SYNTHESIZING PLAY

EXPLORING THE USE OF ARTIFICIAL INTELLIGENCE TO EVALUATE GAME USER EXPERIENCE

by

SAMANTHA NICOLE STAHLKE

A thesis submitted to the School of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

Faculty of Business and Information Technology

UNIVERSITY OF ONTARIO INSTITUTE OF TECHNOLOGY
(ONTARIO TECH UNIVERSITY)

OSHAWA, ONTARIO, CANADA

March 2020

THESIS EXAMINATION INFORMATION

Submitted by: Samantha Nicole Stahlke

Master of Science in Computer Science

Thesis Title: Synthesizing Play: Exploring the Use of Artificial Intelligence to Evaluate Game User Experience

An oral defence of this thesis took place on March 25, 2020 in front of the following examining committee:

Examining Committee:

Chair of Examining Committee	Dr. Faisal Qureshi
Research Supervisor	Dr. Pejman Mirza-Babaei
Examining Committee Member	Dr. Jarek Szlichta
Thesis Examiner	Dr. Shahram Heydari

The above committee determined that the thesis is acceptable in form and content and that a satisfactory knowledge of the field covered by the thesis was demonstrated by the candidate during an oral examination. A signed copy of the Certificate of Approval is available from the School of Graduate and Postdoctoral Studies.

ABSTRACT

Digital games are a complex interactive medium providing a multitude of different experiences. The field of games user research (GUR) is dedicated to investigating and optimizing user experience in games. Such inquiries are of both commercial and academic importance, enhancing product quality and our understanding of human behaviour. A common GUR methodology is usertesting, where researchers gain insights from human users interacting with products. However, usertesting is expensive in terms of expert labour, time, and resource costs. To address these concerns, we developed PathOS, a free, open-source tool for game testing with AI agents. PathOS simulates player navigation in games using a basic model of human behaviour. We conducted an evaluation of PathOS with developers, finding that it provides valuable predictions of user behaviour in the iterative design process. Ultimately, we aim to give the game development community a useful and versatile augmentation to their testing processes.

Keywords: games user research; human-computer interaction; user experience; video games; artificial intelligence

AUTHOR'S DECLARATION

I hereby declare that this thesis consists of original work which I have authored. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize the University of Ontario Institute of Technology to lend this thesis to other institutions or individuals for the purpose of scholarly research. I further authorize the University of Ontario Institute of Technology to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research. I understand that my thesis will be made electronically available to the public.

The research work in this thesis that was performed in compliance with the regulations of the university's Research Ethics Board under REB Certificate #15453.

Samantha Stahlke

Samantha Stahlke

For Victoria, from Elizabeth
(and for loops)

STATEMENT OF CONTRIBUTIONS

Some ideas and portions of figures have appeared previously in the following publications:

> Samantha Stahlke, Atiya Nova, and Pejman Mirza-Babaei. “Artificial Playfulness: A Tool for Automated Agent-Based Playtesting”. In: *Proceedings of CHI 2019 Extended Abstracts*. Glasgow, Scotland, UK: ACM, 2019, LBW0176:1–LBW0176:6. DOI: 10.1145/3290607.3313039

> Samantha Stahlke and Pejman Mirza-Babaei. “Usertesting Without the User: Opportunities and Challenges of an AI-Driven Approach in Games User Research”. In: *ACM Comput. Entertain* 16.2 (2018), 18 pp. DOI: 10.1145/3183568

> Samantha Stahlke and Pejman Mirza-Babaei. “Usertesting Without the User: Introducing PathOS, a Framework for AI-Based Games User Research”. In: *Proceedings of the 2017 CHI PLAY Workshop on Exploiting Players*. Amsterdam, Netherlands, 2017, 5 pp.

The prototype developed for this work (described in Chapter 3) was created with the help of Atiya Nova, who worked as a research assistant on this project. I wrote the majority of the code for the prototype presented here. A full history of all code contributions is available on the project’s Github page:
https://github.com/samanthastahlke/pathos.

The user study performed was conducted both remotely and on the university campus. I performed all of the interviews and was the primary contact for study participants.

I hereby certify that I am the sole author of this thesis. All ideas and prior work from others referenced in this work has been attributed appropriately following standard practices.

Parts of this work have been heavily edited and adapted for submission as a paper at ACM CHI PLAY 2020. The writing of this thesis predates the preparation of this submission.

ACKNOWLEDGEMENTS

Completing this research has been a long and exciting journey, which, while not without its difficulties, has been incredibly rewarding. I would not have been able to complete this work without the support of those around me.

I would like to thank my supervisor, Dr. Pejman Mirza-Babaei, for his mentorship and advice. When I told him three years ago that I wanted to "make an AI stupid, like a person" to try and test games, he encouraged me to run with the idea and has helped me in this work ever since. I have learned so much from Pejman over the past five years, and I hope we can continue to collaborate in the years to come. I would also like to thank Atiya Nova, whom I have worked with on this project among many others. Her dedication and research ability are admirable, and her involvement in this work has made it all the better. For all the time and effort she put into this project, I am very grateful.

To all my friends and colleagues in the graduate game lab and our faculty, thank you for putting up with my constant running around and non-stop chatter. For everyone who participated in this study, who shall remain anonymous for obvious reasons, I am incredibly appreciative of the fact that you were willing to lend your time to our research. This work would not exist without your participation.

Ontario Tech University is a wonderful place, and I am proud to have called it my home for the past six years. I am immensely grateful for the university's support in providing the resources necessary for this work, in particular through the Dr. Deborah Saucier Early Researcher Award. I would also like to thank the Natural Sciences and Engineering Research Council of Canada, the Vector Institute, and the Government of Ontario for additional funding supporting this work.

On a personal note, I am most thankful for the support of those I love, throughout this process and always. My friends have been with me through many long nights of writing, coding, and knocking my head against a keyboard. Emilian, comrade, thank you for helping me turn every moment into an opportunity for laughter, and for being the best partner I could imagine. Helen, I'm glad we ended up in graduate school at the same time, for our joyful commiseration has been a lovely addition to our typical shenanigans. Josh and Owen, I couldn't have made it through undergrad without you, and I'll never forget all our adventures.

Lastly, I would like to thank my mother, Svetlana, who has held me up through the past twenty-odd years of schooling it's taken me to get here. Through all of the laughing, and the crying, and the inevitable yelling at submission systems into the odd hours of the morning, you've been there for me. Your love and patience has made me into the person that I am today, and none of this would have been possible without you. Thank you, for everything.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

AI	Artificial Intelligence
API	Application Programming Interface
EMG	Electromyography
FPS	First-Person Shooter
HCI	Human-Computer Interaction
GSR	Galvanic Skin Response
GUR	Games User Research
GVGAI	General Video Game Artificial Intelligence
HUD	Heads-Up Display
IL	Imitation Learning
MCTS	Monte Carlo Tree Search
ML	Machine Learning
NPC	Non-Player Character
PCG	Procedural Content Generation
RL	Reinforcement Learning
RPG	Role-Playing Game
QA	Quality Assurance
QC	Quality Control
UI	User Interface
UX	User Experience

Part I

INTRODUCTION

The act of play, user experience, and the rise of the machines

1

INTRODUCTION

Blackness, illuminated suddenly by a vivid tableau of action and fantasy. The machine below whirs to life, powering the theatrics of its tiny virtual inhabitants. A few inches from the glow of the screen, its owner sits in rapt attention, hands tensely hovering above the keyboard.

The act of play has captivated humans throughout history. Games have taken on a myriad of fascinating forms, from unstructured roleplay to intricate digital epics taking years to master. Modern video games redefine the boundaries of play with innovations in mechanics, world design, and narrative that challenge users' reaction time, creativity, and critical thinking.

As a leisure activity, games can be relaxing. They can permit us to escape the ills of everyday life, serving as a gentle transport into a tranquil world without conflict or strife. In other incarnations, they take the form of chaotic maelstroms built to push the extremes of player coordination and endurance. Games can provide comfort, and excitement. They can move us to laugh, to cry, to clench our fists in rage, or raise them in victory.

Perhaps this versatility is what has made the medium of video games so pervasive. In just a few short decades since their introduction to the mass market, they have grown to become ingrained in the landscape of modern entertainment, technology, and pop culture. This ubiquity is reflected in their commercial importance, as the games industry now drives billions of dollars in consumer spending globally each year.

Regardless of individual variation, most all digital games share a remarkable ability to engage players at a deep level. With every passing frame, the user is tasked with perceiving the virtual world, understanding the entities therein, and contemplating their next action. Every step, every leap, every sword swing—with each interaction, players have the power to affect the game and their experience. The rich interaction of games can allow users to impart their will on the virtual world, crafting unique and interesting experiences.

The depth of interaction afforded by digital games means that players are often faced with an overwhelming number of possible actions at any given point during play. Playing a game is a process filled with decisions, from low-level tasks such as navigation, to the execution of grand long-term strategies. These decisions are affected by a number of factors, such as player skill, past experience, and varying motivations between individual players. While one player may seek out danger and face the most daunting enemies head-on, another may keep to the shadows, more interested in mastering their environment than picking a fight.

Individual variation between players compounds with the inherently large possibility space of games to produce a vast range of potential user experiences. Consequently, the task of understanding how these experiences unfold, how they can be improved, and how they contribute to our broader understanding of human behaviour, is a difficult research challenge.

Games are undoubtedly intricate systems, and yet their intricacy is far outshone by the almost incomprehensibly complex creatures that interact with them.

1.1 GAMES USER RESEARCH

Our relationship with computers has become an indispensable part of modern life. Computer systems help facilitate the global economy, power our homes, and keep us connected to one another. Though these systems are often focused on providing some degree of automation, they are still ultimately designed for human use, at least for the time being. Ensuring successful interaction with these systems is a crucial part of our ability to improve our lives through technology. The field of human-computer interaction (HCI) is concerned with understanding these interactions and how we can apply the resulting insights to the development of new systems.

The particular challenge of understanding our interaction with game systems adds an additional layer of complexity. While productivity applications may have straightforward metrics for measuring their efficacy—the time taken to complete a task, for instance—the definition of a game experience as "successful" is far murkier. Games are a special case in that their primary goal is typically to provide enjoyment, rather than efficiency in completing some task. While this is perhaps the easiest distinction to draw between game systems and those designed for productivity, a myriad of other differentiating factors compound this disparity. For instance, they seek to provide variation in a user's experience, rather than consistency, and often pose a much larger possibility space [4].

So complex is the relationship between humans and games that an entire field of research has emerged within the last few decades dedicated to its study. Games user research (GUR) is a subfield of HCI focused specifically on understanding user experience (UX) in games [5, 6]. Like HCI, GUR is a multidisciplinary field informed by knowledge from several other domains, such as computer science, sociology, and psychology.

Some of the core objectives of GUR revolve around improving game UX, creating novel experiences and development tools, and contributing to our understanding of human behaviour. These goals manifest in a diverse body of work spanning both academic and commercial applications. In commercially-oriented GUR, work is often focused on dissecting or improving UX for a particular title [7], or creating tools to improve developer productivity and product quality [8]. Within the academic realm, projects may also investigate the development of novel hardware and exotic interactions [9], or pursue specialized work related to topics such as game accessibility [10] or representation [11]. Perhaps unsurprisingly, collaboration between academia and industry is also common in GUR, serving as a bridge to help connect commercial and scholarly interests.

A common library of research techniques is shared by much of the work in GUR. Regardless of the context of a particular project, researchers are ultimately concerned with understanding UX in games. Thus, a central methodology in GUR is playtesting, whereby a variety of data are collected from human participants engaging with a game system [5]. These data are then analyzed to answer questions regarding game UX, ranging from the broad "Do players find this game fun?" to more specific inquiries such as "How can we adjust the laser rifle to ensure combat feels balanced?"

Much like usertesting in general HCI, the specific set of data collection and analysis procedures employed in playtesting can vary immensely according to project scope, technical feasibility, and the research questions under investigation. To gather data regarding players' in-game actions, simple observation may be used, often in conjunction with screen and input capture software to keep a precise log of gameplay events.

Another approach leverages the notion of game metrics (e.g., position in game world, number of deaths, etc.), which are collected automatically through instrumentation of a game's code [12, 13]. In addition to the convenience of their collection, metrics are also useful in that they can be used to gather data from players en masse remotely. For instance, after an online game is launched, the developer may collect information on how often players log in, for how long they play, and how much progress they make in-game. As opposed to other techniques for capturing gameplay data (such as screen-recording), metrics also have the advantage of being almost immediately suitable for quantitative analysis and comparatively lower data storage requirements.

Aside from the actions taken by players, UX researchers are also interested in understanding how players think and feel about a game, and how individual moments contribute to overall experience. For instance, during play, researchers may ask players to think aloud and verbally express how they feel about in-game events [5]. More objective measures, such as facial expression recognition or the use of physiological sensors, may also be used to help capture information regarding a player's mental state [5, 14].

After play is finished, other tactics can be employed to probe players' experience and follow up on significant events that may have occurred during gameplay. For instance, questionnaires and interviews may be used to gather subjective feedback from users [5]. Other techniques, such as skill-check interviews, can help to test players' understanding of a game's mechanics [15].

Irrespective of the particular selection of methods used, any given playtest has the same core structure. After participants are recruited, researchers observe their interactions with a game and attempt to understand how these interactions affect their experience. In other words, the goal of a playtest is to find out what users do, why they do it, and how this makes them feel. In commercial GUR, UX researchers are tasked with providing the development team with feedback as to where the actual experience deviates from designer intention. For instance, designers may want to check whether the intended level of difficulty holds in actual playtesting, to ensure that players feel challenged without becoming frustrated.

Playtesting has become an integral part of the game development process; to ensure that the game functions as a product and delivers the intended experience, verification of its technical quality (e.g., bugtesting, quality control) is insufficient. Rather than studying the system in isolation, understanding the journey experienced by real users is critical. Playtesting helps designers to identify flaws in a game's design, and to understand how changes to that design can affect end user experience. The insights gained may serve to contradict expectations and reveal surprising issues, or confirm a game's strong points and help to identify opportunities for fine-tuning and further improvements. At any rate, playtesting serves as an important part of the game development process, as a validation of the intended experience and an opportunity to understand user behaviour.

Despite its value, playtesting faces a number of obstacles which can prove troublesome, especially for commercial GUR practitioners. An important factor in playtesting is the degree to which recruited participants reflect the characteristics of a game's target audience [16]. Depending on the game, this audience might be incredibly diverse, or embody some niche quality (such as experience with a particular game genre). Accounting for these requirements can prove difficult, requiring game studios to maintain a database of potential players that can be screened for desirable characteristics [17]. If a test is completed on short notice, developers run the risk of not being able to find suitable participants at all [16].

In addition to the challenge of recruitment, playtesting can be incredibly time- and resource-intensive to orchestrate. Each playtesting session requires some degree of researcher supervision, with time required to set up, observe players, conduct interviews, and so on. In scaling the number of players, whether sequentially or in parallel, tests require more time and more research personnel, and may require additional equipment. Though some of these challenges can be mitigated by using remote testing tactics (typically at the expense of rich data collection), the expense of researcher time, test scheduling, and participant compensation is always a concern. These demands are especially restrictive for small studios and independent developers, where human labour is at a premium and budgets are often extremely limited [17].

Even with virtually unlimited fiscal resources, the necessary time investment required to facilitate the scheduling and execution of playtests makes it difficult to conduct them repeatedly throughout development, or on short notice. Without a reliable means to efficiently assess player behaviour at each stage of a design iteration, designers are relegated to relying on data from prior tests or educated guesses when attempting to achieve a particular intended experience.

This work is motivated by a desire to help empower game creators, particularly independent developers, to make more informed decisions when iterating on their designs. Rather than relying exclusively on human testers, we propose that approximate predictions of player behaviour may be used early in the development process to help make decisions about a game's world and level design. In doing so, designers can help to avoid glaring usability issues or deviations from the intended experience, without incurring significant expenses or requiring a full game build suitable for testing. In service to this goal, we hope to augment the GUR toolkit with tactics drawn from another field intimately connected with the study of games: artificial intelligence.

1.2 AI AND GAMES

Humans are social creatures that seek out contact as if by instinct, but we have yet to find another form of conscious life. Whether by lack of awareness or merely a cruel coincidence of nature, humans are, for now, alone in the universe. It is pure speculation to say that this isolation is what has driven us to attempt the creation of synthetic companions possessing some aspect of our own likeness. Nonetheless, for whatever reason, the pursuit of artificial intelligence has in some respect fascinated humanity for generations, long before the advent of the digital age.

Humanlike automata have been the subject of myths since antiquity, with the first functioning mechanical humanoids created during the Renaissance. While these early creations largely served as showpieces, precisely hand-tuned to complete specific tasks, they can be thought of as a primitive foray into the field of robotics. The advent of modern computation in the 20th century revolutionized the idea of automation, allowing machines to "think" digitally rather than mechanically.

Today, computer systems are capable of performing many tasks once thought exclusive to human intelligence unassisted. While the term artificial intelligence (AI) lacks a specific and universally accepted definition, it can be understood as the ability of a computer system to acquire and apply skill in the execution of some task. Such systems can be hand-programmed by a human expert to behave as intended. More recently, AI systems often eschew this hand-coded approach for machine learning (ML) protocols, whereby a system learns to function from a library of training data, rather than being explicitly programmed.

The current and theoretical applications of AI are numerous, from simple data processing tasks, to economic forecasting, to the creation of art and music. Given the state of the art, it is difficult to say what as-yet unimaginable possibilities may emerge with another few decades of advancement. Perhaps it is this uncertainty that makes AI so fascinating, and the subject of such media attention. In popular culture, the future of AI has been depicted as both a scientific marvel responsible for humanity's next great leap, and the path to a dystopia responsible for its destruction.

Technological grandstanding aside, the practical benefits of AI make it a worthy area of research, reflected in a constantly growing body of work regarding AI techniques and applications. Interestingly, the development of AI has been historically entwined with games, both in the use of games to showcase and test AI techniques, and the use of AI to enhance games and the process of game development.

Due to their interactive and often skill-demanding nature, games have long served as a safe environment and testbed of sorts for the development of AI systems. The defeat of human masters in classic board games has served to create landmarks in the advancement of AI, such as the victory of IBM's Deep Blue in 1997 [18] and Google's AlphaGo in 2016 [19]. Video games have also been used to push the boundaries of AI, for instance, by overcoming the complex cooperative strategies of human players in *Dota 2* (Valve Corporation, 2013) [20] or combining computer vision with reinforcement learning to "play from pixels" in classic arcade games [21].

While games have been leveraged as a tool for the betterment of AI, the converse is also true; AI has long been used to improve both game experiences and the process of game development itself. AI has driven dynamic opponents and companions in games for decades, from the simpleton ghosts of *Pac-Man* (Bandai Namco, 1980), to the grand strategy of world leaders in *Civilization VI* (Firaxis Games, 2016). Though in-game AI is often hand-coded and comparatively simple in nature, the illusion of life and intelligence serves as a key piece of immersion in many titles.

The creation of game AI whose purpose is to enhance user experience carries unique challenges; rather than striving to maximize skill, developers are often concerned with making behaviour appear "interesting" or "believable". While an opponent along the lines of AlphaGo may present the ultimate challenge, it is also less than ideal for inclusion in a consumer entertainment product. Aside from the obvious mismatch in terms of computational requirements, the behaviour of a perfect adversary can seem jilted in addition to being frustratingly unbeatable. To this end, game AI can be made intentionally imperfect, both as a cover for technical limitations and a means to create more lifelike behaviour.

For the purposes of this work, the creation of humanlike game AI is particularly intriguing; if agents can be made to mimic the qualities of human play, those agents may conceivably used to estimate how real players would act in-game. This prediction, coarse as it may be, could then be used to help designers make decisions on how a game might be tweaked to provide the intended user experience. Agents that emulate player behaviour in some respect may therefore serve as proxies for human playtesters, at least under certain circumstances.

Though using AI agents to test games is far from commonplace, the use of AI as an aid to game developers and UX researchers is well-established. Simple rule-based systems are regularly used as a development tool by way of procedural generation, whereby different forms of content, from in-game items to entire levels, are generated automatically. First appearing in classic titles such as *Elite* (Braben and Bell, 1984) and *Rogue* (A.I. Design, 1980), procedural generation is still widely used to create entire worlds in games like *Spelunky* (Mossmouth, 2008) and *No Man's Sky* (Hello Games, 2016).

Within the realm of game testing and UX research, AI has been employed in several different applications. These efforts have included the automation of quality control (QC) tasks [22], the use of agents as "bugtesters" under human supervision [23], and the analysis of data gathered from human players [24]. While work in the area of AI-augmented GUR is still fairly nascent, these tools are already finding use in industry projects, demonstrating the promise that they hold for empowering game developers and UX professionals.

With the advent of such tools, the relationship between AI and games has shifted, to a degree. Where games once served as purely as challenges for AI to overcome, they now provide a rich collection of opportunities for improving user experience and augmenting the development pipeline. Today, AI is no longer a mere curiosity in the domain of games; instead, we may view it as a powerful assistant in game creation and a newfound collaborator in game research endeavours.

1.3 GUR AND AI

Throughout the course of a given inquiry in games research, a great deal of work is required in the course of organizing playtests, collecting data, analyzing data, and presenting findings to the development team. Traditionally, this work is entirely dependent on human labour, as its complexity forbids the use of routine automation. However, many of these tasks are suitable for the application of AI, whether as an outright replacement for human labour or an assistant to help optimize researcher efforts.

In theory, AI can be used to augment nearly every stage of the GUR process, from test orchestration to the presentation of findings. Though largely preliminary in nature, existing work has investigated several of these opportunities in GUR and related fields, for example, by using AI to help facilitate data analysis [25, 26]. More pertinent to this work, the use of AI for game testing has been explored in applications spanning quality control [22, 23], the validation of playability and difficulty [27–29], and basic UX investigations [30]. A comprehensive overview of this work will be explored further in Chapter 2.

For the time being, let us examine how existing AI techniques can support the GUR process to help illustrate the motivation for this work and the opportunities for future research in this domain. To this end, consider three hypothetical cases using AI-driven tools to help expedite and enrich the investigation of common UX research questions:

CASE 1: PROBING EMOTIONAL EXPERIENCE. A UX researcher on a small team has been tasked with evaluating players' emotional experience in a story-driven roleplaying game currently under development. Specifically, they aim to find out how players react to key moments in the game's narrative and identify any significant changes in player mood during gameplay. To do this, they recruit several players and record footage of their play while connected to physiological sensors monitoring heart rate and skin conductance, before conducting post-gameplay interviews.

Traditionally, inferring player emotion is a challenging problem demanding attentive researcher observation and painstaking analysis of physiological data. Instead of relying on constant note-taking during the session at the risk of missing important moments, our hypothetical researcher uses real-time facial expression recognition to pinpoint moments of intense player reaction for following up during the post-session interview. To further identify key in-game moments which affect player emotion, they visualize players' physiological signals over time with the help of predictive software which identifies and flags outliers likely to be overlooked by a human observer. This serves to both speed up their analysis process and help prevent them from missing potentially important insights.

Interview transcription, normally completed at the cost of hours of researcher time or outsourced to external labourers, is expedited through voice recognition and a speech-to-text system, leaving a clean document for the researcher to sift through. In addition to their own analysis, they employ a sentiment recognition algorithm to gain an impression of players' overall opinions, as well as automatically pulling strong statements from participants for inclusion in their final report to the development team. Lastly, they use a visual summarization utility to automatically select segments from participants' video recordings representative of the average player's experience to assist in their presentation. With the time saved in transcription, analysis, and presentation, they are able to deliver findings more quickly, ultimately saving resources for the development team.

CASE 2: EXAMINING RETENTION POST-LAUNCH. An analyst at a large AAA studio has been asked to evaluate retention in a newly-released online multiplayer game. Specifically, the company is hoping to identify what makes players quit, and if players at risk of quitting could be selectively offered incentives to keep playing.

Several years ago, this analyst might have deployed a survey, or theorized on what might prompt players to quit and manually probe basic metrics collected from players to try and prove his hypotheses. Today, they use machine learning to help automate this process, making analysis of data from millions of players feasible instead of near-impossible to manage. They employ pattern mining to identify potential sequences of game events leading to players' failure to log on for several days, and discover that long matchmaking times, repeated play of certain maps, and earning low amounts of in-game currency often lead to players' withdrawal.

Based on these findings, they suggest contextual incentives may be offered to help boost overall retention: for players that experience several long queues in a row, their matchmaking range should be relaxed to facilitate quicker play; the probability that a troublesome map is selected should be reduced, and the level design team should look into players' experience in that area; and finally, for players routinely earning small amounts of currency, a bonus or free in-game item could be offered selectively to help preserve their motivation. By using a system capable of analyzing massive quantities of data, the analyst is able to discover more valuable information at a faster pace than he otherwise might have been.

case 3: validating world design. A small team of level designers working on an open-world game wants to make sure that players will be incentivized to explore the entire map and interact with all the content it has to offer. They also want to ensure that novice or hesitant players will not be overly intimidated by the dangers present, without alienating more experienced players looking for a challenge.

Before a build suitable for playtesting is ready, they use a behavioural prediction tool to simulate players' navigation through the game world, creating a heatmap of player activity. Within the tool, they customize the AI agents deployed to mimic the behaviour of both inexperienced, cautious players, and aggressive veterans. Finding that certain spots towards the centre of the map have sparse activity, they note that "novice" agents seem to avoid the area, despite several points of interest being present. Noting that the central area of the map is gated on nearly all sides by enemy camps, the designers create an additional path allowing for a more stealth-based option to become viable in the area. Re-running the simulation, they find that this has made the spread of player activity more uniform throughout the map.

After correcting other similar issues using these predictions, the initial version of the world handed off for playtesting has already undergone several rounds of informed iteration. Testing with human users then uncovers subtler issues which may otherwise have been masked by more glaring deviations from the intended experience. In between rounds of testing, predictive tools are still used to help estimate the impact of any changes on players' actual paths through the game world. In doing so, the final version of the game's world will have undergone significantly more iteration while remaining cost-efficient in terms of the researcher and participant time required.

To a UX researcher, the tools described above would be immensely helpful in the course of their work. Just a few years ago, the existence of such tools may have seemed like an impossible dream, or at least, one which could only be realized in the far future. However, while the cases presented above are theoretical in nature, the applications described are not. Each is grounded in the existing literature; though these tactics are not yet commercially widespread, they are no longer pure speculation.

The expression recognition and emotional inference described in Case 1 has been explored in the context of GUR by [25], and factors contributing to perceptual deficits of human analysts have been investigated by [31]. The conversion of speech-to-text and voice recognition used for transcription has long been a subject of interest in general AI literature [32], as has the idea of sentiment analysis [33]. Lastly, automatically selecting representative visuals given a large collection of data is a problem which has also been explored in the computer vision literature [34].

With regard to Case 2, the large-scale analysis of game metrics via machine learning has been applied previously in both the identification of gameplay styles [7] and the prediction of player retention [35]. The use of pattern mining and sequence recognition to predict player behaviours such as in-game goals and time-of-purchase has also been investigated and suggested as a mechanism for application adaptability [36, 37].

Finally, the scenario outlined in Case 3 is based on this work and our prior publications on the subject [1–3], as well as other explorations in the analysis, simulation, or prediction of player's navigation behaviours [38, 39]. Simulating differing player motivations has also been investigated by [30], to help reproduce the variation in gameplay styles observed in real user populations.

With this work, we aim to produce a tool that will help to make automated testing a viable option for everyday GUR practitioners. By creating an openly available utility for automated testing, we hope to improve not only the toolkits of games researchers, but those of countless industry professionals and independent creators.

1.4 PATHOS - AN AI-ASSISTED TOOL FOR GAME WORLD DESIGN

Game world and level design is a complex creative task with significant impact on players' eventual experience. Rich level design encourages players to explore, interact, and make the most of their time in a game's environment. Without proper incentive, players may not realize all a game world has to offer, or deliberately avoid certain areas because they find them uninteresting or intimidating. Furthermore, poor world design may fail to reveal all options available to a player, causing them to become lost or miss sections of content entirely. This is an especially pressing concern in non-linear games, such as open-world titles, where much of the content is optional. Optional content and freeform navigation can help to preserve players' sense of agency; if executed poorly, however, players can easily miss out on the optimal intended experience.

Playtesting helps to reveal issues with a game's level design before launch, allowing developers to make changes improving the flow of content experienced by an average player, or gently nudge players along the intended path. Post-launch, metrics data gathered from the actual player-base may be used to inform changes, but this is only true of online games, and may result in user confusion depending on the adjustments made. Furthermore, such changes made after release may prove "too little, too late" in terms of retaining players and swaying critics.

Ideally, designers want to ensure that the experience they have created aligns with their intentions before a game is released. However, repeatedly conducting playtests to verify the effects of changes is resource-intensive. Attempting to test minor adjustments or multiple alternatives with participants is also logistically challenging, or even impossible for smaller studios. Lastly, the continual emergence of basic level design issues may obscure other important UX questions, such as players' engagement with other game systems (e.g., narrative, combat).

To help address these challenges, this work explores the use of AI agents to replace human users in early-stage testing of a game's world design, allowing designers to pursue informed iteration earlier in the development process. By simulating the logic of human players at a high level, these agents can help designers to estimate player behaviour. Developers can then use this information as input to their iterative design process, helping to push their creations towards producing the intended player experience. For instance, designers may want to maximize the amount of time players spend in a particular area, or the percentage of game entities a player will interact with.

The idea of such a system is not to replace human playtesters; rather, it aims to provide designers with a cost-effective tool for informing their designs earlier in the development process. Such a tool would be used to explore the impact of changes during the iterative design process, before a testing build is available, or to predict the outcome of small changes in between test rounds with human users.

This work outlines the development of PathOS, a level design utility allowing game creators to predict player navigation with AI agents. The PathOS framework is a lightweight, all-in-one tool featuring customizable agent behaviour, real-time and accelerated in-engine simulation, and data visualization. Agents are driven by a simple behavioural model informed by existing work on player psychology. Most importantly, they reflect players' imperfections; rather than attempting to navigate the level in the most optimal manner as a traditional pathfinding AI might, agents are driven by a configurable set of heuristics reflecting the motivations of real users. Furthermore, agents are non-omniscient; instead, a basic model of player perception and memory limits available information to what a human player would see and remember.

These characteristics are intended to help agents navigate more like human players, who may retrace their steps, temporarily become lost, lose track of what objectives are available, or choose to ignore sections of a level which they find unappealing. It is important to note that this tool is concerned only with estimating navigation in the virtual world, rather than more complex gameplay interactions (such as combat). With this in mind, it is our hope that PathOS will serve as a first step of sorts in the eventual development of more complete AI-driven testing solutions for designers and UX researchers to predict player behaviour.

A typical use case for PathOS would be comparing projected navigation patterns between two candidate level alternatives. First, designers define the desired population of users and configure agents accordingly; depending on a game's target audience, this might be highly skilled completionist players, or cautious novices focused on self-preservation. Then, they define a starting point and allow the simulation to run; depending on the designer's workflow, they may accelerate its progress, or observe in real-time as agents move through the game world. Afterwards, the agents' navigation can be visualized as a heatmap or individual playtraces, allowing designers to compare results between the evaluated levels. Depending on the design goal (e.g., maximizing the spread of activity throughout a level), the team can then make a more informed choice between the two alternatives, or recombine them to create the desired experience.

PathOS has been developed as an extension for Unity[1], a freely available and popular commercial game engine. We chose Unity as a platform for developing this tool as it is widely used among small development studios and individual creators, thus providing maximal opportunity for the tool to assist independent developers. PathOS integrates directly with the Unity editor, allowing designers to leverage it as a natural addition to their existing workflow.

This thesis describes the design, development, and evaluation of PathOS, investigating how it can be used to assist in the process of game world design. Chapter 2 examines the precedent for this work, reviewing existing research regarding the applications of AI in GUR, software testing, and data analysis. The design and technical development of the PathOS framework is detailed in Chapter 3. Chapter 4 describes a user study conducted to evaluate the tool's utility for game level design and identify opportunities for its improvement. Finally, a discussion of results and exploration of PathOS' potential applications is given in Chapter 5, before concluding in Chapter 6.

In developing PathOS, our goal is to to provide a free and open-source tool for creators to improve their design workflow and create more engaging player experiences. Ultimately, we aim to both empower developers and support the emergence of AI tools as a valuable part of the GUR landscape.

Our relationship with games is nothing short of magical. Even after decades of work, comprehending how we interact with these rich, engaging experiences continues to be a worthy research challenge. As GUR continues to move forward, we will only enhance our understanding of interactivity and user behaviour.

Perhaps in doing so, we will make these experiences even more captivating.

[1] https://unity.com/

2 RELATED WORK

Just a few decades ago, AI research existed largely separate from practical applications, focused instead on developing new techniques to solve classic problems, such as the mastery of boardgames. The next set of great chess-playing bots or primitive neural networks, while important milestones in the field, were hardly suitable for extensive cross-domain applications. This is in stark contrast with current attitudes towards AI, where computational intelligence finds some new industrial or consumer application on a daily basis. Today, AI is no longer a mere curiosity: it can recognize your voice and schedule your next hair appointment[1], help write your emails[2], replace irritating traditional customer service with irritating automated customer service[3], unlock your phone[4], critique your photography[5], and give you someone to talk to when the other humans simply will not suffice[6].

Once an exotic idea reserved for science fiction in popular culture, AI is now touted as driving everything from quirky video filters to powerhouse data analytics tools aimed at massive corporations. For better or worse, the term "machine learning" may now be misappropriated in marketing perhaps more often than defined in a classroom. Nonetheless, this first step in the generalization of AI has proven itself as a cross-disciplinary boon to researchers as well.

Pertinent to this work, researchers have investigated the role that AI can play in augmenting GUR methodology to become more robust, scalable, cost-effective, and resilient to human error. When considering the challenge of UX evaluation (i.e., playtesting), the GUR process can be loosely divided into two phases: gathering data from players, and processing that data to derive insights regarding UX. Accordingly, applications of AI in GUR can be assigned a similar categorization. First, we have tools for analysis: those which are designed to support researchers in data processing by discovering trends and handling large datasets, for example. Second, we can consider the concept of simulation-driven testing: utilities that are capable of testing some aspect of a game autonomously, or effectively generating test data by predicting the behaviour of human players.

The following sections explore both of these application areas, and how they have evolved to support the work of GUR professionals over the past several years.

[1] Google Assistant: https://assistant.google.com/

[2] Google Smart Compose: https://www.blog.google/products/gmail/subject-write-emails-faster-smart-compose-gmail/

[3] ChatBot: https://www.chatbot.com/

[4] Face ID: https://www.apple.com/ca/iphone-xs/face-id/

[5] Everypixel Aesthetics: https://www.everypixel.com/aesthetics

[6] Replika: https://replika.ai

The integration of AI into GUR analysis workflows has empowered researchers to more efficiently process data, identify notable patterns, and handle large datasets. In this context, AI can be used to process, filter, or group data, effectively extending the limits of researcher observation. So-called "intelligent" interfaces for visualizing or reviewing data can augment human analysis by learning and adapting to user preferences, or automatically flagging anomalous patterns and notable trends for further investigation.

When dealing with especially large datasets, such as metrics acquired from many hundreds or even thousands of players, the problem of data classification and sorting quickly exceeds the scope of human labour. In these situations, AI-driven clustering and pattern mining approaches are now relatively commonplace in attempting to build player models and identify behavioural patterns from enormous collections of data. This has proven particularly useful in the study of player typology, an area historically reliant on purely human insight [40, 41]. Though the advent of AI-powered analysis in GUR is fairly recent as of this writing, these techniques have already proven valuable in improving our ability to process and extract insights from player data.

2.1.1 *Evaluating User States*

The challenge of understanding certain aspects of game UX, such as player emotion, has led researchers to employ a variety of data collection techniques aimed at capturing information relating to players' psychological states. Attempts to objectively measure player emotion, for instance, have been largely supported by the use of physiological sensors (e.g., galvanic skin response [GSR] and electromyography [EMG]) [14, 42]. Such methods are limited by their susceptibility to signal interference, challenges in interpretation, and an inability to map directly from a given physiological response to a specific emotional state [14].

Mandryk and Atkins [43] created an expert system using fuzzy logic to assess player emotion based on physiological data. Performance of the resulting system was similar to a manual data processing approach when assessing users' emotional valence (i.e., pleasantness or unpleasantness) and arousal (i.e., excitement). These dimensions were then used by the model to provide a continuous classification of emotion, providing cogent labels for a users' psychological state. The research team proposed that such a system would dramatically improve the efficiency of experience evaluation, for example, by supplementing qualitative video analysis with a continuous timeline of emotional data from which notable features could easily be gleaned.

Extending existing methods of observation with AI can improve our ability to non-invasively measure nuances of user experience, such as moment-to-moment changes in a player's emotional state. ML has been extensively explored as a tactic for recognizing emotions based on facial expression, with many systems capable of real-time performance and modern approaches achieving mean accuracy scores of over 90% [44–46]. Speech recordings have also been used as a basis for emotion recognition, with proposed applications in the development of adaptive software interfaces [47, 48]. More recently, multimodal approaches to analysis have combined visual expression recognition with the processing of audio and gestural information [49].

These techniques hold promise in overcoming the aforementioned challenges of sensor-based methods. Firstly, expression recognition provides somewhat more of a direct mapping, capable of discriminating between emotions such as happiness and surprise or anger and disgust [45]. Furthermore, AI emotion detection can be noninvasive, for example, using a webcam recording of players' faces to infer emotional state rather than requiring players to wear electrodes. This characteristic is also advantageous in that it may be less prone to signal interference (e.g., such as the artifacts in EMG data caused by speaking [14]).

Roohi et al. [25] predicted user affect from webcam footage of players' faces using a deep neural network trained to recognize facial expressions. In particular, output from the network was used to create a "gradient" of changes in affect following in-game events (e.g., getting killed, defeating an enemy, winning the game). Results of this analysis were in line with the findings of past work using EMG to assess affect in response to game events. However, the researchers noted that, similarly to sensor-based approaches, classification of emotion through facial recognition is limited by lack of context and oversimplified categorization (e.g. a concentrated frown classified as sadness, or an ironic laugh classified as happiness). Nonetheless, the technique was noted as especially applicable to online or remote usertesting, where the use of sensor technologies is rendered infeasible. AI-driven affect recognition is still a relatively novel field of research; to our knowledge, use of this technology in the context of GUR is far from commonplace. More exploration of these techniques is warranted to determine whether they can effectively supplant or augment sensor-based methods for assessing player emotion.

Another method of note for the potential evaluation of user reactions is opinion or sentiment mining, whereby a language processing model is trained to extract high-level meaning or intent from written corpora [50]. In GUR, manual qualitative analysis of game reviews has been used to help investigate user experience, for instance, by identifying usability problems [51]. Manual coding of these data can be extremely labour-intensive and time-consuming, or even infeasible for larger datasets, such as collections of user reviews which may number in the hundreds or thousands. Pan et al. [52] explored the use of sentiment recognition models to achieve opinion classification across different domains, including user reviews of video games. This technique may be of particular interest in commercial GUR, where developers can have access to thousands of user reviews or social media posts reflective of player sentiment. AI-powered analysis techniques have the potential to reduce human error and drastically improve researcher efficiency in extracting insights from such content.

2.1.2 *Intelligent Data Analysis and Visualization*

Following the orchestration of playtesting sessions, researchers are left to sift through vast quantities of varied data—for example, in-game actions, comments, facial expressions, interview responses, and so on. A number of GUR software tools have been developed to facilitate the exploration and analysis of this data, many of which rely on visualization techniques [53].

Developing and working with visualization interfaces presents challenges including scalability and the integration of heterogeneous data (e.g., game events and physiological data) [54]. Cheong et al. [55] created ViGLS, a system for automating the creation of visual gameplay summaries based on logs of in-game actions. Using an action planner, ViGLS infers causal relationships between game events and applies heuristics based on viewer preference to select events for inclusion in a summary. Though ViGLS was proposed as a utility for players to review prior gameplay sessions, similar timelines and summaries have also been used in the visualization of data from playtesting sessions [56]. Automatic extraction and summarization of important gameplay arcs may be useful for GUR visualization tools, especially for games without a linear summary of events available *a priori* (e.g., open-world games).

Improving the utility of visualization and analysis tools has also been explored through the development of "intelligent" interfaces which provide varying degrees of configurability, assistance, or adaptation to user needs. Such work attempts to support the task of a human analyst by improving efficiency (e.g., by reducing clutter [57]) and accounting for user objectives (e.g., by suggesting a change in data representation suitable for a task such as comparison between time-series [58]).

Gotz and Wen [58] developed a rule-based adaptive visualization interface to recommend actions supportive of goals speculated from user interaction patterns. Similarly, Brown et al. [59] used several ML approaches to predict user behaviour and performance using a visual interface based on patterns of past interaction. This tactic was proposed as a potential basis for future work in the creation of an interface capable of adapting to inferred user characteristics, such as personality and task completion time. ML has also been employed in the prioritization of data points in a visualization based on predicted level-of-relevance to the user [60].

Beyond supporting the goals of a human researcher, AI has also been applied in the near-complete automation of certain analysis tasks. Southey et al. [26] presented SAGA-ML, a system using machine learning to analyze gameplay data generated through random sampling of designer-specified scenarios. SAGA-ML was used to identify potential player exploits and unintended behaviour in a commercial title, and presented as a basis for future work in the automation of gameplay analysis. Today, a growing body of work in GUR applies machine learning to the problem of large-scale data analysis and user modelling, as described in the following subsection.

2.1.3 *Player Modelling and Clustering*

Developing models to classify, describe, and predict user behaviour has been a longstanding challenge in GUR. Player typology attempts to answer this challenge by manually developing archetypes or trait-based theories to classify players into groups based on their gameplay habits and underlying psychological characteristics [61]. The earliest well-known example of a typology framework was proposed by Bartle [40], who described four archetypal players in online multiplayer games. Subsequent typologies have been largely reliant on expert knowledge of game design and psychology (e.g., [41, 62]). Despite the allure of these expertly-crafted models, they are subject to a number of pitfalls, including limitations in their generalization across game genres, a reliance on self-reporting for classification, and research divides in the field of psychology [61]. Of course, typological approaches are also heavily reliant on researcher insight, meaning that they are substantially time-consuming to produce and apply.

The classification of player behaviour in modern GUR is far from dependent on handcrafted typologies. The term player modelling generally refers to the study and creation of computational models, typically accomplished through the application of AI or statistical computation [63] (the related challenge of behavioural prediction is explored in Section 2.1.4). While some broader definitions of player modelling include typology-based approaches [64], here we focus on exploring the use of machine intelligence in the creation of player models (i.e., a "bottom-up" approach [63]). Over the course of approximately the past decade, apparent research interest in player modelling has grown significantly. Several applications of player models have been proposed and explored, including more believable game AI [63], adaptive difficulty adjustment [65, 66], and intelligent monetization systems [37, 63].

Developing a player model without an existing theoretical basis is ultimately a question of establishing relationships based on a collection of user data. This typically reduces itself to problems of classification and regression, making the challenge of player modelling particularly well-suited to machine learning and statistical analysis [63]. Furthermore, the use of automated approaches can be used to handle large collections of data from many hundreds or thousands of players [7, 24], infeasible for purely manual inspection. Several questions arise in tailoring these approaches to player modelling. What sort of data can be used—in-game behaviours (e.g., location, combat, deaths, etc.), play habits (e.g., session length and frequency), player reactions (e.g., facial expression, physiological measures)? At what granularity should data be analyzed? How can we interpret the meaning of a model generated with little or no human intervention? Researchers are in the process of exploring these questions through the development of several different techniques driven by clustering and machine learning.

In-game behavioural data, collected via telemetry, is commonly used as a basis for investigating player motivation and typology. Melhart et al. [67] used preference learning to demonstrate the use of gameplay data as a predictor of self-reported player motivation factors on a pre-established scale. Gameplay data has also been used as a basis for generating new classifications of player behaviour through clustering. Drachen et al. [68] present a comparison of several unsupervised clustering algorithms used to classify players based on character level and log-on patterns in the game *World of Warcraft* (Blizzard Entertainment, 2004). The authors propose that archetypal analysis, a soft-clustering method built upon the unsupervised identification of extreme behaviours, results in a favourable combination of interpretability and adequate separation of different player groups. This technique has also been used in the development of player models based on more game-specific telemetry data, such as player deaths, the use of special in-game abilities, and setting adjustments [7].

Players' in-game location can also serve as a foundation for this process. Spatial clustering is often focused on establishing partitions within the game world rather than the player population, helping designers to understand areas of importance in game levels and patterns of player behaviour in navigating those areas. Thawonmas, Kurashige, and Chen [38] explored landmark detection and the calculation of transition probability based on player location data. The authors further clustered players based on their movement patterns to create navigation visualizations representative of different player groups. More recently, Bauckhage et al. [69] explored several trajectory clustering techniques to create interpretable in-world partitions reflective of player movement patterns.

Different levels of granularity have been used in the development of datasets for behavioural clustering approaches. Aggregate data collected over the course of play (e.g., total shots fired, total number of deaths) has been used to identify groups of players with common high-level traits. Drachen, Canossa, and Yannakakis [24] applied self-organizing maps, a type of unsupervised neural network, to cluster players based on high-level gameplay data in *Tomb Raider: Underworld* (Crystal Dynamics, 2008). Lower-level data (e.g., specific action sequences rather than aggregate play statistics) has also been used to develop behavioural player models. Chen et al. [70] used sequential pattern mining and logistic regression to build a model associating in-game action sequences with player characteristics including game experience and personality traits. While the predictive power of the resulting model was only significant for game expertise, the researchers note that the use of lower-level data allows for the generation of more specific insights than aggregate metrics. Multilevel approaches have also been proposed, applying different clustering algorithms at varying levels of granularity to produce more complete player models [71].

2.1.4 *Behaviour Prediction*

A key challenge within user or player modelling is behavioural prediction, where a model attempts to foretell future user actions. The ability to predict player behaviour is promising even at a coarse level, with the potential to help developers create more adaptive experiences. Can we predict when users will quit or give up, to know when to encourage them? Can we predict what their strategy will be, so that enemy AI can become more cunning and lifelike? Can we tell what they are likely to purchase, so that we can create a personalized special offer? Researchers are in the process of exploring these and similar questions both in GUR and other domains, where user modelling has been applied to predicting search patterns [72] and purchase habits [73]. As discussed in Section 2.1.3, these models are typically trained on existing gameplay metrics, which can be sourced, for example, by mining gameplay data from a game's userbase [63].

Mahlmann et al. [35] used high-level gameplay data from several thousand players of *Tomb Raider: Underworld* (Crystal Dynamics, 2008) to predict player retention via expected total gameplay time and final level reached. In pursuit of this objective, the researchers explored several ML tactics to generate various predictive models, including logistic regression, decision tree induction, and artificial neural networks. The models achieved moderate accuracy when predicting the final level reached by players, though significant errors were incurred when attempting to predict completion time. Researchers experienced several challenges when working with commercial datasets, including noise introduced by missing information and unreliable interfacing of game and metrics software. Despite these obstacles, the team noted the potential of such models to help developers detect pain points and improve our understanding of player engagement.

Lower-level predictions, such as the inference of player objectives and strategy, have also been achieved through the application of various ML algorithms. Extensive work on player strategy prediction has been conducted using the real-time strategy (RTS) game *StarCraft* (Blizzard Entertainment, 1998) [74–76]. For instance, Weber and Mateas [76] achieved moderate accuracy in using regression models and decision trees to anticipate player strategy. The authors concluded that trained models could serve as a basis for improved strategy game AI agents, minimizing the need for human-authored behaviours.

The applications of recognizing player intentions extend beyond the creation of enemy AI. In nonlinear games, goal recognition has been proposed as a tactic for creating a more adaptive game experience, for example, by reminding users of their personal objectives after returning to play [77]. Such models may also be used to provide contextual hints, guide the behaviour of non-player characters (NPCs), or tailor the availability of resources, to name just a few applications. Goal recognition via the analysis of in-game action sequences has been achieved using several different methods, including input-output hidden Markov models [77] and long-term short-term memory networks [36].

Beyond the applications of AI in analyzing data from human players, there also exists the prospect of using AI itself as a stand-in for human participants in game testing, both for quality assurance and UX evaluation purposes. Work on the development of AI agents for game testing is relatively novel and less widespread when compared with the development of in-game or tournament AIs. The goals of these two areas are vastly different; though both almost invariably demand playing proficiency from the agents in question, the objective of AI developed for testing is to uncover issues with a game (e.g., bugs, unintended behaviour, impossible levels), rather than to provide an interesting opponent for players.

Several challenges are presented in the design and development of these agents; for example, the simulation of human-like rather than ideal play behaviour when attempting to predict UX issues, or the generalization of testing agents across multiple games or game genres. By exploring ways to overcome these challenges and develop robust testing agents, researchers aim to supplement traditional testing procedures with semi-automated solutions that promise increased efficiency and reduced human labour requirements.

2.2.1 *QA and Bugtesting*

Quality assurance (QA) is generally defined as the process by which software is inspected or tested to reduce the number of defects present upon release [78]. In this case, defects refers to bugs or logical errors in a system, rather than problems with its design or usability. In the case of games, QA issues might include, for example, physics bugs or glitches with a game's mechanics (e.g., enemies respawning incorrectly). It is important to emphasize that QA and UX testing are fundamentally separate pursuits; while QA aims to uncover technical defects, UX is focused on understanding how humans interact with and respond to a system. Though GUR is traditionally unconcerned with QA, research in its automation has significant technical and methodological overlap with the development of agents for usability and UX testing.

Automated testing of web software and productivity applications leverages a wide variety of techniques for systematically generating and executing test cases [79, 80]. Game QA poses unique challenges when compared with these domains; comparatively speaking, the possibility space presented by a game can be much larger, with far more complex modes of interaction. Current approaches to QA in games can thus rely on human testers manually filing bug reports, which can be expensive and time-consuming. Researchers have explored a number of solutions to this problem, including the use of runtime code monitoring for the detection of rule violation [81].

More pertinent to our discussion is the development of autonomous agents that function as testers themselves. The goal of such agents is to generate the same data that would be required from human testers in a particular context (e.g., an input log and accompanying gameplay footage), while saving the resources required to orchestrate in-person QA tests. Smith [82] describes the case of so-called trace samplers, autonomous agents which use several tactics (e.g., systematic enumeration of action sequences, random input combinations) to "interact" with a game in the same fashion that a large group of players might. The applications of such agents range from bug detection (e.g., the identification of invisible collision boxes) to playability evaluation (discussed in Section 2.2.2). Smith notes that an obstacle to the widespread use of samplers is the technical overhead incurred by game-specific solutions, suggesting that generalized or reusable samplers could help to further the development of more advanced intelligent design and testing tools.

Pfau, Smeddinck, and Malaka [22] developed ICARUS, an automated QA system for testing adventure games developed in the Visionaire[7] engine. The system uses a generalized reinforcement learning (RL) agent as a "player", offering automatic detection and reporting of crashes, rendering failures, and unsolvable game states supplemented with action logs. To mimic the conditions of real play, actions are simulated in real-time, with the agent achieving a total playtime comparable to a "speedrun" (e.g., completing the game as quickly as possible) executed by a human player. The researchers argue that, beyond fully automating the detection of game-breaking bugs, human QA personnel might use simultaneous observation of AI playthroughs to detect more nuanced issues (e.g., animation glitches) more efficiently.

The use of AI-created playtraces for bug detection has also been explored in more general game testing frameworks. Ariyurek, Betin-Can, and Surer [83] developed gameplay agents for automated bug-finding within the General Video Game Artificial Intelligence (GVG-AI) framework[8]. Agents are governed by RL, Monte Carlo tree search (MCTS), and inverse reinforcement learning to create variations of both "synthetic" and "humanlike" behaviours aimed at supporting QA testing in games. For the experiments described, bug-finding is framed as a problem of behaviour generation, with bugs detected automatically by verifying constraints for generated gameplay data. The authors found that gameplay from agents was competitive with that from human testers in terms of QA value for the context presented (e.g., encountered similar numbers of bugs), with agent playtraces encountering more bugs for some of the variants tested.

Machado et al. [23] created Cicero, an intelligent game design tool which provides users with AI assistance in developing and testing game prototypes. Like the agents described above, Cicero's design tools are based on GVG-AI, so as to facilitate generality. Its toolkit includes autonomous agents capable of general play

[7] https://www.visionaire-studio.net/

[8] GVG-AI includes a descriptive language which can be used to create a variety of games, and is used as a testbed for game AI: http://www.gvgai.net

using a combination of heuristics and a forward model (e.g., MCTS). Additionally, Cicero features Kwiri [84], a utility for querying agent gameplay data to assist in data analysis. Cicero's potential as a QA tool was evaluated by comparing the performance of users attempting to find buggy game objects by playing a prototype themselves versus observing an AI agent play the game. Multiple task variations were presented with differing gameplay contexts and types of bugs. In a debugging task based on the classic arcade game *Space Invaders* (Tomohiro Nishikado, 1978), AI-assisted users found significantly more bugs and made fewer errors, despite the fact that users without AI assistance were permitted an unrestricted number of play sessions. While this did not hold true for the other debugging scenarios presented, it speaks to the promise of AI agents as an efficient aid to quality assurance in game testing.

2.2.2 *Evaluating Playability*

Aside from assisting in QA tasks, AI agents can also help to evaluate game playability. Here we use the term playability as a measure of objective qualities about players' ability to complete a game or game segment. This is in contrast to user experience (UX), which attempts to understand subjective qualities. An evaluation of playability can answer questions such as, is it possible to finish this level given the player's current jump height? How long might we expect players to take? How many solutions exist to this puzzle? Can we estimate the difficulty of this level? It is important to note in differentiating between the approaches described here and those in the following section that the division between UX and playability evaluations is not a strict dichotomy. A question relating to user experience, such as whether a game is frustrating, can easily interact with a question of playability, such as whether parts of a game are too difficult or impossible to complete under certain circumstances. In playtesting with human participants, questions of playability are typically answered by having multiple users play through segments of a game to ensure an appropriate level of difficulty for the game's target market. Obvious playability problems can also be uncovered during the design process, as internal testing can quickly reveal issues with, for example, the physical possibility of completing a level. Regardless, judgements regarding playability are traditionally reliant on human labour and insight.

High-level evaluations of playability are generally objective—asking, for example, how many puzzle solutions exist—making them more straightforward to automate when compared with subjective inquiries (e.g., how frustrated players become). Several methods have been explored to reduce the burden on human labour in these cases, including AI player surrogates, logical inference based on a game's rules and level layout, and algorithms tailored to specific questions (e.g., flood simulations to check whether a level is continuous) [85]. Zook, Fruchter, and Riedl [86], proposed ML as a way to expedite the process of tuning parameters to balance game difficulty by training a design model on data from real playtesters.

Of particular interest to the eventual goal of replacing human playtesters is the use of AI "players" for similar purposes.

Automated playtesting agents have been extensively explored as utilities for empowering the design process, by allowing designers to test their creations quickly and inexpensively and supporting iterative development methods. Bell and Goadrich [28] created Cardstock, an automated testing framework for card games written in a custom descriptive language. Cardstock agents implement MCTS to simulate play, generating data used to investigate turn order advantages, game length, and possible player strategies. More recently, Keehl and Smith [87, 88] developed Monster Carlo, a MCTS-based Unity framework for gameplay simulation. The authors propose that Monster Carlo could be used to shorten playtesting turnaround when designers aim to investigate the impact of changes to a game's balance or ruleset on factors such as challenge.

Another general design framework integrating automated testing is Gamika [89], a prototyping tool for mobile games. Gamika includes a configurable AI testing agent that designers can customize by specifying "tactics", or pattern-action pairs defining typical gameplay. Designers can make changes based on observed agent performance and the effectiveness of different available tactics, as well as orchestrating automated runs that attempt to investigate the results (e.g., expected play time) of tweaking game parameters within a certain range.

AI testing tools have also been created for game-specific use cases, tailoring AI agents to operate under the specific constraints of a given game. Agents based on a modified version of the A* pathfinding algorithm were used to exhaustively test gameplay trajectories in *The Sims Mobile* (Maxis Redwood Shores, 2018), with the aim of identifying balance issues for different in-game careers and relationship progression [90]. Another example is Ropossum, a procedural level generation tool equipped with an AI agent capable of verifying level validity (i.e., whether it is possible to complete a given level) [29]. Based on the mechanics of the popular physics puzzler *Cut the Rope* (ZeptoLab, 2010), Ropossum uses a rule-based agent to evaluate possible actions via tree search. This playability check is used in combination with an evolutionary procedural content generator to autonomously create a variety of playable levels. Smith, Butler, and Popovic [27] similarly use a rule-based system to verify the integrity of level designs in Refraction, an educational puzzle game. The Refraction generator has also been extended to include a system for automatically sequencing a progression of levels, found to be similar in performance to human-authored difficulty curves [91].

Automated playability validation has also been proposed in the creation of so-called mixed initiative design frameworks, which integrate computer intelligence as an assistant to human creative labour. Tanagra, for instance, is an AI-assisted design framework for creating two-dimensional platforming levels [92]. Tanagra combines a pattern-based level authoring system with automated search-based playability verification and procedural level generation to effectively facilitate collaboration between computational intelligence and a human designer.

Morai Maker is another example of a level design tool which has been extended to support mixed-initiative design, or so-called "co-creation" between a human and an AI assistant [93]. The tool allows users to create platforming levels based on the classic /emphSuper Mario games. In the mixed-initiative variant of Morai Maker, a variation of the A* algorithm is used to check for reachability between regions of a level and provide a coarse evaluation of difficulty. Platforming games in general often serve as a testbed for the development of AI player agents [94]. In some cases, these agents have been further developed to imitate aspects of human behaviour and individual play characteristics, as discussed in the following section.

2.2.3 *Behaviour Simulation and UX Evaluation*

At the frontier of AI playtesting is the effort to craft agents that simulate aspects of human decision-making, perception, and individual variations in behaviour. In the pursuit of more sophisticated testing AI, we strive to make not just "agents that play", but *agents that play like humans*. Notably, there exists a fair amount of overlap in technology and application between agents meant to evaluate playability and those meant to imitate humans; perhaps it is best to think of this distinction as more along a continuous spectrum rather than a hard boundary to avoid confusions arising from terminology. Nonetheless, in this category we consider agents that help to answer questions dependent on the qualities of the human player, not just the game itself. Instead of asking whether a level is possible, we might ask, what path will a cautious player take through this level? Where could players become disoriented, or frustrated? What will they find most interesting?

We have previously proposed the simulation of human play characteristics as a tool for identifying usability issues and unintended behaviour throughout an iterative design process [2]. By imitating the motivations of a particular player, designers may be able to gain a more complete understanding of how different players experience a given game, by reviewing the behaviour of a large population of AI agents tailored to mimic specific groups of players (e.g., experienced but cautious players versus inexperienced aggressive players). However, this area of work is still largely in its infancy. Roohi et al. [95] present a review of work on simulating intrinsic motivation in game testing agents, finding that current applications are sparse and that some qualities of motivation, such as immersion, remain largely unexplored.

Borovikov et al. [96] present a comprehensive overview of agent-based testing in games, with a focus on machine learning methods for the generation of humanlike behaviours. The authors use commercial case studies to investigate the potential of agent-driven testing, including the use of RL agents to evaluate the pace of progression in a mobile game, with the intent of validating intended playtime and player engagement. They conclude that the task of probing player experience is both important to providing insight for designers and deepening our understanding of game interactions in general.

Of particular interest to our current work is navigation through a virtual world, an interaction which is fundamental to the core experience of many video games. Consequently, imitating human behaviour in the traversal of these worlds has been the subject of significant work investigating the use of AI for playtesting (or integration of AI "test simulations" into a design workflow). Borovikov and Beirami used ensemble Markov models trained on human gameplay data to generate training data for a neural network meant to emulate human-style play in an open-world shooter game [97]. Becht and Bakkes [98] combined a behavioural classification model with inverse reinforcement learning to predict differences in trajectory between players with different roles in a competitive game. Awareness of a player's gameplay goals was also used as the basis for a pathfinding tool developed by Tremblay et al. [39] to provide a probabilistic representation of player movement through levels in a stealth game. The tool produces a predicted heatmap of players' positions, allowing designers to verify if their expected or intended trajectory is likely without necessitating the involvement of human testers. A different approach to humanlike navigation was taken by Tomai, Salazar, and Flores [99], based on existing human playtraces. The authors used probabilistic sampling to generate "humanlike" agent paths given a set of landmarks, suggesting that machine learning might also be applied in the reproduction of "human" pathfinding behaviour.

Different technical approaches may be taken in the mimicry of human behaviour, such as imitation learning (IL), where an agent learns by training itself on the behaviour of an "expert" (in our case, a human) [100]. Hybrid approaches combining IL with reinforcement learning have also been proposed for the generation of humanlike game behaviours [101]. The authors found that hybrid agents demonstrated higher performance than agents trained using IL alone while exhibiting subjectively more "human-like" behaviour than those trained using RL alone. While these methods are obviously of interest to the subject at hand, the idea of learning gameplay behaviours from humans has also been applied in the creation of more believable in-game AI, such as the so-called Drivatars from the recent *Forza Horizon* games (Playground Games, 2014 and 2016) [102]. Many of the qualities that make human opponents interesting are arguably vital to the simulation of more realistic user behaviour for playtesting. Humans generally play suboptimally, are prone to mistakes, may wander off of their own accord, and so on. Furthermore, and perhaps most importantly, there is a great deal of individual variation between human players, arising from differences in psychology, game preferences, and past experience, for example. Simulating this variation is crucial in the development of truly versatile humanlike testing agents.

As mentioned in Section 2.1.3, the classification and description of player behaviour has been studied through both typological and computational modelling approaches. Our understanding of individual play differences established through this work can be applied in subsequent efforts to reproduce this phenomenon in

AI player populations. Emulation of human strategy in board games has been explored using both ML agents trained on human play records [103] and rule-based systems meant to reflect strategic archetypes observed in real players [104]. Ortega et al. [105] compare hand-coded, supervised, and unsupervised learning to simulate human playing styles in Infinite Mario Bros, a public domain variant of the classic platforming game *Super Mario Bros* (Nintendo, 1985). In this case, evolutionary unsupervised learning achieved the best performance, both through measured path similarity with a human player and through subjective assessment of believability by human judges (i.e., a Turing test).

Most work in the area of behaviour simulation has been concerned with the technical questions surrounding reproduction of human characteristics, rather than explicitly leveraging this capability for game and UX evaluation. One such case is presented by Liapis et al. [106], through the use of AI "personas" to judge level design quality in a dungeon crawler game. Controlled by single-layer neural networks, these personas are evolved to play in the fashion different player archetypes (e.g., achievement-oriented, combat-focused, etc.). The agents "critique" levels after simulating play using a scoring function specific to each persona considering events that occurred during the play simulation. Later work used an updated version of the game testbed and agents powered by a modified MCTS to improve computational performance and support more complex level mechanics [30]. The use of AI personas is suggested as an efficient means of automated playtesting when acquiring human feedback is infeasible.

The AI persona concept has since been extended for the testing of Match-3 games by Mugrai et al. [107]. For this application, evolutionary algorithms were used to create MCTS agents mimicking different human play-styles. Agents were developed to simulate behaviours ranging from score minimization (emulating novice play) to the long-term strategization of more experienced players. The authors suggest that automated playtraces could be used to estimate level difficulty for the purposes of game balancing.

Tools developed for both simulation-driven testing and data analysis tasks hold promise in improving researcher efficiency and improving developers' ability to validate their creations even in the absence of suitable participant groups. However, these tools present unique research challenges of their own. Their creation demands solving complex problems such as generalized modelling of game mechanics, human behaviour prediction, and vision-based processing. Furthermore, many of these tools break new ground, as it were, often with little precedent in established frameworks and heavily abstracted inner workings. This brings about a vital question to their eventual success: How can we validate the operation of these tools and assess their utility?

2.3 Evaluating AI GUR Tools

For any piece of software to be successful in a practical use context, its evaluation is of paramount importance. Concerns regarding performance, usability, and utility can be investigated with a variety of methods, depending on the nature of the system under review. In the particular case of AI-driven GUR tools, little precedence exists for evaluation of the novel systems created by researchers. Furthermore, these tools present unique challenges in selecting and designing appropriate evaluation methods. For example, how do we define system accuracy for a game testing framework, when we want agents to display at-times erratic or "incorrect" play patterns meant to reflect human behaviour? How can we effectively design plausible scenarios to measure the impact of AI assistance on the workflow of designers and game researchers?

The sections that follow explore these and other questions related to validating the effectiveness of AI GUR frameworks. Informed by an understanding of HCI methods and existing evaluation efforts for the work described throughout this chapter, the following subsections describe how different dimensions of system effectiveness can be validated.

2.3.1 *Computational Performance*

Many AI systems can be computationally intensive to run; in this sense, there can be a trade-off between requirements for human and machine labour. The resultant costs in time and processing power can help gauge a system's practicality, improvements over past work, and accessibility to developers with average or limited computing resources. Depending on the nature of system tasks, these costs may vary greatly; a system simulating a single gameplay agent in real time will likely have lower processing requirements than a high-resolution facial expression classifier operating on thousands of video frames, for example.

When developing a tool intended for broader use, it is important to provide a clear picture of system performance and scalability. For ML systems, the time required for both training and execution (e.g., classifying an example, simulating an agent playthrough) should be considered. Scalability should also be evaluated with respect to performance—in the case of analysis tools, how does the system respond to a much larger dataset? Game-specific qualities may also be of interest, for instance measuring the impact of a game's branching factor on the processing time of simulated agents (e.g., [28]). Ideally, benchmarks should be conducted on different hardware configurations to provide an impression of performance under "average" conditions. Specific techniques for benchmarking calculations are not specific to the context of GUR, and are beyond the scope of this review. For a complete overview of these methods, the reader is referred to existing work in the software design and evaluation literature such as that presented by Everett and McLeod [108].

2.3.2 *Accuracy and Quality of Output*

An important consideration in validating AI tools is the accuracy of system output. Not all frameworks will be suitable for evaluation of "accuracy" per se, however. In the case of an intelligent visualization system, for example, it may be more helpful to classify behaviour as helpful rather than correct. This is a question of usability rather than concrete accuracy, and is thus more appropriately investigated through the measures described in Section 2.3.3. Nonetheless, an evaluation of accuracy, as defined by the fraction of system output that properly reflects some ground truth, is applicable to the majority of AI GUR tools. Broadly speaking, we can categorize these tools in two main groups: agent-based systems (e.g., agent testing tools described in Section 2.2), and non-agent-based systems (e.g., analysis tools described in Section 2.1). The purpose and output of these two types of systems is vastly different; consequently, techniques for evaluating their accuracy differ as well.

Many non-agent-based systems ultimately solve classification problems (e.g., facial expression recognition or the inference of player strategy). With supervised learning, a labelled dataset is already available for validation, and new examples may be introduced to assess the generalizability of the model developed. In these cases, it may be sufficient to start by expressing overall accuracy as a ratio between the number of correct results and the total number of results produced. However, for non-binary classifiers, a single measurement omits a great deal of information (e.g., the distribution of error across different outputs), and a more complete description of the system's performance should be provided, such as a confusion matrix (e.g., [25]). Other measures, such as precision and recall, may also be used for further insight into a system's correctness and classification ability. These techniques are described extensively in the machine learning literature, for example, by Alpaydin [109].

Predictive algorithms may be assessed in a similar fashion as those for classification, when real outcomes are available for verification. Nominal predictions (e.g., which purchase a user will make next) may be treated identically, while continuous quantitative predictions can be assessed in terms of relative error. An example of such assessment is used by Mahlmann et al. [35] in the prediction of player retention discussed in Section 2.1.4. The evaluation of "correctness" for unsupervised models is less straightforward. Beyond the subjective assessment of whether output is useful or makes sense, it may be desirable to quantify the quality of system output, especially when comparing different approaches. Clustering algorithms, for instance, may be assessed based on the distribution and comparability of clusters [110]. Generally, this and similar assessments can be limited by the opacity of ML algorithms; while it may be straightforward to quantify a system's accuracy, it can be difficult or impossible to trace the origin of errors. Recent research in visualizing ML algorithms (e.g., latent space visualization) may be used to supplement these evaluations to improve depth and understandability (e.g., [111, 112]).

Agent-based systems are radically different in that their output generally consists of action logs, win rates, path traversals, and other gameplay artifacts. For agents intended to test play scenarios for bugs or playability, accuracy of behaviour may be analogous to competence in play. This may be evaluated in a variety of ways, for example, human review of gameplay traces, win rate against humans or other agents in competitive games, or completion times compared with optimal or "expert" play for a subsample of play scenarios. In systems where agents' sole purpose is to determine whether or not a level is playable, output may be treated in the same manner as a non-agent-based binary classifier, by providing a human-labelled subsample for validation. For tools where these agents are used to create or sequence designs, their output may be compared with human-authored content (e.g., [91]). Lastly, gameplay agents may benefit from methods in the automatic generation of test cases for navigation agents described in the AI and robotics literature [113]. To our knowledge, such techniques have yet to be applied in the evaluation of AI GUR frameworks.

Defining "correct" behaviour is far more challenging for agents meant to emulate the decision-making and idiosyncrasies of human players. Since actions objectively identifiable as gameplay mistakes may be desirable in such cases, assessing the quality of generated behaviour is far from trivial. Two main approaches exist for evaluating the "humanness" of agent behaviour. First, if playtraces from humans are available, comparisons between human and agent play can be conducted. For instance, navigation traces may be compared through visual similarity or measuring distances between traversal waypoints (e.g., [99, 105]). Gameplay decisions may be validated by checking for agreement in the actions taken by humans versus AI players placed in identical play scenarios (e.g., [114]). These approaches are limited by the challenge of recruiting participants representative of sufficiently diverse play-styles, as well as their tendency to penalize behaviour that is feasible but deviates from available human samples. Another method less sensitive to this effect is the Turing test, where humans observe gameplay and attempt to discern whether it originated from a human player or AI agent [105, 115, 116]. While this exercise is highly subjective, it can provide an indication of believability for complex behaviours where other forms of evaluation are difficult or infeasible.

2.3.3 *Usability*

It may sound odd at first to perform usability testing on tools which might be used in a usability evaluation themselves, but this step is vital for any utility intended to move into a broader use context. A tool can only be used effectively if it is straightforward to understand and operate. Even though many of the frameworks described operate largely autonomously, users must still be able to configure them, navigate their interfaces, and interpret their results. A usability evaluation for any tool should take into account the needs of the target user—thus, in the case of GUR utilities, tests should be conducted with designers, developers, or researchers where appropriate. Before designing the test, it is important to have a clear idea of key questions that the evaluation should investigate. For instance, can non-programmers understand the output of this clustering algorithm? How easy is it for first-time users to customize the behaviour of gameplay "persona" agents? Do users find the tool intimidating to use? These questions can help in guiding the design of the test session and in recruiting a group of participants well-suited to the specific insights required.

Many of the tactics already employed in GUR to evaluate games (or more broadly, in HCI to evaluate interactions with other software and hardware systems) can be used to assess the usability of a research or design tool. External expert review, for instance, may help to refine a system's interface, the presentation of output, or the general flow of user interaction. Once a usertest is planned, different methods before, during, and after interaction with the system can be used to identify usability issues and key strengths. Pre-session interviews may help to provide context for a participant's past experience with similar tools or their typical workflow before exposure to a new system. During a session, the think-aloud method can provide insights to a user's decision-making process and identify points of confusion as they arise. Skill-check interviews during a session might be used to probe user understanding. After a session is complete, a questionnaire can be administered to capture a structured impression of key system qualities across all users. Lastly, semi-structured interviews can be conducted to give participants an opportunity to reflect on their experience and express their opinions and concerns. An extensive review of these methods and their use in usability testing is given by Drachen, Mirza-Babaei, and Nacke [5].

2.3.4 *User Performance Benefit*

Though many of the same usability testing techniques apply, the impetus for evaluation of a GUR tool is far different from that of a digital game. Generally speaking, a game has no high-level formal requirements apart from being enjoyable (serious games aside). However, a design or research tool must benefit the workflow of the user in some way to provide value. Beyond being usable, it must also be useful. Questions to explore this characteristic might include whether the tool reduces the time needed to complete a task, if the tool improves the quality or quantity of insights gained, and if the tool helps to minimize human error.

There is some overlap, to a degree, between evaluation of a tool's benefit to user performance, and its accuracy (see Section 2.3.2) or usability (see Section 2.3.3). However, a tool may produce accurate output but have poor usability, or conversely good usability but high rates of error. Furthermore, even if an AI tool produces accurate data and is straightforward to use, it may not provide a tangible benefit to the user. Ideally, the usability and accuracy of the tool should be evaluated, and the results of such evaluations acted upon, before attempting to assess its usefulness. It is important to understand how the interplay between the user and the system in question contributes to accomplishing (or failing to accomplish) the goal which prompted its development. Evaluations of this nature in any field should treat the relationship between users and AI systems as collaborations, rather than attempting to isolate each party and evaluate their performance in a vacuum [117].

Designing scenarios to assess a tool's utility should reflect the expected "real-world" use of the system as much as possible. For instance, an AI-assisted debugging tool should be tested with a game prototype known to contain bugs, as demonstrated in the evaluation of [23] (see Section 2.2.1). Evaluation of an open-ended design tool should give users enough time to explore the creative space and assess the extent of the tool's capability. For instance, Powley et al. [89] conduct a preliminary evaluation of Gamika (described in Section 2.2.2) with an extended case study comprising the design of two game prototypes. At any rate, scenarios should be conducted comparatively where possible, resources permitting, to establish a baseline for productivity with a user's existing workflow. Additionally, before running scenarios with the target user group, it is important to consider whether users will need any additional support to use the tool effectively. In an agent-based system, for instance, users may require a replay function to skip back and forth through a "recording" of gameplay in the same manner they might for human players (an example of such a system is provided by Machado, Nealen, and Togelius [118] with SeekWhence).

After a test is conducted, procedures for assessing results will vary depending on the nature of the tool and the test conducted. Measurements of interest for quantifiable tasks (e.g., identifying bugs in a game prototype) may include the number of work items accomplished (number of bugs found), the time taken, and any errors made (writing down a non-existent or incorrectly described bug). Assuming a comparative scenario design has been employed and appropriately counterbalanced, results can be contrasted with those of users working without the AI tool under study. Qualitative performance improvements, such as the "goodness" of levels created with the assistance of AI playtesters, can be more difficult to define. In these cases, it may be important to consider a subjective component of the evaluation - does the user feel as if the tool is useful or improves their workflow? For certain utilities this quality in itself may provide value to the user, for example, if the tool inspires confidence or creativity.

2.3.5 *Other Considerations*

When considering the "real-world" utility of a GUR framework, it is important to acknowledge and criticize characteristics beyond formal testing. Generalizability, for instance, is a defining factor in determining whether a tool will gain widespread use. Evaluating this quality is a matter of considering its ability to be reapplied outside of the original use context. A game-specific tool has no or low generalizability (e.g., AI playability evaluation for platforming levels with fixed mechanic/object definitions), whereas a context-agnostic tool has high generalizability (e.g., AI playtesting framework with configurable inputs and mechanics capable of adapting to any first-person shooter game).

Depending on the nature of the tool, software design quality may be of interest as well. If a lower-level tool, such as a code library for playtesting agents, is meant for large-scale use, its success will likely depend partially on developers' ability to extend and customize functionality beyond the original scope. If such use is intended, qualities such as code modularity and understandability should be assessed, for example, via expert review. Shehory and Sturm [119] provide a set of heuristics for evaluating agent-based modelling techniques, based on more general software design criteria proposed by Ardis et al. [120]. Several examples of similar methods for evaluating system design exist in the software engineering literature.

Another consideration is the feasibility and resource efficiency of the tool in question. If it requires any specialized hardware (e.g., physiological sensors) or software systems (e.g., metrics collection) to function, it is rendered less accessible to smaller development teams with limited budgets. Resource efficiency is also of interest: does the tool's function demonstrably reduce time, labour, or financial requirements for the quality of work completed? Many frameworks are developed with commercial applications in mind; as the field continues to move forward, such questions may become more pressing. Potential time savings, cost analyses, and a salient explanation of any trade-offs associated with tool use (e.g., sacrificing the depth of playtesting feedback from humans to perform more quick, iterative tests with AI) are all important factors in the assessment of tool practicality.

2.4 SUMMARY

The investigation of AI as a means to supplement existing methodologies and develop new research tools has become a subject of substantial interest in GUR. Such applications have been explored both in supporting data analysis and in simulating the process of game testing itself. Techniques such as facial or voice recognition and sentiment analysis can be used to enhance user observation (e.g., [25]). During analysis, intelligent interfaces have the potential to help compensate for researcher errors and support efficiency in analysis tasks. When working with larger datasets, machine learning can help to cut through the noise and make identifying patterns in player behaviour a feasible endeavour (e.g., [24]).

Within the realm of game and UX testing, AI can be used in part to fulfill roles previously reliant on purely human labour. Near-autonomous systems have been developed for quality control and playability evaluation [22, 28]. Frameworks for verifying playability through gameplay constraints and play simulation have also been deployed as tools for active use during the design process [29, 92]. The more challenging task of predicting player behaviour has also been explored, for inferring players' in-game goals [121], supporting the level design process [39], and criticizing level designs from the perspective of different player types [30]. Though the evaluation of such tools presents its own unique challenges, they are already proving themselves valuable in the hands of developers and UX researchers.

The GUR landscape will continue its evolution as more AI-driven tools are developed and explored. With each new tactic brought into the mainstream, professionals and academics alike benefit from access to new approaches that make the course of innovation and discovery replete with even more opportunities.

Part II

THE PATHOS FRAMEWORK

An empathic approach to simulating navigation

3 SYSTEM DESIGN AND DEVELOPMENT

This project was motivated by common pain points in games user research: the pressure and time restrictions imposed by development timelines, the costs associated with hosting human participants, and most importantly, the challenge of recruiting representative user groups.

Players and developers, generally speaking, are ultimately aligned in their goals. In vastly oversimplified terms, people play games to have fun, and developers try to create games which provide fun and engaging experiences. However, gaps in understanding from developer to player can lead to detrimental effects on the end experience. The entire impetus of playtesting is to bridge this gap, to test what works and what does not, and to ensure that an experience fulfills the needs and desires of its target audience.

The caveat imposed by appealing to a particular audience is that participants recruited for playtesting should obviously reflect the characteristics of this audience as accurately as possible. However, this is often somewhat overlooked out of necessity, limited by a lack of resources and participant availability. The result is that playtesters are not always as representative of end users as they should be.

In some cases, only a slight divergence may be present. Testing a competitive multiplayer title with both novice and veteran users, excluding only the uppermost tier of professional tournament-goers, may well be acceptable to gain an understanding of game experience across all skill levels.

In others, the dichotomy between player and playtester can be more exaggerated, restricting or distorting the insights that can be gained from user studies. Take, for example, an educational game intended for young children. Recruiting children as playtesters can be challenging for a myriad of reasons including legal and ethical concerns, and timing sessions to work with the schedule of both parent and child. However, while testing with the more readily available adult population may be able to reveal some glaring issues, such a scenario fails to capture the unique cognitive characteristics affecting children's ability to enjoy and learn from a game.

Pondering this quandary and others like it led to a question of whether current trends in automation and artificial intelligence could help put these issues to rest—can we simulate variation in user behaviour, reducing the time required to "playtest" a game and cutting out the pain of recruitment entirely?

Of course, the idea of completely replacing humans in usertesting is still far away, at least for now. This project is intended to serve as a first step, acting as a complement rather than a substitute for existing methods.

3.1 SYSTEM OVERVIEW

PathOS is an all-inclusive, lightweight tool for assisting game developers by acting as a predictor of user behaviour. The framework is intended to support designers as a playtesting surrogate of sorts, providing a rough estimate of how players would navigate through a game's world.

These predictions are meant to support the work of designers in a few different ways. First, they facilitate a direct comparison between the intended or ideal path through a level and the projected path of real players. Moreover, they can help to estimate the impact of changes, allowing designers to compare multiple alternatives of a level, helping to optimize world design in advance of testing with real users. Additionally, they can help predict rate of engagement with different entities in a game's world, identifying potential hotspots or deadzones of player activity.

The tool provides users with end-to-end support for simulation-driven testing of game levels, from customization of AI behaviour, to watching agents navigate in real time, to recording and visualizing behaviour for later analysis. The design and implementation of these subsystems, along with the front-end interfaces provided to the user, are explored throughout the remainder of this chapter.

The target users of the PathOS system are practicing level designers in the games industry, particularly those working on small development teams with limited resources and no existing tools for semi-automated testing. Secondarily, depending on the scope of a given team or project, user research professionals may also use the framework as a preparatory step before testing with human users to provide feedback to level designers.

In contrast to existing tools and AI testing approaches, which are often highly game- or genre-specific and frequently propietary or largely commercial endeavours (e.g., [22, 24, 26, 30]), PathOS is open-source and designed with generality in mind.

We can summarize our goals in creating PathOS with four key design objectives:

- **Ease the burden of playtesting:** Provide designers with an approximate solution to playtesting early in the development process with much lower labour requirements than human usertesting.
- **Developer accessibility:** Make the tool free and open-source, so that resource limitations do not prevent developers from accessing the tool.

- **Designer usability:** Make the tool easily usable and understandable for designers, rather than requiring manipulation of a technical back-end to extract any real value.
- **Generalizability:** Make the tool adaptable to a variety of different projects, so that it can be helpful to as many creators as possible.

3.1.1 *Design Process and Justification*

In developing PathOS, one of our core aims was to ensure that the tool could integrate well into users' existing level design workflows. Thus, we decided early on to develop the framework as an extension for an existing game engine, rather than a separate utility. This prevents putting an unnecessary burden on users to install and learn tools foreign to their design environment, as well as removing the requirement to have a standalone game build available for testing.

Functionally speaking, our primary motivation was to address the difficulties associated with recruiting players who represent a game's target audience. We wanted to ensure that AI agents could easily be customized to reflect a variety of playing styles, based on differing player motivations and levels of gaming experience.

To achieve the goal of predicting navigation behaviour for different player types, we considered using a machine learning model trained on data from playtesting sessions with human users. Similar models have been used in prior work to derive player types from gameplay data [24] and predict aspects of player behaviour [35]. However, due to the volume of training data required and the limited generalizability of such models across different titles, we opted instead to create an expert system based on existing knowledge of player motivation and gameplay behaviours. In particular, we employ a novel planning algorithm which uses a simple model of player motivation to select destinations for an agent to target based on its knowledge of the game's world. This approach is similar to other planning algorithms used in games, such as goal-oriented action planning [122], though we are focused solely on navigation, and are not concerned with developing AI to be used for in-game characters, but rather game testing. Each part of the algorithm used is detailed in the remaining subsections of this chapter.

PathOS agents attempt to predict human navigation behaviour by using a simplified model of player decision-making in games. This takes into account the gathering of information (perception), retention and retrieval of information (memory), and the selection of navigation targets. In other words, agents "see", "remember", and "think". This loosely emulates the process by which human players acquire and process information about a game's world (e.g., by looking around) before deciding on their next action.

The decision-making process itself is highly variable depending on the motivations of a given player, and it is this variation which we are most interested in capturing for the purposes of modelling a given target audience. For instance, experienced and efficient players might approach a level by heading straight for the objectives needed to complete it. By contrast, a curious player with a completionist attitude might meander around the game world collecting objects before trying to "finish" the level. To ensure agent behaviour reflects these differences, the framework allows users to adjust agent motives (e.g., aggression, curiosity, efficiency, etc.) in accordance with different player types.

To inform the development of the agent logic described above, we conducted a survey of literature surrounding player behaviour, game design, and human cognition. Obviously, deriving a "perfect" model of human gameplay behaviour is currently an impossible challenge, both from a computational and design perspective. Therefore, our goal in conducting this review was to distill the available knowledge into a model which provided an adequate approximation of human gameplay logic while maintaining understandability and transparency. Insights gained from this review are explored in Sections 3.2 and 3.3, alongside a detailed outline of agent logic.

In addition to the agent logic itself, the framework also encapsulates functionality for the customization of agent profiles, orchestration of simulated testing runs, and the review of data collected from agents. These features have been integrated into a front-end user interface, so that the framework can be used in a game development project with no need for additional programming work. We developed the user interface for the tool iteratively, with the primary goals of ensuring understandability and accessibility for new users. Our interface design is explored in-depth in Section 3.4.

Together, this collection of features forms an end-to-end solution for agent-driven testing, comprising the current prototype of PathOS.

3.1.2 *System Prototype*

The PathOS prototype has been created as an extension for Unity, a freely available commercial game engine. We chose Unity due to its popularity, especially among independent game creators, our core target user base for this framework. Unity's CEO claims that approximately half of all digital games are powered by the engine[1].

Many games across different genres have been developed with Unity, including AAA titles such as *Hearthstone* (Blizzard Entertainment, 2014) and *Fallout Shelter* (Bethesda Game Studios, 2015). Several notable games created by independent developers, or "indies", have also been created using Unity, with titles like *Ori and*

[1] https://techcrunch.com/2018/09/05/unity-ceo-says-half-of-all-games-are-built-on-unity/

the Blind Forest (Moon Studios, 2015), *Subnautica* (Unknown Worlds Entertainment, 2018), and *Slime Rancher* (Monomi Park, 2016) powered by the engine. Unity's flexibility and generality across game genres was another key factor in its selection as a platform for the development of this tool.

PathOS is written in C#, with front-end interactions using Unity's Editor API and integrating into the existing Editor interface in Unity. All low-level functionality (e.g., checking agent field-of-view, pathfinding along terrain) is abstracted where possible and offloaded to Unity's physics and rendering systems. This was done to ensure that the tool would be lightweight, and require as little modification to a game project as possible. For instance, rather than requiring designers to instrument a level with some custom collision logic, we simply tap into Unity's native representation of level geometry.

The PathOS prototype is suitable for scenarios in which players navigate a 3D environment, a common feature of many game genres (e.g., first-person shooter, action-adventure, role-playing game, etc.). The current iteration of the tool is limited in this respect, in that it focuses on navigation in a world with a relatively defined "ground plane". Expansion of the tool to better support verticality in level design is left as a task for future development.

Before using PathOS to test a level or world layout, designers need only ensure that they have enabled Unity's navigation mesh ("navmesh"), a physics-based representation of level geometry used for pathfinding. Many projects will have this functionality already enabled, as it is typically used to facilitate pathfinding for any AI characters in-game.

With these conditions met, the framework prototype can be used to predict player behaviour in levels of arbitrary size and layout, allowing designers to help verify the expected user experience of their creations. First, designers mark up their level with a simple utility allowing them to tag various objects according to their in-game function. Then, they customize one or more AI agents to meet the desired user profile. AI behaviour can be observed in real time or accelerated for faster simulation, and recorded for later review. A designer-facing perspective of this workflow is illustrated in Section 3.4.

Before moving on to understand the end-to-end operation of the framework, the sections that follow detail the model used by agents to simulate player behaviour in the game world.

3.2 AGENT PERCEPTION AND MEMORY

The core purpose of PathOS agents is to predict player navigation. In other words, they decide where to go, based on the available alternatives. To make such a decision, the agent must have some representation of the game's world; in simple terms, it must know where "things" are located, and how to get there.

Broadly speaking, we can divide information about a game level into two categories. First, spatial information such as boundaries, terrain, obstacles, walls, and so on. Second, semantic information about the game objects, or "entities", in a level. For instance, where interactive objects are located, and what purpose they serve in-game (e.g., enemy, collectible item, etc.). Before an agent can decide where to go, we must address the challenge of how an agent should represent and access this information.

One approach to this problem would be to simply give the agent a complete representation of the level and allow it to select destinations based on this. For instance, we could provide agents with a list of all interactive game elements in addition to the navmesh. In this case, the agent has immediate access to all level data.

If our goal was to achieve optimal play, such a solution would be acceptable, even desirable. Omniscient agents could simply skim through a list of level elements, identify those necessary for level completion (e.g., mission markers, exit), and ignore other interactions. However, we are concerned instead with approximating the behaviour of human players navigating in 3D space. Not only is the decision-making process of humans variable and often "sub-optimal" (e.g., prioritizing fun over efficiency), but human players generally do not have immediate and perfect access to level information.

Consider a game involving navigation in a 3D world, such as a first-person shooter. Before players decide where to go, they must gather information about the game's world by observing their surroundings. From the first-person viewpoint of their character, players must look around and explore to gather information about a level's layout and the entities (e.g., enemies, ammunition, power-ups, health packs, etc.) contained therein. To navigate effectively, players must remember a level's layout and the location of game entities as they move through a game's world.

Depending on the specifics of a game's interface, some aids might be provided to the player in this respect. For instance, many games provide some form of map to players, showing a level's spatial layout. Generally, these maps are "filled in" as players move around, or require some form of interaction with the world to unlock information about various areas. For instance, in *The Elder Scrolls V: Skyrim* (Bethesda Game Studios, 2011), city maps are filled in as the player walks around. Information about specific entities can also be made explicitly available to assist players with important game objectives. To continue with our *Skyrim* example, players are given a "quest journal" which can be used to mark objectives on the in-game map as well as the in-game heads-up display (HUD).

The need to discover and remember information is a key differentiator between the ability of a human player and an omniscient AI agent to make navigation decisions. Even if an agent's motivations could be matched exactly to a human player, the limits on the information available to a human player could result in drastically different behaviours.

An omniscient agent mimicking an aggressive player could visit every hostile entity in a game's world immediately. However, a human player would first need to explore the level looking for targets. They might miss certain enemies which are difficult to see from a given vantage point, forget about previously seen encampments while exploring a new area, or simply become lost. All of these events are of interest to a level designer, as they may indicate design issues which could negatively impact player experience. For instance, though a designer might intend to have some "secrets" in a level, if most players will never find a power-up meant to help them succeed, then their experience will be unintentionally frustrating. Thus, this inherently imperfect process of players gathering and representing information is a key factor in estimating their behaviour.

To approximate the task of player perception and recall, along with its imperfections, PathOS agents use a simple model of sight and memory to gather and store information about the game environment. First, agents "see" the game world, accessing information about visible game entities and level layout. Then, this information can be committed to memory, where it is subject to decay and imperfect recall. It is the information contained in an agent's memory, rather than a complete "perfect" representation of the game's level, which is then used to make a decision. The remainder of this section details the operation of agents' perception and memory models; how this information is used to control agent navigation is explored in Section 3.3.

3.2.1 *Agent "Sight"*

Every agent is equipped with a camera, functioning as the agent's "eyes". This camera mimics players' view in-game, and can be adjusted to match the settings of a particular game's camera by adjusting parameters such as field-of-view angle and clipping planes. Information visible to the agent can be transferred to the agent's memory and used to make navigation decisions.

Determining what the agent can "see" is accomplished by interfacing with Unity's physics and rendering subsystems. Following the earlier established categorization of level information, we are concerned with determining the visibility of both spatial data (e.g., map boundaries) and entity data (e.g., the type and location of game entities).

Spatial data is conveyed by querying Unity's navigation mesh from the agent's current POV, casting rays along different sight lines. This provides the agent with an indication of the "explorable space" in all directions. Additionally, rays' intersection with level geometry gives the agent information regarding the location of obstacles and map boundaries. Together, this information is used to build a "memory map" of the level's spatial layout, explained further in Section 3.2.2. This map is used both in the selection of potential exploration targets (see Section 3.3) and in route planning.

With respect to game entity data, we make the assumption that players can generally infer an object's in-game function so long as it is in their field-of-view, based on its appearance and the game's context. Typically, game creators make use of visual features such as colour, shape, and iconography, in conjunction with contextual cues, to convey the function of game objects. For instance, a healing item may be placed near an enemy encampment and made to resemble a first-aid kit.

To allow agents to make similar distinctions, we provide a tagging system for designers to mark game entities with their in-game function. The categorization of entity functions used is informed by a review of the game design literature and discussed further in Section 3.3, with the user interface developed for this process detailed in Section 3.4.

Tagged entities are flagged as visible based on a simple visibility check, which determines first if the entity falls within the frustum of the agent's camera, and then uses raycasting to check for occlusions by other objects. Visible entities can be transferred to the agent's memory of game entities, explained in Section 3.2.3. Entities available in memory can then be selected as navigation targets for the agent. The process of selecting destinations is discussed in Section 3.3.

It should be noted that this physics-based approach to determining what agents can "see" is an approximation of vision which fails to capture certain features and nuances of human perception. For instance, it does not consider the impact of colour, luminance, or shape contrast in determining the visibility of an object. A human player, for example, may not be able to detect a poorly lit object at a distance, even without any physical obstructions blocking the object from view. Furthermore, this model does not account for the difference in visual attention between central and peripheral vision, which may cause users to miss objects depending on their gaze focus on-screen.

Accounting for these subtler perceptual factors could be theoretically be achieved through a more sophisticated system built on computer vision. In fact, agents learning to "play from pixels" have already been explored as a means to generalize game AI, for instance, among classic arcade games [21]. However, training such a system to generalize between the aesthetic styles and entity appearance of arbitrary games is currently infeasible, and inflexible to unique design choices. Furthermore, such an approach is currently too computationally demanding, since our system operates in real-time atop the game environment. Nonetheless, the implementation of a more robust vision system incorporating consideration for factors such as visual contrast presents an interesting opportunity for future work.

3.2.2 *Agent Spatial Memory*

An agent's internal representation of level layout is maintained in the form of a "memory map". While the "mental map" model of human navigation has largely

SPATIAL CODE	DESCRIPTION
Unknown	Assigned by default when no information is available.
Seen	Indicates a clear tile which has been visible via line-of-sight.
Explored	Indicates a clear tile which has been traversed by the agent.
Obstacle	Indicates a non-traversable tile that has been observed by the agent through a line-of-sight raycast (e.g., level border, obstacle).

Table 1: Codes used to fill in the agent's memory map. This information is used to determine which parts of the map are traversable and have been visited.

been eschewed in favour of the idea that humans navigate based on landmarks [123], this representation was chosen for its understandability and resemblance to information available to players in-game. The mental map can be rendered on-screen to help designers understand agent logic (see Section 3.4), and is reminiscent of the "mini-maps" common in 3D games.

The memory map is maintained as a grid-based representation of level geometry with adjustable granularity to fit the world scale used by the level designer. As the agent looks and moves around the level, information from raycasts along Unity's navigation mesh is used to populate the map with information. Each square in the map's grid is coded with one of the identifiers given in Table 1.

Information in the memory map is used to evaluate potential opportunities for exploration by the agent's decision logic (see Section 3.3). Additionally, it is selectively used by agents in route planning when travelling to a given destination.

Low-level agent movement is handled via Unity's NavMeshAgent system. Each PathOS agent has a NavMeshAgent component, allowing them to navigate automatically within the bounds of a game's movement restrictions (e.g., movement speed, collision detection, etc.). However, after selecting a destination, higher-level route planning (pathfinding) can follow one of two alternative protocols.

The first protocol simply uses Unity's automatic pathfinding along the navigation mesh, which will have the agent take an optimal path avoiding any obstacles in the way. Since agents are only able to target locations or entities they have observed, this was deemed as an acceptable approximation of player wayfinding in general. However, depending on a level's layout, relying solely on navmesh pathfinding could take the agent through previously unseen and unexplored territory. For instance, if the agent is able to see a landmark at a distance, but no straight unobstructed path is visible, pathfinding along the navigation mesh could take the player through unexplored territory.

To account for the fact that some players may backtrack along known paths to reach a destination, or prefer to stay in "safe" explored territory out of caution (see Section 3.3), we also implemented the ability for agents to wayfind based on the contents of their memory map. When wayfinding from memory, agents use a simple A* algorithm to navigate from their origin to their destination, with penalties for hazardous areas calculated based on the agent's motivation profile (see Section 3.3) and the contents of the agent's entity memory (see Section 3.2.3).

To better replicate the manner in which humans store spatial information, this system could be extended to support landmark detection, perhaps in conjunction with a computer vision-based approach as noted in the previous section. Additionally, it could be specialized on a per-project basis to reflect game-specific location semantics, such as differing terrain types, to provide a more robust routing system.

3.2.3 *Agent Entity Memory*

The second component of an agent's memory is comprised of game entities. The memory of an entity stores two key pieces of information: its location in the level, and its type, as denoted by the aforementioned level markup/tagging system. Agents do not have a perfect memory; instead, the information contained therein is subject to mutation, deletion, and recall penalties, to mimic the limitations of human memory.

Agent memory is governed by a series of operations meant to loosely imitate the procedures of human information storage and recall, and is informed by existing research in cognitive science. To illustrate this process, let us explore how the agent's representation of a game entity evolves from entering field-of-view, to retrieval from long-term memory.

VISIBILITY AND TRANSFER TO SHORT-TERM MEMORY. In humans, "iconic memory" represents a very short-term, high-fidelity representation of what is currently visible [124]. From iconic memory, elements (e.g., images, objects) can be transferred to short-term memory on a sub-second timescale [125, 126]. To mimic this process, entities are only transferred into short-term memory after they remain in an agent's field of view for a short amount of time. Thus, entities visible for less than a few hundred milliseconds (e.g., during rapid camera movement) will not be registered in memory.

SHORT-TERM MEMORY STORAGE. In humans, memory is accepted as being divided into short-term and long-term memory [127]. Short-term memory (STM) is volatile, temporary, and has a fairly limited capacity, whereas long-term memory (LTM) is permanent and stores a much larger amount of information.

After initial transfer to STM, the memory of a game entity is augmented with an indication of the time elapsed since the entity in question was last observed by the agent. In humans, STM has a limited capacity, which is typically expressed in terms of "chunks" representing cohesive pieces of information [128]. For our purposes, we consider each game entity as a chunk, and limit the capacity of agent STM to 3-5 entities in accordance with research on the capacity of human STM [128, 129]. The actual size of an agent's STM varies along this range based on their simulated level of game experience, configurable by the user. This is based on the logic that, as a player is more proficient with games, they become better at managing and recalling game information.

Entities in STM which are currently visible cannot be forgotten, since their information is directly visible and thus available to the agent. For entities which are no longer visible, agent STM is limited to the capacity noted above, and memories are discarded above this limit according to the time elapsed since their formation. In other words, the entities which have been unseen for the longest will be forgotten first, in accordance with the idea that human memory decay is largely dependent on the time since formation [130, 131].

TRANSFER TO LONG-TERM MEMORY. Humans transfer information from STM to LTM by rehearsal [127]; essentially, mental repetition or repeated exposure. To mimic this process, once in agent STM, entities can be transferred to LTM depending on the length of time they are visible and the number of times they have been seen. Since we lack a precise model of human visual attention (see Section 3.2.1) and do not attempt to simulate players' manual rehearsal of game information to memorize it, this is only a coarse approximation. After meeting a visibility time threshold on the order of several seconds, or having been registered in field-of-view on several separate occasions, entities can be transferred to agent LTM, which has unlimited capacity.

Once in LTM, entities can still be forgotten if they remain unseen for an extended period of time, again reflecting temporal memory decay. Again, this period of time scales along an interval according to the agent's simulated level of gaming experience. An exception to this rule is when a designer marks an entity as "always known", effectively noting that players would always have access to an entity's location (e.g., via a mission marker or quest journal). Such entities are persistent in LTM from the beginning of a game session, even if they have not been seen, and cannot be forgotten.

RECALL. When recalling a memory, there is a mental cost on the order of several milliseconds associated with retrieving the information [127]. As the amount of information retrieved increases, so too does the amount of time needed to recall it. To mimic the effect of this factor on player's decision-making, the time taken between an agent's successive re-evaluation of all available destinations in a level is scaled according to the number of entities currently present in memory. The result is that an agent will re-evaluate its options and "decide" more quickly if it has less information to remember.

Human recall is also subject to imprecision, or "noise" in the storage of sensory information [132]. To imitate this characteristic, the agent's memory injects noise into the remembered location of entities which are not currently visible. This effectively means that agents will estimate, rather than precisely recall, the location of unseen entities.

In addition to game entities indicated by the level designer, the agent's memory also maintains a short list of potential "exploration targets" (See Section 3.3). The capacity of this list is limited to the same size as agent STM. Whenever a new exploration target is evaluated by the agent, it is added to this list. The list's capacity is maintained by continually removing the targets evaluated as least promising by the agent; the rationale being that the most interesting targets would be those remembered by the player.

While this model serves as an approximation of human memory, it has been simplified out of necessity, both for computational reasons and in the interest of keeping design complexity feasible and transparent. Future work could explore the augmentation of this model taking into account phenomena such as the need to "focus" memories retrieved from LTM and the association of related memories increasing memory capacity [128], as well as a more complete model of memory decay. Nonetheless, for our purposes, which are focused on the approximation of player navigation, this model serves as the basis for determining the information available to agents when selecting in-game destinations.

3.3 AGENT DECISION-MAKING

The navigation of PathOS agents is based on a continuous loop of selecting a destination from available alternatives. To simulate the cost of recall in humans, the delay between successive re-evaluations is dependent on the number of entities in the agent's memory, as discussed in the prior section. There are two types of destinations evaluated by the agent, each corresponding to part of the agent's memory: entity targets, and exploration targets.

ENTITY TARGETS. Entity targets represent the agent explicitly travelling to a game entity registered in the agent's memory. The agent's destination is set to the location of the entity in-world if it is selected as the current target. Each time available destinations are re-evaluated, every game entity in the agent's memory is considered as a potential target. However, once an entity has been visited by the agent, it will not be considered as a target again; this is based on the assumption that most game encounters (e.g., completing a mission, killing an enemy, consuming a pickup) are one-time interactions.

EXPLORATION TARGETS. Exploration targets represent spatial exploration, and do not have an associated game entity as the final destination. Exploration targets are based on an origin point (e.g., the agent's current location) and a direction. If selected as the current target, the agent's destination is set to the farthest location that can be reached by heading out in a straight line from the origin point in the given direction. This location is ascertained by performing a raycast on the navigation mesh. Exploration targets are generated each time destinations are re-evaluated by using the agent's current location and a series of outward vectors evenly spaced apart. Exploration targets existing in the agent's memory (see Section 3.2.3) are also evaluated as potential alternatives.

Each of these target types is scored based on the logic described in Section 3.3.3. At the end of the evaluation process, the target with the highest score is selected as the agent's current destination. Target evaluation is based on a model of player motivation informed by our review of the literature in game design and player typology. This forms the core of PathOS agent logic and ultimately depends on two categorizations of game entities and motivations respectively, explored in the following subsections.

3.3.1 *Game Entity Types*

Game entities are classified into nine categories defined by their semantic meaning and potential interactions in-game. Designers mark level objects as belonging to one of these categories before starting an agent simulation using our level markup tool (see Section 3.4). This classification is based on existing frameworks for design patterns proposed by Bjork and Holopainen [133] and the formal elements of games as described by Fullerton [134]. Furthermore, these categories reflect game world interactions associated with known player motivations (see Section 3.3.2).

It is impossible to claim that such a system will be able to cleanly delineate all entities within a game of arbitrary design and mechanics. However, these categories were chosen as a general representation suitable for the majority of design scenarios. Our interface for level markup allows designers to tag entities with no appropriate categorization to avoid confusion during the testing and review process. Additionally, adding new entity types can be done programmatically to accommodate specialized scenarios, if desired.

The nine entity types are as follows:

OPTIONAL GOAL. This tag describes an objective marker which is not necessary for the completion of the level. In-game, optional goals may help the player work towards some achievement, acquire desirable rewards such as powerful items, or simply discover more of the game's narrative content.

General Examples:	Optional mission markers, puzzles, time trials.
Game Example:	Sidequest markers in *The Elder Scrolls V: Skyrim* (Bethesda Game Studios, 2011).

Refer to: *Objective* [134], *Optional Goals* [133], *Goal Points* [133]

mandatory goal. In contrast to optional goals, mandatory goals are those which must be interacted with in order to complete the level. All mandatory goals (if present) must be visited before the agent can "interact" with the final goal (below), if one is present in the level.

General Examples:	Main mission markers, mandatory puzzles.
Game Example:	Main quest markers in *The Legend of Zelda: Breath of the Wild* (Nintendo, 2017).

Refer to: *Objective* [134], *Committed Goals* [133], *Goal Points* [133]

final goal. The final goal is used to trigger the end of the level, and indicates a completion milestone in-game (specifying such an entity is, as with all other entity types, optional). Once visited by an agent, the simulation can optionally terminate automatically if configured to do so by the user. Due to this characteristic, the agent's evaluation of the final goal will always take into account the agent's estimation of benefits from other objects which remain unvisited in the level. That is to say, if an agent is incentivized to keep exploring the level, it will avoid the final goal until such incentivization has been reduced (by interacting with other objects).

General Examples:	End-of-level teleporter/door/gate/etc., main quest ending marker.
Game Example:	End-of-level flag poles in *Super Mario Bros.* (Nintendo, 1985).

Refer to: *Objective*, *Outcome* [134]

enemy hazard. The enemy tag is used to indicate a hazard which would incite a combat scenario in-game if provoked. This represents a danger which requires active effort on the part of the player to overcome, and poses a potential threat to the well-being of their character.

General Examples: Monsters, enemy soldiers, hostile characters.
Game Example: Skeletons in *Dark Souls* (FromSoftware, 2011).

Refer to: *Enemies*, *Agents*, *Combat* [133]

environmental hazard. This type of hazard is one that does not require the player to participate in combat, but is rather environmentally-based. Such areas have the potential to harm the player's character on contact or prolonged stay in the area.

General Examples: Poisonous plants, bear traps.
Game Example: Lasers in *Portal 2* (Valve Corporation, 2011).

Refer to: *Deadly Traps*, *Obstacles* [133]

collectible ("achievement resource"). A collectible is an in-game item which can be picked up by the player and most often serves no direct immediate benefit to their character. However, the accumulation of such items may lead to unlocking game content or achievements, or simply grant the player a score of some sort.

General Examples: Stars, trinkets, and other "flavour" items.
Game Example: Moons in *Super Mario Odyssey* (Nintendo, 2017).

Refer to: *Resource* [134], *Objective* [134], *Collecting* [133], *Pick-Ups* [133]

self-preservation ("essential resource"). These are in-game items which can be picked up to increase the player's chances of surviving, by providing some sort of necessary resource. Collection of these items contributes positively to the player's well-being.

General Examples: Ammunition, health packs, power-ups.
Game Example: First-aid kits in *Left 4 Dead* (Valve South, 2008).

Refer to: *Resource* [134], *Power-Ups* [133], *Resources* [133]

POINT-OF-INTEREST. A POI indicates an environmental feature, landmark, or setpiece providing some form of visual interest or message to indicate potential curiosities in the area. The value of such an entity is largely exploration-based.

General Examples:	Statues, buildings, altars.
Game Example:	Setpieces in *Don't Starve* (Klei Entertainment, 2013).

Refer to: *Exploration* [133]

NPC. A non-hostile, non-player character who may be interacted with to discover narrative content, obtain items, start a sidequest, and so on. (Hostile non-player characters are classified as enemies instead.)

General Examples:	Townspeople, friendly creatures, AI allies.
Game Example:	Villagers in *Animal Crossing: New Leaf* (Nintendo, 2012).

Refer to: *Characters*, *Agents* [133]

A tenth additional "null" tag is available for designers to flag interactive objects that do not fall into these categorizations. The scoring of these entities is not affected by agent motives (i.e., agents will not be drawn to them), and so this can be used to avoid confusion in wondering why a particular area was not visited.

It should be noted that in the current prototype, these entity types are static; that is to say, they do not change throughout the course of the level, regardless of where the agent travels. This was done in the interest of reducing complexity, and avoiding the guesswork associated with attempting to simulate interaction with a game's mechanics while preserving generality across genres and game scenarios. However, future work could explore the extension of this system to support more dynamic interactions. For instance, triggers could be created on objective markers to "unlock" the interactivity of other objects, changing their type from null to optional objective.

With the types of game entities defined, there is one other system component instrumental in determining how goals are evaluated—agent motivations.

3.3.2 *Agent Motives*

Each PathOS agent has a motivation profile comprising seven distinct player motives based on existing theories of user psychology and player typology. Each of these motives can be adjusted on a normalized scale to create a unique agent profile, and can be used to recreate classic player archetypes (e.g., the Bartle player types [40]). Additionally, an experience slider can be used to improve the agent's memory capacity and storage, as discussed in Section 3.2.3. The notion of "experience" in this context is similar to the notion of skill-based player characteristics (e.g., *Resourcing*, *Speed*, *Decisiveness*, *Control Skill*) discussed by Cowley [62].

The seven motives driving agent behaviour are as follows:

CURIOSITY. Curiosity represents a player's drive to explore the game world and see all it has to offer, uncovering secrets and traversing as much of the map as possible. Curious players are also interested in uncovering more information about a game's lore and narrative, and are always on the lookout for new content.

Example behaviour. Drawn to Points-of-Interest (POIs) and Non-Player Characters (NPCs). More likely to select exploration targets. Looks around frequently to search for new entities and parts of the map to discover. Less likely to route based on spatial memory (See Section 3.2.2), favouring autogenerated navmesh paths which may take the agent through unfamiliar territory. Less likely to complete a level's final goal if much of the level remains unexplored.

Refer to: *Explorer* [40], *Seeker* [41], *Curiosity-Cognitive* [135], *Immersion-Discovery* [136], *Adventurer* [137]

ACHIEVEMENT. Players driven by achievement seek to earn in-game rewards and receive credit for feats of skill, endurance, or even sheer luck. They want to feel accomplished and perhaps praised for their in-game actions.

Example Behaviour. Drawn to Goals of all types and Collectibles, in pursuit of special achievements.

Refer to: *Achiever* [40], *Achiever* [41], *Optimisation* [62]

COMPLETION. Related to achievement, completionists seek to fill out every tick box in their quest journals and entry in their bestiaries. They want to find every object there is to find, so long as there's some sort of progress bar or list they can consult to see that shiny "100%".

Example Behaviour. Drawn to Goals, Collectibles, and most other interactive entities (Hazards, POIs, NPCs) in pursuit of filling out every completion metric the game has to offer.

Refer to: *Completionist* [138], *Thoroughness* [62]

AGGRESSION. Aggression represents a player's desire to exert dominance over the game world, primarily through combat. Aggressive players are frequently looking to pick a fight, throwing caution to the wind and charging down enemy encampments.

Example Behaviour. Drawn to Enemies and a smaller extent Environmental Hazards, will not shy away from hazardous areas when routing a path based on spatial memory.

Refer to: *Killer* [40], *Aggression* [62], *Conqueror* [41], *Competitor* [138], *Brawn* [139], *Mercenary* [137]

ADRENALINE. Similar to aggression, adrenaline is associated with daredevil-type behaviours, but moreso drawn to environmental hazards and challenges, rather than enemy encounters. Players with high adrenaline motivation are thrill-seekers and take pleasure in surviving dangerous scenarios.

Example Behaviour. Drawn to Environmental Hazards and to a lesser degree Enemies, calculates lower penalties for traversing hazardous areas when routing from memory.

Refer to: *Daredevil/Survivor* [41], *Daredevil* [137], *Challenge* [135]

CAUTION. Opposing danger and prioritizing their well-being, players motivated by caution will be less likely to charge into hazardous areas, especially unprepared. They are always on the lookout for danger, and tread lightly if placed into such situations.

Example Behaviour. Drawn to Self-Preservation items and repelled by Hazards of any kind. More likely to route paths based on memory ("backtracking"), applying harsher penalties to hazardous areas when routing from memory.

Refer to: *Caution/Resourcing* [62]

EFFICIENCY. Efficient players care about finishing a level as quickly as possible, finding mandatory goals and then getting out. They do not care for sticking around to find everything in the game's world; instead, their primary motivation is to simply complete the game's objectives.

Example Behaviour. Heading straight for Mandatory and Final goals.

Refer to: *Speed* [62], *Achievement-Advancement* [136], *Mastermind* [41]

These motives form the core of an agent's destination selection logic, used to score individual alternatives based on their appeal to the agent's motivation profile. This process is explored in the following section.

3.3.3 *Destination Evaluation*

During the agent's evaluation of destination alternatives, each potential exploration and entity target is assigned a score based on the agent's motivation profile. The motivation profile consists of a series of values unique to the agent representing the strength of each of the seven motives in the agent's decision-making process. Each of the motives can be assigned a value on the interval [0, 1] by the designer.

To evaluate potential alternatives within the context of the game environment, a scoring (weight) matrix W defines the relationship between each motive dimension and game entity type. This matrix is global and referenced by all agents, and can be customized by the designer if desired to reflect the needs of a specific project. For instance, if combat difficulty is low and poses little risk to players, the negative association between the Caution motive and Enemy entity type could be reduced.

The default values assigned to the matrix, based on the categorization of motive-specific behaviours outlined in Section 3.3.2, are given in Table 2.

Note that rows of the matrix have a cumulative weight of 1 when summed across all motives. This is done such that an agent with equal motivation across all categories would score game entities equally regardless of their type, representing "agnostic" player behaviour.

The final score of a given destination candidate (exploration or entity target) is given by the following equation:

$$score = entity\ score + bias \tag{1}$$

BIAS. The Bias term represents the "inherent value" of the destination itself, and is calculated differently for exploration and entity targets.

For exploration targets, the initial bias term is zero.

For entity targets, the bias is calculated as the sum of all scores in the scoring matrix *W* (see Table 2) for the row corresponding to the entity's type, with each element scaled by the corresponding value of the agent's motivation profile. For instance, the bias for an entity target with the POI type would be calculated as:

$$entity\ bias = 0.9 \ast Curiosity + 0.1 \ast Completion \tag{2}$$

More generally, we can express this as the dot product of a column vector containing values for each of the agent's seven motivations, and a row vector taken from the scoring matrix corresponding to the entity's type:

$$entity\ bias = M \ast W_i \ast d \tag{3}$$

The entity bias is scaled by a factor d, which is proportional to the inverse square of the distance to the target, to allow the agent to prioritize objectives closer to its current in-game location. Furthermore, entity targets are granted a slight additional "interactivity bonus" over exploration targets. This bonus is only added to the entity's score if it is otherwise deemed a favourable target for the agent.

Any target is also given an additional bias if it is already the active destination target of the agent. That is to say, though agents will continually re-evaluate available alternatives when en route to a given destination, that destination is given a slight preference to incentivize finishing the current trajectory. A new destination will only be chosen once the current destination is reached, or if a significantly better alternative is evaluated while en route.

In addition, any target type is given an additional positive score contributor based on the number of "unexplored" tiles in the agent's memory map that would be crossed if travelling in the desired direction and how far the agent could travel along that trajectory. The magnitude of this contribution is scaled by the agent's Curiosity motive.

ENTITY SCORE. The Entity Score term is calculated based on the agent's current location and the direction to their potential destination. It represents the desirability of taking a direct path to the destination, based on the entities that the agent could encounter as a result. When calculating this term, all entities contained in the agent's memory are considered, ignoring those that have already been visited and thus have no new value or pose no new threat to the agent.

$$entity\ score = \sum (M \ast W_i \ast d) \tag{4}$$

The terms M and W_i are the agent's motive vector and a row taken from the scoring matrix according to the type of each entity considered, in the same fashion as the *entity bias* term described in Equation 3.3.3. Their dot product is scaled by the factor d, a term inversely proportional to the square of the distance between

	curiosity	achievement	completion	aggression	adrenaline	caution	efficiency
Null	0	0	0	0	0	0	0
Optional Goal	0	0.5	0.5	0	0	0	0
Mandatory Goal	0.1	0.2	0.2	0	0	0	0.5
Final Goal	0.1	0.2	0.2	0	0	0	0.5
Collectible	0	0.5	0.5	0	0	0	0
Self-Preservation	0	0	0	0.1	0.1	0.8	0
Enemy	0.1	0	0.2	1	0.7	-1	0
Environment Hazard	0.1	0	0.1	0.3	1	-0.5	0
POI	0.9	0	0.1	0	0	0	0
NPC	0.9	0	0.1	0	0	0	0

Table 2: Default values for the motive-entity scoring matrix.

the entity and the agent's location. The factor *d* is further affected by the angle between the agent's potential trajectory and a vector drawn from the agent to the entity; for entities behind the agent, this factor will have a value of zero, for instance.

As stated in Equation 3.3.3, the final score of a target is taken as the sum of its calculated *bias* and *entity score* terms. Thus, the desirability of each destination is not only dependent on the potential value of arriving at the destination itself, but the collection of encounters an agent might expect to have along the way. This ensures that agents consider the game world as a connected system, rather than a list of disparate entities. For instance, when confronted with one path leading to just one collectible item, and another leading to a trove of several, a completion-oriented agent will first prioritize the path containing more rewards.

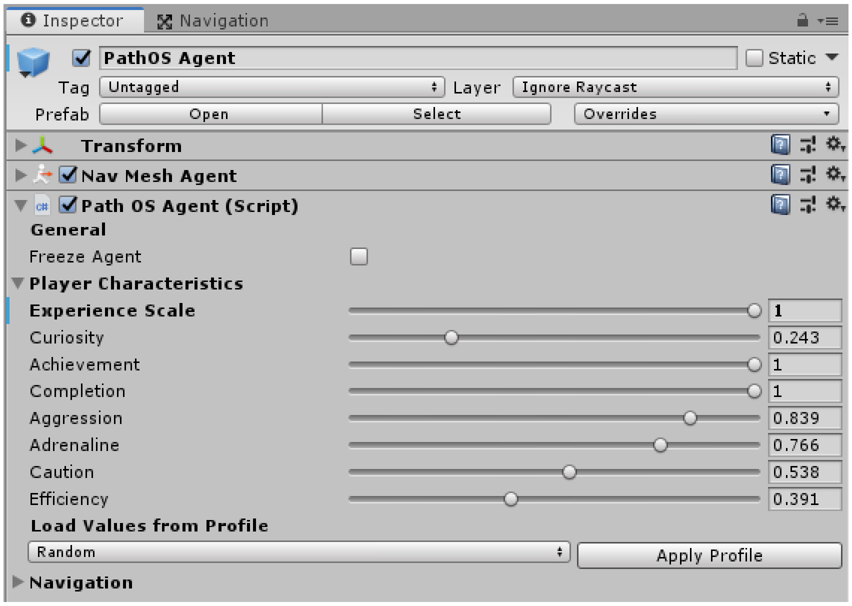
Throughout the destination re-evaluation process, the agent will update its target destination to the current alternative, if the current alternative is evaluated as more favourable. A slight degree of stochasticity is injected into this process to mimic the imprecise nature of subjective player decision-making. For instance, if two alternatives score equally, one will be chosen at random. As the difference between their scores is exaggerated, the likelihood that the higher-scoring alternative will be chosen increases rapidly and reaches certainty after the difference exceeds a given threshold.

Once a destination has been confirmed as the agent's current target, it is routed towards that destination based either on automatic pathfinding along Unity's navigation mesh or via routing on the agent's memory map, as noted in Section 3.2.2. This process of re-evaluating available destinations, selecting a navigation target, and moving throughout the game world forms the core behavioural loop of agents during simulated testing.

Given this understanding of how agents navigate the game world, we can move on to explore how users can interact with this system to perform simulated testing of candidate level designs.

3.4 USER INTERFACE AND DATA VISUALIZATION

As we intend for the tool to be used primarily by level designers, our focus is on front-end interaction through a graphical user interface (GUI) abstracting much of the functionality discussed in the prior sections. However, we also want the tool to be as flexible as possible, and as such, it is open-source to allow for back-end modification to suit individual projects. For instance, new motives and game entity categorizations can be added with relative ease. Such additions could be made, for example, to mimic game-specific playing styles (e.g., stealth versus melee combat) or create special categories for game-specific items (e.g., separating out different types of collectibles).



☑ PathOS Agent

□ Static

Tag: Untagged

Layer: Ignore Raycast

☑ Nav Mesh Agent

☑ Path OS Agent (Script)

General

Freeze Agent □

Player Characteristics

Experience Scale	1
Curiosity	0.243
Achievement	1
Completion	1
Aggression	0.839
Adrenaline	0.766
Caution	0.538
Efficiency	0.391

Load Values from Profile

Random

Figure 1: The Inspector interface for customizing PathOS agent behaviour. GUI elements allow users to tweak agent profiles and navigation characteristics.

Turning our attention to the user-facing front end of the tool, the PathOS user interface has been created to integrate as seamlessly as possible with existing UI conventions in Unity's main level and scene creation interface, the Unity Editor. PathOS scripts are condensed into Unity components which can be dragged and dropped following the conventions of existing Unity script and game object components, such as colliders and character controllers. Modifiable properties (e.g., agent motivation profiles, simulation time limit, etc.) are exposed through Unity's Inspector, a mainstay of the Unity interface. An example of Unity's Inspector interface for PathOS agents is shown in Figure 1.

By using the existing design language of Unity's Editor, we aim to ensure that PathOS will have a reduced learning curve and pose little interruption to designers' existing workflow while offering value in return. A complete enumeration and explanation of all PathOS interface elements can be found in Appendix A, the user manual for the PathOS tool. In this section, we will briefly explore the tool's user interface from setting up simulated testing to analyzing data from agent testing runs.

LEVEL MARKUP. First, designers instrument the Unity scene with markup indicating the types of entities present in the game level. To facilitate this process, the tool provides a point-and-click interface for selecting entity types and tagging game objects. First, the designer selects the desired entity type from a palette of labelled icons. Then, they can move through the scene, clicking to tag objects highlighted under the cursor as having a particular type. Screenshots of this process are shown in Figure 2.

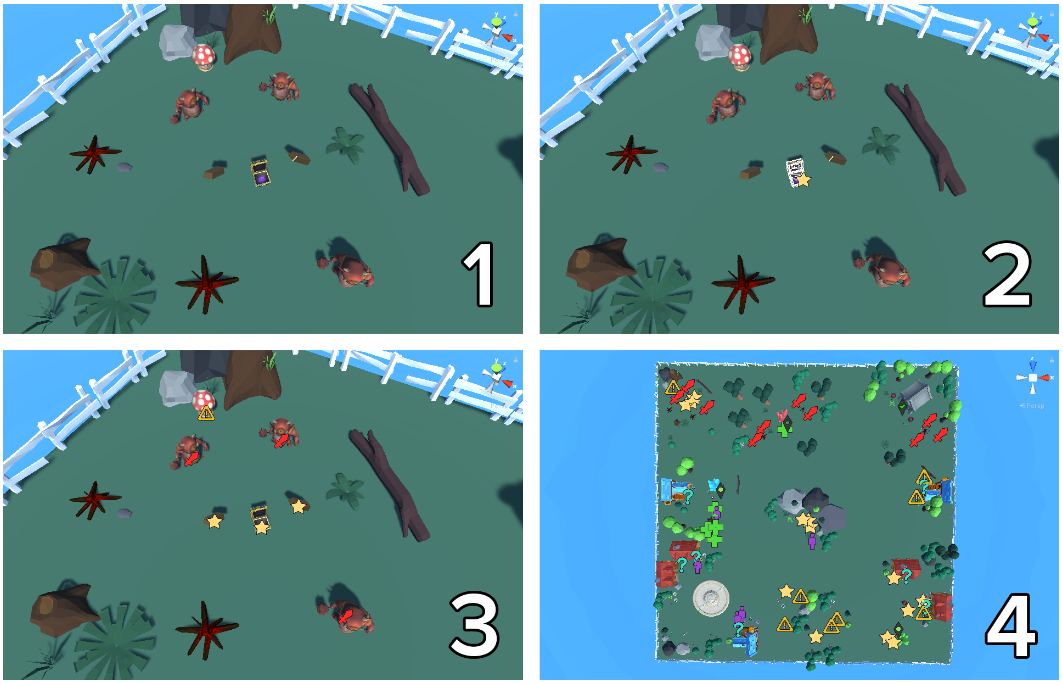


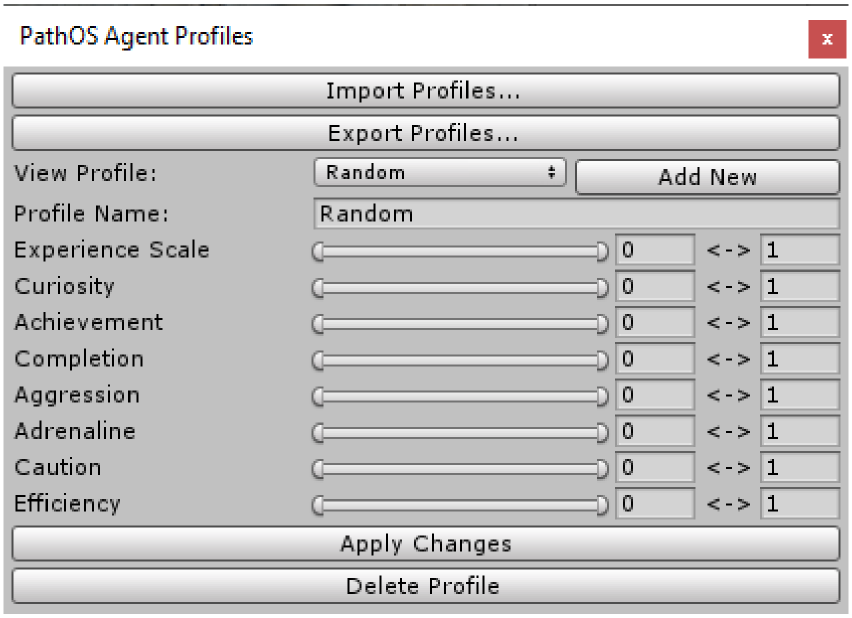
Figure 2: The process for instrumenting a level design with entity labels:
1. A subsection of a level with interactable objects.
2. Selecting an object to tag as a collectible object.
3. Completed entity labels for level subsection.
4. Level overview showing Gizmo icons for all interactive entities.

Designers can also manipulate individual entities through a list view, which can be used to change entity types or flag entities as persistently known by an agent. The list view can also be used to delete swaths of entity tags en masse, as an alternative to the deletion mode of the visual markup tool.

When the PathOS Inspector pane is active, designers can toggle the visibility of icons overlain in the scene indicating the location of various entity tags. These are drawn using Unity's Gizmos system, a framework used by the engine for displaying information in-scene. The display of all Gizmos can be customized from the main Editor window, allowing users to adjust the interface to their liking.

AGENT CUSTOMIZATION. To create a PathOS agent, the designer can simply create an instance of the provided Unity prefab (pre-fabricated object; in other words, a template). Alternatively, they can drag and drop the PathOS Agent script onto the object or model of their choosing in the scene. Tweaking an agent's motivation profile is then achieved by manipulating sliders on the agent's Inspector, as shown in Figure 1. Other factors governing the agent's movement speed, view camera, and so on, can also be configured from this panel.

Custom profile templates containing ranges for each of the motives can be managed from an additional Editor window (see Figure 3). These templates can



View Profile: Random

Profile Name: Random

Experience Scale 0 <-> 1

Curiosity 0 <-> 1

Achievement 0 <-> 1

Completion 0 <-> 1

Aggression 0 <-> 1

Adrenaline 0 <-> 1

Caution 0 <-> 1

Efficiency 0 <-> 1

Figure 3: Editor window allowing for the creation and customization of agent profiles.

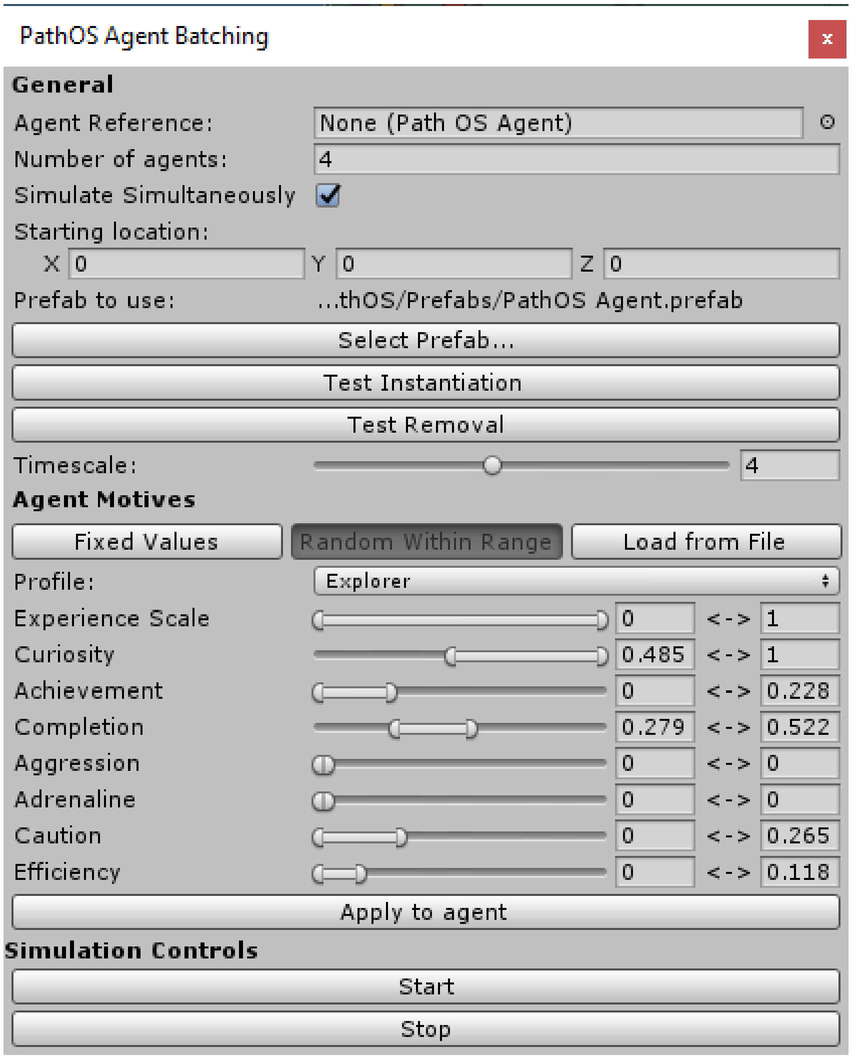
be selected within the agent's Inspector to automatically randomize the agent's motives within the ranges specified in the template.

From the main PathOS inspector, designers can also input a custom scoring matrix, as well as import or export different scoring matrices from disk. This functionality may be useful, for example, in emulating the behaviour associated with different gameplay strategies specific to a particular project.

RUNNING SIMULATIONS. Agent simulation can be triggered simply by hitting the "play" button in the Unity Editor, at which point all enabled PathOS agents in the scene will begin simulation. The PathOS Inspector allows the user to specify when the simulation should be terminated (i.e., after a set amount of real time has elapsed, or all agents have reached the final goal in the level, if one exists).

Testing sessions can also be orchestrated in batches, through a separate Editor window accessed via Unity's menu bar. Figure 4 displays the layout of this window, which allows users to specify the desired number of agents to simulate and a means for generating motivation profiles. Users can specify whether agents should use the same motivation profile, profiles drawn from customizable ranges per-motive, or profiles loaded from a file on disk.

When simulating at scale, agents can run consecutively or concurrently. For concurrent runs, the number of agents active simultaneously is limited to a set batch size, over which batches of agents will be simulated in succession. Designers can also specify an accelerated timescale to reduce the time needed for simulated testing, while preserving their ability to observe live agent behaviour directly.



PathOS Agent Batching

General

Agent Reference: None (Path OS Agent)

Number of agents: 4

Simulate Simultaneously ☑

Starting location:

X 0 Y 0 Z 0

Prefab to use: ...thOS/Prefabs/PathOS Agent.prefab

Timescale: 4

Agent Motives

Profile: Explorer

Experience Scale	0	<->	1
Curiosity	0.485	<->	1
Achievement	0	<->	0.228
Completion	0.279	<->	0.522
Aggression	0	<->	0
Adrenaline	0	<->	0
Caution	0	<->	0.265
Efficiency	0	<->	0.118

Simulation Controls

Figure 4: Editor window allowing for the orchestration of agent testing in batches.

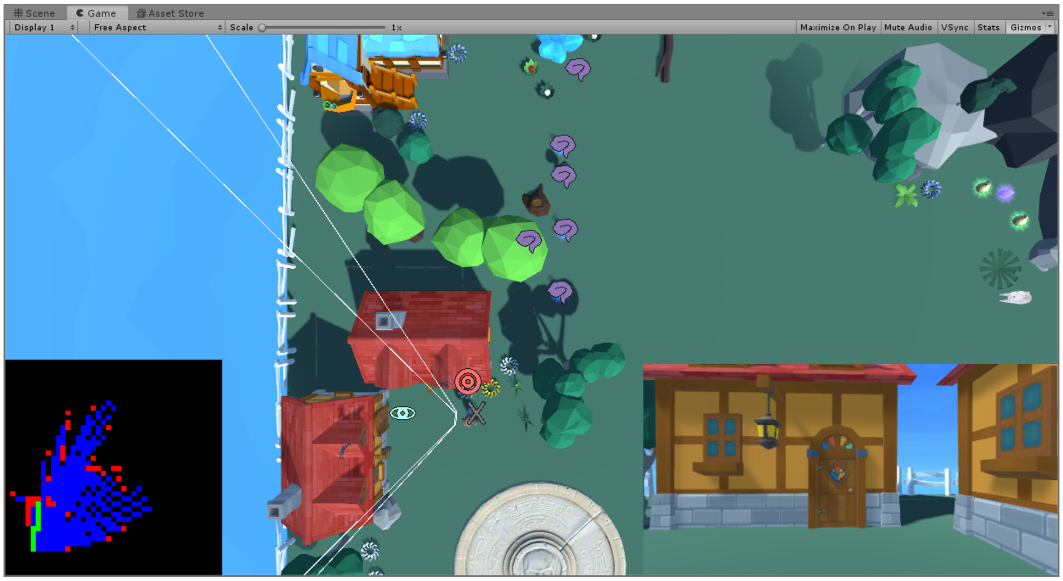


Figure 5: User interface for runtime observation of agent behaviour. Onscreen icons display the status of level entities in agent memory. The agent's mental map and point-of-view are also shown in the lower left and right-hand corners respectively.

REAL-TIME OBSERVATION. During simulation, designers can use PathOS' optional observation camera to move around the scene from Unity's game view, observing agent movement in real-time. They can also select an agent in Unity's hierarchy to view a simplified on-screen representation of the agent's logic, shown in Figure 5.

Icons are overlain on entities in the scene to reveal their state in the agent's decision logic, indicated as either currently targeted, already visited, visible, or in memory. In the lower left corner of the screen, the agent's mental map grid (see Section 3.2.2) is rendered. The lower right corner shows a view through the agent's "eyes", representing player point-of-view. Additionally, an on-screen legend explaining entity icons and mental map coloration can be toggled using the spacebar.

This simple visual feedback can help designers to understand the rationale for agent behaviour; for instance, if a particular entity goes unvisited, a designer may note that it was never flagged as visible, and is thus difficult to find from a player's vantage point.

LOGGING AND VISUALIZATION. To facilitate the analysis of agent data in groups, and reflection on agent behaviour post-testing, PathOS includes a simple logging utility which can record agent behaviour to files on disk. These logs contain information regarding an agent's navigation trace through the game world, which entities the agent visited, and the agent's motivation profile.

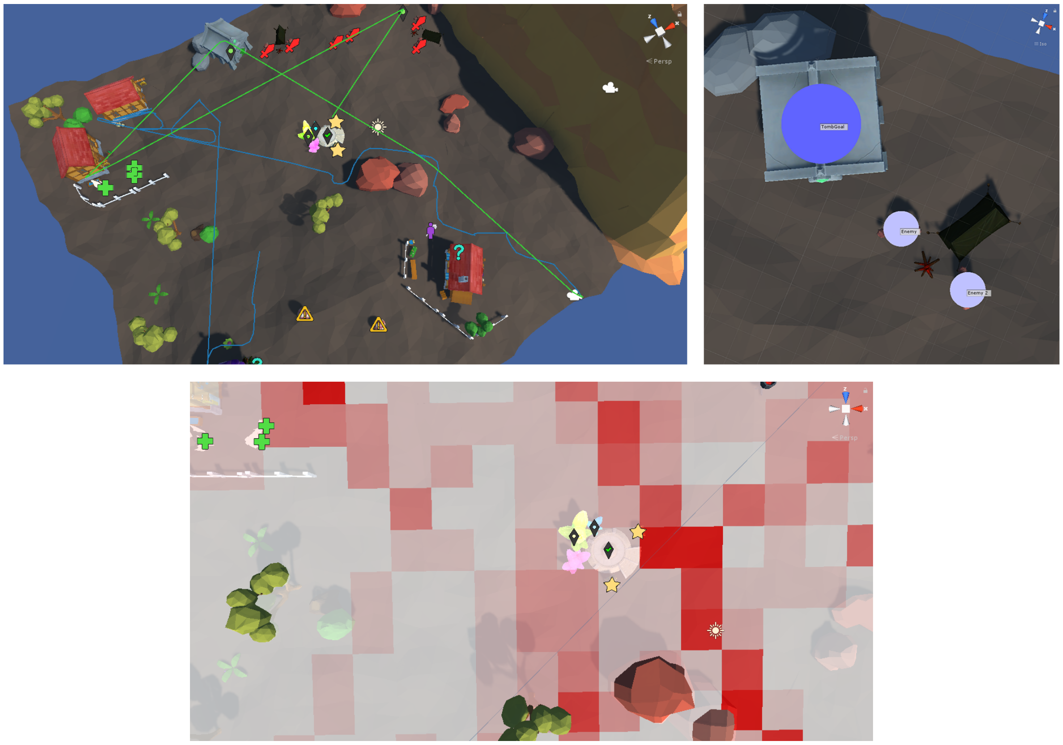


Figure 6: Different visualizations of agent playtrace data. The framework renders visualizations as an overlay on level geometry in the Unity editor. Clockwise from top left: Individual agent paths, entity interactions, heatmap of agent navigation.

The PathOS visualization tool can then be used to load these files and display a visualization of agents' behaviours overlain in the Unity scene, as shown in Figure 6. This utility includes customizable views of individual agent trajectories through the world, heatmaps, and entity interactions (i.e., what proportion of the "testing population" visited each game entity). Data can be viewed on a per-agent or group basis. Additionally, the motivation profile for each agent can be reviewed and copied to an agent in the scene for further live experimentation.

Our overall design philosophy in creating the PathOS UI was to ensure fluid integration with the Unity UI and require as little instruction as possible. In doing so, we hope to provide maximal value for users looking for a way to better inform their choices during the level design process.

4 SYSTEM EVALUATION

The value of any development tool is broadly determined by two factors: its usability, and its utility (or usefulness). Between two tools that are equally easy to learn and navigate, that which provides more useful features for its intended audience will prove more valuable. Likewise, between two tools that offer equally helpful features, it is preferable to interact with the more usable of the two. Both of these qualities play a key role in how users perceive and interact with a tool, and whether it has the ability to reach widespread adoption.

To assess the usability and usefulness of PathOS, and to understand how the tool might be improved, we conducted a user study with game development professionals, all of whom had at least 3 years of relevant experience. This chapter describes our study method and participants, as well as providing an overview of the study results. An in-depth discussion of study results is given in Chapter 5.

4.1 METHOD

The purpose of this evaluation was to gather feedback on developers' impressions of PathOS' usability and impact on the level design process, in addition to their experiences learning to use the tool. Additionally, we wanted to compile a list of any issues users encountered, as well as usability improvements and any "wish-list" items for enhanced functionality, to inform future development of the tool. In other words, our evaluation seeks to answer the following two questions:

1. *How can PathOS contribute to the game design process?*
2. *How can PathOS be improved to better suit the needs of game developers?*

4.1.1 *Webinar*

Before conducting the user study, we organized an online seminar[1], discussing research in the area of AI-driven playtesting and the PathOS tool. The seminar was streamed live on YouTube, delivered by myself and two members of our research team, and lasted for one hour. We advertised the seminar among members of the game development and UX research community via Eventbrite, LinkedIn, and

[1] A recorded archive of the seminar is available here: https://www.youtube.com/watch?v=stPpc1pp6IE

Twitter. Our primary goals for this seminar were to assess potential users' initial reactions to PathOS, in addition to spreading knowledge of AI testing in general and the availability of our tool (since it is open source). Additionally, we hoped that some viewers would express interest in participating in the user study, though no participants were directly recruited through this method.

During the initial live broadcast, approximately 35 viewers watched the seminar. As of this writing, the seminar has a view count of 170 including both live viewership and those who watched the recording of the broadcast after the fact. Questions from the audience were largely focused on clarifying agent behaviours (e.g., how agents make decisions), the availability of specific features in the tool, and potential future developments for the tool (e.g., integrating audio feedback, testing multiplayer games).

4.1.2 *User Study*

The core of our evaluation was a three-part user study comprising two interviews (pre and post) and a level design exercise using the PathOS framework. Both interviews followed a semi-structured protocol; a copy of the interview guide used is available in Appendix C. Interviews were completed in person and remotely via Discord and Google Hangouts depending on participants' location and schedule. Interviews were recorded using Open Broadcaster Software. As the interview sessions involved on-screen demonstration of the tool and participants' work from the design exercise, both audio and screen content were recorded. Participants were compensated for participating in each of the pre- and post-interview sessions, with participation in the post-interview contingent on completing the exercise given.

PRE-INTERVIEW SESSION. This session lasted approximately 30-45 minutes and occurred either in-person or remotely as described above. The primary goals of this session were to understand participants' existing game design processes and to introduce them to the PathOS tool. After providing consent, participants filled out a short questionnaire collecting demographic data and information about their development experience. Following this, participants completed the pre-interview, aimed at exploring their role in game development, their experiences, and their design process. The researcher then gave participants a brief introduction to the PathOS tool, explaining its basic use and demonstrating the basic functions needed to use the tool on a laptop computer (for in-person sessions) or via screen-sharing (for remote sessions).

At the conclusion of the first session, the researcher briefly explained the design exercise to the participants and provided resources to the participants for completion of the exercise: a link to download PathOS, a handout explaining the task and basic use of the tool (see Appendix B), the PathOS user manual (see Appendix A), and a link to the online seminar recording (timestamped at our live demonstration of the tool). These resources were provided to simulate the availability of instructional materials to a "real" end user. It is common for similar tools to have so-called "quickstart" guides, written manuals, and walkthrough videos available to suit the learning preferences of different users.

LEVEL DESIGN EXERCISE. For this exercise, participants were asked to design two levels for a hypothetical game with different design intentions for each level. Participants were asked to create one level incentivizing players to explore and collect lots of items, and one level where players would need to fight many enemies and visit in-game goal markers. The exercise was completed using the Unity game engine.

We gave participants a template project in Unity containing the PathOS tool and a collection of "prefab" assets that could be dragged and dropped to quickly build each level. Participants were told to use PathOS as naturally as possible, using only the features that they felt necessary or helpful, or those that they wanted to explore out of personal interest. The provided project also contained a simple character controller that allowed participants to play through their levels themselves if they wanted (i.e., without using AI agents).

This part of the study was a "take home" exercise; participants could complete it on their own terms and were not instructed to finish it in one sitting or in a particular environment. Participants were told to spend about 2-3 hours on the exercise in total, though they were not restricted from experimenting with the tool for longer if they chose to. Minimal contact with the researcher regarding the study occurred during this time, though participants were allowed to ask questions via direct message if they ran into a question or technical problem that critically impacted their ability to complete the exercise. No major incidents of this nature occurred, though some bugfixes were made during the study based on minor technical issues encountered by participants. These adjustments did not affect the core functionality or user interface of the tool in any way, and are thus assumed to have no bearing on the results of our evaluation.

POST-INTERVIEW SESSION. This session lasted approximately 40-60 minutes, and occurred either in-person or remotely, as with the pre-interview session. The primary goal of this session was to explore how participants used PathOS, their opinions regarding its usability, and whether/how the tool could be useful in the game development process. At the start of the session, participants were asked to share the levels they created on a laptop computer (for in-person sessions) or via screen-sharing (for remote sessions). Participants provided a copy of their levels (as Unity scenes) to the researcher for later reference. The post-interview questions were then answered through discussion. During this time, participants could refer back to their levels visually or demonstrate their points by interacting with the

PARTICIPANT	AGE	OCCUPATION	UNITY EXP.
P1	29	Graduate Student	Advanced
P2	N/A (Withdrawn)		
P3	25	Tracking Data Manager (Mobile)	Intermediate
P4	29	ML Expert (AAA)	Intermediate
P5	26	Data Tracking Manager (Mobile)	Novice
P6	23	Programmer (Indie)	Advanced
P7	24	UX Researcher (Consultant)	Intermediate
P8	23	Programmer (AAA)	Intermediate
P9	23	Programmer (Indie)	Advanced
P10	23	Graduate Student	Intermediate

Table 3: Basic demographic information and self-reported experience working with the Unity engine for study participants.

PathOS interface. After completing the interview, participants were thanked for their time and debriefed on our research.

PARTICIPANTS. We recruited ten participants, all of whom had either completed an undergraduate degree in game development and/or were currently working in the gaming industry. One participant (P2) withdrew from the study after the pre-interview component due to a time conflict, leaving nine participants that completed the study. A summary of demographics and development experience for these participants is given in Table 3.

Of the nine participants that finished the study, eight completed the study as prescribed above. One participant (P4) was a special case, as they had independently been using the tool themselves before being recruited to participate. This participant had been introduced to PathOS by watching the webinar and used the tool for a few weeks prior to contact with the researcher. After discussing the tool with the researcher, they agreed to participate in the study. Due to the participants' existing experience with the tool and time constraints for the participant, they did not complete a pre-interview or design exercise, and completed a slightly modified version of the post-interview (introductory questions from the pre-interview were added, and questions specific to the level design exercise were removed).

Two participants had some minor irregularities relating to the study procedure and their understanding of the task respectively. One participant (P7) only created one level for the exercise, due to time constraints. Another participant (P8) initially misunderstood the purpose of the tool, assuming that it was meant primarily as a driver for in-game character AI, rather than as a testing utility. This became clear during their post-interview; however, discussion with the researcher clarified this point and questions regarding the tool's utility and use cases were answered after this clarification. The participant in question had nonetheless used the tool's output to make adjustments to their level design during the exercise (i.e., as a testing tool), and so it is assumed that this confusion did not have a notable impact on their use of the tool during the exercise. Therefore, for the purposes of this analysis, it is assumed that neither of these instances had any significant effect on the outcome of this evaluation.

ANALYSIS PROCEDURE. Analysis was completed by myself and another member of our research team. In-depth analysis was only performed on post-interview data, as no evaluative questions were asked in the pre-interview and we did not aim to make any correlations between pre- and post-interview responses for individual participants. However, participant comments from pre-interview data have been quoted and summarized for the purposes of discussing common design and development practices in our discussion of results.

After transcribing interview data, we constructed two spreadsheets for analyzing the post-interview data, each with a different evaluative intent. Both follow a deductive coding approach to extract comments from interview data. The first spreadsheet (SUMMARY) is aimed at extracting a high-level synopsis of participants' insights regarding their use of the tool and how it impacted their design process. Here, we collected short quotations from participants and summarized their interview responses. Categories for these excerpts were created deductively before transcripts were analyzed, based on the interview questions used. A list of the categories used is given in Table 4. During the analysis process, quotations and summary notes were placed into the appropriate category. Categories were allowed to contain multiple entries, and were not mutually exclusive (e.g., an excerpt positively discussing the usefulness of a particular feature could be placed in both the "Usefulness" and "Liked Features" categories).

It is important to note that the quotes and responses entered for each category did not necessarily come from participants' immediate answers to questions explicitly referenced by that category. We chose to use a semi-structured protocol for interviews to allow for natural discussion; consequently, participants would often reference prior questions, add to previous thoughts, or "jump ahead" during the interviews.

The second spreadsheet (FEATURES) is designed to capture comments made about the tool's individual features. Categories were created deductively based on the features of the tool (a list is provided in Table 5). For each feature category, comments made were subcategorized as positive, negative (indicating an issue), or suggesting a change. Negative comments were additionally classified as having low, medium, or high severity. Subcategories were mutually exclusive. However, participants' comments would occasionally include multiple features discussed in the same sentence or with the same conclusion, and could thus belong to multiple feature categories (a one-to-many mapping). An example of this is the comment "I think the visualization is cool. Seeing the paths and seeing the heatmaps.", which belongs to both the PATH VIS and HEATMAP categories as a positive comment.

SHORT NAME	DESCRIPTION
PROCESS	Notes on when and how participants used PathOS for testing while they were creating levels.
LEVEL CHANGES	Whether participants made changes to the levels they created based on output from the tool (e.g., agent behaviours), and the nature of changes made.
LIKED FEATURES	Features highlighted by participants as easy to use, useful, interesting, etc.
ISSUES	Problems or concerns encountered by participants while using the tool.
WISHLIST	Suggested changes to the framework.
LEARNING	Notes on participants' first impressions, any confusion encountered, and the resources provided (handout, manual, video).
USEFULNESS	Whether participants stated the framework was useful when asked, and why.
APPLICATION	General application areas participants feel the tool is best suited for (e.g., playtesting vs. QA).
USE CASES	Examples given by participants of how they could use the tool in their work or personal projects (current or hypothetical future).

Table 4: Categories for summarizing responses and participant comments used in the SUMMARY spreadsheet.

FEATURE NAME	DESCRIPTION
LEVEL MARKUP	Utility for labelling game entities according to their interaction type.
AGENT PERSONALITY ADJUSTMENT	Customizing the personality of agents in the Unity Inspector.
AGENT BEHAVIOUR	Logic for agent behaviour simulation.
RUNTIME GIZMOS	Gizmos displayed in the runtime game view as an indicator of agent logic.
MENTAL MAP	Spatial memory of the agent displayed in runtime game view.
AGENT VIEW	Agent POV camera displayed in runtime game view.
RUNTIME UI (OTHER)	Used for comments about the runtime UI non-specific to gizmos, mental map, and agent view.
DATA LOGGING	Recording of agent logs for later inspection and visualization.
HEATMAP	Heatmap visualization from agent data logs.
PATH VIS	Visualization of individual agent paths from data logs.
ENTITY VIS	Visualization of agent interactions with game entities from data logs.
VIS (OTHER)	Used for comments about the visualization utility non-specific to heatmap, path, or entity visualizations.
BATCHING	Ability to run multiple PathOS agents in sequence or parallel.
PROFILES	Utility for quickly assigning agent personalities based on default (included) or customized player profiles.
INSPECTOR (OTHER)	Used for comments regarding the Unity Editor/Inspector interface not specific to other categories.

Table 5: Feature categories used in the Features spreadsheet (subcategories for positive comments, negative comments, and changes exist for each of the categories described here). Features of the tool are discussed in detail in Chapter 3, particularly in Section 3.4.

We also recorded the total number of participants that made any comments belonging to a particular category (rather than the total number of comments in that category) to assist in evaluating the impact of each feature on participant experience. This decision was made for a few reasons. First, since our evaluation is aimed primarily at understanding users' impressions and identifying opportunities for improvement, it is more important for us to consider the number of participants that had a particular impression or issue, rather than the number of individual comments contained in each category. Additionally, participants would often jump between thoughts, reiterate themselves, reword similar thoughts, or add on to their thoughts later in the interview. This makes it difficult to meaningfully separate whether certain excerpts count as one or multiple comments, a distinction which is not significant to the goal of our evaluation. Lastly, while a summary of the number of comments made in a particular category might help in the prioritization of issues, we decided to assess issue severity separately. This allows us to better account for the semantic meaning of comments, rather than just their volume (e.g., a single comment describing an issue with strong words could indicate higher severity than two comments describing a different issue as a mild inconvenience).

Both researchers completed the Summary and Features spreadsheets independently while reviewing the post-interview transcripts. Afterwards, the researchers met to ensure that 100% agreement was reached. Since the researchers were able to choose which quotations to include, we did not consider the use of slightly different quotations with the same meaning (e.g., participants rewording comments from a previous sentence) or differently trimmed excerpts as disagreement. For the Features spreadsheet, any comments that were initially assigned different categories were discussed and re-assigned to the most appropriate category. For the Summary spreadsheet, notes were compared to ensure that interview responses were interpreted equally by both researchers (e.g., whether the tool was useful). No discrepancies were encountered in the interpretation of interview responses in the Summary spreadsheet. Quotations included in the Summary spreadsheets from both researchers were combined to produce a list of high-value excerpts for inclusion in discussion of the results.

4.2 RESULTS

As discussed in Section 4.1, this evaluation has two primary goals. First, we aim to understand PathOS' contributions to the process of game design and development in a practical setting. The task presented to participants served as a microcosm of this process; instead of taking many days or weeks to create a series of levels or an expansive open world, participants had a few hours to create a small interactive scene with the assets provided. Despite the difference in scale, however, the motivation and outcome of the smaller task is not so different. In both cases, the creator is motivated by some design goal in creating a virtual space for players to explore. Thus, if we are able to understand how participants used PathOS for this exercise and their impressions of its utility, we can begin to discover how the tool might be useful to professionals in a broader context. Results relating to this goal are presented in Section 4.2.1.

Our second objective is to identify issues with the tool's implementation and user interface, as well as opportunities to enhance its functionality and usability. To this end, feature-level impressions have been compiled in Section 4.2.2.

4.2.1 *Participant Design Process and Broad Impressions*

As discussed in Section 4.1.2, participants were free to use the tool as they naturally might, rather than being asked to follow any particular design process. Accordingly, the results discussed here reflect a great deal of variety in not only the levels created, but the design and validation tactics employed, as well as use cases suggested by participants.

CREATED LEVELS. Participants were asked to create one level aimed at getting players to explore and collect items (the EXPLORATION level), and one focused on fighting enemies to reach mission markers (the COMBAT level). With the exception of P4 (who completed a modified version of the study excluding the exercise, as noted in Section 4.1.2) and P7 (who created only one level due to time constraints), each participant created two distinct levels reflecting these objectives.

Three participants (P3, P5, and P9) used a physical layout which was largely the same for both levels (e.g., physical features such as landscape, trees, buildings, and pathways), using variation in interactive entities (e.g., enemies and pickups) to create a different experience. The remaining participants made substantial variation to both the physical layout and interactive objects between each of the levels created. As one would expect, the primary difference between levels was the density of interactive objects aligned with each of the given design goals. That is to say, COMBAT levels contained a high density of enemies, and EXPLORATION levels contained a high density of collectible items.

Interestingly, one participant (P3) created a level with substantial elevation changes (a cliff face which could be climbed along with a cave underneath that could be explored). Although the current prototype of PathOS is not yet built to work correctly with such scenarios (as it uses a 2D spatial approximation of the world layout), the participant did not experience any significant issues, noting only that the "mental map" visualization on the runtime interface would display incorrectly for the affected area.

There was a fairly substantial amount of overall design variation in the levels created by different participants. Many participants had a fairly detailed "story" in mind of what players would do as they explained each of their levels, or expressed

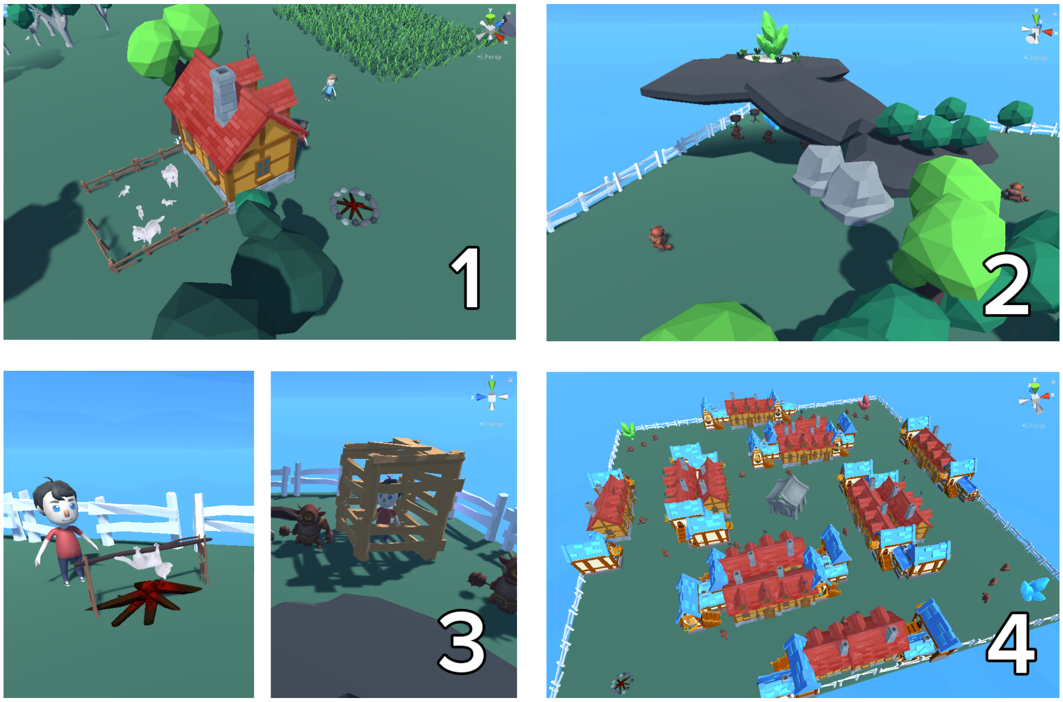


Figure 7: Screenshots of levels created by participants:
1. Farm area created by P1 as part of their exploration level.
2. Overhang and cave area created by P3 as part of their combat level.
3. Two scenes from P6's combat level showing vignettes of non-player characters.
4. Combat level created by P8 following a city theme.

that their level layout was meant to evoke a particular theme (e.g., farm, village, wilderness, etc.). As previously mentioned, participants worked from a collection of prefab assets included with the provided Unity project. Thus, there is a degree of visual similarity between the levels shown, despite the variations in design between individuals. Representative screenshots of the Unity scenes created by participants are given in Figure 7.

participants' use of pathos. Of the eight participants that completed the design exercise, all indicated that they used the AI agents to test their designs. Four participants stated explicitly that they either never (P6, P7) or rarely (P1, P9) tested the levels themselves (i.e., only used the AI agents), despite having the option to run through the level themselves using the provided first-person controller.

Two other participants stated that they frequently used the tool during the design process, before completing the layout of their level. One, P8, noted that they initially observed agents as a "sanity check" after completing an area to ensure that they had tagged objects correctly using the provided markup and observe "how the AI reacted to the new additions in the world". The other participant, P3, stated that they "started testing with the AI from the get go", saying that "As soon as I made any change, the main thing was, can the AI path to it?".

The two remaining participants (P5 and P10) stated that they primarily started using the tool after completing their initial level layout.

visualization vs. real-time observation. Half of the participants that completed the exercise (P1, P6, P9, P10) experimented with using the tool's logging and visualization features as part of their process. The remaining four participants only observed agents in real time. P1, who primarily used the batching utility and visualization to inspect agent behaviour, stated that they felt "just looking at the resulting data is probably more helpful than watching him".

Participants who focused on real-time observation often commented on the information present in the runtime UI overlay as helpful to their process. P3, for instance, stated that "those two tools, the combination of the two of them, drove a lot of my decisions" (referring to the mental map and agent view components of the runtime interface). P5 stated in reference to the runtime interface that "for what I need it's already very useful to have this view in the game scene".

changes made to levels. Five of the eight participants that completed the design exercise (P1, P3, P6, P8, P9) stated that they made changes to their levels as a consequence of observing agents' behaviour (either at runtime or through visualization of data logs). Among the three remaining participants, one (P10) noted that "I could see a use for making some changes as I noticed the behaviour with the agent", but stated that they were more interested in using their time to explore the visualization system. One other participant (P5) had specific ideas of what they would want to change in their levels based on the agents' behaviour ("could've also put some bigger points of interest around the opening [to the mines], kind of lead the AI closer") but did not pursue changes in the time they had for the exercise.

Of the five participants that did make changes during the exercise, the changes that were made included small geometry adjustments meant to prevent unwanted shortcuts, larger layout changes meant to give players more options, and placing additional objects for players to interact with. P1, for instance, made changes to prevent players from getting stuck on a troublesome part of the navmesh, as well as attempting to occlude player vision of a faraway objective at the start of the level. P3 noted that they made several changes throughout their design, primarily focused on guiding the agent towards their intended path ("Let's block this off. Let's make sure they have to pivot. Maybe when they pivot they catch something in their peripheral [...]"). P8 used their observations of agent behaviour to guide placement of additional health resources for the player, stating "I had an 'aha!' moment [...] just let me put the healthpacks before the major battles [...] the healthpack is where the agent would generally go before going to the crystal."

GENERAL IMPRESSIONS OF PATHOS. Participants' general impressions of the tool were quite positive overall. Particular areas highlighted by participants include:

The tool's interface (P1: "It was all super intuitive, easy to use", P3: "[...] felt very much like this is Unity, I know what I'm doing here [...] the only thing that looked visually different from what I was used to was the level markup tool and I knew exactly what it was supposed to be", P5: "It's very clean, and very concise with its information.");

Technical implementation (P4: "That it kinda worked out of the box [...] being able to download code that works, and does, you know, something, is already impressive"; P6: "It's a really good tool. I love the way it walks around. [...] really impressive. I don't actually have any clue how you would approach this. If you wanted to be more like an actual player [...] That's so far down the road though. This is great.");

Usefulness (P3: "[PathOS] improves the process a lot. Without this tool I don't think I could've made something that I'm as happy with in such a short amount of time", P10: "It definitely helps give the designer [an idea] of how they want the level to go and important points that they want a user to experience [...] Even though it's an agent you can get an understanding like 'Oh, maybe a player might not visit this'.");

Ease of setup or integration (P1: "It was pretty straightforward learning how to use the tool", "P7: "I think generally it's pretty easy to use, it's pretty easy to plug and play", P8: "I just saw PathOS and I was like, okay great, that's easy to know that everything I need is sorted into that already").

FIRST-TIME USER EXPERIENCE. Participants did not express any major concerns with the process of setting up and learning how to use PathOS. It should be noted that participants were given a brief tutorial on the tool at the first interview session, as well as being provided with documentation and a walkthrough video. However, this instruction was not in excess of what is normally available online for development tools "in the wild", which often provide extensive video tutorials and help documentation.

Many participants remarked that the tool was easy to learn when asked about their first experiences. P4, who did not receive the tutorial session (since they had been working with the tool independently), stated that "it was easy to see how to replicate the stuff in the video" (referring to the webinar). P3 mentioned that the tool was "extremely easy to use", saying "Once I got started, I didn't refer to the video again, it was very intuitive." P9 noted that despite having the user manual open, they were "able to figure out most of it just by trying to do something" rather than having to consult documentation.

Two participants (P5 and P10) stated that the number of features and settings available could be daunting (P5: "It's a little overwhelming at first, but I think if you take it in strides that works", P10: "It's a lot to take in at first, all the different settings"). However, P5 noted that the modular design of the tool helped to alleviate this factor, saying "It's great because you can choose to do only some of the features and not all of the features and have it still work and be effective". P10 stated that the documentation helped them to keep track of everything, saying that "having that outline on the side really helped knowing the steps I had to do".

Participants also commented positively on the documentation provided, both in terms of the written manual and the walkthrough video. P1 stated that the "user manual and the handout helped a lot" in learning to use the tool. P7 said that the documentation was "mostly very clear", suggesting that additional information on where to click in the Unity interface to access certain settings might be beneficial to help prevent frustration for less experienced Unity users. P4 found the video particularly useful, stating that "If I had not watched the video, I would have found it, kind of difficult [...] in the fact that I would have needed to read a lot of the code." However, P4 also stated that for basic use of the tool, "If I had the Unity prototype that came with PathOS [...] even without watching the video, anyone should be able to replicate the demo."

PERCEIVED USEFULNESS. When asked whether they thought the tool was useful, all nine participants said yes. Upon elaboration, participants cited several different reasons for their assessment. A common reason given was the potential time or labour savings over other methods of testing. P3 noted of the tool's time-saving ability, saying "I can see results, take action on those results, and it's been five minutes. That's extremely valuable." P3 also alluded to a reduction in the burden of recruitment, stating "I can get an AI to run through it without me having to bug the three people that are near me that can actually test something, and realistically you can only get them [meaning humans] to test it once." P9 said using PathOS "was much faster than me having to go and test everything [...] It would take even longer for a person to go through that. So being able to have 8 people go at once on the same level 8 times as fast as normal I think will always be useful." P1 mentioned the hands-off nature of the tool, saying that "you can just run a whole bunch of playthroughs all at once, come back in an hour later and have all of your data, right. It's basically unsupervised, you don't have to, like, you don't have to sit there and watch it run."

Participants also commented on the value and usability of information gleaned from watching agents or visualizing their behaviour. Talking about the tool's visualization features, P1 said that "being able to see the general flow of where players go, how long they spend at each place. I think that's really useful." P6 stated that they were "interested in the flow [of the level] and the paths that they take", saying "I think I got a lot of feedback from the AI about the layout of the level". P10 argued that the information obtained from PathOS could potentially provide insights that would be difficult to obtain with human players: "[...] Like get an understanding of the experience a little bit. Which you don't get from players. With this agent you see what the thinking process is. Oh the agent is heading in this direction now, they're looking for this item or something."

Ease of integration and generalizability to other game projects were also highlighted by participants as enhancing the tool's usefulness. P4 noted that "I think it's useful because it's something that can be extended very easily. And for, probably a lot of games, it doesn't need to be extended." P6 said that they felt the tool "would work with any game and not cause any issue because it's not actually modifying any game objects". P7 stated that they think the tool "has quite a bit of versatility with the different genres it could be used with".

The last major factor participants referenced when discussing the utility of PathOS was its positive contribution to the general process of level design. P3 noted that "it worked really well for my workflow, or my design flow, where I want to build something, test something, see how it works in that moment [...] it helped me prototype a lot faster." P7 mentioned that the tool could potentially be used very early in the design process, before any assets or gameplay mechanics are implemented: "even just having blocks and cubes and that kind of stuff, and absolutely baseline models, I think this could be a very good prototyping tool for level designers." P10 noted that the tool (particularly level markup) could help keep one's design intent in focus, saying "[...] where are the objectives supposed to be, where is the goal supposed to be. It helps the designer also keep track of [their design], like having these collectibles and objectives".

APPLICATION AREAS OF PATHOS. When asked whether they thought the tool might be better suited for playtesting or QA applications, most participants said that they felt PathOS could work both for probing user behaviour (P1: "I think it's useful. Just for general level flow sort of thing", P7: "Figuring out an approximate of what your player is going to do, I think your tool would be really good for that") and finding bugs (P5: "this would definitely help alleviate a lot of the trivial tasks that QA needs to go through when it comes to checking collisions and stuff like that"). Five of the nine participants (P3, P4, P6, P8, P10) expressed no strong leaning as to whether they thought the tool would be more useful for one application over another.

The remaining four participants were split on whether they thought the tool would be more well-suited to support playtesting (P1, P7) or QA (P5, P9). The participants who stated the tool was more useful for a playtesting-type application cited that the types of bugs encountered by human testers would not necessarily align with those encountered by an agent (P7: "Bug testers like, they spend hours running into walls [...] QA testers do some weird things that you don't expect"). Conversely, those who associated the tool more strongly with QA echoed a similar sentiment regarding UX (P5: "Because again player research requires players to research on, at the end of the day", P9: "right now it seems like it's more useful for validation of a level than necessarily how a "real player" would interact with it I guess. 'Cause real people are weird and do weird stuff").

POTENTIAL USE CASES. There was a great deal of variety in the answers given by participants when asked about potential use cases for PathOS in the context of their own experience and work or personal projects. Some participants highlighted open-world games as an opportunity to use the tool (P8: "In AAA, probably like an open-world game. [...] I know AAA always [needs] collectibles and corners and this would be amazing for it.", P10: "Definitely if I'm working on a game like an open-world game, or different levels, I'd like to implement this to get those quick tests, see if this is the right placement of objects.")

When asked whether they might be able to use PathOS for their own personal game projects, participants discussed the tool's potential utility for "indie" projects. P3 envisioned the tool as useful for a small project with a limited pool of potential playtesters: "I'm imagining an indie project where I'm just making this one game, this one level, it's a small exploration game. And I have maybe 5 people who can playtest it for me every now and then [...] just being able to have the dummy run through it, super helpful, super valuable." P7 noted that they thought the tool could work for projects of differing genres, giving the examples of management-RPG style games such as *Stardew Valley* (ConcernedApe, 2016), or dungeon crawlers.

Participants also mentioned a few specific use cases where PathOS could provide an advantage over traditional approaches. P1 explained that the tool could be useful for game jams, saying "there's no time to get actual playtesters during a game jam. So, being able to run like a hundred agents through your level just to see, you know, how it flows. I think that would be useful." P5 suggested that the tool might be helpful in the early stages of a project where confidentiality is a major concern: "Maybe you're working with NDA secure products so you're pretty hesitant to show it to players. So this might be a really good supplement to that."

P4 noted that PathOS could be especially useful in testing areas without complex obstacles or enemies, that rely heavily on NPC interaction—such as cities. They noted that PathOS could be useful in coverage testing to verify playability: "If I had defined, like, this is how interactions with NPCs work, and they go talk to NPCs, and that is the only thing necessary to solve quests, is it possible for an agent to 100% the game? [...] As a human, to test that, that would be really tedious and take a lot of time. Whereas, if I got one PathOS agent to do that, and verified that it's working 100% of the time, that's cool." Another design-specific use case was proposed by P6, who viewed the tool as a potentially valuable complement to procedural level generation: "I think if I were to create a procedurally generated level this tool would let me test all of the parts [...] And that's a big problem when you're making a game I'm guessing, you can't test everything. There's problem areas, and I can see this tool being really great for procedural levels."

In summary, participants expressed a great deal of variety in the scenarios they described when discussing potential use cases for the tool. All nine participants indicated that they might be able to make use of PathOS at some point in their current or future game development projects when asked.

4.2.2 *Feature-Level Impressions*

This section describes a summary of our findings from compiling the Features spreadsheet, collecting participant comments on individual features of PathOS. For a summary of the functionality provided by each of these features, the reader is referred to Table 5, with a more in-depth overview present in Section 3.4. Throughout the remainder of this subsection, a high-level summary of the comments given pertaining to each feature is provided.

level markup. The level markup tool was well-received overall, with all nine participants reporting positive impressions in the interview (P1: "really nice", P3: "I can see that it's a tool for designers", P5: "intuitive [...] clean and clear", P10: "very easy to select a markup and paint [...] I liked how there was an entity list I could go and modify"). However, all nine participants also commented on encountering small usability or quality-of-life issues with the tool. Participants mentioned that, without a way to quickly tag large groups of objects, the tagging process could become lengthier than it otherwise would be (P3: "somewhat tedious to tag all of [the game entities]").

Some participants also had issues with accidentally tagging objects by misclicking, and then having to undo their actions. Some of the suggested changes related to these issues, such as the ability to lock objects from the tagging system (P9: "put them on a different layer right, and not have it be selected") or adjusting the design of the markup cursors to include a precision indicator (P3: "having a small cursor point to what I'm looking at would be helpful"). All negative (i.e., issue-related) comments regarding the level markup were classed as low priority, except for one comment describing the entity list as "ridiculously huge" for more complex levels, suggesting that the list interface could be improved to better handle larger collections of objects.

agents. Participants noted that the process of changing agent personalities was straightforward (P7: "clear to use") and could suit different design intentions (P8: "you can change the weights of the agent to modify whatever your needs of what the level is"). One participant (P8) remarked that the first time using the tool, they "didn't really understand the player characteristics", since the Unity Inspector tab was closed. The participant additionally suggested that adding tooltips for the agent characteristics explaining their meaning could be helpful, a sentiment also expressed by another participant (P10).

With respect to agent behaviour, participants were mixed in terms of whether they found agents to act believably or interestingly (P7: "I estimated what I expected players to do based on how I laid out my level and generally the AI responded as expected", P6: "And I thought that it's really, you know, interesting that they kind of stop and look around"), or instead unconvincingly or confusingly at times (P6: "still a far cry from how a player would actually interact with these levels", P10: "I was confused, like what the agent was doing.").

The only high-priority negative comment relating to agent behaviour was given by P8, noting that the lack of a fast-forward option for single agent simulation could be frustrating: "I was having a miserable time just watching them. You know the late stages of design? Where you have to test this at the end? And like come on. Just get there. [Talking about having to wait for agents to reach the end part of a level.]" (It should be noted here that users are not *required* to watch agents in real-time, since the tool offers accelerated batch simulation and visualization capabilities. However, for users that prefer to watch agents "live" as part of their process, the ability to manipulate the simulation's timescale could improve the tool's flexibility.)

RUNTIME INTERFACE. Participants were largely quite positive about the runtime interface for viewing agent navigation through a level in real time. Regarding the interface as a whole, participants commented on the completeness of information available (P4: "easy to explain why the agent was doing something, and I think that was really important", P10: "everything I needed to know about the agent") and its clarify (P6: "Understandability, information was great.", P7: "all the pieces working together makes it pretty easy to understand what's going on").

None of the negative comments made across the categories relating to the runtime UI were classed as higher than low priority. Those comments that were made related to minor issues with identifying what to click in displaying or customizing the interface, in addition to a concern regarding reduced accessibility of the mental map colour scheme for users with impaired colour vision.

DATA LOGGING AND VISUALIZATION. Participants expressed concern with regard to setting up data logs. One participant, P6, mentioned that the logging system "felt like a bit of an afterthought", and another (P5) did not realize that logging had to be explicitly enabled for the visualization tools to work, suggesting that "maybe some sort of prompt would be helpful to remind me to enable logging". Participants also suggested various quality-of-life improvements for the logging system (P5: "having a default directory might be good", P9: "being able to rename what the folder is, like it's made by the logger").

Several positive comments were made in reference to the visualization system, particularly regarding the heatmap (P4: "I think the heatmap thing was really useful", P6: "I think the visualization is cool. Seeing the paths and seeing the heatmaps.") and individual path visualization (P1: "the individual routes was useful", P10: "I liked that there were different colours and you can select the agent you want to focus on"). Negative comments made within this category generally pertained to confusion regarding visualization settings (P10: “I was confused I couldn’t see the heatmap [referring to forgetting to adjust alpha of the display]”) or the encoding of information in the visualization (P1: “I wasn’t really sure how useful it was to see the labels there [in the entity visualization]”, P10: “a little hard to tell maybe the direction [of the agent] sometimes [in the path visualization]”).

batching and profiling utilities. Only a few participants commented substantially on the batching and profiling utilities, likely due to the fact that these tools are not necessary for basic operation of the framework (as such, many participants had not used them extensively). Positive comments were made on both batching (P1: “useful for running a lot of simulations”) and agent profiles (P5: “I can click apply profile and it saves me a ton of time”, P10: “I felt like the built in ones were optimal settings”). One participant (P10) raised a concern with the batching utility interface, noting difficulties in navigating the Unity hierarchy to view the runtime interface for individual agents during a group simulation.

general interface. Like the level markup utility, participants remarked positively on the usability of the PathOS Inspector elements in general (P1: “super intuitive”, P5: “very clean”, P7: “I liked the organization of your menu systems”, P10: “if you want to customize, it’s there for you”). Two participants remarked that knowing what to click could be difficult at times, citing an overabundance of text in the Unity Inspector (P7) or a lack of clarity in how to display different parts of the interface (P8, though they noted that it became “pretty straightforward” after having a bit of time to get used to the tool).

suggested changes. Several potential changes to the framework’s interface and functionality were discussed with participants, both in response to issues encountered by participants and as standalone suggestions. These changes were captured both in the “change” subcategories of comments collected in the Features spreadsheet and the “Wishlist” category of the Summary spreadsheet (a few comments in the “Wishlist” category did not apply to any one feature of the tool). Since we are especially interested in improving this tool for future use, we created an exhaustive changelist for future improvement, given in Appendix D.

To create the changelist, we enumerated comments from all participants from the relevant categories and collapsed them into a single change where appropriate (e.g., several participants independently suggested tagging objects en masse as an improvement to the markup tool). Where appropriate, we kept track of how many participants had discussed a particular suggestion. Changes were classified into four different categories based on their nature:

Usability changes are aimed at preventing user error or making the tool easier to understand (e.g., adding a precision indicator to the level markup cursor to prevent misclicks).

Quality-of-Life changes are relatively minor changes (from a technical standpoint) which improve the user's experience by making certain operations more streamlined or efficient (e.g., timescale acceleration for individual agent simulation), or enabling new modes of use (e.g., an additional method for filtering visualizations).

Accessibility changes improve the ability of the tool to suit users with different needs (e.g., adding icons on Inspector panel headers to help users with dyslexia easily pick out the panels they need).

Additions are substantial changes which enhance the functionality of the tool or improve its ability to generalize to new use cases (e.g., allowing users to create their own level markup tags).

For a complete summary of the changes proposed, the reader is referred to Appendix D. Additionally, the prioritization of these changes and their potential impact on the tool's usability and applicability to different projects is discussed in Section 5.3.

5 DISCUSSION

In developing PathOS, we set out to provide a tool that could help any game creator to better inform the design of their game's world. Our next goal is to improve this tool and to make sure that developers who may benefit from PathOS can easily learn about and acquire it for their own use. Before moving forward, however, it is important to understand the tool's current state in terms of its usability, integration into the design process, and areas for improvement. To this end, our user study (described in Chapter 4) aimed to investigate these factors, asking *How can PathOS contribute to the game design process?* and *How can PathOS be improved to better suit the needs of game developers?* This chapter will reflect on the results of this evaluation and examine how PathOS can be used in various game development contexts.

5.1 USER STUDY RESULTS

Generally speaking, users' impressions of the tool were very positive, particularly regarding its usefulness as part of a developer's toolkit. Though the study we conducted was fairly preliminary in nature, participants represented our target users quite well, with development experience across projects and studios of all sizes (i.e., student, indie, mobile, AAA). Thus, their experiences with PathOS provide us with a valuable indication of how the framework performs in the hands of real users.

Returning to our initial impetus for the design of PathOS, recall the following primary design goals of the tool:

- **Easing the burden of playtesting:** We want PathOS to help in reducing the effort and resources needed to acquire data on how players will interact with a game. At the cost of providing an approximation, the tool could offer a dramatic reduction in time and labour costs associated with running playtests and recruiting participants.

- **Accessibility for developers:** PathOS is free and open-source, with the intention that any developer can use the tool. However, making the tool available is only the first step; it also needs to be easy to set up and integrate into a game project for developers to want to use it.

- **Usability for designers:** Since the tool is intended to provide insights that inform level design, it needs to be usable by designers, not just programmers.

PathOS' front-end interface is designed to be understandable and usable regardless of an individual's particular area of expertise in game development.

- **Generalizability:** We aim for PathOS to be capable of working in any game project that requires players to control a character moving around in a game world (e.g., FPS, RPG, action-adventure, etc.) with as little modification as possible.

With regard to reducing the costs associated with playtesting, participants commented positively on the time-saving potential of using the framework, as well as alleviating problems associated with recruiting human participants. In terms of developer accessibility, all participants were able to set up the framework without issue and were largely positive on setup, often describing their first-time experience as "straightforward". Participants also commented on the tool's interface, with several individuals describing the PathOS UI as "intuitive". While it should be noted that many of our participants possessed a technical background, no interaction with the back-end of the framework was required for our study, and participants explicitly described the tool as suitable for designers (P3: "[...] it was nice because I can see that it's a tool for designers"). Lastly, to the point of generalizability, participants were positive about the framework's versatility, providing a substantial variety of use cases when asked whether they might be able to use it themselves.

In general, we can conclude that this iteration of the framework has been largely successful in meeting its basic design goals when deployed with real users. To probe the tool's performance further, let us return to the inquiries posed at the beginning of our evaluation: *How can PathOS contribute to the game design process?* and *How can PathOS be improved to better suit the needs of game developers?* These questions are addressed in the following subsections.

5.2 HOW CAN PATHOS CONTRIBUTE TO THE GAME DESIGN PROCESS?

Based on the results of our evaluation, PathOS can effectively serve as a low-cost alternative to playtesting in the early stages of development with minimal barriers to entry, providing insights that can help improve a game's world design and synergize with different designer workflows. To understand how this contribution can be realized, we will examine our central question in three parts, concerning the tool's overall utility, application to different tasks, and integration with existing processes. First, we will review the benefits of the tool, and how it provides value for its users.

5.2.1 *How is PathOS useful?*

When discussing the framework's utility, participants frequently mentioned the time savings afforded by PathOS in comparison to playtesting with human users. Starting with the time spent during a playtest session, testing with agents negates the need for supervision, and allows for a simulation to be accelerated—something which is obviously impossible with human testers. Apart from the session itself, PathOS also saves time in terms of the additional tasks required to orchestrate a playtest. P3 describes traditional playtesting as a "much longer process" than testing with AI, noting the necessity of having to create a build suitable for testing, find a participant, and gather feedback before having the ability to make changes. Contrastly, in working with PathOS, they can "see results [and] take action on those results" in just a few minutes.

Participants also noted that the framework can save time when compared against testing their creation themselves. P4 described a coverage testing scenario, noting that testing as a human "would be really tedious and take a lot of time", and that having the option of AI to validate a design after each change made "scales the amount of saving time", accumulating benefits during development.

A particularly challenging aspect of playtesting is the recruitment process, another factor participants discussed as an obstacle overcome by using PathOS. P3 noted the difficulty of finding participants for a personal or independent project, saying that their "pool of potential playtesters is extremely limited". Furthermore, they note that any one tester is incapable of being used to evaluate first-time experience more than once, while "an AI is completely fresh slate." Another participant, P5, mentioned that recruitment can be especially problematic when working on a new project with strict non-disclosure requirements. Using AI as an alternative during the early stages of production can erase such concerns while still allowing designers to gain an approximation of how players will interact with their creations.

Another obvious benefit of the tool is its financial cost in comparison with other methods. Apart from developer time required for setup, learning, and running simulations, PathOS can be used for free. It does not require a dedicated testing environment, agents can run unsupervised, and no participants need to be compensated. P7 noted that the tool being free and open-source is "fabulous", saying that PathOS is "a good tool for people who want to start with user research but don't have the funds to do so."

PathOS' potential resource savings only provide us with a partial understanding of its usefulness. After all, if a tool intended for design feedback costs nothing, but also provides no useful information, it ultimately provides no value to users. Despite the fact that the framework provides only an approximation of player behaviour, however, participants did find observing agents' behaviours to be useful to their designs. As previously discussed, most participants made adjustments to the levels they created based on output from the framework. Among the few participants who did not make any changes, the vast majority still explicitly noted that they either thought the data gathered from agents could be used to tweak their design, or had specific ideas about what to improve based on agents' behaviour.

Though a few participants mentioned that agents did not always behave in a strictly humanlike fashion (P10: "I was confused like what the agent was doing"), or rather were incapable of reproducing the full spectrum of behaviour exhibited by human players (P8: "There are some crazy players and they do random stuff"), participants also commented positively on the ability of agents to reproduce humanlike qualities. This was true of both specific instances of agent behaviours (P9: "It was really interesting to see some of the [agents] go kill one of these enemies and go to the objective. That's fair, I've seen people to do that in real games.") and the traits exhibited by agents as a collective (P8: "I feel like it's good for estimating the general populace.").

Participants also commented explicitly on the value of the information they gleaned from observing and/or visualizing data from PathOS agents navigating a level. P6 mentioned that they "got a lot of feedback from the AI about the layout of the level," despite the fact that an agent is "still a far cry from how a player would actually interact." Indeed, although PathOS agents do not attempt to capture all the nuances of human gameplay, their behaviour can still serve to provide insights on where players will go, and what they will be tempted to interact with. P3 mentioned the framework's ability to answer questions in a design context, providing the example "Are players getting stuck in certain spots?" as something that could be addressed by a designer using the framework.

Lastly, the tool's generalizability was highlighted by participants as a valuable feature. Participants remarked that they thought the tool could be used for games ranging from exploration-focused adventures, to RPGs, to dungeon crawlers. Additionally, some participants noted that the tool might be potentially modified to work with even more diverse games (P4: "it's something that can be extended very easily", P6: "it might be interesting to expose an API so you can program your own agent with custom movement that follows your own rules for navigation"). However, this modification was not seen as strictly necessary to ensure versatility (P4: "For, probably a lot of games, it doesn't need to be extended").

To answer our initial question, PathOS provides value for developers by offering an approximation of player behaviour which can be used to extract actionable design feedback at an extreme cost reduction when compared with human usertesting. With this in mind, we can move on to identify the tasks and application areas for which PathOS is best suited.

5.2.2 *What can PathOS be used for?*

During the level design exercise, several participants made changes to their creations based on their observation of PathOS agents. The specific nature of these changes can help us to determine the types of issues PathOS can identify, and the tasks that designers would perform as part of working with the tool.

Several participants referred to the intended path or "flow" of a level when discussing how they used agent behaviour as feedback to inform their design. When creating a level or world, designers can have a relatively definite idea of how they want players to experience the space. That is to say, there is a plan of sorts in place for what players will see and when, the actions players will take, and the number of different options available for players to shape their own course through a game's world. PathOS provides an avenue to validate whether this plan will carry forward through the experience of real users, at least in the approximate. The tool's interface provides users with the ability to understand what agents are seeing, where they go, and what game entities they interact with. If discrepancies exist between the observed behaviour and the intended course of action, a design can be changed accordingly.

In keeping with this idea, several participants made changes to their levels intended to correct issues with the overall flow of a level. Some of these changes included removing unintended shortcuts, opening up closed-off areas to give players more options, or strategically placing level geometry to guide players towards an intended objective. The ability to sit back and observe agents in real time—viewing what the player would see and understanding an agents' mock mental model—makes PathOS especially well-suited to such tasks.

Other design choices participants used PathOS to inform included the placement of game resources and the identification of small technical issues. With respect to the addition of resources, a path commonly taken by agents en route to an enemy-dense area could serve as an obvious opportunity to place health pickups allowing players to prepare for or recover from a fight. Regarding technical issues, a few participants mentioned making small adjustments to correct bad geometry on the navigation mesh after noticing agents become stuck or squeeze through a small unintentional gap. Similarly, it was also proposed that the framework could be an easy way to identify poorly placed collision boxes.

Stepping back to examine the tool's use in a more general sense, participants were somewhat divided on whether the framework was more useful in the context of QA or playtesting-type applications. Most participants asserted that the framework could be useful for both, giving examples of different ways that the tool might be useful in each context (e.g., evaluation of "level flow" as a playtest-centric goal, or finding collision errors as a QA-centric goal). Since we originally designed PathOS as a UX-focused tool, we did not include any features meant to explicitly support QA tasks. Nonetheless, several participants pointed out that the tool could also be useful for QA, a surprising and welcome result.

With an idea of both the value that the framework provides and the tasks it can be used for, we are free to consider the last factor in assessing how PathOS can contribute to the game development cycle: its ability to fit in with developers' existing processes and toolkits.

5.2.3 *How can PathOS integrate into existing workflows?*

The integration process for a development tool should ideally require as little work as possible, to prevent excessive overhead from making its adoption infeasible. This process can be thought of as comprising a few distinct stages: initial setup, learning to use the tool, and fitting the tool into a regular workflow. The first steps of this procedure can prove incredibly daunting, frustrating, and potentially resource-intensive if a tool is difficult to learn or requires extensive technical modifications to function with a particular project.

Fortunately, participants commented positively on the tool's ease of setup and ability to work "out of the box". It should be noted here that the participants who completed the design exercise were all given a Unity project with the framework pre-installed so as to expedite the setup procedure. However, the one participant (P4) who did not complete the exercise, as they had independently integrated the tool into a separate Unity project of their own, remarked specifically on the ease of setup (P4: "it kinda worked out of the box [...] being able to download code that works, and does, you know, something, is already impressive [...] easy to see how to replicate the stuff in the video"). P4 noted that for setup with a new project, they felt the need to "dig into the code" to double-check which scripts needed to be present on certain objects. This suggests that a quickstart-type guide, similar to the handout given to participants who completed the exercise, might further improve the setup process.

All participants made positive comments in terms of their experience navigating the tool's interface and learning how to use various features, describing the UI as "intuitive" and "straightforward". Participants also liked the documentation resources provided, with a couple of participants suggesting that additional technical documentation and a concrete API could make the tool more useful for users inclined to modify or augment the framework's functionality.

With respect to workflow integration, based on participants' description of how they used the tool themselves, it is clear that PathOS can be used quite differently depending on the needs of the user. Some participants used the tool frequently throughout the process of creating their levels to validate each new addition, while others blocked out the entirety of their initial designs before switching to an evaluation phase of sorts. In either case, participants were able to gather feedback that allowed them to make informed changes to their design, supporting the idea that PathOS can be used as part of an iterative development process.

The tool's various features led to it being used in a few different modes of operation. Some participants exclusively watched agents in real time, others focused almost solely on visualizing behaviour after the fact, and still others used a combination of both approaches. Some participants orchestrated agents to meander about a level en masse, collecting a great deal of data, where others preferred to inspect the behaviour of a single agent navigating a level in real time. These approaches obviously differ in terms of scalability, with the more hands-off approach of simulating agents unsupervised and reviewing data after the fact being far more feasible than real-time observation for large worlds. Nonetheless, the fact that participants displayed this degree of variety in their use of the framework on the small scale of the exercise is promising in terms of the tool's ability to support users with different needs and development styles.

In summary, we can now provide an answer to the first overarching question governing our evaluation. PathOS can contribute to the game development process by providing low-cost predictions of player behaviour useful for informing design decisions. This is supported by its ability to adapt to different projects and the workflows of designers with varying needs and habits.

5.3 HOW CAN PATHOS BE IMPROVED TO BETTER SUIT THE NEEDS OF GAME DEVELOPERS?

Although participants' feedback on the tool was largely positive, discussion with participants was rich with suggestions to further improve the framework's usability and versatility. Throughout this subsection, we will briefly re-examine the features most liked by participants, usability and quality-of-life issues encountered, and changes discussed to improve PathOS. As a note, all of the changes illustrated here were discussed explicitly in interviews with participants. Some changes were suggested directly by participants unprompted, whereas others were initially suggested by the researcher as a direct response to a comment made by participant regarding an issue or opportunity for enhancement.

LIKED FEATURES. Participants were especially positive regarding the level markup tool, runtime interface, custom Unity Inspector panels, and path visualization. The workflow for level markup was described as easy and intuitive, with the visual design of elements praised as clean and understandable. This is especially encouraging, as the markup process is a necessary part of using PathOS, regardless of what other functionality users aim to explore. Other parts of the custom Inspector interface (e.g., visualization settings panels, agent personality customization UI) were similarly highlighted as intuitive and well-organized. Participants also appreciated the variety of settings offered for customizing features such as the visualization.

The runtime interface was described as easy to understand, and containing all of the information necessary to understand agent behaviour. Participants liked the visual design of the icons used to indicate an agent's understanding of its surroundings, the inclusion of the agent POV, and the ability to visualize an agent's trajectory using the mental map. Trajectory viewing through the included path visualization created from agent logs was also appreciated by participants as a way to quickly assess the flow of a level.

QUALITY OF LIFE CHANGES. Many of the changes discussed with participants were intended to improve "quality of life" in using the tool. That is to say, they aim to make completing certain tasks easier or less time-consuming, or represent small additions that require little technical overhead to implement (e.g., exposing additional information to users).

One of the top quality-of-life improvements suggested by participants was the inclusion of support for tagging multiple objects at once in the level markup tool. Currently, users are required to individually click each object they wish to tag with a given entity type. Allowing users to quickly tag a group of objects based on a hierarchy selection, or their Unity prefab type, would greatly expedite this process when large collections of interactive objects are present.

Another timesaving improvement suggested by participants is the inclusion of a fast forward option for individual agent simulation, similar to the timescale slider implemented for simulating batches of agents. Strictly speaking, this is already possible through Unity's project settings. However, including a shortcut to speed up the simulation within the PathOS UI would make the adjustment of time scaling for the purposes of testing much easier to access.

Participants suggested several improvements to the logging and visualization system which could make it easier to use and enable new analysis tasks. For instance, the process of setting up logging could be expedited through the provision of a default output directory, rather than requiring users to browse manually. A couple of participants indicated that they would like to have a way of quickly understanding an agent's orientation on the path visualization, suggesting small arrowheads as a way of conveying additional information. The ability to filter agents included in a visualization automatically by their simulated player profile was also suggested as a means to quickly compare differences in behaviour between groups of agents.

USABILITY CHANGES. Some of the changes discussed with participants were conceptualized in response to issues encountered by participants in using the tool or navigating its interface. These changes are thus intended primarily to prevent user error and make the user interface more understandable.

When working with the level markup tool, some participants commented that they would occasionally misclick on an object, tagging it accidentally. Two potential changes arose in response to such precision-related concerns: the ability to lock objects such as level geometry and prevent them from being tagged, and the addition of a small precision indicator on the markup cursor to improve user accuracy. Another small issue relating to the markup tool was that a couple of participants were confused as to why the markup brush would deactivate after panning the in-scene camera. This behaviour could be altered to restore the state of the markup brush after users are finished manipulating their scene view, allowing for a more seamless workflow.

A key component of the runtime UI and level markup tool is a set of custom Unity Gizmos, icons overlain on screen to display information about game objects. A few participants either had Gizmos off by default or had Gizmos configured in such a way that they became difficult to see under typical viewing conditions. Though this is a Unity-level setting removed from the PathOS tool, a warning could be included in PathOS to inform users of this eventuality, suggesting optimal settings to prevent users from not being able to see all of the information available. A further change suggested in reference to the runtime UI was the option to move its contents to a separate view, preventing elements of the interface from obscuring the game window depending on a user's desired view.

Lastly, a couple of users expressed that they initially forgot to enable logging in order to make use of visualizations (one participant did not realize that logging needed to be enabled for certain features such as the heatmap). Participants suggested that a warning or dialog to this effect could be included to ensure that users would remember to configure logging if they wanted to use the visualization features.

ACCESSIBILITY CHANGES. This category contains adjustments that would help to make the framework more usable for users with specific accessibility concerns. Changes of this nature were only discussed with one participant (P7), who works as a UX consultant and expressed during their session that they tend to be more accessibility-minded as a result.

While the PathOS interface was often described as straightforward by participants, it is quite text-heavy (a characteristic shared by much of the Unity interface in general). For users with dyslexia, this can pose issues in identifying which regions of the interface to interact with at-a-glance. The inclusion of icons in the Inspector window to delineate different panels was suggested as a way to remedy this problem.

The runtime interface was noted as another opportunity to make the tool's visual design more easily accessible. Though agents' mental map is high-contrast for users with normal colour vision, it is harder to distinguish different areas for those with common forms of colour vision deficiency (e.g., red-green colourblindness). To mitigate this issue, an option for an alternate colour scheme could be included to suit users with colour vision deficiencies. A comparison showing one such potential colour scheme is shown in Figure 8.

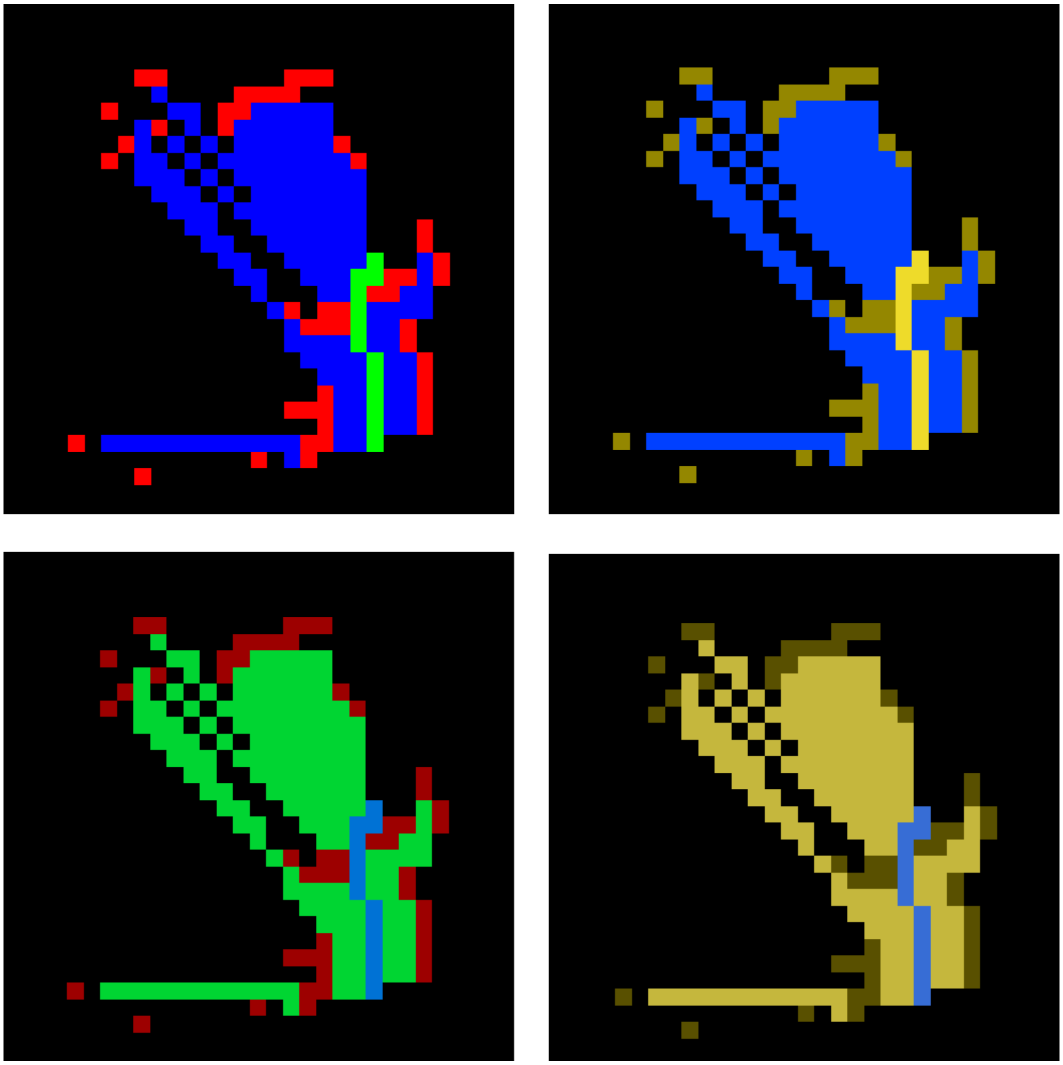


Figure 8: Two colour schemes for the mental map interface. Leftmost images display colours as seen by an individual with typical colour vision. Rightmost images display colours as seen by an individual with a common form of red-green colourblindness (deuteranopia). The top row shows the current colour scheme used (note that colourblind users may have trouble distinguishing between the visited (green) and blocked off (red) sections of the map at a glance. The bottom row shows an alternate scheme which could be provided as an option for colourblind users to enhance contrast.

TECHNICAL ISSUES. Though our evaluation was focused on usability and utility of the tool, rather than technical fidelity, we did take note of any technical issues encountered by participants. During the study, a small bug was fixed in the tool which improved agents' accuracy in identifying game entities as physically unreachable for players. Three participants also experienced some issues with agents occasionally becoming inexplicably stuck on a level's navigation mesh. However, this bug occurred rarely and was not significant enough to interfere with participants' ability to complete the exercise. Technical improvements to the robustness of agents' pathing, reducing the likelihood of such occurrences (except when they represent identification of an issue with level geometry), is part of our continued development.

FRAMEWORK ADDITIONS. Changes classified as additions represent substantial enhancements intended to alter the tool's functionality beyond its current scope. Several such changes were proposed by and discussed with participants during the study, primarily in the interest of making the tool capable of adapting to more complex game logic. For instance, multiple participants expressed a desire to support dynamic game entities; that is to say, the ability to support interactive objects changing throughout the course of a game. Such features might include support for spawners that would automatically tag game entities (e.g., enemies) as they are created at runtime, the ability to integrate with a quest or mission system (e.g., only making objectives available after certain items are interacted with), or objects that can change type at runtime (e.g., NPCs becoming hostile).

Another addition suggested to make agent behaviour more realistic in the context of a particular game was the ability to define custom resources which would factor into an agent's logic. For instance, agents might have a health resource which could decrease upon interaction with an enemy and increase when picking up a healing item. Such a system might integrate a basic form of reinforcement learning to allow agents to interact with a game's mechanics beyond navigation. In a similar vein, it was suggested that an API could be provided to allow for programming custom agent behaviours, enabling the tool to adapt to the unique needs of different game projects.

Lastly, several participants suggested that PathOS' Unity interface could be condensed into a central control panel, similar to other parts of the main Unity Editor (rather than having some settings contained on individual scripted objects). The provision of such an interface would at the very least offer users the option of more centralized control, and is thus a worthwhile addition to the core UI design of the framework.

The changes described here represent those with the highest priority, as they were either discussed with multiple participants or addressed issues with a comparatively high level of severity. However, a number of other small changes, particularly with respect to quality-of-life, were identified during the study and noted for future development of the framework. For a full changelist comprising all potential improvements to PathOS derived as a result of our user study, the reader is referred to Appendix D.

5.4 SYSTEM APPLICATIONS

As with any other development tool, a key factor in maximizing the framework's utility is to know when and how it should be deployed. It is also important to recognize which scenarios are most appropriate for its use, and likewise, which situations may not be suitable. The following serves as a brief overview of how the PathOS framework can be applied in the context of game development processes, answering the following key questions:

When should PathOS be used? This question concerns the general integration of PathOS into the development cycle. Key information addressed here includes when in the development cycle the framework should be used, what general research questions it can help to answer, and how it can be effectively blended with human testing at different points of development.

What projects should PathOS be used for? An obvious consequence of the nature of games as an interactive medium is that they vary wildly in design. Perhaps the easiest way to delineate these design variations is through categorization by genre, though even this can fail to capture a great deal of mechanical diversity. This question gives a relatively high-level overview of which game projects are most suitable for testing with PathOS.

How can PathOS adjust to specific situations? Clearly, the PathOS framework will not be suited, or at least not immediately so, to every development project. For those projects where it is nearly appropriate, or is limited to assessing a small portion of a game's experience, it may be possible to adapt the framework to better meet the needs of individual developers. This question explores how some of these challenges might be met to improve the range of projects for which the tool could be applied.

5.4.1 *Use Cases - When should PathOS be used?*

There are a few points in the iterative development cycle where PathOS could be used most effectively to provide designers with a means for informing or validating their decisions. Though these points may vary depending on the specifics of a particular project, in general, the tool is best suited to three main applications: live exploration of behaviour before a level design is complete, comparing design alternatives, and investigating the impact of changes.

LIVE BEHAVIOUR EXPLORATION. During the early stages of level creation, designers can create rough layouts of level alternatives, iteratively placing objects, examining the scene, and modifying level geometry on an *ad hoc* basis. At this point, quick tests with the PathOS framework could be used to improve designers' "best guess" as to how players will navigate certain areas. Individual agents could be observed to check for basic issues such as the visibility of interactive objects as they are placed, and whether players can suitably navigate different sectors of a level as they are created.

COMPARING DESIGN ALTERNATIVES. After the initial prototyping phase is complete, but before testing with human users, designers might use the tool to perform comparisons between different level designs. This could include assessing which levels are most suited to a particular player type (e.g., net the highest level of activity with interactive objects) or which encourage players to stay the longest before moving on, for example. Such information could be used to choose a subset of levels for playtesting, assist in the sequencing of levels during a playtest, or determine who should be recruited.

INVESTIGATING CHANGES. In between playtest rounds, the tool could be used to assess the impact of changes made in response to player feedback and identified issues. For instance, consider a case where the visibility of interactive objects was noted as a problem during playtesting. Designers might address this problem in several ways, for instance, by moving these objects, reducing visual clutter by removing some decorative objects, adjusting level geometry, or adding additional markers to point players in the right direction. By observing the impact of each of these changes on agent behaviour, designers could gain an indication of which of these adjustments best meet their design objectives, reducing the likelihood that an issue will persist into future testing rounds.

Outside of these applications, it is also important to note which UX questions the framework is most suited to address. Broadly speaking, we can place these questions into three groups: asking what players do, why they do it, and how this makes them feel.

The PathOS tool is only capable of addressing the first category of inquiries. It can be used to investigate high-level questions such as "Where will players go?", "How many players will interact with optional mission markers?", and "How long will players spend in this level?". Without game-specific adaptation, it is not suited to answering more specific questions in this category, such as "How many times will players die?" or "How difficult is this level compared to the previous level?".

Looking beyond these queries, the framework is not currently well-suited to addressing the why and how questions noted above. To address why-type questions, designers would essentially require agents to explain the reasoning behind their actions. While the framework contains a model of player logic, it does not express this logic explicitly, though some inferences can be made in a general sense (e.g., cautious players avoided a certain portion of the level since the hazard density was quite high, players did not collect a certain item because it never entered their field of view). Future work could explore how agent logic might be recorded for parsing by a human analyst after simulation.

With respect to how-type questions, PathOS makes no attempt to simulate users' emotional response to in-game events, or how their emotions might evolve as they progress through a level. Thus, it is not suitable for addressing questions regarding how players feel about their interactions—such inquiries should only be intended for testing with human users, or perhaps in the future, a far more advanced automated solution.

Lastly, it is crucial to note that PathOS is not meant as a substitute for playtesting with human users, but rather as an augmentation for existing workflows to provide designers with more information on player behaviour where it would otherwise be unavailable. In some cases, this unavailability may be a matter of resource limitations, such as not having sufficient time to playtest all candidate modifications to a design after an initial round with human users. In others, it may be due to the state of development; playtesting "blocked-out" level designs before a game's mechanics are in place might simply be infeasible. At any rate, the framework should be thought of as a supplement to existing testing approaches, rather than a replacement.

5.4.2 *Suitability for Different Scenarios - What projects should PathOS be used for?*

In general, the PathOS framework can be of use for any game where navigation in a virtual world forms a substantial part of the end user's experience.There are some exceptions to this rule—for instance, when navigation is contingent on complex player movesets (e.g. in *Mirror's Edge* (EA DICE, 2010)) and would thus require adaptation of the framework (see Section 5.4.3). Additionally, the framework would not be suitable where level geometry does not obey a consistent physical ruleset (for example, the non-Euclidean environment of *Antichamber* (Demruth, 2013)).

Beyond this simple caveat, it is difficult to prescribe a definite set of criteria for whether or not PathOS could be of use in any given project. Variation in game mechanics, particularly those related to navigation, may render the framework more or less appropriate even between otherwise similar game scenarios. Nonetheless, based on a broad categorization of genre, some general assumptions about the tool's ability to adapt to different types of game projects can be made. A high-level assessment of the framework's utility for different game genres is given below.

FPS (FIRST-PERSON SHOOTER) GAMES. Example: *Doom* (id Software, 2016). First-person shooters take place in 3D environments, and often require players to complete a set of definite objectives, such as navigating between entry and exit points, retrieving specific items, or killing specific enemies. Thus, PathOS should be generally suitable for predicting player traversal in such situations. However, it should be noted that a great deal of the challenge in FPS games comes from understanding enemy behaviour, managing resources such as health and ammunition, and negotiating combat scenarios. All of these factors can have an impact on player navigation as gameplay progresses. Since the framework does not account for interactions with a game's mechanics, it might best be used for validation of level playability in these scenarios (e.g., determining whether players can successfully find and reach key items or mission objectives).

ACTION/ADVENTURE GAMES. Example: *Assassin's Creed Odyssey* (Ubisoft Quebec, 2018). Many modern action games require players to negotiate 3D worlds as they complete missions, dispatch enemies, and interact with NPCs. Apart from the limitations imposed by a lack of support for considering individual game mechanics as noted above, this makes them generally suitable for use with the PathOS framework. Specifically, it could be used to investigate whether players adhere to designers' ideal path, or "golden path" in games with linear objective design, where players are expected to encounter a specific series of challenges without being explicitly pressed to do so.

RPGS (ROLE-PLAYING GAMES). Example: *The Legend of Zelda: Breath of the Wild* (Nintendo, 2017). For RPGs with 3D worlds, the framework is well-suited to predicting player traversal in an environment with multiple available alternatives. It can also be used to predict how players might negotiate more limited spaces, such as dungeons or challenge gauntlets with a linear path. This is especially true of Western-style RPGs, where exploring and navigating a complex world forms a substantial portion of the game's experience. For Japenese-style RPGs, or JRPGs (e.g., *Bravely Default* (Silicon Studio, 2012)), the focus is often on tactical, turn-based combat and narrative content as core drivers of the experience. For such titles, the tool is only suitable for assessing the navigation of any overworld or level maps which might be present.

PLATFORMERS. Example: *Hollow Knight* (Team Cherry, 2017). Most platformers grant the player a repertoire of specialized moves to use in conjunction with standard "run and jump"-type navigation. Accounting for the impact of these mechanics on player traversal is challenging, and thus the framework would generally be unsuitable for predicting behaviour in these games without some modifications (e.g., those mentioned in Section 5.4.3). Additionally, many platformers take place in 2D environments, requiring a restructuring of agents' spatial logic to function properly. However, for 3D titles (e.g., *A Hat in Time* (Gears for Breakfast, 2017)), use of the tool could be appropriate if accommodations were made for unique movement mechanics.

PUZZLE GAMES. Example: *Portal 2* (Valve Corporation, 2011). Due to puzzle games' reliance on players thinking through a variety of logical and often abstract problems, PathOS is generally unsuitable for these titles. Though many puzzle games do involve navigating a 3D game space, traversal is often heavily dictated by players' ability to solve these challenges (e.g., solving physics challenges to move through a level in *Portal 2* (Valve Corporation, 2011)). Nonetheless, the framework could be suitable for assessing player navigation in games where puzzle-solving is at least partially decoupled from traversal (e.g., individual areas in *The Witness* (Thekla, Inc., 2016)).

OTHER GENRES. Many games feature navigation or exploration of a virtual space as an integral part of the experience, as noted above. However, several genres of games are commonly completely divorced from the concept, and thus inappropriate for use with the PathOS framework. These categories of games include management or simulation games, in which players manage a system such as a business or group of virtual characters (e.g., *The Sims* (Maxis, 2000)); strategy games, in which players use a complex set of combat, diplomatic, or other tactics to outsmart an opponent or overcome some immense challenge (e.g., *StarCraft* (Blizzard Entertainment, 1998)); sports games, in which players compete in often true-to-life sports scenarios (e.g., *FIFA 19* (EA Vancounver and EA Romania, 2018)); and idle games, which typically rely on a series of repetitive actions with light strategy to accrue a vast quantity of game resources (e.g., *Clicker Heroes* (Playsaurus, 2015)). As a word of caution, this is not to say that such games will never be suited to automated testing; instead, they should simply be thought of as unsuitable for the approach described in this work.

Apart from the design qualities of a given game project, it should also be noted that the current framework prototype has some technical restrictions limiting its suitability. Namely, the tool is only suitable for Unity projects where a Unity Navmesh can be used—these limitations are discussed further in the tool's user manual, given in Appendix A.

5.4.3 *System Adaptation - How can PathOS adjust to specific situations?*

For situations where the framework is not immediately appropriate for assessing all or part of the core game experience, it could be adapted to work with specific projects. At the cost of additional development time, support for unique level geometry, physics, or different game mechanics could be added.

To adapt to unique level geometry or physical rules (e.g., teleportation pads, reversing gravity, etc.), the pathfinding logic of the framework could be swapped out to use a custom solution, rather than relying on Unity's Navmesh API. Depending on the situation, this could be a relatively trivial process, if the developer already has a solution in place for driving the pathfinding logic of any in-game

AI. Regardless, the modular design of the framework means that its pathfinding logic can be exchanged with relatively little development effort.

Addressing the question of game mechanics, this might take the form of augmenting an agent's navigation "moveset" and adjusting pathfinding accordingly. For example, players' ability to fly, execute dashes, or use devices such as a grappling hook in game could be added to agents' low-level navigation logic.

It might also be desirable to simulate other mechanics unrelated to navigation. For example, a developer might want to estimate the outcome of combat scenarios, in the interest of predicting deaths of the player character to gauge difficulty and the potential for frustration. This could be accomplished through the direct simulation of combat mechanics (e.g., shooting, melee combos, etc.), though this would likely require a significant technical effort. Existing AI behaviours for enemies or allies in-game could be used, though this would fail to capture the variation between different player types. Alternatively, developers might consider implementing a simple calculation for estimating the outcome of different combat scenarios based on factors such as enemy type and key agent motivations (e.g., aggression, experience), to provide an automatic, coarse prediction of the result of encounters in a level. Similar predictors might be used to estimate the impact of other mechanics on eventual player experience, such as the time lost in solving a particular puzzle.

For projects built on a different platform (i.e., using a different game engine), the framework could be ported to work with a different navigation back-end and physics API. It could also be adapted to work with an existing system for visualizing playtesting data, rather than the utility provided with the framework. This might be of interest to larger or more established developers with in-house engines or data analysis tools looking for extensions to their existing processes. Future work might also explore whether the overhead associated with creating custom automated testing solutions tailored to individual large (i.e., AAA) projects is a favourable sacrifice, given the much larger budgets and far larger-scale testing schedules of such titles.

Part III

CONCLUSION

Reflections on this work and moving the field forward

6 CONCLUSION

The appreciation of games, like any other art form, is nuanced and complex. Enjoyment of such media is brought about by a cocktail of immersion, escapism, accomplishment, and the ever-elusive concept of fun—among a myriad of other emotional, physical, and mental factors. Reflecting on this notion, it is far from surprising why an understanding of user experience has proven so critical in the modern games industry.

As in so many other fields, we are now driven to question how we can take this vital part of the game development process and optimize it. Such motivation in the industry may be driven purely out of commercial interest or even desperation, but also appeal to scientific curiosity and developer creativity. One simple solution, born out of a need to cut resources without cutting corners, can form the foundation for future innovations. Consider the case of procedural content generation (PCG) [140], whereby part of a game, such as level geometry, is generated algorithmically. On the surface, PCG is a tactic for overcoming resource limitations. From a creative standpoint, it saves on human labour by no longer requiring designers and artists to create everything seen in a game from scratch. From a technical perspective, it reduces storage requirements by only needing space for the content-generation algorithm, rather than the content itself, which can be produced and reproduced at runtime.

These pragmatic benefits provided a clear advantage to early games using PCG, such as *Elite* (Braben and Bell, 1984) and *Rogue* (A.I. Design, 1980), with small development teams and severe computational limitations. Today, such advantages are still important, allowing for indie developers to deliver on more content than they could hope to author by hand (e.g., world design in *Terraria* (Re-Logic, 2011)) or generate boundless worlds where persistent storage would challenge even modern supercomputers (e.g., in *No Man's Sky* (Hello Games, 2016)). However, PCG can also improve UX by creating content beyond what a human designer might imagine [140]. Today, designers have experimented with entire rulesets devised by AI, pushing the boundaries of how generated content can shape player experience [141].

It is our hope that AI-driven testing can spark a similar spirit of innovation, forming part of the path towards our next milestones in game AI and understanding user experience. Could replicating the frailties of human perception and memory help to make for more convincing in-game AI partners and adversaries? Might a more holistic approach to simulating play help in the quest for more general game AI? Can we strive to simulate more challenging components of player behaviour, or perhaps the emotional and mental consequences of a particular game experience?

Attempts to answer such questions are predicated on the existence of a foundation for simulating human play behaviours. The ultimate goal of this work is to contribute to such a foundation. Presented here is a framework for simulation-driven testing aimed at reproducing human navigation behaviours. The concluding thoughts that follow serve as a reflection on its successes and failings, and provide some reasoned speculation as to how this research might be expanded upon in future investigations. In creating this tool, and inspiring further research in AI-driven testing, it is our hope that this work can help to drive the field forward.

6.1 LIMITATIONS

Given the potential scope of AI-driven user experience evaluation as a concept, we must tread lightly in drawing conclusions regarding its utility in general on the basis of a single newly-developed system. This work is just that: an initial prototype and exploratory evaluation of a tool to assist in evaluating game world design. Thus, it is impossible to make definitive statements on the success of automated testing for games in general, or to reliably predict the value of related endeavours, such as the simulation of players' emotional reactions or more complex gameplay behaviours (e.g., combat, puzzle-solving).

It is also important to be cognizant of the fact that the evaluation presented limits our ability to fully validate the system developed, both in terms of its technical fidelity and utility to developers. The chief limitation of this work is that we have yet to conduct a rigorous evaluation of the framework's accuracy; that is to say, we cannot yet prove that the behaviour predicted aligns with the actions of real human users. We are also limited in our ability to evaluate the potential use of the framework in the long term, due to the short-term use explored in our user study.

Evaluating similar systems has proven a challenging cause for researchers working on similar projects aimed at reproducing humanlike behaviour in game environments, as discussed in Chapter 2. This difficulty arises from a number of factors; for instance, preventing the accumulation of error when comparing human and agent behaviour. Furthermore, comparison necessitates that the "personality" of an agent matches its human counterparts; this presents the further challenge of accurately assessing the motivational profiles of human players. Such agreement might require the deployment of typology or motivation questionnaires (e.g., [41]), or an analysis of previous play behaviours to categorize player type. Lastly, even with an adequate strategy for behavioural comparison, the task of evaluating individual agents is further complicated by the fact that chaotic, imperfect behaviour is often the target when mimicking human play.

Apart from validating the accuracy of agent behaviour, the presented evaluation also focuses on a subjective evaluation of tool utility. A more complete evaluation of its deployment could include a comparison of the design process and its output between designers with and without access to the framework. This might include simple objective measures, such as the time taken to create and refine level designs, or the density of objects placed in created levels. Most importantly, it should include an evaluation of UX with human users for the levels crafted. This would allow us to draw conclusions about whether, for example, the tool could help designers to create better world designs, or create equally suitable designs while saving time.

The completion of such an evaluation, fully addressing the remaining concerns of behavioural accuracy and tangible process improvement, is left to future work. Nonetheless, results from our user study indicate that the existing prototype is largely successful in providing designers with a low-cost, easy-to-use option for informing their level design during the development cycle.

6.1.1 *Limitations of the PathOS prototype*

The current prototype of the PathOS system is intended primarily for use during the initial stages of level design, when designers may "block out" level geometry and place initial collections of objects for players to interact with. As previously noted, the tool could also be used between design iterations later in the development process. However, in its current state, the tool is somewhat restricted in its ability to adapt to the needs of different projects, or more complex design scenarios.

First, the prototype only works well with planar level layouts (e.g., outdoor terrain, single-floor layouts), as agents' spatial memory is represented two-dimensionally. This could be extended to a voxelized, three-dimensional representation to better support complex world layouts with overlapping layers of content (e.g., urban environments).

Additionally, agent logic is based on interactive objects being largely static; that is to say, fixed in position and always-available. During the initial phases of level design, before game logic is implemented, this is an adequate approximation. However, when trying to account for how players will actually experience content, this will not always be sufficient. While interactive objects can frequently remain static in games (e.g., characters which stand in place, collectibles hidden in the environment), many are dynamic in nature (e.g., enemies on patrol, mission markers that unlock only when certain conditions are met).

To support predicting "true-to-life" player behaviour in such situations, the framework would need to adapt to the presence of dynamic interactions accordingly. For instance, to reason with moving entities, agents could mimic the imperfect process of trajectory estimation. To deal with conditionally available entities, such as mission markers, a simple set of rules could be exposed for designers to specify when entities should be activated or deactivated.

Overcoming these restrictions would help the framework to simulate player navigation in a wider range of game situations. However, the evaluative power of PathOS is still inherently limited by its scope as a navigation-focused tool. To provide a more complete prediction of player behaviour, agents would need to interact with a game's mechanics (e.g., combat, special movement abilities, etc.). Such interactions are omitted from this work in the interest of both maintaining feasibility and preserving generality between different game genres and specific projects.

Aside from its restrictions in terms of supporting complex level designs and different game scenarios, the framework is also limited in terms of its perception and memory models (described in detail in Chapter 3). For instance, entity visibility is based on a geometric representation of level contents, failing to account for the influence of factors such as colour, shape, and contrast. Spatial navigation is based on a "mental map" rather than landmarks, and memory decay is implemented as a rudimentary check of the time elapsed since exposure to a given entity. The choice to simplify these models was made in the interest of feasibility. Implementation of more complex models may serve to create more accurate predictions of player behaviour. Alternatively, the explicit creation of such models may be foregone by using a machine learning approach. Section 6.2.1 discusses how this framework could be extended or re-implemented with ML, and the challenges which could arise in doing so.

6.1.2 *Limitations of game testing with AI*

It is impossible to say to what extent AI-driven testing may eventually supplant traditional usertesting approaches. Perhaps in the near future, a combination of increasingly sophisticated gameplay agents and growing computational power may make it possible to identify most all QC issues in a game via brute-force—locating physics bugs, logging crashes, and so on. Conceivably, with further exploration in the realm of player prediction, evaluation of game difficulty and retention will also eventually be possible through automated approaches alone. This may soon become feasible as more data gleaned from today's massive player populations becomes available, and machine learning approaches more common. Stepping back from speculation, however, AI-driven testing, particularly in its current state, has a few key drawbacks.

First, to be able to assist in evaluating the complete game experience, an AI testing agent should be able to interact with all of a game's mechanics; in other words, they should have the exact same tools at their disposal as a human player. However, today's systems generally require heavy modifications to adapt from one game to another, even within the same genre. Such an accomplishment would thus require a form of general video game artificial intelligence (GVGAI)[1] capable of interacting with games which the system has not previously experienced [142]. GVGAI remains an unsolved problem as of this writing, though previous work has explored the creation of game-agnostic agents for simple arcade-style games specified in a common language [143].

Without AI capable of transitioning between games with relative ease, the required system adaptations for use in multiple projects means that AI testing still carries a resource burden for developers. Additionally, the technical nature of this burden requires specialized labour to overcome. This may keep the widespread adoption of AI-driven testing limited to sizeable members of the industry for the near future.

There are also some aspects of UX evaluation which, due to the nature of AI, will conceivably prove challenging in the course of developing fully-automated testing solutions. For instance, even if player behaviour can be predicted perfectly, a key aspect of GUR is understanding users' emotional responses during play. Given the complex nature of human emotion, attempting to infer a players' feelings in reaction to game events is a monumental challenge. Context can make all the difference in the impact of a game event. Consider the instance where a player's character is killed. This may be frustrating (causing the loss of progress and having to restart a challenging gauntlet of foes), relieving (allowing for a break in the action to regroup), amusing (in a hectic multiplayer game with friends), or even encouraging (if accompanied by a few kind words and a quick, penalty-free restart of the current challenge).

Even if a system were trained to predict the likely emotional consequence of a sequence of events (e.g., using a statistical model built up in a fashion similar to that described by Roohi et al. [25]), it would be an entirely different matter to extract the reason why such a prediction was made. A key advantage of human participants is that they can explain their decisions and reactions in plain language. Current AI systems, especially machine learning systems, can be incredibly opaque. Without the ability to break down the reasoning behind a player's emotional response in terms a human can understand, tuning a game's experience is made far more difficult.

Similar challenges may arise in attempting to automate evaluation of arguably more nuanced aspects of a game's design, such as art style, sound design, and narrative content. On top of the advanced sensory and cognitive models required to assess such features in the way that a human might, explaining the reasoning for the assessments made would present similar obstacles to those described above. Additionally, it is uncertain how such a system could model the impact of subjective individual differences (such as a predisposition to certain styles of music, favourite colours, or nostalgia for characters present in other works), the nature of which within a game's target audience may be unknown.

[1] Here used to refer to the concept of such an AI in general, this is not to be confused with the GVG-AI development framework referenced in prior chapters.

Regardless of these challenges, AI has already proven useful as a new avenue in game testing, particularly in QC applications [22, 23] and as an assistant of sorts in UX evaluation [25, 30]. As the field continues to progress, more complex applications of AI-driven testing may become feasible, reshaping the way iterative development functions in the games industry.

6.2 FUTURE WORK

Moving forward with the development of the PathOS framework, our first steps will be implementing the improvements discussed in Section 5.3 and extending the tool to support some of the more complex game scenarios described in Section 6.1.1. This includes support for dynamic level geometry and entities, and better adapting to unusual or layered level terrain with volumetric representations of space in agents' memory. Additionally, we plan to add a ruleset designers can use to specify relationships between entities in the scene. For instance, developers may wish to create a chain of mission markers, each unlocking the next, to simulate a linear sequence of objectives (e.g., in an action game or FPS). Different rules within the system could be used to facilitate such tasks, for instance, by toggling the interactive state of level entities based on agent interaction with other entities in the scene.

In addition to these general extensions of the tool, we would also explore the possibility of adding support for basic interaction with a game's mechanics. To remain as game- and genre-agnostic as possible, this could consist of a library of common possible actions (e.g., swinging a weapon, using a healing item, talking to an NPC) which would be triggered contextually based on the entities an agent interacts with. These actions could have an associated risk and estimated completion time associated with them, allowing designers to roughly estimate variables such as time spent in a level or the frequency of player death.

We also wish to pursue a more thorough evaluation of the system's capability as a design aid and validation of agents' behaviour as a suitable predictor of human players. To this end, we would work with a new or existing game project to compare the traces of human players with those from PathOS agents. Additionally, we would recruit a group of level designers to create game maps with and without the framework, and assess whether any tangible differences in quality or creation efficiency arise when using the tool.

Ideally, we would also work with a commercial partner, such as an indie development studio, to prove the tool's utility in a "real-world" context. This could take the form of an extended case study, examining the amount of work necessary to adapt the framework to the needs of a specific game project (e.g., one with unique movement mechanics). Moreover, it would allow us to investigate when the tool is most useful during the development cycle. We could also use this as an opportunity to discover what extensions to the tool would be most beneficial to developers.

6.2.1 *Augmenting the PathOS framework*

To provide a more accurate and robust model of human player behaviour, several extensions could be made to the framework in future endeavours. Separating the system into its component parts (perception, memory, planning, and navigation), each in turn could be modified in the interest of improving accuracy and system generality. Agents' basic "sight" model, for instance, could be overhauled to allow designers to check for visibility issues caused by aesthetic choices or visual clutter. Memory and planning, currently implemented as an expert system, could be replaced with an ML-based approach to work with existing tools for large-scale collection of player metrics. Navigation driven by pathfinding and automatic locomotion could be replaced with direct input manipulation, reducing the control gap between AI and human players.

Starting with agent perception, entity visibility and immediate spatial perception could leverage a computer vision system, as opposed to raw geometry information taken from the game engine. Such a system could be trained for object detection on collections of screenshots, using detected entities as candidate destination points. Depending on the amount of training data available, the system could then attempt to classify detected entities (e.g., friend or foe) or interface with the game engine to determine the nature of seen objects. Tactics such as simulated visual attention [144] could be used to more accurately mirror the way humans recognize patterns in images. By applying rules such as contrast thresholds and minimum detected object size, issues with visibility could be identified. Furthermore, developers could note any decorative or "background" elements misidentified as interactive, to help avoid player confusion.

Navigation could also be altered such that agents would simulate player input, rather than directly controlling their position in-world. This could be used to better support game-specific movement systems, serving to validate playability as a side-effect by, for example, identifying areas unreachable with the move-set provided. Additionally, limiters simulating player reflexes could be imposed to evaluate the difficulty of traversing certain areas.

Depending on the nature of abilities available to a player and their interaction with a game's other mechanics, employing a reinforcement learning system to ensure agents' competence with a game's moveset may be desirable. The use of reinforcement learning as a mechanism to drive gameplay agents has been heavily explored in previous work (e.g., [21]). After a basic level of interaction competence is achieved, these agents could then be customized to reflect higher-level variations in behaviour and planning (i.e., motivational profiles).

Lastly, the framework could be adapted to learn from existing player data, rather than employing rule-based logic to drive agents' behavioural planning. This approach would require massive amounts of player data for training, such as the metrics datasets collected by enterprise-scale developers. Previous work has used these datasets for analyzing behavioural clusters [68], predicting retention [35], and driving imitation-learning systems to mimic individual styles for in-game AI [102].

If using an ML-driven approach based on real player data, several different incarnations could be explored. Data could be generated in aggregate, rather than through direct gameplay simulation (e.g., by visually generating playtraces given level geometry and conditioned on player profile data, based on existing examples). Imitation learning could be used to drive moment-to-moment decisions in gameplay based on the playing style of individual humans.

Regardless of the implementation desired, without GVGAI, such a system would be entirely dependent on access to training data directly applicable to the situation at hand (or suitable for transfer learning). Thus, for the time being, an ML-based approach would be most suitable for iterating on existing content, creating new content for an already released, active title, or working on a similar project with shared mechanics and gameplay logic (e.g., the next title in a franchise).

6.2.2 *The future of AI-driven testing*

Looking beyond how this work may be directly expanded upon in the future, we can also speculate as to how related work could evolve in the coming years. At a high level, the framework described here aims to help designers answer the question "What are players going to do?". By comparing predicted player action with intended player action, developer intention can be validated, or a design can be adjusted to encourage players to take the intended course of action. Going forward, automated testing approaches might attempt to answer the questions of why players would take a certain action and how this makes them feel as a result. If such inferences are available, this could provide designers with a more complete understanding of how specifically a game or its content could be modified to shape the experience of the end user as desired.

In attempting to answer questions of player reasoning and emotional responses, automated approaches may transition from purely predictive mechanisms into those that provide some form of critique. This notion has already been explored by Holmgård et al. [30], though it is unclear how such a system might best present agent "reasoning" or draw high-level conclusions regarding the overall quality of a game experience.

Future work may centre on the simulation or evaluation of game subsystems, much in the same way this work focused on the evaluation of level design from a player navigation perspective. For instance, behavioural simulation in combat may be used to assess difficulty and engagement across different player skill levels. Player combat simulation through reinforcement learning, for instance, has already been investigated as a testing mechanism in commercial titles [145].

AI-driven testing could also be used to attempt more subjective "opinion-based" evaluations, such as critiquing a game's art style or the aesthetics of in-game landscapes. Existing services (e.g., Everypixel[2]) already leverage ML approaches to evaluate the aesthetic quality of photographs, for example. Conceivably, a system oriented towards critiquing game aesthetics could be trained using style influences specified by the developer, such as sequences of screenshots from other games used as artistic inspirations.

While still far out of reach today, one can eventually envision that AI will provide a complete set of utilities for game user experience evaluation. Perhaps automated agents will eventually allow designers to answer each of the core questions surrounding game UX in turn—What are players doing? Why are they doing it? How does this make them feel? Perhaps one day we will not only be able to "ask" a system how players would progress through a game, but why a certain character might come off as controversial, or a given level seems more beautiful than the rest.

Perhaps one day, there will be no limits on what we can glean from our simulated counterparts.

6.3 CONCLUSION

The introduction of video games as an interactive medium has forever changed how we define and experience entertainment. In around just half a century, games have evolved from monochromatic lights playing across the screen of an oscilloscope to heavily immersive, detailed experiences often indistinguishable from reality at first glance. The complexity of interaction with these systems has grown accordingly, necessitating a better understanding of how humans experience them. Thus, games user research has grown as an area of both academic and commercial interest, helping us to gain insight into human behaviour, our relationship with technology, and how we can create better interactive experiences for one another.

A cornerstone of GUR is observing how humans interact with game systems through playtesting. Playtesting is a critical component of both academic and commercial games research, but it incurs immense time and resource expenses due to the nature of testing with human participants. Motivated by a desire to overcome these obstacles, the PathOS framework predicts player behaviour through the use of AI agents as proxies for human users. More specifically, it simulates player navigation, so that designers may evaluate the ability of game worlds to guide users to intended gameplay objectives, resources, discoveries, and so on.

[2] Everypixel Aesthetics: https://www.everypixel.com/aesthetics

While AI testing is still a nascent subfield of GUR, work within the past decade, and particularly the last few years, has explored the development of automated approaches for applications in both game QC and UX evaluation. An overview of existing research, laying a foundation for this work, was given in Chapter 2. Prior research can be classified broadly into approaches for supporting data analysis, and those for simulation-driven testing. This work is positioned within the realm of simulation-driven testing applied to UX evaluation, by allowing game creators to compare intended and predicted player behaviour to improve level and world design.

Our core goals in developing the PathOS framework are accessibility and generality. It is our hope that the increasing democratization of development tools will help to prevent a lack of resources from limiting the ability of independent creators to express their creativity. Thus, PathOS is an open-source utility for a freely-available engine, rather than a proprietary commercial tool. Additionally, the tool is as game- and genre-agnostic as possible, at the expense of simulation complexity (i.e., not tuned to interact with the mechanics of a particular game). Any developer can use PathOS to evaluate their designs, so long as navigation in a 3D environment is a component of gameplay. Details of the technical design and development of the PathOS framework are provided in Chapter 3.

An evaluation of our current prototype, outlined in Chapter 4, revealed that game developers are successfully able to use PathOS as part of their design workflow. Participants responded positively to the tool not only in terms of its user interface and feature set, but its general utility and the value it can provide, particularly for projects where resource limitations are a significant concern. From this evaluation, we were also able to gather a great deal of ideas on how the framework can be modified to improve its usability and generalizability to a wide variety of game projects.

In its current state, the PathOS tool is still, fundamentally, a prototype of simulation-driven testing. Future work should expand its core functionality, improve its generalizability, and investigate how it can be adapted to support specific game scenarios without sacrificing portability. As simulation-driven game testing in general continues to advance, researchers can move on to answer more complex questions of user experience with the help of AI, shortening development cycles and helping to ensure games are tailor-made to their exact target audience.

Beyond the realm of GUR, it will be fascinating to see how the relationship between games and AI continues to grow and change in the coming years. Historically, games have served as a testbed for novel AI systems, as they still do in providing increasingly complex scenarios. However, while games have nearly always supported the growth of AI, the converse has become true as well. AI can serve not only as a player, but an adversary or friend to human users, an author of game content, or even a game's designer.

It is a near certainty that AI will continue to change the way we develop, evaluate, and interact with games, though it is a mystery as to exactly how these changes may manifest. As we begin to explore the notion of technology as a collaborator of sorts, rather than simply a static tool, the possibility space of design will continue to grow. Hopefully, in the near future, this will enable us to bring marvellously engrossing experiences to life, the likes of which are scarcely imaginable today.

Thank you for reading.

Part IV

APPENDICES

A

USER MANUAL

A copy of the PathOS user manual is included in the following pages. This document was provided to participants of the user study described in Chapter 4 for their reference. It is also available on the project's Github page for users of the tool to download.

PATHOS

User Manual

What is PathOS?

PathOS is an end-to-end, lightweight framework for simulating player behaviour. PathOS agents approximate player navigation in a game's world, and can be viewed in real-time or recorded for later visualization. Agents can also be customized to mimic different player motivations.

PathOS is built for Unity, and designed to operate on top of your existing game projects, requiring no instrumentation or modification of game assets or code.

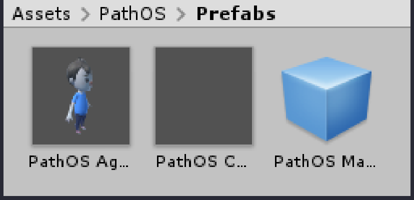
Happy playtesting!

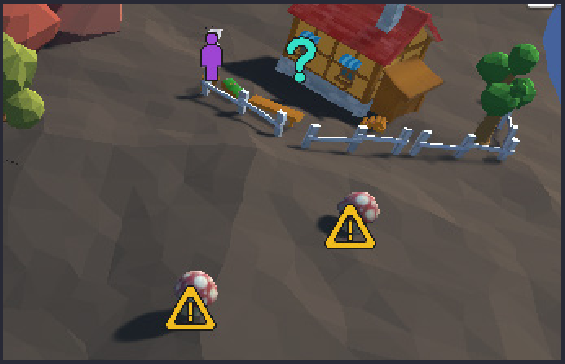
Contents

Quickstart Guide

First thing's first

Set up the Unity Navmesh if you haven't already from Window > AI > Navigation. Make sure you have PathOSManager and PathOSAgent objects in the scene - prefabs can be found in PathOS/Prefabs.





Level Markup

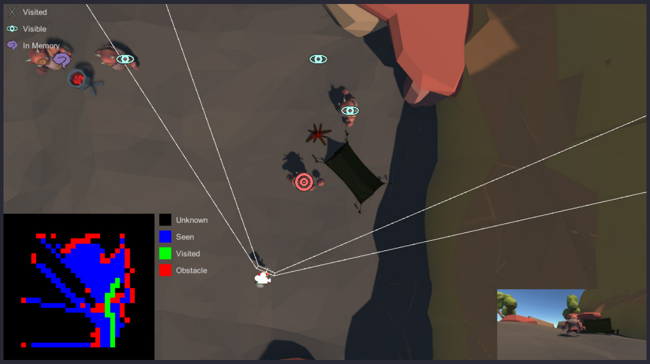
Use the **Level Markup** tab of the Manager Inspector (see *Level Markup*) to label important or interactive objects in the scene (e.g., enemies, collectibles). Tag any objects that would be indicated on a player's compass or minimap with the "Always Known" flag in the Manager's Entity List.

Agent Set-Up

Adjust the motive sliders on the Agent object to reflect the desired profile, or select a profile from the available presets and apply it to the agent (see *Agent Customization* and *Custom Profiles*).

Running the Simulation

Hit play to start the simulation and you can watch the agent navigate in realtime. Select it in the hierarchy to view an onscreen overlay showing its targeting logic, mental map, and player view (see *Runtime Interface*).



Extras

You can run multiple agents automatically as part of a testing batch (see *Batch Simulation*) and record data for later review and visualization (see *Data Recording & Visualization*). For a complete explanation of the system's components and Unity Inspector properties, check out the system reference pages!

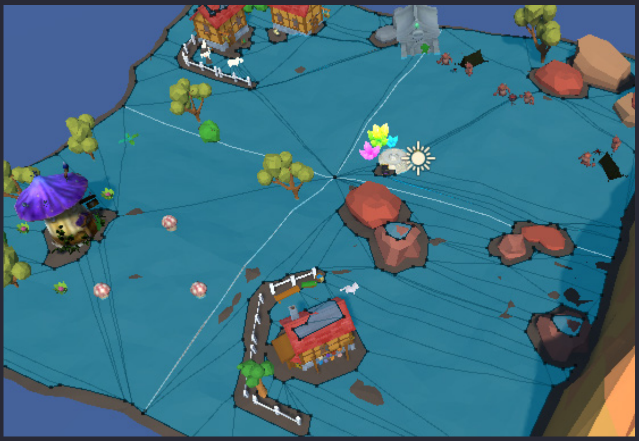
Project Set-Up

Demo Project

The included demo project and prefabs are set up to work "out of the box" - all you'll have to do is hit the Bake button in Unity's Navigation panel to re-bake the Navmesh after changing the level layout.

Will my project work with PathOS?

Hopefully! If your game involves players moving around a 3D world, the answer is probably yes. As long as you can bake a Unity Navmesh for your scene, you can use PathOS to test your level designs. However, the tool has its limitations - be sure to read "A Few Caveats" below to see if the framework is a good fit for you.



Navmesh

PathOS Agents work with Unity's Navmesh system for pathfinding. For the tool to work, you'll need to bake your Navmesh from Window > AI > Navigation. If you're starting from scratch, make sure that the baked agent settings (height, radius, etc.) match the settings on the Unity NavMeshAgent component of your PathOS Agent prefabs and GameObjects.

A Few Caveats

Tool Usage. PathOS is a tool for simulating navigation, not complete gameplay. Agents' navigation is dependent on the contextual information you provide - which GameObjects are collectibles, enemies, goals, and so on - but agents don't interact with your game's mechanics. Agents can't fight enemies to tell you if your combat system is too difficult. At least, not yet.

Level Layout. Agents navigate in 3D, but their spatial logic is planar. This means PathOS works best when your level is mostly lain out at a uniform altitutde with a defined ground plane - or when you can test your level in separate, mostly flat sections. PathOS will not work properly for levels with vertical layering.

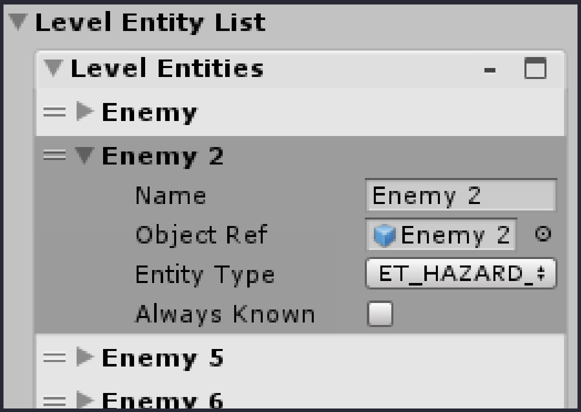
Visibility. Whether or not agents can "see" game objects is based on Unity's physics system, so anything you want to occlude visibility should have colliders attached. Visibility is calculated approximately, so don't be surprised if what you see through the player's POV camera doesn't match up exactly with the agent's logic.

Level Markup

For agents to navigate in the context of your game, they need to understand which objects in the level can be interacted with, and what purpose they serve. To tag game objects, use the **Level Markup** tab of the PathOS Manager Inspector.

Markup Brush

In the **Level Markup** tab of the Inspector, you can click on one of the entity types to activate tagging for that type (your cursor will change). In the scene, click on objects to tag them with the selected entity type. You can also use this mode to change the tag on already labelled objects, or clear tags from labelled objects.



▼ Level Entity List

▼ Level Entities

▶ Enemy

▼ Enemy 2

Name: Enemy 2

Object Ref: Enemy 2

Entity Type: ET_HAZARD_

Always Known: □

▶ Enemy 5

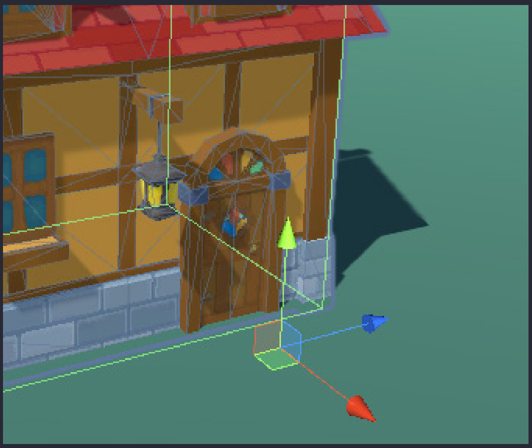
▶ Enemy 6

Entity List

You can also edit tags via the **Level Entity List** tab. You can add and remove tags using the '+/-' buttons at the bottom of the list. Here, you can also change the GameObjects referenced by each tag, or set an object as "always known" (mimicking the effect of entities which would be indicated on a player's minimap or compass).

A Note on Entity Locations

When agents target and visit game entities, they use the position of the Transform on the tagged GameObject. For large objects with colliders attached (e.g., buildings), a parent or proxy GameObject with the desired "visit location" should be used as the tag reference. The prefabs included in the demo project have already been set up with parent GameObjects with suitable pivot points chosen.



Game Entity Types

There are nine different tags available for level objects during the markup process. Here is a summary of their meaning. Type tags are used by agents to help drive their navigation through the game world.

Optional Goal. In-game missions or objectives that are optional. (e.g., sidequest marker)

Mandatory Goal. Objective that must be completed to finish the level. (e.g., main mission marker)

Final Goal. Objective that would allow the player to complete/exit the level, if applicable.

Collectible. Item that can be collected in game for achievement value. (e.g., treasure)

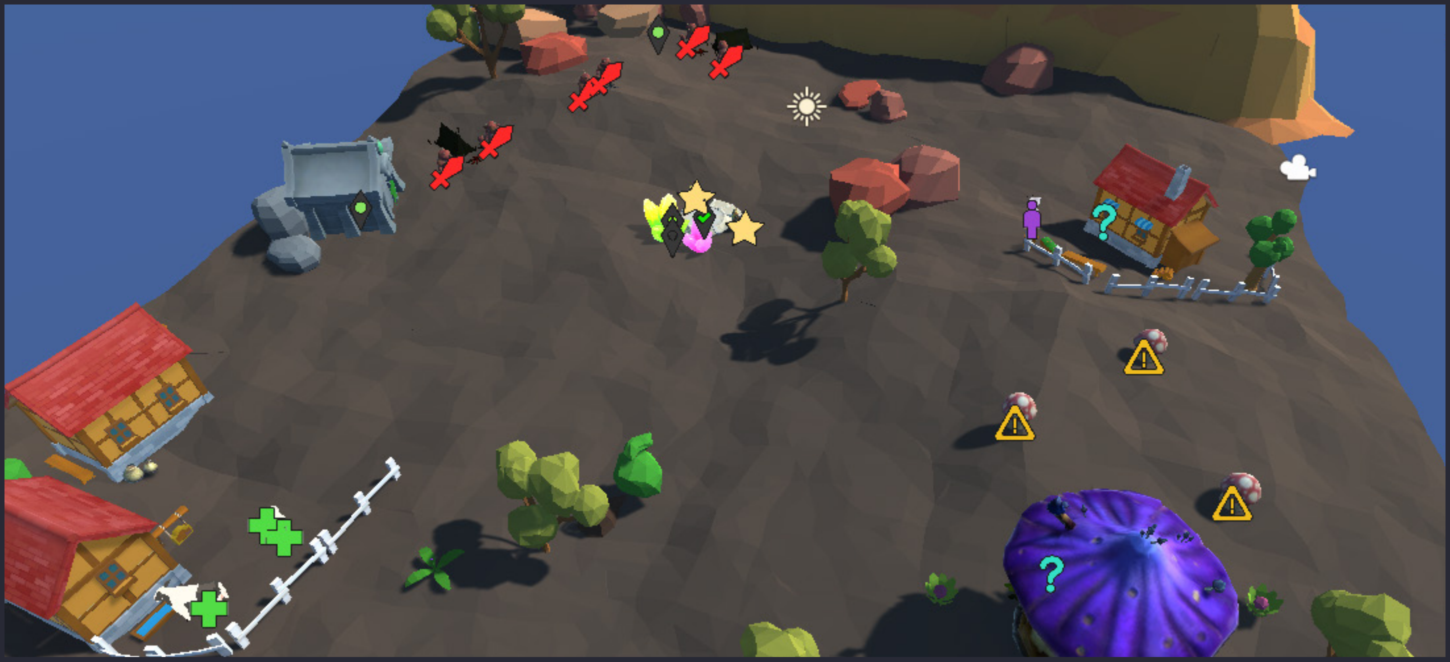
Self-Preservation Item. Item that can be collected to boost player survivability. (e.g., health/ammo)

Enemy Hazard. Hazard that could result in a combat encounter if engaged. (e.g., monster)

Environment Hazard. Interactive hazard that will not result in a combat encounter. (e.g., traps)

Point-of-Interest. Environment landmark intended to draw in players for exploration. (e.g., setpieces)

NPC. Non-hostile character that can be interacted with. (e.g., questgiver)



Runtime Interface

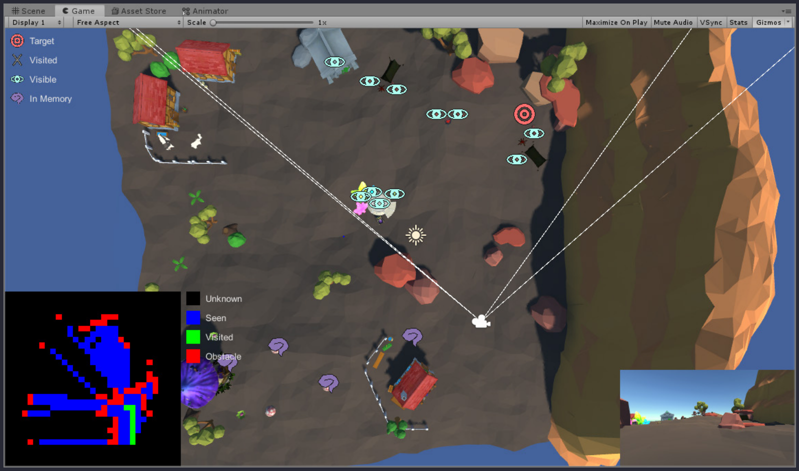
During playmode, select the agent in the hierarchy expand the PathOS Renderer component in the Inspector to enable the PathOS Agent UI. Unchecking the "3D Gizmos" option in Unity is recommended.

Controls

Spacebar. Toggle on-screen legend for mental map and Gizmos.

Click and drag. Pan the PathOS World Camera (if present).

Mouse wheel. Zoom in/zoom out with the PathOS World Camera (if present).



UI Layout

In the lower left corner, the agent's mental map is displayed. In the lower right, the agent (player) POV camera view is rendered. Gizmos are displayed on level entities indicating their state in the agent's world model.

Mental Map

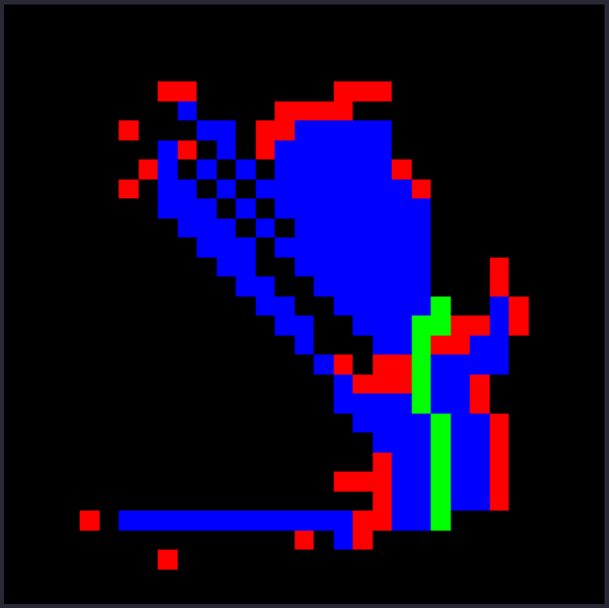
The mental map shows the agent's internal, tile-based representation of the scene's spatial layout. Four colours are used to indicate the state of each tile as perceived by the agent:

⬛ Black - Unknown

🟦 Blue - Seen as unobstructed (e.g., flat ground)

🟥 Red - Seen as obstructed (e.g., wall)

🟩 Green - Visited/traversed by the agent

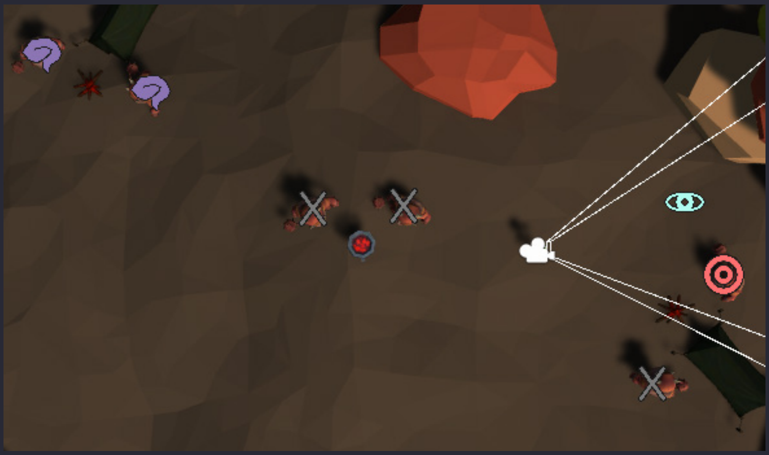


Player View

The player view displays what the agent is currently "seeing" through its player POV camera. This can be used to double-check visibility of game objects outside the approximate system used by agents.

Entity Gizmos

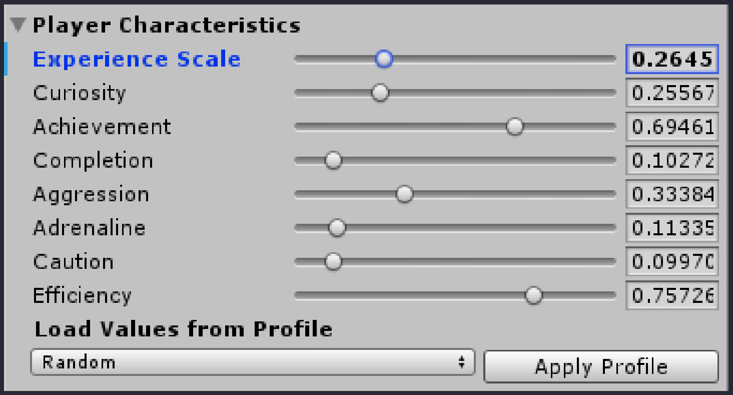
For all game entities tagged using the markup system, Gizmos are displayed indicating how they factor into the agent's logic at any given time. The meaning of these gizmos is as follows:



[NO ICON]	Entity is not affecting agent logic - it is not visible, remembered, or previously visited.
◎	Currently targeted by the agent. Can be applied to a game entity, or an empty point in the scene (while the agent is exploring).
🧠	Entity is contained in the agent's memory.
X	Entity has been previously visited.
👁	Entity is currently visible to the agent.
🚫	Entity has been determined to be unreachable (the agent cannot navigate to it using the Navmesh).

Agent Customization

Agents are governed by their motives, which reflect player motivations and can be customized for each agent. To change an agent's behaviour, you can use the **Player Characteristics** tab of the PathOS Agent Inspector. Here you can tweak individual motives, or apply a custom profile preset (see below).



Player Characteristics

Experience Scale	0.2645
Curiosity	0.25567
Achievement	0.69461
Completion	0.10272
Aggression	0.33384
Adrenaline	0.11335
Caution	0.09970
Efficiency	0.75726

Load Values from Profile

Random

Motives

There are seven agent motives in addition to the experience scale (e.g., amount of prior game experience). These motives affect the agent's behaviour and affect the way it will evaluate tagged entities as potential destinations. These motives are as follows:

Curiosity. The motivation to explore for exploration's sake, and discover all a level has to offer.

Achievement. Wanting to earn achievements, complete game objectives, and rack up a high score.

Completion. The desire to complete every in-game log, find every collectible, and so on.

Aggression. A drive to seek out conflict and combat, dominating the game world.

Adrenaline. Thrill-seeking, not only in combat, but in besting challenges or environmental gauntlets.

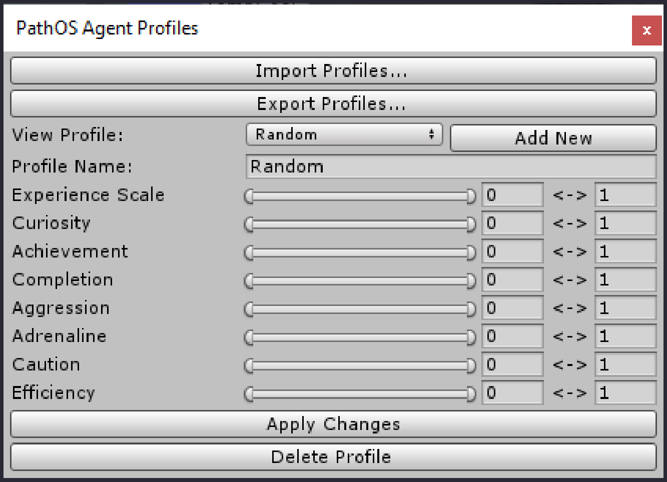
Caution. Taking care to maximize survivability, avoiding combat and hoarding resources.

Efficiency. Wanting to get through a level as quickly as possible, prioritizing necessary goals.

As a note for advanced users, you can view and edit the relationship between these motives and level markup tags in the **Motive Weights** tab of the PathOS Manager Inspector. Positive weights between a motive and entity tag indicate that an agent with a high value for that motive will be drawn to entities of that type. Conversely, negative weights will cause repulsion, and a zero weighting indicates no effect of the chosen motive on the agent's behaviour around entities of the chosen type.

Custom Profiles

To manage agent profiles, go to Window > PathOS Profiles. From here, you can create and edit profiles, as well as loading or saving files to carry profile sets between projects. Each profile has a range defined for the seven agent motives, as well as experience. When a profile is applied to an agent, values are picked randomly from these ranges.

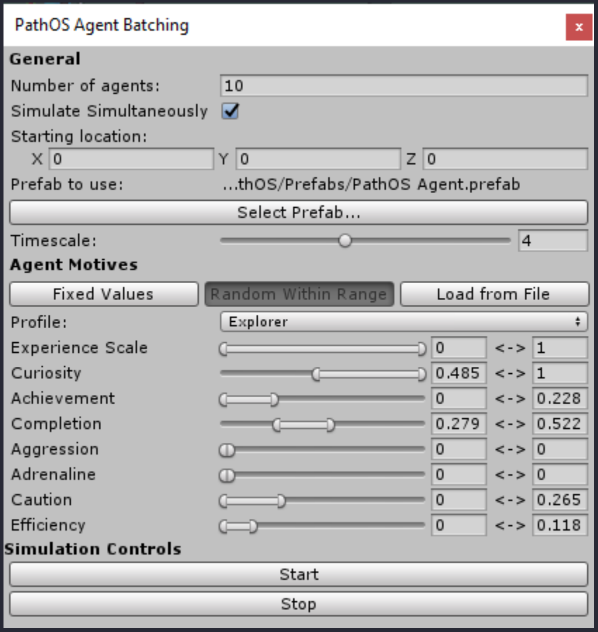


View Profile: Random

Profile Name: Random

Experience Scale	0	<->	1
Curiosity	0	<->	1
Achievement	0	<->	1
Completion	0	<->	1
Aggression	0	<->	1
Adrenaline	0	<->	1
Caution	0	<->	1
Efficiency	0	<->	1

Batch Simulation



Number of agents: 10

Simulate Simultaneously ☑

Starting location: X 0 Y 0 Z 0

Prefab to use: ...thOS/Prefabs/PathOS Agent.prefab

Timescale: 4

Profile: Explorer

Experience Scale	0	<->	1
Curiosity	0.485	<->	1
Achievement	0	<->	0.228
Completion	0.279	<->	0.522
Aggression	0	<->	0
Adrenaline	0	<->	0
Caution	0	<->	0.265
Efficiency	0	<->	0.118

To simulate multiple agents automatically, go to Window > PathOS Agent Batching. From here, you can choose whether to simulate agents consecutively or simultaneously. For consecutive simulation, be sure to drag in a reference to the agent in your scene. For simultaneous simulation, specify a prefab for the system to instantiate for each agent needed, as well as a starting position for agents in world space. For simultaneous simulation, the number of agents active at once is capped at 8. Any number greater than this will be divided into batches of 8 or less at a time, and automatically run in sequence.

You can also adjust the timescale of the simulation to speed things up - note that any time limit set in the Manager inspector will proceed in real time, however.

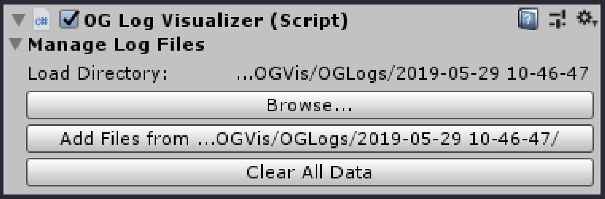
Agent motives will be initialized automatically based on the settings specified - either using fixed values, randomizing them within a range, or loading their values from a file. For range initialization, a custom profile (see above) can be selected to automatically define ranges. For file initialization, a .CSV file should be specified containing values for each of the desired agents (a sample file is included with the demo project).

Data Recording & Visualization

To automatically record logs containing information on agents' navigation and visiting level entities, toggle the "Enable Logging" setting on the OGLog Manager component. This will record logs both if playmode is triggered manually, or if simulation is handled en masse through the agent batching window.

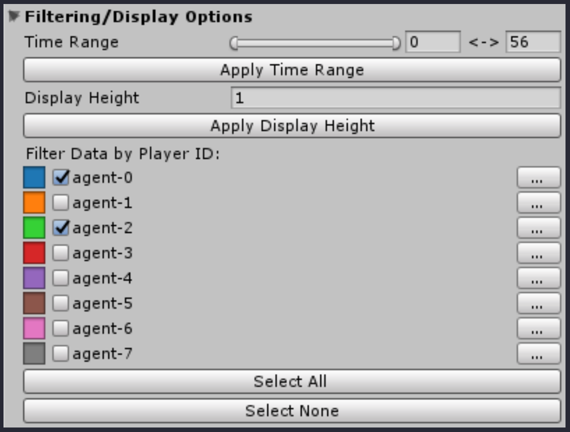
Loading Data Logs

Load logs through the OGLog Visualizer component in the Inspector. Logs are stored as CSV files. To load them, select the directory where logs are located and hit "Add Files from...".


☑ OG Log Visualizer (Script)

Manage Log Files

Load Directory: ...OGVis/OGLogs/2019-05-29 10-46-47


Filtering/Display Options

Time Range 0 <-> 56

Display Height 1

Filter Data by Player ID:

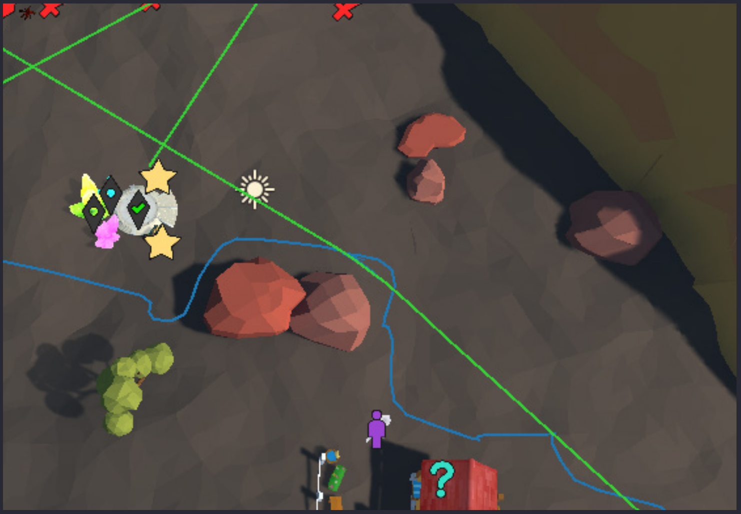
🟦 ☑ agent-0 ...
🟧 □ agent-1 ...
🟩 ☑ agent-2 ...
🟥 □ agent-3 ...
🟪 □ agent-4 ...
🟫 □ agent-5 ...
■ □ agent-6 ...
⬛ □ agent-7 ...

Display Filters

In the **Filtering/Display Options** tab, you can control what data is visible. The "time range" (in seconds) will exclude data from outside the specified range. The "display height" controls at what altitude (in units) visualization elements will be rendered in the scene. You can also choose which agents should be included or excluded from the visualization. The profiles of the agents used to create the data can be viewed by clicking the ellipsis next to each agent name.

Viewing Agent Paths

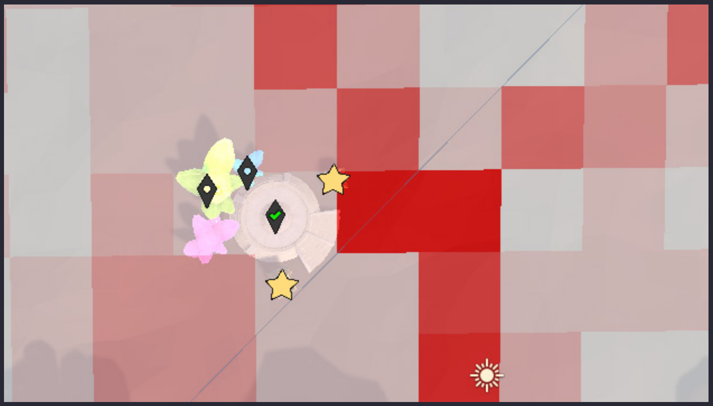
Individual agent paths through the world can be toggled from the **Individual Paths** tab. Enabling "individual interactions" will also show a record of individual agents visiting level entities along their journey through the level. From here, you can also specify colours for displaying each agent's trajectory.

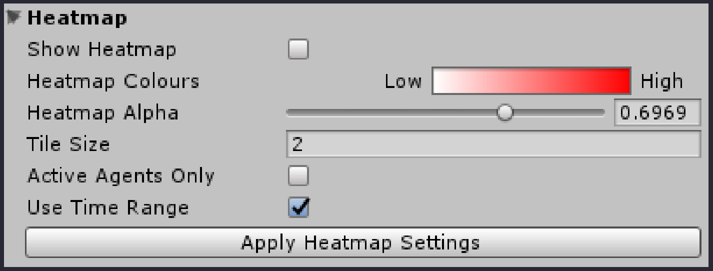


Heatmaps

To use the heatmap functionality, ensure that a child object of the OGLog Visualizer has the OGLog Heatmap Visualizer component attached (the provided prefab is set up with this configuration).

The **Heatmap** tab can be used for customization (e.g., colour scheme). The “tile size” attribute can be used to adjust the heatmap granularity. If “active agents only” is enabled, only data from agents enabled in the **Filtering** tab will be used. If “use time range” is enabled, only data from the time range specified in the **Filtering** tab will be used.





Heatmap

Show Heatmap ☐

Heatmap Colours Low High

Heatmap Alpha 0.6969

Tile Size 2

Active Agents Only ☐

Use Time Range ☑

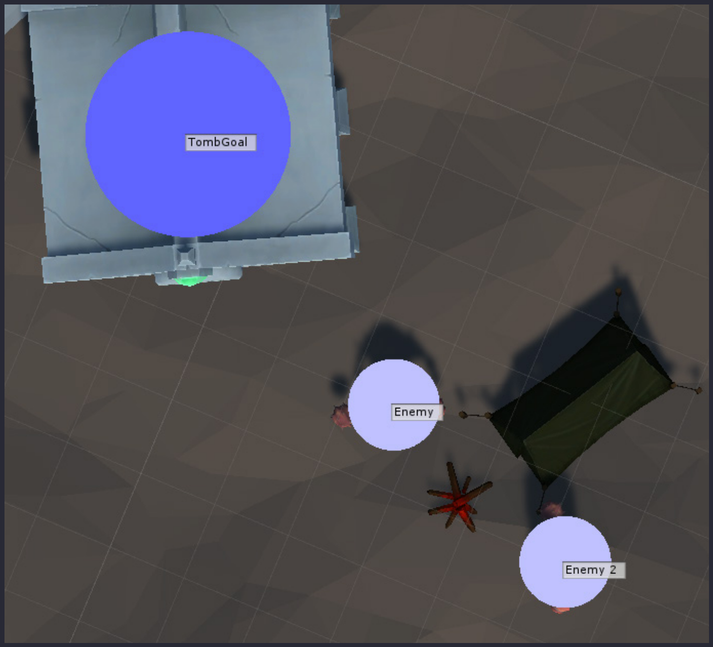
Apply Heatmap Settings

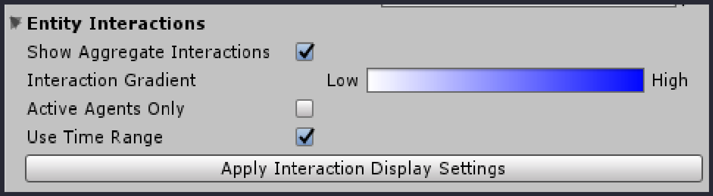
Viewing Agent Interactions

In the **Entity Interactions** tab, you can visualize how many agents visited different level entities in the scene. Each entity is shown as a circle, with scale and colour adjusted to reflect the proportion of agents which visited each entity.

If “active agents only” is enabled, only data from agents enabled in the **Filtering** tab will be used. Circles will be scaled according to the number of total agents in the enabled group. If disabled, data from all agent logs loaded will be used and scaled according to the total number of logs loaded.

If “use time range” is enabled, only data from the range in the **Filtering** tab will be used (i.e., entities visited outside this time range are excluded from the visualization).





Entity Interactions

Show Aggregate Interactions ☑

Interaction Gradient Low High

Active Agents Only ☐

Use Time Range ☑

Apply Interaction Display Settings

Complete System Reference

This section explains the function of each of the included script components, as well as the variables exposed in the Inspector. Most Inspector properties have tooltips in Unity for your reference, and the included prefabs (PathOS > prefabs) are designed to work out-of-the-box.

AI Agents & Configuration

PathOS Manager

There should be one PathOS Manager in the scene. It is responsible for storing level markup data (see *Level Markup*) and providing agents with a lookup table for motive scoring (see *Agent Customization*).

Limit Simulation Time. If enabled, once testing begins, playmode will exit automatically after a time limit.

Max Simulation Time. The time limit for the above (in real time).

Final Goal Triggers End. If enabled, once all agents have reached the object tagged as the final goal (if one exists), playmode will exit automatically.

Show Level Markup. If enabled, Gizmos indicating level markup will be displayed when the Manager is selected and the PathOS Manager component is expanded in the Inspector.

PathOS Agent

This component makes a GameObject function as a testing agent. When it is added, a Unity NavMeshAgent component will be added automatically, along with the PathOS Agent Memory, Eyes, and Renderer.

Freeze Agent. If enabled, the agent's logic and movement will be frozen (can be toggled during playmode).

Player Characteristics. Control the agent's motivation profile (see *Agent Customization*).

Explore Degrees. When exploring, the agent will cast out rays in its FOV every X degrees.

Explore Degrees (Back). When exploring, the agent will cast out rays outside of its FOV every X degrees.

Look Degrees. When looking around, the agent will rotate to either side by X degrees.

Visit Threshold. How close (in units) the agent needs to pass by a game entity to consider it visited. This value should be positive. If it is too large, the agent will consider entities visited when it is still very far away.

Explore Threshold. How close (in units) two exploration targets must be to be considered the same.

Explore Target Margin. When exploring, the radius the agent uses to find a target position on the Navmesh, in units (outside this radius, the agent will give up and deem the location unexplorable).

PathOS Agent Memory

Controls agent memory. Added automatically with the PathOS Agent component.

Grid Sample Size. How large (in units) each tile of the agent's mental map will be.

PathOS Agent Eyes

Controls agent perception. Added automatically with the PathOS Agent component.

Player Camera. The camera which should be used for the agent's "eyes", mimicking player view. The camera used should be a child of the agent GameObject and have its Camera component disabled.

Visibility Size Threshold. The minimum size (measured as a factor of player camera viewport width) that an entitiy must appear on-screen to be considered visible. This value is automatically adjusted to account for aspect ratio when examining entity height.

Raycast Distance. The distance (in units) that the agent "looks" across the Navmesh for obstacles. This value should be positive.

Raycast Height. The height (in units) at which rays will be casted across the Navmesh.

PathOS Agent Renderer

Controls the runtime agent UI. Added automatically with the PathOS Agent component. Renders only if the agent is selected and the Renderer component is expanded in the Inspector.

Show Legend. If enabled, a legend for the agent's mental map and displayed Gizmos will be shown.

Show Memory Map. If enabled, the agent's mental map will be rendered on screen.

Map Screen Size. The onscreen size of the mental map (in pixels).

Show Player View. If enabled, the view from the agent's player camera will be rendered on screen.

View Screen Size. The onscreen size of the player view (in pixels).

PathOS World Camera

This component can be attached to an overhead camera for ease of viewing agents during playmode.

Scroll Speed. How quickly the camera will dolly in and out when the mouse wheel is scrolled.

Pan Speed. How quickly the camera will pan around when the mouse is clicked and dragged.

Data Visualization

OGLog Manager

Handles recording of data logs for agent behaviour and navigation.

Enable Logging. If enabled, data logs will be recorded for all agents in the scene.

Log Directory. The directory to which logs should be written.

Log File Prefix. How agent files should be named (*"prefix-#.csv"*).

Sample Rate. How often (per second) agent position should be sampled for logging.

OGLog Visualizer

Handles visualization of agent logs. See the *Data Recording & Visualization* section for an explanation of Inspector settings for visualization. Needs an OGLog Heatmap component attached to a child object in order to render heatmaps.

OGLog Heatmap

Controls the rendering of heatmap visualizations.

B

USER STUDY HANDOUT

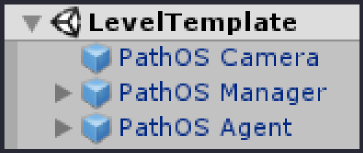
The following page contains a copy of the handout given to participants explaining the level design task for the user study described in Chapter 4.

PathOS Unity Demo - Level Design Exercise

Getting Started

For this exercise, use the provided Unity project. A template scene can be found in the User Study Project/Scenes folder - start each level by creating a copy of this template. Create 2 levels:

- One where players should explore and collect lots of items
- One where players need to fight their way through enemies to visit several goal markers



LevelTemplate
PathOS Camera
PathOS Manager
PathOS Agent

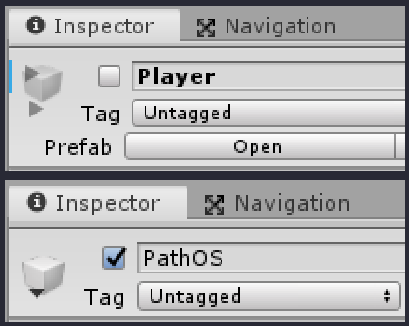
To use PathOS, simply ensure that you keep the AI Manager and Agent GameObjects from the provided template. Should these be deleted, you can recreate them from the PathOS/Prefabs folder.

Level Creation

Create your levels using assets from the Prefabs folder. To create each level, you'll need to do the following:

1. Place objects in the scene as desired.
2. Mark up your level using the PathOS Manager (see *Level Markup* in the user manual).
3. Re-bake the Unity Navmesh from the Navigation panel (Window > AI > Navigation).

Make sure that steps 2 & 3 are completed before you test with PathOS, and be sure to re-bake the navmesh after making changes to the level's layout.



Inspector Navigation
□ Player
Tag Untagged
Prefab Open

Inspector Navigation
☑ PathOS
Tag Untagged

Testing

To test your level layouts, you can use the PathOS framework; ensure that the Player GameObject is disabled and the PathOS container object is enabled. If you prefer, you can test your level by controlling a character yourself in addition to using the framework; to do this, enable the Player GameObject and disable the PathOS container object.

Note: To avoid Navmesh issues, avoid editing values on the NavMeshAgent component of the PathOS Agent.

C RESEARCHER INTERVIEW GUIDE

(Note: This appendix has been slightly modified from its original form to fit the typesetting style of the rest of this document.)

Questions in **bold** are numbered to indicate a rough order (may be adjusted within sections depending on the flow of discussion) and should be asked regardless of participants' previous answers, unless they are answered naturally during discussion that has already occurred during the interview.

Non-bolded questions are optional follow-up questions which can be asked at the discretion of the researcher to facilitate further discussion and clarify participant intent where appropriate.

As this interview follows a semi-structured protocol, the researcher may rephrase the questions stated herein and ask additional follow-ups not listed at their discretion to improve the insights which can be gained.

SESSION 1 - PRE-EXERCISE INTERVIEW

Purpose: Initial exploration of user process and workflow. Researcher should take notes on any points of interest for follow-up during the post-exercise interview.

1. Could you explain your role/area of expertise in game development?

2. Could you briefly explain your experience with level design?

3. Have you used Unity before? How do you feel about using it?

4. What is your typical process when designing a new level or map for a game? Say you're designing a dungeon for an action-adventure game, what's your workflow?

What tools do you typically use? Is there anything you would change about your workflow, if you could?

(After answering these questions, the participant will be introduced to the design exercise and the PathOS tool.)

SESSION 2 - POST-EXERCISE INTERVIEW

Purpose: Explore and understand users' design process during the exercise. Evaluate tool usability and feature set, understand how users apply the tool in their work, gather feedback and suggestions for improvement.

1. Could you briefly explain the levels you created?

2. What was your general process in creating each level?

What steps did you take to refine your design? How/when did you test your levels? Did you use the AI agents and/or test yourself?

3. Did you ever make changes to a level based on output from the tool?

What kind of changes did you make and why?

4. Did you feel like you could express your creativity with the tools provided? Why or why not?

5. If you completed this exercise again, is there anything you would do differently?

6. What were your first impressions of the tool?

7. What, if anything, did you like about the tool?

On mentioning a particular element, example follow-up questions:

Could you elaborate on why you liked that feature? Do you think that feature could be made even more useful? What do you think that feature would be most useful for?

8. What, if anything, did you dislike about the tool?

On mentioning a particular element, example follow-up questions:

Could you elaborate on why you disliked that feature? Do you have any suggestions as to how that feature could be improved?

9. What, if anything, would you change about the tool?

Were there any features you found yourself using often? Were there any features you didn't use very much?

Follow up on the reasons for using or not using individual features. Go over any notes that the participant kept during their time completing the exercise and review their use of different features.

10. How did you find learning to use the tool?

Did you notice the tooltips in the Unity Inspector? Did you find them helpful? Do you think this tool would benefit from a guided walkthrough? Do you think this tool would benefit from a tutorial video?

On mentioning a particular difficulty: Why do you think that feature was difficult to understand?

11. How did you find the interface?

How did you feel specifically about the in-game view - agent view, agent mental map, and entity gizmos?

How did you feel specifically about the Unity Inspector widgets for the agents and entity tagging system?

12. How did you feel about integrating the tool into your workflow?

Follow up on whether they felt like the tool was a natural extension of their process, and any pain points on making it part of their workflow.

13. Do you think this tool is useful? Why or why not?

If yes: What situations do you think the tool would be most useful in? How do you think the tool could be made more useful?

If no: Do you think this tool has the potential to be useful? Do you think this tool might be useful for a developer with a different role? Do you think the tool could be changed to be more useful? How?

14. Do you think you could use this tool or something like it in your work or personal projects?

If yes: Describe a situation in which you could see yourself using this tool.

D

LIST OF SUGGESTED CHANGES

This list collects all of the changes discussed with participants during the user study. Changes have been assigned a category based on their nature:

Usability changes would help prevent user error or confusion.

Accessibility changes would make the framework more accessible (e.g., in terms of UI visibility for those with colour vision deficiency, etc.).

QoL (quality-of-life) changes would improve user experience by expediting certain tasks or providing additional information in the UI.

Additions would be more substantial modifications allowing for additional functionality and/or an enhanced ability to suit individual game projects.

The number in brackets next to a change indicates the number of participants with which the potential change was discussed. (Several changes were brought up by multiple participants independently, and have thus been collapsed into one representative entry here.)

Level Markup

(Usability) (2) Lock things as untaggable (P8, P9)
(Usability) (2) Remember markup brush state during scene camera adjust (P5, P9)
(Usability) (1) Additional indicator on cursor to improve precision (P3)
(Accessibility) (1) Improve visibility of panel/keep it open persistently (P7)
(QoL) (4) Tag groups of objects from selection (P1, P5, P6, P9)
(QoL) (2) Tag objects by prefab type (P3, P9)
(QoL) (2) Quick search for entity type in hierarchy or entity list (P4, P8)
(QoL) (1) Clarify that objects can only have one type (P4)
(QoL) (1) Click-and-drag mode for markup brush (P5)
(QoL) (1) Make markup gizmos always visible (P9)
(QoL) (1) Visual representation of visit radius (P10)
(Add) (1) Allow objects to have more than one entity type (P9)
(Add) (1) Custom entity types (P6)

Agents

Personality Adjustment:

(QoL) (2) Tooltips for motives (P8, P10)

Behaviour/Logic:

(QoL) (2) Fast forward for individual agent simulation (P1, P8)
(QoL) (1) Force stop option for agents if technical glitches are encountered (P9)
(Add) (4) Dynamic entities (e.g., spawners, conditional logic) (P4, P3, P9, P10)
(Add) (3) Adding resource (risk/reward) logic (P1, P3, P9)
(Add) (2) API for custom agent behaviours and/or navigation logic (P4, P6)

Runtime UI:

Gizmos:

(Usability) (2) Warning to enable gizmos/2D vs. 3D/auto-enable (P3, P10)
(QoL) (1) Gizmo on agent (P10)

Mental Map:

(Usability) (1) Preview of where mental map is drawn if agent is not selected (P7)
(Accessibility) (1) Colourblind-friendly colour scheme (P7)
(Add) (1) Move to own view and render texture each frame (P6)

Player View:

(Usability) (1) Make display customization options more obvious (P3)

Other:

(Usability) (2) Move runtime UI to another window (access from menu) (P1, P6)
(QoL) (1) Make runtime UI elements available in scene view (P8)
(QoL) (1) Display summary of manager settings on runtime UI (P10)
(Add) (2) Give the agent a thought bubble/planning logic display (P6, P10)
(Add) (1) Ability to peek inside the agent's memory (P4)
(Add) (1) Central control panel for runtime view with multiple agents (P8)

Data Logging & Visualization

Logging:

(Usability) (2) Prompt to enable logging/enable by default (P5, P10)
(Usability) (1) Secondary dialog to load logs after selecting directory (P9)

(QoL) (2) Default output destination (P1, P5)
(QoL) (1) Crawl subdirectories automatically when loading logs (P1)
(QoL) (1) Ability to unload individual logs (P1)
(QoL) (1) Rename folder created by logger/include name of scene in default (P9)

Heatmap:

(Add) (1) Add information on game performance (e.g., GPU load) (P4)
(Add) (1) Add ability to export heatmaps as images (P6)

Individual Path Vis:

(QoL) (2) Add arrowheads along the path to indicate direction of travel (P5, P10)
(Add) (1) Add time data to path visualization (colour/timestamps) (P9)
(Add) (1) Aggregate path visualization option (P10)

Entity Vis:

(QoL) (1) Increase scaling for improved viewing at a distance (P10)

General/Other Vis:

(QoL) (2) Filtering by entity personality (P4, P5)
(QoL) (1) Fix alphabetization of agents to use expected numeric order (P9)
(QoL) (1) Repeat default colours after running out (instead of using white) (P9)
(Add) (1) Live replay of agent behaviour (P10)
(Add) (1) Summarization of agent interactions (P10)

Batching:

(QoL) (1) Allow batching tool to dock with Inspector panels (P9)
(QoL) (1) Instantiate agents under parent for easier hierarchy navigation (P10)
(Add) (1) Ability to batch with parallel instances of the game running (P4)

General/Other:

(Usability) (1) Button for rebaking navmesh in PathOS/"smart" auto-rebake (P8)
(Accessibility) (1) Icons for Inspector subpanels to improve visibility (P7)
(QoL) (1) More tooltips (e.g., on vis options) (P6)
(QoL) (1) Add message when simulation is finished explaining exit reason (P10)
(Add) (4) Centralized PathOS UI with its own Inspector tab (P3, P6, P7, P8)
(Add) (1) Redirect Unity help button in GUI to PathOS documentation (P6)

BIBLIOGRAPHY

[1] Samantha Stahlke, Atiya Nova, and Pejman Mirza-Babaei. "Artificial Playfulness: A Tool for Automated Agent-Based Playtesting". In: *Proceedings of CHI 2019 Extended Abstracts*. Glasgow, Scotland, UK: ACM, 2019, LBW0176:1–LBW0176:6. DOI: 10.1145/3290607.3313039.

[2] Samantha Stahlke and Pejman Mirza-Babaei. "Usertesting Without the User: Opportunities and Challenges of an AI-Driven Approach in Games User Research". In: *ACM Comput. Entertain* 16.2 (2018), 18 pp. DOI: 10.1145/3183568.

[3] Samantha Stahlke and Pejman Mirza-Babaei. "Usertesting Without the User: Introducing PathOS, a Framework for AI-Based Games User Research". In: *Proceedings of the 2017 CHI PLAY Workshop on Exploiting Players*. Amsterdam, Netherlands, 2017, 5 pp.

[4] Randy J Pagulayan et al. "User-centered design in games". In: *The Human-Computer Interaction Handbook*. Ed. by Julie A. Jacko and Andrew Sears. L. Erlbaum Associates Inc., 2002, pp. 883–906. ISBN: 0-8058-3838-4.

[5] Anders Drachen, Pejman Mirza-Babaei, and Lennart E Nacke, eds. *Games User Research*. 1st ed. Oxford, United Kingdom: Oxford University Press, 2018. ISBN: 978-0-19-879484-4.

[6] Regina Bernhaupt, ed. *Game User Experience Evaluation*. Human–Computer Interaction Series. Springer International Publishing, 2015. ISBN: 978-3-319-15984-3.

[7] Rafet Sifa et al. "Behavior Evolution in Tomb Raider Underworld". In: *Proceedings of CIG 2013*. 2013, 8 pp. DOI: 10.1109/CIG.2013.6633637.

[8] Brandon Drenikow and Pejman Mirza-Babaei. "Vixen: Interactive Visualization of Gameplay Experiences". In: *Proceedings of FDG 2017*. FDG '17. ACM, 2017, 3:1–3:10. DOI: 10.1145/3102071.3102089.

[9] Joshua Tanenbaum et al. "Costumes and Wearables as Game Controllers". In: *Proceedings of TEI 2015*. TEI '15. event-place: Stanford, California, USA. New York, NY, USA: ACM, 2015, pp. 477–480. DOI: 10.1145/2677199.2683584.

[10] Frederico da Rocha Tomé Filho et al. "Let's Play Together: Adaptation Guidelines of Board Games for Players with Visual Impairment". In: *Proceedings of CHI 2019*. CHI '19. event-place: Glasgow, Scotland Uk. New York, NY, USA: ACM, 2019, 631:1–631:15. DOI: 10.1145/3290605.3300861.

[11] Jong-Eun Roselyn Lee and Sung Gwan Park. ""Whose Second Life Is This?" How Avatar-Based Racial Cues Shape Ethno-Racial Minorities' Perception of Virtual Worlds". In: *Cyberpsychology, Behavior, and Social Networking* 14.11 (2011), pp. 637–642. DOI: 10.1089/cyber.2010.0501.

[12] Anders Drachen. "Behavioral Telemetry in Games User Research". In: *Game User Experience Evaluation*. Ed. by Regina Bernhaupt. Human–Computer Interaction Series. Springer International Publishing, 2015, pp. 135–165. ISBN: 978-3-319-15984-3.

[13] Anders Drachen and Shawn Connor. "Game Analytics for Games User Research". In: *Games User Research*. Ed. by Anders Drachen, Pejman Mirza-Babaei, and Lennart E Nacke. 1st ed. Oxford, United Kingdom: Oxford University Press, 2018, pp. 333–353. ISBN: 978-0-19-879484-4.

[14] Lennart E. Nacke. "Games User Research and Physiological Game Evaluation". In: *Game User Experience Evaluation*. Ed. by Regina Bernhaupt. Human–Computer Interaction Series. Springer International Publishing, 2015, pp. 63–86. ISBN: 978-3-319-15984-3.

[15] Steve Bromley. "Interviewing Players". In: *Games User Research*. 1st ed. Oxford, United Kingdom: Oxford University Press, 2018, pp. 163–188. ISBN: 978-0-19-879484-4.

[16] David Tisserand. "It is all about process". In: *Games User Research*. 1st ed. Oxford, United Kingdom: Oxford University Press, 2018, pp. 163–188. ISBN: 978-0-19-879484-4.

[17] Pejman Mirza-Babaei, Naeem Moosajee, and Brandon Drenikow. "Playtesting for Indie Studios". In: *Proceedings of Mindtrek 2016*. Tampere, Finland: ACM, 2016, pp. 366–374. ISBN: 978-1-4503-4367-1. DOI: 10.1145/2994310.2994364.

[18] Murray Campbell, a. Joseph Hoane Jr., and Feng-hsiung Hsu. "Deep Blue". In: *Artificial Intelligence* 134.1-2 (2002), pp. 57–83. DOI: 10.1016/S0004-3702(01)00129-1.

[19] David Silver et al. "Mastering the Game of Go with Deep Neural Networks and Tree Search". In: *Nature* 529.7587 (2016), pp. 484–489. DOI: 10.1038/nature16961.

[20] Greg Brockman et al. *OpenAI Five*. June 2018. URL: https://openai.com/blog/openai-five/.

[21] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533. DOI: 10.1038/nature14236.

[22] Johannes Pfau, Jan David Smeddinck, and Rainer Malaka. "Automated Game Testing with ICARUS: Intelligent Completion of Adventure Riddles via Unsupervised Solving". In: *Proceedings of CHI PLAY 2017 Extended Abstracts*. 2017, pp. 153–164. DOI: 10.1145/3130859.3131439.

[23] Tiago Machado et al. "AI-assisted game debugging with Cicero". In: *Proceedings of CEC 2018*. IEEE, 2018, 8 pp. DOI: 10.1109/CEC.2018.8477829.

[24] Anders Drachen, Alessandro Canossa, and Georgios N. Yannakakis. "Player modeling using Self-Organization in Tomb Raider: Underworld". In: *Proceedings of CIG 2009*. 2009, pp. 1–8. DOI: 10.1109/CIG.2009.5286500.

[25] Shaghayegh Roohi et al. "Neural Network Based Facial Expression Analysis of GameEvents: A Cautionary Tale". In: *Proceedings of CHI PLAY 2018*. CHI PLAY '18. Melbourne, VIC, Australia: ACM, 2018, pp. 429–437. DOI: 10.1145/3242671.3242701.

[26] Finnegan Southey et al. "Semi-Automated Gameplay Analysis by Machine Learning". In: *Proceedings of AIIDE 2005*. 2005, pp. 123–128.

[27] Adam M. Smith, Eric Butler, and Zoran Popović. "Quantifying over Play: Constraining Undesirable Solutions in Puzzle Design". In: *Proceedings of FDG 2013*. 2013, pp. 221–228. URL: http://www.fdg2013.org/program/papers/paper29_smith_etal.pdf.

[28] Connor Bell et al. "Automated Playtesting with Recycled Cardstock". In: *Game & Puzzle Design* 2.1 (2016), pp. 71–83.

[29] Mohammad Shaker, Noor Shaker, and Julian Togelius. "Evolving Playable Content for Cut the Rope through a Simulation-Based Approach". In: *Proceedings of AIIDE 2013*. 2013, pp. 72–78.

[30] Christoffer Holmgård et al. "Automated Playtesting with Procedural Personas through MCTS with Evolved Heuristics". In: *IEEE Transactions on Games* 11.4 (2018), pp. 352–362. DOI: 10.1109/TG.2018.2808198.

[31] Rafael Veras and Christopher Collins. "Saliency Deficit and Motion Outlier Detection in Animated Scatterplots". In: *Proceedings of CHI 2019*. CHI '19. Glasgow, Scotland, UK: ACM, 2019, 541:1–541:12. DOI: 10.1145/3290605.3300771.

[32] Alex Graves and Navdeep Jaitly. "Towards End-to-End Speech Recognitionwith Recurrent Neural Networks". en. In: *Proceedings of ICML 2014*. Vol. 32. Beijing, China: ACM, 2014, pp. II–1764–II–1772.

[33] C. J. Hutto and Eric Gilbert. "VADER: A Parsimonious Rule-Based Model for Sentiment Analysis of Social Media Text". In: *Proceedings of the 2014 AAAI Conference on Weblogs and Social Media*. 2014. (Visited on 07/12/2019).

[34] Gunhee Kim, Leonid Sigal, and Eric P. Xing. "Joint Summarization of Large-scale Collections of Web Images and Videos for Storyline Reconstruction". In: *Proceedings of CVPR 2014*. IEEE, 2014, pp. 4225–4232. DOI: 10.1109/CVPR.2014.538.

[35] Tobias Mahlmann et al. "Predicting Player Behavior in Tomb Raider: Underworld". In: *Proceedings of CIG 2010*. IEEE, 2010, pp. 178–185. DOI: 10.1109/ITW.2010.5593355.

[36] Wookhee Min et al. “Player goal recognition in open-world digital games with long short-term memory networks”. In: *Proceedings of IJCAI 2016*. 2016, pp. 2590–2596. DOI: 10.5555/3060832.3060983.

[37] Ondřej Pluskal and Jan Šedivý. “Predicting Players Behavior in Games with Microtransactions”. In: *Proceedings of the 7th European Starting AI Researcher Symposium*. Amsterdam, Netherlands: IOS Press, 2014, pp. 230–239. DOI: 10.3233/978-1-61499-421-3-230.

[38] Ruck Thawonmas and Masayoshi Kurashige. “Detection of landmarks for clustering of online-game players”. In: *The International Journal of Virtual Reality* 6.3 (2007), pp. 11–16. URL: http://140.109.19.187/pub/thawonmas07_landmark_ijvr.pdf.

[39] Jonathan Tremblay et al. “An Exploration Tool for Predicting Stealthy Behaviour”. In: *Proceedings of AIIDE 2013*. 2013, pp. 34–40.

[40] Richard Bartle. “Hearts, Clubs, Diamonds, Spades: Players who suit MUDs”. In: *Journal of Virtual Environments* 1.1 (1996), 19 pp. URL: http://mud.co.uk/richard/hcds.htm.

[41] Lennart E Nacke, Chris Bateman, and Regan L Mandryk. “BrainHex: A Neurobiological Gamer Typology Survey”. In: *Entertainment Computing* 5.1 (2014), pp. 55–62. DOI: 10.1016/j.entcom.2013.06.002.

[42] Regan L. Mandryk, Kori M. Inkpen, and Thomas W. Calvert. “Using psychophysiological techniques to measure user experience with entertainment technologies”. In: *Behaviour and Information Technology* 25.2 (2006), pp. 141–158. DOI: 10.1080/01449290500331156.

[43] Regan L. Mandryk and M. Stella Atkins. “A fuzzy physiological approach for continuously modeling emotion during interaction with play technologies”. In: *International Journal of Human Computer Studies* 65.4 (2007), pp. 329–347. DOI: 10.1016/j.ijhcs.2006.11.011.

[44] Spiros V. Ioannou et al. “Emotion recognition through facial expression analysis based on a neurofuzzy network”. In: *Neural Networks* 18.4 (2005), pp. 423–435. DOI: 10.1016/j.neunet.2005.03.004.

[45] M. R. Mohammadi, E. Fatemizadeh, and M. H. Mahoor. “PCA-based dictionary building for accurate facial expression recognition via sparse representation”. In: *Journal of Visual Communication and Image Representation* 25.5 (2014), pp. 1082–1092. DOI: 10.1016/j.jvcir.2014.03.006.

[46] Yubo Wang et al. “Real Time Facial Expression Recognition with Adaboost”. In: *Proceedings of ICPR 2004*. 2004, pp. 926–929. DOI: 10.1109/ICPR.2004.1334680.

[47] K. Sreenivasa Rao et al. “Emotion Recognition From Speech Signals”. In: *International Journal of Computer Science and Information Technologies* 3.2 (2012), pp. 3603–3607. ISSN: 0975-9646. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.437.2775&rep=rep1&type=pdf.

[48] Thurid Vogt, Elisabeth André, and Johannes Wagner. "Automatic Recognition of Emotions from Speech: A Review of the Literature and Recommendations for Practical Realisation". In: *Affect and Emotion in Human-Computer Interaction*. 2008, pp. 75–91. DOI: 10.1007/978-3-540-85099-1_7.

[49] Samira Ebrahimi Kahou et al. "Combining modality specific deep neural networks for emotion recognition in video". In: *Proceedings of ICMI 2013*. 2013, pp. 543–550. DOI: 10.1145/2522848.2531745.

[50] Fatemeh Hemmatian and Mohammad Karim Sohrabi. "A survey on classification techniques for opinion mining and sentiment analysis". In: *Artificial Intelligence Review* 52 (2017), pp. 1496–1545. DOI: 10.1007/s10462-017-9599-6.

[51] David Pinelle, Nelson Wong, and Tadeusz Stach. "Using genres to customize usability evaluations of video games". In: *Proceedings of Future Play 2008*. 2008, pp. 129–136. DOI: 10.1145/1496984.1497006.

[52] Sinno Jialin Pan et al. "Cross-domain sentiment classification via spectral feature alignment". In: *Proceedings of WWW 2010*. 2010, pp. 751–760. DOI: 10.1145/1772690.1772767.

[53] G. Wallner and S. Kriglstein. "Visualization-based analysis of gameplay data - A review of literature". In: *Entertainment Computing* 4.3 (2013), pp. 143–155. DOI: 10.1016/j.entcom.2013.02.002.

[54] Shixia Liu et al. "A survey on information visualization: recent advances and challenges". In: *The Visual Computer: International Journal of Computer Graphics* 30.12 (2014), pp. 1373–1393. DOI: 10.1007/s00371-013-0892-3.

[55] Yun-Gyung Cheong et al. "Automatically Generating Summary Visualizations from Game Logs". In: *Proceedings of AIIDE 2008*. Stanford, California, USA: AAAI Press, 2008, pp. 167–172. DOI: 10.5555/3022539.3022570.

[56] Pejman Mirza-Babaei et al. "How Does It Play Better? Exploring User Testing and Biometric Storyboards in Games User Research". In: *Proceedings of CHI 2013*. Paris, France: ACM, 2013, pp. 1499–1508. DOI: 10.1145/2470654.2466200.

[57] Rafael Veras and Christopher Collins. "Optimizing Hierarchical Visualizations with the Minimum Description Length Principle". In: *IEEE Transactions on Visualization and Computer Graphics* 23.1 (2017), pp. 631–640. DOI: 10.1109/TVCG.2016.2598591.

[58] David Gotz and Zhen Wen. "Behavior-Driven Visualization Recommendation". In: *Proceedings of IUI 2009*. 2009, pp. 315–324. DOI: 10.1145/1502650.1502695.

[59] Eli T. Brown et al. "Finding Waldo: Learning about Users from their Interactions". In: *IEEE Transactions on Visualization and Computer Graphics* 20.12 (2014), pp. 1663–1672. DOI: 10.1109/TVCG.2014.2346575.

[60] Duen Horng "Polo" Chau et al. “Apolo : Making Sense of Large Network Data by Combining Rich User Interaction and Machine Learning”. In: *Proceedings of CHI 2011*. 2011, pp. 167–176. DOI: 10.1145/1978942.1978967.

[61] Chris Bateman, Rebecca Lowenhaupt, and Lennart E Nacke. “Player Typology in Theory and Practice”. In: *Proceedings of DiGRA 2011*. Utrecht, The Netherlands, 2011, 24 pp. URL: http://www.digra.org/wp-content/uploads/digital-library/11307.50587.pdf.

[62] Benjamin Cowley and Darryl Charles. “Behavlets: a method for practical player modelling using psychology-based player traits and domain specific features”. In: *User Modelling and User-Adapted Interaction* 26.2-3 (2016), pp. 257–306. DOI: 10.1007/s11257-016-9170-1.

[63] Georgios N. Yannakakis et al. “Player Modeling”. In: *Dagstuhl Seminar on Artificial and Computational Intelligence in Games*. Vol. 6. 2013, pp. 45–59. DOI: 10.4230/DFU.Vol6.12191.45.

[64] Adam M. Smith et al. “An Inclusive View of Player Modeling”. In: *Proceedings of FDG 2011*. 2011, pp. 301–303. DOI: 10.1145/2159365.2159419.

[65] Olana Missura and G Thomas. “Player Modeling for Intelligent Difficulty Adjustment”. In: *Proceedings of DS 2009*. 2009, pp. 197–211. DOI: 10.1007/978-3-642-04747-3_17.

[66] Chang Yun et al. “PADS: Enhancing Gaming Experience Using Profile-Based Adaptive Difficulty System”. In: *Proceedings of the 2010 SIGGRAPH Symposium on Video Games*. 2010, pp. 31–36. DOI: 10.1145/1836135.1836140.

[67] David Melhart et al. “Your Gameplay Says It All: Modelling Motivation in Tom Clancy’s The Division”. en. In: *Proceedings of CoG 2019*. IEEE, 2019, 8 pp. DOI: 10.1109/CIG.2019.8848123.

[68] Anders Drachen, Christian Thurau, and Christian Bauckhage. “A Comparison of Methods for Player Clustering via Behavioral Telemetry”. In: *Proceedings of FDG 2013*. arXiv:1407.3950. 2013, pp. 245–252.

[69] Christian Bauckhage et al. “Beyond heatmaps: Spatio-temporal clustering using behavior-based partitioning of game levels”. In: *Proceedings of CIG 2014*. 2014, 8 pp. DOI: 10.1109/CIG.2014.6932865.

[70] Zhengxing Chen et al. “Modeling Individual Differences through Frequent Pattern Mining on Role-Playing Game Actions”. In: *2015 AIIDE Player Modeling Workshop*. 2015, pp. 2–7.

[71] Daniel Ramirez-Cano, Simon Colton, and Robin Baumgarten. “Player Classification Using a Meta-Clustering Approach”. In: *Proceedings of the 2010 International Conference on Computer Games, Multimedia & Allied Technology* (2010), pp. 297–304. DOI: 10.5176/978-981-08-5480-5_071.

[72] Eytan Adar et al. “Why we search: visualizing and predicting user behavior”. In: *Proceedings of WWW 2007*. 2007, pp. 161–170. DOI: 10.1145/1242572.1242595.

[73] Armando Vieira. “Predicting online user behaviour using deep learning algorithms”. In: *ArXiv Pre-Print* (2016). arXiv:1511.06247, 21 pp.

[74] Ho Chul Cho, Kyung Joong Kim, and Sung Bae Cho. “Replay-based strategy prediction and build order adaptation for StarCraft AI bots”. In: *Proceedings of CIG 2013*. IEEE, 2013. DOI: 10.1109/CIG.2013.6633666.

[75] Ji Lung Hsieh and Chuen Tsai Sun. “Building a player strategy model by analyzing replays of real-time strategy games”. In: *Proceedings of IJCNN 2008*. IEEE, 2008, pp. 3106–3111. DOI: 10.1109/IJCNN.2008.4634237.

[76] Ben G. Weber and Michael Mateas. “A data mining approach to strategy prediction”. In: *Proceedings of CIG 2009*. 2009, pp. 140–147. DOI: 10.1109/CIG.2009.5286483.

[77] Kevin Gold. “Training Goal Recognition Online from Low-Level Inputs in an Action-Adventure Game”. In: *Proceedings of AIIDE 2010*. 2010, pp. 21–26.

[78] Jeff Tian. “Defect Prevention and Process Improvement”. In: *Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement*. 2005, pp. 27–39. DOI: 10.1002/0471722324.ch13.

[79] Nadia Alshahwan and Mark Harman. “Automated Web Application Testing Using Search Based Software Engineering”. In: *Proceedings of ASE 2011*. 2011, pp. 3–12. DOI: 10.1109/ASE.2011.6100082.

[80] M. Linares-Vásquez, K. Moran, and D. Poshyvanyk. “Continuous, Evolutionary and Large-Scale: A New Perspective for Automated Mobile App Testing”. In: *Proceedings of ICSME 2017*. 2017, pp. 399–410. DOI: 10.1109/ICSME.2017.27.

[81] Simon Varvaressos et al. “Automated Bug Finding in Video Games”. In: *Computers in Entertainment* 15.1 (2017), pp. 1–28. DOI: 10.1145/2700529.

[82] Adam Smith. “Open Problem: Reusable Gameplay Trace Samplers”. In: *Proceedings of AIIDE 2013*. 2013, pp. 22–27. URL: http://www.aaai.org/ocs/index.php/AIIDE/AIIDE13/paper/viewFile/7471/7638.

[83] Sinan Ariyurek, Aysu Betin-Can, and Elif Surer. “Automated Video Game Testing Using Synthetic and Human-Like Agents”. In: *IEEE Transactions on Games* (2019), 21 pp. DOI: 10.1109/TG.2019.2947597.

[84] Tiago Machado et al. “Kwiri - What, When, Where and Who: Everything you ever wanted to know about your game but didn’t know how to ask”. en. In: *2nd Workshop on Knowledge Extraction from Games (co-located with AAAI 2019)*. 2019, 8 pp. URL: http://ceur-ws.org/Vol-2313/KEG_2019_paper_7.pdf.

[85] M J Nelson. “Game Metrics Without Players: Strategies for Understanding Game Artifacts”. In: *2011 AIIDE Workshop on Artificial Intelligence in the Game Design Process*. 2011, pp. 14–18. URL: http://www.scopus.com/inward/record.url?eid=2-s2.0-84862684087&partnerID=40&md5=e71a4c9276adc25e668359a9beab5b18.

[86] Alexander Zook, Eric Fruchter, and Mark O Riedl. "Automatic Playtesting for Game Parameter Tuning via Active Learning". In: *Proceedings of FDG 2014*. 2014, 8 pp. URL: https://www.cc.gatech.edu/~riedl/pubs/zook-fdg14.pdf.

[87] Oleksandra Keehl and Adam M. Smith. "Monster Carlo: An MCTS-based Framework for Machine Playtesting Unity Games". en. In: *Proceedings of CIG 2018*. Maastricht: IEEE, 2018, 8 pp. DOI: 10.1109/CIG.2018.8490363.

[88] Oleksandra Keehl and Adam M. Smith. "Monster Carlo 2: Integrating Learning and Tree Search for Machine Playtesting". en. In: *Proceedings of CoG 2019*. London, United Kingdom: IEEE, 2019, 8 pp. DOI: 10.1109/CIG.2019.8847989.

[89] Edward J. Powley et al. "Semi-automated level design via auto-playtesting for handheld casual game creation". In: *Proceedings of CIG 2016*. 2016, 8 pp. DOI: 10.1109/CIG.2016.7860438.

[90] Fernando de Mesentier Silva et al. "Exploring Gameplay with AI Agents". en. In: *Proceedings of AIIDE 2018*. arXiv:1811.06962. AAAI, 2018, 7 pp.

[91] Eric Butler et al. "Automatic Game Progression Design through Analysis of Solution Features". In: *Proceedings of CHI 2015*. 2015, pp. 2407–2416. DOI: 10.1145/2702123.2702330.

[92] Gillian Smith, Jim Whitehead, and Michael Mateas. "Tanagra: Reactive planning and constraint solving for mixed-initiative level design". In: *IEEE Transactions on Computational Intelligence and AI in Games* 3.3 (2011), pp. 201–215. DOI: 10.1109/TCIAIG.2011.2159716.

[93] Andrew Hoyt et al. "Integrating Automated Play in Level Co-Creation". In: *2019 AIIDE Workshop on Experimental AI in Games*. arXiv: 1911.09219. 2019.

[94] Julian Togelius, Sergey Karakovskiy, and Robin Baumgarten. "The 2009 Mario AI Competition". In: *Proceedings of CEC 2010*. 2010, 8 pp. DOI: 10.1109/CEC.2010.5586133.

[95] Shaghayegh Roohi et al. "Review of Intrinsic Motivation in Simulation-based Game Testing". In: *Proceedings of CHI 2018*. 2018, 347:1–347:13. DOI: 10.1145/3173574.3173921.

[96] Igor Borovikov et al. "Winning Isn't Everything: Training Agents to Playtest Modern Games". In: *2019 AAAI Workshop on Reinforcement Learning in Games*. 2019, p. 9.

[97] Igor Borovikov and Ahmad Beirami. "Imitation Learning via Bootstrapped Demonstrations in an Open-World Video Game". In: *2018 NeurIPS Workshop on Reinforcement Learning under Partial Observability*. 2018, 3 pp.

[98] Inge Becht and Sander Bakkes. "meIRL-BC : Predicting Player Positions in Video Games". In: *Proceedings of FDG 2014*. 2014, pp. 1–8.

[99] Emmett Tomai, Rosendo Salazar, and Roberto Flores. "Mimicking Humanlike Movement in Open World Games with Path-Relative Recursive Splines". In: *Proceedings of AIIDE 2013*. 2013, pp. 93–99.

[100] Jonathan Ho and Stefano Ermon. "Generative Adversarial Imitation Learning". In: *Proceedings of NIPS 2016*. 2016, 9 pp. DOI: 10.1016/j.compeleceng.2013.11.024.

[101] Rousslan Fernand Julien Dossa et al. "A Human-Like Agent Based on a Hybrid of Reinforcement and Imitation Learning". In: *Proceedings of IJCNN 2019*. Budapest, Hungary: IEEE, 2019, pp. 1–8. DOI: 10.1109/IJCNN.2019.8852026.

[102] Jordan Golgon. "In Forza Horizon 2, Computers Finally Drive as Crazy as Humans". In: *Wired* (Sept. 2014). URL: https://www.wired.com/2014/09/forza-horizon-2-drivatars/.

[103] Hendrik Baier et al. "Emulating Human Play in a Leading Mobile Card Game". In: *IEEE Transactions on Games* 11.4 (2018), pp. 386–395. DOI: 10.1109/TG.2018.2835764.

[104] Fernando de Mesentier Silva et al. "AI-based playtesting of contemporary board games". In: *Proceedings of FDG 2017*. 2017, 10 pp. DOI: 10.1145/3102071.3102105.

[105] Juan Ortega et al. "Imitating human playing styles in Super Mario Bros". In: *Entertainment Computing* 4.2 (2013), pp. 93–104. DOI: 10.1016/j.entcom.2012.10.001.

[106] Antonios Liapis et al. "Procedural personas as critics for dungeon generation". In: *Lecture Notes in Computer Science*. Vol. 9028. 2015, pp. 331–343. DOI: 10.1007/978-3-319-16549-3_27.

[107] Luvneesh Mugrai et al. "Automated Playtesting of Matching Tile Games". In: *Proceedings of CoG 2019*. arXiv: 1907.06570. London, United Kingdom: IEEE, 2019, 7 pp. DOI: 10.1109/CIG.2019.8848057.

[108] G D. Everett and Raymond Jr. McLeod. "Performance Testing". In: *Software Testing: Testing Across the Entire Software Development Life Cycle*. 2007, pp. 129–149. ISBN: 978-0-471-79371-7.

[109] Ethem Alpaydin. "Design and Analysis of Machine Learning Experiments". In: *Introduction to Machine Learning*. 3rd ed. The MIT Press, 2014, pp. 547–591. ISBN: 978-0-262-02818-9.

[110] J He et al. "On Quantitative Evaluation of Clustering Systems". In: *NETA* 11 (2003), pp. 105–133. DOI: 10.1007/978-1-4613-0227-8_4.

[111] Shixia Liu et al. "Towards Better Analysis of Machine Learning Models: A Visual Analytics Perspective". In: *Visual Informatics* 1.1 (2017), pp. 48–56. DOI: 10.1016/j.visinf.2017.01.006.

[112] Mengchen Liu et al. "Towards Better Analysis of Deep Convolutional Neural Networks". In: *IEEE Transactions on Visualization and Computer Graphics* 23.1 (2017), pp. 91–100. DOI: 10.1109/TVCG.2016.2598831.

[113] Jeremy Straub and Justin Huber. "A Characterization of the Utility of Using Artificial Intelligence to Test Two Artificial Intelligence Systems". In: *Computers* 2 (2013), pp. 67–87. DOI: 10.3390/computers2020067.

[114] Christoffer Holmgård et al. "Evolving personas for player decision modeling". In: *Proceedings of CIG 2014*. IEEE, 2014, pp. 1–8. DOI: 10.1109/CIG.2014.6932911.

[115] Noor Shaker et al. "The turing test track of the 2012 Mario AI Championship: Entries and evaluation". In: *Proceedings of CIG 2013*. 2013. DOI: 10.1109/CIG.2013.6633634.

[116] Chen Si and Chek Tien Tan. "Believable Exploration: Investigating Human Exploration Behavior to Inform the Design of Believable Agents in Video Games". PhD Thesis. University of Technology Sydney, 2017. DOI: 10.13140/RG.2.2.12353.04963.

[117] Prithvijit Chattopadhyay et al. "Evaluating Visual Conversational Agents via Cooperative Human-AI Games". In: *Proceedings of HCOMP 2017* (2017). arXiv:1708.05122, 9 pp.

[118] Tiago Machado, Andy Nealen, and Julian Togelius. "SeekWhence: A Retrospective Analysis Tool for General Game Design". In: *Proceedings of FDG 2017*. 2017, pp. 1–6. DOI: 10.1145/3102071.3102090.

[119] Onn Shehory and Arnon Sturm. "Evaluation of Modeling Techniques for Agent-Based Systems". In: *Proceedings of the Fifth International Conference on Autonomous Agents*. 2001, pp. 624–631. DOI: 10.1145/375735.376473.

[120] Mark A. Ardis et al. "A framework for evaluating specification methods for reactive systems". In: *IEEE Transactions on Software Engineering* 22.6 (1996), pp. 378–389. DOI: 10.1109/32.508312.

[121] Wookhee Min et al. "Multimodal Goal Recognition in Open-World Digital Games". In: *Proceedings of AIIDE 2017*. 2017, pp. 80–86.

[122] Jeff Orkin. "Three States and a Plan: The A.I. of F.E.A.R." In: *Game Developers Conference 2006* (2006), p. 18.

[123] Patrick Foo et al. "Do Humans Integrate Routes Into a Cognitive Map? Map- Versus Landmark-Based Navigation of Novel Shortcuts". In: *Journal of Experimental Psychology: Learning, Memory, and Cognition* 31.2 (2005), pp. 195–215. DOI: 10.1037/0278-7393.31.2.195.

[124] Claire Bradley and Joel Pearson. "The Sensory Components of High-Capacity Iconic Memory and Visual Working Memory". In: *Frontiers in Psychology* 3 (2012), Article No. 355. DOI: 10.3389/fpsyg.2012.00355.

[125] Karl R. Gegenfurtner and George Sperling. "Information transfer in iconic memory experiments". In: *Journal of Experimental Psychology: Human Perception and Performance* 19.4 (1993), pp. 845–866. DOI: 10.1037/0096-1523.19.4.845.

[126] Vincent Di Lollo. "Temporal characteristics of iconic memory". In: *Nature* 267.5608 (May 1977), p. 241. ISSN: 1476-4687. DOI: 10.1038/267241a0. URL: https://www.nature.com/articles/267241a0 (visited on 07/16/2019).

[127] R C Atkinson and R M Shiffrin. "The control of short-term memory". In: *Scientific American* 225.2 (1971), pp. 82–90. URL: http://www.ncbi.nlm.nih.gov/pubmed/5089457.

[128] Nelson Cowan. "The magical number 4 in short-term memory: A reconsideration of mental storage capacity". In: *Behavioral and Brain Sciences* 24.1 (2001), pp. 87–114.

[129] Earl K. Miller and Timothy J. Buschman. "Working memory capacity: Limits on the bandwidth of cognition". In: *Daedalus* 144.1 (2015), pp. 112–122. DOI: 10.1162/DAED_a_00320.

[130] Pierre Barrouillet and Valérie Camos. "As Time Goes By: Temporal Constraints in Working Memory". In: *Current Directions in Psychological Science* 21.6 (2012), pp. 413–419. DOI: 10.1177/0963721412459513.

[131] Timothy J. Ricker, Evie Vergauwe, and Nelson Cowan. "Decay theory of immediate memory: From Brown (1958) to today (2014)". In: *Quarterly Journal of Experimental Psychology* (2016). DOI: 10.1080/17470218.2014.914546.

[132] Wei Ji Ma, Masud Husain, and Paul M. Bays. "Changing concepts of working memory". In: *Nature Neuroscience* 17 (2014), pp. 347–356. DOI: 10.1038/nn.3655.

[133] Staffan Bjork and Jussi Holopainen. *Patterns in Game Design*. Game Development Series. Rockland, MA, USA: Charles River Media, 2004. ISBN: 1-58450-354-8.

[134] Tracy Fullerton. *Game Design Workshop: A Playcentric Approach to Creating Innovative Games*. 2008. ISBN: 978-0-240-80974-8. DOI: 10.1007/s13398-014-0173-7.2.

[135] Thomas W Malone. "Toward a Theory of Intrinsically Instruction Motivating". In: *Cognitive Science* 5.4 (1981), pp. 333–369. DOI: 10.1207/s15516709cog0504_2.

[136] Nick Yee. "Motivations of Play in MMORPGs - Results from a Factor Analytic Approach". In: *CyberPsychology & Behavior* 9.6 (2006), pp. 772–775. DOI: 10.1089/cpb.2006.9.772.

[137] Jukka Vahlo et al. "Digital Game Dynamics Preferences and Player Types". In: *Journal of Computer-Mediated Communication* (2017). DOI: 10.1111/jcc4.12181.

[138] Adam S. Kahn et al. "The Trojan Player Typology: A cross-genre, cross-cultural, behaviorally validated scale of video game play motivations". In: *Computers in Human Behavior* 49 (2015), pp. 354–361. DOI: 10.1016/j.chb.2015.03.018.

[139] Anders Tychsen and Alessandro Canossa. "Defining personas in games using metrics". In: *Proceedings of Future Play 2008*. 2008, pp. 73–80. DOI: 10.1145/1496984.1496997.

[140] Georgios N. Yannakakis and Julian Togelius. “Generating Content”. In: *Artificial Intelligence and Games*. Cham, Switzerland: Springer International Publishing, 2018, pp. 151–202. ISBN: 978-3-319-63518-7.

[141] Cameron Browne and Frederic Maire. “Evolutionary Game Design”. In: *IEEE Transactions on Computational Intelligence and AI in Games* 2 (2010), pp. 1–16. DOI: 10.1109/TCIAIG.2010.2041928.

[142] Georgios N. Yannakakis and Julian Togelius. “Playing Games”. In: *Artificial Intelligence and Games*. Cham, Switzerland: Springer International Publishing, 2018, pp. 91–150. ISBN: 978-3-319-63518-7.

[143] D. Perez-Liebana et al. “The 2014 General Video Game Playing Competition”. In: *IEEE Transactions on Computational Intelligence and AI in Games* 8.3 (Sept. 2016), pp. 229–243. DOI: 10.1109/TCIAIG.2015.2402393.

[144] Karol Gregor et al. “DRAW: A Recurrent Neural Network For Image Generation”. In: *Proceedings of ICML 2015*. arXiv: 1502.04623. Lille, France, 2015, 10 pp.

[145] Olivier Delalleau. *Smart Bots for Better Games: Reinforcement Learning in Production*. Tutorial Presentation. San Francisco, USA, Mar. 2019. URL: https://schedule.gdconf.com/session/ml-tutorial-day-smart-bots-for-better-games-reinforcement-learning-in-production/864367.

LUDOGRAPHY

A.I. Design. (1980). *Rogue*. PC, Epyx.

Bethesda Game Studios. (2011). *The Elder Scrolls V: Skyrim*. PC, Bethesda Softworks.

Bethesda Game Studios and Behaviour Interactive. (2015). *Fallout Shelter*. Android/iOS, Bethesda Softworks.

Blizzard Entertainment. (1998). *StarCraft*. PC, Blizzard Entertainment.

Blizzard Entertainment. (2004). *World of Warcraft*. PC, Blizzard Entertainment.

Blizzard Entertainment. (2014). *Hearthstone*. PC, Blizzard Entertainment.

David Braben and Ian Bell. (1984). *Elite*. PC, Acornsoft.

Demruth. (2013). *Antichamber*. PC, Demruth.

Crystal Dynamics. (2008). *Tomb Raider: Underworld*. PC, Eidos Interactive.

EA DICE. (2010). *Mirror's Edge*. PC, Electronic Arts.

EA Vancouver and EA Romania. (2018). *FIFA 19*. PC, EA Sports.

ConcernedApe. (2016). *Stardew Valley*. PC, Chucklefish.

Firaxis Games. (2016). *Civilization VI*. PC, 2K Games.

FromSoftware. (2011). *Dark Souls*. PC, Namco Bandai Games.

Gears for Breakfast. (2017). *A Hat in Time*. PC, Humble Bundle.

Hello Games. (2016). *No Man's Sky*. PC, Hello Games.

id Software. (2016). *Doom*. PC, Bethesda Softworks.

Klei Entertainment. (2013). *Don't Starve*. PC, 505 Games.

Maxis. (2000). *The Sims*. PC, Electronic Arts.

Maxis Redwood Shores. (2018). *The Sims Mobile*. Android/iOS, EA Mobile.

Monomi Park. (2017). *Slime Rancher*. PC, Monomi Park.

Moon Studios. (2015). *Ori and the Blind Forest*. PC, Microsoft Studios.

Mossmouth. (2013). *Spelunky*. PC, Mossmouth and Microsoft Studios.

Bandai Namco Entertainment. (1980). *Pac-Man*. Arcade, Bandai Namco Entertainment.

Nintendo. (1985). *Super Mario Bros.* Nintendo Entertainment System, Nintendo.

Nintendo. (2012). *Animal Crossing: New Leaf*. Nintendo 3DS, Nintendo.

Nintendo. (2017). *The Legend of Zelda: Breath of the Wild*. Nintendo Switch, Nintendo.

Nintendo. (2017). *Super Mario Odyssey*. Nintendo Switch, Nintendo.

Playground Games. (2014). *Forza Horizon 2*. Xbox One, Microsoft Studios.

Playground Games. (2016). *Forza Horizon 3*. Xbox One, Microsoft Studios.

Playsaurus. (2015). *Clicker Heroes*. PC, Playsaurus.

Re-Logic. (2011). *Terraria*. PC, Re-Logic.

Silicon Studio. (2012). *Bravely Default*. Nintendo 3DS, Nintendo.

Team Cherry. (2017). *Hollow Knight*. PC, Team Cherry.

Thekla, Inc. (2016). *The Witness*. PC, Thekla, Inc.

Tomohiro Nishikado. (1978). *Space Invaders*. Arcade, Taito.

Ubisoft Quebec. (2018). *Assassin's Creed Odyssey*. PC, Ubisoft.

Unknown Worlds Entertainment. (2018). *Subnautica*. PC, Unknown Worlds Entertainment and Gearbox.

Valve Corporation. (2013). *Dota 2*. PC, Valve Corporation.

Valve Corporation. (2011). *Portal 2*. PC, Valve Corporation.

Valve South. (2008). *Left 4 Dead*. PC, Valve Corporation.

ZeptoLab. (2010). *Cut the Rope*. Android/iOS, Chillingo.