Bachelor Thesis
Digital Game Development
Thesis no: TA-2013:05
05 2013

Game Mechanics Integrated with a Lindenmayer System

Per Fornander

School of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona
Sweden

This thesis is submitted to the School of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Bachelor of Science in Digital Game Development. The thesis is equivalent to 10 weeks of full time studies.

Contact Information:
Author:
Per Fornander 910417-2771
E-mail: pefd10@student.bth.se

University advisor(s):
M.Eng Petar Jercic
School of Computing

School of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona
Sweden

Internet : www.bth.se/com
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

Abstract

Creating content for games costs a lot of time and money. This thesis will use the Lindenmayer system for procedural content creation inside the game to answer the question, how can L-systems be exploited for gameplay mechanic purposes? The application implemented in Unity 3D is a form of tree L-system with player controlled recursion, independent branches and poisonous branches. The recursion can allow the player to grow strategically, independent branches can be used as seed and grow on its own if they break off, poisonous branches to force players to think when navigating.

Keywords: L-system, Lindenmayer, game, procedural

Contents

Chapter 1

Introduction

In modern society, games have become the norm in our daily lives and as it reaches out to more people there is a larger demand for games [5]. Games are complex, time consuming and expensive to make. One of the way game studios tackle these problems are with procedural generation, where tools such as World Machine but also whole games like Minecraft [4] extensively use them. Another approach is to use a complete game engine with a known, easy pipeline and fast work flow, as the Unity 3D engine. Unity is a cross-platform engine for the web, PC, Mac, Linux, iOS, Android, Xbox360, PS3 and the Wii. Unity also have a free yet still fully capable quality version of their engine. With all of these positive aspects of Unity, it sure looks like the future of game development. The only comparable game engine is the Unreal Engine with its UDK kit. However compared to the Unity engine, the Unreal engine is more complex, reaches out to fewer platforms and that is why Unity will be this thesis's project platform.

Considering the fact that there already exists a lot of procedural generation tools, this thesis will focus on a procedural generation algorithm which is not commonly used inside games as systems, rather outside in the procedural content generation pipeline. The Lindenmayer system also know as L-system is a parallel rewriting system and will be explained in Section 1.2, which is most commonly used for simulation of plant growth but also other complex fractal patterns [8]. In games, L-systems are used in the content creation pipeline for trees but usually not in the actual game and even less as the actual game world.

To save developers time and money to develop better and unique games, this thesis combines Unity and L-systems to achieve this, by focusing on implementing an L-system into Unity. The reason for this implementation is also to explore and suggest more variations of gameplay mechanic creation in games with procedural methods, which will give the game industry more time and money to spend on other important tasks.

1.1 Game Mechanics

"Game mechanics are methods invoked by agents for interacting with the game world." states Sicart [10] in his article about defining game mechanics. Agents can mean either human or AI players in this context. This thesis will focus on the human player interaction with the world. Examples of game mechanics in the game Minecraft would be jumping, placing or destroying a voxel block and attacking.

1.2 L-systems

L-system is a fractal based parallel rewriting program created by Astrid Lindenmayer 1968 [8], an example is the snowflake generation in Figure 1.1. An L-system is built with a start piece, also known as the Axiom or initiator. The second piece is a modifier or otherwise known as a rule or generator, an L-system can have multiple rules for different scenarios. The Axiom exist from the beginning but gets modified every recursion by the modifiers. The purpose of L-systems are to simulate fractal growths which occur frequently in the organic world by trees, plants, bushes, cells and flowers. The Axiom and rules are built with help of an alphabet that the Axiom and the rules are made from, which is what is actually being modified and works as an interface between the L-system and the application.

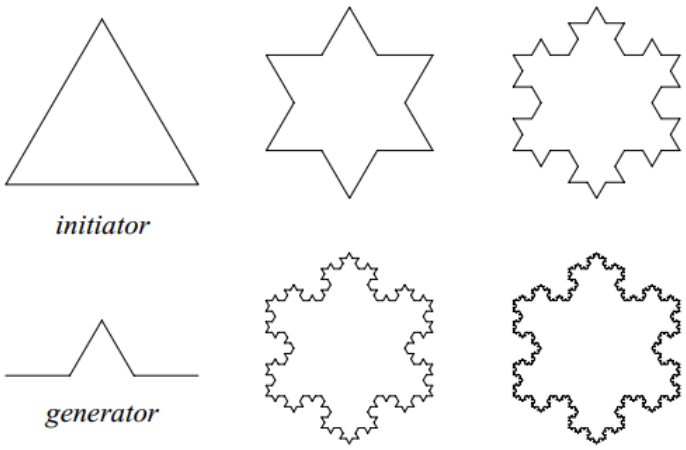

Figure 1.1: Construction of a fractal snowflake with four recursions.

1.2.1 DOL-systems

Variables	A B
Constants	none
Axiom	A
Rules	(A → AB), (B → A)

Table 1.1: Algae DOL-system variables, axiom and rules.

Recursion	Generated String	Applied rules
0	A	Only Axoim
1	AB	Runs rule 1x(A → AB) on the Axoim
2	ABA	Runs rule 1x(A → AB) and 1x(B → A) on recursion 1
3	ABAAB	Runs rule 2x(A → AB) and 1x(B → A) on recursion 2
4	ABAABABA	Runs rule 3x(A → AB) and 2x(B → A) on recursion 3
5	ABAABABAABAAB	Runs rule 5x(A → AB) and 3x(B → A) on recursion 4

Table 1.2: Algae DOL-system iterations.

DOL-systems are the simplest form of L-system and are context-free and deterministic. The **DOL** acronym stands for a **D**eterministic **L**-system with **0** interactions. Lindenmayer's original L-system for modeling the growth of algae is a good and simple example for displaying and explaining the basics of L-systems. The algae system consists of variable A and B, and the rules are (A → AB) which means that a string "A" will be replaced with "AB" and the second rule (B → A) will remake "B" to "A". Combining the rules in Table 1.1 allow to mimic the fractal pattern of algae, and starting with the Axiom A it will play out iteratively as in Table 1.2.

1.3 Procedural Generation

"Procedural techniques are code segments or algorithms that specify some characteristics of a computer-generated model or effect." [2]. An example of procedural generation is the color values on a digital marble surface. Instead of using an image to determine the color values, the color values will be determined by algorithms and mathematical functions.

Procedural generation is widely used in games because it allows for smaller a size on the hard disk drive compared to defining all the complexity of a complex

geometric shape, terrain for example. Procedural generation also cuts down development time and expenses, can allow for an almost infinite amount of world and general content variation for the player and the complex patterns used can bring interesting new experiences. Some of the games known for usage of procedural generation inside the games are Minecraft [7], Terraria [9], Dwarf Fotress [1]. A large number of games also uses procedural generation tools to create content for the game outside the actual game, in the pipeline using applications like SpeedTree by IDV.inc for trees and World Machine by World Machine Software for terrain.

1.4 Development Tools

This section covers a general knowledge of the applications used in this thesis project.

1.4.1 Unity 3D

Unity, is a 3D cross-platform game engine from Unity Technologies covering platforms such as web plugins, iOS, Android, Windows, Mac, Linux, PS3, XBOX 360 and WiiU. Covering this many platforms while also having a fast pipeline and work flow makes Unity a popular game engine for developers and have a developer base on over one million[11]. The game engine framework is written in C/C++ and it can be controlled with scripts written in C♯, Boo and Javascript.

1.4.2 Blender

Blender is a free and open-source 3D computer graphics software by Blender Foundation, used mainly for films, special effects and games. Blenders platforms are Mac OS X, Linux and Windows, it is writting in C, C++ and Python. Blender is controlled with Python and a graphical interface. Blender has many features, this thesis will be using 3D modelling and UV unwrapping to create assets for the project using Blender v2.66.

1.4.3 xNormal

xNormal is a free application that generates normal, ambient occlusion and displacement maps from a highpoly model on to a lowpoly model. xNormal can also generate more less common maps like cavity, curvature, thickness maps that Blender and other 3D-applications do not have support for.

1.4.4 Photoshop

Photoshop by Adobe systems, is a graphics editing program with a wide usability for everything graphic related.

1.4.5 Sculptris

Sculptris is a free 3D sculpting software created by Tomas Pettersson and is currently being developed by Pixologic.

1.5 Research Question

"How can an L-system be exploited for gameplay mechanic purposes?"

This research question will be explored and answered by combining L-system and gameplay mechanics to create an L-system with integrated mechanics that will allow the player to interact with the game world. The resulting gameplay mechanics will be the answer of how one can go about combining L-systems with game mechanics.

1.6 Document Roadmap

This document is structured as follow:

Chapter 2 will present "state of the art" of current known implementations and roles of L-systems.
Chapter 3 will state the research questions.
Chapter 4 will discuss design decisions of the project.
Chapter 5 will go through the process of reaching the final implementation.
Chapter 6 will present the results.
Chapter 7 will present conclusions, discussion and possible future work.

Chapter 2

L-Systems: State of the Art

L-systems was invented by an academic[8], and a lot of research on L-systems is being done by academics, examples of these below.

However actually utilizing L-system in a large scale and get the results out to the masses is up to the entertainment industry. The industry creates their own L-system tools to speed up production and lower costs when creating complex structures like trees and plants. The predominant L-system software will be briefed below in Section 2.4.

2.1 The algorithmic beauty of plants

"The algorithmic beauty of plants" by Prusinkiewicz and Lindenmayer [8], lays the foundation for a large portion of L-system research. In the book, Lindenmayer system that can generate plants or trees with a set of rules and factors. The book covers a lot of different areas where L-systems allows for simulated growth modeling of organic structures, such as trees, bushes, flowers, plants and cellular layers. Most of the L-systems are turtle graphic based and grow with each recursion by applying the rules. Turtle graphics is a method of programming vector graphics using a relative cursor or "turtle" upon a Cartesian plane. The basic factors of the growth besides the rules are at which recursion it is through and the angle value of the turn for the branch.

2.2 TreeSketch:Interactive Procedural Modeling of Trees on a Tablet

"TreeSketch:Interactive Procedural Modeling of Trees on a Tablet" [6] by Longay, Runions, Boudon, and Prusinkiewicz, is about an artistic and interactive software named TreeSketch. TreeSketch is a software that allows for creating complex trees with a natural look by using a multi-touch tablet. The platform allows them to have an aesthetic and creative approach in creating trees by simple brush strokes guiding the direction of how the tree grows and forms. It focus on interactive design and software between L-systems and humans by using games as a medium

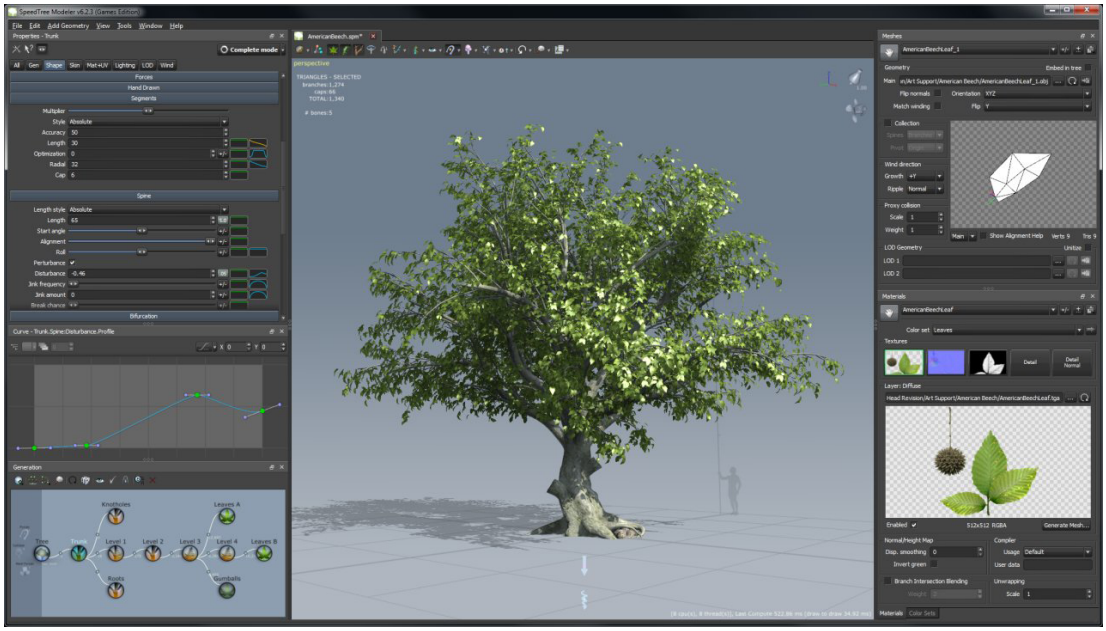

Figure 2.1: Screenshot of Speedtree Modeler.

this thesis proposes another approach to let players interact with L-systems. This thesis interacts by both controlling the growth recursions and be affected by the branches that spawn, with the player.

2.3 Context dependent generation of trees for computer games

"Kontextberoende generering av träd för dataspel" [3] (Translation from Swedish to English: "Context dependent generation of trees for computer games") is a bachelor thesis by Stefan Enberg who examines if context dependent tree generation is appropriate for games. Enberg used three factors as context, sunlight, wind and gravitation to shape the way the the trees are formed. The result was positive to using context dependent generation in games. "Kontextberoende generering av träd för dataspel" shares the same focus as this thesis, namely on L-systems with games. Enbergs thesis is about interaction from the world onto the trees, this thesis is approaching the interaction from the other side so that the tree interacts with the player. Interactions are the gameplay mechanic implementation where the toxic branches hurt the player.

Figure 2.2: Screenshot from Battlefield 3.

2.4 Industry L-systems

The majority of today's professional L-systems are used in game, architecture or film productions for botanic modeling and generation. For film, architecture or advertisement L-systems often exist as plugin-programs in 3D software applications like Maya, 3Ds Max and Blender. The most famous and dominating software on trees is SpeedTree by IDV.Inc which is divided into three software categories, movies, architecture and games. Famous movies like Avatar and "Pirates of the Caribbean: on Stranger Tides" were using SpeedTree to create their realistic digital vegetation landscapes. As for games, famous and successful titles like Battlefield: 3, the Witcher 2, Gears of War 3 used SpeedTree in the content creation process, Figure 2.2 shows SpeedTree trees in Battlefield: 3. SpeedTree for games enable sophisticated procedural generation as well as modeling, animation, rendering of trees and a rich library of up to 180 different species, the modeling interface can be seen in Figure 2.1.

Chapter 3

Design

This chapter will discuss some of the major design decisions from the start, during the course and acknowledge challenges and how they were dealt with.

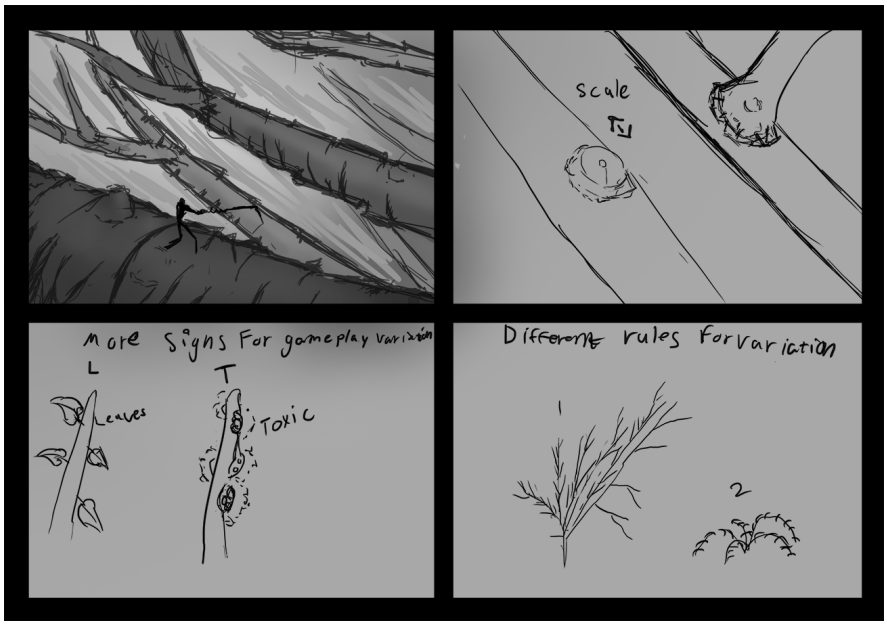

Figure 3.1: First concept of the project.

3.1 First Concept

To help understanding what the goal for the project was, concept art in the form of four thumbnails was drawn to illustrate how an L-system could be used in the context of a game. The thumbnails are in Figure 3.1, starting from the left top illustrate the idea of a player in a world generated by L-systems in form a of an enormous tree that the player navigates through. Thumbnail right top illustrate a a solution to how the growth will feel interactive and immerse by having an

animation that start as a very small part of the branch extruding in an interesting fashion into its full grown form. Bottom left is the key idea to create additional variables for gameplay purposes, it illustrate a safe branch with leaves and a dangerous infected branch containing toxic substances. The last thumbnail to the bottom right illustrate variation in game levels by using different rules.

Figure 3.2: Shows the first problem with scaling happening with cubes.

Figure 3.3: Illustrates the scaling problem in the later stages when a proper L-system was in place.

Figure 3.4: Shows problem with additional branches.

3.2 Challenges

Prefabs will be mentioned in this chapter and further below from here on. A prefab is a type of asset that functions as a reusable game object, all prefabs in the scene are clones of the original prefab.

3.2.1 Cloning

There were two major problems occurring during the project, one was visually obvious and the other missed in plain sight. The obvious one was a scaling issue where some of the branches would get very odd scaling numbers, examples in Figure 3.2 and Figure 3.3. The second issue was not noticed until later because it followed the structure of the L-system was additional branches as shown in Figure 3.4 . Both of these problems occurred for the same reason. When a prefab in Unity has a script that creates an instance of the same prefab, the copy will also be running the script on the same part of the script as the prefab creator. The prefab should only make a copy of itself, but by also copying the running script in action it will when cloning three times first create one, then the original and that one will create one each, and then those three will create one each, which will result in six prefabs instead of three. To solve the problem, an in-between

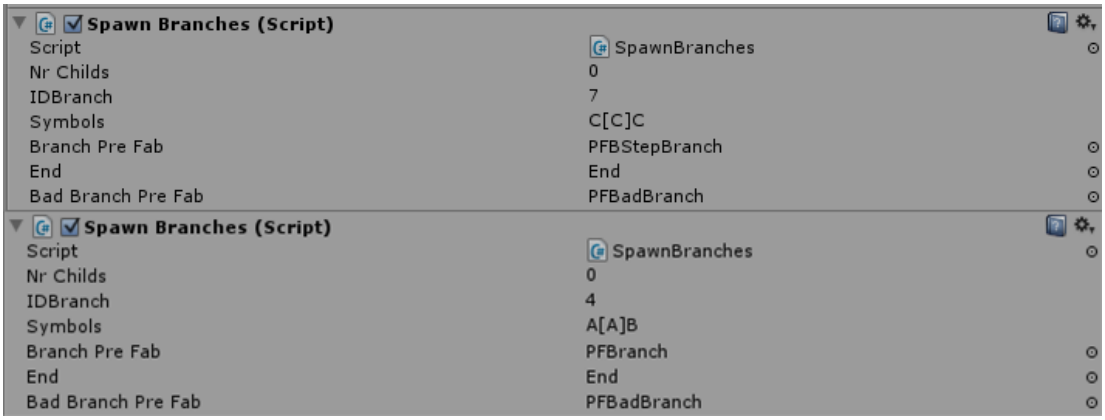

Figure 3.5: This shows two components of spawnBranches script. The top component is for the PFBranch and the bottom is for the PFStepBranch.

prefab was inserted, because the parent would no longer create an instance of itself and instead the other prefab which is controlled. The in-between branch prefab is an exact copy of the branch prefab, a branch creates in-between branches while an in-between branch creates branches so that they never create a version of themselves. Figure 3.5 shows the component interface where a normal branch prefab has an "in-between branch" as input and the "in-between branch" has the normal branch as input to create new branches with.

3.2.2 Transformation

One of the easier design decisions was to decide how the next coming branch should spawn, preferably the next branch could start where the other branch ends. The harder and more time consuming part would be to use the position, length and rotation of the parent branch to decide where the branch end, this would also not give a simple all around solution if you want to work with different kind of parts. The solution for this project was to let the branch prefab have an empty(end) where the branch or trunk ends. The end object will be transformed by the branch prefabs transformation and give the correct coordination of end of the branch.

3.2.3 Diversity vs Control

Another challenge was to decide between a controlled L-system or a semi "random" L-system that loses some control but keeps the idea of a tree and adds more variation. Control would result in the same kind of tree growing but a more accurate representation of a "correct" tree structure, while "random" could mean a merge of different growth structures giving an impression of a tree rather

than an accurate representation. Considering that this thesis is about procedural generation targeted towards the game industry, it is better to go for less control because it will create more scenarios with less effort. When creating a branch it inherits the vector of the direction its parent have and additionally adds rotation with the vector (Random.value*100-50,0,Random.value*100-50)". These random values for x and z will be between -50 to 50 and is inspired by Zachernuk's L-system implementation in Unity [12]. The positive thing about this value scale is that it generates arbitrarily similar vector directions as the Bracketed OL-systems examples in The Algorithmic Beauty of Plants [8]. This also gives random chance for a branch to turn left, right or any other direction.

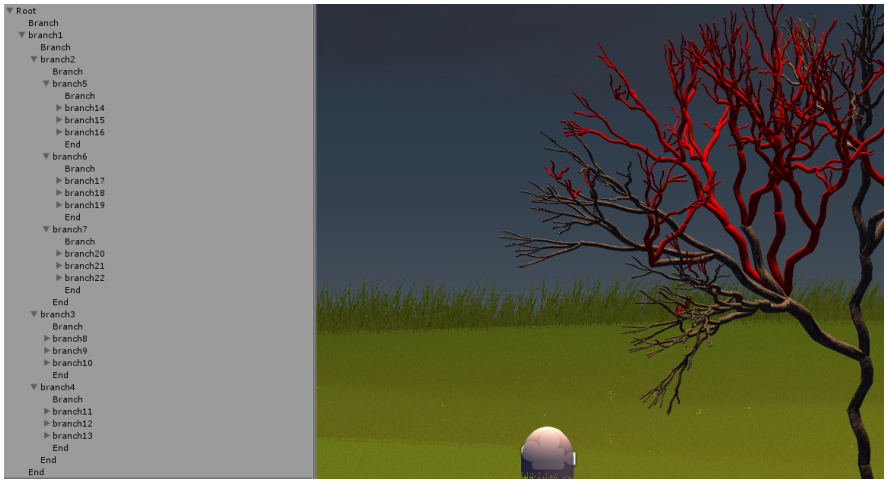

Figure 3.6: Shows a chunk of the hierarchy structure from the generated tree

3.3 System

The system is built upon a key concept of independence where each branch has its own logic. The reason for independence is to support future changes like additional variables and branches, or if a branch falls off it can continue to grow on its own. This project is built on the idea of turtle graphics, where the branch dependent on the former branch's position, though it does not only use the translation and rotation but also the scale. Knowing the kind and position of a branch can be important if one wants to use it for the progress in the game or change the visuals of the branch into for example, a trunk if it has no parents or a branch with leaves if it has no children.
That is the reason why a branch created becomes a child of the branch that created it, example of the structure shown in Figure 3.6 . The systems recursions are controlled by the player and for every click a new recursion is created.

Chapter 4

Implementation

4.1 Development Tools

This thesis project uses one primary application, the game engine Unity 3D and four applications mainly for aesthetics, which is Sculptris, Blender, xNormal and Photoshop. This project uses the free version of Unity, and Unity version 4.1.2. Windows will be used as the platform and C♯ as the main language. Blender is the program to create the lowpoly with its UV layout and to handle all the 3D models created before sending it into Unity. Blenders ".blend" files will be the import file format for the Unity 3D assets. xNormal version 3.17.14.38247 and will use the output maps normal, occlusion, cavity and curvature to create aesthetic pleasing textures for the game models. Using Photoshop CS6 with the xNormal maps to create the final textures for the game models. Sculptris is used to sculpt the highpoly branches and trunk for this thesis, using Sculptris version Alpha 6.

4.2 First steps

The first step was to make something that could duplicate itself like a cell. Using instancing in a script on a prefab cube accomplices just that. The recursion control was given to the player so that each time left mouse button was clicked it would trigger a new recursion. The next step was to build some kind of structure out of the cubes that spawned, adding an offset by its length to create a pillar or what could later be a longer branch. Adding rotation and scaling would need a more sophisticated system or another solution than offset, so an end point was added at the end of the cube to allow for a simple solution of position placement after transformations on the parent. Once this was implemented it was tested by adding rotation and forming a large spiral in Figure 4.1 .

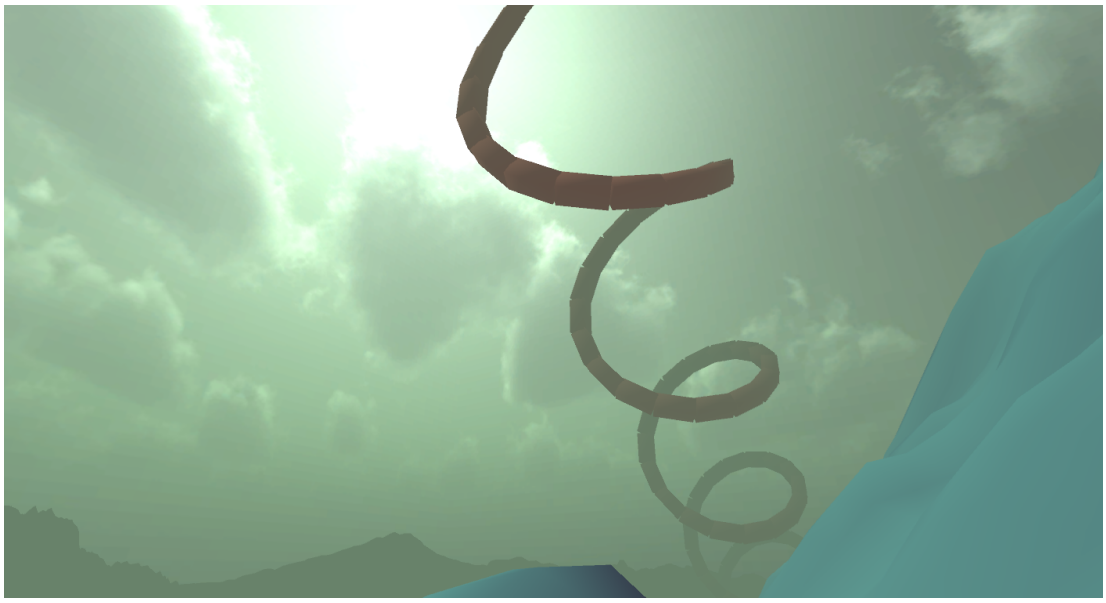

Figure 4.1: Simple cubes generate domino spiral effect

4.3 L-system

The second step was to implement some sort of L-system to get started, the algae was simple enough to begin with. The basics of this L-system was constructed with nested if statements, symbols where introduced and a cube or branch now holds grammar for the branches that will sprung out from it.
Hierarchy is also added now to keep count on order and for future advances like simple swaying animations. Rotation is added to actually see the structure algae creates in Figure 4.2. Figure 4.3 shows the replacement of cubes to branch geometry and a new set of rules that create a more tree like structure. The new L-system is a simple branching pattern, following the rules of A→A[A[B and B→A[B. "[" means the symbol before it will turn in a vector direction of (random*100-50, 0, random*100-50) which can create a vector in the value ranges of (-50 to 50, 0, -50 to 50) it will otherwise follow the form branch's direction, "B" and "A" represent a branch growing out from the branch with the symbol. Figure 4.3 unfortunately also have the cloning problem, discussed in chapter 4.2.1, this is corrected in Figure 4.4, the scaling and amount of branches spawning now work as it should.

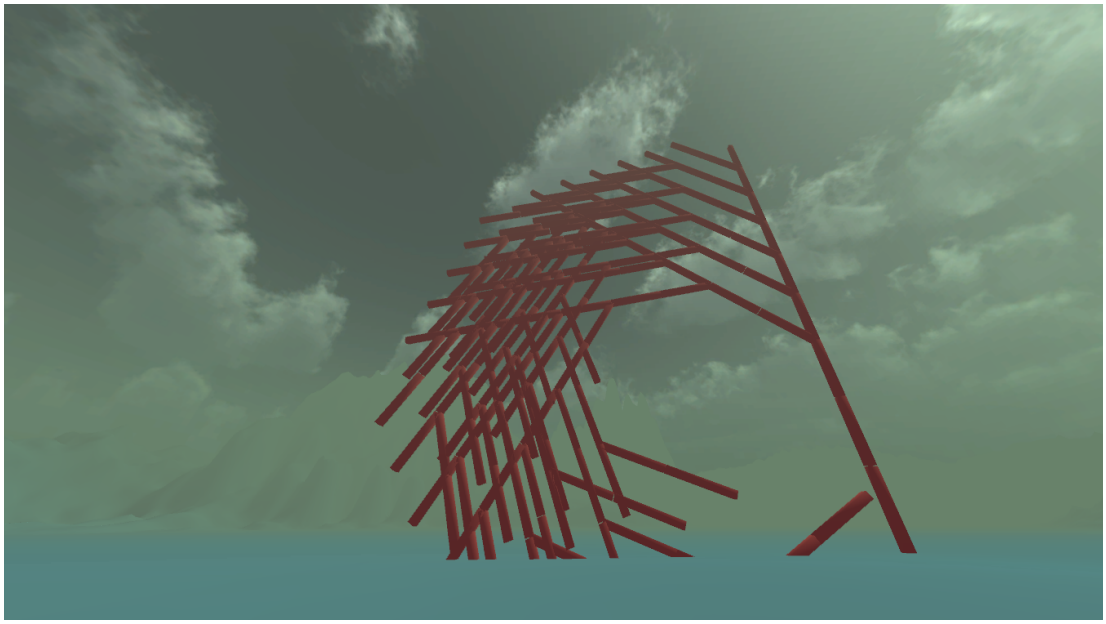

Figure 4.2: Cubes following the algae L-system with a constant rotation factor

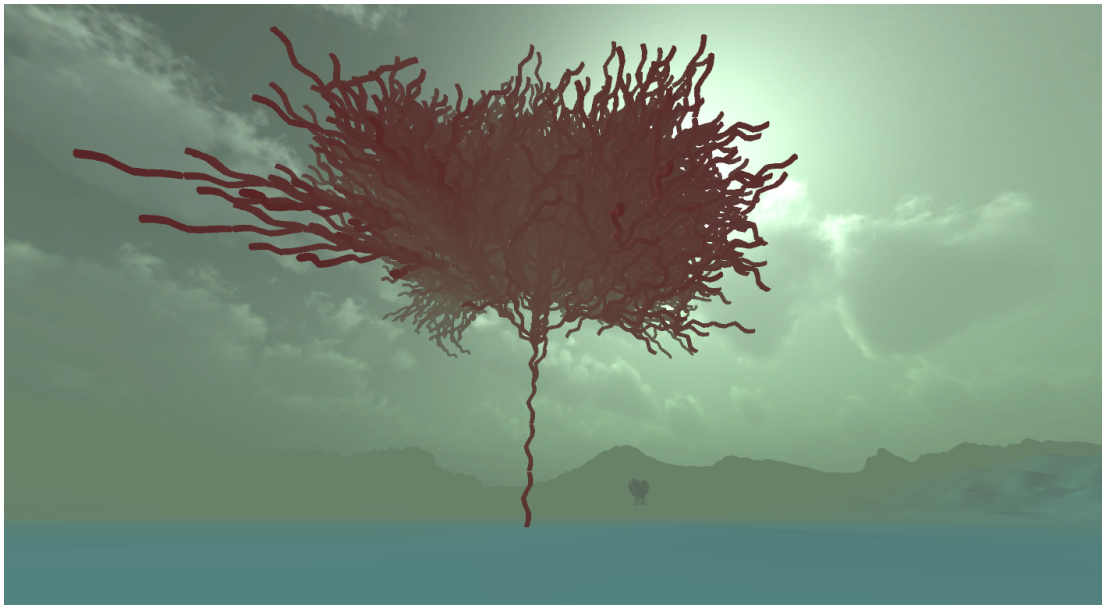

Figure 4.3: A generated tree with no scaling and too many branches

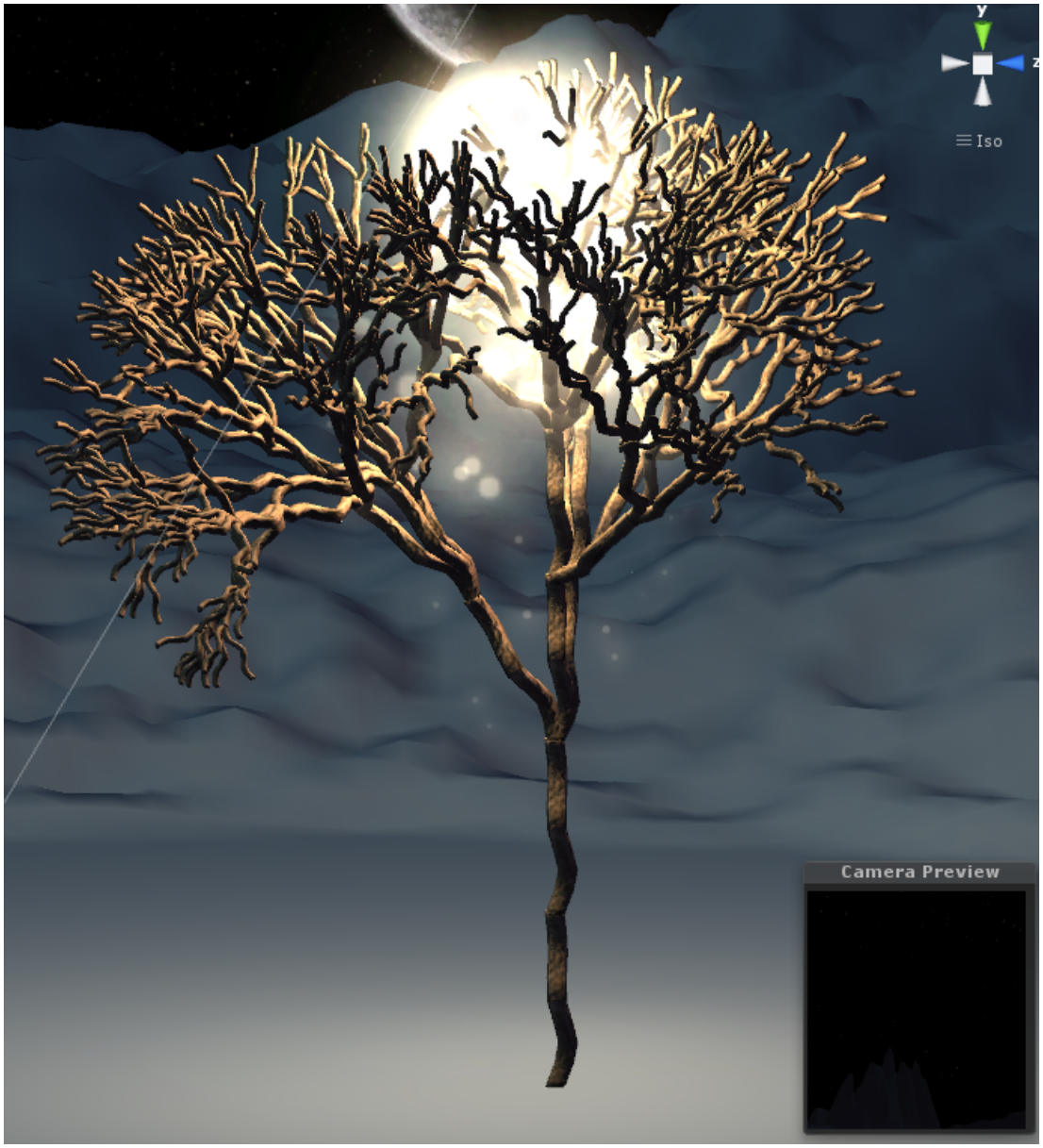

Figure 4.4: A tree generated with correct scaling and number of branches

4.4 Gameplay L-system

At this point, two gameplay features were implemented and presently previously, player control over the recursions and individual growth for branches that are cut off from a tree.
The third gameplay feature is adding possibility to spawn other kinds of prefab branches for special gameplay purposes. These special branches could also use their own rules for further growth. There are many kinds of special branches that could enrich the gameplay experience, poison branch could damage the player, jump branch could give extra force to next jump. This thesis focuses on the system and implements one, the poison branch. To begin with the system needs to handle extra variables and add a new set of rules. Poison branches will be represented by the symbol "C" but follow the same rules as "A" and "B", the "C" have a chance of one in ten to be written by "A" or "B" branch. In Figure 4.5 there is an example the new tree. The red colored branches are poisonous. Using L-systems to generate and model up trees is a matter of aesthetics and because of that there is a need for proper representation of the branches. Using Blender, Sculptris, xNormal and Photoshop the following game models are made a branch, poisonous branch and a trunk seen in Figure 4.6 and in action in Figure 4.7. In Figure 4.8 a particle system is added to the poisonous branch to enhance the idea of danger and poisonous mechanics. To measure the execution time of each recursion, the stopwatch of .NET framework was used. Number of branches is stored in the root or axiom branch and the rest of the data was collected from the in-game statistics window.

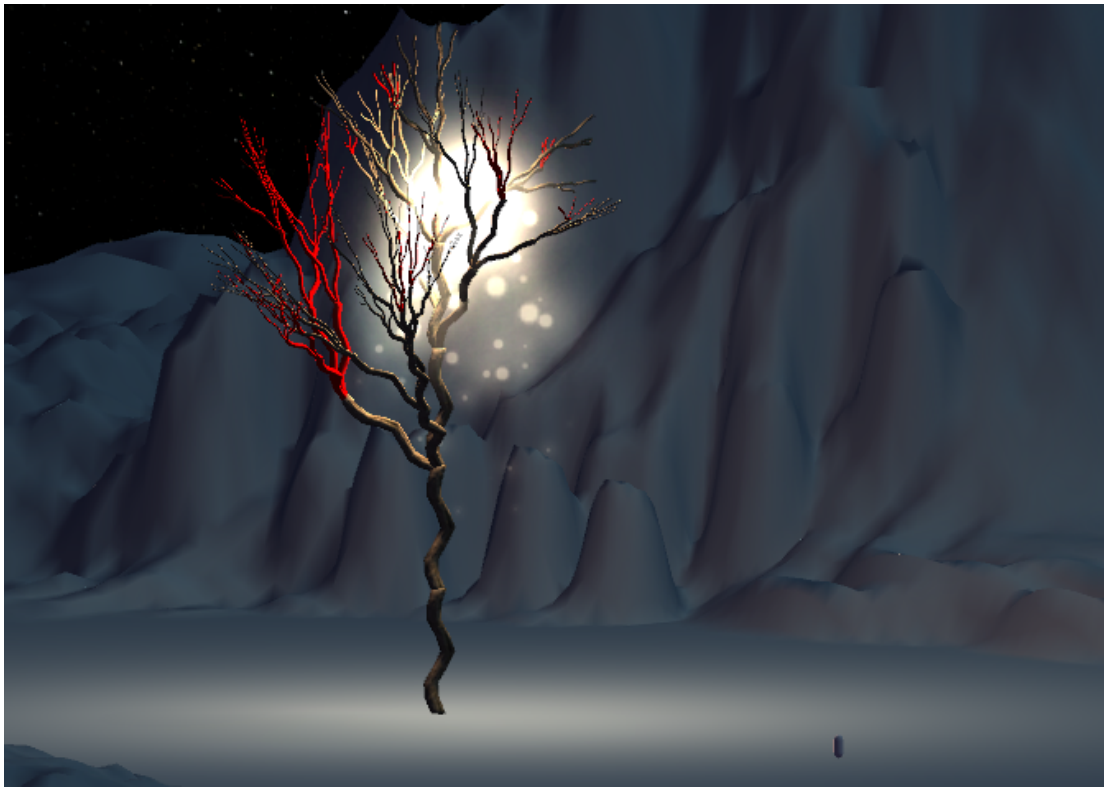

Figure 4.5: Shows the implementation of poison branches

Figure 4.6: The game models to make a tree, the trunk, branch and the poison branch

Figure 4.7: Shows the new branches and the trunk before modifying the xNormal bakes in Photoshop

Figure 4.8: Shows the textured game models and the poison particles

Chapter 5

Result

The result of this thesis is in the form of an L-system in the Unity game engine, the L-system is created with gameplay in consideration.

The non-contextual L-system consist of the following rules, variables and axiom in Table 5.1. Table 5.2 holds data related to the recursions of a tree generated with these rules. "Number of branches" stands for the total number of branches in the scene, "Triangles" is the number of mesh triangles in the scene, "Time" is the amount of milliseconds for recursion to execute on, "FPS" stands for frames per second and is the average fps in the scene.

In context the variables "A" and "B" create the prefabs normal branch, "C" creates the "special", or in this case "bad" branch of the prefab, "[" makes the former symbols object rotate with the vector ((random*100-50, 0, random*100-50)). A prefab branch has a visual representation, an end point as well as a "SpawnBranches.cs" script, seen dissected in Figure 5.1. A branch prefab does not have to be a normal branch and can be a special branch depending on what it contains and the inputs in the "SpawnBranches.cs" script.

The L-system is independent, which means that each part, if cut off from the hierarchy can grow on its own, the structure and system is similar to how armature structures in games are, so the child inherit all the parents transformations, such as scaling and rotation. The assets in Figure 4.6 were made for this project to create a clear visual difference between the poisonous and normal branches. Figure 5.2 shows the resulting trees from the L-system, and Figure 5.3 shows a player climbing among the tree tops.

Variables	A B C [
Axiom	A
Rules	(A→A[A[B), (B→A[B), (A or B have a 10% chance → C[C[C)

Table 5.1: Rules, axiom and variables from this thesis projects L-system.

Recursion	Number of branches	Triangles	Time	FPS
0	1	400	0 ms	2900
1	2	800	2 ms	2900
2	5	2100	5 ms	2900
3	13	8300	45 ms	2100
4	32	22200	81 ms	1600
5	78	52000	182 ms	900
6	193	137900	498 ms	550

Table 5.2: Iteration data from this thesis projects L-system.

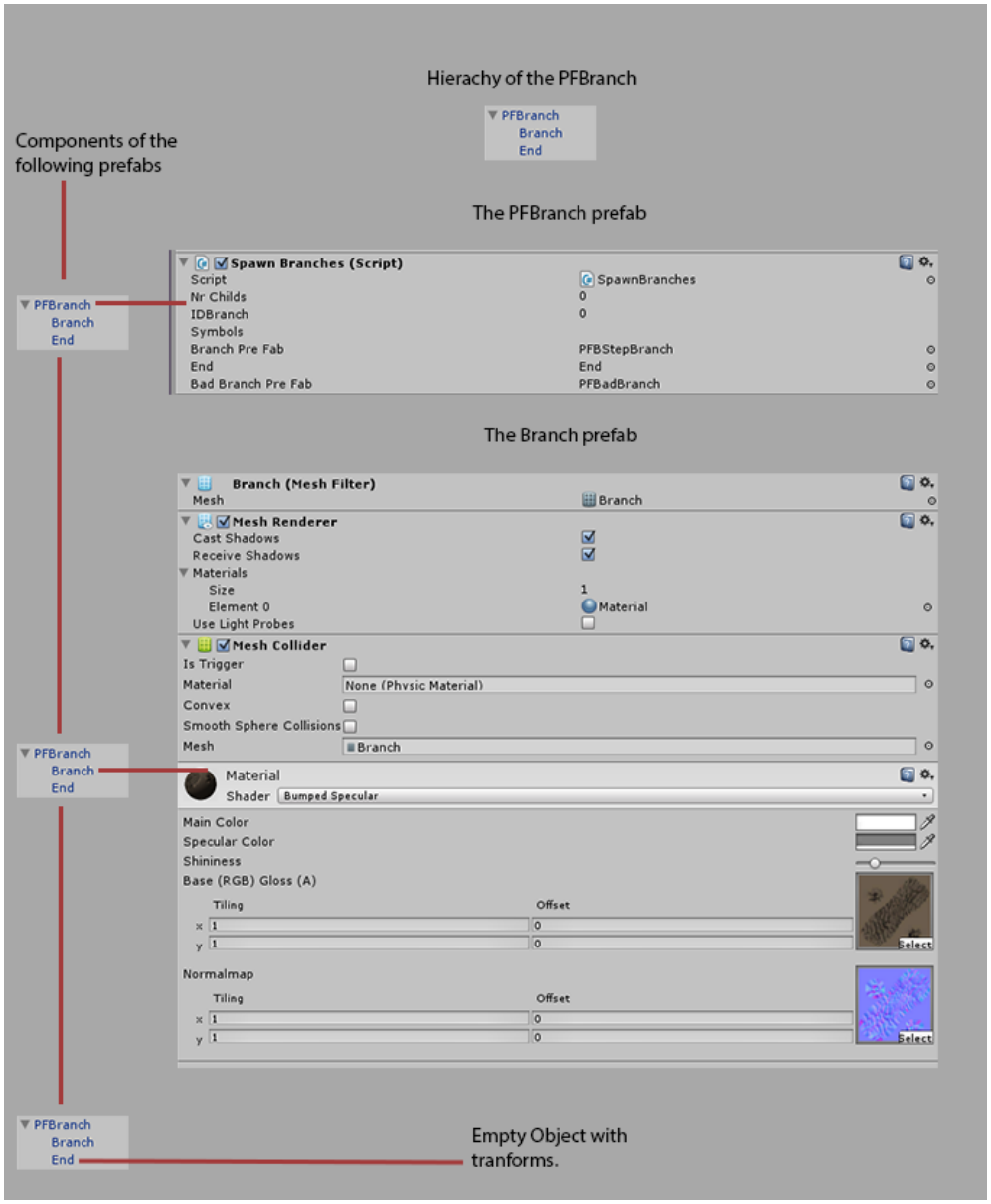

Figure 5.1: A dissected branch prefab.

Figure 5.2: Generated tree with poison.

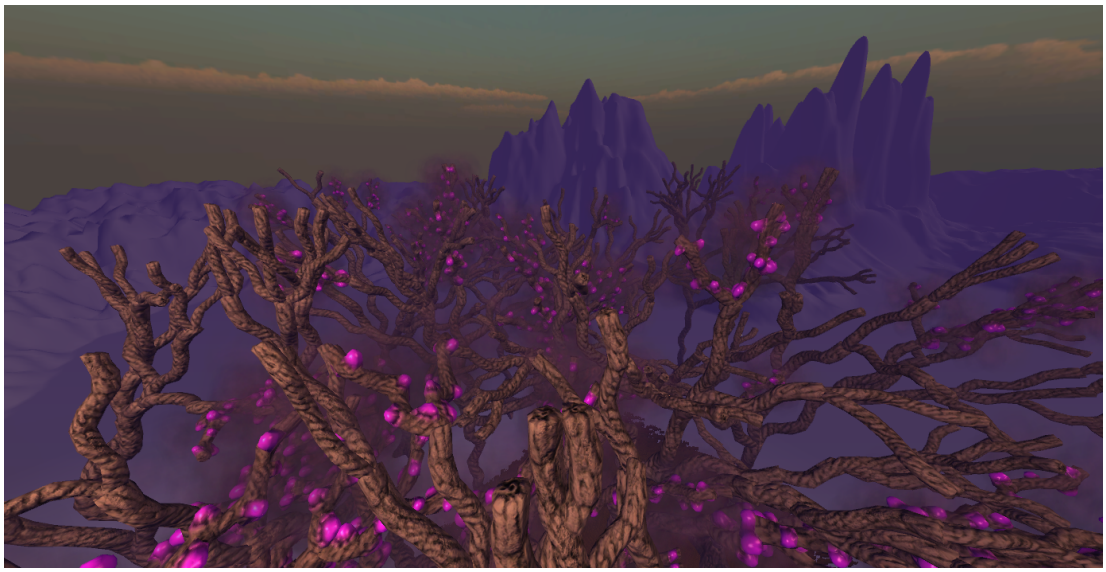

Figure 5.3: A players view among the tree tops.

Chapter 6

Conclusion

The research question have been answered with the implemented gameplay mechanics stated in Section 6.1 as examples of how gameplay mechanics can be integrated with L-systems.

6.1 Gameplay Mechanics

Recursion control enables the player to shape the game world by controlling the growth of the L-system, for example this can allow the player to build structures that will help the player avoid or climb above obstacles.

Different kind of branches in the prefab allows for adding additional branches easily by just creating the new prefab and insert it into the branch that will spawn it. Examples of gameplay mechanic abilities a branch could have are, bouncing branches that either move around a lot or gives the player extra jump height, the implemented special branch in the project was the poison branch that generated dangerous gasses and hurt the player when up close.

Independence for branches mean that it is its own controller, this allows for gameplay mechanics such as branches that fall off can continue to grow on its own. Other uses could be that the player can use branches as seeds and strategically place the spots for the trees to grow.

This thesis project have created an L-system in a game engine, with integrated game mechanics such as recursion control, different kinds of branches and independence for each branch.

6.2 Discussion and Reflection

The methodology was made with consideration of the first concept in Figure 3.1, and ended up using the free version of the Unity game engine using the C♯ language. Unity was very easy to work with considering not having coded with it before which supports the idea of using Unity instead of UDK that is known for being more complex and harsher to beginners. The methodology, while following the ideas of the first concept, it was also a trail and error process to evaluate the current methodology and pick the right direction for the design of the L-system implementation.

The first implementation was to bind the iteration control of the system to the players right mouse button because it was the simplest thing to do and helped gaining debug control. Each branch had its own logic to approach L-systems with the original thought of cell division, while making it easier to program, it might have been better to pursue the standard practice of having one global controller. A controller would ease the process of a much needed implementation of culling optimization and a queue order for creating new branches once the iterations became larger to avoid "freeze lag". Once the L-system was in place, support for different kinds of branches was added and due to the fact they controlled themselves they could also hold their own rules and variables, it was easy adding new branches. The second branch that was implemented was the poison branch due to being a simple implementation. Any other kind of branch, or more branches could have been implemented, but was not because of time constraints.

Looking back, what could have been done better generally is to have a more specified research question to begin with. Testing the gameplay mechanics in a real gameplay scenarios would have been interesting as well.

6.3 Future Work

As for future work one can work on optimisation for extremely large tree worlds that can grow. Optimisations such as specialised culling techniques for L-systems in games could be used for large tree worlds.

One can take it further by focusing on rules that focuses on gameplay and builds interesting shapes for the player. Another approach is making real time plant or tree growth in games seamless by using morph and skeletal animations. There are also other options for L-system and games, one can create an L-system for tunnel or corridor world generators that could have rules that optimize design patterns to enhance the player experience with procedural content.

References

[1] Bay12Games. Dwarf fortress homepage. `http://www.bay12games.com/dwarves/`. Accessed May 13, 2013.

[2] David S Ebert. *Texturing and modeling: a procedural approach*. Morgan Kaufmann, 2003.

[3] Stefan Enberg. *Kontextberoende generering av träd för dataspel*. PhD thesis, University of Skövde, 2007.

[4] Stefan Greuter, Jeremy Parker, Nigel Stewart, and Geoff Leach. Undiscovered worlds–towards a framework for real-time procedural world generation. In *Fifth International Digital Arts and Culture Conference, Melbourne, Australia*, 2003.

[5] Mark Hendrikx, Sebastiaan Meijer, Joeri Van Der Velden, and Alexandru Iosup. Procedural content generation for games: a survey. 2012.

[6] Steven Longay, Adam Runions, Frédéric Boudon, and Przemyslaw Prusinkiewicz. Treesketch: interactive procedural modeling of trees on a tablet. In *Proceedings of the international symposium on sketch-based interfaces and modeling*, pages 107–120. Eurographics Association, 2012.

[7] Mojang. Mojang creators of minecraft homepage. `http://mojang.com/`. Accessed May 13, 2013.

[8] Przemyslaw Prusinkiewicz, Aristid Lindenmayer, James S Hanan, F David Fracchia, Deborah R Fowler, Martin JM de Boer, and Lynn Mercer. *The algorithmic beauty of plants*, volume 2. Springer-Verlag New York, 1990.

[9] RE-LOGIC. Terraria homepage. `http://www.terraria.org/about.html/`. Accessed May 13, 2013.

[10] Miguel Sicart. Defining game mechanics. *Game Studies*, 8(2), 2008.

[11] Unity Technologies. Fast facts. `http://unity3d.com/company/public-relations/`. Accessed May 13, 2013.

[12] Brandel Zachernuk. L-system in unity3d, with source. `http://www.zachernuk.com/2010/12/30/l-system-in-unity3d-with-source/`. Accessed May 13, 2013.