Your Buddy, The Grandmaster: Repurposing the Game-Playing AI Surplus for Inclusivity

Batu Aytemiz, Xueer Shu, Eric Hu, Adam M. Smith
Design Reasoning Lab
University of California, Santa Cruz
{baytemiz, xzhu54, erjihu, amsmith}@ucsc.edu

Abstract

Advances in artificial intelligence (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 the game-playing experience for our players. This paper is inspired by the positive impact of accessibility modes in recent games as well as previous calls in games research literature to focus on player experience. Responding to these calls, we propose utilizing GPAI techniques not to beat the player, as is traditionally done, but to support them in fully experiencing the game. We claim that utilizing GPAI agents to help players overcome barriers is a productive way of repurposing the capabilities of these agents. We further contribute a design exercise to help developers explore the space of possible GPAI-driven assistance methods. This exercise helps developers discover types of challenges and ideate methods that vary in magnitude and assistance type. We first apply this design exercise to explore the design space of possible assistance methods for the action platformer game *Celeste*. We then implement two of the discovered methods that target different challenge types in a Unity clone of *Celeste*. Through this implementation, we discover several additional research questions we must answer before GPAI-driven assistance methods can be truly effective. We believe this research direction furthers the discussion on how to utilize GPAI in service of the player experience and also contributes to the creation of more inclusive games.

Since the mid-2010s, there have been several high profile victories for game-playing AI (GPAI) techniques. In the videogames domain, the research community made progress towards overcoming several difficult problems: an immense branching factor in Go (Silver and Huang 2016); multiagent communication in DOTA2 (Berner 2019); working with hidden information in poke (Brown and Sandholm 2019); parsing the screen with Atari (Badia et al. 2020); and balancing low-level control with high-level decision making in Starcraft 2 (Vinyals and Babuschkin 2019). 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 paper, 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 (Yannakakis and Togelius 2018). Both PCG and Player Modeling have had wide adoption from the game industry (Fernandes, Castanho, and Jacobi 2018) (Bakkes, Spronck, and van Lankveld 2012). 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 (Yannakakis and Togelius 2018). 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 (Schwab 2011). 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 (Krakovna and Uesato 2020; Lehman 2018). 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 primar-

ily 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 (Pitaru 2008). Every player is different and might have different needs to fully engage with a game (Holmes 2018). 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 (Liu 2018). 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 (Matt Makes Games 2018) 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 (Andersen et al. 2012). One reason why players stop playing games is that they don't fully grasp how to play the game itself (Cheung, Zimmermann, and Nagappan 2014). 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 paper, 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 (Smith and Mateas 2011).

In summary, this paper 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.

Related Work

The ability to play games has been a long-lasting goal for AI systems. In 1959, Arthur Samuel, a pioneer of AI research, used the game of checkers to study machine learning (Samuel 1959). In the early 1990s, temporal difference learning was used to play backgammon (Tesauro 1995), and in 1997, IBM's chess-playing computer, Deep Blue, beat the former World Chess Champion Gary Kasparov (Campbell, Hoane, and Hsu 2002).

Kasparov's reaction to this defeat was very forward-facing. The following year, in 1998, he announced Advanced Chess (Kasparov 2017), a fresh take on chess where each human player uses a computer chess program to explore the possible results of candidate moves (de Vassal). 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 (Case 2018). 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" (Kasparov 2017).

In the last decade, there have been several highly publicized results with AlphaGo (Silver and Huang 2016), OpenAI Five (Berner 2019), Atari benchmarks (Badia et al. 2020), and Starcraft (Vinyals and Babuschkin 2019). However, most of this research, especially deep reinforcement learning, has been focused on pushing the score benchmarks instead of on enhancing the player experience. AI agents are getting stronger and stronger, but the research in creating different processes to utilize this strength for the gameplay experience is still ripe for exploration. Using Kasparov's terminology, this paper attempts to contribute one such process

in how we can utilize game-playing AIs to support player experience.

We must note, however, that there have already been many past developments in using AIs in the videogame context for the user experience. Focused research on pathfinding (Abd Algfoor, Sunar, and Kolivand 2015) has allowed even larger amounts of units to navigate. PCG methods increased the replayability of many games (Shaker, Togelius, and Nelson 2016) and became a key component of the roguelike and rogue-lite genres. Interactive storytelling research has resulted in more believable characters and richer storylines (Lebowitz and Klug 2012). Player modeling research has allowed deeper insights into the way our players behave (Hooshyar, Yousefi, and Lim 2018).

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 (Klepek 2019). 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 (Thompson 2019; Kuchera 2017). Challenge in many games is essential for the design and is shown empirically to increase enjoyability (Cox et al. 2012; Petralito 2017). 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 (Fortes 2017), 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 (Consalvo 2009).

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 (Segal 2020) 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 repurpuse GPAI techniques to assist our players.

Design Exercise for Discovering Game-playing AI Driven 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.

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 (Aytemiz and Smith forthcoming) (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" (Klepek 2019); 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**.

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 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.

Figure 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.

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.

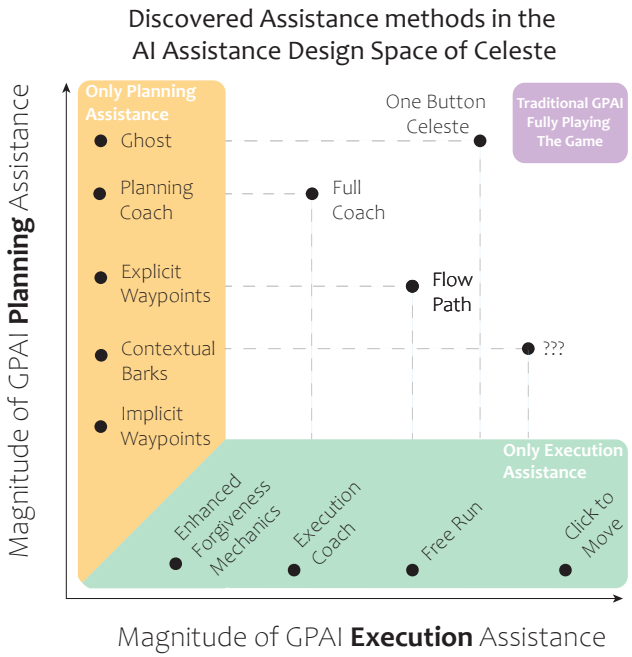

Figure 1: Each point represents a possible GPAI-driven assistance method discovered by applying the design exercise. The graph is not descriptive–it does not visualize preexisting data points. Rather, it is explorative, for it is used to discover assistance methods that map to points in the design space.

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 (Seth Coster 2020). 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.

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.

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.

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 (Smith and Mateas 2011) 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 (Togelius et al. 2013), 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 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.

Ghost Implementation Details

With the Ghost (fig 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 manner. 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. In contrast, having access to the Ghost assistance

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

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

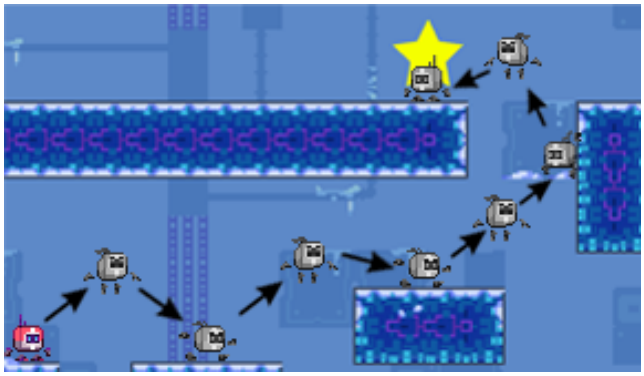

Figure 2: Our implementation of the Ghost planning assistance. When a player asks for help, we spawn a copy of the player, and it takes necessary actions to reach the goal. It is up to the player to repeat those actions.

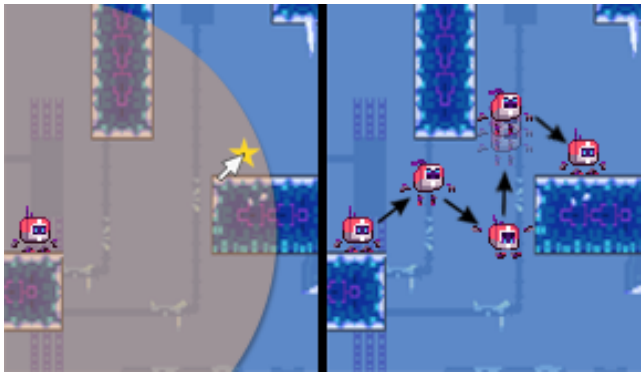

Figure 3: Our implementation of the Click to Move execution assistance. The player clicks anywhere within the radius and the AI handles the execution challenge by moving the player character to the target location.

method helps guide the player regardless of their progress in the game. While this assistance seemed to be effective, it was not clear what the limitations of using it should be, or if using it should be even limited in the first place.

Move to Click 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 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.

Discussion and Future Work

Implementing the assistance methods showed us that before GPAI assistance methods can be claimed as elective, 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 (Andersen et al. 2012). 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 paper, 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.

References

Abd Algfoor, Z.; Sunar, M. S.; and Kolivand, H. 2015. A comprehensive study on pathfinding techniques for robotics and video games. *International Journal of Computer Games Technology* 2015.

Andersen, E.; O'Rourke, E.; Liu, Y. E.; Snider, R.; and others. 2012. The impact of tutorials on games of varying complexity. *Proceedings of the*.

Aytemiz, B., and Smith, A. M. forthcoming. *Proceedings of the 15th International Conference on the Foundations of Digital Games*.

Badia, A. P.; Piot, B.; Kapturowski, S.; Sprechmann, P.; Vitvitskyi, A.; Guo, D.; and Blundell, C. 2020. Agent57: Outperforming the atari human benchmark.

Bakkes, S. C. J.; Spronck, P. H. M.; and van Lankveld, G. 2012. Player behavioural modelling for video games. *Entertain. Comput.* 3(3):71–79.

Berner, C. e. a. 2019. Dota 2 with large scale deep reinforcement learning.

Brown, N., and Sandholm, T. 2019. Superhuman AI for multiplayer poker. *Science* 365(6456):885–890.

Campbell, M.; Hoane, A. J.; and Hsu, F.-H. 2002. Deep blue. *Artif. Intell.* 134(1):57–83.

Case, N. 2018. How to become a centaur. *Journal of Design and Science*.

Cheung, G. K.; Zimmermann, T.; and Nagappan, N. 2014. The first hour experience: how the initial play can engage (or lose) new players. In *ACM SIGCHI symposium on HCI in play*, 57–66. dl.acm.org.

Consalvo, M. 2009. *Cheating: Gaining Advantage in Videogames*. MIT Press.

Cox, A.; Cairns, P.; Shah, P.; and Carroll, M. 2012. 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, 79–88. New York, NY, USA: Association for Computing Machinery.

de Vassal, T. Advanced chess. http://www.ficgs.com/wiki_en-advanced-chess.html. Accessed: 2020-6-2.

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

Fortes, R. P. M. e. a. 2017. Game accessibility evaluation methods: A literature survey. In *Universal Access in HCI. Design and Development Approaches and Methods*, 182.

Holmes, K. 2018. *Mismatch: How Inclusion Shapes Design*. MIT Press.

Hooshyar, D.; Yousefi, M.; and Lim, H. 2018. Data-driven approaches to game player modeling: a systematic literature review. *ACM Computing Surveys (CSUR)* 50(6):90:1–90:19.

Kasparov, G. 2017. Don't fear intelligent machines. work with them.

Klepek, P. 2019. 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. Accessed: 2020-5-8.

Krakovna, V., and Uesato, J. e. a. 2020. Specification gaming: the flip side of AI ingenuity. https://deepmind.com/blog/article/Specification-gaming-the-flip-side-of-AI-ingenuity. Accessed: 2020-5-18.

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

Lebowitz, J., and Klug, C. 2012. *Interactive Storytelling for Video Games*. Taylor and Francis.

Lehman, J. e. a. 2018. The surprising creativity of digital evolution: A collection of anecdotes from the evolutionary computation and artificial life research communities.

Liu, Y. 2018. *Disabled Gamers: Accessibility in Video Games*. Ph.D. Dissertation, Carleton University.

Matt Makes Games. 2018. Celeste. Windows PC version.

Petralito, S. e. a. 2017. 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, 5087–5097.

Pitaru, A. 2008. *E is for everyone: The Case for inclusive game design*. MacArthur Foundation Digital Media and Learning Initiative.

Samuel, A. L. 1959. Some studies in machine learning using the game of checkers. *IBM J. Res. Dev.* 3(3):210–229.

Schwab, B. 2011. Turing tantrums: Ai developers rant! Game Developer Conference.

Segal, J. 2020. Introducing bob's buddy. https://articles.hsreplay.net/2020/04/24/introducing-bobs-buddy/. Accessed: 2020-6-2.

Seth Coster. 2020. Forgiveness mechanics: Reading minds for responsive gameplay.

Shaker, N.; Togelius, J.; and Nelson, M. J. 2016. *Procedural Content Generation in Games*. Springer, Cham.

Silver, D., and Huang, A. e. a. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529(7587):484–489.

Smith, A. M., and Mateas, M. 2011. Computational caricatures: Probing the game design process with AI. In *Workshops at the Seventh AIIDE Conference*.

Tesauro, G. 1995. Temporal difference learning and TD-Gammon. *Commun. ACM* 38(3):58–68.

Thompson, C. 2019. Sekiro: Accessibility in games is about far more than 'difficulty' - IGN.

Togelius, J.; Shaker, N.; Karakovskiy, S.; and Yannakakis, G. N. 2013. The mario ai championship 2009-2012. *AI Magazine* 34(3):89–92.

Vinyals, O., and Babuschkin, I. e. a. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* 575(7782):350–354.

Yannakakis, G. N., and Togelius, J. 2018. *Artificial Intelligence and Games*. Springer, Cham.