2018

Evolving gameplay elements into virtual terrains

Andrew Pech
Edith Cowan University

Edith Cowan University

Copyright Warning

EDITH COWAN UNIVERSITY (ECU)

DOCTORAL THESIS

Evolving Gameplay Elements into Virtual Terrains

This thesis is presented for the degree of

Doctor of Philosophy (Computer Science)

Andrew Pech

Edith Cowan University

School of Science

2018

EDITH COWAN UNIVERSITY (ECU)

DOCTORAL THESIS

Evolving Gameplay Elements into Virtual Terrains

Author:
Andrew Pech

Supervisors:
Dr. Martin Masek
Associate Professor Chiou Peng Lam
Associate Professor Philip Hingston

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy

in the

School of Science

August 2018

Declaration of Authorship

I, Andrew Pech, declare that this thesis titled, 'Evolving Gameplay Elements into Virtual Terrains' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself

Signed: 

Date: *30/08/2018*

Abstract

With advancements in technology, consumers are expecting higher quality and more detailed video game content. This puts a strain on video game companies and their developers as they are required to manually design and create increasingly complex video game content. Procedural content generation can alleviate this burden by using technology to automatically generate video game content, effectively reducing development time and budget. This thesis presents a novel approach towards procedurally generating video game terrains that meet a set of gameplay requirements as specified by the user. This approach uses a genetic algorithm to evolve a set of modifications that, when applied to a user-specified terrain, incorporates the desired gameplay elements. This approach can aid developers by reducing time spent manually creating and editing video game terrains.

An important aspect of this research involved designing a set of measures capable of characterising gameplay elements. A collection of isovist and graph-connectivity measures were discovered that can characterise different types of gameplay elements so that they can be automatically identified in a video game terrain. This set of measures may be useful in other procedural methods and related fields.

List of Publications

Pech, A., Hingston, P., & Masek, M., Lam, C.P. (2016a). *Identifying Attributes for Characterizing Game Area Types in Virtual Terrain*. Paper presented at the 1st Joint Interest Conference of DiGRA and FDG, Dundee, Scotland. (Includes work described in Chapter 4)

Pech, A., Lam, C.P., Hingston, P., & Masek, M. (2016b). *Using Isovists to Evolve Terrains with Gameplay Elements*. Paper presented at the Evostar 2016 conference, Porto, Portugal. (Includes work described in Chapter 6)

Pech A., Hingston P., Masek M., Lam C.P. (2015) *Evolving Cellular Automata for Maze Generation*. In: Chalup S.K., Blair A.D., Randall M. (eds) Artificial Life and Computational Intelligence. ACALCI 2015. Lecture Notes in Computer Science, vol 8955. Springer, Cham.

Pech A., Masek M., Lam C.P., Hingston P. (2016) *Game Level Layout Generation Using Evolved Cellular Automata*. Connection Science, 28(1), 63-82. Doi:10.1080/09540091.2015.1130020.

Acknowledgements

This section is an acknowledgement and a thank you to all the people who made completion of this thesis possible. First of all my supervisory team, Martin Masek, my principal supervisor, who sacrificed much of his time to help me in my research and in the reviewing of this thesis, even with his two children, Tony and Francis, being born during this time; my two co-supervisors, Chiou Peng Lam and Philip Hingston, who taught me a lot of what I needed to know to complete this research. I would also like to acknowledge and thank my family for supporting me, emotionally and financially, during this time.

Table of Contents

List of Figures

Table of Tables

Table of Abbreviations

Abbreviation	Meaning
GA	Genetic Algorithm
IEC	Interactive Evolutionary Computation
PCG	Procedural Content Generation
PNN	Probabilistic Neural Network
RTS	Real-Time Strategy
ADS	Area Type Data Set
asm	Area Similarity Measure

Chapter 1. Introduction

1.1 Overview

Video games have come a long way in the last fifty years, initially being entirely coded by a single person who was also responsible for all of the game's content creation. As hardware has evolved and improved, so have video games. Due to the increased complexity of video games, they often incorporate pre-written components, such as physics engines; to reduce the time spent developing them. Common non-code based components of a video game include, 3D models, textures, particle effects, sky boxes or sky domes, and terrain. This content is typically unique for each game, making the use of pre-generated content unsuitable. Therefore the generation of this content, in comparison to the programming component of game development, requires more time to produce and a larger portion of the development budget.

Terrain is one of the most important components of video games that feature exterior environments, as it represents the layout of the environment (i.e. the level layout) and contributes significantly to the aesthetics and realism of the game. It is also often given a significant portion of the time required to draw the game world to the computer screen. Terrain is typically defined as the shape of the ground, excluding natural features such as trees and rocks, which are instead referred to as geometry, which is, in most cases, textured 3D models. Due to its importance, terrain generation tools have been developed that aid developers in the creation of game terrains. Applications such as Artifex Terra 3D (Gradl, 2008), and EarthSculptor (Szoka, 2004), help create terrain by providing brush tools that alter the height of the painted section of terrain.

Procedural content generation (PCG) is the algorithmic generation of media content and can produce content more efficiently than a developer. PCG has been around since the 1980's when it was originally used to reduce memory requirements of early video games to accommodate the hardware of the time. An early example of such a game is "Rogue" (Toy & Wichmann, 1980) which used PCG to generate random dungeon environments to continuously present the player with new dungeons to explore. Since "Rogue", other games such as NetHack (The NetHack DevTeam, 1987), Moria (Koeneke & Todd, 1994), and Diablo (Blizzard North, 1996) have used similar PCG techniques. As computer hardware advanced it was able to support larger memory requirements, therefore motivation for using

PCG techniques stemmed less from memory conservation and more from increasing replay value in video games, since a single PCG technique could produce a near infinite variety of a particular type of content. Now PCG has found another important purpose, to reduce the time developers spend on content creation.

PCG has taken terrain generation further, allowing designers to generate entire terrains automatically in a fraction of the time it would take to manually create them. Now there are terrain generation tools that utilise PCG such as World Machine (Schmitt, 2008) and L3DT (Bundysoft, 2018), which incorporate parameterised PCG methods to generate base terrains, and offer further manual editing through the use of brushes and other tools as base terrains typically lack suitable areas for gameplay and do not meet any desired game level design requirements.

Although PCG methods and tools have been developed that are capable of generating terrains for video games, the majority of these methods do not consider the automatic inclusion of gameplay elements, requiring developers to manually edit the generated terrains to make them game-ready. The term 'gameplay elements' is not unique to this thesis, with other sources using this term to describe various aspects of a video game, from player controls (Epic Games, 2012) to physical objects that are placed in the game environment (IGN Entertainment, 2018).

Various gameplay elements described in existing literature include sniper locations, galleries, choke points, arenas, strongholds, turrets, vehicle sections, hidden areas, and flanking routes (Hullett & Whitehead, 2010). Sniper locations, galleries, and choke points are ideal for setting up ambushes, while large scale battles can be staged in arenas and strongholds. Turrets and vehicle sections allow for alternate gameplay styles to keep gameplay interesting. Hidden areas and flanking routes both promote exploration. Hidden areas do this by providing areas to hide items that may be useful to the player, while flanking routes provide discrete pathways that can allow players to avoid enemies or give them a tactical advantage. Although many gameplay elements have been described in the literature, an extensive search revealed that while there are many terrain generation methods capable of generating visually pleasing terrains, only few terrain generation methods attempt to automatically incorporate gameplay elements. The few existing terrain generation methods that do incorporate gameplay elements, for example, Olsen (2004) and Togelius *et al.* (2010), typically focus on

accessibility of the terrain, with some work in done in the real-time strategy (RTS) genre including placement of player bases and resources on a terrain.

1.2 Statement of the Problem

Traditionally when creating terrain for a video game, designers are required to make manual edits to the height values of a terrain to incorporate the desired gameplay elements. The designer must also manually design the layout of the gameplay elements, so that all desired constraints between them are met. For example, making sure that there is line-of-sight between two areas, while also ensuring that maintaining this constraint does not violate other constraints or negatively impact other incorporated gameplay elements. There are many solutions to a terrain in terms of desired gameplay elements and their associated constraints, as there are many arrangements of gameplay elements that can meet a set of designer requirements. Therefore, a tool that could automatically generate terrains that meets a set of user-specified requirements and presents the designer with a collection of viable terrains to choose from would be valuable.

However, there is a gap in the current state of the art in generating virtual terrains that incorporate gameplay elements, especially from the perspective of automatically generating terrains that are both aesthetically pleasing and that also incorporate user-specified gameplay elements. In this thesis, the term 'gameplay elements' refers to physical game design aspects, including areas that are designed for a specific gameplay purpose and their configuration within the terrain. These gameplay elements are a vital component of video games as they dictate the physical layout of the game environment, which is an important aspect as it can force players to experience events at specific locations in a chronological order. Although dated, Olsen (2004) and Frade *et al*. (2010) is still some of the most relevant work in this field and attempt to address the gap by focusing on the generation of terrains with a desired amount of accessibility. In these works, accessibility is defined by the gradient of the terrain, where a section of terrain is considered accessible if its gradient is below a set threshold. This is important for video games, as most games that include a virtual terrain require flat areas for players to traverse or to place buildings.

However, most video games would require more than just having accessible areas. They require the accessible areas to conform to specific designs so that a player's progression through the game world can be predicted. This allows strategic placement of story elements, ensures that the player experiences these elements in the correct order. Work by Dormans (2010), Dormans

(2011), Hartsook *et al*. (2011), and van der Linden *et al*. (2013) address this by generating level layouts that conform to stories or missions, but these methods do not generate the associated terrain, thus leaving this an open problem to be addressed using terrain generation methods. Andereck's (2014) work is a step towards this goal, generating terrain that conforms to a desired path. This allows designers to manually design a path, specifying where players can go, and then generating the terrain around the path.

Togelius *et al*. (2010) have also addressed a few other gameplay elements, such as resource placement and player base placement, for the RTS genre of games that typically involve players collecting resources and using them to enhance their base and build an army to dominate other players. Therefore, aside from accessibility and some limited work in the RTS genre, the problem of incorporating gameplay features into virtual terrain is largely unaddressed.

On the other hand, terrain generation methods typically use fractal techniques as they are efficient and produce realistic looking terrains. However, these techniques are not well suited when attempting to automatically generate a terrain that incorporates user-specified gameplay elements. Evolutionary approaches, such as genetic algorithms (GA), are better suited towards this goal, as such approaches are capable of generating the desired content in the absence of a known set of instructions for incorporating gameplay elements into virtual terrains. Very limited work has been done in evolving terrains, such as the work by Togelius *et al*. (2010) that evolves terrain with desired locations for player bases and resources. However, their work uses a simple representation that is unable to represent terrains with more advanced gameplay features. Therefore evolving specific gameplay elements, such as hidden areas and strongholds, into a terrain is still an open problem.

There are two key problems associated with evolving gameplay elements into a terrain. The first problem is the representation of the gameplay elements that are to be evolved. Most existing work in evolving gameplay elements into game environments uses a representation that encodes the entire game environment, rather than representing gameplay elements and integrating them into a user-specified environment. In these cases, gameplay elements are enforced by the fitness function, evolving the environment to accommodate the desired elements. An example is the work by Ashlock *et al*. (2011) where they evolved 2D, maze-like environments. The representations used in this work encode entire maze-like environments and uses the chosen fitness functions to ensure desired paths are formed throughout the maze. Encoding an entire terrain would result in a massive search space unless a low-resolution representation is used, for example, the representation used by Togelius *et al*. (2010), which results in much detail being

lost. This makes representing modifications to an existing terrain a suitable option but introduces the additional problem of integrating evolved modifications into an existing terrain.

The second problem is developing a method that can evaluate how well the user-specified gameplay elements suit their desired purpose. Existing works typically use specialised fitness functions to evolve game environments with specific gameplay elements, such as the work done by Ashlock *et al*. (2011), where one fitness function generates mazes with a desired exit path length and another generates mazes with a desired number of culs-de-sac. These methods of evaluation are not extensible, requiring a new fitness function to be developed for each type of gameplay element. Additionally, there are many gameplay elements that have not been considered in evolutionary PCG approaches (i.e. hidden areas and sniper locations), and therefore no known methods are available for evaluating them.

This introduces another problem. To automatically evaluate gameplay elements that are associated with areas on a terrain, for example a hidden area, a method must be developed that is capable of automatically characterising them using a set of quantifiable measures. For instance, the fitness function used in Frade *et al*. (2010) evaluates a generated terrain based on its accessibility; therefore the accessibility of the terrain must be quantified. This is done by calculating the gradient of the terrain at uniformly spaced locations and comparing the calculated values against a gradient threshold to determine if the terrain is accessible at each location. Similarly, to determine if an area is suitable as a hidden area, characteristics of the area that determine how hidden the area is must be quantified. Considering the limited work done in evolving terrains that incorporate gameplay elements, and the lack of quantifiable measures capable of characterising these elements, this is an open problem for any evolutionary approach that attempts to incorporate gameplay elements into a terrain.

1.3 Purpose of the Study

The aim of this thesis is to develop a method of automatically generating terrains that incorporate gameplay elements specified by the game designer who uses this method (the user). The proposed approach allows game designers to specify a set of gameplay elements in a configuration to be evolved into their chosen terrain. The gameplay elements used in this research include five types of areas, where each area type specifies a gameplay purpose for a section of terrain. These area types are hidden area, open area, vantage point, choke point, and stronghold. Additionally, the user of the method may specify constraints between areas to control other physical aspects of the terrain.

A GA-based approach was chosen to address this aim, as there is no known set of instructions that can automatically incorporate gameplay elements into a terrain. Local search techniques, such as hill climbers (Davis, 1991), are not suitable for this approach as they require linear, continuous, non-noisy fitness landscapes, while GA can efficiently locate optimal areas of vast and complex search spaces (García-Martínez & Lozano, 2008), such as the search space of this approach. While local search techniques may be more accurate at discovering optimal solutions in a local space, there is no one optimal solution to level design and therefore GA meets the needs of this approach.

Research Question: How can a GA-based approach be used to produce a set of modifications which, when applied to a user-specified terrain, will generate terrains which incorporate user-specified gameplay elements? The sub-questions that guided this research include:

- **Sub-Question 1:** How can gameplay elements and their associated constraints be represented, such that they can be modelled as a solution in a GA-based approach?
- **Sub-Question 2:** How can a modified terrain be evaluated in a fitness evaluation approach to evolve a set of modifications that captures specified gameplay elements and corresponding constraints?
- **Sub-Question 3:** How can an area (gameplay element) in a terrain be characterised as a set of measures, so that these gameplay elements can be automatically evaluated to evaluate their suitability for their intended gameplay purpose?
- **Sub-Question 4:** How can a set of modifications be incorporated into an existing terrain?

1.4 Contributions of this Study

The key contribution of this thesis is the development of an automatic approach that allows a game designer to specify a configuration of desired gameplay elements and a terrain, in which evolved modifications will incorporate the designer's specifications. Most existing methods that generate terrains focus on aesthetic qualities, such as the appearance of rivers, mountains, and canyons, while few terrain generation techniques are designed to incorporate gameplay elements. The existing techniques that incorporate gameplay elements focus on either accessibility or gameplay elements related to the RTS genre. This thesis presents a novel approach towards generating terrains that incorporates desired gameplay elements by altering a user-specified terrain to form areas suitable for specific gameplay purposes. It does

this by quantifying areas in the modified terrain, using a range of measures from the fields of architecture and graph theory, to evaluate these areas based on the gameplay features they afford. This approach is not limited to a specific game genre and can easily be extended to incorporate additional gameplay features.

While the developed approach is applicable to content generation in computer games, these generic techniques can also be applied to terrain design for physical landscapes as well as terrains for simulation, visualisation, and scenario modelling purposes. Additional contributions made during the development of this approach include:

- **A representation for terrain modifications that captures gameplay elements and their associated constraints.**
  With limited work in the field of evolving terrains, representing terrain modifications that are capable of capturing gameplay elements is not a well-studied area. An evolutionary approach proposed by Frade *et al*. (2010) evolves terrain generation programs, where a terrain generation program is an equation that generates height values for a terrain height map. The goal of the generated programs is to be able to generate a range of terrains, each with a desired amount of accessibility. Although Frade *et al*.'s (2010) method uses an evolutionary approach towards generating terrains, the representation used is a mathematical equation for generating entire height maps.

  Togelius *et al*. (2010) also uses an evolutionary approach towards generating terrain with desirable locations for placing gameplay elements such as player bases and resources. Their work represents an entire terrain as a list of x and z coordinates and height values, where each x and z coordinate represent a location on a flat terrain to create a hill, where the height of the hill is specified by the coordinates associated height value. Although this is a computationally efficient representation, terrains generated using this representation lack detail and realism.

  The approach presented in this thesis uses a graph structure to represent terrain modifications, as graphs are capable of capturing the level layout topology of a terrain. Each node of the graph contains modification information capable of forming an area suitable for a specific gameplay purpose when applied to an existing terrain.

Each edge contains modification information that controls the ruggedness of the terrain between areas. This representation is well suited for evolutionary approaches as it is computationally efficient to use, while also incorporating detail from the original terrain. This representation also has uses outside of evolutionary approaches. Due to its small memory footprint it can be used to reduce storage requirements for applications that require many terrains with different layouts.

- **A method of integrating specific modifications into an existing terrain.**
  As there is limited work in evolving terrain, and representing modifications for them, there were no methods of applying modifications to an existing terrain suitable for this approach. Chapter 3 of this thesis presents the developed method of incorporating a set of modifications, encoded in a graph structure, into an existing terrain. Applying these modifications to an existing terrain not only forms desired gameplay elements, but also shapes the level layout of the terrain.
- **Extending knowledge of characteristics that are useful in identifying gameplay elements.**
  A collection of isovist and graph-connectivity measures are presented in this thesis that are capable of characterising gameplay areas. Many of these characteristics are new to the field of PCG. The ability to obtain quantitative measures is important for any form of objective evaluation. These characteristics are used in this approach to evaluate the similarity of gameplay elements to specific types by comparing the characteristics of areas in a terrain to the characteristics of existing area samples in order to obtain a similarity measure. Although this approach used these measures to characterise areas in an exterior, terrain environment, they are also applicable to indoor environments.
- **Introduction of a novel method of estimating isovist volume that is efficient and more accurate than existing methods.**
  As part of this research, a method of calculating the three-dimensional volume of visible space, known as an isovist, was required. Existing methods, such as Rayburst sampling (Rodriguez *et al*., 2006), are too time consuming for approaches that are computationally expensive, as is the case in the developed method where is required a large number of volume estimations to be completed. Therefore a novel method of estimating the volume of an isovist was developed that takes advantage of the chosen

isovist representation to be more efficient than Monte Carlo (Metropolis & Ulam, 1949) and Rayburst, and generally more accurate.

1.5 Significance

- This study expands knowledge in the field of PCG, specifically in terrain generation and the automatic incorporation of gameplay elements into terrain.
- The developed approach can automatically generate terrains with desired gameplay elements, reducing the workload of game developers and thus reducing development costs. The gameplay elements used in this approach are commonly found in the FPS genre of games, making this approach particularly useful for games of this genre. However, these elements have uses in other genres, for example, a puzzle or exploration game may use a vantage point to view the layout of a level so that a player has an idea on how to get to their destination. Another example is open areas, they may be used in RTS games for placement of player bases, or in role playing games (RPG) to place buildings or small towns.
- The developed approach is significant for other terrain applications that require areas with specific characteristics to be identified or incorporated. For example, military mission planning requires terrain analysis for appropriate setup of their forces, such as taking advantage of choke points in the terrain (Glinton *et al.*, 2004). An approach to identify areas in a terrain would be useful for autonomous agents, such as non-player characters (NPCs) in RTS games, as agents need ways to evaluate their surroundings to make intelligent decisions (Perkins, 2010).
- The novel isovist volume estimation method can aid in other applications where isovists are used, namely in the field of architecture where they are used in the analysis and design of urban spaces.

1.6 Structure of Thesis

This chapter presented an overview of the problem addressed in this thesis along with the purpose, contributions, and significance of the study. The following chapters are organised as follows.

Chapter 2 is a literature review and is divided into two main sections; 1) “State of the Art” describes the work that has been done in PCG, more specifically in generating terrains, and

outlines the gaps in knowledge. 2) “Technical Review of Genetic Algorithms” details GA, their components, and how they work, as GA is a key component of this approach.

Chapter 3 describes the approach of generating terrains by integrating gameplay elements into an existing terrain. This chapter describes how to generate four components that are required to generate the final terrain, and how these components are combined to create the final terrain.

Chapter 4 lists the collection of graph-connectivity and isovist measures used to characterise areas in this approach and details how they are calculated. This chapter also contains details on the novel isovist volume estimation method and how it is calculated.

Chapter 5 describes a study in which the graph-connectivity and isovist measures, detailed in Chapter 4, are used to characterise specific gameplay areas of existing video game levels. These characterised areas are then used to train a collection of classifiers that are able to determine the type of gameplay area based on the collection of graph-connectivity and isovist measures.

Chapter 6 describes the approach and the main contribution of this research, the evolution of a terrain to incorporate gameplay features. This chapter describes the inputs followed by the steps of the approach. This covers the representation used for the GA-based approach, how the population is initialised, the fitness function, how the population is replaced each generation, and the conditions that must be met before the GA-based approach terminates.

Chapter 7 lists the parameter settings that were used in the evaluations of the developed approach for this research. It details the parameter tuning experiments that were run using three well known parameter sets and compares their results.

Chapter 8 details the experiments, and their results, that were conducted to evaluate the approach developed in this research. This chapter is divided into four main sections. The first three of these sections each detail a set of experiments that attempted to generate terrains with desired gameplay elements and constraints. These sets of experiments are grouped based on their complexity, with the first section detailing the simplest experiments and the third section detailing the most complex. The fourth section explores the impact that different initial terrains have on the resulting generated terrain.

Chapter 9 summarises the main findings of this research and how the aims of this project were addressed. This summary is followed by a list of possible avenues for future research.

1.7 Summary

This chapter provided an overview of the problem that this research addresses along with the purpose, contributions, and significance of this study. The next chapter details the state of the art and details the components of genetic algorithms, which are used extensively in this research.

Chapter 2. Literature Review

This chapter provides a review of existing published work that is relevant to this thesis. The chapter is divided into two sections. Section 2.1 describes existing PCG techniques that focus on terrain generation, from aesthetic terrain generation to methods that incorporate some form of gameplay elements. Section 2.1 also outlines the short-comings of these methods and the gaps in knowledge in this area of research. Section 2.2 describes concepts associated with Genetic Algorithms (GA) which is used in the approach presented in this thesis.

2.1 State of the Art

PCG has many uses in game development from generation of game resources (i.e. textures and meshes) to the generation of entire game levels, with generated layouts and objects being automatically placed around the scene. Originally, simple PCG techniques were used to generate random dungeon layouts for a game called "Rogue" (Toy & Wichmann, 1980) and a genre of games thereafter referred to as "Rogue-likes" ("NetHack", 1987; "Moria", 1994; "Diablo", 1996). There have been many contributions to this field of research due to the potential of PCG in the automation of game development. PCG techniques have evolved to be capable of generating level layouts that meet specific gameplay requirements such as generating desired player paths or designing levels based around a story or series of specified events. Examples of these include the work of Ashlock *et al*. (2011) that used a GA to evolve maze-like level layouts. This method provides many adjustable parameters to control how the path through the level is generated such as the path's length, how often it branches, and how often it reconverges. The use of control points gives further control over how this path is generated. Other techniques produce grammars that are capable of generating game levels that meet the criteria of a given story or action graph (Dormans, 2011; Hartsook *et al*., 2011; van der Linden *et al*., 2013). Levels generated using these approaches contain a series of areas that are set out in the logical order for game or story events to unfold.

Advancements in technology have allowed the creation of games that feature open worlds, with expansive exterior environments and detailed interior settings. These games have become increasingly popular and therefore PCG techniques have been created that aid in the generation of these virtual worlds. For interior environments, there are techniques that can generate entire house or building layouts (Lopes *et al*., 2010). These interior levels can then be furnished with other techniques that place objects such as tables, chairs, and even dinner plates and cutlery (Tutenel *et al*., 2009; Taylor & Parberry, 2011). For exterior environments

there are techniques that can generate roads through an existing terrain following an optimal path based on a set of cost functions including terrain slope and the presence of bodies of water (Galin *et al*., 2010). Other techniques can generate entire road networks, villages, and even cities (Parish & Muller, 2001; Emilien *et al*., 2012). The largest and one of the most important components of an exterior environment is the terrain and therefore many techniques that generate terrain have been developed.

Existing terrain generation techniques can be classified as physics-based, example or sketch-based, and procedural. Most of these techniques focus solely on the visual appearance of the terrain without considering how the terrain might be used for gameplay. This means that terrains generated using these methods require post-editing to make them suitable for use in video games. There are few terrain generation techniques that do consider some form of gameplay, but these techniques are usually quite limited as to what gameplay elements they can incorporate. The rest of this section describes existing terrain generation methods and is divided into three categories; Aesthetic-Based Terrain Generation, Semi-Automated and Sample-Based Terrain Generation, and Gameplay-Based Terrain Generation. Aesthetic-Based Terrain Generation methods are fully automated approaches that focus on visually pleasing and realistic looking terrain features, but do not incorporate any gameplay considerations. Semi-Automated and Sample-Based Terrain Generation methods range from tools that aid developers in creating terrains, to methods that piece terrains together from existing terrain samples. These methods do not directly incorporate gameplay elements, but allow developers to manually include them. Gameplay-Based Terrain Generation methods are approaches that consider gameplay in some form, such as ensuring the generated terrain has a suitable amount of accessible area and high connectivity for player traversability.

2.1.1 Aesthetic-Based Terrain Generation

Belhadj & Audibert (2005) introduced a method of terrain generation using particles that moved in fractional Brownian motion over the terrain, tracing outlines of rivers and ridges and setting the terrain's height as they move over it. Once the outlines are set, a form of midpoint displacement is used to set the heights of the terrain between the outlines. This method results in a realistic looking fractal terrain. The user can control the appearance of the generated terrain by adjusting the movement behaviour of the particles, but this affects the terrains aesthetic properties only.

Using a similar technique to Belhadj & Audibert (2005), Doran & Parberry (2010) introduced the use of software agents that adjust the height of the terrain as they move all over it. Their technique utilises different types of software agents, with each type of agent being responsible for generating a specific feature of terrain, such as rivers, hills, mountains, and beaches. For example, the mountain agent moves over the terrain, raising it in V-shapes and periodically changing direction to form zig-zagged mountain ranges. In addition, each agent type has a number of configurable parameters that allow a user to tune the agents' movement patterns and alter the way they shape the terrain. Again, the one focus of this method is on generating realistic natural features of the terrain, and does not consider the inclusion of any gameplay elements.

Kamal & Uddin (2007) proposed a method of generating mountains or craters at user-specified locations in an initially flat terrain. This method partitions the terrain into a collection of polygons and then, starting at an initial polygon, performs a probing process which results in a subset of polygons, including the initial polygon and random adjacent polygons. This subset of polygons are then raised (or lowered if creating a crater) by a small amount. This process is repeated a number of times, with each iteration dividing the subset of polygons into more, smaller polygons, selecting a number of them via probing, and raising/lowering them by a small amount. The initial polygon selected for probing is always the polygon that encloses the desired mountain/crater location to ensure it gets generated at the specified location. This technique gives finer control over where terrain features are placed but lacks a comprehensive collection of features and does not consider gameplay.

De Carli *et al*. (2014) proposed a method of modifying an existing terrain by using a combination of mean shift clustering and image processing to create canyons. This research used Perlin noise (Perlin, 1985) to create initial terrains and created layered terrain plateaus by equalising the height values of sections of terrain that have similar elevation intensities, which are discovered by using the clustering algorithm. This is another technique focused on aesthetic terrain qualities.

Another approach focused on the generation of realistic terrains was proposed by Zhang *et al*. (2016). This method uses an L-system to generate self-similar river networks and a Voronoi diagram to create ridge networks. Once the river and ridge networks have been created, midpoint displacement, inverse midpoint displacement, and diffusion are used to fill out the rest of the terrains elevation data.

A physics-based method of terrain generation was introduced by Cordonnier *et al*. (2016), which simulates tectonic uplift and stream powered erosion to generate realistic terrains. This approach requires a grey-scale uplift map as input, which represents the speed at which the terrain is raised by tectonics. From this input a stream-graph is generated and the stream-powered erosion is applied, creating realistic-looking terrains.

Génevaux *et al*. (2015) presented a novel method of representing a terrain as a tree structure, where leaf nodes represent terrain features such as rivers and mountains as parametrised skeletal primitives. The inner nodes of the tree determine how the leaf nodes are combined to form the final terrain. For example, some features can be carved out of others, while other features can be blended together.

A collection of techniques have also been proposed that take a basic sketch of a terrain as input and outputs a detailed version of the terrain (Rusness et al., 2009; Golubev et al., 2016; Hnaidi et al., 2010; Michel et al., 2015). Most of these approaches allow the user to sketch a terrain as a series of lines that represent ridges, roads, or rivers, and use various algorithms to generate the terrain around them. Rusnell *et al*. (2009) uses Dijkstra's pathfinding algorithm to get the distance between points on the terrain and the sketched ridges, and uses these distance values to blend the ridges into the terrain. This method was later improved upon by Golubev *et al*. (2016) by using different weight functions when calculating terrain heights and reducing the execution time of the algorithm. Hnaidi *et al*. (2010) uses a diffusion method to blend ridges, roads, and rivers into the terrain, and offers users some parameters, such as terrain elevation and roughness, to further control how the sketched terrain features are formed. The method proposed by Michel *et al*. (2015) takes a vector map as input, rather than a sketch of terrain ridges, that indicates coast contours, main rivers, and mountain peak distributions. This vector map is used to calculate possible continental plates from which a fold map representing mountain ranges can be obtained. The vector map is also used to calculate watersheds to add rivers and other bodies of water.

The primary downside to these sketch-based approaches is the amount of input required from the user who must design where every river, mountain ridge and road must go as well as specifying additional parameters such as the roughness and elevation of the terrain for each of the sketched features. Due to the level of input required these techniques may be better suited as tools to aid in terrain creation/editing rather than stand-alone terrain generation techniques.

2.1.2 Semi-Automated and Sample-Based Terrain Generation

There are several tools that incorporate PCG techniques to aid in terrain generation, such as the tool proposed by Gain *et al*. (2009). This tool uses an interactive, sketch-based procedural terrain generation method to aid users in creating terrains. Users are able to draw feature silhouettes in real-time and the terrain will update with these drawn-in features. Gain *et al*. (2015) proposed a similar terrain authoring tool that allowed users to add constraint points and curves to the terrain that they could then modify to deform the terrain in real-time. Users can also paint sections of the terrain as a specific terrain type, such as mountains, and the tool will use existing terrain exemplars to fill out the elevation data of that section of terrain.

Cristea & Liarokapis (2015) created a tool that combines fractal and physics-based PCG methods to produce realistic terrains. This tool is called "Fractal Nature" and uses a fractal PCG method, inspired by the diamond-square algorithm, to generate a base terrain, which can then be updated by applying thermal and hydraulic erosion simulations.

Other tools that are heavily based on physical simulations include Peytavie *et al*.'s (2009) approach that offers high level terrain authoring tools while using physical simulations to automatically stabilise layers of sand and rock to keep the terrain looking natural and detailed. For example, the user is able to add sand to the terrain and have the physical simulation use the sand to automatically fill crevices, or the user may cut out a section of cliff face and the physical simulations erode the terrain and place rock piles where the terrain has crumbled.

Cordonnier *et al*. (2017) introduced another physics-based terrain generation method that represents the terrain using a discrete layered model. The layers in this model collectively represent the state of the scene at each frame and are updated at each time step by geomorphological and ecological events such as rain, gravity, temperature, wind, fire, and lightning. Users can interactively alter the terrain generation process by 1) adjusting environmental parameters such as rainfall patterns, and frequency of lightning strikes 2) editing the layers directly using a collection of brush tools 3) altering the chances that geomorphological events will occur such as forcing lightning to strike at set location, and 4) scripting any of the above alterations to occur at specific times during simulation.

Subsequently in 2018, Cordonnier *et al*. (2018) proposed another physics-based method that interactively generated terrain by allowing the user to use hand gestures on a multi-touch

device to simulate the compression and folding of colliding tectonic plates. These simulations create realistic-looking mountains where the colliding plates push the earth up into mounds.

Although these physics-based tools are useful for creating realistic terrains, they do not incorporate any gameplay-related PCG methods, leaving the inclusion of any game-related elements entirely to the user.

Zhou *et al*. (2007) proposed a sample-based approach to generating terrains, where the generated terrain is represented as a grid of terrain patches. This approach takes two inputs, a sample terrain, and a user sketch specifying the features that are to be included in the generated terrain. The sample terrain is processed to break it down into a collection of patches, where each patch represents a feature of the terrain such as mountain ridges. The features identified in the example terrain are then matched up with the features present in the input sketch to form a new terrain. This method produces aesthetic terrains that contain features such as mountains and rivers, but has no gameplay considerations.

Raffe *et al*. (2011) proposed another sample-based technique that pieces a terrain together using existing terrain tiles via interactive evolutionary computation (IEC). This technique represents terrains as a 2D grid of terrain tiles, which are small square chunks of pre-generated terrain. The IEC used in this technique initialises a population of generated terrains as a random selection of tiles. The IEC is similar to a GA except rather than use an automatic fitness evaluation method and selection scheme the user is required to interact with the process at each generation to select the most desirable terrains to be used in mutation and crossover. The user is also able to specify tiles that they do not want to be used in future generations, giving them finer control over the process. This technique was developed with gameplay in mind but is not an automated process, requiring the user to intervene regularly during the generation process. Also, the grid-like nature of this technique is apparent in the generated terrains making them look more uniform and less natural.

A similar patch-based approach was taken by Antoniuk & Rokita (2015), where each patch was assigned a base-height, dispersion values, and a terrain type. The terrain type determined which algorithm was used to generate the height values of the patch, which were offset by the patch's base height. Once the terrain for all of the patches has been generated, the boundaries between patches are blended to form smooth transitions. Although this technique does not require frequent user input, it does require detailed information to generate the terrain

including a low-resolution height map that stores the base heights of each patch, a text file containing multiple dispersion values for each patch, and a type map that stores the terrain type of each patch.

Ong *et al*. (2005) and Li *et al*. (2006) both proposed similar techniques that involve the user defining the terrain as a set of regions, where each region is assigned a specific terrain type (e.g. forest, desert, etc). Their methods then stitched the generated terrain together from existing terrain samples, where the samples used to form each region are of the same terrain type as the region. For example, when generating desert terrain this technique will use existing examples of desert terrain. These techniques do not suffer from the unnatural grid-like terrains produced by Raffe *et al*. (2011) and are also fully automated processes, but they do not consider any form of gameplay requirements and require a large collection of existing terrain samples to produce visually distinct terrains.

Another sample-based terrain generation method was proposed by Guerin *et al*. (2017), which uses four types of terrain synthesisers to allow the user to interactively edit a terrain. These four synthesisers are used to 1) transform a sketch to a terrain 2) transform a levelset to a terrain 3) erase a section of terrain, and 4) apply erosion to the terrain for added realism. Each synthesiser is a generative adversarial network (GAN) (Goodfellow *et al*., 2014), which are trained on existing terrain exemplars to produce realistic edits to the generated terrain. This method suffers from the same drawback as other sample-based methods in that it requires existing terrain data to train the GANs as well as not considering any form of gameplay.

2.1.3 Gameplay-Based Terrain Generation

Fully automated techniques that generate terrain which meets certain gameplay requirements have also been developed. One such technique is proposed by Togelius *et al*. (2010) where a GA is used to generate RTS game maps, including terrain, resource placement, and player base placement. The chromosome in this approach uses lists of integers to represent 2D location coordinates for bases, resources, and hills, which also include an additional height dimension. The fitness function evaluated the game maps based on five objectives, distance between player bases and resources, distance between player bases and other player bases, distance between resources and other resources, elevation of player bases, and map symmetry. These objectives could be selected to be used in a multi-objective evaluation resulting in a Pareto front of solutions from which a designer could choose. This is a useful

technique although it does not produce aesthetic details and testing has so far been limited to RTS maps.

Another technique for generating terrain maps for RTS games is proposed by Olsen (2004) who developed an efficient erosion algorithm to form aesthetic terrain with a high probability of containing large accessible areas suitable for RTS gameplay. This technique used Voronoi diagrams to represent mountain and hill boundaries, and "pink" noise for terrain detail. An optimised hybrid algorithm was then applied to simulate thermal and hydraulic erosion to add realism. Owing to the use of Voronoi cells to represent mountains and hills, simple image processing techniques could be applied to alter their elevation and widen paths between them to increase accessibility. This technique is not only efficient but also produces aesthetically pleasing terrain while taking accessibility into account. Although, with the right parameters, this technique is likely to produce terrain with a suitable amount of accessible area, there are many other game related aspects that are not considered.

Frade *et al*. (2009) introduced a method of evolving terrain programs via interactive evolutionary computation. A terrain program is a program that can generate multiple terrains that all contain the same morphological characteristics. This work initially required user intervention to determine good terrain programs during the evolution process, but in later works user intervention was replaced with a fitness function. The fitness function proposed in Frade *et al*. (2010) evaluates a generated terrain based on the amount of the amount of terrain that is accessible. The evaluation is performed by creating a binary accessibility map which represents areas of terrain as either accessible or inaccessible based on a slope inclination threshold. After the initial accessibility map has been generated, a component labelling algorithm is run to determine the number of accessible areas in the map, and the number of cells that contribute to each accessible area. Only the largest accessible area remains in the map while the smaller areas are removed. The overall fitness of the terrain is determined by the size of the largest accessible area in comparison to the desired accessible area size. In a later paper, Frade *et al*. (2010) introduced another method of evaluating generated terrains based on obstacle edge length, which promoted the inclusion of inaccessible terrain for the player to navigate around.

Another PCG technique was introduced by Andereck (2014) which generated terrain that conformed to an accessible player path. This technique allows users to specify path control points which are connected by Hermite splines to form a smooth player path. Once the path is

generated, a terrain fitting algorithm is run which transforms a flat plane into believable terrain while ensuring the inclusion of the desired path. Although this technique was not designed with video games in mind, fitting terrain to desired player paths would be a valuable tool in PCG for video games.

Smelik *et al*. (2010) introduced a sketch-based approach for procedurally generating terrains for military training games. This technique allows users to provide a rough sketch of the terrain, including locations to place buildings, roads, and vegetation, converts the sketch to an editable layered terrain map that contains detailed terrain data, and finally generates the terrain as a combination of a height map and 3D models. Although designed for creating game levels this approach still requires a large amount of user input with desired game design elements having to be manually expressed by the user in the sketch.

Another technique proposed by Smelik *et al*. (2011) used two types of constraints to control the generation process of virtual worlds: semantic constraints and feature constraints. Semantic constraints could be specified by the designer and included line of sight, chokepoint, route, and concealment constraints. Feature constraints govern the placement and state of virtual world features such as trees and terrain. The semantic constraints are enforced by an interaction with the feature constraints where feature constraints are required to alter their current state to conform to the requirements of the semantic constraints. For example, a line of sight constraint may be placed between two locations which are separated by terrain that contains two feature constraints, hilly terrain and a forest. The line of sight constraint interacts with the hills and forest features, forcing them to alter their state to conform to the semantic constraint. The hills feature conforms to the semantic constraint by lowering its hills appropriately to make line of sight possible while the forest feature removes trees from the line of sight, meeting the constraint requirements. This technique is a step towards enforcing desired gameplay requirements during virtual world generation, and although constraints may interact with the terrain of virtual worlds, it is not a terrain generation technique and contains no procedural generation of virtual world features.

2.1.4 Analysis of the State of the Art

Currently there are many terrain generation techniques that are focused on aesthetics and few that are focused on gameplay requirements. This forms a noticeable gap in current knowledge and there is still no method that is capable of forming terrains that meet a wide variety of gameplay requirements. Limited work has been done in this area by Olsen (2004), Togelius *et al*. (2010),

and Frade *et al*. (2010) although their sole focus is either accessibility or the RTS genre. Andereck's (2014) work focused on generating terrain to conform to a desired path, which may be useful as a terrain generation component, but is still far from being a complete terrain level generator. Another approach to including game design in terrain uses semantic constraints that can interact with the terrain to ensure it meets the desired design (Smelik *et al*., 2011). Although this approach allows game design constraints to be enforced, the user is still required to manually design and create the virtual world, and implement the desired constraints. With accessibility and the RTS genre being the main focuses in this field there is room for more scalable approaches that can generate terrains with a wider array of gameplay elements for other game genres. In this thesis, such a scalable approach is proposed and evaluated. The approach is based on Genetic Algorithms (GA), a review of which is presented in the next section.

2.2 Technical Review of Genetic Algorithms

GAs are integral to the algorithms developed and evaluated in this thesis. While other meta-heuristic optimisation techniques have been used in other game areas, such as particle swarm optimisation (Fister Jr. *et al*., 2015), a GA was chosen for the developed approach as the focus of this approach is terrain generation, which has a noisy fitness landscape to which GAs are well suited. This section details basic GA concepts for readers who are unfamiliar with GAs. GAs were introduced by John Holland (1992) and is a common search optimisation technique in the field of computational intelligence that evolves a population of candidate solutions by mimicking the biological process of natural selection, mutation, and reproduction.

The GA is an iterative process and requires a population of candidates or potential solutions, which are represented by chromosomes. The population of candidates enter a loop where they undergo evaluation, selection, mutation, and crossover. The loop terminates once a termination condition is met. A diagram of this process is shown in Figure 2.1. The rest of this section describes common elements in GA including chromosomes, fitness functions, selection schemes, mutation operators, crossover operators, termination conditions, and GA parameters.

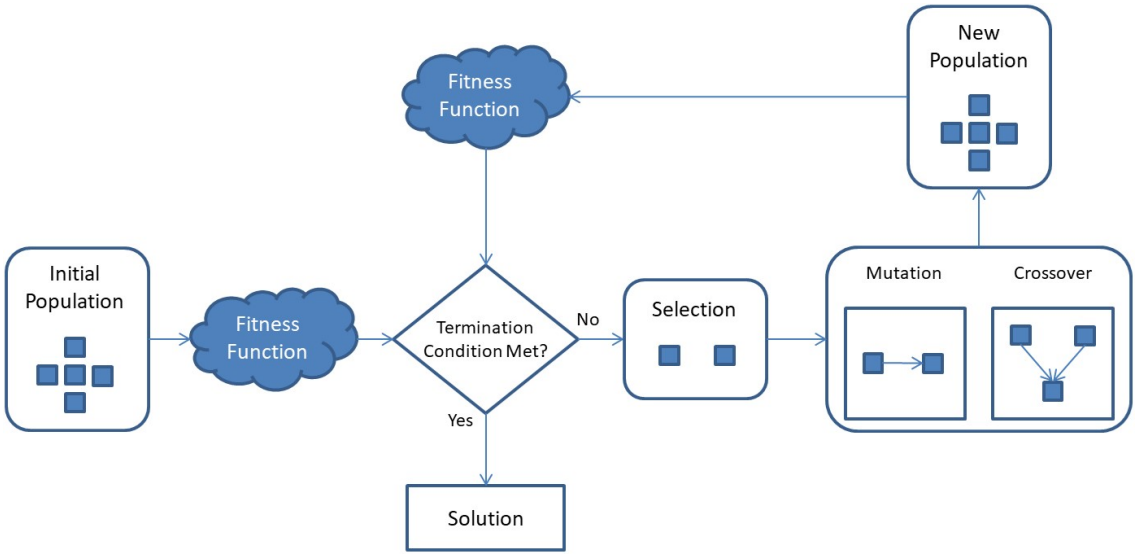


Figure 2.1. A diagram of a basic GA process, where an initial population is evaluated and enters a loop of selection, mutation and crossover, and re-evaluation.

Chromosomes, also called genotypes, are data representations of solutions, called phenotypes, and are made up of a collection of parameter values called genes. A chromosome's genes are manipulated during the GA process to alter the solution that it represents. Chromosomes (genotypes) must undergo a mapping process to form their solution (phenotype), after which the solution is evaluated via fitness function.

The **fitness function** is applied to each candidate solution to evaluate how accurate the solutions are. The measure calculated by the fitness function is called a fitness value and the aim of the GA is to optimise solutions via the fitness function. After evaluation, chromosomes are selected based on the fitness value of their solution.

The **selection scheme** determines how each generation of candidates are chosen and follows the general principle of selecting the fittest solutions to mutate and breed. A commonly used technique selects a chromosome, using a chosen *selection method*, and applies mutation and crossover based on some probability. The altered chromosome is then placed into a new population and this process is repeated until the new population is of appropriate size. Additionally an elitist scheme can be chosen (Mitchell, 1996), where some candidates that are considered 'elite' are selected each generation to be carried over to the new population completely unchanged.

The *selection method* selects a chromosome based on the fitness that was assigned to its phenotype, and there many existing techniques that do this. A common selection method is

tournament selection (Goldberg & Deb, 1991), which is performed by selecting a number of random chromosomes from the GA's population to form a tournament group and picking the chromosome from the group that has the best fitness value. Increasing the tournament group size increases the selection pressure and can help a GA converge faster, but this can also prematurely reduce variation in the GA's population causing the GA to converge at a local optima. A common group size for tournament selection is two, which is referred to as binary tournament selection.

The **mutation** operator gets applied to the chromosome at gene level, where each gene has a chance of being altered based on a mutation probability (*pm*). Genes can be altered in a number of ways, with some common methods including flip bit and uniform mutation (Mitchell, 1996), and Gaussian mutation (Hinterding, 1995). The following section describes Gaussian and uniform mutation as both are used in this study.

Gaussian mutation modifies a gene by adding a Gaussian distributed random value to the gene's current value. Due to the nature of this operator it can only be applied to genes that are represented as numbers. Equation 2.1 shows how Gaussian mutation is calculated, where x is the value of the gene, x' is the modified value of the gene, $N(0,1)$ is a random, Gaussian distributed value in the range of [0, 1], and σ is used to scale the strength of the noise added and is typically set to 10% of the genes value range.

Equation 2.1

$$x' = x + \sigma * N(0,1)$$

Uniform mutation works in a similar manner except rather than modifying a gene by a random value, the gene is completely replaced by a random value within set lower and upper bounds. Again, this operator can only be used on genes that are represented as numerical values.

The **crossover** operator is applied by swapping genetic material between two chromosomes, forming two new chromosomes. Like mutation, there are many types of crossover operators including uniform crossover, which swaps randomly selected genes between two chromosomes, and single-point (or one-point) crossover, which splits two chromosomes at a single point and swaps the segments between the two chromosomes. These crossover methods are described in the book titled "An Introduction to Genetic Algorithms" (Mitchell, 1996). The following section details single-point crossover as it is used in this study.

Single-point crossover is applied to a pair of chromosomes with *pc* probability. The two selected chromosomes are then split at a random location. The location of the split is the same for both chromosomes so that swapping the segments will not change the length of the chromosomes or make them invalid. The tail segments of the chromosomes are then swapped over and joined to the end of the head segment of the other chromosome. This creates two new chromosomes.

A common **termination condition** ends the GA once its population has converged. A population is considered converged once all chromosomes in the population are genetically similar. At this point the population has typically stopped improving and is ideally filled with near-optimal solutions. One method of calculating convergence is by getting the difference between the highest fitness of the population and the average fitness of the population and if the difference is below a set threshold for a set number of generations then the population is considered converged. Another common termination condition terminates the GA after a maximum number of generations. This method is often used as a backup measure in cases where the GA does not converge. A list of common termination conditions for GAs can be found in Safe *et al*. (2004).

Determining suitable values for **GA parameters** such as population size, mutation rate, crossover type, and crossover probability is a common issue with GA. This is a heavily researched issue and is its own research area. There are two primary ways of determining GA parameter values (Eiben *et al*., 1999); parameter tuning and parameter control. Parameter tuning involves running the GA many times while systematically varying the parameters until an ideal set is discovered. Some works (De Jong, 1975; Grefenstette, 1986; Goldberg, 1989) have focused on discovering a set of general GA parameter values that may be used for many search-based problems so that extensive parameter tuning is not required. Parameter control involves altering the parameter values as the GA is running. This can have some benefits such as having a high initial mutation rate to explore new areas of the search space early in the GA and reducing the mutation rate over time to help the GA converge.

2.3 Summary

The original use of PCG in video games was to generate random dungeon environments, which consistently presented players with new levels to explore. Since then, PCG techniques have improved and now levels can be generated to follow storylines and missions, which can also be procedurally generated. Now it is common for games to present the player, not with a

series of levels, but entire environments for them to explore. This has led to a series of PCG techniques that focus on generating interior environments, from generating floor plans and populating them with furniture, to the generation of exterior environments, such as towns, cities, and expansive terrains.

Although techniques have been developed that generate terrain, they have mostly focused on aesthetic appeal and the inclusion of natural features such as rivers and mountains, but do not consider game design principles. Some PCG techniques have been incorporated into semi-automated terrain generation tools to aid developers in producing high quality terrain without burdening them with the small details. Other approaches have been developed to address gameplay in terrain generation to some extent, including generating terrain with large accessible areas, or to conform to a manually designed path. Although these techniques are valid for a video game terrain, there are many other gameplay elements to consider and therefore are far from being a complete solution. Another technique could place player bases, resources, and hills for an RTS game, but the terrain representation was very basic and contained no aesthetic appeal. Another technique uses a form of constraint solving to raise or lower points on an existing terrain to enforce constraints such as a line-of-sight between two locations. However this technique contained no PCG elements and is primarily useful as a tool to aid in the creation of virtual environments. This thesis presents a novel approach to generating terrain that addresses the gap in knowledge by incorporating user-specified game design elements.

Chapter 3. Generating a Terrain Using a User-Specified Terrain

This chapter describes a method of generating a terrain that incorporates modifications as part of the fitness evaluation component of the approach, which is described in Chapter 6. The evaluation of the "*goodness of a solution*" is an important part of any evolutionary approach. In the proposed approach, this evaluation involves a terrain that has incorporated a set of evolved modifications associated with gameplay elements. Hence the ability to incorporate gameplay elements into a user-specified terrain is a key functionality and is used for the evaluation of every individual as part of the fitness evaluation step.

Two inputs are required for this process, namely an *initial terrain*, represented as a height map, and a *terrain graph*. The term 'map' is frequently used in this chapter and refers to a two-dimensional array of values, where the position of each value within the map represents an x and z coordinate associated with that value. The *terrain graph* consists of a collection of nodes and edges, representing a network of areas and paths to be incorporated into the *initial terrain*. Each node and edge contain information that represents a section of a terrain and its desired modifications. Table 3.1 and Table 3.2 list the information associated with each node and edge respectively.

Table 3.1. List of node/area attributes.

Attribute Name	**Attribute Description**
Position X	Geographic location of the node's centre along the X axis.
Position Z	Geographic location of the node's centre along the Z axis.
Offset Y	An offset height from the height of the terrain at the X and Z coordinates of the node.
Diameter	Diameter of the node.
Drop-off	Distance from the perimeter of the node for blending the height of the node with the height of the terrain.

Table 3.2. List of edge/path attributes.

Attribute Name	Attribute Description
Width	Edge width as a percentage of the diameter of the smallest node connected to the edge.
Control Point Offset X	One of two offset values which, when applied to the midpoint between the edge's start point and end point, produce a quadratic Bezier curve control point.
Control Point Offset Z	One of two offset values which, when applied to the midpoint between the edge's start point and end point, produce a quadratic Bezier curve control point.
Drop-off	Distance from the edge's perimeter for blending the edge's height with the terrain's heights.
Roughness	A scalar value used to amplify or dampen height difference between the edge's height and the terrain's heights.

Nodes, with their five attributes, represent circular areas of the terrain with a centre point (*Position X* and *Position Z* attributes), a diameter, and a drop-off radius, as shown in Figure 3.1 (a). The node's *Offset Y* attribute determines the height of the terrain within the node's area.

Edges represent paths between areas by using a quadratic Bezier curve that starts at one of the edge's associated nodes and ends at the other. The curvature of the edge is altered by a control point that is formed by the edge's *Control Point Offset X/Z* attributes. The amount of terrain surrounding the curve that is to be modified is determined by the edge's *Width* and *Drop-off* attributes as shown in Figure 3.1 (a). The reasoning behind using curved edges is to allow curved paths to be formed between areas in the terrain. Curved paths can be a desirable element in video games as they may hide the destination of players who are traversing them. This allows for the placement of ambushes or other obstacles that require the element of surprise. The height of the terrain within an edge's boundaries is linearly interpolated from the heights of the edge's two nodes, using the edge's *Roughness* attribute to control how smooth or rough the enclosed section of terrain is.

Figure 3.1 shows an example of two, equal sized nodes connected by an edge added to an initially flat terrain to illustrate how they represent and modify sections of terrain. Figure 3.1 (a) shows a top-down view of the terrain, where the colour black indicates un-modified terrain and the grey-to-white colours indicate the sections of modified terrain formed by the two nodes and the corresponding edge. The two green circles mark out the circular area of terrain determined by one of the node's *Diameter* and *Drop-off* attributes respectively. The inner blue line marks out the area of terrain determined by the width of the edge, which is calculated as the product of two values, that is, the *Width* attribute of the edge, and the smaller of the *Diameter* attributes associated with the two nodes of the edge. The outer blue line marks out the area of terrain determined by the *Drop-off* attribute of the edge. Figure 3.1 (b) shows a perspective view of the modified terrain.

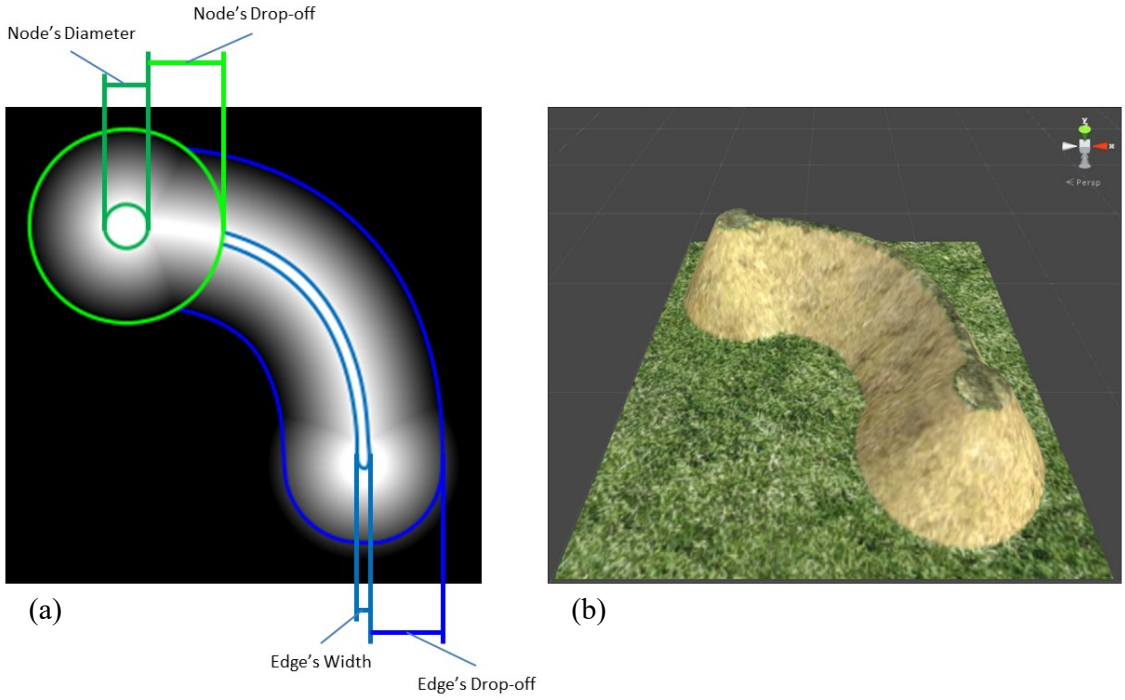


Figure 3.1. Example of how the attributes of two nodes, and the attributes of their connecting edge, determine the sections of terrain that get modified. (a) shows a top-down view of the terrain, where black terrain is unnaffected, and grey-white are affected areas. (b) shows a perspective, textured view of the modified terrain.

The *Drop-off* attributes of nodes and edges are used to smooth the surrounding terrain to blend in with the *initial terrain* to avoid unnatural wall artefacts from forming. This is shown in Figure 3.2, where Figure 3.2 (a) shows an *initial terrain* and Figure 3.2 (b) and Figure 3.2 (c) show the terrain with a generated area, one without blending and the other with blending using the *Drop-off* attributes. As shown in Figure 3.2 (b) circled in red, adjusting the terrain's height values to form the area without blending causes a near 90-degree incline that makes it

look like the surrounding terrain has been cut away. The terrain shown in Figure 3.2 (c) uses blending and it appears much more natural.

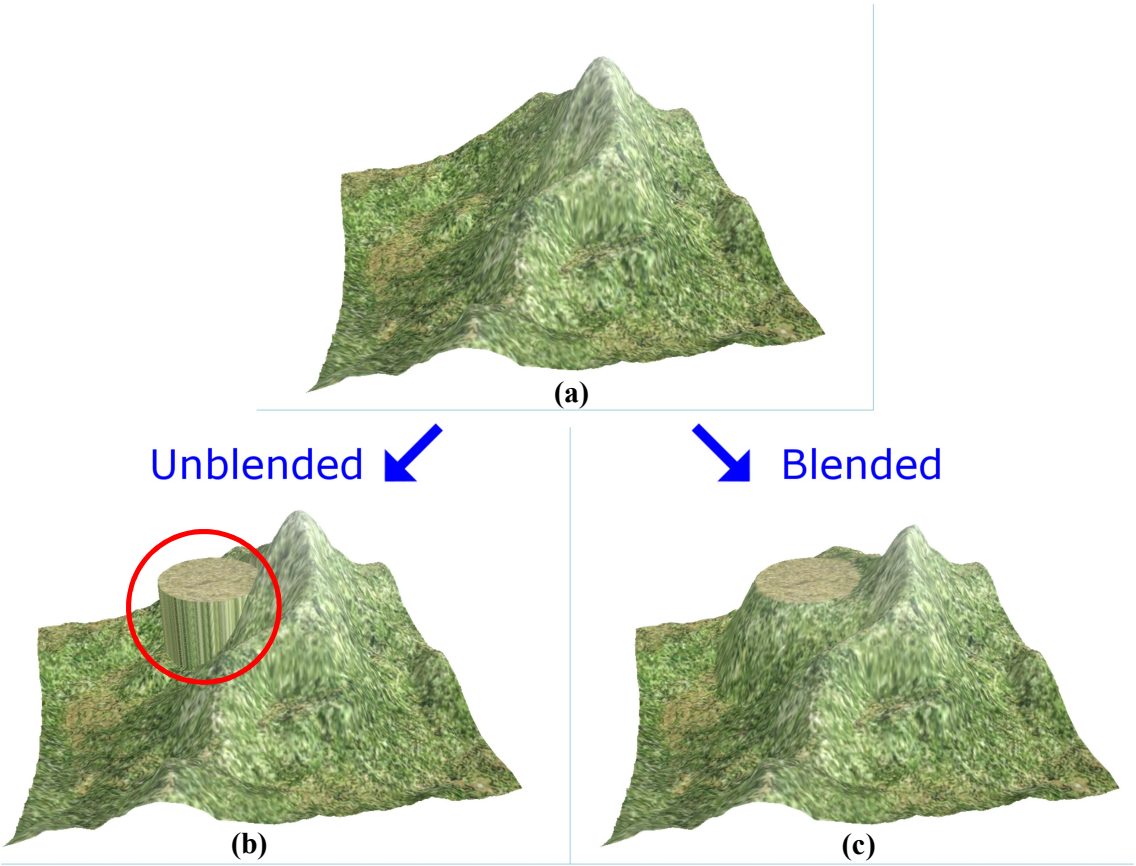


Figure 3.2. Example of a wall artefact. (a) an initial terrain. (b) initial terrain with generated area (wall artefact circled in red). (c) initial terrain with blended area (no wall artefact).

A *terrain of modifications* is formed by incorporating all of the nodes and edges, contained in the *terrain graph*, into an initially flat terrain. The *new terrain* is then formed by merging the *terrain of modifications* with the *initial terrain*. Figure 3.3 shows the steps involved in generating the *new terrain*, where each step produces an artefact that is input into subsequent steps until the *new terrain* is generated. The remainder of this chapter details the steps involved in generating the *new terrain*.

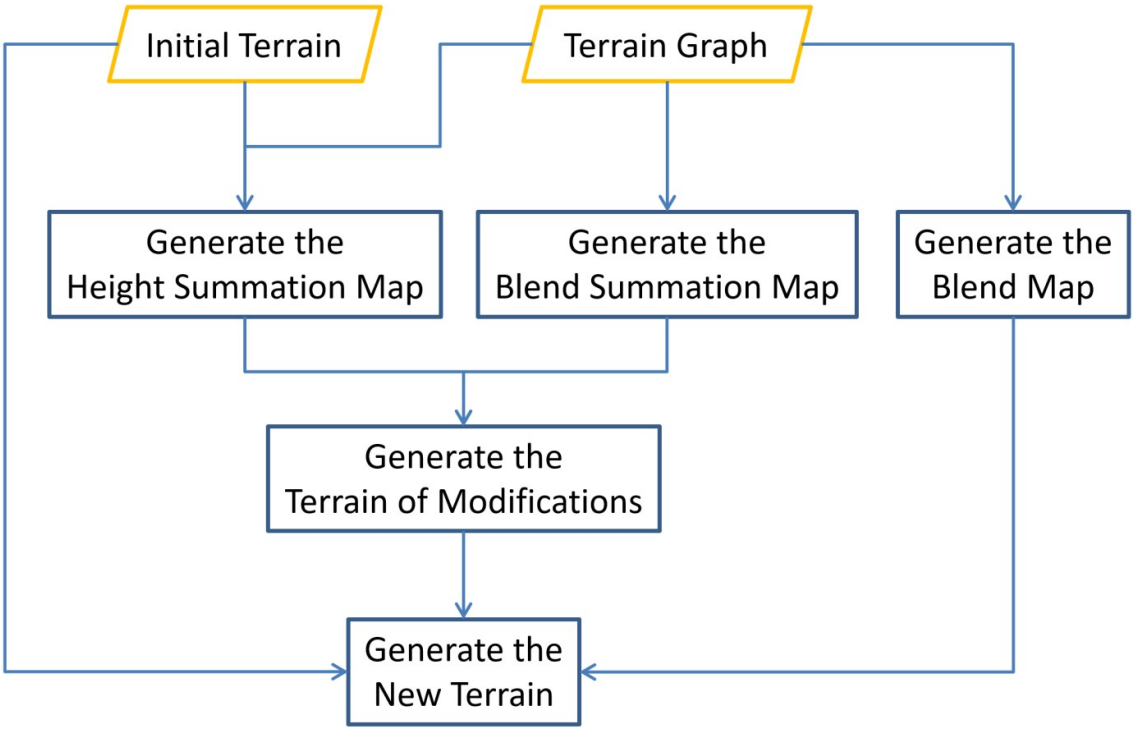


Figure 3.3. An overview of the steps involved in generating the new terrain.

3.1 Height Summation Map

The technique for generating the *height summation map* takes two inputs, the *initial terrain* and the *terrain graph*. Table 3.3 outlines the pseudocode associated with this technique, which starts by initialising the *height summation map*. Initialisation of the *height summation map* sets its size to equal the *initial terrain* and sets all of its values to zero. After the map has been initialised, the technique iterates over each node in the *terrain graph* to calculate each node's height value at each point on the terrain. These height values are stored in the *height summation map*. After the nodes have been processed each edge in the *terrain graph* has their height values calculated and stored in the *height summation map*.

Table 3.3. Pseudocode for generating the height summation map.

```
FUNCTION GenerateHeightSummationMap(initialTerrain, terrainGraph)
    heightSummationMap = EmptyMap()
    InitialiseMap(heightSummationMap)
    FOR EACH node IN terrainGraph DO
        FOR EACH point IN heightSummationMap DO
            bhn = CalculateBaseNodeHeight(node, point, initialTerrain)
            bn = CalculateNodeBlendFactor(node, point)
            hn = bhn * bn
            previousHeight = heightSummationMap.GetValueAt(point)
            heightSummationMap.GetValueAt(point) = previousHeight + hn
        END FOR
    END FOR
    FOR EACH edge IN terrainGraph DO
        FOR EACH point IN heightSummationMap DO
            bhe = CalculateBaseEdgeHeight(edge, point, initialTerrain)
            rhe = CalculateEdgeRoughness(edge, point)
            be = CalculateEdgeBlendFactor(edge, point)
            he = (bhe + rhe) * be
            previousHeight = heightSummationMap.GetValueAt(point)
            heightSummationMap.GetValueAt(point) = previousHeight + he
        END FOR
    END FOR
    RETURN heightSummationMap
END FUNCTION
```

The pseudocode in Table 3.3 shows that height values for nodes and edges (*hn*/*he*) are calculated as the product of a base height value (*bhn*/*bhe*) and a blend factor (*bn*/*be*). Edges require an additional factor when calculating their height, which is referred to as a roughness height value (*rhe*) and is calculated using their *Roughness* attribute. The calculations for values *hn* and *he* are detailed below.

The inputs required to calculate a nodes height value, *hn*, for a single point in the *height summation map* include the node being processed, the *point* of the *height summation map*

being processed, and the *initial terrain*. The value *hn* is calculated as the product of the node's *bhn* and its *bn* as shown in Equation 3.1.

Equation 3.1

$$hn = bhn * bn$$

The calculation for the *bhn* is equal to the summation of the height of the *initial terrain* at the node's centre point, and the node's *Offset Y* attribute. This is shown in Equation 3.2, where *th* is the *initial terrain*'s height at the node's centre, and *oy* is the node's *Offset Y* attribute.

Equation 3.2

$$bhn = th + oy$$

The node's *bn* value is a function of distance from the node's centre, where if the distance between the *point* being processed and the node's centre is less than the node's radius, then *bn* equals one. If the distance is less than the summation of the node's radius and the node's *Drop-off* attribute then *bn* equals a value in the range of [0, 1], where shorter distances result in higher values. If the distance is greater than the summation of the node's radius and the node's *Drop-off* attribute, then *bn* equals zero. The calculation of *bn* is shown in Equation 3.3, where *p* is the *point* (the x and z coordinates of the *height summation map* value being processed), *c* is a two dimensional vector consisting of the node's *Position X* and *Position Z* attributes, *d* being the node's *Diameter* attribute, and *ndo* is the node's *Drop-off* attribute.

Equation 3.3

$$bn = \max\left(0.0, 1.0 - \frac{\max\left(0.0, |p - c| - \frac{d}{2}\right)}{ndo}\right)$$

To calculate an edge's height value, *he*, three inputs are required; the edge being processed, the *point* being processed, and the *initial terrain*. The value *he* is calculated as the product of the edge's *be* and the summation of the edge's *bhe* and *rhe* as shown in Equation 3.4.

Equation 3.4

$$he = (bhe + rhe) * be$$

The edge's *bhe* value is linearly interpolated between the heights of the edge's two nodes, using the distance along the edge's curve as the interpolation factor to form a slope from one

node to the other. This is demonstrated in Figure 3.4, which is a side view of a terrain containing two nodes connected by an edge. The edge height is interpolated from the heights of the two nodes to form a slope between them.

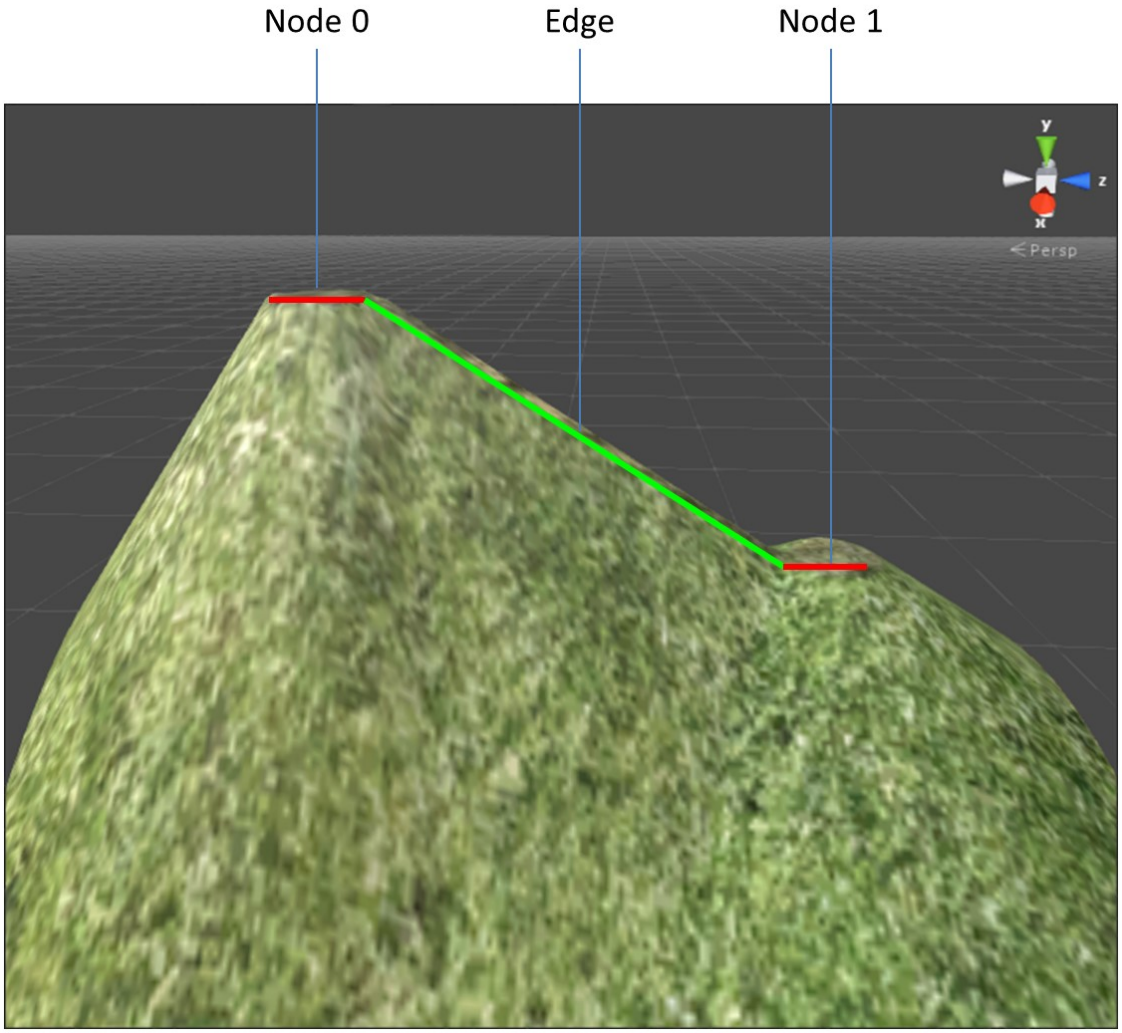


Figure 3.4. Example of an edge's base height being interpolated from its two nodes' heights to form a slope.

Equation 3.5 shows the calculation for *bhe* of a given *point*, where bhn_{n1} and bhn_{n2} are the base heights of the edge's two nodes, and t is a value in the range of [0, 1] representing how far along the *point* is located on the edge's curve. If the *point* is not directly on the edge's curve, then t is calculated for the nearest point on the edge's curve. The pseudocode for getting the nearest point on a quadratic Bezier curve, and its associated t value, is shown in Appendix A.2 and requires four inputs (*p*, *cp0*, *cp1*, *cp2*). The value *p* is the *point*, values *cp0* and *cp2* are points on the circumference of the circle associated with each of the two nodes, where *cp0* is a distance from the centre of the edge's first node, equal to half of the node's *Diameter* attribute, in the direction of the edge's second node. The value *cp2* is the same

except that it is for the edge's second node. Figure 3.5 demonstrates this more clearly, where the directions from the centre of each node, point towards the centre of the opposing node, and the value's *cp0* and *cp2* are placed a distance along their respective directions equal to half of their node's *Diameter* attribute. The value *cp1* is the average of *cp0* and *cp2* plus the edge's *Control Point Offset X/Z* attributes ($cp1 = \frac{cp0+c}{2} + (Control\ Point\ Offset\ X, Control\ Point\ Offset\ Z)$).

Equation 3.5

$$bhe = bhn_{n1} * (1 - t) + bhn_{n2} * t$$

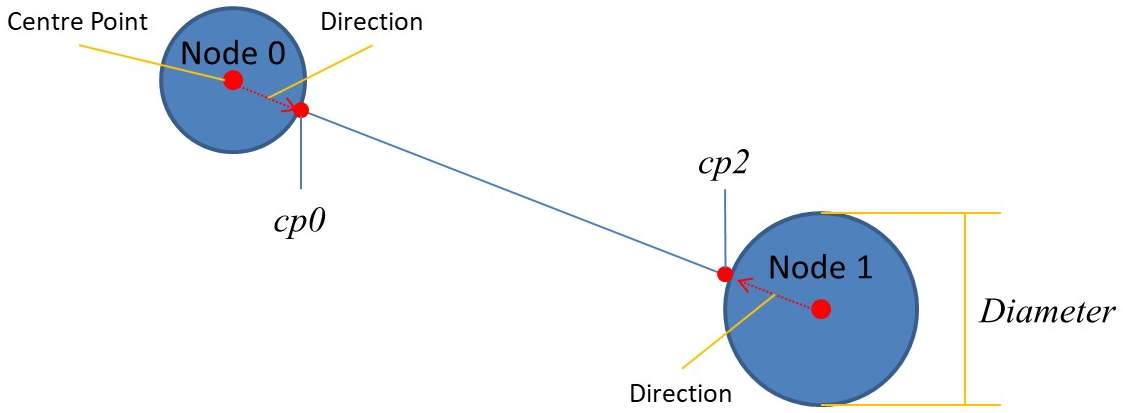
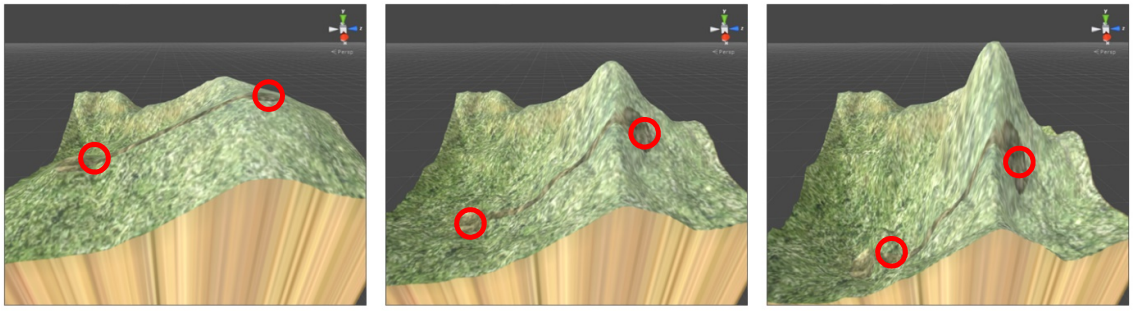


Figure 3.5. Example of how *cp0* and *cp2* are calculated as the node's centre point plus the direction to the node's opposing node multiplied by the node's radius.

The edge's *rhe* value is calculated using the edge's *Roughness* attribute and is used to distort (i.e. to smooth or to roughen) the terrain between its two nodes. Figure 3.6 demonstrates this with three images of a terrain with two generated areas, circled in red, connected by an edge, each generated with a different *Roughness* attribute value. The terrain shown in (a) uses a *Roughness* value of zero which forms a smooth path between the two areas, while (b) uses a *Roughness* value of one, which has no effect on the terrain between the two areas, making it the exact same terrain as the *initial terrain*. The terrain shown in (c) uses a *Roughness* value of two, amplifying the jaggedness of the terrain between the two areas.



(a) *Roughness* = 0.0 (b) *Roughness* = 1.0 (c) *Roughness* = 2.0

Figure 3.6. Example of how different *Roughness* attribute values affect the generated terrain. (a) shows smooth terrain formed by a roughness value of 0. (b) shows initial terrain heights formed by a roughness value of 1. (c) shows amplified heights formed by a roughness value of 2.

The *rhe* is calculated as the difference between the edge's *bhe* and the *initial terrain*'s height, and multiplying it by the edge's *Roughness* attribute. This is shown in Equation 3.6, where *hp* is the height of the *initial terrain* at the *point*'s coordinates, and *r* is the edge's *Roughness* attribute.

Equation 3.6

$$rhe = (bhe - hp) * r$$

As with the node height values, the edge's height value must be multiplied by its blend factor, *be*, before being added to the height summation map. The value *be* is also a function of distance, this time the distance is between the *point* and its nearest point (*np*) along the edge's curve. Equation 3.7 shows how *be* is calculated, where *p* is the *point* on the *height summation map* being processed, *w* is the edge's width, calculated as the product of the edge's *Width* attribute and the smaller of the edge's two nodes' *Diameter* attributes, and *edo* equals the edge's *Drop-off* attribute.

Equation 3.7

$$be = \max\left(0.0, 1.0 - \frac{\max\left(0.0, |p - np| - \frac{w}{2}\right)}{edo}\right)$$

That concludes the calculations required for processing nodes and edges to get their height values. Once a height value has been calculated, it gets summed with the existing value contained in the height summation map at the *point*'s coordinates.

3.2 Blend Summation Map

This section details the technique of generating the *blend summation map*, which is a similar process to generating the *height summation map*. Table 3.3 outlines the pseudocode associated with this technique. This technique takes a single input, the *terrain graph*, and starts by initialising the *blend summation map* in the same manner as the *height summation map*, by setting its size to equal the size of the *initial terrain* and setting all of its values to zero. After the map has been initialised, the nodes and edges in the *terrain graph* are processed to get their blend values (*bn*/*be*), which are then added to the *blend summation map*.

Table 3.4. Pseudocode for generating the blend summation map.

```
FUNCTION GenerateBlendSummationMap(individual)
        blendSummationMap = EmptyMap()
        InitialiseMap(blendSummationMap)
        FOR EACH node IN individual DO
                FOR EACH point IN blendSummationMap DO
                        bn = CalculateNodeBlendFactor(node, point)
                        previousBlend = blendSummationMap.GetValueAt(point)
                        blendSummationMap.GetValueAt(point) = previousBlend + bn
                END FOR
        END FOR
        FOR EACH edge IN individual DO
                FOR EACH point IN blendSummationMap DO
                        be = CalculateEdgeBlendFactor(edge, point)
                        previousBlend = blendSummationMap.GetValueAt(point)
                        blendSummationMap.GetValueAt(point) = previousBlend + be
                END FOR
        END FOR
        RETURN blendSummationMap
END FUNCTION
```

As shown in Table 3.4, the methods required for generating the *blend summation map* are a subset of the methods used to generate the *height summation map*. The methods used to get the values *bn* and *be* are the same methods used to get these values during the generation of

the *height summation map* and are therefore not re-defined here. Once all nodes and edges have been processed, the *blend summation map* is complete.

3.3 Blend Map

The *blend map* stores a blend factor for each point of the terrain. These blend factors determine how much influence the *terrain of modifications* have on the *new terrain* when merging with the *initial terrain*. Blend factors are values in the range of [0, 1], where a value of zero means that the height value in the *new terrain* will be 0% of the modified height plus 100% of the initial height. The converse is true when the blend factor has a value of 1.

Generating the *blend map* is a similar process to generating the *blend summation map*. The only difference is that rather than adding the blend factors to the previously stored value in the map, as shown in Table 3.5, the previously stored value will get replaced if the new blend factor is greater or remain the same if it is not. This is shown in Table 3.6.

Table 3.5. Pseudocode line for summing blend factors when generating the blend summation map.

```
blendSummationMap.GetValueAt(point) = previousBlend + bn  /* or 'be' */
```

Table 3.6. Pseudocode line for replacing blend factors when generating the blend map.

```
blendMap.GetValueAt(point) = Max(previousBlend, bn /* or 'be' */)
```

Making this change to the pseudocode listed in Table 3.4 will result in the complete process of generating the *blend map*.

3.4 Terrain of Modifications

The *terrain of modifications* uses the same representation as the *initial terrain*, a height map, and is initialised to equal the same size as the *initial terrain* with all height values equal to zero. Each height value stored in this terrain is the un-blended, modified height as determined by the modifications encoded by the nodes and edges of the *terrain graph*.

The inputs required to generate this terrain are the *height summation map*, and the *blend summation map*. Once the *terrain of modifications* has been initialised, its height values are set to the *height summation map*'s values divided by the *blend summation map*'s values. Appendix A.6 lists the pseudocode for generating the *terrain of modifications*. Once each

point has been processed, the *terrain of modifications* is complete and can be blended with the *initial terrain* to produce the *new terrain*.

3.5 New Terrain

Generating the *new terrain* requires three inputs; the *initial terrain*, the *terrain of modifications*, and the *blend map*. The *new terrain* is a height map initialised to equal the size of the *initial terrain*. Once initialised, each height value of the *new terrain* is calculated using linear interpolation between the *initial terrain*'s height and the *terrain of modification*'s height, using the blend factors stored in the *blend map* as the interpolation factors. Appendix A.7 lists the pseudocode of this technique.

Once all points of the *new terrain* have been computed, the *new terrain* is complete. Figure 3.7 shows an example of a *terrain graph* that is used to modify an *initial terrain* to create a *new terrain*. Figure 3.7 (a) shows a visual representation of a *terrain graph* that contains modifications for incorporating four areas (green spheres) and three paths. Figure 3.7 (b) shows the *initial terrain* that was generated using the program L3DT (Bundysoft, 2018). This *initial terrain* does not contain many suitable gameplay elements, like areas that are suitable for the placement of objects such as houses. Figure 3.7 (c) displays the *new terrain*, with the four incorporated areas, linked together by smooth, traversable paths. The incorporated areas in the *new terrain* provide more suitable locations for placement of objects, as demonstrated in Figure 3.8. Figure 3.8 shows the *new terrain*, with houses placed on the incorporated open areas to form a village, with trees, rocks, and water, added for aesthetic detail. The two smaller areas that have been incorporated on the hill and mountain side form vantage points, where houses that look over the town have been placed.

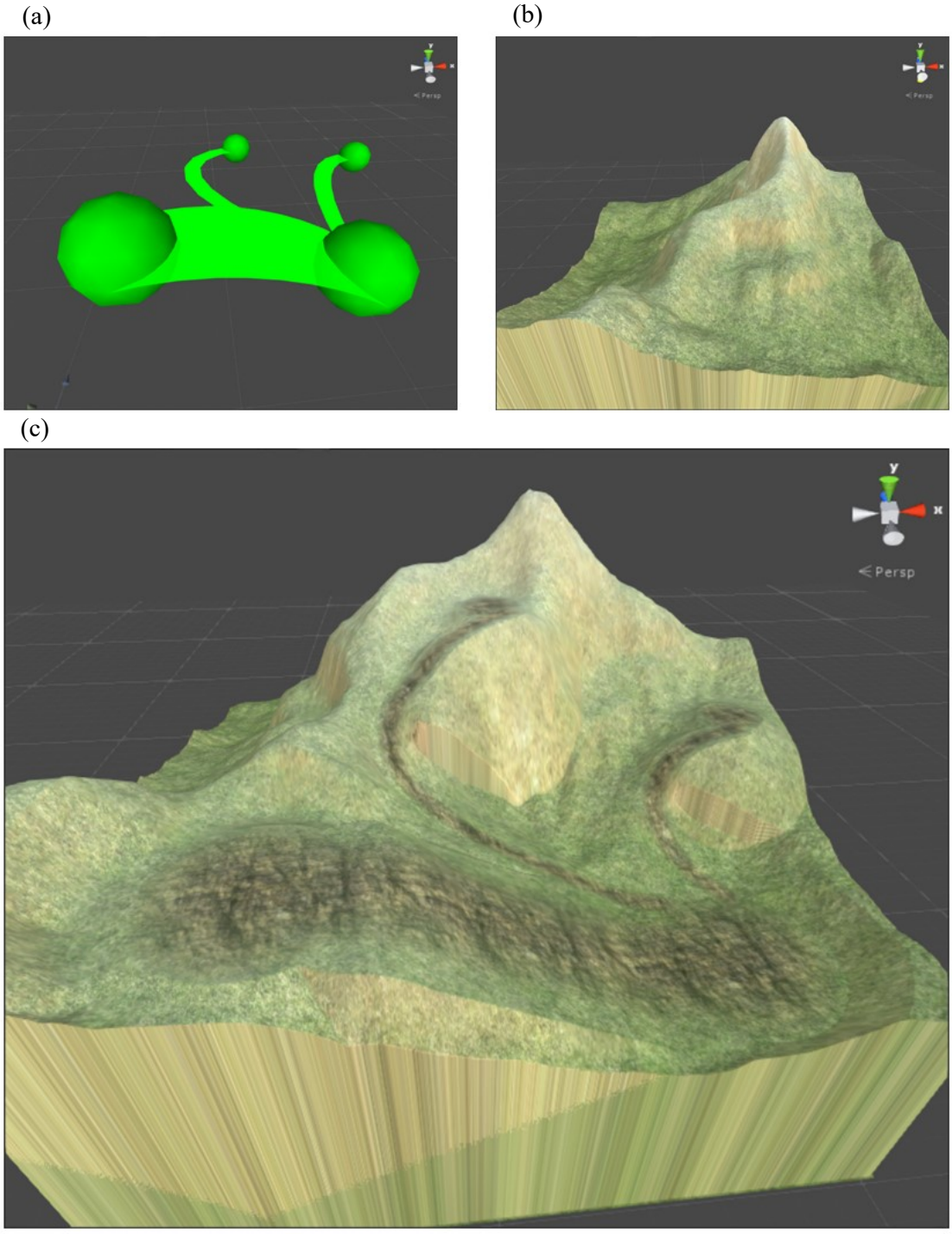


Figure 3.7. (a) A 3D rendered terrain graph representing modifications to an initial terrain. (b) The initial terrain. (c) The generated terrain. Rendered in Unity (2018).

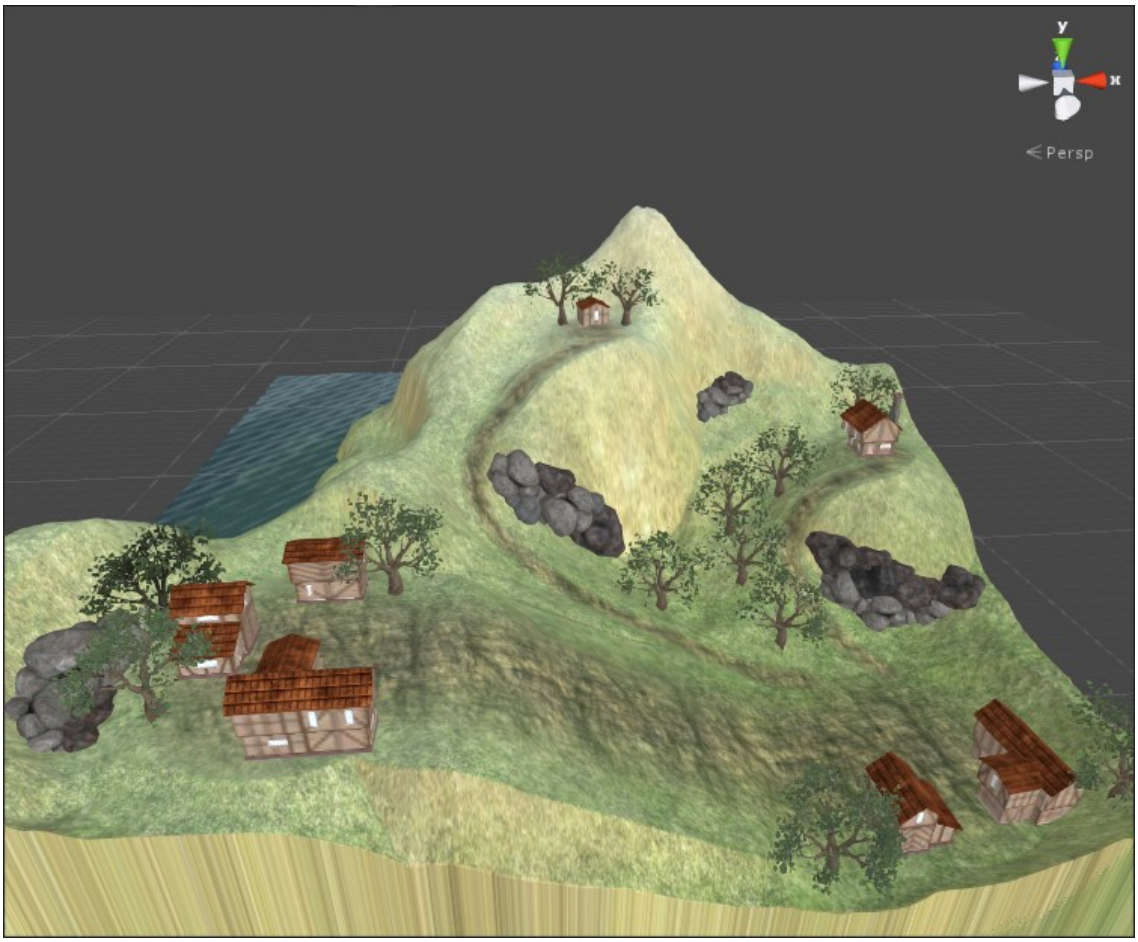


Figure 3.8. The generated terrain populated with trees, houses, rocks, and water. Rendered in Unity (2018).

3.6 Summary

This chapter detailed the method used to generate a *new terrain* from a *terrain graph* and an *initial terrain*, which is used in the evaluation of an individual in the proposed approach. This chapter detailed the generation of four artefacts, a *height summation map*, a *blend summation map*, a *blend map*, and a *terrain of modifications*, followed by how they are used to generate the *new terrain*.

Chapter 4. Measures Used to Characterise Areas on a Terrain

In order to evolve terrains that will contain areas of specific types such as an "open area" or a "vantage point", it would be necessary to have a set of quantifiable measures that is capable of capturing information about them. These measures can be used in an automated process to evaluate the suitability of an area in terms of being a specific type. This chapter details a set of proposed measures for characterising specific area types in a terrain as well as the methods to compute them. This approach uses these measures for an area in a terrain to categorise it as a specific area type. Therefore, being able to represent an area of terrain as a set of quantifiable measures is a key component of the approach.

Three inputs are required to obtain these measures for the areas in a given terrain;

1. The terrain: A height map (two-dimensional array of values) representing the height of the terrain at uniformly spaced locations across the terrains surface.
2. The terrain's accessibility map: A two-dimensional array of values similar to the height map except, instead of storing the height of the terrain at each location, it stores values indicating if the terrain is traversable at each location. Section 4.1 details the method used to generate the accessibility map.
3. A *layout graph*: A graph containing information about the areas that are to be characterised. It is similar to the *terrain graph* described in Chapter 3, except it only contains traversable edges, while a *terrain graph* contains both traversable and non-traversable edges. Traversable edges represent terrain between two areas that is flat enough to be traversed by a player character. Edges in a *terrain graph* may contain modifications that make the terrain between two areas too steep to be traversed. These would be examples of non-traversable edges. Section 4.2 describes the method used to generate this from a *terrain graph*.

Figure 4.1 shows examples of these inputs. Figure 4.1 (a) shows the height map of a terrain as a grey-scale image, where lighter values indicate higher altitudes in the terrain. Figure 4.1 (b) shows the accessibility map of the terrain, where white values indicate traversable sections of terrain and black values indicate terrain that is too steep to traverse. Figure 4.1 (c) is a 3D rendering of the terrain, represented by the height map in Figure 4.1 (a), with an overlayed *layout graph*. Green spheres represent nodes of the *layout graph*, which map to areas on the terrain, and green curves show the edges between nodes. These edges represent traversable paths between areas, which can be seen by the white sections in Figure 4.1 (b).

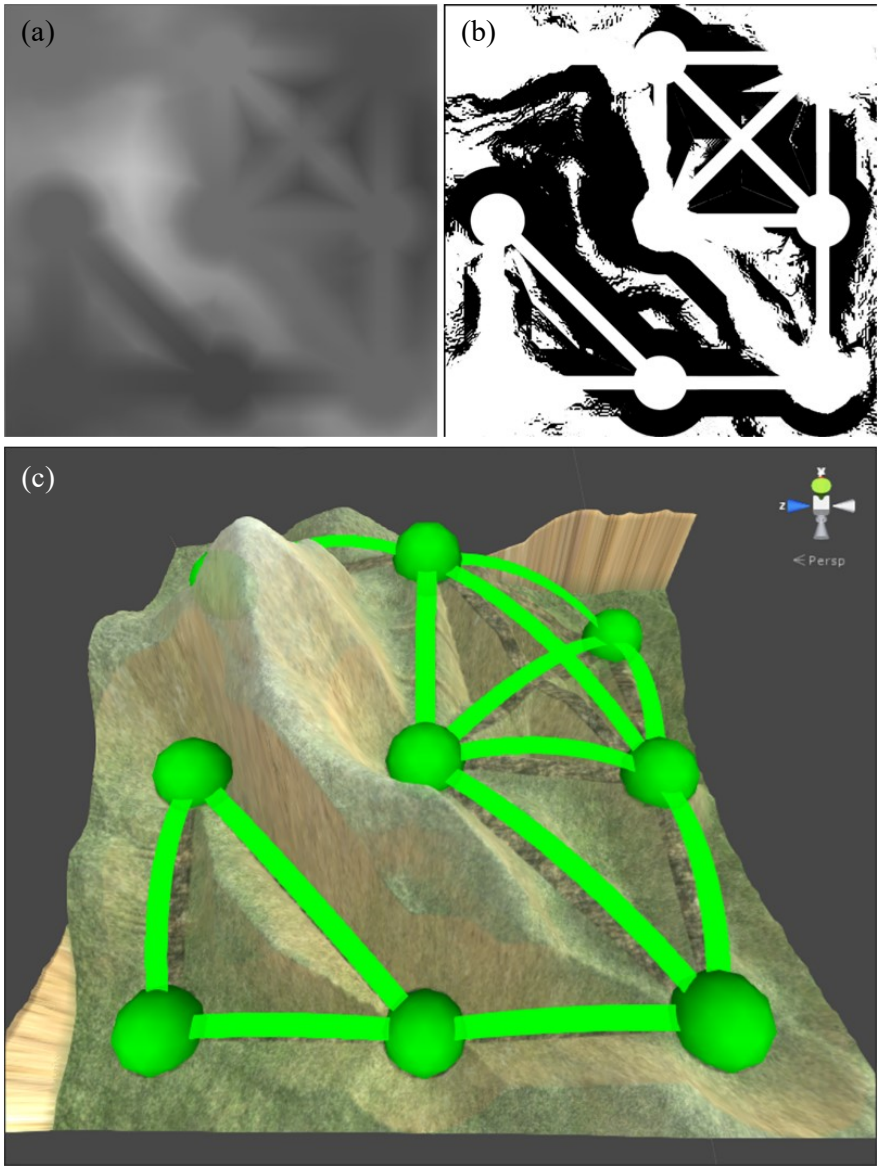


Figure 4.1. Examples of the three inputs required to characterise a terrain's areas. (a): height map. (b): accessibility map. (c): rendered terrain with overlayed layout graph.

Table 4.1 lists the complete set of measures developed in this approach for characterising an area in the terrain. Each measure, with the exception of the *Area Radius*, belong to one of two categories; graph measures (G_1-G_4), and isovist measures (I_1-I_{11}).

An isovist (Benedikt, 1979) is defined as the visible (non-occluded) space surrounding a given point within an environment and is largely used in the field of architecture. Various measures can be calculated for an isovist, such as its size and the maximum distance that can be seen from the isovist's origin. These measures can describe various characteristics of an area, for example, Benedikt (1979) suggests that the privacy of an area could be determined by the size of its isovist's volume and its isovist's skewness.

Dalton (2001) proposed an application that generated isovists in 2D floor plans of buildings or urban spaces, allowing the user to see various measures associated with the generated isovists. Van Bilson & Poelman (2009) proposed using isovist fields in 3D virtual environments, presenting measures from a field of isovists as 2D images for further analysis. Neither of these approaches use space analysis to characterise areas or to categorise them as specific area types, although they do provide measures that may be useful in such an approach.

As isovists have been useful in analysing spaces in 3D environments, a collection of their associated measures is used in this approach to characterise areas in a given terrain. Measures from the field of graph theory are also used as they give information about an area in relation to other areas in the terrain.

The rest of this chapter is laid out as follows. Section 4.1 describes the accessibility map in detail and how to generate it from the height map of a terrain. Section 4.2 describes how to generate a *layout graph* from a *terrain graph* by using the accessibility map to identify and remove the *terrain graph*'s non-traversable edges. Section 4.3 details how to calculate the *Area Radius* measure followed by Section 4.4, which details the methods used to calculate the graph measures. Finally, in Section 4.5 are the details of how to calculate the isovist measures.

Table 4.1. List of the measures used to characterise an area of terrain.

Measure Name	Measure Description
Area Radius	The radius of an area of traversable terrain surrounding the areas centre.
Degree Centrality (G_1)	Number of edges connected to a node.
Eigenvector Centrality (G_2)	The influence of the node in the graph based on its degree centrality and the degree centrality of its neighbours, and their neighbours, etc.
Betweenness Centrality (G_3)	A measure corresponding to the number of shortest paths a node belongs to.
Closeness Centrality (G_4)	A measure of how close a node is to every other node based on the shortest paths between them.
Average Radial Length (I_1)	Average distance from a given location to its isovist's perimeter.
Dispersion (I_2)	The difference between the values of the mean and the standard deviation of the isovist's radial lengths.
Drift 3D (I_3)	The distance between the isovist's origin and the isovist's center of gravity.
Drift 2D (I_4)	Same as Drift 3D but ignores the Y (Up) axis.
Maximum Radial Length (I_5)	Maximum distance from a given location to its isovist's perimeter.
Minimum Radial Length (I_6)	Minimum distance from a given location to its isovist's perimeter.
Skewness (I_7)	Skewness of the distance from a given location to its isovist's perimeter.
Sphericality (I_8)	How well the isovist volume approximates a sphere.
Standard Deviation (I_9)	Standard deviation of distance from a given location to isovist perimeter.
Variance (I_{10})	Variation in distance from a given location to its isovist's perimeter.
Volume (I_{11})	Area that can be seen (is not occluded) from a given location.

4.1 The Accessibility Map

An accessibility map is a 2D array of binary values indicating the traversable (accessible) and non-traversable sections of the associated terrain. Therefore, the accessibility map is of the same size as the associated terrain's height map, where each value represents a single point on the terrain and specifies if that point is either traversable (1) or non-traversable (0). The accessibility map is generated by calculating the gradient of the terrain at uniformly spaced locations and using a gradient threshold to determine if the terrain is too steep to be traversed at each point. Terrain gradients are calculated using a Sobel filter, which returns gradient values in the range of [0, 1], which linearly maps to an angle in the range of [0, 90] degrees. The gradient threshold used in this approach is 0.5, meaning any terrain with a gradient of 45 degrees or greater is considered to be too steep and thus non-traversable. Appendix B.1 presents the pseudocode used to generate the accessibility map.

4.2 Generating a Layout Graph from a Terrain Graph

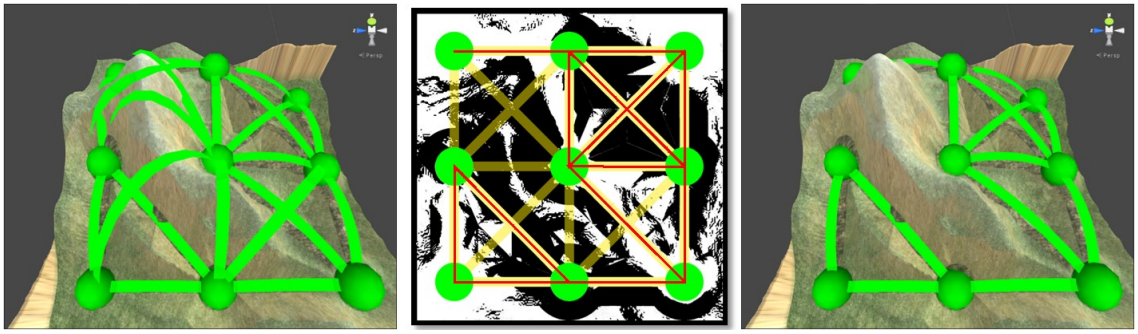
The *layout graph* contains all of the *terrain graph*'s nodes, and only contains those *terrain graph*'s edges that are identified as traversable. The process of generating the *layout graph* is divided into two steps.

Step 1: This step adds all of the nodes from the *terrain graph* as nodes to the *layout graph*.

Step 2: This step adds the traversable edges from the *terrain graph* as edges to the *layout graph*. To evaluate whether an edge is traversable, the proposed method applies the A* path finding algorithm to the terrain's accessibility map. A* (Hart *et al*., 1986) is a common graph traversal algorithm used to calculate the shortest path between two vertices of a graph. In this case the A* algorithm is applied to the accessibility map, where the accessibility map equates to a graph. If the A* algorithm finds a path between the centres of the edge's two nodes it is considered a traversable edge.

When applying the A* algorithm to find a traversable path between two points in the accessibility map, the algorithm must adhere to two restrictions: 1) Only traversable vertices of the accessibility map can contribute to the path. 2) The generated path cannot deviate from the edge's curve by a distance greater than half of the edge's width. If the A* algorithm finds a path while adhering to these restrictions, then the edge is considered traversable and it is added to the *layout graph*. Figure 4.2 demonstrates the process of determining if an edge is traversable and adding it to the graph. Figure 4.2 (a) shows a terrain overlayed with the

terrain graph. This image shows edges passing overs steep sections of terrain that are not traversable. Figure 4.2 (b) shows the accessibility map of the terrain, with black areas showing the non-traversable terrain. The green circles highlight the nodes from the *terrain graph*, while yellow lines highlight the area surrounding each edge (up to a distance equal to half the width of an edge). Red lines show which of these edges are traversable, as discovered by applying the A* algorithm. As shown, no traversable paths cross the black areas of the accessibility map, nor do they deviate from their associated edge by a distance greater than half of the edge's width. Figure 4.2 (c) shows the generated *layout graph* overlaying the terrain. As shown, the *layout graph* consists of all the nodes from the *terrain graph*, while only containing the traversable edges.



(a) (b) (c)

Figure 4.2. (a) A terrain with overlayed terrain graph. (b) Terrains accessibility map showing all traversable paths (red) found between nodes (green) while restricted to edges (yellow). (c) Terrain with overlayed layout graph (containing only traversable edges).

Once the *layout graph* has been generated the *Area Radius* and other measures can be calculated.

4.3 The 'Area Radius' Measure

The *Area Radius* measure is calculated for each area (node) in the *layout graph* and is a measure of the extent of traversable terrain from the centre of the specific area. The *Area Radius* is initialised to be half of the node's *Diameter* attribute, and this is then refined through a binary search as shown in Figure 4.3.

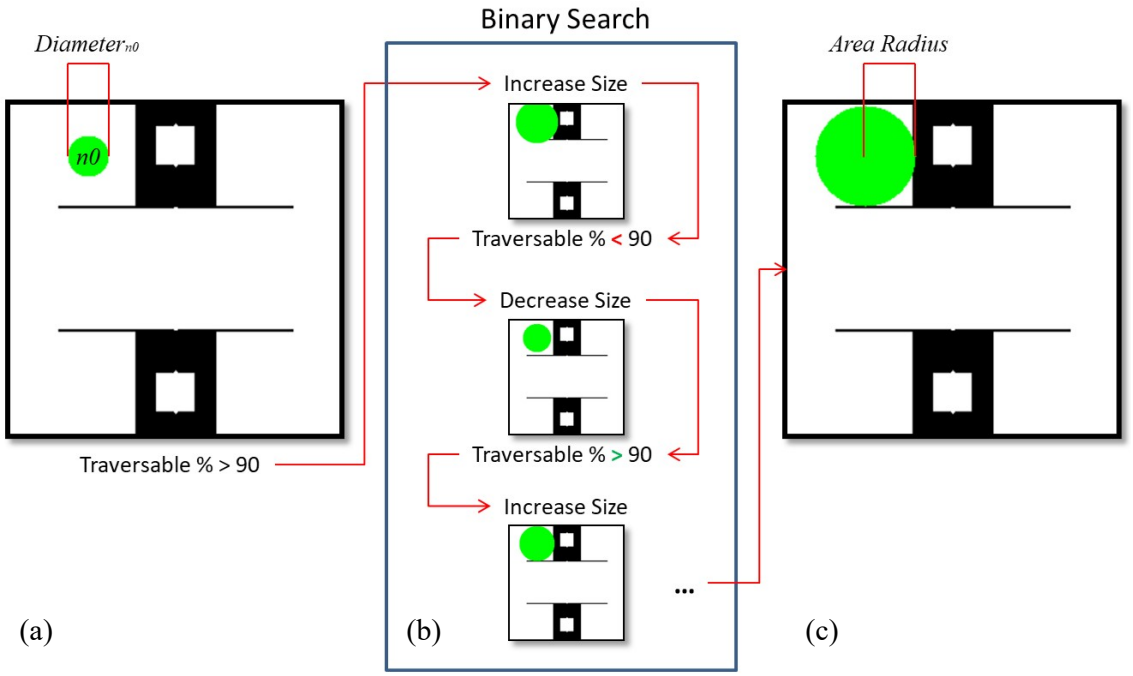


Figure 4.3. The binary search process for calculating the area radius attribute.

Figure 4.3 shows the steps of this process, using an accessibility map to show the traversable vs non-traversable terrain, and a green circle to indicate the area of terrain represented by the *Area Radius* measure. As shown, the *Area Radius* measure is initialised to half of its associated nodes *Diameter* attribute, and then enters the iterative process of the binary search. Each iteration of the binary search calculates the percentage of traversable terrain within the bounds of the *Area Radius* measure. As the percentage of traversable terrain within the initial *Area Radius* is greater than 90%, the *Area Radius* increases in size. The next iteration recalculates the percentage of traversable terrain within the *Area Radius*. As this percentage is now less than 90%, the *Area Radius* decreases in size. This continues for a set number of iterations, resulting in a more suitable *Area Radius* measure. A threshold of 90% was chosen so that small sections of non-traversable terrain would not prevent an accurate measure of traversable terrain. Figure 4.4 shows an example of how small sections of non-traversable terrain could limit an otherwise more accurate *Area Radius* measure.

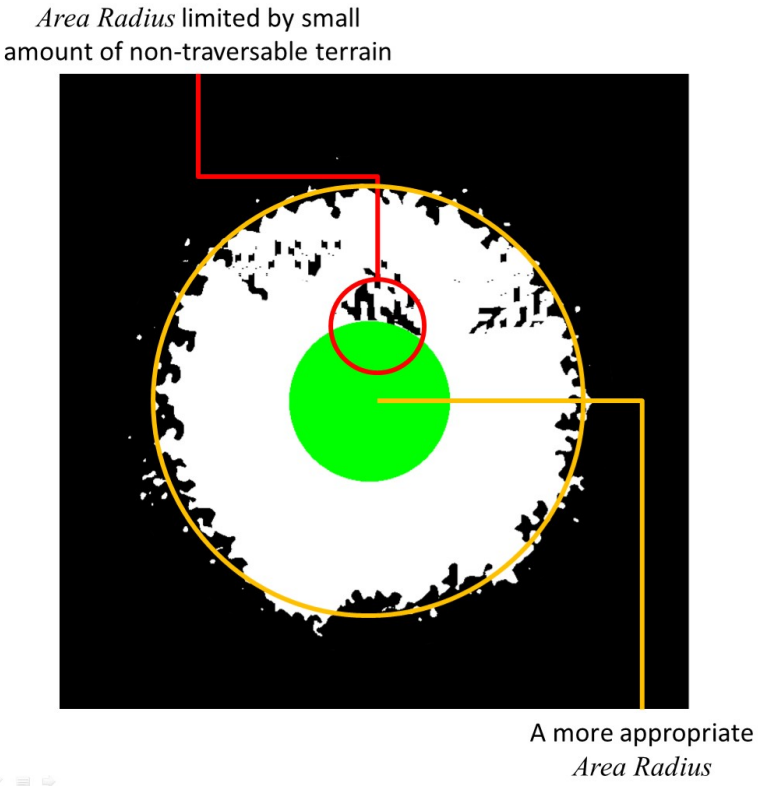


Figure 4.4. An area (green) with an inappropriate *Area Radius* measure caused by small sections of non-traversable terrain. A more appropriate *Area Radius* measure is highlighted in orange.

This figure shows a large area of traversable terrain (white) containing specks of non-traversable terrain (black). Visually, the area circled in orange encapsulates the traversable area better than the area highlighted in green. However, if there was no tolerance for non-traversable terrain within an area, the area of terrain that is encapsulated by the *Area Radius* would be limited to the green highlighted area. Appendix B.3 lists the pseudocode for calculating the *Area Radius* attribute.

4.4 Graph Measures

This section details the methods of calculating the four graph-connectivity measures that are listed in Table 4.1 (G_1-G_4). Graphs are used to capture the layout of the network of areas in the terrain, where each node in the graph represents an area on the terrain and each edge represents a traversable path between two areas. Certain measures that are calculated from a graph can be used to determine various characteristics of an area, such as the likelihood of the area being visited.

Graph measures are calculated for each area in the *layout graph*. These include *Degree Centrality*, *Eigenvector Centrality*, *Betweenness Centrality*, and *Closeness Centrality*

(Newman, 2010). The remainder of this section details the methods of calculating these graph measures.

Degree Centrality is calculated by counting the number of edges that are connected to the node. *Eigenvector Centrality* is a measure of how much influence a node has in a network, or in this case a graph. Calculation of this measure works off the concept that a node's influence is determined by the influence of its neighbouring nodes. Calculating this measure is an iterative process, where each iteration further refines each node's influence based on the influence of its neighbours. This means performing more iterations will result in a more accurate centrality measure. The approach detailed in this thesis performed 16 iterations when calculating *Eigenvector Centrality*. Appendix B.4 lists the pseudocode for the implemented algorithm.

In order to calculate *Betweenness Centrality*, the shortest path between each node in the graph to every other node in the graph is required. The *Betweenness Centrality* for any given node is then calculated as the number of these paths that travel through the given node (excluding the paths that begin or terminate at the given node). For undirected graphs, such as those used in this thesis, this value must be halved as paths get counted twice (e.g. a node that exists on the path between node A and node F will also exist on the path between node F and node A). Once *Betweenness Centrality* has been calculated it is normalised to be in the range of [0, 1]. Appendix B.5 lists the pseudocode for the implemented algorithm.

In order to calculate *Closeness Centrality* of a given node, a set consisting of the shortest path between the given node and every other node in the graph is required. The *Closeness Centrality* is then calculated as the sum of the lengths of the paths in this set. This value is then normalised by dividing the number of paths in the set by the sum of path lengths. In this thesis a path's length is calculated as the sum of its edges' lengths, where each edge length is calculated as the length of its associated quadratic Bezier curve. Appendix B.6 lists the pseudocode of the implemented algorithm for calculating closeness centrality.

4.5 Isovist Measures

This section details the calculation of the 11 isovist measures used in this research to quantify an area in the terrain. The isovist measures are used to capture information about the visible space surrounding a point in the terrain, such as the volume of visible space and the maximum distance that can be seen from the isovist's origin. These measures can describe

various characteristics of an area, for example, Benedikt (1979) suggests that the privacy of an area could be determined by the size of its isovist's volume and its isovist's skewness.

Isovist measures are calculated for each area in the terrain, where each area is represented by a node in the *layout graph*. Appendix B.7 lists the pseudocode associated with calculation of isovist measures for each area in the terrain.

4.5.1 Isovist Representation

An isovist associated with an area is first generated before calculating the measures associated with the area. This approach represents isovists as a set of radial vectors as introduced by Benedikt (1979). This representation consists of a number of equally distributed radial vectors cast out from the isovist's origin (a point in 3D space), as shown in Figure 4.5, where yellow lines represent the isovist's radials. The length of each radial vector is set to a maximum view distance unless terminated by intersecting with artefacts in the environment. Figure 4.5 demonstrates this, where some radials reach the maximum view distance, while others intersect occluding obstacles and terminate early.

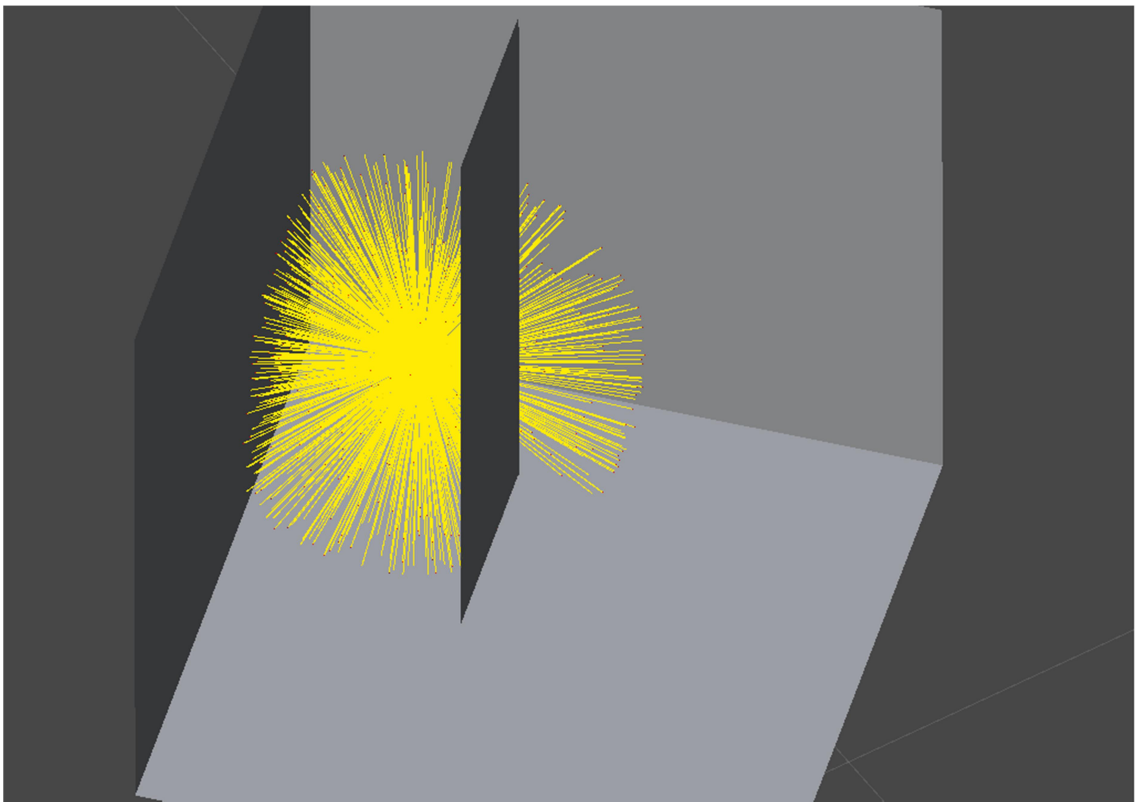


Figure 4.5. An example of a three-dimensional isovist represented as a collection of radial vectors (yellow lines). While some vectors reach a maximum view distance, others hit occluding obstacles and terminate early.

4.5.2 Generating the Isovist

In the proposed approach to generate an isovist, four inputs are required; the *origin* of the isovist, the *maximum view distance*, the isovist's *resolution* (the number of radials), and the *environment* (e.g. terrain). The *origin* is a three-dimensional point that the radial vectors are cast out from, and is set to the centre of an area, which is stored in the corresponding node in the *layout graph*. The height of the *origin* is set to the eye-height of a theoretical character that is standing at the area's centre. This eye-height of a character is a user-specified constant and is set to a value equal to 2.5% of the terrain's width. The *maximum view distance* is the maximum allowed length of a radial, which is a user-specified constant and is set to a value equal to half the terrain's width. The resolution is the number of radials to cast, where greater resolutions result in more accurate measures, but also an increased computation time. The resolution is also a user-specified constant, where this approach uses a resolution of 1024 radials. The environment is used to perform intersection tests with radials to determine if the radial should terminate before reaching the *maximum view distance*. In this case the environment is the terrain.

The process of generating an isovist involves calculating the direction and the length of each radial. Table 4.2 shows pseudocode that outlines this process. The GenerateIsovist function takes the four inputs, creates an isovist, and sets the isovist's origin and radial count. Each of the isovist's radials has its direction randomly generated. A ray is then cast from the isovist's origin, in the generated direction, to determine where it intersects with artefacts in the environment. If the distance between the isovist's origin and the intersection point is less than the *maximum view distance*, then the radials length is set to the distance to the intersection point, otherwise it is set to the *maximum view distance*.

Table 4.2. The pseudocode for generating an isovist.

```
FUNCTION GenerateIsovist(origin, maxViewDistance, resolution, environment)
        isovist = NewIsovist()
        isovist.SetOriginTo(origin)
        isovist.SetRadialCount(resolution)
        FOR EACH radial IN isovist DO
                direction = GetRandomDirection()// See Appendix B.9
                radial.SetDirectionTo(direction)
                distance = RayCast(origin, direction, environment)
                IF distance < maxViewDistance THEN
                        radial.SetLengthTo(distance)
                ELSE
                        radial.SetLengthTo(maxViewDistance)
                END IF
        END FOR
        RETURN isovist
END FUNCTION
```

4.5.3 Calculating Isovist Measures

This section details the calculations of the 11 isovist measures used in this research, which are listed in Table 4.1 (I_1-I_{11}). This collection of measures is selected from those used by Benedikt (1979), Dalton (2001), and van Bilson & Poelman (2009), along with some new measures. These new measures include *Volume* and *Sphericality*, which replace their 2D counterparts, area and circularity, from the above works, and the *Drift 2D* and *Drift 3D* measures, which are variations of the drift measure from Dalton (2001). The rest of this section details the methods of calculating these isovist measures.

The *Minimum Radial Length* and *Maximum Radial Length* measures are the smallest and largest radial lengths of the isovist's radials, respectively. The *Average Radial Length* is the sum of the isovist's radials lengths divided by the number of radials.

Variance is the standard statistical variance calculated using the isovist's radial lengths. To clarify, the method used in this approach calculates *Variance* as the sum of squared

differences between each radial length and the *Average Radial Length*, then divided by the number of radials minus one. Equation 4.1 shows this calculation, where *N* equals the number of isovist radials, r_n is a single isovist radial, and *arl* equals the *Average Radial Length*. The *Standard Deviation* can then be calculated as the square root of the *Variance*.

Equation 4.1

$$Variance = \frac{\sum_{n=1}^{N}(|r_n| - arl)^2}{N - 1}$$

Skewness is calculated in the same manner as *Variance* except for two differences. First the difference between each radial length and the *Average Radial Length* is cubed, not squared; and second, the denominator is the product of the number of radial lengths minus one and the *Standard Deviation* (*sd*) cubed. Equation 4.2 shows this calculation.

Equation 4.2

$$Skewness = \frac{\sum_{n=1}^{N}(|r_n| - arl)^3}{(N - 1) * sd^3}$$

Dispersion is calculated as the *Average Radial Length* minus the *Standard Deviation*. *Sphericality* is calculated as the volume of a sphere, where the sphere's radius is equal to the *Average Radial Length*, divided by the actual volume of the isovist. Equation 4.3 shows this, where *arl* is the *Average Radial Length*, and *v* is the isovist's *Volume*, which is calculated using Equation 4.5.

Equation 4.3

$$Sphericality = \frac{\frac{4\pi}{3}(arl)^3}{v}$$

Drift 3D and *Drift 2D* are calculated as the magnitude of the average of all the isovist's radials. The difference between *Drift 2D* and *Drift 3D* is that *Drift 2D* only uses the x and z components of the isovist radials, while *Drift 3D* includes the y component (the up axis). Equation 4.4 demonstrates this.

Equation 4.4

$$DriftXD = \left|\frac{\sum_{n=1}^{N} r_n}{N}\right|$$

This approach proposes a novel method of estimating an isovist's *Volume*. This method involves calculating the volume of a sphere using the average cubed radial length rather than a sphere's radius cubed. Equation 4.5 demonstrates this. Details on how this novel method works and how it compares to other volume estimation methods are in Section 4.6.

Equation 4.5

$$Volume = \frac{4\pi}{3} \frac{\sum_{n=1}^{N} |r_n|^3}{N}$$

Once the *Area Radius*, the graph measures, and the isovist measures have been calculated for each area (node) in the *layout graph*, then each area in the terrain has been characterised and this process is complete.

4.6 A Novel Method of Estimating the Volume of an Isovist

When working with 3D isovists, volume becomes an important measure. Little existing work has involved using 3D isovists so techniques to calculate the volume of a 3D isovist are limited (Pyssalo *et al*., 2009; Yang *et al*., 2007; Fisher-Gewirtzman & Wagner, 2003; Dalton *et al*., 2015; van Bilsen, 2008; Rodriguez *et al*., 2006; Metropolis & Ulam, 1949). The main problems with current techniques of calculating the volume of an isovist are the complexity of these techniques, and their accuracy. This section describes how the volumes of isovists have previously been calculated, followed by the theoretical basis for the novel volume estimation method used in this approach, and then results from comparing this novel method to existing methods.

4.6.1 Previous Work

Some approaches, such as Pyssalo *et al*. (2009), discretise the visible space into voxels and count these voxels to determine volume. Yang *et al*. (2007) provide a measure of volume by decomposing the isovist volume into a number of triangular fan volumes and summing them. Fisher-Gewirtzman & Wagner (2003) produced a volume-related measure called the Spatial Openness index, determined by summing the 2D areas of a number of vertical and horizontal slices through the volume of visible space. Dalton *et al*. (2015) described two methods, one using the area of three horizontal slices through the isovist, and the other summing the isovist's radial lengths. Results from both of these methods showed a strong correlation with the volumes of the isovists.

In moving towards an accurate estimation of isovist volume, van Bilsen (2008) defined the direction of each radial vector by a pair of angles, defining the volume of the isovist as the continuous double integral across those angles of cubed radial lengths. The problem remains of how to obtain this integral for an isovist volume whose surface cannot be characterised easily by an integral function. One solution to the isovist volume calculation problem is found in the area of 3D laser scanning microscopy. Rodriguez *et al*. (2006) introduced the Rayburst Sampling algorithm where a set of radials are cast from a central point within a structure in a volumetric image towards the surface of that structure. These radials were used to define a triangular surface mesh, and to form a set of pyramidal volumes that, for a star-convex shaped structure, could be summed to determine the volume. A star-convex set (Valentine, 1964) is a shape that is not necessarily convex, but where a line from any point within the shape to the shape's origin is entirely contained within the shape. An isovist is an example of a star-convex set, as it is possible for a ray to be cast from the isovist's origin to any other point within the isovist, without any portion of the ray being outside of the isovist's volume.

Another integration technique suitable in such cases is the Monte Carlo approach (Metropolis & Ulam, 1949), which uses a statistical approach to the calculation of a volume in a multi-dimensional space. Based on a search of the literature, this technique has not been applied to isovist volume measurement before. Monte Carlo integration relies on the unknown volume being enclosed by a known volume and being able to test whether a point is within the unknown volume. A set of random points is taken from within the known volume and the number of these points that also lie inside the unknown volume is determined. The unknown volume is then approximated as the known volume, multiplied by the ratio of points in the unknown volume to the total number of points. As the number of random points approaches infinity, the volume approximation approaches the true volume. For an isovist, the known volume must be large enough to encompass a sphere with a radius equal to the *maximum view distance*. A random point is considered inside the isovist if its distance to isovist's origin is less than the *maximum view distance*, and if there are no occluding obstacles between the point and the isovist's origin. Whilst providing a workable solution, the calculation of isovist volume in this way entails the added computational burden of testing a multitude of random points against the environment.

4.6.2 The Proposed Method

The novel method used in this approach estimates an isovist's volume based on only the set of isovist radial vectors. The algorithm is efficient, as it relies on knowing only the radials, which would in any case need to be obtained to compute other measures of the isovist, such as those discussed in Benedikt (1979) and Dalton *et al*. (2001), in order to characterise the space it represents. The algorithm is also simple, as it does not depend on any relationship between the radials as is needed by techniques that use triangulation, such as in Rodriquez *et al*. (2006).

4.6.3 Theoretical Basis of the Proposed Method

The approximation can be interpreted as using each radial to define a spherical volume where the surface of that sphere touches the isovist surface at the end of that radial. These spherical volumes are then equally weighted, with the sum of weights equal to 1, and summed. For the example of the simplest case, that of a single radial defining the isovist (N=1), the estimated volume, from Equation 4.6, is that of a sphere of radius r_1:

Equation 4.6

$$V_{Isovist_{N=1}} = \frac{4\pi|r_1|^3}{3}$$

For two radials (N=2), the estimated volume is that of two spheres, with radii corresponding to the two radials, each sphere given a weight of 1/N = 0.5:

Equation 4.7

$$V_{Isovist_{N=2}} = 0.5\frac{4\pi|r_1|^3}{3} + 0.5\frac{4\pi|r_2|^3}{3}$$

For N radials, the estimated volume is that of N spheres, each given a weight of 1/N, becoming the generalised equation presented as Equation 4.5.

This technique represents the volume of an isovist, or any volume that has the property of being a star-convex set with respect to the origin of the radials. The volume approximation does not hold for non-convex volumes that are not star-convex, as in such volumes some of the cast out radials will be occluded from interrogating the complete surface of the shape.

4.6.4 Experiments and Results

This novel volume estimation technique, referred to as the *Isovist Method*, is evaluated by comparing it against three other volume estimation techniques (Dalton's sum of radial lengths (*Length Sum Method*), Rodriguez's Rayburst Sampling algorithm, and Monte Carlo integration, described in Section 4.6.1).

The comparison between these four methods is based on results from 2048 repeats of 24 different experiments. Each experiment used the four methods to estimate the volume of an isovist in one of six scenarios at one of four resolutions, where the unique combinations of scenario and resolution formed the 24 experiments.

Each scenario consists of a landscape and an isovist with a *maximum view distance* of 1024. Three of these scenarios contain an isovist whose volume equates to a section of a sphere. These simple scenarios were used since the actual volume of the isovists could be calculated for comparison to the estimations. The other three scenarios are based on more realistic simulated outdoor scenarios. Figure 4.6 displays these three complex scenarios and their associated isovists. Each of these scenarios is situated in a city-block style environment and is labelled (a) Urban, (b) Hidden Area, and (c) Vantage Point. The isovist in the Urban scenario was placed in the middle of a street intersection, while the isovist in the Hidden Area scenario was placed in an alley cul-de-sac. The isovist in the Vantage Point scenario was placed on top of a building.

Due to the complexity of the isovists for these three cases, their volume could not be calculated by any known method and so approximate volumes were calculated using the average estimated volume of 2048 Monte Carlo runs, where each run used 16836 sample points within a box volume, with the dimensions 2560x2560x2560, centred on the isovist's origin. These are the same dimensions used when running Monte Carlo integration in the experiments. Table 4.3 shows the known volumes for the three sphere-based isovists, and the approximated volume (from the 2048 Monte Carlo runs) for the three isovists associated with the three simulated outdoor scenarios.

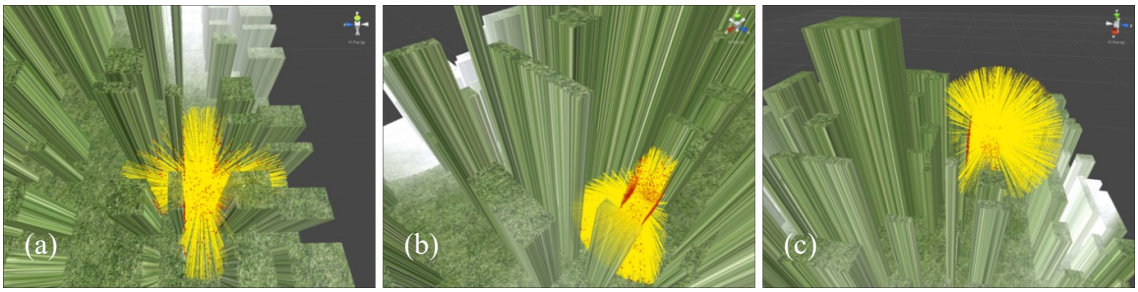


Figure 4.6. Three scenarios, where an isovist has been generated in a complex environment. (a) an isovist generated in the middle of a street in an urban environment. (b) an isovist generated in a hidden alcove. (c) an isovist generated at a vantage point on top of a building.

Table 4.3. The known volumes of the three sphere-based isovists and the approximated volumes of the three complex isovist volumes.

Isovist Shape (Scenario)	**Ground Truth**
Full Sphere	4,497,679,234.84
Sphere Minus Cap (Cap Height = 512)	3,794,916,854.40
Half Sphere	2,248,839,617.42
Terrain Location – Urban	~743,726,500
Terrain Location – Hidden Area	~214,862,000
Terrain Location – Vantage Point	~2,266,592,500

Evaluation of the results from these experiments is divided into three sections. The first section uses Pearson's correlation coefficient (Pearson, 1895) to show that the volume estimations of the *Isovist Method* have a strong relationship to the actual volumes of the isovists in the six scenarios. The second section compares the estimated volumes to the real volumes using box and whisker plots. The third section compares the computation times of each method.

4.6.4.1 Correlation of the Estimated Volumes to the Real Volumes

Dalton *et al*. (2015) stated that the *Length Sum Method* had a strong correlation to the actual isovist volumes. To examine Dalton's claim, Pearson's correlation coefficient (Pearson, 1895) was calculated for all four methods. The result shows that the volume estimations produced using the *Isovist Method* also bear a strong correlation to the actual isovist volumes. Table 4.4 show the calculated coefficients for each method at each of the four resolutions. These values show that the results from each method have a strong correlation to the actual isovist volumes, with higher resolutions showing stronger correlations.

Table 4.4. The Pearson correlation coefficient values for each of the four methods at each of the four resolutions. Each value was calculated using the results from all six scenarios.

	Length Sum	Rayburst	Monte Carlo	Isovist Method
258	0.9924	0.9978	0.9754	0.9979
1026	0.9937	0.9995	0.9933	0.9995
4098	0.9941	0.9999	0.9982	0.9999
16386	0.9942	1.0000	0.9995	1.0000

4.6.4.2 Estimated Volume Results

This section shows the results associated with volume estimation from the 2048 runs of the 24 experiments as a set of box and whisker plots. Figure 4.7 shows the box and whisker plot for the Full Sphere scenario. It is divided into four sections, each associated with one of the four resolutions. Each section shows the minimum, maximum, and interquartile ranges for the four volume estimation methods. The top of the green boxes show the upper quartiles, while the bottom of the red boxes show the lower quartiles, and the median is where the green and red boxes meet. The horizontal, dotted line shows the "*ground truth*" volume that is listed in Table 4.3.

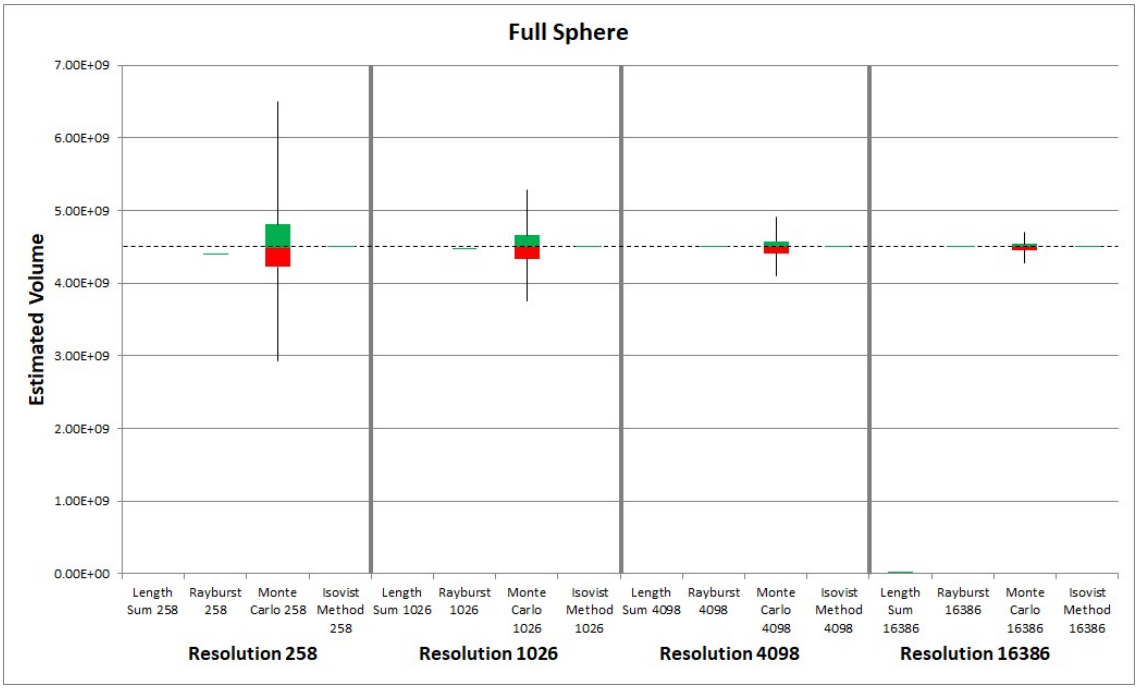


Figure 4.7. The box and whisker plot of the four isovist volume estimation methods for the full-sphere scenario.

Figure 4.7 shows interesting, but not unexpected, results. This plot is associated with the scenario where the isovist is a complete sphere of radius, maximum view distance and have

unobstructed views (i.e. no obstacles). This plot shows that the *Length Sum Method* produces values much lower than the *ground truth* volume – so low in comparison to the real volume that most of them do not display on the chart. This is due to the fact that while this method produces values that have a strong correlation with the real volume, it does not produce accurate volume estimations.

The Rayburst method showed no variance in its volume estimations and only produced accurate volume estimations at high resolutions. It produced less accurate estimations at low resolutions due to the method's inability to fully capture an isovist's volume. Figure 4.8 illustrates why the Rayburst method is less accurate at lower resolutions. This figure shows a 2D example of how the Rayburst method triangulates space. The Rayburst method sums the area of each triangle to calculate the area of the isovist, but this leaves pockets of space that do not have their area included in the calculations. This means that higher resolutions are required to minimise the amount of uncaptured space.

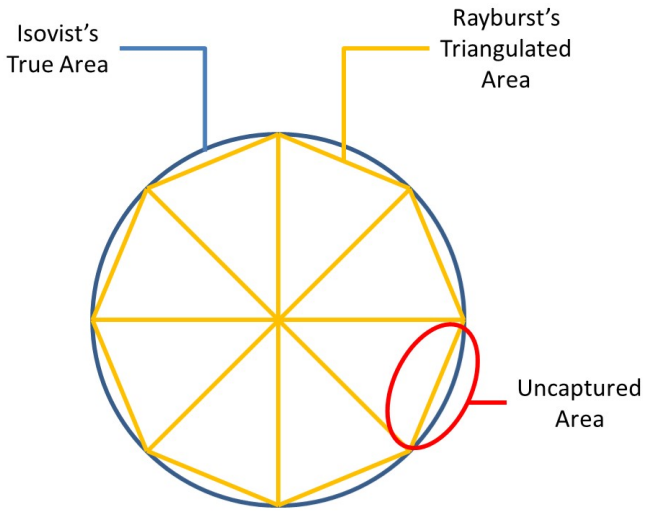


Figure 4.8. Example of how the Rayburst volume estimation method cannot capture an isovist's true volume.

The Monte Carlo method produced accurate median values but also produced a large variation in estimated volumes, specifically at lower resolutions. This is due to an incorrect ratio of randomly sampled points being within the volume of the isovist. This is expected when the number of randomly sampled points is too low, as a large number of points are required to properly sample the space.

The *Isovist Method* estimated the "*ground truth*" volume of the isovist in all of the four resolutions in this scenario. As there are no occluding obstacles, the resulting volume

estimation equates to the average volume of N spheres, where each sphere has a radius equal to the *maximum view distance*.

Figure 4.9, Figure 4.10, Figure 4.11, Figure 4.12, and Figure 4.13 show the box and whisker plots for the estimated isovist volume of the other five scenarios respectively. These figures show that, beside the *Length Sum Method*, the Rayburst Sampling method was the next least accurate in terms of median volume estimations. Even at the highest resolution, Rayburst was typically less accurate than Monte Carlo and the proposed *Isovist Method*, although it does have the least variance in its estimations as shown by its small interquartile range. Monte Carlo integration had produced the greater variance, with the largest interquartile ranges, but produced accurate median values similar to those of the *Isovist Method*. The *Isovist Method* had estimations with smaller interquartile ranges than the Monte Carlo method, making its estimations more reliable, and also produced the most accurate median estimations.

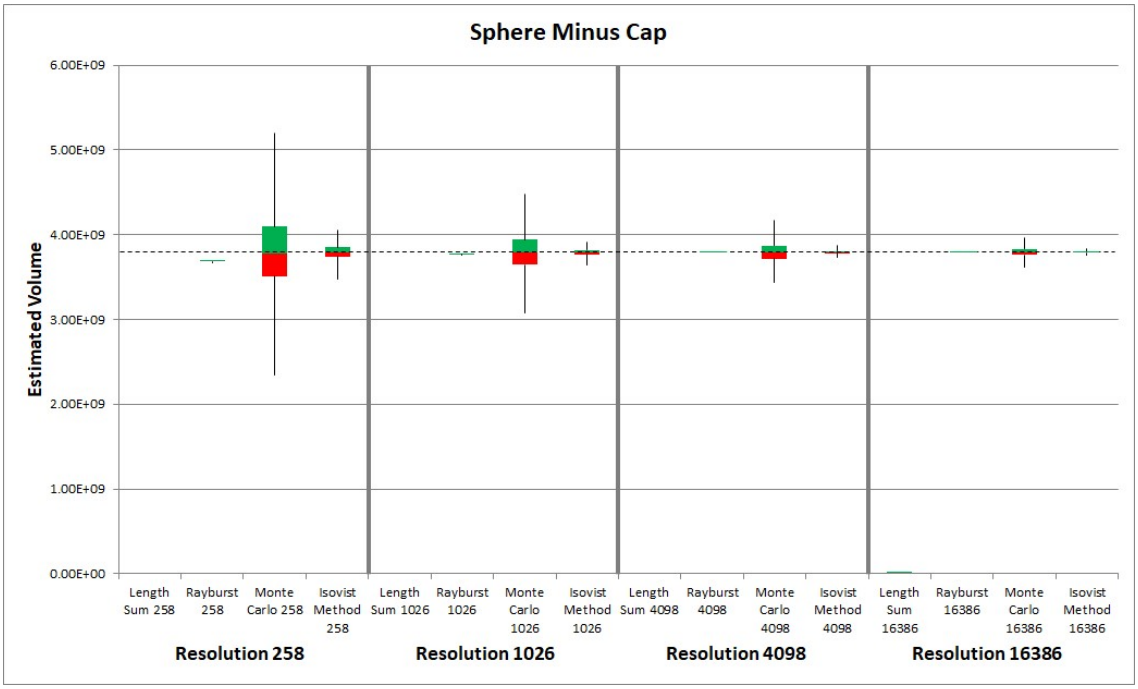


Figure 4.9. The box and whisker plot of the four isovist volume estimation methods for the sphere-cap scenario.

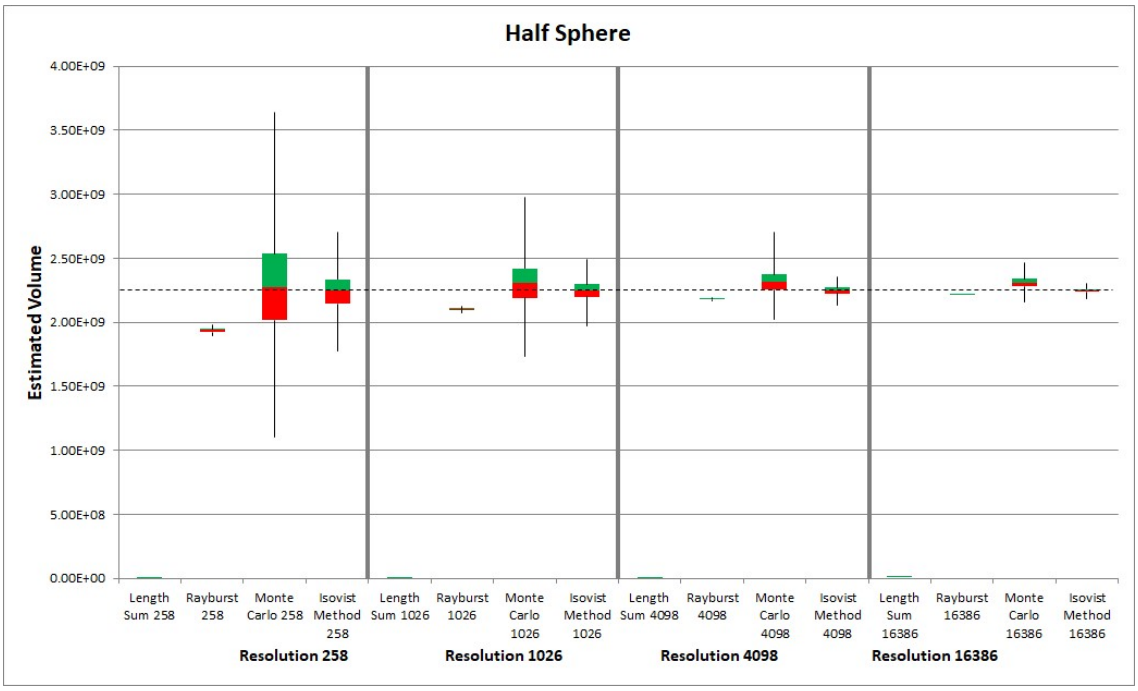


Figure 4.10. The box and whisker plot of the four isovist volume estimation methods for the semi-sphere scenario.

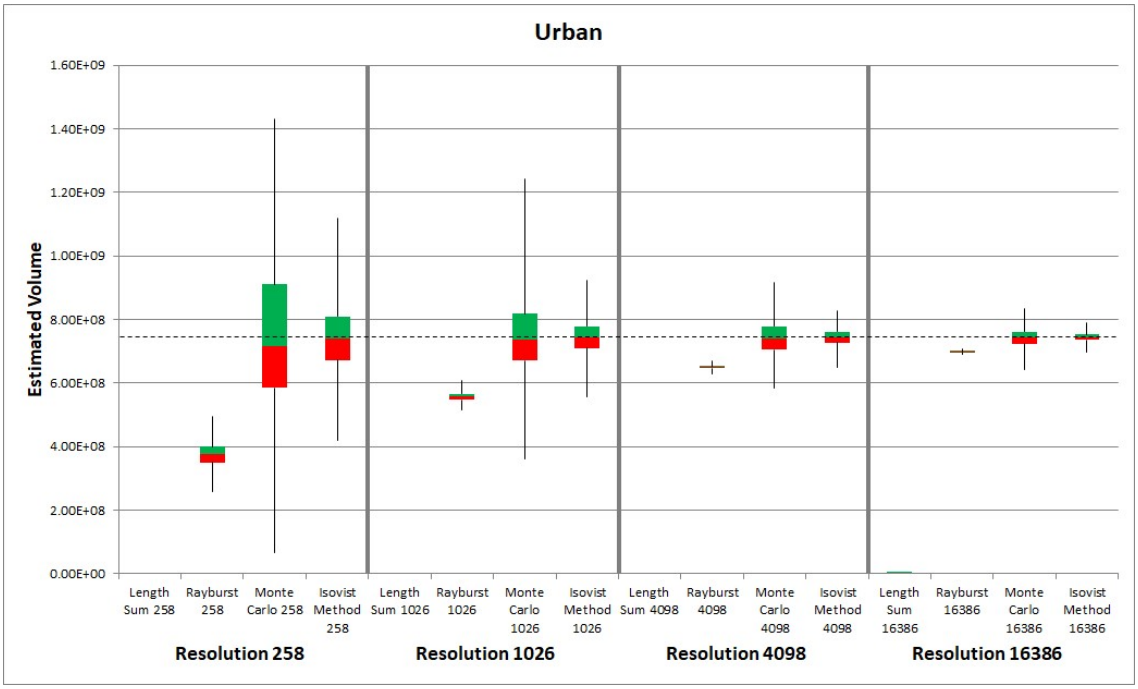


Figure 4.11. The box and whisker plot of the four isovist volume estimation methods for the urban scenario.

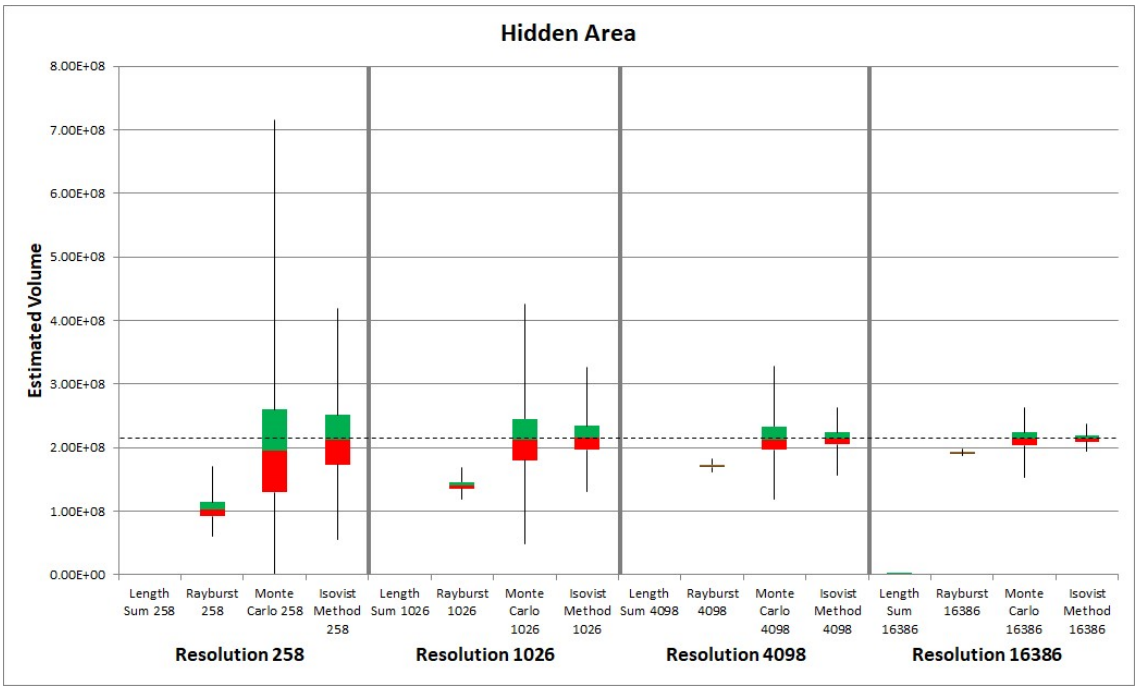


Figure 4.12. The box and whisker plot of the four isovist volume estimation methods for the hidden scenario.

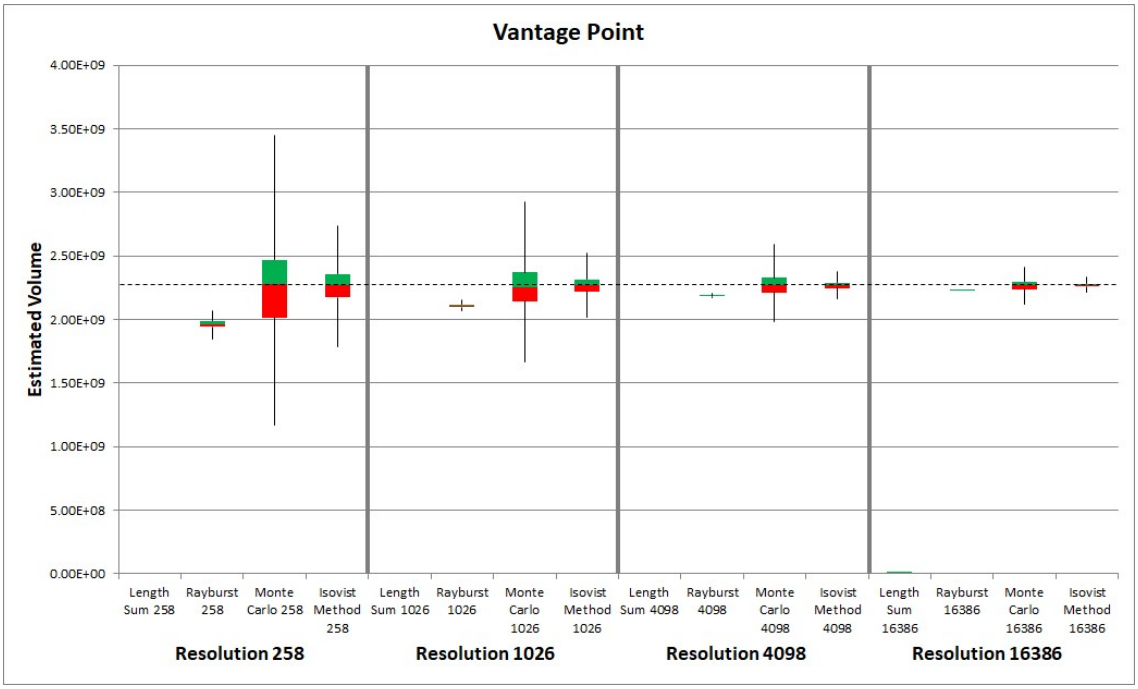


Figure 4.13. The box and whisker plot of the four isovist volume estimation methods for the vantage scenario.

To summarise, the proposed *Isovist Method* produced the most accurate median values with smaller interquartile ranges than Monte Carlo integration. The Rayburst method produced the smallest interquartile ranges, but its volume estimations were less accurate than either the *Isovist Method* or Monte Carlo integration. From the plots it can be seen that as the scenarios increases in terms of the complexity, all three methods (Rayburst, Monte Carlo, and Isovist)

showed more variability in their respective volume estimation at lower resolutions and increasing accuracy as the resolution increases. The proposed *Isovist Method* consistently outperformed the other methods across the four resolutions and scenarios with increasing complexity. Dalton's method was not meant to be a volume estimation method and was included here just for completeness.

4.6.4.3 Efficiency Comparisons Between Volume Estimation Methods

Besides evaluating the methods in terms of their accuracy in volume estimation, computation efficiency of these methods is examined. Table 4.5 shows the run time for each experiment, with the fastest run times bolded. These time values were obtained by averaging the time (in seconds) it took to run the 2048 repeats of each method for each experiment.

From the table, Monte Carlo integration was typically the fastest method, on average taking just under half the time of the next fastest method. The *Length Sum Method* and the *Isovist Method* had similar computation times, while the Rayburst method was the slowest, especially at high resolutions. At resolution 16386 the Rayburst method took, on average, 7.6836 seconds to calculate the volume of a single isovist, compared to the *Isovist Method*, *Length Sum Method*, and Monte Carlo method, which took 0.261, 0.259, and 0.1122 seconds respectively.

To make the time comparisons fair, each volume estimation method was given the same four inputs to estimate each isovist's volume. These four inputs include the isovist's *origin*, the *resolution*, the *maximum view distance*, and the *environment*. Except for the Monte Carlo integration method, the other three methods required an isovist to first be generated before the estimation of its volume. This requirement is a contributing factor as to why the other three methods have a bigger computation time. Thus, in an event requiring computation of various isovist measures in addition to volume, the Monte Carlo method may not be the most suitable approach from a computation efficiency perspective.

Table 4.5. The average times (in seconds) each method took, at each resolution and for each scenario, to calculate an isovists volume. The bolded values highlight the best performing method for each experiment.

Full-Sphere				
	Length Sum	Rayburst	Monte Carlo	Isovist Method
258	0.000198	0.002213	**7.08E-05**	0.000218
1026	0.00078	0.030278	**0.000281**	0.000853
4098	0.003122	0.468179	**0.001096**	0.003365
16386	0.012447	7.432197	**0.00438**	0.013526

Sphere-Cap (h = 512)				
	Length Sum	Rayburst	Monte Carlo	Isovist Method
258	0.003484	0.005513	**0.000206**	0.003572
1026	0.013707	0.043648	**0.000826**	0.013888
4098	0.054784	0.522087	**0.003287**	0.055548
16386	0.219912	7.670553	**0.013045**	0.22061

Semi-Sphere				
	Length Sum	Rayburst	Monte Carlo	Isovist Method
258	**0.000336**	0.002311	0.002448	0.00036
1026	**0.001336**	0.030707	0.009584	0.001436
4098	**0.005351**	0.469202	0.038381	0.005721
16386	**0.021268**	7.436102	0.153449	0.02297

Terrain Location - Urban				
	Length Sum	Rayburst	Monte Carlo	Isovist Method
258	0.004826	0.006883	**0.002111**	0.004939
1026	0.019404	0.048748	**0.008355**	0.019558
4098	0.077398	0.54185	**0.033368**	0.078086
16386	0.309687	7.734353	**0.133455**	0.312255

Terrain Location - Hidden				
	Length Sum	Rayburst	Monte Carlo	Isovist Method
258	0.003298	0.005207	**0.002813**	0.00329
1026	0.012904	0.042288	**0.011044**	0.013095
4098	0.051607	0.515894	**0.044197**	0.052336
16386	0.20653	7.656483	**0.176588**	0.209071

Terrain Location - Vantage				
	Length Sum	Rayburst	Monte Carlo	Isovist Method
258	0.012358	0.014505	**0.003027**	0.012409
1026	0.049322	0.079331	**0.012018**	0.049455
4098	0.196205	0.661852	**0.048133**	0.197227
16386	0.784001	8.171829	**0.192497**	0.787814

To summarise the comparison of the four volume estimation methods; analysis of the results from 24 experiments show that the *Isovist Method* was typically the most accurate method, producing accurate median values with less variation in volume estimations than the Monte Carlo method. The Rayburst method had the least variance in volume estimations, but generally produced less accurate results unless a high resolution was used. Using higher resolutions with the Rayburst method greatly impacted the computation time, while the computation times of the other methods were much less. Monte Carlo integration was timed as being the fastest method, as it does not require an isovist to be generated. So in cases where an isovist must be generated to calculate some of its other measures, the *Isovist Method* would be more appropriate.

4.7 Summary

This chapter described the collection of measures used to characterise an area of terrain, which is important for the automatic evaluation of an area's suitability for specific gameplay purposes. These measures consisted of isovist measures, graph-connectivity measures, and the *Area Radius* measure. This chapter detailed how these measures are calculated, which included details on how to generate an isovist using the representation described by Benedikt (1979). A novel, efficient method for estimating the volume of an isovist was also described, with experiments and analysis comparing the proposed method's performance against three other known methods associated with isovist volume estimation. This analysis revealed that the proposed volume estimation method is typically more accurate than the other methods and is also efficient to compute.

Chapter 5. Classifying Area Types Using Isovist Measures

This chapter details the experiments conducted to determine the effectiveness of the proposed measures described in Table 4.1 for characterising specific area types. The investigation involved using supervised learning techniques to train classifiers using a dataset where each instance belongs to a specific area type with a set of the 16 measures. The hypothesis is that, if classifiers built using the proposed set of measures as relevant features can differentiate different categories of area types with good classification results, then the set of proposed measures is effective for characterising the different area types. This chapter is an extended version of the paper "Identifying Attributes for Characterizing Game Area Types in Virtual Terrain" (Pech, 2016a).

The investigation involved training four different classifiers using a dataset consisting of 70 instances, each associated with one of the five area types. Section 5.1 details the data set used in this research and how it was generated. Sections 5.2, 5.3, 5.4, and 5.5 detail the results of running the dataset through the four classifiers; a J48 Decision Tree, Naïve Bayes, a Multilayer Perceptron, and a Probabilistic Neural Network (PNN).

5.1 The Dataset

The dataset contains a collection of area instances of each of the five area types that were used in this research; hidden area, open area, vantage point, choke point, and stronghold. Table 5.1 lists the number of area instances of each area type. While class imbalance can be a problem for any classifier, due to the number of instances of each class in this dataset, the problem of class imbalance is potentially minimal.

Table 5.1. The number of instances for each area type stored in the area type data set.

Area Type	Number of Instances
Hidden Area	11
Open Area	15
Vantage Point	19
Choke Point	15
Stronghold	10

The dataset was generated by following three steps.

Step 1: Identify existing game levels that contain instances of chosen area types in this study. The dataset used publicly available, player-made maps for the video game "Savage: The Battle for Newerth" (S2 Games, 2003), which were obtained from the website "http://www.newerth.com/".

Step 2: Generate a *layout graph* for each of the selected game levels. These *layout graph*s were manually generated by placing nodes at each manually identified area of the map, then adding edges between areas that had a physically traversable path between them. Figure 5.1 shows an example of a player-made game level for the video game "Savage", with an overlayed, manually generated, *layout graph*. Each green sphere shows a node of the *layout graph*, which corresponds to an area of the game level.

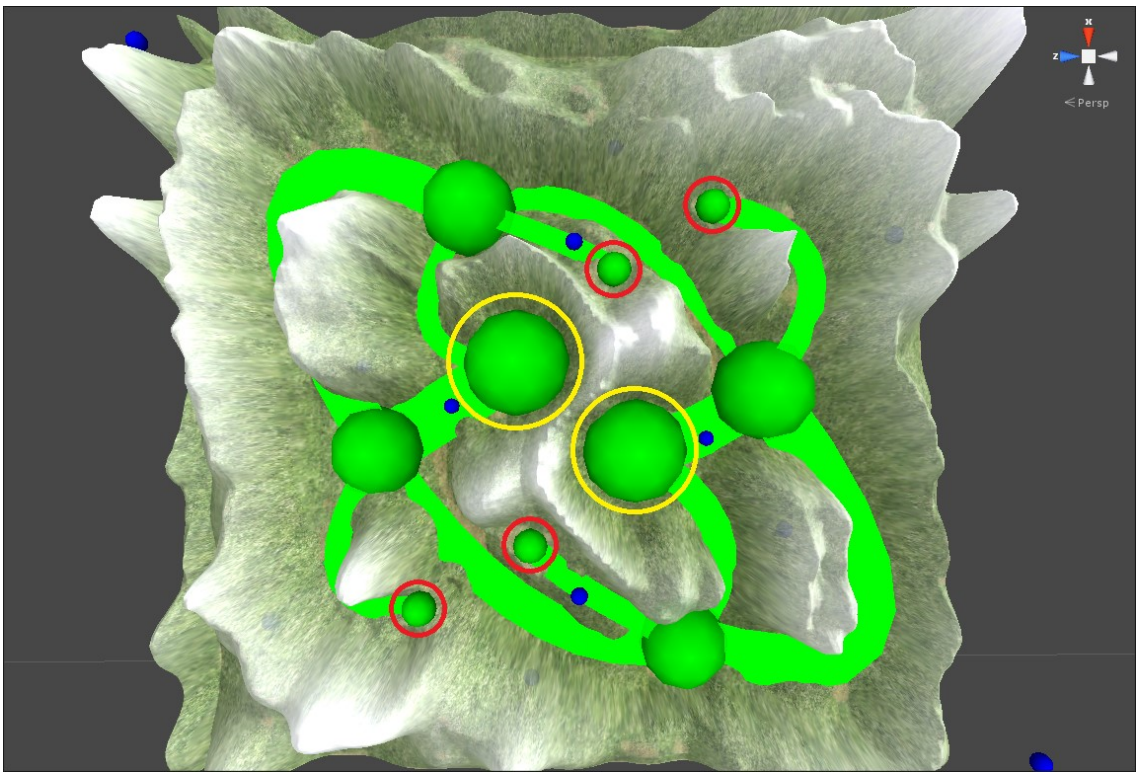


Figure 5.1. An example of a player-made game level for the video game "Savage" with an overlayed layout graph. The circled nodes indicate manually identified vantage points (red) and strongholds (yellow). Image taken from Pech *et al*. (2016a).

Step 3: Manually identify areas of specific types (e.g. vantage points and strongholds) and calculate the set of isovist and graph measures for each area instance, adding the instance to the dataset. These isovist and graph measures were calculated using the methods described in Chapter 4, with the exception of the *Area Radius* measure. As the areas of the game level were manually identified and represented in the *layout graph* as a sphere of appropriate size,

the *Area Radius* measure of an area was manually set to the given area's associated sphere's radius. The circled nodes in Figure 5.1 indicate areas that were manually identified as a specific area type, red circles indicating vantage points, and yellow circles indicating strongholds.

Table 5.2 shows physical descriptions of the five area types. These descriptions were obtained from Hullett & Whitehead (2010) and were used to manually identify instances of the different area types in a terrain. For example, Hullett & Whitehead describe a vantage point, which they call a sniper location, as an elevated position which overlooks a portion of the game level. Therefore all manually identified vantage points in this dataset meet these criteria. Figure 5.2 shows an example of a manually identified vantage point. As shown, it is located at an elevated position and has a good view over a significant portion of the game level.

Table 5.2. A list of the area types that were explored in this research, with descriptions taken from Hullett & Whitehead (2010).

Area Type	**Description**
Hidden Area	Small, well-occluded area, usually off the main route.
Open Area	A large, open area usually used for large-scale combat in FPS style games.
Vantage Point	An elevated location which looks over a portion of the game level.
Choke Point	A small area which must be traversed to reach a portion of the game level.
Stronghold	A well-covered area with limited access points.

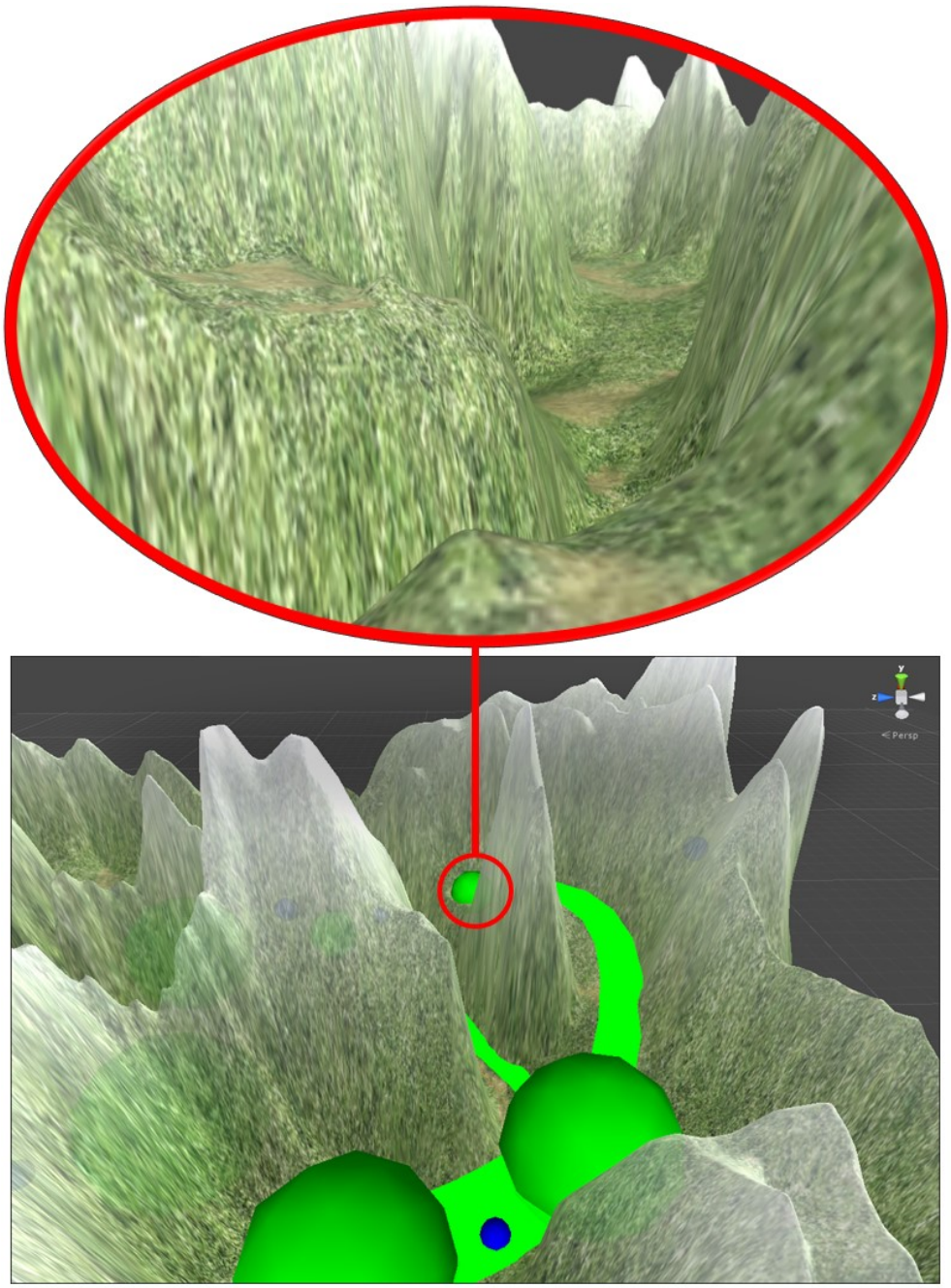


Figure 5.2. An example of a manually identified vantage point meeting the criteria described by Hullett & Whitehead (2010). It is an elevated position and, as shown in the circled image, overlooks a significant portion of the game level. Image taken from Pech *et al*. (2016a).

Once obtained, the generated dataset was used to train four different types of classifiers; a J48 decision tree, naïve Bayes, a multilayer perceptron, and a PNN. These classifiers were chosen as they are each a different type of classifier (i.e. decision tree, probabilistic, black box) and one may be better suited to the problem than others. The first three of these experiments were conducted using the open source machine learning software "Weka", and since Weka does not include a PNN, the open source Python library "NeuPy" was used to run the last

experiment. A 10-fold cross validation was employed to estimate the accuracy of each classifier and to evaluate how well the measures characterise an area as a specific type.

5.2 Evaluation of Measures Using the J48 Decision Tree

J48 is Weka's version of a C4.5 decision tree (Quinlan, 1992) and has a number of adjustable settings that it can be run with. This experiment used Weka's default settings to generate a pruned decision tree. Table 5.3 and Table 5.4 display the classification results and the confusion matrix respectively. When using 10-fold cross validation in Weka the output confusion matrix is formed by generating a confusion matrix for each of the 10 folds and summing their values. Therefore if two of the 10 folds misclassify a single open area as a stronghold, the output confusion matrix will state that two open areas were misclassified as strongholds.

Table 5.3. The classification results from the J48 decision tree experiment.

Correctly Classified Instances	59	84.2857%
Incorrectly Classified Instances	11	15.7134%
Kappa Statistic	0.8014	
Total Number of Instances	70	

Table 5.4. The confusion matrix for the J48 decision tree experiment.

Hidden Area	Open Area	Vantage Point	Choke Point	Stronghold	<-Classified As
10	0	1	0	0	Hidden Area
0	12	0	1	2	Open Area
1	0	18	0	0	Vantage Point
1	0	0	12	2	Choke Point
0	3	0	0	7	Stronghold

As shown in Table 5.3, this classifier has an accuracy rating of 84.29%, correctly classifying 59 out of 70 instances. Some of the misclassified areas could be attributed to the fact that areas of different types share some similar properties. For example, because two open areas were misclassified as strongholds and three strongholds were misclassified as open areas, open areas and strongholds likely share some common properties. This was investigated further by analysing the 10 trees generated during a 10-fold cross validation.

The attributes of the misclassified areas are compared to the rules of the generated decision trees to identify the reason for them being misclassified. Figure 5.3 shows an example of one of the generated trees, but due to the variation in the generated trees across the 10 folds, the tree-specific rules for correctly classifying each of the misclassified areas are listed in Table 5.5. Table 5.6 lists the specific attribute values of the misclassified areas. The values in the ID column of Table 5.5 and represent each areas unique identifier so that they may be referenced in the text. The colours in indicate the actual area type of each area, where yellow indicates a hidden area, green indicates an open area, red indicates a vantage point, blue indicates a choke point, and orange indicates a stronghold.

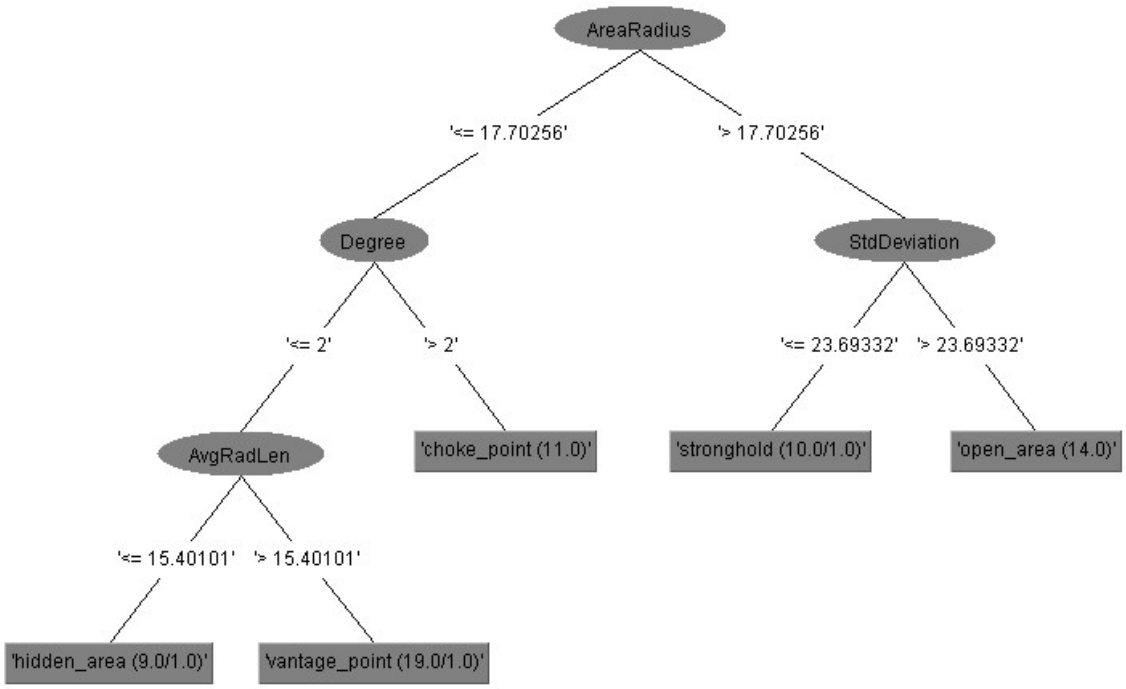


Figure 5.3. A visual representation of one of the 10 decision trees generated during 10-fold cross validation.

Table 5.5. List of rules required to correctly classify each of the misclassified areas in this experiment.

ID	Actual Type	Classified As	Rule
0	Hidden Area	Vantage Point	*Area Radius* <= 17.70256 *Degree Centrality* <= 2 *Average Radial Length* <= 15.40101
11	Open Area	Choke Point	*Area Radius* > 24.32166 *Drift 3D* > 12.14439
18	Open Area	Stronghold	*Area Radius* <= 17.70256 *Drift 3D* > 12.38895
20	Open Area	Stronghold	*Area Radius* > 17.70256 [*Average Radial Length* > 27.42681 OR *Area Radius* <= 26.79486]
36	Vantage Point	Hidden Area	*Area Radius* <= 17.70256 *Degree Centrality* <= 2 *Skewness* <= 0.987486
52	Choke Point	Hidden Area	*Area Radius* <= 17.70256 *Degree Centrality* > 2
57	Choke Point	Stronghold	*Area Radius* <= 16.80288 *Degree Centrality* > 2
58	Choke Point	Stronghold	*Area Radius* <= 17.70256 *Degree Centrality* > 2
61	Stronghold	Open Area	*Area Radius* > 17.70256 *Standard Deviation* <= 23.69332
64	Stronghold	Open Area	*Area Radius* > 24.32166 *Drift 3D* <= 12.14439
69	Stronghold	Open Area	*Area Radius* > 17.70256 *Skewness* > 0.776416

Table 5.6. The attributes of the misclassified instances of this experiment. The number in the left column is the instances unique ID and the colours represent the instances actual area type. Yellow = Hidden Area, Green = Open Area, Red = Vantage Point, Blue = Choke Point, Orange = Stronghold.

ID	Attributes							
	Area Rad	*Avg Len*	*Disp*	*Drift3D*	*Drift2D*	*Max Len*	*Min Len*	*Skew*
0	14.20	22.38	-2.38	11.39	0.98	64	3.2001	1.0424
	Sphere	*STD*	*Var*	*Vol*	*Deg*	*Eigen*	*Close*	*Between*

	0.1644	24.76	612.89	285497	1	0.57	0.0070	0
11	*Area Rad*	*Avg Len*	*Disp*	*Drift3D*	*Drift2D*	*Max Len*	*Min Len*	*Skew*
	20.59	28.45	2.91	13.30	5.40	64	3.2001	0.5383
	Sphere	*STD*	*Var*	*Vol*	*Deg*	*Eigen*	*Close*	*Between*
	0.2626	25.54	652.29	367168	3	1.79	0.0120	0.6428
18	*Area Rad*	*Avg Len*	*Disp*	*Drift3D*	*Drift2D*	*Max Len*	*Min Len*	*Skew*
	26.79	26.04	1.95	12.34	4.41	64	3.2001	0.7764
	Sphere	*STD*	*Var*	*Vol*	*Deg*	*Eigen*	*Close*	*Between*
	0.2391	24.09	580.55	309502	3	2.64	0.0099	0.2500
20	*Area Rad*	*Avg Len*	*Disp*	*Drift3D*	*Drift2D*	*Max Len*	*Min Len*	*Skew*
	32.85	27.15	3.23	12.42	1.37	64	3.2001	0.7026
	Sphere	*STD*	*Var*	*Vol*	*Deg*	*Eigen*	*Close*	*Between*
	0.2625	23.92	572.23	319349	3	2.02	0.0112	0.3214
36	*Area Rad*	*Avg Len*	*Disp*	*Drift3D*	*Drift2D*	*Max Len*	*Min Len*	*Skew*
	4.54	19.06	-5.42	9.17	6.72	64	0.8424	1.1368
	Sphere	*STD*	*Var*	*Vol*	*Deg*	*Eigen*	*Close*	*Between*
	0.1196	24.48	599.21	242306	1	0.57	0.0052	0
52	*Area Rad*	*Avg Len*	*Disp*	*Drift3D*	*Drift2D*	*Max Len*	*Min Len*	*Skew*
	8.10	15.35	-3.41	6.89	0.15	64	3.2001	1.8909
	Sphere	*STD*	*Var*	*Vol*	*Deg*	*Eigen*	*Close*	*Between*
	0.1120	18.77	352.16	135452	2	1.13	0.0128	1.2857
57	*Area Rad*	*Avg Len*	*Disp*	*Drift3D*	*Drift2D*	*Max Len*	*Min Len*	*Skew*
	17.70	24.70	1.34	11.32	1.13	64	3.1979	0.8827
	Sphere	*STD*	*Var*	*Vol*	*Deg*	*Eigen*	*Close*	*Between*
	0.2257	23.36	545.65	279533	4	3.44	0.0076	3.1071
58	*Area Rad*	*Avg Len*	*Disp*	*Drift3D*	*Drift2D*	*Max Len*	*Min Len*	*Skew*
	24.32	23.86	1.48	11.11	1.05	64	3.1884	0.9702
	Sphere	*STD*	*Var*	*Vol*	*Deg*	*Eigen*	*Close*	*Between*
	0.2253	22.38	500.70	252512	3	1.90	0.0053	1.1250
61	*Area Rad*	*Avg Len*	*Disp*	*Drift3D*	*Drift2D*	*Max Len*	*Min Len*	*Skew*
	27.96	26.37	2.14	12.14	1.34	64	3.2001	0.7886
	Sphere	*STD*	*Var*	*Vol*	*Deg*	*Eigen*	*Close*	*Between*
	0.2414	24.22	586.69	318098	3	2.09	0.0120	1.2679
64	*Area Rad*	*Avg Len*	*Disp*	*Drift3D*	*Drift2D*	*Max Len*	*Min Len*	*Skew*
	37.29	27.37	3.84	12.39	1.65	64	3.1860	0.6895
	Sphere	*STD*	*Var*	*Vol*	*Deg*	*Eigen*	*Close*	*Between*
	0.2736	23.52	553.45	313786	2	1.03	0.0140	0.5714
69	*Area Rad*	*Avg Len*	*Disp*	*Drift3D*	*Drift2D*	*Max Len*	*Min Len*	*Skew*
	44.12	27.43	5.00	12.06	1.63	64	3.2001	0.6609
	Sphere	*STD*	*Var*	*Vol*	*Deg*	*Eigen*	*Close*	*Between*
	0.2970	22.43	502.92	290975	2	1.74	0.0052	0.2857

The hidden area had an appropriate *Area Radius* of 14.2 and *Degree Centrality* of 1, but had a larger than usual *Average Radial Length* of 22.38 causing it to be misclassified as a vantage point. The first open area was misclassified as a choke point due to having an *Area Radius* of 20.59, which was less than the rule's threshold of 24.32166. Although this was a higher threshold than most of the trees which used a threshold of 17.70256, in which case the open area would have been classified correctly. The following two open areas 18 and 20 where misclassified as strongholds, with area 18 being misclassified due to a *Drift 3D* attribute value of 12.34, slightly less than the rule's threshold of 12.38895. The rule that misclassified area 20 was different than the rule used to classify area 18, which incorrectly classified the open area due to the area having an *Average Radial Length* of 27.15, slightly less than the threshold of 27.42681, and an *Area Radius* of 32.85, larger than the threshold of 26.79486.

The vantage point, area 36, had an appropriate *Area Radius* and *Degree Centrality*, but was misclassified as a hidden area due to a *Skewness* attribute value of 1.1368, slightly higher than the rule's threshold of 0.987486. Of the three misclassified choke points, 52, 57, and 58, the first is the only choke point in the dataset with a *Degree Centrality* less than 3, causing it to be misclassified. The following two choke points were misclassified because their *Area Radius* attribute values of 17.70 and 24.32 were larger than the rules' thresholds of 16.80288 and 17.70256 respectively. The three strongholds, 61, 64, and 69 were misclassified for different reasons as they were each classified using different rules. Stronghold 61 was misclassified because its *Standard Deviation* of 24.22 was slightly larger than the threshold of 23.69332. Stronghold 64 was misclassified because its *Drift 3D* of 12.39 was slightly larger than the threshold of 12.14439. Lastly, stronghold 69 was misclassified because its *Skewness* of 0.6609 was slightly less than the threshold of 0.776416.

Most of these misclassifications occurred due to small differences in the defining attributes, which is evidence that 1) multiple area types share defining attributes, and 2) multiple area types have similar values for their shared defining attributes. An example is the open area and stronghold area types. A key defining attribute for both of these area types is the *Area Radius*, of which both the open area and stronghold share a similar value for.

5.3 Evaluation of Measures Using the Naïve Bayes Classifier

Naïve Bayes (Bayes, 1763) is a common probabilistic classifier that Weka offers. Again, this experiment was run using Weka's default settings. Table 5.7 and Table 5.8 display the classification results and confusion matrix respectively.

Table 5.7. The classification results from the Naïve Bayes experiment.

Correctly Classified Instances	60	85.7143%
Incorrectly Classified Instances	10	14.2857%
Kappa Statistic	0.8194	
Total Number of Instances	70	

Table 5.8. The confusion matrix from the Naïve Bayes experiment.

Hidden Area	Open Area	Vantage Point	Choke Point	Stronghold	<-Classified As
10	0	1	0	0	Hidden Area
0	12	0	1	2	Open Area
1	1	16	1	0	Vantage Point
0	0	0	14	1	Choke Point
0	0	0	2	8	Stronghold

This classifier performed slightly better than the J48 decision tree, with a classification accuracy of 85.71%. Six of the incorrectly classified areas were the same areas that were misclassified in the J48 experiment, with five of these six being misclassified as the same area type across both experiments. The similarities are as follows; one hidden area misclassified as a vantage point, two open areas misclassified as strongholds, one vantage point misclassified as a hidden area, and one choke point misclassified as a stronghold. There was also a stronghold that got misclassified in both experiments. It was classified as an open area in the J48 experiment and as a choke point in this experiment.

5.4 Evaluation of Measures Using the Multilayer Perceptron

The multilayer perceptron (Rumelhart *et al.*, 1986) in Weka is an artificial neural network (ANN) that is trained using back propagation. This experiment used Weka's default settings, which generated an ANN with a single hidden layer consisting of 10 nodes. Table 5.9 and Table 5.10 display the classification results and the confusion matrix respectively.

Table 5.9. The classification results from the Multilayer Perceptron experiment.

Correctly Classified Instances	61	87.1429%
Incorrectly Classified Instances	9	12.8571%
Kappa Statistic	0.8373	
Total Number of Instances	70	

Table 5.10. The confusion matrix from the Multilayer Perceptron experiment.

Hidden Area	Open Area	Vantage Point	Choke Point	Stronghold	<-Classified As
10	0	1	0	0	Hidden Area
0	12	0	1	2	Open Area
0	0	18	1	0	Vantage Point
1	0	0	13	1	Choke Point
0	1	0	1	8	Stronghold

This classifier performed better than the previous two classifiers with an accuracy of 87.14%. Again, many of the misclassified instances in this experiment were the same instances that were misclassified in the previous two experiments. The hidden area that was misclassified as a vantage point is the same hidden area that was misclassified previously. The open area that was misclassified as a choke point is the same one in the naïve Bayes experiment. This experiment and the naïve Bayes also share a vantage point and a stronghold that were misclassified as choke points. This and the J48 experiment both misclassified the same choke point as a hidden area, and all three experiments misclassified the same hidden area, open area, and choke point as a vantage point and two strongholds respectively.

5.5 Evaluation of Measures Using the PNN

The PNN (Specht, 1990) is a feedforward neural network, which uses a distance measure between the input data and the instances contained in the training dataset along with a radial basis function (RBF) to classify the input data. This experiment used the commonly used Euclidean distance function with a Gaussian RBF, which requires a sigma value as input. Due to the PNN using the Euclidean distance between the attributes of the input data and the attributes of each instance contained in the training dataset, a normalised dataset had to be used. If the data was not normalised then attributes with larger value ranges would have a greater impact on the resulting distance, even though attributes with smaller value ranges may be more indicative of area type.

As the PNN is a distance-based model the training dataset must be normalised during the PNN's construction. Standardisation (Mendenhall & Sincich, 2015) was chosen in place of normalisation as it still results in scale invariance of attributes but also allows comparisons of attributes from different area types, as different area types may have different normal distributions of the same attribute. This method requires a mean and a standard deviation value for each attribute of an area instance (i.e. for each isovist/graph measure). The mean

and standard deviation values are calculated for each collection of area instances (A) belonging to a specific area type. This divides the training dataset into five collections of areas, the hidden areas A_{HA}, the open areas A_{OA}, the vantage points A_{VP}, the choke points A_{CP}, and the strongholds A_{SH}. Therefore a set of mean (M) and standard deviation (S) values, where the set contains a mean/standard deviation value for each attribute, was calculated for each area type to form the sets M_{HA} and S_{HA} for hidden areas, M_{OA} and S_{OA} for open areas, M_{VP} and S_{VP} for vantage points, M_{CP} and S_{CP} for choke points, and M_{SH} and S_{SH} for strongholds. Equation 5.1 and Equation 5.2 show how the mean and standard deviation values are calculated for one of the area type collections (A_{HA}), where $A_{HA_i}$is a single area instance in the collection A_{HA}, I is the number of area instances in A_{HA}, n is the current attribute value (i.e. the value for Average Radial Length), and N is the number of attributes.

Equation 5.1

$$M_{HA_n} = \frac{\sum_{i=1}^{I} A_{HA_{i_n}}}{N}$$

Equation 5.2

$$S_{HA_n} = \sqrt{\frac{\sum_{i=1}^{I} A_{HA_{i_n}}{}^2}{N-1}}$$

Once the mean and standard deviation values have been calculated, the attribute values for each area instance in each area collection are normalised. Equation 5.3 shows how the attribute values for each area instance contained in A_{HA} are standardised.

Equation 5.3

$$A_{HA_{i_n}} = \frac{A_{HA_{i_n}} - M_{HA_n}}{S_{HA_n}}$$

During the classification phase of the PNN the area type of the instances to be classified are unknown, therefore a standardised version of each area instance must be calculated for each area type. This creates five standardised versions of the area instance (E) that is to be classified. These five standardised instances are labelled E_{HA}, E_{OA}, E_{VP}, E_{CP}, and E_{SH}, where each standardised instance is calculated using the mean and standard deviation values for its associated area type. Equation 5.4 shows how each attribute (n) of the area instance (E) is standardised for one of the area types (HA).

Equation 5.4

$$E_{HA_n} = \frac{E_n - M_{HA_n}}{S_{HA_n}}$$

In the hidden layer of the PNN, a distance measure is calculated between each instance contained in the training dataset and the standardised version of E that is associated with the area type of the training instance. The rest of the PNN functions in a standard fashion.

The PNN was chosen as the fourth classifier because it classifies areas by calculating a form of similarity measure between the input area and each of the area types. This similarity measure is useful to help guide the GA-based approach, detailed in Chapter 6, towards generating terrain with more desirable areas.

Due to the role the PNN plays in the approach, its one input parameter, the sigma value, was tuned to get the optimal value. Parameter tuning was conducted by performing 10-fold cross validation 100 times for each of 500 sigma values, where the sigma values ranged between 0.01 and 5.0 at increments of 0.01. The average observed accuracy for each sigma value was compared, where the sigma value of 1.7 resulted in the highest average observed accuracy of 87.53% and is therefore used in this experiment. Table 5.11 and Table 5.12 display the classification results and confusion matrix respectively.

Table 5.11. The classification results from the PNN experiment.

Correctly Classified Instances	60	85.7143%
Incorrectly Classified Instances	10	14.2857%
Kappa Statistic	0.8183	
Total Number of Instances	70	

Table 5.12. The confusion matrix from the PNN experiment.

Hidden Area	Open Area	Vantage Point	Choke Point	Stronghold	<-Classified As
10	0	1	0	0	Hidden Area
0	14	0	1	0	Open Area
1	1	16	1	0	Vantage Point
0	0	2	12	0	Choke Point
0	3	0	1	7	Stronghold

This classifier performed similarly to the J48 and Naïve Bayes experiment with an accuracy of 85.71%. From analysing the individual folds of the 10-fold cross validation it was discovered that six of the misclassified instances were also misclassified in the previous experiments. All four classifiers misclassified the same hidden as a vantage point. The naïve Bayes, multilayer perceptron, and this experiment all misclassified the same open area, vantage point, and stronghold as choke points. The stronghold that was misclassified as an open area in the multilayer perceptron experiment was also misclassified as an open area in this experiment. Additionally one of the strongholds that were misclassified as an open area in the J48 experiment was also misclassified as an open area in this experiment.

5.6 Conclusion and Discussion

This chapter presented the results of four classifiers training on a dataset consisting of 70 area instances. The purpose of these classification experiments was to determine if areas could be identified as a specific area type from the 16 attributes used to characterise them. The results show that all of the classifiers performed well, with observed classification accuracies ranging from 84.29% to 87.14%. These accuracy ratings are considered high as the expected accuracy of a random classifier would be approximately 20%. This suggests that the measures used in this research to characterise areas are strong indicators of an area's type.

Although misclassification is not ideal, there are reasonable explanations for why some areas were misclassified in the four experiments. Some area types share similar properties, such as open areas and stronghold both being large areas. Due to these similarities some areas may be suitable as multiple area types. For example, a hidden area may also meet the criteria of a vantage point. This increases the difficulty for a classifier to correctly classify every instance. Based on the analysis of these results, the measures presented in Chapter 4 will be considered sufficiently effective at characterising area types in the context of the proposed evolutionary approach.

Chapter 6 is not included in this version of the thesis

Chapter 7. GA Parameter Tuning

The aim of parameter tuning when using a GA is to find an appropriate set of GA parameter values that obtains good solutions in a reasonable computation time for a specific problem. Finding good parameter settings is a heavily researched area and is typically done through either parameter tuning or parameter control as described in Section 2.2. These are typically time-consuming processes. Fraser and Arcuri (2011) had argued that if parameter tuning is not performed properly, it is highly possible to result in obtaining parameter configurations which performed worse than standard parameter settings already found in the literature. In their study, they concluded that "*using default values coming from the literature is a viable option*". Standard parameter settings widely known in the literature include Goldberg's (1989), Grefenstette's (1986), and De Jong's (1975).

In order to find appropriate parameter values in this research, empirical parameter tuning was performed using these three, commonly used, sets of GA parameters, which are listed in the highlighted rows in Table 7.1. De Jong's experiments were run using a population size of both 50 and 100, as these are both common population sizes used with De Jong's parameters. Each set of parameters were used in a tuning trial to evolve a set of modifications for a terrain. Each tuning trial used the same inputs and were repeated 10 times. These inputs are:

- The *initial terrain* was generated using Perlin noise (Perlin, 1985), as shown in Figure 7.2 (a).
- The *ADS* is the one detailed in Section 5.5.
- The *input graph* contained five desired area types, four vantage points and an open area, and 50 constraints, making it one of the most complex *input graph*s used in this research.

Additionally, an elitist scheme was used in all experiments. The elitist scheme carried the two best individuals across to the new population each generation.

Table 7.1. Sets of GA parameter settings used for parameter tuning: Grefenstette's, Goldberg's, and De Jong's (highlighted).

Parameter Set	Goldberg	Grefenstette	De Jong
Population Size	30	30	50/100
Mutation Probability	0.033	0.01	0.001
Crossover Probability	0.6	0.95	0.6
Mutation Type	Gaussian + Uniform	Gaussian + Uniform	Gaussian + Uniform
Crossover Type	One Point	Two Point	One Point
Maximum Generations	4000	4000	4000
Selection	Binary Tournament	Binary Tournament	Binary Tournament
Elitist	2	2	2

7.1 Results of GA Parameter Tuning

The results of the tuning runs are evaluated in terms of fitness values achieved and execution duration. The fitness value of the best individual from each of the 10 GA runs were evaluated to determine how consistently the GA found good solutions using each of the three parameter sets. This evaluation showed that there was little difference in the fitness values of these individuals, with the fitness of the best individuals from 10 runs, for Goldberg's, Grefenstette's, De Jong's (50), and De Jong's (100) trials, being in the range of [0.894, 0.96], [0.889, 0.97], [0.880, 0.933], and [0.851, 0.96] respectively.

Table 7.2 shows the average execution time (in minutes) of the 10 runs of tuning trials for each set of parameter values. These results show that experiments using De Jong's (50) parameters took 45% and 38% longer to run than the experiments run using Goldberg's and Grefenstette's parameters respectively. Experiments run using De Jong's (100) parameters took over three times as long to run, which was expected as the computation load is $4 * 10^5$ (i.e. 4000 * 100) evaluations. Each tuning trial using Goldberg's and Grefenstette's parameters had similar execution times, as they have a computation load of $1.2 * 10^5$ (i.e. 4000 * 30) evaluations.

Table 7.2. Average execution times (in minutes) of a GA using Goldberg's, Grefenstette's, and De Jong's parameter sets.

Parameter Set	Goldberg	Grefenstette	De Jong (50)	De Jong (100)
Execution Time (minutes)	203	212.4	294.5	647.4

Execution speed is important in this approach as each evaluation requires a terrain and a number of isovists to be generated, making evaluations computationally expensive. Given that there is no significant difference in fitness values associated with the three parameter sets and execution time associated with using De Jong's parameter set is significantly longer, it is eliminated from contention. Goldberg's and Grefenstette's parameters performed very similarly in terms of execution time and fitness values achieved and Goldberg's parameters were chosen for the experiments for evaluating the proposed approach.

Obviously, execution duration for each run of the developed GA-based approach is strongly linked to the number of generations employed for the evolution process. In the tuning trials employed here, the termination condition employed was 4000, a number selected to ensure that sufficient time is provided for a GA to converge. From a subsequent analysis of the plots of fitness values associated with the 10 runs, and visual inspection of the generated terrain, negligible improvement was seen beyond 2000 generations.

7.2 Analysis of Generated Terrain

This section analyses a terrain generated using Goldberg's set of parameter values. The analysed terrain was generated from the individual with the highest fitness across the 10 runs with the termination condition of 2000 generations. Figure 7.1 shows a goal layout where the green sphere represents the open area and the red spheres represent the four vantage points that surround the open area. The *input graph* used in these experiments was extracted from this goal layout and it includes five desired area types to be incorporated into the *initial terrain*. These five area types include one open area (*oa1*) and four vantage points (*vp1*, *vp2*, *vp3*, and *vp4*). Five constraints were placed between each pair of these desired areas. These five constraints were geographical distance, path distance, traversable, line-of-sight, and reverse line-of-sight, since a line-of-sight constraint is one-way.

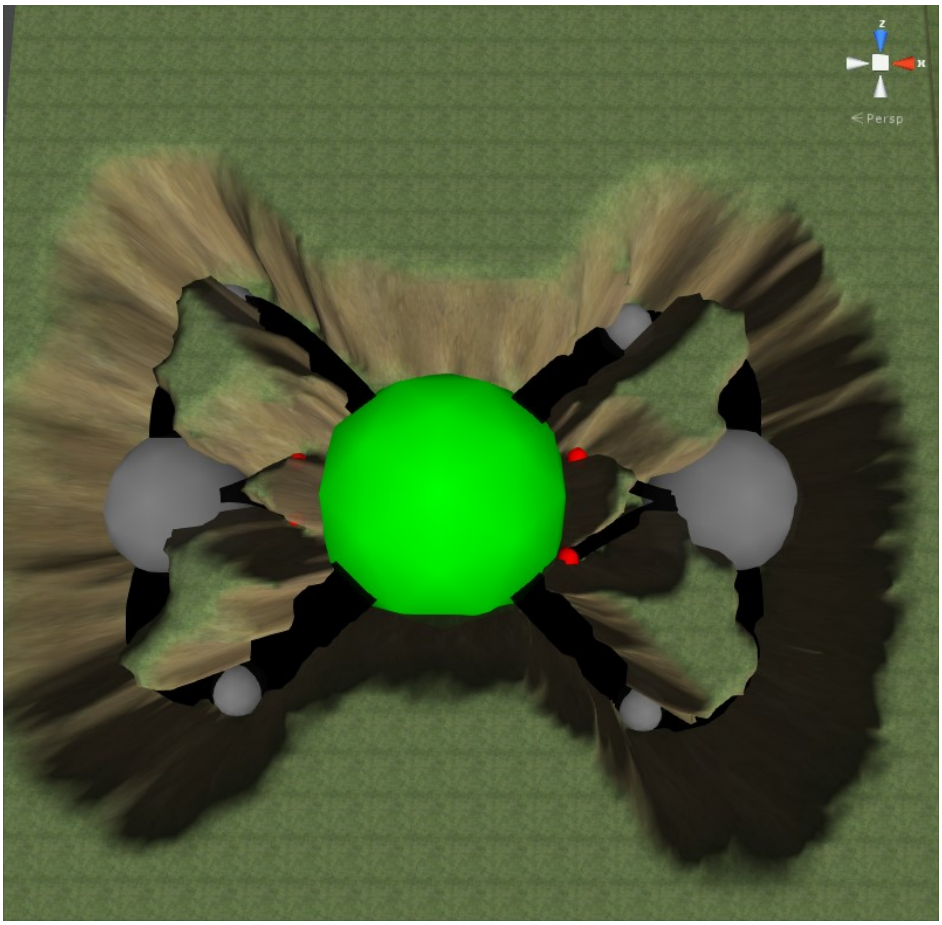


Figure 7.1. Top down view of the goal layout used in the GA parameter tuning experiments. From this goal layout, an input graph was extracted containing a single open area (represented as a green sphere), and four vantage points (represented by red spheres). Rendered in Unity (2018).

Analysis of the generated terrain to evaluate how well it meets user specification involved carrying out attribute analysis and visual analysis. Attribute analysis looks at two aspects: the area similarity measures (*asm*) of the desired area and how well the constraints between these desired areas are maintained in the generated terrain. On the other hand, visual analysis focuses on whether the desired areas and constraints are visually identifiable in the generated terrain, and how visually suitable the area types are for their intended purpose.

Attribute analysis for the constraints shows that most geographical and path distance constraints were closely met, as shown in Table 7.3 which list the desired geographical and path distances as well as their corresponding actual distance values obtained from the generated terrain. These tables show that the geographical distances in the generated terrain were generally close to their target, deviating from their desired values by an average of 6.85 units, which is $\frac{6.85}{221.7} * 100\% = 3\%$ of the maximum value. The path distances deviated from

the desired values by an average of 33.82 units, which is $\frac{33.82}{2194.03} * 100\% = 1.5\%$ of the maximum value. Table 7.4 and Table 7.5 compare the traversable and line-of-sight constraints respectively, which were met with 100% accuracy.

Table 7.3. Comparison of the desired and actual geographical and path distance constraint values of the terrain generated from the best individual across 10 runs of Goldberg's experiment.

Areas	**Geographical Distance**		**Path Distance**	
	Desired	*Actual*	*Desired*	*Actual*
vp1-vp2	9.16	32.19	43.55	68.55
vp1-vp3	43.27	12.37	224.16	299.52
vp1-vp4	46.56	46.42	225.39	240.00
vp1-oa1	2.11	5.81	109.72	192.27
vp2-vp3	45.77	45.71	224.39	230.97
vp2-vp4	44.94	46.80	225.62	179.46
vp2-oa1	1.63	6.57	109.95	123.72
vp3-vp4	12.25	12.60	50.27	51.52
vp3-oa1	0.01	2.14	114.44	107.25
vp4-oa1	0.01	1.37	113.50	55.74

Table 7.4. Comparison of the desired and actual line-of-sight and reverse line-of-sight constraint values of the terrain generated from the best individual across 10 runs of Goldberg's experiment.

Areas	**Line-of-Sight**		**Reverse Line-of-Sight**	
	Desired	*Actual*	*Desired*	*Actual*
vp1-vp2	No	No	No	No
vp1-vp3	Yes	Yes	Yes	Yes
vp1-vp4	Yes	Yes	Yes	Yes
vp1-oa1	Yes	Yes	No	No
vp2-vp3	Yes	Yes	Yes	Yes
vp2-vp4	Yes	Yes	Yes	Yes
vp2-oa1	Yes	Yes	No	No
vp3-vp4	No	No	No	No
vp3-oa1	Yes	Yes	No	No
vp4-oa1	Yes	Yes	No	No

Table 7.5. Comparison of desired and actual traversable constraint values of the terrain generated from the best individual across 10 runs of Goldberg's experiment.

Areas	**Traversable**	
	Desired	*Actual*
vp1-vp2	No	No
vp1-vp3	No	No
vp1-vp4	No	No
vp1-oa1	No	No
vp2-vp3	No	No
vp2-vp4	No	No
vp2-oa1	No	No
vp3-vp4	No	No
vp3-oa1	No	No
vp4-oa1	No	No

As discussed in Chapter 6, an *asm* is a value between zero and one, representing the similarity between the attribute values of an area in the generated terrain and the attribute values of a set of instances for the specific area type in *ADS*, where higher values indicate greater similarity. Details on *asm*s and how they are calculated are in Section 6.2.3.1. The *asm* values of the open area and vantage points in the generated terrain were mostly high, with values typically greater than 0.66. The open area was assigned the highest *asm* value of 0.85 while the vantage points were assigned *asm* values of 0.8, 0.66, 0.52, and 0.50. Although two of the vantage points were assigned *asm* values below 0.6, they were still considered to be acceptable *asm* values. The reason they are considered acceptable is because an *asm* can theoretically be less than zero, but is cropped to the range of [0, 1].

Visual analysis examines the generated terrain visually, examining how well the layout of the desired areas specified in the goal terrain is captured in the generated terrain, and how well the desired areas in the generated terrain meet their intended gameplay purpose. Figure 7.2 (a) displays the initial terrain and Figure 7.1 showed the goal layout used in generating the terrain displayed in Figure 7.2 (b).

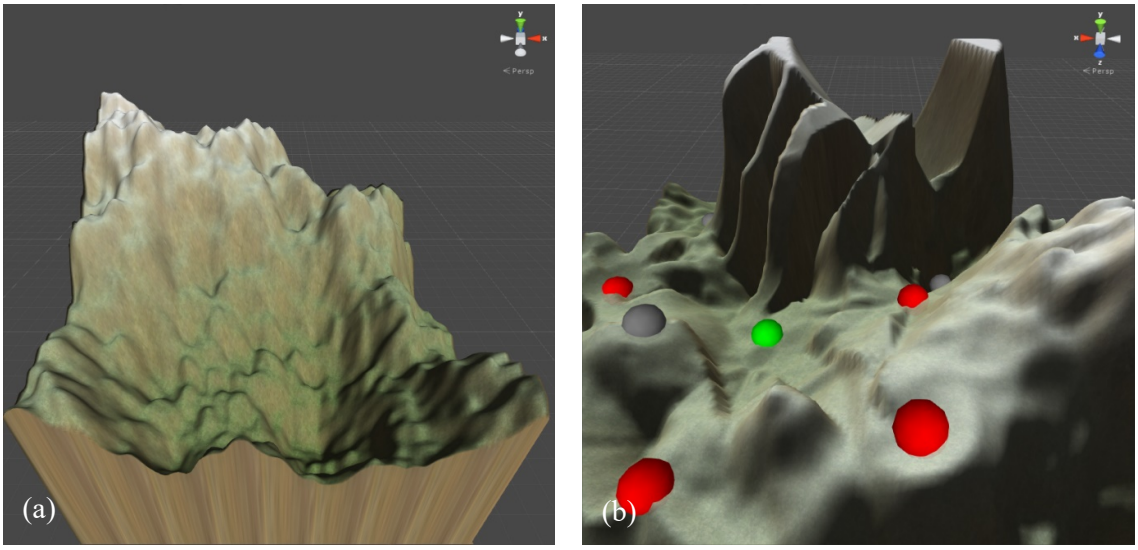


Figure 7.2. (a) The initial terrain used to generate the generated terrain. (b) The terrain generated from the best individual across the 10 runs of the Goldberg experiment. The green sphere represents the open area, the red spheres represent the vantage points, and the grey spheres indicate other generated areas.

The terrain shown in Figure 7.2 shows that the open area (green) is placed in the centre of the four vantage points (red) in a similar fashion to the specified goal layout, which is desired. The vantage points are all placed on higher ground and have line-of-sight to the open area, while remaining well concealed from it. This is also desired and shown more clearly in Figure 7.3.

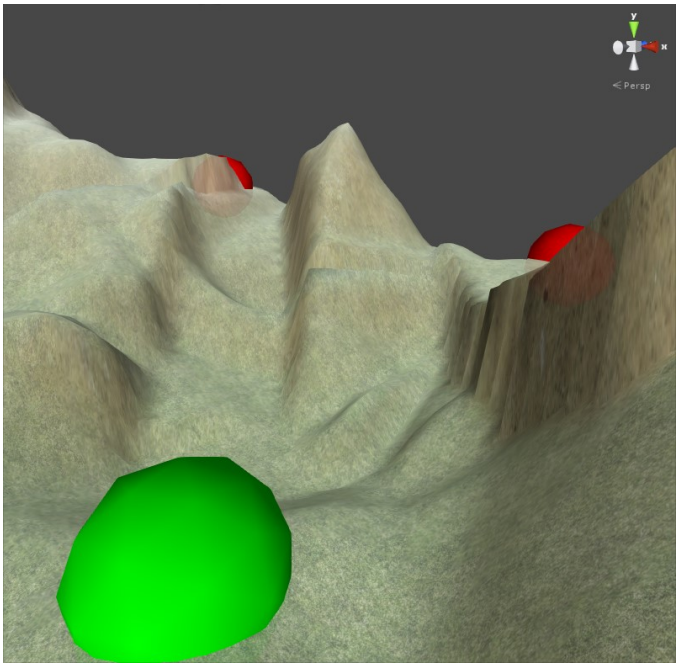


Figure 7.3. A close-up view of the open area in the terrain generated by the best individual across 10 runs of the Goldberg experiment. This figure depicts how the generated vantage points are placed at elevated locations to have line-of-sight to the open area while being obscured from view from the open area.

This visual analysis, along with the constraint analysis and the *asm*s show that terrains generated from highly ranked individuals in the 2000th generations meet the user's requirements.

7.3 Summary

From the analysis of the fitness and convergence plots, Goldberg's parameters were chosen for this approach. Further analysis of these plots associated with all the runs of the developed approach showed that the highest fitness value of a population typically does not improve significantly beyond generation 2000. Given the computation effort required, the termination condition was changed from 4000 to 2000 generations. Attribute and visual analysis of a generated terrain using the best individual from one of the 10 runs also showed that the generated terrain incorporated the user-specified gameplay elements. Table 7.6 displays a complete list of the GA parameters and stopping conditions used in Chapter 8.

Table 7.6. List of GA parameters used for the experiments conducted during this research.

Parameter	**Value**
Population Size	30
Mutation Probability	0.033
Crossover Probability	0.6
Mutation Type	Gaussian + Uniform
Crossover Type	One-Point
Selection Method	Binary Tournament
Elitism	Keep Top 2 Individuals Unchanged
Termination Condition	2000 Generations

Chapter 8. Evaluations and Results

This chapter details the results from the evaluations conducted to evaluate the approach detailed in this thesis. The aim of this approach is to evolve a set of modifications that captures user-specified gameplay elements and their associated constraints for incorporation into a user-specified terrain. A systematic evaluation of this approach was performed by evolving terrains in a set of evaluations that are broken up into four categories. Category I consists of five single area experiments, each attempting to evolve a single area of a specific type into an *initial terrain*. Category II consists of four multiple area evaluations, which attempt to evolve multiple areas of varying types into an *initial terrain*, while maintaining a few manually chosen constraints. Category III involved goal layouts and consists of the most complex evaluations in the evaluation. These five evaluations attempt to capture the layout from existing game levels and evolve a new terrain that captures the same layout in terms of included area types and their associated constraints. Category IV explored the use of different *initial terrain*s to examine how well the developed approach performs with different types of *initial terrain*. This category consists of 25 evaluations, attempting to evolve sets of modifications using the five goal layouts used in Category IV evaluations, into each of five different *initial terrain*s.

Each evaluation, in all four categories, consists of 30 independent runs, using the developed GA-based approach and the GA parameters listed in Table 7.6. Given the stochastic nature of the developed GA-based algorithm, a number of independent runs were completed (in this case, 30) to evaluate the consistency of its performance. All evaluations used the same area type dataset (*ADS*) that is detailed in Section 5.5. The same *initial terrain*, generated using Perlin noise (Perlin, 1985) and displayed in Figure 8.1, is used as input for all evaluations in Category I, II, and III. Evaluations in Category IV involved five different *initial terrain*s and were each used with five different *input graph*s, presented in Figure 8.29. The inputs for the evaluations of Category I, II, and III are shown in Table 8.1, and the inputs for the 25 evaluations of Category IV are summarised in Table 8.2.

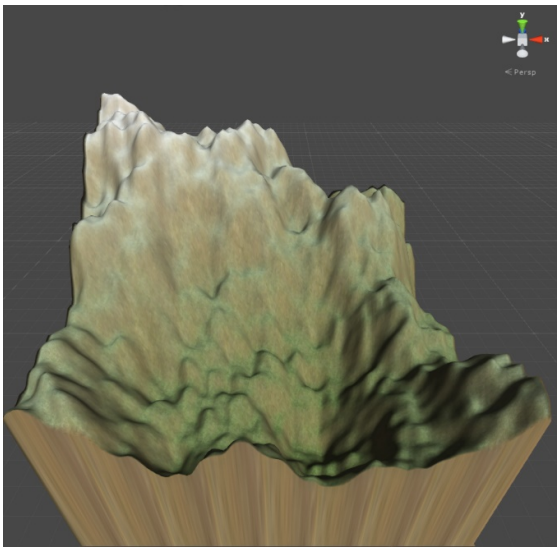


Figure 8.1. The initial terrain, used for Category I, II, and III evaluations, is generated using Perlin noise (Perlin, 1985).

Table 8.1. List of inputs (initial terrain and input graph) used in evaluations for Category I, II, and III.

Category	I	II	III
Initial Terrain	Figure 8.1	Figure 8.1	Figure 8.1
Input Graphs	1 Hidden Area	2 Hidden Areas	Figure 8.29 (a)
	1 Open Area	1 Stronghold 1 Hidden Area	Figure 8.29 (b)
	1 Vantage Point	1 Vantage Point 1 Open Area	Figure 8.29 (c)
	1 Choke Point	2 Strongholds 1 Choke Point	Figure 8.29 (d)
	1 Stronghold		Figure 8.29 (e)

Table 8.2. List of inputs (initial terrain and input graph) used in the evaluations for Category IV.

Input Graph	*Initial Terrain*				
Figure 8.29 (a)	Figure 8.43 (a)	Figure 8.43 (b)	Figure 8.43 (c)	Figure 8.43 (d)	Figure 8.43 (e)
Figure 8.29 (b)					
Figure 8.29 (c)					
Figure 8.29 (d)					
Figure 8.29 (e)					

In order to evaluate the performance of the developed approach in each evaluation, results are examined in three ways.

- Visual analysis of the generated terrains is performed to determine how well the integrated areas match the description of their associated area type. In the goal layout experiments, the layout configuration of areas is also visually analysed.
- Constraint analysis is performed to determine how closely the specified constraints are met in the generated terrains.
- Fitness values of evolved individuals are analysed to determine how consistently the developed approach found good solutions.

Additionally, for the single area evaluations, attribute analysis is also performed to show how changes in the attributes of an area affect its *asm*. Table 8.3 summarises key results.

Table 8.3. Summary of key results from the four categories of experiments.

Category	Key Results
Category I	Fitness values from these experiments ranged between 0.85 and 0.88. Most generated areas capture the desired characteristics, except vantage points that were seldom placed at high locations. Further experiments address this.
Category II	Fitness values from these experiments typically range between 0.78 and 0.95, with the exception of the experiment that incorporates two hidden areas. Incorporating multiple areas of the same type sometimes results in them getting placed at the same location in the terrain, depending on how many constraints this would violate.
Category III	Terrains evolved from the three least complex experiments show similar layouts to their goal layouts. Terrains evolved from the two most complex experiments show some similarity to their goal layout but capture the topology less accurately. The best individual from each experiment has a fitness value greater than 0.95.
Category IV	The *initial terrain* did not have a big impact on the fitness of evolved individuals, as generated terrains still incorporate desired gameplay elements. The *initial terrain* did have a strong impact on the visual appearance of generated terrains, as mountainous *initial terrains* produce mountainous generated terrains, and flat *initial terrains* produce flat generated terrains.

8.1 Category I: Single Area Evaluations

This section describes the evaluations conducted to examine the performance of the developed approach in incorporating a single area of a specified type into a user-specified terrain. The area types used in this evaluation include hidden area, open area, vantage point, choke point, and stronghold. The *initial terrain* used here is shown in Figure 8.1.

8.1.1 Incorporating a Hidden Area

This evaluation examines the performance of the developed approach to evolve a single hidden area into a user-specified terrain. At the completion of the 30 independent runs, the best individual (the individual with the highest fitness) was selected to generate the terrain presented in Figure 8.2. This individual has a fitness of 0.88 and the hidden area (represented by a yellow sphere) in the generated terrain has an *asm* of 0.87. The grey spheres in the terrain are the other eight areas encoded in the node chromosome that were not mapped to a desired area.

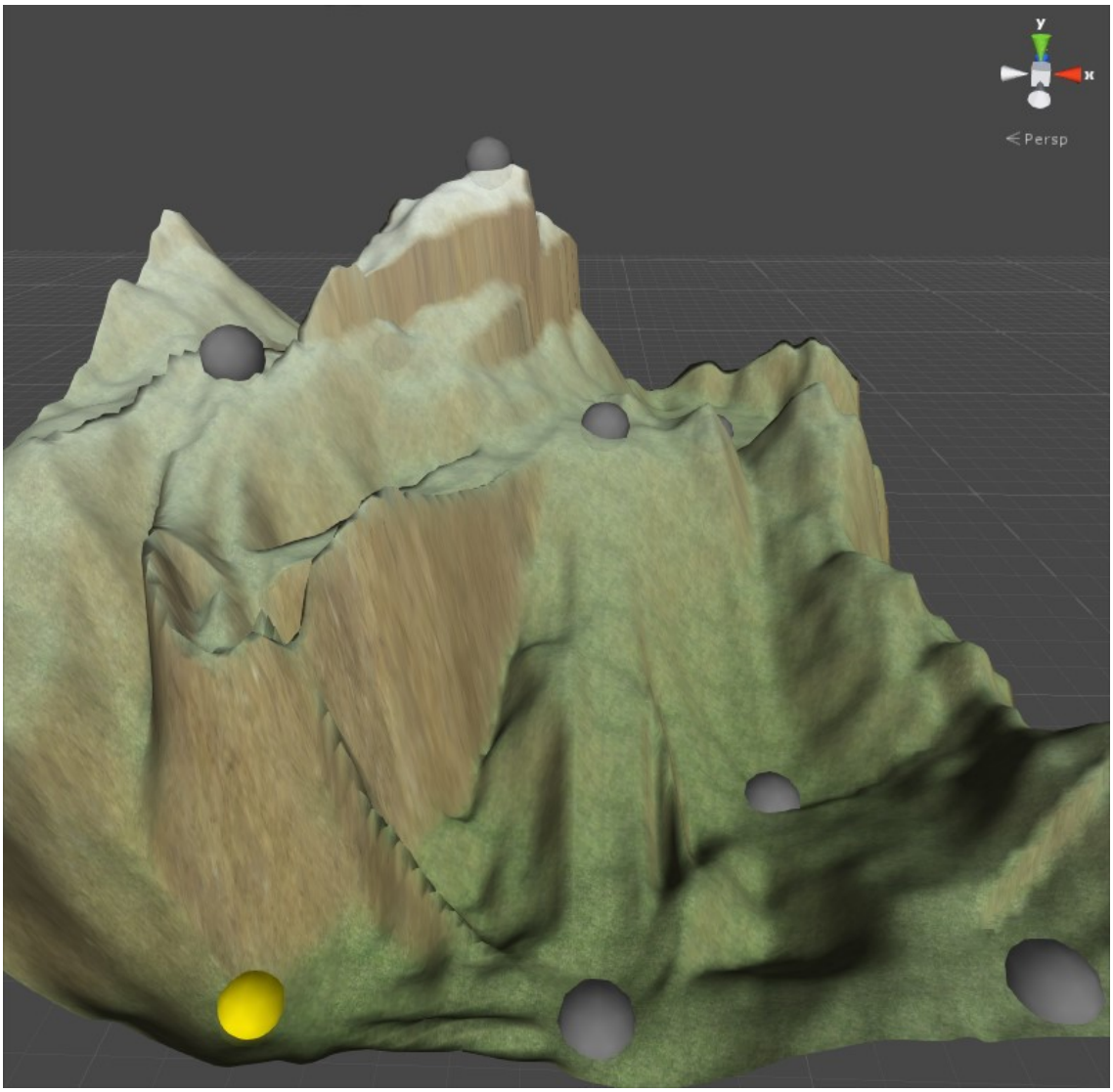


Figure 8.2. Terrain generated using the individual with the highest fitness from 30 runs of the single hidden area evaluation. The hidden area is represented as a yellow sphere. Grey spheres represent the areas associated with the nodes in the node chromosomes that were not mapped to any desired area types.

A visual examination of the generated terrain showed that this hidden area exhibits the physical characteristics of its namesake, with a single path leading to it, well placed away from other areas in the terrain, and surrounded by elevated terrain. Figure 8.3 shows the hidden area from an in-game perspective, where it can be seen that the hidden area is well occluded from player sight.

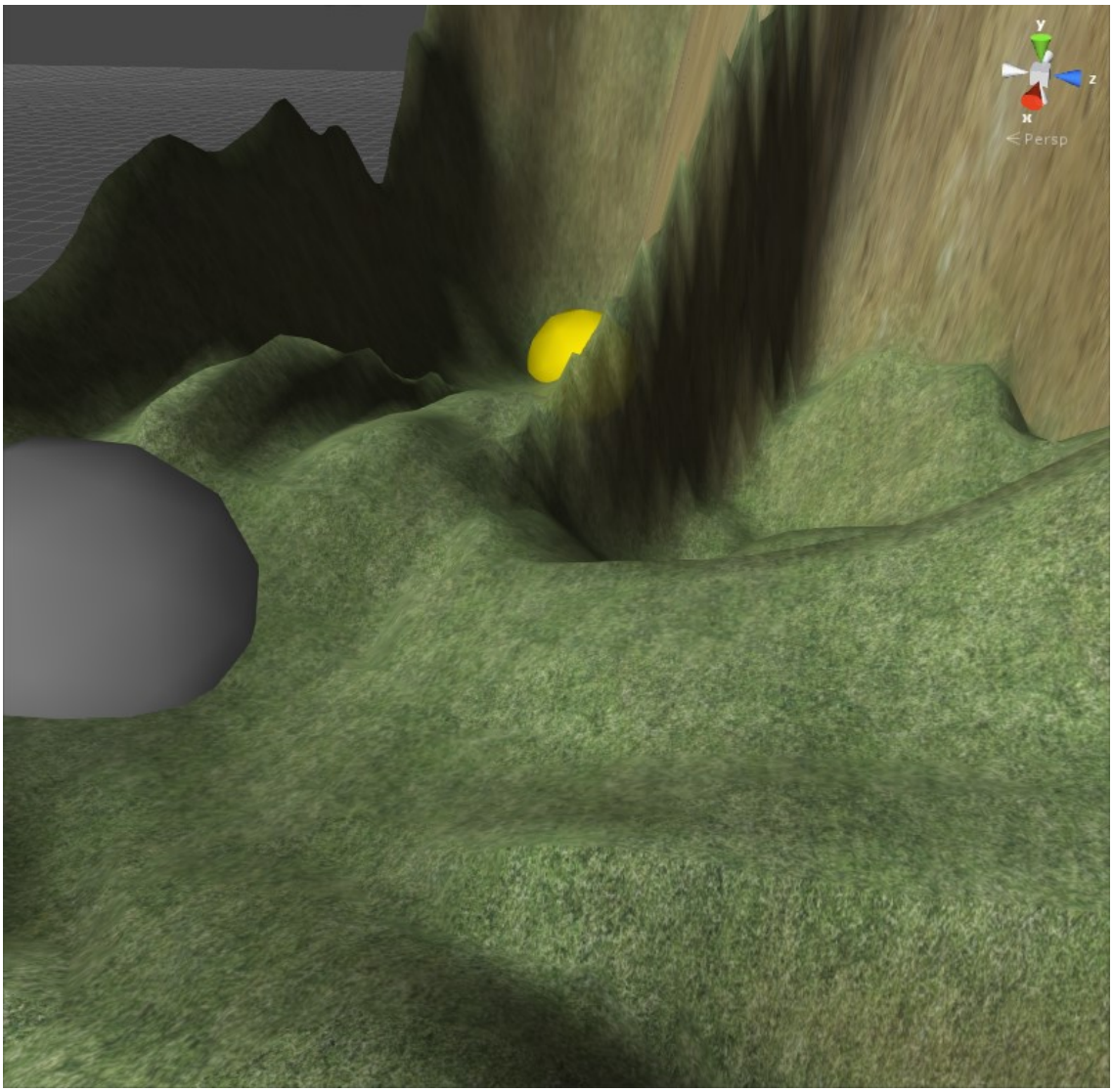


Figure 8.3. A close-up of the hidden area in the best terrain evolved for the hidden area evaluation. The hidden area is represented by a yellow sphere.

Figure 8.4 shows the fitness plot associated with the 30 independent runs for this evaluation. For each run, the fitness value for the best individual in each generation is recorded. The fitness plot shows the highest, median (red), and lowest fitness values associated with the best individual from each generation in the 30 runs. This plot is used to examine the consistency in the performance of the developed algorithm across the 30 runs. Ideally the variance should be small as the process approaches convergence, although a measure of variance is always expected as GA-based is a non-deterministic algorithm. From the plot, it can be seen that there is a big difference in the best fitness value early in the evolution (e.g. generation 1 – one run has a value below 0.5 while another has a value of 0.85). The difference in the fitness value associated with the best individuals associated with the 30 runs becomes increasingly

small as the number of generations approaches 2000 and demonstrates consistent performance of the developed approach. The median fitness of the best individuals from the 30 runs being 0.8873. This plot also shows that by the 480th generation, the best individual in the population of each of the 30 runs had a fitness between 0.86 and 0.88.

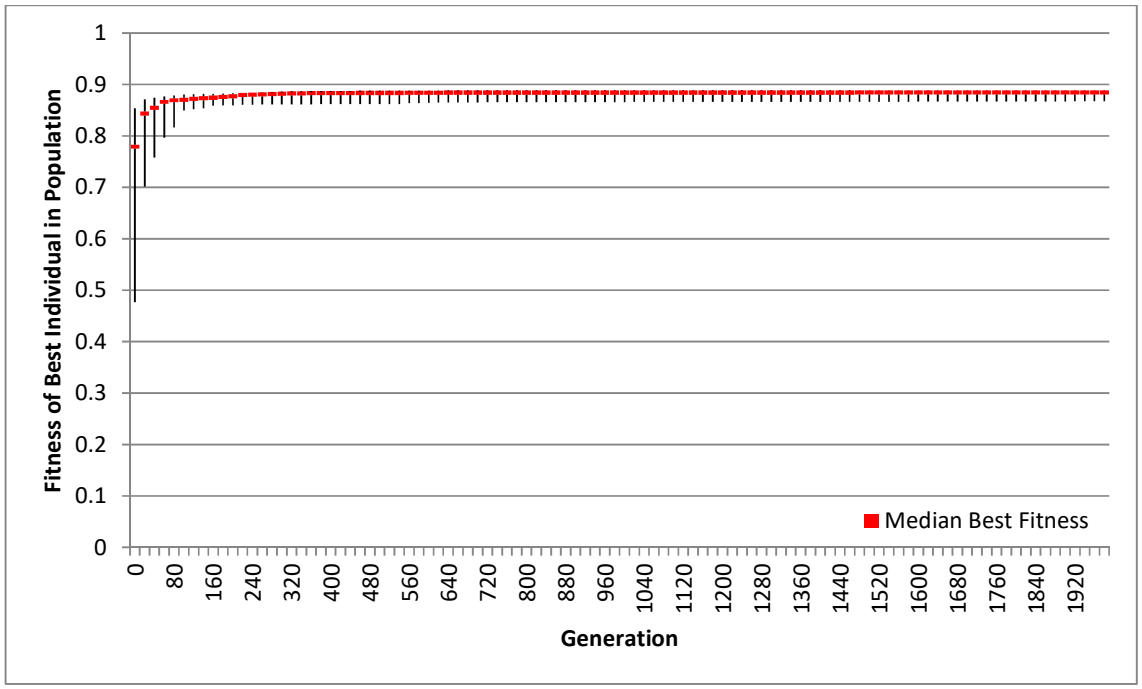


Figure 8.4. Fitness plot of the single hidden area evaluation showing the highest, median, and lowest fitness values of the best individuals across the 30 runs of this experiment.

Figure 8.5 shows a generated terrain using a lower fitness individual from the same population as the terrain presented in Figure 8.2, where the *asm* for the hidden area is 0.7. Although the hidden area still exhibits some of the desired characteristics, it is not as well concealed as the hidden area shown in Figure 8.3.

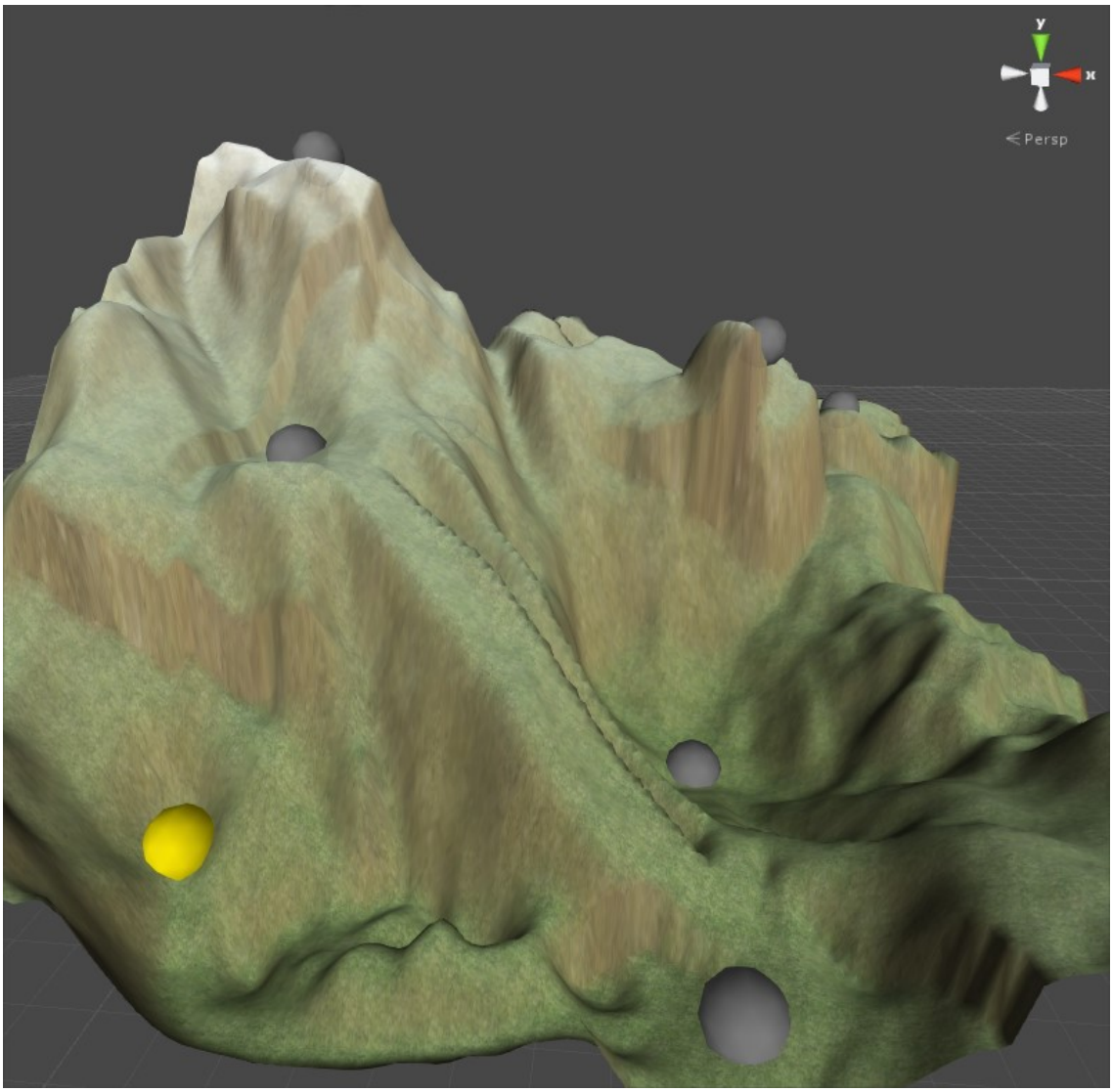


Figure 8.5. An example of a terrain that was evolved to contain a single hidden area, which is represented by the yellow sphere. The hidden area in this terrain was assigned an *asm* of 0.7 as it is less concealed than the hidden area shown previously in Figure 8.2.

Table 8.4 displays the attributes of the two hidden areas shown in Figure 8.3 and Figure 8.5, labelled hidden area 1 and hidden area 2 respectively. Many of the attribute values between the two areas are similar. Notable differences include the *Area Radius* and *Volume* attributes, which represent the traversable terrain that the area consists of, and the amount of space visible from the area, respectively. Although hidden area 1 is larger than hidden area 2, its higher *asm* is mostly due to its lower *Volume* attribute, which indicates hidden area 1 has better cover, concealing it from view.

Table 8.4. List of attribute values of two hidden areas, one with a high *asm* and the second with a mediocre *asm*.

Attribute	Hidden Area 1 *asm*: 0.87 (high *asm*)	Hidden Area 2 *asm*: 0.7 (mediocre *asm*)
Area Radius	6.00	2.00
Degree Centrality	1.00	1.00
Eigenvector Centrality	0.79	0.50
Betweeness Centrality	0.00	0.00
Closeness Centrality	0.01	0.01
Average Radial Length	13.63	19.56
Dispersion	-6.24	-4.84
Drift 3D	7.92	11.54
Drift 2D	3.56	7.62
Maximum Radial Length	64.00	64.00
Minimum Radial Length	2.63	2.37
Skewness	2.03	1.17
Sphericality	0.07	0.13
Standard Deviation	19.87	24.41
Variance	394.84	595.78
Volume	144887.80	248667.94

8.1.2 Incorporating an Open Area

This evaluation examines the performance of the developed approach to evolve an open area into a user-specified terrain. Of the 30 runs of the GA-based approach, the evolved individual with the highest fitness was used to generate the terrain presented in Figure 8.6. The open area is represented here by the green sphere.

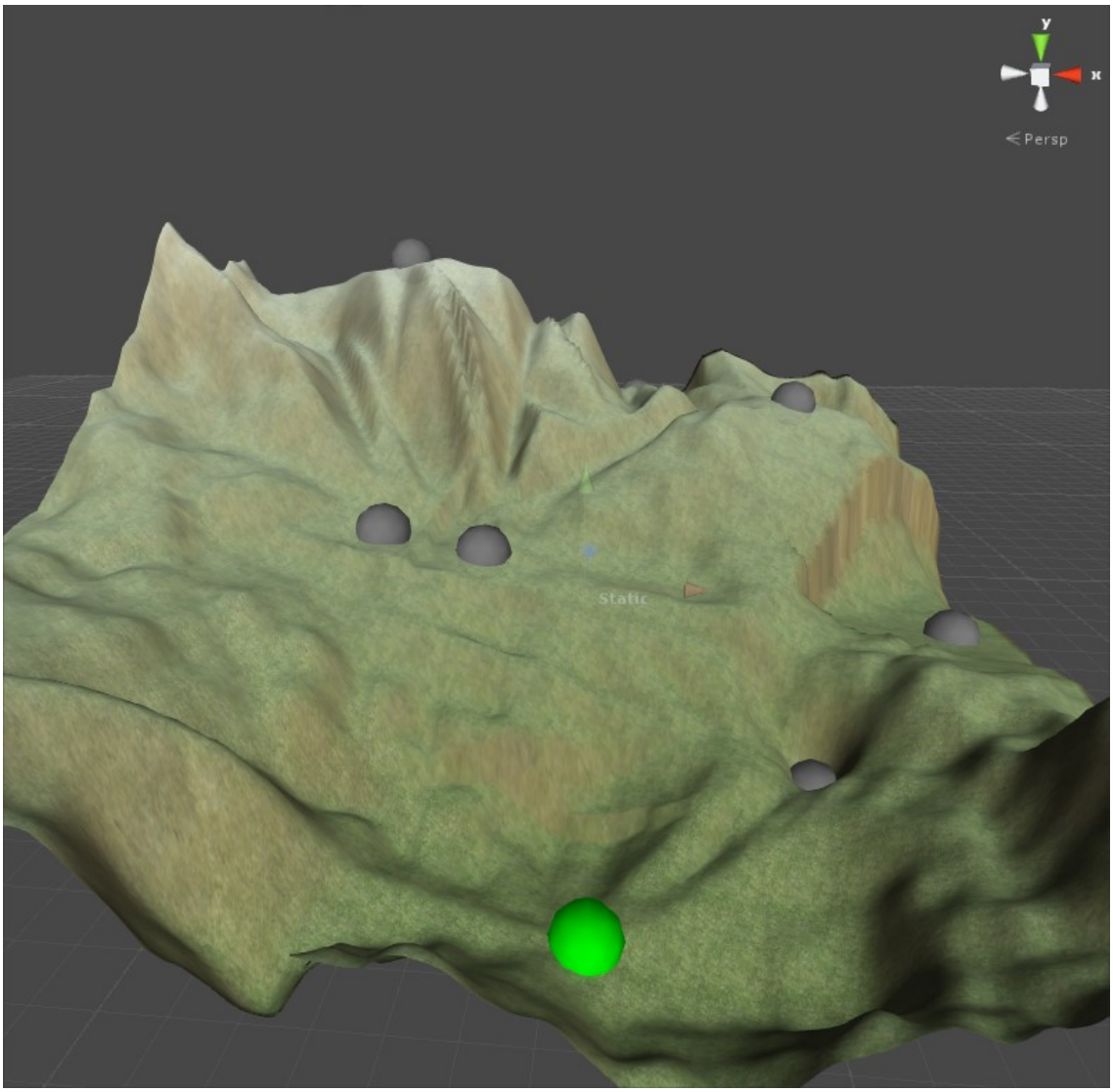


Figure 8.6. Terrain with highest fitness from 30 runs of the single open area evaluation. The open area is represented by the green sphere.

This image shows that the area is in an open space, away from high walls that would limit access. These are desirable traits for an open area, which is reflected by its *asm* of 0.86.

Similar to Figure 8.4, Figure 8.7 shows the fitness plot associated with the best individual in each generation from the 30 independent runs of the GA-based approach. The fitness plot shows the highest, median (red), and lowest fitness values associated with the best individual from each generation in the 30 GA runs. As the number of generations approaches 2000, it can be seen that these individuals from the 30 runs have similar fitness values, within the range of [0.85, 0.88].

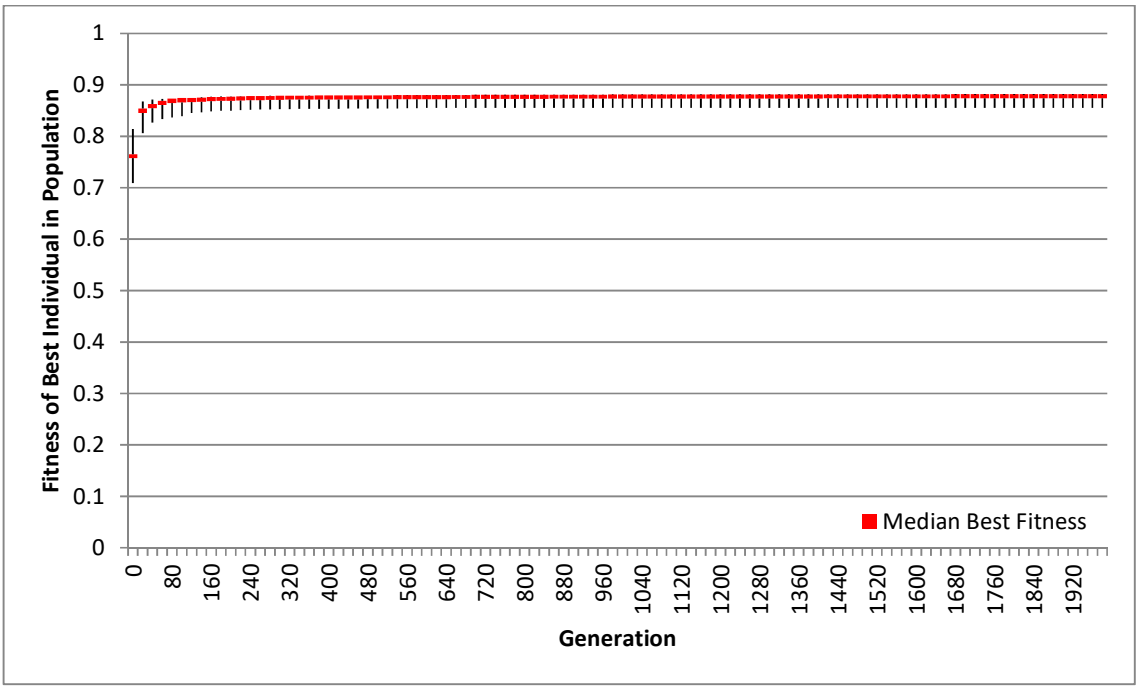


Figure 8.7. Fitness plot of the single open area evaluation showing the highest, median, and lowest fitness values of the best individuals across the 30 runs of this evaluation.

To have a better picture of how a terrain generated from an individual with a low fitness value would compare to one generated from one with a high fitness value, two terrains using individuals with fitness of 0.12 and 0.85 were generated and analysed. Visually the two generated terrains are similar, with a few differences.

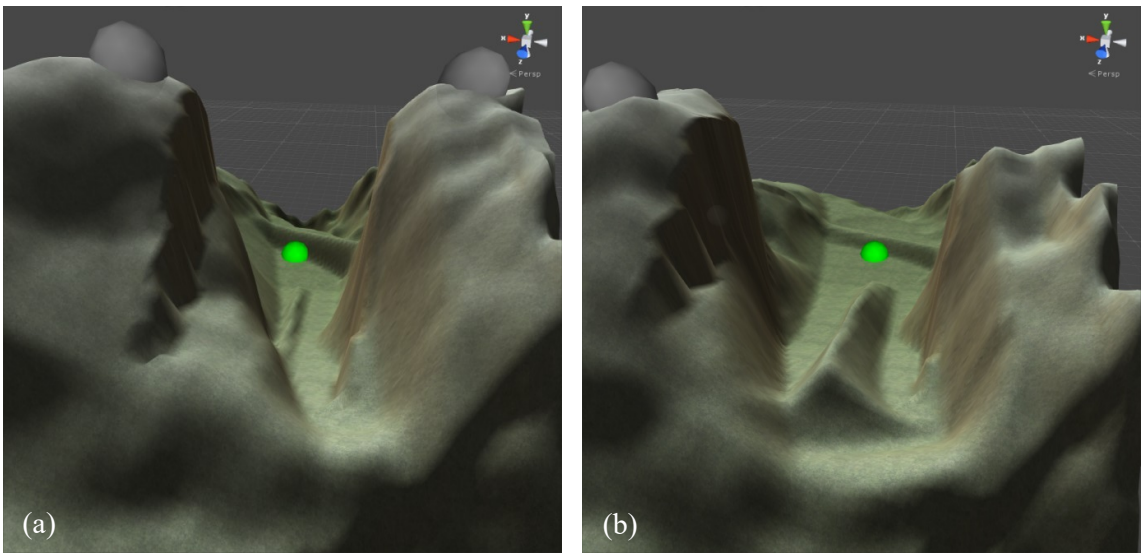


Figure 8.8. Comparison images of two terrains, each containing an open area (green sphere). (a) Generated terrain with fitness of 0.12. (b) Generated terrain with fitness of 0.85.

The key differences between the two terrains are the positions of the two areas on either side of the open area, the 'right area' and the 'left area'. These areas are represented by grey

spheres. In the generated terrain in Figure 8.8 (b), the *right area* and *left area* associated with the open area are lower in height, providing traversable paths between them and the open area. However, these paths are cut off in the generated terrain in Figure 8.8 (a) due to these two areas being higher, making the terrain steeper. Additionally, there is a path between the *right area* and another area to the left of the open area. This path can be seen as a short wall directly behind the open area in both Figure 8.8 (a) and Figure 8.8 (b). In the generated terrain from the higher fitness individual, this path was generated lower to the ground, making the height difference small enough that it can be traversed over, allowing access between the open area and other areas. In the generated terrain in Figure 8.8 (a) this path has been raised due to the *right area* being raised. This increase in the height of the path prevents access to the open area from behind. These differences ultimately restrict access to the area, preventing it from being open.

This difference in accessibility can be more clearly seen in the accessibility maps shown in Figure 8.9, where white areas indicates traversable terrain and black areas non-traversable. The open area is displayed as a green circle. Figure 8.9 (a) shows the accessibility map for the individual with low fitness, while Figure 8.9 (b) shows the accessibility map for the individual with higher fitness. These accessibility maps clearly show that the open area from the higher fitness individual is more open in terms of accessibility. The open area from the lower fitness individual only has a single entrance as non-accessible terrain has been formed around it, cutting off other entrances.

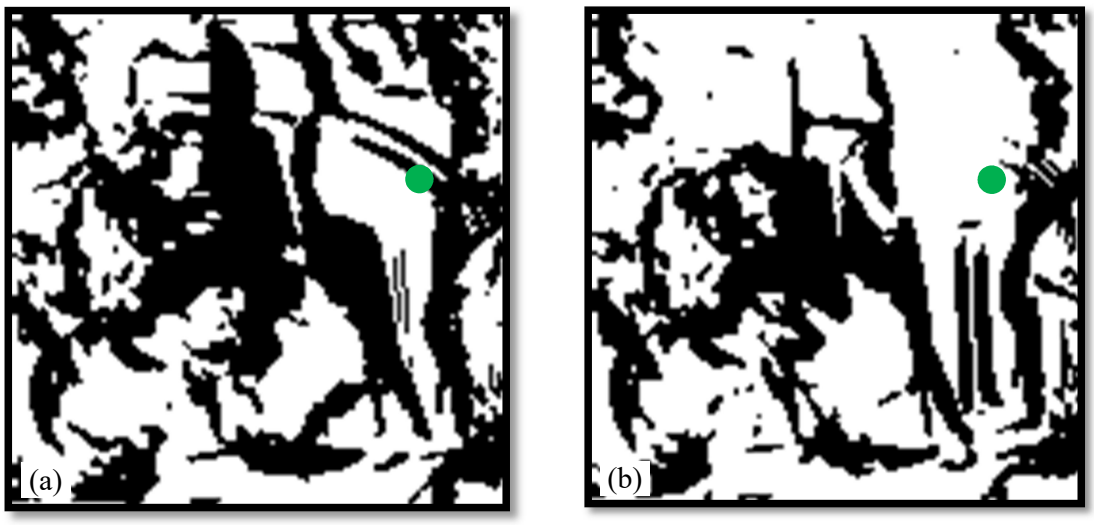


Figure 8.9. Accessibility maps for individual with low fitness (a) and individual with high fitness (b) from one of the 30 runs of the open area evaluation. White areas indicate traversable sections of terrain, while black areas are non-traversable. The open area is represented by a green circle.

Table 8.5 displays a list of the attribute values of these two open areas. Two of the most notable differences between the attributes of these two areas are the graph measures and the *Area Radius* attribute. Since the open area of the lower fitness individual is cut off from all other areas, it has a *Degree Centrality*, *Eigenvector Centrality*, *Betweenness Centrality*, and *Closeness Centrality* of 0. This caused the most significant loss in the *asm* of this area. The *Area Radius* attribute of this area is also significantly smaller, where open areas typically have a large *Area Radius* attribute values, as shown in the higher fitness individual. Other notable differences include the *Dispersion*, *Drift 2D*, and *Minimum Radial Length* attributes, which were affected by the wall forming so close to the centre of the open area in the lower fitness individual.

Table 8.5. List of attribute values of two open areas, one with a low *asm* and the second with a high *asm*.

Attribute	Low Fitness Individual ***asm*: 0.0**	High Fitness Individual ***asm*: 0.83**
Area Radius	2.00	42.38
Degree Centrality	0.00	2.00
Eigenvector Centrality	0.00	0.65
Betweeness Centrality	0.00	0.11
Closeness Centrality	0.00	0.01
Average Radial Length	23.36	30.12
Dispersion	-1.86	4.57
Drift 3D	12.18	13.53
Drift 2D	0.32	2.91
Maximum Radial Length	64.00	64.00
Minimum Radial Length	1.99	3.21
Skewness	0.82	0.36
Sphericality	0.18	0.30
Standard Deviation	25.22	25.54
Variance	635.88	652.30
Volume	295024.20	386491.50

8.1.3 Incorporating a Vantage Point

This evaluation examines the performance of the developed approach to evolve a vantage point into a user-specified terrain. Of the 30 runs of the GA-based approach, the evolved

individual with highest fitness was used to generate the terrain presented in Figure 8.10. The vantage point is represented by a red sphere. The generated area has some characteristics of a vantage point including its relatively small area and its limited access. This can be more clearly seen in the close-up shown in Figure 8.11, and in its accessibility map shown in Figure 8.12.

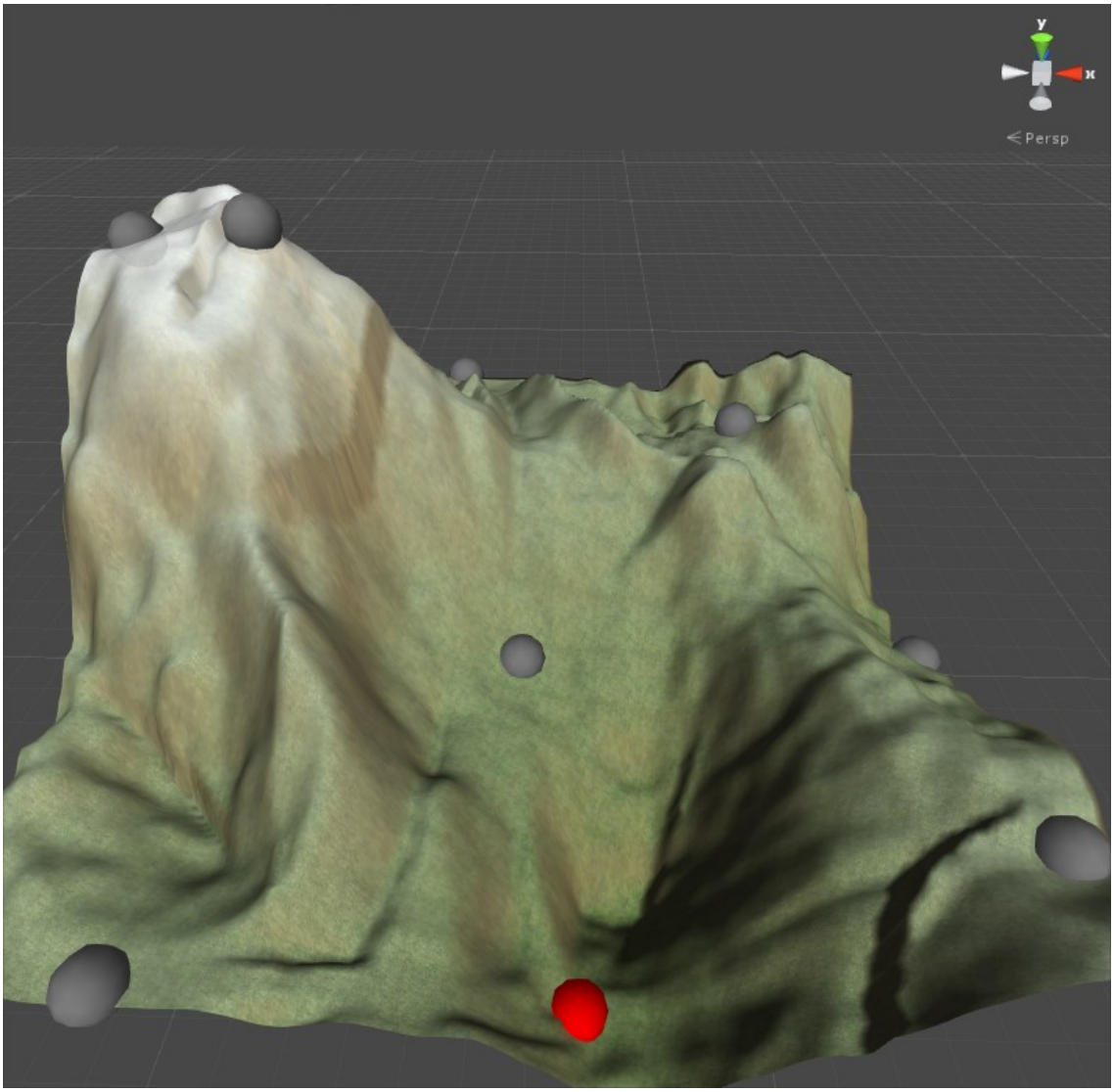


Figure 8.10. Generated terrain using the highest fitness individual from 30 runs of the single vantage point evaluation. The vantage point is represented as a red sphere.

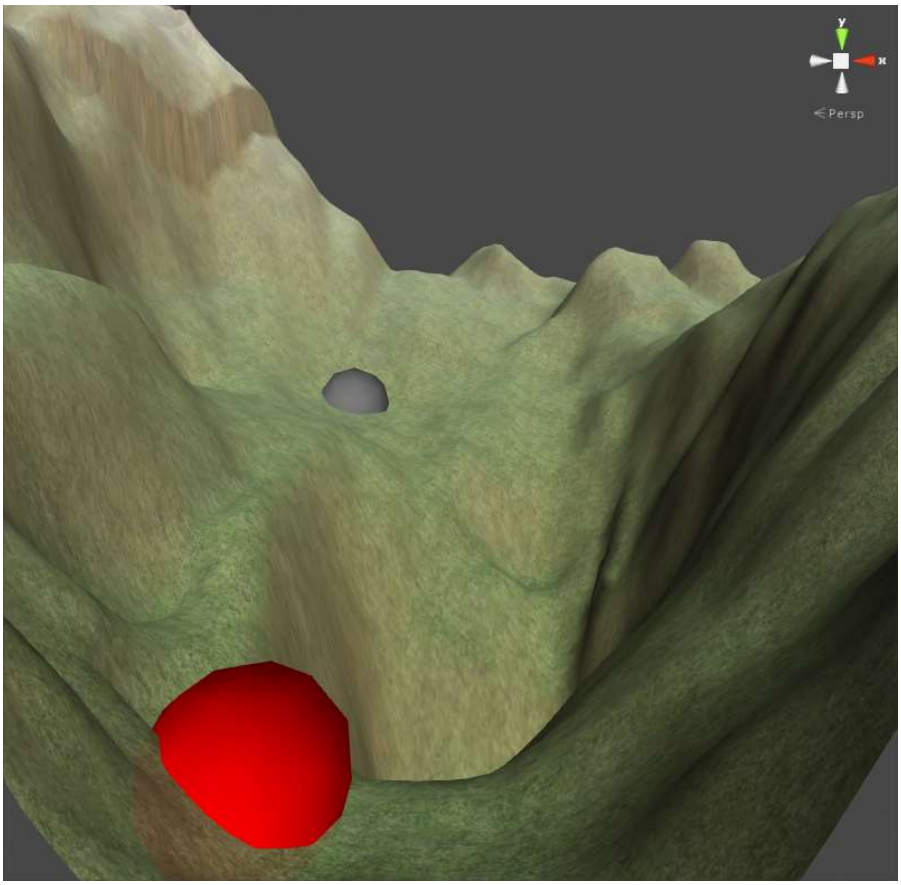


Figure 8.11. A close-up of the vantage point from the generated terrain using the highest fitness individual from 30 runs of the single vantage point evaluation. The vantage point is represented by a red sphere.

Figure 8.12. Accessibility map for the best individual of the single vantage point evaluations. The vantage point is represented as a red circle.

The accessibility map in Figure 8.12 shows that the vantage point, represented by a red circle, is a relatively small area, and that it has only a single entrance as it is mostly surrounded by impassable terrain. Although the generated vantage points from this evaluation typically exhibited these desirable characteristics, few of them were placed in high locations giving them a good vantage over sections of the level. This suggests the isovist and graph measures used in this approach were not fully capable of capturing this quality. The multiple area evaluations, detailed in Section 8.2, further explored the incorporation of vantage points, using constraints to capture this quality. These evaluations showed more desirable results with vantage points being placed at higher locations with a good vantage over a specified area of the terrain.

Figure 8.13 displays the fitness plot for this evaluation, which shows that in all 30 runs of the GA-based approach, the best fitness individuals have fitness values of ~0.86 by 2000 generations (termination condition).

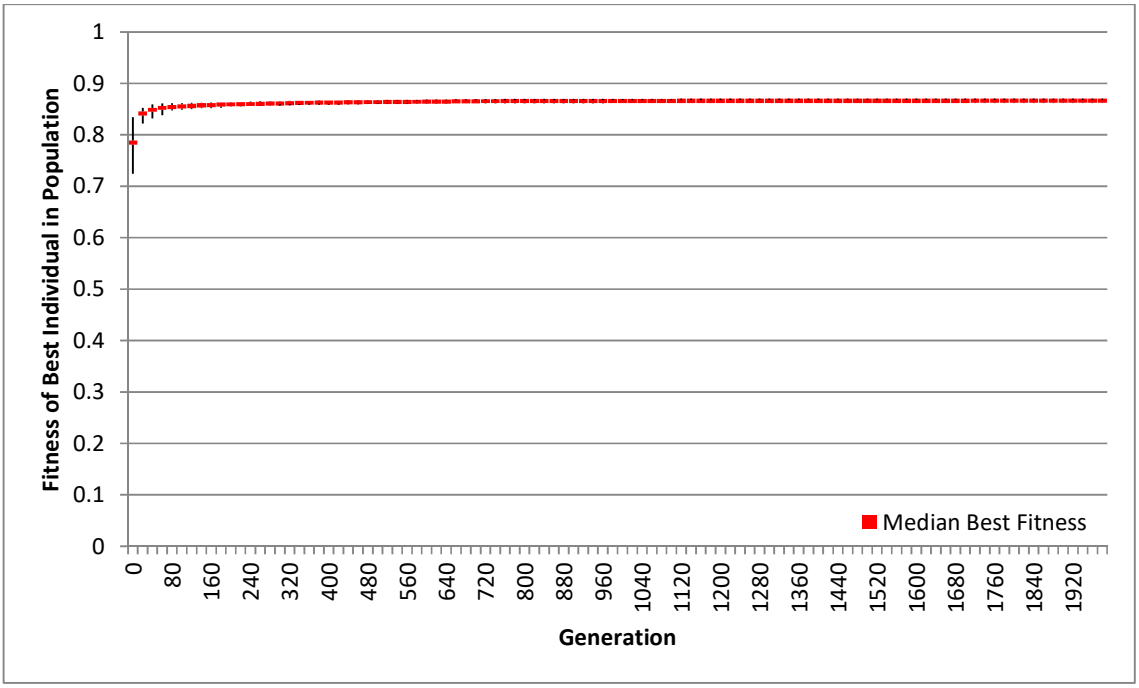


Figure 8.13. The fitness plot for the single vantage point evaluation. It shows the highest, median, and lowest fitness values of the best individual in the final populations of the 30 runs of the evaluation.

8.1.4 Incorporating a Choke Point

This evaluation examines the performance of the developed approach to evolve a choke point into a user-specified terrain (Figure 8.1). Figure 8.14 shows a generated terrain using the best individual from the 30 runs. This image shows that the generated choke point, represented as

a blue sphere, is placed along a narrow, critical path between two sections of the terrain. This is highlighted by the white lines between the generated areas in the terrain, which show the connectivity of the terrain's *layout graph*. This means the only traversable path between the majority of the game level and the section at the front of the terrain goes through the choke point. The choke point is also placed on low ground, exposing it to possible attackers from high ground. These characteristics are desirable as they make the area an ideal place for an ambush, which is a common use for choke points in video games.

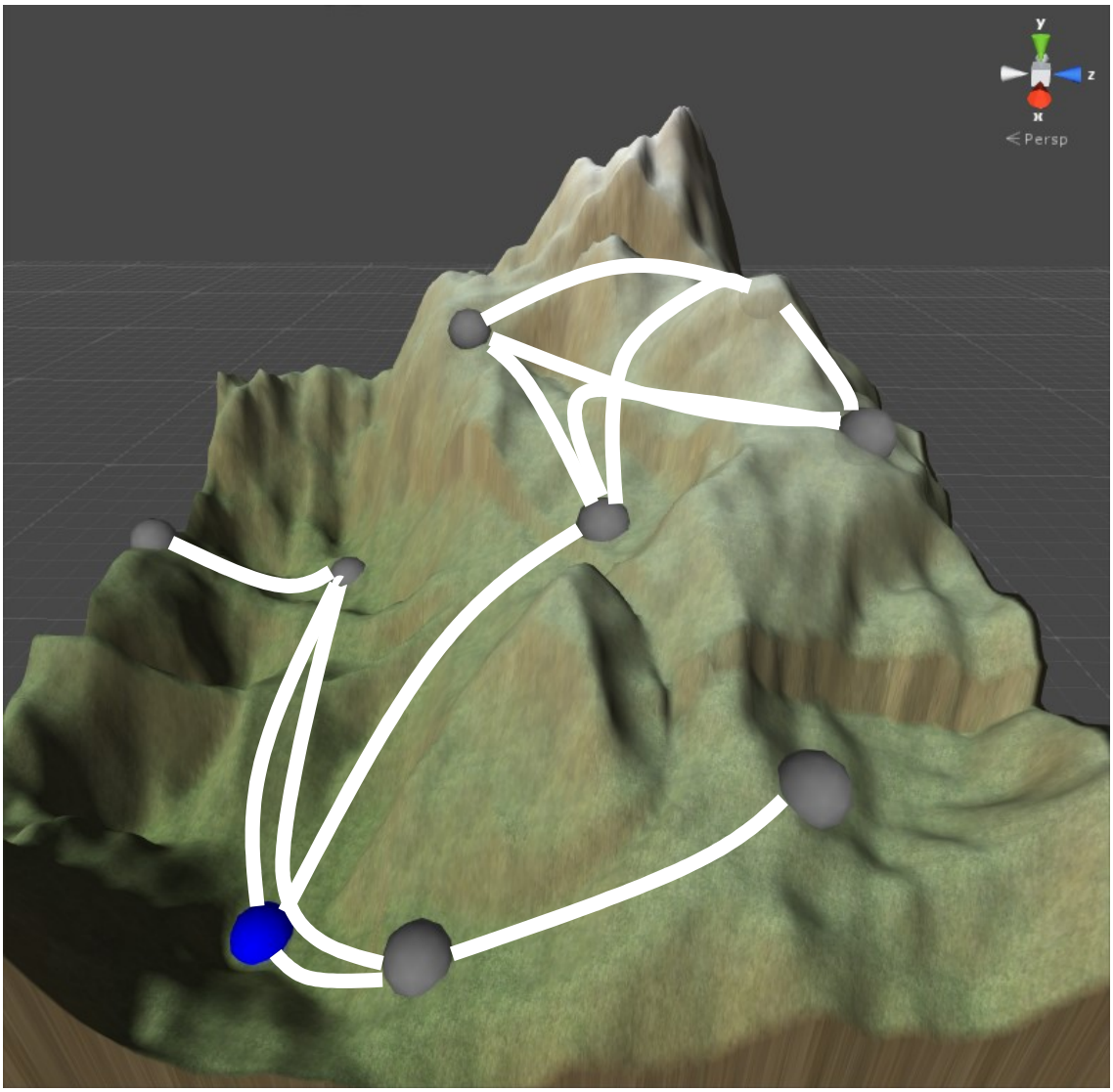


Figure 8.14. Generated terrain using highest fitness individual from 30 runs of the single choke point evaluation. Connectivity between generated areas, as captured by the layout graph, is shown by the white lines.

Although most runs of this evaluation produced choke points with similar physical characteristics, placement of the choke point relative to other areas in the terrain is also important. Choke point placement is important as choke points are often used to partition

sections of a level, forcing players to travel through them in order to progress. The collection of graph-connectivity metrics, G_1-G_4, listed in Table 4.1 are used to capture this quality in the *layout graph* of a terrain. However, although the connectivity of a *layout graph* captures the accessibility of the terrain between two areas, it does not encompass information about the accessibility of the terrain surrounding these two areas. An example of this can be seen in Figure 8.15, where, following the connectivity of the *layout graph* (white lines), the evolved choke point must be travelled through to reach the area circled in red. Unfortunately, as shown in the accessibility map displayed in Figure 8.16, the area of interest (grey circle) is mostly surrounded by traversable terrain, making travelling through the choke point (blue circle) unnecessary.

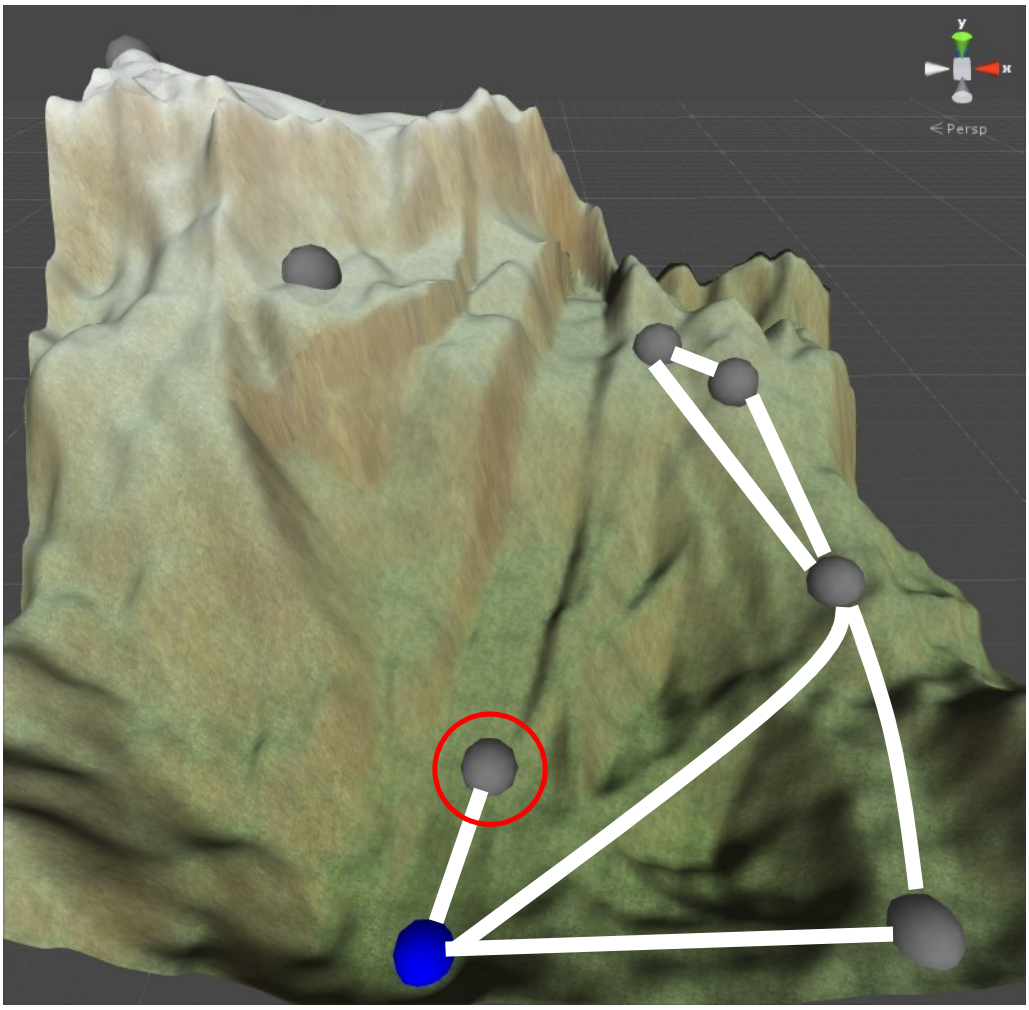


Figure 8.15. Example of a terrain evolved in the single choke point evaluation. White lines show the connectivity between areas according to the layout graph of the terrain. This connectivity shows that the choke point, represented as a blue sphere, separates the circled area from the rest of the level.

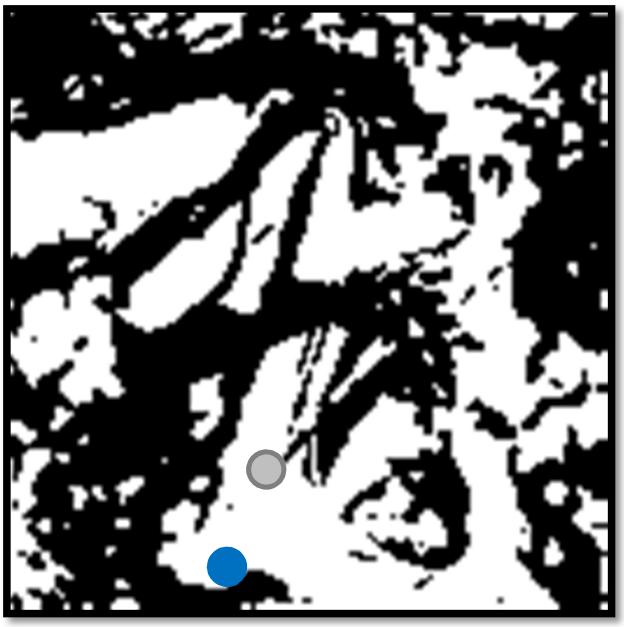


Figure 8.16. The accessibility map of the terrain containing a single choke point. This accessibility map shows that the choke point (blue circle) needn't be travelled through to reach any other part of the terrain, despite the terrains layout graph.

Figure 8.17 shows the highest, median (red), and lowest fitness values associated with the best individual from each generation in the 30 runs of the GA-based approach. This plot shows that every repeat of the 30 evaluation runs produced an individual with a high fitness of ~0.86, similar to the open area and choke point evaluations.

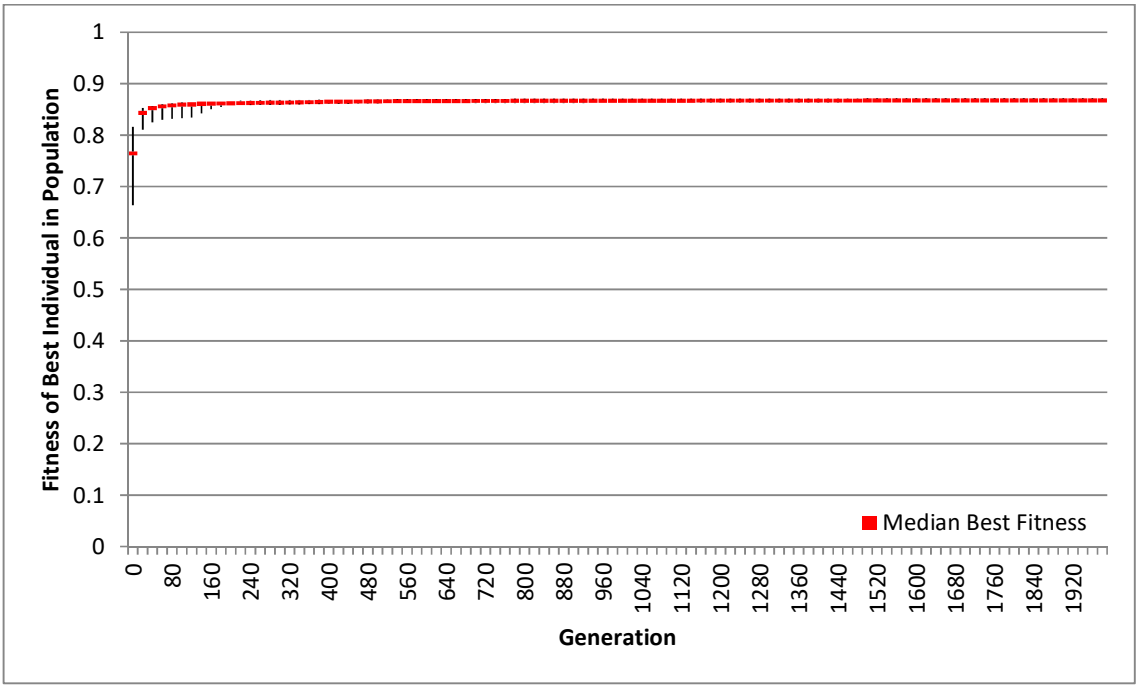


Figure 8.17. The fitness plot of the single choke point evaluation. Shows the highest, median, and lowest fitness values of the best individual in the final populations of the 30 runs of the evaluation.

To summarise, all 30 runs of this evaluation incorporated choke points into the *initial terrain*, where the choke points exhibited ideal physical characteristics. The majority of these choke points were placed at ideal locations in the level layout topologies, separating two segments of the traversable terrain, forcing players to travel through the choke point in order to traverse between segments. Occasionally the *layout graph*s of generated terrains did not accurately capture the topology of the level in terms of accessible terrain. This due to the graph-connectivity of the generated *layout graph* being inaccurate.

8.1.5 Incorporating a Stronghold

This evaluation examines the performance of the developed approach to evolve a stronghold into a user-specified terrain. Most strongholds that were generated in the 30 repeated runs were assigned high *asm* values of ~0.85. These highly ranked strongholds were large areas and well surrounded by elevated terrain, which provided the stronghold with cover, similar to their description. They usually had between one to two entrances and were otherwise surrounded by impassable terrain, limiting access. All these characteristics are desirable in a stronghold. An example of a terrain, generated using the best individual across the 30 final populations from the 30 repeated runs, is displayed in Figure 8.18, where the stronghold is represented as an orange sphere.

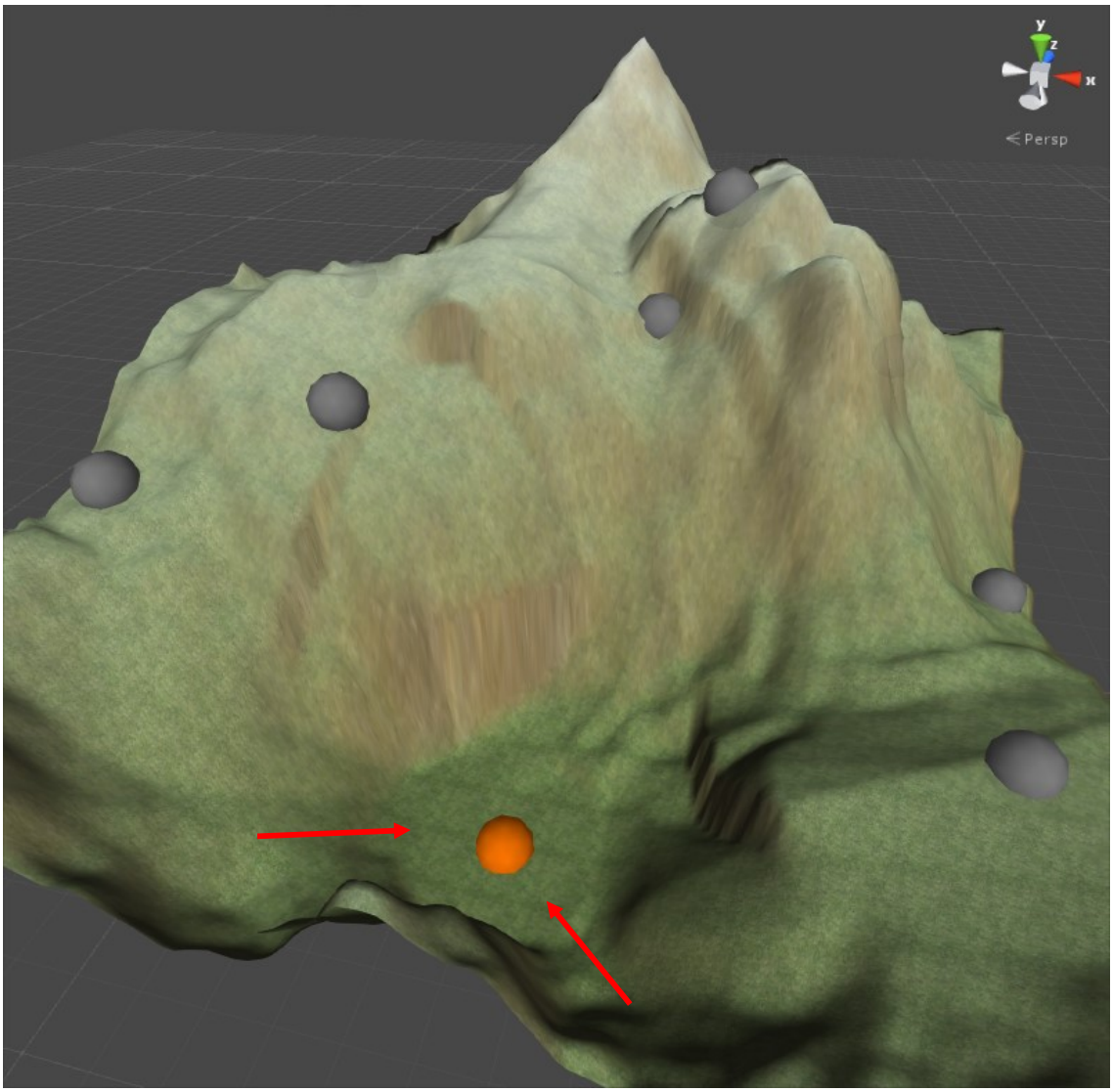


Figure 8.18. Generated terrain using the highest fitness individual from 30 runs of the single stronghold evaluation.

This figure shows a stronghold that is surrounded by elevated terrain, providing cover. The stronghold has two entry points, highlighted by red arrows, where terrain elevation is low enough to allow access to and from the stronghold.

Some strongholds had their two entrances on opposite sides of the area, which increases the difficulty of defending the stronghold. Ideally entrances should be placed next to one another to make defending the area easier since players don't have to watch their back. One potential way to handle this issue is to incorporate a user input option for adjusting and controlling this aspect of generated strongholds, allowing control over the difficulty of the game level.

Figure 8.19 shows the fitness plot for this evaluation. All 30 runs of the GA-based approach produced individuals with high fitness values. Most runs resulted in an individual with a fitness value above 0.86 as shown by high the median (red) value.

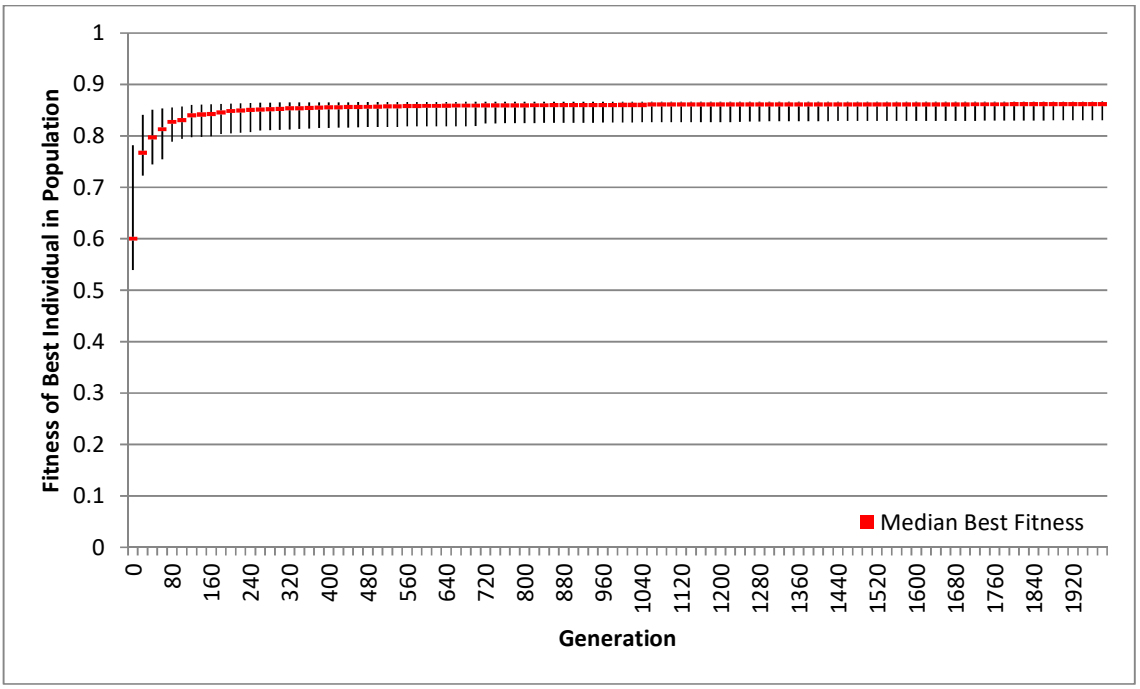


Figure 8.19. The fitness plot for the single stronghold evaluation. It shows the highest, median, and lowest fitness values of the best individual in the final populations of the 30 runs of the evaluation.

8.2 Category II: Multiple Area Evaluations

This section describes the evaluations conducted to examine the performance of the developed approach in incorporating more complex configurations of gameplay elements into a user-specified terrain. Each of these evaluations attempt to evolve between two and three areas of specific types into the *initial terrain*, while maintaining up to four manually specified constraints. The *initial terrain* used for evaluations in this category is shown in Figure 8.1.

8.2.1 Incorporating Two Hidden Areas with Associated Constraints

This evaluation employed the developed approach to evolve two hidden areas, into the *initial terrain*, that were separated by a geographical distance of 181 units, ideally placing the two areas at opposite corners of the terrain. The value 181 was calculated as the Euclidean distance between two corners of the terrain, where the terrain's length is 128 units, resulting in the equation $181 = \sqrt{128^2 + 128^2}$. This test was designed to be the simplest of the multiple area evaluations, as it only incorporates two desired areas and has a single constraint placed between them. Hidden areas were chosen due to the results of the single area

evaluations, which showed hidden areas to be one of the easiest area types to be evolved using the developed approach.

Figure 8.20 shows a terrain generated using the individual with the best fitness from the 30 runs. This terrain contains two hidden areas, represented as yellow spheres, with *asm* values of 0.85 and 0.83 and separated by a geographical distance of 165 units. As shown, both hidden areas are surrounded by elevated terrain and are out of the way from other areas in the level, as is expected of hidden areas with a high *asm*. They have been placed at opposite corners of the terrain, as enforced by the geographical distance constraint. The circled image in Figure 8.20 views one of the hidden areas from a different location, as it is hidden behind a mountain in the main figure.

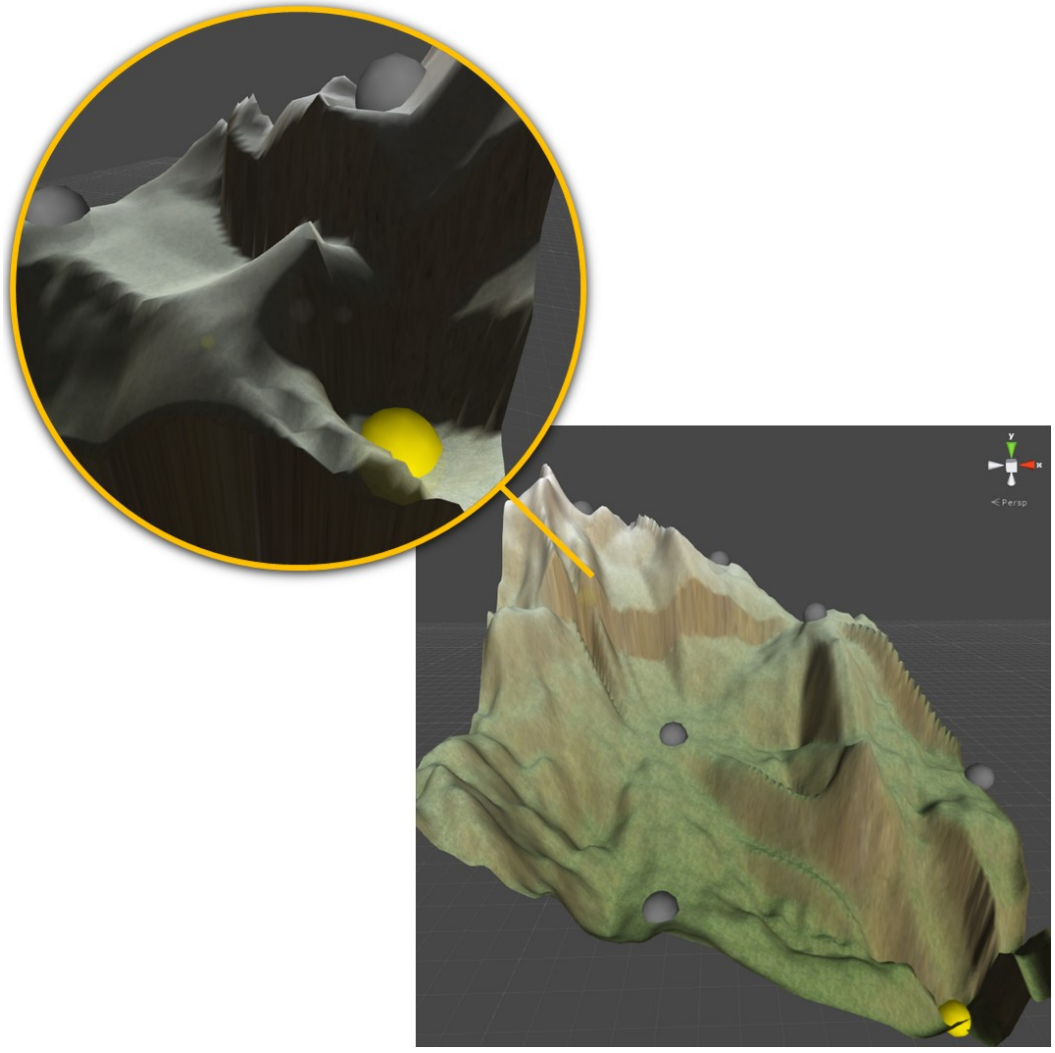


Figure 8.20. A terrain generated from the best individual evolved during the two hidden areas evaluation. The yellow spheres represent the two hidden areas.

Figure 8.21 shows the fitness plot associated with the 30 independent runs for this evaluation. For each run, the fitness value for the best individual in each generation is recorded. The fitness plot shows the highest, median (red), and lowest fitness values associated with the best individual from each generation in the 30 runs. The plots showed variability in the fitness value of the best individual, ranging from 0.88 to 0.61 as the number of generations approach 2000. In at least one of the 30 runs, the best individual has probably converged to a local optima, resulting in a fitness value of 0.61. The median (50% of the individuals) has a fitness value of approximately 0.8.

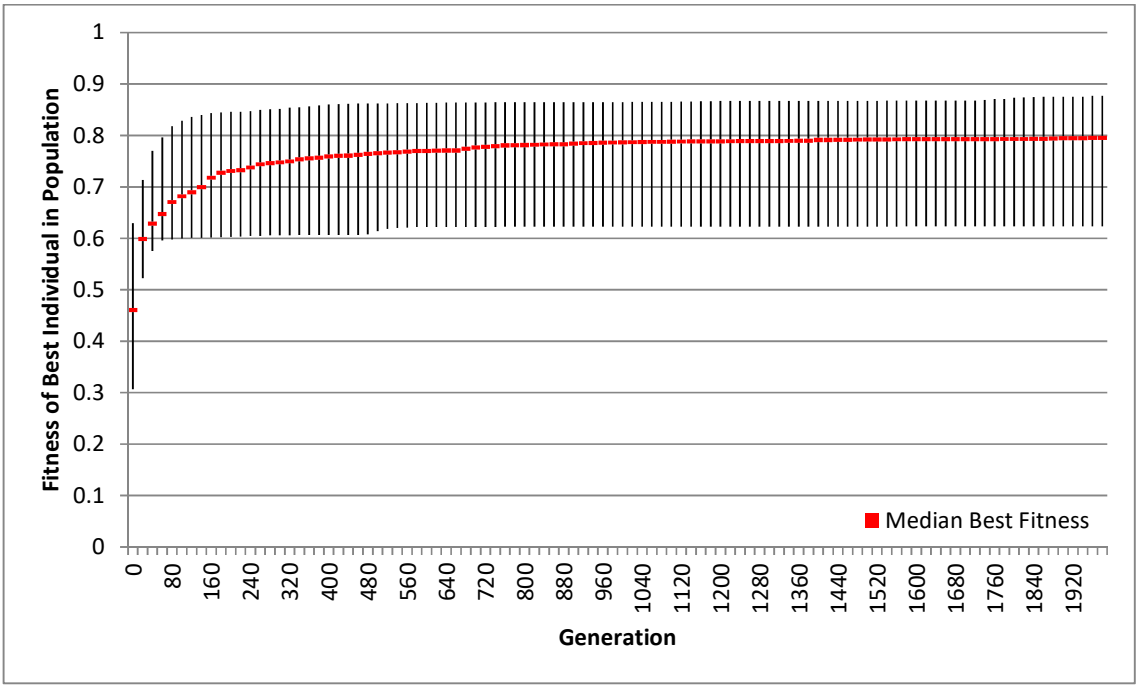


Figure 8.21. Fitness plot for the two hidden areas evaluation. It shows the highest, median, and lowest fitness values of the best individual that was evolved in each of the 30 runs of the evaluation. In at least one of the 30 runs, the best individual has probably converged to a local optima, resulting in a fitness value of 0.61.

The best individuals from the 30 runs that have low fitness values were analysed. The majority of evaluations that resulted in the best individual with a fitness in the range of [0.7, 0.8] showed that good hidden areas were found, but the geographical distance was less than the user-specified value. Six of the 30 runs resulted in a best individual with a fitness value below 0.7 and of these six, four individuals used the same generated area as both hidden areas while completely ignoring the distance constraint. In examining the weighting components of the fitness function (Equation 6.15, and Equation 6.16), it can be seen that the situation arises as the aggregated *asm* values of the two areas skews the fitness value of the terrain and resulting in the aggregated constraint having little impact (as in this case there is one

constraint). This scenario is less likely to happen in the event of more constraints being included.

8.2.2 Incorporating One Stronghold and One Hidden Area with Associated Constraints

This evaluation employed the developed approach to evolve a terrain with a stronghold and a hidden area that were separated by a small geographical distance of 16 units, which is one eighth of the terrain's width, but a high path distance of 512 units, which is four times the terrain's width. These constraints were chosen to simulate a common game level pattern, where the level forms a loop. This pattern was particularly common in the video game The Elder Scrolls V: Skyrim (2011), where each dungeon would loop the player back to the entrance so they didn't have to backtrack through a completed dungeon to return to the world map.

As with previous evaluations, the generated terrain using the individual with the best fitness from the 30 runs is displayed in Figure 8.22. This terrain shows that the stronghold and hidden area meet the criteria of their area types and, although placed close to one another geographically, there is no direct path between them. Owing to this condition, players are forced to travel the desired distance around the terrain to get to one area from the other. This figure shows the stronghold, represented as an orange sphere, having a single entrance, while being mostly surrounded by elevated terrain, granting the area cover. The hidden area is represented by a yellow sphere and is surrounded by elevated terrain with only a single entrance point. Although the two areas are placed next to one another, a steep section of terrain prevents direct access.

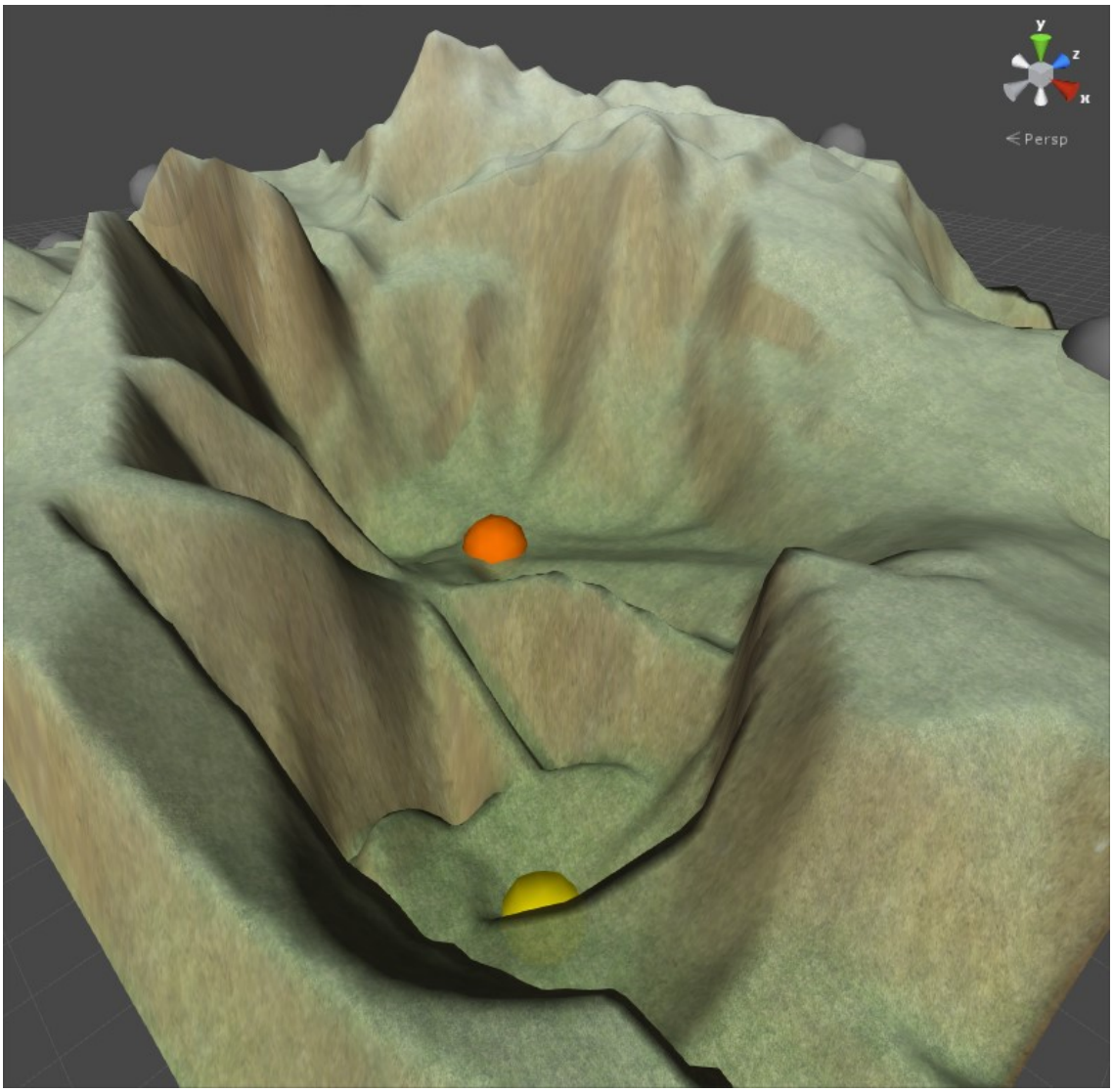


Figure 8.22. Generated terrain using the individual with the highest fitness from the 30 runs. The stronghold and hidden area are represented by the orange and yellow spheres respectively and are separated by a small geographical distance and large path distance.

Figure 8.23 shows the accessibility map of the terrain, representing this stronghold as an orange circle, and the hidden area as a yellow circle. A grey line shows the path that must be travelled to reach one area from the other. This is a good demonstration of obtaining the desired result, where the two areas are geographically close, but a large portion of the terrain must be traversed to reach one area from the other.

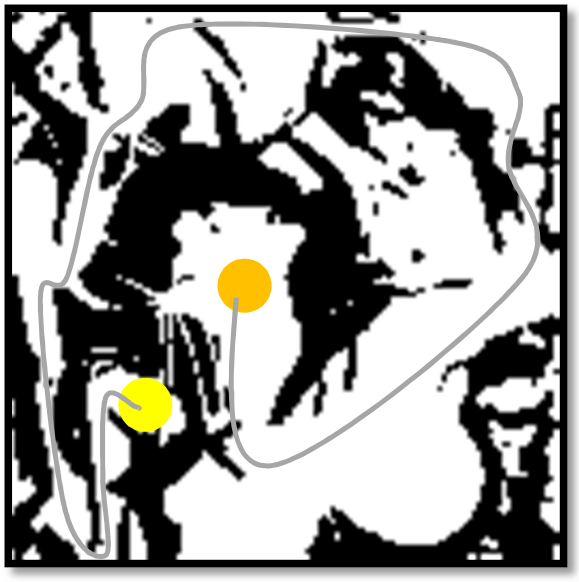


Figure 8.23. The accessibility map of the terrain containing a stronghold and hidden area. This accessibility map shows that the stronghold (orange circle) is geographically near the hidden area (yellow circle), but is cut off from it by non-traversable terrain. Therefore the path (grey) must be travelled to reach one area from the other.

Figure 8.24 shows the fitness plot associated with the 30 independent runs for this evaluation. The fitness plot shows the highest, median (red), and lowest fitness values associated with the best individual from each generation in the 30 runs. From the plot, it can be seen all 30 runs resulted in good solutions, with fitness values for the best individual in the range of [0.86, 0.91].

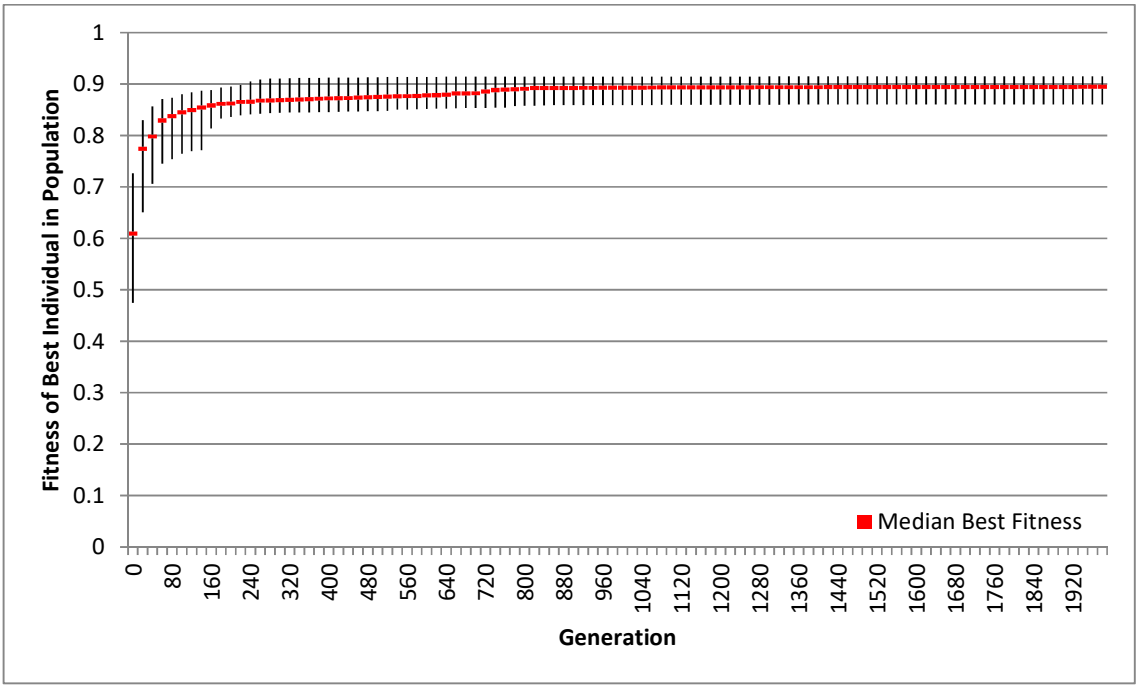


Figure 8.24. Fitness plot for the one stronghold and one hidden area evaluation. The plot shows the highest, median, and lowest fitness values of the individual with the highest fitness in each generation for the 30 runs of the evaluation.

From analysing individuals with lower fitness, it was discovered that they typically sacrificed accuracy for meeting the path distance constraint in order to satisfy the other fitness criteria. This meant the path distance between the stronghold and hidden area was shorter than the desired length. This is likely a result of using a higher theoretical maximum to normalise the path distance value, as this allows larger deviations between the generated path distance and its desired value, while resulting in small changes in the fitness of the individual.

8.2.3 Incorporating One Vantage Point and One Open Area with Associated Constraints

This evaluation used the developed approach to evolve a vantage point that looks over an open area into the *initial terrain*. Four constraints were placed between the two area types including a non-traversable constraint so that there was no direct path between the two areas, a small geographical distance constraint of 16 units, a line-of-sight constraint from the vantage point to the open area, and a no line-of-sight constraint from the open area to the vantage point.

The terrain generated using the individual with the highest fitness value from the 30 runs is shown in Figure 8.25 and represents a typical result for the 30 runs. This terrain shows an open area (green sphere) that is overlooked by a vantage point (red sphere) placed in the hillside. The vantage point is placed higher up to have a proper vantage over the open area and it is well covered by surrounding terrain. It is also not easily accessed.

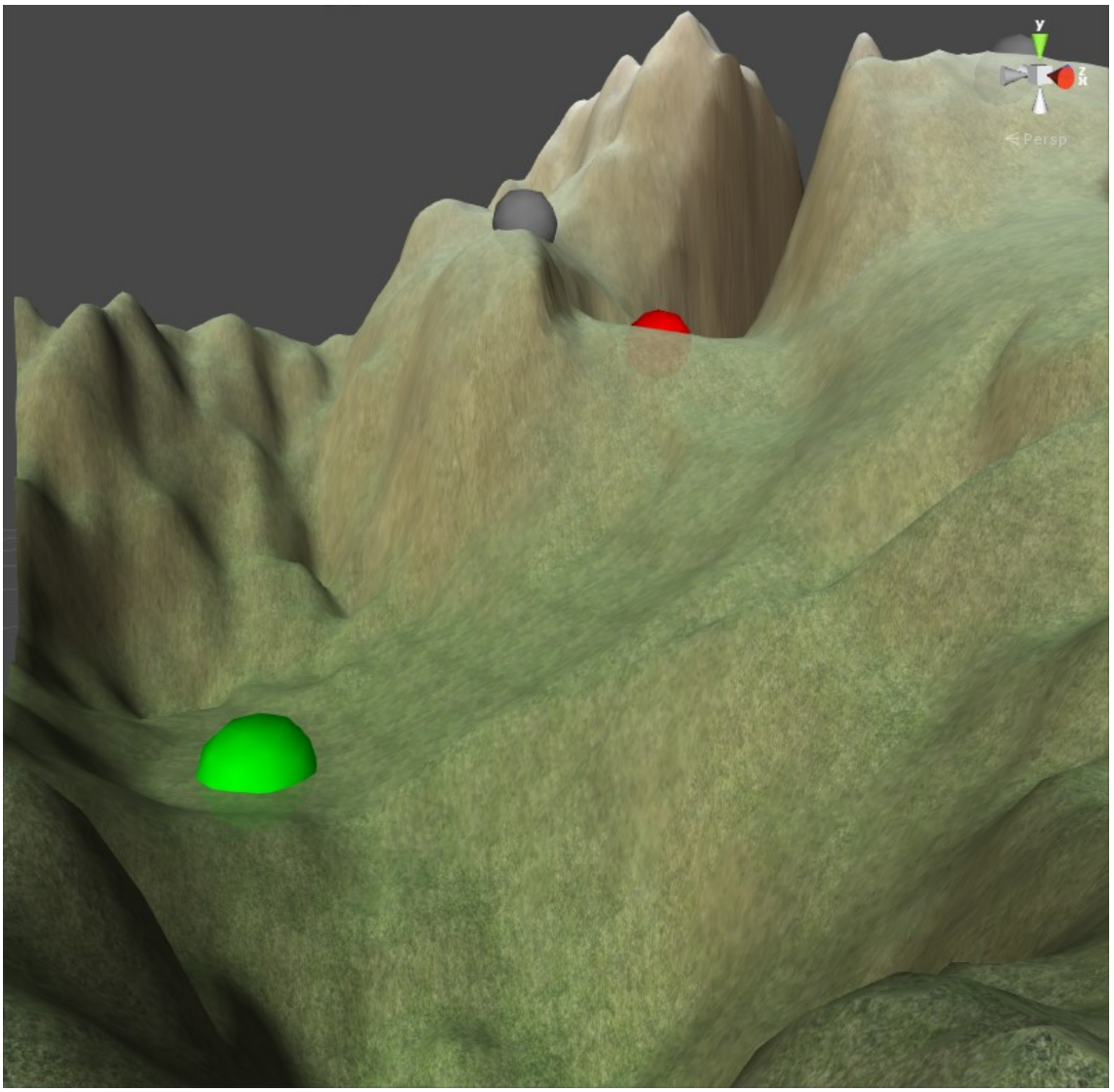


Figure 8.25. Generated terrain using the individual with the highest fitness from 30 runs of the vantage point and open area evaluation. The green and red spheres represent the open area and vantage point respectively.

Figure 8.26 shows the fitness plot associated with the 30 independent runs for this evaluation. This plot shows that all 30 runs of this evaluation found good results with the resulting individuals being assigned fitness values between 0.93 and 0.95.

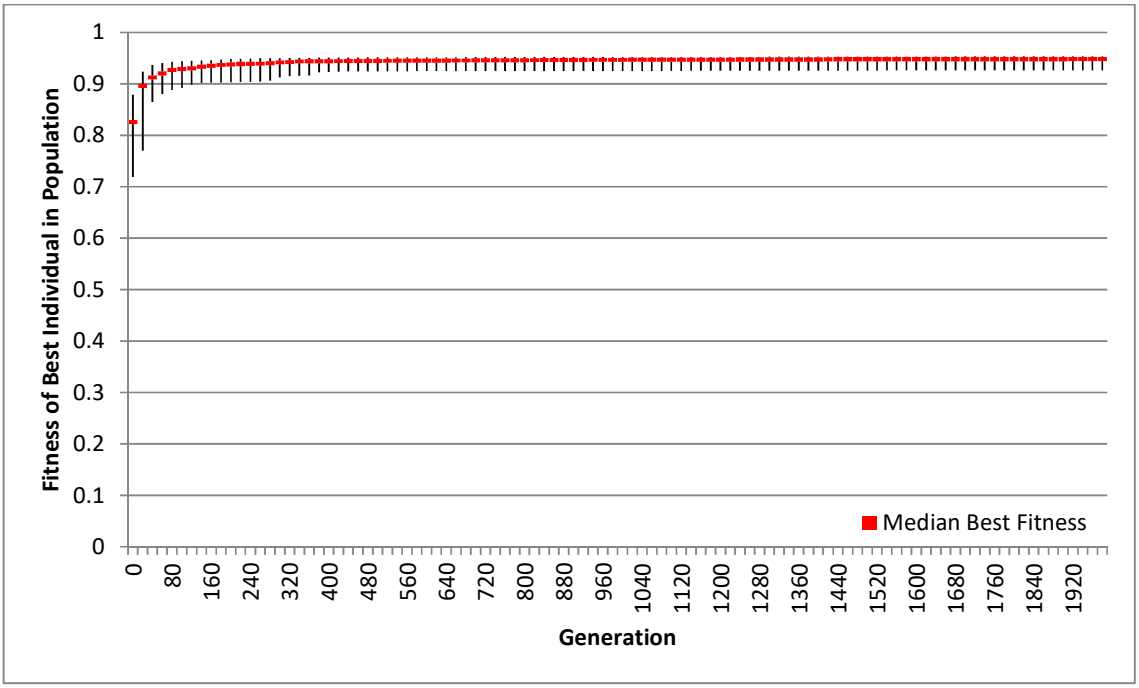


Figure 8.26. Fitness plot for the one vantage point and one open area evaluation. Shows the highest, median, and lowest fitness values of the best individual that was evolved in each of the 30 runs of the evaluation.

Although the single vantage point evaluations, detailed in Section 8.1.3, did not produce the most desirable areas, the evaluations described in this section show that, with the addition of a few constraints, this approach is capable of producing good vantage points with desirable characteristics.

8.2.4 Incorporating Two Strongholds and a Choke Point with Associated Constraints

This evaluation is to use the developed approach to evolve more than two desired areas into the *initial terrain*. The goal here was to generate a terrain with two strongholds, at opposite sides of the terrain, and have them separated by a choke point placed halfway between the two strongholds. To accomplish this, three geographical distance constraints were added, one to separate the two strongholds by 128 units (the length of the terrain), and two to separate each stronghold from the choke point by 64 units, which should place the choke point in the middle of the strongholds.

The terrain generated using the individual with the highest fitness from 30 runs is shown in Figure 8.27, where the strongholds are represented as orange spheres and the choke point is represented as a blue sphere. The generated strongholds exhibit some of the desired characteristics, as they are partially surrounded and have limited access. However, as shown

in the figure, the strongholds appear to be more exposed than the strongholds evolved in the evaluations detailed in Section 8.1.5. This is reflected by their lower *asm* values of 0.7 and 0.78. An explanation for why the strongholds were assigned lower *asm*s is that the distance constraint forced the strongholds to be placed in less suitable locations, such as the edge of the terrain. The choke point was assigned an *asm* of 0.85 and is placed at the entrance of one of the paths leading to one of the strongholds, although an alternate path to the stronghold is also present that does not require traversing through the choke point. The distance constraints were met almost perfectly with a distance of 127.9 units between the two strongholds, and distances of 63.5 and 64.8 units between the strongholds and the choke point.

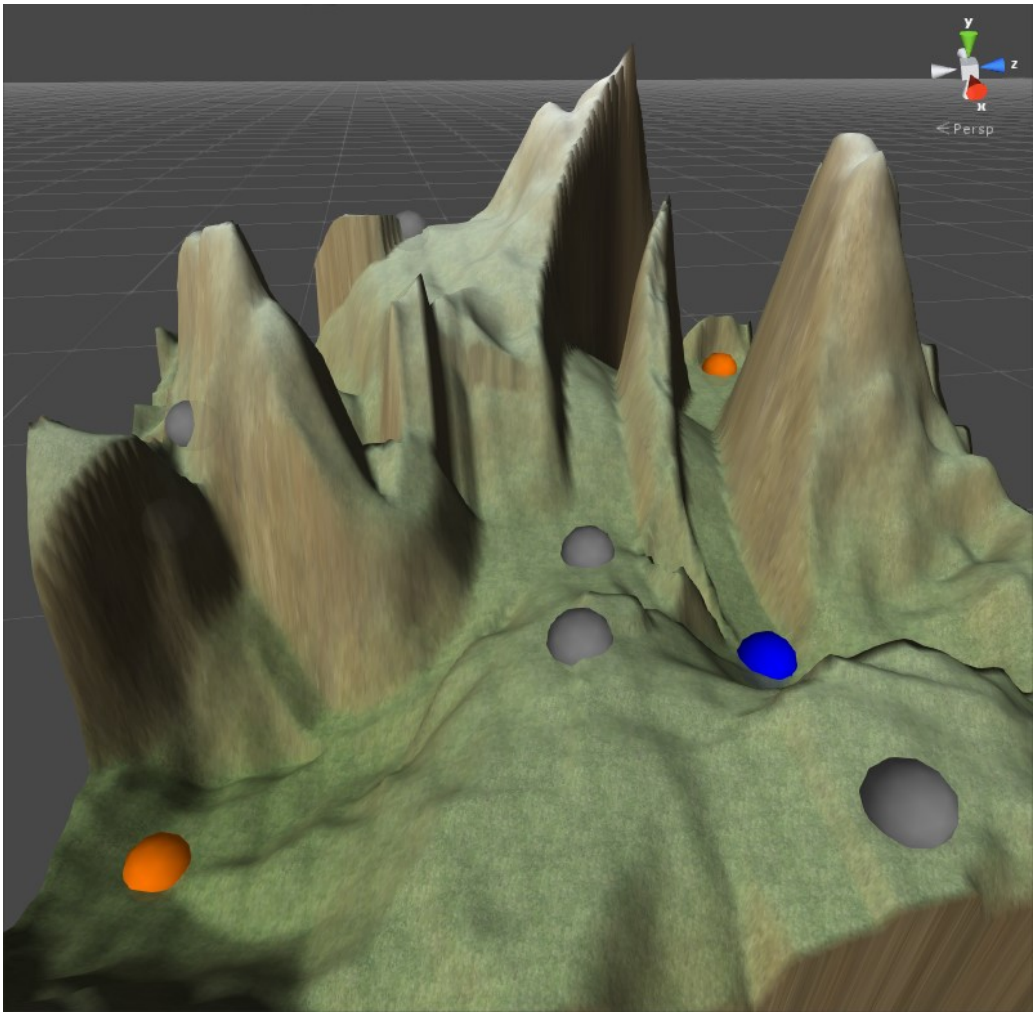


Figure 8.27. Generated terrain using the individual with the highest fitness from 30 runs of the two strongholds and one choke point evaluation.

Figure 8.28 shows the fitness plot associated with the 30 independent runs for this evaluation. Across the various generations in the 30 runs, it can be seen that the fitness value of the best individual lies in the range of approximately 0.78 to 0.89.

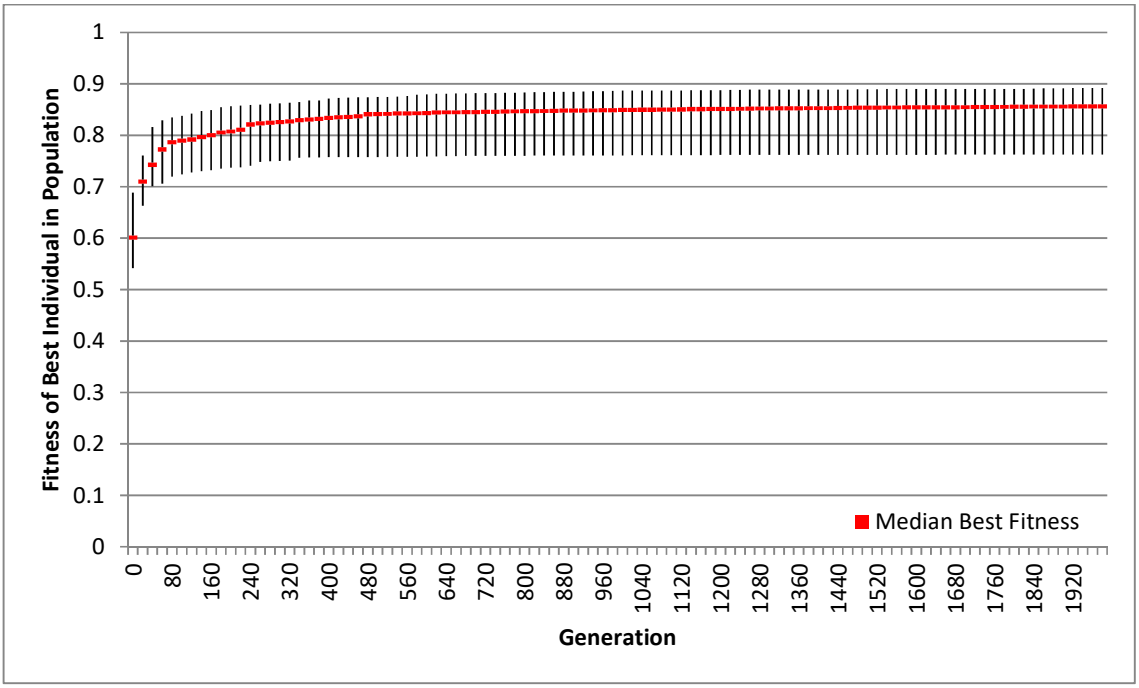


Figure 8.28. Fitness plot for the two strongholds and one choke point evaluation. Shows the highest, median, and lowest fitness values of the best individual that was evolved in each of the 30 runs of the evaluation. In at least one of the 30 runs, the best individual has probably converged to a local optima, resulting in a fitness value of 0.77.

8.3 Category III: Goal Layout Evaluations

The *initial terrain* used for evaluations in this category is shown in Figure 8.1. In addition, these evaluations will use five level designs, extracted from existing terrains. These five existing terrains are player-made for the video game "Savage: Battle for Newerth" (S2 Games, 2003) and were retrieved from the website "http://www.newerth.com/". These terrains were selected based on two main factors, the presence of known level design patterns, such as symmetry, and the inclusion of desired area types. These game levels were converted to input graphs by manually identifying the areas of specific area types (the hidden areas, open areas, vantage points, choke points, and strongholds) and adding them as desired areas to the *input graph*. The constraints between all pair combinations of the desired areas were then computed and added to the *input graph*. These five game levels are displayed in Figure 8.29, with each level's *layout graph* overlaying the terrain. Yellow, green, red, blue, and orange spheres represent manually identified hidden areas, open areas, vantage points, choke points, and strongholds respectively, and the black sections represent manually identified paths between areas.

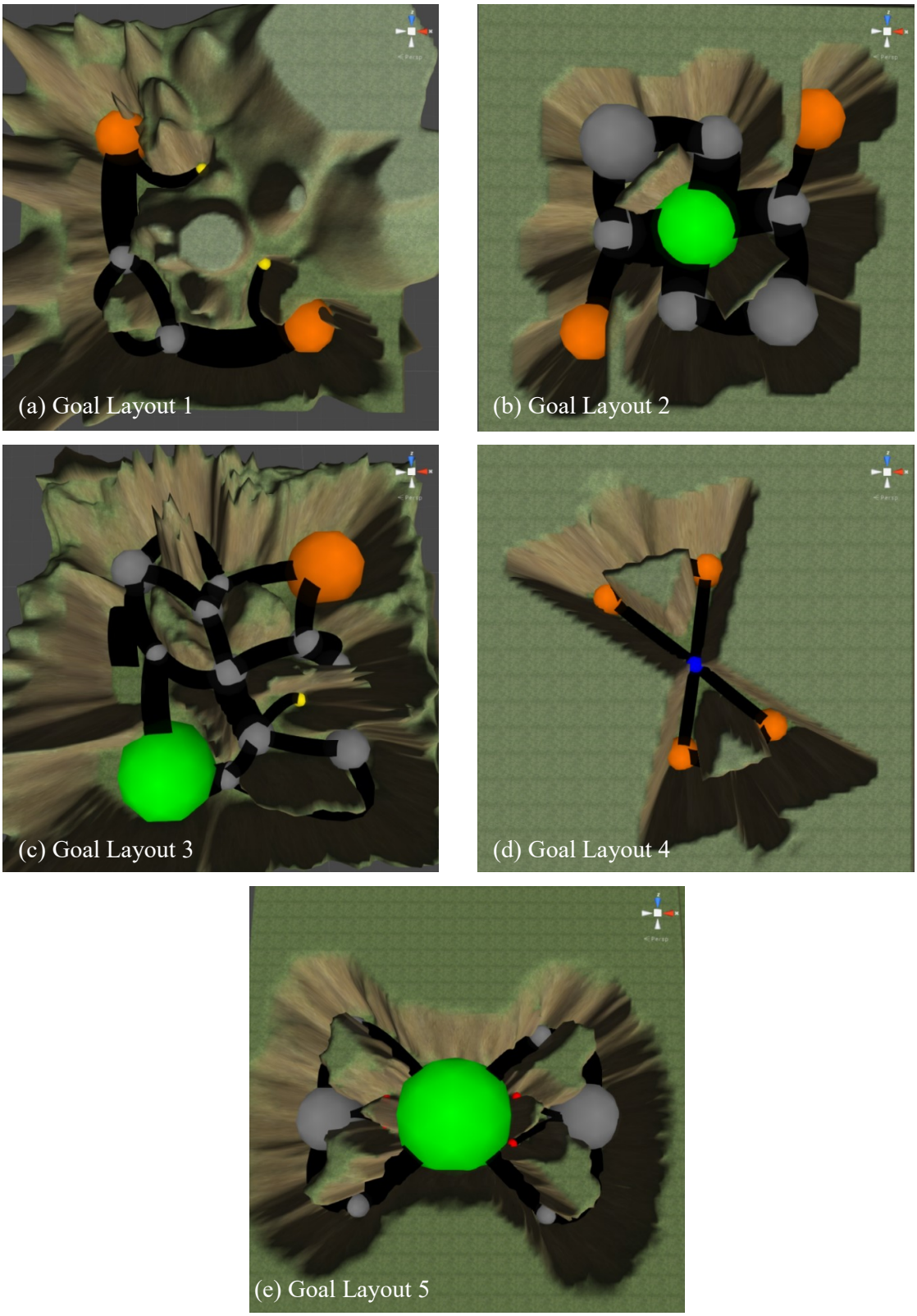


Figure 8.29. Top down views of the five player-made levels for the video game "Savage" that were used as goal layouts for the goal layout evaluations. These images are screenshots taken from Unity (2018). Yellow, green, red, blue, and orange spheres represent manually identified hidden areas, open areas, vantage points, choke points, and strongholds respectively, and black sections represent manually identified paths between areas.

Goal layout 1 shows a symmetrical level, with a stronghold placed at each end of the terrain and hidden areas placed nearby. Goal layout 2 is another symmetrical level, where players can make their bases in the two strongholds and meet in the open area, placed in the centre, for open combat. Goal layout 3 is asymmetrical, and therefore lacking the structure of the previous two layouts. These types of layouts are typically better suited for exploration as players are more likely to get confused and lost. This layout contains an open area, a stronghold, and a hidden area, where the player may find useful items. Goal layout 4 consists of four strongholds, two placed at each end of the map, and a single choke point separating them. This setup forces players on opposing sides to confront each other at a central location, with no areas to explore or alternate routes for stealth attackers. Goal layout 5 is another symmetrical level, where two opposing teams can meet in the central open area for combat, while also offering vantage points for a stealthy approach, using snipers to attack from a distance.

The five evaluations detailed in this section are labelled EXP1, EXP2, EXP3, EXP4, and EXP5, which employed the developed approach to evolve the extracted goal layouts 1, 2, 3, 4, and 5 respectively into a user-specified terrain. Figure 8.30 shows top down views of terrains generated using the individual with highest fitness evolved during each evaluation.

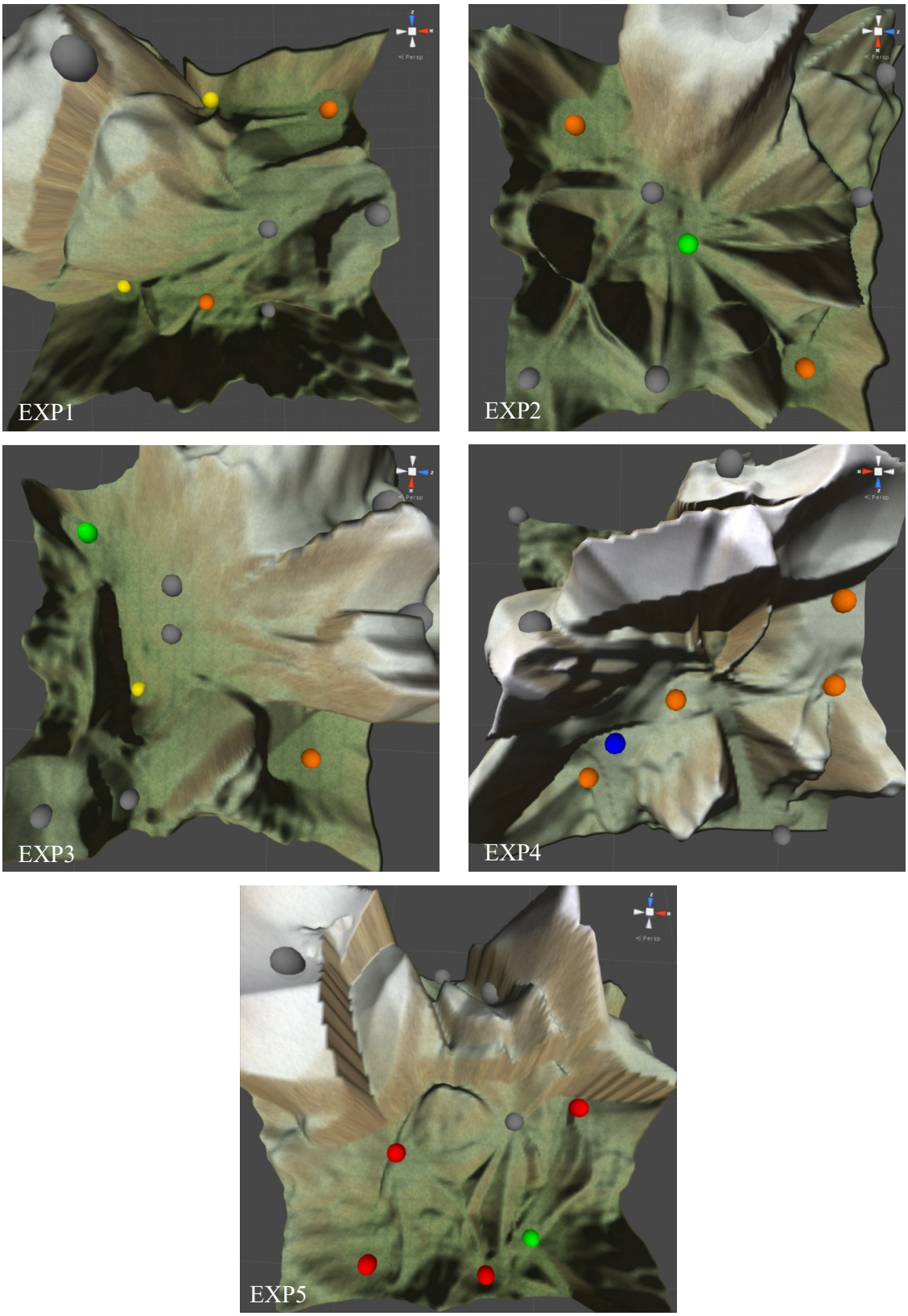


Figure 8.30. Images of the terrains generated from the best individual evolved in the evaluations EXP1, EXP2, EXP3, EXP4, and EXP5 respectively. Yellow, green, red, blue, and orange spheres represent hidden areas, open areas, vantage points, choke points, and strongholds respectively.

These images show that most of the evolved level layouts are similar to their goal layouts in terms of the placement of areas relative to one another. The terrain for EXP1 shows two strongholds at opposite ends of the terrain, with a hidden area placed near both strongholds. The top-right and the bottom-left strongholds were assigned asm values of 0.82 and 0.76 respectively. The top-right stronghold is well covered with two entry points, while the bottom-left stronghold is more open with three entry points. The two hidden areas, both assigned *asm*s of 0.85, are placed in small nooks that are well surrounded by elevated terrain.

The terrain for EXP2 shows two strongholds at opposite corners of the terrain, suitable for player bases, and an open area in the middle, where players can meet for combat. Both strongholds are well surrounded with limited access and were assigned *asm* values of 0.82 and 0.77. The open area was assigned an *asm* of 0.86, is placed in the open, and is easily accessible.

The terrain for EXP3 shows an open area and a stronghold placed at opposite corners of the terrain with a hidden area placed in between, in a similar manner to goal layout 3. The open area, with an *asm* of 0.81, is well exposed, but its access is restricted by its placement in the corner of the terrain. The hidden area was assigned an *asm* of 0.86 and is a small area surrounded by elevated terrain. The stronghold is a larger area, well covered by elevated terrain, has limited access, and was assigned an *asm* of 0.84.

EXP4 and EXP5 where the most complex experiments in terms of the number of desired area types to be incorporated and the number of constraints. The terrains from these two evaluations were not as accurate in meeting the desired level layout. The choke point in EXP4 was not placed in the centre of the four strongholds, and neither was the open area placed in the centre of the four vantage points in EXP5. Additional analysis of all the generated terrains revealed that none of the desired areas were placed at the same terrain location, as was the case in the evaluations detailed in Section 8.2.1.

Figure 8.31 and Figure 8.32 show box and whisker plots that display the deviations between desired distance constraint values and the actual distance constraint values in the 30 generated terrains of each goal layout evaluation. Each plot shows the minimum, maximum, and interquartile ranges of these deviations, where the tops of the green boxes show the upper quartiles, the bottoms of the red boxes show the lower quartiles, and the median is where the green and red boxes meet. The vertical axis of these plots shows the deviation distance,

where values of zero are desired. This axis ranges from 0 to the theoretical maximum of the associated constraint for the plot (221 for geographical distance and 2194 for path distance). Calculation of the theoretical maximum values is detailed in Section 6.2.3.2.

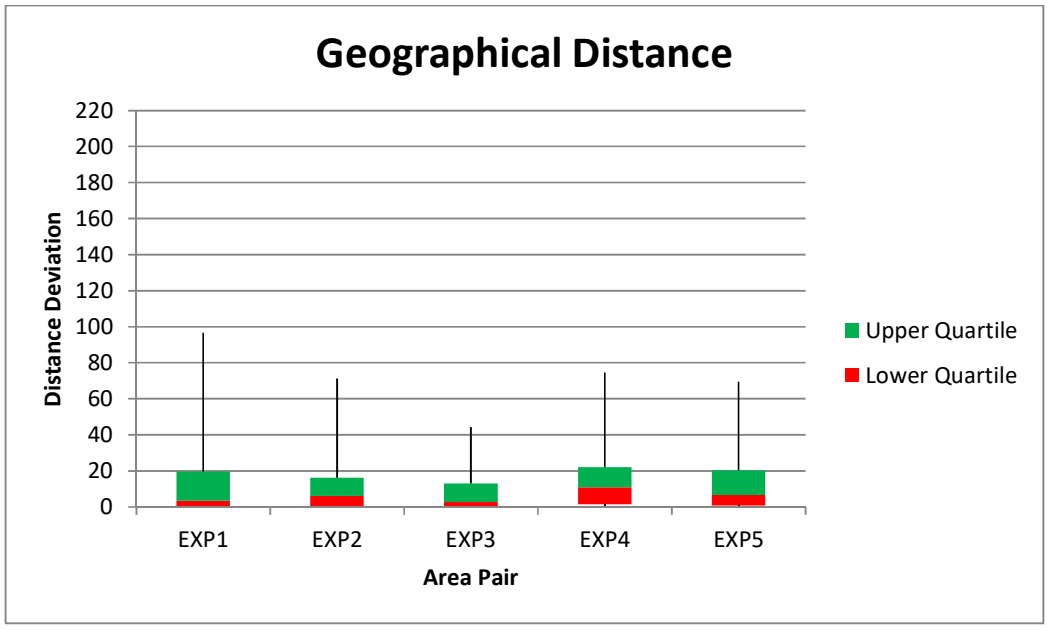


Figure 8.31. Box and whisker plot showing the minimum, maximum, and interquartile ranges of the deviations between desired and actual geographical distance constraint values across the five evaluations.

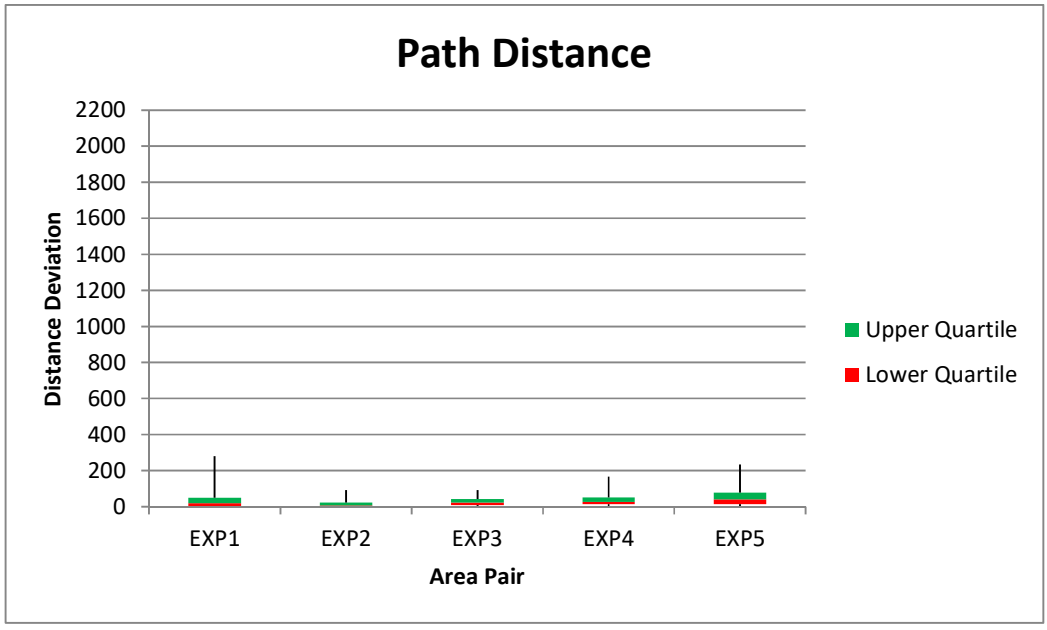


Figure 8.32. Box and whisker plot showing the minimum, maximum, and interquartile ranges of the deviations between desired and actual path distance constraint values across the five evaluations.

The geographical distance plot show that the majority (~75%) of geographical distance constraints deviate from their desired values by 20 units or less. The median values show that

at least 50% of the geographical distance constraints for EXP1, EXP2, EXP3, and EXP5 deviate from their desired values by 6.5 units or less. The median value for EXP4, one with one of the most complex *input graph*s used for evaluations, is 11 units. The path distance plot shows that 75% of all path distance constraints in EXP1, EXP2, EXP3, and EXP4, deviate from their desired values by less than 26 units. The upper quartile for EXP5, one of the most complex evaluations, is 88 units.

Comparing the two plots from Figure 8.31 and Figure 8.32, it is clear that the path distance constraints were typically more accurately met than the geographical distance constraints, in comparison to their respective theoretical maximum values. Additionally, the two simplest evaluations, EXP2 and EXP3, typically met their geographical and path distance constraints more accurately than the more complex evaluations.

The line-of-sight and traversability constraints were almost always met, with 1 to 4 out of 30 terrains from each evaluation, not meeting all constraints. The one exception to this is EXP5, where only four terrains met all line-of-sight and traversablility constraints. Only two of the 300 traversability constraints (10 per terrain) failed, but 77 of the 600 line-of-sight constraints (20 per terrain) failed. This is likely due to the large number of vantage points in these terrains, where each vantage point should have line-of-sight to the open area, while the open area should not have line-of-sight back to the vantage point, which is a more difficult constraint to meet than having line-of-sight both ways or not having line-of-sight at all.

Figure 8.33, Figure 8.34, Figure 8.35, Figure 8.36, and Figure 8.37 show box and whisker plots for the *asm* values for each desired area type in each of the five evaluations. The theoretical maximum *asm* value is 1. These plots show that the *asm* values of the areas evolved in the three evaluations, EXP1, EXP2, and EXP3, are typically high, although some area types generally achieved higher *asm* values than others. Almost all hidden areas in these three evaluations have an *asm* between 0.8 and 0.88, and almost all open areas in these evaluations have an *asm* of 0.85. Stronghold *asm* values are generally lower, rarely reaching an *asm* above 0.8.

The areas evolved involving the two most complex goal layouts, EXP4 and EXP5, typically have lower *asm* values than areas evolved in the other three evaluations, as shown by their larger interquartile ranges. The lower quartile of *asm* values for EXP4 is typically below 0.4, and the upper quartile between 0.6 and 0.8. The *asm* values of evolved areas in EXP5 are

generally higher than those of EXP4, with the lower quartiles being greater than 0.65 and the upper quartiles always above 0.8.

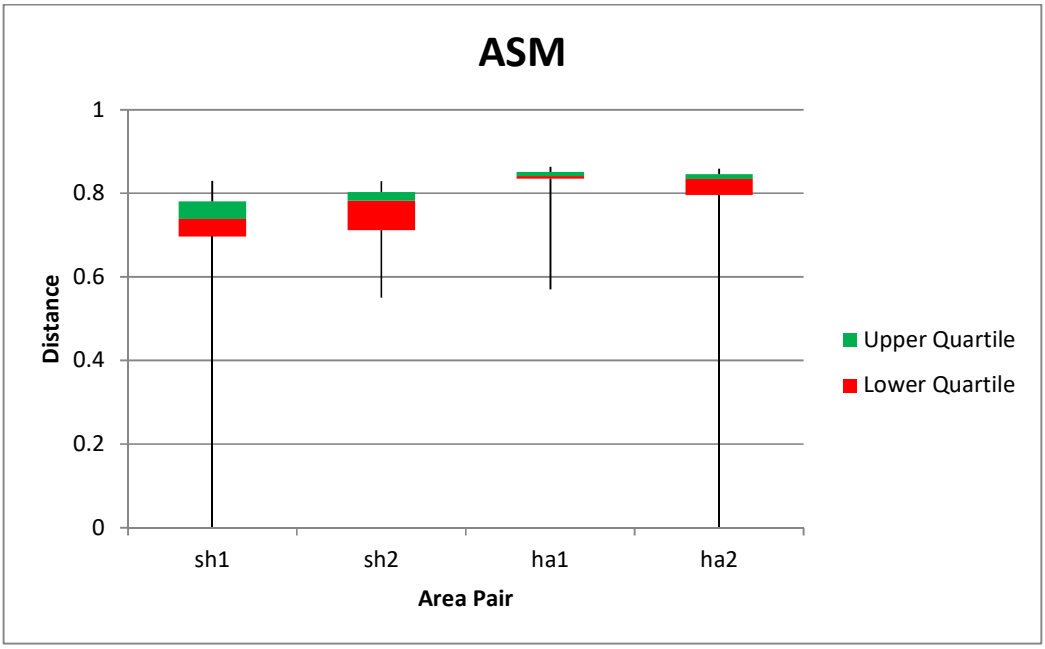


Figure 8.33. Box and whisker plot showing the interquartile range of *asm* values for desired area types across the 30 final terrains produced in EXP1.

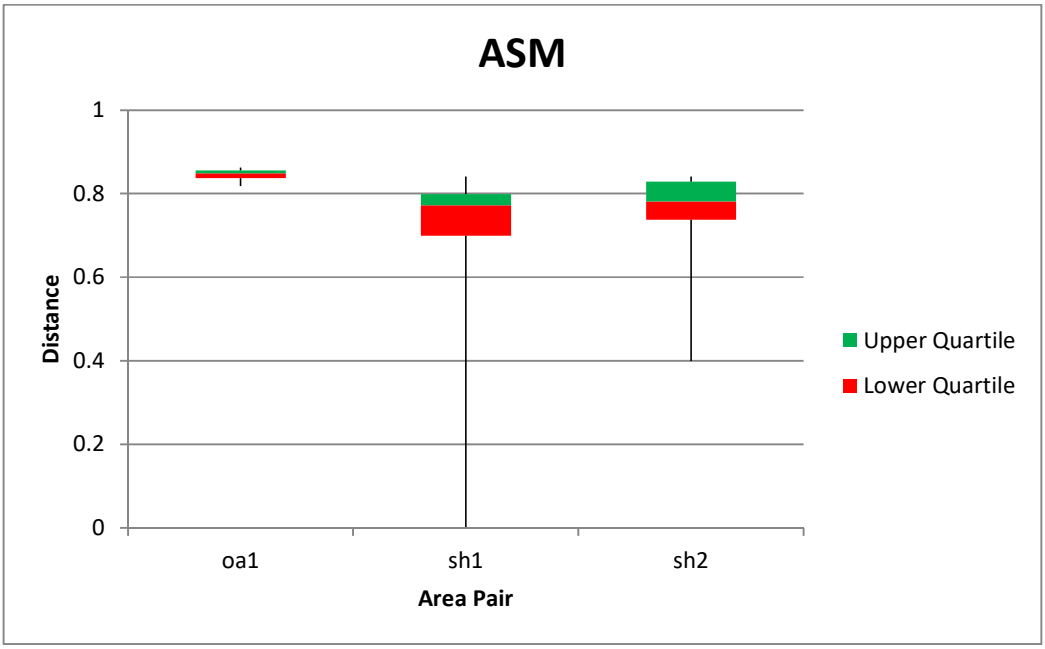


Figure 8.34. Box and whisker plot showing the interquartile range of *asm* values for desired area types across the 30 final terrains produced in EXP2.

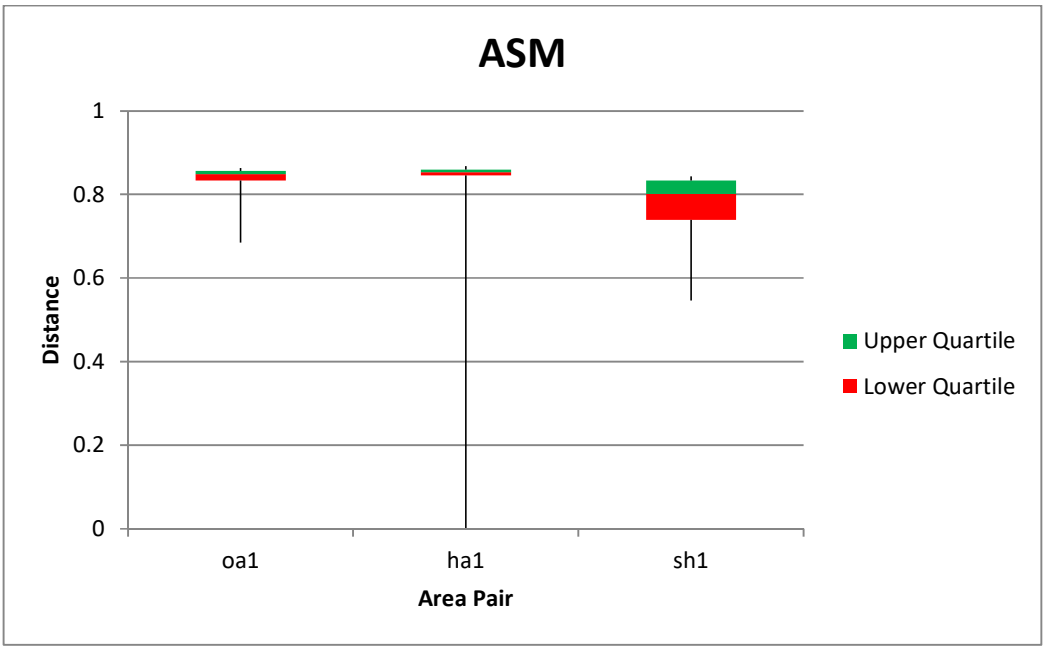


Figure 8.35. Box and whisker plot showing the interquartile range of *asm* values for desired area types across the 30 final terrains produced in EXP3.

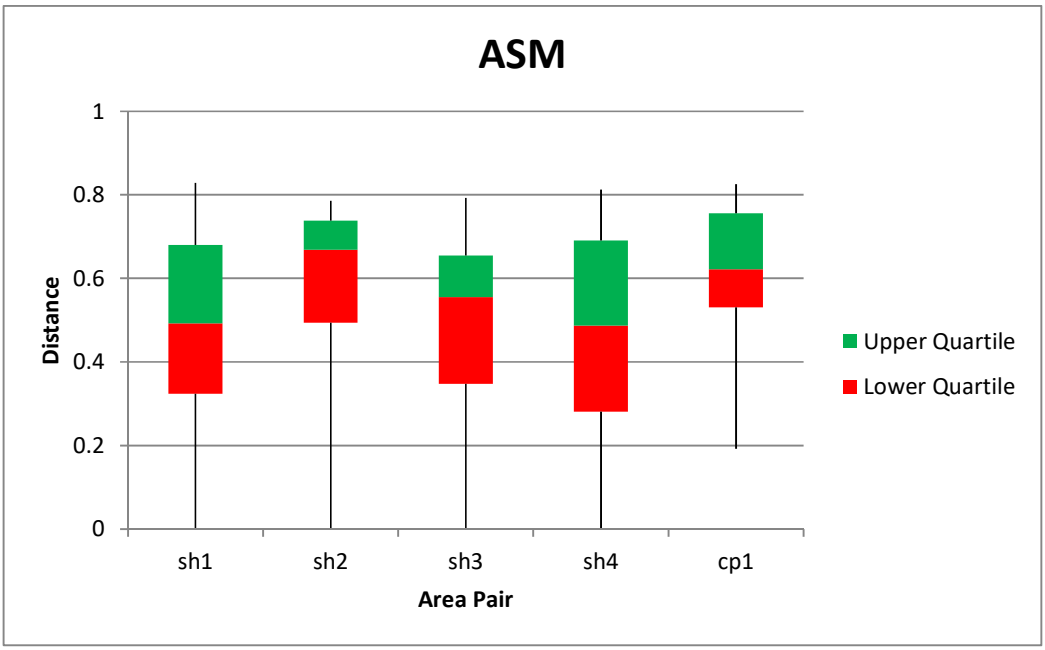


Figure 8.36. Box and whisker plot showing the interquartile range of *asm* values for desired area types across the 30 final terrains produced in EXP4.

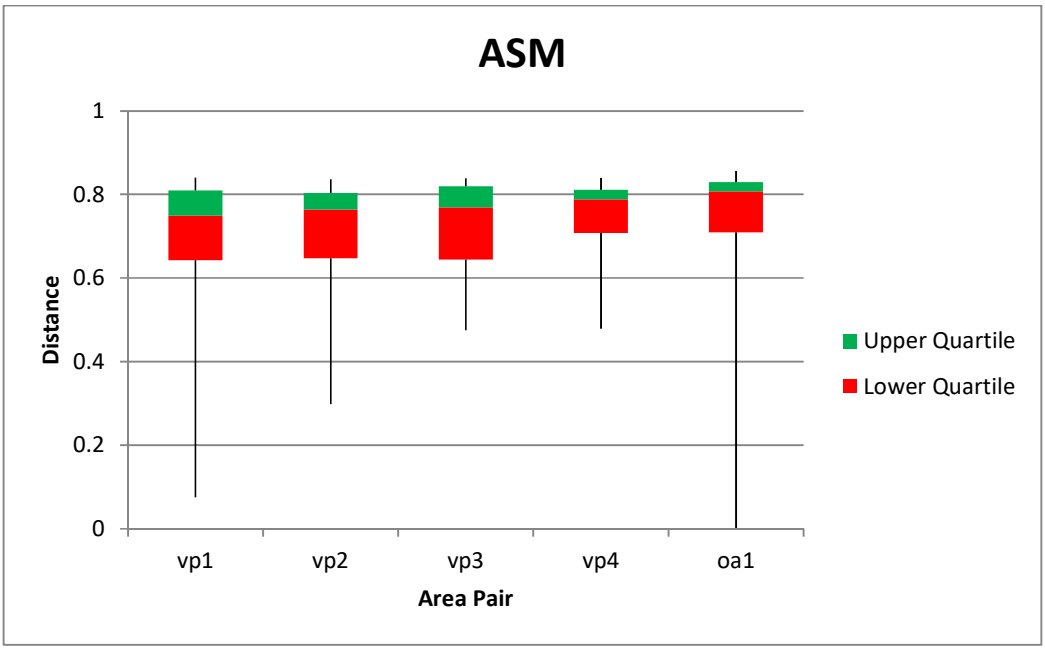


Figure 8.37. Box and whisker plot showing the interquartile range of *asm* values for desired area types across the 30 final terrains produced in EXP5.

Figure 8.38, Figure 8.39, Figure 8.40, Figure 8.41, and Figure 8.42 show the fitness plots associated with the 30 independent runs for these five evaluations. The plots show that most runs of these five evaluations evolved good individuals with the best individuals having fitness values equal to or greater than 0.97, except for EXP4, where the best individual has a fitness value of 0.95. The median values show that at least 50% of the evolved individuals in EXP1, EXP2, and EXP3 are assigned fitness values in the range of [0.96, 0.97]. The median values for the two most complex evaluations, EXP4 and EXP5, are still high values in the range of [0.91, 0.92]. The fitness of the lowest fitness individuals among the 30 individuals associated with each generation of the 30 runs in these two evaluations is 0.86, whereas in the simpler evaluations these individuals have approximate fitness values of 0.9.

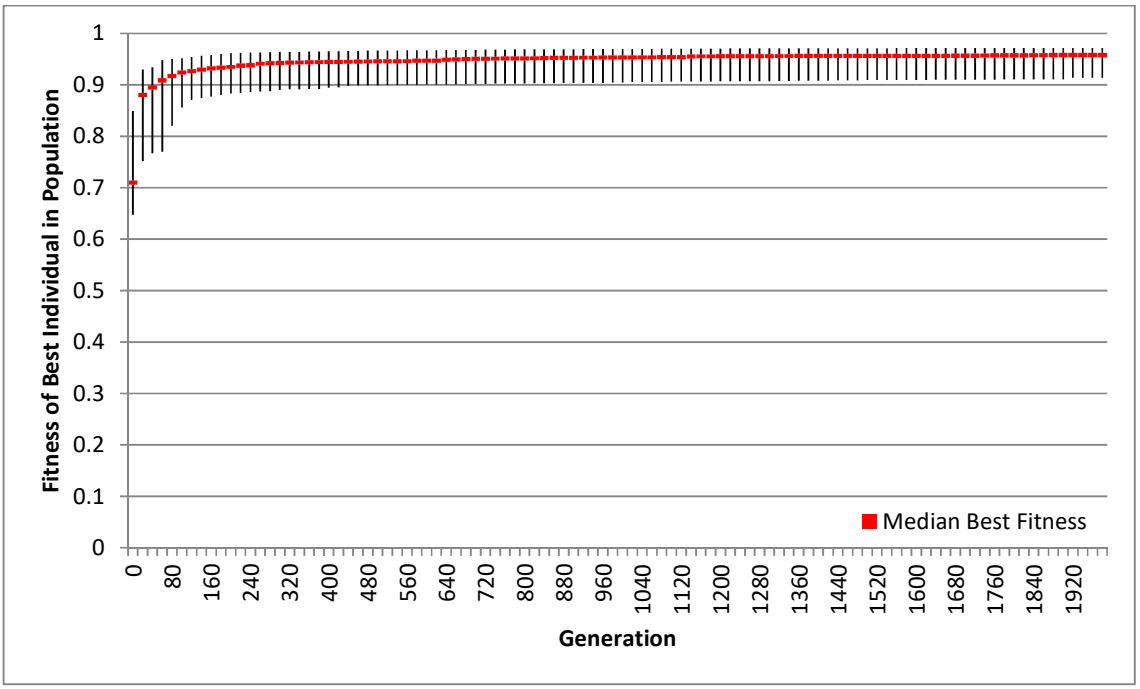


Figure 8.38. The fitness plot for evaluation EXP1. The plot shows the highest, median, and lowest fitness values of the individual with the highest fitness in each generation for the 30 runs of the evaluation.

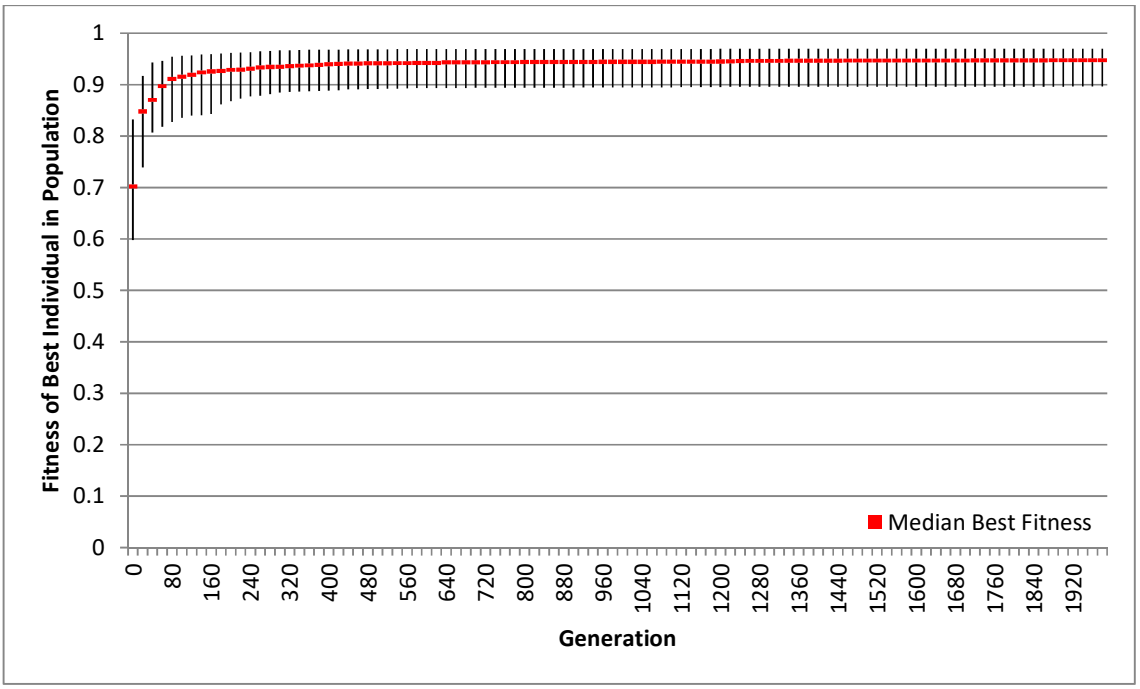


Figure 8.39. The fitness plot for evaluation EXP2. The plot shows the highest, median, and lowest fitness values of the individual with the highest fitness in each generation for the 30 runs of the evaluation.

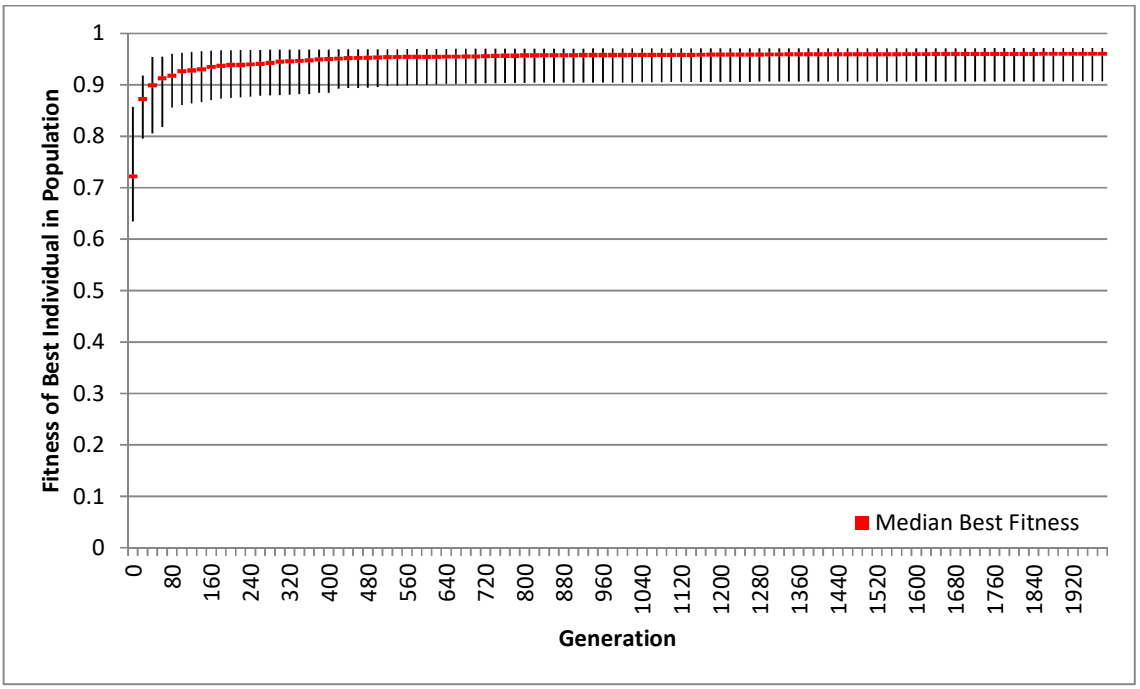


Figure 8.40. The fitness plot for evaluation EXP3. The plot shows the highest, median, and lowest fitness values of the individual with the highest fitness in each generation for the 30 runs of the evaluation.

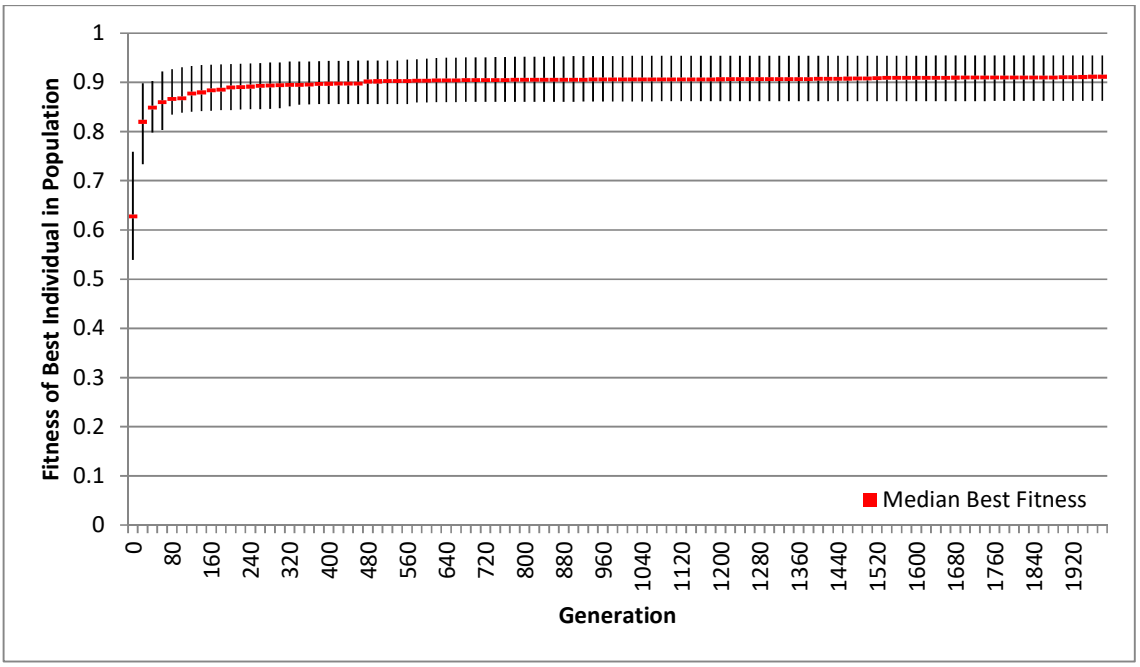


Figure 8.41. The fitness plot for evaluation EXP4. The plot shows the highest, median, and lowest fitness values of the individual with the highest fitness in each generation for the 30 runs of the evaluation.

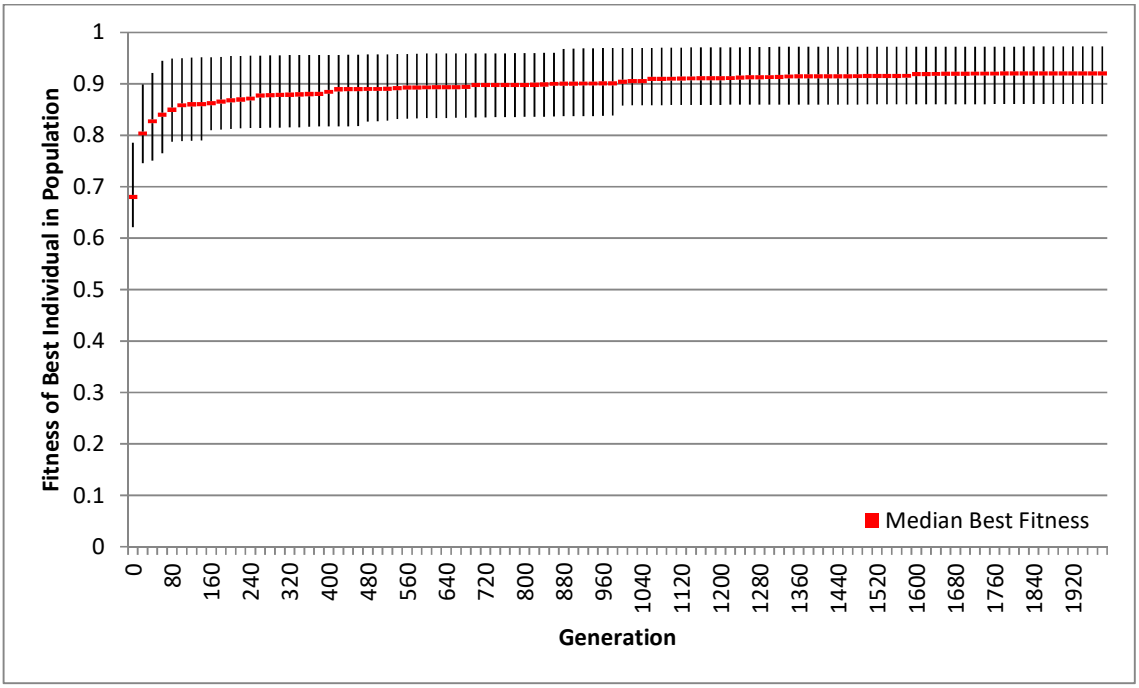


Figure 8.42. The fitness plot for evaluation EXP5. The plot shows the highest, median, and lowest fitness values of the individual with the highest fitness in each generation for the 30 runs of the evaluation.

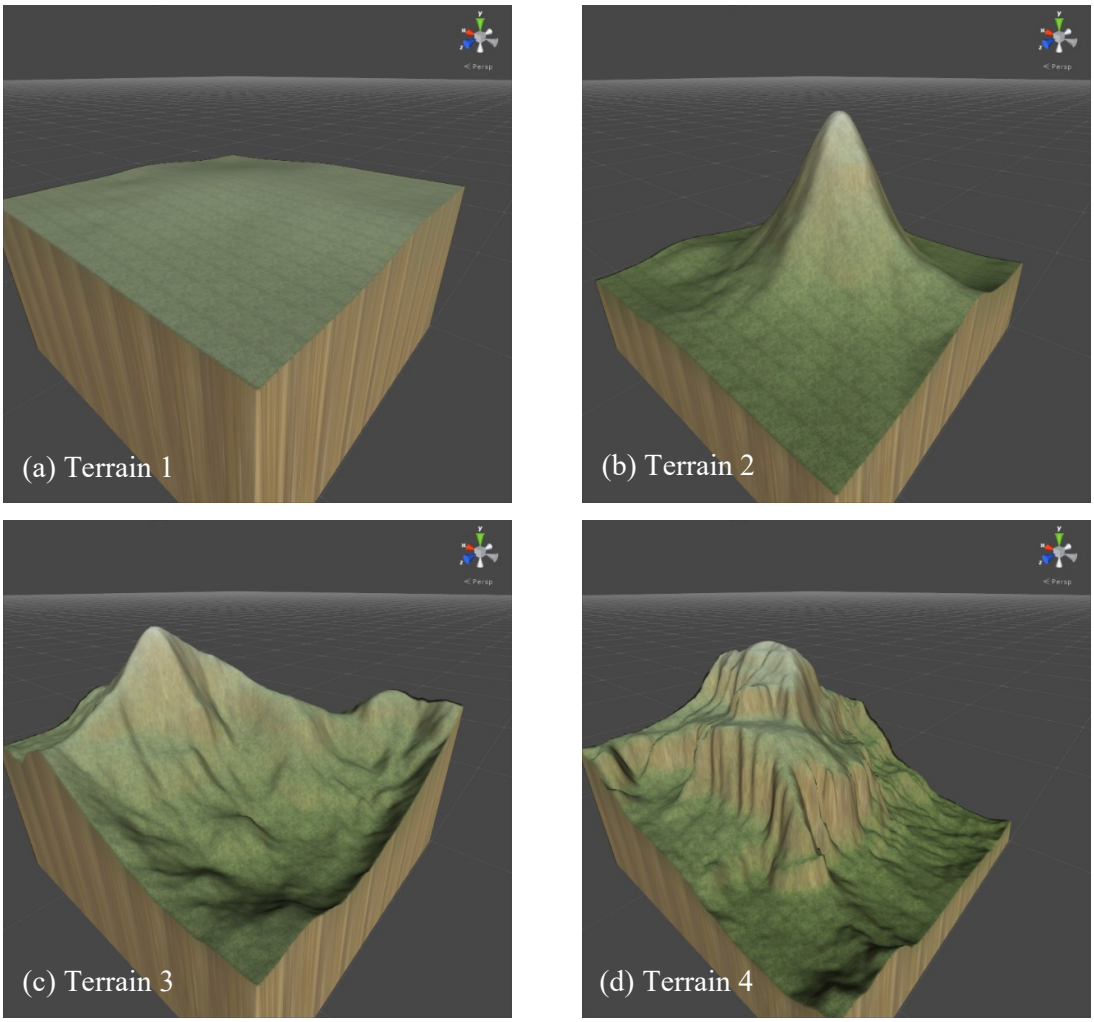
This section detailed the results of five evaluations, each using the developed approach to evolve a goal layout into the *initial terrain*. The results from these evaluations show that goal layouts can be evolved accurately. The simpler evaluations that consisted of goal layouts with less desired area types and constraints, typically performed better than those with the more complex goal layouts, but good solutions were found in all evaluations.

8.4 Category IV: Evaluation of the Effects from Using Different Initial Terrains

This section evaluates how well the developed approach is at incorporating the same configuration of area types into different initial terrains. For example, the approach may evolve better hidden areas in a rough terrain compared to a smooth terrain that has no mountains to provide cover, or it may find smooth terrains better for incorporating open areas. This evaluation is performed by examining the *asm* values of evolved areas from 25 evaluations and is detailed in Section 8.4.1. Additionally the generated terrains are evaluated to determine how much the evolved modifications deformed the initial shape of the terrain, where minimal deformation is prefered as the *initial terrain* is chosen by a game designer for aesthetic reasons. This evaluation is performed through visual analysis of the generated terrains and a deformation measure as detailed in 1.1.1.

8.4.1 Analysis of ASM Variances From Using Different Initial Terrains

Five different *initial terrain*s were used in these evaluations and these are shown in Figure 8.43. These terrains range from an almost flat terrain (Terrain 1), to a very rough terrain with almost no flat areas (Terrain 5). Each of these terrains was generated using the program L3DT v16.05 (Bundysoft, 2018), achieving terrains with varying ruggedness by manipulating the generation parameters that L3DT provides. For each of these *initial terrain*s five evaluations were conducted, each using one of the five input graphs used in the goal layout evaluations detailed in Section 8.3, resulting in a total of 25 evaluations. Each of these 25 evaluations was runs 30 times.



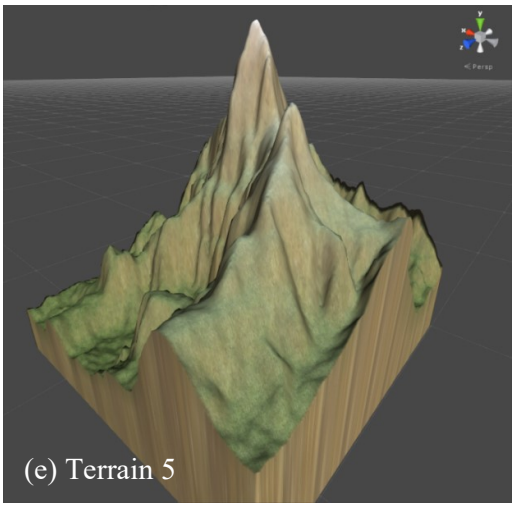


Figure 8.43. Images of the five initial terrains that were used in these evaluations. These terrains range in roughness from smoothest (terrain 1) to roughest (terrain 5).

Table 8.6 lists the average *asm*s of each area type in the evolved terrains across the 30 runs of each of the 25 evaluations, where each evaluation is defined by a unique combination of *input graph* and *initial terrain*. Area types are referenced as follows: HA = Hidden Area, OA = Open Area, VP = Vantage Point, CP = Choke Point, and SH = Stronghold. This table shows that varying the *initial terrain* typically did not significantly impact the *asm* values of the generated areas. For example, the 30 runs of the evaluations that used *input graph* 1, which attempted to incorporate two strongholds and two hidden areas, resulted in an average stronghold asm in the range of [0.70, 0.72].

One-way analysis of variance (ANOVA) tests were used to analyse this data using a significance level of 0.05. These ANOVA tests are used to determine if there is a significant variation in the *asm* values of a specific area type when varying the *initial terrain*. Two, one-way ANOVAs, were performed for *input graph*s 1, 2, 4, and 5, as these *input graphs* incorporated two different area types. Three, one-way ANOVAs, were performed for *input graph* 3, as it incorporates three different area types, OA, HA, and SH. Each ANOVA had five independent groups, one for each of the *initial terrain*s. Each group consisted of the *asm* values of the generated areas of the specified type from the best individual of each of the 30 runs of the evaluation. Therefore the ANOVA for the area type ‘stronghold’ for *input graph* 1 consists of five independent groups. Group 1 consists of the 60 stronghold *asm* values, as there are two strongholds in each terrain, from the 30 runs of the evaluations that used terrain 1. Group 2 consists of the 60 stronghold *asm* values from the 30 runs of the evaluations that used terrain 2. Groups 3, 4, and 5 contained the same data from the evaluations that used terrains 3, 4, and 5.

Table 8.6. A list of the average *asm* values across the 30 runs of each evaluation. The average *asm*s are categorised by the initial terrain of the evaluation (column), the goal layout of the evaluation (row 1) and the areas' type (row 2), where HA = hidden area, OA = open area, VP = vantage point, CP = choke point, and SH = stronghold.

		Terrain 1	Terrain 2	Terrain 3	Terrain 4	Terrain 5
Input Graph 1	*SH*	0.721113	0.690805	0.725407	0.703143	0.701748
	HA	0.79192	0.783122	0.819043	0.732579	0.763951
Input Graph 2	*OA*	0.855213	0.840717	0.848932	0.839848	0.840335
	SH	0.75998	0.727355	0.766858	0.758924	0.747344
Input Graph 3	*OA*	0.84033875	0.823364	0.844107	0.824287	0.817187
	HA	0.764024097	0.852829	0.852345	0.849061	0.843798
	SH	0.765819167	0.759228	0.763019	0.771533	0.747203
Input Graph 4	*SH*	0.484926	0.476719	0.495141	0.488836	0.520233
	CP	0.639478	0.631372	0.651472	0.674544	0.669747
Input Graph 5	*VP*	0.691745	0.664477	0.705542	0.708364	0.733968
	OA	0.743097	0.785019	0.812445	0.780009	0.789832

Only two of the ANOVA results showed that there was a significant variation in their *asm* values. These ANOVA tests were for hidden areas in *input graph* 3 and vantage points in *input graph* 5. Post-hoc tests were performed for these two cases by performing two-sample t-tests and the Bonferroni correction between each pair of groups. Since there are five groups in each ANOVA test, ten t-tests were performed for each ANOVA. This made the corrected significance level equal 0.005. These tests failed to show a significant variance between any two groups for the hidden areas in *input graph* 3, but looking at the mean values listed in Table 8.6 its clear that the average hidden area *asm* for terrain 1 is lower than the average *asm* values for the other four terrains. Looking at the *asm* value of each individual hidden area in the generated terrains of these five evaluations, it was discovered that most hidden areas that were evolved into terrain 1 had similar *asm* values to those in the other four terrains and that the average was brought down by three hidden areas that were assigned an *asm* of 0. These three hidden areas were placed in the open with multiple paths leading into them. This suggests that it is more difficult for the approach to find hidden areas in flat terrain, which was expected as there is no elevated terrain to provide cover for the generated areas. Although it is possible to evolve cover for areas via mutations, as was the case for the other 27 hidden areas evolved in this evaluation.

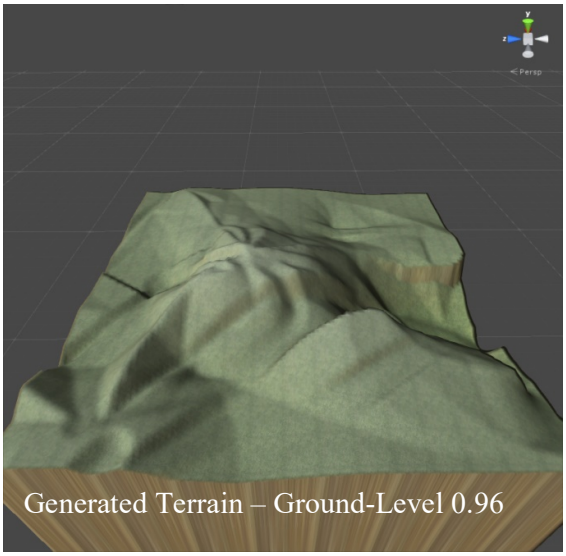
The two-sample t-tests perfomed between the groups for the vantage points in input graph 5 showed that there was a significant variation in the *asm*s between terrain 2 and terrain 5, with a p-value of 0.00003 which is less than the corrected significance level of 0.005. The values in Table 8.6 show that these two groups contained the largest difference in the mean *asm* value. It was theorised that rougher terrains would contain more areas that were suitable for vantage points and these results support this theory as the roughest terrain resulted in the highest average *asm* for vantage points and the two smoothest terrains resulted in the two lowest average *asm*s.

1.1.1 Analysis of Terrain Deformation From Using Different Initial Terrains

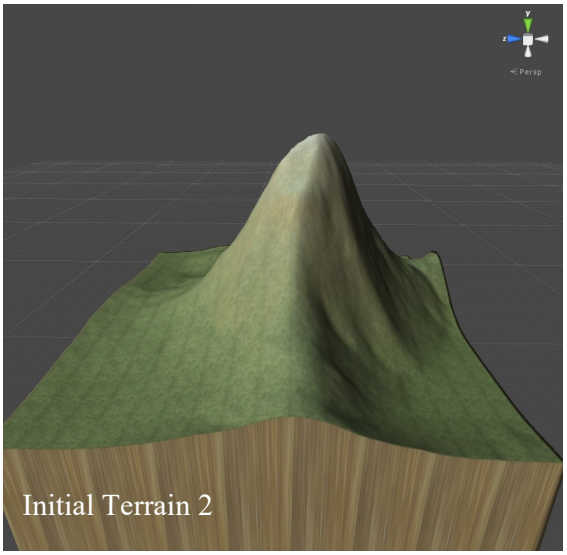
It is preferable that evolved modifications deform the *initial terrain* as little as possible while still incorporating all of the user-specified requirements. The ground-level measure, discussed in Section 6.2.3.3, is designed to reduce deformation so that the generated terrain more accurately resembles the *initial terrain*. It does this by penalising the fitness of generated terrains when areas are created at higher or lower altitudes than the initial height of the terrain. Creating areas at higher or lower altitudes creates mountains or craters that can significantly alter the landscape. Generated terrains across the 30 runs of the 25 evaluations achieved an average ground-level measure of 0.897, where a theoretical maximum of 1.0 indicates that no penalty was incurred. As the size of the terrain is 128 units, the average difference between the modified height of a generated area and the height of the *initial terrain* at the centre of the generated area is $(1 - 0.897) * 128 = 13.184$ units.

The generated terrains with high ground-level measures were analysed to determine the degree of deformation present in these generated terrains. Some of these terrains are shown in Figure 8.44, which displays the five *initial terrain*s next to the generated terrain with the highest (best) ground-level measure of all the terrains that were evolved using the *initial terrain*. These terrains show that while the generated terrains maintained their initial shape to some degree, they ultimately formed their own shape to incorporate the desired areas and constraints. Regardless of the final form of the generated terrains, it is obvious that the *initial terrain*s had a strong impact on their visual appearance. For example, the generated terrain associated with *initial terrain* 1, that contains no mountains or craters that significantly alter its relatively flat visual appearance to incorporate the user-specified goal layout, while the generated terrains associated with the more mountainous *initial terrain*s, 4 and 5, are still mountainous.

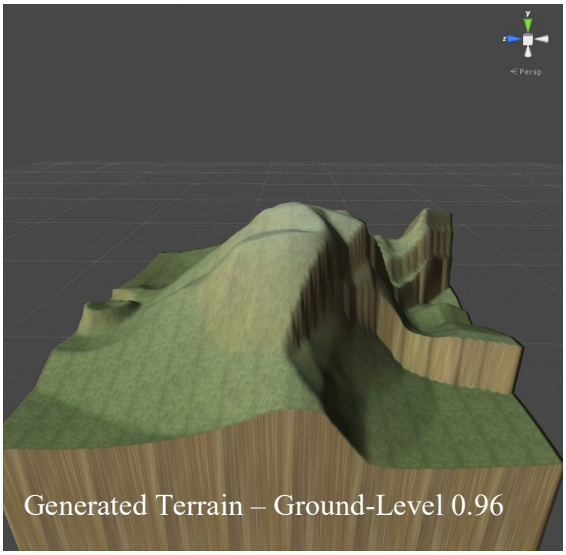
Initial Terrain 1



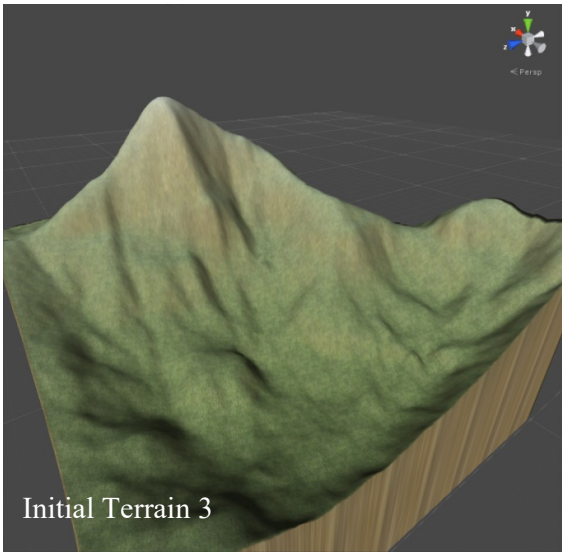
Generated Terrain – Ground-Level 0.96



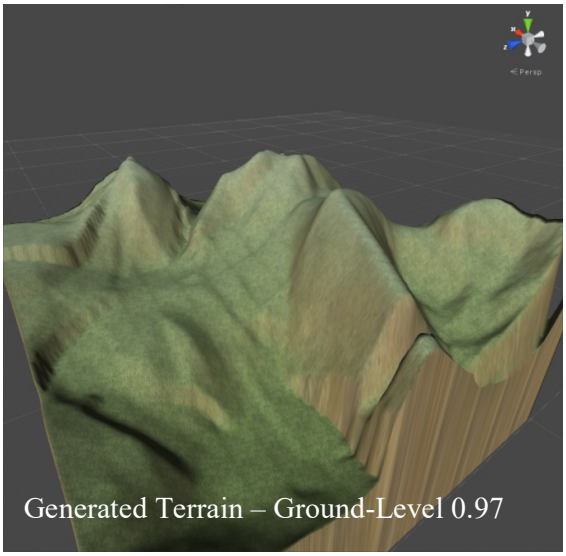
Initial Terrain 2



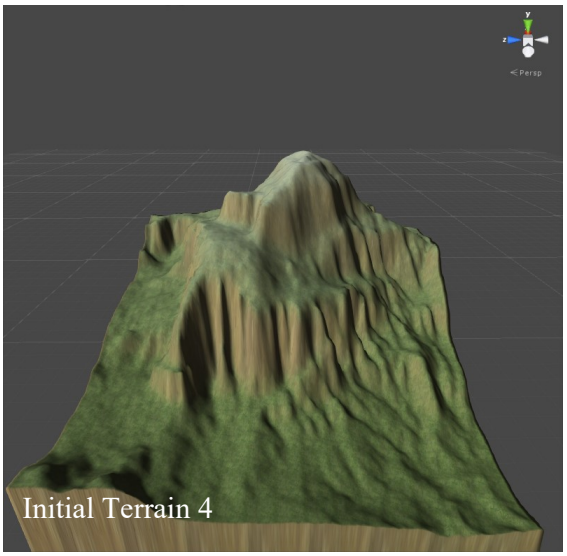
Generated Terrain – Ground-Level 0.96



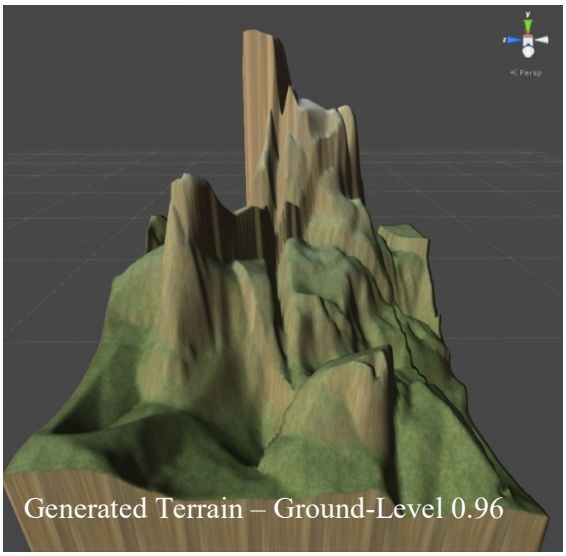
Initial Terrain 3



Generated Terrain – Ground-Level 0.97



Initial Terrain 4



Generated Terrain – Ground-Level 0.96

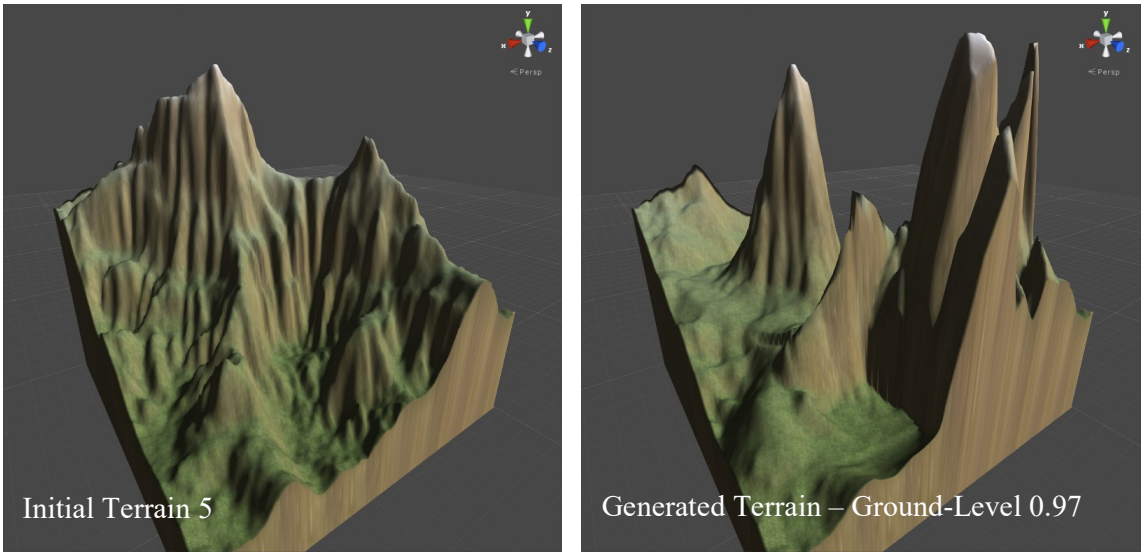


Figure 8.44. Images of the five initial terrains used in these evaluations paired with its corresponding generated terrain with the best evolved ground-level measure, where a ground-level measure of 1 indicates the least deformation to the initial terrain.

A deformation factor was devised to quantify the amount of deformation in a generated terrain. Obtaining this factor requires the heightmaps of the *initial terrain* and the generated terrain, and is calculated as the average of the absolute differences between the height of the *initial terrain* and the heights of the generated terrain at each (x, z) coordinate. The average value is divided by the maximum terrain height to normalise. Therefore a deformation factor of 0.0 indicates that the two terrains are identical and a factor of 1.0 indicates the maximum possible deformation. The average deformation factors across the 30 runs of each evaluation is listed in Table 8.7.

Table 8.7. List of average deformation factors of each evaluation.

	Terrain 1	**Terrain 2**	**Terrain 3**	**Terrain 4**	**Terrain 5**
Input 1	0.079496	0.071932	0.069708	0.094033	0.092366
Input 2	0.087703	0.08923	0.089261	0.116724	0.094939
Input 3	0.097263	0.093813	0.07971	0.10443	0.102991
Input 4	0.097672	0.112135	0.097079	0.124783	0.117366
Input 5	0.121514	0.13416	0.116149	0.131224	0.132202

These values show a noticable trend in the average deformation factors. The more rough the *initial terrain* was, the higher the deformation factor. This is contrary to what was expected as it was hypothesised that the smoothest *initial terrain* would require the most deformation to produce suitable areas. This trend could be attributed to the fact that there are more peaks

in rough terrains that need to be flattened when an area is generated near them, even if the generated area has a ground-level measure of one. Therefore if an area is created at the top of a peak, even if it is created on the peaks surface, to create a flat area it will need raise all the surrounding heights significantly. Creating a flat area on a flat terrain, even if the area is slightly offset from ground level, will typically cause less deformation. This is demonstrated in Figure 8.45, which shows a side view of a terrain, where blue circles show initial heights of the terrain and red circles show modified heights. In Figure 8.45 (a) a flat area has been created at the top of a peak at ground level. This would result in a high deformation factor because the surrounding heights have to be raised significantly to make the area flat. In Figure 8.45 (b) an area is created, slightly offset from the ground level, but because the surrounding heights are all at a similar height, the deformation factor is less.

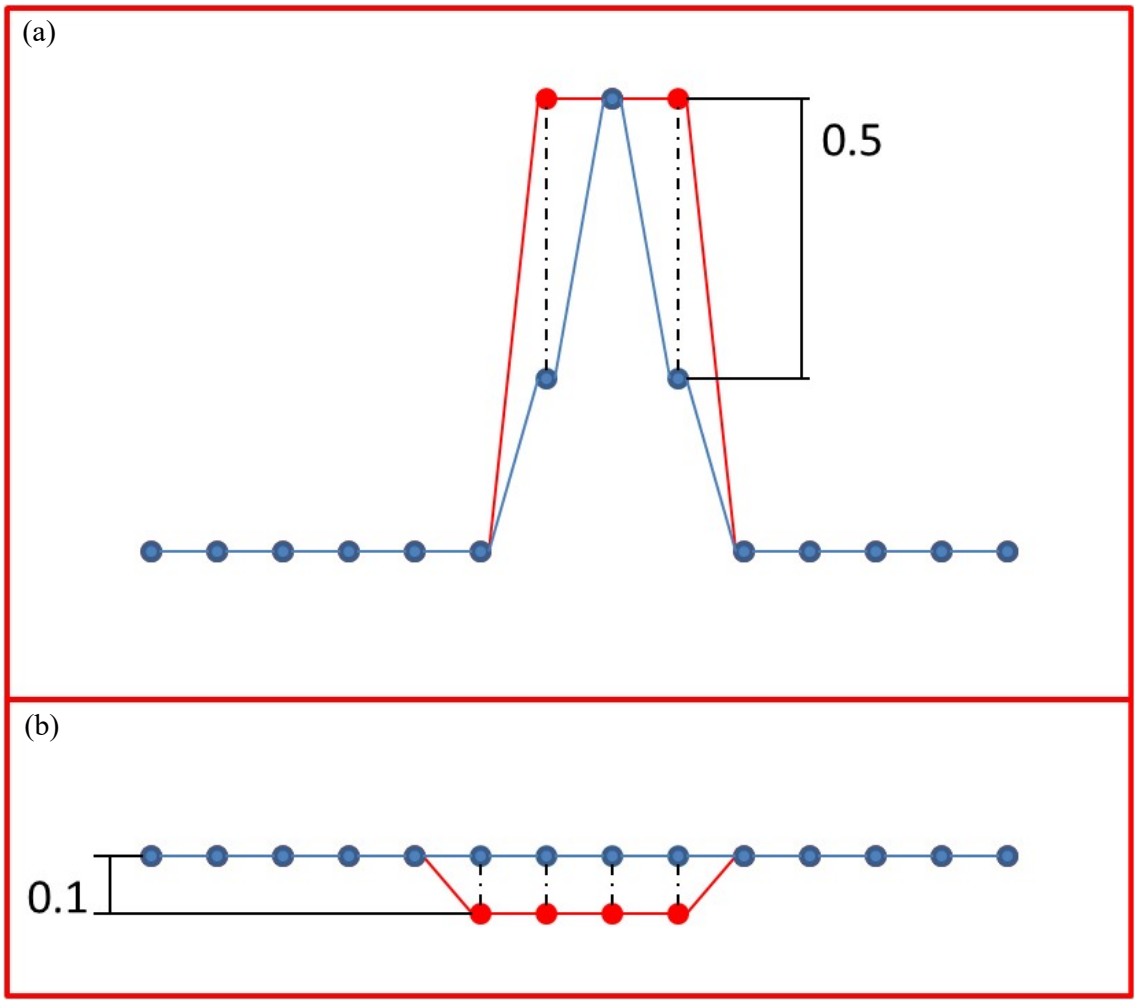


Figure 8.45. A visual example of how peaks may cause more terrain deformation despite good offset penalty scores. (a) Shows how an area generated at ground level on a peak can cause significant deformation. (b) Shows how an area generated below ground level on a flat surface causes less deformation.

This section discussed the impact that different *initial terrain*s have on the quality of generated areas, in terms of *asm* values of the areas. This was examined by running evaluations that used different *initial terrain*s and performing analysis of variance tests on the *asm* values of generated areas. These results showed that typically there is no significant difference in *asm* values of generated areas based on the chosen *initial terrain*. This section also examined the level of deformation applied to an *initial terrain* in order to incorporate the user-specified requirements. This examination revealed that modifications typically deformed the rough terrains to a higher degree than the smoother terrains, and that the chosen *initial terrain* has an obvious impact on the visual appearance of the generated terrains.

8.5 Examples of Using a Generated Terrain

This section shows three examples of using the evolved terrains generated by the approach. Figure 8.46 shows the terrain generated using the *initial terrain* shown in Figure 8.1, and the associated set of evolved modifications produced by the approach and is completed with objects, such as rocks and trees. The goal layout used in the evolution process in shown in Figure 8.29 (a). Figure 8.47 (a) and Figure 8.47 (b) show the terrain from the perspective of a player character. Figure 8.47 (a) is looking towards a path leading to a hidden area. The trees in this image help provide the cover concealing the area from view. Figure 8.47 (b) is a view from the path leading to the small hidden area where helpful items may be placed.

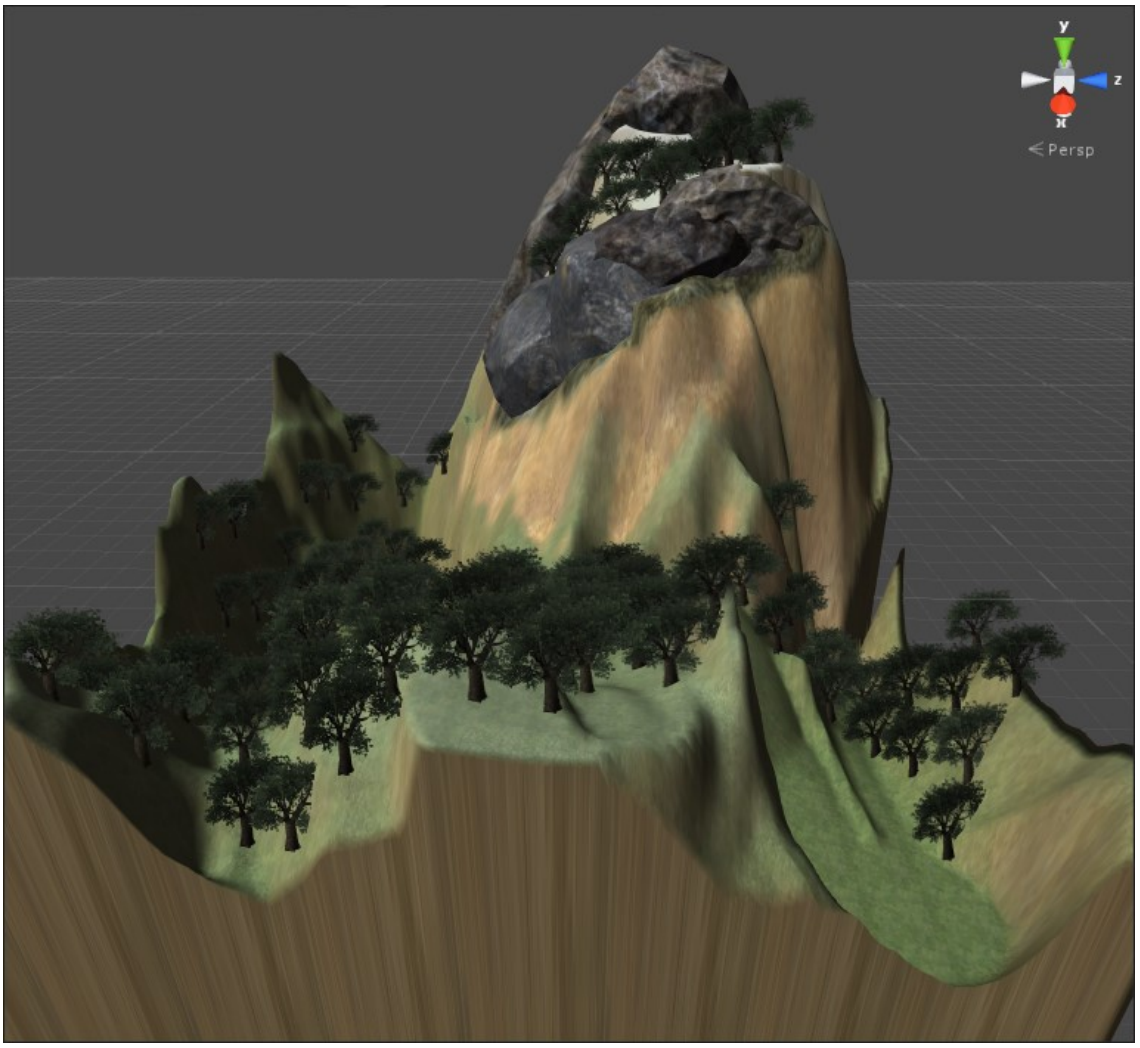


Figure 8.46. A terrain generated using the initial terrain shown in Figure 8.1, and the goal layout shown in Figure 8.29 (a), populated with objects such as trees and rocks.

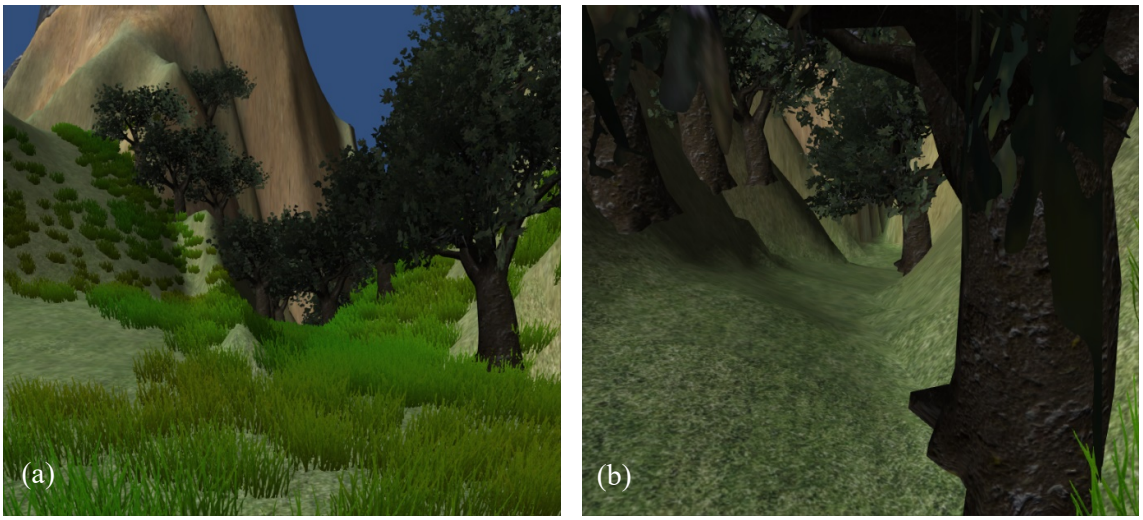


Figure 8.47. Images of the generated terrain in Figure 8.46, from the perspective of a player character. (a) shows the first person view of the entrance to a path leading to a hidden area. (b) shows the first-person view of the path leading to the hidden area.

Figure 8.48 shows another example of using a generated terrain. This terrain is generated using the *initial terrain* shown in Figure 8.43 (d), and the associated set of evolved modifications produced by the approach. The goal layout used in the evolution process is shown in Figure 8.29 (e). This terrain contains four vantage points and an open area. Figure 8.49 (a) shows the terrain from the perspective of a player character, who is positioned at one of the vantage points. This figure shows the player has good vantage over three other players (white capsules) who have been positioned in the open area. A fourth player (white capsule) has been positioned in another of the vantage points and is circled in red. Figure 8.49 (b) shows the terrain from the perspective of one of the three players in the open area, looking up at the fourth player positioned in the vantage point, again circled in red.

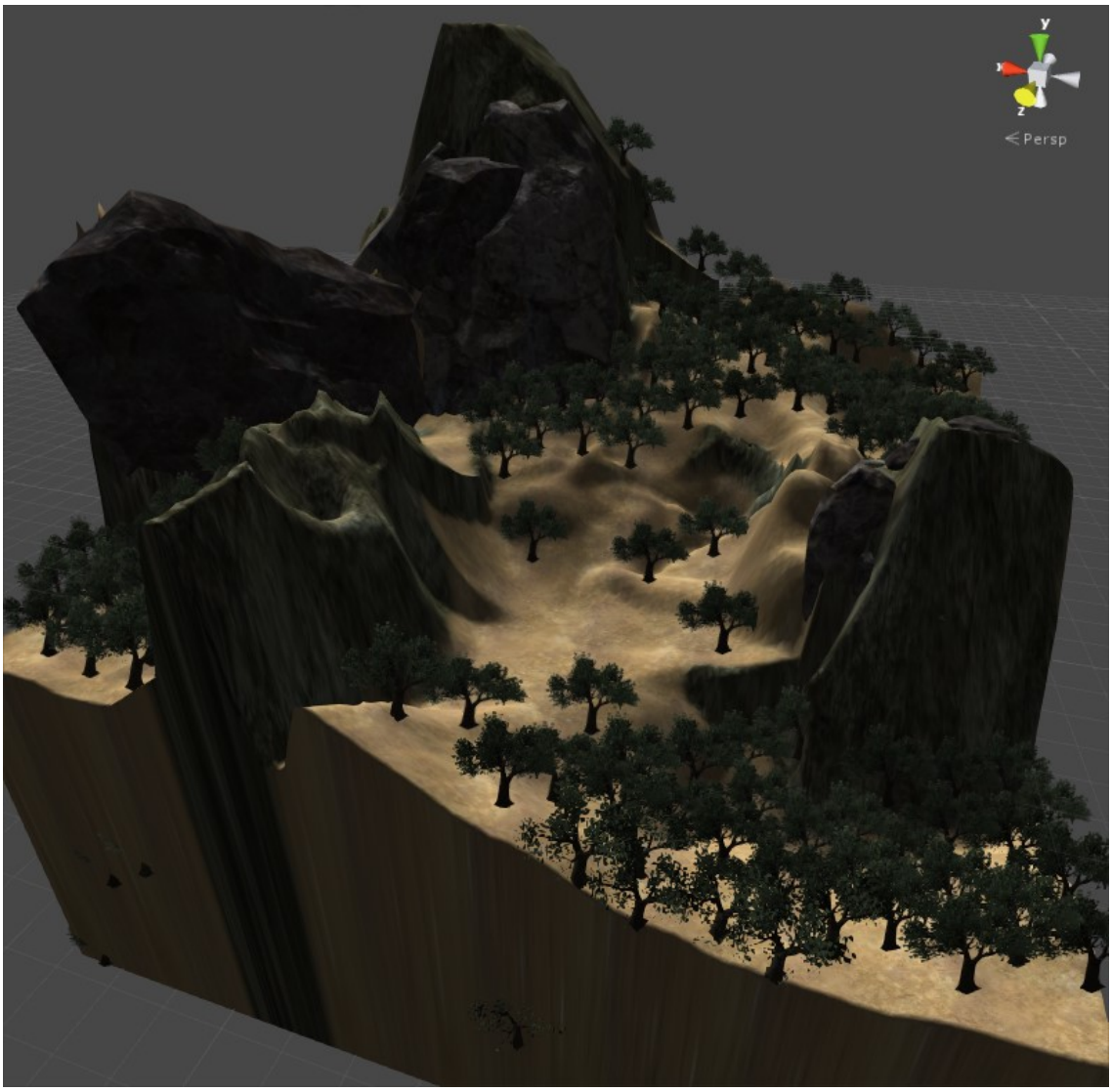


Figure 8.48. A terrain generated using the initial terrain shown in Figure 8.43 (d), and the goal layout shown in Figure 8.29 (e), populated with objects such as trees and rocks.

Figure 8.49. Images of the generated terrain in Figure 8.48, from the perspective of a player character. (a) shows the first-person view from a vantage point overlooking three other players (represented as white capsules) and another player on a vantage point circled in red. (b) shows the first-person view from one of the three players on low ground, looking up at a player positioned on a vantage point (circled in red).

In another example, the terrain shown in Figure 8.50 is generated using the *initial terrain* shown in Figure 8.43 (e) and the associated set of evolved modifications produced by the approach. The terrain has been populated with various objects to finalise the game level. The goal layout used in the evolution process is shown in Figure 8.29 (d), with four strongholds and a choke point. Figure 8.51 (a) shows the terrain from the view of a player character from one of the strongholds, facing the choke point. Figure 8.51 (b) shows the terrain from a player character's perspective from the choke point. This figure shows two additional paths (red arrows) that also lead into the choke point from the other strongholds.

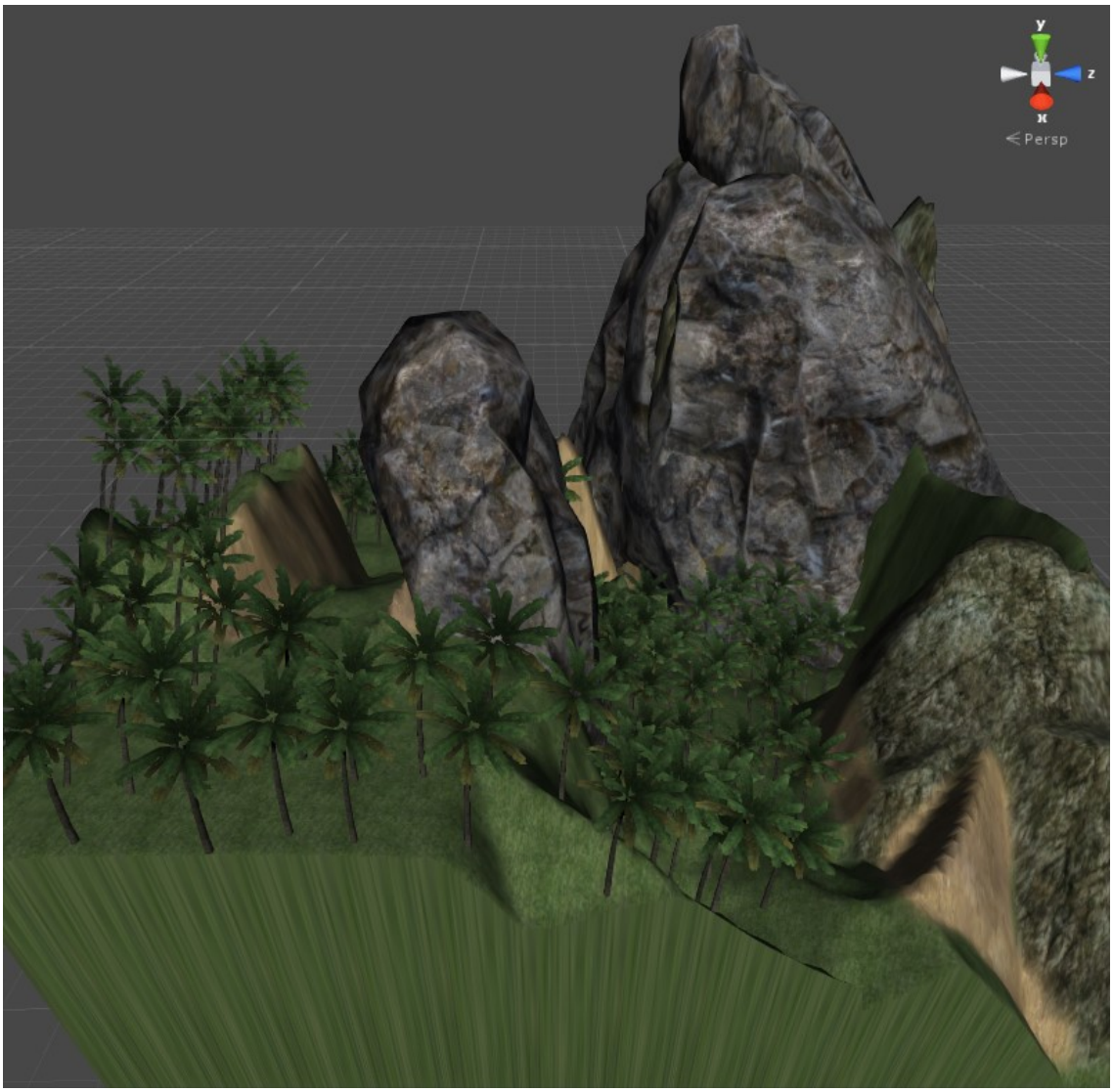


Figure 8.50. A terrain generated using the initial terrain shown in Figure 8.43 (e), and the goal layout shown in Figure 8.29 (d), populated with objects such as trees and rocks. This terrain conforms to the goal layout by incorporating four strongholds connected to each other through a choke point.

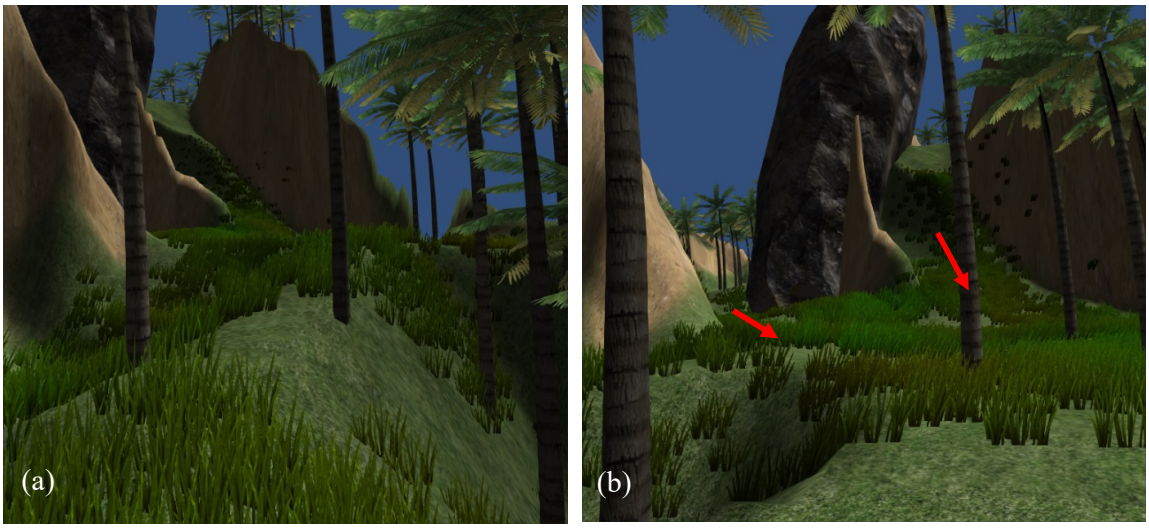


Figure 8.51. Images of the generated terrain in Figure 8.50, from the perspective of a player character. (a) shows the first-person view of the entrance to the choke point. The first-person view from the choke point is shown in (b), revealing two other paths (red arrows) that converge at the choke point's location.

8.6 Summary

This chapter explored the effectiveness of *asm* values, the ability to evolve single and multiple areas of specific types into an *initial terrain*, the ability to evolve terrains to meet a goal layout extracted from existing game levels, and the effects that the *initial terrain* has on both the generated terrain and the developed approach's ability to find suitable areas of specific types. The evaluation results show that an *asm* is good at indicating how similar an area's characteristics are to a specific area type by visually analysing generated areas with both good and bad *asm* values. The results of this analysis show that the higher the *asm* the more similar the area's physical characteristics are to the characteristics of areas of a specific area type.

This approach is able to incorporate single areas and multiple areas of specific types into an *initial terrain* and maintain constraints placed between the areas. Constraints were maintained well with distance constraint values closely matching the desired values, and traversable and line-of-sight constraints being maintained with near perfect accuracy. The *asm* values of generated areas were typically between 0.70 and 0.85, which is considered good; especially as *asm* values can theoretically be much lower than 0. One problem made apparent in the multiple area evaluations involved desired areas of the same type being assigned the same physical area in the terrain. This was not prohibited in this research as some area types share similar properties to other area types, and the approach was designed to allow different desired area types to be placed at the same physical location if no distance constraints were placed between them. Unfortunately, the lack of this restriction caused the developed

approach to favour placing two areas of the same type in the same location. Evaluations with layouts containing more constraints alleviate this problem to some degree.

The evaluation results show that this approach can generate terrains with similar layouts to specified goal layouts that were extracted from existing game levels. These goal layouts varied in complexity with the more complex layouts consisting of more desired areas and constraints. Although the approach evolved suitable terrains for each layout, more complex *input graph*s do have a negative influence on the performance of the GA-based approach.

Lastly, the *initial terrain*s used in these evaluations had impact on the visual appearance of the terrains that were generated. Aside from the visual impact, in most cases the *initial terrain* did not have a significant effect on the *asm* values of generated areas or the overall fitness of an individual.

Chapter 9. Conclusion and Future Work

This thesis presented a novel approach to generating terrains that meet desired gameplay criteria. This chapter summarises the key findings of this research in Section 9.1, followed by potential areas of future work in Section 9.2.

9.1 Conclusion

Video game content is usually generated manually by teams of developers and is one of the most time-consuming components of game development, which also makes it one of the most expensive components. PCG is the process of algorithmically generating media content and can be a useful tool in game development as it can reduce the time spent manually generating content. This thesis contributes to the field of PCG by proposing a novel approach to generating terrains for video games that incorporate desired gameplay elements.

Although a number of terrain generation techniques exist, few methods incorporate gameplay requirements and those that do tend to focus on a single gameplay element. This approach attempted to address these issues by allowing the user to specify a number of specific area types, where each area type is designed for a specific gameplay purpose, to be incorporated into an *initial terrain*, while maintaining specific constraints that can be placed between desired area types. This approach was also designed to be scalable such that new area types could easily be added by updating one of the inputs with existing examples of the new area types.

The primary research question was addressed by developing a novel approach towards generating terrains that incorporate a range of gameplay elements. This GA-based approach evolves a set of modifications that, when applied to a user-specified terrain, incorporates a desired configuration of gameplay elements, as specified by the user. The gameplay elements used in this approach include five area types; hidden area, open area, vantage point, choke point, and stronghold. Each of these five area types are designed for a specific gameplay purpose. For example, open areas can be used for staging large-scale battles, and hidden areas for hiding items and promoting exploration. Geographical distance, path distance, traversable, and line-of-sight constraints can be specified between areas to control the configuration of these areas in the generated terrain. In order to address this primary question, four sub-questions first had to be addressed.

This study addressed Sub-Question 1 by using an individual that consists of three chromosomes, the node chromosome, the edge chromosome, and the mapping chromosome, and is detailed in Section 6.2.1. The node and edge chromosomes encode a graph structure, referred to as a *terrain graph*, which represents a set of modifications that form areas and paths when applied to a user-specified terrain. The mapping chromosome encodes information that determines the gameplay purpose of a subset of the areas encoded by the *terrain graph*. For example, if the user desires an open area to be incorporated into the terrain, the mapping chromosome dictates which of the encoded areas the "open area" is. Each node in the *terrain graph* represents a set of modifications that, when applied to a user-specified terrain, form an area that may be suitable for a specific gameplay purpose. Each edge in the *terrain graph* represents a set of modifications that controls the ruggedness of the terrain between areas, forming traversable paths between areas, or making the terrain impassable. Details of a *terrain graph* and the method used to apply these modifications are presented in Chapter 3 and addresses Sub-Question 4.

Sub-Question 2 was addressed by developing a fitness evaluation method, detailed in Section 6.2.3, for the GA-based approach that consists of a weighted aggregation of three components; area evaluation, constraint evaluation, and node evaluation. Area evaluation involves calculating the average *asm* value of the gameplay areas that are specified by the user. An *asm* determines how closely the attributes of an area in the terrain match the attributes of known areas of a specific type. For example, if the attributes of an area in the terrain are similar to the attributes of known hidden areas, then the area has a high *asm* for the type "hidden area". This calculation requires a collection of areas of a specific area type to which the attributes of an area in the terrain can be compared. This study obtained such a dataset, referred to as an *ADS*, by manually identifying areas of specific types in existing game levels.

Developing a set of measures for Sub-Question 3 was required to quantify areas in a terrain so that the suitability of an area for specified gameplay purpose could be measured. This sub-question was addressed by using the attributes, detailed in Chapter 4, to characterise areas in a terrain. These attributes consist of an *Area Radius* measure, a collection of graph-connectivity measures, and a collection of isovist measures. The *Area Radius* measure indicates the amount of traversable terrain that forms an area in the terrain. The graph measures capture information about the location of an area relative to other areas in the

terrain, and lastly the isovist measures capture information about the visible space surrounding an area. Development of these measures led to an important contribution of this research; a novel method of estimating the volume of an isovist that is efficient and more accurate than existing methods. This novel isovist volume estimation method is detailed in Section 4.6.

9.2 Future Work

During this research certain limitations of the developed approach were identified. These limitations are listed below as well as possible directions for future research in this field.

- The gameplay elements used in this research were limited to those that are geographically restricted to a localised area in the terrain, such as hidden areas for hiding items or vantage points to give long-range attackers an advantage over targets on low ground. These areas are characterised using 16 isovist and graph measures. New area types that adhere to this restriction can easily be included in the developed approach by adding measures of these new area types to area type data set (*ADS*). Other types of gameplay elements that are not restricted to a localised area in the terrain, such as a flanking route as described in Hullet & Whitehead (2010), cannot be characterised as a single set of these 16 measures. Adding the capability to characterise these different types of gameplay elements is an area of possible further research.
- The approach developed in this work was targeted at producing terrain. The approach also however lends itself to indoor environments or gameplay that contains a mixture of outdoor/indoor play. Both the graph-based and isovist measures are particularly suited to building layouts, and are already employed in the field of urban design for those purposes. Future work could include using this approach along a similar line as in this thesis, by starting with an initial building layout and modifying the placement of walls, doors, and windows to achieve particular gameplay elements.
- The developed approach did not focus on the aesthetic quality of generated terrains as this is already a well-researched area. Therefore, gameplay elements are incorporated into an *initial terrain* that is specified by the user based on the aesthetic qualities of the specific terrain. A ground-level measure was incorporated into the fitness function of the approach to minimise the deformation of the *initial terrain* when incorporating the user-specified gameplay elements. It did this by penalising the fitness of

individuals when areas were generated at heights that differed from the ground level of the *initial terrain*, as greater height differences can form mountains and craters that significantly alter the appearance of the *initial terrain*. Future research could address this issue by incorporating aesthetic-based requirements into the representation, or by adding aesthetics measures to the fitness function. Some aesthetic considerations that could be measured from a set of example levels include colour balance, lighting, and types of textures used.

- The developed approach uses a *layout graph* to capture the connectivity of areas in a generated terrain. Graph measures used to characterise gameplay elements are calculated from this *layout graph* and contribute to the similarity measure that determines how suitable an area is for a specific gameplay purpose. Besides the *layout graph*, the structure of the initial input terrain also influences accessibility between areas but this is not captured in the measures. An area of future work will be to investigate similarity measures that are more representative of the final terrain. This can be approached in a number of ways, by either modifying terrain to remove accessibility not specified in the *layout graph*, or by incorporating the additional accessibility that is inherent in the terrain into the similarity measures.
- As this is a GA-based approach, there are several avenues for future work regarding its GA component, such as using adaptive mutation. Adaptive mutation starts with a larger mutation probability to avoid prematurely converging and reduces the mutation probability as a GA runs to help it converge in later generations. This could be beneficial to the developed approach as sometimes a small mutation to an individual may result in oscillation of the average fitness value for the population. The application of adaptive mutation will remove the need to make non-trivial decisions beforehand as to what are appropriate, avoid being trapped in a local optima, and may be used to speed up convergence.
- Multi-objective evolutionary algorithms (Coello, 1999) could also be useful for the developed approach, as the developed fitness function consists of three evaluation components that are currently combined using weighted aggregation into one fitness function. In a problem involving multiple, possibly competing, objectives (which is the case in point), a multi-objective approach would find a set of solutions that define the best trade-off between the competing objectives.

- Memetic algorithms (Moscato, 1989) are another avenue for future research. These algorithms are similar to GAs but include a local search as part of the evolutionary process to prevent the algorithm resulting in premature convergence. The benefits of incorporating a local search helps the memetic algorithm to find optimal solutions more quickly in scenarios where using just the evolution process alone would take too long, as well as being able to find solutions which are otherwise unreachable by just using an evolution strategy of a local search method on its own.
- The developed approach was verified by using it to replicate existing game level layouts and comparing the terrains that it generates to these goal layouts. This approach could be incorporated into a tool for game developers to aid them in creating game-ready terrains. Once a tool has been developed, systematic user studies can be conducted to validate its usefulness to game developers.

References

Andereck, M. (2014). *Procedural Terrain Generation Based on Constraint Paths.* (Master of Science), The Ohio State University,

Antoniuk, I., & Rokita, P. (2015). *Procedural Generation of Adjustable Terrain for Application in Computer Games Using 2D Maps*, Cham.

Ashlock, D., Lee, C., & McGuinness, C. (2011). Search-Based Procedural Generation of Maze-Like Levels. *IEEE Transactions on Computational Intelligence and AI in Games, 3*(3), 260-273. doi:10.1109/TCIAIG.2011.2138707

Bayes, T. (1763). An essay towards solving a problem in the doctrine of chances. *Phil Trans R Soc, 53*, 370-418.

Belhadj, F., & Audibert, P. (2005). *Modeling landscapes with ridges and rivers: bottom up approach*. Paper presented at the Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia, Dunedin, New Zealand.

Benedikt, M. (1979). To Take Hold of Space: Isovists and Isovist Fields. *Environment and Planning B: Planning and Design, 6*(1), 47-65. doi:10.1068/b060047

Bethesda Game Studios. (2011). The Elder Scrolls V: Skyrim. Rockville, Maryland, USA: Bethesda Softworks.

Blizzard North. (1998). Diablo: Blizzard Entertainment. Ubi Soft Entertainment, Electronic Arts.

Bundysoft. (2018). L3DT.

Coello, C.A. (1999). A Comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and Information Systems*. An International Journal, vol. 1, no. 3, pp. 269–308, Aug. 1999.

Cordonnier, G., Braun, J., Cani, M., Benes, B., Galin, E., Peytavie, A., & Guérin, E. (2016). Large Scale Terrain Generation from Tectonic Uplift and Fluvial Erosion. *Computer Graphics Forum, 35*(2), 165-175. doi:doi:10.1111/cgf.12820

Cordonnier, G., Cani, M., Benes, B., Braun, J., & Galin, E. (2018). Sculpting Mountains: Interactive Terrain Modeling Based on Subsurface Geology. *IEEE Transactions on Visualization and Computer Graphics, 24*(5), 1756-1769. doi:10.1109/TVCG.2017.2689022

Cordonnier, G., Galin, E., Gain, J., Benes, B., Guérin, E., Peytavie, A., & Cani, M. (2017). Authoring landscapes by combining ecosystem and terrain erosion simulation. *ACM Trans. Graph., 36*(4), 1-12. doi:10.1145/3072959.3073667

Cristea, A., & Liarokapis, F. (2015, 16-18 Sept. 2015). *Fractal Nature - Generating Realistic Terrains for Games.* Paper presented at the 2015 7th International Conference on Games and Virtual Worlds for Serious Applications (VS-Games).

Dalton, N., Marshall, P., Dalton, R., Peverett, I., & Clinch, S. (2015). *Thee Dimensional Isovists for the Study of Public Displays*. Paper presented at the 10th Space Syntax Symposium, London.

Dalton, R., & Dalton, N. (2001). *OmniVista: An Application for Isovist Field and Path Analysis*. Paper presented at the 3rd International Space Syntax Symposium, Atlanta, Georgia, USA.

Davis, L. (1991). Bit-climbing, representational bias, and test suit design. *Proc. Intl. Conf. Genetic Algorithm, 1991*, 18-23.

De Carli, D., Pozzer, C., Bevilacqua, F., & Schetinger, V. (2014, 26-30 Aug. 2014). *Procedural Generation of 3D Canyons.* Paper presented at the 2014 27th SIBGRAPI Conference on Graphics, Patterns and Images.

De Jong, K. (1975). Analysis of the behavior of a class of genetic adaptive systems. *Technical Report No. 185, Department of Computer and Communication Sciences, University of Michigan*.

Doran, J., & Parberry, I. (2010). Controlled Procedural Terrain Generation Using Software Agents. *IEEE Transactions on Computational Intelligence and AI in Games, 2*(2), 111-119. doi:10.1109/TCIAIG.2010.2049020

Dormans, J. (2010). *Adventures in level design: generating missions and spaces for action adventure games*. Paper presented at the Proceedings of the 2010 Workshop on Procedural Content Generation in Games, Monterey, California.

Dormans, J. (2011). *Level design as model transformation: a strategy for automated content generation*. Paper presented at the Proceedings of the 2nd International Workshop on Procedural Content Generation in Games, Bordeaux, France.

Eiben, A., Hinterding, R., & Michalewicz, Z. (1999). Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation, 3*(2), 124-141. doi:10.1109/4235.771166

Emilien, A., Bernhardt, A., Peytavie, A., Cani, M., & Galin, E. (2012). Procedural generation of villages on arbitrary terrains. *The Visual Computer, 28*(6), 809-818. doi:10.1007/s00371-012-0699-7

Epic Games. (2012). Unreal Engine 3 / UDK. Retrieved from https://api.unrealengine.com/udk/Three/GameplayHome.html

Fisher-Gewirtzman, D., & Wagner, I. (2003). Spatial Openness as a Practical Metric for Evaluating Built-up Environments. *Environment and Planning B: Planning and Design, 30*(1), 37-49. doi:10.1068/b12861

Fister Jr., I., Perc, M., Ljubič, K., Kamal, M., Iglesias, A., & Fister, I. (2015). Particle Swarm Optimization for Automatic Creation of Graphic Characters. *In Chaos, Solitons & Fractals*, (Vol. 73, pp. 29-35).

Frade, M., de Vega, F., & Cotta, C. (2009). Breeding Terrains with Genetic Terrain Programming: The Evolution of Terrain Generators. *International Journal of Computer Games Technology, 2009*, 13. doi:10.1155/2009/125714

Frade, M., de Vega, F., & Cotta, C. (2010). *Evolution of Artificial Terrains for Video Games Based on Accessibility*, Berlin, Heidelberg.

Frade, M., de Vega, F., & Cotta, C. (2010, 18-23 July 2010). *Evolution of artificial terrains for video games based on obstacles edge length.* Paper presented at the IEEE Congress on Evolutionary Computation.

Fraser, G., & Arcuri, A. (2011). Evolutionary generation of whole test suites. *2011 11th International Conference on Quality Software (QSIC)*, 31-40.

Gain, J., Marais, P., & Straßer, W. (2009). *Terrain sketching*. Paper presented at the Proceedings of the 2009 symposium on Interactive 3D graphics and games, Boston, Massachusetts.

Gain, J., Merry, B., & Marais, P. (2015). Parallel, Realistic and Controllable Terrain Synthesis. *Computer Graphics Forum, 34*(2), 105-116. doi:doi:10.1111/cgf.12545

Galin, E., Peytavie, A., Maréchal, N., & Guérin, E. (2010). Procedural Generation of Roads. *Computer Graphics Forum, 29*(2), 429-438. doi:doi:10.1111/j.1467-8659.2009.01612.x

Génevaux, J., Galin, E., Peytavie, A., Guérin, E., Briquet, C., Grosbellet, F., & Benes, B. (2015). Terrain Modelling from Feature Primitives. *Computer Graphics Forum, 34*(6), 198-210. doi:doi:10.1111/cgf.12530

Glinton, R., Owens, S., Giampapa, J., Sycara, K., Grindle, C., & Lewis, M. (2004, July). *Terrain-Based Information Fusion and Inference*. In Proceedings of the Seventh International Conference on Information Fusion (pp. 338-345).

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. New York: Addison-Wesley.

Goldberg, D., & Deb, K. (1991). A Comparative Analysis of Selection Schemes Used in Genetic Algorithms. In G. J. E. Rawlins (Ed.), *Foundations of Genetic Algorithms* (Vol. 1, pp. 69-93): Elsevier.

Golubev, K., Zagarskikh, A., & Karsakov, A. (2016). Dijkstra-based Terrain Generation Using Advanced Weight Functions. *Procedia Computer Science, 101*, 152-160. doi:https://doi.org/10.1016/j.procs.2016.11.019

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., . . . Bengio, Y. (2014). *Generative adversarial nets*. Paper presented at the Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2, Montreal, Canada.

García-Martínez, C., & Lozano, M. (2008). Local Search Based on Genetic Algorithms. In P. Siarry & Z. Michalewicz (Eds.), *Advances in Metaheuristics for Hard Optimization* (pp. 199-221). Berlin, Heidelberg: Springer Berlin Heidelberg.

Grefenstette, J. (1986). Optimization of Control Parameters for Genetic Algorithms. *IEEE Transactions on Systems, Man, and Cybernetics, 16*(1), 122-128.

Guérin, E., Digne, J., Galin, E., Peytavie, A., Wolf, C., Benes, B., & Martinez, B. (2017). Interactive example-based terrain authoring with conditional generative adversarial networks. *ACM Trans. Graph., 36*(6), 1-13. doi:10.1145/3130800.3130804

Hart, P., Nilsson, N., & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics, 4*(2), 100-107. doi:10.1109/TSSC.1968.300136

Hartsook, K., Zook, A., Das, S., & O. Riedl, M. (2011, Aug. 31 2011-Sept. 3 2011). *Toward supporting stories with procedurally generated game worlds.* Paper presented at the 2011 IEEE Conference on Computational Intelligence and Games (CIG'11).

Hinterding, R. (1995, Nov. 29 1995-Dec. 1 1995). *Gaussian mutation and self-adaption for numeric genetic algorithms.* Paper presented at the Proceedings of 1995 IEEE International Conference on Evolutionary Computation.

Hnaidi, H., Guérin, E., Akkouche, S., Peytavie, A., & Galin, E. (2010). Feature based terrain generation using diffusion equation. *Computer Graphics Forum, 29*(7), 2179-2186. doi:doi:10.1111/j.1467-8659.2010.01806.x

Holland, J. (1992). Genetic Algorithms. *Scientific American, 267*(1), 66-73.

Hullett, K., & Whitehead, J. (2010). *Design patterns in FPS levels*. Paper presented at the Proceedings of the Fifth International Conference on the Foundations of Digital Games, Monterey, California.

IGN Entertainment. (2018). Spacelings Wiki Guide. Retrieved from http://au.ign.com/wikis/spacelings/Level_and_Gameplay_Elements

Kamal, R., & Uddin, Y. (2007). *Parametrically controlled terrain generation*. Paper presented at the Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia, Perth, Australia.

Koeneke, R., & Todd, J. (1983). Moria.

Li, Q., Wang, G., Zhou, F., Tang, X., & Yang, K. (2006). *Example-Based Realistic Terrain Generation*, Berlin, Heidelberg.

Lopes, R., Tutenel, T., Smelik, R., Kraker, K., & Bidarra, R. (2010). *A constrained growth method for procedural floor plan generation.* Paper presented at the Proceedings of GAME-ON 2010.

Mendenhall, W., & Sincich, T. (2015). *Statistics for Engineering and the Sciences, Sixth Edition*. New York: Chapman and Hall/CRC.

Metropolis, N., & Ulam, S. (1949). The Monte Carlo Method. *Journal of the American Statistical Association, 44*(247), 335-341. doi:10.1080/01621459.1949.10483310

Michel, E., Emilien, A., & Cani, M. (2015, 2015-05-04). *Generation of Folded Terrains from Simple Vector Maps.* Paper presented at the Eurographics 2015 short paper proceedings, Zurich, Switzerland.

Mitchell, M. (1996). *Introduction to Genetic Algorithms*.

Moscato, P., 1989, On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms, *Caltech Concurrent Computation Program*, C3P Report 826.

Newerth.com. (2006-2018). Newerth. Retrieved from http://www.newerth.com/

Newman, M. (2010). *Networks: An Introduction*: Oxford University Press, Inc.

Olsen, J. (2004). *Realtime Procedural Terrain Generation - Realtime Synthesis of Eroded Fractal Terrain for use in Computer Games.* University of Southern Denmark.

Ong, T., Saunders, R., Keyser, J., & Leggett, J. (2005). *Terrain generation using genetic algorithms*. Paper presented at the Proceedings of the 7th annual conference on Genetic and evolutionary computation, Washington DC, USA.

Parish, Y., & Müller, P. (2001). *Procedural modeling of cities*. Paper presented at the Proceedings of the 28th annual conference on Computer graphics and interactive techniques.

Pearson, K. (1895). Note on Regression and Inheritance in the Case of Two Parents. *Proceedings of the Royal Society of London, 58*, 240-242.

Pech, A., Hingston, P., & Masek, M., Lam, C.P. (2016a). *Identifying Attributes for Characterizing Game Area Types in Virtual Terrain*. Paper presented at the 1st Joint Interest Conference of DiGRA and FDG, Dundee, Scotland.

Pech, A., Lam, C.P., Hingston, P., & Masek, M. (2016b). *Using Isovists to Evolve Terrains with Gameplay Elements*. Paper presented at the Evostar 2016 conference, Porto, Portugal.

Perkins, L. (2010). *Terrain analysis in real-time strategy games: an integrated approach to choke point detection and region decomposition*. Paper presented as the 6th AAAI conference, Stanford, California, USA.

Perlin, K. (1985). Image Synthesizer. In *Computer Graphics (ACM)* (Vol. 19, pp. 287-296).

Peytavie, A., Galin, E., Grosjean, J., & Merillou, S. (2009). Arches: a Framework for Modeling Complex Terrains. *Computer Graphics Forum, 28*(2), 457-467. doi:doi:10.1111/j.1467-8659.2009.01385.x

Pyysalo, U., Oksanen, J., & Sarjakoski, T. (2009). *Viewshed Analysis and Visualization of Landscape Voxel Models*. Paper presented at the 24th International Cartographic Conference, Santiago, Chile.

Quinlan, R. (1992). *C4.5: Programs for Machine Learning*. San Mateo, California, USA: Morgan Kaufmann.

Raffe, W., Zambetta, F., & Li, X. (2011). *Evolving patch-based terrains for use in video games*. Paper presented at the Proceedings of the 13th annual conference on Genetic and evolutionary computation, Dublin, Ireland.

Renders, J., & Bersini, H. (1994). *Hybridizing genetic algorithms with hill-climbing methods for global optimization: two possible ways.* Paper presented at the Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence.

Rodriguez, A., Ehlenberger, D., Hof, P., & Wearne, S. (2006). Rayburst sampling, an algorithm for automated three-dimensional shape analysis from laser scanning microscopy images. *Nature Protocols, 1*, 2152. doi:10.1038/nprot.2006.313

Rumelhart, D., Hinton, G., & Williams, R. (1986). Learning representations by back-propagating errors. *Nature, 323*, 533. doi:10.1038/323533a0

Rusnell, B., Mould, D., & Eramian, M. (2009). Feature-rich distance-based terrain synthesis. *The Visual Computer, 25*(5), 573-579. doi:10.1007/s00371-009-0332-6

S2 Games. (2003). Savage: The Battle for Newerth.

Safe, M., Carballido, J., Ponzoni, I., & Brignole, N. (2004). *On Stopping Criteria for Genetic Algorithms*, Berlin, Heidelberg.

Smelik, R., Galka, K., Kraker, K., Kuijper, F., & Bidarra, R. (2011). *Semantic constraints for procedural generation of virtual worlds*. Paper presented at the Proceedings of the 2nd International Workshop on Procedural Content Generation in Games, Bordeaux, France.

Smelik, R., Tutenel, T., Kraker, K., & Bidarra, R. (2010). Declarative terrain modeling for military training games. *Int. J. Comput. Games Technol., 2010*, 1-11. doi:10.1155/2010/360458

Specht, D. (1990). Probabilistic neural networks. *Neural Networks, 3*(1), 109-118. doi:https://doi.org/10.1016/0893-6080(90)90049-Q

Szoka, E. (2004). EarthSculptor.

Taylor, J., & Parberry, I. (2011). *Randomness + Structure = Clutter: A Procedural Object Placement Generator*, Berlin, Heidelberg.

The NetHack DevTeam. (1987). NetHack.

Togelius, J., Preuss, M., & Yannakakis, G. (2010). *Towards multiobjective procedural map generation*. Paper presented at the Proceedings of the 2010 Workshop on Procedural Content Generation in Games, Monterey, California.

Gradl, T. (2008). Artifex Terra 3D.

Toy, M., & Wichman, G. (1980). Rogue: Epyx.

Tutenel, T., Bidarra, R., Smelik, R., & Kraker, J. (2009). Rule-based layout solving and its application to procedural interior generation. doi:urn:NBN:nl:ui:24-uuid:1322562f-7363-4891-bc1c-25ab8debaa91

Unity. (2018). San Francisco: Unity Technologies SF.

Valentine, F. (1964). *Convex Sets*. New York.

van Bilsen, A. (2008). *Mathematical explorations in urban and regional design.* Retrieved from http://resolver.tudelft.nl/uuid:7d347e20-d14d-43f1-8283-b1e8e06979c6

van Bilson, A., & Poelman, R. (2009). *3D Visibility Analysis In Virtual Worlds: The Case of Supervisor*. Paper presented at the 9th International Conference on Construction Applications of Virtual Reality, Sydney, NSW, Australia.

van der Linden, R., Lopes, R., & Bidarra, R. (2013). *Designing procedurally generated levels.* Paper presented at the 2013 AIIDE Workshop (WS-13-20).

Yang, P., Putra, S., & Li, W. (2007). Viewsphere: A GIS-Based 3D Visibility Analysis for Urban Design Evaluation. *Environment and Planning B: Planning and Design, 34*(6), 971-992. doi:10.1068/b32142

Zhang, H., Qu, D., Hou, Y., Gao, F., & Huang, F. (2016). Synthetic Modeling Method for Large Scale Terrain Based on Hydrology. *IEEE Access, 4*, 6238-6249. doi:10.1109/ACCESS.2016.2612700

Zhou, H., Sun, J., Turk, G., & Rehg, J. (2007). Terrain Synthesis from Digital Elevation Models. *IEEE Transactions on Visualization and Computer Graphics, 13*(4), 834-848. doi:10.1109/TVCG.2007.1027

Appendices

Appendix A

Chapter 3 related appendices.

Appendix A.1

Pseudocode for the generation of a *height summation map*.

```
FUNCTION GenerateHeightSummationMap(initialTerrain, terrainGraph)
    heightSummationMap = EmptyMap()
    InitialiseMap(heightSummationMap)
    FOR EACH node IN terrainGraph DO
        FOR EACH point IN heightSummationMap DO
            bhn = CalculateBaseNodeHeight(node, point, initialTerrain)
            bn = CalculateNodeBlendFactor(node, point)
            hn = bhn * bn
            previousHeight = heightSummationMap.GetValueAt(point)
            heightSummationMap.GetValueAt(point) = previousHeight + hn
        END FOR
    END FOR
    FOR EACH edge IN terrainGraph DO
        FOR EACH point IN heightSummationMap DO
            bhe = CalculateBaseEdgeHeight(edge, point, initialTerrain)
            rhe = CalculateEdgeRoughness(edge, point)
            be = CalculateEdgeBlendFactor(edge, point)
            he = (bhe + rhe) * be
            previousHeight = heightSummationMap.GetValueAt(point)
            heightSummationMap.GetValueAt(point) = previousHeight + he
        END FOR
    END FOR
    RETURN heightSummationMap
END FUNCTION
```

Appendix A.2

Pseudocode for getting the nearest point on a quadratic Bezier curve.

```
FUNCTION GetNearestPointOnQuadraticBezierCurve(p, cp0, cp1, cp2)
        A = cp1 - cp0
        B = cp2 - cp1 - A
        M = cp0 - p

        a = dot(B, B)
        b = 3 * dot(A, B)
        c = 2 * dot(A, A) + dot(M, B)
        d = dot(M, A)

        solution = ThirdDegreeEquation(a, b, c, d)// See Appendix A.4

        d0 = length(cp0 - p)
        d2 = length(cp2 - p)

        IF dot(solution, solution) > 0 THEN
                t = solution.x
                IF t >= 0 AND t <= 1 THEN
                        // See Appendix A.3 for function GetPointOnQuadraticBezierCurve
                        pos = GetPointOnQuadraticBezierCurve(t, cp0, cp1, cp2)
                        dist = length (p - pos)
                        IF dist < distMin THEN
                                tMin = t
                                distMin = dist
                                posMin = pos
                        END IF
                END IF
                t = solution.y
                IF t >= 0 AND t <= 1 THEN
                        pos = GetPointOnQuadraticBezierCurve(t, cp0, cp1, cp2)
                        dist = length (p - pos)
                        IF dist < distMin THEN
                                tMin = t
                                distMin = dist
                                posMin = pos
                        END IF
                END IF
                t = solution.z
                IF t >= 0 AND t <= 1 THEN
                        pos = GetPointOnQuadraticBezierCurve(t, cp0, cp1, cp2)
```

```
                    dist = length (p - pos)
                    IF dist < distMin THEN
                        tMin = t
                        distMin = dist
                        posMin = pos
                    END IF
                END IF

                IF tMin != 0 AND distMin < d0 AND distMin < d2 THEN
                    RETURN posMin AND tMin
                END IF
            END IF

            IF d0 < d2 THEN
                distMin = d0
                tMin = 0
                posMin = cp0
            ELSE
                distMin = d2
                tMin = 1
                posMin = cp2
            END IF

            RETURN posMin AND tMin
END FUNCTION
```

Appendix A.3

Pseudocode for getting a point along a quadratic Bezier curve.

```
FUNCTION GetPointOnQuadraticBezierCurve(t, cp0, cp1, cp2)
        a = (1 - t) * (1 - t)
        b = 2 * t * (1 – t)
        c = t * t
        point1 = (0, 0, 0)
        point1.x = a * cp0.x + b * cp1.x + c * cp2.x
        point1.y = a * cp0.y + b * cp1.y + c * cp2.y
        RETURN point1
END FUNCTION
```

Appendix A.4

Pseudocode for solving the third-degree equation required for getting the nearest point on a quadratic Bezier curve.

```
FUNCTION ThirdDegreeEquation(a, b, c, d)
    zeroMax = 0.0000001
    solution = (0, 0, 0, 0)

    IF abs(a) > zeroMax THEN
        z = a;
        a2 = b / z;
        b2 = c / z;
        c2 = d / z;
        p = b2 - a2 * a2 / 3
        q = a2 * (2 * a2 * a2 - 9 * b2) / 27 + c2
        p3 = p * p * p
        D = q * q + 4 * p3 / 27
        offset = -a2 / 3
        IF D > zeroMax THEN
            z = sqrt(D)
            u = (-q + z) / 2
            v = (-q - z) / 2
            IF u >= 0 THEN
                u = pow(u, 1 / 3)
            ELSE
                u = -pow(-u, 1 / 3)
            END IF
            IF v >= 0 THEN
                v = pow(v, 1 / 3)
            ELSE
                -pow(-v, 1 / 3)
            END IF
            solution.x = u + v + offset
            solution.w = 1
            RETURN solution
        ELSE IF D < -zeroMax THEN
            u = 2 * sqrt(-p / 3)
            v = acos(-sqrt(-27 / p3) * q / 2) / 3
            solution.x = u * cos(v) + offset
            solution.y = u * cos(v + 2 * π / 3) + offset
            solution.z = u * cos(v + 4 * π / 3) + offset
            solution.w = 3
            RETURN solution
```

```
            ELSE
                IF q < 0 THEN
                    u = pow(-q / 2, 1 / 3)
                ELSE
                    u = -pow(q / 2, 1 / 3)
                END IF
                solution.x = 2 * u + offset
                solution.y = -u + offset
                solution.w = 2
                RETURN solution
            END IF
        ELSE
            a2 = b
            b2 = c
            c2 = d
            IF abs(a2) <= zeroMax THEN
                IF abs(b2) <= zeroMax THEN
                    RETURN (0, 0, 0, 0)
                ELSE
                    solution.x = -c2 / b2
                    solution.w = 1
                    RETURN solution
                END IF
            END IF
            D = b2 * b2 - 4 * a2 * c2
            IF D <= -zeroMax THEN
                RETURN (0, 0, 0, 0)
            END IF
            IF D > zeroMax THEN
                D = sqrt(D)
                solution.x = (-b2 - D) / (2 * a2)
                solution.y = (-b2 + D) / (2 * a2)
                solution.w = 2
                RETURN solution
            ELSE IF D < -zeroMax THEN
                RETURN (0, 0, 0, 0)
            ELSE
                solution.x = -b2 / (2 * a2)
                solution.w = 1
                RETURN solution
            END IF
        END IF
        RETURN solution
```

```
END FUNCTION
```

Appendix A.5

Pseudocode for the generation of a *blend summation map*.

```
FUNCTION GenerateBlendSummationMap(individual)
    blendSummationMap = EmptyMap()
    InitialiseMap(blendSummationMap)
    FOR EACH node IN individual DO
        FOR EACH point IN blendSummationMap DO
            bn = CalculateNodeBlendFactor(node, point)
            previousBlend = blendSummationMap.GetValueAt(point)
            blendSummationMap.GetValueAt(point) = previousBlend + bn
        END FOR
    END FOR
    FOR EACH edge IN individual DO
        FOR EACH point IN blendSummationMap DO
            be = CalculateEdgeBlendFactor(edge, point)
            previousBlend = blendSummationMap.GetValueAt(point)
            blendSummationMap.GetValueAt(point) = previousBlend + be
        END FOR
    END FOR
    RETURN blendSummationMap
END FUNCTION
```

Appendix A.6

Pseudocode for the generation of a *terrain of modifications*.

```
FUNCTION GenerateTerrainOfModifications(heightSummationMap, blendSummationMap)
    terrainOfModifications = EmptyMap()
    InitialiseMap(terrainOfModifications)
    FOR EACH point IN terrainOfModifications DO
        shv = heightSummationMap.GetValueAt(point)
        sbv = blendSummationMap.GetValueAt(point)
        modifiedHeight = shv / sbv
```

```
            terrainOfModifications.GetValueAt(point) = modifiedHeight
        END FOR
        RETURN terrainOfModifications
END FUNCTION
```

Appendix A.7

Pseudocode for generation of a *new terrain*.

```
FUNCTION GenerateNewTerrain(initialTerrain, terrainOfModifications, blendMap)
        newTerrain = EmptyMap()
        InitialiseMap(newTerrain)
        FOR EACH point IN newTerrain DO
            ihv = initialTerrain.GetValueAt(point)
            mhv = terrainOfModifications.GetValueAt(point)
            bv = blendMap.GetValueAt(point)
            newHeight = bv * mhv + (1.0 – bv) * ihv
            newTerrain.GetValueAt(point) = newHeight
        END FOR
        RETURN newTerrain
END FUNCTION
```

Appendix B

Chapter 4 related appendices.

Appendix B.1

Pseudocode for the generation of an accessibility map.

```
FUNCTION GenerateAccessibilityMap(terrain)
    accessibilityMap = EmptyMap()
    accessibilityMap.SetSizeTo(terrain.GetSize())
    FOR EACH point ON terrain DO
        gradient = GetSobelAtPoint(terrain, point)// See Appendix B.2
        IF gradient IS LESS THAN 0.5 THEN
            accessibilityMap.GetValueAt(point) = 1
        ELSE
            accessibilityMap.GetValueAt(point) = 0
        END IF
    END FOR
    RETURN accessibilityMap
END FUNCTION
```

Appendix B.2

Pseudocode for a Sobel filter that returns the gradient of a terrain at given coordinates.

```
FUNCTION GetSobelAtPoint(terrain, point)
    x = point.GetXCoordinate()
    z = point.GetZCoordinate()

    centerDepth = terrain.GetHeightAtCoordinates(x, z)
    depthsDiag = (0, 0, 0, 0)
    depthsAxis = (0, 0, 0, 0)

    xp1 = Clamp(x + 1, 0, terrain.GetWidth())
    xm1 = Clamp(x - 1, 0, terrain.GetWidth())
```

```
        zp1 = Clamp(z + 1, 0, terrain.GetDepth())
        zm1 = Clamp(z – 1, 0, terrain.GetDepth())

        depthsDiag.x = terrain.GetHeightAtCoordinates(xp1, zp1)
        depthsDiag.y = terrain.GetHeightAtCoordinates(xm1, zp1)
        depthsDiag.z = terrain.GetHeightAtCoordinates(xp1, zm1)
        depthsDiag.w = terrain.GetHeightAtCoordinates(xm1, zm1)

        depthsAxis.x = terrain.GetHeightAtCoordinates(x, zp1)
        depthsAxis.y = terrain.GetHeightAtCoordinates(xm1, z)
        depthsAxis.z = terrain.GetHeightAtCoordinates(xp1, z)
        depthsAxis.w = terrain.GetHeightAtCoordinates(x, zm1)

        depthsDiag -= centerDepth
        depthsAxis -= centerDepth

        horizDiagCoeff = (1, 1, -1, -1)
        vertDiagCoeff = (-1, 1, -1, 1)
        horizAxisCoeff = (1, 0, 0, -1)
        vertAxisCoeff = (0, 1, -1, 0)

        sobelH = depthsDiag * horizDiagCoeff + depthsAxis * horizAxisCoeff
        sobelV = depthsDiag * vertDiagCoeff + depthsAxis * vertAxisCoeff

        sobelX = LengthSquared(sobelH)
        sobelY = LengthSquared(sobelV)
        sobel = SquareRoot(sobelX * sobelX + sobelY * sobelY)

        RETURN sobel
END FUNCTION
```

Appendix B.3

Pseudocode for calculating the *Area Radius* attribute for each node in a *layout graph*.

```
FUNCTION CalculateAreaRadii(layoutGraph, terrain, accessibilityMap)
    FOR EACH node IN layoutGraph DO
        centre = node.GetGeographicalLocation()
        radius = node.GetDiameter() / 2
        step = terrain.GetWidth() / 8
        FOR 32 iterations DO
            circle = (centre, radius)
            valuesInCircle = accessibilityMap.GetValuesWithinCircle(circle)
            traversableCount = valuesInCircle.GetNumberOfValuesThatEqual(1)
            percent = traversableCount / valuesInCircle.GetNumberOfValues()
            IF percent > 90 THEN
                radius = radius + step
            ELSE
                radius = radius - step
            END IF
            step = step / 2
        END FOR
        node.SetAreaRadiusTo(radius)
    END FOR
END FUNCTION
```

Appendix B.4

Pseudocode for calculating the *Eigenvector Centrality* attribute for each node in a *layout graph*.

```
FUNCTION CalculateEigenvectorCentrality(layoutGraph)
    FOR EACH node IN layoutGraph DO
        node.SetEigenvectorCentralityTo(1)
        node.SetTotalInfluenceTo(0)
    END FOR
```

```
        FOR 16 iterations DO
                maxEigenvector = 16
                FOR EACH node IN layoutGraph DO
                        FOR EACH neighbour OF node DO
                                c = neighbour.GetEigenvectorCentrality()
                                t = node.GetTotalInfluenceValue()
                                node.SetTotalInfluenceTo(t + c)
                        END FOR
                END FOR
                FOR EACH node IN graph DO
                        c = node.GetTotalInfluence()
                        node.SetEigenvectorCentralityTo(c / maxEigenvector)
                END FOR
        END FOR
END FUNCTION
```

Appendix B.5

Pseudocode for calculating the *Betweenness Centrality* attribute for each node in a *layout graph*.

```
FUNCTION CalculateBetweennessCentrality(layoutGraph)
        FOR EACH node IN layoutGraph DO
                node.SetBetweennessCentralityTo(0)
        END FOR
        FOR EACH nodeA IN layoutGraph DO
                FOR EACH nodeB IN layoutGraph DO
                        IF nodeA IS NOT nodeB THEN
                                path = CalculateShortestPathBetween(nodeA, nodeB)
                                IF path.DoesExist() THEN
                                        FOR EACH nodeC ON path DO
                                                IF nodeC IS NOT nodeA OR nodeB THEN
                                                        c = nodeC.GetBetweennessCentrality()
                                                        c = c + 1
                                                        nodeC.SetBetweennessCentralityTo(c)
```

```
                                        END IF
                                    END FOR
                                END IF
                            END IF
                        END FOR
                    END FOR
                    N = layoutGraph.GetNumberOfNodes()
                    FOR EACH node IN layoutGraph DO
                        c = node.GetBetweenessCentrality()
                        c = c / 2                                   // Correction for undirected graph
                        c = c / ((N - 1) * (N - 2) / 2)             // Normalisation
                        node.SetBetweennessCentralityTo(c)
                    END FOR
END FUNCTION
```

Appendix B.6

Pseudocode for calculating the *Closeness Centrality* attribute for each node in a *layout graph*.

```
FUNCTION CalculateClosenessCentrality(layoutGraph)
    FOR EACH nodeA IN layoutGraph DO
        reachable = 0
        nodeA.SetClosenessCentralityTo(0)
        FOR EACH nodeB IN layoutGraph DO
            IF nodeA IS NOT nodeB THEN
                path = CalculateShortestPathBetween(nodeA, nodeB)
                IF path.DoesExist() THEN
                    reachable = reachable + 1
                    c = nodeA.GetClosenessCentrality() + path.GetLength()
                    nodeA.SetClosenessCentralityTo(c)
                END IF
            END IF
        END FOR
        nodeA.SetClosenessCentralityTo(reachable / nodeA.GetClosenessCentrality())
```

```
    END FOR
END FUNCTION
```

Appendix B.7

Pseudocode for calculating the isovist measures for each node in a *layout graph*.

```
FUNCTION CalculateIsovistMeasures(layoutGraph, terrain)
    maximumViewDistance = terrain.GetWidth() / 2
    resolution = 1024
    eyeHeight = terrain.GetWidth() * 0.025
    FOR EACH node IN layoutGraph DO
        terrainHeight = terrain.GetHeightAt(node.PositionX, node.PositionZ)
        origin = (node.PositionX, terrainHeight + eyeHeight, node.PositionZ)
        // See Appendix B.8 for function GenerateIsovist
        isovist = GenerateIsovist(origin, maxViewDistance, resolution, terrain)
        node.SetAverageRadialLengthTo(CalculateAverageRadialLength(isovist))
        node.SetDispersionTo(CalculateDispersion(isovist))
        node.SetDrift3DTo(CalculateDrift3D(isovist))
        node.SetDrift2DTo(CalculateDrift2D(isovist))
        node.SetMaximumRadialLengthTo(CalculateMaximumRadialLength(isovist))
        node.SetMinimumRadialLengthTo(CalculateMinimumRadialLength(isovist))
        node.SetSkewnessTo(CalculateSkewness(isovist))
        node.SetSphericalityTo(CalculateSphericality(isovist))
        node.SetStandardDeviationTo(CalculateStandardDeviation(isovist))
        node.SetVarianceTo(CalculateVariance(isovist))
        node.SetVolumeTo(CalculateVolume(isovist))
    END FOR
END FUNCTION
```

Appendix B.8

Pseudocode for the generation of an isovist.

```
FUNCTION GenerateIsovist(origin, maxViewDistance, resolution, environment)
    isovist = NewIsovist()
```

```
        isovist.SetOriginTo(origin)
        isovist.SetRadialCount(resolution)
        FOR EACH radial IN isovist DO
                direction = GetRandomDirection()// See Appendix B.9
                radial.SetDirectionTo(direction)
                distance = RayCast(origin, direction, environment)
                IF distance < maxViewDistance THEN
                        radial.SetLengthTo(distance)
                ELSE
                        radial.SetLengthTo(maxViewDistance)
                END IF
        END FOR
        RETURN isovist
END FUNCTION
```

Appendix B.9

Pseudocode for calculating a random direction in 3D space.

```
FUNCTION GetRandomDirection()
        phi = π * 2 * RandomValueBetween(0, 1)
        theta = 2 * asin(sqrt(RandomValueBetween(0, 1)))
        x = radius * sin(theta)
        y = x * sin(phi)
        x = x * cos(phi)
        z = cos(theta)
        RETURN (x, y, z)
END FUNCTION
```