Proceedings of the Nineteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2023)

Dynamic Difficulty Adjustment via Procedural Level Generation Guided by a Markov Decision Process for Platformers and Roguelikes

Colan F. Biemer

Northeastern
biemer.c@northeastern.edu

Abstract

Procedural level generation can create unseen levels and improve the replayability of games, but there are requirements for a generated level. First, a level must be completable. Second, a level must look and feel like a level that would exist in the game, meaning a random combination of tiles that happens to be completable is not enough. On top of these two requirements, though, is the player experience. If a level is too hard, the player will be frustrated. If too easy, they will be bored. Neither outcome is desirable. A procedural level generation system has to account for the player's skill and generate levels at the correct difficulty. I address this issue by showing how a Markov Decision Process can be used as a director to assemble levels tailored to a player's skill level, but I've only demonstrated that my approach works with surrogate agents. For my thesis, I plan to build on my past work by creating a full roguelike and platformer and running two player studies to validate my approach.

Introduction

Procedural level generation (PLG) is a sub-area of procedural content generation which aims to generate game levels (Shaker, Togelius, and Nelson 2016). Game levels have two major requirements. The first requirement is that a generated level should look and feel like a level in the target game. For example, a *Mario* (Nintendo 1985) level looks distinctly different from a level built for *Rogue* (A.I. Design 1980). The second requirement is that whatever is generated must be completable. If a level generated is not completable, it is bound to result in a bad experience for the player. This alone is a difficult task, but it isn't enough for a PLG system to be used in an actual game because the player's experience (Thue and Bulitko 2012) and skill (Massoudi and Fassihi 2013) should be taken into account when generating a level: a highly skilled player will be bored if the PLG system outputs an easy level, a low skilled player will be frustrated if the PLG system outputs a hard level. My thesis aims to address this with one pipeline.

My work started by generating level segments for platformers and roguelikes (Biemer, Hervella, and Cooper 2021) using a modified version of MAP-Elites (Mouret and Clune 2015). A full level is assembled by concatenating level segments together. This led to my second paper (Biemer and Cooper 2022) which linked level segments together to form a larger level. A linking algorithm was necessary to ensure that an assembled level was completable and matched the style of the target game. With these segments and links, I built a Markov Decision Process (MDP) to make levels where the MDP acted as a director to assemble levels based on an ideal difficulty for the player (Biemer and Cooper 2023). I ran the MDP on a set of agents which mimicked different play styles and found promising results, even when I tested switching agents.

For my thesis, I plan to address several weaknesses in my work. The first is the lack of human player evaluation. The agents I wrote were simple, but they only approximate the basics of a human player. For my work to be of use, it needs to be validated by real people. The second area for improvement is the poor approximation of difficulty used by the MDP to assemble levels. The third and final piece is to show how the pipeline, which I describe below, can be used in an actual game. To that aim, I am actively developing a platformer and a roguelike that will show how my work can be put together to make a game with a procedural level generation system that adapts to the player by generating levels at an ideal target difficulty.

Related Work

The basic building block of my thesis comes from the work of Dahlskog, Togelius, and Nelson (2014) which showed that an n-gram (Jurafsky 2000) can intake a corpus of levels, where a level is broken down by columns, and output levels that look and feel like the original game. A problem with using n-grams to generate levels is that an n-gram with an n that is too large will memorize whole sequences while an n that is too small will generate seemingly random sequences, and, regardless of the size of n, there is no guarantee that a level generated by an n-gram is completable.

MAP-Elites (Mouret and Clune 2015) is a quality-diversity genetic algorithm. It works by assessing solutions based on behavioral characteristics (BCs) for diversity (e.g. density or linearity of a level (Smith and Whitehead 2010)) and fitness for quality (e.g. how far an agent can make it through a given level). Each BC represents an axis that has been tessellated to form a grid of n dimensions, where n is

the number of BCs. Within each square of the grid, there may be elites that have been assessed based on fitness. An elite is placed into a given square based on its characteristics. To generate new solutions, elites are selected from the grid, and genetic operators—mutation and crossover—are run. The resulting solution is assessed and placed into the grid. Placement occurs if there is no elite at the target grid square or if the newly generated solution has better fitness than the current elite. The output of MAP-Elites is a corpus of solutions, which are, in my case, levels.

Dynamic difficulty adjustment (DDA) (Hunicke 2005) can take many forms (e.g. adjusting player health, changing NPC behavior, etc.), but the specific form related to my thesis is DDA through level generation. The earliest work that I'm aware of comes from Jennings-Teats, Smith, and Wardrip-Fruin (2010) and they ran a data collection process that had players play a selection of levels and then assess the difficulty. A model was trained on this data and then used to rank generated level segments. With a model to rank a player's skill level, levels were selected to match the player's current skill level. Another work, by González-Duque et al. (2020) used MAP-Elites to generate a grid of levels for multiple agents, where the objective is to find a level with a sixty percent win rate for the target agent. With these grids, they tested how quickly a grid can be updated for a new agent with the intelligent trial-and-error algorithm (Cully et al. 2015). Overall, this is very similar to my work. We both use the MAP-Elites grid as a starting point and an offline reinforcement learning method (in my case, an MDP) in an online environment. However, unlike my work, when working with real players, they need an agent that can reasonably approximate the player for an update, whereas my work does not rely on a surrogate for the player. Gonzalez-Duque, Palm, and Risi (2021) addressed this concern in a follow-up work using Bayesian optimization. They found that their solution could find an ideal level based on a desired target difficulty—difficulty was approximated as the time to solve a level—in approximately four iterations. In comparison to my approach which used an MDP, we found that an MDP took ≈ 7 levels to adapt to a new player when given an extreme case of a player switching, see the paper for more details (Biemer and Cooper 2023).

A different approach is to focus on the player's experience (Thue and Bulitko 2012). This can be seen as a more all-encompassing approach where a generator is optimizing for a wide range of variables such as fun, sense of influence, etc. Thue and Bulitko use an MDP to represent their game and use a manager to observe the player and make modifications to the MDP. For example, the player intensity is tracked, one variable is the player's mouse movement, and if that value exceeds a threshold, then the MDP is modified to stop spawning enemies for a certain time. A more recent approach (Shu, Liu, and Yannakakis 2021) developed a model to track the player's experience and that model was used to evaluate future levels in a reinforcement learning process.

Work to Date

Gram-Elites: N-Gram Based Quality Diversity Search

Gram-Elites (Biemer, Hervella, and Cooper 2021) builds on top of MAP-Elites by modifying genetic operators to be based on n-grams, see the code[1] or paper for more information. The result is that every level generated is guaranteed to be well-formed, meaning there are no broken in-game structures, such as malformed pipes in *Mario*. We tested Gram-Elites against MAP-Elites for two platformers and a roguelike, and found that Gram-Elites created more levels that were completable and well-formed in less time for each game.[2]

On Linking Level Segments

When we attempted to connect the level segments built with Gram-Elites to form a larger level, we found that even if a level segment was guaranteed to be playable, it did not mean that the concatenation of two or more completable level segments was itself completable (Biemer and Cooper 2022). The solution[3] we developed had two steps: structure completion and a tree search for completability. Structure completion used n-grams to ensure that the level segments being connected didn't have incomplete structures at either side and if either did, an n-gram completed those structures. After structure completion, completability was tested with an agent. If the level was not completable, a tree search ran on level slices (column or row of a level depending on the game's orientation) that did not have structures—the tree search can be sped up by manually defining these level slices to reduce the search space—until a max depth was reached or a completable level was found. A link generated was guaranteed to result in a completable and well-formed level.

To test linking, we used the resulting grids from Gram-Elites for all three games. Linking was tested for neighbors (i.e. four cardinal directions in two dimensions) in the grid and compared to concatenation. For the horizontal platformer and roguelike, linking was a hundred percent effective in generating links. For the vertical platformer, it was ninety-nine percent effective, whereas concatenation only resulted in a completable level eleven percent of the time. We also tested using the links generated to form a level of more than two segments. In the case of both platformers, this was completely fine. For the roguelike, though, there were long-term dependencies due to a stamina mechanic which made linkers built for two-level segments fail when used for larger levels. We proposed one solution which required the link generated to have an item that gave stamina back to the player, but the more viable solution would be to run the linking algorithm on all three level segments.

Level Assembly as a Markov Decision Process

When the past two works are put together, the result is a graph where Gram-Elites builds the nodes (level segments)

[1] https://github.com/bi3mer/GramElites

[2] https://github.com/bi3mer/GramElitesData

[3] https://github.com/bi3mer/LinkingLevelSegments

and the linking algorithm builds the edges (links), and this graph can be used as the structure for an MDP to generate levels for dynamic difficulty (Biemer and Cooper 2023).[4] The goal of the MDP is to generate levels that maximize its reward, where the reward is a mixture of the designer reward, which we define as difficulty and player reward. Starting with designer reward, we averaged the behavioral characteristics of a level segment as a stand-in for difficulty; this is a limitation that I discuss in the Future Work section. The player reward represents what the player enjoys and we created surrogate player models to represent different kinds of enjoyment (e.g. likes hard levels) and skills (e.g. low skill). The MDP starts biased completely towards the designer and adapts to the player both in terms of skill and preferences.

To form a level with the graph, there is a start node and a death node. The start node is where level formation begins. Any node that the player has previously visited—meaning that the player played that level segment and completed all of it—has an edge from the start node. From there, the edges are based on linking. In testing, we compared our MDP with a director that randomly selected nodes and a greedy director which selected the highest reward. We found that the MDP outperformed both directors in terms of reward, but failed to adapt if there was a player switch. To remedy this, we introduced a modified optimization algorithm which we called adaptive policy iteration. This algorithm removed neighbors from the start node based on how often the player failed. The result was a director that performed better than the base MDP and one that quickly adapted to new players even in the most extreme case of a switch between a highly-skilled player that liked difficult levels and a low-skilled player that liked easy levels.

Future Work

Agent-based simulations are an interesting testbed but have many limitations (Manzo and Matthews 2014). The agents that I developed were simplistic by design. They had random chance built into them, but their skills did not improve or regress as they played. They did not grow frustrated by levels that were too hard or show any signs of boredom playing levels that were too easy. The result was an interesting algorithm on paper, but not useful because it was not validated by real players.

To address this, I plan to run a player study on two games I am developing, where a greedy director is compared to an MDP-based director. The first game is a simple web-based platformer, similar to *Mario*, to test my existing work. The second game takes my work a step further and is a web-based roguelike. The player has to find a set of items to unlock a door and proceed deeper into the dungeon. It builds on top of all the previous work by using segments generated with Gram-Elites and combining them with links to form a graph. The MDP selects rooms to be placed in a dungeon. An extra layer of difficulty based on player progression is added through a layout generator which tells the MDP where rooms need to be placed. Hallways between rooms are filled in based on the layout. This changes the requirements for a level generated where completability is based on whether an agent can access a hallway to the north, south, east, or west from one or more of the corresponding hallways. A room must have at least one such path, but more allows for them to be used in different contexts of the layout. The layout itself guarantees that a path exists between every room. The rooms selected cannot invalidate those paths. The resulting dungeon level will be fully playable, but there is one extra addition that I'm working on. I am designing the agent to not use any in-game items (e.g. the lightning scroll, which shoots lightning and automatically kills the closest, in-sight enemy) to guarantee that a level is completable. The result will be that every dungeon generated has a guaranteed path where the player does not have to interact with or damage any enemy.

For both games, I need to use an improved measure of difficulty for the MDP to optimize around.[5] One approach is to have an agent solve a level and then solve the level again, but requirements like keys are removed. The difference between path lengths can be used as an indicator of difficulty (Weeks and Davis 2022). In my work, though, I need to validate a measure of difficulty before using it. As a result, I am planning two player studies—one for the platformer and one for the roguelike—that evaluate difficulty. Depending on how many levels I can feasibly have evaluated, I may be able to take the approach that Reis, Lelis, and Gal (2015) took and have every level segment evaluated. If not, I will either train a model to predict difficulty based on heuristics as input (Jennings-Teats, Smith, and Wardrip-Fruin 2010; Wong et al. 2012) or on the level as input (Guzdial, Sturtevant, and Li 2016). I believe the former approach is of more value as I will be able to identify what heuristics will be useful in identifying what makes a game difficult.

[4] https://github.com/crowdgames/mdp-level-assembly

[5] Another possibility is to use a multi-objective MDP where multiple skills are used to represent different kinds of difficulty, but I believe that this approach is out of scope for my thesis.

References

A.I. Design. 1980. Rogue.

Biemer, C.; Hervella, A.; and Cooper, S. 2021. Gram-Elites: N-Gram Based Quality-Diversity Search. In *Proceedings of the FDG workshop on Procedural Content Generation*, 1–6.

Biemer, C. F.; and Cooper, S. 2022. On Linking Level Segments. In *2022 IEEE Conference on Games (CoG)*, 199–205. IEEE.

Biemer, C. F.; and Cooper, S. 2023. Level Assembly as a Markov Decision Process. *arXiv preprint arXiv:2304.13922*.

Cully, A.; Clune, J.; Tarapore, D.; and Mouret, J.-B. 2015. Robots that can adapt like animals. *Nature*, 521(7553): 503–507.

Dahlskog, S.; Togelius, J.; and Nelson, M. J. 2014. Linear levels through n-grams. In *Proceedings of the 18th International Academic MindTrek Conference: Media Business, Management, Content & Services*, 200–206.

González-Duque, M.; Palm, R. B.; Ha, D.; and Risi, S. 2020. Finding game levels with the right difficulty in a few trials

through intelligent trial-and-error. In *2020 IEEE Conference on Games (CoG)*, 503–510. IEEE.

Gonzalez-Duque, M.; Palm, R. B.; and Risi, S. 2021. Fast game content adaptation through Bayesian-based player modelling. In *2021 IEEE Conference on Games (CoG)*, 01–08. IEEE.

Guzdial, M.; Sturtevant, N.; and Li, B. 2016. Deep static and dynamic level analysis: A study on infinite Mario. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 12, 31–38.

Hunicke, R. 2005. The case for dynamic difficulty adjustment in games. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*, 429–433.

Jennings-Teats, M.; Smith, G.; and Wardrip-Fruin, N. 2010. Polymorph: dynamic difficulty adjustment through level generation. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, 1–4.

Jurafsky, D. 2000. *Speech & language processing*. Pearson Education India.

Manzo, G.; and Matthews, T. 2014. Potentialities and limitations of agent-based simulations. *Revue française de sociologie*, 55(4): 653–688.

Massoudi, P.; and Fassihi, A. H. 2013. Achieving dynamic AI difficulty by using reinforcement learning and fuzzy logic skill metering. In *IGIC*, 163–168.

Mouret, J.-B.; and Clune, J. 2015. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*.

Nintendo. 1985. Super Mario Bros.

Reis, W. M. P.; Lelis, L. H. S.; and Gal, Y. K. 2015. Human computation for procedural content generation in platform games. In *2015 IEEE Conference on Computational Intelligence and Games (CIG)*, 99–106.

Shaker, N.; Togelius, J.; and Nelson, M. J. 2016. *Procedural content generation in games*. Springer.

Shu, T.; Liu, J.; and Yannakakis, G. N. 2021. Experience-driven PCG via reinforcement learning: A Super Mario Bros study. In *2021 IEEE Conference on Games (CoG)*, 1–9. IEEE.

Smith, G.; and Whitehead, J. 2010. Analyzing the expressive range of a level generator. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, 1–7.

Thue, D.; and Bulitko, V. 2012. Procedural game adaptation: Framing experience management as changing an MDP. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 8(2): 44–50.

Weeks, M.; and Davis, J. 2022. Procedural dungeon generation for a 2D top-down game. In *Proceedings of the 2022 ACM Southeast Conference*, 60–66.

Wong, H.; et al. 2012. Rating logic puzzle difficulty automatically in a human perspective. In *Proceedings of DiGRA Nordic 2012 Conference: Local and Global–Games in Culture and Society*.