Artificial Intelligence in the Game Development Process

Derek Li
BASIS Independent Silicon Valley, San Jose, USA

Email: 2007derekli@gmail.com
Manuscript submitted January 10, 2022; revised May 8, 2022; accepted August 15, 2024; published November 26, 2024
doi: 10.18178/JAAI.2024.2.2.265-274

Abstract: This paper studies three other papers in relation to video games and AI, with a specific focus on the use of AI in the game development process. Developers can enhance their workflow through the use of procedural content generation, game playing AI and AI playtesting, and AI-powered assistive tools. The papers examined include in-depth case studies of specific games and interviews with game developers. This paper concludes on the unpredictable but seemingly exciting future for AI and video game development and the balance between creativity and automation.

Keywords: artificial intelligence, video games, game development

1. Introduction

Over the past century, video games have cemented themselves as not only a successful industry but also as a medium for creativity and cultural exchange. Since their advent, numerous new genres, themes, and game-playing mechanics have appeared in the video gaming world, spurred by rapid progress in computing technology. The growing affordability and advancement of consumer-grade hardware has made video gaming more accessible and their social influence more profound. Progress in fields such as artificial intelligence has also long presented new opportunities for the video game industry, including exciting new means of game playing. During the dawn of AI technologies in the 1950s, software was already being created to play simple games like Tic-Tac-Toe and chess, while in the contemporary era, AI-related features are frequently involved in games such as No Man's Sky (2016) and Diablo III (2012), as stated by Xia and Ye *et al.* [1].

New innovations in the AI space also present opportunities for game developers themselves. For example, the development of experimental game-playing agents can soon help developers play-test their games, reducing the need for logistically challenging and potentially expensive human testing. Computing methods such as procedural content generation give game developers the chance to implement new and creative gameplay mechanics and perhaps reduce the workload of development teams when it comes to asset and level creation. Popular video games such as Super Mario Bros. and Angry Birds have served as testbeds of video game procedural content generation (for these two games, procedural level generation), according to Xia and Ye *et al.* [1]. In the case of Super Mario Bros., for example, new frameworks and machine learning methods allowed for effective level generation and the possibility of co-creation between humans and AI agents.

This paper will explore projects and research regarding the relationship between artificial intelligence and video games, including the potential for AI to assist in the game development process. The findings by Risi and Preuss [2] will be used to discuss game-playing AI and video game procedural content generation in general. Two case studies by Zhao *et al.* [3] will be used to examine the role of AI in playtesting and providing

useful feedback to a development team. Finally, work by Igras-Cybulska *et al.* [4] will be used to discuss developer perspectives on an AI-assistant concept aimed at streamlining the development process for virtual reality multiplayer games.

2. Background

2.1. Procedural Content Generation

In the context of video games, procedural content generation is the creation of objects within the game through algorithms or computing procedures instead of strictly by hand. This can involve random processes but not always. The chart of game AI applications by Risi and Preuss [1] cites levels, maps, assets, and rules under procedural content generation.

A famous example of procedural content generation (specifically, procedural level generation) in video games can be seen in Minecraft, the best-selling video game in history as of this writing, according to Wikipedia contributors [5]. In the game, "infinite" worlds are created via procedural level generation, complete with biomes, terrain features like mountains and seas, man-made structures, and game characters ("mobs").

The worlds are generated through a pseudo-random process. The way a world generates is determined by a seed, such that when the same seed is used to generate two worlds (given that the same version of the game is used), the exact same world will result. If no seed is specified upon world creation, a random seed is used. Of course, due to computing resource limitations, the "infinite" worlds are not truly geographically infinite per se, but large enough such that their limited size will never be of concern to most players.

It goes without saying that this approach to creating game levels takes much of the work of designing many custom levels away from developers, and instead leaves them with the challenge of developing an algorithm capable of repeatedly generating unique but reasonable assets and levels. As stated above, procedural content generation does not apply only to levels and maps but also to assets in general or even rulesets. For a generic example, consider a multiplayer shooter where not only the game maps are different for each match, but also the properties of each weapon and the winning objective of the game.

2.2. Artificial Intelligence Algorithms

There are many algorithmic procedures applicable to video games and AI, some of which will be discussed here.

Reinforcement learning, according to Risi and Preuss [2], is where "an agent learns to perform a task through interactions with its environment and through rewards". This can be used in conjunction with deep neural networks to become deep reinforcement learning, allowing training data to include, as stated by Risi and Preuss [2], "high-dimensional sensory systems". This will be especially applicable to the case studies examined in Section 4.

Risi and Preuss [2] state that evolutionary algorithms seek to emulate real-world biological evolutionary processes, creating "candidate solutions" and picking the best ones of each generation to create more, better candidates, describing them as a "global optimization method". This is also applicable to Section 4.

The A* algorithm is a pathfinding algorithm using a weighted graph to find the optimal path to a certain goal. This goal can be an objective in a video game with the graph weights being various scores or parameters within the game, such as score accumulation. A heuristic may or may not be used to tailor the algorithm's behavior. Further information about this algorithm can be found in work by Hart *et al.* [6]. This will be discussed further in Section 4.

2.3. Assets

In general terms, an asset in a video game is a component in building the game environment at large. Notable examples include 3D models, textures and materials, and sound files. The definition used by Igras-Cybulska [4] for "assets" is "any reusable components or resources used in video game development. These can include a wide range of elements such as code libraries, predesigned graphics, animations, sound effects, music tracks, or even pre-built game level designs". The paper by Igras-Cybulska [4] revolves heavily around developing a custom asset utilizing AI to assist Unity game developers.

3. Game-Playing AI and Procedural Generation

The study by Risi and Preuss [2] offers an overview of research developments in the relationship between artificial intelligence and video games. The paper places much focus on the idea of an AI "learning to play". Two types of "learning to play" are distinguished: that from "states and actions" and that from "pixels".

The former is used to describe traditional board games or similar, where the number of valid actions that can be taken by a player or an opponent is small such that each action or outcome can be individually examined, and the consequences thereof can be analyzed. With a combination of deep learning and the Monte Carlo tree search, it was shown that after the machine learning process had been "seeded" (given human game data to work off of), algorithms like AlphaGo (an AI agent capable of playing Go on the professional level) was essentially able to improve its skill level by playing against itself. This mostly boils down to the "states and actions" nature of games like Go–it is reasonable (in terms of computing resources and time) for an algorithm to be made that evaluates the current "state" of the game and subsequently evaluates the best "action" to take. This type of algorithm has been used to create AI agents that can play games like Chess, Shogi, and Poker. Risi and Preuss [2] conclude that in terms of AI algorithms, self-play may not be a silver bullet for all games. They are, however, clearly well-suited for board games and card games, possibly due to these games having a more reasonable number of possible states and actions such that this kind of algorithmic approach is effective.

The latter, playing from "pixels", such as with a purely digital video game, has shown to be more complex. A method applicable in this case is deep reinforcement learning, which can be used to train AI agents with "high-dimensional input such as images, videos or sounds without the need for human design features or preprocessing," as stated by Risi and Preuss [2]. Often, deep reinforcement learning is implemented by creating a model of the environment and "thinking ahead" and create a plan for the future. The example given by Risi and Preuss [2] is the World Model where an AI agent was able to effectively learn the environment of a challenging and complex video game and use it to create a plan for what to do next. Evolutionary algorithms are also useful in this area particularly when a game is too difficult to be solved with deep reinforcement learning.

Risi and Preuss [2] also describe the possibility of merging the ideas of "states and actions" and "pixels". An example given is AlphaStar, an AI agent able to play the video game StarCraft at a human level. The "states" that are given to the algorithm include the status of various game elements, and the "pixels" includes the minimap at the bottom of the interface. The aforementioned self-play is also utilized in AlphaStar, but only with the "seeding" of human game data as seen earlier. Risi and Preuss [2] note that these applications of "AI playing games" are tailored towards playing specific games rather than learning how to play any given game ("General Video Game Playing"). General video game playing, on the other hand, remains an unsolved challenge. As video games continue to improve and become more complex, they may have the potential to serve as environments to train general intelligence.

Risi and Preuss [2] reason that procedural content generation can help train AI algorithms (such as game-

playing AI) to adapt to a variety of different environments and potentially bring them closer to general intelligence.

Although not explicitly discussed by Risi and Preuss [2], these circumstances can prove to be beneficial for video game developers. As procedural content generation improves, it is likely that game developers will be able to create game levels or maps (or portions thereof) with relative ease. An example of this can be seen in the State of Unreal Keynote at Game Developers Conference 2023 [7], where Unreal Engine's experimental procedural content generation tools are demonstrated. Through "procedural assemblies", the developer is able to place large, pre-made objects into a game level, which adapts to the environment around the assembly and the positioning of the assembly itself. With the tools shown in the Unreal Engine presentation [7], developers and artists only build a portion of the level by hand. This is then scaled up with procedural tools which can significantly increase the efficiency of creating and editing large levels. The method of generation is deterministic and defined by a set of parameters assigned by the developer. One can interpret the set of parameters as a kind of seed for map generation, with the final map being built according to a deterministic algorithm for a given seed. In other words, using the exact same generation parameters twice for building a map will result in the exact same map being built, with the same trees in the same places and the same rocks in the same places, such that a developer can go back to a previously found configuration. Regardless, the final result still feels natural and realistic as long as the parameters entered are reasonable (e.g., if one of the parameters is tree density, a high value would result in extremely overcrowded tree growth, which doesn't look very natural). As for gameplaying AI, the research by Risi and Preuss [2] is also promising for game developers and players. A rather obvious application of more advanced game-playing AI is the creation of more robust and realistic teammates and opponents for more immersive and engaging gameplay. As can be seen with the following topics, game-playing AI can find a useful role in play-testing as well.

4. AI for Playtesting Games

Zhao *et al.* [3] developed human-level game-playing AI agents that attempt to provide useful feedback to game developers, as opposed to simply playing against other humans or playing a game with admirable efficiency. The methods used to train these AI agents and evaluate their performance as well as technical information and context behind the project are described in great detail by Zhao *et al.* [3], but this paper will focus mainly on some of the ideas at work in the training process, a high-level overview of how the AI agents are assessed, and the benefits this could entail for game developers.

Training such an AI agent requires the observation of game states and for the agent to make decisions based on these observations. It is noted by Zhao *et al.* [3] that using a frame buffer for such a purpose is inefficient, and that a "lower-dimensional engineered representation of game state" was created instead. Methods used to achieve this include abstraction to create smaller models of the environment and running the game at a higher clock speed.

The paper by Zhao *et al.* [3] includes two case studies involving "playtesting AI agents": one for providing feedback on player experience and one for providing feedback on player progression.

An important note made by Zhao *et al.* [3] is that these studies aim to model how a human would play the game rather than attempting to achieve a maximally impressive, potentially inhuman result. It is thus that the resulting data can be of value to game designers.

The Electronic Arts game The Sims Mobile is used by Zhao *et al.* [3] for studying the former case as the game environment is more lax in terms of objectives–there is no set path to take per se, and the game interface is designed such that a simple model of the game state can be created for efficient training. Furthermore, the player has the option to select different" careers" which determine the experience a player will have. These career paths have goals that are relatively easy to model and could realistically be achieved by an AI agent.

Ideally, each career choice should be roughly the same difficulty, determining which would require playtesting. Zhao *et al.* [3] studies the methodology and feasibility of gaining this information via an AI agent as opposed to human testers.

To accomplish this, a heuristic value with a weighted sum is created to correlate with a player's progress, which is used with an A* algorithm. An example of such a weighted sum would be that of the career level, experience points, and number of finished events. With knowledge of the full game state and with the game mechanics being fully deterministic, a state transition graph can be used to model the game, which can be solved via the A* algorithm. As mentioned in Section 2, this algorithm is commonly associated with pathfinding, making it appropriate in this scenario. The heuristic value can be customized based on what part of the game the developers intend to observe or study, such as the number of "appointments" needed to complete each career.

The evolutionary strategy solution of an optimization problem model of career progression is used for comparison with the A* results. Rather than using a heuristic, this solution attempts" to achieve a high environment reward against selected objective, e.g., reach the end of a career track while maximizing earned career event points "as stated by Zhao *et al.* [3]. The agent will make actions" based on a probabilistic policy by taking a softmax on the utility...measure of all the actions in a game state," according to Zhao *et al.* [3] The measure of utility is used as the grounds for an "action selection mechanism." Zhao *et al.* [3] continues by stating that an evaluation function is defined and that function is optimized against. The mathematical basis for such a solution is thoroughly described by Zhao *et al.* [3] and thus won't be repeated here.

When the results of both methods are compared, the A* algorithm and the evolutionary strategy performed similarly in many career choices, with one notable outlier being the Barista career. In this case, the evolutionary strategy approach performs significantly more actions (279) compared to the A* algorithm (75). Zhao *et al.* [3] states that "this can be from the fact that this career has an action that does not reward experience by itself, but rather enables another action that does it. This action can be repeated often and can explain the high numbers". It can be inferred that the A* algorithm did not elect to repeat these actions. Additionally, given the nature of the A* algorithm, agents utilizing it play in a manner with no variance. The evolutionary strategy agent results in high variance. Therefore, many runs are needed in order for meaningful results to be produced. In the study by Zhao *et al.* [3], 2,000 runs were used.

Fig. 1 in the paper by Zhao *et al.* [3] shows four careers and the number of actions needed to complete them using both A* and the evolution strategy. Based on the output data, the culinary, fashion, and medical careers are of similar difficulty, and the number of actions needed to complete them varies little on play style. There is evidence to suggest the barista career's difficulty may depend on the style of play as there is a significant difference between the A* algorithm and evolution strategy results, as mentioned above.

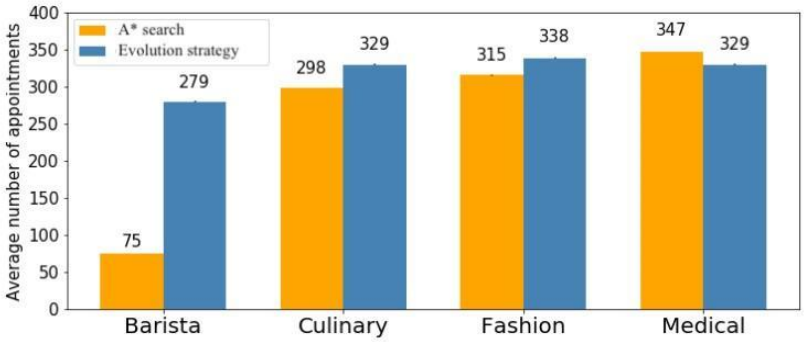

Fig. 1. Number of actions needed to complete a given career path using A* versus the prediction strategy as shown by Zhao *et al.* [3].

The second study by Zhao *et al.* [3] about AI playtesting is in regards to "measuring competent player progression". The study concerns an unspecified "real-time multi-player mobile game, with a stochastic environment and sequential action". This game is more elaborate than The Sims Mobile, thus requiring more skillful strategies in order for the player to succeed. The game involves resource collection and management with the aim of leveling up to progress through the game. Players may also perform upgrades requiring resources; whether or not an upgrade is possible with the player's current resources is available information. This second case study aims to examine how an AI agent would progress through the game, that is, how it balances resources and makes strategic decisions for the developer to get a feel for how a human would play the game.

Zhao *et al.* [3] uses a "simplified state space that contains information about the early game". There are only about 150 continuous and discrete variables in the simplified state space. There are also only about 25" action classes", from which actions can be generated. These actions may not be valid at all times as their validity depends on the present game state. Whether or not an action is valid may not be known information to the player. If an invalid action is attempted, nothing will happen in-game.

The training process involves evaluating valid and invalid actions with the goal being an agent that takes actions and therefore behaves like a human. A feedforward neural network is used in the process with two hidden layers, each with 256 neurons.

An "episode" is created and defined as a goal in the game that could be reached by a skilled human player within a certain time frame. The agent then attempts to perform actions, some of which may be invalid, one of which is defined as "do nothing". A "rewarding mechanism" assigns scores to the actions the agent attempts. For example, one point (a "+1" reward) is added when a goal is reached, one point is deducted ("−1" reward) when an invalid action is attempted, no points ("0" reward) are given for valid actions, and 0.1 points are deducted ("−0.1" reward) when the "do nothing" action is attempted.

Zhao *et al.* [3] uses two versions of the observation space, one "naive" and one "augmented." The naive state space only considers information that is readily available to the player, that is, it can be immediately seen on the game interface. The augmented observation space includes the naive state space but also" information the agent would infer and retain from current and previous gameplays".

Fig. 2. Graphs showing the average cumulative reward for each agent during training and evaluation. The X-axis is the number of iterations as shown by *Zhao et al.* [3].

Four types of agents were trained in this study and their average cumulative return (the previously mentioned rewards) were recorded as a function of the number of iterations, each worth about 60 min of playtime. The average return is capped at 1 because this would indicate a goal having been reached, though this may sometimes be impossible as in some circumstances negative rewards are inevitable.

The four agents are: two Deep Q Network (DQN) agents, one with a complete state space (1), the other with

an augmented observation space (3); two Rainbow (an agent shown to perform better than the standard DQN in a publication by Hessel *et al.* [8]) agents, one with complete state space (2), the other with augmented observation space (4). Rainbow and DQNs are model-free reinforcement learning methods that train agents to seek a "reward" given when favorable actions are performed (Rainbow is actually an extension to the standard DQN methodology with expanded functionality, see the paper by Hessel *et al.* [8] for further information). Zhao *et al.* [3] points out that a benefit of DQNs is that they" can use convolutional function approximators as a general representation learning framework from the pixels in a frame buffer without need for task-specific feature engineering".

It should be noted that the complete state space contains information not available to the regular human player and is therefore likely being used as a control to test the augmented observation space results against. Zhao *et al.* [3] defines augmented spaces as "the space observable by humans in addition to inferred information, which is much smaller than the complete space". Therefore, a poorer result is expected from the augmented space tests than the complete state space.

The results align with these expectations, with the augmented observation space performing worse than the complete state space. It was found that "the agent keeps attempting invalid actions in some cases as the state remains mostly unchanged after each attempt and the policy is (almost) deterministic". A graphical representation of the results can be seen in Fig. 2.

As for feedback to game developers, Zhao *et al.* [3] states that the data indicates that a human would likely have trouble keeping up with such a complex set of actions and their validity. This was further confirmed through additional (real) human testing- that the game interface and how it presented actions and their validity needed to improve such as to not hinder a human player's progress.

It is noted that the methods used in the above case study are not especially time efficient. Zhao *et al.* [3] hopes that once the game is released that it would be possible to use human play data to experiment with imitation learning, making the training process for these agents more efficient.

5. AI Assistance in Developer Workflows

There is a recent paper by Igras-Cybulska *et al.* [4] that explores the prospect of AI tools being directly implemented in game development engines, assisting developers in asset creation and debugging processes. This study is notable as it contains feedback from actual game developers which could be of great help should these AI tools be pursued or popularized. The tool developed and evaluated by Igras-Cybulska [4] is a Metaverse Unity Networking (MUN) asset that uses AI "to facilitate the development of multiplayer VR games in Unity, thereby addressing identified challenges".

The MUN asset discussed by Igras-Cybulska [4] is" supported by a dual component AI framework" which contains a recommendation engine to help with debugging and programming and a behavior analytics system to make NPC avatars behave more believably and realistically in a virtual reality environment.

The next part of the paper considers an "exploratory study" of game developers using the Unity game engine to get an idea of developers' expectations and feedback regarding AI-based game assets, including potential automation and quality-of-life features.

The study included interviews with various developers, sampled to encompass a diverse range of experience levels and backgrounds. These interviews concerned challenges and obstacles developers face, the typical workflows of game developers, and ideas for potential solutions in the form of" supportive assets."

These interviews by Igras-Cybulska *et al.* [4] concluded with the following key points: the testing phase of multiplayer game development often carries the most challenges and thus an asset would be helpful. The asset should be within the control of the developers at all times (e.g. its actions must be approved of by the developers) and it should adjust its behavior to fit a developer's needs. The asset can perform "monotonous

tasks requiring speed and infallibility" automatically, again with developer approval. While the asset can perform these mundane tasks "devoid of creativity" fully automated, they should not hinder creative processes and leave that area fully open to developer input. The potential for a voice assistant is not popular among the developers interviewed. Ideally, the asset should provide real-time, up-to-date feedback, give detailed suggestions as opposed to "binary judgments", and include a "search engine" similar to that of the website Stack Overflow. The asset must act as a "supportive assistant rather than an authoritative future", leaving room for the developer's own ideas and potential rejection of the assistant's suggestions altogether.

A focus group interview was also conducted, in which three participants discussed concepts for an "ideal asset," weighing the positives and negatives of potential ideas, to develop a reasonable approach to tackle the challenges and expectations of developers in the context of such an AI asset. The key points cited by Igras-Cybulska *et al.* [4] (which were discussed during the study) include the following: a sort of modular approach for game elements like server hosting and matchmaking that can be "easily integrated into games", with a lookup tool to find solutions via a "classification system quickly". The case of visually representing developer tasks and workflows was seen positively, with the concept of "step-by-step guides" and visual bot design tools (to simulate real players) being suggested. Recommendations by an AI asset should be implemented in a manner that is friendly to the developer, that is, easily accessible and quality-of-life focused, such as suggestions next to faulting code or the integration of interactive tutorials as part of recommendations, such to create not only a solution but a learning experience for developers. The asset's error-handling role should not be fully automated and should not be disruptive of the developer's work.

The work by Igras-Cybulska *et al.* [4] also concludes that many developers were hesitant to use new assets due to poor experiences with existing assets where the result of their features was underwhelming, the possible learning curve needed to learn such assets, and frequently inadequate documentation for new software.

In the context of the above observations, the aforementioned MUN asset is presented as the solution by Igras-Cybulska *et al.* [4]. The asset has features such as partial automation of the development process, a motion library to create realistic interactions and multiplayer systems more conveniently, and resources for server management in a VR multiplayer environment.

Igras-Cybulska *et al.* [4] indicates two main fields in which the AI in the MUN asset was applied: a recommendation system and a movement analysis system. The recommendation system is essentially a code correction and suggestion assistant that boasts impressive accuracy in the testing described by Igras-Cybulska *et al.* [4]. This should hopefully take much of the workload of mundane programming and debugging off developers, leaving them with more time to develop more creative aspects of the game.

The "automatic movement detection and prediction feature" utilizes machine learning to analyze body motion of users which creates for a more immersive VR experience, owing to its ability to accurately predict user movements.

Developers interviewed by Igras-Cybulska *et al.* [4] stated that their most favored features of the MUN asset included "ready-made prefabs", tutorials to aid in the development process, and automated code review features. Igras-Cybulska *et al.* [4] concludes that the best AI-powered asset for game developers would be one with a significant degree of personalization capabilities and programming assistance tools (such as the recommendation engine).

The MUN asset is currently published online on the project's GitLab Repository by Babiuch [9].

6. Conclusion

There is no doubt that the rapid development of AI technologies in the past decades has led to new advancements in video game development, and will likely continue to do so. Complex machine learning

techniques like reinforcement learning are becoming more popular, and the video game industry is still growing stronger. Not only are game-playing AI agents able to play traditional, "states and action" oriented games but also video games that involve a user interface and continuously evolving, multi-dimensional environment, as seen in the study by Risi and Preuss [2]. New procedural content generation technologies can also help game developers build game levels, making the development process more efficient. Such a technology can be seen in the Unreal Engine Keynote [7], and based on developer interviews and feedback collected by Igras-Cybulska *et al.* [4], there also appears to be demand for such tools related to "readymade prefabs".

The potential for game-playing AI as a means to playtest video games while they are in development is explored in the study by Zhao *et al.* [3]. This paper covered two of the case studies presented by Zhao *et al.* [3], each demonstrating how modern AI methodologies can be used to give useful feedback to developers. In the study by Igras-Cybulska *et al.* [4], interviewees expressed that the testing phase is among the most challenging parts of game development, and thus this was a prime consideration when developing AI assets that would assist developers. The demonstrations by Zhao *et al.* [3] are tailored to specific games in somewhat idealized circumstances, and were designed to work in games that did not focus on VR. However, the basis for these studies (such as the idea of creating simplified game state representations and certain concerns regarding computing resources) can likely be widely applied, and the computing procedures have a relatively general application (e.g., the A* algorithm). Thus, the approach and methodologies by Zhao *et al.* [3] would likely be a good starting point for creating developer-assistant assets like those described by Igras-Cybulska *et al.* [4].

An emphasis made by Igras-Cybulska *et al.* [4] is that game developers are concerned about AI assets and tools potentially being too controlling or eventually stunting creative potential. A point is made that automated or assisted tasks should only be the most mundane and repetitive such as server setup and programming, such that more time can be allocated to more creative pursuits. In a way, this relates to the idea of procedural content generation for maps and the question of whether the creation of large worlds is considered mundane work or a strictly creative construct that should be out of reach of automation tools. In the Keynote by Unreal Engine [7], the demonstration uses a base of hand-built assets by artists before "scaling up" the world using procedural content generation, as determined by a set of parameters. While populating most of the level via procedural content generation, the procedural assemblies used are still created manually. This can potentially be seen as a balance between automating repetitive tasks (after all, adding similar rock formations and trees across a map in a pseudo-random way by hand is extremely tedious and likely isn't creatively stimulating) and preserving the creative freedom of game designers and artists.

It is likely that progress in the field of AI and video games will only continue to accelerate. Powerful computing hardware will likely become more accessible, making studies like those seen in the study by Zhao *et al.* [3] more applicable. As indicated by Risi and Preuss [2], procedural content generation tools have the potential to provide grounds for the development and testing of different AI solutions, and perhaps even general game-playing AI in the future. Software like the MUN asset described by Igras-Cybulska *et al.* [4] are already being released onto the market, with the developer interviews showing considerable demand in this space. While the future of any field of study is without doubt uncertain, the reader should view the future of AI and video games, especially the role of AI in assisting developers with their workflow, with a degree of optimism.

Conflict of Interest

The authors declare no conflict of interest.

Acknowledgment

The author would like to thank Prof. Bill Nace of Carnegie Mellon University for his valuable guidance and advice in the process of writing this paper.

References

[1] Xia, B., Ye, X., & Abuassba, A. (June 2020). Recent research on AI in games. *Proceedings of 2020 International Wireless Communications and Mobile Computing (IWCMC)* (pp. 505–510). doi: 10.1109/IWCMC48107.2020.9148327

[2] Risi, S., & Preuss, M. (2020). From chess and atari to starcraft and beyond: How game AI is driving the world of AI. arXiv: 2002.10433. Retrieved from https://arxiv.org/abs/2002.10433

[3] Zhao, Y. *et al.* (May 2020). Winning is not everything: Enhancing game development with intelligent agents. *IEEE Transactions on Games PP*, 1–1. doi: 10.1109/TG.2020.2990865

[4] Igras-Cybulska, M., *et al.* (June 2024). Supporting unity developers with an AI-powered asset: Insights from an Exploratory user study on multiplayer game development. *Proceedings of the 1st International Workshop on Designing and Building Hybrid Human–AI Systems (SYNERGY 2024)*, Arenzano (Genoa), Italy.

[5] Wikipedia contributors. (2024). List of best-selling video games—Wikipedia, The Free Encyclopedia. Retrieved from https://en.wikipedia.org/w/index.php?title=List of best-selling video games&oldid=1232332912

[6] Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics, 4(2)*, 100–107. doi: 10.1109/TSSC.1968.300136

[7] Unreal Engine. (2023). State of Unreal—GDC 2023—Epic Games. Presented at Game Developers Conference (GDC) 2023 by Epic Games, the creators of Unreal Engine. Retrieved from https://youtu.be/gcElD8KvDLs?t=501

[8] Hessel, M. *et al.* (2017). Rainbow: Combining improvements in deep reinforcement learning. arXiv: 1710, 02298 [cs.AI]. Retrieved from https://arxiv.org/abs/1710.02298

[9] Babiuch, P. (December 2022). Metaverse-unity-networking. Retrieved from https: //gitlab.com/public-repos9/metaverse-unity-networking