Investigating Automated Level Design

Dudley Dawes [15017153]
Department of Computer Science and Creative Technologies
University of the West of England
Coldharbour Lane
Bristol, UK
dudley2.dawes@live.uwe.ac.uk

Summary Video:
https://youtu.be/tlCWWixwqvA

Abstract
The aim of this project is to research and implement multiple learning-based and search-based automated level design techniques. The types of techniques used are influenced by structures used in games and in other forms of media. This project hopes to give an insight into interesting ways level design automation can be conducted by altering commonly used structural and other techniques.

Author Keywords
Level Design; Procedural Generation; Automation

Introduction
This project consists of multiple prototypes developed to test automated level design techniques. The prototypes test one main technique each, which draws inspiration from other fields of research discussed in each of the following sections. However, due to unforeseen issues and the development of new techniques, crossovers between multiple techniques happens.

The following paragraphs will discuss the prototypes developed for this project.

Prototypes
The following sections contain an overview, discussion and evaluation of different prototypes developed for this project.

Sentence Generator
The *Sentence Generator* aims to automate level design by defining the sections found in platformer-game levels and replace them with words, using a parse tree (Setzer, 2014). The parse tree technique was used by (Matsubara, Kato and Egawa, 2008) to segment sentences into a noun (NP), verb (VP) and other parts. However, instead of compressing sentences using the parse tree this prototype uses the parse tree as a layout for English and level parts.

There are two versions of the *Sentence Generator*, generating sentences using English grammatical parts such as Determiner, Noun and Adverb, and generating sentences using common platformer-game level parts such as Safe Beginning, Skill Acquisition and Boss Fight. When researching platformer games such as *Ratchet & Clank* (Insomniac Games), *Super Mario Bros* (Nintendo Creative Department) and *Crash Bandicoot*

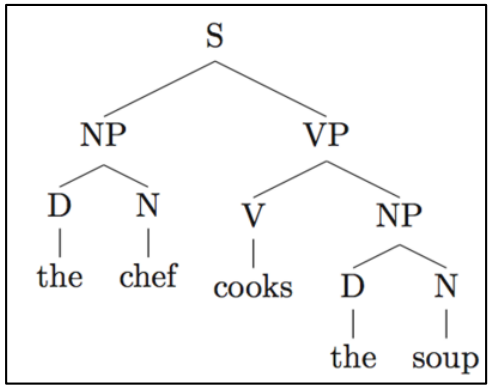


Figure 1 Parse tree for sentence structure.

(Naughty Dog) a clear correlation can be drawn between the types of parts a platformer-game level includes.

This prototype has shown that the parse tree is a good solution for level design automation when there is a pre-existing structure to replicate. The parse tree provides an organised way to structure sentences and levels. As seen in *The Hero's Journey* prototype this technique can be expanded to other forms of generation, game genres and levels which have a pre-existing structure.

Unlike other automation techniques researched in this project such as *Heights and Deltas* and *3D Level Generation*, the parse tree cannot learn and requires an explicit rule set to generate from (*figure 1*). To enable learning-based or search-based automation a system or technique such as a Neural Network (Summerville and Mateas, 2016) or Markov Chain (Snodgrass and Ontañón, 2013) could be utilised. Additionally, the *Sentence Generator* shows the capability of the parse tree to allow for complex data to replace words (*Appendix 4*).

The Hero's Journey

This prototype aims to automate the level design process by using the parse tree technique and the narrative structure The Hero's Journey. The Hero's Journey is a commonly used narrative structure used in films such as *Star Wars* (George Lucas) and *Shrek* (William Steig). It has twelve individual parts which can be used to define different sections in a platformer-game level. For the purpose of this prototype two parts which be chosen; *Crossing the Threshold* and *Returning Home* (Lim and Lee, 2014).

This prototype uses the parse tree technique implemented in the *Sentence Generator*. Instead of using the parse tree with The Hero's Journey sections, it is used to automate the platform layout, where the parse tree nodes become the platforms. The Hero's Journey sections are used as mechanics applied to the platforms depending on their position in the generation process (*Appendix 2*).

The Hero's Journey is easy to automate and define as a rule-set as it has an explicit structure. This prototype is an example of how structural forms in non-game specific media can be used as inspiration for procedural systems. The two elements of The Hero's Journey used in this prototype were easily integrated within a platformer-game environment. Therefore, it can be concluded that The Hero's Journey structural elements have the potential to be used as mechanics in other game environments, narratives or experiences.

The success of this prototype led to the development of prototypes which utilises other forms of non-game specific media structures (*Appendix 4*). Like the *Sentence Generator*, this prototype uses no learning techniques, however, there is clear potential for integration or crossover with other automation techniques such as learning a parse tree layout using the Markov Chain technique used for the *3D Level Generation* prototype.

Four Colour Theorem

This prototype looks at level design automation for exploration based game genres such as Rogue-likes along with world overviews and layouts such as the *New Mario Bros* (Nintendo EAD) world lobby view. For this purpose, the Four Colour Theorem is used. The

Four Colour Theorem is typically used to colour groups of shapes such as Maps (Thomas *et al.*, 1997). Due to the guarantee that no same colour will be next to each other, it makes it a viable solution for level automation.

This prototype uses the technique to procedurally generate levels similar to Rogue-likes or dungeon crawlers where each rectangle represents a different type of exploration zone or area. The algorithm used in this prototype is inspired by the Four Colour Theorem but does not use the exact algorithm due to the complexity of the mathematics. Instead, simple rules are used to stop same colours being generated next to each other.

Unlike previous prototypes, this prototype does not follow a pre-existing set of generation parameters. It does, however, give the user the ability to alter parameters of the generation rules (*Appendix* 3). Giving the user the ability to make alterations to the generation process is a technique used in other prototypes including *3D Level Generation* and is a way to utilise humans to tweak the automation process.

The output of the system shows that if developed further this prototype could be a viable solution to the automation of game worlds. The algorithm used in this prototype was inspired by the Four Colour Theorem and did not use the exact equation. This caused errors to become exponentially more computationally expensive as more rectangles are generated. Despite this, the prototype shows the potential for colours to be replaced with complex data which could describe the types of zones or exploration areas. For example, if generating a lobby view similar to *New Super Mario Bros* (Nintendo EAD) the colour could represent either a Water, Sky or Desert view with additional data detailing the difficulty and cost of failure. Another alternate use case for this prototype would be, instead of creating a dungeon crawler layout, to generate the colours onto a grid. A pathfinding algorithm could draw a line between the start and end, and use the sequence of colours as the zones or areas in a particular level or world.

3D Level Generation

This prototype aims to automate level design by learning from a pre-existing data set and uses a Markov Chain to learn the probability of transitions to each tile. The prototype is inspired by (Snodgrass and Ontañón, 2013) implementation. However, instead of parsing columns in a 2D tile map, this prototype parses (learns from) a 3D tile-based level in a forward, up and right direction. This prototype is also inspired by the *Number Generator* (*Appendix 5*) which learned the transitions between values in a sequence of numbers.

Similar to the *Four Colour Theorem* this prototype lets the user define the generation parameters such as the size and either First-Order or High-Order Markov Chain. The Markov Chain transition matrix is filled using a *.csv* file. Additionally, this technique shows how effective a transition matrix is when learning data. The technique is used again in the *Heights and Deltas* prototype.

In previous prototypes, the stochastic aspects are undirected and are used for selecting and placing objects or words. In this prototype, the stochastic elements are directed by the probability values learned from previously parsed levels. This leads to the issue that to generate a new or vastly different level a lot of data has to be fed into the system. In this prototype, only a single level is parsed to generate a new level

meaning generated levels always share similarities with the pre-designed level which was parsed. This could be positive or negative depending on the desired automation solution, however, the *Heights and Deltas* prototype aims to solve this issue by parsing multiple levels before generating a new level.

There are two main issues with the design of this prototype. The first is that Air tiles are not managed correctly. Air tiles like other tiles should be accounted for and added to a transition matrix in order to learn from the empty spaces. However, for a user to have to place Air tiles when creating a level is additional work and not ideal for professional projects. The second issue is that tiles are viewed as single entities. Levels are typically made with intricacies and features which want to be preserved. In this prototype, each tile is handled separately with leads to level features being ignored, scrambled or lost. This issue is addressed in the *Shapes* prototype.

Shapes

Inspired by Piet Mondrian's piece, *Composition II in Red, Blue and Yellow* and the main issues with the *3D Level Generator* prototype, this prototype aims to learn the individual features in a level instead of the individual tiles. The system parses a group of different shapes which learns their position, size and type. These variables are used by the generation system, along with the probability of the transitions between types, to generate a new shape in the world (*Appendix 7).*

Despite the short development period the prototype was implemented in, it is clear that the aim has potential use for mitigating issues found in parsing tile-based levels. The prototype did work as expected and individual objects were considered as a whole instead of groups of tiles however this leads to another issue, that many 2D levels are built with 2D tile sheets. This issue was tackled in this prototype by making a group of tiles a child objects of a single object. This method worked in this prototype but it difficult to see it as a solution without it being developed further. This is because when using the child object method attributes about the shape are lost.

Heights and Delta Values

This prototype aims to automate the level design process by learning from levels in which each tile is a height value. The system is inspired by research projects such as *Launchpad* (Whitehead *et al.*, 2011) and the *Markov Chain Number Generator* (*Appendix 5*). Unlike the *Launchpad* this prototype views individual tiles as values, which is similar to the *Markov Chain Number Generator* prototype. Instead of being arbitrary values, however, the values used in this prototype represent either a height or a change in height. The prototype also makes use of a simplified version of the transition matrix. It also takes into consideration the required solutions learned from the *3D Level Generator* and *Shapes* prototype in managing level features. However, unlike the *Shapes* prototype, this prototype uses backtracking.

There are two versions of this prototype both of which use probabilities and backtracking. Backtracking is the process of taking into consideration a number of previous tiles to the current tile. A higher backtracking value means that more tiles prior to the current tile are being taken into consideration. In all cases when a low backtracking has been used the tile heights fluctuate more variedly.

The first version of the prototype uses height values to identify the position of the tile, *Heights*. The second version uses the change in height, *Deltas*, to determine the position of the tile. The *Deltas* prototype was made to test the difference in output between the two techniques. It was found that less data was needed in the initialisation of the system making *Deltas* less complex. In *Heights*, prior to parsing and generation, the user needs to define the height values the tiles will be generated at. This is not ideal for a professional solution as it could lead to confusion in the initialisation process. In *Deltas*, this risk is mitigated as only a single initial position is required to generate from. This makes more sense as a ground level is a common feature in platformer games.

When using a low backtracking value both *Heights* and *Deltas* have fluctuating height variations compared with higher backtracking. However, in *Deltas*, the tiles plateau before fluctuating (*Appendix 8*). In contrast, when using high backtracking the tile heights fluctuate less. Two potential reasons for this is that there is not enough data to feed the system and the levels, one-hundred tile long, are too short for a high backtracking value.

Unlike the *3D Markov Chain* implementation, five levels were parsed. This number leads to interesting generations with a low backtracking value and keeps levels flat using high backtracking.

In this prototype, the aim is to generate levels which can always be completed. Unlike prototypes which are not learning-based such as the *Sentence Generator* and the *Four Colour Theorem*, this prototype only generates levels which are permutations of the parsed levels. If all the parsed levels can be completed, then any permutation must have the ability to be completed as they share the same characteristics. This theory was not correct. As seen in *Appendix* 8, situations arise when a level is unable to be completed. Typically, levels could not be completed as either, the players start position would be invalid. All the pre-designed levels tested followed the level design features; the Safe Beginning and Safe End Point.

Evaluation

This project has investigated learning-based and search-based level design automation techniques. It has aimed to use the techniques learned in each prototype as inspiration for new prototypes so that new prototypes can learn from the mistakes or issues which arise. Many of the prototypes were to exhibit a single technique. However, throughout the project techniques which were used found uses in other prototypes or crossover of techniques. Some example of this can be seen in *The Hero's Journey*, the *3D Level Generator* and the *Heights and Deltas* prototypes.

The prototypes developed for this project have touched the surfaces of an ever-growing body of academic and industry research into procedural generation and automated level design. The aim of this project was to look at the research through a different lens. To see what techniques can be used, found and inspired by to automate level design.

References

[1] Lim, G., Lee, J-D. (2014) Storytelling Design for Collaborative Learning based on the Hero's Journey. *International Journal of Software Engineering and Its Applications* [online]. 8 (8), pp. 105 – 118. [Accessed 20th April 2018].

[2] Matsubara, S., Kato, Y., Egawa, S. (2008) Sentence Compression by Structural Conversion of Parse Tree. *2008 Third International Conference on Digital Information Management*. London, UK, 13 – 16 November. IEEE, 544 – 550.

[3] Setzer, A. (2014) CS 275 Automata and Formal Language Theory. *Computer Science* [online]. Available from: https://pdfs.semanticscholar.org/presentation/efea/ab2dba40b91f079966171d510e62882066c6.pdf?_ga=2.236180865.1297610802.1524260225-2081825316.1524260225 [Accessed 20th April 2018].

[4] Snodgrass, S., Ontañón, S. (2013) Generating Maps using Markov Chains. *Artificial Intelligence and Game Aesthetics: Papers from the 2013 AIIDE Workshop (WS-13-19).* Boston, MA, USA, 14 – 18 October 2013.

[5] Summerville, A-J., Mateas, M. (2016) Super Mario as a String: Platformer Level Generation Via LSTMs. *DiGRA/FDG '16 - Proceedings of the First International Joint Conference of DiGRA and FDG*. 13 (1). [Accessed 10th January 2018].

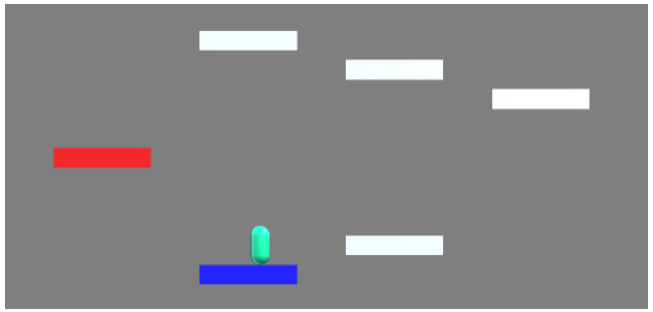
[6] Thomas, R., Seymour, P., Sanders, D., Robertson, N. (1997). The Four-Colour Theorem. *Journal of Combinatorial Theory* [online]. 70, pp. 2 – 44 [Accessed 15th April 2018].

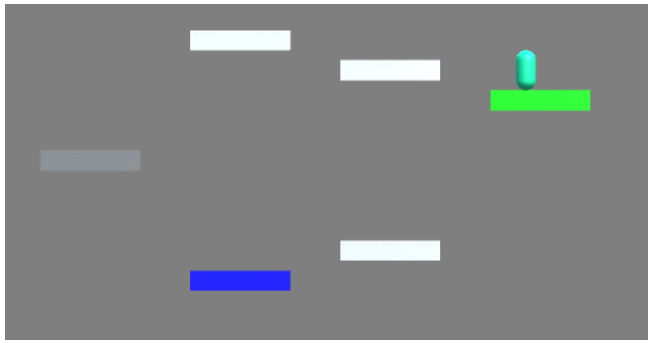
[7] Whitehead, J., Smith, G., Mateas, M., Treanor, M., March, J. Cha, M. (2011) Launchpad: A Rhythm-Based Level Generator for 2-D Platformers. *IEEE Transactions on Computational Intelligence and AI in Games [online]. 3, 1-16. [Accessed 13 October 2017].*

Appendix

[1] Sentence Generator
https://github.com/DudleyHK/LevelGenerationConsoleTest

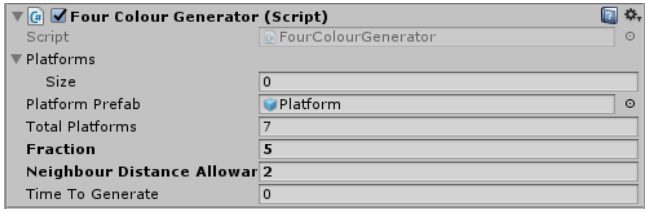
[2] The Hero's Journey
https://github.com/DudleyHK/UnityProceduralPlatformer





The blue box represents *Crossing the Threshold*, at this point the first box turns red. The player cannot return home. Once the player gets to the end, the end box turns green and the red turns to grey. The player can now return home, completing the level.

[3] Four Colour Theorem
https://github.com/DudleyHK/UnityProceduralPlatformer



This box gives the user the ability to edit the generation parameters.

[4] Poetry and Song Structure
https://github.com/DudleyHK/UnityProceduralPlatformer

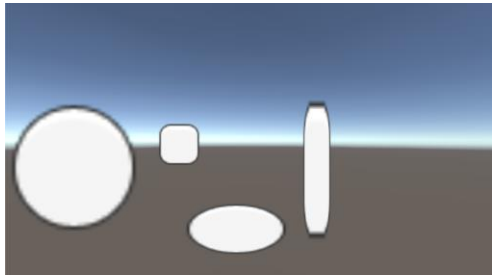
```
public class Obstacle
{
    public Obstacle(string name, ushort difficulty, ushort costOfFailure)
    {
        Name          = name;
        Difficulty    = difficulty;
        CostOfFailure = costOfFailure;
    }
```

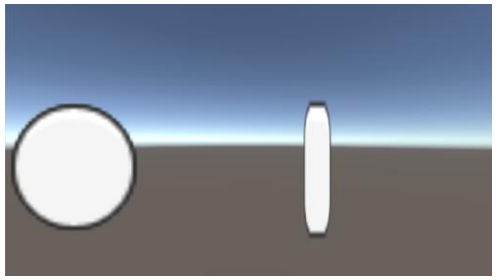
This is the class and constructor to replace the words in the generation system. The *Obstacle* class is acting as a complex data type.

[5] Number Generator
https://github.com/DudleyHK/LevelGenerationConsoleTest

[6] 3D Level Generator
https://github.com/DudleyHK/CTPPrototype

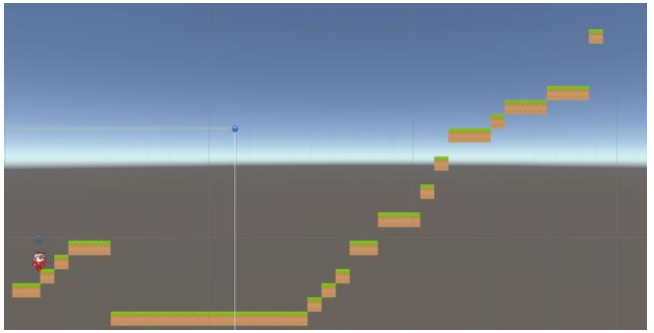
[7] Shapes
https://github.com/DudleyHK/CTPPrototypeProjects



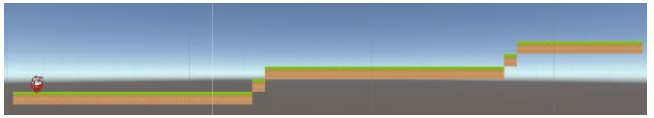


The shapes type, size and position are learned and used to build a transition matrix. Unlike other prototypes which use transition matrix, this data is not saved to a file.

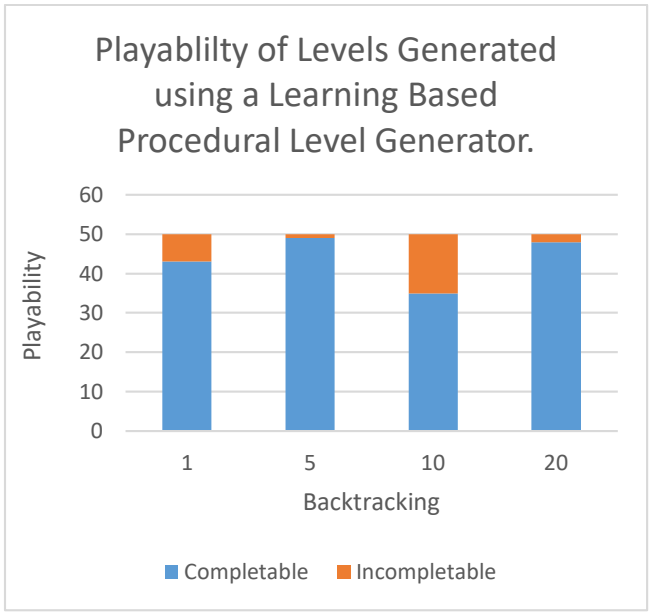
[8] Heights and Deltas
https://github.com/DudleyHK/CTPPrototypeProjects



Generated level with a backtracking of one. This is an example of using the change in height position.



Generated level with a backtracking of 10. This is an example of using the change in height position along with the plateau effect.



This test was run using the height positions as opposed to the change in heights. Each level was generated one-hundred times with each backtracking value.