CHALMERS
UNIVERSITY OF TECHNOLOGY

Flock O' War

A Procedural Strategy Game Based on Real-Time Flocking Behaviour

Bachelor's thesis in Information Technology

Erik Jergéus
Maria Fornmark
Max Arfvidsson Nilsson
Sigbjørn Kjesbu Drøsshaug
Simon Olsson
Tobias Karlsson

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

BACHELOR THESIS 2020

Flock O' War

A Procedural Strategy Game Based on Real-Time Flocking Behaviour

Erik Jergéus
Maria Fornmark
Max Arfvidsson Nilsson
Sigbjørn Kjesbu Drøsshaug
Simon Olsson
Tobias Karlsson

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Flock O' War
A Procedural Strategy Game Based on Real-Time Flocking Behaviour

Supervisor: Mads Rønnow, Department of Computer Science and Engineering, Chalmers University of Technology
Examiner: Michael Heron, Department of Computer Science and Engineering, Chalmers University of Technology

Bachelor Thesis 2020
Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2020

Abstract

In nature, it is common for animals to move together in a cohesive pattern, but without any apparent direction. This is referred to as flocking, herding or schooling etcetera depending on the species. The purpose of this bachelor's thesis is to investigate whether the basic rules of flocking can be utilised as a gameplay mechanic to create a fun and realistic looking game. To accomplish this, a strategy game with flocking as a key feature was created within the Unity and C# environment. A large focus was put on gameplay elements, such as animations, sound, and visual appearance, for the players to have a more enjoyable experience. The result is an entertaining strategy game for two players where each player is in charge of their own army of troops, that move according to the rules of flocking, battling against each other on a procedurally generated landscape.

Sammandrag

I naturen är det vanligt att djur rör sig tillsammans i ett sammanhängande mönster, dock utan någon uppenbar riktning. Detta fenomen bildar flockar, hjordar eller stim, etcetera, beroende på arten. Syftet med det här kandidatarbetet är att undersöka ifall de grundläggande reglerna för flockbeteende kan användas som en spelfunktion, för att skapa ett spel som är roligt och ser realistiskt ut. För att uppnå detta skapades ett strategispel, med flockbeteende som en huvudfunktion, inom Unity- och C#-miljön. För att spelarna skulle ha en bättre upplevelse så lades ett stort fokus på spelfunktioner, så som animationer, ljud, och utseende. Resultatet är ett underhållande strategispel för två spelare där varje spelare är ledare över sin egen armé, som rör sig enligt reglerna för flockbeteende och slåss mot varandra på ett procedurellt genererat landskap.

Keywords: flocking behaviour, aggregations, video-game, strategy, procedural generation, human, simulation, unity.

Acknowledgements

We would like to thank our supervisor, Mads Rønnow, for his support during this project. Your aid and encouragement allowed us to go further and explore more possibilities. We would also like to thank Brackeys and Board To Bits Gaming for helping us with their programming tutorials on YouTube. Without their help, we would have not come as far as we did.

Group DATX02-20-88, Gothenburg, May 2020

Contents

List of Figures

List of Tables

1
Introduction

Flocking is the phenomenon where many animals are moving together as a collective unit and appears among many species in nature, such as schools of fish or flocks of birds [1]. The interest in understanding the origins and simulating this behaviour has resulted in findings that show the behaviour to be based on simple rules followed by the individuals [1, 2]. Although the concept of simulating flocking was initially developed in the field of computer graphics, there are also many applications within various other research fields[1, 2].

Simulating flocks is interesting in the field of computer graphics because it is desirable to create a realistic rendering of flocks for movies and games [2]. However, flocking is also interesting in the field of evolutionary biology and behavioural ecology, where simulations of flocking are used to observe how particular individual behaviour facilitates the emergence of flock behaviour [3]. For example, the rules of flocking have been used to simulate the emergence of new behaviours in the presence of evolution. By extending the standard rules of flocking to include behaviours such as the ability to give birth and the need for feeding, the behaviour of a multicellular organism emerged [3]. These simulations can be useful as a complement to experiments in nature, as simulations are less time consuming and easier to organise. They are also easier to verify since it is possible to control all the parameters [2]. Other situations could also be modelled as a flocking behaviour, for example, traffic flow simulations could be modelled as a flocking situation with added constraints such as lanes [2].

As mentioned above, flocking is normally a concept used to describe a collective behaviour among animals, such as birds or fish, and less often to describe the behaviour of humans in groups. An interesting question is whether these rules of flocking can be applied to humans moving in groups. Especially since the experiments on evolution and flocking indicate that by adding additional rules to the flocking, it is possible to create complex behaviour. This means that human flocking possibly could be simulated by adding behaviours significant for humans. To do that it is important to understand the rules of human flocking. Do humans ”flock” according to the same rules as animals? If not, what is the difference? Can the basic rules of flocking be extended to create a seemingly realistic human behaviour?

1.1 Aim

The aim is to investigate whether the basic rules of flocking can be utilised as a gameplay mechanic to create a fun and realistic looking game. A strategy game is developed, where the flocking is used to simulate the behaviour of two opposing forces. Further, it is investigated whether the use of flocking as a movement strategy retains a human-like appearance when used to move armies. Lastly, it is investigated how the standard algorithm can be extended to implement a fight or flight behaviour and differentiate the behaviour of different kinds of troops.

1.2 Scope

The main focus of the project is to develop an entertaining game using flocking behaviour. Particular emphasis is on creating good core gameplay elements and extending the flocking algorithm in order to be a compelling part of the game. The project's focus is not on optimising the algorithm to allow for as large flocks as possible, but some optimisation might still be needed to make the game enjoyable on most computers. Neither is the project focusing on user interaction, although some design work is required for improving the gameplay.

2
Theory

This chapter gives a theoretical background to introduce the key concepts relevant to this report. First, the phenomenon of flocking in nature is presented, followed by a review of the work that has been done on simulating flocking behaviour. Then, flocking mechanisms among humans are discussed as well as methods for optimising flocking algorithms. Finally, some game design concepts are introduced.

2.1 Aggregations and Flocking

Understanding the mechanisms of flocking in nature is fundamental when creating simulations of flocks, which is a kind of aggregation. Aggregations in nature have been researched by Parrish et al. [4] and are described as the formation of a whole from many units. The resulting formation is coherent and cohesive, thus removing the focus of the observer from the individual members to the group. Flocking is the phenomenon of animals moving in groups, such as flocks of birds and schools of fish, and emerges from the phenomenon of aggregation [1].

Biologists have researched mechanisms of aggregations, especially how the individual benefits from the group [1]. There lies and self-evident advantage in being indistinct within a group, for example when attacked by predators. However, there are still many questions to answer on why this type of behaviour, where similarity and conformity are enforced, has emerged in nature.

According to Parrish et al. [4], aggregations can be divided into two categories; passive and active aggregations. In passive aggregations, the members of the flock are passively brought together by outside forces, while in active aggregations the group is formed by active decisions from its members. An example of passive aggregation could be jellyfish or algae gathered together by currents in the ocean. An active aggregation could for example form around a food source that many individuals gather around. To avoid a too dense aggregation, which can lead to depletion of for example food or oxygen, repulsive forces develop to avoid the individuals ending up too close to each other. The combination of attractive and repulsive forces lead to group structures, much similar to the aggregations formed by atoms and molecules, by reaching an equilibrium of attracting and repulsive forces.

A congregation is an aggregation resulting from attraction in the group itself regardless of outside forces, such as flocks of birds and schools of fish [1, 4]. The formation emerges from a combination of mutual attraction and repulsion between the individuals. Although the type of interaction differs among different species, there are some general features ascribed to congregations [1]. There are usually distinct edges, the density is fairly uniform and often all units are facing the same direction. Moreover, the individuals have the freedom to move and there are often coordinated movement patterns.

Parrish et al. [4] summarise a few interesting topics to investigate regarding animal congregation. They reflect on the importance of the individual member, how the group is functioning and how the boids work together to form a group:

- What does the individual know about the whole group?
- What are the benefits and costs of being in the group?
- Is there an optimal group size?
- How are the boundaries determined and why are they distinct?
- Why are there patterns?
- What are the assembly rules?
- Can models of the whole predict individual behaviour?

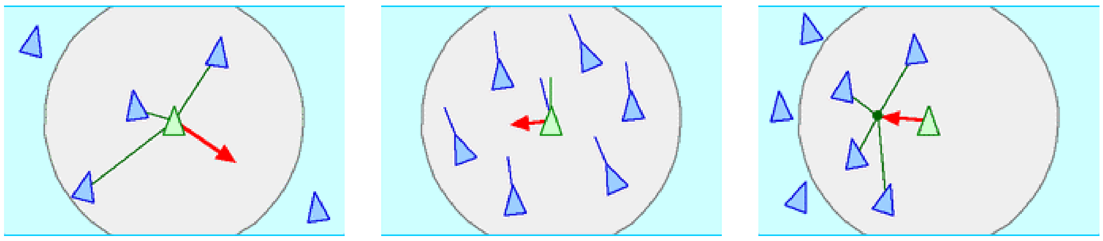
2.2 Simulation of Flocking

When simulating flocking one can not simply simulate the thought process of the animals involved. Instead one needs to create rules based on the behaviour observed in nature. When this is done the rules can be converted into code which can run on a computer. Here it is very important that the rules are simple enough that the simulation can run while still managing to mimic reality.

2.2.1 The Basic Rules of Flocking

The first algorithm for simulating flocking behaviour was presented by Reynolds in 1987 [2]. This publication is now regarded as a fundamental piece within the field and is widely cited in later works [1, 3]. In his pivotal piece, Reynolds introduced the concept bird-like object, boid, and presented the boid model. A boid is a general term for a member in a flock and the boid model presents how the boids interact with each other to form a flock [2].

Reynolds viewed simulating flock behaviour as similar to simulations of particle systems, with the exception that instead of single point particles it is comprised of boids with geometric shapes and physical attributes. As opposed to particles, boids also have an orientation and can interact with each other. These interactions can be summarised into three main rules: collision avoidance, velocity matching and flock centring, also named separation, alignment and cohesion respectively. A visualisation of the rules is seen in Figure 2.1



(a) Separation Rule. From [5], public domain. **(b)** Alignment Rule. From [6], public domain. **(c)** Cohesion Rule. From [7], public domain.

Figure 2.1: A visualisation of the three rules of flocking, separation, alignment and cohesion.

Reynolds' proposed algorithm represents the internal state of each boid as a vector representing the direction and the speed of that boid. The impact of the three rules mentioned above is evaluated based on the distances to flock mates and their directions. To obtain realistic flocking behaviour, the individual forces have to be balanced through weights.

Reynolds' ideas are more explicitly described by Spector et al. [3], where the velocity vector $\vec{V}$ is composed of the sum of five different components each with a weight c_i. In addition to the original three flocking rules, two components were added to further tailor the behaviour to the desired situation. The resulting behaviour is shown in Equation 2.1:

$$\vec{V} = c_1\vec{V_1} + c_2\vec{V_2} + c_3\vec{V_3} + c_4\vec{V_4} + c_5\vec{V_5} \tag{2.1}$$

$\vec{V_1}$ is the movement away from crowds, $\vec{V_2}$ is the movement towards the centre of the world, $\vec{V_3}$ is the average neighbour velocity, $\vec{V_4}$ is moving towards the centre of all agents and $\vec{V_5}$ is a random factor. Just like the centre of the world and the random factor ($\vec{V_2}$ & $\vec{V_5}$) behaviours that were added here, other additional behaviours could be added and influence the behaviour of the boids, resulting in more complex behaviour.

Another important aspect to consider is the boid's perception of its surroundings. Failing to provide the boid with a realistic amount of information gives an unnatural flocking behaviour [2]. An animal in the real world has a limited field of view, and mathematical models need to reflect that aspect to produce a realistic result [1]. Hence, Reynolds allowed the sensitivity of nearby neighbours to decrease by an inverse exponential of the distance.

The complexity of simulating both perception and behaviour means that testing the flocking model is difficult [1]. This is mainly due to the large number of parameters and special conditions present, meaning that there is a vast amount of cases to be tested. Verifying the behaviour against real-world data was done only relatively recently [8].

2.2.2 Extending Flocking Rules

While the first simulation of flocking was presented by Reynolds in 1987, there were also others trying to define mechanisms for flocking behaviour. In 1986, the mathematician Okubo [9] had theories on how flocking behaviour could be described by mathematical models. In 1990, Heppner and Grenander [10] presented another computer simulation which was based on differential equations but still involved repelling and attracting forces.

Most of the work from the 1990s and onwards is based on Reynolds' work, investigating the effect of adding more behaviours, specific behaviours for different species, and experimenting with changing the values of the weights. The lack of research within this field, especially during the 1990s, might be due to the advanced mathematics and computational resources required, in combination with difficulties in observing the resulting behaviour [1].

In 1993 and 1994 Reynolds [11, 12] published some work on the evolution of flock behaviour and was followed by Spector in 2002 and 2003 [13, 3]. This was mainly done by introducing new behaviours and evaluating the effect of prioritising these behaviours. A few examples of new behaviours were fear of predators, avoiding obstacles and introducing food sources.

After 2003 there has been plenty of work involving flocking simulations, most of them based on extending or modifying behaviours in Reynolds' model. For example in 2006, Hartman [14] added behaviour for defining the leader of a flock based on behaviours seen among birds. In 2007, Delgado-Mata [15] worked on introducing behaviour responding to fear. Also, in 2010 Hildenbrandt [8] compared studies of real bird flocks of starlings in Rome with their model for flocking, with satisfying results. The model was based on Reynolds' theories with added behaviours specific for starlings, such as aerodynamics, number of flock neighbours, and movement around sleeping sites.

2.3 Flocking Behaviour in Human Agents

Human's ability to efficiently communicate during the organisation of group movement generally overshadows the use of non-communicative strategies, such as flocking. Even so, spontaneous flocking has been recorded to occur in human agents when they have limited perception radius, means of communication, and no externally set goals. In other words, there exists an inherent desire for humans to spontaneously act as a flock, without a requirement of external goals [16].

S. Frey & R. L. Goldstone describes in [17] that human flocking arises due to social norms, which in turn originate from human's ability to guess the reasoning of other similar agents. They also pose the idea that human groups converge due to shared priorities and resources, comparable to animal aggregations. Research on consensus decision making in human groups confirms this behaviour by describing how groups of human agents possessing conflicting priorities still facilitate efficient consensus decisions in a flocking manner [18].

Reynold's simulations used a model based on physics that determined the velocities based on an acceleration that was applied in accordance with the composite behaviour in each time step. This worked properly for the simulation of agents that moves fast in comparison to their acceleration since it resulted in smooth changes of direction. Z. Shen and S. Zhou [19] describe how a position-based steering algorithm simulated their human agents in military operations more efficiently, due to these agents moving slowly and changing direction often. This algorithm, however, is optimised for smaller groups moving in urbanised terrain, which creates a large focus on the individual, ignoring some factors regarding group movement.

One way to tailor the flocking algorithm towards simulating large groups of humans in combat is described in [20]. Reynold's traditional flocking algorithm [2] is used as the backbone of formation keeping, but it is expanded upon by introducing leaders. These leaders have direct communication with their group of agents and issue directional and combat commands to get the group to work towards some goal. While it better accommodates the battle scenario, it does leave some important aspects of flocking behind; the decision making of individuals in the absence of direct communication.

2.4 Procedural Generation of Terrain

Procedural generation is the process of creating data through an algorithm rather than doing it manually [21]. This technique has been used to generate content for games as early as the 1980s.

A common method to generate procedural terrain is through the use of Perlin noise. Perlin noise is a form of generated data that can be distributed across a 2D plane [22]. Instead of being completely random, the data tends to group up and be similar to nearby points of data. If you display this data on a 3D graph, with the values being the height, you get something reminiscent of hills and valleys , see Figure 2.2

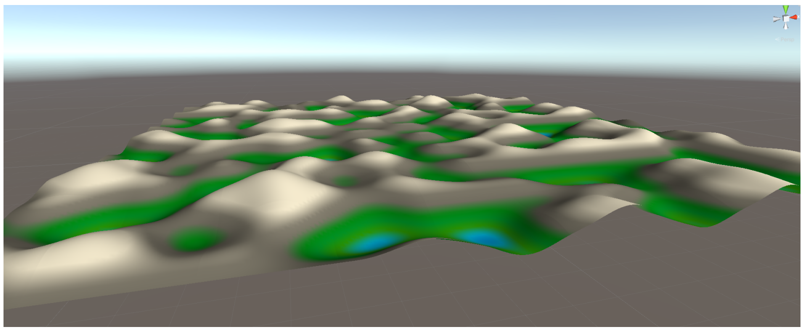


Figure 2.2: Procedurally generated terrain using perlin noise.

To make the terrain more complex it is possible to stack several layers of Perlin noise on top of each other [23]. This technique is called using octaves, where the number of layers is referred to as the number of octaves used. When using octaves, it is important to resize the layers over time, so that the different layers can represent different levels of detail. This is done through the two variables lacunarity and persistence. Lacunarity controls how much the size will decrease in the x and z-axis for every new layer, which makes the groupings of data smaller and closer to each other. Persistence instead changes how much the height decreases. If these are adjusted correctly, the first octave will represent hills, while the third might represent potholes and larger stones, see Figure 2.3.

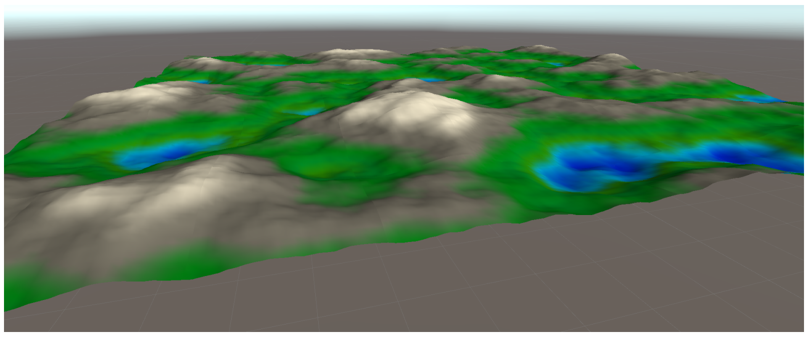


Figure 2.3: Shows procedurally generated terrain using perlin noise and octaves.

2.5 Game Design

When designing a game the creator has to make several design choices, which will impact the expectations and perception of the game. In the book *The Art of Game Design* [24], Jesse Schell describes many aspects of game design with a focus on the process the creators go through when making decisions. Furthermore J. Schell discusses the benefits and dilemmas of common design decisions.

2.5.1 Type and Genre

In [25] L. Grace describes game types as critical aspects of a game which heavily influence how the game is played and the skills required from the player. For example, in action games, the main allure is the intensity of the action and good reflexes often determine the skill of the player, while strategy games rely on problem-solving and game knowledge. By choosing appropriate game types the designer can get an understanding of their intended audience's skills and reason for playing and as such adapt the game to fit them.

L. Grace [25] continues with explaining ”Genre” as an aspect that influences the look and feel of the game and often affects the structure of narrative elements. Since both the genre and type influences the narrative style, a thought out connection between the two creates a more cohesive experience. For example, the horror genre blends well with the role-playing type, while a simulation type would have a hard time amplifying horror aspects.

2.5.2 Game World

J. Schell [24] expresses that decisions regarding how the game world's setup lets the designers pick formats that works best with their theme, interactions, and the experience they want to mediate. He defines the game world as the medium that translates an idea into something tangible; a description of the game's setting that allows the player to understand the spatial relationship between objects. Some core design decisions in regards to the game world are whether the world is an open-world (continuous) or level-based (discrete), if the world is constructed in 2D, 3D or even in text format, as well as if the game is linear (often seen in books) or continuous.

The decision to create an open-world scenario facilitates more freedom and creative possibilities at the expense of the designers' ability to tailor the experience towards the intention and theme of the game. Therefore many designers choose to implement a mix of both. The most common way is by having certain parts of the world be portioned off and more heavily scripted, often referred to as dungeons or instances. Giving control to the players is another way to create the freedom that is more tailored towards simulation games. This way the player can tailor the game towards their preferences and create the perfect map, character, or game element for them. However, J. Schell [24, p. 203] argues that while the player wants to have an entertaining experience, they also want to make the game easier for themselves. Therefore it is likely that they give themselves a lot of power, which in turn makes the standard game less enjoyable since they now feel a lack of power.

Choosing a 2D world focuses the player on the gameplay as the world is usually less complex in comparison with 3D, where the players can better relate to and thereby appreciate what is happening in the world. Text-based-adventures were common in the earlier days of game development since they were easier to make and allowed the player to use their imagination to experience the world, however, they had limitations in how descriptive they could be and required more effort from the player [24, p. 143–144].

2.5.3 Player Interaction

K. D. Saunders and J. Novak [26] express the importance of the interface in regards to interactivity in games. It is what makes it possible for a player to connect with the game and immerse themselves within the world. To facilitate immersion, the interface needs to be intuitive, give feedback and be easy to interact with. By simplifying the interface and following standards in the industry it becomes easier for the player to know how to perform the action they intend.

T. Fristoe describes [27] how important it is for the game to respond to the player's actions to establish that the player has an impact on the world. By the game's units behaving differently depending on what the player did, it becomes apparent that the player affected the state of the game. Another common way of interacting is player-to-player, where the game provides a framework for the players to come together and create exciting challenges. Furthermore, Fristoe discusses how a designer needs to be aware of too much interaction, which can result in an adverse effect of chaos and loss of player control. Therefore it is most often better to have fewer and well-polished interactions, rather than overwhelm the players.

2.5.4 End Condition

While a game is not required to have a clear goal or end condition to yield an enjoyable experience, it is still an aspect that every game designer should discuss. By introducing ways to win or lose the game designer gets a seamless tool to create tension, risks, and goals for the player. To accentuate the end condition's importance, it is critical for the player to be challenged by the game and realise that they need to act in order to win. If the end condition is imbalanced the game becomes boring and pointless since either the game is too easy and there are no risks for the player, or the player gets frustrated from repeated losses [24].

2.5.5 Resources

For a player to be able to feel accomplished in a challenge a game must allow for strategy, which in turn requires that a player has access to the strategic management of resources. The resources themselves can be any sort of commodity that the player can affect and use and the management of them is what encourages strategic gameplay. Balancing these resources is important to make the game feel fair, allow options for the player, and at the same time be scarce enough so that the player wants to use them in the best way possible [24].

3
Method

This chapter outlines the methods used during the course of the project. It presents the tools used during the game's development, both for the game itself and for the process surrounding it. It also describes the implementation details of the game, the iterative process of developing the flocking behaviour used in the game, and the game design decisions that were made. Lastly, it also describes how the game was tested throughout the project.

3.1 Tools

During the course of the project, an array of tools were used. Many of these have been crucial to the success of the project, as they have provided the most basic functionalities for creating the game. Among these are tools specifically developed for game development, but also other tools for software development in general.

3.1.1 Game Engine

When making a game, the use of a game engine is a great choice to help facilitate the implementation, as it alleviates many of the commonly faced challenges during development. Some of these challenges are how to render graphics on the screen correctly, adding audio and animations, as well as organising a large number of assets and components needed, which many game engines can handle for the developer. While there are many game engines available to choose from, after some initial discussion, the choice stood between Unity [28] and Unreal Engine [29].

The group felt more comfortable working in Unity's C# environment as opposed to the Unreal Engine's C++ environment, since C# is syntactically similar to Java, which the group was proficient in already. Additionally, the group's supervisor recommended Unity, as it was considered to have a less steep learning curve. Taking these things into consideration, the group decided to go with Unity.

Unity is a cross-platform game engine developed by Unity Technologies [28] and is free to use for non-commercial use and smaller companies generating less than 100 000 US dollars annually [30]. Some of the benefits Unity provides are rendering, garbage collection, a built-in physics system with collision detection out of the box, terrain-generating tools, a What You See Is What You Get (WYSIWYG) editor enabling quick prototyping, and an asset store [31].

3.1.2 Unity Asset Store

The Unity asset store offers a variety of assets - either for free or for sale. Developers get access to many different types of assets, ranging from 2-D and 3-D models, to sound effects, visual effects, and more. Some of the assets available in the store come in pre-packaged bundles, ready to use out of the box.

The group decided upon an asset pack containing many different assets for a medieval-themed game such as knights and castles. These can be found in Appendix A.2. Using game assets allowed the group to focus more on the implementation details and game design of the project, as opposed to texturing and modelling custom assets from the ground up.

3.1.3 Visual Studio IDE

Scripts in Unity are used to control the behaviour of objects in the game world [32]. All objects in Unity, which are called “Gameobjects” [33], can have scripts attached to them, which allows for the implementation of complex behaviour. When installing Unity, it comes with the option to also install Visual Studio [34], which is a popular IDE developed by Microsoft. This led the group to use this IDE to implement the scripts needed to model the behaviours used in the game.

3.1.4 Git Version Control

The Git version control system [35] was used to keep track of development efforts across team members. Using git alleviates many of the most commonly faced challenges during development by allowing the developer to experiment and save code state at different points in time, which can later be merged into a stable codebase, or reversed in case of bugs. This enables a programmer to develop in safe, small increments without fear of making irreversible errors.

3.1.5 GitHub

To synchronise the work done by team members, the group used GitHub [36], which allows the group to work in parallel on the project, with the use of branches [37]. The master branch is conventionally the main branch of the project where a stable version of the project lives. To ensure a working and stable master branch, the group made use of code reviews before allowing members to merge their code into the master branch.

When a member of the group finishes a task on their branch, they create a pull request [38], asking to merge the work on their branch into the master branch. The pull request signals for the rest of the team to review the changes before approving or disapproving the merge. The practice of reviewing new features before adding them to the main project is a commonly used technique for improving code quality while increasing the likelihood of finding bugs and enforcing agreed-upon code practices.

Pull requests also tend to become a focal point for discussion during meetings, which in turn proliferates knowledge about the codebase across team members.

3.1.6 Remote Communication

During the project, weekly meetings and discussions were done in person, but as the project progressed, it became necessary to do meetings and discussions remotely. Slack [39] was chosen for group communication, as its support of threads [40] helped organise lengthy discussions in one place. Slack was also used to coordinate remote meetings that would later be done through Zoom [41]. A GitHub bot was added to the Slack workspace [42] that would automatically post a notification to Slack when a new pull request was pushed to the GitHub repository, allowing team members to keep up with changes easier, and be notified when a pull request was awaiting approval. Communication with the supervisor was done mainly through Signal [43].

3.2 Flock Behaviour Development

The beginning of the project was mostly spent on trying to convert the traditional version of flocking into something resembling human behaviour. After this much time was spent on extending the flocking behaviour adding components to make it behave more like an army at war. Towards the end time was mostly spent on fine tuning the behaviour to make it more versatile and reliable.

3.2.1 Prototyping

The project's initial flocking behaviour implementations were based on a Unity tutorial series done by the YouTube channel Board to Bits Games [44], which laid the foundation for the implementation chosen for the game. After some initial development, where different unit types were added, the need arose for more complex behaviour beyond what was provided in the basic implementation of the flocking algorithms. The default flocking behaviour produces some interesting behaviour, but left to itself, it is without direction and purpose. This seemingly random behaviour might suffice for a flock of birds to appear realistic, but did not give the desired outcome for army units moving around on a battlefield.

3.2.2 Composite Behaviours

Given that the game called for more sophisticated behaviour than randomly flocking units, composite behaviours were implemented to allow for the combination of the three base behaviours; separation, alignment and cohesion, with additional behaviours (see Figure 3.1). These composite behaviours were used to give units a specific behaviour depending on its type. For instance, scout-types can detect units further away and tries to steer flocks towards winnable battles, while infantry tries to follow scouts and occupy the front line in battle.

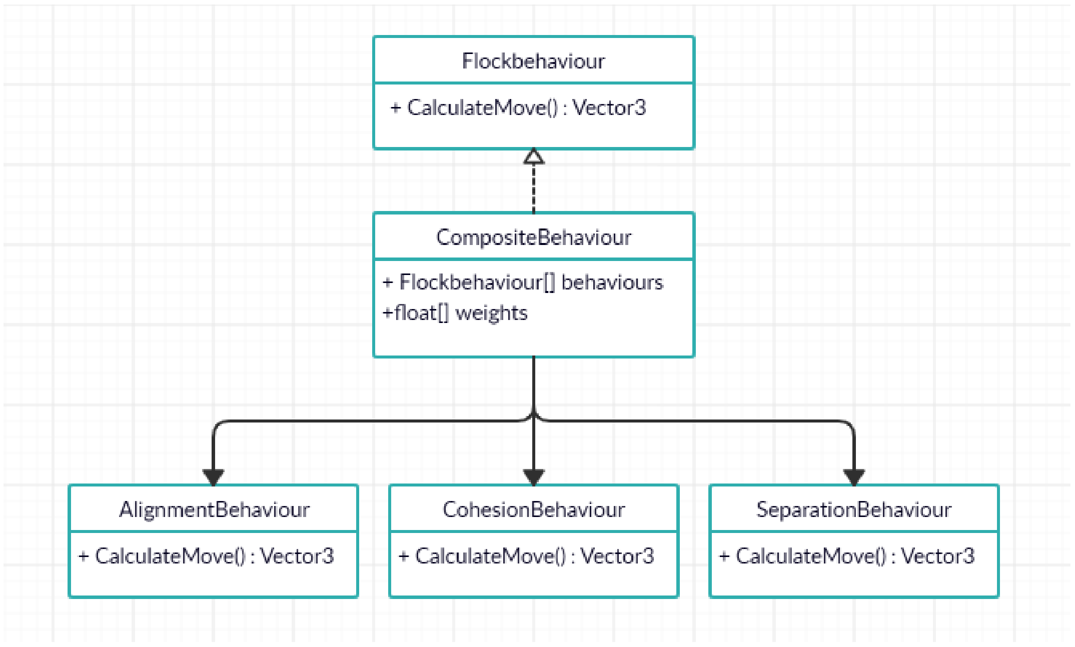
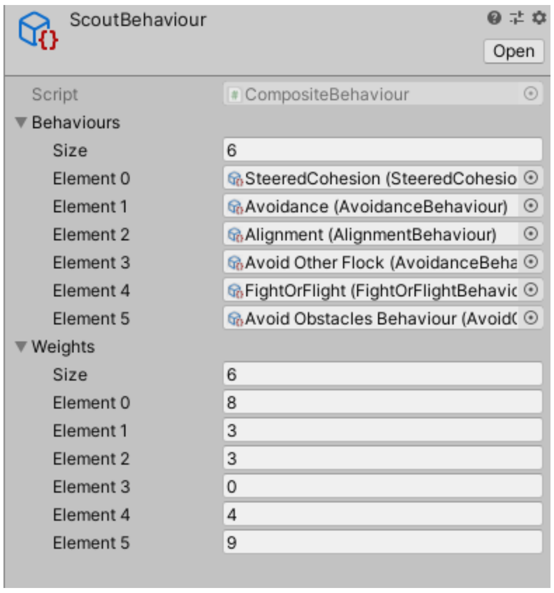

Flockbehaviour
+ CalculateMove() : Vector3
CompositeBehaviour
+ Flockbehaviour[] behaviours
+float[] weights
AlignmentBehaviour
+ CalculateMove() : Vector3
CohesionBehaviour
+ CalculateMove() : Vector3
SeparationBehaviour
+ CalculateMove() : Vector3

Figure 3.1: UML diagram showing the relation between the three base behaviours and the CompositeBehaviour. The CompositeBehaviour can be extended with additional behaviours.

By combining several behaviours into a composite behaviour and assigning weights to each specific behaviour, the composite behaviour can be tailored to represent how a specific type of unit should behave. This implementation allowed for a streamlined process when adding new unit types, as it allowed for the combination of already existing base behaviours, which every unit has in common, and the new behaviour (see Figure 3.2).

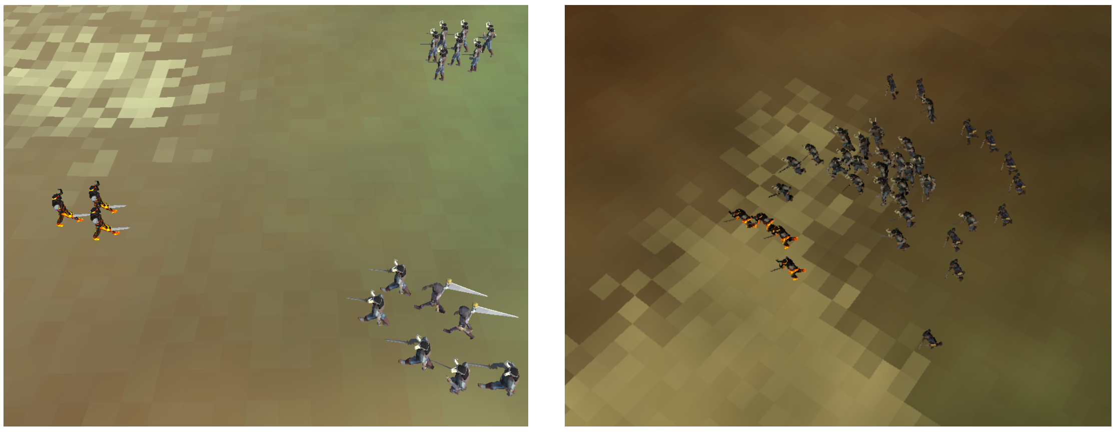


Script	CompositeBehaviour
▼ Behaviours	
Size	6
Element 0	SteeredCohesion (SteeredCohesio
Element 1	Avoidance (AvoidanceBehaviour)
Element 2	Alignment (AlignmentBehaviour)
Element 3	Avoid Other Flock (AvoidanceBeha
Element 4	FightOrFlight (FightOrFlightBehavic
Element 5	Avoid Obstacles Behaviour (Avoid(
▼ Weights	
Size	6
Element 0	8
Element 1	3
Element 2	3
Element 3	0
Element 4	4
Element 5	9

Figure 3.2: CompositeBehaviour for the scout-type unit in Unity showing behaviours and their associated weights.

3.2.3 Additional Flocking Behaviours

In a typical flocking scenario, agents run around and flock with each other depending on the weights of their specific behaviours. To have a more realistic behaviour for the armies, units should only want to flock together with other units on the same team, while trying to attack or avoid the other team's units depending on the immediate circumstances. If a smaller group of units is overwhelmed by a larger one, they should fall back, but if they instead outnumber the enemy, they should go on the offensive (see Figure 3.3). The decision making of each unit is based on the given weights to its behaviours in the composite behaviour for that specific unit type. The weights given to these behaviours determine how strongly it wants to stay together with its own, and how aggressive it is, among other things.



(a) A scenario where the scouts leads a group of units towards the enemy, while the group without scouts are clueless.

(b) A scenario where the blue team is outnumbering the red team, which is fleeing.

Figure 3.3: Scenarios illustrating scouts increased field of vision and the fight or flight behaviour.

To ensure that the game finishes in a reasonable amount of time, behaviour to make the armies converge on a common destination was added. Different solutions were considered, such as making every unit have a preference for walking towards the centre of the map to battle it out, or having a natural focal point such as each team having a castle they need to defend. The castle gives the player incentive to spawn units near it to survive or attack the other team's castle to win. After some playtesting, castles were chosen as they felt like the most intuitive way of achieving the goal while also adding a new gameplay element.

3.2.4 Optimisation and the $O(n^2)$ Problem

There is a common problem in flocking scenarios which comes from how an agent calculates its movements from other agents in its surroundings, and the scaling problem that naturally comes with this. In a straightforward implementation, each agent checks every other agents position and makes calculations for its movements based on that position, giving $O(n^2)$ every frame update.

One solution to this problem is to divide the playing field into equal parts, by using a grid pattern, then when the movement calculation is run, only the agents within the same cell or neighbouring cells are checked, to reduce the number of iterations needed. This solution runs into the same problem if a lot of agents happen to be within the same cell. The group looked into potential solutions to this problem, such as using quadtrees [45] for subdividing cells containing some amount of agents that is above some predefined threshold, but instead decided to use an already existing functionality within Unity. The solution was to use Unity's built-in physics system through utilising colliders to detect nearby units that happen to be within a given range. The OverlapSphere method [46] in the physics class of Unity's scripting API is often used for range detection and was thus a suitable solution to this problem, see Figure 3.4.

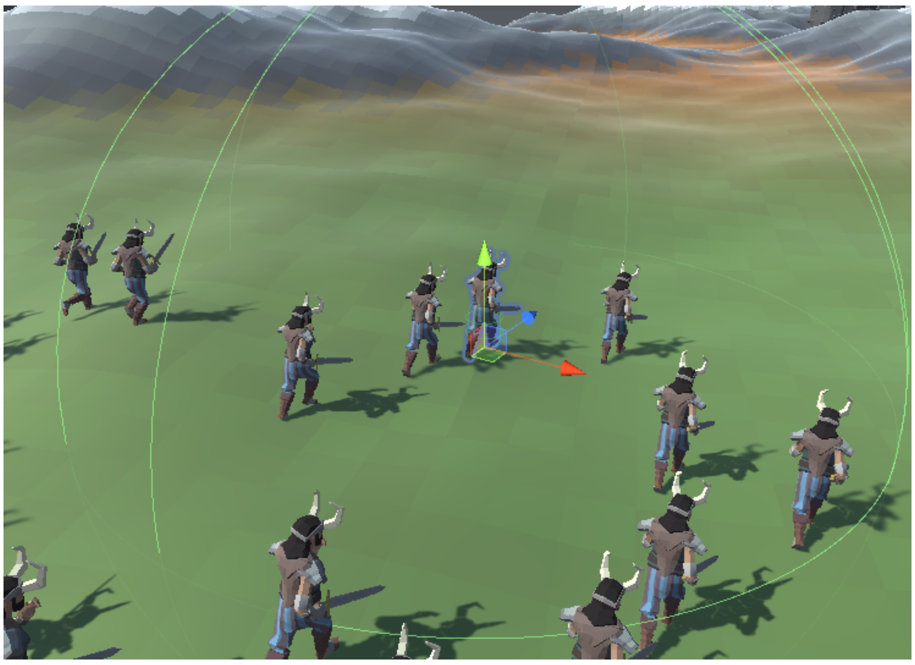


Figure 3.4: The collider used to detect nearby units, which in this case results in the targeted unit detecting seven neighbours.

3.3 Game Design

During the course of the project, game design focus went through several phases. Initially, focus was on prototyping and brainstorming game mechanics to make an entertaining game. The middle parts of the project were instead spent on refining game mechanics and adding content. During the last weeks of the games development, focus shifted more towards finishing up features, fixing bugs and improving the game experience.

3.3.1 Gameplay

During the development of the game, a decision was made early on to focus on the gameplay aspects of the game, rather than the technicalities of the flock simulation. A major motivator was to have fun during development and while playing the game, and the group consensus steered towards focusing on the fun factor of the game and its gameplay, over optimising for a maximum number of agents simulated at once. This gameplay driven development focus laid the foundation for the decision making detailed in later parts of this section.

3.3.2 World Generation

This game is done in a 3D environment to increase the complexity of the game's visual aspect, which is important for a simulation-based game without continuous interaction. Furthermore, it is implemented in the form of procedurally generated terrain with mountains and valleys that form and blend seamlessly. Players can control the size of the map and the frequency of mountains at the beginning of the game, see Figure 4.2. The map is then generated with the use of perlin noise and octaves as described in 2.4, and finally, surrounding walls are added to restrict the playing field, see Figure 3.5.

As this project aims to get the player to experience and play with the flocking behaviour, it was decided that a certain amount of freedom was required to achieve a player experience that facilitated creativity. By suggesting restrictions of world elements and resources, as seen in 2.5.5, the players will start by experiencing the game in the way that was intended and balanced for, while still allowing the player to experiment with the environment if they would like to. To enforce the restrictions on the players, a linear gameplay style could be used to unlock options to change the world, but this was decided against to accommodate the audience of players that plays solely for experimentation.

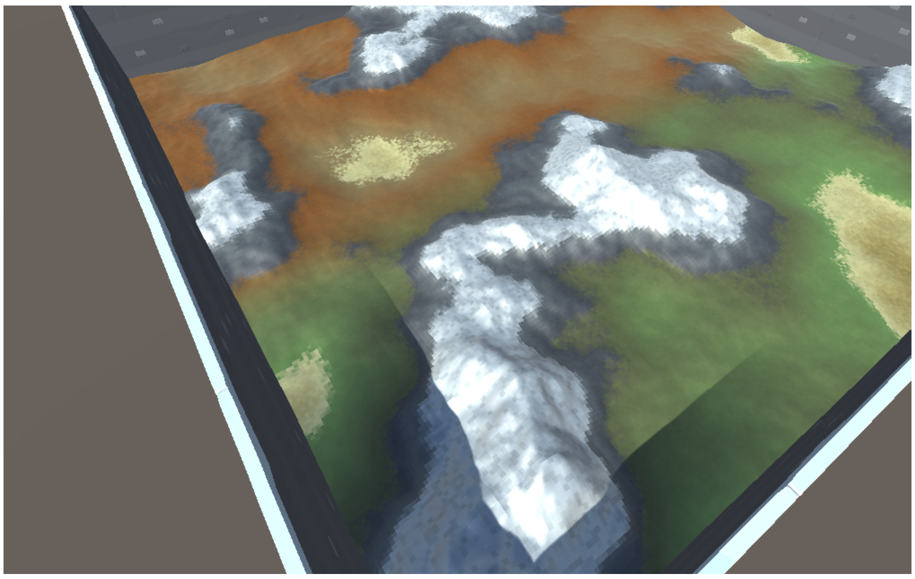


Figure 3.5: The procedurally generated map, complete with walls, mountains, valleys and plains

3.3.3 Sound Effects

To make the game more interactive it was decided that sound effects would be used to entice immersion in the battles. Unity has great support for sound effects through the AudioListener [47] and AudioSource [48] components that can be added to game objects. The AudioListener component is often attached to the player's camera, which is also something that was done for this project. By attaching the listener to the camera, the sounds heard in the game world feel more realistic e.g. they fade when they are further away from the camera, and is heard from the correct direction relative to the player's view. The AudioSource component was added to all of the unit prefabs, such that when a unit want to play a sound effect, Unity can play the sound effect from the location of that unit in the game world.

While playtesting it quickly became apparent that having hundreds of units clash together in battle, without any restriction on the number of concurrent sound effects playing, the noise was deafening. To tackle this problem, a custom AudioManager class was created, to route all playback requests through, such that it can keep track of the number of sound effects currently being played. To ensure coordinated access to the audio resources, the AudioManager class was implemented as a singleton [49].

When a unit wants to play a sound effect, it sends a request to the AudioManager class, and with it, a reference to its AudioSource component. The AudioManager first checks if the AudioSource is already playing a sound effect, and if it is, the request will be denied. If however, the AudioSource is ready, the AudioManager plays a sound effect, using the given AudioSource, and increments a counter, keeping track of the currently playing sound effects. As the counter is incremented, a coroutine [50] is started that waits for the duration of the sound effect, before decrementing the counter.

As some of the sound effects used in the game were quite short (less than a second), an offset was added to the coroutine waiting duration to have a minimum waiting period between playbacks on the same AudioSource. This allowed for more audio sources to be allowed to play a sound effect, resulting in spread out sound effect playback on the battlefield, making it feel more realistic as more units get to play their sound effects. By restricting the concurrent sound effects playing at once, and limiting the playback rate on the same AudioSource, battles felt more in line with the desired user experience.

3.3.4 Animations

Animations were implemented using unity compatible animations from Mixamo, see Appendix A.3. These animations were placed into an animations controller and integrated to work with the assets already used in the game. The base of the torso is used as an anchor point to the physics representation meaning that if the upper body is moving in the animation, it is instead the legs that will be moving in-game which would look unnatural. Because of this animations with not to much upper body movement where selected.

Furthermore, four different movement animations were selected to fit four different movement speeds: walking, jogging, running, and fast running, as can be seen in Figure 3.6. This was to ensure that the movement of the feet of the unit was travelling at the same speed as the ground beneath them. To make this even more precise new variations of the different movement animations were created with the speeds slightly altered to enable even greater speed matching. After observing that some units would switch between different animations very frequently causing them to stutter, a new system for delaying transitions was added. This system made it so that the units have to exceed the speed needed to transition to the next animations by a certain percentage. The same was added for when a unit is decreasing in speed needing them to reach a certain percentage lower than the threshold.

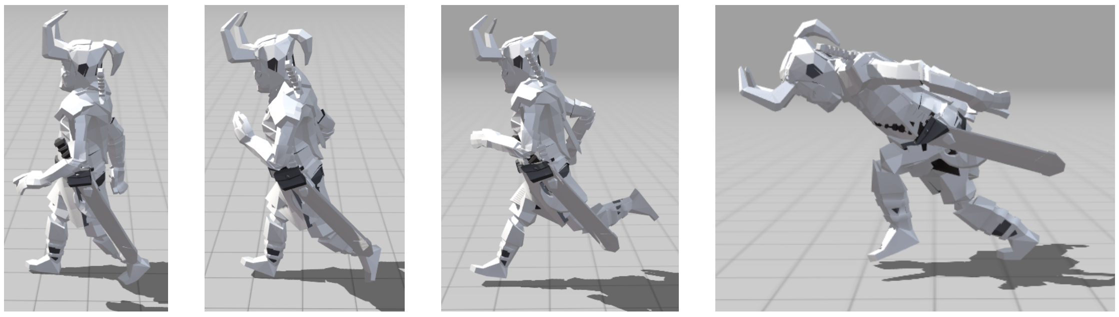

(a) Walking **(b)** Jogging **(c)** Running **(d)** Fast running

Figure 3.6: Shows the four different movement animations used in the game.

Other animations for fighting and idling were also added, see Figure 3.7. Different fighting animations for archers and melee units were used and a system was implemented that created a cooldown for the attack animations so that they were not triggered too often. The idle animation was added and set to trigger under a very low speed so that soldiers barely moving had a natural resting pose. Smooth transitions between animations were also added to make switches of animations during runtime unnoticeable.

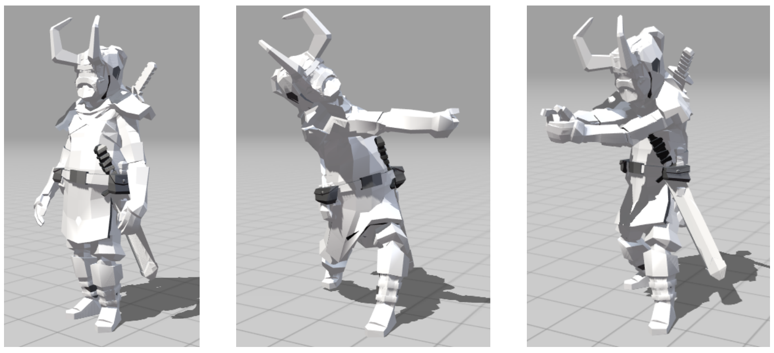

(a) Idle (b) Melee attack (c) Range attack

Figure 3.7: Animations for idle, melee attack and range attack.

3.3.5 Camera

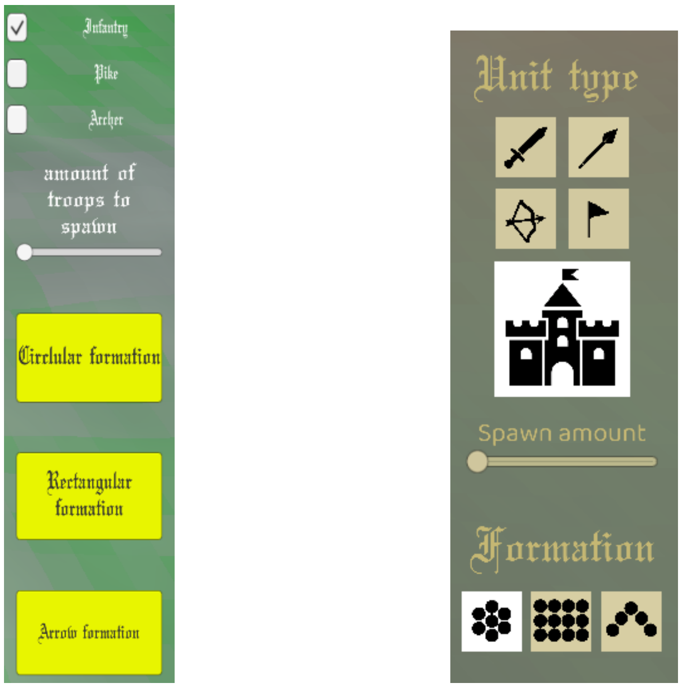
When starting development, proper camera controls were deemed essential. A static camera overlooking the battlefield would look stale and the players would not feel as involved in the game. It would also have lessened the effects that the animations would have as the player would barely be able to see the units, especially if the players would choose a larger playing area.

The main idea with the camera controls was to have a focal point that the camera is always pointing at that the player controls instead of controlling the camera directly. This approach makes the controls feel much more natural, especially when zooming and rotating. To speed up the development process the camera started of in an orthographic perspective as the features that were chosen to be implemented were much easier to implement with those settings and it would not have taken long to convert everything for the perspective camera. The features that were implemented were; zooming, rotating, movement in a 2D plane, and simple speed adjustment. All but the speed adjustment functionality were later added as mouse controls as well.

3.3.6 Interface

When implementing the interface, a stylised simplistic design was preferred, with buttons having pictures of available unit formations, and unit types, instead of text and sliders where possible. Unit costs and available money are displayed by a text that updates as the player places units on the battlefield. The focus of the design is to require as little effort as possible from the player for them to do what they want.

This includes minimum amounts of button presses, reducing the switching needed between mouse and keyboard, and is also a reason why buttons opted to use images. An additional aim with the use of images on the buttons is to give an intuitive idea of what the buttons do at a glance, while simultaneously reducing the amount of text that needs to be processed by the user.



(a) Old **(b)** Updated

Figure 3.8: The difference in readability between the two renditions of the interfaces is significant. Further updates in regards to size and colour also help.

During the simulation phase, the interface displays how many units are alive in each army, and how many have been killed, see Figure 4.7. The user interface is kept simple to avoid cluttering the screen with superfluous information, so the user can stay focused on watching the battle unfold. When the game ends, and the end screen is shown, additional information is displayed as there is no concern that it will take away from the focus of the player.

3.3.7 Game Balance

During game development, the question of game balance comes up naturally: Who gets to go first? How much should a unit cost? How much damage does a pikeman do? Some initial numbers were tested, with units given different costs based on their perceived usefulness. After playtesting the numbers would be tweaked further. Some of the values looked at were amount of health points, and damage dealt. The goal was to make sure that combat did not feel unfair e.g. no unit-type greatly outshines any other, and that units lived for a certain amount of time, such that battles did not end too quickly, or last for too long. This iterative process was repeated until the group felt the values gave the intended game experience.

3.4 User Testing

The game was tested continuously throughout the development process by the group members. An approval review was required from at least one other group member, before allowing a new feature to be merged into the master branch, as described in 3.1.5. The reviews included both code review, functionality and appearance, as well as design work tested on many different screen sizes.

Testing with external users was initially planned, but could not be done properly due to unforeseen external circumstances. Since user testing is a time-consuming process, the original plan was to conduct only a simple version where friends of the group would try out the game before giving feedback. The increased difficulty of conducting the tests in a satisfying manner led the group to settle with mainly internal testing for this project.

4
Results

This section contains the results of the project. First, the gameplay is described and after that, the specifications for the flocking behaviour is presented. The last sections describe the work done on optimising the game and show the procedurally generated terrain.

4.1 Gameplay

The gameplay consists of four different scenes, which can be seen in Figure 4.1. The game starts at the startup scene, which leads to setup scenes for each player. The players set up their side of the playing field one at a time, where player 1 goes first followed by player 2. When both players are done, the game moves on to the simulation scene where the battle is simulated. If there is a winner within a certain time, the game moves on to the end scene. From the end scene, the player can choose to start a new game. If the game did not end within the time limit, a new turn is started and the game returns to the setup scenes, keeping all resources and troops. In the consecutive turns the players get money in accordance with the number of enemies killed as well as a hefty default amount and has the option to spawn new troops. Spawning troops is limited to a short range around their castle or through the built-in spawning queue within the castle.

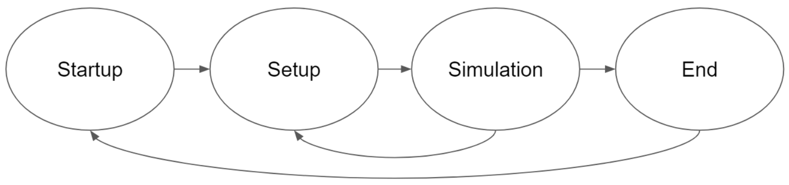


Figure 4.1: The four stages of the game

4.1.1 Startup Scene

Initially, players are presented with a startup scene, as shown in Figure 4.2. On the left, there are options to adjust map size, mountain density, input-seed for terrain generation, and amount of starting money. When the players have made their choices, they can press Start to move on to the next scene, the setup scene for player 1.

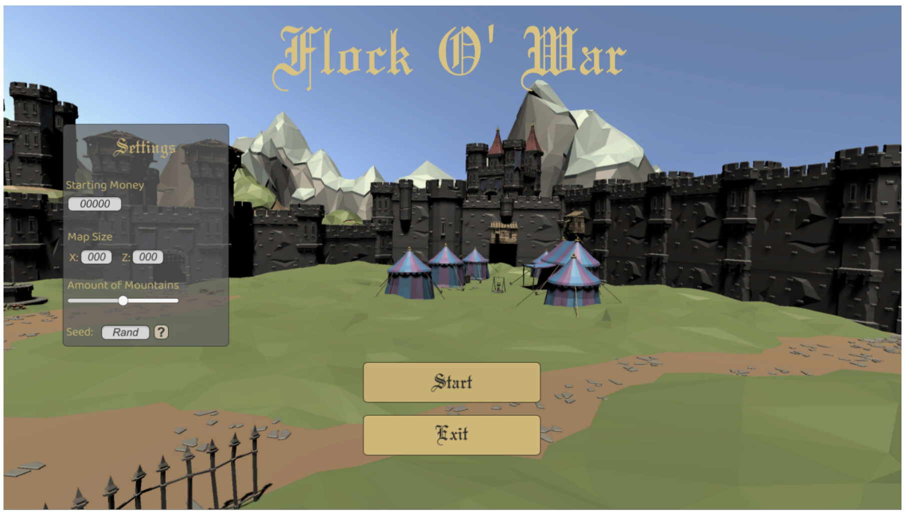


Figure 4.2: The menu scene of the game, with options to customise certain aspects before starting the simulation. The background image is taken from the example scene provided in the Polygon Knights Pack (see Appendix A.2 for references to assets used in the game).

4.1.2 Setup Scenes

The setup scene for player 1 (Blue team) is shown in Figure 4.3 and the setup scene for player 2 (Red team) looks the same. As can be seen, the battlefield is divided into two areas, with red and green ground. The blue team can only place its units in the green area and the red team can only place in the red area. The first time the setup scene is loaded, the terrain is also generated based on the seed provided in the menu scene. More details on that can be found in Section 4.4.

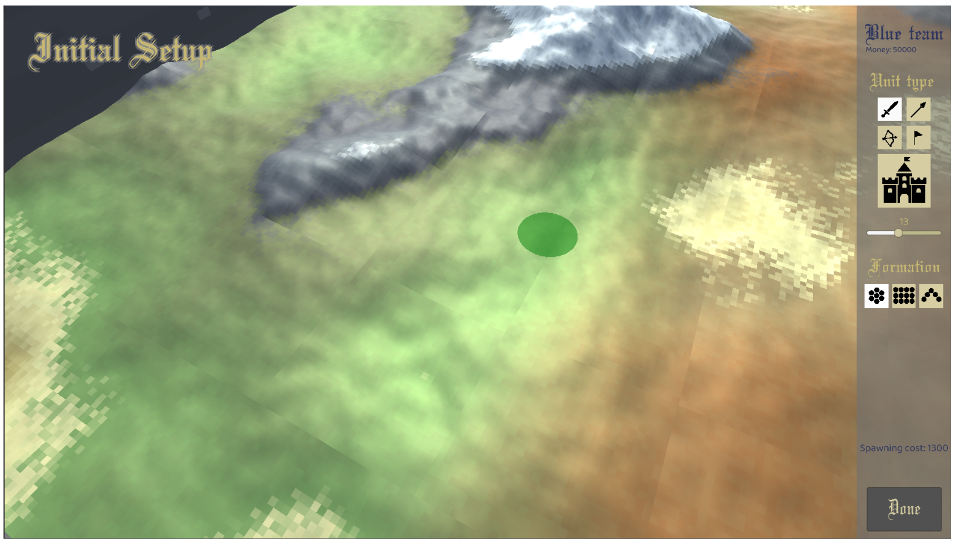


Figure 4.3: The setup scene where the players can purchase and place their troops and castles. See Appendix A.2 for references to assets used in the game.

4.1.3 Unit Spawning

During the setup stage, the players can use the provided money to purchase different types of units, by selecting the corresponding button located at the top right of the screen. To adjust the number of units purchased at a time, the slider can be used, and the formation in which the units are placed is chosen using the formation buttons.

When this stage is initialised, a green indicator area that follows the cursor will appear, see Figure 4.4, with its size and form depending on the number of troops and formation chosen, respectively. This area is used to visualise to the player approximately where the troops will be spawned if the player chooses to do so. The purchase is then done by left-clicking on the playable battlefield at the desired spawn location, which is calculated using Unity's Raycast feature [51]. It is also possible to hold left click and drag to continuously place units. If the player tries to place a unit outside their playing area or the wall, the unit will not be placed, an error message will be shown, and the player will not be deducted money for it. Similarly, if a player tries to spawn a troop on a location that already has a troop, the spawning will be cancelled.

If, for some reason, a player regrets a troop placement, holding right-click will enable a deletion indicator area, which can be resized by using the mouse scroll wheel. This area follows the mouse cursor and can be used to delete troops that are inside of it. This will also refund the proper amount of money to the player. A unit re-position functionality was also considered, but ultimately not implemented, since the deletion mechanic was considered more straightforward.

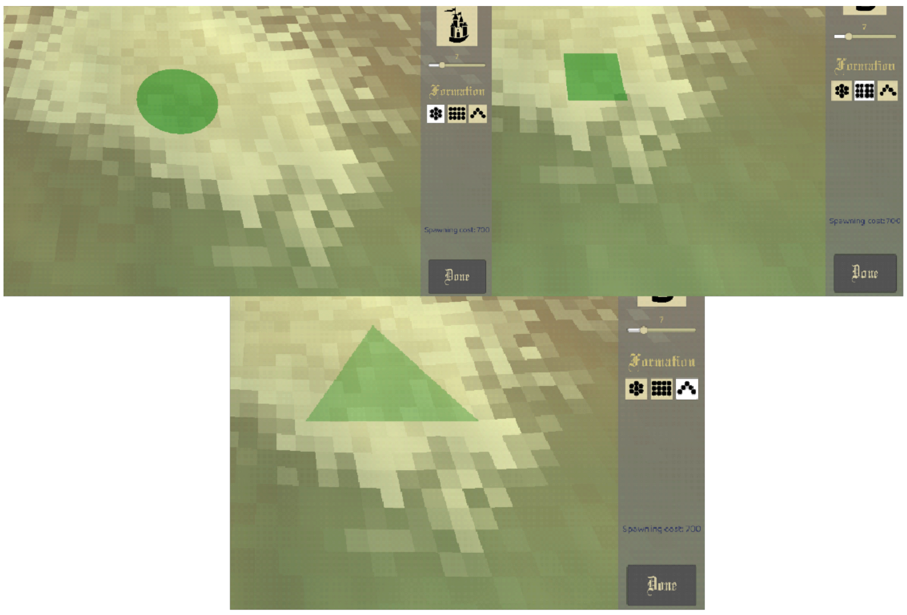


Figure 4.4: The spawning indicators for each formation

4.1.4 Unit Types

The different types of units and their properties are shown in Table 4.1. There are five types of units in the game, each with different specifications, and it is up to the player to choose how many of each unit to purchase and how to place them to win the game. The units also have a behaviour, determining how they move in the game. The composition of the behaviours is further explained in Section 4.2.

The basic units are Infantry, Pikemen, and Gunmen, which all have similar behaviour, but different values for health, damage, cost and reach. The scout is a more special unit with the purpose to scout the battlefield and find where the enemy army and castle are located. See Figure 4.5 and 4.6 for the visual appearance of these units. The castle also has its specific purpose, being the base camp of the army. It can not move or attack, but if it is destroyed the other team wins. All teams are required to place at least one castle, which is why the first castle has a cost of 0 (otherwise 3000).

Table 4.1: An overview of unit types and their properties.

Unit	Health	Damage	Cost	Reach	Behaviour
Infantry	200	40	100	1.5	Default
Pikemen	250	30	150	2.5	Default
Gunmen	100	20	300	8	Default
Scouts	50	1	300	1	Scout
Castle	1000	0	0/3000	0	Nothing

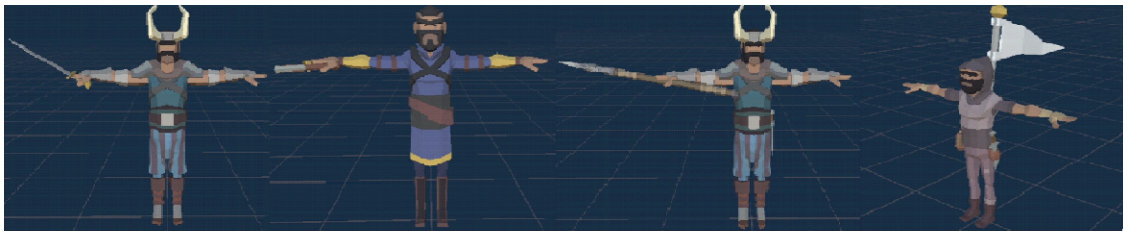


Figure 4.5: The units for the first army (excluding castle), from left to right: Infantry, Gunmen, Pikemen, Scout

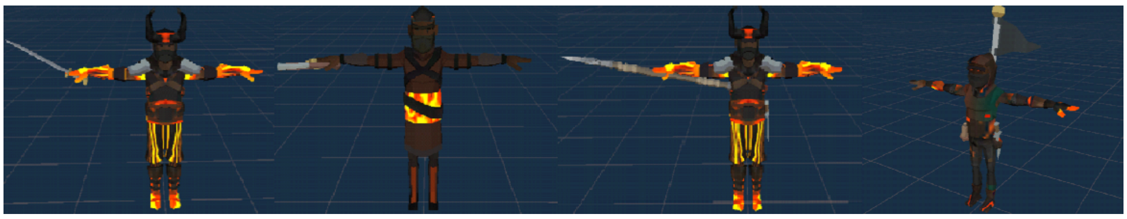
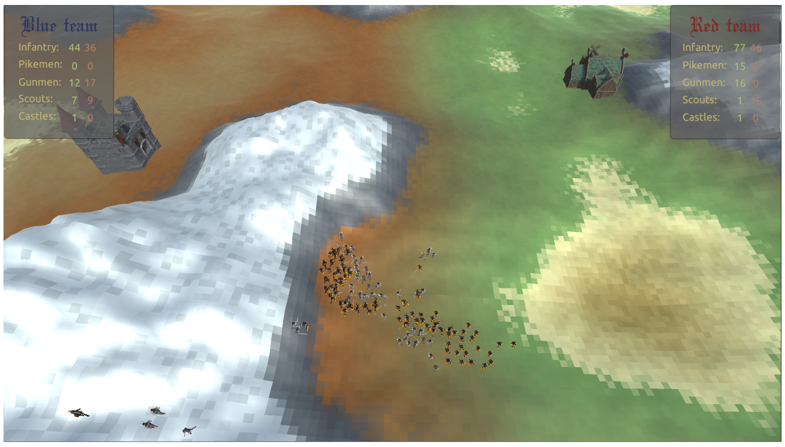


Figure 4.6: The units for the second army (excluding castle), from left to right: Infantry, Gunmen, Pikemen, Scout

4.1.5 Simulation Scene

When both players are satisfied with the placements of their troops, they are brought to the simulation scene, shown in Figure 4.7. During this stage, the player armies will be moving around the battlefield according to their behaviours, flocking, attacking, and fleeing based on the immediate situation. The status for each team is shown in the upper corners, where the number of living and dead units of each unit type is visible. The simulation continues until there is a winner or the time limit has passed. If the time limit has passed, a new turn starts and the players can choose to place more units. To win, one of the teams has to eliminate all of the enemy units or destroy all enemy castles.



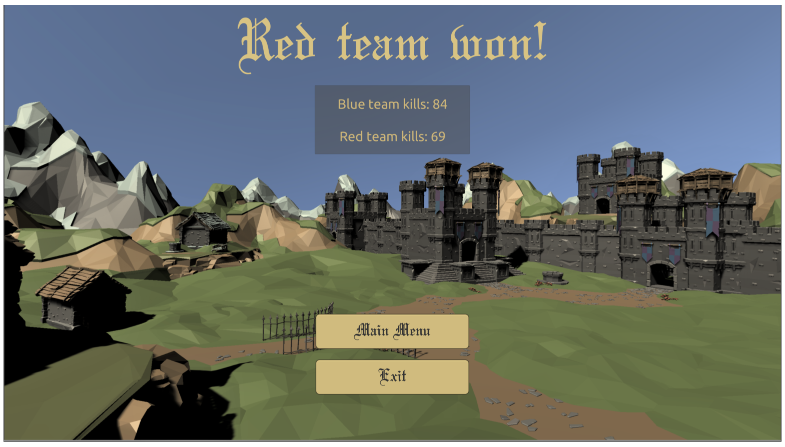
Blue team

Infantry:	44	36
Pikemen:	0	0
Gunmen:	12	17
Scouts:	7	9
Castles:	1	0

Red team

Infantry:	77	46
Pikemen:	15	6
Gunmen:	16	0
Scouts:	1	15
Castles:	1	0

Figure 4.7: The simulation scene, where the two castles are visible together with fighting troops.



Red team won!

Blue team kills: 84

Red team kills: 69

Figure 4.8: The end scene of the game, which shows a summary of the game that was just played. The background image is a screenshot from the example scene provided in the Polygon Knight Package (see Appendix A.2 for reference).

4.1.6 End Scene

When a player has won the game it is brought to the last scene, shown in Figure 4.8. Here, the winner is announced, along with the kill count of both teams. The players are then given the option to either return to the main menu or to quit the game.

4.2 Flocking Behaviour

The movement of each unit in the simulation is determined by its assigned behaviour. The behaviour is a composition of many components responding to different situations. The composition behaviour is based on the theory described in Section 2.2.1 and the components are summarised together similarly as in Equation 2.1, where different components are summarised with weights and then normalised.

Table 4.2 shows the behaviours created for the simulation and how they are weighed in the composite behaviours. Currently, there are two composite behaviours, Default behaviour which is implemented by Infantry, Pikemen, and Gunmen, and Scout behaviour which is implemented by the Scout. Both composite behaviours consist of the same components, the only difference is how they are weighted when they are summarised. In particular, it can be observed that the scout is not motivated to avoid the enemy or tend to its own group which allows it to move more freely and discover new grounds.

Table 4.2: An overview of behaviours and how they are weighted in the respective composite behaviours, default behaviour and scout behaviour.

Component	Filter	Default behaviour weight	Scout behaviour weight
Alignment	Same flock	15	3
Attack	Other flock	-	-
Avoid obstacles	Obstacle	1000	9
Avoid other flock	Other flock	2	0
Avoidance	Same flock	90	3
Fight or flight	-	20	4
Cohesion	Same flock	10	8

Among the behaviours listed in Table 4.2, Alignment, Avoidance, and Steered cohesion are the three rules for flocking described by Reynolds (see Section 2.2). These components create basic flocking behaviour, resulting in the army moving together cohesively. However, when there is an enemy army present, additional components are required so that the troops are attracted to their allies and run away from the enemy. Furthermore, the units have to decide when it is a good idea to attack and when they should run away. This is handled by the fight or flight behaviour.

The fight or flight behaviour can decide, depending on the number of enemies, allies and where they are, to attack or retreat. This is done by creating a strength value for all allies and comparing that to the corresponding value for all enemies. When the units attack, they run towards the centre of the enemies. When they instead decide to run away, they run in a direction which is a combination of the vector directed away from the enemy and the vector directed towards the centre of their allies. Finally, since the battlefield is finite and enclosed by walls, the units need to know how to behave when they are close to a wall. This is regulated in the component Avoid Obstacles.

4.3 Optimisation

To keep the game running smoothly it was continually tested to make sure that no changes would break the game's performance. The minimum requirement is that the game stays above 30 frames per second (fps) in any feasible scenario when there is a significant amount of troops on the playing field. When each player has 100 or more troops, their flock is large enough to require strategy and planning, while smaller stray groups are still significant enough to have to be taken into account. Therefore a requirement of 30 fps when each player has 100 troops was decided to be the limit for when added complexity was too costly. For example, scouts had to be tweaked to only have increased range in the *Fight or flight* behaviour.

Table 4.3: An overview of the performance depending on the number of infantry units and if they are battling. The tests were done on the same computer (see Appendix B.1 for specifications) and they denote the fps with a density corresponding to the natural one achieved while flocking. 60+ frames are the max measured and everything above 30 fps was playable without significant disturbances in the experience.

Number of units (infantry only)	fps while flocking	fps while flocking & battling
600	20	10
400	30	15
300	45	20
200	60+	40
150	60+	50
100	60+	60+

The game is sufficiently optimised for the intended purpose. As shown in Table 4.3, 200 units can battle in a high-density scenario while staying above our fps limit. When playing the game with the default settings of 10000 starting money, it is infeasible to reach more than 400 units, despite the money awarded between rounds and the type of unit spawned. Furthermore, when spawning through consecutive turns, it is less likely that the flock is able to stay perfectly cohesive, and as such the density and the number of attacking troops is generally much lower than the testing scenario, resulting in even higher frame rates.

As mentioned in the method section 3.2.4, the flocking algorithm avoids quadratic computations, in regards to the number of units. Instead, the computations are linearly dependent on the number units and density, which yields significant improvements. However, the behaviours' complexity is large in the implementation and as such, the units' field of vision has been decreased, until the density is sufficiently small to make the algorithm computable in a reasonable time frame. This change results in flocks more commonly breaking apart, as smaller fluctuations in cohesiveness results in split up when their field of vision is limited to a few time their body lengths.

This also creates problems for ranged units, as they need to see the unit they are going to shoot at. Ranged units lose their purpose if an enemy can not be shot before it is a few body lengths away, as they will engage with enemies within melee range. Fortunately, the field of view is still significant enough that melee units consistently form a front line that protects their ranged units.

4.4 Procedurally Generated Terrain

The terrain in the game is generated procedurally using the seed given in the start scene of the game. This terrain is represented through a mesh with the height levels for the mesh being procedurally generated through Perlin noise to form mountains and valleys. Several octaves of Perlin noise are used in balance to form a more complex and natural-looking appearance. The colour of the mesh depends on the height level, where the deepest valleys are sand coloured, the upper parts of the mountains are white, while the lower parts of the mountains are dark grey. The ground level above the sand is green or red, which is used to clearly indicate to the players which team's side it is.

The height of the mountains can be regulated using the slider on the settings panel on the starting page. Figure 4.9 shows the difference in the terrain generated when the settings are at maximum and minimum mountain height, respectively. As can be seen, the change in the appearance of the terrain is dramatic, allowing for different styles of gameplay with different kinds of strategies.

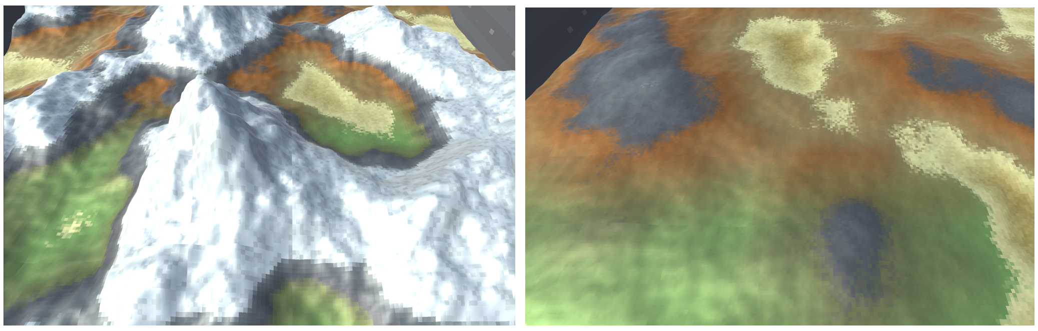


Figure 4.9: The resulting terrain when the number of mountains is set to high and low, respectively

5
Discussion

This chapter examines how entertaining the game is and how well the implemented flocking behaviour and procedural terrain contributes to that experience. Furthermore, possible improvement ideas are discussed in regards to the project's final state. It also contains a description of the project's contents in regards to ethics and conclusions of our thoughts regarding the results of the project.

5.1 Gameplay

The aim of the project was to create a game based on flocking behaviour and to see if Reynold's flocking rules could be used and extended to simulate the movement of two armies battling. The resulting product is a strategy game where the players choose among the available units to compose an army that can defeat the other player. When evaluating the game, an important criterion is whether the game is good or not. However, that is a subjective question and comes with many aspects such as: Is the game fun to play? Is the game fair and balanced in a proper way? Is it intuitive? Is it based on skill? Is it easy to understand?

5.1.1 Development Process

As the development of the game proceeded, it became obvious that implementing flocking behaviour was only a minor part of the development of a proper game that would be enjoyable and challenging to play. The flocking itself was not entertaining enough, and additional features would have to be added to create a playable game. Adding different unit types such as gunmen and pikemen created the possibility for the player to outline a strategy on how to compose their army. This created the need for a money system to have a boundary for the number of troops that could be created. However, the pacing was still an issue as the units kept wandering around aimlessly on the battlefield with no intention to find or kill the other army. This was, as is explained in Section 5.3, mostly due to the units having a very short field of vision, only reaching as far as a few body lengths away from them. This has partially been countered through optimising the flocking behaviour and introducing the scout and castle units. The castle provides an extra winning condition and a clear goal of where to go, while the scouts help to gather the army and bring them to the castle. To further increase the interactivity of the game, rounds were introduced, allowing the player to place defensive units against attackers.

What all these new features have in common is that they aim to make the game more entertaining and challenging. During the course of the project, it has become apparent that flocking by itself does not make a good game. Flocking is but a small detail which has to be combined with other features for a game to be fun to play. The rules, the winning conditions and the pacing are all crucial to keeping the player entertained.

Another important aspect is the user interaction and user experience. Although the design has not been the main priority of the project, it quickly became apparent that the feel and experience of the game was a large part of what makes the game entertaining. In the very beginning, the design was non-existent with no harmonised colour scheme, and the units were floating in a t-pose because there were no animations. To remedy this a simple design was created with a colour palette and background images matching the assets from the Polygon Packages (see Appendix A.2). In addition, animations and sounds were added to make the troops feel more alive. Although this was outside our main scope, it turned out to be a vital part in improving the precise interactivity and polish of the game.

5.1.2 Challenges

Balancing, as in deciding reasonable values for costs, health, damage, and all other parameters of the game has proven to be quite challenging. Since there are many possibilities when the player is placing their army it is difficult to test all possible combinations. This means that it is difficult to completely understand the impact of changing a value. To fully evaluate whether or not the game is balanced, extensive and thorough testing would be required. Also, it would have been beneficial to have a more clear design process of the game from the beginning to have a better understanding within the group on what parameters the game has and how they affect each other, especially when adding new features and units. For example, one interesting consequence of the design decisions was that the exposure of the castle was very dependent on how many mountains there are around the castle. Since the playing area is not symmetrical, it is often unfair with one side having a much better castle placement. In rare occasions, there would even exist a placement completely surrounded by mountains where no unit would ever reach.

Another challenge has been the navigation abilities of the units. One of the first issues which were discovered was the units not being able to leave lower parts of the map. This was partly because the units were too weak to overcome the incline of the terrain, but also because the units had no strong enough reason to move against the terrain. The first partial solution was the use of a stronger aerodynamic coefficient coupled with stronger acceleration. This made it so that the units could change speed quicker while still having a similar top speed. This essentially made the slope of the map less of a factor when it comes to the total acceleration of a unit, which in turn made them more able to move up slopes. An unfortunate side effect of this was that it made the units more twitchy as they where now able to change directions very fast, giving them an unnatural appearance. This was solved by adding a maximum change rate to the acceleration. This made the changes in a direction less dramatic, while still allowing a high enough acceleration to resist the slope of the map. Another solution that was added was the addition of angular drag. This made it so that the units could not turn faster than a certain rate, making them more determined when walking up slopes.

5.1.3 Suggested Improvements

A critical thing to add if we would like to distribute the game to other people would be to add instructions and ways to inform the player how the game work. As it is now, the game lacks instructions for the user. If a new player tries to play the game it will be hard, since no information is given to the player on how to play the game. There is no information about how the game works, what controls there are and the specifications of the different units.

Another suggested improvement would be to add more units to allow for more complex strategies and letting the player feel more in control. One suggestion is to add a commander unit. This unit would always know where the enemy castle is, and could thus lead the other units there even if it is beyond a mountain range. To decrease the impact on performance, the commander would be able to cycle through a list of castles removing the need for a very large field of vision. A new behaviour would have to be implemented to make units follow the commander more than other units. If the commander were to die it would leave the units lost again. Thus the commander would have to avoid enemy units, or at least have a very high number of health points so that it is unlikely to die. Since the game is designed in such a way that makes it easily extendable, adding new units would not require much effort.

It has also been discussed that the area where you may spawn units could dynamically change. One way of doing this would be to divide the playing board into squares and let a player spawn within a square if that player has the largest number of units within that square. This means that the player firstly could choose to reinforce an army heading for the enemy castle and secondly could redirect the army by making the new troops face the desired direction. It would not be game-breaking, as the enemy player would be able to counter by spawning soldiers in a nearby square. To make it more strategic and realistic, a requirement could be added where a player would need to control a line of connected squares to any place where he or she wishes to spawn units. This would force the attacking player to balance the troops properly to avoid a surrounding manoeuvre by the enemy. The turn duration would probably also have to be shortened so that the formation on the battlefield does not change too much between turns. This so that line breakthroughs and surrounding manoeuvres do not happen to easily as they can be countered in time. Combining this grid solution with the previous solution could also work to greater effect.

5.2 Flocking Behaviour

Another part of the aim of this project was to see if the basic flocking rules could be used as a foundation to simulate the movement of armies in a seemingly realistic way. This turned out to also be a subjective question since quantitative testing would require setting up studies of humans moving in groups, which is outside the time frame and scope of this project. This means that the evaluation could mainly be done qualitatively by discussing the look and feel of the flocking behaviour.

As mentioned previously in Section 5.1.1, the flocking behaviour itself needed to be adjusted to improve the gameplay aspects and create a more interesting simulation. In the end, components for avoiding the other flock, deciding whether to fight or flee and attacking were fundamental parts of creating a suitable behaviour for the game. In the end, our method follows the same idea as described in [20], where the basic rules of flocking were used to simulate large combats by introducing leaders (see Section 2.3). A notable difference is that the scouts do not directly communicate strategies or goals with their allies, but instead only communicate their intent through their alignment and velocity.

After the introduction of additional components in the behaviour as well as the scout unit, a decent behaviour of the armies has been achieved. The resulting flocking behaviour provides value to the game, in our opinion, and is not perceived as particularly unnatural. However, there are a few issues. Firstly, the most notable issue is that sometimes a smaller group breaks off from a larger group, which does not make sense since they should want to stick to the group. This is due to the field of view of each unit being too limited, meaning that they do not see the whole group if it is large enough. This limitation is set due to performance reasons and is further discussed in Section 5.3. Another issue is that scouts sometimes go in the wrong direction when there exist multiple forces opposing each other, misleading the whole army. This happens because the scouts have no mechanism for choosing between two armies and thus instead runs in the average direction. There have been a few suggested possible solutions, for example, the scout could be more drawn towards the centre of the map or the castle, but we worry that this might influence the flocking behaviour too much.

In conclusion, the resulting flocking behaviour works well for this game and contributes to the experience. Initially, there were concerns that the flocking rules could not be used on humans since, after all, humans are not particularly similar to birds or fish. However, for this specific purpose, the method turned out to work well and resulted in a behaviour that is well suited for the purpose.

5.3 Optimisation

As the development of the game progressed, it became increasingly apparent that optimisation could not be completely ignored. While measures were taken to improve the flocking algorithm's complexity, there were still a few unimplemented ideas which were hypothesised to boost the performance. Unfortunately, they were deemed too time-consuming and priority was instead placed on developing key gameplay features. It was argued that a fun but slightly worse-performing game is more appropriate for the project's aim than a boring but better-performing one. To accommodate for the non-optimal performance, the slider for determining the number of units spawned at a time was limited, as well as the default amount of money. This resulted in the player spawning fewer troops and as such keeping the game playable, while within our performance limit (stated in 4.3).

One idea to possibly improve the performance is to implement a dynamic field of vision, through adjusting a unit's field of vision to depend on the number of other units nearby. This would mean that units who are at the edge of the flock would be able to expand their fields of vision, making them better at spotting both enemies and friends. Additionally, the units at the centre of the flocks would be able to shrink their vision to only see their closest allies, resulting in fewer computations for the flock.

Another solution, which could have been combined with the previously mentioned one, is the introduction of skewed or angled fields of vision. If a unit is mostly looking forward, then it can see further without having too many neighbouring units within its field of vision. This would make the flock more sophisticated, as the front units, which possess the most amount of information, would have a greater impact on the group's movement. If for example, a larger enemy flock appears in front of the flock, the front units could decide to turn away, while ignoring the units behind them who are still unknowingly heading towards the enemy force. Further justification for this idea is that forward-oriented vision is more realistic, as it more closely mimics human vision, which is relevant in regards to the project's aim.

A notable problem with a forward-facing vision for the flock is the lack of other senses. As soon as a unit turns away from the enemy, it has instantly forgotten about it, as it has no memory. And since it also has no hearing, it has no way of noticing if someone is sneaking up on it from behind. Not even if that someone is an entire army. These issues were also observed when a simple version of this idea was tested in the game. The units would turn around to run from the enemy, but as soon as they faced away from the enemy they would stop turning away, believing the enemy was gone. In the end, it was thought to be too complex to try to implement a working version of the solution and the behaviour was ironically deemed more realistic with circular vision.

As it stands, the calculations for the game are still made in 3D space using three-dimensional vectors, despite most of the movement primarily being in two dimensions. Doing all the logic in 2D would provide a performance boost as all computation with the y-dimension would be removed. Though this would require some changed logic as the slope of the map would still need to be accounted for in some way. That is, it would look weird to simply project a 2D world on to a 3D world since the units would be able to walk up mountains just as easily as they would walk along flat ground. In the end, the change was deemed too time-consuming to test, especially since it was not clear if the decrease in complexity would outweigh the computations accounting for the terrain's slope.

The last optimisation that was considered was the storing of several computations that were repeated over multiple behaviours. Currently, the different behaviours do all of their calculations separately despite many of them being the same. Examples of these calculations include filtering of the neighbours' team and the distance to units. This could be optimised by instead doing the computations ahead of time and then sending the values on to the behaviours. It would require infrastructural changes to the code and make it less neat but would provide a considerable performance boost.

5.4 Procedural Terrain

The use of procedural terrain in the project has been a heavily discussed topic. At the beginning of the project it was adopted as a way of giving the game more replay value, but it has later on been questioned as ineffective use of time more needed for other parts of the project. As it stands the generation does provide some replay value as it enables for every game to be unique. On top of this, the ability to adjust allows for different kinds of gameplay depending on the roughness of the terrain.

Where it is lacking though is in the aesthetics. There were plans to add textures to the generated mesh to provide a more realistic appearance, but this was not prioritised as other parts of the project were deemed more important. The same reasoning was used for the plans to add trees and rocks. As the implementation is now it would have been more time-efficient to manually create a few better-looking maps. This would have had the additional benefit of freeing up time, which could have been used to make the game run better or to implement more units.

As it stands the current implementation of the procedural terrain does provide a good platform for future development and would, if the time necessary was available, probably have been extended to include textures, forests, and other structures. Within the given timeframe though, there is a strong argument for using premade maps instead.

5.5 Ethical Aspects

The use of flocking in a war scenario can be seen as a way of relating war to natural behaviour and thus arguing that human conflict is a natural phenomenon. But as the implementation used differs in the scenario to that of war, it would be far fetched to argue that it would be possible to do a proper adaptation. An important aspect of war is good communication and formations, while the flocking application is all about the lack of communication and the resulting behaviour that comes from this.

One might instead argue that the technology used could be reused to wage real wars. While this could be done to an extent, implementing the games´ flocking and adjusting it to be the behaviour for modern drones, it would not be better than an already existing implementation. The tactics used by the units in the game are simply too primitive. Additionally, a more centralised form of command is likely more effective than flocking, implying that even a highly advanced version of the game's implementation would be unsuitable for any military application.

Lastly, we have the issue that games containing violence might inspire similar actions in impressionable groups such as children. This has been an ongoing discussion for many years. This game is less gruesome than many similar games on the market and does not contain any blood. Neither does the player perform the actions of violence, as the gameplay is centred around building your army, not actual participation in the battles. Despite this, there is still violence in the game, which could have been avoided by for example displaying the soldiers as blobs of slime or children running around with toy swords and water-guns. But this could, in turn, have been seen as a way of promoting war as something fun. In the end, war is not used in the game for the sake of war, rather, it is used because it provides a good story basis for the typical player vs player strategy game.

5.6 Conclusion

Considering all aspects discussed above, it is clear that there are many things left to develop until the game is perfect. However, the group is still satisfied with how far we have come in the development process. Although some more work would be required for perfection, especially regarding optimisation, there are many features implemented in the game that contributes to an enjoyable experience, such as money, turns and castles. In addition, there are several types of units and it is possible to see the differences between them and use them in different combinations to obtain different results. Furthermore, the design and animations also look good in the group's opinion and contribute to creating the right feel of the game.

As it stands, the flocking behaviour also has some issues, but works well for this particular purpose and its extendable design means that it could easily be modified. Overall, the opinion of the group is that the game satisfies the scope of the project, which was to be entertaining. The game is re-playable enough that it could keep players occupied and entertained for a significant amount of time.

Bibliography

[1] I. L. Bajec, N. Zimic, and M. Mraz, “Simulating flocks on the wing: the fuzzy approach,” *Journal of Theoretical Biology*, vol. 233, no. 2, pp. 199 – 220, 2005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0022519304004746

[2] C. W. Reynolds, “Flocks, herds and schools: A distributed behavioral model,” *SIGGRAPH Comput. Graph.*, vol. 21, no. 4, p. 25–34, Aug. 1987. [Online]. Available: https://doi.org/10.1145/37402.37406

[3] L. Spector, J. Klein, C. Perry, and M. Feinstein, “Emergence of collective behavior in evolving populations of flying agents,” *Genetic Programming and Evolvable Machines*, vol. 6, pp. 111–125, 03 2005.

[4] J. K. Parrish, W. M. Hamner, and C. T. Prewitt, “Introduction – from individuals to aggregations: Unifying properties, global framework, and the holy grails of congregation,” in *Animal Groups in Three Dimensions: How Species Aggregate*, J. K. Parrish and W. M. Hamner, Eds. Cambridge University Press, 1997, p. 1–14.

[5] C. Reynolds, “Rule separation,” 1995, [Online; Accessed May 14, 2020], Public domain, via Wikimedia Commons. [Online]. Available: http://www.red3d.com/cwr/boids/

[6] ——, “Rule alignment,” 1995, [Online; Accessed May 14, 2020],Public domain, via Wikimedia Commons. [Online]. Available: http://www.red3d.com/cwr/boids/

[7] ——, “Rule cohesion,” 1995, [Online; Accessed May 14, 2020], Public domain, via Wikimedia Commons. [Online]. Available: http://www.red3d.com/cwr/boids/

[8] H. Hildenbrandt, C. Carere, and C. Hemelrijk, “Self-organized aerial displays of thousands of starlings: A model,” *Behavioral Ecology*, vol. 21, no. 6, pp. 1349–1359, 10 2010. [Online]. Available: https://doi.org/10.1093/beheco/arq149

[9] A. Okubo, “Dynamical aspects of animal grouping: Swarms, schools, flocks, and herds,” *Advances in Biophysics*, vol. 22, pp. 1 – 94, 1986. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0065227X86900031

[10] F. Heppner and U. Grenander, “A stochastic nonlinear model for coordinate bird flocks,” *The Ubiquity of Chaos, AAAS publication*, vol. 89, pp. 233–238, 01 1990.

[11] C. W. Reynolds, “An evolved, vision-based behavioral model of coordinated group motion,” in *From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior (SAB92).* Cambridge, MA: MIT Press, 1993, pp. 384–392.

[12] ——, “Evolution of obstacle avoidance behavior: Using noise to promote robust solutions,” in *Advances in Genetic Programming.* Cambridge, MA, USA: MIT Press, 1994, p. 221–241.

[13] L. Spector and J. Klein, “Evolutionary dynamics discovered via visualization in the breve simulation environment,” in *Workshop Proc. of ALife VIII.* Sydney: UNSW Press, 2002, pp. 163–170.

[14] C. Hartman and B. Benes, “Autonomous boids,” *Computer Animation and Virtual Worlds*, vol. 17, no. 3-4, pp. 199–206, 2006. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cav.123

[15] C. Delgado-Mata, J. Ibáñez-Martínez, S. Bee, R. Ruiz-Rodarte, and R. Aylett, “On the use of virtual animals with artificial fear in virtual environments,” *New Generation Comput.*, vol. 25, pp. 145–169, 2007. [Online]. Available: https://doi.org/10.1007/s00354-007-0009-5

[16] M. Belz, L. W. Pyritz, and M. Boos, “Spontaneous flocking in human groups,” *Behavioural Processes*, vol. 92, pp. 6 – 14, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0376635712001921

[17] S. Frey and R. L. Goldstone, “Cognitive mechanisms for human flocking dynamics,” *Journal of Computational Social Science*, vol. 1, no. 2, pp. 349–375, 2018. [Online]. Available: https://doi.org/10.1007/s42001-018-0017-x

[18] J. R. Dyer, C. C. Ioannou, L. J. Morrell, D. P. Croft, I. D. Couzin, D. A. Waters, and J. Krause, “Consensus decision making in human crowds,” *Animal Behaviour*, vol. 75, no. 2, pp. 461 – 470, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0003347207003764

[19] Z. Shen and S. Zhou, “Behavior representation and simulation for military operations on urbanized terrain,” *Simulation*, vol. 82, pp. 593–607, 09 2006.

[20] A. Boccardo, R. De Chiara, and V. Scarano, “Massive battle: Coordinated movement of autonomous agents,” *3D Advanced Media In Gaming And Simulation (3AMIGAS)*, pp. 35–42, 2009.

[21] G. Smith, “An analog history of procedural content generation,” *FDG*, 06 2015. [Online]. Available: http://www.fdg2015.org/papers/fdg2015_paper_19.pdf

[22] S. Gustavson, “Simplex noise demystified,” 03 2005. [Online]. Available: http://staffwww.itn.liu.se/~stegu/simplexnoise/simplexnoise.pdf

[23] Y. Scher, “Playing with perlin noise: Generating realistic archipelagos,” *Medium*, 11 2017. [Online]. Available: https://medium.com/@yvanscher/playing-with-perlin-noise-generating-realistic-archipelagos-b59f004d8401

[24] J. Schell, *The Art of Game Design: A Book of Lenses.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2008.

[25] G. Lindsay, “Game type and game genre,” *Google Scholar*, vol. 22, p. 2009, 01 2005.

[26] E. Adams, *Fundamentals of Game Design*, 3rd ed. USA: New Riders Publishing, 2014.

[27] F. Teale, “Game elements: Interaction,” *League of Gamemakers*, 07 2016.

[28] Unity real-time development platform. (Accessed: 30.03.2020). [Online]. Available: https://unity.com/

[29] Unreal engine. (Accessed: 07.02.2020). [Online]. Available: https://www.unrealengine.com/en-US/

[30] Unity - plans and pricing. (Accessed: 05.05.2020). [Online]. Available: https://store.unity.com/#plans-individual

[31] Unity asset store. (Accessed: 05.05.2020). [Online]. Available: https://assetstore.unity.com/

[32] Unity - manual: Scripting. (Accessed: 05.05.2020). [Online]. Available: https://docs.unity3d.com/Manual/ScriptingSection.html

[33] Unity - scripting api: Gameobject. (Accessed: 05.05.2020). [Online]. Available: https://docs.unity3d.com/ScriptReference/GameObject.html

[34] Visual studio ide. (Accessed: 05.05.2020). [Online]. Available: https://visualstudio.microsoft.com/

[35] git - version control system. (Accessed: 05.05.2020). [Online]. Available: https://git-scm.com/

[36] Github. (Accessed: 05.05.2020). [Online]. Available: https://github.com/

[37] Github - branches. (Accessed: 05.05.2020). [Online]. Available: https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/about-branches

[38] Github - pull requests. (Accessed: 05.05.2020). [Online]. Available: https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/about-pull-requests

[39] Where work happens | slack. (Accessed: 05.05.2020). [Online]. Available: https://slack.com/intl/en-se/

[40] Slack - threads. (Accessed: 05.05.2020). [Online]. Available: https://slack.com/intl/en-se/help/articles/115000769927-Use-threads-to-organize-discussions-

[41] Zoom. (Accessed: 05.05.2020). [Online]. Available: https://zoom.us/

[42] Slack - workspace. (Accessed: 05.05.2020). [Online]. Available: https://slack.com/intl/en-se/help/articles/212675257-Join-a-Slack-workspace

[43] Signal » home. (Accessed: 05.05.2020). [Online]. Available: https://signal.org/

[44] Board to bits games. playlist: Flocking algorithm in unity. (Accessed: 07.02.2020). [Online]. Available: https://www.youtube.com/watch?v=mjKINQigAE4&list=PL5KbKbJ6Gf99UlyIqzV1UpOzseyRn5H1d

[45] R. Finkel and J. Bentley, "Quad trees: A data structure for retrieval on composite keys." *Acta Inf.*, vol. 4, pp. 1–9, 03 1974.

[46] Unity - scripting api: Physics.overlapsphere. (Accessed: 30.03.2020). [Online]. Available: https://docs.unity3d.com/ScriptReference/Physics.OverlapSphere.html

[47] Unity - scripting api: Audiolistener. (Accessed: 05.05.2020). [Online]. Available: https://docs.unity3d.com/ScriptReference/AudioListener.html

[48] Unity - scripting api: Audiosource. (Accessed: 05.05.2020). [Online]. Available: https://docs.unity3d.com/ScriptReference/AudioSource.html

[49] Singleton design pattern. (Accessed: 05.05.2020). [Online]. Available: https://sourcemaking.com/design_patterns/singleton

[50] Unity - manual: Coroutines. (Accessed: 05.05.2020). [Online]. Available: https://docs.unity3d.com/Manual/Coroutines.html

[51] Unity - scripting api: Physics.raycast. (Accessed: 14.05.2020). [Online]. Available: https://docs.unity3d.com/ScriptReference/Physics.Raycast.html

A
Appendix 1: Assets

In this appendix, all the assets used in the game are listed.

A.1 Icons

The icons used for buttons in the game come from The Noun Project. They are licensed under the Creative Commons license and it is allowed to use them in any way wanted as long as the creator is referenced.

- **Sword icon:** `https://thenounproject.com/search/?q=sword&i=2444713`, ProSymbols US
- **Spear icon:** `https://thenounproject.com/search/?q=spear&i=819835`, Hamish
- **Bow icon:** `https://thenounproject.com/search/?q=bow&i=2851797`, Nick-lasR, AT
- **Flag icon:** `https://thenounproject.com/search/?q=flag&i=714884`, Maxim Kulikov
- **Castle icon:** `https://thenounproject.com/search/?q=castle&i=28149`, Joel McKinney

A.2 Asset Packages

A bundle with licenses to several assets packages was purchased. All units and their equipment were taken from these packages. The background images in the game were taken from screenshots in the sample scenes that came with the Knights package. The packages are listed below:

- **Polygon Adventure Pack:**
  `https://syntystore.com/products/polygon-adventure-pack`
- **Polygon Knights Pack:**
  `https://syntystore.com/products/polygon-knights-pack`
- **Polygon Vikings Pack:**
  `https://syntystore.com/products/polygon-vikings-pack`
- **Bundle**
  `https://www.humblebundle.com/software/best-of-polygon-game-dev`

A.3 Animations

Listed below are the animations used. The license covers the use of the animations as in-game figures, free or commercial.

- **walk**:
  `https://www.mixamo.com/#/?page=1&query=walk`
- **run**:
  `https://www.mixamo.com/#/?page=1&query=male+weighted+run`
- **fast run**:
  `https://www.mixamo.com/#/?page=1&query=running+leaning+back+or+forth`
- **naruto run**:
  `https://www.mixamo.com/#/?page=1&query=female+ninja+run`
- **idle**:
  `https://www.mixamo.com/#/?page=1&query=weight+shift+idle`
- **melee attack**:
  `https://www.mixamo.com/#/?page=1&query=right+to+left+attack+with+axe`
- **range attack**:
  `https://www.mixamo.com/#/?page=1&query=firing+a+gun`

A.4 Sounds

Listed below are the sounds used. All the sounds are under the CC-BY lisence.

- **Arrow (by EverHeat)**:
  `https://freesound.org/people/EverHeat/sounds/205563/`
- **Sword Slice 11 (by Black%20Snow)**
  `https://freesound.org/people/Black%20Snow/sounds/109420/`
- **Sword 4 (by CpawsMusic)**
  `https://freesound.org/people/CpawsMusic/sounds/437113/`
- **sword7 (by Streety)**
  `https://freesound.org/people/Streety/sounds/30248/`
- **06SWORD05 (by lostchocolatelab)**
  `https://freesound.org/people/lostchocolatelab/sounds/1452/`
- **Sword Clash and Slide (by FunWithSound)**
  `https://freesound.org/people/FunWithSound/sounds/361485/`
- **Epic Sword Clang 2 (by ethenchase7744)**
  `https://freesound.org/people/ethanchase7744/sounds/439538/`
- **Metal Ping1 (by timgormly)**
  `https://freesound.org/people/timgormly/sounds/170964/`
- **metal_rings_04 (by Department64)**
  `https://freesound.org/people/Department64/sounds/95275/`
- **Hardcore Kick (by taylorevanmcalister)**
  `https://freesound.org/people/taylorevanmcalister/sounds/223530/`
- **Metal hit with metal bar resonance (by jorickhoofd)**
  `https://freesound.org/people/jorickhoofd/sounds/160045/`

- **Stab, Metal Knife in Lettuce, D (by InspectorJ)**
  `https://freesound.org/people/InspectorJ/sounds/413493/`
- **Death Pain (by AlineAudio)**
  `https://freesound.org/people/AlineAudio/sounds/416838/`

B

Appendix 2: Performance Test

B.1 Computer Specifications for Performance Tests

Listed here is the computer specifications for performance tests.

- **Operating System** - Windows 10.0.18362 Build 18362 x64
- **Processor** - Intel(R) Core(TM) i7-6700k CPU @ 4.00 GHz, 4001 Mhz, 4 Cores, 8 Logical Processors
- **Physical Memory** - 16.0 GB
- **Graphics Card** - NVIDIA GeForce GTX 970