Borovikov, J. Harder, M. Sadovsky, and A. Beirami, “Towards interactive Training of Non-Player Characters in Video Games,” At ICML 2019 Workshop on Human in the Loop Learning (HILL), June 2019.

[23] I. Borovikov, Y. Zhao, A. Beirami, J. Harder, J. Kolen, J. Pestrak, J. Pinto, R. Pourabolghasem, H. Chaput, M. Sardari, L. Lin, N. Aghdaie, and K. Zaman, “Winning Is Not Everything: Training Agents to Playtest Modern Games,” AAAI Workshop on Reinforcement Learning in Games 2019, January 2019.

[24] I. Borovikov, and A. Beirami, “From Demonstrations and Knowledge Engineering to a DNN Agent in a Modern Open-World Video Game,” Proceedings of the AAAI 2019 Spring Symposium on Combining Machine Learning with Knowledge Engineering (AAAI-MAKE 2019), 2019.

[25] I. Borovikov, “AI-Driven Autoplay Agents for Prelaunch Game Tuning,” in S. Rabin (ed.), *Game AI Pro - Online Edition 2021*, CRC Press, Chapter 10, 2021, Available online: http://www.gameaipro.com/GameAIProOnlineEdition2021/GameAIProOnlineEdition2021_Chapter10_AI-Driven_Autoplay_Agents_for_Prelaunch_Game_Tuning.pdf.

[26] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “OpenAI Gym,” arXiv preprint arXiv:1606.01540, 2016.

[27] R. A. Brooks, “A robust layered control system for a mobile robot,” IEEE Journal of Robotics and Automation. 2 (1), pp. 14-23, 1986.

[28] R. A. Brooks, “Elephants Don’t Play Chess,” Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back. MIT Press, 1990.

[29] C. Buche, C. Even, and J. Soler, “Autonomous Virtual Player in a Video Game Imitating Human Players: the ORION Framework,” 2018 international Conference on Cyberworlds, 2018.

[30] C. Buche, C. Even, and J. Soler, “Orion: A Generic Model and Tool for Data Mining,” in M. L. Gavrilova, C. K. Tan, and A. Sourin (eds.), “Transactions on Computational Science XXXVI: Special Issue on Cyberworlds and Cybersecurity. Lecture Notes in Computer Science,” Springer, Berlin, Heidleberg, pp. 1-25, 2020.

[31] S. Burton, I. Habli, T. Lawton, J. McDermid, P. Morgan, and Z. Porter, “Mind the gaps: Assuring the safety of autonomous systems from an engineering, ethical, and legal perspective,” Artificial Intelligence, Volume 279, February 2020.

[32] D. Camarena, N. Counter, D. Markelov, P. Gagliano, D. Nguyen, R. Becker, F. Firby, Z. Rahman, R. Rosenbaum, L.A. Clarke, and M. Skibinski, “Little Learning Machines: Real-Time Deep Reinforcement Learning as a Casual Creativity Game,” Proceedings of the Experimental Artificial Intelligence in Games Workshop co-located with the 19th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2023), Salt Lake City, Utah, USA, October 8, 2023.

[33] A. Canossa, A. Drachen, “Patterns of Play: Play-Personas in User-Centred Game Development,” in Proceedings of the DiGRA International Conference, 2009.

[34] A. Canossa, D. Salimov, A. Azadvar, C. Harteveld, and G. Yannakakis, "For Honor, for Toxicity: Detecting Toxic Behavior through Gameplay," Proceedings of the ACM on Human-Computer Interaction, Volume 5, Article No. 253, pp 1-29, 2021.

[35] J. M. Carroll, "Making Use: Scenario-Based Design of Human-Computer interactions," MIT Press, 2003.

[36] A. Champandard, and P. Dunstan, "The Behavior Tree Starter Kit," in S. Rabin (ed.), *Game AI Pro* CRC Press, Chapter 6, pp. 73-91, 2013, available online: http://www.gameaipro.com/GameAIPro/GameAIPro_Chapter06_The_Behavior_Tree _Starter_Kit.pdf.

[37] O. Chapelle, P. Shivaswamy, S. Vadrevu, K. Weinberger, Y. Zhang, and B. Tseng, "Boosted Multi-task Learning," Mach. Learn., 85(1-2):149-173, Oct. 2011.

[38] Y. Chow, O. Nachum, E. Duenez-Guzman, and M. Ghavamzadeh, "A lyapunov-based approach to safe reinforcement learning," arXiv preprint arXiv:1805.07708, 2018.

[39] W. G. Cochran, "Sampling Techniques," 3rd Edition, New York: John Wiley and Sons inc., 1977.

[40] J. Cohen, "Statistical Power Analysis for the Behavioral Sciences," 2nd Edition, Hillsdale: Lawrence Erlbaum Associates, 1988.

[41] T. Cokelaer, "FITTER," 2019. Retrieved 28.09 2022: https://fitter.readthedocs.io

[42] M. Colledanchise, and P. Ögren, "How Behavior Trees Modularize Robustness and Safety in Hybrid Systems," Proceedings of IEEE/RSJ international Conference on intelligent Robots and Systems, pp. 1482-1488, June 2014.

[43] M. Colledanchise, R. Parasuraman, and P. Ögren, "Learning of Behavior Trees for Autonomous Agents," IEEE Transactions on Computational intelligence and AI in Games, Vol. 11, Issue 2, pp. 183-189, March 2018.

[44] M. Colledanchise, and P. Ögren, "How Behavior Trees Modularize Hybrid Control Systems and Generalize Sequential Behavior Compositions, the Subsumption Architecture, and Decision Trees," in "IEEE Transactions on Robotics," Vol. 33. Issue: 2, pp. 372-389. April 2017.

[45] M. Colledanchise, and P. Ögren, "Behavior Trees in Robotics and AI: An introduction," CRC Press, 2018.

[46] G. Costikyan, "Uncertainty in Games,“ Massachusetts: MIT Press, 2013.

[47] D. Crankshaw, X. Wang, J. E. Gonzalez, and M. J. Franklin, “Scalable Traning and Serving of Personalized Models,” in Workshop on Machine Learning Systems at NIPS, December 2015.

[48] C. A. Cruz, and J. A. R. Uresti, “HRLB^2: A Reinforcement Learning Based Framework for Believable Bots,” Applied Sciences, 8(12):2453, 2018.

[49] M. Csı’kszentmihĺyi, “Flow: the psychology of optimal experience,” New York: Harper & Row, 1990.

[50] R. B. D’Agostino, A. J. Belanger, and R. B. Jr. D’Agostino, “OA suggestion for using powerful and informative tests of normality,” American Statistician 44, pp. 316-321, 1989.

[51] M. Dann, F. Zambetta, and J. Thangarajah, “Deriving Subgoals Autonomously to Accelerate Learning in Sparse Reward Domains,” Proceedings of the AAAI Conference on Artificial intelligence, Vol. 33(01), pp. 881-889, 2019.

[52] R. Davidson, and J. G. MacKinnon, “Bootstrap tests: How many bootstraps?,” Econometric Reviews. 19(1), pp. 55-68, 2000.

[53] D. L. Davies, and D. W. Bouldin, “A Cluster Separation Measure,” IEEE Transactions on Pattern Analysis and Machine intelligence. PAMI-1 (2), pp. 224-227, 1979.

[54] P. Dayan, and G. Hinton, “Feudal Reinforcement Learning,” Advances in Neural Information Processing Systems 5, 10, 2000.

[55] A. Debner, “Scaling up Deep Reinforcement Learning for intelligent Video Game Agents,” 2022 IEEE international Conference on Smart Computing (SMARTCOMP), 2022.

[56] S. Devlin, R. Georgescu, I. Momennejad, J. Rzepecki, E. Zuniga, G. Costello, G. Leroy, A. Shaw, and K. Hofmann, “Navigation Turing Test (NTT): Learning to Evaluate Human-Like Navigation,” Proceedings of the 38th International Conference on Machine Learning (ICML), pp. 2644-2653, 2021.

[57] R. Dey, and C. Child, “QL-BT: Enhancing Behaviour Tree Design and Implementation with Q-Learning,” 2013 IEEE Conference on Computational inteligence in Games (CIG), pp. 1-8, 2013.

[58] T. J. DiCiccio, and B. Efron, “Bootstrap confidence intervals,” Statistical Science. 11(3), pp. 189-228. August 1996.

[59] K. Dill, “What is Game AI?,” in Rabin S. (ed.), “Game AI Pro,” CRC Press, Chapter 1, pp. 3-9, 2013, available online: http://www.gameaipro.com/GameAIPro/GameAIPro_Chapter01_What_is_Game_AI.pdf.

[60] A. Dockhorn, T. Tippelt, and R. Kurse, “Model Decomposition for Forward Model Approximation,” 2018 IEEE Symposium Series on Computational intelligence (SSCI), 2018.

[61] A. Drachen, A. Canossa, and G. N. Yannakakis, “Player Modeling using Self-Organization in Tomb Raider: Underworld,” Proceedings of IEEE Computational intelligence in Games, pp. 1-8, 2009.

[62] A. Drachen, R. Sifa, C. Bauckhage, and C. Thurau, “Guns, Swords and Data: Clustering of Player Behavior in Computer Games in the Wild,” Proc. IEEE CIG 2012, pp. 163-170, 2012.

[63] A. Drachen, C. Thurau, R. Sifa, and C. Bauckhage, “A Comparison of Methods for Player Clustering via Behavioral Telemetry,” Proceedings of the Eighth international Conference on the Foundations of Digital Games, pp. 245-252, 2013.

[64] A. Drachen, P. Mirza-Babaei, and L. E. Nacke (eds.), “Games User Research,” Oxford University Press, 2018.

[65] P. Duersch, M. Lambrecht, and J. Oechssler, “Measuring skill and chance in games,“ European Economic Review, Volume 127, 2020.

[66] B. Efron, and R. Tibshirani, “An introduction to the Bootstrap,” New York: Chapman & Hall, 1993.

[67] B. Efron, T. Hastie, “Computer Age Statistical Inference: Algorithms, Evidence, and Data Science,” Cambridge University Press, 2016.

[68] M. S. El-Nasr, A. Drachen, and A. Canossa, “Game Analytics Maximizing the Value of Player Data,” Springer, 2013.

[69] M. S. El-Nasr, T. D. Nguyen, A. Canossa, and A. Drachen, “Game Data Science,” Oxford University Press, 2022.

[70] A.E. Elo, “The Proposed USCF Rating System, Its Development, Theory, and Applications,“ Chess Life. XXII (8): 242-247, August 1967.

[71] M. P. P. Faria, R. M. S. Julia, and L. B. P. Tomaz, “Evaluating the Performance of the Deep Active Imitation Learning Algorithm in the Dynamic Environment of FIFA Player Agents,” 18th IEEE international Conference on Machine Learning and Applications (ICMLA), pp. 228-233, 2019.

[72] G. Flø’rez-Puga, M. A. Gø’mez-Martı’n, P. P. Gø’mez-Martı’n, B. Dı’az-Agudo, P. A. Gonzĺez-Calero, “Query enabled behaviour trees,” IEEE Trans. Comput. Intell. AI Games 1(4), pp. 298-308, December 2009.

[73] L. Floridi, J. Cowls, T. C. King, and M. Taddeo, “How to Design AI for Social Good: Seven Essential Factors,” Science and Engineering Ethics, Volume 26, pp. 1771-1796, April 2020.

[74] A. Fod, M.J. Mataric, and O.C. Jenkins, “Automated Derivation of Primitives for Movement Classification,” Autonomous Robots 12(1), pp. 39-54, 2002.

[75] K. French, S. Wu, Z. Zhou, and O. C. Jenkins, “Learning Behavior Trees from Demonstration,” 2019 international Conference on Robotics and Automation (ICRA), pp. 7791-7797, 2019.

[76] Y. Fu, L. Qin, and Q. Yin, “A reinforcement learning behavior tree framework for game AI,” in 2016 International Conference on Economics, Social Science, Arts, Education and Management Engineering, Atlantis Press, 2016.

[77] F. Fuchs, Y. Song, E. Kaufmann, D. Scaramuzza, &, and P. Dürr, “Super-human performance in Gran Turismo Sport using deep reinforcement learning,” in IEEE Robot. Autom. Lett. 6, pp. 4257-4264, 2021.

[78] T. Fullerton, “Game Design Workshop: A Playcentric Approach to Creating innovative Games,” CRC Press. Taylor & Francis Group, 2014.

[79] B. G. Galef, “Imitations in animals: History, definitions, and interpretations of data from the psychological laboratory,” in T. R. Zentall, and Jr. B. G. Galef (eds.), *Social Learning*. Lawrence Erlbaum, pp. 3-28, 1988.

[80] M. Gardner, “Mathematical Games: The Fantastic Combinations of John Conway’s New Solitaire Game ’Life’,” Scientific American, 223, pp. 120-123, 1970.

[81] J. Gemrot, R. Kadlec, M. Bída, O. Burkert, R. Píbil, J. Havlícek, L. Zemĉák, J. Ŝimloviĉ, R. Vansa, M. Ŝtolba, T. Plch, and C. Brom, “Pogamut 3 can assist developers in building AI (not only) for their videogame agents,” in “Agents for games and simulations,” pp. 1-15. Springer, 2009.

[82] A. Géron, “Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow,” O’Reilly, 2nd Edition, 2019.

[83] A. Ghasemi, and S. Zahediasl, “Normality Tests for Statistical Analysis: A Guide for Non-Statisticians,” International Journal of Endocrinology and Metabolism, 10(2), pp. 486-489, 2012.

[84] V. Giammarino, M. F. Dunne, K. N. Moore, M. E. Hasselmo, C. E. Stern, and I. Ch. Paschalidis, “Learning from Humans: Combining Imitation and Deep Reinforcement Learning

to Accomplish Human-Level Performance on a Virtual Foraging Task," arXiv preprint arXiv:2203.06250, 2022.

[85] J. Gillberg, J. Bergdahl, A. Sestini, A. Eakins, and L. Gisslén, "Technical Challenges of Deploying Reinforcement Learning Agents for Game Testing in AAA Games," Proceedings of 2023 IEEE Conference on Games (CoG), pp. 1-8, 2023.

[86] T. Gilovich, D. Griffin, and D. Kahneman, "Hueristics and Biases: The Psychology of Intuitive Judgment," Cambridge University Press, 2002.

[87] I. Goodfellow, Y. Bengio, and A. Courville, "Deep Learning," MIT Press, 2016.

[88] D. Gotojuch, "Towards Controlled Design of Learning Agents for Automated Video Game Playing," in P. Roberts (ed.), *Game AI Uncovered: Volume Two*. CRC Press, Chapter 10, pp. 85-94, 2024.

[89] S. Grand, D. Clif, and A. Malhotra, "Creatures: Artifcial life autonomous software agents for home entertainment," Proceedings of the first international conference on Autonomous agents, pp. 22-29, 1997.

[90] A. Gronlund, K.G. Larsen, A. Mathiasen, J.S. Nielsen, S. Schneider, and M. Song, "Fast Exact K-Means, k-Medians and Bregman Divergence Clustering in 1D," arXiv preprint arXiv:1701.07204, 2017.

[91] C. Guceklsberger, C. Salge, J. Gow, and P. Cairns, "Predicting Player Experience Without the Player An Exploratory Study," in CHI PLAY '17: Proceedings of the Annual Symposium on Computer-Human interaction in Play, pp. 305-315, October 2017.

[92] S. F. Gudmundsson, L. Cao, and R. Meurling, "Human-Like Playtesting with Deep Learning," 2018 IEEE Conference on Computational intelligence and Games (CIG), pp. 1-8, 2018.

[93] C. Guerrero-Romero, S. M. Lucas, and D. Perez-Liebana, "Using a Team of General AI Algorithms to Assist Game Design and Testing," 2018 IEEE Conference on Computational intelligence and Games (CIG), pp. 1-8, 2018.

[94] C. Gulcehre, Z. Wang, A. Novikov, T. L. Paine, S. G. Colmenarejo, K. Zolna, R. Agarwal, J. Merel, D. Mankowitz, C. Paduraru, J. L. Dulac-Arnold, M. Norouzi, M. Hoffman, O. Nachum, G. Tucker, N. Heess, de, and N. Freitas, "RL Unplugged: A Suite of Benchmarks for Offline Reinforcement Learning," in Neural information Processing Systems, pp. 7248-7259, 2020.

[95] D. Hafner, K-H. Lee, I. Fischer, and P. Abbeel, "Deep Hierarchical Planning From Pixels," Advances in Neural Information Processing Systems 35 (NeurIPS 2022), 2022.

[96] S. Hanlon, and C. Watts,"Dragon Age Inquisition's Utility Scoring Architecture," in S. Rabin (ed.), *Game AI Pro 3: Collected Wisdom of Game AI Professionals*, Taylor & Francis, pp. 371-379, 2017.

[97] J. Harmer, L. Gisslèn, del, J. Val, H. Holst, J. Bergdahl, T. Olsson, K. Sjöö, and M. Nordin, "Imitation Learning with Concurrent Actions in 3D Games," arxiv preprint arXiv:1803.05402, 2018.

[98] R. Herbrich, T. Minka, and T. Graepel, "TrueSkill™: A Bayesian Skill Rating System," in Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference, MIT Press, pp. 569-576, 2007.

[99] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband, G. Dulac-Arnold, J. Agapiou, J. Leibo, and A. Gruslys, "Deep Q-learning From Demonstrations," Proceedings of the AAAI Conference on Artificial intelligence, Vol. 32(1), 2018.

[100] P. Hingston, "Believable Bots: Can Computers Play Like People?," Springer, 2012.

[101] J. Ho, and S. Ermon, "Generative Adversarial Imitation Learning," NIPS'16: Proceedings of the 30th international Conference on Neural information Processing Systems, pp. 4572-4580, 2016.

[102] C. Hodent, "The Gamer's Brain: How Neuroscience and UX Can Impact Video Game Design," CRC Press. Taylor & Francis Group, 2018.

[103] E. Hollnagel, "Human Reliability Analysis: Context & Control," Academic Press, 1994.

[104] C. Holmgård, M. C. Green, A. Liapis, and J. Togelius, "Automated Playtesting with Procedural Personas through MCTS with Evolved Heuristics," IEEE Transactions on Games, Volume 11, pp. 352-362, 2018.

[105] N. van Hoorn, J. Togelius, D. Wierstra and J. Schmidhuber, "Robust player imitation using multiobjective evolution," 2009 IEEE Congress on Evolutionary Computation, Trondheim, Norway, pp. 652-659, 2009.

[106] F.-H. Hsu,"Behind Deep Blue: Building the Computer that Defeated the World Chess Champion," Princeton N.J. Princeton University Press, 2002.

[107] C. Huang, L. Zhang, Y. Jing, and D. Zhou, "Efficient Imitation Learning for Game AI," in 2020 IEEE Conference on Games (CoG), pp. 128-135, 2020.

[108] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation Learning: A Survey of Learning Methods," ACM Computing Surveys, Volume 50, Issue 2, Article No. 21, pp. 1-35, 2017.

[109] J. Huizinga, “Homo Ludens,” Routledge. Digital Edition 2009, 1938.

[110] M. Iovino, E. Scukins, J. Styrud, and C. S. Ögren, “A Survey of Behavior Trees in Robotics and AI,” Robotics and Autonomous Systems. Vol. 154. August 2022.

[111] M. Iovino, I. F. Doğan, I. Leite, and C. Smith, “Interactive Disambiguation for Behavior Tree Execution,” arXiv preprint arXiv:2203.02994, 2022.

[112] N. Iskander, A. Simoni, E. Alonso, and M. Peter, “Reinforcement learning agents for Ubisoft’s Roller Champions,” arXiv preprint arXiv:2012.06031, 2020.

[113] G. F. Jenks, “The Data Model Concept in Statistical Mapping,” International Yearbook of Cartography 7, pp. 186-190, 1967.

[114] O. Johnson, “Information Theory and The Central Limit Theorem,” Imperial College Press. London, 2005.

[115] M. Johnson, K. Hofmann, T. Hutton, and D. Bignell, “The Malmo Platform for Artificial intelligence Experimentation,” in “IJCAI,” pp. 4246-4247. 2016.

[116] S. M. Jordan, Y. Chandak, D. Cohen, M. Zhang, and P. S. Thomas, “Evaluating the performance of reinforcement learning algorithms,” in international Conference on Machine Learning, 2020.

[117] K. Judah, A. P. Fern, T. G. Dietterich, and P. Tadepalli, “Active Imitation Learning: Formal and Practical Reductions to I.I.D. Learning,” Journal of Machine Learning Research, Vol. 15, pp. 4105-4143, 2014.

[118] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar, and A. Lange, “Unity: A General Platform for intelligent Agents,” arXiv preprint arXiv:1809.02627, 2018.

[119] A. Juliani, A. Khalifa, V.-P. Berges, J. Harper, E. Ten, H. Henry, A. Crespi, J. Togelius, and D. Lange, “Obstacle Tower: A Generalization Challenge in Vision, Control and Planning,” IJCAI’19: Proceedings of the 28th International Joint Conference on Artificial Intelligence, pp. 2684-2691, August 2019.

[120] K. Jung, J. Lee, V. Gupta, and G. Cho, “Comparison of Bootstrap Confidence interval Methods for GSCA Using Monte Carlo Simulation,” Frontiers in Psychology, Vol. 10, 2019.

[121] Z. Juozapaitis, A. Koul, A. Fern, M. Erwig, and F. Doshi-Velez, “Explainable reinforcement learning via reward decomposition,” in IJCAI/ECAI Workshop on Explainable Artificial intelligence, 2019.

[122] N. Justesen, P. Bontrager, J. Togelius, and S. Risi, "Deep Learning for Video Game Playing," in IEEE Transactions on Games, Vol.: 12, Issue: 1, pp. 1-20, March 2020.

[123] J. Juul, "The game, the player, the world: looking for a heart of gameness," Level Up: Digital Games Research Conference Proceedings, Vol. 1, pp. 30-45, 2003.

[124] J. Juul, "Handmade Pixels Independent Video Games and the Quest for Authenticity," MIT Press, 2019.

[125] D. Kahneman, P. Slovic, and A. Tversky, "Judgement under uncertainty: Heuristics and biases," Cambridge University Press, 1982.

[126] D. Kahneman, and A. Tversky, "Choices, Values, and Frames," Cambridge University Press, 2000.

[127] D. Kahneman, and J. Riis, "Living, and thinking about it: Two perspectives on life. The science of well-being,", pp. 285-304, 2005.

[128] F. Kamrani, L. J. Luotsinen, and R. A. Løvlid, "Learning Objective Agent Behavior using a Data-Driven Modeling Approach," Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, Budapest, Hungary, pp. 2175-2181, 9-12 October 2016.

[129] A. Kanervisto, V. Hautamäki, and C. Scheller, "Action Space Shaping in Deep Reinforcement Learning," 2020 IEEE Conference on Games (CoG), pp. 479-486, 2020.

[130] A. Kanervisto, J. Pussinen, and V. Hautamäki, "Benchmarking End-to-End Behavioural Cloning on Video Games," IEEE Conference on Games 2020. Virtual, pp. 558-565, 2020.

[131] S. Karimi, S. Asadi, F. Lorenzo, and A. H. Payberah, "CandyRL: A hybrid reinforcement learning model for gameplay," in IEEE International Conference on Machine Learning and Applications, 2022.

[132] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaskowski, "Vizdoom: A doom-based AI research platform for visual reinforcement learning," arXiv preprint arXiv:1605.02097, 2016.

[133] G. Klein, "Sources of Power: How People Make Decisions," MIT Press, 1998.

[134] A. Kobanda, C.A. Valliappan, J. Romoff, and L. Denoyer "Learning Computational Efficient Bots with Costly Features," arXiv preprint arXiv:2308.09629, 2023.

[135] E. Kohler, C. Keysers, M.A. Umiltà, V. Gallese, L. Fogassi, and G. Rizzolatti, "Hearing Sounds, Understanding Actions: Action Representation in Mirror Neurons," Science 297, pp. 846-848, 2002.

[136] S. Kolouri, P. E. Pope, C. E. Martin, and G. K. Rohde, "Sliced Wasserstein Auto-Encoders," International Conference on Learning Representations 2019, 2019.

[137] R. Koster, "A Theory of Fun for Game Design," O'Reilly Media inc., 2013.

[138] T. D. Kulkarni, K. Narasimhan , A. Saeedi, and J. Tenenbaum, "Hierarchical deep reinforcement learning: integrating temporal abstraction and intrinsic motivation," Advances in neural information processing systems 2016, pp. 3675-3683, 2016.

[139] A. Kumar, J. Hong, A. Singh, and S. Levine, "When Should We Prefer Offline Reinforcement Learning Over Behavioral Cloning?," arXiv preprint arXiv:2204.05618, 2022.

[140] J. Lai, X. Chen, and X. Zhang, "Training an Agent for Third-person Shooter Game Using Unity ML-Agents," 2019 international Conference on Artifical Intelligence and Computing Science, 2019.

[141] J. E. Laird, van, and M. Lent, "Human-level AI's Killer Application: interactive Computer Games," Proceedings of the Seventh National Conference on Artificial intelligence (AAAI), pp. 1171-1178, 2000.

[142] H. M. Le, N. Jiang, A. Agarwal, M. Dudik, Y. Yue, III, and H. Daumé, "Hierarchical Imitation and Reinforcement Learning," Proceedings of the 35th international Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

[143] M. Lewis, and K. Dill, "Game AI Appreciation, Revisited," in S. Rabin (ed.), *Game AI Pro 2* CRC Press, Chapter 1, pp. 3-9, 2015, available online: http://www.gameaipro.com/GameAIPro2/GameAIPro2_Chapter01_Game_AI_Appreciation_Revis

[144] L. Li, L. Wang, Y. Li, and J. Sheng, "Mixed Deep Reinforcement Learning-behavior Tree for Intelligent Agents Design," Proceedings of the 13th international Conference on Agents and Artificial intelligence, Vol. 1: ICAART, pp. 113-124, 2021.

[145] L. Liden. "Artificial stupidity: The art of intentional mistakes" in S. Rabin (ed.), *AI Game Programming Wisdom 2*, Charles River Media, pp. 41-48, 2003.

[146] C. Lim, R. Baumgarten, and S. Colton, "Evolving Behaviour Trees for the Commercial Game DEFCON," European Conference on the Applications of Evolutionary Computation, pp. 100-110, 2010.

[147] J. MacGlashan, E. Archer, A. Devlic, T. Seno, C. Sherstan, P. R. Wurman, and P. Stone, "Value Function Decomposition for Iterative Design of Reinforcement Learning Agents," arXiv preprint arXiv:2206.13901, 2022.

[148] T. Machado, D. Gopstein, A. Nealen, O. Nov, and J. Togelius, "AI-assisted game debugging with Cicero," 2018 IEEE Congress on Evolutionary Computation (CEC), July 2018.

[149] P. Maes, "Artificial life meets entertainment: lifelike autonomous agents," Communications of the ACM, Volume 38, Issue 11, pp. 108-114, November 1995.

[150] T. Mahlmann, A. Drachen, J. Togelius, A. Canossa, and G. N. Yannakakis, "Predicting player behavior in Tomb Raider: Underworld," Proceedings of the 2010 IEEE Conference on Computational intelligence and Games, pp. 178-185, 2010.

[151] T. W. Malone, "Heuristics for designing enjoyable user interfaces: lessons from computer games," Proceedings of the 1982 Conference on Human Factors in Computing Systems, pp. 63-68, 1982.

[152] K. Manabe, and Y. Miyake, "Game Balancing using Genetic Algorithms to Generate Player Agents," in S. Rabin (ed.), *Game AI Pro - Online Edition 2021*, Chapter 17, 2021, Available online: http://www.gameaipro.com/GameAIProOnlineEdition2021/GameAIProOnlineEdition2021_Chapter17_Game_Balancing_using_Genetic_Algorithms_to_Generate_Player_Agents.pdf.

[153] A. Matthias, "The responsibility gap: Ascribing responsibility for the actions of learning automata," Ethics and Information Technology, Volume 6, pp. 175-183, 2004.

[154] J. McCarthy, "Chess as the Drosophila of AI," in: T. A. Marsland, and J. Schaeffer (eds.), *Computers, Chess, and Cognition*. Springer, pp. 227-237, 1990.

[155] P. McCorduck, "Machines Who Think: A Personal inquiry into the History and Prospects of Artificial intelligence," 2nd Edition. CRC Press. 1979.

[156] W.S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," Bulletin of Mathematical Biophysics, 5, 1943.

[157] T. Micceri, "The unicorn, the normal curve, and other improbable creatures," Psychol. Bull. 105, pp. 156-166, 1989.

[158] S. Milani, A. Juliani, I. Momennejad, R. Georgescu, J. Rzepecki, A. Shaw, G. Costello, F. Fang, S. Devlin, and K. Hofmann, "Navigates Like Me: Understanding How People Evaluate Human-Like AI in Video Games," CHI '23: Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, Article No.: 572, pp. 1-18, April 2023.

[159] G. A. Miller, "The magical number seven, plus or minus two: Some limits on our capacity for processing information," Psychological Review. 63(2), pp. 81-97, 1956.

[160] I. Millington, "AI for Games," Third Edition, CRC Press, 2019.

[161] T. Minka, R. Cleven, and Y. Zaykov, “Trueskill 2: An improved bayesian skill rating system,” Technical Report, 2018. Retrieved 29.02 2024:

[162] O. Missura, and T. Gärtner, “Player Modeling for Intelligent Difficulty Adjustment,” Proceedings of the 12th international Conference on Discovery Science (DS), 2009.

[163] T. Mitchell, “Machine Learning,” McGraw-hill New York, 1997.

[164] Y. Miyake, “Current Status of Applying Artificial Intelligence in Digital Games,”, in R. Nakatsu, M. Rauterberg, and P. Ciancarini (eds.), *Handbook of Digital Games and Entertainment Technologies*, Springer, pp. 253-347, 2017.

[165] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing Atari with deep reinforcement learning,” arXiv preprint arXiv:1312.5602, 2013.

[166] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” Nature, 518(7540), pp. 529-533, 2015.

[167] P. Moschovitis and A. Denisova, “Keep Calm and Aim for the Head: Biofeedback-Controlled Dynamic Difficulty Adjustment in a Horror Game,” IEEE Transactions on Games, pp. 1-10, 2022.

[168] L. Mugrai, F. de Mesentier Silva, C. Holmgård, and J. Togelius, “Automated playtesting of matching tile games,” in Conference on Games. IEEE, 2019.

[169] E. Murphy-Hill, T. Zimmermann, and N. Nagappan, “Cowboys, ankle sprains, and keepers of quality: how is video game development different from software development?,” in the ICSE 2014: Proceedings of the 36th International Conference on Software Engineering, pp. 1-11, May 2014.

[170] S. Mysore, B. E. Mabsout, R. Mancuso, and K. Saenko, “Honey. I Shrunk The Actor: A Case Study on Preserving Performance with Smaller Actors in Actor-Critic RL,” 2021 IEEE Conference on Games (CoG), pp. 01-08, 2021.

[171] O. Nachum, H. Tang, X. Lu, S. Gu, H. Lee, and S. Levine, “Why does hierarchy (sometimes) work so well in reinforcement learning?,” arXiv preprint arXiv:1909.10618, 2019.

[172] N. Nacsimento, P. Alencar, C. Lucena, and D. D. Cowan, “A Context-Aware Machine Learning-based Approach,” in ther Proceedings of the 28th Annual international Conference on Computer Science and Software Engineering, Markham, Ontario, Canada, pp. 40-47, October 2018.

[173] M. J. Nelson, “Game Metrics Without Players: Strategies for Understanding Game Artifacts,” Proceedings of the AAAI Conference on Artificial intelligence and interactive Digital Entertainment, 7(3), pp. 19-24, 2011.

[174] M. Newman, “Atari Age: The Emergence of Video Games in America,” MIT Press, 2017.

[175] A. Y. Ng, and S. Russell, “Algorithms for inverse reinforcement learning,” Proceedings of the Seventeenth International Conference on Machine Learning, pp. 663-670, 2000.

[176] M. Nicolau, D. Perez-Liebana, M. O’Neill, and A. Brabazon, “Evolutionary Behavior Tree Approaches for Navigating Platform Games,” in “IEEE Transactions on Computational intelligence and AI in Games,” Vol. 9. Issue 3. September 2017, pp. 227-238, 2016.

[177] A. Noblega, A. Paes, and E. Clua, “Towards Adaptive Deep Reinforcement Game Balancing,” Proceedings of the 11th international Conference on Agents and Artificial intelligence Vol. 2: ICAART, Prague. Czech Republic, pp. 693-700, 2019.

[178] M. A. North, “A Method for Implementing a Statistically Significant Number of Data Classes in the Jenks Algorithm,” FSKD’09: Proceedings of the 6th international conference on Fuzzy systems and knowledge discovery, Vol. 1, China, pp. 14-16, August 2009.

[179] S. Ocio, “Adapting AI Behaviors To Players in Driver San Francisco Hinted-Execution Behavior Trees,” Proceedings, The Eighth AAAI Conference on Artificial intelligence and interactive Digital Entertainment, 2012.

[180] J. Ortega, N. Shaker, J. Togelius, and G. N. Yannakakis, “Imitating human playing styles in Super Mario Bros,” in “Entertainment Computing,” Vol. 4, Issue 2, pp. 93-104, April 2013.

[181] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, and J. Peters, “An Algorithmic Perspective on Imitation Learning,” Foundations and Trends in Robotics, Vol. 7, No. 1-2, pp. 1-179, 2018.

[182] V. M. Panaretos, and Y. Zemel, “Statistical Aspects of Wasserstein Distances,” Annual Review of Statistics and Its Application. Vol. 6, pp. 405-431. March 2019.

[183] Z. Pang, R. Liu, Z. Meng, Y. Yu, and T. Lu, “On Reinforcement Learning for Full-length Game of StarCraft,” Proceedings of the AAAI Conference on Artificial Intelligence, 33(01), pp. 4691-4698, 2019.

[184] N. Partlan, A. Madkour, C. Jemmali, J. A. Miller, C. Holmgård, and M. S. El-Nasr, “Player Imitation for Build Actions in Real-Time Strategy Game,” AIIDE workshop on Artificial Intelligence for Strategy Games, 2019.

[185] S. Pateria, B. Subagdja, A. Tan, and C. Quek, “Hierarchical Reinforcement Learning: A Comprehensive Survey,” ACM Computing Surveys. Vol. 54, Issue 5, Article No.: 109, pp. 1-35, June 2022.

[186] T. Pearce, and J. Zhu, “Counter-strike deathmatch with large-scale behavioural cloning,” 2022 IEEE Conference on Games (CoG), Beijing, China, pp. 104-111, 2022.

[187] T. Pearce, T. Rashid, A. Kanervisto, D. Bignell, M. Sun, R. Georgescu, S. V. Macua, S. Zheng’Tan, I. Momennejad, K. Hofmann, and S. Devlin, “Imitating Human Behaviour with Diffusion Models,” ICLR, 2023.

[188] C. Pedersen, J. Togelius, an G. N. Yannakakis, “Modeling Player Experience for Content Creation,” IEEE Trans. Computational intelligence and AI in Games, Vol. 2. no. 1, pp. 54-67, 2010.

[189] V. M. Petrović, “Artificial intelligence and Virtual Worlds - Toward Human-Level AI Agents,” in “IEEE Access,” Vol.: 6, pp. 39976-39988, 2018.

[190] J. Pfau, R. Malaka, and J. D. Smeddinc, “Towards Deep Player Behavior Models in MMORPGs,” CHI PLAY ’18: Proceedings of the 2018 Annual Symposium on Computer-Human interaction in Play, pp. 381-392, October 2018.

[191] J. Pfau, J. D. Smeddinck, and R. Malaka, “The Case for Usable AI: What industry Professionals Make of Academic AI in Video Games,” CHI PLAY ’20: Extended Abstracts of the 2020 Annual Symposium on Computer-Human interaction in Play, pp. 330-334, November 2020.

[192] A. Plaat, “Learning to Play: Reinforcement Learning and Games,” Springer, 2020.

[193] D. N. Politis, J. P. Romano, and M. Wolf, “Subsampling,” Springer Verlag, 1999.

[194] C. Politowski, Y-G. Guéhéneuc, and F. Petrillo, “Towards Automated Video Game Testing: Still a Long Way to Go,” 2022 IEEE ACM 6th international Workshop on Games and Software Engineering (GAS), May 2022.

[195] M. L. Puterman, “Markov Decision Processes: Discrete Stochastic Dynamic programming,” John Wiley & Sons, 2014.

[196] S. Rabin, “The Illusion of intelligence,” in S. Rabin (ed.), “Game AI Pro 3,” CRC Press, Chapter 1, pp. 3-9, 2017, available online: http://www.gameaipro.com/GameAIPro3/GameAIPro3_Chapter01_The_Illusion_of_Intelligence.p

[197] S. Rabin (ed.), “Game AI Pro 360 Guide to Architecture,” CRC Press. Taylor & Francis Group, 2020.

[198] S. Rabin (ed.), “Game AI Pro 360 Guide to Movement and Pathfinding,” CRC Press. Taylor & Francis Group, 2020.

[199] A. Ramdas, N. Garcia, and M. Cuturi, “On Wasserstein Two Sample Testing and Related Families of Nonparametric Tests,” arXiv preprint arXiv:1509.02237, 2015.

[200] J. Randløv, “Learning Macro-Actions in Reinforcement Learning,” Advances in Neural information Processing Systems 11 (NIPS 1998), 1998.

[201] N. M. Razali, and B. W. Yap, “Power Comparisons of Shapiro-Wilk, Kolmogorov-Smirnov, Lilliefors and Anderson-Darling Tests,” Journal of statistical modeling and analytics, 2(1), pp. 21-33, 2011.

[202] S. Risi, and M. Preuss, “From Chess and Atari to StarCraft and Beyond: How Game AI is Driving the World of AI,” KI - Künstliche Intelligenz, Vol. 34, pp. 7-17, 2020.

[203] P. Roberts, “Artificial Intelligence in Games,” CRC Press, 2023.

[204] P. Romov, and S. Korolev, “Performance of Machine Learning Algorithms in Predicting Game Outcome from Drafts in Dota 2,” Communications in Computer and information Science, Vol. 661, 2016.

[205] S. Ross, G. Gordon, and D. Bagnell, “A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning,” in Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, PMLR 15:627-635, 2011.

[206] M. B. Rosson, and J. M. Carroll, “Scenario-Based Design,” in “Human-Computer interaction,” CRC Press, pp. 161-180, 2009.

[207] J. Roy, R. Girgis, J. Romoff, P. Bacon, and C. Pal, “Direct Behavior Specification via Constrained Reinforcement Learning,” Proceedings of the 39th International Conference on Machine Learning, PMLR 162:18828-18843, 2022.

[208] Y. Rubner, C. Tomasi, L. J. Guibas, “The earth mover’s distance as a metric for image retrieval,” Int. J. Comput. Vis. 40(2), pp. 99-121, 2000.

[209] S. J. Russell, and A. Zimdars, “Q-decomposition for reinforcement learning agents,”International Conference on Machine Learning (ICML), 2003.

[210] Electronic Arts Search for Extraordinary Experiences Division, “Experimental Self-Learning AI in Battlefield 1,” 2018. Retrieved 07.01 2023: https://www.ea.com/seed/news/self-learning-agents-play-bf1

[211] I. Sagredo-Olivenza, P. P. Gómez-Martín, M. A. Gómez-Martín, and P. A. González-Calero, “Combining Neural Networks for Controlling Non-Player Characters in Games,” in I. Rojas, G. Joya, and A. Catala (eds.), *Advances in Computational intelligence*, Lecture Notes in Computer Science, Springer International Publishing, pp. 695-705, 2017.

[212] I. Sagredo-Olivenza, P. P. Gómez-Martín, M. A. Gómez-Martín, and P. A. González-Calero, “Trained Behaviour Trees: Programming by Demonstration to Support AI Game Designers,” IEEE Transactions on Games. Vol. 11, pp. 5-14, 2019.

[213] P. Samarati and L. Sweeney, “Protecting Privacy when Disclosing Information: k-Anonymity and its Enforcement through Generalization and Suppression,“ Technical Report. SRI International Computer Science Laboratory, 1998. Retrieved 15.06 2025: https://dataprivacylab.org/dataprivacy/projects/kanonymity/paper3.pdf

[214] A.L. Samuel, “Some studies in machine learning using the game of Checkers,” IBM Journal of research and development, 3(3), pp. 210-229, 1959.

[215] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, and T. Graepel, “Mastering Atari, Go, Chess and Shogi by planning with a learned model,” Nature 588, pp. 604-609, 2020.

[216] J. Schrum, I. V. Karpov, Miikkulainen, and R, “UT$\hat{2}$: Human-like behavior via neuroevolution of combat behavior and replay of human traces,” in “Computational intelligence and Games (CIG),” 2011 IEEE Conference, pp. 329-336, 2011.

[217] H. V. Seijen, M. Fatemi, J. Romoff, R. Laroche, T. Barnes, and J. Tsang, “Hybrid reward architecture for reinforcement learning,”. in Conference on Neural information Processing Systems (NeurIPS), 2017.

[218] A. Sestini, J. Bergdahl, K. Tollmar, A. D. Badganov, and L. Gisslén, “Towards Informed Design and Validation Assistance in Computer Games Using Imitation Learning,” in Conference on Neural information Processing Systems (NeurIPS), 2022.

[219] A. Sestini, L. Gisslén, J. Bergdahl, K. Tollmar, and A. D. Bogdanov, “CCPT: Automatic Gameplay Testing and Validation with Curiosity-Conditioned Proximal Trajectories,” arXiv preprint arXiv:2202.10057, 2022.

[220] N. Shaker, S. Asteriadis, G. N. Yannakakis, and K. Karpouzis, “Fusing Visual and Behavioral Cues for Modeling User Experience in Games,” IEEE Transactions on Cybernetics, Vol. 43, no. 6, pp. 1519-1531, 2013.

[221] N. Shaker, J. Togelius, and M. J. Nelson, “Procedural Content Generation in Games,” Springer, 2016.

[222] K. Shao, Z. Tang, Y. Zhu, N. Li, and D. Zhao “A Survey of Deep Reinforcement Learning in Video Games,” arXiv preprint arXiv:1912.10944, 2019.

[223] J. Sharp, and D. Thomas, “Fun, Taste, & Games: An Aesthetics of the Idle, Unproductive, and Otherwise,” MIT Press, Kindle Edition, 2019.

[224] J. Shen, Y. Qu, W. Zhang, and Y. Yu, “Wasserstein Distance Guided Representation Learning for Domain Adaptation,” Proceedings of the AAAI Conference on Artificial Intelligence, 32(1), 2018.

[225] Y. Shin, J. Kim, K. Jin, and Y. B. Kim, “Playtesting in match 3 game using strategic plays via reinforcement learning,” IEEE Access, 2020.

[226] T. X. Short, and T. Adams (eds.), “Procedural Generation in Game Design,” CRC Press Taylor & Francis Group, 2017.

[227] M. Sicart, “Play Matters,” MIT Press, 2014.

[228] L. C. Siebert, M. L. Lupetti, E. Aizenberg, N. Beckers, A. Zgonnikov, H. Veluwenkamp, D. Abbink, E. Giaccardi, G. Houben, C. M. Jonker, J. van den Hoven, D. Forster, and R. L. Lagendijk, “Meaningful human control: actionable properties for AI system development,” AI and Ethics, Volume 3, pp. 241-255, 2023.

[229] de, Mesentier, F. Silva, I. Borovikov, J. Kolen, N. Aghdaie, and K. Zaman, “Exploring Gameplay with AI Agents,” AIIDE’18: Proceedings of the Fourteenth AAAI Conference on Artificial intelligence and interactive Digital Entertainment, Article No.: 23, pp. 159-165, November 2018.

[230] G. A. de Silva, M. W. de Souza Ribeiro, “Development of Non-Player Character with Believable Behavior: a systematic literature review,” SBC - Proceedings of SBGames 2021, 2021.

[231] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, “Mastering the game of Go with deep neural networks and tree search,” Nature 529, pp. 484-489, 2016.

[232] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, “Mastering the game of Go without human knowledge,” in Nature 550, pp. 354-359, 2017.

[233] M. Skarupke, “Automated AI Testing: Simple tests will save you time,” in S. Rabin (ed.), *Game AI Pro - Online Edition 2021*, Chapter 1, 2021, Available online: http://www.gameaipro.com/GameAIProOnlineEdition2021/GameAIProOnlineEdition2021_Chapter01_Automated_AI_Testing_Simple_tests_will_save_you_time.pdf.

[234] I. S. Sprague, and P. Ögren, “Adding Neural Network Controllers to Behavior Trees without Destroying Performance Guarantees,” 61th IEEE Conference on Decision and Control (CDC 2022), Cancun, Mexico, 2018.

[235] P. Spronck, E. André, M. Cook, and M. Preuß, “Artificial and Computational intelligence in Games: AI-Driven Game Design. Dagstuhl Reports,” Vol. 7, Issue 11, pp. 86-129, 2017.

[236] S. N. Stahlke, and P. Mirza-Babaei, “User Testing Without the User: Opportunities and Challenges of an AI-Driven Approach in Games User Research,” Computers in Entertainment, Vol. 16. Issue 2, Article No.: 9, pp. 1-18, April 2018.

[237] S. Stahlke, A. Nova, and P. Mirza-Babaei, “Artificial players in the design process: Developing an automated testing tool for game level and world design,” in Proceedings of the Annual Symposium on Computer-Human Interaction in Play, pp. 267-280, 2020.

[238] R. S. Sutton, and A. G. Barto, “Reinforcement Learning: An Introduction,” 2nd Edition. The MIT Press. Cambridge, Massachusetts. London, England, 2020.

[239] P. Sweetser, and P. Wyeth, “Gameflow: a model for evaluating player enjoyment in games,” Computers in Entertainment (CIE) 3(3), 3, 2005.

[240] H. J. Thode, “Testing for normality,” New York, Marcel Dekker, 2002.

[241] T. Thompson, “The Changing Landscape of AI for Game Development,” in P. Roberts (ed.), *Game AI Uncovered: Volume One*. CRC Press, Chapter 1, pp. 1-11, 2024.

[242] R. L. Thorndike, “Who belongs in the family?,” Psychometrika. Vol. 18, pp. 267-276, 1953.

[243] C. Thurau, C. Bauckhage, and G. Sagerer, “Imitation learning at all levels of game AI,” Proceedings of the International Conference on Computer Games, Artificial Intelligence, Design and Education, 2004.

[244] J. Togelius, “Evolution of a subsumption architecture neurocontroller,” Journal of Intelligent & Fuzzy Systems. 15(1), pp. 15-20, 2004.

[245] J. Togelius, G. N. Yannakakis, S. Karakovskiy, and N. Shaker, “Assessing believability,” in P. Hingston, “Believable bots,” Springer, pp. 215-230, 2021.

[246] J. Togelius, “AI researchers, Video Games are your friends!,” in “Computational Intelligence,” Vol. 669, pp. 3-18. Springer. 2017.

[247] J. Togelius, “Playing Smart,” Cambridge, MA: MIT Press, 2018.

[248] T. Tullis, and B. Albert, “Measuring the User Experience: Collecting, Analyzing and Presenting Usability Metrics,” Morgan Kaufmann, 2008.

[249] A. Tychsen and A. Canossa, “Defining Personas in Games Using Metrics,” in Proceedings of the 2008 Conference on Future Play: Research, Play, Share. ACM, pp. 73-80, 2008.

[250] Ubisoft La Forge, “Ubisoft La Forge - Pushing State-Of-The-Art AI in Games To Create The Next Generation Of NPCs,” 2022. Retrieved 07.01 2023: https://montreal.ubisoft.com/en/ubisoft-la-forge-pushing-state-of-the-art-ai-in-games-to-create-the-next-generation-of-npcs

[251] Unity Technologies, “Training ML-Agents,” 2018. Retrieved 01.07 2024: https://github.com/Unity-Technologies/ml-agents/blob/develop/docs/Training-ML-Agents.md#behavior-configurations

[252] Unity Technologies, “Class DecisionRequester,” 2024. Retrieved 01.07 2024: https://docs.unity3d.com/Packages/com.unity.ml-agents3.0/api/Unity.MLAgents.DecisionRequester.html

[253] Unity Technologies, “Unity Discussions ML-Agents,” 2024. Retrieved 01.07 2024: https://discussions.unity.com/tag/ml-agents

[254] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu, “FeUdal Networks for Hierarchical Reinforcement Learning,” Proceedings of the 34th international Conference on Machine Learning, Vol. 70, pp. 3540-3549, 2017.

[255] C. Villani, “Optimal Transport: Old and New,” A Series of Comprehensive Studies in Mathematics, Springer, Volume 338, pp. 93-111, 2009.

[256] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Phlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver, “Grandmaster level in StarCraft II using multi-agent reinforcement learning,” in Nature 575, pp. 350-354, 2019.

[257] J. Von Neumann, “Theory of Self-Reproducing Automata,” University of Illinois Press, 1966.

[258] B. Wagner, “Liable, but Not in Control? Ensuring Meaningful Human Agency in Automated Decision-Making Systems,” Policy & Internet, Vol. 11, No. 1, Wiley Periodicals Inc., 2019.

[259] Z. Wang, J. Merel, S. E. Reed, N. D. Freitas, G. Wayne, and N. Heess, “Robust Imitation of Diverse Behaviors,” NIPS’17: Proceedings of the 31st international Conference on Neural information Processing Systems, pp. 5326-5335, December 2017.

[260] Wei, J. Chen, X. Ji, H. Qin, M. Deng, S. Li, L. Wang, W. Zhang, Y. Yu, L. Liu et al., “Honor of kings arena: an environment for generalization in competitive reinforcement learning,” Advances in Neural Information Processing Systems 2022, 35: pp. 11881-11892, 2022.

[261] M. West, “Intelligent mistakes: How to incorporate stupidity into your AI code”. Game Developer Magazine - Digital Edition, 2008.

[262] R. Wilcox, “Introduction to Robust Estimation and Hypothesis Testing,” 5th Edition, Elsevier, 2021.

[263] B. Wu, Q. Fu, J. Lian, P. Qu, X. Li, L. Wang, W. Liu, W. Yang, and Y. Liu, “Hierarchical Macro Strategy Model for MOBA Game AI,” arXiv preprint arXiv:1812.07887, 2018.

[264] R. P. Wurman, S. Barrett, K. Kawamoto, J. MacGlashan, K. Subramanian, T. J. Walsh, R. Capobianco, A. Devlic, F. Eckert, F. Fuchs, L. Gilpin, P. Khandelwal, V. Kompella, H. Lin, P. MacAlipine, D. Oller, T. Seno, C. Sherstan, M. D. Thomure, H. Aghabozorgi, L. Barret, R. Douglas, D. Whitehead, P. Dürr, P. Stone, M. Spranger, and H. Kitano, “Outracing champion Gran Turismo drivers with deep reinforcement learning,” Nature 602, pp. 223-228, 2022.

[265] D. Xiaoqin, Qinghua, L, and H. Jianjun, “Applying Hierarchical Reinforcement Learning to Computer Games,” Proceedings of the IEEE international Conference on Automation and Logistics, Shenyang, China, August 2009.

[266] T. Xu, Z. Li, and Y. Yu, “Error Bounds of Imitating Policies and Environments,” in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 44, no. 10, pp. 6968-6980, 1 Oct. 2022.

[267] G. N. Yannakakis, “Game AI revisited,” Proceedings of the 9th conference on Computing Frontiers, ACM Press, pp. 285-292, 2012.

[268] G. N. Yannakakis, P. Spronck, D. Loiacono, and E. André, “Player Modeling,” Dagstuhl Seminar on Artificial and Computational intelligence in Games 2013, 2013.

[269] G. N. Yannakakis, and H. P. Martı’nez, “Ratings are Overrated!,” Frontiers in ICT, 2:13, 2015.

[270] G. N. Yannakakis, and J. Togelius, “Artificial intelligence and games,” Springer, 2018.

[271] A. E. Youssef, S. E. Missiry, I. N. El-gaafary, J. S. ElMosalami, K. M. Awad, and K. Yasser, “Building your kingdom Imitation Learning for a Custom Gameplay Using Unity ML-agents,” 2019 IEEE 10th Annual information Technology, Electronics and Mobile Communication Conference (IEMCON), pp. 0509-0514, 2019.

[272] K.K. Yu, M. Guzdial, N.R. Sturtevant, M. Cselinacz, C. Corfe, I.H. Lyall, and C. Smith, “Adventures of AI directors early in the development of Nightingale,” in Proceedings of the Eighteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE’22), Vol. 18. AAAI Press, Article 9, pp. 70-77, 2022.

[273] P. Zackariasson, and T. L. Wilson (eds.), “The Video Game industry: Formation, Present State, and Future,” New York: Routledge, 2012.

[274] Q. Zhang, Y. Quanjun, and K. Xu, 2016, “Towards An integrated Learning Framework for Behavior Modeling of Adaptive CGFs,” Proceedings of the IEEE 9th international Symposium on Computational intelligence and Design (ISCID), Hangzhou, China, VBolume 2, pp. 7-12, 10-11 December 2016.

[275] Q. Zhang, J. Yao, Q. Yin, and Y. Zha, “Learning Behavior Trees for Autonomous Agents with Hybrid Constraints Evolution,” Applied Sciences 8(7):1077, July 2018.

[276] Z. Zhang, H. Li, L. Zhang, T. Zheng, X. Hao, X. Chen, M. Chen, F. Xiao, and W. Zhou, “Hierarchical Reinforcement Learning for Multi-agent MOBA Game,” arXiv preprint arXiv:1901.08004, 2019.

[277] Y. Zhao, I. Borovikov, A. Beirami, J. Rupert, C. Somers, J. Harder, de, Mesentier, F. Silva, J. Kolen, J. Pinto, R. Pourabolghasem, H. Chaput, J. Pestrak, M. Sardari, L. Lin, N. Aghdaie, and K. Zaman, “Winning Is Not Everything: Enhancing Game Development with intelligent Agents,” IEEE Transactions on Games, Vol. 12. Issue 2, pp. 199-212, June 2020.

[278] X. Zhu, “Behavior Tree Design of intelligent Behavior of Non-player Character (NPC) based on Unity3D,” Journal of intelligent & Fuzzy Systems, Vol. 37. Number 5, pp. 6071-6079, 2019.

[279] J. Zhu, J. Villareale, N. Javvaji, S. Risi, M. Löwe, R. Weigelt, and C. Harteveld, “Player-AI Interaction: What Neural Network Games Reveal About AI as Play,” CHI ’21: Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems, Article No.: 77, pp. 1-17, May 2021.

[280] D. W. Zimmerman, “A note on preliminary tests of equality of variances,” Br J Math Stat Psychol, 57 (Pt 1), pp. 173-81, May 2004.

[281] A. E. Zook, and M. O. Riedl, “A Temporal Data-Driven Player Model for Dynamic Difficulty Adjustment,” in AIIDE’12: Proceedings of the Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, pp. 93-98, 2012.

[282] A. Zook, E. Fruchter, and M. O. Riedl, “Automatic Playtesting for Game Parameter Tuning via Active Learning,” arXiv preprint arXiv:1908.01417, 2019.

[283] E. Zuniga, S. Milani, G. Leroy, J. Rzepecki, R. Georgescu, I. Momennejad, D. Bignell, M. Sun, A. Shaw, G. Costello, M. Jacob, S. Devlin, and K. Hofmann, “How Humans Perceive Human-like Behavior in Video Game Navigation,” CHI ’22 Extended Abstracts: CHI Conference on Human Factors in Computing Systems Extended Abstracts, New Orleans, LA, USA, April 2022.

[284] D. Zwillinger, and S. Kokoska, “OCRC Standard Probability and Statistics Tables and Formulae,” Chapman & Hall: New York. 2000.

[285] R. de Pontes Pereira, and M. P. Engel, “A Framework for Constrained and Adaptive Behavior-Based Agents,” arXiv preprint arXiv:1506.02312, 2015.