Artist Away: Procedural Generation of Terrain

Sam Connolly

8 April 2017

A project report submitted in partial fulfilment for the degree of

**Bachelor of Science in Computer Games Development**

**School of Physical Sciences and Computing**

**University of Central Lancashire**

Abstract

The abstract is the summary of the project report within one page (aim for about 500 words). Unnumbered chapter headings, as above, are entered using the ‘Heading (Unnumbered)’ style, which automatically starts a new page.

This template starts the page numbering at the foot of this page. That is, the first page does not have a number.

It is suggested that the abstract be structured as follows:

Problem: What you tackled, and why this needed a solution

Objectives: What you set out to achieve, and how this addressed the problem

Methodology: How you went about solving the problem

Achievements: What you managed to achieve, and how far it meets your objectives.

Currently throughout the modern-day games industry, artists are required to work alongside games developers to assist with the creation of models and levels. This presents problems throughout industry as artists and developers frequently have very different ideas about what is required from art assets, and how they should be implemented. There is a balance between implementing realistic models and artwork, and efficient rendering of an entire scene, however frequently this balance is not met due to a difference of opinion between artists and developers. Constant battling between artists and developers can be a drain on company funds, with so many games studios going out of business, if a company can save money by not having to hire an artist or have an artist’s input on a technical solution then it makes sense not to.

Artist Away set out to generate a realistic looking terrain which could be used as a level of a game, without having to plan the game level beforehand. This would involve the generation of the shape of the level, layering textures on the generated level shape to represent real world terrain, creating level entities such as trees and plants, presenting realistic day and night scenes, and finally presenting a body of water which runs underneath the level. I believe these are the core components of a game level, and all can be generated without the requirement for an artist. Artist Away would do all of this, and could save a company the salary of an artist, and potentially keep that company in business.

Artist Away makes use of procedural generation by generating a height map, the engine then controls plotting vertices per the provided height map, and decides which type of terrain area each vertex belongs to per the vertex’s relative height. The engine then textures the vertices with the appropriate texture depending on the area type, and then generates entities in each area. The engine then defines the vertices for a body of water, and positions it underneath the terrain, all properties of the water will be controlled through the engine. The body of water adds realism to the scene. Finally, day and night cycles are triggered through the engine class, it should simply perform a gradual change of the skybox colour which will represent a time of day in which the sun is at a different position.

Artist Away successfully uses Simplex Noise to generate a height map, and the engine then loads in this height map through one of two interfaces, a pre-generated ‘.map’ file or a dynamic two-dimensional array of values. The engine is responsible for dictating which vertex of the terrain belongs in which area, it then makes use of HLSL shaders to render the appropriate texture to each area of the terrain model. The engine also has a small change to generate a tree or plant in grass areas, and position them at a random rotation. Day and night cycles area achieved by changing the colour of the skybox. Water is created as a plane which sits underneath the terrain, and the properties of the water are controlled through the engine.

Attestation

I understand the nature of plagiarism, and I am aware of the University’s

policy on this.

I certify that this document reports original work by me during my University project.

**Signature**  **Date**

Acknowledgements

Acknowledge anyone who has helped you in your work such as your supervisor, technical support staff, fellow students or external organisations. Acknowledge the source of any work that is not your own.

Laurent Noel is my project supervisor and has helped with technical explanations of areas which I then went on to implement within my project.

ASSIMP model library is a third-party library which I used for the loading of models within my engine, while I have set up this library myself I have not implemented the core mechanics behind most model loading.

AntTweakBar is a third-party library which provides an interface for controlling things within my engine, while I have set up this library myself, I have not implemented the core mechanics behind drawing the tweak bar and responding to events in the tweak bar.

3D Game Programming with DirectX 11 – Book by Frank Luna which heavily influenced my implementation of the game engine timer class. While my implementation is not identical, the structure of the timer is the same.

<http://www.rastertek.com> is a group of tutorials which supplied me with the structure for my Direct X 11 engine, while there are similarities between my classes and shaders, there are also a wide variety of differences.

Table of Contents

[Abstract i](#_Toc478568129)

[Attestation iii](#_Toc478568130)

[Acknowledgements iv](#_Toc478568131)

[Table of Contents v](#_Toc478568132)

[List of Figures ix](#_Toc478568133)

[List of Tables x](#_Toc478568134)

[List of Listings xi](#_Toc478568135)

[1 Introduction 1](#_Toc478568136)

[1.1 Background and Context 1](#_Toc478568137)

[1.2 Scope and Objectives 1](#_Toc478568138)

[1.3 Achievements 1](#_Toc478568139)

[1.4 Overview of Report 2](#_Toc478568140)

[2 Literature Review 3](#_Toc478568141)

[2.1 Introduction 3](#_Toc478568142)

[2.2 Game Engine Structures 3](#_Toc478568143)

[2.2.1 Class Hierarchy Based Engines 3](#_Toc478568144)

[2.2.2 Component Based Engines 4](#_Toc478568145)

[2.3 Procedural Generation 4](#_Toc478568146)

[2.4 Summary 5](#_Toc478568147)

[3 Project Planning 6](#_Toc478568148)

[3.1 Introduction 6](#_Toc478568149)

[3.2 Methodology 6](#_Toc478568150)

[3.2.1 Scheduling 6](#_Toc478568151)

[3.2.2 Development Lifecycle 6](#_Toc478568152)

[3.2.3 Testing Plan 7](#_Toc478568153)

[3.3 Requirements 7](#_Toc478568154)

[3.3.1 Selecting a Game Engine and Framework 7](#_Toc478568155)

[3.3.2 Creating a Game Engine 7](#_Toc478568156)

[3.4 Potential Solutions 8](#_Toc478568157)

[3.4.1 Implementation of a Class Hierarchy Based Engine 8](#_Toc478568158)

[3.4.2 Generating Noise 9](#_Toc478568159)

[3.4.3 Generating Terrain from Height Maps 9](#_Toc478568160)

[3.5 Tools and Techniques 9](#_Toc478568161)

[3.6 Legal and Ethical Issues 10](#_Toc478568162)

[3.6.1 Is It Ethical to Automate a Person’s Role? 10](#_Toc478568163)

[3.6.2 Automation In Industry 10](#_Toc478568164)

[3.6.3 Legal Use of Third Party Tools and Resources 11](#_Toc478568165)

[3.7 Potential Technical Issues 11](#_Toc478568166)

[3.8 Algorithms 12](#_Toc478568167)

[3.9 Summary 12](#_Toc478568168)

[4 Design 13](#_Toc478568169)

[4.1 Introduction 13](#_Toc478568170)

[4.2 System Design 13](#_Toc478568171)

[4.2.1 Game Engine Design 13](#_Toc478568172)

[4.2.2 Procedural Generation 14](#_Toc478568173)

[4.3 User Interface Design 14](#_Toc478568174)

[4.3.1 Prio Engine 14](#_Toc478568175)

[4.3.2 Artist Away 15](#_Toc478568176)

[4.4 Summary 16](#_Toc478568177)

[5 Prio Engine 17](#_Toc478568178)

[5.1 Introduction 17](#_Toc478568179)

[5.2 Timing 17](#_Toc478568180)

[5.3 Layout 17](#_Toc478568181)

[5.3.1 Build Structure 17](#_Toc478568182)

[5.3.2 Usage 17](#_Toc478568183)

[5.3.3 Debugging Tools 17](#_Toc478568184)

[5.4 Framework and Libraries 17](#_Toc478568185)

[5.4.1 Choice of Framework 17](#_Toc478568186)

[5.4.2 AntTweakBar 17](#_Toc478568187)

[5.4.3 ASSIMP 17](#_Toc478568188)

[5.5 Rendering Models 17](#_Toc478568189)

[5.5.1 Predefined Models 17](#_Toc478568190)

[5.5.2 Loading Models 17](#_Toc478568191)

[5.5.3 Rendering Models 17](#_Toc478568192)

[5.5.4 Rendering Terrain 17](#_Toc478568193)

[5.5.5 Rendering Water 17](#_Toc478568194)

[5.5.6 Rendering a Skybox 17](#_Toc478568195)

[5.5.7 Rendering Clouds 17](#_Toc478568196)

[5.6 Lighting 17](#_Toc478568197)

[5.7 Rendering 2D Images 18](#_Toc478568198)

[5.8 Particle Systems 18](#_Toc478568199)

[5.8.1 Rain 18](#_Toc478568200)

[5.9 Summary 18](#_Toc478568201)

[6 Artist Away 19](#_Toc478568202)

[6.1 Simplex Noise Generation 19](#_Toc478568203)

[6.2 Creating and Exporting Height Maps 19](#_Toc478568204)

[6.3 Implementing User Interface 19](#_Toc478568205)

[6.3.1 Controlling the Game 19](#_Toc478568206)

[6.3.2 AntTweakBar Usage 19](#_Toc478568207)

[6.4 Concurrent Programming 19](#_Toc478568208)

[6.4.1 Multithreading Noise Generation 19](#_Toc478568209)

[6.4.2 Preventing Multiple GPU Instructions 19](#_Toc478568210)

[6.5 Updating Terrain During Run Time 19](#_Toc478568211)

[6.5.1 Generating New Height Map 19](#_Toc478568212)

[6.5.2 Destroying Old Entities 19](#_Toc478568213)

[6.5.3 Placing New Entities 19](#_Toc478568214)

[6.5.4 Updating Water 19](#_Toc478568215)

[7 Test Strategy 20](#_Toc478568216)

[7.1 Introduction 20](#_Toc478568217)

[7.2 Section 20](#_Toc478568218)

[7.3 Section 20](#_Toc478568219)

[7.4 Summary 20](#_Toc478568220)

[8 Evaluation, Conclusions and Future Work 21](#_Toc478568221)

[8.1 Project Objectives 21](#_Toc478568222)

[8.2 Evaluation 21](#_Toc478568223)

[8.3 Applicability of Findings to the Commercial World 21](#_Toc478568224)

[8.4 Conclusions 21](#_Toc478568225)

[8.5 Future Work 21](#_Toc478568226)

[8.6 Concluding Reflections 21](#_Toc478568227)

[9 Bibliography 22](#_Toc478568228)

[Appendix 1 – Title of Appendix 23](#_Toc478568229)

[Appendix 2 – Title of Appendix 24](#_Toc478568230)

[Appendix 3 – Title of Appendix 25](#_Toc478568231)

List of Figures

[Figure 2‑1 Class Hierarchy Structure 3](#_Toc478568232)

[Figure 2‑2 Entity Component Structure 4](#_Toc478568233)

[Figure 3‑1 Overview of Prio Engine Class Diagram 8](#_Toc478568234)

[Figure 3‑2 Artist Away Use Case Diagram 9](#_Toc478568235)

[Figure 3‑3 Model Class Diagram Within Prio Engine 10](#_Toc478568236)

List of Tables

Similarly, you can automatically generate a list of ‘Tables’. Select a table, right-click it and add a caption labelled ‘Table’ and ‘above selected item’. To update this after revisions, right-click in this table and choose Update Field (or use F9) and then choose to update the entire table. Delete this paragraph before submission.

[Table 1 - Test Results 6](#_Toc476373581)

List of Listings

This list only applies to you if you use code snippets in your report. If you don’t have any listings, remove this whole section including the heading ‘List of Listings’.

You can automatically generate a list of ‘Listings’. After formatting your Code, move the cursor to the first line below your code block and click ‘References’ -> ‘Insert Caption’ in the ribbon menu. Select the label ‘Listing’ or add a new Label called ‘Listing’ if it does not yet exist. To update this after revisions, right-click the table and choose Update Field (or use F9) and then choose to update the entire table. Delete this paragraph before submission.

[Listing 1 - [Main.java] The main class of the program 5](#_Toc476373582)

# Introduction

## Background and Context

With the release of No Man’s Sky (Hello Games, 2016), procedural generation has become a trending topic in recent times. Procedural generation is the creation of an entity, these entities can range from small entities such as trees, to larger entities such as a game level. Artist Away attempted to tackle the challenge of procedurally generating a level without the requirement for an artist, in theory a programmer would be able to instruct the program to design a level and the remainder of work would be handled by the engine.

This presented a challenge; which engine would be most suited for a procedural generation project? While many engines would allow for procedurally generated worlds, few would provide any form of flexible framework. By creating a bespoke engine, it is ensured that procedural generation would be fully supported by the engine. To implement Artist Away, a Direct X 11 based engine named Prio Engine has been designed.

## Scope and Objectives

Prio Engine is designed to support generation of terrain, however there are other core mechanics which are warranted by a game engine such as mesh loading, model control, render management, debugging logs, image loading, text rendering, accepting user input and lighting. Prio Engine must also support procedural generation specific functions, such as the loading of height maps, plotting terrains, the ability to integrate with the Artist Away project. All the while, Prio Engine should be kept as a flexible Direct X 11 engine to serve as a useful tool for future projects.

Artist Away on the other hand serves the purpose of creating height maps through noise algorithms, and must export the height maps in a format which Prio Engine can load. The ongoing integration between the two projects is crucial to success of this project.

## Achievements

I have successfully created a Direct X 11 Engine, which can be exported as a static library, and from the static library can be used to load, control and manipulate models. The engine can also accept height maps in one of two forms, a two-dimensional array, or a text file (exported from the Artist Away project) and from these height maps, procedurally generate terrain. The terrain is successfully textured using different textures per area, and a smooth blend is used between each of the textures. The engine defines a body of water, and successfully refracts the terrain tiling beneath the water.

I have also created a C++ program called Artist Away which uses Prio Engine as a static library, and can interface successfully with the engine to draw terrains, and supply the engine with height maps which are generated through Simplex Noise. Artist Away can also export ‘.map’ files which can be loaded using Prio Engine, as well as 2 dimensional arrays.

Summarise what you have achieved.

## Overview of Report

Briefly overview the contents of what follows in the report. Overview (1-2 lines per chapter):

'Chapter 3 describes the investigation of the problem and presents the top-level analysis as a Yourdon dataflow diagram. ... Chapter 4 contains an overview of the design architecture and examines the key design issues’.

# Literature Review

## Introduction

Each of your chapters should have an introduction to tell your readers what they will find in the chapter.

Summarise current knowledge and what others have done in the various topics of your dissertation – in the application area and in the various technologies that you might have used or did use. Write for someone familiar with computing, but not necessarily expert in the particular topics of your project. Give references to other work by using cross-references to entries in the References section, like this (Turner & Jennings, 2002).

## Game Engine Structures

### Class Hierarchy Based Engines

Class hierarchy structures are found throughout a wide range of industry software. Class hierarchies are typically found in object oriented programming due to the representation of real world objects within classes. However, Pereira, et al. (2013) explains that as a hierarchy grows larger it grows more difficult to maintain. This proves to be problematic during development of game engines, as flexibility is of the utmost importance and as a result, classes can be added at any point in time which results in a rapidly expanding hierarchy. Figure 2-1 demonstrates a real-world implementation of a class hierarchy.



Figure 2‑1 Class Hierarchy Structure

Despite the drawback of large networks of classes, class hierarchies are an incredibly effective design tool for games engines due to the constant ongoing communication amongst multiple classes per frame. The use of a class hierarchy ensures that objects only communicate with relevant instances of classes within their sub-hierarchy, and as a result reduces unnecessary coupling between classes.

Hierarchy based engines are reasonably flexible and can be extended to implement new features in a game. For example, a hierarchy based engine could be extended to implement a phone in a game through the structure shown in figure 2-1, allowing the engine to classify phones and apply different behaviours for different classifications of phones.

### Component Based Engines

Component based engines operate in a very similar manner to hierarchal based engines, in fact most still make use of hierarchies. What separates a component based engine from a standard hierarchal based engine is that a component based engine will define each world object as an entity, each entity will contain multiple components to help it display the properties of a real-world component. Figure 2-2 demonstrates the use of components with a theoretical game entity. Component based engines provide flexibility to add and remove components, for example if a vehicle were to run out of fuel, the movement component could be taken away from the instance of the vehicle entity and the object would remain in the game.



Figure 2‑2 Entity Component Structure

## Procedural Generation

Procedural generation is often found in games when generating heightmaps and drawing different areas based on their height in relevance to the rest of the height map. However, a wide variety of content can be procedurally generated such as levels, adventures, characters, weapons planets, and plants (Julian, et al., 2013). Procedural generation has very few limitations within games, in theory entire games could be procedurally generated, as demonstrated in No Man’s Sky.

Procedural generation usually occurs through artificial intelligence (AI) methods within computer science. For instance, height maps can be generated through Perlin Noise or Simplex Noise algorithms, a very basic form of AI can then plot these heights into a grid, and dictate how each position of a grid at that height should be coloured.

## Summary

# Project Planning

## Introduction

Writing a game engine is no small task, it requires a large amount of planning. Writing an engine which is to be flexible enough to be used by other applications in future complicates matters further. By planning the structure of how applications will interface with an engine, we can reduce unforeseen errors encountered when using the engine over a variety of applications. It also allows us to plan how the engine can be updated in future, as we know what can change and in what way the methods of the engine can change, if we are aware with how other applications will interact with the engine.

## Methodology

### Scheduling

### Development Lifecycle

An agile approach was selected for Prio Engine since the overall project in fact consists of two separate sub projects. By adopting an agile development lifecycle, Prio Engine can run independently from Artist Away, and can be used with a wide variety of other projects.

Prio Engine consists of many components, and as such both the design and development are required to be as flexible as possible. It is well renowned how fragile IT systems can be if they are not developed in an agile manner (Jacobson, et al., 2016). Developing Prio Engine in an agile format enables the developers to support a wide range of unknown applications in future. However, development of Prio Engine could not be completely agile, as there is a time constraint in the form of the project deadline. This resulted in practicing certain agile methodologies while leaving out others. One agile methodology which enabled planning of Prio Engine was the use of tasks, stories, hills, and epics. A task is a small task, a story consists of many tasks to complete a larger challenge, a hill may have many stories and often refers to a project, and an epic is the combination of these and describes the product in its entirety. The use of tasks within Prio Engine allowed the developer to break up the required work into much smaller tasks, developers can then place importance and urgency on these tasks, and decide on a timescale for each task. This ensures that the crucial components of the engine are developed at the correct times.

The development of Artist Away while agile, was much smaller. This made it a simple development process where the developer could implement the core mechanics of Prio Engine, and then write the height map generation for Artist Away, furthermore Prio Engine could be adapted to accept this height map and the remainder of development and expansion could occur within Prio Engine.

### Testing Strategy

In an agile life cycle, development typically occurs in chunks and as a result so does testing. These chunks are known as iterations and are incredibly important to the success of continuous integration testing, as if the testing of an area is cut short the most important areas of each iteration can be prioritised to minimalize ineffective testing (Jiang & Chan, 2016).

In an ideal world, there would be a set of automated test cases which the engine must pass before progressing into the next iteration, however due to time constraints there is simply no available time for the developer to write a multitude of test cases. Instead, it proves more efficient for the developer to conduct manual test cases upon the engine and continuously test integration with Artist Away at the end of development in every iteration.

Bach (1998) outlines the methodology named ‘Good Enough Testing’ which is an alternative to exhaustive testing, and allows testers to determine when a solution has been deemed to pass testing. I believe ‘Good Enough Testing’ is both an important and effective methodology for the testing of Prio Engine and Artist Away, as the time constraints placed upon the project render exhaustive testing both impossible and ineffective. A period at the end of each iteration is dedicated to testing, with more important test cases prioritised. If minor bugs occur, the project will not be deemed to have failed testing, instead bug fixes will be placed on a development queue at low priority, if time allows towards the end of the project then bug fixing will occur.

## Requirements

### Selecting a Game Engine and Framework

The games industry provides are large list of existing engines to select from, however many of these engines prove not to be flexible enough and limit the scope of projects (Ali, 2016). Through careful planning, selected engines can be eliminated should they not meet the project requirements, however using an existing engine still prevents future flexibility of the project. Often in software development, there are requirements which go unforeseen at the start of a project, and later in the development lifecycle become more apparent. By creating a bespoke game engine, a project can be flexible in the future as the engine can be modify to suit project needs.

Direct X 11 has had overwhelming amounts of support from the development community since its release and has continued to thrive since the release of Direct X 12. This is primarily due to the complexity of Direct X 12 which makes it incredibly difficult to develop with, and is designed solely for experts. Microsoft recognised the complexity of Direct X 12, and released Direct X 11.3 in 2015 as a Direct X 12 alternative. As a result, Direct X 11 has continued to thrive throughout the release of Direct X 12, and has widely available support and resources, while supporting modern features such as geometric shading and tessellation.

Direct X 12 is also limited to Windows 10 platforms only. This drastically reduces the target market for any engine, and coupled with the complexity of the engine. Direct X 11 has been selected as the development framework of choice in this project due to its superior support and simplicity in comparison to Direct X 12.

### Creating a Game Engine

Prio Engine must be able to load, control and render models at 60 frames per second when using the minimum specification graphics processing unit (GPU) which is an NVIDIA GTX 960. Model control is the core mechanic behind games engines, however it is useless if it cannot render at a smooth and realistic speed.

The engine must be able to be exported as an external library, this will enable other applications to use it. Furthermore, the engine must be able to generate terrain from a height map provided by another application. Another requirement from the engine is that it must be able to control model position, rotation and scale and render them correctly, outside of the engine, and without being exposed to the complexity of the engine.

The height map must be generated by Artist Away and be stored in a two-dimensional array or a ‘.map file’, the engine will be able to read either of these methods. Artist Away must make use of the Simplex Noise algorithm to smoothly generate a realistic looking terrain shape.

The complete project must generate terrain which reflects a real-world environment not just in shape, but in appearance too. The terrain should be decorated with a variety of entities which result in the terrain looking more realistic, and less bland.

## Potential Solutions

### Implementation of a Class Hierarchy Based Engine

To plan Prio Engine, I started with a diagram of how I expected the engine to look: 

Figure 3‑1 Overview of Prio Engine Class Diagram

At a glance, it appeared this was a reasonably flexible solution that would account for future changes. For example, should I wish to change the engine from running on Direct X to OpenGL, theoretically, I should only be required to change the Direct3D class. However, this isn’t quite the case as there are some differences between the types of buffers and how they are used in Direct X and OpenGL. It also demonstrates that we could use it as a static library by providing methods for users through the Engine class, and allow the engine itself to handle the inner workings through a network of classes.

Another potential solution which was considered was an entity based system which would control all entities within the scene throughout an entity based manager. Overall, this may have been the better choice, however I believed it would not have brought enough benefit for the overall purpose of this game engine which is procedural generation of terrain to warrant implementation, it would only provide benefits in flexibility for future use.

### Procedural Generation

Different types of generators can create drastically different content and result in two incredibly different levels. It can be difficult to recognize and evaluate the difference in procedural generators (Horn, et al., 2014) and therefore careful planning of how the content will be generated is incredibly important.

Perlin noise is a very popular choice of noise used for height maps, however Simplex noise has now been released, which as described earlier is Ken Perlin’s new and improved version of Perlin noise. By creating a custom Simplex noise implementation, we can ensure that data generated is formatted correctly, and can be manipulated according to parameters which the user can define.

We can then allow Prio Engine to process the height map, create a terrain from the height map, and divide the terrain into areas according to their height in relevance to the rest of the terrain tiles. After dividing terrain tiles into areas, two things may occur; firstly, a relevant texture can be selected for that area, secondly, we can generate a chance to create an entity on that area of terrain. For example, a terrain tile may sit at 50% of the total height within the height map, the engine may decide this is a grass tile, the engine also then states each grass tile has a 5% chance to have a tree at this position. The engine can also analyse nearby tiles and take their properties into account. For example, if a tree exists in a tile within 10 units, then increase the chance to create a tree entity on this tile by 10%.

## Tools and Techniques

GitHub was used as a tool for source control, it enabled modifications to be submit and the ability to roll back to previous versions of the code base for both Artist Away and Prio Engine.

AntTweakBar was used to give users of Artist Away control over select variables used throughout the application. The use case diagram below details how users can interact with Artist Away while it is running, and provide custom parameters to artist away.



Figure 3‑2 Artist Away Use Case Diagram

ASSIMP was used to load a wide variety of models in different file formats into the engine. It is used within Prio Engine by parsing information about meshes and loading in the relevant files, and storing this information in member variables, before passing it over to buffers to be used with HLSL shaders. The class diagram below is an overview of how the mesh loading process works within Prio Engine.



Figure 3‑3 Model Class Diagram Within Prio Engine

## Legal and Ethical Issues

### Is It Ethical to Automate a Person’s Role?

As technology becomes more complex, users of technology are required to adapt to the latest versions of technology. In some scenarios this is perfectly acceptable, however in the modern-day games industry with the expectations of developers constantly rising, small independent companies simply cannot afford to spend time training their artists to use the latest technology and present it in the format required, as well as simultaneously developing the code behind the game. There is also no guarantee on how long it will take an artist to create a resource, due to a lack of reliability caused by human errors. According to (Neumann, 2016), we will continue to face these issues as they are a by-product of human labour. Automating the artistic process would result in a more consistent product being produced, as it programmatically executes a process, and there are very few variables involved in the production methodology. Given independent games companies are typically not renowned for being particularly wealthy, and are renowned for shutting down due to a lack of funds before games are released, it seems logical for smaller companies to risk hiring an artist to design levels, when a stable option of procedural generation can be used to complete the same process.

### Automation In Industry

As automation becomes increasingly popular and more attractive to employers, regulations and legislation regarding automation are being introduced. Unfortunately, we cannot predict what laws will be introduced in the future, however we can analyse the legislation introduced surrounding driverless cars, a popular form of automation. The Vehicle Technology and Aviation Bill imposed the idea that insurance companies which insured the automated vehicle would be responsible for reimbursing any damage caused by that vehicle. If we take similar logic and apply it to the process of procedural generation within games, we can rule out the requirement to pay damages as no insurance companies are involved, and no physical damage can come of a computer program which generates a game level. Because the company would be responsible for any mistakes within the game level, it would not matter if it were an artist or a computer program which designed the level.

While the legal side of automation is clearly not much of a barrier to the games development process, the ethical question remains: is it acceptable to remove an entire role from the industry to save money? There is no clear-cut answer to this question, and the answer is entirely situational and depends on the size, budget, and longevity of a games studio. This author believes it is only acceptable to automate a process when all other options have been exhausted, and it is simply deemed not feasible to hire an artist as a level designer. Business owners have a responsibility to provide careers to working class people, particularly larger business owners. This keeps the economy within the country stable, and provides workers with money to spend on business products.

### Legal Use of Third Party Tools and Resources

Prio Engine makes use of two third party libraries (AntTweakBar and ASSIMP), we firstly need to ensure that the licensing on these products allows for use within a final year project. AntTweakBar uses the zlib/libpng license, which claims the library is both free to use and redistribute, meaning that it is a completely eligible choice for use with Prio Engine and Artist Away. ASSIMP holds a 3 clause BSD license, the three clauses summarise to claim that the copyright notice must be displayed unmodified on binary and source files, and that ASSIMP cannot be used to endorse Prio Engine or Artist Away. Prio Engine can meet these conditions and therefore licensing of third party libraries has provided no issues.

Prio Engine makes use of textures and models, to ensure there were no legal issues Prio Engine has exclusively used free to use and royalty free resources.

## Potential Technical Issues

The first issue I face is keeping the game engine separate from the height map generation, yet keeping procedural generation itself within the engine. My design research resulted in the model where Prio Engine is a static library, which is included within Artist Away. This enabled us to generate height maps in two dimensional arrays, and pass them to Prio Engine as two dimensional arrays, from which Prio Engine could procedurally generate terrain. However, it did not seem feasible for future use to tie the generation of terrain to Artist Away, so a method was placed into Artist Away to export ‘.map’ files, which were rows and columns of floating point numbers which represent a height map. After implementing the ability to import ‘.map’ files in Prio Engine, and place the extracted values into a two dimensional array, we ended up with a method which meant Prio Engine could be run without any requirement for Artist Away, unless you wanted to modify the height map during runtime.

## Algorithms

Procedural generation can take many forms, but for an effective and smooth transition in height as is with terrain, some form of noise generation algorithm is required. (Mikuličić & Mihajlović, 2016) discuss the use of Perlin Noise to generate height maps, combined with Fractional Brownian Motion. This seems like a perfectly reasonable solution to generate height maps, however in 2001 Ken Perlin presented his improved version of Perlin Noise, known as Simplex Noise. According to (Perlin, 2002) Simplex Noise is quicker to generate than Perlin Noise, and in theory could be expanded into multiple dimensions rather than just two. While development of Artist Away has very little interest in the multiple dimension aspect of Simplex Noise, improving the efficiency of generating height maps is an important part of development. Simplex Noise was chosen as the more appropriate algorithm to use within Artist Away, as there was very little change in the complexity of the algorithm and implementation was worthwhile given the performance benefits.

## Summary

Write a short summary at the end of each chapter.

# Design

## Introduction

Prio Engine targets developers while Artist Away targets users of the game. Having two separate target audiences requires two separate design techniques, as Prio Engine is required to be somewhat complex, while Artist Away is required to be as simplistic and intuitive as possible.

Prio Engine and Artist Away are both written in C++ making use of object oriented methods to simplify how components within both projects interact with one another.

## System Design

### Game Engine Design

(Gregory, 2012, pp. 11-12) defines a game engine as a piece of architecture in which the artistic resources are separated from the core components such as the rendering system, collision detection system or sound system. From this definition, we can interpret that users of the engine will be responsible for loading of resources, but not implementing the code behind it. This is applicable to our engine as a user may want to create a terrain height map as a resource, and have the engine load this. We accommodate for this feature by accepting a map file which simply contains the same information that would be contained in the two-dimensional array generated by Artist Away.

The first step to designing Prio Engine was to research game engine architectures which would enable Prio Engine to be flexible and run with a variety of projects. Continuing from architecture research, a class diagram was drawn up to help visualise interaction throughout the engine (Figure 1).

The next design choice was to move the engine components into a dynamically linked library (DLL) or static library. DLLs in Visual C++ (Microsoft, 2015) describes a DLL as an executable file with shared functions and libraries. Whereas static linking involves exporting code as objects, and importing the functions and code through library objects. While DLLs are more efficient in terms of file size as they only load the required functions into memory, it made more sense to use Prio Engine as a static library as there is a whole range of optimisations that could occur by the optimising compiler in release mode of visual studio, which developers would be completely unaware of. It was deemed the benefits of using the engine as a dynamically linked library simply did not outweigh the security of using the engine as a statically linked library.

### Procedural Generation

## User Interface Design

Onal, et al. (2014) found user interface of a game can drastically impact the level at which a user will co-operate with a game, and can define the level of interest a user has for a game. As Prio Engine and Artist Away are split into two projects, two user interfaces must be defined. Prio Engine will be used by developers and therefore the engine class will be the interface, whereas Artist Away can be used by any none technical user, and as a result requires a more simplistic user interface.

### Prio Engine

Prio Engine is a code driven engine, and does not make use of scenes, demonstrated in larger engines such as Unity and Unreal Engine 4. The user interface for Prio Engine is designed for developers and not end users, the engine can be controlled through the engine class. The engine class may return certain elements such as a pointer to a mesh, so that the user can create models from this mesh, however for the most part the engine will oversee managing the scene and this will be hidden from the developer. Through the engine class interface, the developer will maintain the ability to initialise, shut down and get the time it took to process the last frame for the engine. These three parts of functionality are the core of Prio Engine; however, the developer will not have to implement everything they require the engine to do as a lot of functionality is already covered.

#### Predefined Shape Creation

Prio Engine makes use of predefining shapes such as cubes and triangles, they are incredibly useful when debugging code and requiring a model, instead of the programmer defining the vertices and adding render code, they just call the CreatePrimitive function from the engine class, and Prio Engine will handle the rendering of the predefined vertices. The vertices and indices are all defined in the ‘PrioEngine’ namespace, which can be found in ‘PrioEngineVars.h’. Users can also remove the shapes by a function named ‘RemovePrimitive’.

#### Terrain Creation

Terrain can be created and initialised through the engine class interface. Two methods exist which allow it to import a text file which contains heights where each element is separated by a space, and each row is separated by a new line. Terrain can also be imported through a two-dimensional array of type double. The terrain creation functions return a pointer to the terrain, so the user can modify the terrain elsewhere. The user also maintains the ability to update an existing terrain entity through the engine class.

#### Meshes

Meshes are loaded and destroyed through the engine class interface, the model creation for a mesh is done through the mesh object which is returned through the engine. By loading meshes through the engine, we can ensure that optimisation techniques through frustum culling are handled by the engine, and all models and meshes are rendered in the correct order.

#### 2D UI Images

Prio Engine also offers the ability to load in user interface images which will be rendered last, and therefore sit on top of everything else which is rendered in the window. These are particularly useful for showing UI elements like health, completion, and mana pools in a wide variety of games. The ability to create and remove these from the scene through the engine is an incredibly powerful tool to developers.

#### In Game Text

Prio Engine offers a method of displaying text in a game through Direct X 11, after the functionality was removed in Direct X 10, this can prove to be an incredibly time consuming feature to implement. It also allows text to be updated without having to destroy and recreate the text object, you must simply pass the new string you wish to display to the text object.

#### Skybox Environment

Developers can also update the skybox through Prio Engine to give a level a different feel. By default, the skybox will cycle through day, evening and night times. However, developers can disable automatic cycling, change the time between cycles and select a specific colour they want the skybox to be.

### Artist Away

#### Tweak Bar

AntTweakBar allows users to interact with the program and modify variables from within the executable during runtime. Prio Engine fully supports AntTweakBar usage, and demonstrates the use of AntTweakBar within Artist Away. Through AntTweakBar, users can modify the height map during run time, which also allows for a wider variety of terrains to be generated. Furthermore, the user is given control over the parameters used for Simplex Noise.

Using call-back functions, the tweak bar can access any of the Prio Engine class functions and execute them, allowing flexible demonstrations without the requirement to recompile, and an intuitive interface which is presented to the user on the screen. This interface presents a list of modifiable in game options to the user, and prevents the user requiring to read any documentation to understand how the project works.

#### Control

Camera movement within artist away represents the standard camera controls for most first-person shooter games, and role playing games throughout the industry, ‘W’, ‘A’, ‘S’ and ‘D’ keys are used to control the movement of the camera, while left, right, up and down keys control the rotation of the camera. These keys are standard throughout the industry, and therefore any user who is familiar with a wide variety of games will not be required to read through large amounts of documentation.

Some keys are bound to the function keys on the keyboard, such as F1 which is bound to toggle wireframe, and F2 which is bound to toggle full screen mode.

## Summary

The user interface for Prio Engine exists for developers and thus is much more complex, while Artist Away is designed for end users and therefore is much more simplistic. A combination of allowing developers freedom over engine use and placing power in the end user’s hands allows for an incredibly powerful and flexible user interface, with unlimited potential.

# Prio Engine

## Introduction

## Timing

## Layout

### Build Structure

### Usage

### Debugging Tools

#### Logging

#### Wireframe Mode

## Framework and Libraries

### Choice of Framework

### AntTweakBar

### ASSIMP

## Rendering Models

### Predefined Models

### Loading Models

### Rendering Models

### Rendering Terrain

#### Importing Height Maps

#### Calculating Terrain Type

#### Positioning World Entities

#### Updating Terrain

### Rendering Water

### Rendering a Skybox

#### Day, Night, and Evening Cycles

### Rendering Clouds

### Optimisations

#### Sphere Based Frustum Culling

## Lighting

## Rendering 2D Images

## Particle Systems

### Rain

## Summary

# Artist Away

## Simplex Noise Generation

## Creating and Exporting Height Maps

## Implementing User Interface

### Controlling the Game

### AntTweakBar Usage

## Concurrent Programming

### Multithreading Noise Generation

### Preventing Multiple GPU Instructions

## Updating Terrain During Run Time

### Generating New Height Map

### Destroying Old Entities

### Placing New Entities

### Updating Water

# Test Strategy

## Introduction

Each of your chapters should have an introduction to tell your readers what they will find in the chapter.

## Section

To add a caption to a table, either select the whole table (e.g. by clicking on the + symbol in the upper left corner of the table), right-click the it and choose ‘Insert Caption’ or click in any table cell and select ‘References’ -> ‘Insert Caption’ from the ribbon menu. Choose ‘Table’ as label and ‘above the item’ as position. Add the caption text in the box, separated with a dash as the example below shows.

Table 1 - Test Results

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## Section

Text goes here.

## Summary

Write a short summary at the end of each chapter.

# Evaluation, Conclusions and Future Work

## Project Objectives

Summarise what you have achieved.

## Evaluation

Stand back and evaluate what you have achieved and how well you have met the objectives. Evaluate your achievements against your objectives in section 1.2. Demonstrate that you have tackled the project in a professional manner.

(The previous paragraph demonstrates the use of automatic cross-references: The ‘1.2’ is a Cross-reference to the text in a numbered item of the document, it is not literal text but a field. The number that appears here will change automatically if the number on the referred-to section is altered, for example if a chapter or section is added or deleted before it. Cross-references are entered using Word's **Insert** or **References** menu. Cross-references are set to update automatically when printed, but may not do so on-screen beforehand; you can update a field manually on-screen by right-clicking on it and selecting Update field from the pop-up menu or by selecting the whole document and pressing F9.)

## Applicability of Findings to the Commercial World

Summarise what you have achieved.

## Conclusions

Summarise what you have achieved.

## Future Work

Explain any limitations in your results and how things might be improved. Discuss how your work might be developed further. Reflect on your results in isolation and in relation to what others have achieved in the same field. This self-analysis is particularly important. You should give a critical evaluation of what went well, and what might be improved.

## Concluding Reflections

Summarise what you have achieved.

# Bibliography

Ali, Z. (2016) *'A Framework for Game Engine Selection for Gamification and Serious Games',* San Francisco, IEEE, pp. 1199-1207.

Gregory, J. (2012) *'Game Engine Architecture',* 2nd edition, Natick: A K Peters, Ltd..

Hello Games, (2016) *'No Man's Sky',* Guildford: Sony Interactive Entertainment.

Jacobson, I., Spence, I. & Seidewitz, E. (2016) 'Industrial-Scale Agile--From Craft to Engineering', *Communications of the ACM,* 59(12), pp. 63-72.

Julian, T., Preuss, M., Beume, N. & Wessing, S. (2013) 'Controllable Procedural Map Generation via Multiobjective Evolution', *Genetic Programming and Evolvable Machines,* 14(2), pp. 245-277.

Microsoft, (2015) *DLLs in Visual C++,*   
https://msdn.microsoft.com/en-us/library/1ez7dh12.aspx  
(accessed 25 03 2017).

Microsoft, (2015) *Walkthrough: Creating and Using a Static Library (C++),*   
https://msdn.microsoft.com/en-us/library/ms235627.aspx  
(accessed 25 03 2017).

Mikuličić, N. & Mihajlović, Ž. (2016) *'Procedural Generation of Mediterranean Environments',* Opatija, Information and Communication Technology, Electronics and Microelectronics (MIPRO).

Neumann, P. G. (2016) 'Risks of Automation: A Cautionary Total-System Perspective of Our Cyberfuture', *Communications of the ACM,* 59(10), pp. 26-30.

Onal, E. et al. (2014) *'Decision-making in abstract trust games: A user interface perspective',* New Orleans, Cognitive Methods in Situation Awareness and Decision Support (CogSIMA).

Pereira, D., Zárate, L. & Song, M. (2013) *'Applying Formal Concept Analysis to Assist Class Hierarchy Construction in Biomedical Systems',* Hangzhou, IEEE.

Perlin, K. (2002) 'Improving Noise', *ACM Transactions on Graphics,* 21(3), pp. 2-3.

Appendix 1 – Title of Appendix

You may have one or more appendices containing detail, bulky or reference material that is relevant though supplementary to the main text: perhaps additional specifications, tables or diagrams that would distract the reader if placed in the main part of the dissertation. Make sure that you place appropriate cross-references in the main text to direct the reader to the relevant appendices.

Do not blindly include all of your code in the appendix or the body. Only include the parts you refer to in the report. You can but those parts either in the appendix or in the body (e.g. in the “Implementation” part).

Appendix 2 – Title of Appendix

You may have one or more appendices containing detail, bulky or reference material that is relevant though supplementary to the main text: perhaps additional specifications, tables or diagrams that would distract the reader if placed in the main part of the dissertation. Make sure that you place appropriate cross-references in the main text to direct the reader to the relevant appendices.

Do not blindly include all of your code in the appendix or the body. Only include the parts you refer to in the report. You can but those parts either in the appendix or in the body (e.g. in the “Implementation” part).

Appendix 3 – Title of Appendix

You may have one or more appendices containing detail, bulky or reference material that is relevant though supplementary to the main text: perhaps additional specifications, tables or diagrams that would distract the reader if placed in the main part of the dissertation. Make sure that you place appropriate cross-references in the main text to direct the reader to the relevant appendices.

Do not blindly include all of your code in the appendix or the body. Only include the parts you refer to in the report. You can but those parts either in the appendix or in the body (e.g. in the “Implementation” part).