Pathfinding Agents for Platformer Level Repair

Seth Cooper and Anurag Sarkar
Northeastern University
se.cooper@northeastern.edu, sarkar.an@northeastern.edu

Abstract

A key concern for procedural level generation is ensuring that generated levels are in fact playable. One approach for determining playability is using pathfinding agents to test if generated levels can be completed. This typically involves following an iterative loop of generating candidate levels until the agent is able to successfully complete a playthrough, discarding the failed level and starting the generation process over each time the agent fails. In this paper, we present a new approach for generating playable levels that leverages pathfinding agents, not to simply test levels for playability, but to perform level repair to fix generated levels that are unplayable. By augmenting a simple pathfinding agent to be able to take moves that would require modifying the level, our approach can fix unplayable levels without having to regenerate a new level from scratch. We compared our repair agent to a default agent on three PCGML level generators each for *Super Mario Bros.* and *Kid Icarus*, and found the repair agent was able to fix the majority of levels, taking time that ranged from similar to the default agent to around 4x its time.

Introduction

Procedural content generation (PCG) refers to a set of approaches for algorithmically generating content for games. An important consideration for PCG methods is ensuring functionality of generated content, which in the specific context of game level generation means ensuring that generated levels are playable. One approach for doing so, particularly for techniques involving search-based PCG (Togelius et al. 2011) and PCG via machine learning (PCGML) (Summerville et al. 2018), is to follow a generate-and-test methodology. Here, as the name suggests, a test is used to check if a generated level satisfies desired constraints (such as playability). If this test fails, the level is discarded, a new level is generated, and the process continues until a desired number of valid levels has been generated.

A commonly used test in such approaches is the use of pathfinding agents to check for playability, filtering out those generated levels that the agents fail to complete. As discussed in the related work, this approach has been used by several prior works for generating playable levels; it restarts the generation process no matter how the agent failed to complete the level. However, not all unplayable levels are generated the same, and a simple set of modifications might be enough to render some of them playable without having to regenerate them from scratch. Additionally, such modifications could be determined by leveraging information about where and how the agent would need to do something impossible to complete the level and thus the ability to repair an unplayable level could be incorporated into the pathfinding agent itself, thereby aiming to replace the generate-and-test loop with a single generate-and-fix pass, or increase the likelihood of the test passing.

In this paper, we introduce a new pathfinding agent capable of repairing levels as part of testing levels for playability. The agent is able to make some impossible moves as it searches for a path through the level, and, if the resulting path includes these moves, they are used to modify the level in an attempt to repair it. In our approach, the level remains unmodified as the agent searches for a path, which can lead to finding paths with contradictions in them, such as both moving through a non-solid tile and then jumping off of it as if it were solid. We attempt to guide the agent away from such moves.

For evaluation, we compared our repair agent to a default agent on three different PCGML level generators (two n-grams and one VAE) each for *Super Mario Bros.* and *Kid Icarus*, and found the repair agent to be able to repair the majority of levels. The execution time for the repair agent ranged from being similar to that of the default agent to approximately 4x its time.

Related Work

A body of prior PCG research has utilized pathfinding agents, specifically based on A^* search, to help generate playable levels. This has been particularly popular in search-based PCG techniques (Togelius et al. 2011) where playability as determined by the A^* agent is incorporated into the objective function being optimized to search for levels. Additionally, use of A^* agents has also been popular in PCGML techniques where these agents have been used to annotate levels with A^* paths with the hope that models trained on them will generate levels with such traversable paths as well.

Overall, there have been two popular approaches to pathfinding agents that have been used in a number of works

for evaluating level playability and generating playable levels. First, Robin Baumgarten's physics-simulation agent (Togelius, Karakovskiy, and Baumgarten 2010), used with Markov models (Snodgrass and Ontañón 2014), GANs (Volz et al. 2018), Quality-Diversity algorithms (Withington 2020), the GVG-AI domain (Green et al. 2018; 2020) and the Mario AI framework (Khalifa et al. 2019). Second, Adam Summerville's tile-based agent (Summerville, Philip, and Mateas 2015), used with LSTMs (Summerville and Mateas 2016), Markov models (Snodgrass and Ontañón 2016; 2017) and VAEs (Sarkar et al. 2020). Note that in all these cases, the agent is used to determine whether a generated level is playable or not rather than to modify the level. Our work differs in that our agent can actively attempt to repair a generated level if it is not playable.

A number of prior works have focused on repairing generated levels. Jain et al. (2016) trained autoencoders to generate and repair Mario levels while both Mott, Nandi, and Zeller (2019) and Shu et al. (2020) attempted to repair defective GAN-generated Mario levels, with the former using LSTMs and the latter using a hybrid approach combining a multi-layer perceptron and genetic algorithms. We also attempt to fix defective generated levels in this work but utilize our pathfinding agent, instead of a trained model, to make repairs.

Method

Here we describe the basic approach we used to allow a pathfinding agent to attempt to repair a platformer level.

For pathfinding, we started with the pathfinding agent provided by the Video Game Level Corpus (VGLC) (Summerville et al. 2016). This tile-based pathfinding agent considers two types of tiles: solid and non-solid. We made a few minor updates, such as adding support for start and goal tiles, and removing some shortcuts the agent could take to move through solid tiles. We refer to this as the *default agent*. Running this agent on a level gives a path from the start to the goal, if there is one, which determines if the level is playable.

We augmented the default agent to create the *repair agent*. The repair agent can take additional actions: in some cases it can move through solid blocks or jump while above non-solid blocks. However, these actions have a very high cost, high enough that the agent will not take them unless necessary. After the repair agent has found a path, it goes back through the path and looks for any high cost moves. If it finds any, it then *modifies* the level appropriately: modifying a solid tile to be non-solid if it was moved through, and modifying a non-solid tile to solid if it was jumped off of.

While this approach is relatively quick, it can result in the repair agent making contradictory moves and modifications. For example, the agent can jump up through a non-solid tile, and then jump again from above that same tile, which would result in a modification to place a solid tile, even though if that solid tile had been there originally, the first jump would not have been possible.

To reduce the chance of these kinds of contradictory moves, we prevent the agent from making some moves in cases that seem likely to introduce contradictions, based on which *tiles* have already been visited by the pathfinding search. These include preventing the agent from walking on solid tiles that have been visited, or starting a jump above solid or non-solid tiles that have been visited. We also prevent moves that would modify the start and goal tiles and those under them. Notably, this does mean that the agent will have different moves available depending on the order in which nodes (and thus tiles) are visited by the pathfinding search; thus we do not expect the agent to find an optimal set of modifications. To check if the level is playable by the default agent with the modifications, we confirm the default agent can follow the path the repair agent took after applying modifications.

If the repair agent can find a path through the level, and the default agent can follow that path after applying modifications, we consider the level *repaired*.

level	playable			modifications (in repaired levels)			agent time (ms)	
	w/o rep.	with rep.	not	min	med	max	default	repair
SMB N1-1	995	5	0	1	1	1	282 (40)	338 (46)
SMB N1-3	782	218	0	1	1	2	235 (47)	333 (82)
SMB VAE	953	47	0	1	1	3	259 (47)	320 (55)
KI N1	6	893	101	1	7	22	150 (55)	612 (137)
KI N4	1	801	198	1	7	24	156 (49)	609 (156)
KI VAE	31	840	129	1	4	19	177 (83)	583 (140)

Table 1: Summary of results for attempting to repair 1000 generated levels for each generator. The number of levels that were playable without repair, with repair, and were not playable after attempted repair; the minimum, median, and maximum number of modifications used in levels that were repaired; and the mean and standard deviation of time taken by the default agent and the repair agent.

Analysis and Discussion

To evaluate the performance of this approach, we ran the repair agent and the default agent on levels generated via two PCGML approaches, trained using levels from the games *Super Mario Bros.* (SMB) and *Kid Icarus* (KI), taken from the VGLC (Summerville et al. 2016).

First, we used n-grams (Dahlskog, Togelius, and Nelson 2014). For SMB, we generated levels using trigrams; one trained on level 1-1 *(SMB N1-1)* and one on level 1-3 *(SMB N1-3)*. For KI, we generated levels using bigrams, one trained on level 1 *(KI N1)* and one on level 4 *(KI N4)*. Levels generated were 100 columns/rows for SMB/KI.

Second, we used variational autoencoder (VAE)-based sequential level generation models for SMB *(SMB VAE)* and KI *(KI VAE)* as described in Sarkar and Cooper (2020). The model for each game was trained on 16x16 level segments of that game extracted from all their levels in the VGLC and is capable of generating whole, continuous levels consisting of these 16x16 segments. Levels contained 6 segments each, resulting in 96 columns/rows for SMB/KI.

For each generator, we generated 1000 levels. We ran the default agent and repair agent on each to see which lev-

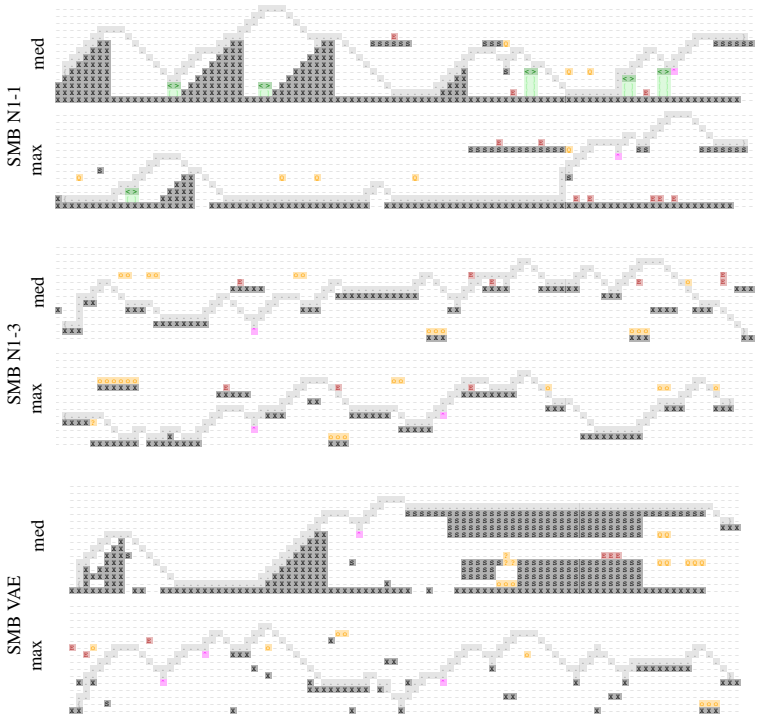

Figure 1: Example generated *Super Mario Bros.* levels using each generator with the median and maximum number of modifications (most of which are 1). Paths are shown as dots; removed solids are cyan + and added solids are magenta ^.

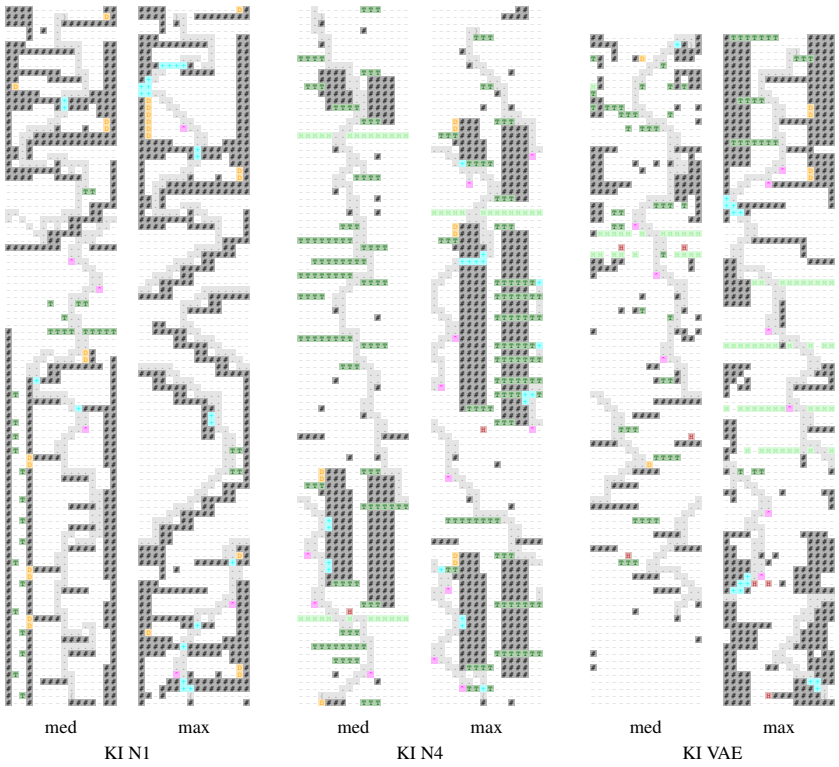

Figure 2: Example generated *Kid Icarus* levels using each generator with the median and maximum number of modifications. Paths are shown as dots; removed solids are cyan + and added solids are magenta ^.

els were playable by the default agent and which could be repaired to be playable by the repair agent, and gathered modification and timing information. The agents were implemented in Python.

For KI, we updated the agents to allow wrapping around the sides of the level; for simplicity, we considered one-way platforms as solid and moving platforms as non-solid, and used the jump definitions from SMB.

A summary of the results is given in Table 1, with example repaired SMB levels in Figure 1 and KI levels in Figure 2.

In the analysis, we found that most levels were able to be repaired to be playable. The SMB levels generated were mostly playable already, but the repair agent was able to make all of them playable. Only one or two modifications were needed to repair the levels, usually adding a solid tile to make a jump possible.

On the other hand, the KI levels generated were almost entirely unplayable initially. The repair agent was able to get to around 80-90% playable levels. Several modifications were often needed to repair KI levels, sometimes carving out several solid tiles to make a path. We suspect that these paths may be due to certain modifying moves being prevented by visited tiles, which the agent compensates for by modifying solid tiles. It is possible that the low playability of the generated KI levels was related to, for example, the simplicity of the n-gram model creating large open vertical sections. Playability may also have been impacted by simplifying the platform mechanics and using SMB jumps; a higher-fidelity approximation of KI mechanics could have found the levels to be more playable.

We found a moderate increase in the relative time taken by the repair agent, which appears to increase as the proportion of initially playable levels decreases. For SMB N1-1, which had 99.5% initially playable levels, the repair agent took only 120% of the time of the default agent. For KI N1, which had only 0.6% initially playable levels, the repair agent took 408% of the default agent's time. We expect this is due to the relatively short time in which the unplayable levels can be rejected by the default agent.

Conclusion

In this work, we proposed a pathfinding agent that can attempt to repair platformer levels by modifying solid tiles to be non-solid or vice versa. In an analysis of procedurally generated levels, we found the repair agent can improve playability at the cost of an increase in time over an agent that only evaluates playability.

There are a variety of future directions and variations on this work. Other types of modifications may be possible, such as placing items, and other types of games could be explored with other tile types. Different pathfinding search heuristics may influence the modifications made. More accurate models of player movement (than tile-based approximations) can also be explored. Future work could also examine the impact of repair on expressive range (Smith and Whitehead 2010), for example, comparing the expressive range of the generated levels pre- and post-repair.

While we considered a level repaired with any number of modifications, designers might want to place a limit on the number, type, or location of modifications allowed before rejecting a level. The application of repair agents as a support tool for hand-authored levels, along with the types of feedback they can provide to designers on their levels, may be an area for further study.

References

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.

Green, M. C.; Khalifa, A.; Barros, G. A.; Nealen, A.; and Togelius, J. 2018. Generating levels that teach mechanics. In *Proceedings of the 13th International Conference on the Foundations of Digital Games*, 1–8.

Green, M. C.; Mugrai, L.; Khalifa, A.; and Togelius, J. 2020. Mario level generation from mechanics using scene stitching. *arXiv:2002.02992 [cs]*.

Jain, R.; Isaksen, A.; Holmgård, C.; and Togelius, J. 2016. Autoencoders for level generation, repair and recognition. In *Proceedings of the ICCC Workshop on Computational Creativity and Games*.

Khalifa, A.; Green, M. C.; Barros, G.; and Togelius, J. 2019. Intentional computational level design. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 796–803.

Mott, J.; Nandi, S.; and Zeller, L. 2019. Controllable and coherent level generation: A two-pronged approach. In *AIIDE Workshop on Experimental AI in Games*.

Sarkar, A., and Cooper, S. 2020. Sequential segment-based level generation and blending using variational autoencoders. In *Proceedings of the 11th Workshop on Procedural Content Generation in Games*.

Sarkar, A.; Summerville, A.; Snodgrass, S.; Bentley, G.; and Osborn, J. 2020. Exploring level blending across platformers via paths and affordances. In *Sixteenth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Shu, T.; Wang, Z.; Liu, J.; and Yao, X. 2020. A novel CNet-assisted evolutionary level repairer and its applications to Super Mario Bros. *arXiv preprint arXiv:2005.06148*.

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*.

Snodgrass, S., and Ontañón, S. 2014. Experiments in map generation using Markov chains. In *Proceedings of the 9th International Conference on the Foundations of Digital Games*.

Snodgrass, S., and Ontañón, S. 2016. Controllable procedural content generation via constrained multi-dimensional Markov chain sampling. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, 780–786.

Snodgrass, S., and Ontañón, S. 2017. Learning to generate video game maps using Markov models. *IEEE Transactions on Computational Intelligence and AI in Games* 9(4):410–422.

Summerville, A., and Mateas, M. 2016. Super Mario as a string: Platformer level generation via LSTMs. In *Proceedings of 1st International Joint Conference of DiGRA and FDG*.

Summerville, A. J.; Snodgrass, S.; Mateas, M.; and Ontañón, S. 2016. The VGLC: The video game level corpus. In *Seventh Workshop on Procedural Content Generation at First Joint International Conference of DiGRA and FDG*.

Summerville, A.; Snodgrass, S.; Guzdial, M.; Holmgård, C.; Hoover, A. K.; Isaksen, A.; Nealen, A.; and Togelius, J. 2018. Procedural Content Generation via Machine Learning (PCGML). *IEEE Transactions on Games* 10(3):257–270.

Summerville, A. J.; Philip, S.; and Mateas, M. 2015. MCMCTS PCG 4 SMB: Monte Carlo tree search to guide platformer level generation. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*.

Togelius, J.; Yannakakis, G. N.; Stanley, K. O.; and Browne, C. 2011. Search-based procedural content generation: A taxonomy and survey. *Computational Intelligence and AI in Games, IEEE Transactions on* 3(3):172–186.

Togelius, J.; Karakovskiy, S.; and Baumgarten, R. 2010. The 2009 Mario AI competition. In *IEEE Congress on Evolutionary Computation*, 1–8. ISSN: 1941-0026.

Volz, V.; Schrum, J.; Liu, J.; Lucas, S. M.; Smith, A.; and Risi, S. 2018. Evolving Mario levels in the latent space of a deep convolutional generative adversarial network. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 221–228. ACM.

Withington, O. 2020. Illuminating Super Mario Bros: quality-diversity within platformer level generation. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 223–224.