A Comparison of Procedural Content Generators

COMP110 - Computer Architecture Essay

AR185160

December 2, 2015

This paper will evaluate and compare three articles; A Rhythm-Based Level Generator for 2-D platformers[1], Procedural Content Generation Using Occupancy-Regulated Extension[2] and Procedural Content Generation Using Patterns as Objectives[3]. These articles all have their own techniques to procedurally generate content for a 2-D platformer. This article recomends that Launchpad: A Rhythm-Based Level Generator is the most appropriate PCG if a Indie developer is wanting to make a procedurally generated 2-D platformer.

1 Introduction

The way in which we use Procedural Content Generators (PCG) to develop content for games is a vital part of the video game design process. With the recent release of popular procedurally generated games, such as Minecraft, Elite: Dangerous and The Binding of Isaac, it is important to choose a procedural content generator that suits your needs, as there are a lot of

different implementations of PCG to choose from, each designed for a particular enviroment. The work is motivated by the complexity and variety of all the different approaches to Procedural Content Generation, and aims to provide a comparison of three different papers to make a recommendation based on qualities that would suit the needs for an Indie developer that would like to implement a PCG into their 2-D platformer game.

There is always going to be advantages and disadvantages of one PCG over another depending on the context. However in the context of a 2-D platformer, these are three PCG articles that are designed to generate levels in the style of Super Mario Bros(SMB).

1: Launchpad: A Rhythm-Based Level Generator for 2-D platformers[1]

2: Procedural Level Generation Using Occupancy-Regulated Extension [2]

3: Procedural Content Generation Using Patterns as Objectives[3]

2 Review of Procedural Content Generation Articles

There are many different approaches to implementing PCG in games, the reason I chose these three articles is because they were specified towards 2-D platformers. In particular, they were all loosely based of SMB, and generate content in that same style.

The recommendation is based off the desirable characteristics in each PCG, in the context of what a Indie developer that would like to implement a PCG. The metrics that will be important to a Indie developer are flexibility(ease of implementation and customization of technique), playability, *leniency* , *Linearity*[1], [4] and amount of human authoring needed for content/levels.

[1]leniency is how forgiving the level is to the player, E.g. levels with less obstacles or enemy's are less lenient. Linearity is a measure of the varying geometry of a level

2.1 Launchpad: A Rhythm-Based Level Generator

Launchpad generates levels out of small segments called "*rhythm groups*" which are short, non-overlapping areas that contain a challenge for the player. I.e. a series of quick jumps or jumping back and forth between platforms. The rhythm groups are generated using a 2 tiered grammar [2] based approach.

The first stage of this creates a set of beats that correspond to a player action, this is good because it means the game can be improved by adding more varying player actions. The second stage takes the player action and creates geometry based on its parameters, this is then added to the level as a "rhythm group".

To achieve guaranteed playability of levels, Launchpad generates many different levels which are then tested against their leniency and linearity to find the best level. This design works well because it lets the program be very flexible, and each stage has its own setting that can be edited to make the program look how the designer intended. Furthermore this will help eliminate any unwanted linearity that may occur.

However this approach is designed for a "speedrun" type of gameplay, and does not currently work well with exploration type of gameplay.

2.2 Procedural Level Generation Using Occupancy-Regulated Extension

This procedural level generator is to be able to generate levels without knowing anything about how the game works. This method works by using a library of chunks and assembling them based on a complex set of parameters.

This Algorithm generates the content in 3 stages, firstly it selects the

[2] Grammar in this context meaning a set of rules for rewriting strings, which forms the basics of how procedural content generation works[5].

location at which to start generating geometry. It then selects chunks from the library that is then tested with its filtering and selection algorithms to see if it is appropriate to place at that location. Lastly that chunk is integrated within the existing content.

The only aspect that is not general purpose is the post-processing algorithm which is very specific to the game, because it adds textures to the sides of blocks and changes the amount of power-up boxes and coins on the level.

the levels that it generates can be very creative and complex

It is not very quick at generating levels with large chunk- library's

Relies on a large chunk library to work well

Requires a lot of post-processing

This approach does not guarantee playable levels, which is a very undesirable property when creating a level.

2.3 Procedural Content Generation Using Patterns as Objectives

They take an interesting approach to PCG, by looking at the way levels are structured in an original game such as SMB, then analyzing that level design to generate levels that use the same design techniques.

This method works by identifying "micro-patterns" and "meso-patterns" in an existing game, such as SMB. Micro-patterns are a vertical 1-block wide slice of the level, and meso-patterns are the way in which micro-patterns are arranged. The generator then uses the micro-patterns as building blocks to generate a new level.

This PCG is not ideal for an indie developer as it requires a version of an existing game to work, this means that they must create a game and a level in order to generate more levels.

2.4 comparisons

Flexibility

Amount of human authoring needed for content/levels

Leniency

Linearity

Playability

Ease of implementation

3 Recommendation

Your essay must make a clear recommendation, in terms of which of the three techniques you have reviewed is the best according to whichever metric or metrics you feel is most appropriate. You must justify your choice, backing it up with empirical evidence. However remember that an academic essay is not a murder mystery: you should already have briefly discussed your recommendation in the introduction and in other parts of the essay. Do not save it for a grand reveal at the end.

4 Conclusion

Write your conclusion here. The conclusion should do more than summarise the essay, making clear the contribution of the work and highlighting key points, limitations, and outstanding questions. It should not introduce any new content or information.

References

[1] G. Smith, M. Treanor, J. Whitehead, and M. Mateas, "Rhythm-based level generation for 2d platformers," in *Proceedings of the 4th International Conference on Foundations of Digital Games*, pp. 175–182, ACM, 2009.

[2] P. Mawhorter and M. Mateas, "Procedural level generation using occupancy-regulated extension," in *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pp. 351–358, IEEE, 2010.

[3] S. Dahlskog and J. Togelius, "Procedural content generation using patterns as objectives," in *Applications of Evolutionary Computation*, pp. 325–336, Springer, 2014.

[4] G. Smith and J. Whitehead, "Analyzing the expressive range of a level generator," in *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, p. 4, ACM, 2010.

[5] N. Shaker, J. Togelius, and M. J. Nelson, *Procedural Content Generation in Games: A Textbook and an Overview of Current Research.* Springer, 2015.