Southampton Solent University

SCHOOL OF MEDIA, ARTS AND TECHNOLOGY

**BSc (Hons) Computer Games (Software Development)**

**2019**

**Flavio Fiori**

**“Procedural Terrain Generation”**

***“The title of your dissertation”***

Supervisor : Philip Alassad

Date of submission : Working on Progress

Contents

[1. Progress Report 4](#_Toc2325341)

[1.1 Overview 4](#_Toc2325342)

[1.1.1 Project 4](#_Toc2325343)

[1.1.2 Why I want to undertake this project and how it is going to challenge me 4](#_Toc2325344)

[2.1.3 Aims 5](#_Toc2325345)

[2.1.4 Objectives 5](#_Toc2325346)

[2.1.5 Stretch Goals 6](#_Toc2325347)

[2.2 Project Specification 6](#_Toc2325348)

[2.2.1 Software 6](#_Toc2325349)

[2.3 Research 7](#_Toc2325350)

[2.3.1 Perlin Noise 7](#_Toc2325351)

[2.3.2 Diamond-Square Algorithm 11](#_Toc2325352)

[2.3.3 Simplex Noise 15](#_Toc2325353)

[2.3.4 Thermal erosion 15](#_Toc2325354)

[2.4 Prototyping 16](#_Toc2325355)

[2.5 Project Management 17](#_Toc2325356)

[2.5.1 Methodologies 17](#_Toc2325357)

[2.5.2 Tools for Time Management 20](#_Toc2325358)

[2.6 Design 22](#_Toc2325359)

[2.6.1 High Level Flow Chart 23](#_Toc2325360)

[2.6.2 Perlin Noise Class 24](#_Toc2325361)

[2.6.3 Mesh Generation Class 24](#_Toc2325362)

[2.6.4 Map Generation Class 25](#_Toc2325363)

[2.7 Project Plan 26](#_Toc2325364)

[2.7.1 Time estimation 26](#_Toc2325365)

[2.7.2 Feasibility and Risk 29](#_Toc2325366)

[2.7.3 Risk Analysis 30](#_Toc2325367)

[References 31](#_Toc2325368)

[Bibliography 33](#_Toc2325369)

[Appendix 36](#_Toc2325370)

[Appendix A) Reading List 36](#_Toc2325371)

[Appendix B) Literature review 37](#_Toc2325372)

[B.1) Terrain Synthesis Techniques Research 37](#_Toc2325373)

[B.2) Perlin Noise Research 38](#_Toc2325374)

[B.3) Diamond-Square Research 40](#_Toc2325375)

[Appendix C) Images 41](#_Toc2325376)

[C.1) Prototype using the 2D Perlin noise function that I created 41](#_Toc2325377)

[C.2) Prototype using the Unity Perlin Noise Function 41](#_Toc2325378)

[C.3) Perlin Noise from my function 42](#_Toc2325379)

[C.4) Perlin Noise from unity 43](#_Toc2325380)

[C.5) Ken Perlin’s original gradient function 43](#_Toc2325381)

[C.6) Riven Calculate Perlin Noise twice as fast 44](#_Toc2325382)

[C.7) Noise Displaced super quadric with old interpolants 44](#_Toc2325383)

[C.8) Noise Displaced super quadric with new interpolants 45](#_Toc2325384)

[C.9) Risk and Analysis 46](#_Toc2325385)

[C.10) Gantt Chart 47](#_Toc2325386)

# Progress Report

## Overview

### Project

The goal is to create a procedural generation of terrain, where the user can define the size of it. Randomly generate a smooth transition from mountains to valleys, simulating a terrain and applying textures in relation of the type of terrain. The end goal is to be able to create a plug-in for Unity that allows random terrain generation inside the editor, without the need for the user to design and generate terrain.

### Why I want to undertake this project and how it is going to challenge me

Terrain generation is something that I find fascinating. Due to this reason I find that undertaking this project not only is going to drastically improve my knowledge in Unity and C# but also is going to increase my job opportunities in the game industry. Programming a random terrain generation using Hight maps, is going to be one of the biggest challenges for me. Since I do not have an idea where to start. Long hours of study and programming are going to be required for me to complete this project.

### 2.1.3 Aims

The aim of this project is to create a procedural terrain generation.

This project will contain different researches into what are the most cost-efficient terrain synthesis techniques to use for procedural terrain generation.

When the terrain generation is successfully complete, a method for synthesis of eroded terrain is going to be implemented (Olsen 2004).

### 2.1.4 Objectives

* Research the best algorithms to create procedural terrain.
* Implement one of the three algorithms to create procedural generated terrain:
* Perlin Noise
* Diamond-Square
* Simplex Noise
* Create a 3D wireframe in unity using the algorithm selected
* Implement an algorithm for eroded terrain
  + Thermal erosion
* Implement the Diamond-Square algorithm to create the noise and generate the height maps using this algorithm.
* Implement the Simplex Noise algorithm to create the noise and generate the height maps using the algorithm.
* In unity editor the user can chose what algorithm use when generating the terrain.

### 2.1.5 Stretch Goals

* Create the procedural terrain generation plug-in for unity
* Implement the terrain synthesis techniques in C++ and OpenGL
* Create the Graphic class
* Create the Mesh Class
* Generate a 3D terrain using the terrain synthesis technique

## 2.2 Project Specification

### 2.2.1 Software

The required software needed is Visual Studio, Unity, Microsoft Excel, Microsoft Word, GitHub, Gantt Project, Visio, Photoshop, Kanbanflow, HacknPlan, and Google Drive.

* Visual studio is my favourite IDE for creating the scripts in C#.
* Unity is the 3D engine that will be responsible to handle and render the procedural terrain.
* Microsoft Excel is the program that will be responsible to create the work schedule for each week
* Microsoft Word allows me to create the documentation for the project.
* Github and Google Drive is where the backups of the project are stored.
* Gantt Project is the responsible program to create the Gantt chart for this project.
* Visio is where I am going to create the UML Class Diagrams for the project.
* Photoshop is used for the making of the terrain textures.

## Research

### 2.3.1 Perlin Noise

#### 2.3.1.1 What it is?

Perlin Noise is a type of gradient noise created by Ken Perlin in 1983. Noise can be used to create a wide variety of natural looking textures. When combining noise into mathematical expressions is possible to achieve procedural textures. The great thing about procedural textures is that they do not need a source texture image. They can be applied directly to 3D objects without the need to create the texture mapping responsible to wrap the texture around the object.

#### 2.3.1.2 How it works?

1. In the noise method parameter, an initial value is given

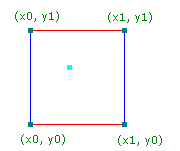


Figure 1 - Initial values the 4 grid points

1. The original algorithm created the pseudo-random gradient the following way: . Currently the most common uses are for n=1 (animations), n=2 (cheap texture hacks), n=3 (less-cheap texture hacks) (Perlin 2001).
2. For each coordinate:
   1. Generates a pseudo-random gradient vector
   2. This gradient vector defines a positive direction and a negative one.

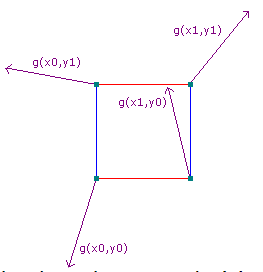


Figure 2 - Pseudorandom gradient

1. Linearly combine a weighted sum, using the function to ease the curve in each dimension . This function is applied to the U and V values. This make changes more gradual as one approaches integral coordinates.

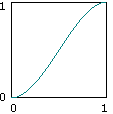


Figure 3 - Ease Curve

##### 2.3.1.2.1 The 4 neighbours in two dimensions

In two dimensions, there are 4 surrounding grid points. For each grid point, a vector going from the grid point to (x,y) is generated, this is easily obtained by subtracting the grid point from (x,y).

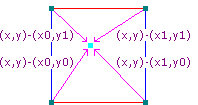


Figure 4 - The 4 neighbours

#### 2.3.1.3 Changes in the Perlin Noise Algorithm

During the research was discovered that the original Perlin Noise created in 1980 had two deficiencies in the algorithm. One of them was the gradient computation, to fix this problem Ken Perlin discovered that was not necessary the G equation to be random at all. In the journal Improving Noise the G function was replaced by 12 vectors defined by the directions from the centre of a cube to its edges instead of the original G equation that take each grid point and assigns a pseudorandom vector of length 1 in R^2 (Perlin 2002). The other deficiency was found in the function to ease the curve, see ([Appendix](#_Noise_Displaced_super) C.7) ([Appendix](#_Noise_Displaced_super_1) C.8)

“The above algorithm is very efficient but contains some deficiencies. One is in the cubic interpolant function's second derivative 6-12t, which is not zero at either t=0 or t=1. This nonzero value creates second order discontinuities across the coordinate-aligned faces of adjoining cubic cells.” (Perlin 2002)

### 2.3.2 Diamond-Square Algorithm

#### 2.3.2.1 What it is?

Diamond-Square algorithm is a method to create a fractal landscape. A fractal landscape is a simulation of a natural terrain appearance. This algorithm was introduced by Alain Fournier, Don Fussel and Loren Carpenter at the annual conference on computer graphics named SIGGRAPH (Fournier, Fussell and Carpenter 1982).

#### 2.3.2.2 How it works?

Diamond-Square algorithm works basically this way, first you need to create a 2D vertex field, where it is expressed in the following form: . This makes sure that for every set of vertices on the corner, there will always be a vertex on the centre (Olse 2004).

The algorithm is made by three steps:

1. Initialize Values
2. Perform the diamond step
3. Perform the Square step

##### 2.3.2.2.1 Initialize Values

The algorithm starts to initialize the array values for the initial four corner points within a specific range set by the user.

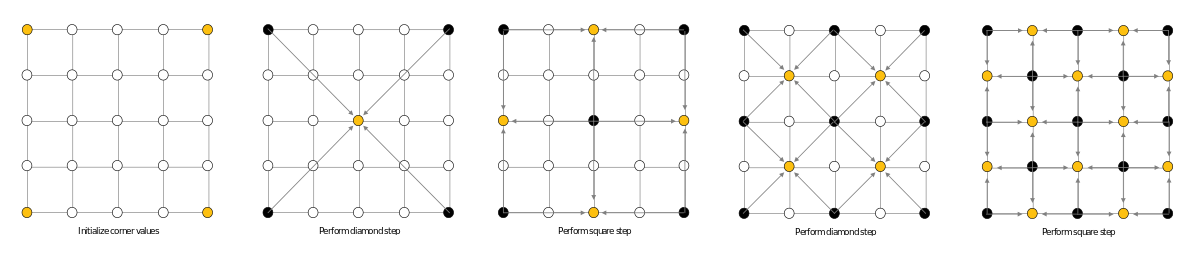


Figure 5 - Diamond-Square Initializing the corner values

##### 2.3.2.2.2 Diamond Step

The diamond step, consists of going through each square in the array, set a centre of that square to be the result of the average of the four corner points plus a random value.

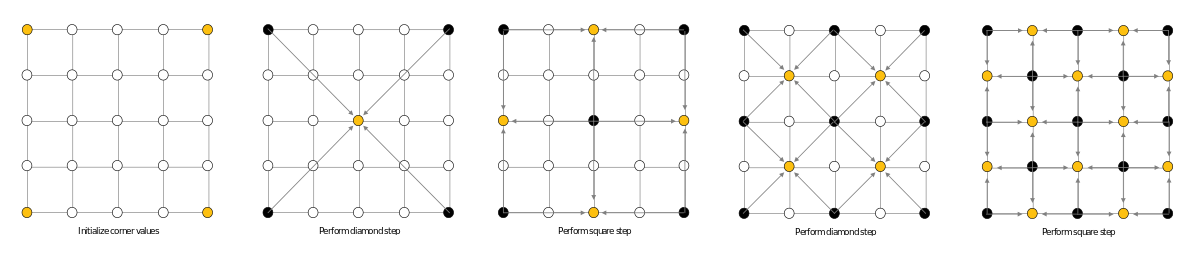


Figure 6 - Diamond-Square performing the diamond step

##### 2.3.2.2.3 Square Step

In the square step for each diamond in the array, sets the midpoint of that diamond to be the average of the four corners plus a random value. For each iteration the random value is reduced, until there are no more vertices.

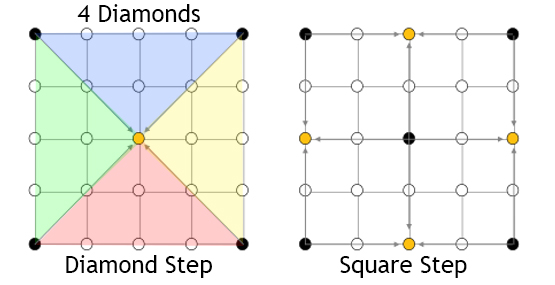


Figure 7 - Diamond-Square performing the square step

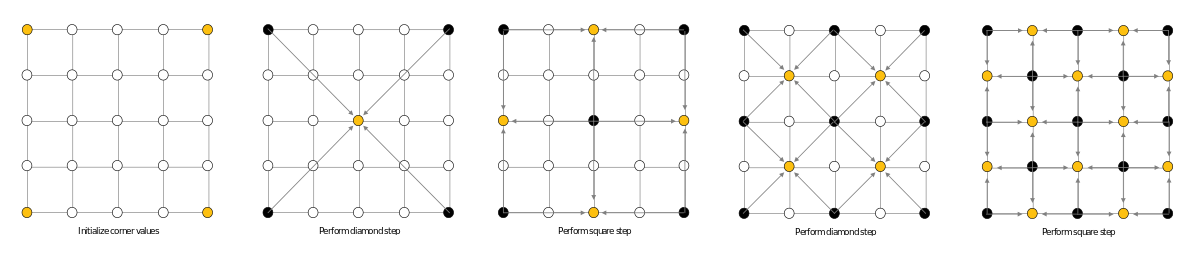


Figure 8 - Diamond-Square iteration

### 2.3.3 Simplex Noise

Simplex Noise is an algorithm created in 2001 by Ken Perlin to address the limitation of the Perlin Noise algorithm.

#### 2.3.3.1 Advantages compared to Perlin Noise

* Lower computational complexity, requiring a small number of multiplications, making it cheaper to compute than the Perlin Noise.
* Is visually isotropic, this means that the magnitude has the same value when measured in different directions.
* There are no more grid artefacts.
* Easier to implement in hardware.

### 2.3.4 Thermal erosion

Thermal Erosion is a process that simulates the breaking down of material, this breaking down then is moved by the wind or water to the bottom (Olse 2004; Rohde 2013).

#### 2.3.4.1 How it works?

An amount of the material that is at the top of a slope whose inclination is above a threshold value, will be moved down in the slope.

## 2.4 Prototyping

When developing a prototype of the procedural terrain, the premade Unity Perlin Noise function was used to generate a noise map and after converting it to a colour map ([Appendix](#_Prototype_using_the) C.2), this allowed to get an idea of the result of the prototype.

With the prototype in hand, now was time to implement the Perlin Noise function. This function was based on the java reference implementation of improved noise function with slight modifications.

When creating the noise function, few problems raised, one of them is the original Ken Perlin’s gradient function using complicated, and confusing bit-flipping code to calculate the dot product of a randomly selected vector and the 8 location vectors ([Appendix](#_Ken_Perlin’s_original) C.5) (Perlin 2002).

An easy alternate way of writing the code and even fast to compile in many languages, is to write a switch statement with all the 16 possible cases from the 8 location vectors ([Appendix](#_Riven_Calculate_Perlin) C.6) (Riven 2010).

Is possible to see the first result of the 2D Perlin noise function that was implemented in the ([Appendix](#_Prototype_using_the_1) C.1). While this function was far from perfect due to the fact that sometimes the output was negative, the result was satisfactory. To improve the function it was necessary to add the third axis (Z), to create a the correct output value ([Appendix C](#_Perlin_Noise_from).3) similar to the original Unity function ([Appendix](#_Perlin_Noise_from_1) C.4).

## 2.5 Project Management

### 2.5.1 Methodologies

#### 2.5.1.1 Waterfall

In the waterfall methodology the scope of the project is defined in the beginning and cannot be changed once development starts.

The advantages of using waterfall methodology is the fact that it is going to help organize the project with defined tasks. In the future, if all the steps are followed, a complete product is achieved. The waterfall model is simple and easy to understand, each phase of the model has specific deliverable dates and review processes, helping to create a linear structure for the project (ISTQB 2013).

##### 2.5.1.1.1 Pros

* Simple to Implement
* Easier to manage due to the fact that the stages are more rigid
* Is great for small projects

2.5.1.1.2 Cons

* Difficult to make changes once the project begins
* Software is created near the end of the development cycle
* The risks are high and unpredictable
* Does not work well for long projects

#### 2.5.1.2 Agile

Agile use sprints, each sprint is a time period that is allocated for each phase of the project. A sprint is complete when the time defined for it expires. If the development of the project is not satisfactory in that sprint, or the tasks were not met, then that sprint is considered a fail, sending all the failed tasks to the next sprint and so on.

##### 2.5.1.2.1 Pros

* Software is created in the beginning of the development cycle
* Decreases the risk associated with software development
* Requirements can change during the development of the project, implementing new features, this helps companies create the right product for the market

##### 2.5.1.2.2 Cons

* It is more focused on the developer instead of the client
* Easy to get lost if no clear focus is applied in the final outcome
* Less focus on the product design
* Ineffective in large organizations

#### 2.5.1.3 Time-Box

Time-box is a short period of time that person works towards a specific goal. If this goal is not reached when the time period ends, then the person stops working on the project and start evaluating what went well, what went wrong, has the goal been met, or to some extent met (Rothman 2017).

##### 2.5.1.3.1 Pros

Easy to implement

Easy to follow

Always updated with the project progress

Focus on the important things under time constraints

##### 2.5.1.3.2 Cons

Time consuming

Sometimes work cannot be completed on time, causing unfinished products.

Time estimation is not accurate

### 2.5.2 Tools for Time Management

#### 2.5.2.1 Trello

Trello is a project management application that uses the Kanban system. The projects are represented by boards, each of the boards contains a list of tasks. The tasks are represented by cards, that are created inside the boards.

“Trello is an online tool for managing projects and personal tasks. That may sound rather prosaic. But this increasingly popular app often inspires the sort of passion usually reserved for consumer apps like Pinterest or Instagram. It’s the kind of business software that slips into businesses through the backdoor, just because individual employees like how it works.” (Finley 2014)

#### 2.5.2.2 KanBanFlow

KanBan Flow is a cloud-based project management solution similar to trello, it uses the Kanban system but there is one thing that it excels compared to trello. KanbanFlow features the pomodoro technique for time tracking of tasks. This feature makes KanbanFlow my number one choice in time management tools (Software Advice 2017).

#### 2.5.2.3 HacknPlan

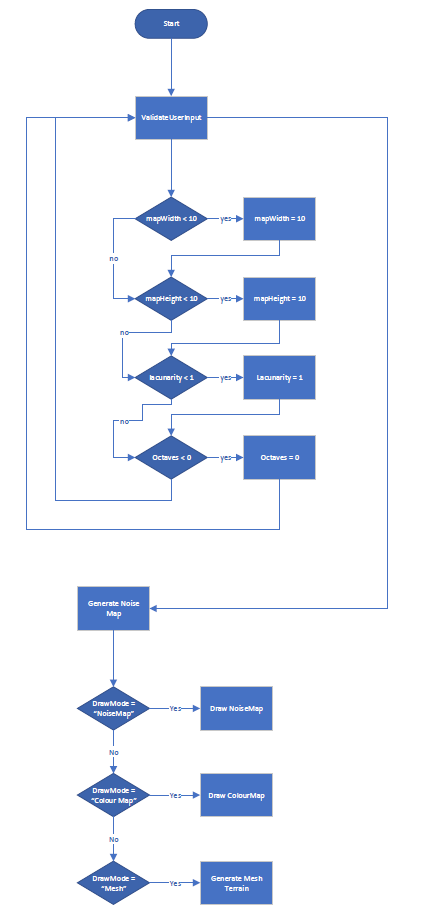
Basically, HacknPlan is an alternative of trello for game developers. The difference is that HacknPlan brings game design and task management together to provide a better organization while creating a game project. HacknPlan features stock headings with categories like programming, art, design, writing and more. Not only that but also each heading features different subcategories; In programming there is AI, Physics and more (Estevez 2019; GameDesigning 2018).

#### 2.5.2.4 Microsoft OneNote

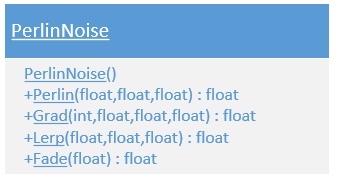
Microsoft OneNote is an app that simulates a notebook, were the user can gather handwritten or typed notes, drawings or even audio commentaries.

## 2.6 Design

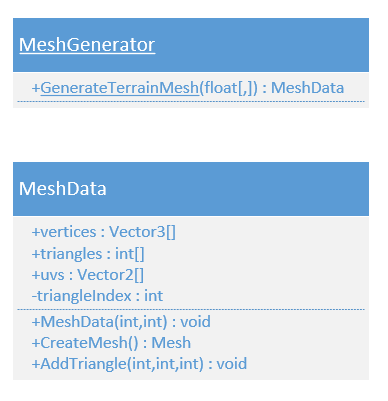
### 2.6.1 High Level Flow Chart



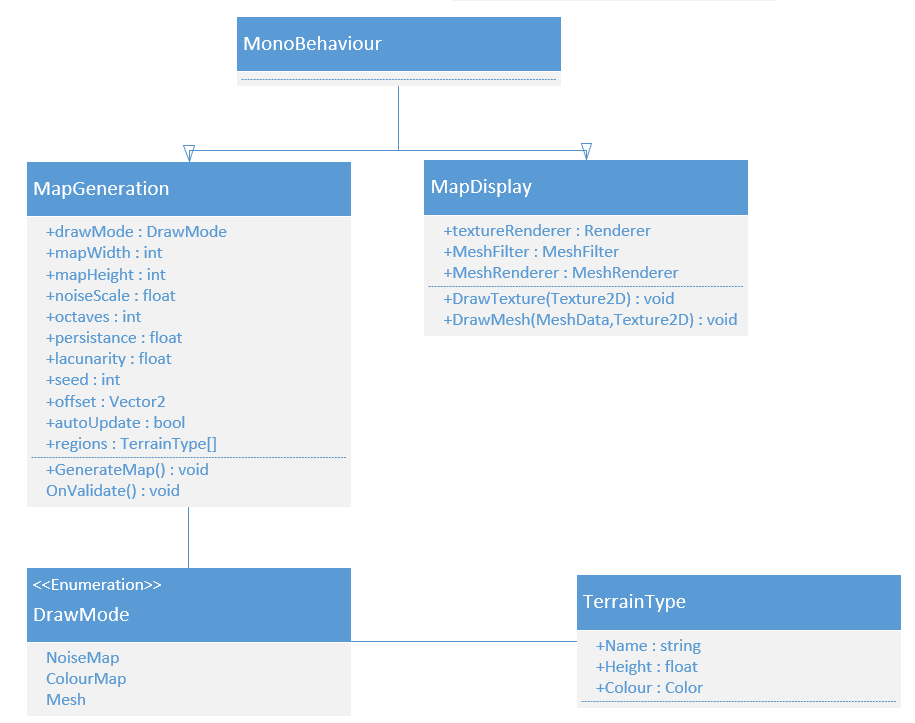
### 2.6.2 Perlin Noise Class



### 2.6.3 Mesh Generation Class



### 2.6.4 Map Generation Class



## 2.7 Project Plan

Planning a project beforehand is a critical process to ensure that the project is going to be completed successfully. To ensure that everything goes as planned, a methodology named “Time-Box” is going to be applied to the project.

At the start of each week, a Gant Chart and a work break down structure is going to be created, this will be called a sprint. The risks will be managed at the end of each sprint.

### 2.7.1 Time estimation

To see the full Gantt Chart, see [Appendix C.10](#_C.10)_Gantt_Chart)

#### 2.7.1.1 Sprint 1 – Initial Setup

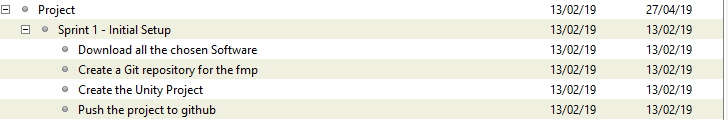


Figure 9 - Gantt Chart Sprint 1

#### 2.7.1.2 Sprint 2 – Algorithm Researching

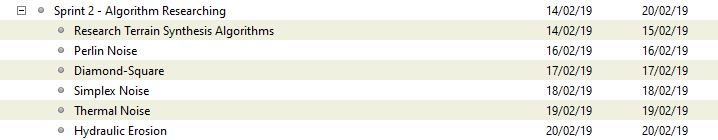


Figure 10 - Gantt Chart Sprint 2

#### 2.7.1.3 Sprint 3 – Finishing the Perlin Noise Function and Research in Terrain Synthesis techniques

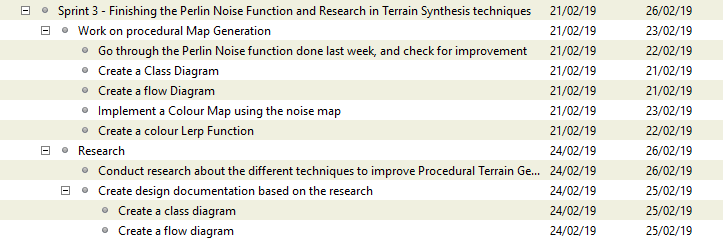


Figure 11 – Gantt Chart Sprint 3

Week 13/02/2019 – 23/02/2019

WBS

1. Work on the Procedural Map Generation (50%) (15 hours)
   1. Create a Class Diagram (5% Design)1h30m
   2. Create a flow Diagram (5% Design)1h30m
   3. Go through the Perlin Noise function done last week, and check for improvement (10% Programming)3h
   4. Implement a Colour Map using the noise map (20% Research/Programming)6h
   5. Create a colour Lerp Function (10% Research/Programming)3h
2. Research (50%) (15 hours)
   1. Conduct research about the different techniques to improve Procedural Terrain Generation (20% Research)
   2. Conduct research on the Diamond-square algorithm (20% Research)
   3. Create design documentation based on the research (10% Design)
      1. Create a class diagram (5% Design)
      2. Create a flow diagram (5% Design)

#### 2.7.1.4 Sprint 4 – Implementing the Mesh Generator class to display the heightmap



Figure 12 - Gantt Chart Sprint 4

#### 2.7.1.5 Sprint 5 – Graphic interface to change the terrain generation on the go



Figure 13 - Gantt Chart Sprint 5

#### 2.7.1.6 Sprint 6 – Implement Thermal Noise algorithm for terrain erosion



Figure 14 - Gantt Chart Sprint 6

#### 2.7.1.7 Sprint 7 – Implement Hydraulic Erosion algorithm



Figure 15 - Gantt Chart Sprint 7

#### 2.7.1.8 Sprint 8 – Create a Unity Plug-in and Work on the final report



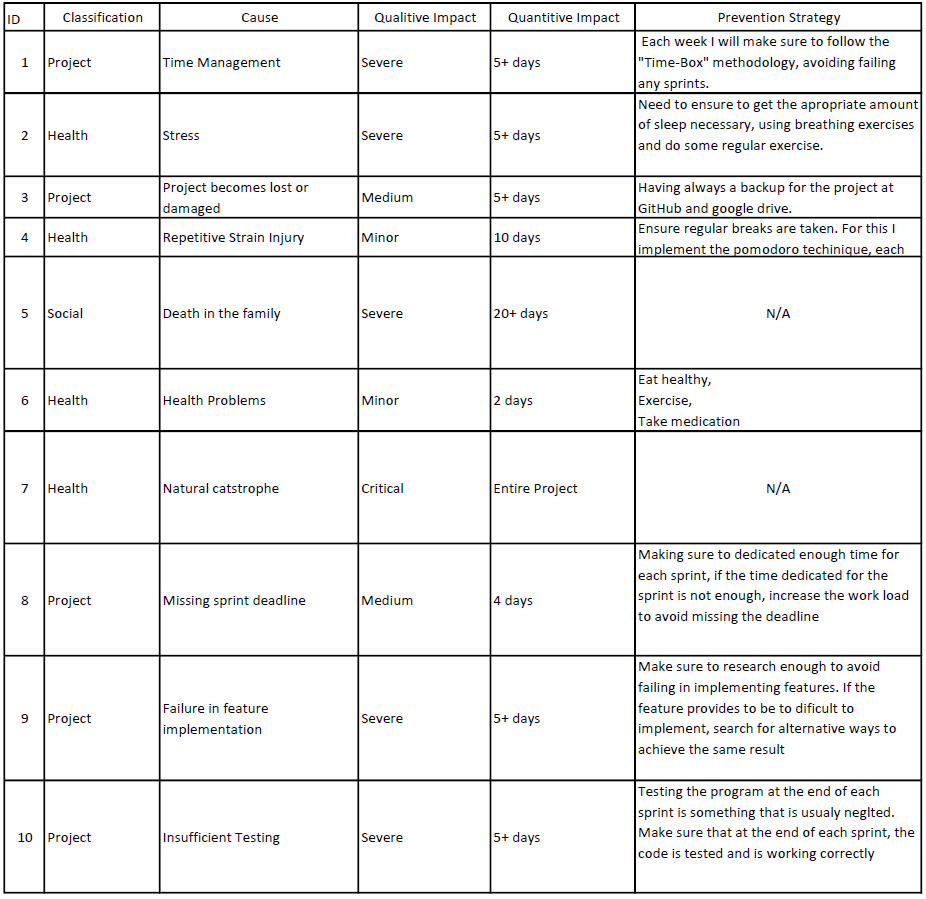
Figure 16 - Gantt Chart Sprint 8

### 2.7.2 Feasibility and Risk

To prevent any risk from occurring, I am going to use time management techniques like pomodoro and Kanban (Cirillo 2006; Cummings 2018; Cirillo 2017; Marcus Hammarberg and Joakim Sundén 2014; Anderson 2010; Peterson 2015).

I believe that creating a terrain generator with Hight maps is a feasible project, for avoiding some of the risks, I will make sure that I work at least 30 hours per week on the project. To see the risks that may occur during the development of this project see the Risk Analysis ([Appendix](#_Risk_and_Analysis) C.9).

### 2.7.3 Risk Analysis



# References

ANDERSON, D.J., 2010. Kanban. Sequim, Wash: Blue Hole Press

CIRILLO, F., 2006. The Pomodoro Technique. [viewed 02/05/ 2018]. Available from: <http://baomee.info/pdf/technique/1.pdf>

CIRILLO, F., 2017. Do more and have fun with time management. [viewed 02/05/ 2018]. Available from: <https://francescocirillo.com/pages/pomodoro-technique/book/>

CUMMINGS, T., 2018. The Pomodoro Technique: Is It Right For You? [viewed 02/05/ 2018]. Available from: <https://www.lifehack.org/articles/productivity/the-pomodoro-technique-is-it-right-for-you.html>

FINLEY, K., 2014. WHY A SIMPLE TO-DO LIST TOOL IS WINNING OVER LEGIONS OF FANS [viewed 23/02/ 2019]. Available from: <https://www.wired.com/2014/08/why-a-simple-to-do-list-tool-is-winning-over-legions-of-fans/>

FOURNIER, A., D. FUSSELL and L. CARPENTER, 1982. Computer rendering of stochastic models. Communications of the ACM, Jun 1, 371-384

MARCUS HAMMARBERG and JOAKIM SUNDÉN, 2014. Kanban in Action. Manning Publications

OLSE, J., 2004. Realtime Procedural Terrain Generation. Department of Mathematics and Computer Science. University of Southern Denmark

OLSEN, J., 2004. Realtime Synthesis of Eroded Fractal Terrain for Use in Computer Games. Department of Mathematics and Computer Science

PETERSON, D., 2015. What is Kanban? [viewed 02/05/ 2018]. Available from: <http://kanbanblog.com/explained/>

PERLIN, K., 2002a. Improved Noise reference implementation [viewed 13/02/ 2019]. Available from: <https://mrl.nyu.edu/~perlin/noise/>

PERLIN, K., 2002b. Improving noise. ACM Transactions on Graphics (TOG), 21(3), 681-682

RIVEN, 2010. Calculate PerlinNoise faster with an optimization to grad(). In: August, Available from: <http://riven8192.blogspot.com/2010/08/calculate-perlinnoise-twice-as-fast.html>

ROHDE, S., 2013. Weathering and Erosion Basics [viewed 28/02/ 2019]. Available from: <https://www.youtube.com/watch?time_continue=0&v=CNUzTmPKxv8>

ROTHMAN, J., 2017. Why We All Use Timeboxes Available from: <https://www.agilealliance.org/why-we-all-use-timeboxes/>

SOFTWARE ADVICE, 2017. KanbanFlow Software [viewed 23/02/ 2019]. Available from: <https://www.softwareadvice.com/uk/project-management/kanbanflow-profile/>

ZUCKER, M., 2001. The Perlin noise math FAQ [viewed 10/02/ 2019]. Available from: <https://mzucker.github.io/html/perlin-noise-math-faq.html>

# Bibliography

ALEXANDRE PHILIPPE MANGRA, ADRIAN SABOU and DORIAN GORGAN, 2016. TSCH Algorithm - Terrain Synthesis from Crude Heightmaps. Romanian Journal of Human - Computer Interaction, 9(2), 119Top of Form

Bottom of Form

ANDERSON, D.J., 2010. Kanban. Sequim, Wash: Blue Hole Press

BECK, S., 2013. Minecraft in Unity 3D [Flashing Images] - One-Week Programming Challenge [viewed 10/01/ 2019]. Available from: <https://www.youtube.com/watch?v=qdwUkYrHosk&t=>

CIRILLO, F., 2006. The Pomodoro Technique. [viewed 02/05/ 2018]. Available from: <http://baomee.info/pdf/technique/1.pdf>

CIRILLO, F., 2017. Do more and have fun with time management. [viewed 02/05/ 2018]. Available from: <https://francescocirillo.com/pages/pomodoro-technique/book/>

CUMMINGS, T., 2018. The Pomodoro Technique: Is It Right For You? [viewed 02/05/ 2018]. Available from: <https://www.lifehack.org/articles/productivity/the-pomodoro-technique-is-it-right-for-you.html>

ESTEVEZ, C., 2019. What is HacknPlan? [viewed 23/02/ 2019]. Available from: <https://hacknplan.com/knowledge-base/what-is-hacknplan/>

FINLEY, K., 2014. WHY A SIMPLE TO-DO LIST TOOL IS WINNING OVER LEGIONS OF FANS [viewed 23/02/ 2019]. Available from: <https://www.wired.com/2014/08/why-a-simple-to-do-list-tool-is-winning-over-legions-of-fans/>

FLAFLA2, 2014. Understanding Perlin Noise [viewed 28/02/ 2019]. Available from: <http://flafla2.github.io/2014/08/09/perlinnoise.html>

FOURNIER, A., D. FUSSELL and L. CARPENTER, 1982. Computer rendering of stochastic models. Communications of the ACM, Jun 1, 371-384

GAMEDESIGNING, 2018. We Review The Hacknplan App [viewed 23/02/ 2019]. Available from: <https://www.gamedesigning.org/gaming/hacknplan/>

GÉNEVAUX, J. et al., 2013. Terrain generation using procedural models based on hydrology. ACM Transactions on Graphics (TOG), 32(4), 1-13

JAVIDX9, 2017. Programming Perlin-like Noise (C++) [viewed 01/03/ 2019]. Available from: <https://www.youtube.com/watch?v=6-0UaeJBumA>

NESO ACADEMY, 2018. Bitwise Operators in C (Part 2) [viewed 01/03/ 2019]. Available from: <https://www.youtube.com/watch?v=8aFik6lPPaA>

OLSE, J., 2004. Realtime Procedural Terrain Generation. Department of Mathematics And Computer Science, University of Southern Denmark,

OLSEN, J., 2004. Realtime Synthesis of Eroded Fractal Terrain for Use in Computer Games. Department of Mathematics And Computer Science,

PERLIN, K., Noise and Turbulence [viewed 28/02/ 2019]. Available from: <https://mrl.nyu.edu/~perlin/doc/oscar.html#noise>

PERLIN, K., 1985. An Image Synthesizer

PERLIN, K., 2001. Noise Hardware [viewed 27/02/2019]. Available from: <https://www.csee.umbc.edu/~olano/s2002c36/ch02.pdf>

PERLIN, K., 2002a. Improved Noise reference implementation [viewed 13/02/ 2019]. Available from: <https://mrl.nyu.edu/~perlin/noise/>

PERLIN, K., 2002b. Improving noise. ACM Transactions on Graphics (TOG), 21(3), 681-682

RIVEN, 2010. Calculate PerlinNoise faster with an optimization to grad(). In: August, Available from: <http://riven8192.blogspot.com/2010/08/calculate-perlinnoise-twice-as-fast.html>

ROHDE, S., 2013. Weathering and Erosion Basics [viewed 28/02/ 2019]. Available from: <https://www.youtube.com/watch?time_continue=0&v=CNUzTmPKxv8>

ROTHMAN, J., 2017. Why We All Use Timeboxes Available from: <https://www.agilealliance.org/why-we-all-use-timeboxes/>

SMELIK, R.M. et al., 2009. A Survey of Procedural Methods for Terrain Modelling.

SOFTWARE ADVICE, 2017. KanbanFlow Software [viewed 23/02/ 2019]. Available from: <https://www.softwareadvice.com/uk/project-management/kanbanflow-profile/>

THE CODING TRAIN, 2015. I.5: Perlin Noise - The Nature of Code [viewed 28/02/ 2019]. Available from: <https://www.youtube.com/watch?v=8ZEMLCnn8v0>

THE CODING TRAIN, 2016. 13.1: Introduction - Perlin Noise and p5.js Tutorial [viewed 28/02/ 2019]. Available from: <https://www.youtube.com/watch?v=Qf4dIN99e2w>

ZUCKER, M., 2001. The Perlin noise math FAQ [viewed 10/02/ 2019]. Available from: <https://mzucker.github.io/html/perlin-noise-math-faq.html>

# Appendix

## Appendix A) Reading List

ALEXANDRE PHILIPPE MANGRA, ADRIAN SABOU and DORIAN GORGAN, 2016. TSCH Algorithm - Terrain Synthesis from Crude Heightmaps. Romanian Journal of Human - Computer Interaction, 9(2), 119

Top of Form

Bottom of Form

FOURNIER, A., D. FUSSELL and L. CARPENTER, 1982. Computer rendering of stochastic models. Communications of the ACM, Jun 1, 371-384

GÉNEVAUX, J. et al., 2013. Terrain generation using procedural models based on hydrology. ACM Transactions on Graphics (TOG), 32(4), 1-13

OLSE, J., 2004. Realtime Procedural Terrain Generation. Department of Mathematics and Computer Science, University of Southern Denmark,

OLSEN, J., 2004. Realtime Synthesis of Eroded Fractal Terrain for Use in Computer Games. Department of Mathematics and Computer Science,

PERLIN, K., 1985. An Image Synthesizer

PERLIN, K., 2001. Noise Hardware [viewed 27/02/2019]. Available from: <https://www.csee.umbc.edu/~olano/s2002c36/ch02.pdf>

PERLIN, K., 2002. Improving noise. ACM Transactions on Graphics (TOG), 21(3), 681-682

SMELIK, R.M. et al., 2009. A Survey of Procedural Methods for Terrain Modelling.

## Appendix B) Literature review

### B.1) Terrain Synthesis Techniques Research

**THE CODING TRAIN, 2015. *I.5: Perlin Noise - The Nature of Code*[viewed 28/02/ 2019]. Available from:**[**https://www.youtube.com/watch?v=8ZEMLCnn8v0**](https://www.youtube.com/watch?v=8ZEMLCnn8v0)

**THE CODING TRAIN, 2016. *13.1: Introduction - Perlin Noise and p5.jsTutorial*[viewed 28/02/ 2019]. Available from:**[**https://www.youtube.com/watch?v=Qf4dIN99e2w**](https://www.youtube.com/watch?v=Qf4dIN99e2w)

These two videos introduced to me the Perlin Noise algorithm and how it works. They discuss that Perlin Noise is a pseudo-random noise, meaning that the values are not random at all, but the increment in a slow way, between the value 0 and 1.

**BECK, S., 2013. *Minecraft in Unity 3D [Flashing Images] - One-Week Programming Challenge*[viewed 28/02/ 2019]. Available from:**[**https://www.youtube.com/watch?v=qdwUkYrHosk&t=**](https://www.youtube.com/watch?v=qdwUkYrHosk&t=)

This link shows a finish product in unity, where the user develops a voxel world like Minecraft, using procedural terrain generation.

### B.2) Perlin Noise Research

**PERLIN, K., 2002. Improving noise. *ACM Transactions on Graphics (TOG),*21(3), 681-682**

This was the first Perlin Noise academic journal that I came across, practically this journal refers that the original algorithm of noise had two deficiencies.

Un-optimal gradient computation, in other words the gradient function was replaced by 12 defined vectors. The other deficiency was found on the fade function that was after replaced by .

**FLAFLA2, 2014. *Understanding Perlin Noise*[viewed 28/02/ 2019]. Available from:**[**http://flafla2.github.io/2014/08/09/perlinnoise.html**](http://flafla2.github.io/2014/08/09/perlinnoise.html)

“Understanding Pelin Noise”, was in this web page that I learned all about the algorithm and how to recreate a working version of the Perlin Noise. This is just a copy of the original Perlin Noise but updated to the improved version of the algorithm.

**ZUCKER, M., 2001. *The Perlin noise math FAQ*[viewed 10/02/ 2019]. Available from:**[**https://mzucker.github.io/html/perlin-noise-math-faq.html**](https://mzucker.github.io/html/perlin-noise-math-faq.html)

This source is a combination from the Ken Perlin’s Making Noise web site, and the Higo Elia’s page, but unfortunately this web pages were taken down.

**PERLIN, K., *Noise and Turbulence*[viewed 28/02/ 2019]. Available from:**[**https://mrl.nyu.edu/~perlin/doc/oscar.html#noise**](https://mrl.nyu.edu/~perlin/doc/oscar.html#noise)

In this link Ken Perlin shows how to implement Perlin Noise function and how to do it using different dimensions functions.

**PERLIN, K., 1985. An Image Synthesizer**

Talks about five ideas for visual texture synthesis. The author introduces the concept of Pixel Stream Editor. Explaining how to develop efficient naturalistic looking textures. The author talks about how he used this system to create a believable representation of fire, water, clouds, waves, marble and oil films.

**RIVEN, 2010. Calculate Perlin Noise faster with an optimization to grad(). In: August, Available from:**[**http://riven8192.blogspot.com/2010/08/calculate-perlinnoise-twice-as-fast.html**](http://riven8192.blogspot.com/2010/08/calculate-perlinnoise-twice-as-fast.html)

The author discovered an efficient way to simplify the Perlin Noise algorithm changing the gradient function and making it twice as fast, returning the gradient directions in a simplified way instead of doing the complicated bit manipulation that Ken Perlin does on the original algorithm.

**NESO ACADEMY, 2018. *Bitwise Operators in C (Part 2)*[viewed 01/03/ 2019]. Available from:**[**https://www.youtube.com/watch?v=8aFik6lPPaA**](https://www.youtube.com/watch?v=8aFik6lPPaA)

This author thought me how the bitwise operators worked. This helped me understand how the bit manipulation of the gradient function of Ken Perlin worked.

**JAVIDX9, 2017. *Programming Perlin-like Noise (C++)*[viewed 01/03/ 2019]. Available from:**[**https://www.youtube.com/watch?v=6-0UaeJBumA**](https://www.youtube.com/watch?v=6-0UaeJBumA)

This author gives an overview and explains more in detail how to implement the algorithm in a different language like C++.

### B.3) Diamond-Square Research

**OLSE, J., 2004. Realtime Procedural Terrain Generation. Department of Mathematics and Computer Science, University of Southern Denmark**

In this journal the author implements the Diamond-Square algorithm, explaining how it works.

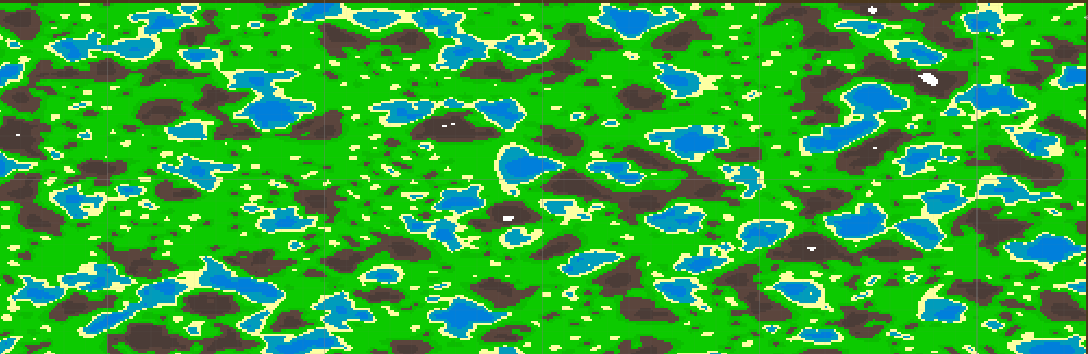
**OLSEN, J., 2004. Realtime Synthesis of Eroded Fractal Terrain for Use in Computer Games. *Department of Mathematics and Computer Science***

In this journal is were the idea of the diamond-square algorithm was introduced.

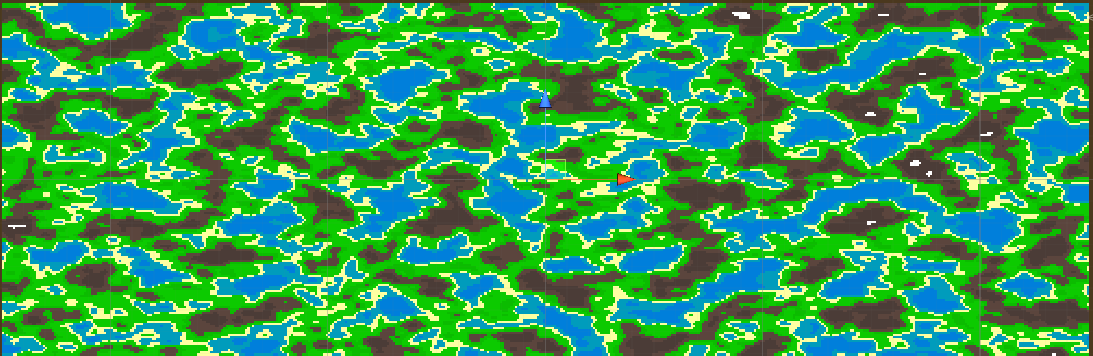
“First, we compute the boundary of the patch, using the one-dimensional version of the algorithm to the level desired. We then fill the square for each level, computing the centres, then the sides, using at each step the four neighbours (diagonally for the centres, horizontally and vertically for the sides). At each step the new point is computed as a Gaussian pseudo-random variable, whose expected value is the mean of the four neighbours at this level, and whose standard deviation is , with the level, H the self-similarity parameter, and c a constant to be adjusted to fit the application” (Olse 2004)

## Appendix C) Images

### C.1) Prototype using the 2D Perlin noise function that I created



### C.2) Prototype using the Unity Perlin Noise Function



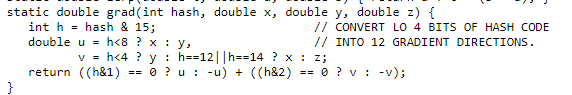
### C.3) Perlin Noise from my function



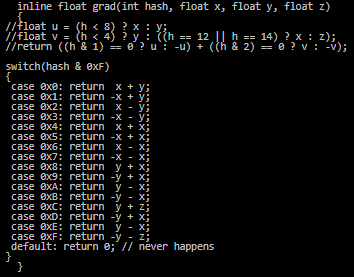
### C.4) Perlin Noise from unity



### C.5) Ken Perlin’s original gradient function



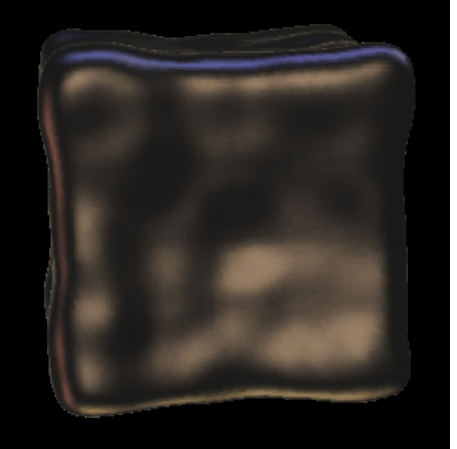
### C.6) Riven Calculate Perlin Noise twice as fast



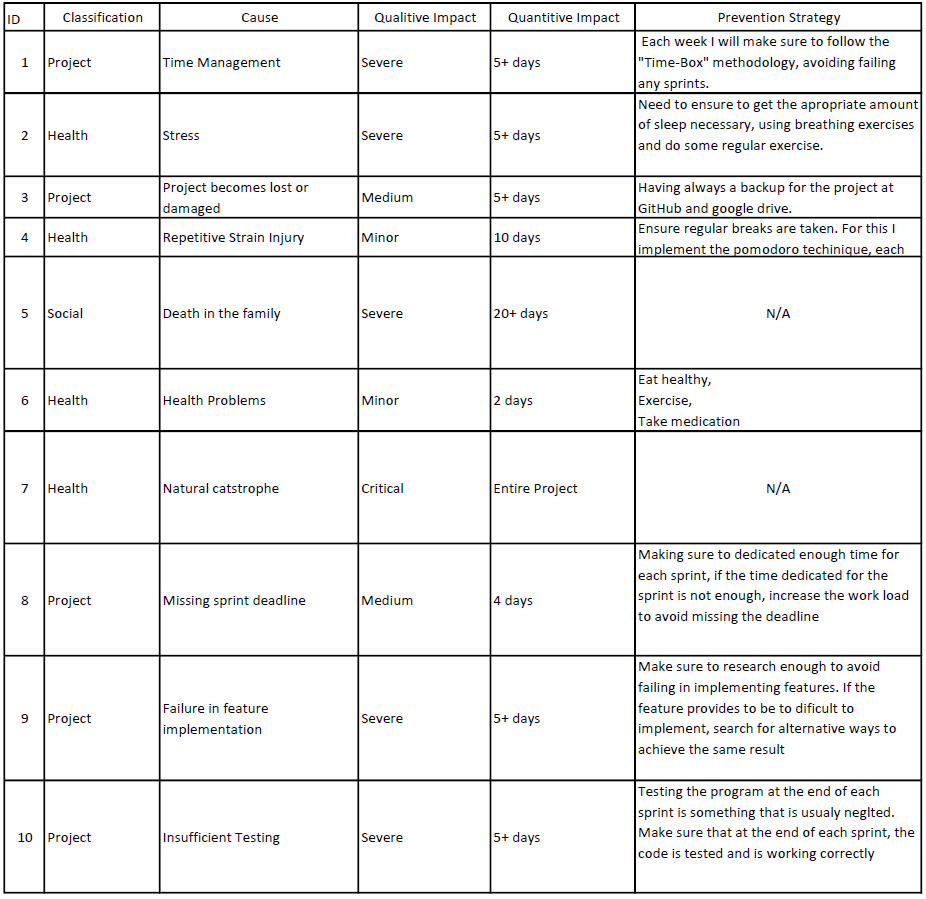
### C.7) Noise Displaced super quadric with old interpolants



### C.8) Noise Displaced super quadric with new interpolants



### C.9) Risk and Analysis



### C.10) Gantt Chart

