SOUTHAMPTON SOLENT UNIVERSITY

BSc (Hons) Computer Games (Software Development)

Academic Year 2017 – 2019

Terrain Generation Report

Flavio Fiori

# Abstract

Contents

[1. Abstract 2](#_Toc1662416)

[2. Progress Report 5](#_Toc1662417)

[2.1 Overview 5](#_Toc1662418)

[2.1.1 Project 5](#_Toc1662419)

[2.1.2 Why I want to undertake this project and how it is going to challenge me 5](#_Toc1662420)

[2.1.3 Aims 6](#_Toc1662421)

[2.1.4 Objectives 6](#_Toc1662422)

[2.1.5 Stretch Goals 6](#_Toc1662423)

[2.2 Project Specification 7](#_Toc1662424)

[2.2.1 Software 7](#_Toc1662425)

[2.3 Research 8](#_Toc1662426)

[2.3.1 Perlin Noise 8](#_Toc1662427)

[2.3.2 Diamond-Square Algorithm 8](#_Toc1662428)

[2.3.3 Simplex Noise 8](#_Toc1662429)

[2.4 Prototyping 8](#_Toc1662430)

[2.5 Project Management 9](#_Toc1662431)

[2.5.1 Methodologies 9](#_Toc1662432)

[2.5.2 Tools for Time Management 9](#_Toc1662433)

[2.6 Design 9](#_Toc1662434)

[2.6.1 Entity Relationship Diagram 9](#_Toc1662435)

[2.6.2 Perlin Noise Class 9](#_Toc1662436)

[2.6.3 Mesh Generation Class 9](#_Toc1662437)

[2.6.4 Map Generation Class 9](#_Toc1662438)

[2.7 Project Plan 9](#_Toc1662439)

[2.7.1 Time estimation 10](#_Toc1662440)

[2.7.2 Feasibility and Risk 11](#_Toc1662441)

[2.7.3 Risk Analysis 11](#_Toc1662442)

[Appendix 12](#_Toc1662443)

[A) Prototype using the 2D Perlin noise function that I created 12](#_Toc1662444)

[B) Prototype using the Unity Perlin Noise Function 12](#_Toc1662445)

[C) Perlin Noise from my function 13](#_Toc1662446)

[D) Perlin Noise from unity 14](#_Toc1662447)

[E) Ken Perlin’s original gradient function 14](#_Toc1662448)

[F) Riven Calculate Perlin Noise twice as fast 15](#_Toc1662449)

[G) Noise Displaced super quadric with old interpolants 15](#_Toc1662450)

[H) Noise Displaced super quadric with new interpolants 16](#_Toc1662451)

[I) Risk and Analysis 17](#_Toc1662452)

[Reference 17](#_Toc1662453)

# 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
  + Hydraulic 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

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 G](#_Noise_Displaced_super)) ([Appendix H](#_Noise_Displaced_super_1))

“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.”

-Ken Perlin

### 2.3.2 Diamond-Square Algorithm

### 2.3.3 Simplex Noise

## 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 B](#_Prototype_using_the)), 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 slightly 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 E](#_Ken_Perlin’s_original)) (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 F](#_Riven_Calculate_Perlin)) (Riven 2010).

Is possible to see the first result of the 2D Perlin noise function that was implemented in the ([Appendix A](#_Prototype_using_the_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)) similar to the original Unity function ([Appendix D](#_Perlin_Noise_from_1)).

## 2.5 Project Management

### 2.5.1 Methodologies

### 2.5.2 Tools for Time Management

#### 2.5.2.1 Trello

#### 2.5.2.2 KanBan Flow

#### 2.5.2.3 HacknPlan

#### 2.5.2.4 Microsoft OneNote

#### 2.5.2.5 Microsoft Excel

#### 2.5.2.6 GitHub

## 2.6 Design

### 2.6.1 Entity Relationship Diagram

### 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

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

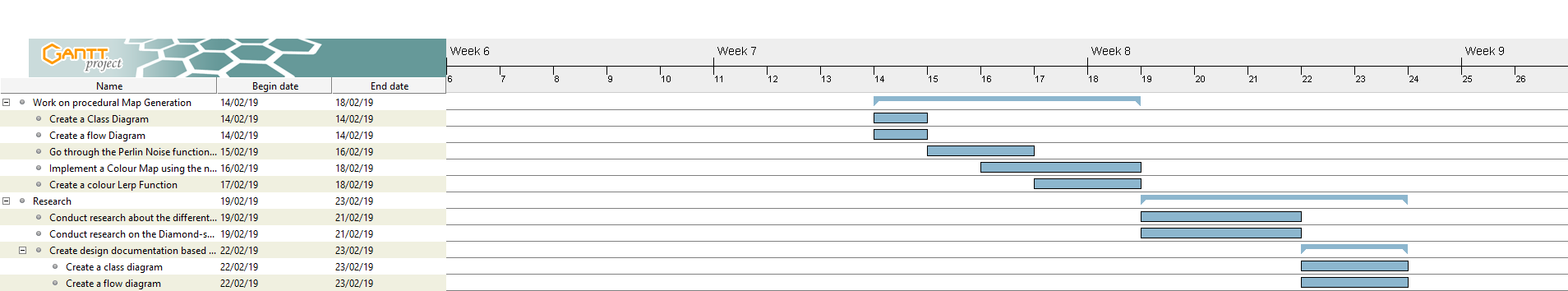


Figure – Weekly Gant Chart using the Time-Box methodology

Week 13/02/2019 – 23/02/2019

WBS

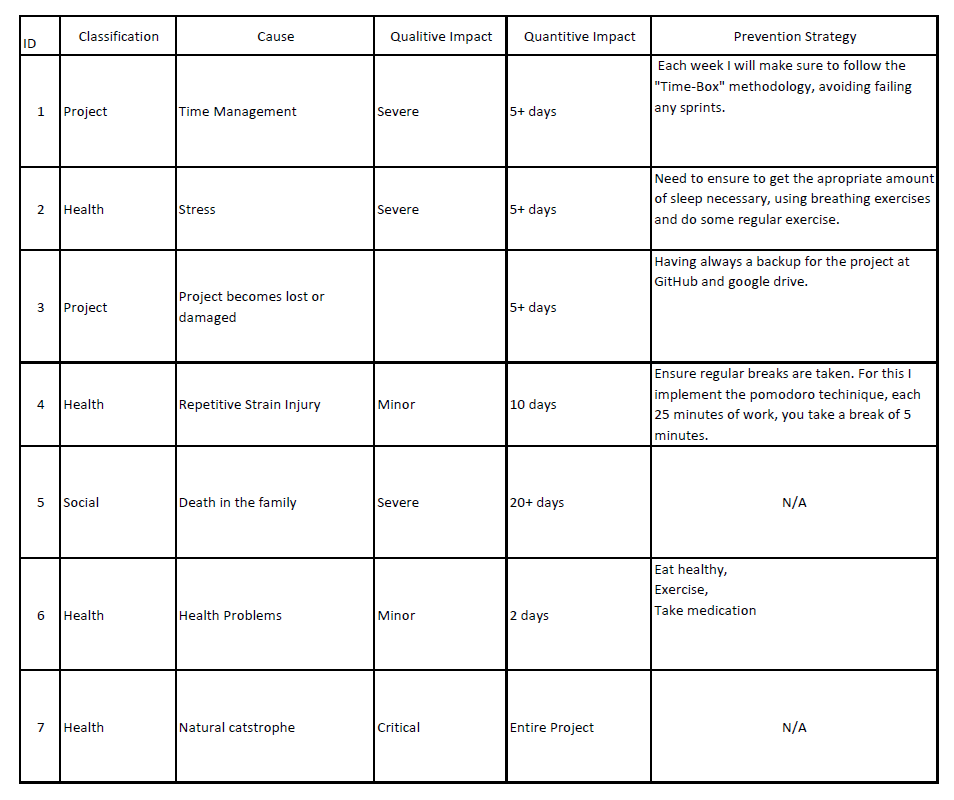
1. Work on the Procedural Map Generation (50%) (15 hours)
   1. Create a Class Diagram (5% Design)1h30m
   2. Create a flow Diagram (5% Design)1h30m
   3. Go through the Perlin Noise function done last week, and check for improvement (10% Programming)3h
   4. Implement a Colour Map using the noise map (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)

### 2.7.2 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).

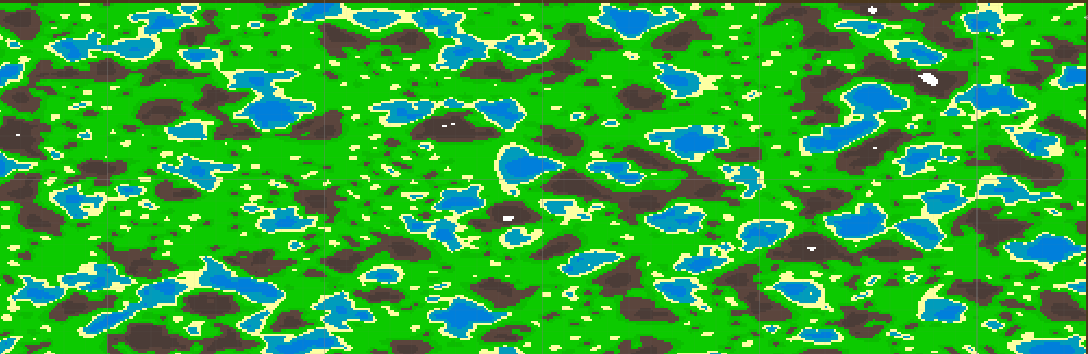
I believe that creating a terrain generator with Hight maps is a feasible project, for avoiding some of the risks, I will make sure that I work at least 30 hours per week on the project. To see the risks that may occur during the development of this project see the Risk Analysis ([Appendix I](#_Risk_and_Analysis)).

### 2.7.3 Risk Analysis

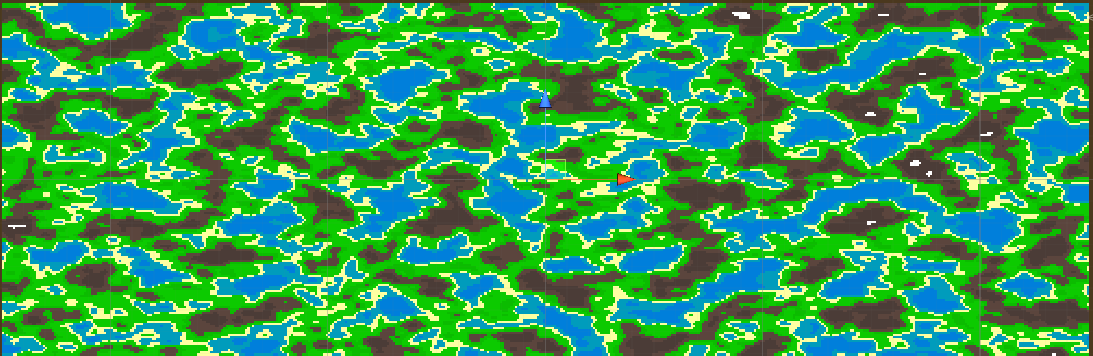


# Appendix

## Prototype using the 2D Perlin noise function that I created



## Prototype using the Unity Perlin Noise Function



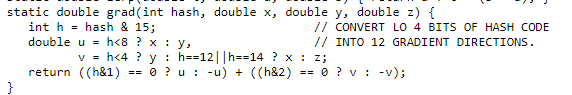
## Perlin Noise from my function



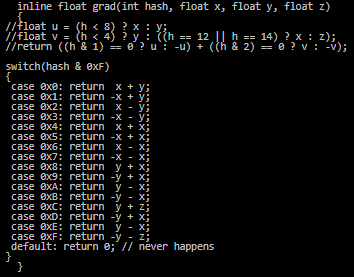
## Perlin Noise from unity



## Ken Perlin’s original gradient function



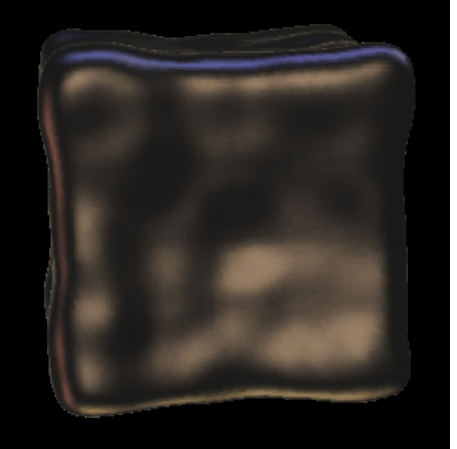
## Riven Calculate Perlin Noise twice as fast



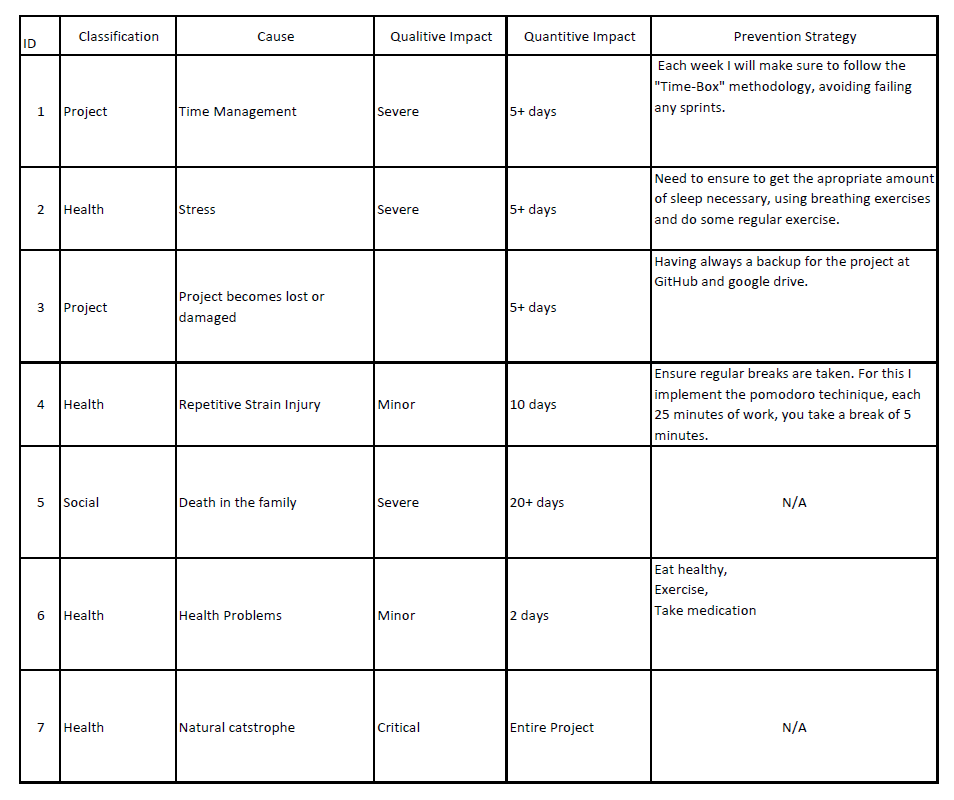
## Noise Displaced super quadric with old interpolants



## Noise Displaced super quadric with new interpolants



## Risk and Analysis



# Reference

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>

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

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>

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