Generating Playable RPG ROMs for the Game Boy

ISAAC KARTH, University of California, Santa Cruz, USA
TAMARA DUPLANTIS, University of California, Santa Cruz, USA
MAX KREMINSKI, University of California, Santa Cruz, USA
SACHITA KASHYAP, University of California, Santa Cruz, USA
VIJAYA KUKUTLA, University of California, Santa Cruz, USA
AARON LO, University of California, Santa Cruz, USA
ANIKA MITTAL, University of California, Santa Cruz, USA
HARVIN PARK, University of California, Santa Cruz, USA
ADAM M. SMITH, University of California, Santa Cruz, USA

The handheld Game Boy console has seen widespread popularity, with over a hundred million sold. It continues to be relevant as a retro-computing platform, and through emulation a Game Boy ROM can be run in the browser. However, despite being a well-defined, stable platform, there have been very few procedural generation projects that target the Game Boy. By leveraging existing ecosystems of development tools for the Game Boy platform, we construct a game generator that outputs playable ROMs. Paired with this, we seek to improve the ecological validity of game generation by building a generator that acts as a bridge to an active community of non-academic developers. The system interprets and outputs the same data artifacts used in the wild, allowing it both learn from and contribute to development communities, as well as tapping into the corpus of existing design work.

CCS Concepts: • **Software and its engineering → Interactive games**; • **Applied computing → Computer games**.

Additional Key Words and Phrases: game generation, ecological validity, retrocomputing

1 INTRODUCTION

Academic game generation can often be an ivory tower exercise. While many projects, such as ANGELINA [8], attempt to engage with game development communities—for example by participating in game jams [9]—academic game generators seldom produce artifacts that game developers can directly use. While previous game generation systems have produced interesting results, they are typically not immediately actionable for human developers.

This is a problem because we want our results to be useful. To borrow a term from psychological research, we want *ecological validity* [3], a demonstration that the things that we generate can be useful to developers in non-academic contexts. The first step of that is generating things in such a way that the generated artifacts are available to non-academic developers.

Ideally, we would also like this relationship to be bi-directional: just as developers should be able to use the things that our generator produces, our generator should be able to tap into an existing corpus and learn from the community. Such a system would need a game representation that is shared by both the community and the generative system.

To address this, we implemented a game generation system targeted at generating playable ROMs for the Nintendo Game Boy platform (Fig. 1)[1]. Our system autonomously generates, compiles, and launches complete, playable top-down RPG/adventure games. The ROMs it creates are playable on the original hardware, in console emulators, and in the browser (both desktop and mobile). The system generates the level design, NPC dialog, title art, and so forth, all of which

[1]The source code is available at https://github.com/ikarth/game-boy-rom-generator, a video of the generator running is available at https://youtu.be/G5KKxxNevVQ, and generated games can be played at https://isaackarth.com/games/rom_gen_test_5/

Fig. 1. The generated game *Princess's Mystic Ash Wizard: Ocean of the Dread*, running on the Super Nintendo, via the Super Game Boy cartridge. The Super Game Boy is an official Nintendo peripheral that allows Game Boy cartridges to be played on a Super Nintendo Entertainment System, and is commonly used for streaming Game Boy video, due to the original Game Boy lacking a video out.

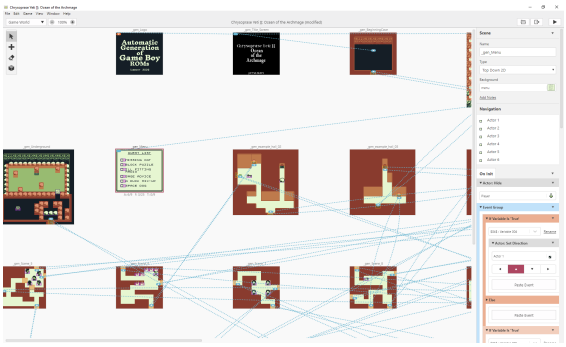

Fig. 2. The GB Studio 1.2.1 interface, displaying the project file for the generated game *Crysopraise Yeti][: Ocean of the Archmage*. The game is fully editable by the user, and can be compiled into a playable ROM.

can be reopened in the game engine IDE and reused by game developers. It also includes basic automated playtesting via the Go Explore algorithm [12] incorporating itself into the wider AI gym research, where test environments are created to train new AI models.

We use GB Studio project files as an intermediate game representation format, which we term GBS. GBS is a JSON data structure that acts as a genre-and-platform specific game definition language (GDL) which we used to inform the ontology of our generator. By borrowing the concepts the GB Studio engine uses, our data structures can build on top of genre expectations and platform history to create ROMs that are culturally recognizable as games.

Additionally, by using an existing engine, our system can act as a bridge between the research community and an active community of active game developers. This allows us to learn from the data artifacts that they create and share. Since our generator produces an intermediate format that the engine can open, the relationship is bi-directional, opening the potential for developers to reuse elements from the games that our generator produces.

1.1 The Platform

We chose the Nintendo Game Boy as a platform because:

- The Game Boy is a fixed target. The platform has not had a backwards-compatibility breaking change in decades. The games we generate will be playable for the indefinite future.
- The Game Boy has a mature emulation ecosystem. The games that we generate can run on desktops, in browsers, and on mobile devices.
- Retrocomputing minimizes the ecological footprint of the generator: once generated, the games do not need expensive GPU processing (or new hardware at all).
- Despite this, new compatible hardware *is* being produced, in the form of the Analog Pocket[2].
- Due to the extensive existing emulation, AI training gyms (such as OpenAI's Gym Retro [24]) already support the format and can play our games.

[2] https://www.analogue.co/pocket

- We were able to leverage local expertise with the hardware, incorporating deep knowledge of how the platform operates.
- At the same time, the existence of an integrated development environment for the platform (in the form of GB Studio) meant that we could involve inexperienced or less technical contributors in the project.
- Leveraging GB Studio meant that we could borrow its ontology of scenes, triggers, and actors, giving us a starting point for elaborating our own ontology.
- There is an active community of developers who will be able to open and understand the generated games. There are also many existing games to use as data, both in GBS format and as ROMs.
- Part of our goal was to generate games that are culturally recognizable as games, with the objective of having them fit in a recognizable cultural lineage. By establishing a framing [10] that positions them as videogames, we make it easier to innovate in ways that push the narrative side of game design.
- Our long-term research is interested in addressing the orchestration problem [19], which requires generating games with multiple facets to orchestrate.
- Nostalgia was not a factor: indeed, many members of the team working on the project were born after the release date of the original Game Boy, and the majority of the team has never owned the original hardware.

1.2 GB Studio

One motivation driving our technical decision-making was the desire to incorporate the tools that communities are already using in the wild. A successful procedural generation research project should be in conversation with the communities that already make and use the kinds things they are generating. Therefore, we chose to target the GB Studio format to build on the existing community and tools.

GB Studio (Fig. 2) is an open-source "visual game builder" [22] that acts as an interface between the user and the Game Boy Developer's Kit (GBDK), an unofficial set of open source C tools and libraries for developing software that runs on the Game Boy platform. Our generator targeted GB Studio as an intermediate format, leveraging its existing build pipeline for our generator.

The core of our generator is a Python program that outputs the JSON specification that GB Studio compiles (via GBDK) into a ROM that can run on a Game Boy or emulator.

2 RELATED WORK

Game generation has been around for some time. One strand arose out of the *newsgame* [33] movement, inspired by the contemporary game studies focus on procedural rhetoric [2]. Newsgames influenced the Nelson and Mateas generation project [23] which generates WarioWare-esque microgames by combining a few sets of stock mechanics with entity movement behaviors constrained by commonsense knowledge of object types from the ConceptNet [21] and WordNet [29] databases.

Another early approach by Togelius and Schmidhuber [32] evolves arcade games within a search space defined by an internal representation of game rules. Smith and Mateas's *Variations Forever* system [30] similarly makes use of an internal-only representation of game rule structure for minigame generation—this time leveraging answer set programming instead of evolutionary algorithms to search the space for viable rulesets.

Cook's ANGELINA series of game generation systems continued the evolutionary strategy [8]. More recent versions of ANGELINA [9] have experimented with recontextualizing game generation as a continuous [7] and culturally

engaged process, with ANGELINA drawing on querying its Twitter followers, using news articles for themes, and entering its games into game jams.

WikiMystery [1] is another example of culturally-engaged videogame generation, using real-world data (specifically Wikipedia data about networks of well-known historical figures) to generate murder mystery adventure games that evoke a specific cultural context. Machine learning approaches to the acquisition of game design knowledge for game generation have recently come into use [25]. Guzdial and Riedl [15] combine these approaches with conceptual expansion to invent novel gameplay through the recombination of mechanics learned from existing games.

The idea of using an intermediate format to allow the system to reason about the games being generated is not new: PuzzleScript [18] is a non-academic language for describing 2D tile-based block-sliding games. In addition to being used for human-created puzzle games, it has been used for both level generation [16] and full game generation including the invention of novel mechanics [20].

Other game generators also make use of intermediate formats—most notably through game description languages (GDLs). These include the Stanford GDL [14], which focuses primarily on turn-taking boardgames. The Ludi GDL is a more focused boardgame description language leveraged by the Ludi generator to explore abstract games, leading to multiple commercially published games [4]. VGDL [27] is targeted at 2d arcade videogames [11] and is used by GVGAI, a game AI research toolkit, as a shared representation for both automated playing and game generation [26]. The Cygnus game description language [31], developed in response to the limitations of VGDL, is capable of expressing a strict superset of the games that can be expressed in VGDL. Cygnus powers the abstract game generator Gemini [31], which has been used to generate small arcade games included in the narrative game *Emma's Journey* [13] as thematically appropriate procedural accompaniment to a text-driven narrative. It also powers the Gemini-based mixed-initiative game creation tool Germinate [17].

Cook has recently argued [6] that game generation tools intended to fit into human game development workflows could benefit from targeting general-purpose programming languages directly, rather than targeting a narrower GDL. Such tools would need to both read human-authored code (perhaps structured in a way that makes this code more amenable to automatic analysis) and to emit human-readable code.

3 DESIGN AND IMPLEMENTATION

Our generator takes GB Studio project files as input, turns them into templates for inclusion in the level generation, generates new games, exports the files to GB Studio, compiles it to a ROM, and generates a web page that acts as a catalog of generated games (Fig. 4).

Because we borrow the GBS ontology, the basic atomic units the generator uses are scenes, triggers, and actors. Scenes are discrete map spaces, triggers are areas in space that run scripts when the player enters them, and actors are Non-Player Characters (NPCs) (who can also run scripts). Our ontology augments this: for example, we added the concept of *connections*. Connections are pairs of triggers in neighboring scenes that bidirectionally move the player between scenes: a common implicit gameplay feature that our ontology makes explicit, enabling the generator to directly reason about connections.

Scenes can be hand-authored. Or example scenes in GB Studio format can be imported through a process that turns them into scene template functions (which create a new scene based on the original example, but with added variation). The GBS scripting is converted into Python; the import and export conversion is synced with changes in the underlying scripting language by running a module that generates Python code that matches the GBS commands. Since the templates import GBS scripting, non-technical users can visually construct complex scenes in the IDE and import

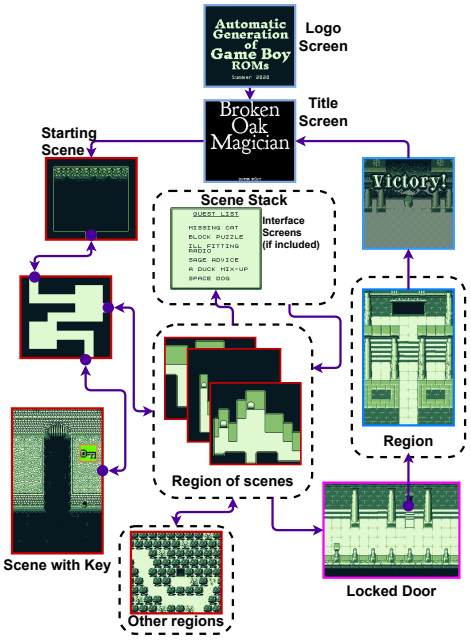

Fig. 3. Diagram of the basic layout of a generated game. Execution begins with the intro and title screens (top left), from which the player can start a new game or load a saved game. New games start in the scene the generator has designated as the starting scene. The scenes are connected via bidirectional trigger links. Connections are tagged with their region, and locked gates are placed between some regions, which can be unlocked by the player finding the matching key. Additional interface scenes (menus, combat, etc.) can be accessed using GB Studio's scene stack, though the current unified generator doesn't use that feature.

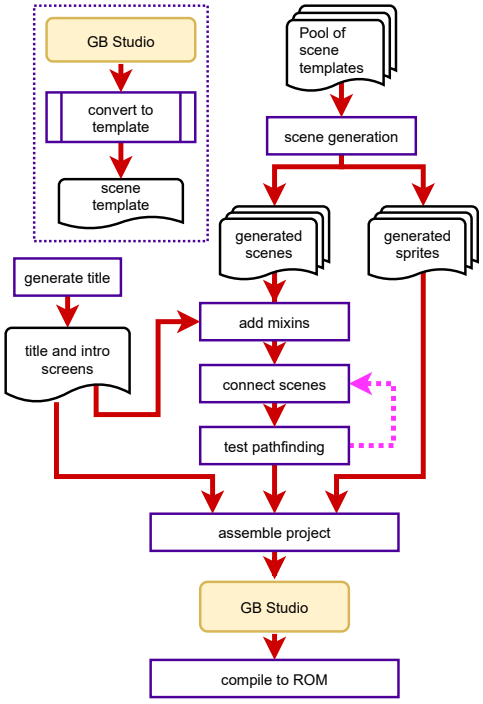

Fig. 4. The generative pipeline in version 1.0 of the generator. The generator starts with a pool of scene templates. The title of the game (generated via Tracery [5]) is used to generate the title screen (via seam-carving [28]), and a MacGuffin is selected based on the title. The MacGuffin and other mixins (such as NPCs) are incorporated into the generated scenes. The scenes are then connected via creating bi-directional links between scenes, with a generate-and-test loop to ensure that the player can pathfind from the start to the end of the game. The generated data is assembled into a project data structure, read into GB Studio, and compiled into a ROM.

them as templates. Scene template functions can be further customized: our victory screen started as an auto-generated template that was customized with additional functionality. Our generator is designed to support multiple ways of running the generation process, to the point where importing an example game from GB Studio also generates a function that will generate games using only the newly imported scenes. For purposes of this demo, we constructed a unified generator that draws on over 70 scene templates and assembles them into a completed game.

The unified generator generates a title (via Tracery [5]) and then selects a MacGuffin[3] for the player to quest for by analyzing the title and generating a quest object that reflects the title. The title is rendered as a box cover (for the web page) and a title screen (for display in-game), using seam carving [28] to adjust the layout of the text (Fig. 5).

The library of scene templates are divided into regions. A subset of templates are sampled without replacement from each region to arrive at the final scene list. Scenes that contain linked behavior, such as lock-and-key puzzles, are

[3]In the sense used by Hitchcock: A MacGuffin is a thing that motivates the characters but whose contents don't matter to the narrator [34].

Fig. 5. A selection of title screens generated by the project. Titles are generated via a Tracery grammar, which is then rendered into the title font. The title image layout is resized with seam carving, which both rearranges the elements for better layout in the design and sets the image ratio to match the Game Boy screen resolution.

included as joint groups. Scenes are then connected together, using the previously-detected connection points, to form the final map. The connection process places the locked gates between regions. A generate-and-test process using a pathfinding test is run to ensure that the player will be able to traverse the game from beginning to end. Additional actors (NPCs, signs, and other objects that can be interacted with) are added to some scenes, with narratively-relevant dialog generated via a Tracery [5] grammar that was itself generated via Gitta [35].

Once a complete project is generated and verified, it is written to disk (as a GBS JSON file) together with its associated assets. GB Studio is launched and automatically compiles the game into a finished ROM. A webpage cataloging all of the games generated in the current batch is also generated, with embedded links to download the user-editable project files. The generated ROM can be played in an emulator, on the web, or on the original Game Boy hardware (Fig. 1).

From here, the game can optionally be automatically machine playtested using the OpenAI Gym Retro API. We apply the Go-Explore algorithm [12] (one of many available automated game exploration methods [36]) to find samples of diverse pathways through the game's state space.

4 CONCLUSION

We built a generator that demonstrates ecological validity by being able to both take input from and generate output for a non-academic game development IDE that is actively used by a community of developers. Non-technical users can create content for the generator and use content generated by the generator without needing an understanding of the technical details of the generator.

By building a generator targeting the Game Boy, we are able to participate in a mature ecosystem of development tools and emulation, ensuring that the games generated by the system will be playable for the foreseeable future. We leveraged the existing GB Studio ontology and constructed our own ontology of RPG implementation on top of it (mixin NPCs, connections, lock-and-key quests, etc.). The generator creates complete, playable games with unique titles and varied layouts.

Looking to the future, a major limitation of the current version of the generator is that the pipeline has many interconnected dependencies, as is characteristic of the *orchestration problem* [19]. The next phase of development for the generator will address this directly. This will make it much easier to create content for the generator and improve its expressive range.

However, even in its present state, the generator fulfills its initial goal of demonstrating the possibility of bi-directional content exchange between the generator and and the tools used by an existing community of developers.

REFERENCES

[1] Gabriella Alves Bulhoes Barros, Michael Green, Antonios Liapis, and Julian Togelius. 2019. Who killed Albert Einstein? From open data to murder mystery games. *IEEE Transactions on Games* (2019).

[2] Ian Bogost. 2007. *Persuasive Games: The Expressive Power of Videogames*. MIT Press.

[3] M. B Brewer. 2000. Research design and issues of validity. In *Handbook of research methods in social and personality psychology*, H. T. Reis & C. M. Judd (Ed.). Cambridge University Press, 3–16.

[4] Cameron Browne and Frederic Maire. 2010. Evolutionary game design. *IEEE Transactions on Computational Intelligence and AI in Games* 2, 1 (2010).

[5] Kate Compton, Quinn Kybartas, and Michael Mateas. 2015. Tracery: an author-focused generative text tool. In *International Conference on Interactive Digital Storytelling*. Springer, 154–161.

[6] Michael Cook. 2020. Software Engineering For Automated Game Design. In *IEEE Conference on Games*. IEEE.

[7] Michael Cook and Simon Colton. 2018. Redesigning computationally creative systems for continuous creation. (2018).

[8] Michael Cook, Simon Colton, and Jeremy Gow. 2016. The ANGELINA videogame design system—part I. *IEEE Transactions on Computational Intelligence and AI in Games* 9, 2 (2016), 192–203.

[9] Michael Cook, Simon Colton, and Jeremy Gow. 2016. The ANGELINA videogame design system—part II. *IEEE Transactions on Computational Intelligence and AI in Games* 9, 3 (2016), 254–266.

[10] Michael Cook, Simon Colton, Alison Pease, and Maria Teresa Llano. 2019. Framing In Computational Creativity-A Survey And Taxonomy.. In *ICCC*. 156–163.

[11] Marc Ebner, John Levine, Simon M Lucas, Tom Schaul, Tommy Thompson, and Julian Togelius. 2013. Towards a video game description language. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[12] Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. 2019. Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995* (2019).

[13] Jacob Garbe, Max Kreminski, Ben Samuel, Noah Wardrip-Fruin, and Michael Mateas. 2019. StoryAssembler: an engine for generating dynamic choice-driven narratives. In *Proceedings of the 14th International Conference on the Foundations of Digital Games*.

[14] Michael Genesereth, Nathaniel Love, and Barney Pell. 2005. General game playing: Overview of the AAAI competition. *AI Magazine* 26, 2 (2005), 62–72.

[15] Matthew Guzdial and Mark Riedl. 2018. Automated game design via conceptual expansion. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 14.

[16] Ahmed Khalifa and Magda Fayek. 2015. Automatic puzzle level generation: A general approach using a description language. In *Computational Creativity and Games Workshop*.

[17] Max Kreminski, Melanie Dickinson, Joseph Osborn, Adam Summerville, Michael Mateas, and Noah Wardrip-Fruin. 2020. Germinate: A Mixed-Initiative Casual Creator for Rhetorical Games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 16. 102–108.

[18] Stephen Lavelle. 2013. PuzzleScript. http://puzzlescript.net.

[19] Antonios Liapis, Georgios N Yannakakis, Mark J Nelson, Mike Preuss, and Rafael Bidarra. 2018. Orchestrating game generation. *IEEE Transactions on Games* 11, 1 (2018), 48–68.

[20] Chong-U Lim and D Fox Harrell. 2014. An approach to general videogame evaluation and automatic generation using a description language. In *2014 IEEE Conference on Computational Intelligence and Games*. IEEE.

[21] Hugo Liu and Push Singh. 2004. ConceptNet—a practical commonsense reasoning tool-kit. *BT Technology Journal* 22, 4 (2004), 211–226.

[22] Chris Maltby. 2021. *itch.io: GB Studio by Chris Maltby*. https://chrismaltby.itch.io/gb-studio

[23] Mark J Nelson and Michael Mateas. 2007. Towards automated game design. In *AI*IA 2007: Artificial Intelligence and Human-Oriented Computing*. Springer, 626–637.

[24] Alex Nichol, Vicki Pfau, Christopher Hesse, Oleg Klimov, and John Schulman. 2018. Gotta Learn Fast: A New Benchmark for Generalization in RL. *arXiv preprint arXiv:1804.03720* (2018).

[25] Joseph C Osborn, Adam Summerville, and Michael Mateas. 2017. Automated game design learning. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 240–247.

[26] Diego Perez-Liebana, Jialin Liu, Ahmed Khalifa, Raluca D Gaina, Julian Togelius, and Simon M Lucas. 2019. General video game AI: A multitrack framework for evaluating agents, games, and content generation algorithms. *IEEE Transactions on Games* 11, 3 (2019), 195–214.

[27] Tom Schaul. 2013. A video game description language for model-based or interactive learning. In *2013 IEEE Conference on Computational Inteligence in Games (CIG)*. IEEE.

[28] Vidya Setlur, Saeko Takagi, Ramesh Raskar, Michael Gleicher, and Bruce Gooch. 2005. Automatic Image Retargeting. In *Proceedings of the 4th International Conference on Mobile and Ubiquitous Multimedia* (Christchurch, New Zealand) *(MUM '05)*. Association for Computing Machinery, New

York, NY, USA, 59–68. https://doi.org/10.1145/1149488.1149499

[29] Push Singh. 2002. The public acquisition of commonsense knowledge. In *Proceedings of AAAI Spring Symposium: Acquiring (and Using) Linguistic (and World) Knowledge for Information Access*.

[30] Adam M Smith and Michael Mateas. 2010. Variations Forever: Flexibly generating rulesets from a sculptable design space of mini-games. In *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*. IEEE, 273–280.

[31] Adam Summerville, Chris Martens, Ben Samuel, Joseph Osborn, Noah Wardrip-Fruin, and Michael Mateas. 2018. Gemini: bidirectional generation and analysis of games via ASP. In *Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference*.

[32] Julian Togelius and Jurgen Schmidhuber. 2008. An experiment in automatic game design. In *2008 IEEE Symposium On Computational Intelligence and Games*. IEEE, 111–118.

[33] Mike Treanor and Michael Mateas. 2009. Newsgames: Procedural Rhetoric Meets Political Cartoons. In *DiGRA Conference*.

[34] François Truffaut, Helen G. Scott, and Alfred Hitchcock. 1984. *Hitchcock*. Simon and Schuster. http://hdl.handle.net/2027/mdp.39015008348644 Translation of *Le cinéma selon Hitchcock*.

[35] Thomas Winters and Luc De Raedt. 2020. Discovering Textual Structures: Generative Grammar Induction using Template Trees. arXiv:2009.04530 [cs.CL]

[36] Zeping Zhan, Batu Aytemiz, and Adam M Smith. 2019. Taking the Scenic Route: Automatic Exploration for Videogames. In *Proceedings of the Second AAAI Workshop on Knowledge Extraction from Games*.