Debugging Procedural Level Designs with Mental Maps

Riemer van Rozen
rozen@cwi.nl
Centrum Wiskunde & Informatica
The Netherlands

Georgia Samaritaki
samaritakigeorgia@gmail.com
University of Amsterdam
The Netherlands

Joris Dormans
joris@ludomotion.com
Ludomotion
The Netherlands

ABSTRACT

Procedural Level Generation provides tools and techniques for generating many game levels from a single specification. Instead of creating levels by hand, level designers make use of generators that automate the creation process. However, iteratively improving a level's design requires encoding generators of adventures, puzzles and encounters in notations that bear little resemblance to generated content. Raising the level quality is difficult, because it is hard to reason about bugs that can manifest inside generated content.

We take the position that *debugging* requires special attention. We argue that advancing the area of PCG calls for tools and debugging techniques that speed up procedural level design and empower level designers. We propose exploring how Domain-Specific Languages can help in authoring a level's design, validating the generator's code, and debugging issues in generated content. We introduce Mental Maps, a visual language that expresses the spacial relations between rooms, objects and paths. We discuss how Mental Maps can serve as generator blueprints before the generation happens, and as debugging lenses for projecting issues afterwards.

KEYWORDS

procedural content generation, level design, debugging

1 INTRODUCTION

Procedural Level Generation studies how to generate many game levels from a single specification, e.g., for games such as platformers or dungeon crawlers [11]. Over the years, authors have described many versatile techniques for creating generators [5, 11], exploring design spaces [8], analyzing expressive ranges [2, 10], and creating mixed-initiative design tools [1, 5, 6]. Although *debugging* is widely understood as an important facet of Procedural Content Generation (PCG) [9], few approaches propose *debugging techniques* that directly integrate with the source code of generation engines [7, 13].

In this position paper, we argue that advancing theory and practice of PCG requires studying how *debugging techniques* can help to automate and speed-up procedural level design. In particular, we explore how Domain-Specific Languages (DSLs) can help in authoring a level's design, validating the generator's code, verifying generated content, tracking issues, and debugging these artifacs simultaneously. To support our argument, we discuss 1) why a need for debugging arises in a typical automation process and 2) how *'mental maps'*, visual DSLs for bridging the gap between level designs and generated content, can help improve debugging facilities.

2 PROCEDURAL LEVEL DESIGN

Game levels of Roguelikes consist of content such as forests, roads and clearings that situate activities, quests and puzzles involving enemies, objects and skills. Designers make sketches, draw diagrams and write texts that compose this content for wondrous adventures, thrilling encounters, perilous challenges and glorious victories. Instead of creating the levels by hand, game developers can opt for constructing *generators* that help to automate the creation process. The challenge is to devise a *content generation strategy* for producing interesting variations and combinations that realize the design's quality, overall structure and gameplay goals. Figure 1 illustrates activities (rounded rectangles) in the automation process that consume and produce (arrows) different artifacts (rectangles).

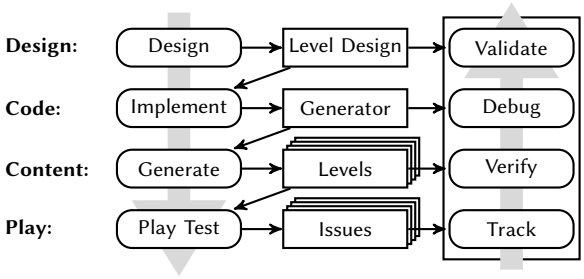

Figure 1: Designing and debugging generated game levels

The implementation of a generator involves translating its level design into source code. Engines represent and transform content, using formalisms and techniques such as design patterns [1], Answer Set Programming or logical constraints [8], transformative grammars (L-Systems) [5] and SMT solvers [14]. To accommodate evolving level designs and gameplay goals, designers constantly require changes to the code controlling the algorithms and transformation pipelines that produce the levels. However, improving the quality is difficult, because it is hard to reason about bugs that can manifest inside potentially generated content. As a result, many changes are complex, time-consuming and error-prone. To get to grips with this complexity, designers need better tools and debugging techniques for analyzing the impact of changes [2, 10, 13].

Play testing generated levels is particularly complex. In practice, this means playing generated levels and *eyeballing* the content, which is costly, repetitive and takes too much time. For better test coverage, level designers need test automation techniques that can identify issues automatically. For instance for: 1) *verifying* the *structures* of levels against the design and 2) *testing* dynamic interaction sequences, for ensuring *valid* game states and progressions.

Additionally, when players report bugs about a particular level, these issues need to be recorded and tracked for making fixes. The challenge is relating the issues to specific parts of the generated content, the design and the generator's sources. Level designers need tools and techniques for reproducing the issues and debugging the related level artifacts simultaneously in an integrated manner. Next, we discuss addressing these challenges with *'mental maps'*.

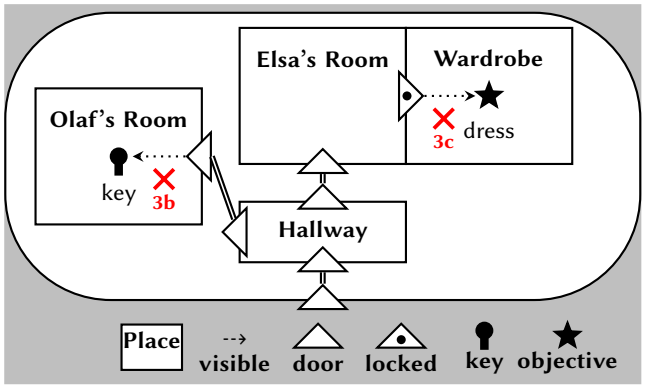

Figure 2: Mental Map of Elsa's Magic Ice Castle

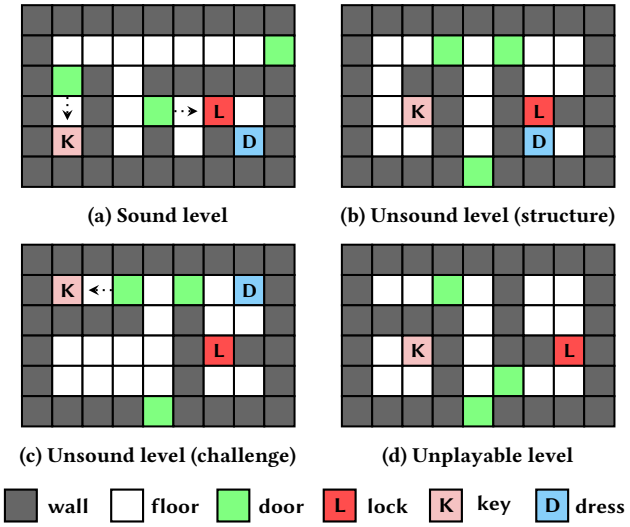

Figure 3: Tile Maps of Example levels

3 MENTAL MAPS

In a recent blog post, Dormans describes how Ludomotion's content generation strategy has been instrumental for the innovative procedural level designs of Unexplored 2 [4]. Ludomotion uses visual level designs that resemble generated game levels. The diagrams depict abstract spacial relations between places, objects and paths, and serve as *blueprints* for the game's *grammar-based* level generator Ludoscope [3, 5]. Ludoscope offers a visual editor for expressing transformation pipelines but lacks a debugger. Once encoded, the level blueprints cannot easily be inspected or used to track content.

Based on this notation, we propose Mental Maps, a visual Domain-Specific Language for procedural level design for controlling the variability of level generators. Figure 2 shows a design of a castle level with a hallway and two connected rooms. The player has to get a key from Olaf's room to obtain a dress from Elsa's wardrobe.

First, we explain how to manually derive *level properties* from Mental Maps. We observe the following: 1) in Olaf's room is a key; 2) the key can be obtained; 3) the key is visible from the door; 4) in Elsa's room is a wardrobe. 5) the wardrobe is locked; 6) that lock is visible from the door; 7) in the wardrobe is a dress; and 8) the dress can be obtained; 9) the dress cannot be obtained without the key (we omit the rest). Next, we compare these constraints with four simplified levels. Figure 3 shows tile maps that illustrate good and bad qualities. Only the first one (Figure 3a) conforms to every design constraint. Figure 3b violates constraint 3, Figure 3c violates constraints 6, 7 and 9 and Figure 3d violates constraints 2 and 7.

4 DISCUSSION

Here we discuss how Mental Maps, a conceptual approach exemplified by the DSL of Section 3, could help to debug generated content. We propose extending tools such as Ludoscope with a means for authoring Mental Maps as blueprints that control the variability of the generator. Instead of manually translating hand-written diagrams into source code, the tool derives content constraints automatically.

This approach builds on existing theory. In particular, textual notations for expressing level properties have been used for identifying [13] and preventing [7] low quality combinations of rooms, quests and objects. Using *origin tracking* [12], the names and *source locations* (visual placements, lines and column numbers) of the content can be passed to a generator's implementation. In debug mode, the generator can copy these locations into the generated content at no additional cost. The debugger can trace the origins of reported issues inside generated levels through the generator's sources back to the design, as shown in Figure 1. Once identified, issues can be marked on the Mental Map (as a debug lens), as shown in Figure 2.

Of course, we are still in the process of constructing the tools that implement our conceptual approach for test automation and source level debugging. However, it could provide the following benefits: 1) expressing level designs visually; 2) mapping cognitive design processes to the content generation; 3) raising the level quality.

Mental Maps are generator-agnostic, and not limited to grammar-based level generation. For instance, generators based on ASP or SMT solvers use constraints as first-class citizens to prevent low quality levels. Deriving constraints directly from Mental Maps level designs could help ease the authorial burden. Finally, issue tracking using Mental Maps does not depend on a particular kind of content representation or generation technique.

5 CONCLUSION

We have argued for advancing procedural level generation with *debugging techniques* and tools that speed up the level design process and empower designers. In addition, we have described Mental Maps as a promising research direction to help improve debugging.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their constructive comments that have helped to improve this paper.

REFERENCES

[1] Alexander Baldwin, Steve Dahlskog, José M. Font, and Johan Holmberg. 2017. Mixed-Initiative Procedural Generation of Dungeons using Game Design Patterns. In *IEEE Conference on Computational Intelligence and Games, CIG 2017, New York, NY, USA, August 22-25, 2017*. IEEE, 25–32. https://doi.org/10.1109/CIG.2017.8080411

[2] Michael Cook, Jeremy Gow, Gillian Smith, and Simon Colton. 2021. Danesh: Interactive Tools for Understanding Procedural Content Generators. *IEEE Transactions on Games* (2021), 1–1. https://doi.org/10.1109/TG.2021.3078323

[3] Joris Dormans. 2012. *Engineering Emergence: Applied Theory for Game Design*. PhD Thesis. University of Amsterdam.

[4] Joris Dormans. 2021. The Theory of The Place: A Level Design Philosophy for Unexplored 2. *Gamasutra* (Oct. 2021). https://www.gamedeveloper.com/blogs/the-theory-of-the-place-a-level-design-philosophy-for-unexplored-2

[5] Daniel Karavolos, Anders Bouwer, and Rafael Bidarra. 2015. Mixed-Initiative Design of Game Levels: Integrating Mission and Space into Level Generation. In *Proceedings of the 10th International Conference on the Foundations of Digital Games, FDG 2015, Pacific Grove, CA, USA, June 22-25, 2015*. Society for the Advancement of the Science of Digital Games.

[6] Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. 2013. Sentient Sketchbook: Computer-Aided Game Level Authoring. In *Proceedings of the 8th International Conference on the Foundations of Digital Games, FDG 2013, Chania, Crete, Greece, May 14-17, 2013*. Society for the Advancement of the Science of Digital Games, 213–220.

[7] Georgia Samaritaki. 2022. *Debugging Grammars for Level Generation*. Master's Thesis. Master of Software Engineering, University of Amsterdam.

[8] Adam M. Smith and Michael Mateas. 2011. Answer Set Programming for Procedural Content Generation: A Design Space Approach. *IEEE Trans. Comput. Intell. AI Games* 3, 3 (2011), 187–200. https://doi.org/10.1109/TCIAIG.2011.2158545

[9] Gillian Smith. 2013. *Level Design Processes and Experiences*. AK Peters/CRC Press, Chapter Procedural Content Generation: An Overview, 159–183.

[10] Gillian Smith and Jim Whitehead. 2010. Analyzing the Expressive Range of a Level Generator. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games, PCGames '10, Monterey, California, USA, June 18, 2010*. ACM, 4:1–4:7. https://doi.org/10.1145/1814256.1814260

[11] Roland van der Linden, Ricardo Lopes, and Rafael Bidarra. 2014. Procedural Generation of Gungeons. *IEEE Trans. Comput. Intell. AI Games* 6, 1 (2014), 78–89. https://doi.org/10.1109/TCIAIG.2013.2290371

[12] Arie van Deursen, Paul Klint, and Frank Tip. 1993. Origin Tracking. *J. Symb. Comput.* 15, 5/6 (1993), 523–545. https://doi.org/10.1016/S0747-7171(06)80004-0

[13] Riemer van Rozen and Quinten Heijn. 2018. Measuring Quality of Grammars for Procedural Level Generation. In *Proceedings of the 13th International Conference on the Foundations of Digital Games, FDG 2018, Malmö, Sweden, August 07-10, 2018*. ACM, 56:1–56:8. https://doi.org/10.1145/3235765.3235821

[14] Jim Whitehead. 2020. Spatial Layout of Procedural Dungeons Using Linear Constraints and SMT Solvers. In *FDG '20: International Conference on the Foundations of Digital Games, Bugibba, Malta, September 15-18, 2020*. ACM, 101:1–101:9. https://doi.org/10.1145/3402942.3409603