Bachelor Project

Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Computer Graphics and Interaction

Viktor Bobůrka

Supervisor: Ing. Ladislav Čmolík, Ph.D.
January 2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Bobůrka** Jméno: **Viktor** Osobní číslo: **474674**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**

Studijní program: **Otevřená informatika**

Specializace: **Počítačové hry a grafika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

3D tahová strategická hra

Název bakalářské práce anglicky:

3D turn-based strategy game

Pokyny pro vypracování:

Seznamte se s principy tvorby počítačových her. Proveďte analýzu herních principů používaných ve strategických počítačových hrách, zejména ve hrách tahových. Na základě analýzy navrhněte 3D tahovou strategickou počítačovou hru. Dále vytvořte modulární komponenty, ze kterých se budou skládat jednotlivé úrovně, jejich materiály a textury. Dle návrhu vytvořte s využitím modulárních komponent alespoň pět hratelných úrovní hry. Výslednou hru otestujte pomocí kvalitativních testů alespoň s šesti hráči.

Seznam doporučené literatury:

1) R. Koster. Theory of Fun for Game Design, 2nd edition, O'Reilly Media, 2013.
2) J. Schell. The Art of Game Design: A book of lenses. CRC Press, 2008.
3) B. L. Mitchell. Game Design Essentials, John Wiley & Sons, 2012.
4) S. Rogers. Level up! the Guide to Great Video Game Design, John Wiley & Sons, 2014.
5) E. De Nucci and A. Kramarzewski. Practical Game Design: Learn the art of game design through applicable skills and cutting-edge insights, Packt Publishing, 2018.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Ladislav Čmolík, Ph.D. Katedra počítačové grafiky a interakce

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **02.02.2022** Termín odevzdání bakalářské práce: _____

Platnost zadání bakalářské práce: **30.09.2023**

Ing. Ladislav Čmolík, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____ Datum převzetí zadání

_____ Podpis studenta

Acknowledgements

First and foremost, I would like to thank Ing. Ladislav Čmolík, Ph.D., who supervised this thesis and provided numerous consultations and much advice, for the opportunity to work on this thesis. I would also like to thank everyone who helped me test the early prototypes for their feedback, and the testers for taking their time to play this game.

Declaration

I declare that this is all my own work and that I have cited all the sources I have used in the bibliography.

Prague, May 19, 2022

Prohlašuji, že jsem zadanou práci vypracoval samostatně a že jsem uvedl všechnu použitou literaturu.

V Praze, 19. května 2022

Podpis:.....

Abstract

This thesis focuses on developing a turn-based strategy computer game. This process consists of game design, the implementation of game logic and the creation of modular components.

Keywords: Game development; Unity engine

Supervisor: Ing. Ladislav Čmolík, Ph.D.

Abstrakt

Tato práce se zabývá vývojem tahové strategické počítačové hry. Skládá se především z herního návrhu, implementace herní logiky a tvorby modulárních komponent.

Klíčová slova: Vývoj počítačových her; Unity engine

Překlad názvu: 3D tahová strategická hra

Contents

Figures

Chapter 1

Introduction

Computer games are a significant part of the entertainment industry, and by comparison of their global revenue, they might even become more influential than the film industry. They have been growing in popularity for the past 40 years, and according to DFC [Int21], there are currently an estimated three billion players worldwide. Whether computer games are an emerging art form or a new type of entertainment, they are massively popular with the general public. Therefore, it is worth studying how they are produced and learning to create them.
This thesis focuses on the process of creating a computer game. Due to its scope, it will cover a subset of computer games called tactical games. They are a very compelling genre of games and have a long and rich history. They have been invented as board games in ancient civilizations and continue to be popular today. Some have been played for several thousands of years. Others are, despite their long existence, still being reinvented and massively popular. Because of their variability and longevity, they make good material for further study.

Figure 1.1: Medieval chess pieces.[oS22]

The world tactical comes from the Greek word taktike (the art of arrangement), which reflects the historical function of the games. Some have even been used to educate military leaders and hone their abilities on the

battlefield, for example, the Prussian game Kriegsspiel. Other ones are still being played today and are getting mainstream attention, such as chess or go. What makes tactical games interesting is that they still act as battles, only they don't happen physically but intellectually. For example, Pirie [Mad19] claims chess has been viewed as a tool to demonstrate intellectual superiority over the Western world by the Soviet Union.
When considering the digital space, battles are still the primary motivation of tactical games. There, they don't have to happen between two humans but can be a challenge given to one player. Creating such a challenge is going to be the topic of this thesis.
In order to create a tactical game, existing games must be first analyzed. The second chapter of this thesis will focus precisely on that. I will explore what tactical games are, how they can be classified, and what makes them fun to play. Then I will look at examples of individual games and what I can learn from them. The third chapter will focus on designing a tactical game using the insight gained from the analysis. The design will partly stem from existing game principles and partly try to implement new mechanics.
In chapter four, the design will be analyzed from the technical perspective, and different approaches that will be suitable for the design will be examined. The algorithms needed in order to realize the game mechanics as well will be analyzed as the basics for a turn based game. Then, the creation of 3D assets will also be described and different approaches will be discussed.
In the fifth chapter, the implementation of the game will be covered. The creation of modular components that will be used in the game will also be documented. Finally, the fifth chapter will also contain testing of the game and its evaluation.

Chapter 2

Analysis

In this chapter, I will look into guidelines for creating a fun computer game. I will start with some general principles that can be used when making any game, and then I will analyze successful tactical games to see what they can teach us. Later, I will use these principles to fuel design decisions.

2.1 What makes a game fun?

Before analyzing games, one crucial question needs answering – what makes a game fun? Considering there isn't even a single definition of what a game is, it would be a difficult task to thoroughly define what it is about a game that makes it fun. While entire books are written on this topic, there is no consensus as it is a highly subjective matter. However, some attempts to answer it can give us insight into the mindset that can be helpful when trying to create a game.

2.2 Fun from mastery

Let's start with a general idea of what makes a game fun. According to Raph Koster [Kos05], "fun from games arises out of mastery. It arises out of comprehension. It is the act of solving puzzles that makes games fun. In other words, learning is the drug." Most things you do in games can be derived from a desire to learn. This desire can be applied to learning about the story or the world of the game. However, it is of most importance when mastering the underlying game mechanics.
The point at which a game ends is usually when the player runs out of things to learn. Once they can predict what happens at every point in the game, it becomes predictable and boring. Game designers know this, of course, and they know helpful tools to prolong the lifetime of a game.
There are several ways to let the player learn for as long as possible. One possibility is to use levels. Most games have different levels that the player has to go through because the presented challenge can change significantly with variations in the player's surroundings. Sometimes, they randomly generate content, hoping that ever-changing playing fields will keep the player engaged

indefinitely.
Some games unveil their mechanics one at a time so the player can keep learning for a longer time. It is a quite common strategy across game genres. For example, in platformers, the player can discover new possibilities of movement and can be presented with new challenges. In role-playing games, the player can unlock new abilities to interact with the world in a novel way. In tactical games, new units can be introduced, and the player has to learn how to play with or against them.
Perhaps the oldest and most effective example is multiplayer games. There are as many different challenges in multiplayer as there are players. Then, if a game has a vast possibility space, people can keep playing it for centuries. The most known examples of this are chess or go. They both feature a so-called combinatorial explosion of possibilities. It is the effect of having, for example, twenty possible first moves in chess, but after four moves, there are 197 742 possible layouts. Having this many possibilities makes the game too complicated to be solved. Therefore, you can spend your whole life learning it.

Figure 2.1: A: Visual style of the game Manifold Garden [Chy], B: Enticing discovery in Zelda: BotW [IGD22]

2.3 Other sources of fun

While most games are primarily focused on mastery of the game mechanics, there are also ways to classify fun into different aspects. Those can help us clarify which events the game can focus on to deliver a coherent experience to the player. According to Robert Hunicke [RH], there are eight sources of fun in gameplay:

1. Sensation is the pleasure of the senses. Traditionally, the source can be a striking visual style or a satisfying sound effect. Notable examples are Manifold Garden, Mirror's Edge, and Mario Odyssey.

2. Fantasy can let you escape into the game's make-believe world if it's captivating and more pleasing than the real world. Famous examples are Assasin's Creed and Final Fantasy.

3. Narrative can be understood as the story told by the game, but some games utilize a so-called emergent narrative. This kind of narrative can emerge naturally and directly from the game mechanics. Some games even focus almost solely on the story, for example, Firewatch and What Remains of Edith Finch. Good examples of emergent narratives are Crusader Kings or the Far Cry series.

4. We have already considered challenge as a source of fun in games. There the pleasure comes from overcoming obstacles or mastering the game mechanics. Almost all games utilize this source, but some games focus on it more than others, such as Spelunky or Counter-Strike.

5. Fellowship is a source that utilizes other players and social structures. It is very commonly drawn from board games but has its place in many multiplayer games, too. Co-op games like the board game Pandemic, or It Takes Two utilize this. Many MMORPGs also work with this idea on a much larger scale, such as World of Warcraft.

6. When discussing discovery, what is usually meant is exploring the game's world. The pleasure comes from finding places of interest carefully placed by the level designer and generally containing some rewards or easter eggs. There are many examples, such as Zelda: Breath of the Wild, Outer Wilds, and Subnautica.

7. Expression is fun through self-discovery. It means the player can adjust the game world to their own image. It is tough to achieve it in a game with a narrative (unless it is very reactive like in Dungeons and Dragons), but a familiar feeling in sandbox games like Minecraft.

8. At last, submission is a complete immersion in a game. It happens when the player doesn't consider it a single experience but rather an ongoing hobby. Some examples include collecting games like Warhammer: 40 000 or multiplayer pastime games like League of Legends.

2.4 Fun in tactical games

Tactical games are a highly analytic genre. The player spends most of the time weighing different moves and calculating outcomes. This leads to some sources of fun being more viable than others. While tactical games can utilize multiple of those sources, I think the most important one is challenge. Tactical games revolve around carefully looking at your options and developing the best plan for a given situation. The pleasure, then, comes either from the player seeing their plan succeed or from learning how to be more successful in the future.
However, the player's plan can be flawed or become outdated, and they have to invent a new one. This dynamic creates an ebb and flow to the game, and the player has to change their plan every couple of turns. The number of turns the player can make before changing their strategy varies significantly

from game to game. For example, in tic tac toe, you can develop an optimal strategy and then don't have to change it at all (which is why it is trivial), but chess grandmasters can think twenty moves ahead [Mos19], and they still might not make the best move. There is no optimal value for how far ahead the player should plan as it changes with their goal, the difficulty of the challenge, and their skill. The only thing we can rule out are the extremes – if there are too few options, you can effortlessly solve the challenge in the first turn, and the rest would be trivial and uninteresting. If there are too many options and you can't predict the outcome, the game is confusing instead of fun [Kos05].
Other sources of fun can also appear in tactical games, such as sensation from visuals and sounds, narrative that can be either written or emergent, discovery of the world map, or immersion in the game.
Fun from sensation can come together when the visuals or sounds of the game feel satisfying to the player. The visuals include visually pleasing 3D models and animations, which can go a long way if the player has time to admire them. Aside from that, many visual effects can be used to heighten the enjoyment. However, games need strong art direction to utilize this source of fun, and it is challenging, if not impossible, to judge or quantify.
Sound design is also an essential aspect of a game, even though it often goes unnoticed. Most games feature a musical score, sometimes adapting to in-game events. Then, of course, there are sound effects of anything that happens on screen. Sounds, such as footsteps, gunfire, or background noise, help immerse the player and set the atmosphere of a setting.
Another substantial component in tactical games can be narrative. Most feature a linear or branching story, and others even manage to create an emergent narrative through what happens in its gameplay.

2.5 Game mechanics

There are several game mechanics frequently used in tactical games that I would like to analyze as I plan to use them when designing my game. Those include grid and cover systems, feedback loops, randomness and balancing.

2.5.1 Grid

A grid system is a crucial aspect of tactical games. It limits the movement of units to fit into a grid, which allows for clear and well defined boundaries in which the player can act.
There are several types of grids used in turn based games. The most common types are square grid and hexagonal grid. Both have some advantages and disadvantages which we will discuss.
Hexagonal grids, such as one used in the game series Sid Meier's Civilization, have an obvious advantage of universality. That means that every tile is connected to all neighbor tiles in the same way, meaning along one edge. This means that the intended distance from each tile to all neighbor tiles is

the same as the actual distance, and there is very little room for ambiguity. Surprisingly, implementation of hexagonal grids is not complicated and is very similar to implementation of square grids.
Square grids, on the other hand, might be a bit easier to understand for new players, as they mimic grids of popular board games such as chess.
However, they also present a dilemma. The culprit is that squares connect to four other squares via their edges, but to four more squares via their vertices. This means that there are multiple possibilities of what a square's neighbors are and what are the distances to them. It is up to the designer, then, to choose the proper use of a square grid and can give them more options in their approach.
Only using the edges as connections between squares and setting the distance between them to one gives us the Manhattan distance. However, this creates a dissonance between the distance in the game and distances in real life. For example, the distance between squares 1,1 and 2,2 would be two, but in real life it would be around 1.4.
This problem can be fixed by including the diagonal neighbors as well and setting the diagonal distance to 1.4, but then it might be challenging for players to accurately estimate distances on the grid without having to do decimal calculations.
This leads us to the conclusion that the type of grid should be selected based on the situation. In some cases, it is more important that distances and tiles feel more representative of the real world, in others, clarity and breadth of designer choice is more desired.

2.5.2 Cover systems

A cover system comes out very naturally with a grid system. When a game features a tile system, some tiles (or even borders between tiles) can be hidden behind. When a unit hides behind such a cover tile, it can be protected from enemy attacks. That usually means that they will receive less damage, or that they are more difficult to hit. Of course, a single cover tile doesn't have to (and sometimes shouldn't) protect a unit from all sides. As in real life, if you hide behind a rock, people can still see you if they are behind the rock, too.
The effect of a cover system is that it matters much more how a player places their units. For each of their units, they have to balance its cover, ability to attack enemies outside of cover, as well as targets they can reach with their attacks, proximity to other units, and so on. This creates a broader variety of choice for the player, which hopefully presents the player with more interesting choices and dilemmas. They can, for example, run outside of cover and risk receiving a lot of damage, in exchange for getting to a better spot or hurting the enemy units more.

2.5.3 Feedback loops

Feedback loops are invisible parts of games that can stem from other mechanics, rather than being a mechanic on their own. They do, however, influence the overall experience of the game greatly.
Feedback loops come in two forms – positive and negative. Positive feedback loops cause that the more something happens, the more it will happen in the future. Negative feedback loops mean that the more something happens, the less it will happen in the future.
For example, positive feedback loops can occur in team based first person shooters. If one team manages to kill a player from the enemy team, it gains a number advantage which will make it more probable that it will kill more enemies in the future.
Negative feedback loops can be found, for example, in most racing games. There, it stops the player in the first spot from getting too ahead by slowing them down or giving them worse rewards for certain actions. This makes it harder for a player to stay in the first place and motivates players who are behind to keep trying to catch up.
In other cases, feedback loops are left out. In tennis, there is no feedback loop that would give either player an advantage or disadvantage after scoring a point (at least if we disregard the psychological effect). That makes it more fair in each game and gives the players time to change their strategies without being pressured by artificial disadvantages.
The pitfall of feedback loops lies in their strength. If a positive feedback loop is too strong, it might get out of control and give the starting advantage much more impact than it should have. For example, if a hockey team scoring a goal led to the other team losing one player, it would probably be overly difficult to catch up. On the other hand, if a negative feedback loop is too strong, any achievement can feel insubstantial.

2.5.4 Randomness

Randomness is commonplace in video games. It ranges from basic elements like card drawing and hit chances, all the way to sophisticated systems like procedural generations or simulations. It is a very broad topic that is present in most games. Therefore, the goals of randomness can differ greatly.
In appearances like card drawing, the goal is to present the player with a previously unknown situation that they have to react to. This new situation leads to two things. Firstly, the player has to react to the cards they drew. The reaction is that they reevaluate their position and future strategy. During that, they engage with the game more and have to become adaptable.
Secondly, if a game is influenced by card drawing, there is almost an infinite amount of different possible matches. In poker, for example, there are 2,598,960 possible five card combinations. With this many combinations, it rarely happens that any two games are the same. This makes things easier for the developer as they don't have to come up with so many different situations

themselves to keep the player engaged.
An extreme case of this strategy is procedural generation. Games like No Man's Sky boast about providing 18 quintillion planets for the player to explore. There are so many that most of them will probably never be visited by any player. However, in more modest cases, the generated map can keep players engaged for a reasonable amount of time and at the same time save time for level designers who don't have to place every object by hand.
However, there is another kind of randomness, so called output randomness. It influences an action after it has been taken. A very simple example is when the game tells you that your attack can deal four to six damage. But you only learn how much you will actually deal after you attack. This type of randomness makes it more difficult to plan ahead as you never fully know what will become of your actions. With this in mind, there can be a risk and reward scheme in play, which will allow the player to either play conservatively or take a chance when they feel like it. Nevertheless, it has to be balanced properly in order to not overrule any strategic thought.
The amount of randomness in a game influences the overall experience and can override other elements if it is applied too much. Therefore, it has to be carefully balanced. Schell describes it as follows: "Risk and randomness are like spices. A game without any hint of them can be completely bland, but put in too much and they overwhelm everything else. But get them just right, and they bring out the flavor of everything else in your game." [Sch15]

2.6 Analysis of existing tactical games

It is often helpful to examine existing products before creating one's own. They can be used as bases for a further expansion of game mechanics or to use the knowledge of the essential aspects of different games that make them successful.

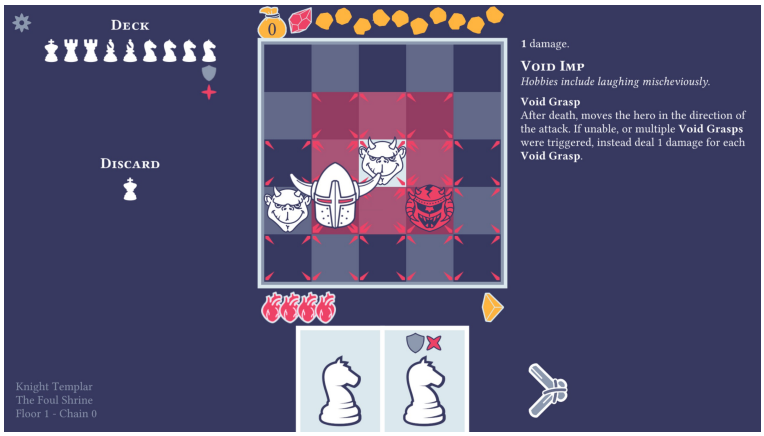

Figure 2.2: The layout of PawnBarian: red-marked tiles will be under attack, and the player can move like a knight in chess would.[j4n21]

2.6.1 Pawnbarian

This game is a perfect example of a small-scope tactical game. It doesn't have a story nor a lot of art and music and focuses primarily on the gameplay. It is heavily inspired by chess, using both the chessboard and the mechanics of unit movement. The player controls a single warrior and can act by using the moves of different chess pieces. Your enemies are more varied, though, as they employ different abilities and attack patterns. It is a game that features some randomness in level generation and utilizes drawing cards for available moves. The randomness helps create unforeseeable challenges and prevents memorizing the level layouts and creating an optimal strategy, rather forcing the player to consider every move individually.

Figure 2.3: Hand-drawn art assets in The Banner Saga[Stu21]

2.6.2 The Banner Saga

The narrative and art direction are the most prominent factors in The Banner Saga. It taps a lot of potential from narrative and sensation as sources of fun. There is an overarching story with cutscenes, a lot of written text fueled by player decisions, and a series of battles in between. The story isn't separate from the gameplay because the player's choices on the road can affect their army's battle morale and battle difficulty, cause their warriors' death or lead to recruiting new ones. Narrative, art, and music can invest the player emotionally in the game. The incorporation of the player decisions and story in battles helps enhance that.

2.6.3 Into the Breach

There is a clever twist in Into the Breach where the player can always see what their enemy will do in the next turn. The goal, then, is to work around that. The player's units have abilities that allow them to manipulate the position of enemies or cancel their attacks to render them ineffective. This predictability removes intuition from the decision process and makes each round feel like a puzzle game where the player needs to figure out the correct sequence of moves to succeed.

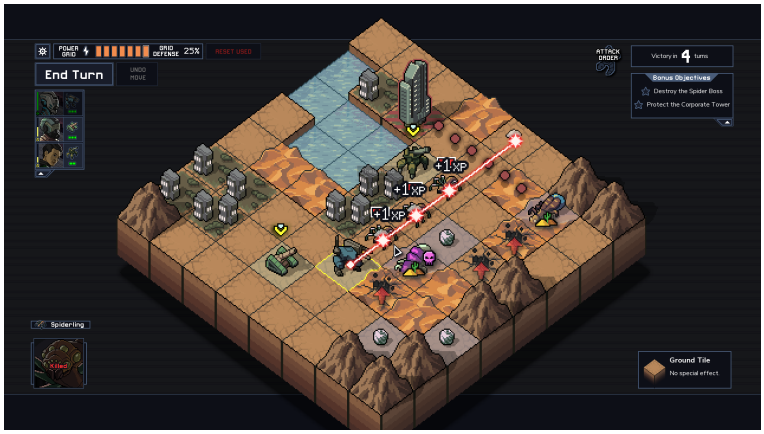

Figure 2.4: The player is on the move and can see where enemies will emerge and attack on their turn.[Gam21]

2.6.4 Summary

There are several takeaways from these three games. In Pawnbarian, we can see that a game doesn't need sophisticated art to be fun. It also shows us how important randomization is for games that want the player to learn to improvize.
The Banner Saga shows us how much great art can elevate a game. There is no shortage of games where the visual aspect is the cornerstone of the experience.
In into the breach, we can see how crucial positioning is in tactical games. It makes the gameplay much more interesting if the player isn't just calculating attacks, but also employs spatial thinking.

Chapter 3

Design

This chapter describes the design decisions process. There are several topics to be covered here, such as the grid system, cover system, feedback loops and player incentives.

3.1 Paper prototype

In game design, it is rare that the designed product will function in the intended way on the first attempt. This is why testing is a crucial aspect of game development since the early stages. However, implementing ideas in Unity might take a very long time and could turn out to be a waste of resources. This is why, when suitable, paper prototypes are used to test out initial game ideas.
In turn-based games, paper prototypes are even more suitable as they can almost completely encompass the game mechanics. For these reasons, I decided to create a paper prototype during the design phase and tested out different mechanics to decide if they should be included in the game.

3.2 Unit actions

This section will describe which actions units can perform and how those actions are performed.

3.2.1 Movement and distances

The game will be played on a grid. Some games employ hexagonal or square grids and both have their advantages and disadvantages, as described in the analysis chapter. For this game, I chose a square grid as it provides more variety in movement and attack choices and makes it easier for the player to estimate distances. It provides another dilemma, which is how to measure distance.
For unit attacks, I decided to use Manhattan distance as it is very clear for the player how far a unit can attack from any given tile. For movement, however, it is quite unintuitive as it doesn't correspond to human behavior.

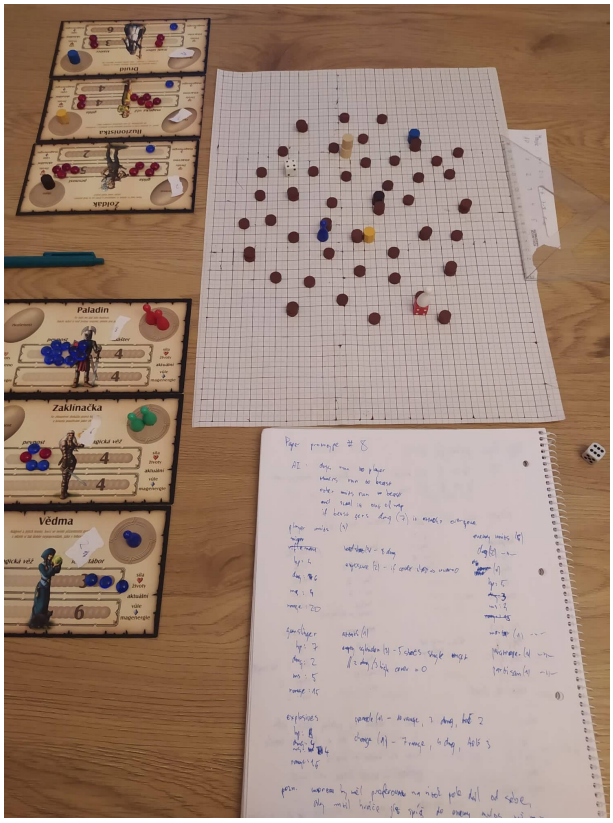

Figure 3.1: The used paper prototype made partly from existing board games (Puerto Rico and Prophecy).

Therefore, I decided that movement will have eight directions instead of four, including diagonals. Moving along a diagonal, a unit will travel the distance of 1.4. That is a closer approximation to what the distance would be in the real world instead of it taking two cardinal movements.

3.2.2 Attacks and cover

Attacking a unit will be as simple as deducting the player's damage from the enemy's health points. However, the game space will also contain objects that provide cover for any adjacent unit. Cover will give the unit damage reduction from its direction.
It would be possible for cover to reduce the hit chance of incoming attacks. However, I want to avoid using probability to derive whether one character will hit another. There are two reasons for this. Firstly, it can be simply frustrating to see a shot miss when the player is counting on it. The other is that the turn will usually end after this randomized action, and the player can't work around the unexpected event anymore. However, without any randomness, games can be too trivial or repetitive. We can work around this by including input randomness at the beginning of the turn rather than output randomness at the end. One example could be that units will be given

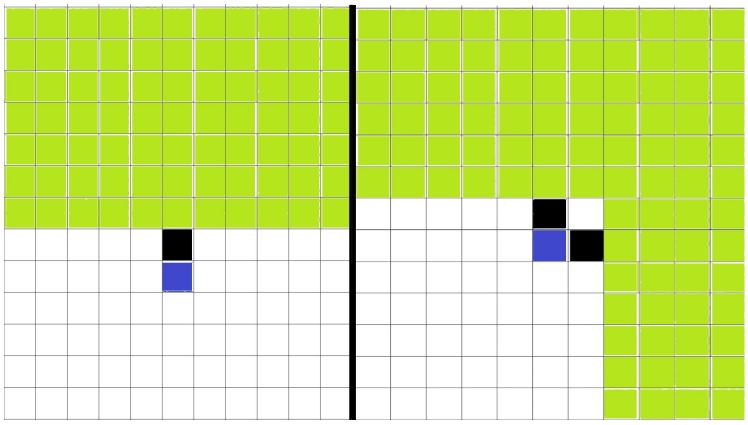

Figure 3.2: This figure shows a unit (blue) behind cover (black), and all the tiles it takes reduced damage from.

a semi-random number at the beginning of every round, affecting how many actions they can take. Let's call it morale.

3.3 Game loops and incentives

In strategy games, it is important to implement a feedback loop that influences the state of the game. This loop should be a positive feedback loop to let the player or the enemy gain and utilize advantages. That means that the more one side wins, the easier it should be to succeed in the future.
Usually, this feedback loop emerges out of existing game mechanics. For example, if the player manages to kill an enemy unit, they become more powerful by having more units than the enemy. However, the turn order can be set up so that it is a very minor disadvantage and if the unit wasn't very strong to begin with, it doesn't affect the game very much.
To make this loop stronger for both sides, we can utilize the focus of a unit. It can be influenced by other events in game, such as the player losing a unit or managing to kill an enemy.
This approach has several advantages. One of them is that it will speed up the late stage of the match because when someone gains the upper hand, they will be able to win the rest of the game more quickly. The other one is that it adds weight to even more minor decisions at the beginning of the game, as they may influence the whole game. There is, however, a danger in this approach. Positive feedback loops can quickly get out of control, so they should either be weak or have a limiting element.
For example, instead of player units gaining or losing focus for every death of any unit, it can be limited to some more important units in the game. This makes the loop both weaker and presents the player with a choice. They can decide if they want to aim on these high-value targets to strengthen their own units. However, to make this choice more interesting, there also have to be drawbacks to it.

To further expand on this mechanic, the high-value targets should be units that are more difficult to reach or have more health than other ones. Other enemies can, then, focus more on endangering the player's units, drawing their attention to themselves. Alternatively, they can target another unit that is perhaps in a more vulnerable position or in a position that has more potential to hurt the player.

Chapter 4

Technical analysis

In this chapter, we will attempt to find the most efficient and robust way to create the game from a technical standpoint. This includes the use of game engines and modelling software, but also more general ideas, such as modular components or map generation.

4.1 Existing software

There are many technical aspects to creating a computer game. It is a multi-disciplinary task and needs complex skills and software to realize. Thankfully, many tools can simplify this process. The most significant one is the game engine. Jason Gregory introduces game engines as "fully featured reusable software development kits that can be licensed and used to build almost any game imaginable." [Gre14] However, this doesn't mean game engines are all you need. Another crucial tool is modelling software. It allows for the creation of 3d assets, as well as animations. Those are then imported into a game engine which can switch between them on command. Then there are graphics editors, which are used to create textures used as properties of materials for 3d models. Finally, digital audio workstations can record, edit, and produce music and sound effects.

4.1.1 Game engines

Game engines usually handle a lot of fundamental computations, such as collision detection, rendering, or physics. In addition to that, they let the developer use code to add more functionalities. They also support the use of 3d models, materials, and animations. There are many commercial game engines used in the industry today, but in this thesis, we will use the Unity game engine. It is currently one of the most widely used game engines. It is very suitable for solo projects due to its simplicity and adaptability. It also is one I have previous experience with and is free to use non-commercially.

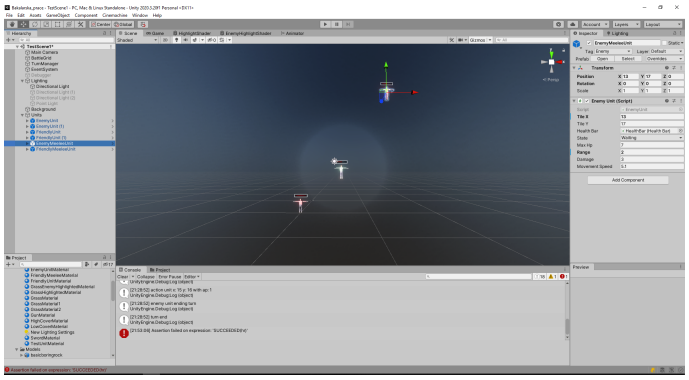

Figure 4.1: Unity game engine.

4.1.2 Modelling software and graphics editors

3D modelling software is used to make 3D models and animations. Those can be imported into Unity where they are controlled by scripts or conditioned transitions. They can also be used as modular components, which will be explained later. For 3D modelling, I will use the open-source program called Blender.
Graphics editors allow for creating textures. Those textures are then used to create a material that serves as a top layer of a 3D model. This layer makes the viewer perceive it as a real substance. This thesis will not focus on creating textures as there are many of them available under the Creative Commons license.

Figure 4.2: A 3D model in blender.

4.1.3 Audio software

Digital audio workstations are programs that provide tools for creating and editing audio. It can be produced either by recording or synthesizing different sounds. Usually, it is a mixture of both. Creating audio is beyond the scope of this thesis. Therefore, the game will use freely available music and sounds.

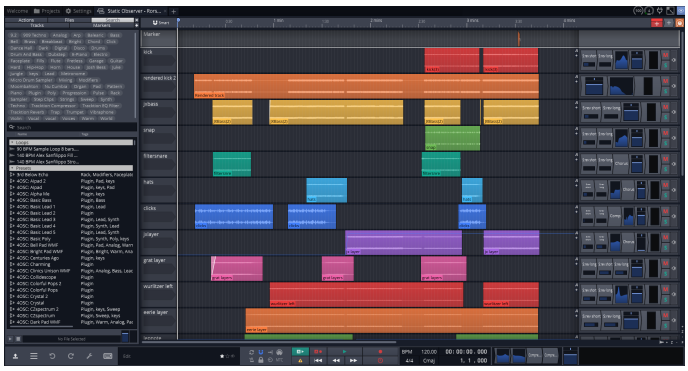

Figure 4.3: Waveform audio software.

4.2 Scripting

In Unity, scripts can be written in C# language and can affect anything in the game. In this thesis, they will be used for grid generation, pathfinding, AI, game logic, user interface and more. I will not go into detailed explanation of those scripts, but will outline the most important algorithms.

4.2.1 Grid generation

The game world will consist of a grid, which will be generated from a text or image file. The advantage of this approach is that it is very easy to modify a level from this file. Instead of placing every tile separately in the Unity editor, they can be rearranged from a text or image editor. The script will then place them into the level as intended. With this technique, the 3D models of some objects can also be randomized, which can lead to more variability.

4.2.2 Pathfinding

Pathfinding is a very common problem in video games. In strategy games, it is an easier one because of the clearly defined grid. Usually, the algorithm has to find a viable shortest path from point A to point B. There are several well defined algorithms to find such a path. The most prominent one is Dijkstra's algorithm. It is a best-first search that explores all possible paths and returns the shortest.
In some cases, exploring all possible paths can be computationally demanding and redundant. This is why we will also use a greedy modification of Dijkstra's algorithm – the A* algorithm. It doesn't have to search through every possibility, because it uses a heuristic function. It can be understood as an estimate of how close a certain position is to the desired goal. Positions that are getting closer to the goal are then searched sooner then ones that are farther from it. While this heuristic doesn't have to yield the best result, it leads to the shortest path most of the time and it does so very quickly.
A* has become the standard for pathfinding in video games and will be used

in this thesis for finding the shortest path. Dijkstra's algorithm will also be used for finding all possible movement options for a given unit.

4.2.3 AI

Artificial intelligence is fundamental for player experience, therefore it has to be balanced carefully and is very difficult to execute well. In turn-based games, there are many possible approaches as it has been a widely explored topic that received mainstream attention. For example, the chess match between Kasparov and Deep Blue is considered a cornerstone of AI research. For turn-based games, there are numerous algorithms, such as the minimax algorithm or Monte Carlo search, that work based on exploring the possibility space as deep as possible. However, they are very complex in the sense that the computations take a long time. According to Russel and Nordvig [SR15], the time complexity of the minimax algorithm is $O(b^m)$, where b is the number of legal moves and m is the depth of the search tree. In this case, there might be a much more effective solution available.
Multi-criteria decision making is often used in small scale tactical games as it doesn't just find a good possible move, but can influence the way different agents behave and make their behavior distinct and unique [Che13]. It is also very computationally undemanding as it doesn't search a whole possibility tree, but only evaluates the current situation.
The core of the algorithm is an evaluating function, that assigns each possible action a score based on several criteria. After all tiles are given a score based on all criteria, the one with the best score is chosen and the action is executed. Those criteria can be different for each use, which makes it very good at distinguishing different behaviors. Weights of different criteria can also be introduced, which makes for an easy introduction of greater variability.

4.3 3D modelling techniques

There are two main approaches to 3D modelling – polynomial modelling and polygonal modelling. Polynomial modelling uses polynomial surfaces to represent a 3D model. The creator of the model decides where the control points of a polynomial surface lay, and the surfaces are approximated between them. Polygonal modelling, on the other hand, consists of point that are connected by triangles or other polygons.
The most important differences in the use of polynomial and polygonal modelling are their precision, scalability and difficulty of use. Due to their nature, polynomial models create smoother surfaces. They can also be scaled to any size without losing any amount of detail. However, they are more difficult to create and render. This is why polygonal modelling tends to be favored in video games. This will also be the case in this thesis.
There are several techniques that are important to polygonal modelling, such as sculpting, vertex editing and the use of modular components. They will be discussed below.

4.3.1 Sculpting

Sculpting is perhaps the most powerful approach in polygonal modelling, but it does come with some disadvantages. It lets the sculptor add or remove material from a model as if they were sculpting from clay. It is very effective for making detailed models of non-exact shapes very quickly. However, it is very hard to precisely control, so it might not be suitable for modelling objects that have strictly set proportions. Sculpted objects also tend to contain many small polygons, making them more computationally demanding. Therefore, sculpted objects are used mainly for baking of normal maps.

4.3.2 Vertex editing

Individual vertices, edges and faces can be edited to put a desired shape together. In its most basic form, this could be a slow process of moving every vertex individually to make them compose a 3D object. Fortunately, there are many tools that quicken this process, most importantly extruding, loops cuts, insetting, and many more.
While vertex editing can be more time demanding, it is more suitable for objects with strictly defined boundaries, such as walls, tables, stairs, rocks and other inanimate objects. It can also be stylized into a low-poly look which embraces its imperfections and makes for a simplified view of the world.

4.3.3 Modular components

Modular components are in-game objects that can be re-used multiple times within the game. This is a common approach in game development that saves time and resources. The idea is to achieve a lot of variability while not creating too many 3d models. For example, instead of making several variations of different pipe layouts, the developer can produce parts that fit into each other and then change their arrangements.
Another use of modular components can be seen when creating a game world. A forest, for example, can only consist of a few different versions of trees that are repeated. We can also use the same models with different size or color to achieve a similar effect.

4.3.4 Animations

Animations are key to make a 3D model move or change shape in a scene. However, to understand animations, we must first understand armatures and rigging.
Armatures are equivalent to skeletons, which consist of bones. Bones are added to parts of the model which should be able to move independently from its other parts. For example, a typical 3D model of an arm has at least eight bones. The upper arm, forearm, and hand have one, and each of the fingers have one or more. With these bones, we can simulate the arm bending in the shoulder, elbow, wrist and finger joints.

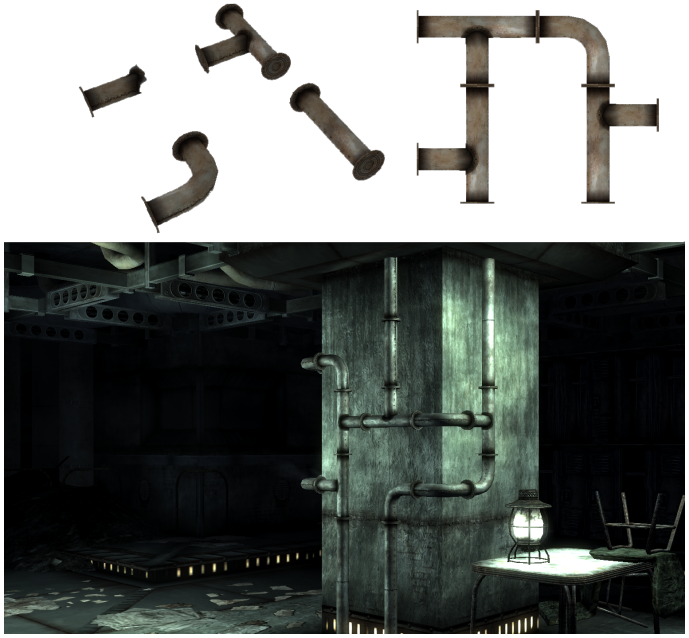

Figure 4.4: An example of modular components usage.[Bur21]

These bones can then be moved around and some parts of the 3D model will follow them. For example, a bone in the upper arm of a modeled character should cause the arm to move, but the rest of the body should stay still. Rigging, then, is selecting which parts of a model correspond to which bones of the armature.

4.3.5 Textures

Textures are used to map colors (or other attributes) onto a 3D object. First, the vertices of the object are given UV coordinates. They connect a given vertex to a given point on the texture. Then, the texture is applied accordingly. Textures can serve multiple purposes. The most important one is color, but there are several other uses. Normal maps, for example, can seemingly give a texture depth where there is none in the 3D model. This is achieved by manipulating the way light bounces off the texture.

4.3.6 Materials

Materials control the appearance of 3D models. They have many attributes that allow them to be very versatile. More importantly, combinations of textures can be used in materials to create realistic visual appearance. Materials can also use different shaders for their appearance.
In Unity, so called PBR (physical-based rendering) materials are used. They are a type of material that uses four types of textures. They are albedo for

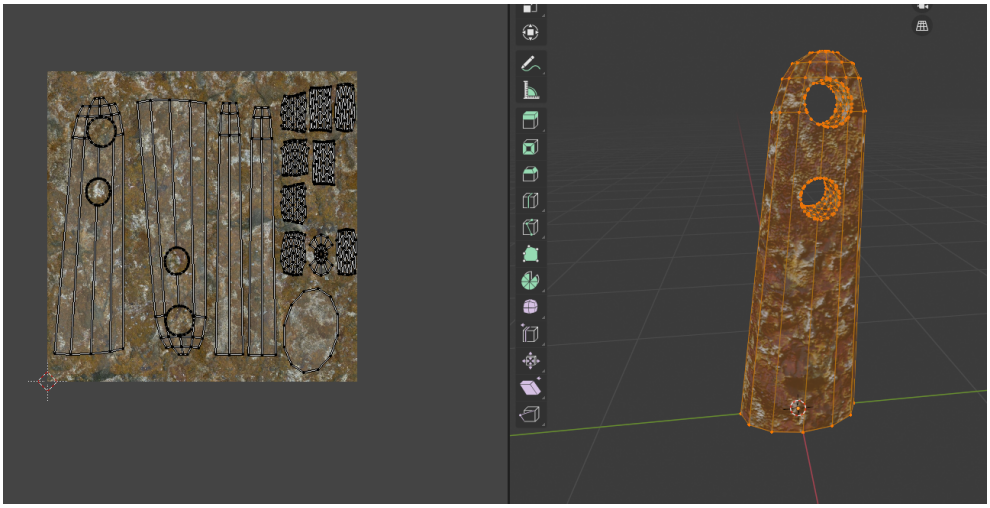

Figure 4.5: An example of UV unwrapping and a color texture.[Tuy21]

the material's color, the metallic and roughness properties for how much light the material reflects, and normal maps to manipulate light reflections to create the illusion of depth.

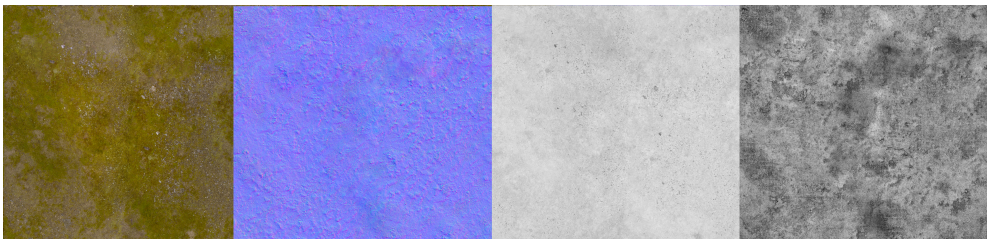

Figure 4.6: An example of a material created with four textures. From the left to the right they are albedo, normal map, roughness and metallic. [Tuy21]

Chapter 5

Implementation

In this chapter, the implementation of the game is described. There are two major parts, scripting and modular components. Scripts are written in the C# language and utilize the Unity framework, as well as some algorithms that will be described below. Modular components were created in Blender and imported into the Unity game engine. Scripts were created directly in Unity.

5.1 Game Systems and Scripts

This section encompasses all the most important game systems and scripts, such as the tile system, AI, pathfinding or map generation. The most important is the turn manager script. Its role is to control the turn order and call other scripts to ensure the correct execution of code. The script called TileGrid contains the representation of the level. Each unit has their own script, which keep track of its attributes and triggers animations or other changes. The interactions between units, namely attacking, is done within the combat manager script. Last but not least, there is an AI script that calculates enemies' behavior.

5.1.1 Tile System

The foundation of the game is the tile system. Before the game starts, a game object for each of the tiles is created. A tile has three important properties, its coordinates, a click handler, and a list of its neighbors. Coordinates and neighbors let algorithms work with the tiles further, for example to use pathfinding or calculate distance. The click handler registers when the user clicks on a tile with the use of a box collider and passes that information to the game manager.
There are several types of tiles – an empty tile, low cover, and high cover. An empty tile is one that units can move to while the other two types provide protection for a unit that hides behind them. This has to be taken into account when moving on the map so that units don't move to an occupied tile. To keep this information organized, a grid manager object keeps track of both actual game objects and their simplified representations.

There are two types of pathfinding used in this game. Breadth-first-search serves to inform the player, and A* calculates the shortest path between two points.
Using breadth-first-search, the game can highlight all available tiles for the unit that is taking its turn. It keeps track of tiles that were encountered but not explored. With each iteration, it removes a first element from this list and finds its neighbors. If they were not explored yet, it adds them to the list. Following these steps, it will be able to visit all available tiles. To stop the algorithm from spreading over the whole grid, the search is cut off in a certain depth depending on the searching unit's movement points.
The A* algorithm is similar to breadth-first-search but uses heuristics and is much faster in most situations. It also keeps a list of encountered nodes, but unlike breadth-first-search, it keeps them sorted depending on two criteria – the distance from the beginning and a heuristic function. The heuristic function determines the distance from the goal. In this thesis, it was achieved by using the euclidean distance.
Another important function of the tile system is calculating whether a unit is in cover in relation to another unit that attempts to attack it. This is done simply by checking neighboring tiles and if they consist of cover in the direction of the attacking unit.

5.1.2 Turn management and turn order

The crucial update loop in the turn manager script has two main branches. One runs if it is the player's turn and the other if it's an enemy's turn. If the player is playing, it works with the grid system to highlight possible moves for a given situation. At the same time, it is waiting for player input from the keyboard, which can result in the game moving forward. The other branch is used for the enemies' turns. It calls upon the AI to perform an action with the active unit. When the AI has no moves left or it wants to finish its turn, the turn manager moves to the next unit.
The order in which units take their turn can dramatically affect the outcome of the game. The starting units have advantage over units that come after them since they have an opportunity to attack the opponent first, possibly removing them from the game before they can take their turn. To give neither side a decisive advantage, the order is evenly distributed based each side's unit count. To preserve some variation, though, the individual units are chosen randomly. To do this, first a ratio of friendly to enemy units is found. Then, units are added according to this ratio so that the sequence is evenly distributed.

5.1.3 AI

Out of the algorithms discussed in the technical analysis chapter, I decided to use the multi-criteria decision making due to its fast computation and variability.

The decisions were based on the following criteria:

Figure 5.1: An enemy unit attacking the player's unit

1. Cover - it is as important for a unit to be in cover as it is for them to deal damage to the player. This score evaluates how many of the player's units that are in range, would have their damage reduced by cover. It also gives a higher score if the cover is more powerful.

2. Player units in range - for a unit to be well positioned, they should be able to fire at the player while not getting hurt back. This score was at its peak when one player unit was in range and decreased as more enemies could potentially hurt it. However, if there was no one in range, it was set to 0 as it meant that the unit couldn't perform an attack.

3. Distance from nearest player unit is a complement of their units in range. It would often happen that units which started the game far away from any player units wouldn't know which direction to go and would just stay in one place until the player came to them. This failed to create pressure on the player. Enemy units were given this score to try to get in range of player units if they were too far to attack.

4. Ally proximity - to stop enemies from clustering in a single position, a negative score was given to a tile that was too close to the unit's allies. This allowed enemy units to spread themselves out and create more difficult challenges for the player.

5. Flanking score was based on how many player units weren't in cover in relation to a given tile. This meant that the unit could attack them with

their full damage potential, which is a considerable difference and an important part of the game.

6. Angle score is an expansion of the flanking score. It is often the case that there are no possible tiles that would be flanking the player's units. Therefore, units try to get to a narrower angle to make flanking easier in the future.

7. Forbidden movement - for tutorial purposes, an extra score was added that would make the AI stay in one place and not move at all. This would cause the enemy unit to only use attacks and was done to not overwhelm the player in the first level.

Each score was also given a modifiable weight that would make for easy testing, iteration and balancing. For different kinds of units, some weights could also be discarded. For example, meelee units would not take cover, flanking and angle scores into account while increasing their score of trying to get closer to the player by 50 percent.
This scoring was, however, made primarily for movement. To be able to compare it to the choice of attacking instead of moving, the attack score was set simply by seeing how much damage a unit could cause if they didn't move, but attacked a player unit. Then, the score was compared to the best score of movement and the more highly evaluated action was chosen to be performed. Such an AI is very functional, but far from perfect. However, it is easy to modify and improve upon. One problem is that if a unit can take multiple actions in one turn, it will still evaluate them separately. This can lead to limited decision making. For example, if the unit has enough action points, it could attack twice instead of movement. To make use of such opportunities, attack score is doubled when such a possibility presented itself.
Nevertheless, this system could be improved by the unit exploring all sequences of moves and picking the one with the best results.

5.2 Modular Components

As this game is meant to be played on a grid, the use of modular components is very straightforward. Each tile on a grid will be represented by one or several components, and the tiles will be combined to create the world map. The tiles will either be empty and consist of a textured cube, or they will contain another object (cover or unit), which will sit on top of the tile.

5.2.1 3D modelling

The 3D models were created with the many tools offered by Blender. The most important ones include mesh editing, bones editing, UV unwrapping and texture painting. As a creation of a single realistic 3D model could expand to or even beyond the scope of a bachelor's thesis, I decided to use stylized low-polygon models. They are called low-polygon because they consist of

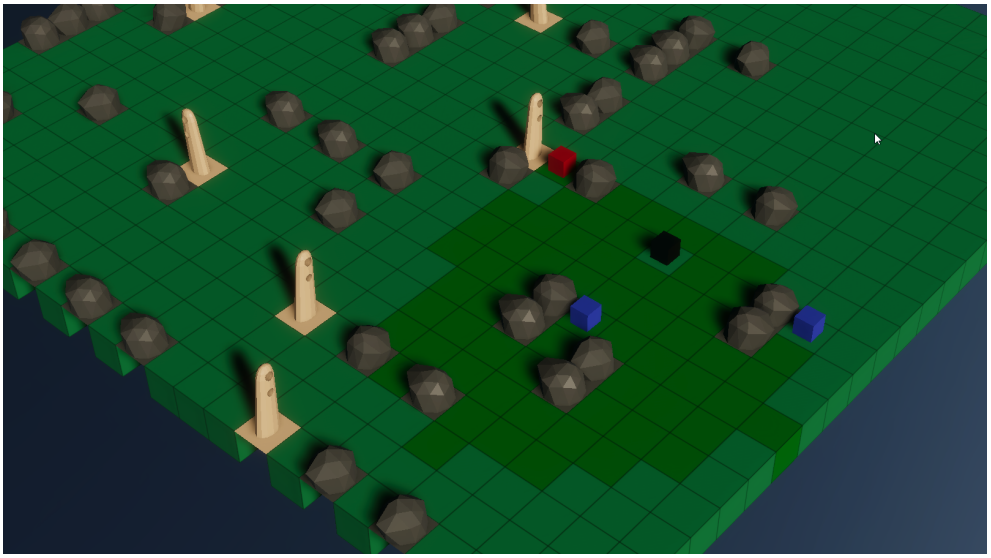

Figure 5.2: An early version of a level consisting of modular components

comparatively very few polygons, the unit model I created, for example, only has about a thousand polygons. Some other objects can contains as few as a hundred.
This approach requires the use of mesh editing rather than sculpting, as sculpting requires the use of numerous polygons to be effective. The materials,

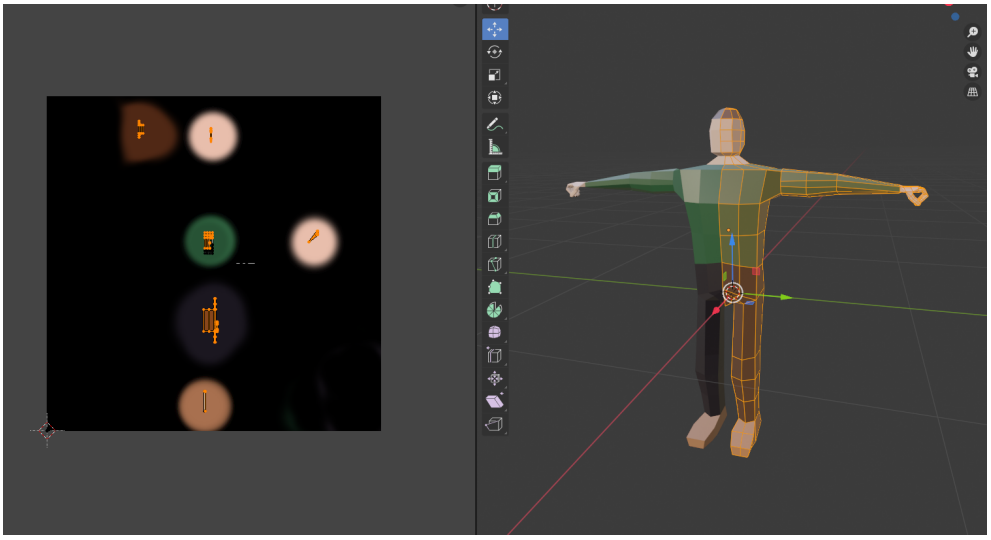

Figure 5.3: A 3D model of a unit and its texture

apart from the tile cube, were created with simplistic, single color textures. However, even with simplistic models, there should be some variability. In this thesis, it came in the form of game objects having several possible models that are being assigned randomly at the time of their generation at the beginning of the level. The same level can, then, be visually different every time it is played, even though systematically it works the same.

To add more variety, the models were also rotated by a random amount. This causes the whole space to not feel as uniform and artificial. In the grass tiles, the texture was also independently rotated in the shader to create a more natural look.

To emphasize the grid and make sure the player understands where they can

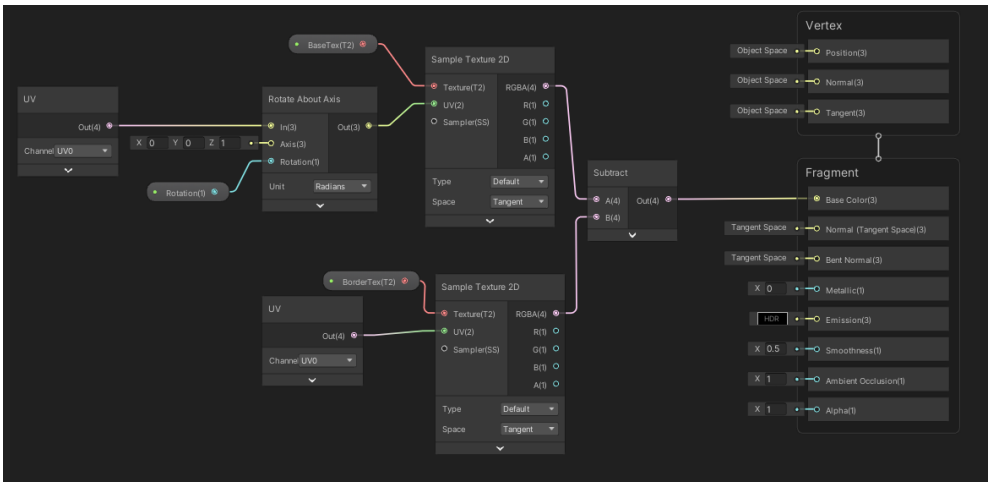

Figure 5.4: A shader that rotates the texture independently of the object

place units and how distances work, the grid needed to be distinguishable. This was achieved by another shader that made the borders between tiles black.

Two more shader were used to highlight tiles that the player can currently move to and to highlight enemies they can currently attack.

Figure 5.5: Tiles the player can move to highlighted in blue and enemies they can attack highlighted in red

5.3 Testing

Testing is crucial for game development, as it can tell us if both the player and the game behaves as intended. Usually, the developer is too invested and familiar with the game so they can overlook some defects or not notice them altogether. It is also important to know if players who have no understanding of the game can correctly interpret what the developer is trying to communicate to them. For these reasons, the game was tested with seven people.

Figure 5.6: The finished game with all models and shaders used

They were free to figure the game by themselves from the visual feedback and short hints at the beginning of each level. They would only receive advice if it was clear they wouldn't be able to progress by themselves. This only happened once, but was to be expected from a person who had almost no previous experience with video games.

5.3.1 Questionnaires

After finishing the game, players were asked to fill out a Likert scale questionnaire. A Likert scale questionnaire consists of questions with a psychometric scale, which lets the tester answer on a scale from one to five between two statements. Those two statements may differ for each question and are indicated in the parentheses beside each question (1 - 5).
The questionnaire featured the following questions:

1. How familiar are you with strategy games? (not at all - very familiar)
2. How do you like strategy games? (not at all - very much)
3. How complicated did you find the game? (not at all - very complex)
4. How understandable were the controls? (very confusing - very intuitive)

5. How often did you have to think about the next move? (almost never - every time)
6. How fun was the game? (not at all - very fun)
7. How well did the game inform you about what was happening? (very poorly - very well)
8. How would you rate the graphics? (very poorly - very well)
9. How would you rate the AI? (trivial - complex)

The results can be seen in figure 5.7.

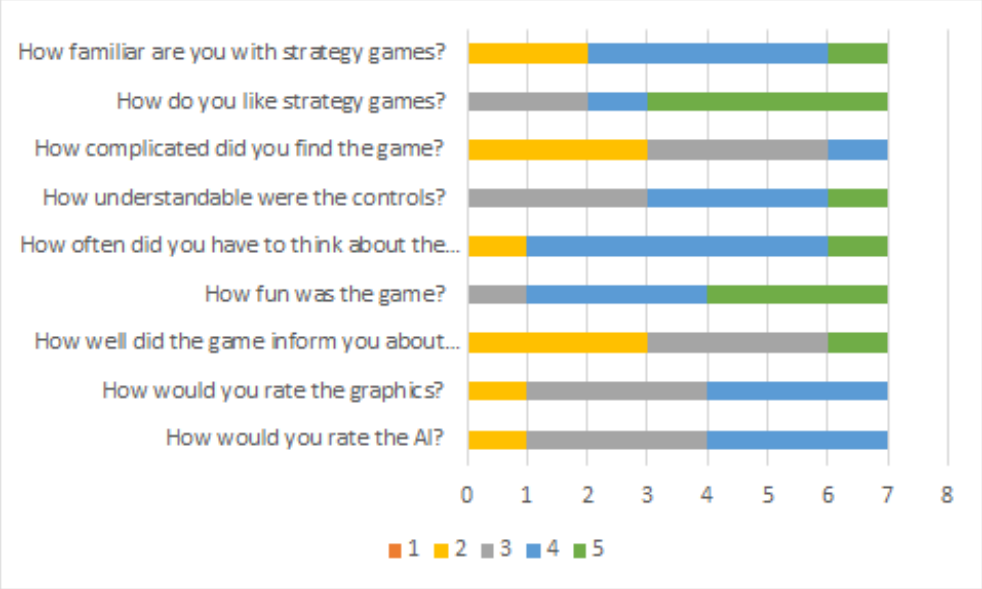

Figure 5.7: Results from Likert scale questionnaire

The players were also asked the following open-ended questions to let them express more thoroughly and focus on their own perceptions:

1. What did you enjoy most in the game?
2. What did you find most annoying in the game?
3. Did you encounter any bugs?

5.3.2 Results

First, let us focus on the Likert scale questionnaires. From the game design perspective, the most important metrics are if the game was fun and if the player had to think about their next move. In that aspect, I believe the game was a success as most players found it enjoyable and the game had them weighing their options before deciding what to do.
However, a lot of players believed the game mechanics were inadequately

explained. It was an intentional decision to not let the player know everything about the game, but more tutorials or visual indications would be an improvement.
There were mixed reactions to AI and graphics. They seemed to satisfy most players, but didn't tend to score very highly. Low poly graphics tend to be somewhat divisive, but could certainly be improved with larger variety of models. The AI was challenging for the players, but as the units didn't have many possible actions to choose from, the AI had no opportunity to impress players.
From the open-ended questions and from watching the testers while playing, we can say that the most enjoyable part of the game was thinking about positioning behind cover and flanking the enemies. A lot of players also liked the minimalist models.
The questions about what the players found annoying and if they encountered bugs were often interchangable. There was one major bug where the game sometimes wouldn't highlight the tiles the player could move to. This would happen once or twice per game and led to some players being confused and sometimes they would skip their turn because they thought they couldn't do anything else.
A more minor bug was that in level four, the players couldn't move the camera to all parts of the map. Fortunately, it only affected a small part of the map and the game was still playable.
Some players also wouldn't notice the differentiation between ranged and meelee units at first, as the models weren't large enough to notice the difference. This would be easily fixed by scaling up their distinctive features.
When the players were asked about what they most enjoyed, about half commented positively on the graphics and the other half commented positively on the gameplay. I think it is a good result where those two aspects are in balance.

Chapter 6

Conclusion

This thesis aimed to create a turn-based 3D tactical game with the use of modular components and test it on at least six players. First, I researched principles of game development and analyzed existing games for inspiration. Then, a paper prototype was created to test initial ideas and shape the form of the game. Once the specifics of the game were more apparent, I weighed different algorithms and approaches that could be used in development. The layout of levels and the modular components needed were also identified.
The game was developed in Unity. First, the scripts were written with placeholder 3D models. The most crucial scripts were turn order, map generation, unit behavior, and enemy AI. All the scripts were written so the game could be modified and balanced, creating editable variables for unit attributes and enemy AI. The map could be generated from an easily editable text file.
After coding the base of the game's logic, the modular components were created. They were modeled and animated in Blender software using the various tools of vertex editing. For some materials, custom shaders were created. Most of those materials used textures made by me. Some exceptions featured textures downloaded from the internet.
After the game was finished, qualitative tests took place with seven participants. Tests revealed some minor bugs, but the game was well received in the most important aspects.
The development of this game deepened my knowledge of coding and modeling tremendously. It also shows how complicated pieces of software games are and how many different aspects and approaches can be used while creating them.

Appendix A

Bibliography

[Bur21] Joel Burgess, http://blog.joelburgess.com/2013/04/skyrims-modular-level-design-gdc-2013.html, dec 2021.

[Che13] Alex Cheng, https://www.gdcvault.com/play/1018058/AI-Postmortems-Assassin-s-Creed, 2013.

[Chy] William Chyr, https://manifold.garden/presskit.

[Gam21] Subset Games, https://subsetgames.com/itb.html, dec 2021.

[Gre14] Jason Gregory, *Game engine architecture*, CRC Press, 2014.

[IGD22] IGDB, https://www.igdb.com/games/the-legend-of-zelda-breath-of-the-wild/presskit, april 2022.

[Int21] DFC Intelligence, *Global video game consumer segmentation*, https://www.dfcint.com/product/video-game-consumer-segmentation-2/, 2021.

[j4n21] j4nw, https://j4nw.com/pawnbarian/presskit.html, nov 2021.

[Kos05] Raph Koster, *A theory of fun for game design*, Paraglyph Press, Inc., 2005.

[Mad19] Pirie Madsen, *The match of the century*, https://www.adamsmith.org/blog/the-match-of-the-century, 2019.

[Mos19] Wojciech Moskwa, *Norweigan teenager to be crowned new chess king*, https://www.reuters.com/article/us-chess-norway-carlsen-idUSTRE5BT17H20091230, 2019.

[oS22] National Museum of Scotland, https://www.nms.ac.uk/explore-our-collections/stories/scottish-history-and-archaeology/lewis-chess-pieces/, feb 2022.

[RH] Rober Zubek Robin Hunicke, Marc LeBlanc, *A formal approach to game design and game research*, https://users.cs.northwestern.edu/~hunicke/pubs/MDA.pdf.

[Sch15] Jesse Schell, *The art of game design*, CRC press, 2015.

[SR15] Peter Norvig Stuart Russell, *Artificial intelligence a modern approach*, Pearson Education, Inc., 2015.

[Stu21] Stoic Studio, `https://bannersaga.com/media`, nov 2021.

[Tuy21] Rob Tuytel, `https://polyhaven.com/a/aerial_grass_rock`, dec 2021.