**Project Plan**

DV1478 Kandidatarbete i datavetenskap

2017-04-30

|  |  |  |
| --- | --- | --- |
| Thesis | Tentative title | Procedural city generation using Perlin noise |
| Classification | Theory of computation, Randomness, geometry and discrete structures, Computational geometry |
| Student 1 | Name | Elias Frank |
| Email | elias.frank@gmail.com |
| Social security number | 19940327–4179 |
| Student 2 | Name | Niclas Olsson |
| Email | ohlsson.niclas@gmail.com |
| Social security number | 19930421–3797 |
| Supervisor | Name | Hans Tap |
| Email | hans.tap@bth.se |

**1 Introduction**

Exploring a huge open world environment is a desirable feature in games. But creating a big open city such as in the *Grand Theft Auto*[1] series and *Batman: Arkham City*[2] involves years of work for a lot of people. Making big open cities in games is simply not feasible for most game companies. These games all have massive success with their big open worlds making a feasible generated city an attractive technique for smaller companies to be able to compete with big open worlds of their own.

To create massive amounts of content without a big workforce algorithmically based solutions exist, *Procedural Content Generation* (PCG). PCG was in the past used to minimize the disk space required for games. *.kkrieger*[3] is an excellent example of this. It has since evolved into a method to minimize workforce required for content. *No man’s sky*[4] is an excellent example of a game using PCG to minimize workforce while maximizing content.

There are many techniques to achieve PCG such as ray marching, squarified treemaps[5], hashing[6], Perlin noise [7], fractals, L-systems[8][9], Shape grammar[10] and rule based subdivision[8] among others. Based on our time constraints, we chose Perlin noise as our technique for procedural generation. In relation to the other techniques mentioned, Perlin noise is easy and fast to implement. In our application, we have decided to use Perlin noise in a hierarchical, top-down, fashion. Each step of the generation process uses the result of the previous step as a starting point. This hierarchical process is described under *Method*.

**2 Aim and objectives**

This work aims to explore the possibility of using Perlin noise to procedurally generate a city that can be used in games. By conducting a user study, we will investigate whether the resulting city is considered believable in a game setting.

We aim to achieve the following objectives:

* Implement a method for procedurally generating cities using Perlin noise.
* Implement a user interface that allows the user to change the way the city is generated.
* Test whether the generated cities are believable.
* Use the test results to answer the research question.

**3 Research question**  
Can Perlin noise be used in a hierarchical manner to procedurally generate a city viable in games?

**4 Method**

The method used to answer the research question is an implementation that can procedurally generate a city using Perlin noise in a hierarchical manner. The city will be visualized in 3D, on a flat terrain and will be the shape of a square, like a cross section of a bigger city. To examine the viability of the cities generated, a user study will be conducted.

## 4.1 Implementation plan

To generate a city viable to use in games, three different generation stages have been recognized: *Districts*, *roads* (creating blocks) and *buildings*. We start by generating the districts based on user input. Then the roads and blocks are generated, taking the districts and its parameters into consideration. Finally, the buildings are generated, using information from the district and block that it belongs to. This is what we refer to as Hierarchical PCG. Four different variables will be controlled by the user:

* Seed
* Building minimum height
* Building maximum height
* Size of the blocks (this naturally also effects the number of roads)

The three last variables are connected to specific districts. This means that three districts, with three variables each, the user can control nine different variables controlling the city generation plus the seed. The seed is a string of characters. The seed will change how the city is generated. This is needed so the user can generate several cities with the same district parameters but different seeds. The seed also makes the generation deterministic meaning that the exact same city can be generated if all the parameters are entered.

## 4.2 Constraints

* Only three different districts will be supported
* The user interface is restricted to four different parameters
* The city edges will not be realistic (i.e. no smaller roads or villages at the edge of the city)
* Do not implement optimizing techniques for the rendering pipeline.
* Limit content generation to the city (i.e. no terrain generation etc.).
* Do not implement property generation (i.e. no cars or street signs etc.)

## 4.3 Experiment design

To start with, each participant will be shown a document that details the experiment. The document will explain how the participants are supposed to complete the experiment. Several images displaying different cities generated through the implementation will be shown. A questionnaire will ask relevant questions about the cities displayed. Examples: *Which of the cities looks like a city that could be used in a game? Which city contain the biggest factory district?*  The participants will answer these questions with a number representing one or more of the images.

## 4.4 Viability

During generation, each part of the city will follow certain guidelines for what is viable. This should ensure that whatever input the user provides, a viable city is generated. These are just guidelines; it is ultimately up to the participants of the user study to determine if the result is believable and viable in games.

* Districts must be a certain distance from each other and from the edges of the city.
* Roads must be far enough apart that buildings can fit between them.
* Buildings must have no more than one identical neighboring building.

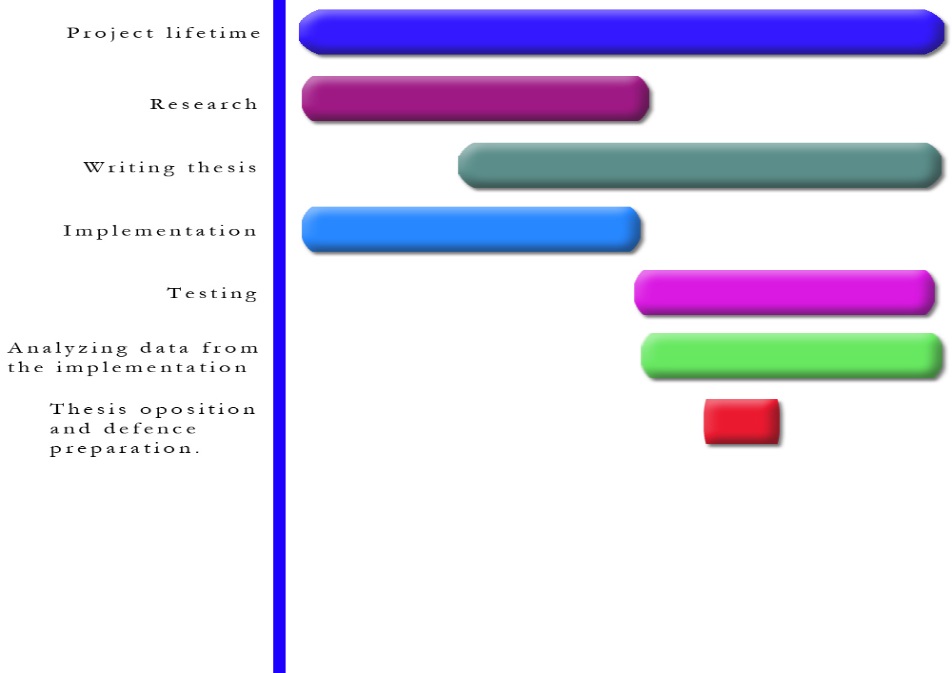
## 4.5 Data Collection

The main way of collecting data is through a user study where participants will fill out a questionnaire. From the result of this user study, we will be able to answer the research question. Apart from the user study, we will keep track of performance information such as loading times, RAM and VRAM.

**5 Expected outcomes**

With Perlin noise many cities will be generated. With the user study, we will gather data through a questionnaire that will show that the users find some of the cities viable to use in games.

**6 Time and activity plan**

  
*Figure 3: Visual representation of activity plan*

The implementation and research will be the first things to be done and they will be done simultaneously. Some parts of the implementation require research before we have the knowledge to implement it, but all the parts of the implementation, such as the rendering pipeline, that do not require any research, will be done in conjunction with the research.

The writing of the thesis will begin shortly after we have some basic research and base implementation done. The thesis will be worked on every week until completion. When the implementation has enough features, data gathering and analysis of this data will begin. This data is crucial to answer the research question and conclude the thesis.

* Project lifetime 2017-03-27 – 2017-06-11
* Research 2017-03-27 – 2017-05-14
* Writing thesis 2017-05-01 – 2017-06-11
* Implementation 2017-03-27 – 2017-05-14
* Testing 2017-05-14 – 2017-06-11
* Analyzing data 2017-05-14 – 2017-06-11
* Thesis opposition and defense 2017-05-22 – 2017-05-28

**7 Risk management**

|  |  |  |  |
| --- | --- | --- | --- |
| **Risk** | **Probability (1-5)** | **Severity (1-5)** | **Mitigation** |
| Procedurally generating a city is difficult and consumes too much time. | 3 | 5 | Only implement the most crucial features to answer the research question. |
| Failing to communicate with thesis partner. | 2 | 4 | Have daily meetings and work together when possible. |
| Technical issues | 2 | 5 | Make sure all work is on several hard drives. Use git as source control. |
| Collect data from the implementation wrong, twisting the results. | 3 | 4 | Carefully decide what parameters in the implementation to collect data from and in what form to collect this data. |
| Defective construction of the implementation. | 2 | 4 | Before coding the implementation make sure there is a good plan to follow. |

# **References**

[1] Rockstar Games. (1997-2013). Game. *Grand Theft Auto Series.*

[2] Rocksteady Studios. (2011). Game. *Batman: Arkham City.* Warner Bros, Square Enix.

[3] Farbrausch. (2004). Game. *.kkrieger.*

[4] Hello Games. (2016). Game. *No man’s sky .* Hello Games, Sony.

[5] Fernando Marson, S. M. (2010). *Automatic Real-Time Generation of Floor Plans Based on Squarified Treemaps Algorithm.* Brazil: International Journal of Computer Games Technology.

[6] Stefan Greuter, J. p. (2003). *Real-time procedural generation of `pseudo infinite' cities.* Australia: Graphite.

[7] Ken Perlin (2002), *Improving noise. SIGGRAPH 02*

[8] Muller, P. (2006). *Procedural Modeling of Cities Part VI.* Boston, Massachusetts: ACM SIGGRAPH.

[9] Müller, P. (2001). *Procedural modeling of cities.* New York: SIGGRAPH '01.

[10] *Advanced procedural modeling of architecture*, Pascal Müller, Michael Schwarz, ACM