Procedural Tool for Designing Platformer Levels

Dudley Dawes
15017153
Sean Butler
https://youtu.be/76XambrRW_M

* * *

A learning-based procedural level generation tool, allowing designers to iterate through different permutations of premade tile-based levels.

Literature Review

The project discussed in this document is a learning-based, procedural level generation tool, which allows designers to input a premade, tile-based, level into the system. The system learns from the data and outputs a permutation of the level. This computer aided design, is a repeatable process, where the computer works with the designer.

The intention of this document, is to give an insight into the research phase of this project. It will discuss areas of research surrounding some of the underlying topics of the project. In the following paragraphs the structure of the document will be presented in two parts.

The first part is the *Research* section. This section contains five sub-sections which discuss the main areas of research undertaken as a part of the research phase of this project. Moreover, the section will discuss the original research points bulleted in the *Proposal* document. The second part, is the *Planned Research Implementation* section. This section includes a summary of the next stage in development of this project. Finally, following the *References* and *Bibliography* are two additional sections; the *Log Sheets*, containing details on the bi-weekly meeting with the tutor, and the *Appendix*, a section of the document containing additional images and research, as part of the research phase.

The *Research* section gives a breakdown of the main areas of research undertaken for this project. It contains details of applications developed in other research, as well as prototypes developed for this project, to test the applicability of ideas and generative systems. The following paragraphs detail each sub-section of the *Research* section.

Structure, Rhythm and Patterns

The research in this section originally intended to meet the requirement of experiencing different games with the aim of finding a uniform vocabulary which defines the structure of platformer levels. This lead to research into; level design theory, reading *Level Design: Concept, Theory & Practice* (Kremers, 2009), rhythm-based platforming with a paper by Whitehead *et al*. (2011) and watching Let's Play videos of 3D platforming games such as *Ratchet and Clank* (Insomniac Games, 2016), and playing *Crash Bandicoot N. Sane Trilogy* (Vicarious Visions, 2017) which provided additional data on the way rhythm is used to force players to make quick decisions.

Using Structures from Other Media

This section discusses the two prototypes developed using the Parse Tree structure in Unity. The first prototype uses the research collated from story structure, The Hero's Journey, a commonly applied narrative structure used in story-driven media. And discusses the development of a 3D platformer visualisation of the Hero's Journey. The second prototype is inspired by research taken from the application *Launchpad* (Whitehead *et al.,* 2011) and discusses the 2D prototype using rhythm theory.

Finally, Rational Design was used in the *Hero's Journey Platformer* to determine the distance between the platforms a technique used by Brown (2017) when designing levels for *Hyper Light Drifter* (Heart Machine, 2016).

Building a Level out of Text

This section will provide details of the research surrounding the *Sentence Generation* prototype. Research in this field was to later influence a system which generated a level from a text file in the *Markov Chain Prototype* (see *Appendix G*). Another generation system, which could have been used instead of the Parse Tree structure is L-Systems. L-Systems are a fractal based grammar which is typically known for the algorithmic generation of plant structures (Prusinkiewicz and Lindenmayer, 1990) and other fractal based designs such as Road Networks and Cities (Smith and Bryson, 2014). Both systems would work for the *Sentence Generator* however, due to time constraints only the Parse Tree structure was tested.

Learning-based Generation

The implementation of *Rhythm Based Platformer* and the *Hero's Journey Platformer* used a tile-based system of procedural generation. In this section the *Number Generator* implementation will be discussed which uses a learning-based approach.

Research

Throughout the research phase the originally proposed questions have become less pertinent. However, as these questions gave guidance in the early stages of research the following paragraphs shall discuss these some of the ideas.

Research into referencing C++ Static or Dynamic-Link Libraries in Unity (Unity, 2017) is a small part of the research discussed in, *Learning-based Generation*. Though achievable, it is concluded that this technique is not required as prototypes are developed with C# and further development will be conducted in C++.

Procedural content generation (PCG) can be categorised in three ways, learning-based (LB), search-based (SB) and tile-based (TB). LBPCG learns from an existing data set i.e. the Markov Chain, where a probabilistic model of the transitions between states is built (Snodgrass and Ontañón, 2013). SBPCG uses the desired outcome as the data set and an algorithm such as the Genetic Algorithm (Wong and Bhojan, 2016) is used to find the best option. TBPCG builds a level from separate parts (Kazemi, 2013) i.e. the *Spelunky* (Mossmouth LLC, 2008) generator which algorithmically pieces together a level. The type of PCG being taken into further development is a mix between learning-based and tile-based.

Structure, Rhythm and Patterns

Song structure used in pop-music (*Figure 1*), takes advantage of a *Bridge* to break the pattern of a song. Likewise, Iambic Pentameter, found in poetic writing, uses off beats to break the pattern. Breaking patterns is a technique currently adopted in Games Development to create memorable experiences (Taylor, 2013). The *Rhythm Based Platformer* prototype was implemented to test this research.

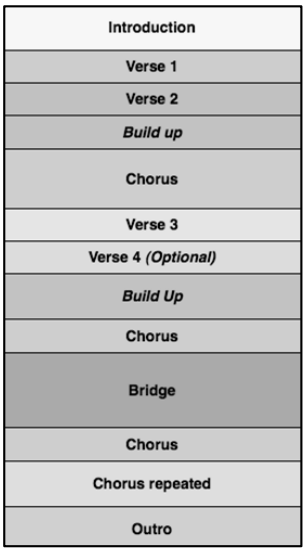


Figure 1 Pop song structure (Depellegrin, 2013)

The film industry uses narrative structures such as the Three-Act structure and the Hero's Journey, to engage the audience. The Hero's Journey includes a structural element, *The Point of No Return,* used to guide the protagonist. This is a narrative structure and therefore can be used in video games. However, to find out if the structure can be used to layout a level, the *Hero's Journey Platformer* prototype was implemented. The Three-Act structure was not developed further due to the positive data collected from the *Hero's Journey Platformer* prototype.

Using Structures from Other Media

The *Hero's Journey Platformer* prototype was developed to understand how *The Point of No Return* mechanic affects the structure and pattern of a level. Furthermore, the prototype employs the Parse Tree, with the aim of confirming its adaptability between the projects.

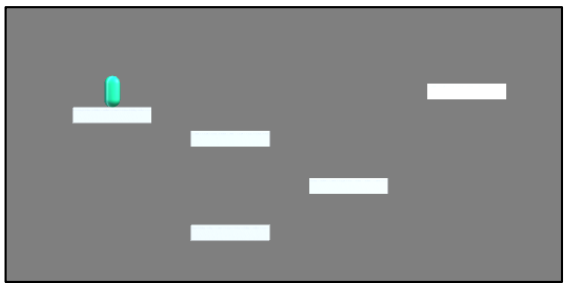


Figure 2 Orthographic view of the Hero's Journey Platformer, made in the Unity Engine.

The Parse Tree's nodes were used as platforms (*Figure 2*). This demonstrates the versatility of the structure and of the potential application of nodes to hold complex data. Additionally, *The Point of No Return* mechanic (*Appendix A*) has verified that structures used in other forms of media can be integrated in platformer level design.

The *Launchpad* (Whitehead *et al.,* 2011) inspired prototype, *Rhythm Based Platformer,* intended to adapt Iambic Pentameter for platformer level design and visualise it in a PCG system. As seen in *Appendix C*, strings were used as data, where each character represented a different obstacle. The system creates a sequence using the on, off pattern. Additionally, the system positions a break in the centre to break the pattern (*Figure 3)*.

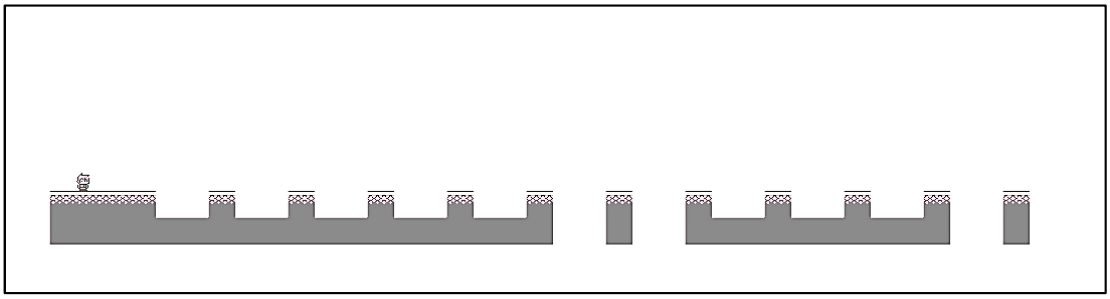


Figure 3 Rhythm Based Platformer output. The player, situated on the left, has Sprint and Double Jump to navigate the level. Made in the Unity Engine.

The system architecture was built in individual parts, a technique later used for the *Markov Chain Prototype* (*Appendix G)*. Additionally, the prototype illustrates how effective rhythm can be in building platformer levels.

Building a Level out of Text

The *Sentence Generator* prototype was implemented in C++ (*Appendix B)* and was the first application which aimed to determine how useful the Parse Tree structure is, for generative systems.

The application was developed to group words together into the structural elements of a sentence (*Figure 4*). The system then randomly selects words to output a different sentence every iteration.

In conclusion the Parse Tree structure is easy to apply. Additionally, the words could be replaced with different data, a theory tested in the *Hero's Journey Platformer.*

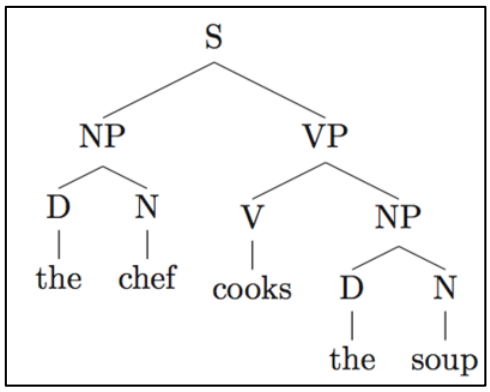


Figure 4 Visual representation of a Parse Tree structure.

Learning-based Generation

The *Number Generator* (*Figure 5)* uses a sequence of numbers as input data into a Markov Chain. A series of permutations are output based on the probability of transitions between numbers.

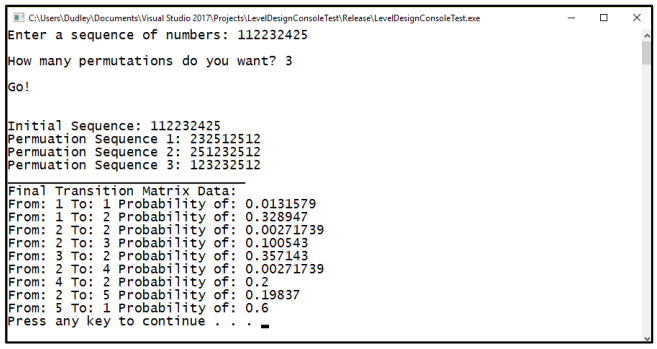


Figure 5 Number Generator console application, developed in C++.

The prototype was inspired by the research paper by Snodgrass and Ontañón (2013). A key difference is that this application uses a First-Order Markov Chain as opposed to the High-Order Markov Chain implemented in the paper. Additionally, the system learns from the previous permutation. This causes the transition matrix to become scrambled and output unvaried permutations. However, it can be concluded that the Markov Chain technique is well suited to a PCG system and will be developed further in the Unity Engine.

Planned Research Implementation

The research phase of this project has provided three main points for further development, using a High-Order Markov Chain to learn a tile-based level, decoupling the architecture of the system, and using C++ and DirectX 11 for the final implementation. This final point is the consequence of the ease of prototyping in Unity and the desire to pursue more of a challenge for the final project. The technical challenges of using C++ have been considered. For example, a system using the DirectX 11 rendering engine has been built and is ready to use with 3D geometry. However, as a failsafe the DirectX Toolkit is available for download and includes many helpful features and functionality.

References

Brown, L (2017) Level Design Workshop: Applying 3D Level Design Skills to the 2D World of 'Hyper Light Drifter'. *GDC Vault* [videocast]. Available from: http://www.gdcvault.com/play/1024306/Level-Design-Workshop-Applying-3D [Accessed 11th October 2017]

Depellegrin, S. (2013) Song Structure – Regular Pop Song (ABABCB). Song Writing with Sonia [blog]. 28th April. Available from: https://songswithsonia.wordpress.com/2013/04/28/song-structure-regular-pop-song-ababcb/ [Accessed 04th January 2018].

Kazemi, D. (2013) Spelunky Generator Lessons. *Tiny Subversions* [blog]. 19th October. Available from: http://tinysubversions.com/2013/10/spelunky-generator-lessons/index.html [Accessed 25th August 2017].

Kremers, R. (2009) *Level Design: Concept, Theory & Practice.* Massachusetts, A K Peters/CRC Press.

McEntee, C. (2012) Features. *Rational Design: The Core of Rayman Origins* [blog]. 27 March. Available from: https://www.gamasutra.com/view/feature/167214/rational_design_the_core_of_.php [Accessed 18 October 2017].

McMillan, L. (2013) Blogs. *The Rational Design Handbook: An Intro to RLD* [blog]. 08 June. Available from: https://www.gamasutra.com/blogs/LukeMcMillan/20130806/197147/The_Rational_Design_Handbook_An_Intro_to_RLD.php [Accessed 18 October 2017].

Prusinkiewicz, P., Lindenmayer, A. (1990) *The Algorithmic Beauty of Plants.* New York: Springer-Verlag.

Smith, A.J., Bryson, J.J. (2014) A logical approach to building dungeons: Answer set programming for hierarchical procedural content generation in roguelike games. *Proceedings of the 50th Anniversary Convention of the AISB.* London, 1 – 4 April 2014.

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.

Taylor, D. (2013) Ten Principles for Good Level Design. *GDC Vault* [videocast]. Available from: http://www.gdcvault.com/play/1017803/Ten-Principles-for-Good-Level [Accessed 23rd August 2017].

Unity (2017) Unity Documentation. Available from: https://docs.unity3d.com/2018.1/Documentation/Manual/NativePlugins.html [Accessed 04th January 2018]

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].

Bibliography

Coppin, B. (2004) *Artificial Intelligence Illuminated.* Sudbury, MA, USA: Jones and Bartlett.

McGuiness, C., Ashlock, D., Lee, C. (2011) Search-Based Procedural Generation of Maze-Like Levels. *IEEE Transactions on Computational Intelligence and AI in Games* [online]. 3 (3), pp. 260-273. [Accessed 17th June 2017].

Prusinkiewicz, P., Lindenmayer, A. (1990) *The Algorithmic Beauty of Plants.* New York: Springer-Verlag.

Richmond, P., Romano, D. (2008) Automatic Generation of Residential Areas using Geo-Demographics. *Advances in 3D Geo-Information Systems* [online]. 1 (2), pp. 401-416. [Accessed 16th June 2017].

Ripamonti, L.A., Mannalà, M., Gadia, D., Maggiorini, D. (2017) Procedural content generation for platformers: designing and testing FUN PLEdGE. *Multimedia Tools and Applications* [online]. 76 (4), pp. 5001-5050. [Accessed 17th June 2017].

Roque, L., Barreto, N., Cardoso, A. (2014) Computational creativity in procedural content generation: a state of the art survey. *Proceedings of Videojogos.* Barcelos, 5 November 2014.

Ruela, A.S., Guimarães, F.G. (2016) Procedural generation of non-player characters in massively multiplayer online strategy games. *Soft Computing* [online]. pp 1-16. [Accessed 17th June 2017].

Smith, G. (2015) An Analog History of Procedural Content Generation. *Proceedings of the 10th International Conference on the Foundation of Digital Games*. Pacific Groove, CA, USA, 22 – 25 June 2015.

Sorenson, N., Pasquier, P. and DiPaola, S. (2011) A Generic Approach to Challenge Modelling for the Procedural Creation of Video Game Level. *IEEE Transactions on Computational Intelligence and AI in Games* [online]. 3 (3), pp. 229-224. [Accessed 16th June 2017].

Wong, H.W., Bhojan, A. (2016) TITAL – Asynchronous multiplayer shooter with procedurally generated maps. *Entertainment Computing* [online]. 16, pp. 81-93. [Accessed 16th June 2017].

Appendixes

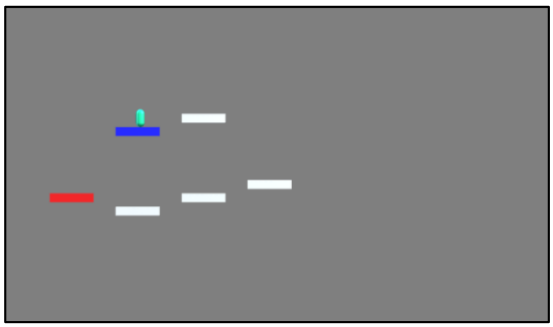
Log Sheets

Date	Tasks	Summary	Questions	References
28/09/2017	Look at level structures.	Try to find a correlation between different platformer levels.	Can platformer level structure be generalised?	Book - Level Design: Concepts, Theory and Practice.
	Update Sentence Generator system.	Update the sentence generator so the string it outputs represents an element of a platformer level, rather than Nouns, Verbs etc.	How can generating sentences be used for generating levels?	Game - Rayman Legends
12/10/2017	Research structures used in other forms of media.	Pick a well established structure found in Film, Literature or Music and use it as a base for a generative system.	Will I still use a Parse Tree structure? How can I break patterns using other creative structures?	The Hero's Journey. Pop-music song structure.
	Research Rational Level Design.	Rational Level Design is a good thing to have in mind when designing levels, look into what it is, where it has been used and how I can use it.	Who else has used these techniques?	The Rational Design Handbook.
26/10/2017	Look at the Four Colour Theorem.	The Four Colour Theorem is a maths problem where a group of joined shapes have to be coloured using as little colours as possible with no two same colours touching.	Is this a suitable generative technique?	

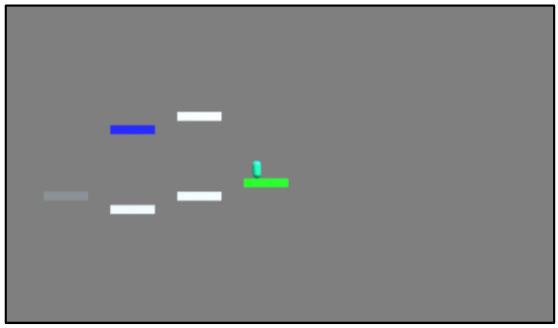
Date	Tasks	Summary	Questions	References
09/11/2017	Research Rhythm.	Develop a system in Unity which groups prefabs of tiles together and uses rhythm to drive the generation process.	How is rhythm used in Games and other forms of media such as Poetry?	Iambic Pentameter. Rhythm Based Platforming Paper.
23/11/2017	Conclude the Four Colour research and prototype.	Write up a paragraph on the out come of the four colour prototype.	How can the Four Colour Theorem project be used to generate a whole world?	
07/12/2017	Conclude the Rhythm prototype.	At this point every prototype has held similarities but with no clear progression or path to follow.	Are Markov Chains a positive way forward?	Paper - Experimenting with Map Generation using Markov Chains.
	Research Markov-Chains.	Research and develop a Markov-Chain number sequence generator. Implement a system which generates a sequence of numbers based on the number sequence inputted into the system.	What are Markov Chains and how do they work?	YouTube Videos and online tutorials.

Further Documentation

Appendix A: The Heros Jounrey Prototype.



Left: An image showing the player (Turquoise) reaching the gate (blue). The start (red) is locked.



Right: An image showing the player (Turquoise) reaching the end (green) and unlocking the start platform (grey).

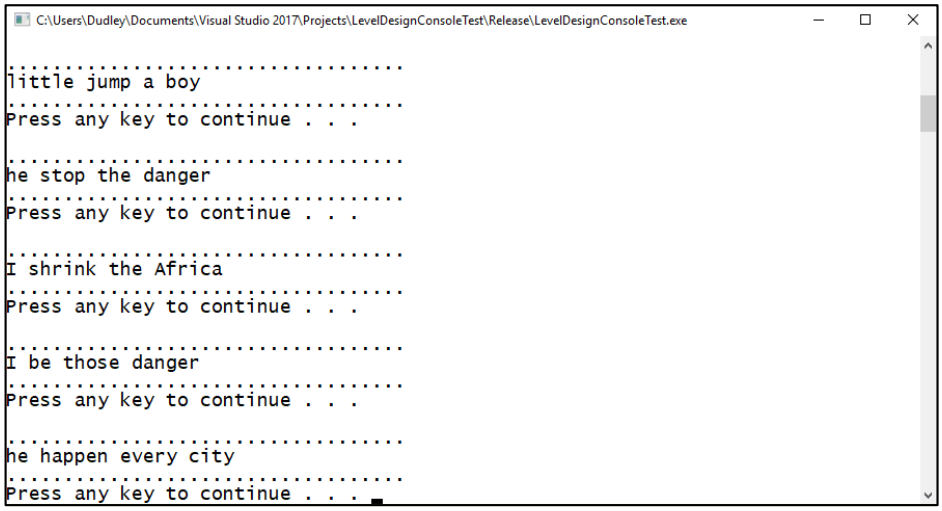
Appendix B: Sentence Generator.

```
void Generator::init()
{
    nouns =
    { ... }
    verbs =
    { ... }
    adjectives =
    { ... }
    adverbs =
    { ... }
    pronouns =
    { ... }
    prepositions =
    { ... }
    conjundetons =
    { ... }
    determiners =
    { ... }
}
```

Left: Initialistion of the word lists used by the generation system.

```
const std::string Generator::generate()
{
    std::string new_str = "";

    new_str += pronouns[getRandVal(0, 4)]    + SPACE;
    new_str += verbs[getRandVal(0, 9)]       + SPACE;
    new_str += determiners[getRandVal(0, 4)] + SPACE;
    new_str += nouns[getRandVal(0, 9)];
    return new_str;
}
```

Right: The C++ implementation of a Parse Tree structure.



Above: The outputted sentences, generated by the system.

Appendix C: Rhythm Based Platformer.

```
public List<Obstacle> RunObstacleGenerator(ushort totalObstacles)
{
    obstacleList = new List<Obstacle>();
    for (int i = 0; i < totalObstacles; i++)
    {
        if(i == 0)
        {
            obstacleList.Add(new Obstacle("FFFF", 1, 1));
            continue;
        }
        else if(i == (totalObstacles - 1))
        {
            obstacleList.Add(new Obstacle("FJJF", 1, 1));
        }
        else if (i == (totalObstacles / 2f))
        {
            obstacleList.Add(new Obstacle("FJJJ", 1, 1));
        }
        else if (i == (totalObstacles / 2f) + 1)
        {
            obstacleList.Add(new Obstacle("FJJJ", 1, 1));
        }
        else
        {
            obstacleList.Add(new Obstacle("FjjF", 1, 1));
        }
    }
```

Left: C# script which uses string data to simplify the generation process. This data will later be read by a parsing system which will convert each character into the correct sprite, thus creating obstacles.

Right: The parsing system. This converts the characters into a sprite prefab which is instantiated into the level.

```
private GameObject GetTileType(char type)
{
    switch (type)
    {
        case 'F': return floorPrefab;
        case 'G': return groundPrefab;
        case 'J': return jumpPrefab;
        case 'j': return null;
        case 'W': return null;
        case 'w': return null;
        case 'D': return null; // Danger
        case '*': return null;
    }
    return debugPrefab;
}
```

Appendix D: Number Generator.

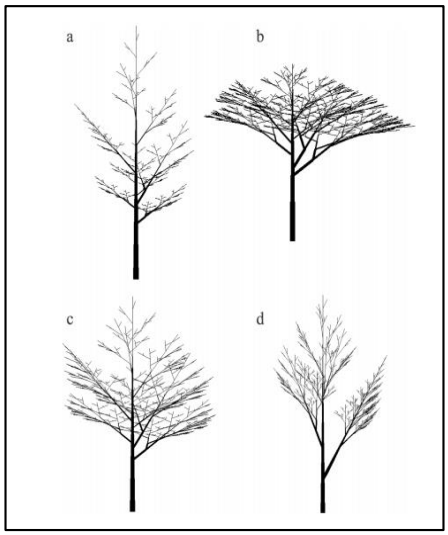
```
void MarkovMain::run()
{
    for(unsigned short i = 1; i <= number_of_permutations; i++)
    {
        fillTransitionMatrix();
        fillDenominatorList();
        calculateProbability();
        createNewSequence(i);
    }

    outputFinalData();
}
```

Above: The number generation loop developed using a First-Order Markov chain and C++.

```
std::vector<std::tuple<float, float, float>> transition_matrix;
std::vector<std::pair<float, float>> denominators;
```

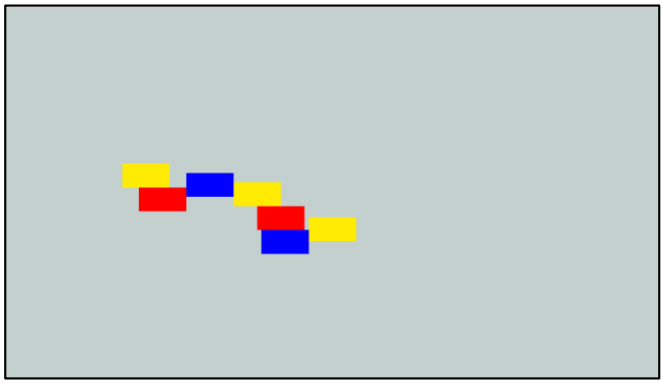
Above: The C++ implementation of the transition matrix and row totals. The transition matrices are filled with from, to and total values.

Appendix E: L-System for Tree Generation.



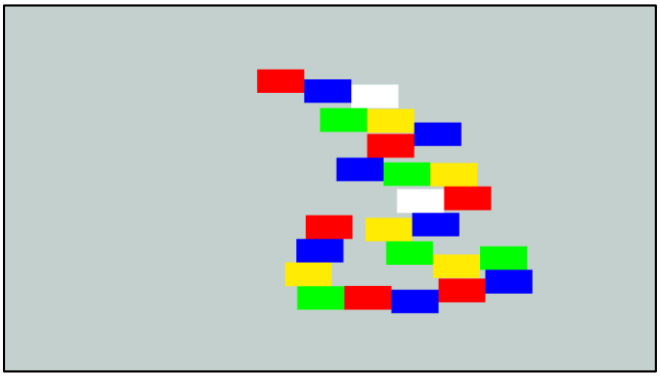
Above: A book by Prusinkiewicz and Lindenmayer (1990) detailing the many methods of using L-Systems to generate plants.

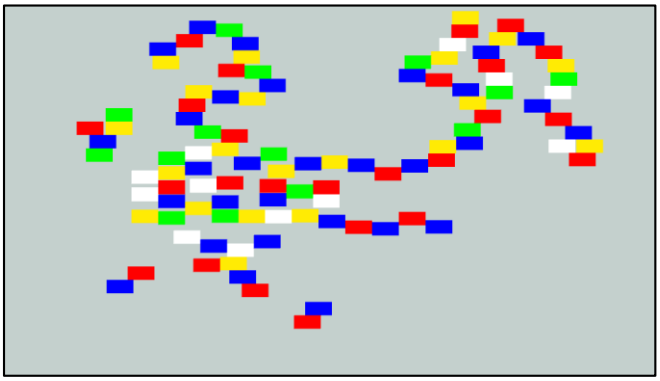
Appendix F: Four Colour Experiement



The Four Colour theory is an algorithm designed to use the least number of colours to fill a space, with no same colours touching. To test this the *Four Colour Experiment* prototype, developed using the Unity Engine, was implemented.

The vision for this prototype was to create a simple version of what, with more development, could be a grid which used Level Types such as Exploratory, Combat and Platforming instead of colours, thus creating an overview of a world similar to the Mario Worldview in *New Super Mario Bros.* (Nintendo EAD, 2006). One difference between the original theory and the prototype implementation is that the application does not consider the number of shapes it needs to colour.



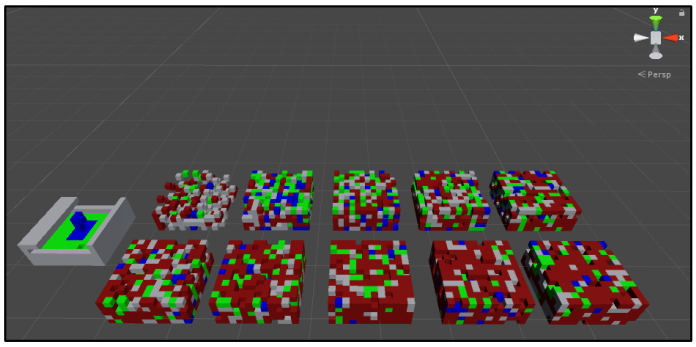


The images shown are the outputs of the prototype each displaying a different number of tiles.

Appendix G: Markov Chain Prototype

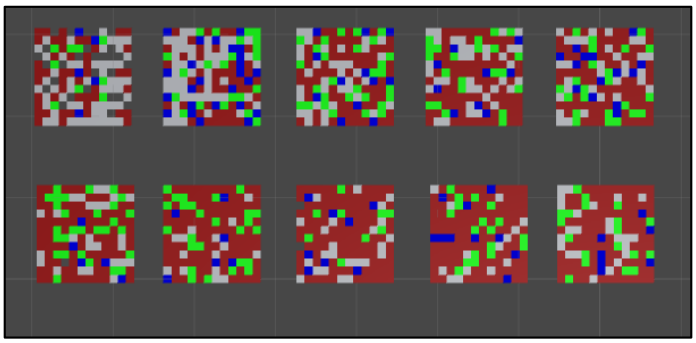
The follow series of images displays the output of a Markov Chain implementation, which uses a High-Order Markov Chain. Additional images show an experiment which aims to map out the speed of generation based on the size and starting tile.

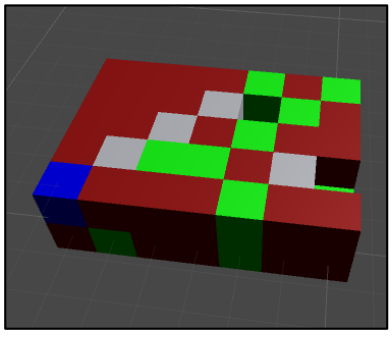
This is the prototype of the system which will be developed using C++. This system was developed in Unity using C#.



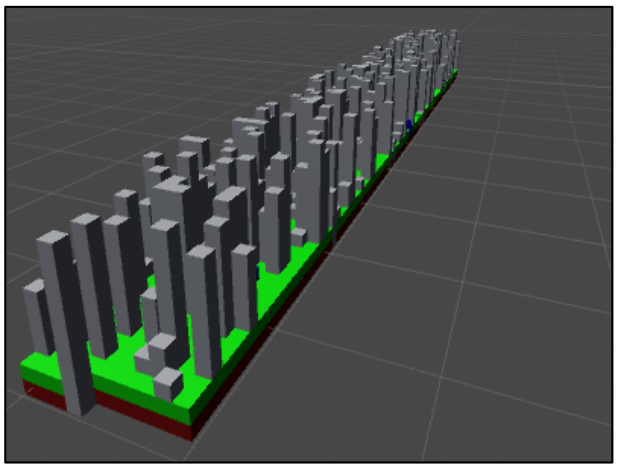
Left: Ten permutations of the far left, pre-made level.

Right: A top-down view of the image above.



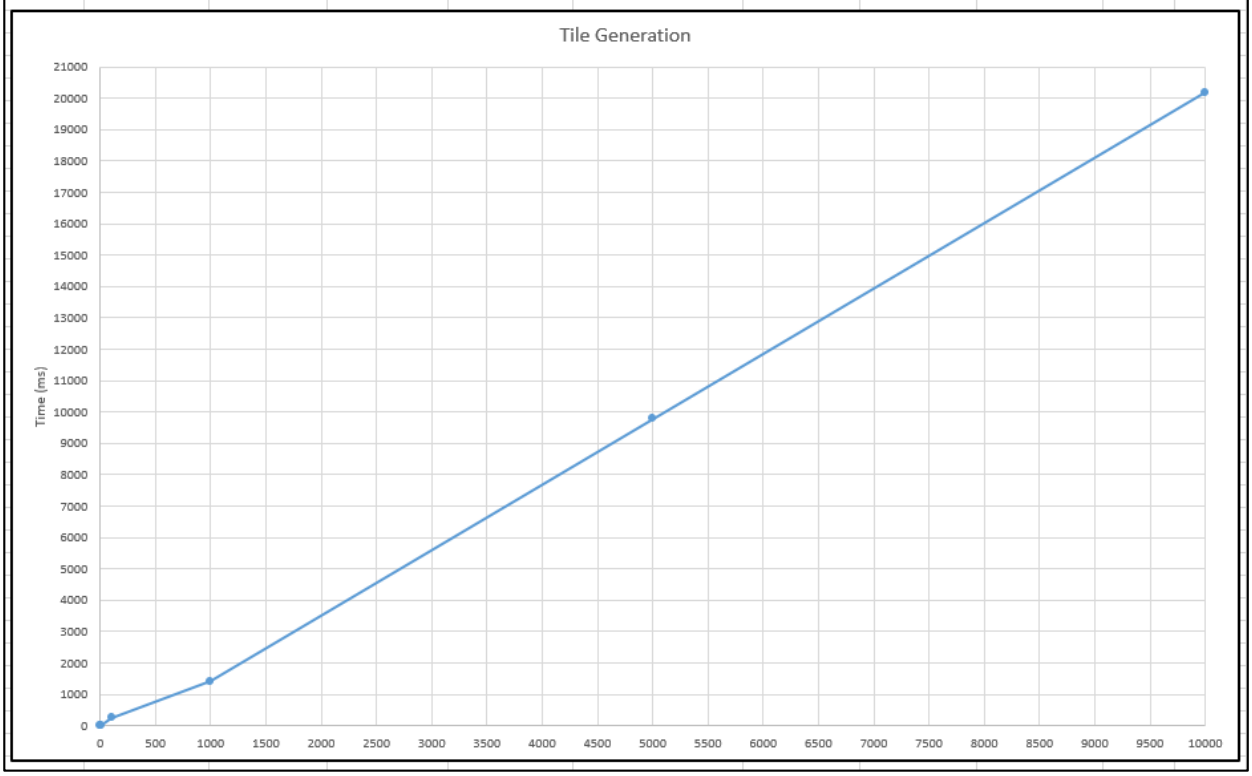


Above: A permutation of a pre-made level with smaller dimensions.



Above: A permutation of a pre-made level with larger dimensions.

	Time Taken (ms)					
Total Tiles	1 - Mud	2 - Grass	3 - Water	4 - Wall	5 - Air	Average
1	12.4088	8.6188	9.1958	8.3975	5.534	8.83098
10	23.4164	28.566	28.6107	30.015	30.2292	28.16746
100	214.3073	232.4245	240.0825	299.9081	299.9933	257.34314
1000	1692.1156	1804.7377	1783.5396	1751.0251	2.2183949	1406.727279
5000	8945.3062	10070.4998	9757.6826	9584.7893	10515.0441	9774.6644
10000	19088.82	19110.7968	21475.3125	19010.4632	22211.7146	20179.42142



Above: The results taken from the experiment aiming to show the speed of generation based on the size and starting tile. At the top are the values collected ad used to draw the graph.

Appendix H: Rational Level Design

The importance of integrating Rational Level Design (McMillan, 2013), discussed by McEntee (2012) former employee of Ubisoft, techniques into this project constricted due to the *Number Generator* prototype. Despite this Rational Level Design was used in *Hero's Journey Platformer* prototype to calculate the distance between platforms.

The technique was not taken further due to the project becoming a LBPCG system.