The Evaluation of Variation and Flexibility of Differing Procedural Content Generators for 2D Platform Games

COMP110 - Computer Architecture Essay

1503048

December 7, 2015

In this paper, three methods of procedural content generation shall be examined, in terms of the variation and flexibility of input provided. The overarching theme of recommendation is influenced by the application of each method with respect to the genre of 2D platform games. The intended recommendation is for an indie game development studio.

1 Introduction

Procedural content generation (PCG) for games is a method of creating in-game objects and components. In terms of a 2D platformer, this can include components such as: enemies, gaps for the player to jump and platforms to land on.

There are a number of issues to consider when using PCG, such as the cost effectiveness in comparison to a designer manually creating a level. However, in the scope of intended replayability, it is suitable for PCG use due to its requirement of varied content generation.

Three methods shall be examined: Smith et al's grammar-based approach using Launchpad [1]; Jennings-Teats et al's Polymorph using Dynamic Difficulty Adjustment (DDA) [2]; and Dahlskog et al's Multi-level Level Generator based on design patterns [3].

The metric used is the variation of generation and flexibility of input that each method provides. Furthermore, the conclusion should draw towards a clear recommendation for implementation in a 2D platform game to be produced by an indie company.

2 Methods

2.1 Introduction

Launchpad is Smith et al's [1] method for procedural content generation, for a 2D platformer level design. This design consists of a two-tiered, grammar-based approach. Their objective is for creating an autonomous, designer-guided level generator. It is a rhythm based level generator, where the term rhythm is explained through player pacing, movements and actions. For example, a player aims to match a series of actions with a rhythm of hitting the controller buttons in a speedrun behaviour [1].

Kremers theory of level design supports this necessity of maintaining pace of a game, as it is described to be crucial to continual player engagement [4, pp. 246].

Jennings-Teats et al [2] discuss the use of Polymorph's dynamic level generation. This method is produced using DDA which changes gameplay on-the-fly. This program aims to produce "continually-appropriate challenge" for its players [5].

Similarly, it has been supported by Hunicke [6], that the appropriate level of difficulty for individual players is essential for enjoyment.

Dahlskog et al's [3] approach uses a search-based, bottom-up approach. This method builds on analysis of Super Mario Bros [7] levels into three levels of abstraction; using micro-, meso-, macro-patterns. These micro-patterns are used as the building blocks for the game producing vertical slices of the level which are then built into meso-patterns.

2.2 Flexibility

Launchpad allows the designer to manipulate a variety of parameters in order to give guidance and provide a rule set for the generator.

The parameter of rhythm can be manipulated through fields such as: evenly spaced beats, short followed by long beats or randomly spaced beats. These variables are also known as "regular", "swing" or "random" respectively [1, pp. 7].

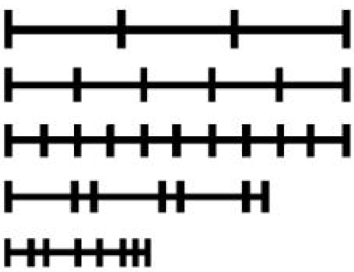

Figure 1: Examples of rhythm modification using type, density and length [1, pp. 7].

Fig. 1 shows examples of how the designer could modify the rhythm groups based on length, type and density.

The parameter input for Polymorph is based on player interaction. This not only allows for dynamic generation of adjacent components, but also the individual flexibility for personalised generation. Therefore, this method of DDA allows the designer to present the user with segments that are then assigned to levels by difficulty.

Enemies	
Enemy	A single enemy
2-Horde	Two enemies together
3-Horde	Three enemies together
4-Horde	Four enemies together
Roof	Enemies underneath a hanging platform making Mario bounce in the ceiling
Gaps	
Gaps	Single gap in the ground/platform
Multiple gaps	More than one gap with fixed platforms in between
Variable gaps	Gap and platform width is variable
Gap enemy	Enemies in the air above gaps
Pillar gap	Pillar (pipes or blocks) are placed on platforms between gaps
Valleys	
Valley	A valley created by using vertically stacked blocks or pipes but without Piranha plant(s)
Pipe valley	A valley with pipes and Piranha plant(s)
Empty valley	A valley without enemies
Enemy valley	A valley with enemies
Roof valley	A valley with enemies and a roof making Mario bounce in the ceiling
Multiple paths	
2-Path	A hanging platform allowing Mario to choose different paths
3-Path	2 hanging platforms allowing Mario to choose different paths
Risk and Reward	A multiple path where one path have a reward and a gap or enemy making it risky to go for the reward
Stairs	
Stair up	A stair going up
Stair down	A stair going down
Empty stair valley	A valley between a stair up and a stair down without enemies
Enemy stair valley	A valley between a stair up and a stair down with enemies
Gap stair valley	A valley between a stair up and a stair down with gap in the middle

Figure 2: Grouped patterns of game component based on theme [3, pp. 2].

Dahlskog et al's method of using a Multi-level Level Generator allows developers to create and analysis the difficulty of combined component groups present in these patterns. For examples of groups, see Fig. 2.

Due to the bottom-up approach of this method, there is not much designer input, beyond the initial extracted sequence. The fitness function used measures the presence and order of existing patterns.

2.3 Variation

Launchpad's use of geometric interpretation of rhythm should be reflective of the designer input. It provides the user with varied geometric displays of rhythm. It also claims to uphold the integrity of the intended rhythm while still generating an "impressive variety of levels" [1].

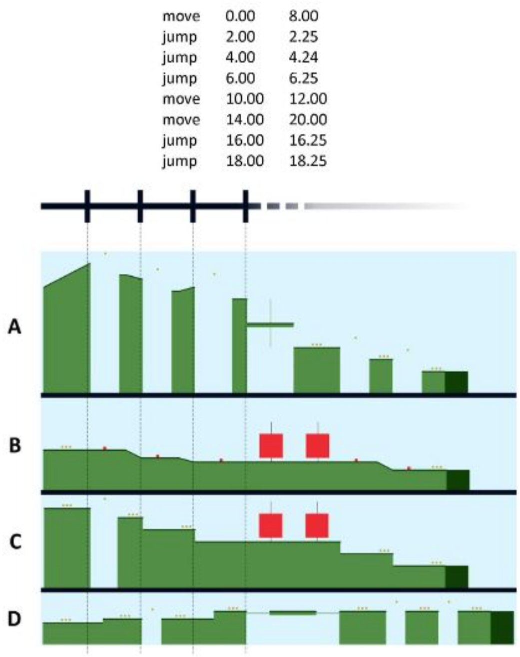

Figure 3: Four variations of geometric interpretation based on the same rhythm parameter set [1, pp. 8].

In Fig. 3, four levels are generated and directly comparable to the base rhythm set provided by the designer. The green platforms can be seen to fall or rise in line with the rhythm pace. The large red squares in level B and

C represent a stomper or a wait component. This has been interpreted in other variations, such as A and D, as a moving platform. It is debatable as to whether the intended rhythm is maintained. There is significant difference in player movement required at the beginning of segment B in comparison to segment A.

Polymorph works by evaluating player behaviour through use of its learned model trained from its collected data. An applied Multilayer Perceptron algorithm is used to rank difficulty of generated level segments. In combination with the data collected from player's current performance, the next segment of the level is generated based on the player's behaviour.

Feature	Correlation coefficient
Gap_Kill	0.7749
JumpUp	0.7095
Gap_Gap	0.6765
Gap_Avoid	0.6177
FlatGap	–0.5316
KillEnemy	–0.5222
Avoid	0.3818
JumpDownGap	0.3219
JumpUpGap	–0.0842
Avoid_Thwomp	0.0709

Figure 4: Correlation between level of difficulty rating and game component [2, pp. 141].

The data collected from gameplay, as shown in Fig. 4, has allowed the Polymorph model to assess the grouped and single components of a game based on difficulty. These results are then fed back into the next segment of the level to be generated and provide the player with appropriate difficulty.

The Multi-level Level Generator uses a fitness function that allows "single-point mutation and one-point crossover" [3, pp. 4]. The mutation basis for generated variation operates through exchange of a set of five vertical slices "at a random starting position with a new random set of five slices" [3, pp. 4].

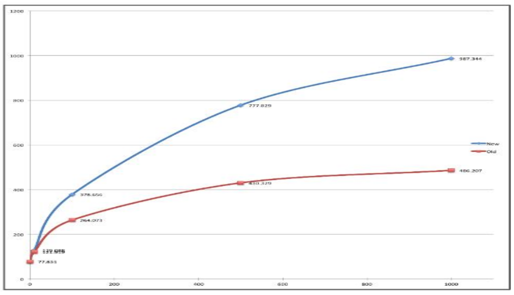

Figure 5: A comparison between the effects of mutation-operators between single (red) and five (blue) slice exchange [3, pp. 4].

This allows for greater variation between levels as larger segmented patterns are altered. This is shown in Fig. 5.

3 Recommendation

Launchpad provides a flexible generator, which allows the designer to clearly manipulate the pacing of the game level as intended with some reliability.

Polymorph has shown to be a very successful data collection generator. This allows for dynamically adjusted and generated gameplay through player interaction. This on-the-fly difficulty adjustment means a reduction in player frustration.

Multi-level Level Generation has shown extensive experience with pattern deconstruction and combination. The mutation-operators have also shown effective in generating new content based on pattern replication and exchange.

Overall, Polymorph's DDA approach provides a level balance of designer input and personalised level difficulty, with variation to generation. For an indie company, this reduces work load while still providing an enjoyable experience for players.

References

[1] G. Smith, J. Whitehead, M. Mateas, M. Treanor, J. March, and M. Cha, "Launchpad: A rhythm-based level generator for 2-d platformers," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, pp. 1–16, Mar. 2011.

[2] M. Jennings-Teats, G. Smith, and N. Wardrip-Fruin, "Polymorph: A model for dynamic level generation," in *Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, (Stanford, California).

[3] S. Dahlskog and J. Togelius, "A multi-level level generator," in *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*, (Dortmund, Germany).

[4] R. Kremers, *Level Design: Concept, Theory, and Practice.* United States: A.K. Peters, 2009.

[5] M. Jennings-Teats, G. Smith, and N. Wardrip-Fruin, "Polymorph: dynamic difficulty adjustment through level generation," in *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, (New York, USA).

[6] R. Hunicke, "The case for dynamic difficulty adjustment in games," in *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, ACE '05, (New York, USA), pp. 429–433, ACM, 2005.

[7] S. Dahlskog and J. Togelius, "Patterns and procedural content generation: revisiting mario in world 1 level 1," in *Proceedings of the First Workshop on Design Patterns in Games*, p. 1, 2012.