Content Reinjection for Super Metroid

Ross Mawhorter, Batu Aytemiz, Isaac Karth, and Adam Smith
University of California, Santa Cruz
1156 High Street
Santa Cruz, CA, 95064

Abstract

Academic procedural content generation (PCG) efforts often yield plausible game content but stop short of fully integrating it into the games that inspired it. This misses opportunities to discover game- and platform-specific constraints that were previously ignored in evaluations of playability (e.g. how invisible objects in a level design are used to explicitly control camera movement). Grappling with existing games can ensure that the PCG community is solving realistic problems, rather than convenient abstractions of them. In this paper, we use technical knowledge from the ROM hacking community along with the WaveFunctionCollapse example-driven generator to reinject controllably-generated level designs into the commercial Super Metroid ROM image (rather than a clone) for execution on physical Nintendo hardware. Our work charts a path for more realistic evaluation of the playability of generated content and highlights challenges for deploying generative methods. These challenges can spark a conversation about the ways that abstractions are used in PCG research.

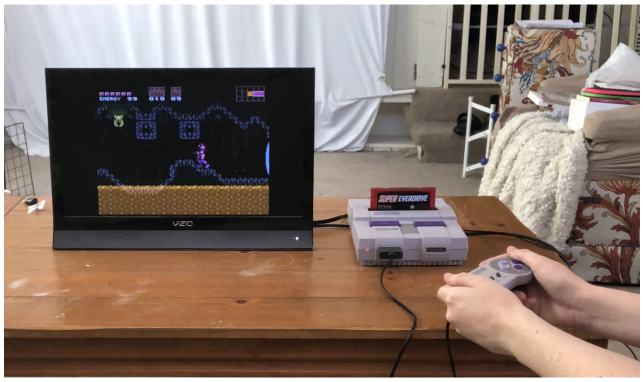

Figure 1: Playing a computer-generated reinjected Super Metroid room. The game is running on the Super Nintendo Entertainment System, loaded from a Super EverDrive X6 flash cartridge.

Introduction

Much of the literature in the field of procedural content generation (PCG) is focused on the problem of generating level designs for videogames (Shaker, Togelius, and Nelson 2016). These systems often attempt to generate level designs in the same style of those in an existing, inspiring game. Data from the original game is sometimes used for training in a machine learning setup (Summerville et al. 2018). However, analyses of the expressive range of these systems often treat the level designs as if they were images: to be passively inspected relative to inspiring inputs or to have item counts and geometric shapes quantified (Smith and Whitehead 2010; Summerville 2018). This analysis only speaks indirectly to playability concerns for the designs, some of which can only be determined by playing through each design in multiple or even all possible ways (Smith, Butler, and Popovic 2013). While Mario-style levels can easily be injected into the open-source game clones like Infinite Mario Bros (Persson 2009) / Infinite Tux (Lewis 2009), gaps between these clones and their inspiration Super Mario Bros (Nintendo Co. Ltd. 1983) go unexamined. While working with the original level data as input provides one kind of research challenge, making sure a generator can output new level data that is perfectly compatible with (safe for reinjection into) the inspiring game provides an orthogonal challenge, one that has so far not yet been addressed by this community.

In this paper, we demonstrate a technique for reinjecting procedurally generated levels into Super Metroid (Nintendo Co. Ltd. 1994). The generated levels can run on the original physical Nintendo hardware, as seen in Figure 1. In order to do this, we use knowledge from the ROM hacking community to identify the format for various data structures within the game. We extract the original game data for a single room of Super Metroid, then use WaveFunctionCollapse (Karth and Smith 2021), an example-guided constrained content generator, to create a new room in the same style. In particular, we apply constraints to the level design to ensure the result is compatible with contextual details assumed by the game. We then reverse the extraction process to reinject the new content, replacing the original room with the generated one. In addition to describing this pipeline in detail, this paper also lays out some of the main challenges and benefits to content reinjection. The engineering challenges of using a particular PCG method that are often abstracted away can actually expose important new research questions.

Background

This paper uses traditional PCG methods in a new domain: consuming and producing data that is directly compatible with the data from an existing game.

Procedural Level Design

A 10-year retrospective on the Procedural Content Generation Workshop (Liapis 2020) identified Super Mario Bros as the most common platformer game targeted. When projects attempt to use data from this game or generate data for it, an abstracted proxy has always been used instead. For example, the Mario level designs available in the Video Game Level Corpus (VGLC) represents human-transcribed level designs in a simplified annotation format intended to abstract across a few different games rather than being accurate to any one of them(Summerville et al. 2016). Meanwhile, when the playability of generated levels is assessed, it is either done by simulation within the Java-clone Infinite Mario Bros (Volz et al. 2018) or using a tile-level abstraction of the game's movement rules that simplifies out notions character momentum (Cooper and Sarkar 2020). In work that targets Super Mario Bros by name, often Infinite Mario Bros is quietly used as a replacement without mention of possible divergences (Lucas and Volz 2019). This approach was codified in the design of the 2010 Mario AI Championship's level design track (Shaker et al. 2011; Togelius et al. 2013). While using a shared testbed makes it easier to make certain kinds of progress in PCG, it leaves room for gaps between the clone and the original game to go unexamined.

Work that modifies the original Super Mario Bros ROM is not unheard of. For example, demonstrations of the Reveal-More (Chang, Aytemiz, and Smith 2019) system involved manually injected changes. The same authors' followup (Chang and Smith 2020) further explored the non-obvious implications of small design changes: moderately altering gravity breaks a *cutscene*, rendering large parts of the game unreachable. These projects show that the playability of a level design does not cleanly reduce to tile-level reachability analysis. Even the idea of a level design as a grid of tiles is itself an abstraction over the game's actual data format. For example, in Super Mario Bros, extended horizontal runs of tiles are a primitive design element (Altice 2017, p. 133). The view of level designs offered by the VGLC suggests more design flexibility than the original games actually support while also not providing enough information to accurately predict playability.

Instead of using this type of abstracted data, we work directly with data from an existing game, Super Metroid. The generator reads and writes data directly to and from the ROM using an existing PCG method that can work accurately with an unspecified data format.

WaveFunctionCollapse

To generate content that is compatible with existing games, a generator must exactly reproduce the kind of data already available for that game. WaveFunctionCollapse (WFC) (Karth and Smith 2021) is an example-driven generator that uses existing game data to define the building blocks and re-composition rules for freshly generated content. In particular, it is a constraint-solving method for which it is easy to mark certain parts of a level design as predesigned and leave the remainder to the generator to fill in in a plausible way. Compared to the use of long short-term memory (LSTM) models (Summerville and Mateas 2016) or as Multi-dimensional Markov Chains (MdMCs) (Snodgrass and Ontanón 2017), WFC allows directly expressing constraints over parts of the level design that are assumed to be in place for the larger game to function properly. While LSTMs and MdMCs can accomodate constrained tiles, that information is only propagated in the direction of generation, creating a problem when only small parts of the level are known rather than an entire prefix (refer to Figure 5).

For content reinjection, many different kinds of generators could be used, but it is important that they be strictly guided by the original game data. In this case, a mistake in a single bit can cause the game to crash or other unexpected behavior. Therefore, a generator that uses discrete tiles is preferred. WFC is in use in several commercial games, such as *Caves of Qud* (Freehold Games 2021) and *Bad North* (Plausible Concept 2018), but this work represents the first effort to use WFC to reinject content into a game not originally designed to showcase PCG.

ROM Hacking

Super Metroid is a platformer videogame released for the Super Nintendo Entertainment System (SNES) in 1994. Players must traverse a large world, collecting items, completing platforming challenges, and using weapons to destroy enemies. Instead of being a single two-dimensional tile grid, The game world is composed of many rooms connected by doors. Figure 2 shows an example level.

Videogames like Super Metroid were commercially distributed in physical cartridges containing read-only memory (ROM) chips with both executable code, as well as game data. In the present, flashcarts (in which rewriteable memory chips replace the ROMs) are used to store ROM images, and interface with the original hardware. Communities exist around making and distributing playable modifications or hacks to these games.[1]

In order to reinject content into ROM images, we need to understand the internal data formats used by the game. In modern games, much of the content is stored in well-documented shared data formats, but Super Metroid and many older games use proprietary data formats that were designed only for use in a single game and as a result are poorly-documented. Fortunately, Super Metroid has a vibrant game-specific ROM hacking community.[2] This group of people has reverse-engineered the code to gain a deep understanding of the internal workings of the game.

This level of understanding has led to a significant non-academic *and* non-commercial application of PCG: Item

[1]https://www.romhacking.net/

[2]https://metroidconstruction.com/

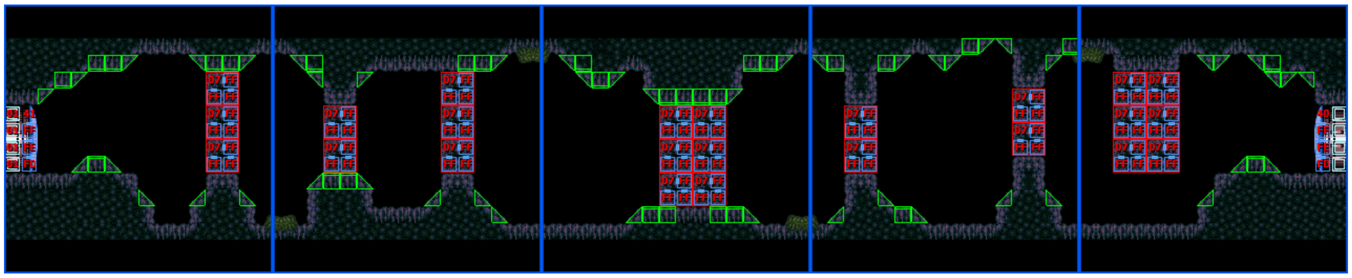

Figure 2: **Original** room design from the commercial game, as seen in the SMILE RF editor. This primarily shows tile graphics data, but any tile with a special Behind The Scenes (BTS) value shows the value in red, or has a green outline if it is a slope. Enemies present in the room are not shown.

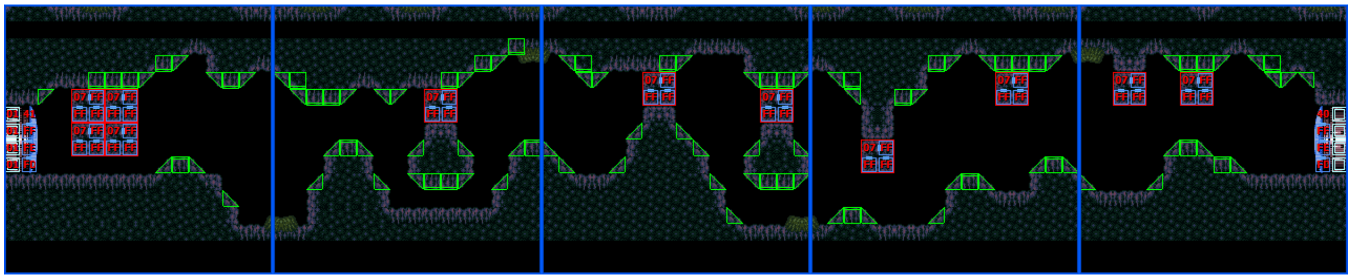

Figure 3: An example **generated** level, as viewed in the SMILE RF editor. The configurations of a few pinned tiles selected in Figure 5 are constrained to match the original level design from Figure 2.

Randomizers. An Item Randomizer is a program that reads in a clean game ROM, and produces a new ROM by moving collectible objects around in the game's world.[3] There is typically logic to ensure that the new placements still result in a playable (completable) game, but players must make new choices about how to traverse the game world. Other randomizers reconnect existing scenes in the game in new ways to create new gameplay experience[4]. To our knowledge, randomizers have not yet attempted to redesign individual scenes at the level of visible tiles in the way that hundreds of many manually created Super Metroid hacks have done.[5]

Despite being almost 30 years old, Super Metroid is still an actively modified game. For example, a recent and glowingly-reviewed ROMhack, V I T A L I T Y,[6] released in 2020, has over 5700 downloads. Using content reinjection we can use and evaluate academic PCG techniques while targeting an audience much larger than that of many researcher-made playable experiences (Treanor et al. 2017).

In this paper we deal with the problem of creating fresh level designs at the tile level. By reading community-made documentation and code, using community-made tools, and learning from community members, we learned how to extract existing content and reinject novel content for Super Metroid. The main challenge of content reinjection is in understanding a game's native data format. For Super Metroid, the internal workings of the game are now particularly well-documented thanks to the extensive work of ROM hackers. For other games, this type of knowledge may or may not be accessible to researchers.

Content Reinjection Demonstration

This section describes how content reinjection is used to demonstrate a kind of design randomization not yet seen in the academic or ROM hacking communities. In particular, the generator will:

- Use only tiles present in the original level design in plausible combinations.
- Enforce the presence of doors and enemies at locations assumed by other parts of the game we are not modifying.
- Yield a tile grid that can correctly load and execute on the original console hardware.

In order to focus on interacting directly with the game ROM, this specification of the problem does *not* require that the level be traversible (e.g. in terms of reaching one door from the other).

Data Format

Level data on the Super Metroid ROM is stored in a compressed format (used by some other Nintendo games, including Super Mario World) associated with the Lunar Compress tool.[7] We can inspect and edit data in this format using

[3]For example http://randommetroidsolver.pythonanywhere.com/ and https://dashrando.github.io/

[4]https://metroidconstruction.com/resource.php?id=102 https://www.worldrandomizer.com/

[5]https://metroidconstruction.com/hacks.php

[6]https://metroidconstruction.com/hack.php?id=623

[7]https://fusoya.eludevisibility.org/lc/

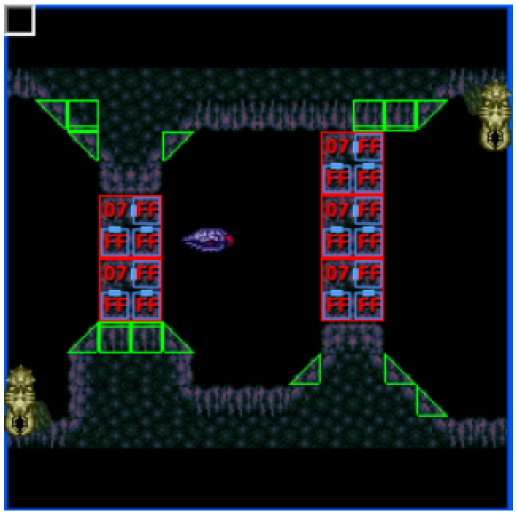

Figure 4: SMILE RF view of one screen of the original game showing the placement of enemies *on top* of tile data. Enemies are not moved in this paper's experiments.

the Super Metroid Integrated Level Editor (ReFactored) or SMILE RF tool created by Scyzer and Jathys.[8]

The decompressed level data is a 2-dimensional array of tiles. Each tile in the decompressed room has up to 40 bits of information, which comprise four main components. First is the tile type, which sets the physical properties of the tile like collision and destructibility. Second is the tile graphics, which indexes into a room-specific graphics table (with up to 1,024 entries). Third is the background graphics index (if present). The last value is known to ROM hackers as "Behind the Scenes" or BTS data. It further differentiates tiles of the same type (e.g. differently shaped slope tiles). The BTS value of a door tile controls which door transition is triggered when the player enters it. Rooms are organized into 16×16 tile screens, with the typical room ranging from 1 to 30 screens.

Figure 2 shows an example room viewed with SMILE RF, which shows the BTS layer with brightly-colored overlays. Figure 4 shows a detail from SMILE RF, with enemy positions overlaid.

In addition to the compressed two-dimensional array of tile data, each room in Super Metroid has additional information, including (but not limited to) where to place specific types of enemies into the room, instructions for camera movement within the room, and a table of door transitions leaving the room and connecting to other rooms. Many existing randomizers focus only on modifying this room-header data. Our present work focuses specifically on replacing the tile data, without modifying these other tables. The curious reader is encouraged to read about the Post-Load Modification (PLM) system by which aspects of a level design can be customized by code executed during live play of the game.[9] This feature, which allows for arbitrary code execution on the SNES main processor, is not used in our current work.

[8]http://sadiztyk.metroidconstruction.com/

[9]https://metroidconstruction.com/SMMM/\#plm-set-pointer

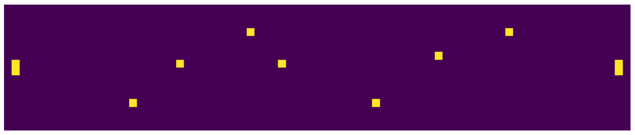

Figure 5: Enemies in *Super Metroid* are not part of the tile grid and cannot easily be generated using WFC. In order to make a coherent and functioning room, we constrained the locations of doors and enemies (shown as yellow tiles) to match the data from the original room.

Reinjection Pipeline

Our content reinjection pipeline begins by extracting the original game data. We used SMILE RF to identify the memory address of the original level data for our specific target room shown in Figure 2. This room, informally known as the Crateria Gauntlet Entrance room, is often traversed from left-to-right much like a Mario level, however it can also be traversed in the opposite direction as the player continues to explore the world after collecting items. We use WFC to generate a new room, but add constraints that improve the playability of the output.

The enemy set for a room is part of the room header information. Since our generator does not modify this data structure, enemies in the generated room will appear in the same positions as in the original room. In order to create a room where enemy placements make sense, we constrain the level data at each enemy position to match the original room (see Figure 5). We also constrain the doors on either side of the room, so that the player may enter and leave the generated room. Enemy and door positions were obtained manually rather than by the generator reading ROM data structures.

Of the possible 40-bit representations for each tile, there are 71 unique tiles used in the original room. From these, we produce the set of 2×2 overlapping tile patterns (329 unique combinations). We then use WFC[10] (with the pinned tile constraints) to generate a new room with the same shape as the original room, the same pattern vocabulary, and using patterns adjacent to one another in only combinations seen in the original data. We used WFC with 2×2 tiles because the destructible blocks in the original level are made of a 2×2 pattern of tiles that cannot be coherently recombined. Keeping the shape of the room the same is important for two reasons. First, if the new room is smaller than the original room, the game will crash after receiving less level data than expected. Second, if the room is larger than the original room, putting the new room back into the ROM might overwrite and garble other level data unless we manually reallocate the other rooms.

After obtaining new grid of tiles using WFC, we compress it, and write the compressed bytes back to the same location in the ROM where the original level data came from. Fortunately, the reverse-engineered compression algorithm

[10]We use the ASP-based rational reconstruction of WFC described by Karth and Smith (2017).

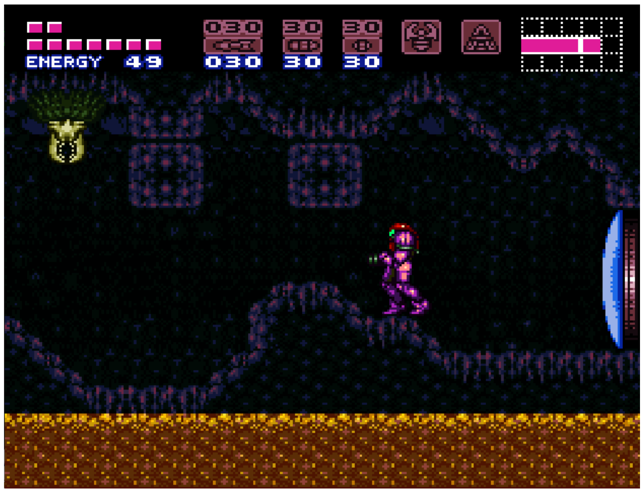

Figure 6: Playing the generated room using the `bsnes` emulator.

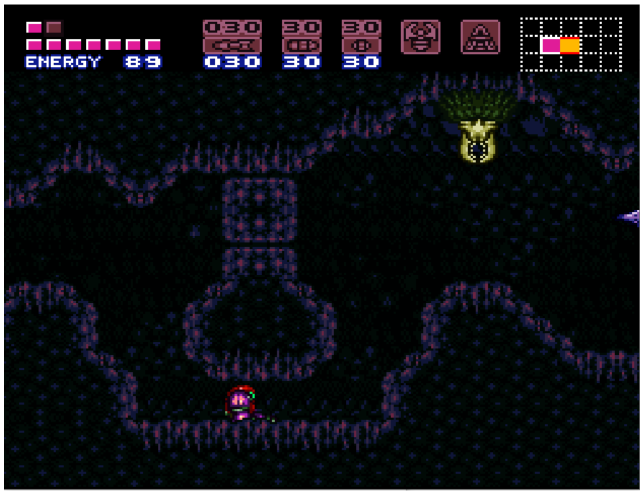

Figure 7: The player can clip into the floor and become stuck at one point in the generated level.

is more efficient than the original algorithm used to create the game, so the generated level will usually be small enough to fit back into the original location. To make the new room easier to demonstrate on the SNES hardware, we made several other small modifications to the ROM (adding starting items so that the player can reach the modified level more easily, and automatically skipping the lengthy Ceres Station introduction and tutorial sequence for the same reason).

Figure 3 shows an image of a generated level design. Figure 6 shows an in-emulator screenshot from within the generated room. Figure 1 shows the game running on the original SNES hardware. In pursuit of using the original hardware to run generated content, our system for reinjection has significant limitations. Our goal in this paper is not to provide a complete system for generating Super Metroid content, but rather to show that it can be done, and use the problems that we encountered to focus future research.

Benefits of Reinjection

The design of Super Metroid exposes some important new facets of creating playable levels. While the WFC algorithm creates a new level in the same style as the original at the resolution of raw tiles, it does not consider the true gameplay semantics. For example, Figure 7 shows a case where the player can clip into the floor and become stuck while playing a generated level.

Rather than viewing this glitch as a shortcoming of Super Metroid, it actually exposes an important responsibility for generators of playable levels. Prior work on Super Mario Bros has largely assumed that the player occupies a single tile at a time (Cooper and Sarkar 2020). In Super Metroid, the player can change their hitbox shape in many ways (e.g. the single-tile morph ball shape or the multi-tile spin-jump shape), and this makes collision detection more complicated. Generators need to be able to recognize and avoid problematic platform placement. Specific examples of bad arrangements might be found by reinjecting generated content back into the original game and testing it there (i.e. by executing the modified ROM in an emulator). Some types of level design mistakes can only be detected by playing the levels. Reinjection makes it possible to find these problems.

Content reinjection also makes it easier for us to play our generated content while ensuring there are no deviations from the original videogame. Even though it took significant effort to learn the necessary technical knowledge, it would have been much more challenging to create a reasonably faithful Super Metroid approximation designed to showcase procedurally-generated content.

There are entire game features that are discovered by doing content reinjection that would otherwise be missed by working with simplified game clones. For example, in Super Metroid, each 16×16 tile section of each room has a *scroll value*, which controls how the camera behaves in that segment. Scroll values are used to create the experience of discovering a hidden passageway by keeping the passage offscreen until the player enters an invisible trigger in the mouth of the passage that temporarily changes the scroll values.

A generator that creates playable Super Metroid levels should be aware of existing scroll values, or consider how to control the camera at the same time that it designs the level geometry. Extra annotations like camera controls and loading triggers are commonly used in contemporary games. These examples show that there is a need for generators to consider level data outside of the tile grid. By learning about existing game data formats, PCG designers can discover other nonstandard types of content that could be generated in addition to level data.

Finally, the lack of available level data is a significant bottleneck in certain types of PCG approaches. Learning existing game data formats also provides a way to automatically extract level data, which can improve the diversity of level design data sets, such as the Video Game Level Corpus.

Conclusion

In this paper we demonstrate a pipeline for generating and reinjecting content for Super Metroid. This technique can be used to replace existing rooms with generated content, which can be played on the original hardware (or in an emulator). We also describe some of the benefits of content reinjection for Super Metroid, exposing new facets of level design generation. External validity is a concept that refers to the generalization of research findings to new settings(L Mitchell and M Jolley 2010, p. 48). The external validity of PCG research relies on developing generation techniques that can be successfully applied to existing games. To ground our research in the cultural and technical background of the practice of games, we need generators that can demonstrate the ability to handle real-world technical challenges by integrating into commercial games that were not originally designed to be PCG demo platforms.

Acknowledgements

The authors would like to thank Tamara Duplantis for creating Figure 1.

References

Altice, N. 2017. *I Am Error: The Nintendo Family Computer / Entertainment System Platform*. Platform Studies. MIT Press. ISBN 9780262534543. URL https://books.google.com/books?id=1r34DwAAQBAJ.

Chang, K.; Aytemiz, B.; and Smith, A. M. 2019. Reveal-More: Amplifying Human Effort in Quality Assurance Testing Using Automated Exploration. In *2019 IEEE Conference on Games (CoG)*. doi:10.1109/CIG.2019.8848091.

Chang, K.; and Smith, A. 2020. Differentia: Visualizing Incremental Game Design Changes. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 16(1): 175–181. URL https://ojs.aaai.org/index.php/AIIDE/article/view/7427.

Cooper, S.; and Sarkar, A. 2020. Pathfinding Agents for Platformer Level Repair. *Proceedings of the Experimental AI in Games (EXAG) Workshop at AIIDE* .

Freehold Games. 2021. Caves of Qud. Grinblat, Jason and Bucklew, Charles Brian.

Karth, I.; and Smith, A. M. 2017. WaveFunctionCollapse is Constraint Solving in the Wild. In *Proceedings of the 12th International Conference on the Foundations of Digital Games*.

Karth, I.; and Smith, A. M. 2021. WaveFunctionCollapse: Content Generation via Constraint Solving and Machine Learning. *IEEE Transactions on Games* doi:10.1109/TG.2021.3076368.

L Mitchell, M.; and M Jolley, J. 2010. *Research design explained*.

Lewis, C. 2009. Infinite Tux. URL https://github.com/qbancoffee/infinite-tux.

Liapis, A. 2020. 10 Years of the PCG workshop: Past and Future Trends. In *Proceedings of the 2020 FDG workshop on Procedural Content Generation*.

Lucas, S. M.; and Volz, V. 2019. Tile Pattern KL-Divergence for Analysing and Evolving Game Levels. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '19, 170–178. New York, NY, USA: Association for Computing Machinery. ISBN 9781450361118. doi:10.1145/3321707.3321781. URL https://doi.org/10.1145/3321707.3321781.

Nintendo Co. Ltd. 1983. Super Mario Bros.

Nintendo Co. Ltd. 1994. Super Metroid.

Persson, M. 2009. Infinite Mario AI Competition. URL https://notch.tumblr.com/post/157149894/infinite-mario-ai-competition.

Plausible Concept. 2018. Bad North. Oskar Stålberg and Richard Meredith and Martin Kvale.

Shaker, N.; Togelius, J.; and Nelson, M. 2016. *Procedural Content Generation in Games*. Computational Synthesis and Creative Systems. Springer International Publishing. ISBN 9783319427164. URL https://books.google.com/books?id=-IdJDQAAQBAJ.

Shaker, N.; Togelius, J.; Yannakakis, G. N.; Weber, B.; Shimizu, T.; Hashiyama, T.; Sorenson, N.; Pasquier, P.; Mawhorter, P.; Takahashi, G.; Smith, G.; and Baumgarten, R. 2011. The 2010 Mario AI Championship: Level Generation Track. *IEEE Transactions on Computational Intelligence and AI in Games* 3(4): 332–347. doi:10.1109/TCIAIG.2011.2166267.

Smith, A. M.; Butler, E.; and Popovic, Z. 2013. Quantifying over play: Constraining undesirable solutions in puzzle design. In *FDG 2013*, 221–228.

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 Ontanón, S. 2017. Procedural level generation using multi-layer level representations with mdmcs. In *2017 IEEE conference on computational intelligence and games (CIG)*, 280–287. IEEE.

Summerville, A. 2018. Expanding expressive range: Evaluation methodologies for procedural content generation. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 14.

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.; and Mateas, M. 2016. Super Mario as a String: Platformer Level Generation Via LSTMs. In *DiGRA/FDG &#3916 - Proceedings of the First International Joint Conference of DiGRA and FDG*. Dundee, Scotland: Digital Games Research Association and Society for the Advancement of the Science of Digital Games. ISBN ISSN 2342-9666. URL http://www.digra.org/wp-content/uploads/digital-library/paper_129.pdf.

Summerville, A. J.; Mateas, M.; Snodgrass, S.; and Ontañón, S. 2016. The VGLC: The Video Game Level Corpus. In *Proceedings of the FDG workshop on Procedural Content Generation*.

Togelius, J.; Shaker, N.; Karakovskiy, S.; and Yannakakis, G. N. 2013. The Mario AI Championship 2009-2012. *AI Magazine* 34(3): 89–92. doi:10.1609/aimag.v34i3.2492. URL https://ojs.aaai.org/index.php/aimagazine/article/view/2492.

Treanor, M.; Warren, N.; Reed, M.; Smith, A.; Ortiz, P.; Coney, L.; Sherman, L.; Carré, E.; Vivatvisha, N.; Harrell, D.; et al. 2017. Playable experiences at AIIDE 2017. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 13.

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*, GECCO '18, 221–228. New York, NY, USA: Association for Computing Machinery. ISBN 9781450356183. doi:10.1145/3205455.3205517. URL https://doi.org/10.1145/3205455.3205517.