UNIVERSITY OF CALIFORNIA
SANTA CRUZ

DESIGN OUT HELPLESSNESS: AI INTERVENTIONS FOR GAME INCLUSIVITY

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTATIONAL MEDIA

by

Batu Aytemiz

March 2022

Table of Contents

List of Figures

Abstract

Design Out Helplessness:

AI Interventions for Game Inclusivity

by

Batu Aytemiz

There are no single-player videogames. When someone plays a game, they are not only interacting with the game's contents, but also engaging with the culture and connections that are built around that game. This channel by which people connect to one another gets cut off when the design of the game leaves a player feeling helpless and excluded. Informed by approaches from game studies, human computer interaction, and artificial intelligence (AI) research, this work explores how we can fight this unintended exclusion.

From a theoretical standpoint, this work expands our vocabulary of failures in games, allowing us to differentiate *in-loop* from *out-of-loop* failures. It teaches how to design AI-powered assistance methods, allowing us to repurpose game-playing AI agents in service of game inclusivity. From a technical standpoint this work contributes three prototype systems: a dynamic tutorial framework, enabling games to respond to the skill levels of players; an AI intervention for motor accessibility, helping players to adapt dexterity based games for deliberative play; and a reinforcement learning based navigation agent, enabling access to a portfolio of assistance methods.

As a whole, this dissertation constitutes a step towards making games more inclusive, not for the sake of games, but for the sake of the people who play them.

Benden önce gelen hocalarıma, ve benden sonra gelecek yeğenime.

Acknowledgments

Only my name is written on the title page, yet, this is a gross under representation of how many people contributed to this dissertation. These contributions came in many forms: conversations over beer, a push back on my ideas, an inspiring talk, a passing remark, an opportunity, a freshly baked cookie and much more. I am only able to write these words because of the tens, if not hundreds of people who explicitly and implicitly supported, challenged, protected, angered, inspired, educated, and nurtured me.

Ablam, bana böyle bir yolun mümkün olduğunu gösterdiğin için;

Anam, her zaman yelkenimdeki rüzgar olduğun için;

Babam, kendime güvenmemi ve hayattan korkmamamı sağladığın için;

Adam M. Smith, for broadening my horizon with every conversation, and exemplifying all-encompassing kindness, boundless intellectual curiosity and relentless bias for action;

Grammy's House, Sarah, Ethan, Mase, SB, Dave, Aya, David, Caleb, and Brushes, for creating a home away from home, and the copious amounts of delicious food;

Dostlarım, Doruk, Ezgi, Ersel, Buse, Deniz, Ecem, Ege, Eylül, Gülsen, Karakaş, Özge, Sarp ve Defne, her evime döndüğümde, sanki hiç ayrılmamışım gibi benimle kaldığımız yerden içmeye devam ettiğiniz için;

Olivier Delalleau, for taking a chance on me when I bombed the interview and, with deep patience, teaching me the craft of reinforcement learning;

My desert-dweller friends, Sofia, Myera, Mona, Spencer, Koh, Su, Attilio, Matilda, Sugandha, Kristopher for helping me step outside my context at least once in a while and your unwavering friendship;

Isabella, for proof reading this document line by line, word by word;

Oskar, for the rolling conversations and showing me even slimy mold can be endlessly intriguing;

Motiv upstairs, for providing a place to work and the napkins that housed the first sketches of my research;

MJ and Tad, for sharing your deep insights on the craft of games and teaching me how to teach;

Anisha, for all the perfectly brewed tea;

Sarah and James, for the many hours of conversations around the fires and the meticulously crafted cocktails;

Jimmy, for getting me a job, hours of media consumption and the idea for my final project;

Cassidy, for inviting me out of my comfort zone;

Johnathan Pagnutti, for answering my endless flurry of questions in my first year;

Gonca Hocam, her memlekete döndüğümde benimle kahve içip dertlerimi dinlediğiniz ve kahkahalara boğduğunuz için;

Leila, for giving me job measuring the size of fish and helping me pay my rent when I got fired;

Alena, for always listening deeply, asking the most thoughtful questions and helping me write one more word at a time;

Lachlan, mate, for all the bottomless pitchers, myriad new perspectives, a kick-start on anarchism and, simply, your guidance and friendship;

Eda and Jon, for the endless laughters, enchanting karaoke performances and many, many graphs;

Christoffer Holmgård, for grabbing coffee with me during my internship and suggesting that I look into getting a Ph.D. after graduation;

Sercan, ÜNOG'da birlikte gerçekleştirdiğimiz projeler için;

Emily and Theo, for ensuring that our department kept functioning and advocating for me;

To my research collaborators, Ken, Isaac, Jesse, Nick, Ross, Zeping, Xueer, Hunter, and Eric for sharing your ideas, time and attention with me;

The Computational Media community for expanding my ideas and being my intellectual springboard;

My committee for reading my dissertation without any compensation;

COLA, Lori and, Cindy for radicalizing me;

To the many foodcarts for feeding me, and bus drivers for transporting me;

To all my teachers, for inspiring and mentoring me;

To all my students, for giving me something to work towards;

I thank you all. I am eternally grateful.

Chapter 1

Introduction

My mother told me that she didn't like playing videogames. She explained how as I was growing up she wanted to play videogames with me, but it never worked out: "When I first tried to play a game with you, I couldn't understand what to do, the game made me feel incompetent, and I felt ashamed." — Personal anecdote from the author.

Videogames are an essential part of millions of peoples' lives. [69, 171] Yet, many others are excluded from experiencing videogames. This exclusion is due to the assumptions games make about what players know and don't know, and what players can and can't do. When a game rejects a player, whether by having the first levels be too difficult to understand or by not being accessible enough, players are not only excluded from engaging with the game's contents, but more importantly, the whole culture and the connections that are built around that game—the late night play sessions, the questionable fan art, the endless arguments over strategies, and more. Given that there are more than 2 billion videogame players out there already using games as a way to connect [220], ensuring everyone has a chance to take part in this community is crucial. With my research, I am working to make games more inclusive, so that no one is excluded from playing games by feeling helpless.

Research to make games more inclusive also benefits players who already "get it." According to Raph Koster's Theory of Fun [189], a widely accepted philosophy in the games industry, "fun is a response to learning." If we develop our understanding of how to support our players' learning, we can make our games more engaging for all our players. This includes experienced videogames players jumping into a new game with less handholding, as well as having systems that can keep training even the expert and competitive players.

Furthermore, I explore the tension between designer intent and player diversity–the myriad of ways players can and prefer to play. Designers who can distill their vision into uncompromising experiences are often applauded by those who can engage with these experiences. But what about those who are left excluded? In my work I present an approach to enhance the accessibility of games while retaining the intent of the designer.

By making games more accessible experiences and more intentional teachers, we can reduce the exclusion of players and ensure that more people can fully engage with all the challenges and joys our games have to offer. My research aims to improve the effectiveness of assistive systems by utilizing artificial intelligence (AI) techniques. In this dissertation I describe the research I have conducted towards this goal while also contextualizing my work in the broader literature.

My dissertation primarily asks two research questions:

RQ1: What lenses on games and the game design process will allow us to improve inclusivity in games without negatively affecting the design intent?

RQ2: How might artificial intelligence techniques be used to intervene in game designs to design out the feeling of helplessness and exclusion?

Rather than answer these questions independently, my proposed work has developed example interventions inspired by preliminary answers to the first question. I intend for my work to impact practicing game designers rather than to catalog abstract knowledge. More specifically, in my dissertation, I have:

- Expanded our understanding of failure in videogames. In many applications, you don't want your users to fail. In games this is not so—failure is an essential component in many games. Therefore, making games easier across the board isn't a viable method of making videogames more inclusive. To make videogames more inclusive while retaining these core aspects, we need to understand the tension between challenge, access, and player failure and have a richer vocabulary to discuss and analyze these concepts. This work was published as the conference paper, "A Diagnostic Taxonomy of Failure [50]".

- Showed that gameplaying AI techniques can be repurposed to make games more inclusive. Advances in AI techniques have resulted in immense breakthroughs in how well we can algorithmically play videogames. Yet, this increased investment in game-playing AI (GPAI) techniques has not translated into a tangible improvement in our players' game-playing experience. I have proposed to repurpose this abundant research to assist the player. This work was published as the conference paper, "Your Buddy The Grandmaster Repurposing the Game Playing AI Surplus for Inclusivity" [49].

- Prototyped an AI intervention for learnability. Most tutorials are cumbersome, and they are notoriously hard to tune appropriately, either by not being comprehensive enough and leaving novices behind or being too overbearing and irritating the experts. Tutorials need to serve all skill levels to ensure people from various backgrounds can fully enjoy the game. I have proposed a dynamic tutorial system to help players who need it, but only when they need it. This work was published as the conference paper, "Talin: A Framework for Dynamic Tutorials Based on the Skill Atoms Theory" [45].

- Prototyped an AI intervention for motor accessibility. Similar to the diversity in game literacy, players are also diverse when it comes to motor abilities. This prototype allows converting high-paced dexterity based challenges to deliberative planning challenges in a narrow set of games that exist in the wild. This work was published as the paper "SMES: Adapting Dexterity-based Games for Deliberative Play [48]".

- Prototyped a reinforcement learning-based navigation AI to create assistive methods. Navigation is ubiquitous in videogames, and yet for those who have zero experience in 3D games, it presents a significant challenge that needs to be mastered before the rest of the game can be enjoyed. In this RLNav project, I built a 3D parkour game with no combat and designed assistance methods that use the reinforcement learning-driven navigation system.

- Released the *NavAssist* research pipeline. For AI driven-assistance methods to be effective, we need both robust agents and also well-designed assistance methods. This need for an interdisciplinary approach makes it difficult for more specialized researchers to make progress in this new area. To help alleviate this problem I have contributed an open-source research pipeline which includes: (a) for reinforcement learning researchers, five OpenAI Gym compatible navigation environments to train more effective agents; (b) for game designers, a Unity game project and the pre-trained models to implement new AI-driven assistance methods; (c) and for human-computer interaction researchers, a set of example assistance methods powered by pre-trained models to investigate and evaluate the effects of those assistance methods.

Chapter 2

Background

With my research, I hope to contribute to videogame inclusivity, not for the sake of games, but for the sake of those who play them. In this chapter, I show how playing even single-player games is a social act; present the tension between challenge and inclusion; describe the lacking state of in-game tutorials; and describe how AI techniques can be utilized to further inclusion.

2.1 There Are No Single-player Games

The COVID-19 pandemic has assisted in convincing people that games are culturally and socially relevant. The World Health Organization has encouraged people to play videogames [24], and as of 2020, there are more than 2 billion game players worldwide, with the largest games reaching more than 300 million players. [6]

Games help people form communities and build connections. Multiplayer games are accepted as important sources of human connection with persistent avatars and consistent interactions [71]. However, I argue that even single-player games play an essential part in how people connect. The lead level designer for the original *Spyro the Dragon* [166] series, Michael John, argues that there are "No Singleplayer Games" [170] and explains that even though single-player games often support just one controllable character, people often push these boundaries and find ways to play together. He describes the existence of a common culture where single-player games are played by multiple people on the same couch taking turns, or one person watching the other play. With the recent advent of livestreaming, the number of people watching has become drastically larger [172]. John shows that even when a play session is solitary, there is a slew of ways people connect through games outside the game sessions. He mentions the prevalence of speedrunning communities [270], creator fan groups [246], and communities of practice [233].

This is all to say when someone is excluded from playing the game, they are not only excluded from the game contents but also excluded from the social network that is weaved around the game and the potential connections brought by this network. Thus we must make games more inclusive not as a way to reach more customers or improving the design of games, but rather, as a necessity to ensure no one is excluded from the larger conversation that humanity is having.

At the outset, making games inclusive might seem like a simple task. There are well-established guidelines for digital accessibility [148, 188], and for an organized design team following compliance rules could be straightforward. However, we will see in the next section that a common game design ethos is prone to designing for an implied player [26] and not recognizing the diverse ways our players engage with the games we develop. This prevalent (and effective!) design ethos suggests that to make games more inclusive our theories must be integrated into game design theories and be in connection with the practices, requirements, and expectations of both players and developers.

2.2 Tension Between Designer Intent and Player Diversity

> "In *Celeste*, we try to gently push the player to do things that they thought were impossible for them. We also accept that every player is different and that people come into the game at many different skill levels." — Matt Thorsen[1]

I grew up playing *Ortada Sıçan*[2]. It is a game where two throwers attempt to pass a ball to each other while a catcher in the middle tries grabbing it. I found this folk game lacking as it pushed me to become frustrated with my friends: the two throwers would stand very far apart and lob the ball high over the catcher. This means the catcher would have almost no chance of grabbing the ball. Sure, the throwers handily won, but the game was utterly boring without any tension. Ignoring my pleas as a budding game designer, my friends had traded fun for an easy victory. As the designer I thought *Ortada Sıçan* was never meant to be played like this and my friends were, utterly and fundamentally wrong.

The sacred "intended way to play" belief is common in game design, and many designers suggest that these types of opinionated games result in more engaging experiences. Soren Johnson, lead designer of *Civilization IV* [116], describes how ultra-

[1]https://www.vice.com/en/article/d3w887/celeste-difficulty-assist-mode

[2]Also known as Keepaway, or Rat, Piggy, Monkey, and other animals in the middle in different cultures

competitive players would abuse game systems to get an edge in the game, while also making the whole experience much more grueling, effectively "optimize the fun out of a game." [173] The Game Director of *Doom Eternal* [165] Hugo Martin explains how there is a correct way to play *Doom*, and if the player isn't doing what they are supposed to, they shouldn't be allowed to succeed–the player shouldn't be able to progress in the game via an "un-fun" way of playing. [23]. In a similar vein, Sébastien Bénard once shared that while engaging with the early iterations of *Dead Cells* [214] players were not "playing in the way he'd intended" and his team had to make drastic design changes to ensure the player wasn't simply picking the most effective way to progress in the game [70].

In all these previous examples, the designers identified a vision of how the implied players [26] should have fun playing their games and took steps to eliminate other ways of progressing in these game. Arguably the games were better for it. However, not every attempted change is successful. When designing *XCOM 2* [118], Jake Solomon wanted the players to take more risks, instead of playing the game in a slow and methodical manner [208]. To incentivize a faster pace, *XCOM 2* introduced turn timers that resulted in a Game Over upon reaching zero. A significant portion of the player base responded negatively to this change [10], with mods to disable the turn timer soon popping up. These examples illustrate the underlying tension between the design intent and the multitude of ways players can engage with a game.

This tension can become more controversial when the design intent is built upon the core idea of the game being difficult, especially when the difficulty of the game results in the game excluding certain players. For example, a high level of dexterity might be a core component of a given action game, yet this dexterity requirement might mean that people with motor disabilities will have problems accessing this game's contents. And yet, many game designers and scholars, myself included, consider challenge and failure an inherent component of games [207, 176, 174]. In various cases, challenge and failure are tailored to be a big part of the whole experience — to the extent that McGonigal [207] claims players spend a significant amount of their time failing. On one hand, certain challenges and failures exclude a portion of the player base, on the other hand, challenge and failure are a core aspect of many games.

I am interested in exploring this tension: with this thesis I argue that there are two areas we should focus on to help make games more inclusive while retaining the challenge and difficulty of games.

1. Developing comprehensive assistive methods to support diverse groups of people
2. Making games more intentional teachers of their own systems

Assistive modes are one of the main ways to make games more accessible, and making games more accessible is essential as videogame players come in all shapes, sizes, ages, and abilities. If a player cannot engage with a game due to a disability they might have, then by definition they will have a harder time participating in the conversation built around that game. Games must be able to support the players without being condescending or removing the challenge altogether.

Making games more intentional teacher is essential as most games assume a certain level of game literacy from their players, which results in players with less gaming experience (whether they be older folks, people who didn't grow up in communities where videogames are popular, or those who simply want to learn games in a different stage of their lives) being excluded. More importantly, however, games must teach to the absolute novices without boring the experts.

Both goals require certain technical capabilities to be reached. If a system is to be an effective teacher, it would be beneficial for the system to be an expert in the competitions it is trying to teach. Similarly, if a system is supporting its players' in playing the game, then the system itself should better be able to play the game itself. Furthermore, it will be useful to model the player in detail so a useful context could be presented to them. Both competently playing videogames and modeling players are problems that can be solved with AI techniques. In the coming sections, I will describe how AI techniques are useful in tackling these problems.

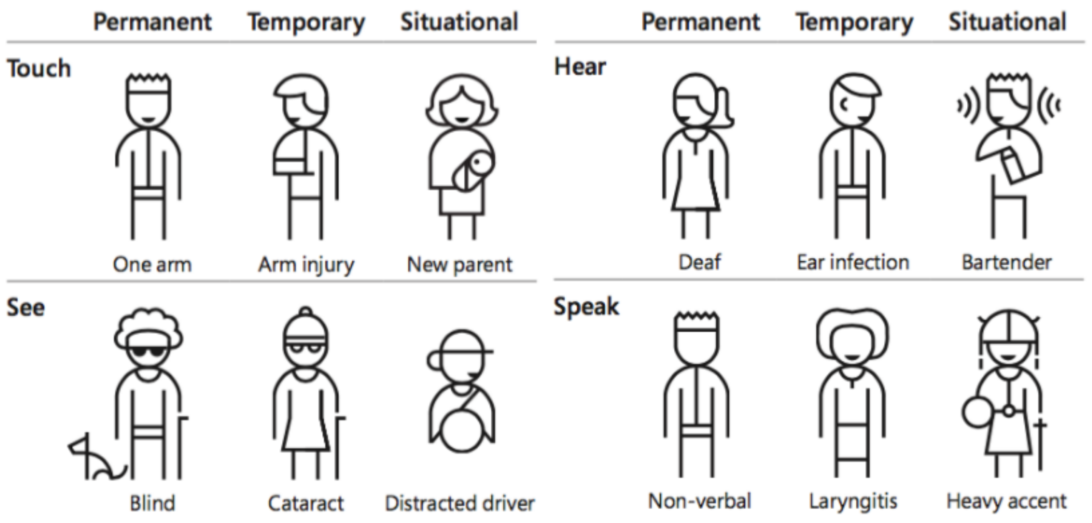


Figure 2.1: Examples of permanent, temporary and situational disabilities. Taken from *Microsoft Inclusivity Handbook* [12]

2.3 Accessing Games

"I just want to grab all you developers and share our tears, our struggles, and just how badly we want to play your games." — Chris "deafgamersTV" Robinson[3]

The World Health Organization defines disability as:

"Disability is not just a health problem. It is a complex phenomenon, reflecting the interaction between features of a person's body and features of the society in which he or she lives." [4]

This perspective states that disability arises from a mismatch between a person and the context they are in [152]. This approach is wildly different from thinking that disability is an unchanging attribute of the person themselves. The mismatch perspective makes it easier to recognize that not all disabilities are permanent. I am typing these words one year after having broken my collarbone on my dominant side—I

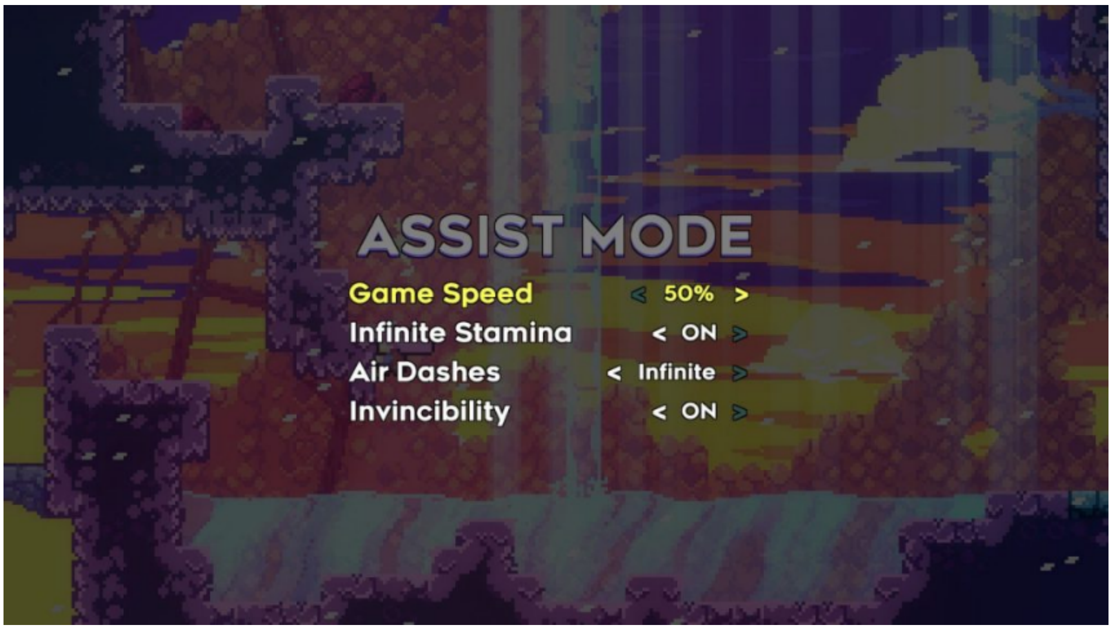
[3]https://twitter.com/GA_Conf/status/1092555146628714496

can easily remember the time in which I was temporarily disabled when it came to using my right arm.

Mismatches can also arise from the context we are in. I do not suffer from color-blindness or any other visual impairment, yet I used the high-contrast visual aid mode while playing *Mario Odyssey* [226]. The reason behind this was related not to me, but to the time of day: I was playing *Mario* with a projector, and whenever I tried to play during the day in my room without blinds, the game's colors would look washed out, and I was not able to differentiate the different colors. I was suffering from a situational mismatch, and having the option to increase the game's contrast helped me solve the issue.

Inclusive Design is a method that aims to minimize these mismatches [278] and also focuses on "designing a diversity of ways for everyone to participate in an experience with a sense of belonging." When I use the term "inclusivity" in my title Design Out Helplessness: AI interventions for Game Inclusivity, I am referring to this perspective of offering more options for players to participate in playing videogames. It is important to note, however, this approach isn't the only aspect of inclusivity in games. The specific work in this thesis leaves out conversations around gender representation [164] and cultural inclusion [96] as a way to focus on the issues of learning and accessibility.

Apart from increasing the effectiveness of game tutorials, another way to make games more inclusive is to create "assist modes" [248]. Every player is different and might have different needs to fully engage with a game [152]. Some players might find a game inaccessible due to its difficulty; other players might find the same game inac-


ASSIST MODE

Game Speed < 50% >

Infinite Stamina < ON >

Air Dashes < Infinite >

Invincibility < ON >

Figure 2.2: The assist mode of Celeste [205]

cessible due to design choices such as color schemes and specific input mappings [199]. Offering a portfolio of assistance methods can help us reduce the mismatches between our players and our games.

Assist modes that provide such assistance methods are becoming more popular. The assistance mode in *Celeste* [205] is one of the most comprehensive and has garnered praise from players and designers alike [187]. With it, the player can tweak almost every aspect of the game's difficulty: game speed, total climbing stamina, the number of allowed dashes, and invulnerability are among the many options.

The inclusion of assist mode in *Celeste*, combined with the release of other difficult games without any sort of assistance, such as *Cuphead* [291] and *Sekiro* [124], sparked a series of discussions surrounding the design intent and the value of difficulty in videogames [191, 302]. Challenge in many games is essential for the design and is shown empirically to increase enjoyability [93]. Assist modes do not aim to make games easier; rather, they make games more accessible. Increasing the player's options decreases the likelihood of a mismatch between the game and the player's capabilities. However, while we have come a long way in improving game accessibility [121], there is still room for improvement.

One of my favorite tools to help navigate this tension between designer intent and accessibility is the Accessible Player Experiences (APX) toolkit [58]. APX is a set of design patterns that invites designers to think about both Access and Challenge. Access refers to the ability of the player to "be able to perceive what is going on in the game and to take control of actions in the game. In other words, players need to be able to sense (see/hear/feel) the output of the game and to provide input to the game (click/tilt/speak)." [58] Before anything else, it is important for players to be able to access the game to start playing. However, even after gaining access to a game, some players might not be able to make progress because of the challenges they face.

The design patterns APX provides offer alternative ways to think about how players can respond to both Access and Challenge problems they might encounter. Each of these patterns acts a lens for the designer to evaluate their own game, and illuminates potential blindspots that the designers might have when it comes to access and challenge.

Access Patterns

- Second Channel
- Flexible Text Entry
- Clear Text
- Same Controls but Different
- Improved Precision
- Do More With Less
- Personal Interface
- Flexible Controllers
- Distinguish This From That
- Leave it There
- Flexible Displays
- Clear Channels

Challenge Patterns:

- Total Recall
- Slow it Down
- Save Early, Save Often
- Moderation in All Things
- House Rules
- Undo Redo
- Helping Hand
- Training Ground
- Bypass
- Play Alongside

All of these patterns are relevant in our goals. However, two of them, Do More With Less and Helping Hand, lend themselves especially well for being powered by AI methods.

Do More with Less is a pattern that invites designers to recognize that not every player can input long sequences of actions reliably. This might be due to motor disabilities, or cognitive disabilities. The proposed design solution is to reduce the number of actions in any sequence so that players can successfully make progress in the game.

Helping Hand invites designers to recognize that some players might need additional support in the game to make progress, it also suggests the implementation of different assistance methods.

I see both of these patterns as two (among many other) opportunities for AI techniques to play a role in making games more inclusive for our players. While neither of these design patterns require an AI technique to be implemented, I argue that using AI techniques can further enhance the effectiveness of these patterns. For Do More With Less, AI systems can chunk groups of actions together in a context-sensitive manner, and for Helping Hand, the assistance might even take the form of a supportive character. I further develop these ideas in my own work "Your Buddy, Grandmaster: Repurposing Game Playing AI Surplus for Game Inclusivity" which I later introduce.

2.4 The State of Tutorials

> "Do you ever feel like too stupid for a game? Like you want to love the game but can't grasp the mechanics for it to be fun?" - A question posted on Reddit [4]

Tutorials are how videogames teach their players how to play. They become especially important if the player doesn't have much videogame experience and has to learn the very basics of the genre. However, if tutorials don't discriminate and explain

[4] https://www.reddit.com/r/patientgamers/comments/et93em/do_you_ever_feel_like_too_stupid_for_a_game_like/

the very basics of the genre to each and every player instead, the play experience of experts is affected negatively. This is such a frequent occurrence that the common consensus around vocal videogame players is that tutorials are almost always bad:

> "No one likes tutorials." [253]
>
> "Onboarding? Vomit." [313]
>
> "Great tutorials are hard to come by, and bad ones are unfortunately in over-abundant supply." [250]
>
> "We all hate in-game tutorials." [169]
>
> "The T-word everyone skips." [128]

Not limiting ourselves with explicit quotes, several other articles that discuss how to design tutorials imply that often they are not seen in a positive light by the players. [30, 59, 98, 110, 311] What we are seeing here might be a selection bias, where players show their dislike for the tutorials they recognize, whereas the tutorials that are effective without being oppressive are not appreciated simply because they have not been recognized explicitly as tutorials. This suggests the need for a validated review of whether these claims are in reality substantiated or at the very least more academic focus on tutorials.

Surprisingly, especially given the amount of possible room for improvement, game tutorials are an underdeveloped field of research. Green et al. [143] also reaches the same conclusion with their paper that "surveys the (scant) literature on game tutorials". This might be because there is an overwhelming amount of research when it comes to using games in learning and education [28, 131, 146, 268] and fewer researchers are concerned with how people learn in commercial videogames over the education system.

Regardless of the research attention tutorials gets, it is immensely important to overall player experience, and whether the game will be commercially successful.

In the Experimental Game Development Podcast, the developer of *Steambirds Survival* [288], Andy Moore, describes how adding a relevant tutorial that helped players learn the game boosted the ratings of the game from a mediocre 5 to a respectable 8. [213]

While the experience of Andy Moore is an individual data point, larger studies involving millions of data points also suggest similar trends. A 2019 report on mobile gaming statistics shows that, on average, over 70% of players stop playing a game within the first day. [20] The report describes the game being too complex for users to understand as one of the major reasons for this low level of retention. [20] Although I am personally uninterested in focusing on maximizing retention numbers, it is an undeniable fact that most people create media so that others engage with it. Data shows that a majority of people's engagement will be, for better or worse, constrained to the initial areas of the game; therefore, such the tutorial segments carry a significant amount of weight in deciding the long-term success of any game.

On the more academic side of things, Cheung et al. try to pinpoint and explain how important the first impressions of a game are [80]. They analyzed over 200 game reviews and interview industry professionals to identify which aspects of an initial play session are relevant to keep players engaged, and whether the first play session is important at all. They, not surprisingly, find that in the first hour "players reconcile their expectations with their initial experience, setting the tone for the rest of the game or, perhaps, giving up on playing the rest of the game entirely." This agreement between industry professionals and academics on the importance of tutorials suggests that research into tutorials is a field ripe for exploration.

What is not explicitly adressed and commented on in these studies and articles, however, is the impact of losing a player during the tutorial stage. From a metrics perspective, it might not matter whether a player stopped playing because they got bored, or because they couldn't understand the game—in the end, the player stopped playing. However, from the player's perspective, there is a huge difference. The feeling of engaging with the game and then finding out you are not enjoying it is very different from being unable to even start playing the game because you couldn't figure it out. A player who experiences the former might simply move on to another game, whereas a player who cannot learn how to play a game from the start might give up on games entirely.

This important (and currently only anecdotally supported) effect of losing players right as they are just beginning a game is why, I argue, *how* people learn games is a crucial facet of game inclusivity. This becomes even more important as more and more people who don't traditionally play games (parents, grandparents, that person who has never used a computer before) start to play.

Through my work, I claim that by increasing the effectiveness of our tutorials, we can make games more inclusive without sacrificing the enjoyment of our more expert players. Additionally I suggest using AI techniques to make games more adaptive when it comes to deciding when and how to teach the player.

2.5 AI in Games

"If It Works, It's Not Al." —Eve M. Phillips [245]

The definition of artificial intelligence (AI) is notoriously difficult to pin down. Given that this dissertation is being written during the machine learning boom, it is important to note that I do not equate AI with deep learning. Instead, for the purposes of this dissertation–making games more inclusive–I will assume a more generous definition: "*Systems that imitate intelligence to accomplish tasks.*" In my work, I operationalize this definition through various prototypes that I developed: an agent that can navigate in physics based videogames, an overlay that represent and enables the editing of user input and a framework that updates and reacts to beliefs of player skills. However, these are only the systems that I have chosen to build–the theories and approaches I contribute in this work are agnostic to the underlying AI systems, and can likely be adapted to use any AI technique.

If we narrow our focus to the videogame technical research domain, the book *Artificial Intelligence and Games* describes three[5] high-level use cases for academic AI in games: Procedural Content Generation (PCG), Player Modelling and Gameplaying. [317]. Among these three fields, I make extensive use of the research in gameplaying and mostly decide not to utilize advances in PCG and player modeling.

[5]This is certainly not to say that there are only these three fields in technical games research, but rather that I find it productive to put my work in conversation with these three fields specifically.

2.5.1 Why Gameplaying?

Throughout my time in graduate school during 2010s there have been several highly publicized and marketed results in gameplaying research with Deep Q Learning [212], AlphaGo [281], OpenAI Five [60], Atari benchmarks [53], and Starcraft [309]. This increased publicity has created a lot unfounded hype around how AI in player-facing videogames has also improved. Yet, most of this research has been focused on pushing the score benchmarks instead of on enhancing the player experience. Unfortunately, even though AI agents have gotten stronger and stronger, this has had disproportionately little impact on the gameplay experience of the players. I claim that harnessing the strength of gameplaying AI agents in the service of inclusivity is an effective method in realigning this body of research with the experience of players. I will include a more detailed discussion of this topic to Chapter 6.

2.5.2 Why Not Procedural Content Generation or Player Modeling?

On the surface, procedural content generation (PCG) and player modeling seem to be an appropriate fit for making games more inclusive. One can generate whatever content is most suitable for the player, and there already exists a body of work already aiming to do so [283, 316, 122, 303]. However, there are two reasons why I personally have decided not to focus on PCG and player modeling techniques in my work.

First, I am more interested in hand-authored experiences. This in no way is an attempt to diminish the authorial expertise of PCG algorithm designers or content authors, but rather, is a recognition of the increased difficulty in conveying design intent through procedural systems. As described before, I am excited to explore the tension between the design intent and the player inclusion. Adding the layer of procedural content generation between the player and the designer, while perfectly feasible, would have made my exploration more difficult.

Second, throughout my Ph.D. research I have grown more and more skeptical about the efficacy of AI systems that force their outputs on their users. Much have been written on this topic from the perspectives of fairness [234], gender [241], race [230], design [90] and more. Inspired by this crucial body of work, and humbled by the mistakes of my own predictive systems, I would like to present a perspective influenced by anarchism.

2.5.3 A Brief Anarchistic Perspective on AI

The term "anarchism" even as a political philosophy, is overloaded and needs some disentangling before we can use it to interrogate AI approaches. Anarchism does not refer to the characterization of molotov-cocktail wielding rioters who supposedly want to burn everything to the ground; but instead a political philosophy that is focused on self-determination and the reduction of hierarchical structures [221].

My preferred definition comes from American linguist and anarchist Noam Chompsky: "Anarchy is the idea that every form of authority and domination and hierarchy, every authoritarian structure, has to prove that it's justified—it has no prior justification." [81]. What, therefore, does authority and domination mean in the context of videogames? I take them to be the sum of the non-negotiable design decisions that the author forces upon the players.

These decisions can be anything: the implementation details of character creators (or lack thereof like *Rust* [114] which assigns the player a random avatar that cannot be customized); the manner in which the players can save the game, at will or predefined; the outputs from an AI system—think generated levels, or adjusted difficulties; or even the viability of certain playstyles. For example, in an interview game director Hugo Martin describes how Doom Eternal [165] kills players who aren't playing the way the game is intended. This is made to ensure the players stay in the fun zone [2].

I am in favor of designers being opinionated in bringing their visions to life, especially because videogames are, almost by definition, consensual activities. The player can opt-out at any given time. In my humble opinion, the authority of *Doom Eternal* over the play-styles of the player is justified.

However, I am skeptical of an automated system making immutable decisions on behalf of the players–say, creating levels they need to traverse–without their input attempting to maximize a specific metric. When we, as the designers, model a player, we make dozens of assumptions. Some of these assumptions are bound to be wrong. Let me illustrate with a fun example: the engineers who designed the automatic transmission of my car assumed that I don't use my car as a dancing aid.

One of the joys of driving is being able to bounce the car up and down to the beat of the music you are playing. For this to work effectively, however, you need to switch to a gear lower than normally suitable for driving at whatever speed you are traveling—the idea is to get a jolt of movement with each press on the gas pedal. An automated transmission makes this almost impossible, as it is primarily focused on providing a smooth driving experience. The assumption that the driver wants a smooth driving experience is unfounded, and the automated system prevents the driver from achieving my goals. Along these lines, I find it unrealistic that an AI system can account for the boundless creativity and diversity of players. Any hierarchical decision that a designer or an AI system makes on behalf of the player is likely to leave some groups out.

To bring this conversation back to videogames and my research, let us use this anarchistic lens of forced design decisions to briefly consider two approaches to difficulty adjustment: Assist modes, where the player makes the decision, and dynamic difficulty adjustments (DDA), where the player model makes the decision.

Dynamic difficulty adjustment [158] is the idea of, as the name suggests, the game automatically adjusting challenges to prevent overwhelming the player or being seen as trivial. In order for a DDA system to work, it needs to be able to assign a value that measures the difficulty level of the game felt by the player for any given game state. This measure is typically done through a set of handcrafted heuristics that use a combination of variables such as score and health points, and is blind to anything that is not instrumented. After this measure is defined, several different techniques can be used to adjust another set of parameters to make the game easier or more difficult. [324]

In most cases, when the DDA system makes a decision the player cannot overrule the authority of the system. This can lead players to the feeling of being patronized [29] due to their lack of agency. Even if DDA improves the player experience for normative play, transgressive playstyles such as speedrunning or simply messing around are most likely not going to be acknowledged and appropriately responded to by the system. How could the designer of the system even know what appropriate might mean?

In contrast, assist modes that expose the options for adjusting difficulty do not act on the behalf of the player and make no assumptions about the state of the player. The agency of the player is respected, and there is much less authority imposed by the designer. I believe this is much more in line with the anarchistic principles of empowerment that I am interested in pursuing with my work: SMES provides an alternative interface that the user can decide not to utilize, while Talin attempts to predict the skill level of the player, In Talin's case, the player is fully empowered to ignore it–more so, if the player ignores the actions of Talin and still makes progress, by design Talin will fade into the background. In Reinforcement Learning Powered Assistance methods, the system never intervenes on behalf of the player, but instead provides a plethora of assistance methods where the player can choose from.

As such, this dissertation is more focused on empowering the player to match the designer's intent, not on changing the game to match the player's perceived state. Following this principle I intend my work not to patronize but rather to empower the player. In the next chapter I will discuss the methods I utilized to reach this principle.

Chapter 3

Research Approaches

In this section, I give a brief overview of how I identified which problems to tackle, as well as the methods I used to make progress in solving them.

3.1 Problem Solving Approaches

Videogames are an inherently multidisciplinary topic, and so is my approach to making them inclusive. Throughout my work I have drawn from several different fields, in such a way that one compliments the other. The following is a brief overview of how I structured and approached each of my research projects.

3.1.1 Game Studies: *Play and Reflect*

It was important for me to ensure that the AI interventions I built did not detract, but rather complimented the vision of the game, whatever the vision might be. Given a naive approach to assistance, the most common risk was hampering challenge aspects of a given game. As such, I had to expand our understanding around challenge and failure.

For the Taxonomy of Failure, I applied the game studies lens to this exploration—that is a significant amount of playing and reflecting. I recorded myself playing several challenging games such as *Getting Over It With Bennet Foddy* [120], *Sekiro* [124], and *Dark Souls* [55] while applying the "talk aloud" [106] method while recording both myself and the screen. During these play sessions I would mumble, groan, curse, cheer, light up, and sigh. Each emotional quip, and sometimes their accompanying self-reflections, became my dataset of responses. I further expanded on this dataset by watching other people play challenging games, mostly on Twitch and YouTube.

The ideas found in the Taxonomy of Failure paper—such *in-loop* and *out-of-loop* failures—and the six classes of failures emerged from analyzing and annotating this dataset, which constitutes my approach to Game Studies.

3.1.2 Research Through Design: *Computational Caricatures*

In the Game Studies approach, a sizeable amount of time was spent constructing theory. For the other projects, especially Talin, SMES, and Repurposing AI Surplus, I chose to spend majority of my time designing and developing to explore and understand the space I am interested in, subscribing to the research through design methodology. [289]

More specifically, I made ample use of the Computational Caricature methodology. [284] A computational caricature exaggerates the salient aspects of the game design process while downplaying or distorting all other points. A computational caricature has claims (to be quickly recognized and understood) and oversimplifications (to be overlooked).

In the Talin project, I claimed that keeping track of and responding to different player skills improved learnability while using a simple technique to model the player skills.

In the SMES project, I claimed that re-shaping input allowed new ways of engaging with games while simplifying UI fidelity, the user experience and the number of games the system supported.

In Repurposing AI Surplus project, I claimed that game-playing AI methods can be used to assist our player and prototype several different instantiations of assistance to target different challenges while simplifying the level design, performance constraints and how the player is introduced to the assistance methods.

3.1.3 Reinforcement Learning Implementations: *Train and Evaluate*

While the computational caricatures allowed me to get a sense of the approaches, there is still a significant gap between a caricature and the reality it is trying to depict that should not be ignored.

In this dissertation, I set out to make games more inclusive. It was important to me that I could see whether the ideas scaled, at least in principle, to the tools that were used in the industry—that is if I couldn't make the theories work in practice as the primary torch carrier of the ideas, it was too naive of me to expect that others could implement and use them.

The technique of choice for my implementations was reinforcement learning [295]. At different points of my research I touched upon all the steps of the traditional reinforcement learning methodology such as adjusting policy model design while monitoring impacts on training metrics, reward shaping and reward redefinition while monitoring impacts on non-reward metrics to disambiguate changes that superficially improve learning from those that lead final policies that actually perform the best, perform ablations to identify which combinations of design features must be combined to yield which outcomes.

3.2 Problem Generation Approaches

I used the methods discussed above to answer my research questions and they served me well. However, these techniques were not very useful in deciding which questions to answer in the first place. I find that we spend a disproportionate amount time discussing how to answer questions without exploring how to find questions worthy of answering in the first place.

This section offers a list of techniques I myself utilized to guide my research as I progressed in my Ph.D., the grounding for which are my emotions and self-reflections. The following is not necessary to understand the rest of the dissertation but aims to give contextualizing insight into how my research questions emerged.

3.2.1 Dissect Concepts

I initially, and thankfully very briefly, thought the job of a Game Designer was to "minimize the failures" of a player. This belief was misguided [50], and even I myself gravitated towards rather challenging games and failure was an important facet of my play. Yet, I responded to some failures with invigorated enthusiasm, while others left me feeling better bitter and had me walking away. The concept of *failure* couldn't differentiate the *good* failures from the *bad* ones.

Dissecting the concept of failure to create a more discriminative and useful vocabulary lead to A Diagnostic Taxonomy of Failure in Videogames, which I expound on in the next chapter of this dissertation.

3.2.2 Follow Frustration

I grew up playing real-time strategy games. One of my favourite memories is my father excitedly showing me a newspaper article he had just read about how strategy games increased intellect. This was in 2003, when I was eight years old. All this is to say, when I started playing *Dawn of War III* [255] in 2017 I had more than a decade of controlling armies in real time strategy games experience under my belt.

In order complete the tutorial level of *Dawn of War*, I had to move the camera to a designated point of the map. The game told me how to accomplish this task through six different means: middle of the screen as the primary objective, top left of the screen as a reminder, bottom of the screen as an instruction, above the designated point as the target, bottom right of the screen via the minimap. The overbearing tutorialization made me feel infantilized. I got frustrated and stopped playing the game.

I switched to playing *Crusader Kings II* [25], another game I recently had purchased. It ended up being way too dense for me, and I couldn't figure out how to play it. The insufficient tutorialization made me feel helpless and stupid. I got frustrated and stopped playing the game.

Following this sense of frustration to find a solution lead to Talin: A Framework for Dynamic Tutorials Based on the Skill Atoms Theory, which I expound on in Chapter 5 of this dissertation.

3.2.3 Repurpose Techniques

I was finishing my undergraduate education when the 2013 paper *Playing Atari with Deep Reinforcement Learning* [212] was published. "*Game AI was solved!*" or so I thought at the time. I spent the next eight or so years being constantly impressed by how powerful reinforcement learning was becoming. These techniques were capable of beating *DotA* [307] and even *Starcraft II* [256] professionals!

Yet even though the number of e-sport athletes that AI agents could win against was increasing steadily, there was a stark consistency in how ineffective these agents were when it came to enhancing the gameplay experience of ordinary players. I imagined reinforcement learning as a great source of energy just waiting to be channeled in the right way. Even thought it didn't end up being the silver bullet my undergraduate self assumed it to be, this line of questioning ended up being productive.

In my paper "Repurpusing AI Surplus for Inclusivity" I focused on utilizing these gameplaying AI agents to serve the player experience. I expound on the main concept of this paper in Chapter 7 of this dissertation.

3.2.4 Reify Theories

"A Diagnostic Taxonomy of Failure" and "Repurposing AI Surplus for Inclusivity" are papers that contribute theories of what to do and how to do them. A rather straightforward way to find research questions is by seeing whether the theories proposed held water by operationalizing them through computational implementations. One might argue that this is no different than the Computational Caricature research method discussed in the previous section. However, I find the engineering challenges to be a salient difference that can, in their own rights, produce research questions.

My projects that focused on reinforcement learning powered assistance methods were a direct reification of the "Re-purposing AI Surplus" paper. Implementing a reinforcement learning powered assistance methods in a relatively realistic 3D game engine showed me several limitations of the system that would have remained unknown had I stayed in the computational caricature scale.

One example concerns the aesthetic style of reinforcement learning agents. Carrying out my implementation not in a simplified 2D grid world, but rather in a more realistic 3D environment showed me that a naive implementation of reinforcement learning agents will most likely not respect the aesthetic style of the given game. If interested a further exploration of this idea can be found in "Designer Centered Reinforcement Learning." [56]

In the following five chapters I will take a deep dive into the the individual projects mentioned in this chapter.

Chapter 4

A Diagnostic Taxonomy of Failure

Large portions of this chapter were published in the Foundations of Digital Games Conference 2020 under the paper title "*A Diagnostic Taxonomy of Failure for Videogames*" [50]. This paper has been co-authored by my advisor Adam M. Smith.

Failure in videogames is not desirable. A cursory look at several beloved videogames is enough to challenge the previous blanket statement. From popular cult classics such as the *Dark Souls* [55] franchise to the intentionally difficult and frustrating games such as *QWOP* [119] and *Getting Over It* [120], many videogames are known to explicitly invite failure. These games are enjoyed, even cherished, by many who play them [5, 35]. When players fail in these challenging games, the failures are seen not as a problem [273], but rather as a part of the game loop [31]. However, not every failure is met with such enthusiasm, nor is every type of failure healthy [264, 9]. If a player is failing because the on-screen text is too small for them to read, or because they do not know the controls then there is a problem—no one is excited to face these types of challenges. So, if some failures are welcome, whereas some are unwanted, and if some failures are opportunities to grow, whereas others make players feel helpless, how do we know, articulate, and communicate these differences? In this chapter, we contribute a diagnostic taxonomy to aid in answering these questions.

Players can fail in a variety of different ways while playing games. Failure can be part of a game and might even be expected to occur naturally in the span of a gameplay session. Failure might also be undesirable and necessitate a response from either the game itself, in the form of a helping action, or from the game designer, in the form of a design change. For example, in an analytical game such as chess, we expect most of the exciting challenge of the game to revolve around the failure to form effective plans (and counter plans or counter-counter plans). If a player is failing because they made an ineffectual plan, this is par for the course and does not require a response from neither the game system itself nor from the game designer. However, if the players are constantly failing because the chess pieces are hard to grasp and keep slipping from their hands then this asks for a better-designed chess set, as moving the chess pieces around should not be where the challenge of chess lies. In the former failure case, the player failed because they didn't plan their moves correctly, which matches the presumed design intent of chess. However, if the player fails at handling the chess pieces this requires an intervention as this type of failure (dropping the pieces) does not match the presumed design intent of chess.

Let us look at another example: Jenga [267]. Jenga is a game of physical skill in which players take turns removing one block at a time from a tower constructed of fifty four wooden blocks. Each block removed is then placed on top of the tower, creating a progressively taller and more unstable structure. In a game of Jenga, in addition to your strategy of deciding which blocks to move, your ability to physically manipulate the blocks also matters—maybe even more so! Contrary to chess, dropping a game piece while performing an action is acceptable because the failure in removing the block by dropping it–after having decided which block to move–matches what we imagine to be the design intent of Jenga.

In chess, after having made a plan (deciding which piece to move), the act of executing that plan should be seamless and should not pose any chance of failure, as it would detract from the play experience. Whereas in Jenga, after having made a plan (deciding which piece to remove), you can still fail at the act of executing that plan, and that failure would not be jarring to the overall experience at all, as it is part of the expected gameloop.

In our Jenga and chess examples, we outlined two similar types of failures (failures that occur when physically handling a game piece) and described how the consequences of those failures on the play experiences can be wildly different. This is why we need to have terms to clearly differentiate the types of failures we want from the types of failures we would rather avoid. In this chapter, we designate the failures that support the design intent of a game as *in-loop* failures (referring to the ideas of a game loop). The existence of *in-loop* failures is not detrimental to the enjoyment of the game, and in many cases, this type of failure is the main component that makes the game challenging and engaging. On the other hand, *out-of-loop* failures represents the unintentional (from the designer's side) failures that detract from the vision of the game and should therefore be minimized. To use our earlier example, dropping the chess piece would be an *out-of-loop* failure whereas dropping the Jenga block (and the tower with it!) would be an *in-loop* failure, even though they are both execution failures. In Jenga, physical manipulation is a part of the intended challenge, whereas in chess physical manipulation of the pieces is definitely not a part of the intended challenge.

In the context of videogames and this chapter, we use the term "failure" to describe the inability to make progress towards the objective (or goal) suggested by the game. It is important to note that just because the game is suggesting an objective, it does not mean the players are going to play the game to accomplish that goal. However, having this terminology will still help designers hone in on what the game is trying to accomplish and if appropriate (such as when discussing creative sand-box games or transgressive play [26]) can be extended to include player-set goals.

Currently, the videogame industry lacks any specific vocabulary to describe the differences pertaining to failures that occur in videogames. This gap makes it difficult to communicate which types of failures are desirable and which should be avoided, especially because communication is one of the problems game development teams often face [244]. This chapter proposes a taxonomy of failures to facilitate this differentiation and increase specificity in communication. If we have a vocabulary that allows us to describe how we, as game designers, are making our players fail, we can investigate whether these failure instances match our design intent. We propose this Taxonomy of Failure as a tool to help designers, critics, analysts, educators, and scholars be more precise in their discussions of failures in videogames.

We envision this taxonomy being used to:

1. describe what categories of failures are *in-loop* and which are *out-of-loop* in the games we design, and
2. classify different instances of failures to better understand which are problematic and should be reduced, and which are essential and should be fostered.

4.1 Failure in Games

Failure is an inherent component of games. In many cases, it is tailored to be a big part of the whole experience—to the extent that McGonigal [207] claims players spend most of their time failing. As such, the importance of failure has been a part of the discourse surrounding games from several different angles. Juul [174] investigates whether the presence of failure affects the enjoyment derived from playing videogames and concludes that players create elaborate theories of failure as a source of enjoyment. Further, in *The Art of Failure*, Juul [176] discusses the aesthetics of failure and argues that it is a crucial aspect of what makes a game a game. Building on this foundation by Juul, Roche describes the notion of "valid" failure[258], investigating the cases where the game recognizes, either from a gameplay or a narrative perspective, that the player has failed.

Another concept tightly related to failure is challenge. Adams, in *Fundamentals of Game Design* provides two broad categories of challenges: cognitive and physical [31]. Thinking about what the challenge facing the player should be is another way to conceptualize and differentiate *in-loop* from *out-of-loop* failures. However, it is important to note that these gameplay categorizations of challenge still struggle with measuring difficulty [33] and does not cover all the different ways a player can fail. Shifting our attention from challenge to failure allows us to engage our users on the users' terms, by exploring how the challenges we design are received by our players.

In *Uncertainty in Games*, Costikyan [91] discusses another aspect crucial to failure: uncertainty. This discussion provides another perspective to describe *in-loop* and *out-of-loop* failures. If the player expects high uncertainty in their actions, blunders resulting from those specific moves are *in-loop* failures because the player acknowledges the likelihood of failing. *Out-of-loop* failures are those in which the player does not expect uncertainty. They do not foresee a potential of failure, and they are jarred if failure does occur. This lens can be another way to reason about how *in-loop* and *out-of-loop* failures affect the play experience.

Another angle to investigate failure is the aspect of learning and mastery. Gee [136] describes good videogames as learning machines and explains that failing and trying again makes failure central to learning and a part of the fun. Similarly, in *The Art of Computer Game Design* [94] Crawford, and in *A Theory of Fun for Game Design* [189], Koster, argues that fun, learning, and failing are intricately connected. In their work exploring failures and breakthroughs [163] Iacovides et al. discuss relationships between learning and in game involvement. Learning and failure have been connected to one another in other domains as well. Kapur introduces the concept of productive failures [181] and shows the effectiveness and how to design for them [182].

Productive failures are the notion of allowing people to fail while attempting to solve complex problems. This kind of failure facilitates the understanding of the problem at hand. It is easy to see the similarities between the good game design argued by Gee and Koster, and the notion of productive failures introduced by Kapur. Anderson digs into this connection by investigating how failure in games promotes learning [39], and Kllevig investigates whether the game can play a role in changing a person's relationship to failure [178]. Desirable difficulty [63] is another theory that puts emphasis on understanding how people fail and —through the failure and difficulty— learn. Desirable difficulty is concerned with matching the learner with the correct difficulty and challenge. In all these cases having a more precise language to specify and identify different types of failures would help the clarity and communication of the ideas.

While the idea of learning is deemed to be important when it comes to enjoying games, the tutorial levels meant to teach players how to play usually leave some things to be desired—even though they have a significant effect on the rest of the game [38]. It is common for players to complain about the overprotective and cautious nature of some in-game tutorials, especially complaining about how the tutorial robs the player of a sense of discovery [250, 313] There has been some work in creating context-aware tutorials [45] and adjusting the game difficulty to fit the player [158] but there is still room for improvement. A taxonomy of failure can allow designers to decide when to provide tutorials and where to let the player explore, which is essential in allowing games to keep the sense of discovery and learning.

Another important benefit of being able to specify different types of failure is related to accessibility in games. There have been improvements when it comes to game accessibility [126] and there are many resources that educate game designers on how to make games more accessible such as the extensive *Game Accessibility Guidelines* [9]. There are still, however, many open research questions in this area. Being able to classify failures will surface cases where the players are failing due to accessibility issues. Classifying failures can also contribute to making games more accessible. Another use for the taxonomy is Universal Design [290], a design approach that prioritizes removing barriers to engagement with experiences. A taxonomy of failure can allow the designers to be more precise in deciding which failures act as barriers to enjoyment (and so should be removed) and which failures are essential for the game experiences (and so should be preserved).

4.2 Taxonomy of Failure

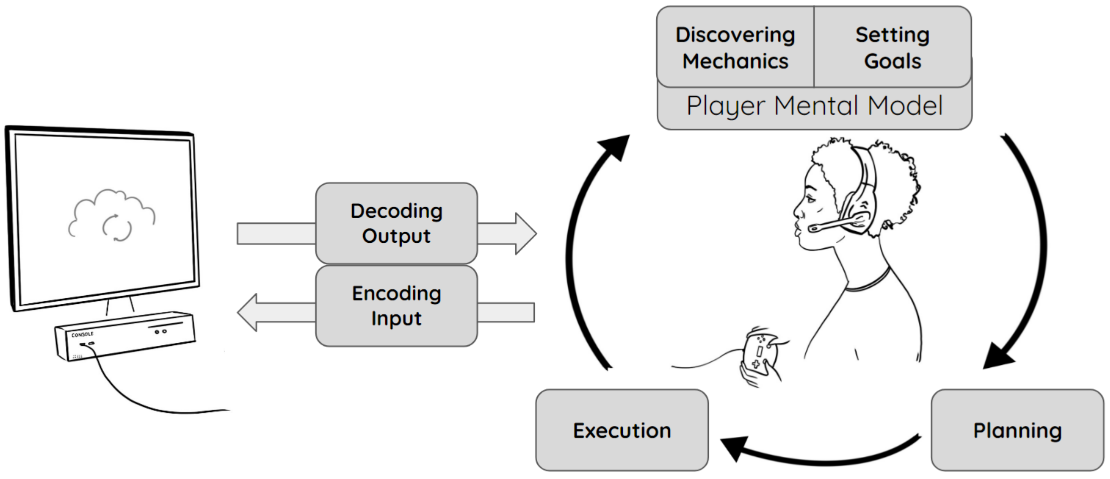


Figure 4.1: The proposed model of how people play games, specialized for delineating different types failure.

In order to classify where people can fail while playing videogames we propose a specific model of how people play videogames (shown in Fig. 8.1). Our model is inspired by and builds upon Swink's feedback loops [296], Sellers interactive loop [272] and Don Norman's conceptual models [231], and is adjusted to make delineating failure classes more straightforward.

Building upon the aforementioned work, we claim that players have a model of the game even before interacting with the game, created by their cultural knowledge, previous game experiences, and the aesthetic and representational aspects that the game is presenting [159]. The model consists of players understanding of the mechanics of the game (what they *can do* in the game) and the goals communicated by the game (what they *should do* in the game). Players then use this model to form a plan that can be arbitrarily short or long, in an attempt to achieve the goals they have set for themselves. Finally, the players attempt to execute this plan. Regardless of whether the plan is successful or not, the feedback from the game allows the player to update their mental model of the game and form a new plan.

However, in order to even engage in this model–plan–execute–update loop, the players need to be able to encode their inputs to the system and decode the outputs that are coming out of the game. Encoding input refers to the ability of the player to engage with the game on the most elementary level, whether it is through pressing the correct buttons, knowing which gestures to make on a touch screen, or being able to voice the correct verbal commands, etc. Decoding output in this context refers to the ability of the player to parse and understand the feedback of the game, whether it be visual, aural, tactile or otherwise.

The steps described in our model of how a player engages with a videogame also represent the different classes of failures—points in our model where a failure can occur.

4.2.1 Encoding Input → What physical action allows me to act in the game?

Before one can truly start playing a game they need to be able to interact with the game system itself. This class refers to the players' knowledge of the control scheme and the physical capability of using it. Is the player aware of the correct buttons to press, the right gestures to make, or in general, the valid set of actions they can take to interact with the system? More importantly, is the player physically capable of taking those actions?

In-loop encoding input failure can arise in puzzle boxes or escape rooms, where the challenge lies in actually figuring out the specific actions to get a response from the system or the correct physical components to push and pull.

It is much more common, however, for failures in the encoding input class to be *out-of-loop*. A simple example would be the player not knowing which buttons used to play the game. A more serious encoding input failure is when a player can not use a functionality due to a physical limitation. A recent example is when a player with a physical disability loses the ability to play *Civilization* due to a change in the user interface [13].

4.2.2 Decoding Output → Can I parse the feedback of the game?

In addition to the ability of conveying input to the game one must also understand what the game is saying in return. Decoding output refers to the player's ability to understand the information presented by the game, whether it be visual, aural, tactile or through any other medium that the game is using.

Games that use decoding output as an *in-loop* class would be spot-the-difference or hidden object games where the challenge lies in parsing a generally complicated scene, a game where the player is trying to find differences between two images or games where part of the challenge lies in overcoming the chaotic visuals[75]. Any game where the challenge lies purely in perception can be considered to have decoding output as an *in-loop* class.

Many *out-of-loop* examples in the decoding output class fall under accessibility issues: If the player can not differentiate enemies from allies due to their color-blindness [138] and the game's color scheme this is a decoding output failure. Further examples would include not being able to read text on screen because of it being too small [125], or not being able to understand the natural language that conveys crucial information about the game.

Encoding Input and **Decoding Output** groups cover the fundamental requirements for any player to give input to the system and receive clear feedback as an output. Only after a baseline proficiency in these two classes are established can the player start engaging with the game productively. It is important to note that many failures in these two classes stem from accessibility issues and it is crucial for game designers to be aware of these cases.

4.2.3 Discovering Mechanics → Do I know what I can do in the game?

This class of failure refers to the player's understanding of the game's mechanics and the set of available verbs offered to the player. In other words, does the player know the capabilities of the tools that they are given?

If the game is about figuring out how different mechanics relate to one another and finding creative ways of solving problems through emergent methods, that game is using discovering mechanics as an *in-loop*-failure class. The joy in playing the game comes from learning what the tools are, making the discovery of what the mechanics are is part of the fun.

On the other hand if the players progression is halted because they could not figure out how a specific lock-and-key style mechanic functions—whether it be not realizing that bombs can blow open cracked walls, or the specifics of how levers and pressure plates work in many adventure games (and the challenge lies in figuring out where you should place the bombs and the order of pulling the levers, instead of figuring out whether you can do those things)—then it is an *out-of-loop* failure in the discovering mechanics class. Another example would be the player repeatedly getting defeated by a specific type of enemy because they didn't realize that a specific tool in their toolkit deals more damage to that type of target. If the joy in the game is purely about mastery over the mechanics, and not in uncovering what the mechanics are, then discovering mechanics is an *out-of-loop* failure.

To be more concrete, in rhythm based games such as *Guitar Hero* [149] or *Tap Tap Revenge* [297], the available set of verbs is simple and should be well communicated, and the joy in the game emerges from getting better at those verbs, not figuring out what the verbs are. Whereas in *Legend of Zelda, Breath of the Wild* [229] understanding how different systems of weather, conductivity, and temperature relate to another would make discovering mechanics an *in-loop* class for that game.

This topic also comes up in design discussions that happen in the game industry. In the Noclip Podcast the game director of *Doom Eternal*[165], Hugo Martin, describes why deciding whether to make game mechanics easy to discover or not is important[2]:

> [*Doom Eternal gets really*] heavy-handed on the player education stuff, because we do not want to make learning [mechanics] part of the experience. As you know in *Sekiro*, a lot of the [mechanics] are there to be discovered and that's a huge part of the experience. For us, I want to make it clear as day. The fun of the game is not figuring [mechanics] out, the fun of the game is mastering it.

4.2.4 Setting Goals → Do I know what I should accomplish in the game?

Any moment a confused player asks "What am I trying to do here?" is a failure instance belonging to the setting goals class. The setting the goal class refers to the moments when the player does not know what to do to make effective progress in the game, or what progress even means. In other words, as long as the player is clear on what their goal is, the failure is not a setting goal failure.

In many open world games figuring out where to go next and who to talk to in order to move the story further is a part of the challenge and enjoyment. In these cases it is acceptable for the player to not know exactly what the proper next step is. The challenge and fun comes from discovering what comes next as well as exploring the space of possible goals. In other games, there are several ways to achieve victory, from conquering the world by force to becoming a diplomatic mastermind. In these games deciding which objective to pursue, and sometimes prioritizing, is fun and, as such, setting goals is an *in-loop* failure class as well.

In many games however the designers want their players to always know what their next objective is. If at any point the player is questioning what they should do next and if this questioning is not a part of the game's design, then failures stemming from this confusion are *out-of-loop* setting goal failures. This is a prominent issue with confusing quest texts and cases where a non-player character shares what the player should do next without recording that information anywhere permanent.

When the player does not know where to go and what to do next in open world survival games, such as *Subnautica* [306], it is not a problem, since the design of the game affords wandering around and setting goals through the experience of spending time in the space. Whereas if a player does not know that the goal of *Super Meat Boy* [299], an action-packed platformer, is to reach the end of the level, then this is an *out-of-loop* failure! In *Super Meat Boy* the goal should be abundantly clear and the challenge should not come from the player understanding what the goal of the level is, but rather, the process to reach that goal.

The **Discovering Mechanics** and **Setting Goals** classes together represent a large portion of the mental model that the player uses to play the game. Players create this model before even playing the game, and through the experience of playing the game, this model is constantly updated. The player continually learns and improves their understanding of what is possible in the game and what they should do next, by forming plans, attempting to execute those plans, and updating their model based on the result of their actions.

4.2.5 Planning → What steps should I take to accomplish my goals?

Planning is the player using their mental model of the game to decide what they will do next, given the goals they have and their awareness of the game's mechanics. A plan might span just the next few actions if the game is a twitchy reflex game, or a plan might be an elaborate battle strategy in a complex turn-based grand strategy game. Failures that fall into this class include the failures that arise from making an ineffectual plan and the failures at simply making a plan.

In many slow-paced turn-based games most of the fun arises in the plans the player generates: Which piece to play in chess, which position to move a soldier next in a turn-based tactics games or which route to take to the next level in a strategy game. Any failure that arises from making the wrong plan would be an *in-loop* planning failure.

If a game is designed to be purely about fast reaction times, or repeated executions and the player finds themselves making longer term plans (and failing at them) this would be an example of an *out-of-loop* failure. To be more concrete, imagine a game where the designer's intent is to have levels that can be progressed with any loadout that the player chooses. If there exists a scenario that a player is left unable to progress due to picking the wrong loadout (a long term plan they made at the start of the level) then this does not match the outlined design intent of the game. This hypothetical scenario is an example of how the Taxonomy of Failure could be used to validate and maintain the design intentions of a game while designing its experience.

4.2.6 Execution → Was I successful in following the steps in my plan?

Execution is the ability to implement the plan formed by the player. There is a failure in this class whenever a player flops and does not succeed in something they had planned to do.

Any game where dexterity is intended to play a role most likely has execution as an *in-loop* failure class. Whenever the player misses the next platform by a pixel because they jumped one moment too late, or they could not play the correct notes to the beat, or they parry one moment to soon and lose their last life, they are experiencing an *in-loop* execution failure.

If the game is not testing the players ability to execute those plans and instead tests the formation of plans, any failure in execution would be an *out-of-loop* failure. If a player miss-clicks and sends their armies to the wrong tile on a hex-based strategy game, if a player clicks on the wrong verb in an interactive narrative or if a player is forced to play a chess piece touched by accident due to the touch-move rule and as a result loses the game, they suffered an *out-of-loop* execution failure.

It is important to discuss the difference between failures in **execution** versus failures in **encoding input** as they might seem similar. The encoding input class refers to the players knowledge about the means in which they can act in the game. Does the player know what actions to take to act in the game and are they generally physically capable of taking those actions? Whereas Execution refers to their moment to moment success in taking those actions in the correct time and order. The ability to press a button does not directly correlate to one's accuracy of pressing it at the right time - an example of execution failure. However if you have a physical limitation that prohibits you from comfortably pressing the button at all then that is an encoding input failure.

In this section we have described the six categories of failure that exist in the Taxonomy of Failure:

- **Encoding Input**: What physical action will allow me to act in the game?
- **Decoding Output**: Can I parse the feedback of the game?
- **Discovering Mechanics**: Do I know what I can do in the game?
- **Setting Goals**: Do I know what I should accomplish in the game?
- **Planning**: What steps should I take to accomplish my goals?
- **Execution**: Was I successful in taking the steps in my plan?

4.3 Classifying Games

Why is this classification useful? One way we can use the Taxonomy of Failure is to classify games by designating which of the classes of failures are *in-loop* and which classes of failures are *out-of-loop*. This classification allows us to look at games we are interested in through another perspective, complementing other ways of understanding games.

If we are in the process of designing a videogame then we can use the taxonomy to act as a map in describing where we want the challenge to exist. This description can be helpful in evaluating the design decisions we come across as the development process continues by ensuring failures happen in the *in-loop* classes, and failures in the *out-of-loop* classes are minimized.

Additionally, the Taxonomy of Failure might prove useful when we are trying to understand games that already exist. After classifying the games that interest us, we can look at how these games actually guide their players away from *out-of-loop* failures and towards *in-loop* failures.

Altough we believe the taxonomy to be useful, it is important to be aware of the limitations. An overemphasis on using the taxonomy during the initial phases of design might pigeon hole the game into a certain idea, and might cause interesting insights to slip by. It is important to be flexible and try to truly understand what makes the game fun! If you observe that your players are enjoying failing in categories you designated as *out-of-loop* failures then it might be valuable to reevaluate your initial classifications.

4.3.1 Super Mario Bros

Let us look at this process through a few examples. We start with *Super Mario Bros* [228] (referred to as *SMB* from here on out). We claim *SMB* is a game about mastering how to move in a world filled with spikes, piranha plants and hazardous landscapes. Given this observation, the *in-loop* classes are **planning** and **execution** and any failure outside of those two classes are *out-of-loop* failures.

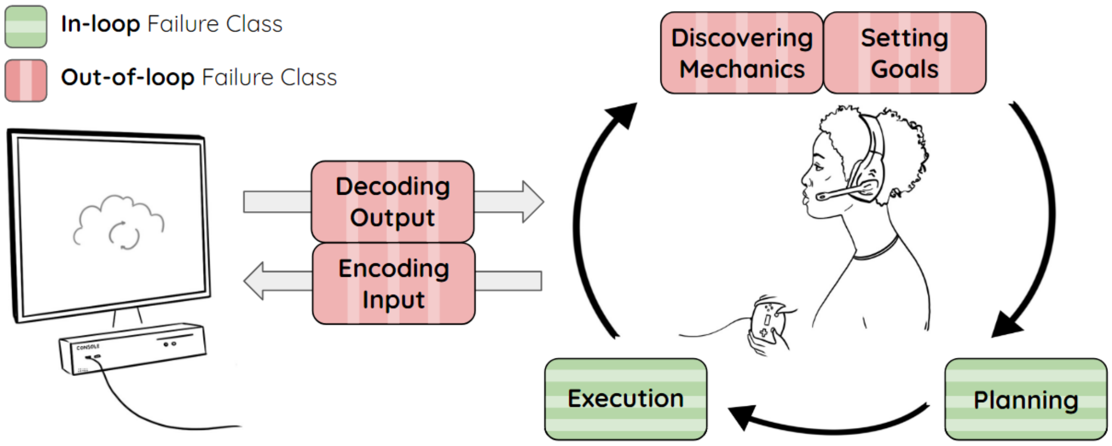


Figure 4.2: The proposed in-loop (green) and *out-of-loop* (red) failure classes of Super Mario Bros.

We can start by digging into the *in-loop* failure classes of SMB. What is the main challenge in *SMB*? First, the player needs to decide which path to take through the obstacles. Is the player going to jump on the goombas, or will they try to dodge them by running on the platforms above? There are many paths in the game that are viable, but there are also many which lead to certain doom. If the player takes a leap of faith and, after falling into the pit, realizes that jumping is not possible, it is not a big problem. They will just take the other route the next time. They conducted a plan to make the jump and executed the plan but the plan was flawed. Thus the player failed, but this sequence was not problematic because **planning** in *SMB* is an *in-loop* failure class. (shown in Fig. 4.2)

After making a plan in SMB, the player actually has to be able to implement that plan. It is one thing to think, "I am going to jump over the goomba, dodge the fireball, and vault over the piranha plant" and another to actually be able to do it! Either way though, if the player is not successful in their execution, they can always try again and no player is expected to successfully execute their plans on their first try, or with have a 100% success rate. When Mario dies there is a relatively short respawn time, and this supports our analysis that players are indeed expected to die several times in each gameplay session. These factors make **execution** also an *in-loop* failure class.

Now, we can also look at the *out-of-loop* classes and see how games attempt to steer the players from failing at those classes. Let us start by investigating the **encoding input** class. When we actually look at the hardware design, we see that the Nintendo Entertainment System controller (Fig. 4.3) has very few buttons, so just by looking at the controller the player can gauge how to give input to the system. Consider playing *SMB* on an emulator on a modern computer. If the player does not know which of the myriad keyboard buttons actually map to the initial NES controller buttons, then they can spend several minutes trying to figure out what hardware buttons to press, that would be a good example of an **encoding input** class of failure.

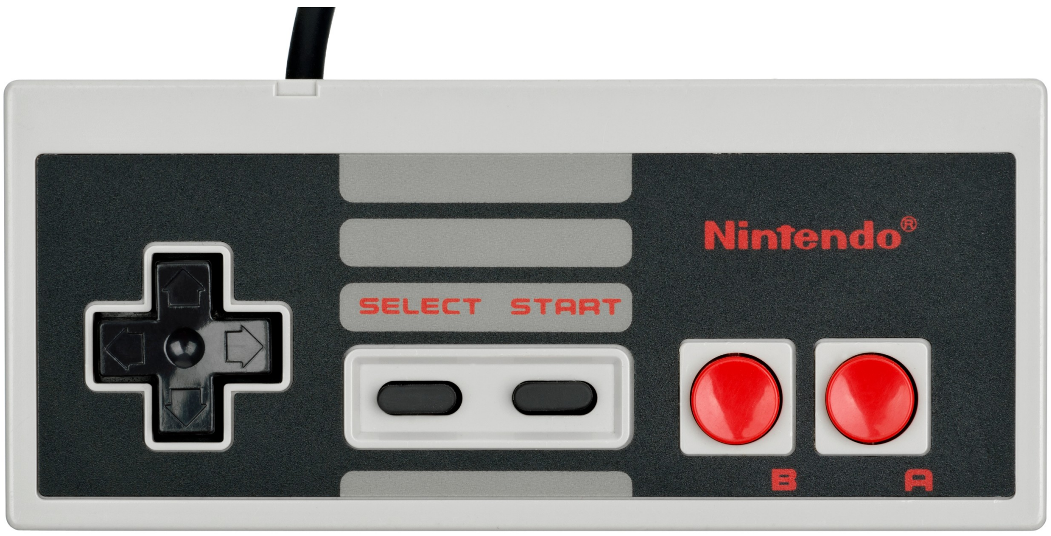


Figure 4.3: The NES controller with no redundancy and a clear way to engage each button. Image from public domain Wikipedia, owned by Evan-Amos.

When it comes to **decoding output**, most *Mario* levels are not about trying to figure out where the obstacles are; rather, they are about avoiding them. Whether it be because of the color palette limitations of the day, or by intentional design (and realistically, a combination of both factors), obstacles in *Mario* are mostly easily discernable, and in many cases deaths are not caused because the player does not see the enemies coming. Even if that ends up being the case, the non-random nature of levels indicates that the designers were comfortable with the players knowing where the hazards were before trying to navigate through them.

In updating the player's mental model, the first level of *Mario* (Fig 4.4) does a lot of heavy lifting. The way *Mario* is positioned when the game initially starts, and how the camera moves in relation to *Mario* all support the idea that *Mario* should go towards the right. Furthermore, the first few minutes of the game, with the goomba

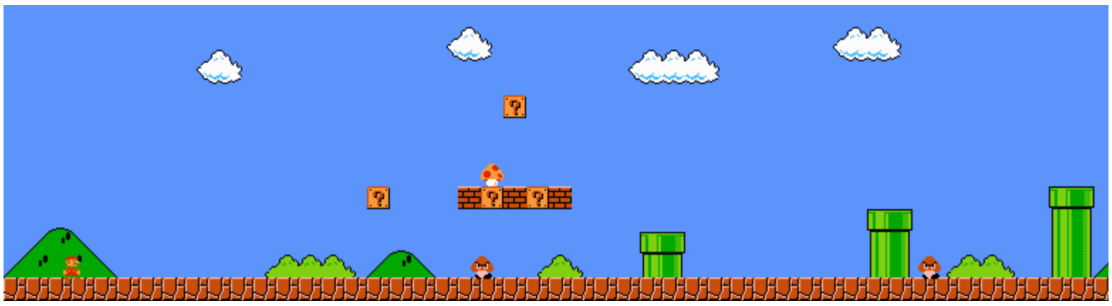


Figure 4.4: The starting area of the first level (1-1) of *Super Mario Bros.* Image taken from Video Game Level Corpus [293].

appearing (and not letting the player continue onwards until they demonstrate they can jump), the specific placement of the question mark tiles with power-ups, and the varying height pipes, teach most if not all the mechanics the player has to know to be effective at the game. If needed, a more detailed breakdown of *Mario* 1-1 can be found elsewhere [113]. Just within minutes the player knows what they can do and what they should do in the game respectively ensuring that no one is failing in **discovering mechanics** or **setting goals**. Miyamoto himself says, "So even within that one section we knew that the player would understand the general concept of what *Mario* is supposed to be and what the game is about" [112]. This gradual learning curve that starts at *Mario* 1-1 and continues throughout the game because *Mario* is not a game about the player exploring how complicated and varied systems interact with each other or figuring out where to go; rather it is about making a plan and executing it.

One might argue that when a speedrunner is trying to finish *Mario* in under five minutes, they are effectively exploring the ways of going faster and, as such, uncovering the hidden (and in this case potentially unintentional) mechanics becomes a part of the fun, making **discovering mechanics** an *in-loop* failure class. This argument goes to show that these types of generalizations using the framework run the risk of assuming the entire game audience engages with the game in an identical way. When using the taxonomy, care must be given to understanding different styles of games that any game can support, which might create several different *in-loop* and *out-of-loop* failure classes for each play style and player segment.

4.3.2 Sid Meier's Civilization

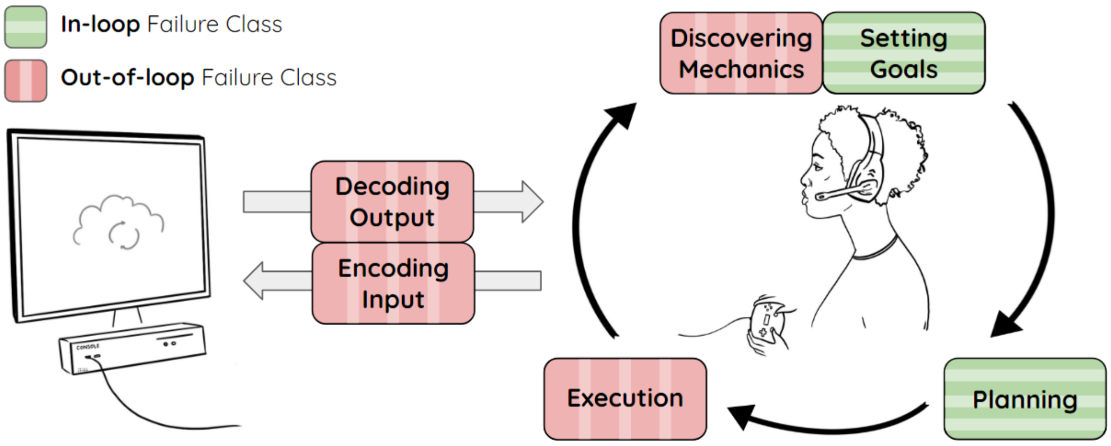


Figure 4.5: The proposed *in-loop* (green) and *out-of-loop* (red) failure classes of Civilization games.

Another game we can look at is *Sid Meier's Civilization* (shown in Fig 4.5) series [117]. Similar to *SMB* both **encoding input** and **decoding output** are *out-of-loop* failure classes. In *Civilization* figuring out which buttons to press and visually identifying what is going on the screen should not be challenging.

When it comes to **discovering mechanics**, most *Civilization*-like games, we claim, expect you to learn the rules of the system and then try to be successful within those bounds. The joy does not come from exploring whether hammers help you build armies faster, or whether apples allow you to grow your cities— these are elementary building blocks of the game that the player is expected to utilize. If the player has difficulty grasping these concepts there is a problem and the game has aggressive in game advisors to ensure that this rarely happens. Thus **discovering mechanics** for *Civilization* is an *out-of-loop* failure class.

The main bulk of the challenge comes from the player making both long and short term plans to decide how to move their units, what buildings to construct, how to conduct diplomacy and what policies to enact. If the player loses the game, it is expected that this is due to making a series of bad decision, which means that this style of loss is within the gameloop. Thus planning is an *in-loop* failure class for the *Civilization* series.

Following from this is the **execution** class. Unlike *SMB*, after having completed the plan, there should be no difficulty in actually implementing that plan. If the player has decided to build a granary, pressing the buttons to do so should be simple. Similarly if the player has decided to invade the neighboring city giving the order should be trivial. If the player fails to finish the granary because the city falls to enemy hands, or the invading army is defeated by the city's defenders, these are not execution failures—the orders were given and executed properly. Rather, they were planning failures, since the plans were not formed effectively which, as discussed above is not problematic. Therefore, this goes to show that in *Civilization* **execution** is definitely an *out-of-loop* failure class.

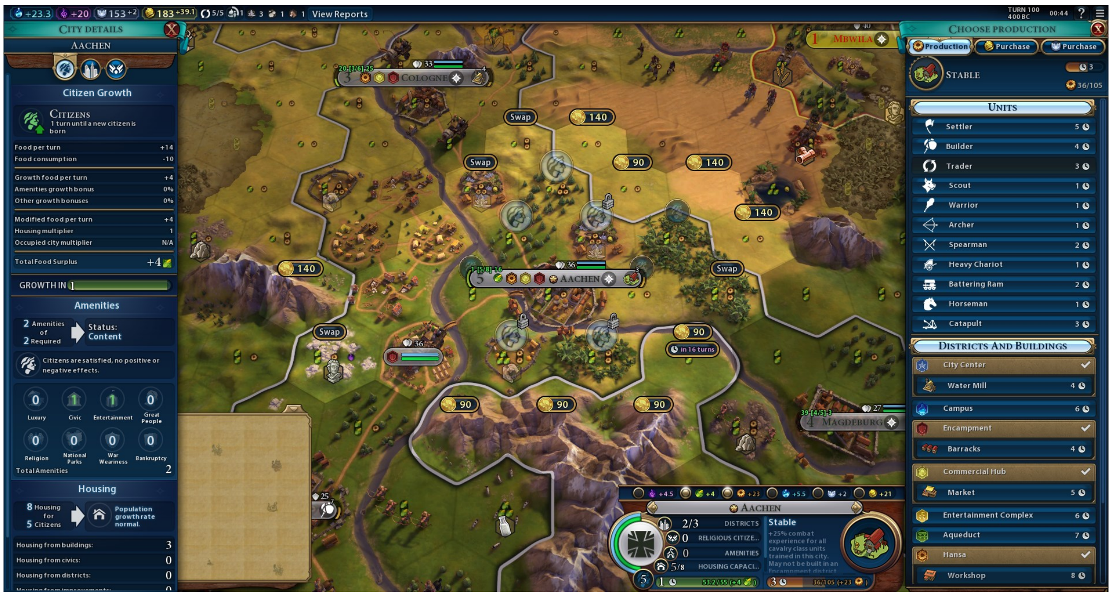


Figure 4.6: The screenshot displaying a gameplay scene of the turn based strategy game *Civilization VI* [117]

Different from *SMB* however, is the **setting goals** class. In *SMB*, the goal is to reach the right side of the screen and that is closely attached to victory. However in *Civilization* there are several ways of achieving victory. You can conquer all the capital cities of your enemies, you can talk your way into becoming a world leader or you can win the space race. Deciding on the victory conditions of the player's goal is a part of the fun. If the player fails at a given playthrough because they did not alighn their goal with the strengths of their nation this is not a problem, because they can learn from their mistakes and try again. As such, setting goals is an *in-loop* failure class for *Civilization*, and not an *in-loop* class for *Super Mario* where the only option is to go right (ignoring transgressive methods of engaging with the game).

4.3.3 Legend of Zelda: Breath of the Wild

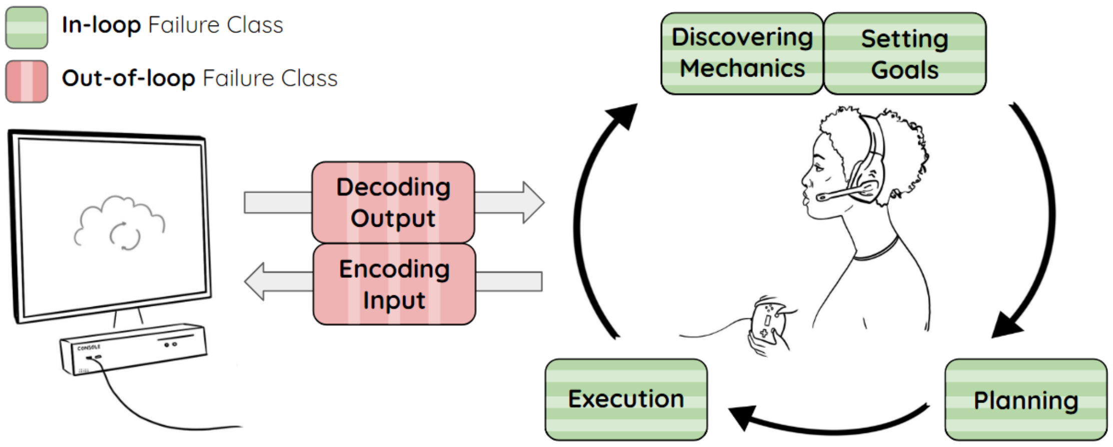


Figure 4.7: The proposed *in-loop* (green) and *out-of-loop* (red) failure classes of *The Legend of Zelda: Breath of the Wild.*

Similar to our earlier examples **encoding input** and **decoding output** are out-of-loop failure classes in *Legend of Zelda: Breath of The Wild* [229] (referred to *BoTW* from here on out). This game is not about figuring out which buttons to press or parsing the screen; rather, it is about, deciding what to do in an open-world, discovering how the game systems interact with one another and fighting different types of enemy characters.

Unlike the previously mentioned games however, a big portion of the fun when it comes to playing *BoTW* comes from gradually learning how the different systems of the game interact with one another. Initially, the player can figure out that wooden weapons catch on fire when held close to a firepit, which can lead to the realization that the shrubbery too is flammable. This understanding culminates in the player figuring out that the fire creates an air current they can hijack to gain some altitude using their paraglider (shown in Fig.4.8). Sequences like these in which the player discovers how primitive pieces interact and create emergent experiences contribute to what makes playing *BoTW* engaging. If the player fails because they perished in the forest fire they just created—showing that they failed because they didn't understand the mechanic previously– this is not a (design) problem. Nor it is a problem if the player has their progress halted while they play around discovering what tools they have to solve the puzzle they just encountered. Thus discovering mechanics is an *in-loop* class for BotW.

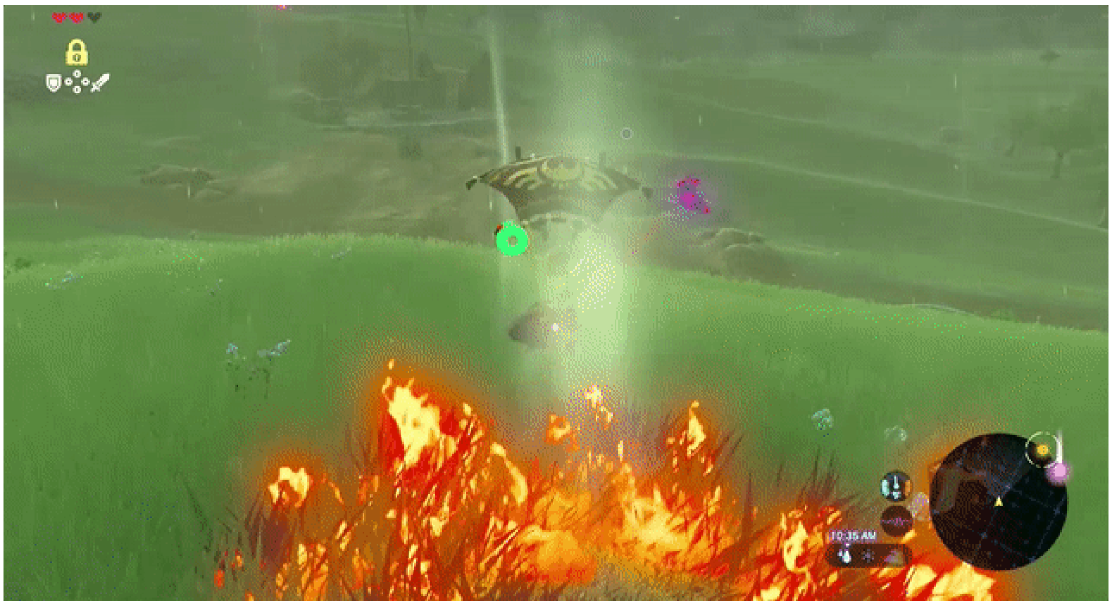


Figure 4.8: The screenshot displaying Link from Breath of The Wild [229] using the updraft created by fire to glide upwards.

When it comes to **setting goals**, most open world games have an *in-loop* class and *BoTW* is no exception. The player is given the overarching victory condition of defeating Ganon in his castle. The player can set his sights on that goal directly, or the player can decide to have their goal be becoming more powerful by recruiting other characters or finishing different storylines—all these goals are supported by the base game. If the player fails to defeat the boss because they set their sights on defeating him too early before accomplishing other goals this is not game breaking failure. The player is allowed to change and readjust their goals and try again later.

Planning and **execution** are also *in-loop* failures as with many other action games. When the player is in combat or climbing a rocky mountain, they first need to plot the actions they want to take and then successfully execute those actions. If they fail because they made the wrong plan—miscalculating how much stamina they would need to climb the hill and slipping down—or because they didn't execute their plan correctly—missing their bow shot and failing to take down the enemy from afar, thereby becoming overwhelmed—the play experience is not hampered and the player can respawn and try again, updating their model with their newfound experience.

4.3.4 Where's Waldo?

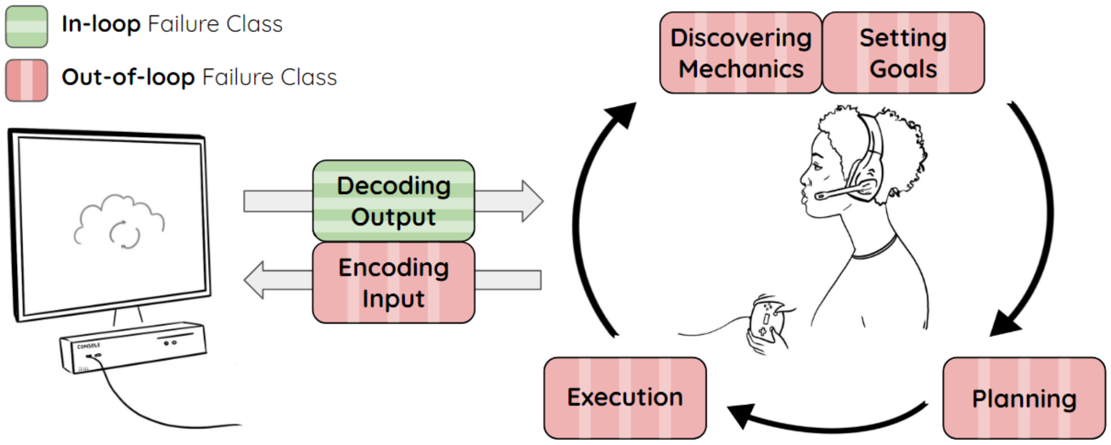


Figure 4.9: The proposed *in-loop* (green) and *out-of-loop* (red) failure classes of *Where's Waldo? The Fantastic Journey.*

We would like to conclude this section with an unorthodox game, *Where's Waldo? The Fantastic Journey* [201] (seen in Fig. 4.10). Although, we argued that in most cases **encoding input** and **decoding output** is an *out-of-loop* failure class, this does not always have to be the case. The only manner in which a player should fail while playing *Where's Waldo* should be in finding Waldo, and as such **decoding output** is an *in-loop* failure class in this case. (Fig. 4.9) The player should know what their goals are, what they could do and which buttons they should press. After spotting Waldo, the rest should pose no difficulty, and as such, all other classes for this game are *out-of-loop* failure classes.

4.4 Classifying Instances of Failures

So far, we have been classifying games, but another use case of the Taxonomy of Failure is to classify instances of failure. We defined failure as "inability to make progress towards the objective suggested by the game" earlier in the chapter. For many games this might be an explicit "Game Over" screen with a call for the player to try again, but for others it might take the form of a temporary setback. What exactly is going to be classified as an "instance of failure" will once again depend on the specifics of the game and the context in which the taxonomy is being used.

We would go through the trouble of classifying how our players are failing if this process would help us identify whether a design change is necessary or not. If we keep observing that the players are failing in a specific *out-of-loop* failure category, we

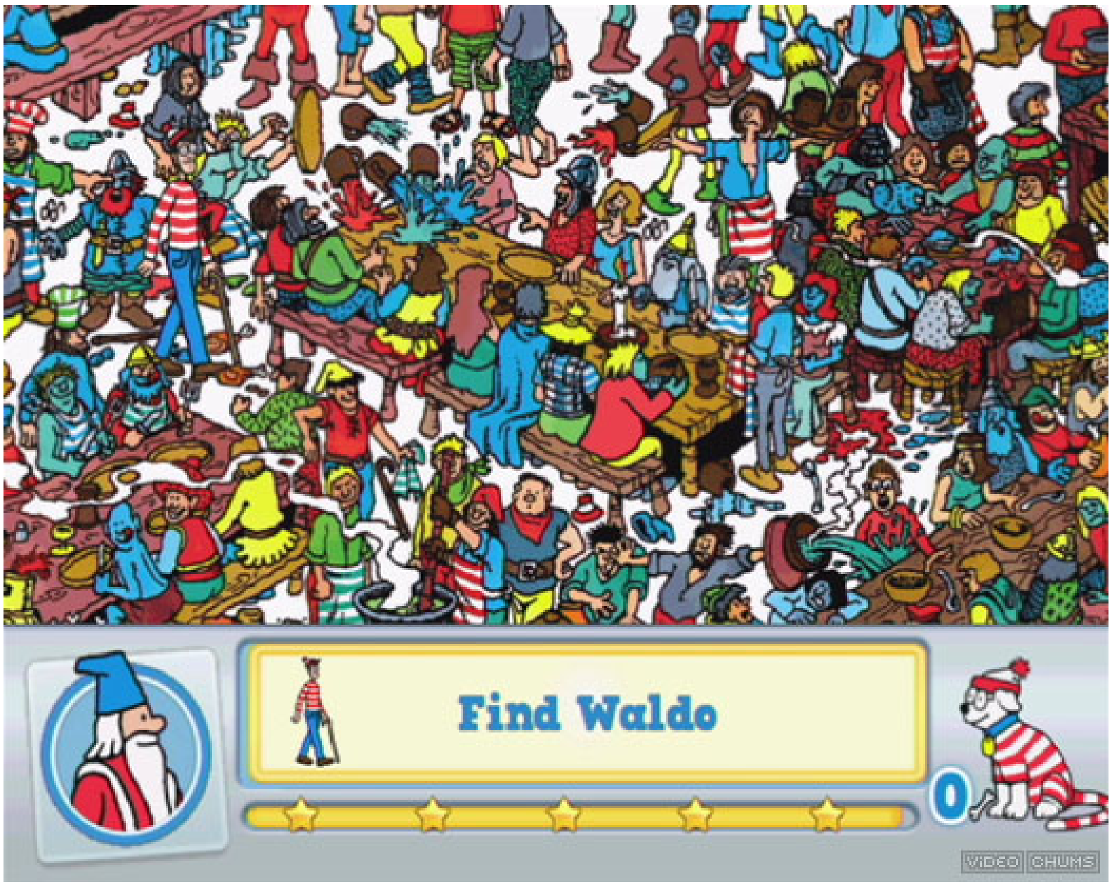


Figure 4.10: A gameplay scene from *Where's Waldo? The Fantastic Journey* [201]. Can you spot Waldo?

can use this to inform our decisions as to what changes to make.

Another capability that is of interest to both industry [61] and researchers is player modeling [315]. In most approaches to player modeling the player failures are not differentiated, if represented at all [154]. Given that player response to different types of failures vary, we expect discriminating different types of failure to be useful for downstream applications.

But is the granularity of the Taxonomy of Failure enough to be a useful tool in the design process? One might argue that there can be cases where two failures of the same class might have opposing impacts on the player; even though they are in the same class, one failure might be *in-loop* and another might be out-of-loop.

A simple example of this is repeated failures. The first time the player misses a difficult jump is fine. Perhaps the first five times it is fine, maybe the first fifty. But for many games there becomes a point in which even an *in-loop* failure can become frustrating and detrimental to the intended play experience. In those cases, the first few failures of the execution class are possibly *in-loop*, and the latter cases are *out-of-loop*.

Another example of this comes in the form of specific mechanical interactions. We argued that in many cases failures in decoding input are *out-of-loop* failures related to accessibility. However, consider the flashbangs in many first-person shooters or the screen obscuring attacks in *Super Smash Bros* [227] and other party games. Suddenly, if a player fails as they could not decode the screen because it was visually washed out, it is once again acceptable.

To account for these type of particular cases, in practice, it is valuable to extend the Taxonomy of Failure by considering instances of failures specific to the game one is working on. Going back to our earlier example, in *Super Smash Bros.* **decoding output** might be deemed an *out-of-loop* failure class, but underneath the decoding output class, the specific instance of "failure due to screen obstruction mechanic" would be designated as an *in-loop* failure instance. The provided Taxonomy of Failure does not come with this level of specific instances by default— we believe that for most games, many of the provided instances would be invalid, and it is better to construct this layer for each game individually. As such, we suggest expanding the taxonomy for the game one is interested in more deeply exploring, by describing the concrete ways how a player can fail in that game. This exercise allows the designers to be more specific when needed, and we also believe that the sole act of outlining the possible failure cases in a game one is designing to be a fruitful activity. Examples of specific failure instances can be seen in Fig. 4.11.

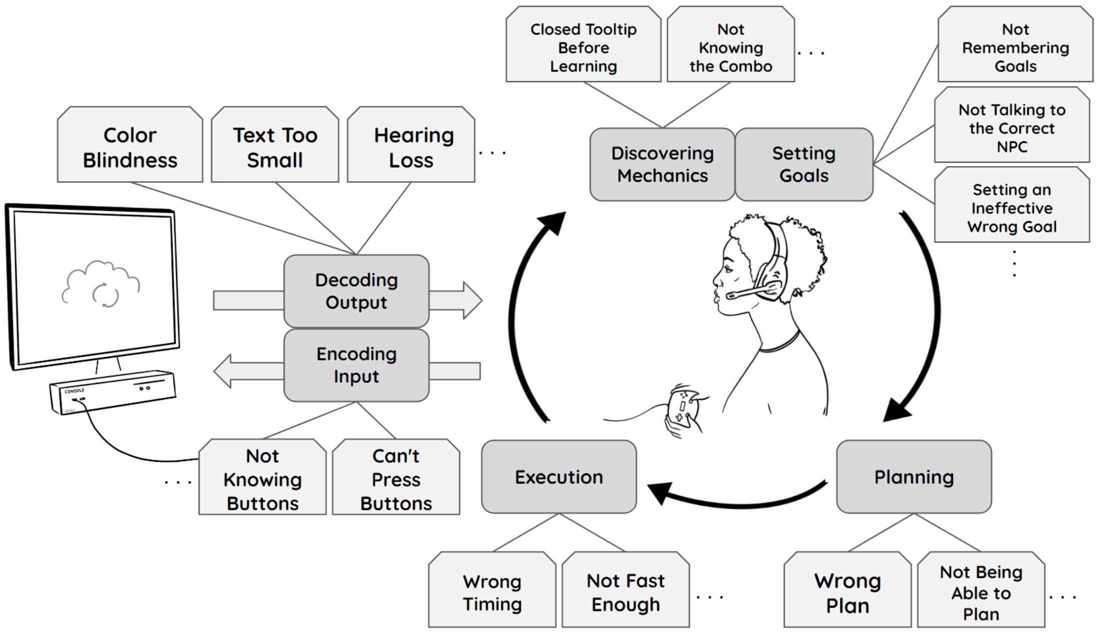


Figure 4.11: The extended Taxonomy of Failure with a series of potential extensions shown.

When it comes to using the Taxonomy of Failure in practice, one cannot simply halt the playtest and ask why the player failed after a loss of health points. While it is possible for the designer of the game to guess what is happening, that would still be a guess. A possible way to alleviate this problem is to record the play session and if time permits, go over the failures with the player and work with them to annotate the failures.

Conclusion

In this chapter I described why it is important to delineate between different types of failures and we offered the Taxonomy of Failure with the six classes of failures to facilitate doing so:

- **Encoding Input**: What physical action will allow me to act in the game?
- **Decoding Output**: Can I parse the feedback of the game?
- **Discovering Mechanics**: Do I know what I can do in the game?
- **Setting Goals**: Do I know what I should accomplish in the game?
- **Planning**: What steps should I take to accomplish my goals?
- **Execution**: Was I successful in taking the steps in my plan?

Further, I discussed how classifying a game using this terminology can be useful for both developing and analyzing games: How, for each given game, each class can be described as *in-loop* or *out-of-loop.* That is to say whether the described class of failure is expected in the game loop or is to be avoided.

I also described how analyzing a game using the Taxonomy of Failure can offer insights into the analysis and critique of the given game and also offer another metric in deciding which design changes are appropriate given the expected players and the state of the game. Our hope is that after reading this chapter the reader's relationship with failure in their favorite games has changed ever so slightly.

The Taxonomy of Failure plays a foundational role in defining the goals of the computational prototypes described later in this dissertation. Each prototype focuses on tackling specific failure classes. These projects contain both player-facing and developer facing interventions. While player-facing interventions attempt to directly assist the players, the developer facing interventions support the game creators in supporting their players.

- *Talin* project, discussed in Chapter 5, tackles the **Discovering Mechanics** failure class by aiding developers create more responsive tutorials.
- *SMES* project, discussed in Chapter 6, converts **Execution** challenges to **Planning** challenges by allowing players to manipulate their playtraces.
- *Nav Assist* project, discussed in Chapter 8, supports players navigation in both **Execution** and **Planning** failure classes. This projects offers both player-facing assistance methods and also a developer-facing framework for future developers to design their own assistance methods.

Chapter 5

Talin: Dynamic Tutorials Based on the Skill Atoms Theory

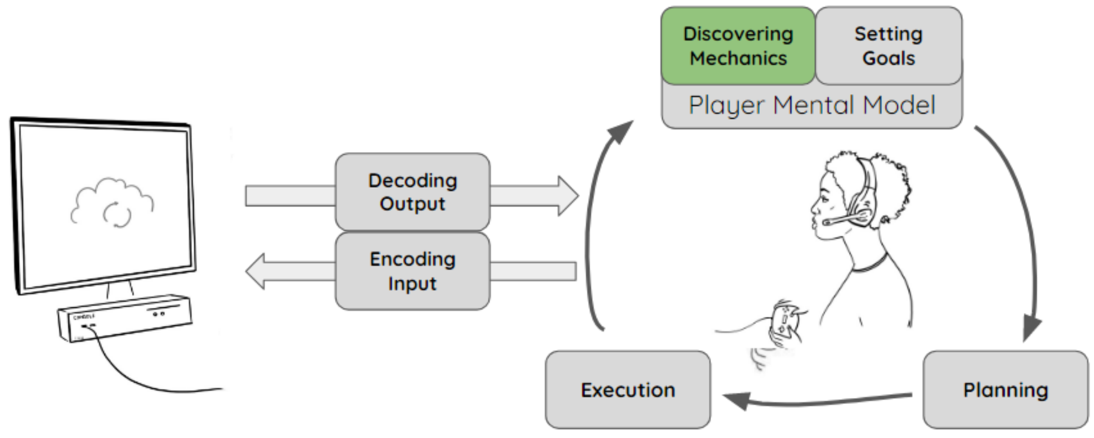


Figure 5.1: Talin assisting players in the Discover Mechanics failure class.

Large portions of this chapter was published in the AIIDE 2018 under the paper title "Talin: Dynamic Tutorials Based on the Skill Atoms Theory" [45] which was co-authored by my peer graduate students Isaac Karth, Jesse Harder and my advisor Adam M. Smith.

The opening minutes of any video game, which often take the form of a tutorial level, are extremely crucial. It is critical that the tutorial captures the attention of the player and teaches them how to read the game's visual interface and usefully act in the game world. If the pace of this initial level is either too slow or too fast the player loses interest: tutorials have a major impact on player progress for games with unfamiliar mechanics, particularly when the tutorials are context-sensitive. At the same time, tutorials can be detrimental in games where players are already familiar with the genre [38]. Therefore, it is in the designers' interest to have tutorials that can provide context-sensitive help when the player is unfamiliar with the game but also completely disappear when the player is comfortable.

Building on top of the Taxonomy of Failure, in this chapter I introduce the Talin framework,[1] to help designers implement adaptive tutorials without the need of any programming skills. Tutorials built with our framework continuously monitor evidence of player skills and act in the game world to give targeted feedback just where needed. Talin supports the players in the *Discovering Mechanics* failure class discussed in Chapter 4.

Not everyone requires an explanation of equivalent depth when it comes to learning new game concepts – not every player has identical gaming literacy. An experienced player will complain if the game locks away the interesting parts behind a series of simple tasks, such as camera control lessons, whereas a person new to video games will feel lost if not enough time is spent teaching the very basics. Rather than binning

[1]https://github.com/batu/talin

players into coarse classes (e.g. newbie versus expert) or projecting them onto a single axis of skill, Talin-supported tutorials can distinguish players along many dimensions, supporting players with distinct constellations of previous experience.

Meanwhile, in many video games, the tutorial is not integrated into the whole experience but rather exists only in the first few levels, or pops into view whenever a new mechanic is introduced. This front-loaded structure makes it difficult for players to remember the components of the game if they ever take a break or if a friend who was just watching the earlier interaction grabs the controls later on. When a player returns to any game after a hiatus, the natural instinct is to pick the game up from where they left off. In most cases this means they will spend some time trying to remember how to interact with the system, making very little progress. In severe cases, they will have to go back and replay the tutorial levels they already completed to bring themselves up to speed. Neither of these circumstances are desirable. Talin-supported tutorials can allow tutorial content to be sprinkled throughout a game's level progression (or even baked into the behavior of common level design elements) with the knowledge that it will only be seen by players who need to see it.

Even if the player does not take a break, it is difficult to gauge the player's understanding of the more complicated concepts. The game designers have to expertly craft their levels to ensure the desired progression is accurately reflected in the player's experience. In most cases, the system has no explicit feedback acting as a model of a player's mastery over the skills, which makes it impossible to give personalized help. The designers have to work with a one-design-fits-all structure which inevitably loses some players through the cracks.

The Talin system provides a solution to these problems by offering a new way of designing tutorials and making it seamless for any designer to implement. The system achieves this by operationalizing Dan Cook's skill atoms theory to build a mastery model representing the player's current understanding of the game. The mastery model can be queried to decide which information the player is lacking at any given moment.

On the high level, the way the system tracks the user's skill mastery is as follows: The designer defines what *skill atoms* need to be tracked, for example attacking breakable doors. The skill *mastery* for each skill is represented as a scalar.[2] Then the designer adds a *detector* to the breakable door object. The detector activates whenever the player is detected by the detector, in this case, when the player is near a breakable door. The detector will then decide how to *adjust* the skill mastery value, either increasing it or decreasing it. If the player is near a breakable door but they are not breaking the door, the skill mastery value will decay. If the player attacks the breakable door, the skill will be considered exercised and the value will increase. The designer can decide to react to the player based on the value of the skill mastery value by activating predetermined *hints*. For example, if the skill mastery value is below a threshold, the breakable door might start shimmering as a form of a subtle hint. If the value decays even further, a textual pop-up might appear reminding the button press for attacking. Through the combination of skill atoms (keeping track of individual proficiencies), detectors (understanding when a skill is relevant) and hints (the way the game responds)

[2]The designer may interpret a value between 0 and 1 as a probability that the player has mastered a given skill, but the framework does not enforce this interpretation.

designers track and respond to individual players.

We built the Talin framework as a Unity asset and focused extensively on usability and expandability. Skill atoms, detectors, and hints can be configured using the engine's graphical editor and placed into useful locations in the 2D/3D scenes of a game. We understand that, in most cases, video game tutorials are designed and implemented by game designers who are not proficient programmers. In order to ensure the tool is accessible and easily introduced into the workflow, we structured Talin to require no programming expertise. We also ensured that the framework is easily extensible if a developer with expertise wants to use the system for more specialized purposes (such as game-specific detector logic). Later in this chapter, we walk through a no-code implementation of dynamic tutorialization for the Unity-provided *2D Game Kit*[3] example game.

This chapter operationalizes Daniel Cook's skill atom theory into a designer-friendly tool. With the Talin framework we hope to streamline the process of making personalized tutorials for game designers and developers alike. The addition of this type of adaptive tutorials should increase the player retention rate as every player could get a tutorial tailored to fit their knowledge level. Adaptive tutorials will also help designers ensure that the player is effectively utilizing all the tools they have created. Overall, we believe this framework can greatly improve the process of authoring personalized tutorials and help create better game experiences.

[3]https://unity3d.com/learn/tutorials/s/2d-game-kit

5.1 Defining the Tutorial

We use the definition and categorization of Darran Jamieson, mainly because it has been picked up by some of the academic work we will be discussing.

Jamieson discusses four different ways games teach players how to play. [169] Non interactive in-game tutorials, interactive in-game tutorials, background tutorials and no tutorials. We will start by going over the definitions and leave the analysis until later.

Non-interactive in-game tutorials are usually displayed in a written form where the player has to read the instructions and learn what to do. Worse cases the player might face an image of a controller and a keyboard with each of the buttons annotated with the in-game action it activates.

Interactive in-game tutorials, instead of just describing what buttons to press, actually requires the player to do so. In most cases the progress is prevented until the player performs the actions that the tutorials are asking them to.

When a game embeds the tutorial text into the game environment it is referred to as background tutorial. While the player can not engage with the tutorial itself, the player is still in the game world and can play the game freely. Contrary to the interactive tutorial however, the player is not asked to accomplish any explicit task and more importantly the player progress is not artificially gated.

Finally, the no tutorial category refers to games where there is no explicit tutorial but rather the player is asked to learn the game through identifying the affordances and exploring the space of actions.

When we look at other articles in the wild we can see almost identical classifications [292] Unfortunately however this categorization leaves several obvious gaps in our understanding of what a tutorial is. First, this completely ignores any learning that takes place outside of the game, both through official (game handbook, marketing) and unofficial means (friends, walkthroughs and let's play). Second, there is no distinction between "No Tutorial" meaning there is no intention in how the game elements are introduced versus, "No tutorial" referring to a very intentionally crafted experience that doesn't use words to explain the mechanics. For a specific example of an intentionally crafted experience you can refer to Anna Anthropy's breakdown of the opening sequence of Super Mario Bros. [42]

Green et al. in their work [142] regarding generation of tutorials don't explicitly create a classification but highlight three common types: Teaching using instructions, teaching using examples [through non-player characters], teaching using a carefully designed experience. The former type is similar to the categories of Jamieson in regards to static instructions and the latter is pointing at the muddiness of "no tutorial" classification. Jamiesson's classification would categorize "teaching through examples" as a background tutorial, as it would be in the game, yet wouldn't be blocking progress.

Andersen et al. instead of categorizing tutorials directly, the authors offer four features to describe the salient aspects of any given tutorial [38]. The first feature is Tutorial presence which simply describes whether the game provides a tutorial to its players or not. The second feature, Context-sensitivity describes whether the suggestions are delivered within the application context or not. Third feature is called Freedom. This is the degree to which tutorials force the users to perform the mechanics being described. The final feature is availability of help which refers to whether the player can access the documentation on demand. We are going to come back to Andersen to delve deeper into this work.

So far we can see that there is no agreed upon way of understanding what a tutorial is and what combinatorial features can be used to describe them. Both Green et. al and Andersen et al. focus on other facets of tutorials rather than giving a throughout categorization. When it comes to industry, there is also no agreed upon, exact definition. This presents an opportunity for further research.

5.2 Measuring the Importance of the First Hour

It is important for both academics and practitioners to deepen their understanding of tutorials because they create the first impression the player has of the game, and it heavily influences whether the player wants to keep playing or not. Furthermore analysis around tutorials shows that there is a difference between how learning in games researchers and those who are more entrenched in game design look at learning in games. Gee's claims always approach games as a whole and do not differentiate between a tutorial level, and the rest [131]. For Gee, all of the game is "a learning machine." Whereas the research surrounding tutorials, by definition, is concerned with only a small part of the game. Then the question rises, is it warranted for us to reduce our scope of our research from the whole game to only the tutorial? I claim that due to the immense impact the first hour has on the whole play experience, the reduction in scope is justified, and even productive. This is one of the first differences between learning in games research and tutorial research. In order to justify my claim in the following section I will deeply look into research that discusses how impactful the first hour is, in informing the overall player experience:

Cheung et. al [80] tries to answer exactly how important the first impressions of a game are. They analyse over 200 game reviews and interview industry professionals to understand what aspects of the initial play session is relevant to keep players engaged, and whether the first play session is important at all.

The authors collect 35 reviews from a variety of gaming websites and 212 reviews from amazon, in 30 different game genres. They decided to study Xbox 360 games due to the "popularity of the console and because of similarities across all first hour experiences such as a common hardware specification. "–and potentially because this research team's host organization Microsoft, is affiliated with the brand Xbox, but this is pure conjecture on my part. These 212 amazon reviews were selected using a sampling of search terms that indicated the play session was constrained to roughly the "first hour". In addition to the reviews they also interview 3 game designers, two game testers and one user researcher with over 100 years experience between them.

Through the qualitative analysis the authors discover a common experience arc that takes place within the first hour: Expectations → Experience → Outcomes. Unsurprisingly players come to the game not as a blank slate but filled with expectations, whether it be through reviews, previous experience or influence of friends. One novel finding is that often, the expectation is a very concrete moment or action in the game and it is usually infeasible for players to reach that point in one go.

While discussing experience the authors first distinguish positive and negative experiences but go further to also distinguish experiences that satisfy a player versus those that intrigue them. The players who are intrigued express further interest in playing the game. In tagging the authors separate satisfying and intriguing moments using the terminology memorable and engaging. Memorable moments are those that conjure fleeting expressions of awe to the events on screen, whereas engaging moments are accompanied by an enthusiasm for more.

On the flipside the negative experiences also have seperate delineations: annoyances, frustrations and tedium. Annoyances are "small negative experiences for gamers such as music, sound effects, or glitches in the camera view of a game" and are considered individually dismissable as long as the frequency of these are kept low. Frustrations on the other hand are "acute, immediately causing players to consider quitting: being unable to play a game because of hours of patching are required, an unbeatable first level, and terrible controls." and evoke an immediate disengaging response from the player. Finally, tedium refers to cases where the player gets bored by doing the same types of events, and might indicate that the player has exhausted the value of the game.

When it comes to the outcome step of the first hour arc, the authors contribute two terms: Abandonments and Holdouts. Abandonment is the reasons (whether gradually or abruptly) that causes the player to stop playing the game. Whereas holdouts are the inverse, these are the reasons that keep the players engaged even if the rest of the game isn't up to their liking. Holdouts could be the earlier mentioned anticipated elements, a single game mechanic or a good narrative.

The authors then move on to the learning aspect of the first hour experience. They state that every game has to be learned on the elementary level, but furthermore, that designers have a clear vision of how the game is supposed to be played–the first hour is also tasked with teaching the player this ideal way. Towards this end the authors identify two challenges: Pitfalls and Tradeoffs.

Pitfalls are when a player misses a key detail in learning about the game, and as such abandons the game. One example of this is a player abandoning a game due to unskippable cutscenes. In reality, in said games the cutscenes were skippable, but the game hadn't made that clear enough to the players.

Tradeoffs refer to the tension between clear directions and a freedom to explore. The analysis shows that when directions are missing players feel frustrated. "Finally after running everywhere, I see there's a hallway up above the room with no access to it for anyone who can't fly. ... Not seeing it made me feel stupid. I don't like it when games make me feel stupid." On the other hand, the authors claim that when the game is too restrictive and doesn't allow for self exploration the players feel frustrated as well.

The authors affirm the importance of the first hour but suggest an unorthodox direction for development. The claim is that "intrigue trumps enjoyment" and the first phases of the games should be primarily concerned with building intrigue with engaging moments instead of evoking fleeting fun with memorable moments. They argue that this is valid as they categorize the players to be looking for the central purpose of the game in the first hour, rather than engaging with the game properly. In the tension between too much and too little hand holding, the authors take the side of more structure that hints to the depth of the game.

The team concludes the paper by stating the need for more research in the first hour. While this paper is not explicitly focusing on tutorials and learning, it is clear that some of the questions they raise pertain directly to these areas and add to our understanding that tutorials in games, combined with a deeper understanding of learning, is a valuable area of research.

5.3 Effects of Tutorials

Hopefully the findings of the previous section have justified focusing on tutorials. Let us go back to Andersen's work and keep investigating their findings. This will help us understand whether the findings in this domain parallel those in learning in games. In their paper they aim to understand the impact of different types of tutorials (as described above) on games of varying complexity. As such, in their study they use games with different levels of difficulty: *Hello Worlds* a puzzle platformer which doesn't stray away from the genre conventions, *Refraction*, a slightly more complicated education game that is used to teach fractions, and finally *Fold It* a game which doesn't easily fit into any genre and other traditional modes of play. None of these games are isolated to lab experiments, and each of them has been released into the wild accruing hundreds of thousands of game players which makes the overall study much more credible.

The papers outlines four hypotheses regarding tutorial design, one for each of the features they are testings:

1. Games with tutorials will exhibit better player engagement and retention than games without tutorials.
2. Tutorials that present instructions in context will be more effective than tutorials that present information out of context.
3. Tutorials that restrict player freedom improve engagement and retention.
4. Having on-demand access to help improves player retention.

The paper measures engagement in three ways. First they count the number of unique levels each player completes. Second, they calculate the total length of time and finally they measure the return rate which is defined as the number of times players restart the game. Because they did not have access to the individual participants–the players are engaging with the game in the wild, not in a lab setting– the quantitative engagement metrics were used to evaluate the effectiveness of the tutorials.

After testing with eight different treatments and over 40.000 players, their findings are as follows: "Tutorials were only justified in the most complex game." In Refraction and Hello Worlds, the lower and medium complexity games, the tutorials had no significant difference in engagement of the players, however, in Fold It the presence of the tutorial increased the levels played by 75%.

Context-sensitivity can improve engagement. In Fold-it context sensitivity played 40% more levels, yet in the other two games there were not significant positive effects. Tutorial freedom did not affect player behavior, in any of the games. To explain the situation the authors hypothesized that the fact that people don't enjoy being restricted might cancel out any positive benefit of focusing the players' attention.

On-demand help harmed and helped player retention. In Fold-it the effects were positive, with a %12 percent increase in playtime. However, in refraction the players with the on-demand help completed 12% fewer levers and played for 15% less time. The results get even less definite when you consider that only 31% of players had ever used the help button. It is possible that the players only click help when they are in need of help and can not figure out the puzzle on their own, which likely is also when they are frustrated. This is one finding that potentially contradicts Gee's principles, in this case on-demand & just-in-time principle.

Overall while this paper gives significant insight into how tutorials impact games, however their results and methodology also shows that there is significant room for further research. First, their methodology doesn't involve any qualitative data, which makes it hard to understand some of the more curious findings. Second, all of the games they test are serious in nature and attract a certain group of people. It is not obvious whether the claims generalize to commercial games. Third, their classification of tutorials fully disregards "teaching through carefully designed experiences." The games in question seem to be well designed, we can reach this conclusion by looking at their player numbers. This then raises the question whether "no tutorial" in this experiment actually means “teaching through carefully designed experience.”

5.4 Industry Wisdom Regarding Tutorials

One thing to recognize through this literature review is that game tutorials is an under researched area of study. Green et al. also reaches the same conclusion with their paper that “surveys the (scant) literature on game tutorials”. [142] Fortunately however, there is a significant body of work that we can explore from the game industry side that is concerned with building more effective and less intrusive tutorials. One aspect that ties many of these pieces together is their depiction of the current state of game tutorials and players’ response to them.

The GDC talk titled “Teaching by Design: Tips for Effective Tutorials from ’Mushroom 11’” by Itay Keren, the designer of the unorthodox puzzle game Mushroom 11 captures the industry wisdom when it comes to designing tutorials. Keren discusses eleven facets that make a tutorial effective. [185] Below I use these 11 tips as an anchor to summarize the industry wisdom around tutorials and describe how many of these tips map to Gee’s principles. Keren’s suggestions follow very closely to those provided in the Game Approachability Principles. [102]

Know what you need to teach. Keren suggests identifying the individual knowledge gaps for the first time players. He brings up the topic of Expert Bias, otherwise known as the curse of knowledge [150], where as experts of our own game we underestimate how much there is to know. This especially becomes problematic when we assume our players have a similar background in playing videogames as we do. Listing all the components of what needs to be taught foregrounds the necessity of addressing them individually.

Teach one thing at a time. Keren refers to the idea of system 1 (reactive) and system 2 (reasoning) thinking [177] and suggests that learning happens when users critically engage with the concepts rather than simply reacting to them. When several complicated concepts are being taught at once, it is likely that players get overwhelmed and start reacting instead of reasoning, which hampers the quality of the learning that takes place. A simple fix to this problem is introducing the concepts one by one. [98] calls this the isolation principle, and surprisingly this concept is missing from most other sources. This maps to Gee's well ordered problems principle.

Learn by play. This tip is based on the notion that self-acquired knowledge is better and longer retained. Keren suggests that instead of delivering the learning outcome to the player, designers should create spaces in which players can interact and discover the outcome themselves. He also mentions the importance of clear cause/effect patterns in the learning stage. Unpredictability inhibits learning making it more challenging to acquire an accurate model of how the system functions.

The idea that the player should learn by playing instead of reading text boxes is among the most common advice that is given.[30, 83, 110, 131, 98, 311] Some sources, such as Keren, are more thorough when detailing how to actually achieve this. This maps to Gee's manipulation principle.

Learn by observation. Sometime it is very difficult for the player to discover what needs to be done by simply exploring the same. In these cases the designer can implement cues that simulate the desired interaction. Instead of directly telling the player what to do, this would mean showing them. Keren suggests that even if the player observes an interaction first, due to the hindsight bias, they will feel as if they had discovered the solution by themselves.

A prominent game critic Egoraptor [110] especially goes deep into this topic, even using menu selection tools and cinematics as examples of how games can demonstrate certain actions.

Provide a safe place to experiment and fail. Keren states that people adjust their behavior according to the perceived risk levels. If players are put into a risky situation they once again start reacting intuitively instead of reasoning through the steps, which similar to tip #2 limits their learning. In cases where the game needs to teach the player about a hazard, putting a save point right before the hazard.

Set clear goals and avoid distractions. It is important for the player to know what the game expects them to do. Keren mentions that humans have a tendency to identify patterns, and when there is no clear goal, the player might identify a false goal and get frustrated when the game isn't responding to their attempts at reaching it. Furthermore, designers should be conscious of filling the scene with irrelevant and misleading clutter. Keren mentions that if players enter a state of functional fixedness [32], it is difficult to lead them back to the correct path. This maps to Gee's fish tank principle.

Make players stop and think. When teaching the player a complicated skill you might need to reduce the pace of the game to allow the player to think. If there is a simple, correctly looking solution that is wrong, the player might get fixated on that mistaken method. An example of these unclearable gaps, that is barely longer than the jump distance of the player character. Players might not recognize the gap is too wide, and keep trying to jump across, hoping that if they time it just right, they will reach to the other side.

This technique is covered by [128], but the terminology used there is "teach through small puzzles". The idea is simple, instead of showing an obvious answer, wrap the teaching idea behind a small challenge. This goes back to the criticisms of Linderoth of games as learning machines, and how games that don't follow this principle can carry the player forward without requiring them to think. This concept is also supported by the learning concepts desirable difficulty and mastery learning.

Hand Holding is sometimes necessary (when all else fails). In order for the player to explore the game they need to know what the elementary verbs and mechanics are. When basic controls of the game are being taught, it is acceptable to use text popups. Even then it is important to keep them brief, and teach them contextually.

I claim this is the fundamental tension when crafting tutorials which I will expand on in a later section. Before we move on however, [98] explains this tension elegantly: "Give too little information, and the player can become totally lost. Flood the player with information or hold the players hand too much and it takes away the satisfaction of discovering a game's complexity."

Repeat for retention, prime before ramping up. Keren claims that spaced repetition helps longer term retention. To act on this knowledge he suggests that whenever a difficult puzzle is going to be introduced, it is useful to prime the player with an easier one right before. This is also related to the idea of spaced repetition [180]. If the player practices concepts through a longer timespan, the retention is increased.

Tune your difficult curve. In addition to mentioning the relatively well known flow diagram, Keren also states that prolonged difficulty affects learning and retention negatively. He suggests instead of increasing difficulty linearly, the difficult should peak mid way. This allows stress to be replaced with euphoria at the endings stages of a grueling challenge, which increases retention. This maps to Gee's pleasantly frustrating principle.

Test your tutorial. As the final tip Keren suggests playtesting widely and repeatedly. He especially suggests playtesting at events, where there is a diverse crowd of people and no shortage of other interesting things that are trying to capture the attention of the player.

Through these eleven points I offer a birds eye view of the best practices when it comes to designing tutorials. This overview allows a comparison between the principles of a good tutorial and Gee's "good games as learning machines" principles.

5.5 Comparing Industry Design Wisdom with Learning in Games Research

There is a lot of overlap between these two camps. Both best practices in tutorial design, and principles of Gee emphasise utilizing the interactive nature of games as the primary means of teaching. The best practices of tutorial design are more specific, which is not surprising given the more applied nature of these ideas.

However not all ideas are represented equally across both domains. While Gee's principles spend a lot of time around the ideas of customization and agency, discussions of these concepts are almost entirely absent in the best practices of tutorial design. This can be explained by referring back to our earlier observation: Gee's principles engage with games as a whole, whereas learning/teaching insights seem to be focused on the first-hour and tutorials. From the perspective of tutorial design, customization and agency seem to be attributes inherent to games as a whole and not only tutorials themselves and as such are not included in the discussion. This showcases the potential for further research in exploring the effects of customization in game tutorials. On the flipside what is missing from Gee's principles is any discussion of usability concerns. This point was also made earlier when discussing breakdowns as the breakdowns related to parsing the screen or taking action were not represented in Gee's principles [259]. This can be explained by acknowledging that Keren's principles are much more practice and design oriented whereas Gee's principles engage with ideas on a higher level.

There seems to be a further need of validation and clarification both in Gee's principles but also the best practices of designing tutorials. For example Anderesen's results of on-demand help reducing engagement shows that sometimes experimental results can go against intuition. Furthermore, similar to the lack of clarity around what "good games" are, there also doesn't seem to be consensus around what a tutorial is. This seems especially problematic given the heavy underlying assumption that players hate tutorials. It would be curious to claim players hate Mario 1-1, one of the most praised levels in gaming history, yet our current understanding of what a tutorial is can not cleanly make the distinction between Mario 1-1 and the tutorials that are disliked. I further claim that this is more than a simple "definition problem" and in reality might have very real consequences. If a designer has the idea "players hate tutorials" ingrained into their brains, then they might simply choose to spend less time working on the tutorial or skip fully. While this last part is pure conjecture on my part, I claim that the dogmatic belief that players hate tutorials is up for a deeper investigation.

Finally the tension between too much & too little hand holding seems to be a concern that arises in both domains. The agent and co-design principles of Gee suggest the importance of free form exploration and active learning. These principles state that excessive structure isn't optimal for learning. Tension is much more apparent when it comes to game design practice. Every single one of the sources that claim tutorials are disliked links the distaste to the failure of the tutorials to balance free exploration with structure. The industry wisdom is clear in stating that hand holding in general is not effective. What the industry wisdom doesn't help with is actually deciding when handholding is necessary. I claim this tension between too much hand holding and not enough guidance can be resolved by looking into the specifics of how people fail.

For the specifics of the Talin project we based our approach on the knowledge model of the player. The initial inspiration for the project was the skill atoms player model developed by Daniel Cook [88]. The player's model of the game system is disassembled into atomic components, individual skills that the player learns and combines to form more complicated skills. The player interacts with the game in a feedback loop, as the actions the player takes update the state of the simulation, which gives feedback to the player. The player uses the feedback to update their internal model of the game and uses that to inform their next decision. Each individual unit of action-simulation-feedback is a skill atom. In Cook's model, skill atoms can be chained together to describe more complex skills that are assembled out of multiple atoms: for example, learning how to stack blocks efficiently in Tetris depends on first learning the skills to move and rotate the blocks.

The skill atom model was originally developed as part of an effort to create a game grammar, and it has been adopted and adapted several times. For example, in the HCI field the model has been amended by Sebastian Deterding to consist of ”goals, actions and objects, rules, feedback, emergent challenge, and motivation” and combined with “design lenses” and “intrinsic integration” to arrive at a theory of intrinsic skill atoms for “gameful design” [103].

Naturally, skill atoms are hardly the first attempt to model user knowledge. For example, a formal knowledge model of a user's understanding can be represented as a graph of the knowledge space [104]. The introduction of Intelligent Tutoring Systems

(ITS) was predicated on personal computers becoming fast enough to generate intelligent computer-assisted instruction based on cognitive science models of how a student acquires new cognitive skills [40]. In contrast, rather than being concerned with the formal accuracy of our player-knowledge model, our approach is oriented toward the process of designer-accessible model specification and the output of the dynamic tutorials in a widely adopted game making environment. In this sense, our work is related to the Vixen project [107] which aims to support visual analytics directly from within the Unity editor UI.

ITSs have been the basis for several tutorial generators. One recent example is the Thought Process Language (TPL). While TPL can "generate explanations for a given problem" it requires that an algorithm for solving the problem first be encoded in TPL [236]. Further, it assumes that the goal of the tutorial system should be to give the user complete understanding of a specific problem solving process. In contrast, our system is designed to be added to an existing game without reimplementing the gameplay elements, focusing on the elements of tutorial generation that are more important for games. Importantly, Talin is designed to suggest ways for the player to interact playfully with the game, not to instruct the player on the optimum solution.

Another relevant concept present in the learning sciences literature is *zone of proximal* development and it refers to the the space between what a learner can do without assistance and what a learner can do with guidance. It is suggested that being in this zone has positive effects on learning effectiveness. [105] The dynamic guidance given by Talin improves the likelihood that the player stays in their zone of proximal development throughout their playtime.

For a game-specific approach, Michael Green et al. suggested ways in which AI techniques could be applied to the tutorial generation problem in the General Video Game AI (GVG-AI) framework [143]. Our goal differs from this project: rather than inferring the game rules that the tutorial explains, we assume that the developer is best equipped to manually specify the elements using our tutorial development toolkit. Our focus is on the dynamic presentation of the tutorial to the player, using the skill atom graph to dynamically focus the tutorial on only the elements that the player needs to have explained.

Another related research area is dynamic difficulty adjustment (DDA) [158]. While dynamic difficulty adjustment systems implicitly have a player model, they are more concerned with keeping the player in the flow channel rather than tracking which skills the player knows about or strategizing how to introduce an unknown mechanic to the player. Dynamic difficulty adjustment can be complementary to a dynamic tutorial system- perhaps the system displays hints before later escalating to adjusting the difficulty, while using the knowledge model to better track if the player is finding particular skills to be too frustrating or boring.

5.6 Talin Technical Design

The Talin framework consists of three modular building blocks that the game designer manipulates to create the dynamic tutorials (Fig. 5.2). Skill Atoms capture

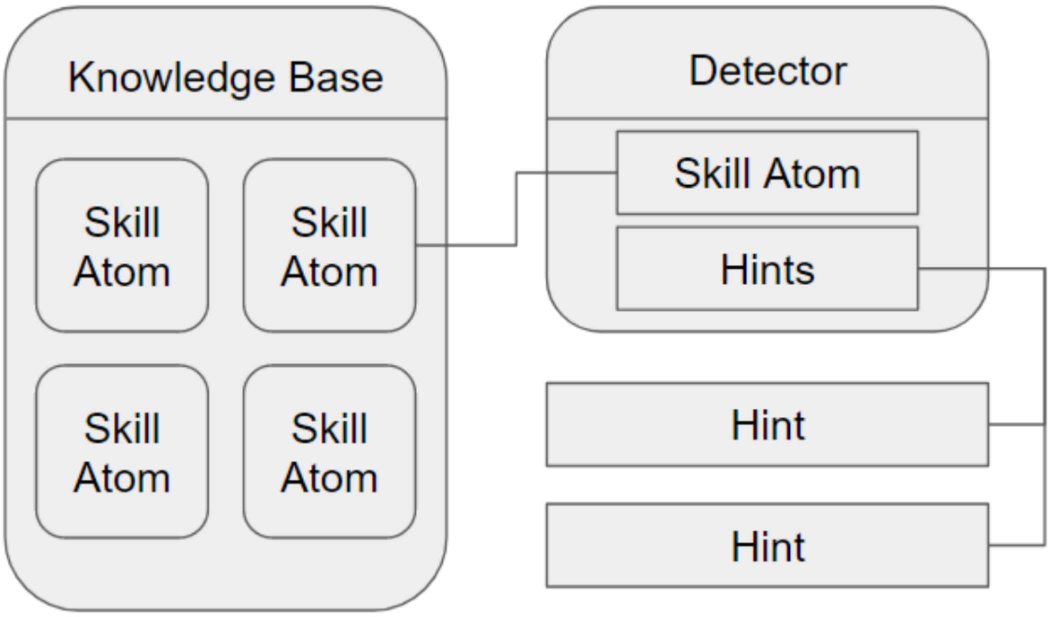
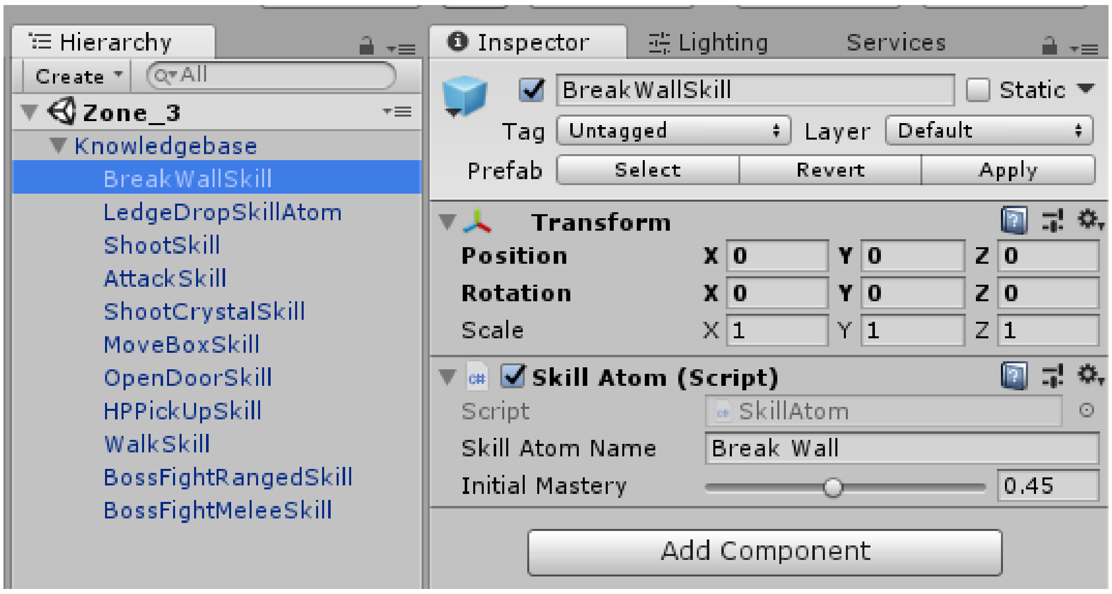


Figure 5.2: Overview of how skill atoms, detectors, and hints are structured.

a player's current mastery of a skill as a scalar value, Detectors decide when a skill mastery update is relevant, and Hints activate whenever the skill mastery value crosses a designated threshold.

Consider the mechanic of attacking a breakable wall in order to remove it. This is a game mechanic that can benefit from being taught in a dynamic fashion for several reasons: First, it is a common game mechanic that seasoned players will have internalized. Second, the mechanic affords a sense of discovery that could be undermined if a tooltip pops up and reveals the salient information too soon. Third, it is a context specific skill that is only applicable when a breakable wall is present. Due to the skill not being in the core gameplay loop, it can be easily forgotten when the player takes an extended break from the game.



Hierarchy

Create ▾ All

- Zone_3
  - Knowledgebase
    - BreakWallSkill
    - LedgeDropSkillAtom
    - ShootSkill
    - AttackSkill
    - ShootCrystalSkill
    - MoveBoxSkill
    - OpenDoorSkill
    - HPPickUpSkill
    - WalkSkill
    - BossFightRangedSkill
    - BossFightMeleeSkill

Inspector | Lighting | Services

✓ BreakWallSkill ☐ Static

Tag: Untagged | Layer: Default

Prefab: Select | Revert | Apply

Transform

Position	X	0	Y	0	Z	0
Rotation	X	0	Y	0	Z	0
Scale	X	1	Y	1	Z	1

✓ **Skill Atom (Script)**

Script	SkillAtom
Skill Atom Name	Break Wall
Initial Mastery	0.45

Add Component

Figure 5.3: In-editor Talin designer interface showing how the Skill Atoms are initialized.

The implementation process starts with the designer creating the skill atom representing the BreakWall skill (Fig. 5.3). The designer then initializes the skill atom with a starting value that represents the initial skill mastery of the player. The designer is free in deciding how to initialize the skill mastery level. If they wish, they can assume prior knowledge of a common game mechanic, such as moving around and initialize the skill at a high value. Otherwise, if the skill in question is more domain specific, they can initialize the skill at a much lower initial value. The skill mastery value range is set to be between 0 and 1, where 1 represents total mastery of the skill and 0 represents lack of any understanding regarding the skill. Regardless of the initialization, as the player progresses through the game this skill mastery value increases or decreases to reflect the understanding of the player for the wall-breaking mechanic.

Not every skill that is required by the game is relevant throughout the experience. BreakWall skill is only of interest when the player is near breakable walls. This is where detectors come in. Detectors allow designers to specify when a skill atom is exercised (the atom's value is increased) or decayed (its value decreased). Designers can choose from several built in detectors covering the majority of simple cases. Those with programming experience can even extend an existing detector class to cover complex edge cases. The most commonly used detectors are proximity detectors and input detectors.

As the name suggests, the proximity detector allows the designer to check whether an object, often the player, is near the detector. For the BreakWall skill, a sensible option is to attach a proximity detector to the breakable wall object and decay the BreakWall skill whenever the player is within a set radius (Fig. 5.4). The proximity detector represents the idea that if a player is standing right next to breakable wall, and not breaking it, they might not realize that the wall is actually breakable. The longer the player actively lingers right next to the breakable wall without making any progress, the more likely it becomes that they need a hint to figure out that the wall is indeed interactable.

Yet, it is not enough to only decay the breakable wall skill value. The designer also needs a method to increase mastery level whenever the player show that they know how to break a wall. To compliment the proximity detector, an input detector can be used. The input detector tracks whether a player successfully attacks and breaks a wall. Using the combination of the proximity detector and the input detector, the designer

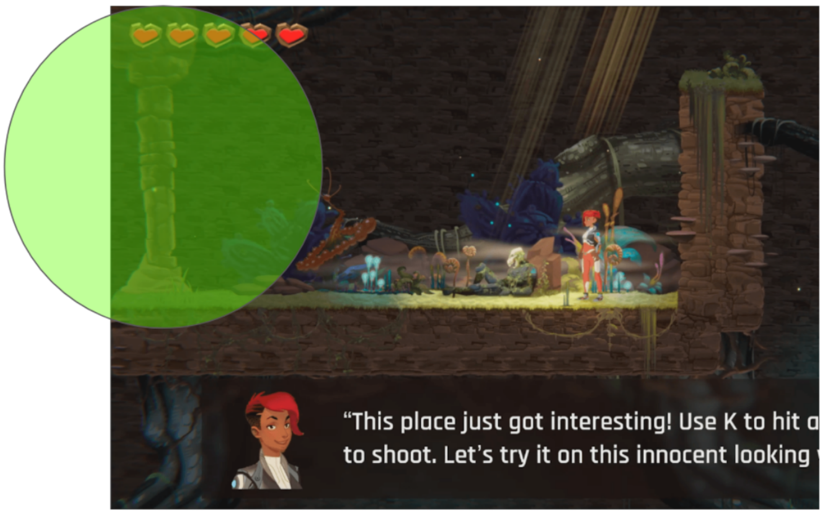

"This place just got interesting! Use K to hit a
to shoot. Let's try it on this innocent looking

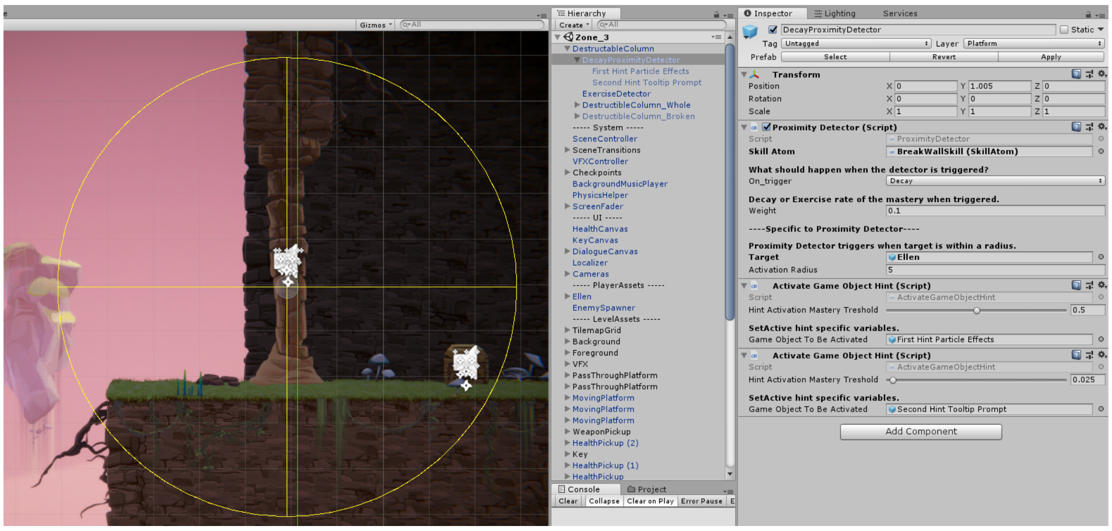
Figure 5.4: Visual representation of the proximity detector, indicated by the green circle. The proximity detector is attached to the breakable wall, and linked to the BreakWall skill.

is able to update the wall breaking mastery of the player simply by observing the play pattern. Due to the context specific implementation, the updates only happen when the skill itself is relevant. In order to allow designers to customize how the skill mastery values change we offer different ways of manipulating the value itself. The designer can pick between a linear change (increase/decrease by a constant amount), logistic change (reduce the distance to a target value by a constant fraction), or exponential change (scale the value by a constant fraction).

Hints are the structured way to create in-game events that trigger in relation to the given skill mastery level. A hint can be a simple tooltip pop up or even a sound cue. Or they can be much more intricate, resulting in substantial changes within the game systems. A hint activates when the mastery value of a skill crosses a designated threshold. It is up to the designer to define their own hints and at which mastery level the hints trigger. There is no limit as to how many hints can be attached to a particular skill atom. This allows the designer to combine different hints as they see fit. Using multiple hints with distinct thresholds, designers can start by introducing subtle hints and gradually become more explicit as the player shows signs of struggling.

Within the tool we offer several pre-built hints such as activating a particle system, playing a sound cue, or introducing a tool tip. For example, for the BreakWall skill, the designer can decide to activate a hint that creates a particle effect system emphasizing the cracks on the wall when the skill atom decays past a certain threshold. If the player stays within the radius of the proximity detector without breaking the wall even further, a second, more explicit hint can activate. Since the player missed both the cracks on the wall and the animated particles, the new hint has grounds to be very direct: a tooltip explicitly stating that the wall is breakable appears. For extreme scenarios, we can even imagine a hint implementation that assumes control of the player character to physically demonstrate the desired action (something the player might perceive as a cutscene).

Using the combination of skill atoms, detectors, and hints, a designer creates levels that adjust to the players. The skill atoms give the designers a method to keep track of what the player knows. The detectors allow the designer to manipulate the skill values. Finally, the hints introduce a convenient way to trigger user-defined in-game events.



```
Zone_3
  DestructableColumn
    DecayProximityDetector
      First Hint Particle Effects
      Second Hint Tooltip Prompt
    ExerciseDetector
    DestructibleColumn_Whole
    DestructibleColumn_Broken
  ----- System -----
  SceneController
  SceneTransitions
  VFXController
  Checkpoints
  BackgroundMusicPlayer
  PhysicsHelper
  ScreenFader
  ----- UI -----
  HealthCanvas
  KeyCanvas
  DialogueCanvas
  Localizer
  Cameras
  ----- PlayerAssets -----
  Ellen
  EnemySpawner
  ----- LevelAssets -----
  TilemapGrid
  Background
  Foreground
  VFX
  PassThroughPlatform
  PassThroughPlatform
  MovingPlatform
  MovingPlatform
  MovingPlatform
  WeaponPickup
  HealthPickup (2)
  Key
  HealthPickup (1)
  HealthPickup
```

```
DecayProximityDetector                      Static
Tag  Untagged            Layer  Platform
Prefab  Select  Revert  Apply

Transform
Position   X 0   Y 1.005   Z 0
Rotation   X 0   Y 0       Z 0
Scale      X 1   Y 1       Z 1

Proximity Detector (Script)
Script                  ProximityDetector
Skill Atom              BreakWallSkill (SkillAtom)
What should happen when the detector is triggered?
On_trigger              Decay
Decay or Exercise rate of the mastery when triggered.
Weight                  0.1
----Specific to Proximity Detector----
Proximity Detector triggers when target is within a radius.
Target                  Ellen
Activation Radius       5

Activate Game Object Hint (Script)
Script                  ActivateGameObjectHint
Hint Activation Mastery Treshold    0.5
SetActive hint specific variables.
Game Object To Be Activated     First Hint Particle Effects

Activate Game Object Hint (Script)
Script                  ActivateGameObjectHint
Hint Activation Mastery Treshold    0.025
SetActive hint specific variables.
Game Object To Be Activated     Second Hint Tooltip Prompt
```

Figure 5.5: In-editor Talin designer interface. The proximity detector is displayed in the level editor view on the left as a yellow circle. In the center, the Detector is listed in the scene hierarchy, parented under the breakable object. On the right is the interface for the proximity detector properties, which allows the designer to completely specify the behavior without needing to use code.

5.7 Tutorial Implementation

Talin can be used without any programming knowledge. All of the skill atoms, detectors and hints can be implemented using the visual interface that Unity offers (Fig. 6.2). We believe it is crucial for the tool not to require programming to function. In most cases the person who is tasked with creating the tutorial is a game designer who might or might not be comfortable with programming. It is important that the tool offers enough flexibility out of the box to seamlessly adapt into the workflow of whoever is using it. If someone can create the rest of their tutorial scene with graphical scene editing tools, those tools should also serve them in making the tutorial dynamic. While we discuss our Unity specific implementation in detail as an example, we expect the developers to re-implement these ideas in manners they see fit for the games they are creating regardless of the technology stack they use.

In this section, we walk through a no-code implementation of dynamic tutorialization for the Unity-provided *2D Game Kit* example game. This example game includes an existing, static tutorial design that we adapt via integration with Talin-provided building blocks.

Referring back to our example, if a designer wants to implement the BreakWall skill they start by instantiating a Knowledgebase prefab[4] in the editor and attaching a skill atom GameObject to it as a child. The skill atom game object has input fields for its name and the initial skill mastery value. The Knowledgebase is simply a container for all skill atoms. Then, they will attach a proximity detector game object to the breakable wall prefab and link it with the BreakWall skill atom through the visual interface. The ability to add a detector as a child of another game object makes it so that only one change to a prefab propagates through all instances of said object throughout the game. Even if the tutorial is being implemented long after other levels are completed, implementing the tutorial at the prefab level makes it relatively easy to embed the dynamic tutorial all throughout the game with very little effort. After linking the skill atoms and the detector all that remains is to set up the desired hints. The hints are also added to the detector object as a script. This allows for the designer to have the ability to have different sets of hints at different detectors. In the case of the BreakWall skill, the designer can select the predefined hint that activates a game

[4]In Unity, prefabs are the editor's abstraction of game object templates. In a lower-level game engine, our framework primitives might be realized as base classes to be instantiated or extended.

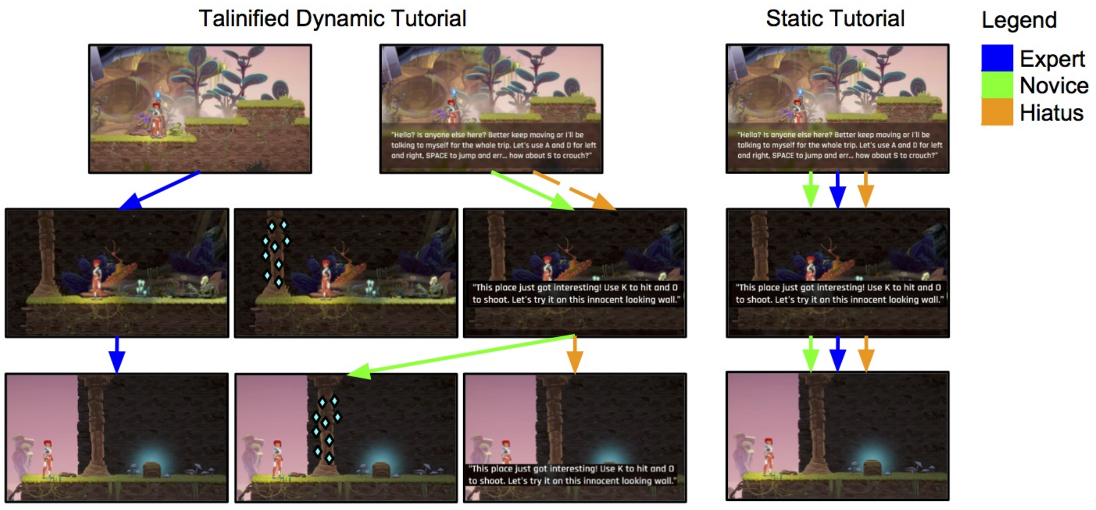
object, and through the visual interface set it up so that whenever the BreakWall skill decays under a certain threshold a tooltip gameobject activates.

While the simple workflow of Talin requires no programming knowledge, the system allows for easy expandability. If a developer wants to have a certain detector trigger in more specific scenarios than simple radius or input situations, they can extend any of the base classes to include the desired functionality. Consider a case where the designer wants to introduce a PickUpHealthPack skill atom. In this context, a standard proximity detector will not be sufficient. If the player is lingering next to a health pick up while they are not missing any health points, the designer can not assume that the player doesn't know how the health pack functions, as they might be deciding not to pick it up so as to save it for later. Thus, the proximity detector can be extended with a few lines of code to decay only when the person is near a health pick up and they are missing a certain amount of health points. While this is a trivial example, the extendibility of the base classes allows the developers to create game specific structures to support the type of tutorials they want to achieve.

The tool also comes with simple debugging support, including visualizations of the values of each skill atom and how they change as the player progresses.

5.8 Example Personas

Compared to the existing static tutorial in *2D Game Kit*, the dynamic tutorial made by Talin offers different experiences to different players (Fig. 5.6). For example



"Hello? Is anyone else here? Better keep moving or I'll be talking to myself for the whole trip. Let's use A and D for left and right, SPACE to jump and err... how about S to crouch?"

"Hello? Is anyone else here? Better keep moving or I'll be talking to myself for the whole trip. Let's use A and D for left and right, SPACE to jump and err... how about S to crouch?"

"This place just got interesting! Use K to hit and O to shoot. Let's try it on this innocent looking wall."

"This place just got interesting! Use K to hit and O to shoot. Let's try it on this innocent looking wall."

"This place just got interesting! Use K to hit and O to shoot. Let's try it on this innocent looking wall."

Figure 5.6: The experience of the different personas through different game moments, contrasting our dynamic tutorial experience with the traditional static tutorial.

the expert player starts the game and immediately knows how to navigate the space. Because they move around with ease, the input detector recognizes the player exercising the movement skill and thus, no hint regarding movement appears.

When the expert player reaches the first breakable wall segment, it takes only a few seconds for them to realize there are cracks on the wall sprite. At this point, due to their previous experience the expert player discovers that they can attack the wall to destroy it. Similarly, when they reach the second breakable wall several levels down the line, they do not need any support.

Overall, the expert player, due to never needing any help, never sees any of the hints. This allows them to discover some of the game mechanics themselves hopefully increasing their enjoyment.

Whereas an expert player doesn't need any help, a novice player might need more pointers than we expect. When the game starts the novice player spends some time exploring the controls but doesn't figure them out. The initial detector tracks that the player is not making any progress and activates the first hint explaining how to navigate the game.

When the player arrives at the second moment, they spend some time near the breakable wall without making any progress. Due to the proximity detector this loitering incrementally decays their BreakWall skill. When the break wall skill crosses the first threshold a particle effect system activates drawing the attention of the player to the well. Yet, the player spends a little bit more time near the wall without any progress. Then the second hint activates, explicitly telling how to make progress at this step.

When it comes to the third moment, another breakable wall several levels into the game, the player still takes a few moments. This time however, they do not need the tooltip to appear as the particle effects are enough for them to remember the break wall mechanic.

Next we consider a player who has gone through the first two moments, then has decided to take a long break from playing the game before coming back. Their memory of the two moments and the tutorial prompts attached to them is blurry. When the player reaches the third moment, they do not remember the break wall mechanic. As they linger, the attached proximity detector decays their BreakWall skill. When the mastery level crosses the first threshold the particle effects appear, but in this case that is not enough. The player spends a few more seconds within the radius of the proximity detector which results in the explicit tooltip hint activating. The player remembers the controls and the mechanics. This allows them to keep on making progress, instead of getting frustrated and stopping playing the game altogether.

Overall every single player gets an experience that is specifically tailored to their skill level, which is made possible by the constant tracking of the individual mastery levels.

Conclusion

In this chapter we presented a new tool designed to help game designers create dynamic tutorials. We showed how Talin operationalizes Dan Cook's skill atom theory to create compartmentalized skill mastery tracking. We showed the steps outlining how to create a dynamic tutorial and discussed how different players engage with the dynamic tutorials.

While this project mainly focused on contributing in the area of tutorials, we believe this granular level of skill mastery tracking can be an incredibly valuable dataset when it comes to understanding the holistic experience of the players going through games.

Talin currently requires the manual specification of skill chains. While this is consistent with our design goal of developer-control, a possible future direction is to build on the research by using Cognitive Task Analysis to identify skill chains [156].

We believe that it should be useful to model the player's mastery level for many more skills than are explicitly addressed with hints. Additional skills that are tracked behind-the-scenes should provide useful context for nuanced gameplay analytics. Beyond knowing where in a given level players are likely to abandon gameplay, we would like to know which concepts they were struggling with. Analytics based on skill level might suggest the need for additional dynamic hints far from the game's traditional tutorial level.

Chapter 6

SMES: Adapting Dexterity-Based Games for Deliberative Play

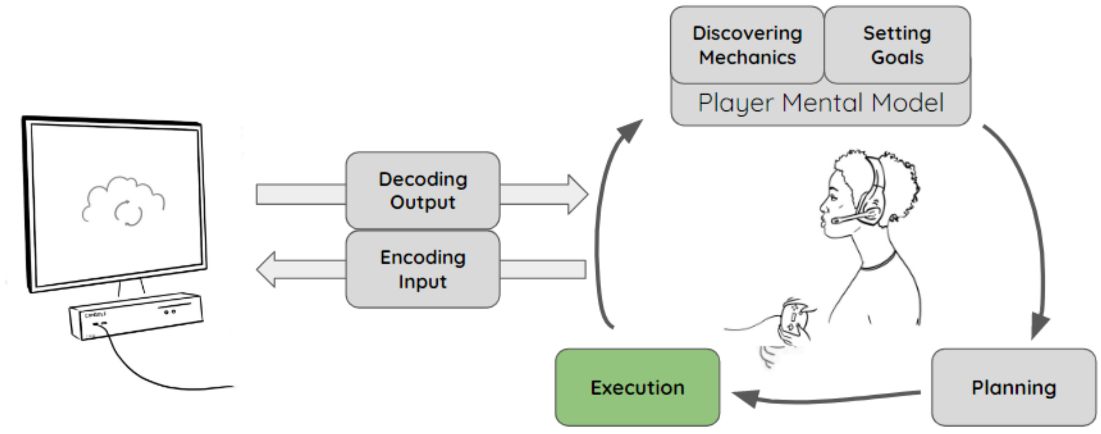


Figure 6.1: SMES assisting players in execution failure class.

Large portions of this chapter was published in the Experimental AI in Games Workshop in the AIIDE Conference 2020 under the paper title *"SMES: Adapting Dexterity-Based Games For Deliberative Play"* [48] which was co-authored by my undergradute mentees Hunter Lynch, Carl Erez, Jesse Harder and my advisor Adam M. Smith

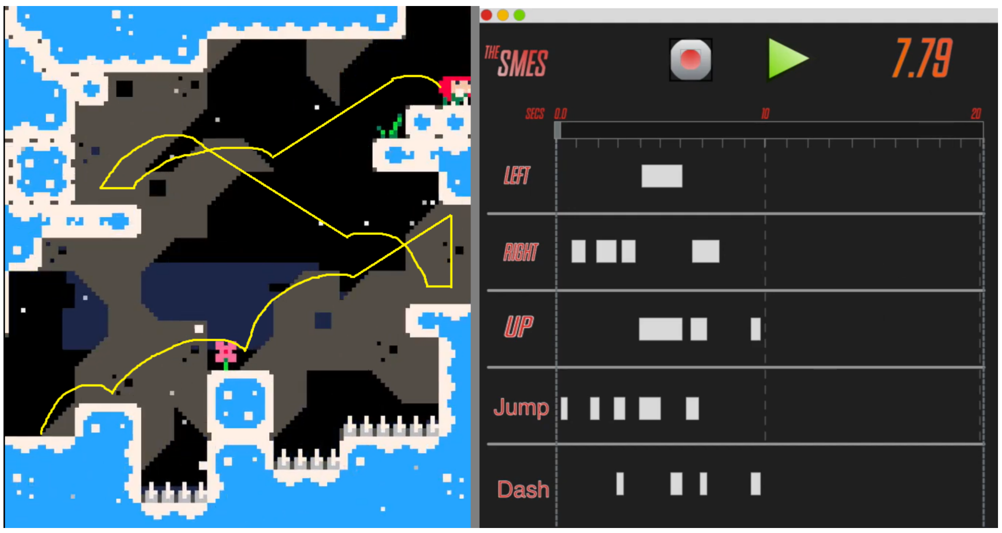


Figure 6.2: On the left, the approximate path that player took from the bottom left of the map to the top right of the map in the game Celeste. On the right, the buttons press sequence that the player pressed in order to take this path.

In the previous chapter, with *Talin*, we looked into how we can support players learning the games. In this chapter, we discuss the *Sharp Multi-input Editing Software* (SMES). The SMES allows converting high-paced dexterity based challenges to deliberative planning challenges in a narrow set of games that exist in the wild. SMES was built with difficult platformers with very short deterministic levels in mind such as Celeste [205] and Super Meat Boy [299].

When faced with barriers to engagement it is common for people to modify games through using the different interfaces to play the game such as switch controls [275] or Microsoft Universal Controller [111], or by changes in software via trainers.

Inspired by the spirit of modifying how people engage with games, SMES allows the players to save, display, modify and replay their keypress sequences. SMES works through the following four steps:

1. The player attempts the challenge and SMES records their keypresses.
2. The SMES visualizes the keypresses over time.
3. The player makes experimental changes to the keypresses.
4. The SMES replays the edited keypresses (and the player returns to step 2).

This loop allows the player to change how they engage with dexterity based games. The challenge stops being about the precise timing of button presses, but rather becomes more like a puzzle game where the player incrementally refines the play trace until they clear the level.

When we extract and replay the playtrace of the player, we treat the sequence of actions as a stateless, fixed policy. Similar stateless action sequences have been used successfully for difficult exploration problems [109] [322] and is shown to be a promising starting point.

We believe that AI techniques can be effective in assisting players while operating on playtraces. Representing dexterity based games as playtraces, and then recording how player changes allows us to get another type of data as to how a player might improve their playtraces.

In the rest of this chapter we will discuss the technical details and the design decisions of SMES, and how the extracted play traces can be a fruitful avenue for AI research.

Videogames provide intrinsically valuable experiences to millions of individuals [175]. Maybe more importantly, games are an important part of how people connect to one another [235]. While multiplayer games offer moment-to-moment connectivity, the connective nature of games also extends to single player games through creating a set of shared experiences [170] and communities [137].

It is important for everyone to be able to engage in this shared culture. Yet, through a combination of design decisions of developers and differing abilities of players, a significant group of people are excluded from engaging with games [298]. To make games more inclusive, in recent years there have been a number of games that include assist modes such as Celeste [205] and Super Mario Odyssey [226]. These assist modes allow the player to change the magnitude of difficulty they encounter by tweaking variables such as game speed, player and enemy attributes and other relevant factors. The addition of assist mode has been met with positive feedback from the players [187].

Most assist modes do not modify what type of challenge there is but rather they work by changing the magnitude of the challenge. In this project we wanted to explore whether we could change the type of challenge that the game presented. We use the Taxonomy of Failure (ToF) to categorize the types of challenges [50]. ToF offers six different categories of failure which map to different types of challenges that exist in a game. For this project the relevant categories are Planning and Execution. Planning refers to the challenge category where the player is deciding on what actions to take. The Execution category on the other hand refers to the challenge of the player to actually take those actions, by pressing the correct buttons in the correct order and timing.

In the context of difficult platformer games, which SMES is targeting, the planning challenge would be deciding what route to take and when to use the relevant abilities such as dash or jump, and the execution challenge would be related to pressing the buttons to activate the relevant abilities at the correct time. Our goal in this project is to replace the challenging execution aspect from playing difficult platformers into a planning one.

Instead of building a new game that has the planning challenge present as a default, in this project we explore a tool that targets already existing culturally relevant games and supports people's ability to engage with them. While there have been many successful games that are specifically designed to target groups with disabilities [141] we believe that due to the social nature of games it is important to support players in engaging with pre-existing cultural artifacts. The player centered assistance nature of the SMES allows the player to be in charge of how to, and with which game, to SMES with.

In building the SMES we were inspired by the ideas of transgressive play [26] and the diversity of ways players customize how they engage with games. One example of this is the WeMod community[1]. WeMod allows players to reconfigure over a thousand single player games through a unified interface, letting them customize the game to their liking, often through changing difficulty and progression variables. As of July 2020 WeMod advertises that it has over 8 million users. This shows that a significant

[1]https://www.wemod.com

population of players are accustomed to modifying their play experience by reaching outside the game.

The Speedrunning community is another group that inspired this project. Speedrunners are interested in completing a game as fast as humanly possible. Tool Assisted Speedruns (TAS), take this concept a step further and use emulators to complete a game as fast as possible using any and all available tools. While most TAS systems run the game in an emulated environment to have frame by frame control, in order to increase generalizability of our system we simply replay keypress events (which is not guaranteed to reproduce gameplay over long periods of time) instead of attempting to emulate the whole game.

The disability community has also been customizing the way they engage with games through a combination of software and hardware [58]. The Accessible Player Experiences [71] design patterns are a great resource when it comes to making games more inclusive. These 22 patterns target both the ability of the players to access the game content, and their ability to adjust the challenges present in the game. In the later chapter we will be discussing how SMES fits these design patterns.

6.1 Technical Design of the SMES

6.1.1 Design Inspirations

Before the development phase of the SMES we reached out to different communities on Reddit to inform our design. We had conversations on the disabledgamers, TAS (tool assisted speed run) and pc gaming subreddits. While few in number, the discussions around how such a tool might be used was immensely valuable in prioritizing certain features over others and understanding potential user needs.

Our design was informed by a public discussion of a system prototype on Reddit's `r/disabledgamers` community and `r/TAS` communities

We incorporated *casual creator* design patterns wherever possible [86]. Casual creator patterns aim to make systems more usable and friendly. While these design patterns are traditionally used in the context of generative systems, we found them useful in our design. Like most casual creators, SMES is a tool that prioritizes approachability and flexibility over control and frame-perfect precision.

Finally SMES also drew inspiration from the Accessible Player Experiences [71] (APX) patterns. APX patterns have two distinct categories: Access patterns support players accessing the game by allowing them to tune the experience to meet their unique needs, and challenge patterns on the other hand allow players to modulate the amount of challenge they want to face.

6.1.2 Recording Key Presses

One decision we had to make early on was what layer of abstraction to have SMES integrate. While extending an emulator would allow us to be much more precise both in recording and replaying our playtraces, it would also limit the games that the SMES would be compatible with to those that could run on the decided emulator. One of our guiding principles while building SMES was to support access to a broader set of culturally relevant games, and we therefore decided not to limit ourselves with any specific emulator.

In order to reach a wider range of games SMES uses the Win32 API[2] to integrate into the keyboard input layer. We also investigated the reverse-engineering tool Frida[3] as means to be fully platform agnostic. This would have allowed us to potentially save and restore game state which would have unlocked a more granular control of how time passes in the game [262]. This control over time could have allowed more comprehensive assistance methods to be implemented. In the end however, we decided to focus on capturing keyboard calls to reduce user burden of instrumenting each game specifically.

After the player starts recording, the SMES intercepts and saves the virtual keyboard calls that happen while the game is focused. When the player stops recording, the keypresses are saved into a text file. Having players seed the starting playtrace with their own action connects with the "No blank slate" pattern of casual creators. While the players can start from scratch to create the full playtrace using SMES, having a starting playtrace to edit makes the iterative process much faster.

To support the quickest edit-and-replay interaction loop, it is the player's responsibility to include input sequences that would reset the game to a safe state (e.g. accessing an in-game menu to restart the level from a checkpoint). If there is no such obvious sequence to include in the playtrace that will accomplish this goal, the player can always use manual controls (similar to how they recorded the original playtrace) to

[2]https://docs.microsoft.com/en-us/windows/win32/api/
[3]https://frida.re/

reset the game before continuing the SMES loop.

6.1.3 Representing Key Presses

It is common for speed-runs to be shared through a BK2[4] file. These files are a way to record which combination of keys were pressed down at which frame. Inspired by this, our intermediate representation is in a simple human readable text file. This was decided in part to support the easy sharing of play traces. The easy sharing component is aligned with the "Self promotion" casual creator design pattern that acknowledges the shared and social nature of playing games.

Having a semi-permanent copy of the playtrace that the player can keep editing also aligns with the "Save Early, Save Often" and "Slow it Down" patterns of APX. The player can take as many iterations as they would like to edit the playtrace. This allows the player to engage with the game on their own pace instead of being locked to the speed the game demands. Furthermore, even if they have to close SMES and come back later, the text file will be there for them to pick up where they left off.

Because SMES can not know what exact frame each button was pressed, instead of matching keypresses to frames, we record the timing of each press. We record the timing of whenever a key is pressed down and later released. However, instead of recording at what time these actions of key_down and key_up happen, we record how much time elapses in the middle. This makes keeping track of multiple keypresses and the duration of a specific keypress easier. To be more specific, our representation is a

[4]http://tasvideos.org/Bizhawk/BK2Format.html

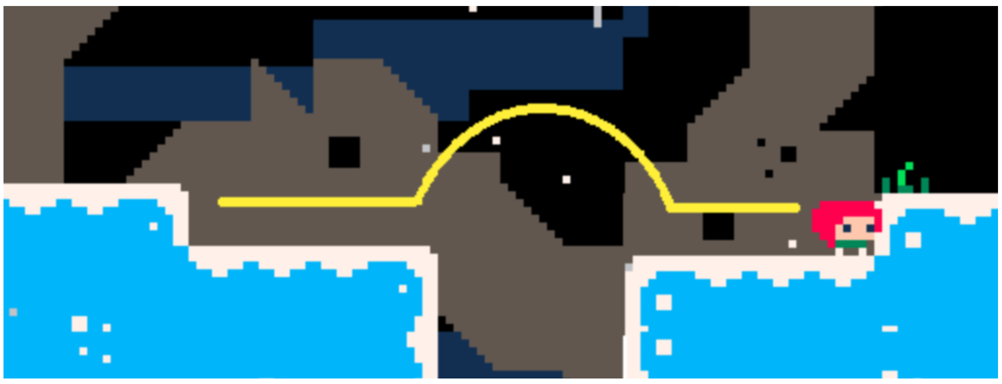


Figure 6.3: The visual path that is described by the given playtrace.

series of two data points, the action_type and the modifier. There are three possible actions, key_down, key_up, or delay, and each action has a modifier associated with it. If the action type is key up or down, then the modifier is the name of the key that is pressed or released. Whereas if the action type is delay, the modifier is how many milliseconds have elapsed since the last entry.

Let us give an example of a playtrace. Imagine a character starts running to the right, then jumps over a gap, and stops running. This can be shown as follows:

```
key_down , d
delay    , 500
key_down , w
delay    , 1000
key_up   , w
delay    , 500
key_up   , d
```

In this example, the player holds down “d” for two seconds, while pressing “w” for a single second in the middle.

6.1.4 Visualizing and Editing Keypresses

After the playtrace is saved into a text file, the SMES GUI loads them and visualizes them. The current GUI is written using the QT framework and is still a work in progress. While designing the visualization we were inspired by digital audio workstation (DAW) software packages. SMES allows the users to manipulate the playtraces by changing when and for how long a key is pressed down. When the desired changes are done the updated playtrace is once again saved into the text document.

This design is aligned with the “Improved Precision” and “Personal Interface” patterns of APX. It is an additional interface that the players can use to engage with dexterity games and by removing the temporal aspect of taking actions the player can be more precise. Depending on motor skills and personal preferences, players may have an easier time with our GUI or with a text editing tool for which they have built deeper fluency (in which case the GUI forms a passive viewer tool).

6.1.5 Replaying the Playtrace

The player can start the replay whenever they want by pressing the replay button. SMES then starts replaying the playtraces that is saved in the intermediate text file. In order to replay the playtrace we use the underlying Win32 API, the same system we used to record the keys. We use the python package winput[5] to send Virtual Key Codes as input to the operating system directly.

Due to the game not running in an emulator and the Win32 API not being

[5]https://pypi.org/project/winput/

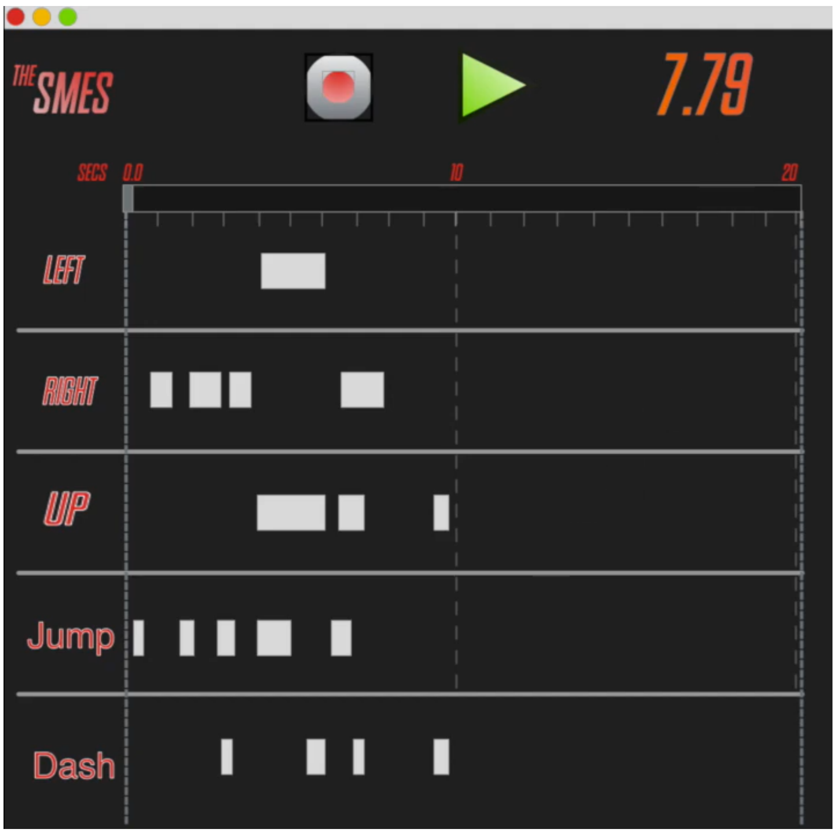


Figure 6.4: The visual mock-up of the editor.

instantaneous the playtrace that is replayed is not completely deterministic. In our experiments there is an 2% average divergence between the length of the input sequence and its realized replay. Because the errors compound in a playtrace this can result in perceivable differences in outcomes across different replays of the same playtrace. This is the main bottleneck that limits the usefulness of the current state of the SMES in supporting longer playtraces or styles of play that rely on precise determinism (e.g. luck manipulation strategies used in TAS). These are not critical limitations, however, because we imagine SMES being applied to localized scenes (which can be re-attempted many times) in games that were otherwise completable within the limits of human dexterity.

6.2 Future Work: Manipulating Playtraces with AI

As mentioned earlier, there is currently no AI system acting on the saved playtraces. However, other work shows that by adding a simple level state awareness reusing parts of play traces can be very useful. For an example from the games industry, in the fighting game Killer Instinct, the enemy shadow AI learns how to play by memorizing sequences of player actions [222]. Then the AI agent replays parts of the player's action sequences when a correct game state is matched. This method proves effective enough to be deployed in a live commercial game, and it has been met with approval from the player base[6].

In the future, we are interested in taking a co-creator approach to supporting playtraces similar to the reinforcement learning based mixed-initiative level creator Morai Maker [144] or the work on user empowerment where a reinforcement learning agent is trained to augment (rather than replace) player input [108].

We imagine that the edits that the user makes to the timeline are not directly applied but are instead taken as training data points to tune a stateful and observation-dependent action policy. In this setup, the player is still making deliberate choices about how to act in specific moments, but the local choices they make can have impact on several other moments (e.g. those that arise in the future or as the result of different

[6]https://forums.ultra-combo.com/t/shadows-ai-retrospective-discussion/24899

stochastic elements of the game).

Furthermore, we believe that the datasets describing how a player iteratively refines their playtrace can be interesting for future applications.

Conclusion

In this chapter we presented the work-in progress Sharp Multi-input Editing Tool (SMES) which allows the players of difficult platformer games to change the challenge type from a dexterity based execution challenge to a cerebral planning challenge. SMES does this by recording the actions of a player, and then presents it to them in a visual interface where they can make changes and replay the updated playtraces. We also discussed how playtraces can be operated on via AI systems with the goal of assisting and empowering players. We hope that with SMES we can contribute to the ongoing conversation of how we can use AI techniques to assist our player.

Chapter 7

Repurposing the Game-Playing AI Surplus for Inclusivity

Large portions of this chapter was published in the Artificial Intelligence and Interactive Digital Entertainment Conference 2020 under the paper title *"Your Buddy, the Grandmaster: Repurposing the Game-Playing AI Surplus for Inclusivity"* [49] which was co-authored by my undergraduate mentees Xueer Zhu, Eric Hu and my advisor Adam M. Smith

The previous chapters of this dissertation expanded our understanding of failure in videogames, and introduced two computational prototypes that used simple systems to assist the players. Now we turn our attention to exploring how we can use gameplaying AI (GPAI) agents to do the same.

Since the mid-2010s, there have been several high profile victories for game-playing AI techniques. In the videogames domain, the research community made progress towards overcoming several difficult problems: an immense branching factor in Go [281]; multiagent communication in DOTA2 [60]; working with hidden information in poke [66]; parsing the screen with Atari [53]; and balancing low-level control with high-level decision making in Starcraft 2 [309]. In all these projects, the AI agents learned how to play the game very well, often even beating the best human opponents they faced. Unfortunately, however, this increase in the game-playing competency of the AI agents has not directly improved the game-playing experience of human players.

We classify game-playing AI techniques to be any function that takes in a game state and maps it to an action for the agent to take. For the purposes of this chapter, we are not interested in the methods used to do the mapping; rather we will focus on how this mapping can assist the player.

The book *AI in Games* describes three high-level use cases for academic AI in Games: Gameplaying, Procedural Content Generation (PCG), and Player Modelling [317]. Both PCG and Player Modeling have had wide adoption from the game industry [115, 54]. However, the recent improvements in game-playing, especially of learning-based methods, have not seen a similar level of adoption.

A significant portion of game-playing research is concerned with creating agents that play games optimally [317]. However, this aspiration for optimality is not found in the game industry. Commercial game developers are not interested in creating unbeatable opponents for their players to face; rather, they are concerned with crafting an engaging experience. Therefore, the need for an optimal AI agent as a non-player character (NPC) is often not very high [266]. Most NPC enemies are alive for a very short time before they are removed from the game. Simpler techniques are enough to achieve the desired design goals in this short span of time. Even the opponents that are expected to put up a good fight can usually do so through cheating behind the scenes or through clever game mechanics.

Even when the goal of the AI agent is to be as strong as possible, there is always an underlying aesthetic constraint. The AI agent's actions must fit the context of the game. Most of the recent reinforcement learning or evolutionary methods–although effective in maximizing the given objective–are horrible at being graceful while doing so [190, 194]. Even though continuously jumping might be the optimal way to traverse a map, it is unlikely that the game designers would be happy to see their special operations soldier NPC doing so.

These obstacles in using game-playing techniques primarily arise when we assume, as it is traditionally done, that the AI will drive an opponent behavior to beat the player. However, there are several different ways our game playing-agents can *assist* the player:

- A search-based agent can highlight a good path through the obstacles in a complicated action-platformer, allowing the user to concentrate on timing their button presses to follow the highlighted path.
- A rule-based agent can offer reminders if the player is consistently using the unintended tool for the job, allowing the player to utilize all the affordances the game provides.
- A policy-based agent can take control of microing units in a real-time strategy game, allowing the player to focus their attention on macro-level strategic choices.

- A value-based agent can pinpoint which action reduced the player's chances of winning, allowing them to improve their game-playing skills more efficiently.

Shifting our focus from beating the player to supporting the player has several benefits. Through assisting, optimality becomes a valuable trait since we would like the AI to help the player in the best way possible! Furthermore, depending on the implementation we choose, we can assist the player without the need for a physical avatar, which makes shaping the aesthetic impact of our AI much easier.

Why should we be concerned with providing assistance to our players? Because doing so makes our games more inclusive, allowing more players to experience our games [248]. Every player is different and might have different needs to fully engage with a game [152]. Some players might find a game inaccessible due to its difficulty. Other players might find the same game inaccessible due to design choices such as color schemes and specific input mappings [199]. Offering a portfolio of assistance methods can help us reduce the mismatches between our players and our games. In recent years, games such as the difficult action platformer Celeste [205] have started including assist modes. We propose using AI agents to assist our player as a personalized and effective extension to preexisting assist modes.

Another reason we should use GPAI to help our players is that contextual help can potentially increase player retention, especially in highly complex games [38]. One reason why players stop playing games is that they don't fully grasp how to play the game itself [80]. We can use GPAI to show players how to proceed whenever they are stuck and at risk of abandoning the game. It is already common for players to refer to walkthroughs and wikis when they cannot proceed, and having a system where the players can seek help from the game itself will give more control to the designers.

However, it might not be immediately obvious which aspects of the game a player might need help with. Additionally, it can be difficult to discern what the different assistance methods might look like. To help address these challenges, we present a design exercise to explore the design space of GPAI-driven assistance methods. This exercise can help us categorize a game's challenges and use these challenge categories to ideate different assistance methods. In this chapter, we apply this design exercise to Celeste to generate a variety of different assistance method ideas. We then implement two of them, one responding to the execution challenge and one responding to the planning challenge. The purpose of this implementation is to better understand the implications of using GPAI to assist players by creating a computational caricature [284].

In summary, this chapter contributes:

1. A reframing to utilize game-playing AI techniques in service of player experience and inclusivity,
2. A design exercise to systematically explore the design space of GPAI-driven assistance methods,
3. An implementation of two different assistance methods targeting two different challenges in a clone action platformer of Celeste.

7.1 *Weak Human + Machine + Superior Process*

in 1997, IBM's chess-playing computer, Deep Blue, beat the former World Chess Champion Gary Kasparov [74]. Kasparov's reaction to this defeat was very forward-facing. The following year, in 1998, he announced Advanced Chess [184], a fresh take on chess where each human player uses a computer chess program to explore the possible results of candidate moves [95]. Several years later, in 2005, a freestyle chess competition was organized, and any AI, human, or centaur team could compete. Surprisingly, the winner was not a grandmaster backed up by a supercomputer; rather, the winners were a pair of amateur chess players who used three ordinary desktop computers. What had allowed them to succeed was not their individual chess knowledge, nor the amount of computation they had, but rather how the pair had utilized the strengths of their AI [77]. After analyzing this victory, Kasparov reached the following conclusion: "Weak human plus machine plus better process was superior to a strong computer alone and, more remarkably, superior to a strong human plus machine plus inferior process" [184]. This project attempts to contribute one such process in how we can utilize game-playing AIs to support player experience.

Some recent games have started including assist modes to let their players adjust game challenges. The assistance mode in Celeste is one of the most comprehensive ones and has garnered a lot of praise [187]. With it, the player can tweak almost every aspect of the game's difficulty: game speed, total climbing stamina, the number of allowed dashes, and invulnerability are among the many options.

The inclusion of assist mode in Celeste, combined with the release of other difficult games without any sort of assistance, such as Cuphead and Sekiro, sparked a series of discussions surrounding the design intent and the value of difficulty in games [302, 191]. Challenge in many games is essential for the design and is shown empirically to increase enjoyability [93, 243]. Assist modes do not aim to make games easier; rather, they make games more accessible. Increasing the player's options decreases the likelihood of a mismatch between the game and the player's capabilities. While we have come a long way in improving game accessibility [121], there is still room for improvement.

Outside of accessibility, there are other reasons why players use techniques that seem to make the games "easier." For example, they can change the main mode of gameplay by using an emulator for Tool Assisted Speedruns (TAS) or by using trainers [87].

Specific techniques can also be used to extract additional information from a game to help make analysis easier. One such tool is Bob's Buddy [271] for the card game Hearthstone. In the Battlegrounds mode, this add-on, given a boardstate, calculates the odds of the player winning, losing, or tying and is often used by the players to improve their play.

Tools such as Bob's Buddy show that players are already creating tools to help them engage with games on a deeper level. Additionally, systems such as tool-assisted speedrun emulators show alternative ways to play. The increasing prevalence of assist modes, along with the positive reactions to them, show that making systems more accessible is beneficial in several different ways. In the following section, we will be describing our attempt at a "better process," to show how we can repurpose GPAI techniques to assist our players.

7.2 Design Exercise for Discovering Assistance Methods

Although it is easy to state that we should utilize GPAI agents to help players, it is relatively difficult to formulate the specifics. First, we must decide on what way the player will be assisted. Then, we must decide on the magnitude of the assistance. Most games offer a variety of challenge types, and not every player will need support in every challenge. Furthermore, while some players would prefer a bit of assistance, others might benefit from a more comprehensive solution. In this section, we describe a design exercise that facilitates the formation of a portfolio of assistance methods that vary in both assistance type and magnitude.

This assistance method ideation exercise has three steps:

1. Identify the different difficulty types of our game,
2. Construct the design space of assistance methods,
3. Explore this space to formulate a portfolio of methods.

7.2.1 Identifying the Challenge Types

The first step is to decide on the primary challenges the player must overcome throughout the game. Challenges can be deemed primary either through simple reasoning or by using a pre-existing framework. We suggest using the Taxonomy of Failure [50] (ToF) to identify the challenge categories in a game. Subscribing to a preexisting framework makes this step more systematic, and using the ToF can help discover unexpected challenge categories, especially when it comes to accessibility. However, the user should employ the categorization method that best fits their game.

The ToF proposes a model of how people play videogames, focusing on areas where players can encounter a failure. According to the ToF, there are six classes of failure. Thus, the players could face a challenge in:

- **Encoding Input**: Is the player physically capable of using the game's controls?
- **Decoding Output**: Can the player parse the feedback of the game?
- **Discovering Mechanics**: Does the player know what they can do in the game?
- **Setting Goals**: Does the player know what they should accomplish in the game?
- **Planning**: What steps should the player take to accomplish their goals?
- **Execution**: Was the player successful in following the steps of their plan?

We can analyse the game in question with the ToF to identify the main categories of challenge. In Celeste, figuring out what the hardware button mappings are should not be challenging (encoding input). Parsing the screen and understanding what is being displayed is also not one of the desired challenges (decoding output). Similarly, while playing Celeste, the player has a simple goal: to reach the end of the level with a limited but expressive skill set of jumping, dashing, and climbing. Discovering the goal and uncovering new mechanics are not part of where the challenge lies (discovering mechanics and setting goals). Instead, the designers are interested in challenging “both the mind and fingers” [187]; players must decide on the correct path of each level while also being able to follow that path. The player must constantly plan their way through each level and execute said plan by pressing the correct buttons at the correct timings, meaning the challenges they constantly face are **planning** and **execution**.

7.2.2 Mapping the Design Space

It is important for the assistance methods to vary in the challenge they target–not every player needs help with the same difficulties–and in the magnitude of help they offer–not every player needs the same degree of assistance. Therefore, a design space of possible assistance methods can be parameterized by target assistance type and the assistance magnitude. To construct this design space, we create a graph where each axis represents the increasing magnitude of assistance in one of the chosen difficulty types.

Typically, one starts with a data set and maps it to selected axes to explore how the data points are related. In this exercise, we do the inverse by starting with an empty graph, and working our way back to the data points: We select a point in the constructed graph and formulate an assistance method that would map to the selected point. This procedure helps systematically discover assistance methods that vary in both targeted difficulty and assistance magnitude. A positive side effect is that it also surfaces interesting combinations where both difficulty types are targeted by the assistive method to varying degrees.

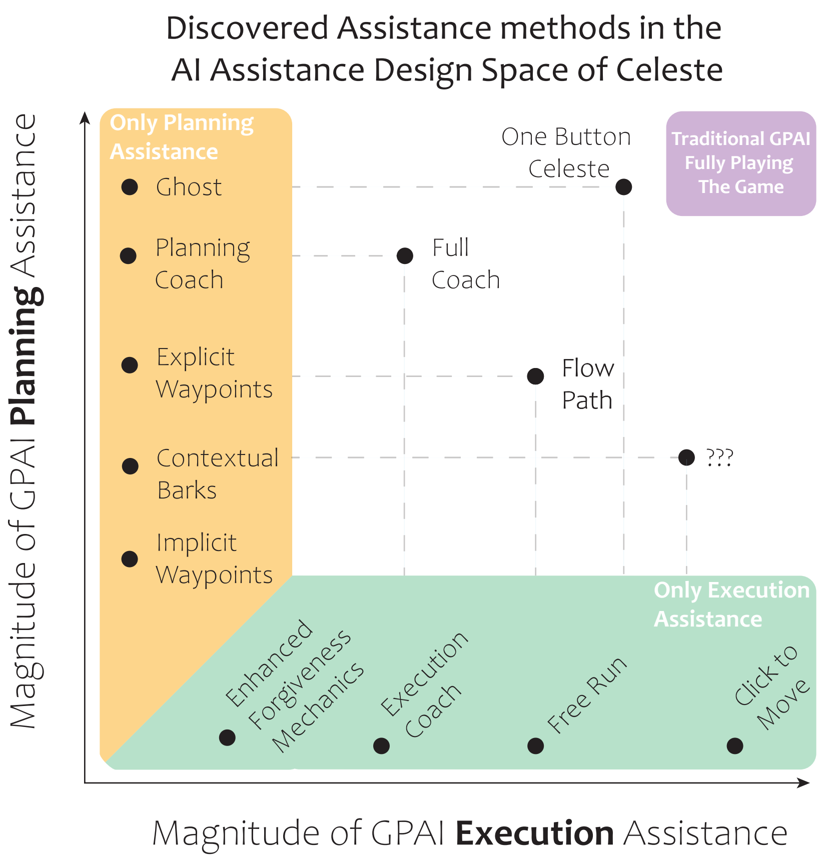

Discovered Assistance methods in the AI Assistance Design Space of Celeste
Magnitude of GPAI **Planning** Assistance
Magnitude of GPAI **Execution** Assistance
Only Planning Assistance
Ghost
Planning Coach
Explicit Waypoints
Contextual Barks
Implicit Waypoints
One Button Celeste
Traditional GPAI Fully Playing The Game
Full Coach
Flow Path
???
Only Execution Assistance
Enhanced Forgiveness Mechanics
Execution Coach
Free Run
Click to Move

Figure 7.1: Each point represents a possible GPAI-driven assistance method discovered by applying the design exercise. The graph is not descriptive–it does not visualize preexisting data points. Rather, it is explorative, for it is used to discover assistance methods that map to points in the design space.

Figure 7.1 shows our attempt at exploring the design space of assistance methods in Celeste. This is not meant to be an exhaustive list; rather, it gives the reader an idea of how we can use the exercise to pick a point in the space and come up with a GPAI-driven assistive method that maps to our selected point. The top right region of the graph represents how traditional GPAI is used: playing the game by fully responding to all of its challenges. The bottom region shows assistance methods that only target the execution challenge, and the left region shows assistance methods that only target the planning challenge. The middle region shows assistance types that target both difficulties.

Below, we give a brief explanation for some of the potential assistance methods we ideated using this design exercise for Celeste. The goal is to showcase the range of assistance methods that can be reached using this exercise, rather than comprehensively specify implementation details.

7.3 Discovered Execution Assistance Methods

Click to Move This assistance style emerged after we asked what it would look like if the AI fully took care of the execution challenge in the game. While this assistance method is active, as its name suggests, the player must solely click wherever they would like their agent to go, and the AI handles the rest. We imagine execution assistance methods to be useful for players who might not have the manual dexterity necessary to execute the complex sequence of actions but still enjoy the planning challenge. This assistance method could also be used by players who are exploring optimal ways to navigate the space.

Free Run Style Assistance In this assistance style, the player still has to move their character in the direction they would like it to go, but the AI takes care of jumping over small obstacles, gaps, and hazards. We were inspired by the freerunning system of the Assassin's Creed games in which the player must direct the character, but the AI system decides which ledge to grab and which corner to step on.

Enhanced Forgiveness Mechanics A lot of platformer games, and among them Celeste itself, implement a series of Forgiveness Mechanics [274]. For example, the characters can jump several frames after they leave a ledge, and if the player presses the jump button before their character hits the ground, the game recognizes the intent of the player to jump, thereby makes the character jump the moment they land. These additions subtly assist the player, and usually are implemented with simple boolean checks. If we have an AI that knows–given a goal–how to get there, we can use this AI system to implement a series of enhanced forgiveness mechanics that span a wider range of assistance.

Execution Coach If we have an agent that has close to optimal play, we can also measure how far the player is from the optimal sequence. Thus, we can use the oracle not to take over the execution aspect of the play, but rather to give the player advice so they can improve their own abilities. This advice would focus on the timing and sequence of actions, and it can also act as a dynamic hint system.

7.4 Discovered Planning Assistance Methods

Ghost The most comprehensive version of the planning assistance is when the AI system directly displays what the optimal path from any given point in the map is without moving the character. Planning assistance methods can be useful when players consistently fail to find the correct path to reach the goal, whether due to a cognitive impairment or otherwise. These methods can also be useful for speedrunners as they attempt to validate the optimal path in a level.

Explicit Way Points In this assistance method, the AI displays points along the optimal path instead of the full sequence of actions. This can be used to show the player which platform to jump from or at what point to dash towards the goal without revealing step-by-step instructions.

Planning Coach If we have an optimal path, it takes just a little more effort–defining a distance metric–to evaluate other paths against it. We can use this capability to give our player tips and tricks as to which path to take. When the AI recognizes that the player is straying too much from the optimal path, or, more likely, when the player asks for help, the AI can annotate at which point the player first left the optimal path.

Implicit Way Points Another way to convey optimal path information is through the environment. Instead of explicitly showing way points, we can use subtle environmental cues. Level designers work hard to guide the player through their levels. Combining implicit way points with effective level design would allow that guidance to be dynamic and adapt to the specifics of the player.

7.5 Combining Assistance Methods

Flow Path In this style, we combine both the explicit waypoints and free run assistance. This combination results in an assistance method in which free run assistance only activates when the player is on the optimal path towards their goal. This method would be similar to a game feature that encourages and rewards players for taking the expected route in lieu of an assistance method.

One Button Celeste In this style, the GPAI agent almost fully takes over playing the game. The path the character takes is fully determined by the AI and–similar to the Free Run execution assistance–the GPAI handles moving through small obstacles and gaps. The assistance can be set up so that the player is only in charge of the jumping capabilities of the character. This means the player can solely engage with the game by using the jump button. This assistance method can help players with motor impairment engage with the game by using switch access control methods.

Ending with One Button Celeste, we described nine assistance methods. Not all of these would be effective when implemented in the game. Instead of specifying one or two robust assistance methods in detail, our goal in this section was to convey that there are a lot of opportunities and diversity in the ways we can repurpose GPAI to assist our players, and show how designers can go through this exercise for the games they are working on. However, in order to start testing the validity of these potential ideas, implementing them in context is necessary.

7.6 Implementing Assistance Methods

Staying in the ideation phase is not sufficient if the goal is to explore the potential design of an assistance method. In this section, we describe how we implemented our game-playing AI for a simplified clone of the game Celeste and used it to prototype different assistance methods. In order to exemplify our solution in this project, we use the Computational Caricature [284] methodology. A computational caricature exaggerates the salient aspects of the game design process while downplaying all other points. A computational caricature has claims (to be quickly recognized and understood) and oversimplifications (to be overlooked). With our following prototype, we:

- **Claim** that game-playing AI methods can be used to assist our player and prototype several different instantiations of assistance to target different challenges.
- **Simplify** level design and performance constraints. More importantly, we oversimplify the means in which the assistance becomes available to the player in addition to the limits of said assistance, both of which are crucial components in the gameplay experience that requires further research.

To put our assistance methods into practice, we built upon the open source code base of André Cardoso's recreation of Celeste in Unity[1]. This implementation uses the default Unity physics engine to move the character. Inspired by [304], we decided to use A* to drive our assistive AI. We parametrized the game space with a seven-dimensional state description: The x and y positions of the character, the x and y velocities of the character, and flags that denote whether the character can jump, dash, and walk.

Within this space, we used the actions walk, jump, and dash to do the search. We used a simple euclidean distance heuristic between the player's character and the goal to guide the search. To run the actual search, we instantiated a separate Unity

[1]https://github.com/mixandjam/Celeste-Movement

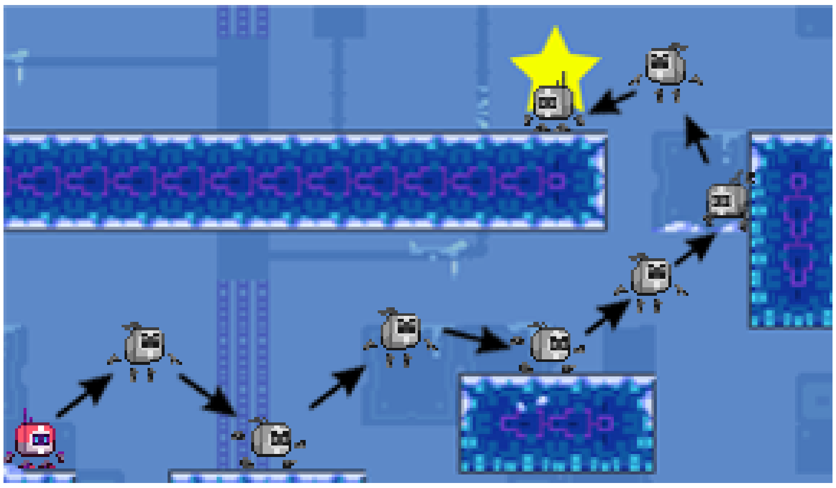


Figure 7.2: Our implementation of the Ghost planning assistance. When a player asks for help, we spawn a copy of the player, and it takes necessary actions to reach the goal. It is up to the player to repeat those actions.

physics scene where we can manually set the physics timestep to be much smaller. We also implemented fuzzy tile matching, which allowed us to do the search more efficiently and within acceptable timeframes. Using this capability, we implemented two of the discovered assistance methods: Ghost to assist with planning challenges and Move to Click to assist with execution challenges.

7.6.1 Ghost Implementation Details

With the Ghost (fig 7.2.) assistance method, the player presses the P key to start the search. When the search is complete, a copy of the character with a grayscale sprite (a.k.a the Ghost) travels directly to the goal. The Ghost shows the optimal path without directly interacting with the player. Thus, the Ghost acts as a guide without taking away player agency.

We realized that Celeste already uses a similar technique in a simplified man-

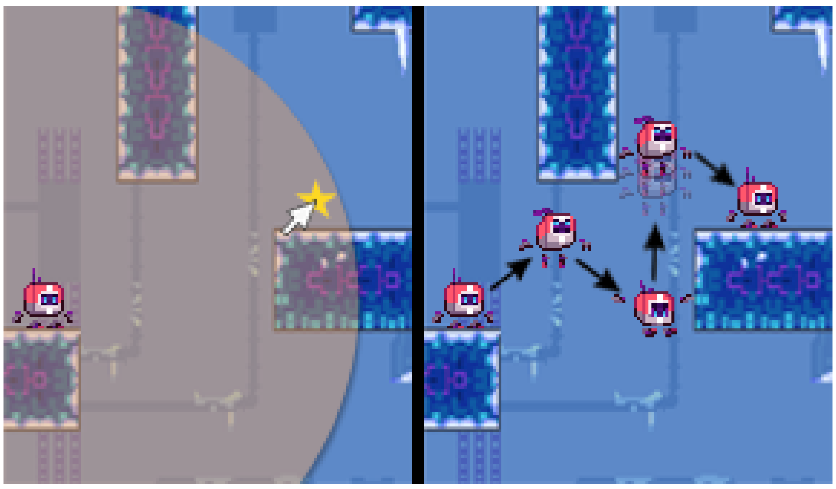


Figure 7.3: Our implementation of the Click to Move execution assistance. The player clicks anywhere within the radius and the AI handles the execution challenge by moving the player character to the target location.

ner. Instead of doing a search from the player's position to the goal, the Ghost in Celeste is simply embedded into the level and keeps looping, showing the player what to do[2]. This visual guide's limitation, however, is its attachment to that specific place in that specific level. For example, if the player takes a break from the game for several months, and cannot remember specific moves when they return, the Ghost embedded in the level would not be helpful.

7.6.2 Click to Move Implementation Details

While implementing the Click to Move assistance method, we decided on a maximum distance of automated movement to be searched. In this implementation we wanted the player to click on the path they would like to take, and as such, we decided to limit the radius where the AI would be active. It is important to note, however, this

[2]https://youtu.be/E1Ox37N1efQ?t=979

is one possible implementation of this assistance mode, and the assistance would have worked without a radius limit as well. In order to use our assistance method, the player clicked within a small radius for the AI to find the path there.

Click to Move created a cycle where the player would click where they would like to move, wait for for the AI to take the required actions, and then find the next position they would like to reach. This cycle resulted in a more staggered play experience. Sometimes the player would try to move up a platform, but the platform would be out of the clickable radius. When the player clicked the closest point possible, usually in the air, the agent would reach that point and fall to the ground before the player could click on the next point.

7.7 Discussion and Future Work

Implementing the assistance methods showed us that before GPAI assistance methods can be claimed as effective, several further research questions need to be answered:

First, simply picking the assistance method is not enough, and much more thought must go into implementation details. While our implementation of the Ghost assistance method seemed to have fulfilled its purpose, the initial design of Click to Move subtracted from the game-playing experience. An iterative process that finetunes the limitations and strengths of the methods is crucial. ***What are the best practices in designing GPAI-driven assistance methods?***

Second, a more comprehensive look into designing the most effective way to introduce these assistance methods is needed. A study conducted by Anderson et al. found that depending on the complexity of the game, adding a help button could harm engagement [38]. It is not entirely clear whether locking these assistance methods behind the options menu is the best option or if there is any way to actively recognize and suggest assistance methods without overstepping boundaries. ***What are the most effective ways of introducing GPAI-driven assistance methods to the player?***

Finally, there still needs to be an empirical evaluation of the effects of these different types of assistance on the player's experience. While assist modes in general have been received positively by those who use them, it is important to test whether the added complexity and diversity of the assistance methods actually translates into enhanced play experiences, and makes games more accessible. ***How effective are specific GPAI-driven assistance methods in improving the gameplay experience?***

Conclusion

In this chapter, we described how repurposing GPAI capabilities to assist the player can enhance the game-playing experience and help make games more inclusive. To assist with the ideation of assistance methods, we proposed a design exercise with two steps: first, we suggest using the Taxonomy of Failure to discover the types of challenges that exist in the game. Second, we show how to use these types of challenges to delineate the design space of assistive methods. We then applied this design exercise to Celeste to generate several assistance methods. We implemented two of the discovered methods, Ghost and Click to Move, to further explore this process. Through our implementation, we discovered several additional research questions that must be answered before GPAI-driven assistance methods can be confirmed as effective. We believe this research direction furthers the discussion on how to utilize GPAI in service of the player experience. We hope crafting our GPAI agents to be supportive buddies to our players, instead of hostile enemies, will contribute to making videogames more inclusive.

Chapter 8

Reinforcement Learning Powered Navigation Assistance

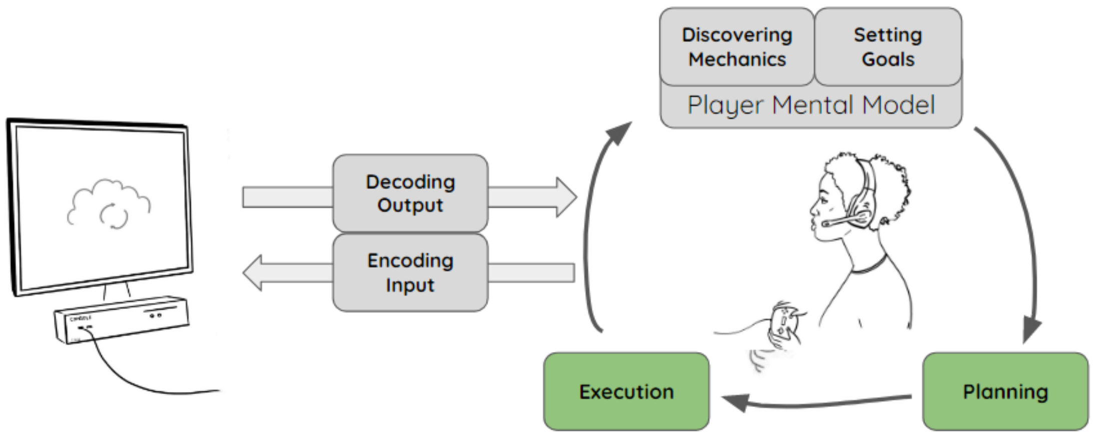


Figure 8.1: RLNav assisting players in both the execution and planning failure classes.

RLNav represents reifying the theories presented in the previous chapter by utilizing traditionally gameplaying focused reinforcement learning techniques for accessibility.

In the previous chapter we explored how, instead of attempting to beat the player, the game playing capabilities of AI agents can be repurposed to support the players to enhance the player experience [49]. In this chapter, we reify the theories discussed in the previous chapter by training a reinforcement learning powered navigation agent.

We claim that using reinforcement learning agents for supporting the player over as non-player character opponents has several benefits. First, when assisting the player in playing the game optimality becomes important. If you are using an AI agent to help someone, you would expect the agent to be really good at what it attempts to do. Currently the capability to defeat world champions in Dota [60] or Starcraft [309] does little to affect the day-to-day experience of majority of players, however, if we start repurposing this competency to support the players this reality might change.

Second, using the agent capabilities for assistance reduces the need for an embodied non-player character avatar to be on the screen which reduces the likelihood of the RL agents' actions disrupting the aesthetics consistency of the videogame [47].

Building on top of this strand of work, this project focuses on helping the players navigate in a complex 3D environment with several movement mechanics such as jumping, double jumping and wall running. We chose navigation as our assistance target because the ability to move intentionally is a core component of many games, and often is a prerequisite for engaging with the rest of the gameplay experience. In this chapter we contribute a playable example of reinforcement learning powered assistance methods.

8.1 Why focus on navigation assistance?

Even though it might come as second nature to those who have played many videogames, navigating in 3D space with multiple movement abilities is a rather difficult task for novices [67, 217, 21]. Most games assume that players are competent in moving around in the created world, and as such, navigation is a prerequisite for accessing many games' contents. Targeting a common aspect of many games also increases the likelihood of the insights to be applied in a variety of different games, compared to focusing assistance in a niche mechanic in a specific game.

Furthermore movement systems, especially those with additional mechanics such as wall jumping, dashing, etc. can have rather high skill ceilings which makes expert play such as speed running possible [270]. So with this one mechanic we can target and assist players on both ends of the skill spectrum–getting novices to start playing and getting experts to push the boundaries.

8.2 What is reinforcement learning?

"Reinforcement learning is learning what to do—how to map situations to actions—so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them." [295] In other words, reinforcement learning is a specific formalism for presenting sequential decision making problems.

This formulation often contains an environment and an agent. The agent is given a set observation every timestep from the environment and is tasked with taking the action that maximizes the cumulative future discounted reward. In our case, the game level is the environment, the character that is navigating within the world is the agent, and the agent is trying to take the action that allow it to reach the navigation target which is the reward.

While there are multiple techniques that can be used to solve reinforcement learning (RL) problems, such as evolutionary algorithms and value iteration look up tables, during the 2010s machine learning boom, RL has almost become synonymous with Deep Reinforcement Learning [123] due to it's proven effectiveness which uses neural networks to parameterize the policies, and gradient descent to train them. In my work I make use of the recent advances in DRL, and use the off-policy Soft Actor Critic algorithm [145].

8.3 Why use reinforcement learning over traditional navigation methods?

While navigation meshes [287] are the industry standard [206] for non-player character navigation they do not scale well with the number of different actions the player can take [44]. In the traditional setup, a designer needs to annotate the map [308] for the different types of movement abilities such as wall-running or double jumping. While these limitations are acceptable when it comes to non-player character navigation, the need of assistance being always available at any point in the map (whether on the ground, climbing stairs, during jumping or while being farther away from a link node) limits the usefulness of navmeshes for assistance methods. Furthermore the fact that RL algorithms share the same action and traversal space as the players also makes it easier to use them in assistance.

Reinforcement learning methods are shown to handle relatively complicated environments and can scale well with multiple actions [36]. Outside of videogames, deep reinforcement learning has a history of being used for navigation problems as well [323], such as navigation using visual input [321], local obstacle avoidance [319], social navigation [294], and multi agent navigation [196]. The success of deep reinforcement learning in these domains suggests that this approach will be effective in videogames as well.

After the training pipeline is set up, RL techniques also reduce the authoring burden on the designers. When a new map needs to be processed, no additional annotation from the designer is necessary compared to manually adding navmesh linking nodes, which can be time consuming [308]. RL techniques can also demonstrate navigation strategies that were not anticipated by the designer. This capability also has benefits that are orthogonal to assistance such as bug-finding and machine playtesting [139].

8.4 Reinforcement Learning Setup

8.4.1 The NavAssist Environment

Even though there are multiple pre-existing reinforcement learning environments, at the time of writing none of them faithfully implement a 3D movement system commonly found in commercial videogames while also being sufficiently easy to use. Because our goal is to create assistance methods that can later be adopted in videogame design practice, it is important that the environment we train our agents on are similar to commercial videogames. Towards this end we based our training environment on the First Person Shooter (FPS) Microgame Template published by Unity.[1]

Using this template as a basis gave us several advantages: First, we can be confident that the character controller implementation we use is a simplified but realistic approximation of the controllers commonly found in other commercial First Person games. Second, the myriad of resources and tutorials available explaining and building on top of the FPS Microgame makes it easier for other researchers to customize the NavAssist project based on their own needs. Furthermore, even though we primarily focus on navigation, the existence of other pre-existing game components such as shooting with different weapons, multiple enemy types, and a variety of implemented modes ensure that the NavAssist project gives us many opportunities to increase the complexity of the assistance methods implementable in the future.

The agent is rewarded for reaching a goal point. At the end of each episode

[1]https://learn.unity.com/project/fps-template

new random positions are sampled for both the agent's starting point and the goal. More specifically, a random position in the 3D space is sampled, which then is used as a starting point for a downward raycast. If the raycast lands on a valid surface, the collision point is used as the spawn for the agent or the goal. This ensures the agents and the goals always spawn on reachable and valid positions. A curriculum that gradually increases the distance between the sampled starting points for the agent and the goals is applied.

On top of the sparse reward for reaching the goal location, an optional dense reward which is given to the agent for moving closer to the goal can be configured. Potential Based Reward Shaping [225] can be enabled for this dense reward. PBRS ensures that, when a dense reward is introduced, the optimal policy for the initial task remains the same–which is to reach the goal as fast as possible, not to simply get close to the goal. This helps the agent in cases where it needs to move away from the goal for a while to reach it.

8.4.2 The *NavAssist* Supported Observations

In each of the environments the agent has access to several configurable observations (sensors), seen in Figure 8.2.

Agent and Goal states: Information about the agent's current state is given including the agent's and the goal's relative position, agent velocity, the number of jumps and air jumps that are remaining, etc. This observation informs the agent of how it is moving and where it should go. Alternate observation encoding schemes are easily

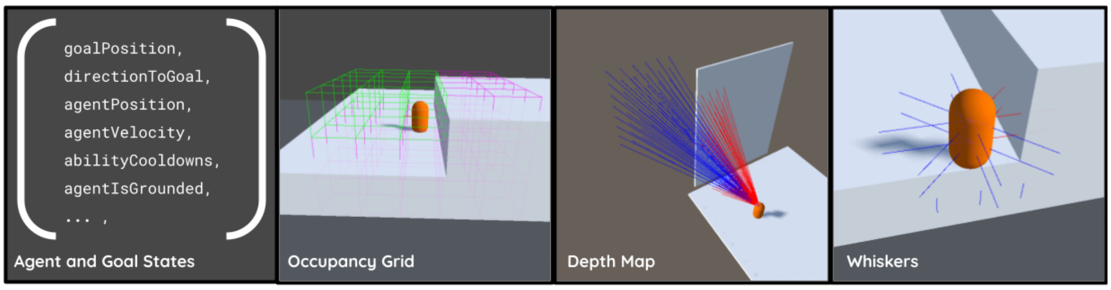


Figure 8.2: Our agents sense the environment via customizable observations.

configured.

3D Occupancy Grid observation: In an offline step we discretize the space into cubes and store whether the cube is occupied by level geometry or not. During play we collect the occupancy of nearby grid cells as an additional observation representing the local structure of the world. The grid resolution and sampling pattern are customizable.

2D Depthmap observation: We cast several rays forward from the agent position and return whether the raycasts hit, and if so, how far away the hit is. This observation informs the agent about the obstacles facing direction. The resolution and maximum depth of this sensor are customizable.

Whiskers observation: We cast a small number of short rays that surround the agent and return whether the ray hit an obstacle, and if so how far the obstacle is. This observation informs the agent of its immediate surroundings, and the edges of the platforms it is on. The number, direction, and depth of these raycasts is configurable.

The default environment baselines use all the observations described above, and the default configurations have been set empirically. Each of these observations are implemented by inheriting from a base observation class which makes it easier for new types of sensors to be implemented and open sourced in the Unity project. We hope that open sourcing this project will make it easier for other researchers to investigate more sophisticated representations of the world (particularly those that lead to better human assistance rather than just improved agent performance).

8.4.3 The *NavAssist* Supported Actions

The user can choose among two continuous action spaces, one that uses agent rotation, and one that does not. Both action spaces share common actions for moving in the x axis, moving in the z axis, and jumping. The continuous jump action outputted by the network is discretized to be a binary action by using a threshold in the environment. These map to the common use of WASD keys for strafe movement, and Spacebar for jumping actions that the human player takes. The action space that uses rotation adds two additional action outputs that control the agent direction in the X and Y axes. These additional actions map to common use of mouse movement for camera controls.

8.4.4 Learning Setup

For our reinforcement learning baselines we decided to use Soft-Actor Critic [145], following the procedure described by Alonso [36]. In the baselines shown we are using the stable-baselines3 library for our algorithm implementation. These baselines are not attempts at state-of-the-art results but instead represent the performance of applying a common reinforcement learning algorithm without any algorithmic improvements, curricula or non-default network architectures.

The network architecture used in the baselines is kept simple. The observation input vector is passed into the network that consists of two fully connected layers of 2048 hidden units each. We experimented with the DenseNet [157] architecture as per [237] however empirically we did not observe any improved performance for using the DenseNet architecture and decided to keep using the simpler multilayer perceptron architecture.

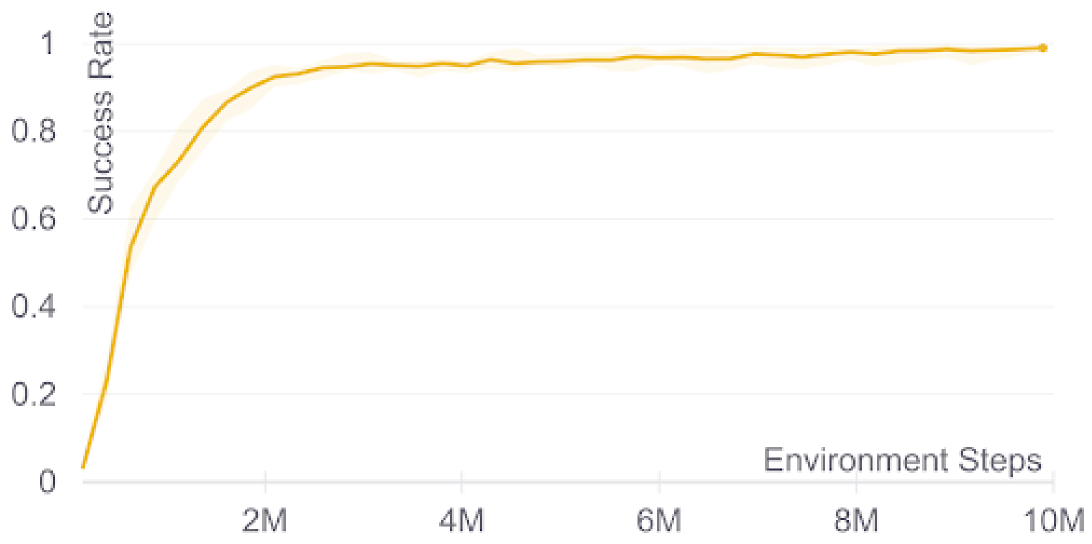


Figure 8.3: The training curve for the navigation environment. The shaded area shows the standard deviation over three different seeds.

We present more details of the training procedure in the Appendix. The presented information includes a comparison between using the default hyperparameters versus fine-tuning them; an ablation study of different observation types and algorithmic improvements; and the success rate changes with network size which we found to be crucial.

8.5 Example Assistance Methods

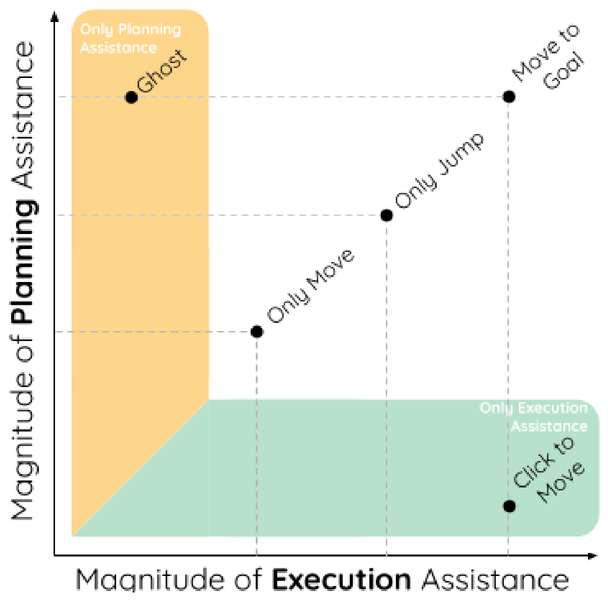

Only Planning Assistance
Ghost
Move to Goal
Only Jump
Only Move
Only Execution Assistance
Click to Move
Magnitude of Planning Assistance
Magnitude of Execution Assistance

Figure 8.4: Our five example assistance methods support planning and execution in different ways.

We also offer five different example assistance methods. The ideas for these example assistance methods were generated using the design exercise suggested by Aytemiz et al. [49] as follows: The game is broken down into the different challenging aspects. Responding to a diagnostic taxonomy of failure in videogames [50], we define a *planning* challenge, deciding which route to take to reach the goal, and an *execution* challenge, the ability to actually press the buttons correctly to follow that route. Then we can explore how to assist the players using our navigation in each of the challenges.

Move To Goal: The agent assists the player in both challenge types by taking full control over the player character and navigating towards the goal. This could be of interest to players who prefer to simply skip certain playing parts of the game without missing out on seeing the kind of play that it would have required.

Ghost: The agent assists the player in the *planning* challenge by taking control of a separate invisible ghost agent that navigates to the goal and leaves a visible trail for the player to follow. The player now knows a viable path to the goal but it is up to the player to follow that path. This could be of interest to players who are stuck and do not know which path to take to the goal.

Click to Move: The agent assists the player in the *execution* challenge by taking control of the player character and navigating towards a point clicked by the player. The player has indicated what route to take by clicking but the agent handles the movement along the indicated path. This could be of interest to players who are interested in self-directed navigation of the world, but they do not want or lack the a capability to execute the complex actions necessary.

Only Jump and Only Move: The agent assists the player in both the execution and planning challenges to a varying degree. In these assistance methods the agent takes control of some aspect of the movement, moving or jumping respectively. In Only Move, the player only has to move towards the goal, and the agent will jump at the correct moments. In the Only Jump assistance, the agent will automatically move towards the goal and the player instead has to time the jumps. This might be of interest to players who have a limited input space when it comes to playing the games, such as people with impaired mobility using sip and puff [215] input systems, but who still want some of the challenge of fine-grained controls.

8.6 System Evaluation

While the post training success rate is an important indicator of how effective the agent would be in assistance it does not tell us the whole story. One main problem is that during training (and final testing) the 1000 sampled goal/agent positions are sampled uniformly over the map. This does not reflect the average play pattern–our players do not randomly move around, they have more specific goals.

Another problem is that when it comes to assistance, some failures are more costly than others. In our environment there are two ways in which an agent can fail, it can either time out by not reaching the goal in the allocated time, or it can fall down the chasm, restarting the level. In the context of assistance the latter case is much more harmful. If the player sees that the agent is stuck repeating the same action again and again they can simply take over and continue play. While definitely not the intended result the impact of this failure on the player is relatively small. On the other hand if the assistance method causes a level reset, not only the agent has not accomplished its given task, it has also affected the progress of the player up to this point.

To tackle these two problems we created a more representative test suite. We placed five goals in hard to reach locations we thought would be analogous to where an ingame objective would be. Then, starting from the same initial position, we recorded ourselves reaching each of the goals respectively. We created a dataset of 943

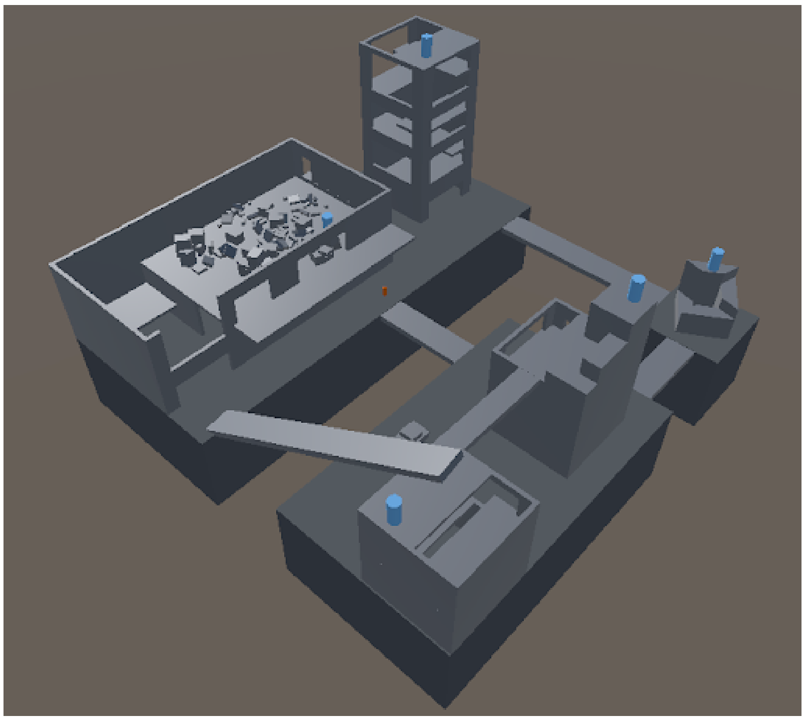


Figure 8.5: Blue columns represent the five goal positions, and the orange capsule is the agent starting position.

Success Rate	Falling Failure Percentage	Timeout Failure Percentage
0.986	23%	77%

Table 8.1: Success rate of our best performing model on the 943 trajectories.

start/end pairs that are representative of where the player could be while navigating to a reasonable goal.

We spawned our agent at each of those 943 points and measured its success rate in reaching the goal of the respective trace. We ran through this dataset 3 times as the model actions are not deterministic and the success rate changes slightly across runs.

Table 8.1 shows the success rate, and the two different possible failure types. Falling failure represents the agent failing to execute a jump and falling down the world, whereas the Timeout failure represents the agent getting stuck and timing out.

By simply implementing these methods and engaging with them ourselves certain design problems in each implementation immediately becomes obvious. For example taking control of the player avatar and camera in Full Gameplay assistance can cause disorientation and motion sickness, causing more harm than good. Sometimes it is challenging to see where the ghost assistance is headed if the target is immediately behind the player. Click to Move assumes the player is able to click on a surface, this makes it very difficult to jump on to the platforms if the surface cannot be seen from below. In Only Move and Only Jump assistance methods, the model is not trained with the human model, and as such, expects the human to act like itself. This is not a valid assumption and can cause the model to expect actions from the player that they won't take in order to progress toward the goal.

While we can show that the technical system will be effective in helping the player, thorough user studies are necessary to uncover the full effects of introducing these assistance methods and our proposed study can be found in Appendix B. Furthermore, we only scratched the surface of the possible assistance methods, and applicable reinforcement learning (or other AI) techniques.

To support a larger body of interdisciplinary researchers to participate in the conducting of AI driven inclusivity research we decided to open source our codebase, experiment baselines, and models as the NavAssist research pipeline.

8.7 NavAssist Research Pipeline

In order for AI-driven assistance methods to be implemented in commercial games and assist the players effectively we need:

- AI researchers to create competent AI agents that can be trained within reasonable timescales,
- Game designers to come up with assistance methods that do not undermine the interesting challenges of gameplay,
- Human Computer Interaction researchers to ensure the assistance methods are contextualized properly and evaluated thoroughly.

Without any one of these different perspectives, it becomes difficult to effectively pursue this line of research end-to-end. If a game designer wants to design AI driven assistance methods they need the AI models to be already developed, if a human-computer interaction (HCI) researcher wants to test different ways of introducing these methods they need the AI assistance methods to be in place, and if an AI researcher is interested seeing their work utilized for assistance they need perspectives of both the game designers and HCI researchers.

To bridge this gap between different disciplinary groups we are contributing the open source *NavAssist* research pipeline (see Figure 8.6) which is a set of components that make it easier to conduct end-to-end AI-driven assistance research. The *NavAssist* research pipeline consists of:

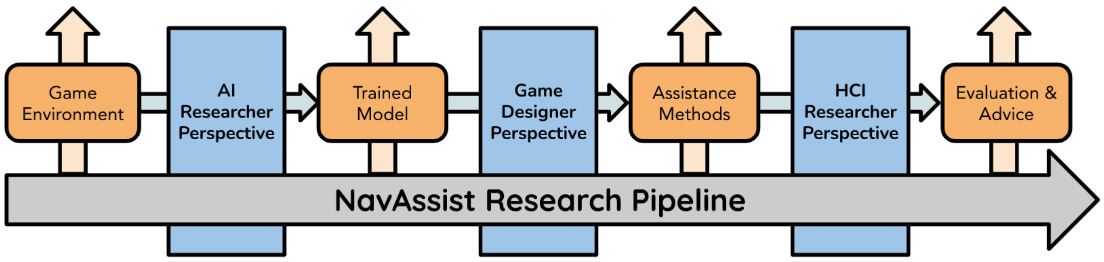

Game Environment
AI Researcher Perspective
Trained Model
Game Designer Perspective
Assistance Methods
HCI Researcher Perspective
Evaluation & Advice
NavAssist Research Pipeline

Figure 8.6: The *NavAssist* research pipeline offers multiple open-source components (orange) that make it easier for researchers with different perspectives (blue) to demonstrate impact with fully-realized player assistance systems.

- A Unity project for a human-playable first person platformer game with complex movement abilities,
- Several levels designs of this game exported in an Open AI Gym environment format,
- Pre-trained reinforcement learning models (policies) capable of navigating in these game levels,
- The learning curves of these trainings experiments for baseline comparisons,
- Implementation of several assistance methods that showcase underlying HCI challenges,
- A human playable game build with the implemented assistance methods.

With the *NavAssist* project a researcher can choose to focus their attention on improving the aspect of AI-driven assistance research in which they personally are invested, while using the provided components for other aspects. With this chapter, our hope is to contribute to the foundations for shared progress and to make it easier for our community to make progress in AI driven accessibility research. All the referred pieces can be found in this Github repository with accompanying notebooks: https://github.com/batu/NavAssist/

8.7.1 The *NavAssist* RL Environments

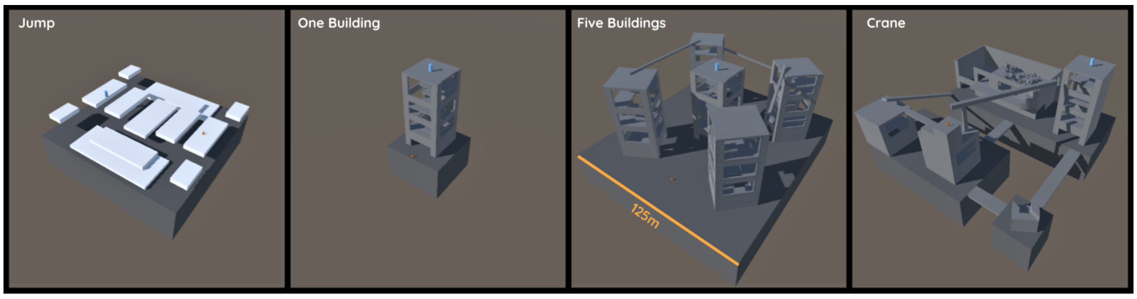


Figure 8.7: Four of our example navigation environments.

Using the Unity project we contribute several different environments with increasing complexities (four are shown in Figure 8.7). While the number of environments in the repository is more than five and steadily increasing, we describe five environment that are representative.

Debug environment contains no obstacles and is used to quickly test learning algorithms. To solve this environment, it is sufficient for the agent to learn to walk towards the goal.

Jump environment contains floating platforms that the agent must jump over. Platforms on the corners of the map are inaccessible from jumps from the ground, but they can be reached by from lower platforms nearby. In order to solve this environment, agents must learn routes more complex than straight lines and to use their double-jump effectively.

One Building environment contains a 5-story tall building that the agent must climb. Each story requires either a jump or a double-jump to ascend. In order to reach the top, the agent needs to jump out of the building through the window and then do a double-jump back to reach the roof.

Five Buildings environment contains five tall buildings (shifted replicas of the building in the previous environment). To traverse this environment effectively, the agent must learn how to jump between buildings.

Crane environment models a 150 meter by 150 meter first person shooter style game map with several buildings, bridges to cross chasms. The Crane environment extends the Urban environment by adding a (static) crane that connects one side of the map to the other. The crane presents an important challenge as it is unlikely for the agents to collect many states from there by following a random exploration. To solve this environment, the agent has to learn routes to climb to points of interest while also being able to reliably execute jumps with precision.

8.8 Nav Assist Pipeline Baselines

For our reinforcement learning baselines we decided to use Soft-Actor Critic [145], following the procedure described by Alonso [36]. In the baselines shown we are using

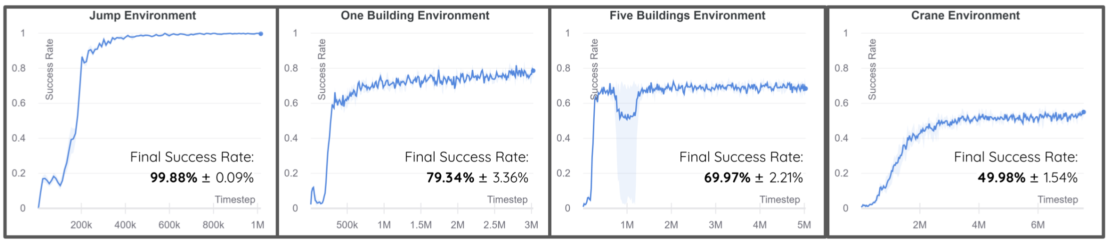


Figure 8.8: Results of the Soft-Actor Critic algorithm on each of the environments confirms their increasing difficulty. These baselines are offered as starting points for exploration of algorithmic improvements, curricula, advanced network architectures, etc. and do not represent the state of the art results on these environments.

the `stable-baselines3` [252] library for our algorithm implementation. These baselines are not attempts at state-of-the-art results but instead represent the performance of applying a common reinforcement learning algorithm without any algorithmic improvements, curricula or non-default network architectures. The hyperparameters were tuned on the One Building environment and kept consistent across all environments. The provided repository also includes results for our best performing navigation agents.

We use one million, three million, five million and seven million five hundred thousand timesteps for the Jump, One Building, Five Buildings, and Crane environments respectively. Results are shown in Figure 8.8. All of the environments use the same observation and action spaces, which includes all the earlier described observations and no rotational actions.

8.9 Questions Addressable with *NavAssist*

We hope that this pipeline will support the shared foundations of AI driven assistance research, and in this section we describe several open research questions that we believe the *NavAssist* research pipeline can be useful in answering.

8.9.1 Open Questions for Reinforcement Learning

The current training pipeline forces the reinforcement learning agent to memorize the level geometry in the neural network weights. Being able to generalize (or use transfer learning) to new maps would increase the effectiveness of RL methods drastically. **How can reinforcement learning agents generalize or transfer to new maps?**

Many games contain different playable characters with different movement characteristics. The common approach currently is to train a new model per movement style which, when combined with the need to train a model for each level, does not scale very effectively. **How can reinforcement learning agents generalize to different characters with differing movement characteristics?**

It is common for trained agents to behave in an unhuman-like manner, which has the potential to break immersion in certain cases. **How can the agents learn to navigate more like a human?**

The observations (the depth map, occupancy grid, and whiskers) used to represent the level geometry are all coarse estimations of the actual environment and can fail in specific configurations. **How can we more effectively represent level geometry to a higher level of fidelity without exploding complexity?**

It is common for games to be tested extensively both during development and also after release. **How can we utilize the human demonstrations available from**

player and quality assurance testing?

Most deep reinforcement approaches use RL to handle every aspect of gameplay. While potentially very effective, this makes it difficult for designers to tune specific aspects of the behavior. **How can we modularly integrate RL into the rest of the game AI stack to ensure the overall behavior is manipulable by the designers?**

8.9.2 Open Questions for Game Design

The provided example assistance implementations are far from an exhaustive list of what is possible in AI-driven navigation assistance. **What are other, potentially more effective AI driven assistance methods?**

Even though in this chapter we assumed the lens of assistance, these capabilities can also potentially become a part of the default gameplay. **How can we use the navigation capabilities to uncover new gameplay opportunities?**

We demonstrated ways of using RL-trained *policies* to assist the player, but many RL methods also trained *value* functions that assign scores to states. **How can trained value functions be used to assemble new assist modes?**

In this chapter we have focused on assisting the player, however the same capabilities can also be used in assisting the game designers, for example when it comes to ensuring all the items placed on a map are reachable. So, generally, **how can we use navigation AI to support game designers?**

Rules that allow the game to respond to the player's intent versus the actual player input are called forgiveness mechanics [130]. One such mechanic for example, is called “coyote time,” and it allows the player to be able to jump for a few frames after they leave a platform. These forgiveness mechanics often result in a more responsive gameplay. **How can we utilize navigation based assistance methods to make the games more responsive?**

8.9.3 Open questions for Human-Computer Interaction

It is imperative that we quantify how much better, if at all, are the AI-driven assistance methods over assistance methods that do not use any AI techniques. **How effective are the assistance methods in actually assisting the players and improving the overall play experience?**

One of the issues that became apparent when assisting is that the player does not exactly know what the AI model is going to do and this reduces the sense of agency which is crucial to maintain when assisting. **How can we ensure that the assistance mode communicates what is going to happen and ensures the player feels in control when the assistance is active?**

We discussed several usability issues that arose in our current implemented examples, such as not being able to target the top of platforms with the Click to Move assistance method. To ensure the methods can be used as intended we would need to know **what some common usability issues that arise are when implementing assistance methods?**

One danger when it comes to offering assistance methods is being condescending to the player. It is important that the player is made aware of the existence of these methods, yet this introduction does not feel patronizing. Towards that end we would need to explore **the best practices in introducing the assistance methods.**

Conclusion

In this chapter we have introduced the *NavAssist* research pipeline to support end-to-end research for AI-driven navigation assistance methods. We described the three main components of the research pipeline:

- The reinforcement learning environments and training baselines for researchers to improve upon,
- A set of pre-trained navigation models for designers to use in creating new assistance methods,
- A set of example AI driven assistance methods to work as inspiration and basis of evaluation.

We hope these building blocks will help our community make progress in this research area. We are looking forward to integrating the requests and ideas of the larger community to the repository in terms of new environments, movement abilities, observations assistance methods and other components.

Chapter 9

Future Work

The projects I presented only scratch the surface of how to utilize artificial intelligence techniques for making videogames more inclusive—plenty of work remains. In this chapter I describe five research directions that can be built on top of the ground covered by this dissertation.

9.1 Explore the Effectiveness of Assist Modes

At the time of writing, even with the growing body of game accessibility research [9, 58, 192, 73], there is still much we do not know about assist modes implemented in commercial videogames. We do not yet know how often they are used by the players, how effective they are in making games more inclusive, or what the best practices are when designing and presenting different accessibility options.

Understanding which accessibility options are most effective and sought after by players can guide future design decisions in the games industry. Understanding the impact of the baseline assist modes is also important when it comes to quantifying the effectiveness of AI-driven assist modes. A proposed user study can be found in Appendix B.

Accessibility modes usually change the abilities of the player, such as making the character invincible in *Celeste* [205], but sometimes they can also change how punishing the game is, such as resetting Mario to the last point he jumped off when he misses a platform in *Super Mario Odyssey* [226]. While the Taxonomy of Failure [50] helps us understand the failure of players, we lack a comprehensive exploration of how the game reacts after a player fails. A deeper understanding of the responses a game can give to player failure—a taxonomy of punishment in videogames, if you will—can assist designers intent on coming up with a more diverse set of accessibility options.

9.2 AI-Driven Coaches

As argued in this dissertation, I believe another approach to making games more inclusive is to ensure they are more effective teachers. I am very much interested in utilizing the gameplaying competencies of AI agents to coach players [305].

As argued in Chapter 5 most games frontload their tutorialization and teach the rules of the game without teaching players how to master the game. While this approach can be sufficient in simpler games, in more complex genres, it leaves groups of players behind. For example, many struggle to learn the multiple subsystems of grand strategy games or understand the intricacies of positioning in fighting games. An AI coach can help identify the aspects the players is lacking in and give precise feedback on how to improve.

Imagine a sports game where after each match, the player can receive an analysis of the blunders that lead them to lose points, and the alternative options they could have taken. This type of feedback could potentially help people improve their gameplay at a much steadier rate, reducing the frustrations of getting stuck in a skill plateau. Any type of function that maps a given state to its utility could be used for this purpose. Conveniently, the critic/value networks [295] in any actor-critic reinforcement learning algorithm does exactly this, and one can get it for free when training an RL model.

9.3 Learning Sciences for Games

Using games for learning is a well studied topic. [249] Educators have tried to understand game design to make learning more effective. I believe we can look into learning sciences to make games more effective.

While "fun is a response to learning" is a rather popular theory in the games industry [189], not a lot of effort has been put into bringing the best practices of the science of learning into game design. Concepts such as, zone of proximal development [105], spaced repetition [43], worked-out examples [257], expertise reversal effect [179], and mastery learning [64] are thoroughly studied in learning sciences and can be integrated into and evaluated in the videogame entertainment context.

For example, the trained agents in the RLNav project can be used to show the players effective ways to traverse the level acting as a worked out example. The knowledge representation of player skill in Talin can be enhanced by incorporating the times in which the players are supported with a skill under a spaced repetition regiment.

9.4 Repurpose Gameplaying AI Agents for Design Insights

This dissertation explores using gameplaying AI agents for game inclusivity to support players. What if we use the same gameplaying capability to support videogame designers as well? Previous work has attempted to address some of these questions [153, 322, 223, 279], and the intent here isn't to suggest an alternative to playtesting with real humans, but rather, using the gameplaying capabilities to distill insights that can inform design decisions.

For example, can we utilize a gameplaying agent to ensure all the items placed on a map are reachable? With a navigation agent can we measure the parts of a game where an important landmark visible? Or, can we use the gameplaying agents to understand the balance of our games? As our ability to train and deploy agents mature, our approaches regarding how to utilize them need to mature alongside them.

9.5 Lean on the Domain Specificity of Videogames

Reinforcement learning research is often conducted with an eye towards *real life*, and videogames are seen as stopgap. This means that improvements and design decisions that can only work in videogames domain aren't celebrated. However, if we are going to be staying in the videogames domain, then there is no reason to ignore videogame specific aspects we can take advantage of.

For example, reinforcement learning agents usually have a rather naive way of perceiving the topology of the world they inhabit—through a number of raycasts. However, if we acknowledge that we are going to be staying in the world of a videogame, then there is no reason not to use the underlying structure that organizes the game's world. In many modern engines, game worlds are represented by scene graphs [62]. We can feed this scene graph directly to our reinforcement learning agents, parsed via graph convolutions [263], in addition to them learning the world through other less structured observations such as raycasts.

Overall, I hope that this dissertation generates more questions than it answers.

Chapter 10

Conclusion

My mother told me that she didn't like playing videogames. She explained how as I was growing up she wanted to play videogames with me, but it never worked out: "When I first tried to play a game with you, I couldn't understand what to do, the game made me feel incompetent, and I felt ashamed."

Videogames made my mother feel helpless and made her decide never to engage with them. Not only she was excluded from the enchanting stories and vibrant worlds of these games, but also from the potential conversations and shared experiences with me. In this dissertation I argued that we should focus on making games more inclusive, not for the sake of game but for the sake of people who play them.

To this end, I set out to discover AI-powered approaches to making games accessible experiences, and more effective teachers. Of course, this goal cannot be achieved through a single body of work. This dissertation, however, marks a number of both theoretical and practical advances towards achieving this goal:

- *A Diagnostic Taxonomy of Failure* project expands our game design vocabulary and contributes a finer grain understanding of failure in videogames.
- *Talin: A Framework for Dynamic Tutorials Based on the Skill Atoms Theory* project recognizes that not every player learns the same way and contributes a to making tutorials more responsive.
- *SMES: Adapting Dexterity-Based Games For Deliberative Play* project acknowledges that there can be many ways of engaging with videogames and contributes a method of converting execution failures to planning failures.
- *Repurposing the Game-Playing AI Surplus for Inclusivity* project reframes our preexisting capabilities in service of the player experience and contributes a design exercise for the generation of assistance methods.
- *Reinforcement Learning Powered Assistance Methods* project operationalizes the proposed theories and contributes several navigation assistance methods.
- *NavAssist Research* pipeline contributes a common platform where interdisciplinary research on reinforcement learning driven accessibility modes can be conducted.

I hope with this work, I contributed to people feeling more welcome in the digital worlds we create.

Bibliography

[1] The designer's notebook: Difficulty modes and dynamic difficulty adjustment. https://www.gamasutra.com/view/feature/132061/the_designers_notebook_.php. Accessed: 2020-6-9.

[2] Designing doom eternal with hugo martin - noclip podcast 22. https://www.youtube.com/watch?v=qBHIalb01ewfeature=youtu.bet=1545.

[3] Difficulty levels - TV tropes. https://tvtropes.org/pmwiki/pmwiki.php/Main/DifficultyLevels. Accessed: 2020-5-6.

[4] Disabilities. https://www.afro.who.int/health-topics/disabilities. Accessed: 2021-2-22.

[5] Dwarf Fortress:Losing is Fun. https://dwarffortresswiki.org/index.php/DF2014:Losing.

[6] Fortnite player count 2020. https://www.statista.com/statistics/746230/fortnite-players/. Accessed: 2021-2-23.

[7] Free cheats and trainers for hundreds of games - WeMod app. https://www.wemod.com/. Accessed: 2020-7-1.

[8] Gamasutra - the chemistry of game design. https://www.gamasutra.com/view/feature/ 1524/the_chemistry_of_game_design.php?print=1. Accessed: 2020-6-9.

[9] Game accessibility guidelines — a straightforward reference for inclusive game design. http://gameaccessibilityguidelines.com/. Accessed: 2020-6-8.

[10] How do you feel about the turn timers in XCOM 2? - XCOM 2 - giant bomb. https://www.giantbomb.com/xcom-2/3030-49817/forums/how-do-you-feel-about-the-turn-timers-in-xcom-2-1791765/. Accessed: 2021-2-23.

[11] Interactive storytelling for video games — ScienceDirect. https://www.sciencedirect.com/book/9780240817170/interactive-storytelling-for-video-games. Accessed: 2020-5-6.

[12] Microsoft design. https://www.microsoft.com/design/inclusive/. Accessed: 2021-2-22.

[13] r/civ - dear firaxis, the change to the great works UI makes it nearly impossible to use for me as a disabled gamer.

[14] A theory of fun 10 years later. https://www.gdcvault.com/play/1016632/A-Theory-. Accessed: 2020-4-29.

[15] A theory of fun for game design. https://www.theoryoffun.com/press.shtml. Accessed: 2020-6-7.

[16] The player involvement model. In Gordon Calleja, editor, *In-Game*. The MIT Press, 2011.

[17] Legends of learning. https://www.legendsoflearning.com/blog/james-paul-gee-game-based-learning/, November 2016.

[18] I used a switch control for a day - 24 accessibility. https://www.24a11y.com/2018/i-used-a-switch-control-for-a-day/, December 2018. Accessed: 2020-7-24.

[19] Shadows AI retrospective discussion. https://forums.ultra-combo.com/t/shadows-ai-retrospective-discussion/24899, November 2018. Accessed: 2020-7-9.

[20] Mobile gaming statistics: 85+ statistics for 2020 — udonis. https://www.blog.udonis.co/mobile-marketing/mobile-games/mobile-gaming-statistics, December 2019. Accessed: 2020-5-28.

[21] What breath of the wild is like for someone who doesn't play games, November 2019.

[22] What games are like for someone who doesn't play games, September 2019.

[23] Designing DOOM eternal with hugo martin - noclip podcast #22, January 2020.

[24] Games industry unites to promote world health organization messages against COVID-19; launch #playaparttogether campaign.

https://www.businesswire.com/news/home/20200328005018/en/Games-Industry-Unites-Promote-World-Health-Organization, March 2020. Accessed: 2021-2-23.

[25] Paradox Development Studio. Crusader kings ii. Windows, 2012.

[26] Espen Aarseth. I fought the law: Transgressive play and the implied player. In Naomi Segal and Daniela Koleva, editors, *From Literature to Cultural Literacy*, pages 180–188. Palgrave Macmillan UK, London, 2014.

[27] Zeyad Abd Algfoor, Mohd Shahrizal Sunar, and Hoshang Kolivand. A comprehensive study on pathfinding techniques for robotics and video games. *International Journal of Computer Games Technology*, 2015, April 2015.

[28] Azita Iliya Abdul Jabbar and Patrick Felicia. Gameplay engagement and learning in Game-Based learning: A systematic review. *Rev. Educ. Res.*, 85(4):740–779, December 2015.

[29] Ernest Adams. The designer's notebook: Difficulty modes and dynamic difficulty adjus, May 2008.

[30] Ernest Adams. The designer's notebook: Eight ways to make a bad tutorial, 2011.

[31] Ernest Adams. *Fundamentals of Game Design.* Pearson Education, 2014.

[32] R E Adamson. Functional fixedness as related to problem solving; a repetition of three experiments. *J. Exp. Psychol.*, 44(4):288–291, October 1952.

[33] Alena Denisova Swansea University, Swansea, Wales, United Kingdom, Christian Guckelsberger Goldsmiths, University of London, London, United Kingdom, and David Zendle University of York, York, North Yorkshire, United Kingdom. Challenge in digital games — proceedings of the 2017 CHI conference extended abstracts on human factors in computing systems. https://dl.acm.org/doi/10.1145/3027063.3053209. Accessed: 2020-1-22.

[34] Ali Alkhatlan and Jugal Kalita. Intelligent tutoring systems: A comprehensive historical survey with recent developments. December 2018.

[35] Fraser Allison, Marcus Carter, and Martin Gibbs. Good frustrations: the paradoxical pleasure of fearing death in DayZ. In *Proceedings of the Annual Meeting of the Australian Special Interest Group for Computer Human Interaction*, pages 119–123. dl.acm.org, 2015.

[36] Eloi Alonso, Maxim Peter, David Goumard, and Joshua Romoff. Deep reinforcement learning for navigation in aaa video games. *arXiv preprint arXiv:2011.04764*, 2020.

[37] Eloi Alonso, Maxim Peter, David Goumard, and Joshua Romoff. Deep reinforcement learning for navigation in AAA video games. November 2020.

[38] Erik Andersen, Eleanor O'Rourke, Yun-En Liu, Richard Snider, Jeff Lowdermilk, David Truong, Seth Cooper, and Zoran Popović. The impact of tutorials on

games of varying complexity. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems.* ACM, 2012.

[39] Craig G Anderson, Jen Dalsen, Vishesh Kumar, Matthew Berland, and Constance Steinkuehler. Failing up: How failure in a game environment promotes learning through discourse. *Thinking Skills and Creativity*, 30:135–144, December 2018.

[40] John R Anderson, C Franklin Boyle, and Brian J Reiser. Intelligent tutoring systems. *Science*, 228(4698):456–462, 1985.

[41] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. July 2017.

[42] Anna Anthropy. Level design lesson: To the right, hold on tight. *Auntie Pixelante*, 8, 2009.

[43] David P Ausubel and Mohamed Youssef. The effect of spaced repetition on meaningful retention. *The Journal of General Psychology*, 73(1):147–150, 1965.

[44] Ramon Axelrod. Navigation graph generation in highly dynamic worlds. *AI Game Programming Wisdom*, 4:2008, 2008.

[45] B Aytemiz, I Karth, J Harder, A M Smith, and others. Talin: A framework for dynamic tutorials based on the skill atoms theory. *Artif. Intell.*, 2018.

[46] B Aytemiz, I Karth, J Harder, A M Smith, and others. Talin: A framework for dynamic tutorials based on the skill atoms theory. *Artif. Intell.*, 2018.

[47] Batu Aytemiz, Sam Devlin, and Katja Hofmann. Designer-centered reinforcement learning. https://www.microsoft.com/en-us/research/blog/designer-centered-reinforcement-learning/, February 2021. Accessed: 2021-9-27.

[48] Batu Aytemiz, Hunter Lynch, Carl Erez, and Adam M Smith. Smes: Adapting dexterity-based games for deliberative play. In *AIIDE Workshops*, 2020.

[49] Batu Aytemiz, Xueer Shu, Eric Hu, and Adam Smith. Your buddy, the grandmaster: Repurposing the Game-Playing AI surplus for inclusivity. *AIIDE*, 16(1):17–23, October 2020.

[50] Batu Aytemiz and Adam M Smith. A diagnostic taxonomy of failure in videogames. In *International Conference on the Foundations of Digital Games*, number Article 18 in FDG '20, pages 1–11, New York, NY, USA, September 2020. Association for Computing Machinery.

[51] Batu Aytemiz and Adam M Smith. A diagnostic taxonomy of failure in videogames. In *International Conference on the Foundations of Digital Games*, number Article 18 in FDG '20, pages 1–11, New York, NY, USA, September 2020. Association for Computing Machinery.

[52] P Backlund and M Hendrix. Educational games - are they worth the effort? a literature survey of the effectiveness of serious games. In *2013 5th International Conference on Games and Virtual Worlds for Serious Applications (VS-GAMES)*, pages 1–8, September 2013.

[53] Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, and Charles Blundell. Agent57: Outperforming the atari human benchmark. March 2020.

[54] Sander C J Bakkes, Pieter H M Spronck, and Giel van Lankveld. Player behavioural modelling for video games. *Entertain. Comput.*, 3(3):71–79, 2012.

[55] BANDAI NAMCO. Darksouls. PC, September 2011.

[56] Sam Devlin Batu Aytemiz, Mikhail Jacob. Acting with style: Towards designer centred reinforcement learning for the videogames industry. *Reinforcement Learning for Humans, Computers and Interaction Workshop at CHI 2021*, 2021, May 2021.

[57] Edward Beeching, Maxim Peter, Philippe Marcotte, Jilles Debangoye, Olivier Simonin, Joshua Romoff, and Christian Wolf. Graph augmented deep reinforcement learning in the gamerland3d environment. *arXiv preprint arXiv:2112.11731*, 2021.

[58] Jen Beeston, Christopher Power, Paul Cairns, and Mark Barlet. Accessible player experiences (APX): The players. In *Computers Helping People with Special Needs*, pages 245–253. Springer International Publishing, 2018.

[59] Nicolae Berbece. This is a talk about tutorials, press a to skip, January 2016.

[60] Christopher et. al Berner. Dota 2 with large scale deep reinforcement learning. December 2019.

[61] Paul Bertens, Anna Guitart, and África Periáñez. Games and big data: A scalable Multi-Dimensional churn prediction model. October 2017.

[62] Lars Bishop, Dave Eberly, Turner Whitted, Mark Finch, and Michael Shantz. Designing a pc game engine. *IEEE Computer Graphics and Applications*, 18(1):46–53, 1998.

[63] E L Bjork and R A Bjork. Making things hard on yourself, but in a good way: Creating desirable difficulties to enhance learning. *teaching.yale-nus.edu.sg*.

[64] James H Block and Robert B Burns. Mastery learning. *Review of research in education*, 4:3–49, 1976.

[65] Fran C Blumberg, Sheryl F Rosenthal, and John D Randall. Impasse-driven learning in the context of video games. July 2008.

[66] Noam Brown and Tuomas Sandholm. Superhuman AI for multiplayer poker. *Science*, 365(6456):885–890, August 2019.

[67] Stefano Burigat and Luca Chittaro. Navigation in 3D virtual environments: Effects of user experience and location-pointing navigation aids. *Int. J. Hum. Comput. Stud.*, 65(11):945–958, November 2007.

[68] E Butler, A M Smith, Y E Liu, and Z Popovic. A mixed-initiative tool for designing level progressions in games. *Proceedings of the 26th annual*, 2013.

[69] Tanya Byron. Safer children in a digital world. *London: DCFS*, 2008.

[70] Sébastien Bénard. How dead cells secretly stops you from dying — audio logs, May 2019.

[71] P Cairns, C Power, M Barlet, and G Haynes. Future design of accessibility in games: A design vocabulary. *International Journal of Human*, 2019.

[72] Paul Cairns, Christopher Power, Mark Barlet, Gregory Haynes, Craig Kaufman, and Jen Beeston. Enabled players: The value of accessible digital games. *Games and Culture*, page 1555412019893877, December 2019.

[73] Paul Cairns, Christopher Power, Mark Barlet, Gregory Haynes, Craig Kaufman, and Jen Beeston. Enabled players: The value of accessible digital games. *Games and Culture*, 16(2):262–282, 2021.

[74] Murray Campbell, A Joseph Hoane, and Feng-Hsiung Hsu. Deep blue. *Artif. Intell.*, 134(1):57–83, January 2002.

[75] Dennis Carr. Nerve damage, Oct 2018.

[76] Chris Carter. Killer instinct's shadow AI mechanic is insanely detailed. https://www.destructoid.com/killer-instinct-s-shadow-ai-mechanic-is-insanely-detailed-293099.phtml, 2015. Accessed: 2020-7-9.

[77] Nicky Case. How to become a centaur. *Journal of Design and Science*, January 2018.

[78] C Catalá, J K Rose, and A B Bennett. Auxin-regulated genes encoding cell wall-

modifying proteins are expressed during early tomato fruit growth. *Plant Physiol.*, 122(2):527–534, February 2000.

[79] Jared E Cechanowicz, Carl Gutwin, Scott Bateman, Regan Mandryk, and Ian Stavness. Improving player balancing in racing games. In *Proceedings of the first ACM SIGCHI annual symposium on Computer-human interaction in play*, CHI PLAY '14, pages 47–56, New York, NY, USA, October 2014. Association for Computing Machinery.

[80] Gifford K Cheung, Thomas Zimmermann, and Nachiappan Nagappan. The first hour experience: how the initial play can engage (or lose) new players. In *ACM SIGCHI symposium on HCI in play*, pages 57–66. dl.acm.org, 2014.

[81] Noam Chomsky. *Understanding power: the indispensable Chomsky.* The New Press, 2002.

[82] Chris Thorn. Jim gee principles on gaming, November 2013.

[83] G Christopher Williams. Active learning: The pedagogy of the game tutorial. https://www.popmatters.com/111485-active-learning-the-pedagogy-of-the-game-tutorial-2496084217.html, September 2009. Accessed: 2020-6-7.

[84] Karl Cobbe, Jacob Hilton, Oleg Klimov, and John Schulman. Phasic policy gradient. September 2020.

[85] Michele Colledanchise and Petter Ögren. *Behavior Trees in Robotics and AI: An Introduction.* CRC Press, July 2018.

[86] Kate Compton and Michael Mateas. Casual creators. In *ICCC*, pages 228–235. axon.cs.byu.edu, 2015.

[87] Mia Consalvo. *Cheating: Gaining Advantage in Videogames.* MIT Press, August 2009.

[88] Daniel Cook. The chemistry of game design, Jul 2007.

[89] Henry Caldwell Cook. *The Play Way: An Essay in Educational Method.* Frederick A. Stokes Company, 1917.

[90] Sasha Costanza-Chock. *Design justice: Community-led practices to build the worlds we need.* The MIT Press, 2020.

[91] Greg Costikyan. *Uncertainty in Games.* MIT Press, 2013.

[92] Ben Cowley, Darryl Charles, Michaela Black, and Ray Hickey. Toward an understanding of flow in video games. July 2008.

[93] Anna Cox, Paul Cairns, Pari Shah, and Michael Carroll. Not doing but thinking: the role of challenge in the gaming experience. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 79–88, New York, NY, USA, May 2012. Association for Computing Machinery.

[94] Chris Crawford. The art of computer game design. 1984.

[95] Thibault de Vassal. Advanced chess. http://www.ficgs.com/wiki$_en - advanced - chess.html. Accessed : 2020 - 6 - 2.$

[96] Lars de Wildt, Thomas H Apperley, Justin Clemens, Robbie Fordyce, and Souvik Mukherjee. (re-)orienting the video game avatar. *Games Cult.*, 15(8):962–981, December 2020.

[97] O Delalleau, E Contal, E Thibodeau-Laufer, R C Ferrari, Y Bengio, and F Zhang. Beyond skill rating: Advanced matchmaking in ghost recon online. *IEEE Trans. Comput. Intell. AI Games*, 4(3):167–177, September 2012.

[98] Design Doc. Teaching game mechanics well - guidance VS. hand holding design doc, August 2016.

[99] Heather Desurvire and Charlotte Wiberg. Master of the game: assessing approachability in future game design. In *CHI'08 extended abstracts on Human Factors in Computing Systems*, pages 3177–3182. ACM, 2008.

[100] Heather Desurvire and Charlotte Wiberg. Game usability heuristics (play) for evaluating and designing better games: The next iteration. In *International Conference on Online Communities and Social Computing*, pages 557–566. Springer, 2009.

[101] Heather Desurvire and Charlotte Wiberg. User experience design for inexperienced gamers: GAP – game approachability principles. In Regina Bernhaupt, editor, *Evaluating User Experience in Games: Concepts and Methods*, pages 131–147. Springer London, London, 2010.

[102] Heather Desurvire and Charlotte Wiberg. User experience design for inexperi-

enced gamers: Gap—game approachability principles. In *Game user experience evaluation*, pages 169–186. Springer, 2015.

[103] Sebastian Deterding. The lens of intrinsic skill atoms: A method for gameful design. *Hum.-Comput. Interact.*, 30(3-4):294–335, May 2015.

[104] J.P. Doignon and J.C. Falmagne. *Knowledge Spaces*. Springer Berlin Heidelberg, 2012.

[105] Peter E Doolittle. Vygotsky's zone of proximal development as a theoretical foundation for cooperative learning. *Journal on Excellence in College Teaching*, 8(1):83–103, 1997.

[106] Anders Drachen, Pejman Mirza-Babaei, and Lennart Nacke. *Games User Research*. Oxford University Press, Inc., USA, 2018.

[107] Brandon Drenikow and Pejman Mirza-Babaei. Vixen: interactive visualization of gameplay experiences. In *Proceedings of the 12th International Conference on the Foundations of Digital Games*, page 3. ACM, 2017.

[108] Yuqing Du, Stas Tiomkin, Emre Kiciman, Daniel Polani, Pieter Abbeel, and Anca Dragan. AvE: Assistance via empowerment. June 2020.

[109] Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Go-Explore: a new approach for Hard-Exploration problems. January 2019.

[110] Egoraptor. Sequelitis - mega man classic vs. mega man X, October 2011.

[111] Kit Englard. Microsoft has created the most accessible gaming controller ever made. https://www.vice.com/en$_us/article/d3kvx7/microsoft-adaptive-controller, 2018. Accessed: 2020-7-24$.

[112] Eurogamer. Miyamoto on world 1-1: How nintendo made mario's most iconic level.

[113] Extra Credits. Design club - super mario bros: Level 1-1 - how super mario mastered level design, June 2014.

[114] Facepunch Studios Ltd. Rust. Windows, 2013.

[115] L V Fernandes, C D Castanho, and R P Jacobi. A survey on game analytics in massive multiplayer online games. In *2018 17th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, pages 21–2109, October 2018.

[116] Firaxis Games. Civilization iv. Windows PC version, 2005.

[117] Firaxis Games. Civilization VI. PC, October 2016.

[118] Firaxis Games. Xcom 2. Windows PC version, 2016.

[119] Bennet Foddy. QWOP. PC, November 2008.

[120] Bennet Foddy. Getting over it with bennett foddy. PC, December 2017.

[121] Renata Pontin M et. al Fortes. Game accessibility evaluation methods: A literature survey. In *Universal Access in HCI. Design and Development Approaches and Methods*, page 182, 2017.

[122] Miguel Frade, Francisco Fernandez de Vega, and Carlos Cotta. Evolution of artificial terrains for video games based on accessibility. In *European Conference on the Applications of Evolutionary Computation*, pages 90–99. Springer, 2010.

[123] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *arXiv preprint arXiv:1811.12560*, 2018.

[124] From Software. Sekiro : Shadows Die Twice. Windows PC version, 2018.

[125] Ethan Gach. The year in tiny video game text, 2019, Dec 2019.

[126] Lee Garber. Game accessibility: enabling everyone to play. *Computer*, (6):14–18, 2013.

[127] Lee Garber. Game accessibility: enabling everyone to play. *Computer*, (6):14–18, 2013.

[128] GDC. How to make great game tutorials, August 2016.

[129] Gdc. This is a talk about tutorials, press a to skip, January 2019.

[130] GDC. Forgiveness mechanics: Reading minds for responsive gameplay, March 2020.

[131] J P Gee. Learning by design: Games as learning machines. *Interactive educational multimedia: IEM*, 2004.

[132] J P Gee. Situated language and learning: A critique of traditional schooling. 2004.

[133] J P Gee. Are video games good for learning? *Nordic Journal of Digital Literacy*, 2006.

[134] J P Gee. Games for learning. *Educational Horizons*, 2013.

[135] James Paul Gee. Good video games and good learning. *Phi Kappa Phi Forum*, 85:33+, 2005.

[136] James Paul Gee. Learning by design: Good video games as learning machines. *E-Learning and Digital Media*, 2(1):5–16, March 2005.

[137] James Paul Gee and Elizabeth Hayes. Nurturing affinity spaces and game-based learning, 2012.

[138] Cameron Gidari. What it's like to play games when you're colorblind, July 2014.

[139] Camilo Gordillo, Joakim Bergdahl, Konrad Tollmar, and Linus Gisslén. Improving playtesting coverage via curiosity driven reinforcement learning agents. March 2021.

[140] David Graeber. Are you an anarchist? the answer may surprise you! *The Anarchist Library*, 2010.

[141] Dimitris Grammenos, Anthony Savidis, Yannis Georgalis, and Constantine Stephanidis. Access invaders: Developing a universally accessible action game. In *Computers Helping People with Special Needs*, pages 388–395. Springer Berlin Heidelberg, 2006.

[142] Michael Cerny Green, Ahmed Khalifa, Gabriella A B Barros, Andy Nealen, and Julian Togelius. Generating levels that teach mechanics. In *Proceedings of the 13th International Conference on the Foundations of Digital Games*, number Article 55 in FDG '18, pages 1–8, New York, NY, USA, August 2018. Association for Computing Machinery.

[143] Michael Cerny Green, Ahmed Khalifa, Gabriella AB Barros, and Julian Togelius. "press space to fire": Automatic video game tutorial generation. *arXiv preprint arXiv:1805.11768*, 2018.

[144] Matthew Guzdial, Nicholas Liao, Jonathan Chen, Shao-Yu Chen, Shukan Shah, Vishwa Shah, Joshua Reno, Gillian Smith, and Mark Riedl. Friend, collaborator, student, manager: How design of an AI-Driven game level editor affects creators. January 2019.

[145] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy maximum entropy deep reinforcement learning with a stochastic actor. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1861–1870, Stockholmsmässan, Stockholm Sweden, 2018. PMLR.

[146] Thomas Hainey, Thomas M Connolly, Elizabeth A Boyle, Amanda Wilson, and Aisya Razak. A systematic literature review of games-based learning empirical evidence in primary education. *Comput. Educ.*, 102:202–223, November 2016.

[147] Juho Hamari and Janne Tuunanen. Player types: A meta-synthesis. 2014.

[148] Vicki L Hanson and John T Richards. Progress on website accessibility? *ACM Trans. Web*, 7(1):1–30, March 2013.

[149] Harmonix Music Systems. Guitar hero. Sony Playstation 2, October 2005.

[150] Chip Heath and Dan Heath. *Made to Stick: Why Some Ideas Survive and Others Die.* Random House Publishing Group, January 2007.

[151] Ralf Herbrich, Tom Minka, and Thore Graepel. TrueSkill™: a bayesian skill rating system. In *Advances in neural information processing systems*, pages 569–576. papers.nips.cc, 2007.

[152] Kat Holmes. *Mismatch: How Inclusion Shapes Design.* MIT Press, September 2018.

[153] Christoffer Holmgård, Michael Cerny Green, Antonios Liapis, and Julian Togelius. Automated playtesting with procedural personas through mcts with evolved heuristics. *IEEE Transactions on Games*, 11(4):352–362, 2018.

[154] Danial Hooshyar, Moslem Yousefi, and Heuiseok Lim. Data-Driven approaches to game player modeling: A systematic literature review. *ACM Comput. Surv.*, 50(6):90:1–90:19, January 2018.

[155] Danial Hooshyar, Moslem Yousefi, and Heuiseok Lim. Data-driven approaches to game player modeling: a systematic literature review. *ACM Computing Surveys (CSUR)*, 50(6):90:1–90:19, January 2018.

[156] Britton Horn, Seth Cooper, and Sebastian Deterding. Adapting cognitive task analysis to elicit the skill chain of a game. In *Proceedings of the Annual Symposium on Computer-Human Interaction in Play*, pages 277–289. ACM, 2017.

[157] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[158] Robin Hunicke. The case for dynamic difficulty adjustment in games. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, ACE '05, pages 429–433, New York, NY, USA, 2005. ACM.

[159] Robin Hunicke, Marc LeBlanc, and Robert Zubek. MDA: A formal approach to game design and game research. In *Proceedings of the AAAI Workshop on Challenges in Game AI*, volume 4, page 1722, 2004.

[160] I Iacovides, J Aczel, E Scanlon, and others. Making sense of game-play: How can we examine learning and involvement? *Transactions of the Digital*, 2013.

[161] Ioanna Iacovides, James Aczel, Eileen Scanlon, Josie Taylor, and Will Woods. Motivation, engagement and learning through digital games. *IJVPLE*, 2(2):1–16, April 2011.

[162] Ioanna Iacovides, James Aczel, Eileen Scanlon, and William Woods. What can

breakdowns and breakthroughs tell us about learning and involvement experienced during game-play? page 7, 2011.

[163] Ioanna Iacovides, Anna L Cox, Patrick McAndrew, James Aczel, and Eileen Scanlon. Game-Play breakdowns and breakthroughs: Exploring the relationship between action, understanding, and involvement. *Human–Computer Interaction*, 30(3-4):202–231, May 2015.

[164] Roziana Ibrahim, Gary Wills, and Lester Gilbert. degendering games: Towards a Gender-Inclusive framework for games. pages 127–130. eprints.soton.ac.uk, July 2010.

[165] id Software. Doom eternal. PC, March 2020.

[166] Insomniac Games. Spyro the dragon. PlayStation, 1998.

[167] Mikhail Jacob, Sam Devlin, and Katja Hofmann. "it's unwieldy and it takes a lot of Time"—Challenges and opportunities for creating agents in commercial games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 16, pages 88–94, 2020.

[168] Mikhail Jacob, Sam Devlin, and Katja Hofmann. "it's unwieldy and it takes a lot of time"—challenges and opportunities for creating agents in commercial games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 16, pages 88–94, 2020.

[169] Darran Jamieson. 4 ways to teach your players how to play your

game. https://gamedevelopment.tutsplus.com/tutorials/4-ways-to-teach-your-players-how-to-play-your-game–cms-22719. Accessed: 2020-4-29.

[170] Michael John. There are no single player video games, June 2020.

[171] Daniel Johnson, Christian Jones, Laura Scholes, and Michelle Carras. *Videogames and wellbeing: A comprehensive review.* Young and Well Cooperative Research Centre, Australia, 2013.

[172] Mark R Johnson and Jamie Woodcock. The impacts of live streaming and twitch.tv on the video game industry. *Media Cult. Soc.*, 41(5):670–688, July 2019.

[173] Soren Johnson. GD column 17: Water finds a crack. https://www.designer-notes.com/?p=369, June 2011. Accessed: 2021-2-23.

[174] Jesper Juul. Fear of failing? the many meanings of difficulty in video games. *The video game theory reader*, 2(01):2009, 2009.

[175] Jesper Juul. *Half-Real: Video Games between Real Rules and Fictional Worlds.* MIT Press, August 2011.

[176] Jesper Juul. *The Art of Failure: An Essay on the Pain of Playing Video Games.* MIT Press, 2013.

[177] Daniel Kahneman. *Thinking, Fast and Slow.* Macmillan, October 2011.

[178] Kathie Anne Kallevig. Perceptions of failure in education: Changing the fear of failure through gamification. 2015.

[179] Slava Kalyuga. Expertise reversal effect and its implications for learner-tailored instruction. *Educational psychology review*, 19(4):509–539, 2007.

[180] Sean H K Kang. Spaced repetition promotes efficient and effective learning: Policy implications for instruction. *Policy Insights from the Behavioral and Brain Sciences*, 3(1):12–19, March 2016.

[181] M Kapur. Productive failure. *Cogn. Instr.*, 2008.

[182] M Kapur and K Bielaczyc. Designing for productive failure. *Journal of the Learning Sciences*, 2012.

[183] Isaac Karth. Ergodic agency: How play manifests understanding. *Engaging with Videogames: Play, Theory and Practice*, pages 205–216, 2014.

[184] Garry Kasparov. Don't fear intelligent machines. work with them, 2017.

[185] Itay Keren. Teaching by design: Tips for effective tutorials from 'mushroom 11'. https://www.gdcvault.com/play/1024187/Teaching-by-Design-Tips-for, 2017. Accessed: 2020-5-27.

[186] Ahmed Khalifa, Philip Bontrager, Sam Earle, and Julian Togelius. PCGRL: Procedural content generation via reinforcement learning. January 2020.

[187] Patrick Klepek. The small but important change 'celeste' made to its celebrated assist mode. https://www.vice.com/en$_us/article/43kadm/celeste - assist - mode - change - and - accessibility, September 2019. Accessed : 2020 - 5 - 8.$

[188] Hannu Korhonen and Elina M I Koivisto. Playability heuristics for mobile games. In *Proceedings of the 8th conference on Human-computer interaction with mobile devices and services*, pages 9–16. dl.acm.org, 2006.

[189] Raph Koster. *Theory of Fun for Game Design.* "O'Reilly Media, Inc.", November 2013.

[190] Victoria Krakovna and Jonathan et. al Uesato. Specification gaming: the flip side of AI ingenuity. https://deepmind.com/blog/article/Specification-gaming-the-flip-side-of-AI-ingenuity, 2020. Accessed: 2020-5-18.

[191] Ben Kuchera. When is exclusion a valid design choice? https://www.polygon.com/2017/10/4/16422060/cuphead-difficulty-exclusion, October 2017. Accessed: 2020-6-2.

[192] Jozef Kulik, Jen Beeston, and Paul Cairns. Grounded theory of accessible game development. In *The 16th International Conference on the Foundations of Digital Games (FDG) 2021*, number Article 28 in FDG'21, pages 1–9, New York, NY, USA, August 2021. Association for Computing Machinery.

[193] Josiah Lebowitz and Chris Klug. *Interactive Storytelling for Video Games.* Taylor and Francis, 2012.

[194] Joel et. al Lehman. The surprising creativity of digital evolution: A collection of anecdotes from the evolutionary computation and artificial life research communities. March 2018.

[195] Wei Li, Tovi Grossman, and George Fitzmaurice. Gamicad: a gamified tutorial system for first time autocad users. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*, pages 103–112. ACM, 2012.

[196] Juntong Lin, Xuyun Yang, Peiwei Zheng, and Hui Cheng. End-to-end decentralized multi-robot navigation in unknown complex environments via deep reinforcement learning. In *2019 IEEE International Conference on Mechatronics and Automation (ICMA)*, pages 2493–2500. IEEE, 2019.

[197] J Linderoth. Why gamers don't learn more: An ecological approach to games as learning environments. *Journal of Gaming & Virtual Worlds*, 2012.

[198] Jonas Linderoth. it is not hard, it just requires having no life – computer games and the illusion of learning. *Nordic Journal of Digital Literacy*, 4(01):4–19, 2009.

[199] Yiyi Liu. *Disabled Gamers: Accessibility in Video Games.* PhD thesis, Carleton University, 2018.

[200] LoF Dev. James paul gee - games for change keynote speech on affinity spaces - 6/20/2012, July 2012.

[201] Ludia. Where's waldo? the fantastic journey. NintendoDS, September 2009.

[202] T Marsh, P Wright, and S Smith. Evaluation for the design of experience in virtual environments: modeling breakdown of interaction and illusion. *Cyberpsychol. Behav.*, 4(2):225–238, April 2001.

[203] Dominic Masters and Carlo Luschi. Revisiting small batch training for deep neural networks. April 2018.

[204] Michael Mateas and Andrew Stern. Ludology meets narratology in game design space. *Worlds in Play: International Perspectives on Digital Games Research*, 21:267, 2007.

[205] Matt Makes Games. Celeste. Windows PC version, 2018.

[206] Colt McAnlis and James Stewart. Intrinsic detail in navigation mesh generation. *AI Game Programming Wisdom*, 4:95–112, 2008.

[207] Jane McGonigal. *Reality Is Broken: Why Games Make Us Better and How They Can Change the World.* Penguin, January 2011.

[208] Alec Meer. Interview: Firaxis' jake solomon on what went right and wrong with XCOM 2. https://www.rockpapershotgun.com/making-of-xcom-2, February 2016. Accessed: 2021-2-23.

[209] Lori S Mestre. Student preference for tutorial design: A usability study. *Reference Services Review*, 40(2):258–276, 2012.

[210] Robinson Meyer. The atlantic. *The Atlantic*, June 2015.

[211] Tom Minka, Ryan Cleven, and Yordan Zaykov. Trueskill 2: An improved bayesian skill rating system. *Tech. Rep.*, 2018.

[212] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis

Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[213] Andy Moore. Indie flash developer discusses the changes he made to develop a hit game. http://www.indiegamepod.com/?p=2076, September 2010. Accessed: 2020-5-15.

[214] Motion Twin. Dead cells. Windows PC version, 2017.

[215] Imad Mougharbel, Racha El-Hajj, Houda Ghamlouch, and Eric Monacelli. Comparative study on different adaptation approaches concerning a sip and puff controller for a powered wheelchair. In *2013 Science and Information Conference*, pages 597–603. ieeexplore.ieee.org, October 2013.

[216] Dinara Moura and Lyn Bartram. Investigating players' responses to wayfinding cues in 3D video games. In *CHI '14 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '14, pages 1513–1518, New York, NY, USA, April 2014. Association for Computing Machinery.

[217] Dinara Moura and Magy Seif El-Nasr. Design techniques for planning navigational systems in 3-D video games. *Comput. Entertain.*, 12(2):1–25, February 2015.

[218] Curtiss Murphy. Why games work and the science of learning. March 2012.

[219] Ilya Musabirov, Paul Okopny, and Denis Bulygin. Analyzing chat logs in online games for tutorial improvement. In *Proceedings of the 2015 Annual Symposium*

on Computer-Human Interaction in Play, CHI PLAY '15, pages 661–666, New York, NY, USA, October 2015. Association for Computing Machinery.

[220] Herman Narula. A billion new players are set to transform the gaming industry. https://www.wired.co.uk/article/worldwide-gamers-billion-players, 2019. Accessed: 2021-2-23.

[221] Dave Neal, Ed Boraas, Gary Elkin, and Iain McKay. An anarchist faq, Mar 2020.

[222] Derek Neal. Designing AI for killer instinct. https://www.youtube.com/watch?v=9yydYjQ1GLg, 2016.

[223] Mark J Nelson. Game metrics without players: Strategies for understanding game artifacts. In *Workshops at the Seventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2011.

[224] Mark J Nelson, David L Roberts, Charles L Isbell, and Michael Mateas. Reinforcement learning for declarative optimization-based drama management. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, AAMAS '06, pages 775–782, New York, NY, USA, May 2006. Association for Computing Machinery.

[225] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pages 278–287, 1999.

[226] Nintendo. Super mario odyssey. Nintendo Switch, 2017.

[227] Nintendo. Super smash bros. Nintendo 64, January 2019.

[228] Nintendo Entertainment. Super mario bros. Famicom, September 1985.

[229] Nintendo Entertainment. The legend of zelda: Breath of the wild. Nintendo Switch, March 2017.

[230] Safiya Umoja Noble. *Algorithms of oppression.* New York University Press, 2018.

[231] Don Norman. *The Design of Everyday Things: Revised and Expanded Edition.* Basic Books, November 2013.

[232] Paul Okopny, Ilya Musabirov, and Daniel Alexandrov. Informal In-Game help practices in massive multiplayer online games. In *Social Informatics*, pages 218–222. Springer International Publishing, 2015.

[233] Martin Oliver and Diane Carr. Learning in virtual worlds: Using communities of practice to explain how people learn from play. *Br. J. Educ. Technol.*, 40(3):444–457, May 2009.

[234] Cathy O'neil. *Weapons of math destruction: How big data increases inequality and threatens democracy.* Crown, 2016.

[235] World Health Organization. Games industry unites to promote world health organization messages against COVID-19; launch #playaparttogether campaign, March 2020.

[236] Eleanor O'Rourke, Erik Andersen, Sumit Gulwani, and Zoran Popović. A framework for automatically generating interactive instructional scaffolding. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 1545–1554. ACM, 2015.

[237] Kei Ota, Devesh K Jha, and Asako Kanezaki. Training larger networks for deep reinforcement learning. February 2021.

[238] Radek Pelánek. Applications of the elo rating system in adaptive educational systems. *Comput. Educ.*, 98:169–179, July 2016.

[239] Radek Pelánek, Jan Papoušek, Jiří Řihák, Vít Stanislav, and Juraj Nižnan. Elo-based learner modeling for the adaptive practice of facts. *User Model. User-adapt Interact.*, 27(1):89–118, 2017.

[240] Caroline Pelletier and Martin Oliver. Learning to play in digital games. *Learn. Media Technol.*, 31(4):329–342, December 2006.

[241] Caroline Criado Perez. *Invisible women: Exposing data bias in a world designed for men.* Random House, 2019.

[242] D Perez-Liebana, J Liu, A Khalifa, and others. General video game AI: A multi-track framework for evaluating agents, games, and content generation algorithms. *IEEE Transactions*, 2019.

[243] Serge et al Petralito. A good reason to die: How avatar death and high challenges

enable positive experiences. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, pages 5087–5097, May 2017.

[244] F Petrillo, M Pimenta, F Trindade, and others. What went wrong? a survey of problems in game development. *in Entertainment (CIE)*, 2009.

[245] Eve Marie Phillips. *If it works, it's not AI: a commercial look at artificial intelligence startups.* PhD thesis, Massachusetts Institute of Technology, 1999.

[246] Martin Picard. Machinima: Video game as an art form. 2006.

[247] David Pinelle, Nelson Wong, and Tadeusz Stach. Heuristic evaluation for games: usability principles for video game design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1453–1462. ACM, 2008.

[248] Amit Pitaru. *E is for everyone: The Case for inclusive game design.* MacArthur Foundation Digital Media and Learning Initiative, 2008.

[249] Jan L Plass, Richard E Mayer, and Bruce D Homer. *Handbook of game-based learning.* Mit Press, 2020.

[250] Jack Pooley. 10 worst video game tutorials everyone hated, September 2019.

[251] Daft Punk. *Harder, Better, Faster, Stronger.* Daft Life, 2001.

[252] Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, and Noah Dormann. Stable baselines3. https://github.com/DLR-RM/stable-baselines3, 2019.

[253] Sheri Graner Ray. Tutorials: Learning to play. https://www.gamasutra.com/view/feature/134531/tutorials_learning_to_play.php, October 2010. Accessed: 2020-5-27.

[254] Razbuten. What breath of the wild is like for someone who doesn't play games. https://www.youtube.com/watch?v=5LdenlAKb2g, November 2019.

[255] Relic Entertainment. Dawn of war iii. Windows, 2017.

[256] Relic Entertainment. Dawn of war iii. Windows, 2017.

[257] Alexander Renkl. Learning from worked-out examples: A study on individual differences. *Cognitive science*, 21(1):1–29, 1997.

[258] Eoin Roche. "valid" failure in the design of digital games: An introspective view from a player's perspective. *scss.tcd.ie*.

[259] William Ryan and Martin A Siegel. Evaluating interactive entertainment using breakdown: Understanding embodied learning in video games. In *DiGRA Conference*, volume 9, 2009.

[260] Vardan Saini and Matthew Guzdial. A demonstration of mechanic maker: An AI for mechanics co-creation. *AIIDE*, 16(1):325–327, October 2020.

[261] A L Samuel. Some studies in machine learning using the game of checkers. *IBM J. Res. Dev.*, 3(3):210–229, July 1959.

[262] Arindam Sarma. *Abstracting AI Evaluation Environments with Virtual Machines*. University of California, Santa Cruz, 2020.

[263] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[264] Mike Schmierbach, Mun-Young Chung, Mu Wu, and Keunyeong Kim. No one likes to lose. *Journal of Media Psychology*, 26(3):105–110, January 2014.

[265] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. July 2017.

[266] Brian Schwab, Dave Mark, Kevin Dill, Mike Lewis, and Richard Evans. GDC: Turing tantrums: AI developers rant, 2011.

[267] Leslie Scott. Jenga. Physical Game, 1983.

[268] Jim Scullion, Mark Stansfield, and Thomas Connolly. A survey of students' improved mastery of game playing skills through informal online game-based learning. In *Proceedings of the 5th European Conference on Games Based Learning*, volume 20, pages 535–542. researchgate.net, 2011.

[269] Rainforest Scully-Blaker. A practiced practice: Speedrunning through space with de certeau and virilio. *Game Studies*, 14(1), 2014.

[270] Rainforest Scully-Blaker. *Re-curating the Accident: Speedrunning as Community and Practice*. Concordia University, 2016.

[271] Jonathan Segal. Introducing bob's buddy. https://articles.hsreplay.net/2020/04/24/ introducing-bobs-buddy/, April 2020. Accessed: 2020-6-2.

[272] Michael Sellers. *Advanced Game Design: A Systems Approach.* Addison-Wesley Professional, October 2017.

[273] Serge Petralito University of Basel, Basel, Basel-Stadt, Switzerland, Florian Brühlmann University of Basel, Basel, Switzerland, Glena Iten University of Basel, Basel, Basel-Stadt, Switzerland, Elisa D. Mekler University of Waterloo, Waterloo, ON, Canada, and Klaus Opwis University of Basel, Basel, BS, Switzerland. A good reason to die — proceedings of the 2017 CHI conference on human factors in computing systems. https://dl.acm.org/doi/10.1145/3025453.3026047. Accessed: 2020-1-22.

[274] Seth Coster. Forgiveness mechanics: Reading minds for responsive gameplay, March 2020.

[275] Hampus Sethfors. I used a switch control for a day - 24 accessibility. https://www.24a11y.com/2018/i-used-a-switch-control-for-a-day/, December 2018. Accessed: 2020-7-24.

[276] Noor Shaker, Julian Togelius, and Mark J Nelson. *Procedural Content Generation in Games.* Springer, Cham, 2016.

[277] Molyneux Shuldham. XXI. account of the sea-cow, and the use made of it. *Philosophical Transactions of the Royal Society of London*, 65:249–251, January 1775.

[278] Albert Shum. Inclusive design toolkit, 2016.

[279] Fernando De Mesentier Silva, Igor Borovikov, John Kolen, Navid Aghdaie, and Kazi Zaman. Exploring gameplay with ai agents. In *Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2018.

[280] Mirna Paula Silva, Victor do Nascimento Silva, and Luiz Chaimowicz. Dynamic difficulty adjustment through an adaptive ai. In *Computer Games and Digital Entertainment (SBGames), 2015 14th Brazilian Symposium on*, pages 173–182. IEEE, 2015.

[281] David Silver and Aja et. al Huang. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016.

[282] A M Smith. Answer set programming in Proofdoku. *Thirteenth Artificial Intelligence and Interactive Digital Games*, 2017.

[283] Adam M Smith, Chris Lewis, Kenneth Hullett, Gillian Smith, and Anne Sullivan. An inclusive taxonomy of player modeling. *University of California, Santa Cruz, Tech. Rep. UCSC-SOE-11-13*, 2011.

[284] Adam M Smith and Michael Mateas. Computational caricatures: Probing the game design process with AI. In *Workshops at the Seventh AIIDE Conference*, October 2011.

[285] Adam M Smith and Daniel Shapiro. Teaching game AI as an undergraduate course in computational media. *adamsmith.as*.

[286] Gillian Smith, Jim Whitehead, and Michael Mateas. Tanagra: A mixed-initiative level design tool. In *Proceedings of the Fifth International Conference on the Foundations of Digital Games*, pages 209–216. ACM, 2010.

[287] Greg Snook. Simplified 3D movement and pathfinding using navigation meshes. *Game programming gems*, 1(1):288–304, 2000.

[288] Spry Fox. Steambirds. IOS, 2010.

[289] Pieter Jan Stappers and Elisa Giaccardi. Research through design. In *The encyclopedia of human-computer interaction*, pages 1–94. The Interaction Design Foundation, 2017.

[290] Edward Steinfeld and Jordana Maisel. *Universal Design: Creating Inclusive Environments*. John Wiley & Sons, April 2012.

[291] Studio MDHR. Cuphead. Windows PC version, 2017.

[292] Paul Suddaby. The many ways to show the player how it's done with In-Game tutorials. https://gamedevelopment.tutsplus.com/tutorials/the-many-ways-to-show-the-player-how-its-done-with-in-game-tutorials–gamedev-400. Accessed: 2020-1-22.

[293] Adam James Summerville, Sam Snodgrass, Michael Mateas, and Santiago Ontañón. The VGLC: The video game level corpus. June 2016.

[294] Libo Sun, Jinfeng Zhai, and Wenhu Qin. Crowd navigation in an unknown and dynamic environment based on deep reinforcement learning. *IEEE Access*, 7:109544–109554, 2019.

[295] R S Sutton and A G Barto. Reinforcement learning: An introduction. *IEEE Trans. Neural Netw.*, 9(5):1054–1054, September 1998.

[296] Steve Swink. *Game feel: a game designer's guide to virtual sensation*. CRC Press, 2008.

[297] Tapulous. Tap tap revenge. IOS, July 2008.

[298] Christopher Taylor. Able gamers includification. 2013.

[299] Team Meat. Super meat boy. Windows PC version, 2010.

[300] Gerald Tesauro. Temporal difference learning and TD-Gammon. *Commun. ACM*, 38(3):58–68, 1995.

[301] C Therrien. " to get help, please press x" the rise of the assistance paradigm in video game design. 2011.

[302] Cherry Thompson. Sekiro: Accessibility in games is about far more than 'difficulty' - IGN, April 2019.

[303] Tommy Thompson and Matthew Syrett. 'play your own way' adapting a procedural framework for accessibility. In *Proceedings of the 13th International Conference on the Foundations of Digital Games*, pages 1–7, 2018.

[304] Julian Togelius, Noor Shaker, Sergey Karakovskiy, and Georgios N Yannakakis. The mario ai championship 2009-2012. *AI Magazine*, 34(3):89–92, 2013.

[305] Mike Treanor, Alexander Zook, Mirjam P Eladhari, Julian Togelius, Gillian Smith, Michael Cook, Tommy Thompson, Brian Magerko, John Levine, and Adam Smith. Ai-based game design patterns. 2015.

[306] Unknown Worlds Entertainment. Subnautica. PC, January 2018.

[307] Valve Corporation. Dota 2. Windows, 2013.

[308] Wouter van Toll, Atlas F Cook, and Roland Geraerts. Navigation meshes for realistic multi-layered environments. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3526–3532, September 2011.

[309] Oriol Vinyals and Igor et. al Babuschkin. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, November 2019.

[310] Helen Wauck and Wai-Tat Fu. A Data-Driven, multidimensional approach to hint design in video games. In *Proceedings of the 22nd International Conference on Intelligent User Interfaces*, pages 137–147. dl.acm.org, 2017.

[311] Matthew M White. *Learn to Play: Designing Tutorials for Video Games.* CRC Press, June 2014.

[312] Josef Wiemeyer, Lennart Nacke, Christiane Moser, and Florian 'Floyd' Mueller. Player experience. In Ralf Dörner, Stefan Göbel, Wolfgang Effelsberg, and Josef

Wiemeyer, editors, *Serious Games: Foundations, Concepts and Practice*, pages 243–271. Springer International Publishing, Cham, 2016.

[313] Alex Wiltshire. Don't hate on tutorials. https://www.rockpapershotgun.com/2017 /06/16/how-usability-testing-makes-games-better/, June 2017.

[314] Narnia Worth. Players and avatars: The connections between player personality, avatar personality, and behavior in video games. 2015.

[315] Georgios N Yannakakis, Pieter Spronck, Daniele Loiacono, and Elisabeth André. Player modeling. 2013.

[316] Georgios N Yannakakis and Julian Togelius. Experience-driven procedural content generation. *IEEE Transactions on Affective Computing*, 2(3):147–161, 2011.

[317] Georgios N Yannakakis and Julian Togelius. *Artificial Intelligence and Games*. Springer, Cham, 2018.

[318] Nicholas Yee. Understanding MMORPG addiction. *Retrieved February*, 15:2008, 2002.

[319] Koki Yokoyama and Kazuyuki Morioka. Autonomous mobile robot with simple navigation system based on deep reinforcement learning and a monocular camera. In *2020 IEEE/SICE International Symposium on System Integration (SII)*, pages 525–530. IEEE, 2020.

[320] Carole L Yue, Alan D Castel, and Robert A Bjork. When disfluency is—and is not—a desirable difficulty: The influence of typeface clarity on metacognitive judgments and memory. *Mem. Cognit.*, 41(2):229–241, February 2013.

[321] Fanyu Zeng, Chen Wang, and Shuzhi Sam Ge. A survey on visual navigation for artificial agents with deep reinforcement learning. *IEEE Access*, 8:135426–135442, 2020.

[322] Z Zhan, B Aytemiz, and A M Smith. Taking the scenic route: Automatic exploration for videogames. *arXiv preprint arXiv:1812.03125*, 2018.

[323] Kai Zhu and Tao Zhang. Deep reinforcement learning based mobile robot navigation: A review. *Tsinghua Science and Technology*, 26(5):674–691, 2021.

[324] Mohammad Zohaib. Dynamic difficulty adjustment (DDA) in computer games: A review. *Advances in Human-Computer Interaction*, 2018, November 2018.

Appendix A

NavAssist Further Experiments

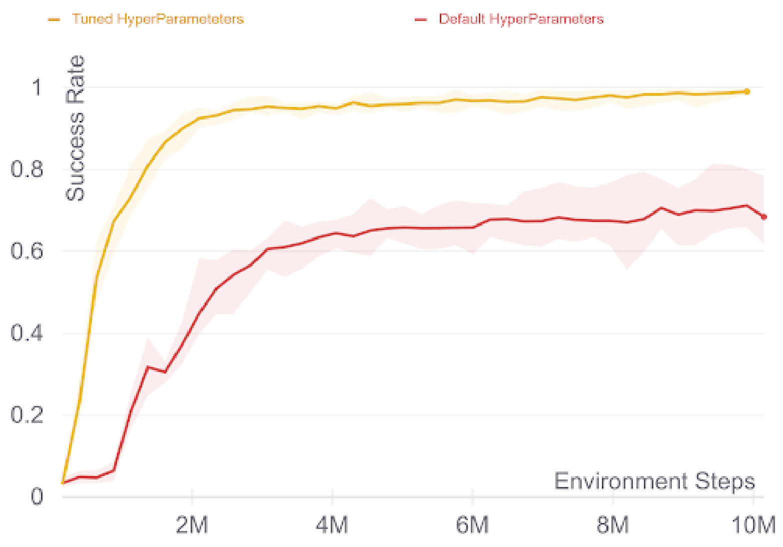


Figure A.1: Tuned hyperparameters versus default hyperparameters of the underlying library in the Urban environment. Shaded regions show standard deviation over 5 seeds.

Often in reinforcement learning research only the final, fully tuned training curves are reported. This however hides the amount of effort necessary in finding the correct hyperparameters and downplays the sensitivity of the algorithms. The experi-

Value Function Hidden Layer Width	Value Function Parameter Count	Mean Final Success Rate	Best Final Success Rate
2560	7.872.000	0.983 ± 0.008	0.992
2048	**5.249.024**	**0.988 ± 0.004**	**0.993**
1536	3.150.336	0.979 ± 0.006	0.988
1024	1.575.936	0.953 ± 0.004	0.960
512	525.824	0.942 ± 0.002	0.946
256	197.376	0.916 ± 0.008	0.930
128	82.304	0.830 ± 0.052	0.875

Table A.1: The results of training the reinforcement learning for navigation agent in the Urban environment with changing critic network sizes. The results show mean over 5 seeds.

Removed Component	Remaining Observation Size	Mean Final Success Rate
Baseline	505	**0.988 ± 0.0049**
Depth Map	450	0.979 ± 0.007
Whiskers	479	0.970 ± 0.013
Occupancy	100	0.934 ± 0.003
Absolute Position	499	0.967 ± 0.007
Curriculum	505	0.952 ± 0.002
Dense Reward	505	0.836 ± 0.020

Table A.2: The results of the ablation study on the reinforcement learning for navigation agent in the Urban environment. The results show mean over 5 seeds for 10 million timesteps.

ments shown are from training Soft-Actor Critic algorithm on the Urban environment used in chapter 8.

In an attempt to be more transparent on the importance of hyperparameter tuning figure A.1 shows the comparison between a highly tuned agent with the default hyperparameters.

In our hyperparameter tuning we found that one of the most important parameters to tune is the size of the value network, which is shown in table A.1. We also ran an ablation study to explore the importance of different types of observations and algorithmic decisions. Table A.2 shows the results of our experiments. The full list of used hyper-parameters can be found here: https://wandb.ai/batu/Urban/runs/1kxiy929/overview

Appendix B

NavAssist User Study Proposal

In this dissertation, I have contributed several computational prototypes (following the computational caricatures research method [284]). In order to learn more about the efficacy of these systems, and how players realistically interact with the systems, user studies need to be conducted. In particular, I am interested in exploring how players utilize and respond to the provided assistance methods. This chapter contributes a detailed study design in order to guide future researchers who want to follow up on the NavAssist project described in chapter 8. In particular, the study design sketched offers a method for evaluating the effectiveness of new assistance modes that might be invented the future. The user study will explore how players respond to the reinforcement learning powered assistance modes. The study will utilize a mixed-methods approach, collecting both quantitative and qualitative information via semi-structured interviews and task-based analysis.

Recruitment

Participants will all be ages 18 and over. They will be recruited online and within the Santa Cruz community to complete one or more tasks each. An intake questionnaire will record demographic info and previous videogame playing experience. We hope to recruit a wide variety of ages and especially experience levels.

Task Based-Analysis

The recruited individuals will be asked to complete the task of navigating to a set of goals in the provided videogame while using different assistance methods. This section will help us gather insights into players' responses to using each assistance method. Then, the recruited individuals will be asked to complete a similar set of navigation challenges, while having the option to use (or not use) whichever assistance method they prefer. This section will give us preliminary data on player choices into when and which assistance methods they use and will inform the design of future more comprehensive longitudinal studies. Before and after participating in these tasks, they will be asked to fill out participation surveys on their emotional states and perception of their experience. Both the screens and faces of the players will be recorded for matching the emotional responses of the player to the events happening on the screen. The players will be prompted to use the speak aloud [106] technique to help communicate their feelings and inner state.

Several metrics will be tracked such as the amount of time it takes for the players to complete the tasks, the number of mistakes they make, the amount of time they spend using each assistance mode when they have the choice. These metrics will be recorded by the game automatically and will be analyzed in conjunction with the qualitative information

Semi-Structured Interviews

As a part of the user study, semi-structured interviews will be conducted that aims to understand players' opinion of and past experience with accessibility modes, and their engagement with the RL-powered assistance methods presented. The interview structure asked in the interviews will be refined in early small-scale pilot studies.

The experiments will continue until the saturation of information. We will be using thematic analysis techniques to process the interviews. We expect the results of this analysis to inform the design of future prototypes and highlight important future research questions.