Implementation of PCG: Three Examples But Which Is The Most Practical?

COMP110 - Computer Architecture Essay

1506530

November 23, 2015

Procedural content generation (PCG) has been used in games for decades now. It is an ever evolving method for creating emersive and reliable game that players can enjoy. This paper will provide three distinct methods for implementing PCG in a 2D platfrom game. They will be of an increasing implementation ease, starting from a relatively simple algorithm that is used to enhance the difficulty of a game. Then moving on to one that uses an already existing game to randomize its levels. Finally creating a complete new game based on PCG.

1 Introduction

Procedural content generation (PCG) is one of many methods used today to create and design levels, particularity 2D platformers. PCG not only creates a new experience for the player with each playthrough, but also when done correctly can keep development costs down, by removing the need for extra level designers[1]. The only real issue with PCG is which one to choose, with

the variety of algorithms that are available it can be daunting to decide on which direction to go. So with PCG its important to have a clear idea of the task you want to accomplish as this will clearly define which algorithm would be the best for that game. With this in mind this paper contains three different algorithms that could be used in a 2D Platform game.

1.1 Motivation

The three examples used are to be discussed on which algorithm would be the easiest to implement while still being the most efficient. They are as follows:

In section 2: An Iterative Content Adaptation Algorithm (ICA) used to enhance difficulty of a game[2].

Section 2.1: Genetic Algorithm (GA) used alongside an Agent-based Digger Algorithm to use and add assets to an established game[1].

Section 2.2: An Algorithm used alongside the Launchpad to create a whole new game[3].

In section 3: A comparison of efficiency will be drawn based on their respective results.

More detailed information can be found here [2, 1, 3] respectively. Aa brief a description as possible for each algorithm and how they work are presented in the following sections. Looking at ease of implementation the obvious choice would be the ICA due to its simplicity and ease of use on more than one game, more on this in section 2.

2 Iterative Content Algorithm (ICA)

The First Algorithm that is looked at is perhaps the easiest to implement, as it uses a graph overlay of a generated map to populate the map with objects

and obstacles. A graph is drawn form the geometry of each level to create something similar to the graph in Fig.1 Prince of Persia was used for these examples [2].

[2]

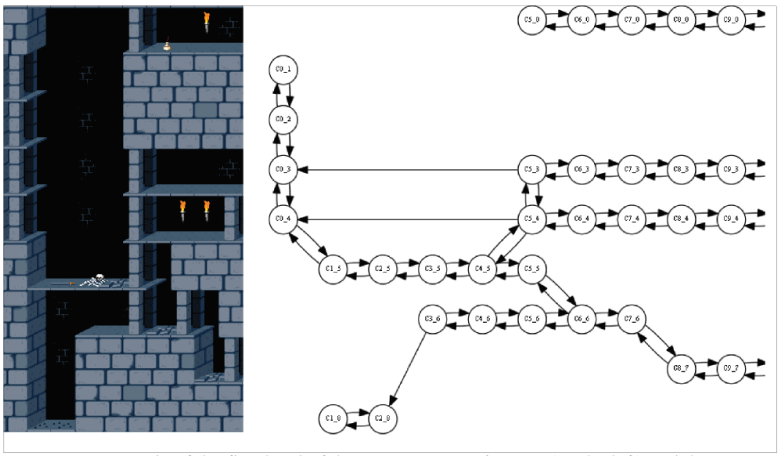


Figure 1: - A Sample of the first level of the game *Prince of Persia* (on the left) and the graph (on the right)

.

As you can see there is quite a bit of information here which requires more computational effort from the algorithm than is required. So the graph needs to be compressed as to reduce the amount of vertices and edges there are. For example the if a vertex is connected directly to another vertex it will stay, if not then its removed to produce a graph like Fig.2[2].

[2]

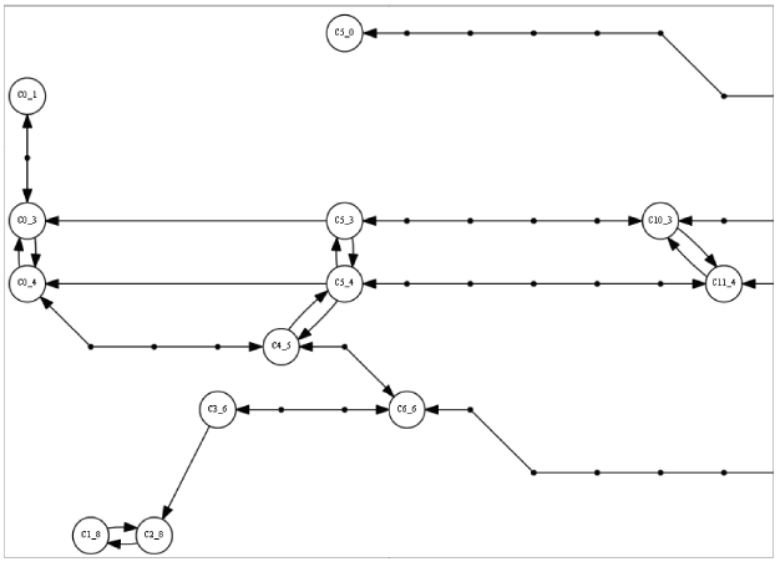


Figure 2: - Example of a compressed graph. The compressed nodes are marked with dots

.

As can be derived from these there is a significant difference. The ICA will then calculate the shortest/longest/alternative routes to the end of each level with the addition of dead-ends and a no way back path. This is gathered into a tree presenting all the information in a more readable manner alongside the graph Fig.3[2].

[2]

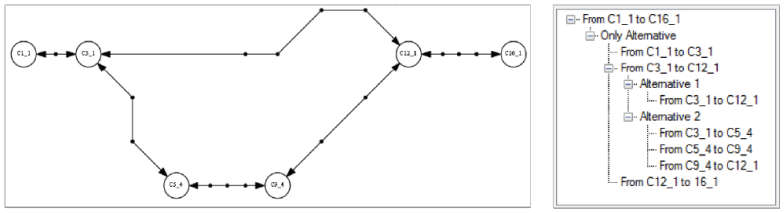


Figure 3: - Example of a graph (on the left) and the corresponding tree (on the right)

.

From this, its is overlayed onto a platform level Fig.4.Then with difficulty applied, in Fig.5 a difficulty level of 100 was tested[2].

[2]

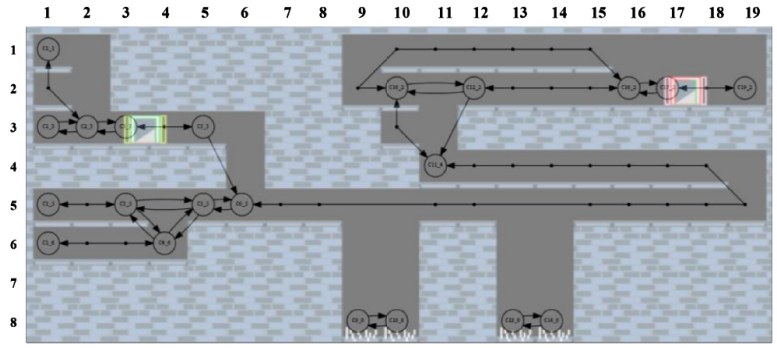
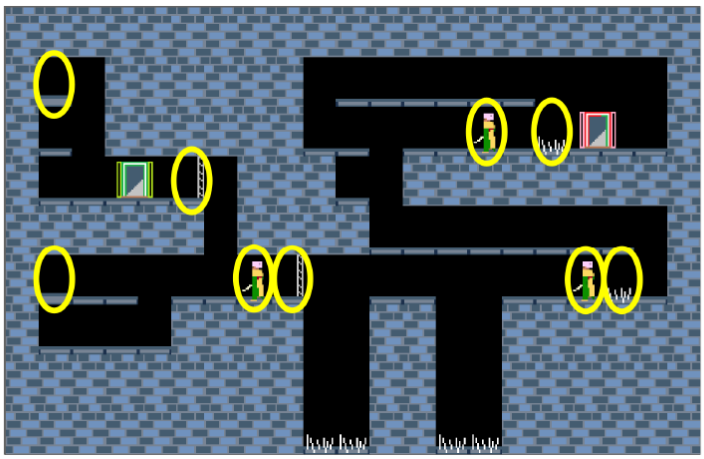


Figure 4: - The Graph generated for the example level



.

Figure 5: - Example of the level after the algorithm has been applied to *Prince of Persia*

.

From Fig.4 its can be seen that the algorithm also used the dead-ends to apply another challenge in the from of switches to remove the obstacles from the path. This method can also be applied to other platform based games as can be seen in Fig.6.7. This can be really useful for cutting costs in

development as it removes the need for object placement in games. Although the developer will still need to create the levels themselves, they would not need to be populated[2].

[2]

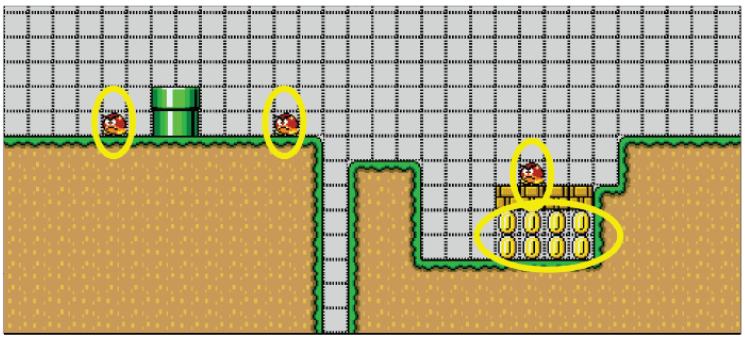
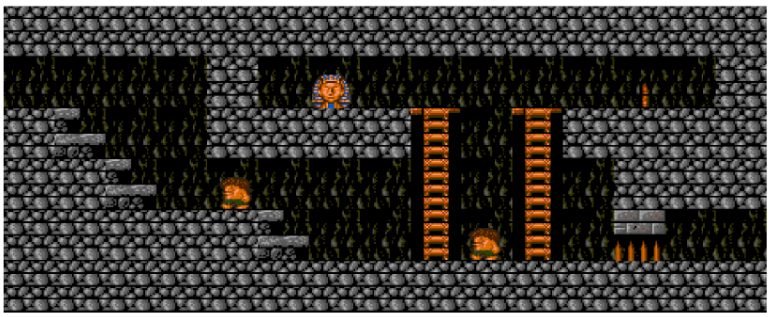


Figure 6: - Example of the level after the algorithm has been applied to *Infinite Mario Bros.*



.

Figure 7: -Example of the level after the algorithm has been applied to *XRick*

.

2.1 Genetic Algorithm (GA)

Genetic Algorithm (GA) is a search heuristic that will mimic the natural selection process, a taxonomy of a evolutionary algorithm that runs through hundreds of evolutions to determine the most optimized method for the task at hand[4]. Using this algorithm, Infispel, an adaptation of the famous

platform game Spelunky is a good example where this is used. A Genetic Algorithm (GA) along with graphs of the rooms and the paths connecting them is used to populate the levels with items and objects. While a agent-based Digger algorithm is used for the rooms structure depending on the difficulty selected Fig.8. Primarily for map building in a first-person shooter games the digger algorithm can provide a platformer game with interesting level builds Fig.8. [1]. The way in which the digger creates a level is its placed in one of the 16 rooms and is set of in a random direction, assigning each cell a value of 1(on) or 0(off). The digger will repeat this for a total of 35 moves, walls are then dug where necessary for room connection, then repeated again for the 16 rooms.

[1]

(a) Difficulty = 10% (b) Difficulty = 50% (c) Difficulty = 90%

Figure 8: - Three Examples of the digger outcome with increasing difficulty .

Through this method a developer could create a PCG game based only on these two Algorithms, but they would require quite a bit of modification before implementation unlike the ICA[1, 2].

2.2 Endless Web (EW)

The last and biggest method would be using the Launchpad along with a PCG algorithm to create a game that continuously builds the game depending on your play style[3]. With EW the approach taken was to use the Launchpad

as the primary PCG source then extensively modify it to suit their needs. Fig.9 shows the capabilities of the end product.

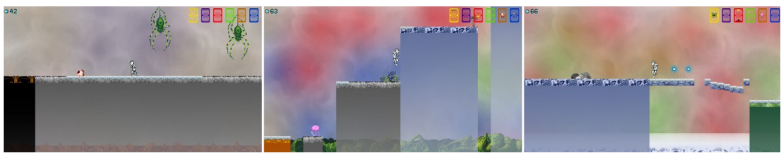


Figure 9: - Three different scenarios that can come from Endless Web
.

[3]

The game is dependant on the way you tune the world around you, by using different (Tuning Portals) and the proximity to friendly NPC's the world around you will automatically generate and change[3]. The development of such a PCG game would require a great deal of time and cost, for a small indie company it would be impractical to use such a method to create a game.

3 Recommendation

Through the Evidence provided in the whole section 2, the Iterative Content Algorithm (ICA) would be the strongest to use for a small indie company. As provided by section 2 the ICA is the easiest most practical method to implement PCG. It can support a wide variety of platfrom games with little to no modification[2]. While the other two methods would require stronger modifications and commitment to achieve[1, 3]

4 Conclusion

Although further research is needed on all three methods presented here, it is clear that PCG is still evolving in many directions. From small item placement as seen with the ICA to churning out an entire game as you play

as with The Endless Web. From this paper it is hoped that the reader can take away a fair and basic understanding of just a few ways which PCG can be done. Along with a good idea of which method would be useful when creating a game for themselves. A further question that could be asked is could these three methods be combined to create a new method of PCG. This would require further research into them for any feasible argument to be made.

References

[1] W. Baghdadi, F. Shams Eddin, R. Al-Omari, Z. Alhalawani, M. Shaker, and N. Shaker, *A Procedural Method for Automatic Generation of Spelunky Levels*, vol. 9028. Springer International Publishing, 2015-03-17.

[2] F. Maurato, F. Birra, and M. Prspero dos Santos, "Enhancing level difficulty and additional content in platform videogames through graph analysis," *ACE'12 Proceedings of the 9th international conference on Advances in Computer Entertainment*, pp. 70–84, 2012-11-03.

[3] G. Smith, A. Othenin-Girard, J. Whitehead, and N. Wardrip-Fruin, "Pcg-based game design: creating endless web," *Proceedings of the International Conference on the Foundations of Digital Games*, pp. 188–195, 2012-05-29.

[4] M. El-Abd and M. Kamel, *A Taxonomy of Cooperative Search Algorithms*, vol. 3636. Springer Berlin Heidelberg 2005, 2005-08-30.