Western Norway University of Applied Sciences

Bachelor Thesis Report

The Gamification of Fjell Fortress

Simon Vaular
Griffin Marshall Retzius
Oneal Didrik Ferkingstad Lane

Computing / Information Technology

Faculty of Engineering and Science

Supervisor: Ilona Heldal

Client: HVL Media Lab

Submission date: 23.05.2022

I confirm that the work is self-prepared and that references/source references to all sources used in the work are provided, cf. Regulation relating to academic studies and examinations at the Western Norway University of Applied Sciences (HVL), § 10.

Report title The Gamification of Fjell Fortress	*Date: 22.05.2022*
Authors Simon Vaular, Griffin Marshall Retzius, Oneal Lane	*Pages without appendices: 62*
	Pages of appendices: 1
Field of study: *Computer engineering / Information technology*	*Amount of floppy discs / CDs:* 0
Contact at study: Ilona Heldal	Classification: No
Notes: None	

Client: HVL Media Lab	*Clients reference:*
Clients contact person: Øyvind Fosse	*Phone number:* 55 58 77 66

Summary:
The goal is to create a historically correct, enjoyable learning experience for pupils by gamifying the previous bachelor project, as well as increase the performance and playability of the game. It is built upon two previous bachelor groups. The application will be available for 9th grade middle school students and possibly an exhibition at the Fjell Festning Museum. Today's solution will further develop existing code based on the preceding bachelor groups. The end result will not only be useful through education, but it will also preserve local Norwegian history in a digital environment.

Keywords:

VR development	Gamification	Unity
Fjell Festning	Map Design	Educational

Table of contents

Terms and Abbreviations

Word	Explanation
VR	Virtual Reality. Virtual reality is about creating a sense of a physical presence within a virtual world (Qualcomm, n.d.). Typically virtual reality hardware consists of the head mounted display, and hand controllers.
AR	Augmented Reality. In AR the physical world is enhanced with digital elements.
XR	Extended Reality is an umbrella term consisting of VR, AR, Mixed Reality and everything in between (Qualcomm, n.d.).
XR Rig	According to Unity: "An XR Rig is the user's eyes, ears, and hands in the virtual world." (Unity Technologies, 2021).
LOD	Level of Detail. A computer graphics method used for rendering objects with lower detail distant from the camera.
NPC	Stands for Non-Player Character. It is used to refer to all in-game characters that are not controlled by the player.
AI	Artificial Intelligence can mean two slightly different things in the context of this project. These days AI typically refers to systems that learn over time based on large quantities of data. In video games however, it typically refers to the system that controls the behavior of non-player characters.
Scene	The 3D environment in which the player, the story and 3d objects reside.
FPS	Frames per second. Refers to how many fully rendered pictures are shown on the screen per second. A 60hz monitor can display a maximum of 60 frames per second.
VFX	Visual Effects. Typically refers to particle simulations, explosions, shockwaves and other purely visual elements.
Shader	Shader - A computer graphics function that produces effects such as shade from lighting.
Mesh	A Collection of polygons, edges and faces that make up a 3D object.
GPU	Graphics processing unit. A piece of computer hardware dedicated to efficiently drawing objects to the screen.
CPU	Central processing unit. Essentially the brain of the computer.
URP	Universal Render Pipeline.

1. Introduction

1.1 Context

During the German occupation of Norway during world war 2, the Germans built a fortress on the mountains above the town of Fjell. It was built to protect the coast outside Bergen from a potential allied invasion. The fortress was built by soviet prisoners of war, as well as local entrepreneurs. At the war's end, uncertain times awaited those left at the fortress. What will happen to the German soldiers, and those who cooperated with them? What would happen to the Soviets upon arrival in their own country? Would Stalin have mercy on them for surrendering to the Germans?

This is a theme that Museum Vest, the current owners of the Fjell Fortress, wants to explore. They want to do this in the form of a video game. Since 2018, Media Lab has, on behalf of Museum Vest, been developing a mobile game called Batteri Fjell 1945. Set in 1945, shortly after the war, the player encounters a cast of characters with their own unique ethical dilemmas, stories and struggles. The story takes place on the fortress, using elements of real events to create a captivating and educational experience.

Figure 1.1: Batteri Fjell 1945

Media Lab is, in the words of Western Norway University of Applied Sciences “the Western Norway University’s competence unit for the development and use of digital media and new technology in teaching, research and dissemination.” (Western Norway University of Applied Sciences, 2020). Media Lab has for this reason an interest in the game as a teaching tool. More

specifically they aim to have the game integrated as a part of middle school 2nd year (9th graders) curriculum in certain schools.

During the process of planning the mobile game, Media Lab got the idea of using the same story but in Virtual Reality (VR), allowing the user to fully immerse themselves into the scenario. It would allow players to wander around a 3D representation of the fortress as it might have looked at the end of the war, and to also get a much greater sense of how the lives of the people there might have been. The decision was made to have a near completely student driven VR version of the game developed in addition to the mobile game.

It is the VR version of the game that this rapport and associated work is built upon.

The VR project has been worked on by two bachelor groups prior to the current. The first group 3D scanned the fortress, and together with data from The Norwegian Mapping Authority, created a rough 3D representation of the terrain. The second group processed the rough terrain, and in combination with different objects textures managed to create a realistic environment within the Unity engine. They even got time to add some characters and gameplay elements.

Figure 1.2: The scene made by the two previous groups.

The current group's job is to build upon the foundation laid down by the previous groups to create an immersive, VR learning experience.

1.2 Motivation

The hope is for the final product to provide players with an experience that is both educational and fun. Virtual Reality has extraordinary potential in the areas of storytelling and education. World War 2 happened a long time ago, and so it can seem very distant for the youth growing up today to relate to. VR addresses this by allowing players to use their senses to a higher degree than traditional games, which often leads to a greater sense of immersion and realism. When discussing serious topics like the treatment of prisoners of war and German soldiers, their girlfriends and children, that extra level of realism and engagement might help players to get a deeper understanding of the stories than they otherwise would.

Letting the player be able to interact with the world not only helps with the realism, but also allows the story to involve the player to a greater extent.

Having a VR representation of the fortress allows players to experience the fortress in an entirely new way. Players can wander the area and experience the fortress as it might have been in 1945. Structures that no longer exist and areas that are closed off can be made accessible. If the game is set up at the museum, it would allow the visitors to explore these areas, and to also have a nice educational experience. The project files can at a later date be repurposed to serve as a 3D reconstruction of the fortress.

With Media Lab being the Western Norway University's competence unit in the use of digital media in learning, they have an interest in exploring VR as an educational tool. They also have an interest in the tools and methods used to achieve this. Particularly the use of 3D scanning to create digital representations of real life objects. Media Lab has a lot of experience with the technology, but its use within video games is uncharted territory. This project allows Media Lab to get a greater understanding of its practical use cases, and the workflows associated with implementing it. Media Lab has also expressed an interest in studying the learning outcomes of the VR and mobile versions.

1.3 Problem description and goal

The VR version in its current state features a 3D environment with some buildings and characters. It contains some gameplay elements, but the functionality is basic, and not fully implemented. The job of the current team is to build upon the previous work, and turn it into a video game that is both engaging and educational.

In the mobile game, the player's goal is to investigate Fjell Festning as a journalist. Your job as the player is to interview different characters, learning about their stories and struggles. Media Lab has created a dialogue script for each character in the game along with a base storyline for the mobile app.

The team's job is to implement the pre-written story, turning what is essentially just a 3D environment into a full game. However the story on its own might not be enough to make the game enjoyable. The team has for this reason a lot of freedom to add elements that they believe will enhance the user experience. On the other hand, the game should be complete at the end of this group's work, so making the scope of the project too big should be avoided.

In order to make the game as accessible as possible, the client wants the game to be playable on mobile VR headsets, including Oculus Quest 2 which is as of february 2022 the headset of choice for 46% of VR users on the SteamVR platform (Ridley, 2022). Having the game accessible on mobile VR platforms would allow institutions, like schools and Museum Vest, to invest only in VR hardware, instead of also having to invest in computers with expensive graphics cards. The problem with this is that the project currently runs poorly even on many gaming computers, and to get the game to run well on these, let alone mobile VR hardware, will require a lot of performance optimizations.

With the tremendous work ahead it leads us to the group's main problem: how can the group finish the VR game within the deadline, and at the same time ensure an engaging, educational and enjoyable user experience?

1.4 Client

Media Lab is the group's client, and are the same people who develop the mobile version. Øyvind Fosse of Media Lab is the project manager and largely the ones that the group has to relate to. Media Lab is the Western Norway University of Applied Sciences's (HVL's) competence unit for the development and use of digital media, new technology in teaching and research (Western Norway University of Applied Sciences, 2020). They explore different technologies and their use cases on behalf of the University. Their work is not limited to the university, as they use their competence to take on external projects.

Media Lab is also responsible for the operations of Learning Lab, a branch within the university, offering both advice and equipment in areas of technology and pedagogy (Western Norway University of Applied Sciences, 2020).

Media Lab has been collaborating with Museum Vest throughout the project's lifetime. Museum Vest is the organization running the Fjell Fortress museum and will be the owner of the final product. The museum will aid the development process through information as well as a guided tour. The script has been written in large part by Vigleik Røkke Mathisen who is the museum educator on Fjell Fortress, and the client and the team's main contact. They will periodically supervise the project, checking in on its current state.

1.5 Report structure

The first chapter of the report will emphasize the goals and motivations behind the project, as well as give a good idea of what the game is about and what the group wants to achieve.

Chapter 2 will help the reader understand the background of the project, and it will be described in more detail. This chapter will also highlight the previous bachelor projects that have taken part in Fjell Fortress.

The different technologies and methods used to carry out the project will be described in chapter 3. Here, the group will evaluate and justify the choices that were made and alternative solutions. Furthermore, the group will also state the chosen solution and the discussion around it, as well as a discussion around that subject. Lastly, project/development methodology will be presented, a project plan, risk evaluation and an evaluation plan.

Chapter 4 will consist of the project's design and structure. The chapter will also include the project's game implementations and how these have been accomplished.

Chapter 5 will include the project's evaluation and results, aslo the methods used for evaluation.

Chapter 6 discusses the project's results and why these results were as they were. The chapter gives an insight of how these results could have also been refined and will also further discuss improvements upon them.

Chapter 7 highlights the further work that could, or should be done in order for the project to be in an ideal state.

Chapter 8 includes all references used in the report.

2. Project description

2.1 Practical background

2.1.1 Previous work

Media Lab is nearing completion of their mobile game Batteri Fjell 1945. The VR version is essentially the same, only within a different environment. Media Lab has recorded voice lines for all 12 dialogue interactions, created 8 different characters and created a bunch of 3D models. The 3D models include the main canon, different variations of barracks as well as other smaller objects.

The first bachelor group who worked on this project used a variety of 3D scanning techniques, as well as online map data from the Norwegian Mapping Authority to create a realistic digital replica of Fjell Fortress and the surrounding areas. The second group focused on the development side, using the Unity engine, they mainly focused on the environment, but also added characters, prefabs and the basis for a dialogue system.

The project consists of a runnable Unity project with a small collection of assets (3D models, textures etc.) and uses Unity 2019.3. The project features dialogues in the form of text. This is implemented for 6 of the 12 total interactions.

Figure 2.1: Screenshot of the scene at the current groups project start.

2.2 Initial requirements and proposed solution

2.2.1 Initial Requirements

The client does not have an extensive list of requirements and instead lets the developers have a large amount of freedom in terms of functionality. The story has however to follow that of the mobile game. All dialogues have been pre-recorded, so changes to the storyline are difficult. The client wants the game to be comparable to the mobile version, but also to utilize the

possibilities that VR provides. Table 2.1 describes the initial requirements presented by the client.

Table 2.1 *Initial Requirement table*

Nr.	Initial Requirements
1	The game must be in a finished, playable state by the deadline. Playable means the story according to the script must be completely implemented, as well as an intro and outro scene.
2	Be able to run on an Oculus Quest 2 device without the use of an external computer.
3	Create the best possible gaming experience for the target audience (9th graders).
4	Add effects to amplify the story, and in turn the user experience.
5	Easter eggs. Things like hidden objects and features. Developers have a lot of freedom here.
6	Use of AI. In this context it refers to NPC behavior and not machine learning. The goal is to add some unpredictability to the game.

2.2.2 Clients proposed solution

The group has been given a large degree of freedom in terms of how the end goal is met. In terms of solutions they recommend the team to investigate the use of 3D scanning in order to accurately recreate the environment.

Media Lab wishes to work closely with the team, recommending the team to use the resources within Media Lab and Learning Lab to aid during the development process. Learning Lab in particular, has both access to appropriate hardware and the competence to guide the team in their use. The team is allowed to take advantage of Media Lab's previous work on the mobile game within the VR game. This includes models, plugins and assets, as well as information on how to use them.

Media Lab encourages the team to add their own flavor to the game through effects and easter eggs. Easter eggs is a term in video games and media referring to hidden features or objects. The client acknowledges that video games have evolved a lot since they were young, and is open for suggestions on how to make the game more appealing to a younger audience.

Ideally the game should be performance optimized to such a level that it can run on an Oculus Quest 2 VR headset. To do this the group plans to apply various performance enhancing methods such as culling and replacing objects with high vertex count from the previous version of the game.

2.3 Limitations

- **Resources:** The project has no funding, this means that some resources are out of reach. These may be expensive 3D models or third party components that could be used in the game.

- **Time:** Time is the greatest limitation in the project considering all the possible implementations, problems, bug fixing, and the short project phase.

- **Lack of experience:** The group has limited experience when it comes to developing in Unity. A challenge for the group will be to put realistic limits to the development. The group will need to focus their efforts and prioritize certain functionalities to implement.

- **Project Design:** The current project builds on previous work done by other bachelor groups. This means that the group needs to further develop existing code based on not necessarily well-documented decisions of the preceding groups.

2.4 Resources

Since the project was originally started by a preceding group, all code, documentation and essential information that correlates to the project is required. This project is composed of multiple files and a large amount of information which has been conveyed by Øyvind Fosse (HVL Media Lab). The game itself has been developed in Unity, which is to be used for further development and improvement of the game.

The aforementioned required information consists of historical lessons and facts that are related to the second world war and Fjell Fortress. This information is important to gain an understanding of the project and increase the focus on historical accuracy. This will help the group during development in relation to historically accurate functionalities within the game.

In addition to this, the group may require new assets, models and more from the Unity Asset Store, Sketchfab and other sources. Items such as interactable objects and particles will help in the gamification.

The team is in possession of great support from both Media Lab and Museum Vest. Support for developing in Unity is supplied by Alexander Miguel Jensen Sewe. He has assisted us with questions regarding development and different techniques within Unity. Øyvind Fosse is the primary client, he assists the group with ideas that can be implemented into the game and important quality control for evaluations. Ilona Heldal provides counseling for the scientific and literary part of the project. Lastly Vigleik Mathisen from Fjell Fortress guides us with documentation and information about the compound.

3. Project design

3.1 Possible solutions

Some technological and software choices were already set by Media Lab and the previous bachelor group. Therefore the project will continue in the Unity game engine, which was the chosen development environment from the previous bachelor groups.

If the project would be remade from scratch, a better choice for this particular project could be choosing the Unreal Engine 5. This would make it easier to create characters by using the engine's MetaHuman creator (Epic Games, n.d.) instead of hand modeling, or buying models, and more importantly the engine allows for directly using photogrammetry scans, or other high poly models in the scene through Unreal Engine's Nanite technology (Epic Games, n.d.). The project does focus heavily on recreating a historical landmark, and directly using photoscanned assets would result in a more realistic end result.

The previous group used Unity version 2019.3 with the SteamVR VR framework. This version of the project runs on the Oculus Quest 2 VR headsets when run from PC, however input from the controllers does not work, making the game unplayable without some changes to the input system. The project is also using the High Definition Render Pipeline (HDRP), which does not support building the game to android, and in return does not work on mobile VR headsets.

3.1.1 Unity version

Sticking with Unity 2019.3 allows for a quicker start to development. It has minimal risks in terms of potential issues, as no project files would be altered significantly. It also ensures compatibility with the plugins used by previous developers.

Upgrading to Unity 2021.2 grants greater flexibility by opening up the possibilities of newer tools, features and workflows that do not exist in older versions. This also allows the developers to explore technologies Media Lab could be interested in for future projects.

3.1.1 Framework and API

When working with a complex input system like a head mounted display, a layer of abstraction is needed. Several VR headset manufacturers provide plugins and assets for Unity that aim to make developing VR applications easier.

The inherited project uses the SteamVR plugin for Unity, a VR framework by Valve that handles input from the most popular VR controllers and head mounted displays (Valve Corporation, 2021). It also provides other tools and assets that aid in the development of VR applications. As the name suggests, it is specifically made to interface with SteamVR. SteamVR is a VR platform that allows users to play VR games, browse and buy games, chat with friends and utilize a variety of different hardware.

In 2019, the OpenXR specification was released (Khronos Group, 2019). OpenXR serves as a layer between the game engine and the VR hardware. This allows developers to write code for the OpenXR API instead of targeting specific hardware. Brent E. Insko, lead XR Architect at Intel, and OpenXR working group chair explains the purpose of OpenXR as follows:

> OpenXR seeks to simplify AR/VR software development, enabling applications to reach a wider array of hardware platforms without having to port or re-write their code and subsequently allowing platform vendors supporting OpenXR access to more applications (Insko, n.d.).

OpenXR differs from SteamVR by first being purely an API. It does not provide any tools or scripts, other than handling input from the hardware. Secondly it is implemented by the hardware manufacturers themselves, and for this reason does not rely on Valve to add support for that particular headset. This means that as long as OpenXR continues to be implemented by headset manufacturers, the VR application will be compatible with all future VR hardware (see Figure 3.1).

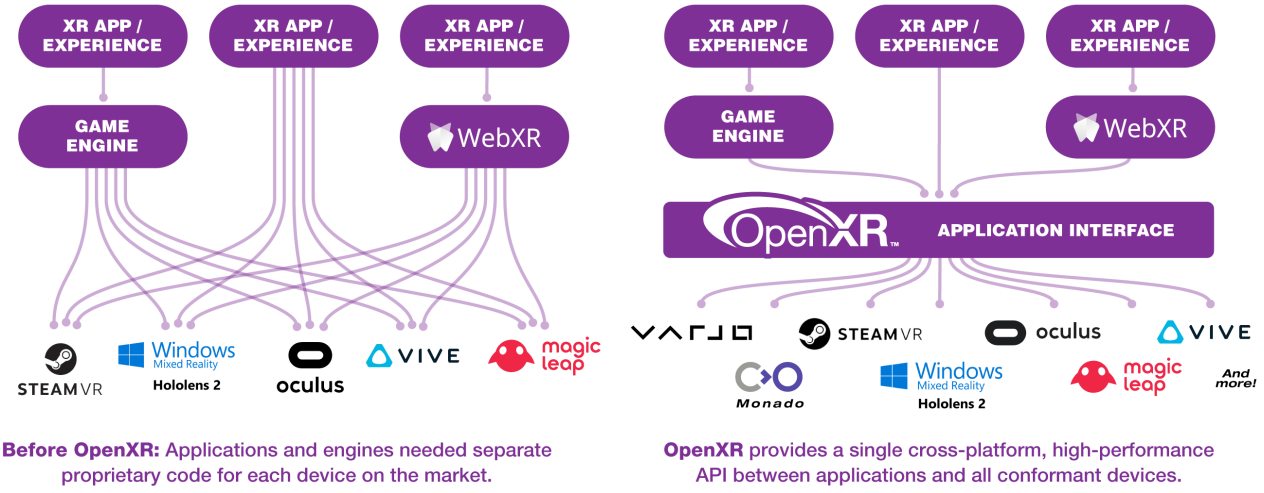

Figure 3.1: Before OpenXR on the left, and after OpenXR on the right (Khronos Group, n.d.).

Since OpenXR only serves as an API between Unity and the hardware, actions like moving the camera with your head, picking up objects and locomotion must be programmed manually. The XR Interaction Toolkit is a framework that does all of this and more, providing developers with easy to use and modifiable scripts for handling interactions and movement as well as premade player controlled rigs.

3.1.3 High Definition Render Pipeline

The existing project uses the High Definition Render Pipeline. According to the Unity manual: "A render pipeline performs a series of operations that take the contents of a scene, and displays them on a screen." (Unity, 2022).

The High Definition Render Pipeline, or simply HDRP, is one of the render pipelines Unity provides. It is designed to take advantage of cutting edge technology to create high fidelity graphics (Unity, 2022). The increase in visual fidelity does however come with a performance cost. HDRP also utilizes compute shaders which require compatible hardware. For this reason, mobile devices including mobile VR headsets like Oculus Quest 2, are not supported build targets.

3.1.4 Universal Render Pipeline

The Universal Render Pipeline, or URP, is a render pipeline with a focus on performance and device compatibility (Unity, 2022). It provides developers with easy to use workflows, and optimized graphics allowing developers to more quickly create efficient and visually impressive games. Unlike the High Definition Render Pipeline (HDRP), URP supports building games for every platform Unity targets which includes Android, which most mobile VR headsets run on.

Converting the project from HDRP to URP could cause issues with the currently used assets. According to the Unity manual: "Porting your project from HDRP to URP and vice versa is possible, but it is not a 1-click operation and will require manual rework of the lighting, the materials, and the shaders!" (Unity, 2022).

3.1.5 Discussion of alternatives

Sticking with Unity 2019.3, SteamVR and HDRP would require very little project setup, which would leave more time for development of gamification elements and environment. However it could lead to a less flexible and more restricted development process down the line, as it prohibits the use of newer technologies and frameworks. Most importantly it would completely rule out mobile VR, for both present and future VR headsets.

Updating Unity, switching framework and render pipeline would significantly increase project setup time, and could carry unforeseeable problems. It does however have several advantages like mobile VR support, immediate compatibility with almost all modern VR headsets, better performance, better framework and plugin support and better documentation.

3.2 Chosen solution

3.2.1 Technical Solution

Considering the extra flexibility updating the project might provide, the choice was made to update the project to Unity 2021.2, and to switch to the Universal Render Pipeline. The game from the previous bachelor group does not run at the recommended frame rate of 72 frames per second , and switching to URP will most likely lead to a noticeable performance increase (Unity, 2021). While it is uncertain whether the game will have good enough performance to run on current generation mobile VR headsets, a big factor in the decision was allowing for the possibility of a mobile VR port now or sometime in the future.

SteamVR will be replaced by OpenXR and the XR Interaction Toolkit. The industry is quickly adopting OpenXR, and we see that continuing in the future. If that prediction is correct, the game would be playable on all upcoming VR devices without having to change or update anything. SteamVR needs more maintenance. The plugin already from the start had to be updated in order to use it with the Oculus Quest 2.

3.2.2 Chosen features

Number one priority will be to finish the required functionality to the game. This will include fundamentals like the XR Rig, which handles head tracking, player movement and hands, as well as the necessary gameplay elements like the dialogue system.

Once these elements are added, the team plans to add a wide array of features that is going to make the game more interesting. Table 3.1 lists some of the features planned to add at this stage of development. A full list of features and non-functional requirements can be found in the vision document (See Appendix A). All features might not make it into the final product due to time constraints.

Table 3.1: *List of features*

Feature	**Description**
Interactable objects	Different objects the player can pick up, inspect, throw and possibly use for something.
Notes	A form of interactable objects that contain some piece of information relating to the story.
Notebook	Hosts the UI for the dialogue system.
Animations	Animations for the characters and for surrounding vehicles.
Improved 3D environment	Includes improving the terrain textures, adding more plants, foliage and rocks, modifying the terrain shape to more accurately represent the terrain at Fjell, removing unnecessary parts of terrain. Adding new objects, buildings, bunkers etc.
Minefields	Populate the area with some form of danger encouraging players to stay on the path.
Timer	Add a timer in order to motivate players to not mess around too much.
Rewards	Gives the player extra time if they complete a small task.
Achievements or score	Score relates to how much information the player has gathered both from dialogues and from inspecting notes and objects.
Facial animations	Synchronize the movement of the mouth to the spoken audio.
Remodel objects	Remodel some of the photo scanned objects in order to reduce polygon.
Intro and outro	New intro scene where the player drives towards the fortress.
Visual Effects	Visual effects like smoke, embers, dust etc.
Performance optimizations	Optimize performance through methods like light baking, occlusion culling and batching.

How the player will interact with the different functionalities are explained in the requirements documentation (See Appendix B).

3.3 Chosen tools

Unity
Unity is a 2D/3D cross - platform game engine used for both game development, animation and simulations. Unity was chosen as the main development environment mainly because the previous bachelor groups used it. This allows us to pick up where they left off without having to remake everything from scratch. Unity has good support for both standard VR and mobile VR.

Unity scripts are written in C#. The team has experience in Java which is very similar.

Blender
Blender is a tool for modeling, texturing, animating and simulating (Blender, 2022). It is used to create handcrafted 3D models for the project. There is basic experience with this application within the team, and it is free to use. Blender was used by Media Lab for some of the existing models. The team has access to the original blender files and can through Blender make modifications to them.

Mixamo
Mixamo is a website from Adobe that offers a wide variety of animations and character models for free. It also offers automated character rigging for humanoid characters. These can be imported from handmade Blender created characters or models from Mixamo (Mixamo, n.d.).

Krita
Krita is an open source free painting program (Krita Foundation, n.d.). In this project the program will mainly be used for tweaking textures such that they correlate with the respected model, for example scaling the texture or altering its basecolor.

xNode
xNode is a very powerful and intuitive node editor framework ideal for coding your own dialogue systems, state machines, procedural generation, behavior trees etc. (Brigsted, 2021).

Oculus Lipsync
The Oculus Lipsync plugin for Unity will be used for handling mouth movement for the NPCs during dialogue. Oculus Lipsync analyzes the audio and maps them to mouth gestures or expressions called visemes (Facebook Technologies, n.d.). This plugin requires 15 different visemes (or mouth expressions) to be defined on a model in the form of blend shapes. Not all models have these visemes. For this reason Oculus Lipsync is used in conjunction with SALSA LipSync Suite.

SALSA LipSync Suite
SALSA LipSync Suite is a plugin for Unity that provides high quality and fast lip sync approximation without phoneme mapping (Crazy Minnow Studio, 2022). It is used as it does not require nearly as many visemes as Oculus Lipsync, and the visemes used are completely customizable, meaning it can be used on nearly all characters, as long as they have more than one mouth blendshape defined.

FL Studio 20/Logic X Pro
Both FL Studio 20 and Logic X Pro are digital audio workstations used for audio sequencing and mixing. In the context of this project they are used to combine and alter sounds for use in the game, as well as recording of temporary voice lines.

3.4 Project methodology

Developing a game takes time, especially with limited resources. This, combined with the group's lack of previous experience in game development means there is a considerable risk of the game not being finished in time. To make sure the end product is as good as possible, a realistic and well structured project plan must be established, as well as a solid development methodology. This is used in conjunction with regular testing with the client, to make the best possible product.

3.4.1 Development methodology

Agile

While the project does not use a specific framework, the agile methodology is central to the way the project is operated. Key points are continuous delivery of working software, welcoming changing requirements, simplicity of solutions and code, and regularly reflecting on the efficiency of the development process. Key points are taken from the Agile Manifesto (Beck et al., 2001).

Weekly client meetings

Meetings discussing weekly progress are hosted at the end of every week. Each team member will give the client a rundown on what they have done, and present any short or long term goals they work on. This is an opportunity for the client and the development team to discuss possible solutions as well as new ideas.

Iterations

Every two to four weeks a running, working version of the game will be built, and made available to Media Lab for testing. These iterations will contain features defined in the project plan, with most important features being prioritized. The first iteration will likely take quite a bit longer than subsequent iterations. This is due to the iterations following the minimal viable product strategy, resulting in the first iteration needing to implement a lot more features to adhere to this principle.

Minimal Viable Product

Seeing as time is the project's primary limitation, it is important to spend it wisely. The aim is for every iteration to be a playable version of the game. A playable version must contain the necessary gameplay elements that enables the story to be told, at least to some extent. Every iteration builds upon the next, adding new features and refining existing ones, making the user experience better.

Weekly development team meetings

At the start of every week, the team hosts a short meeting where each team member informes the team on which features they will work on. This allows for a discussion around prioritization of features, and prevents multiple members working on the same problem, leading to inefficient time management. This also prevents team members from not carrying their load, as they have to present their work at the client meeting at the end of the week.

3.4.2 Project plan

The group has been given a lot of freedom to themselves determine what features to implement. The client might however request ideas and features during development. For this reason, the priorities for the different features are subject to change during development. Figure 3.2 shows the initial plan of when to develop the different features. The project plan serves as a guideline of what features to implement. The team will discuss internally at the start of every week what features to implement with a preference towards important key features and features requested by the client.

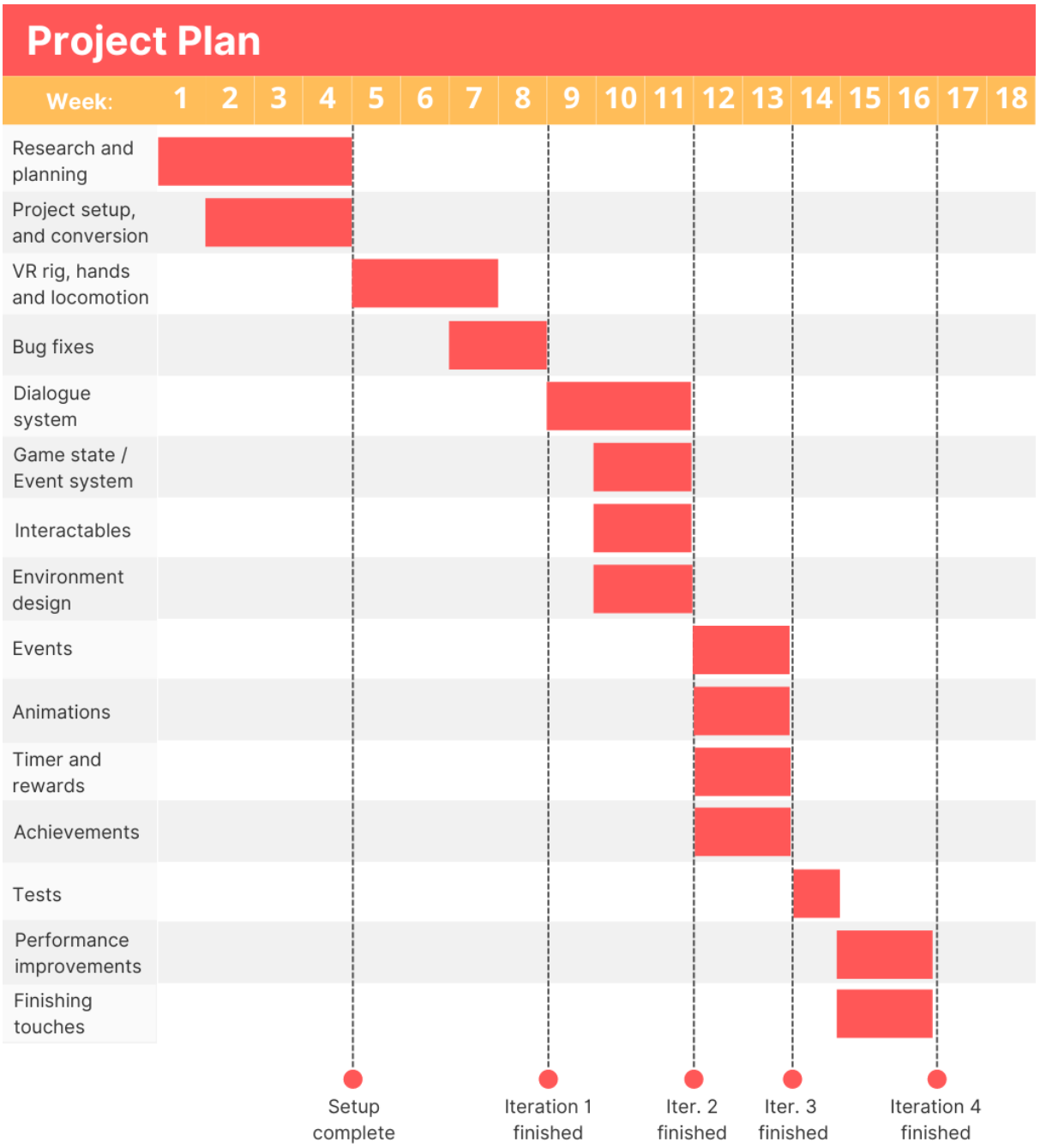

Figure 3.2: Gantt chart of the initial project plan.

The project is composed of a project setup period, followed by 4 iterations. Each iteration is an entirely playable version of the game, however in terms of certain functionality they are

different. Even though the project plan groups the different features into specific iterations, the features are intended to be polished, or even changed based on feedback until it has reached its desired state. The goal is for every iteration to improve the user experience, or the learning outcome.

According to the project the planning phase is limited to the first four weeks, however this is mainly to identify what tools to use and what features to implement. While some features are already planned to a large extent, most will be planned during the course of development, and communicated during the weekly meetings. Sketches like Figure 3.3 can help to communicate planned features with the client.

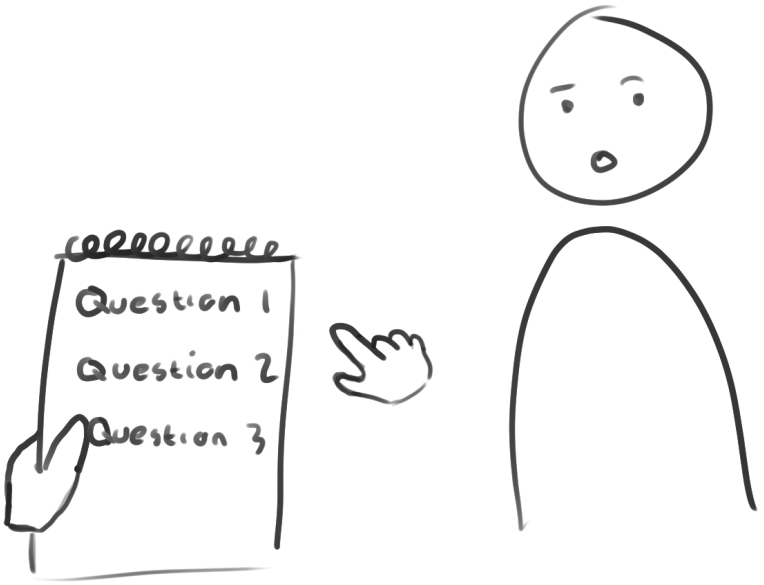

Figure 3.3: Early sketch displaying the dialogue system's notebook UI.

The team aims to communicate features early, and regularly in order to take advantage of the principles of Agile Research. Agile Research is an approach that favors gathering information and input early and often allowing developers to test their ideas and adapt to changes more easily (Knowles, 2020).

Iterations 1-3 each have well defined goals in terms of features. Between iteration 3 and 4, there will be in-house tests, which allows for an opportunity to make changes. Iteration 4 is dedicated to making use of the feedback gathered from the tests, as well as finishing touches, and unfinished features left over from previous iterations. Figure 3.4 shows when the team plans to start developing the different features.

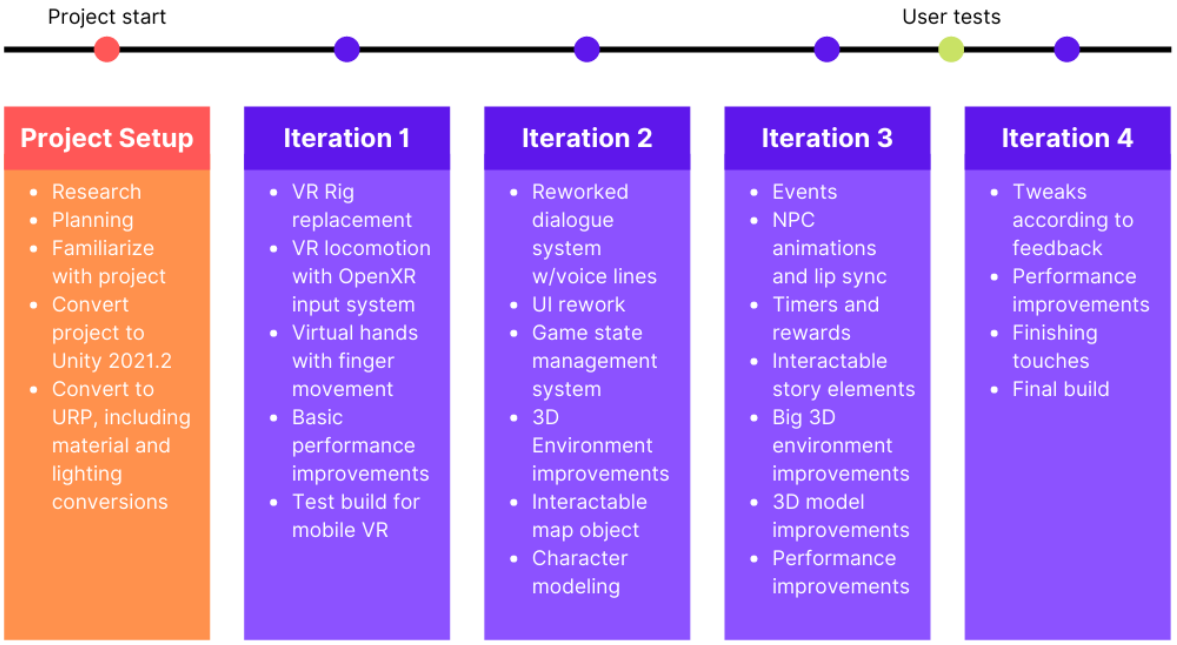

Figure 3.4: List of most important features per iteration.

Iteration 1 will in terms of gameplay be almost identical to the received project, but will be updated and converted as described in chapter 3.2. It will also feature a build for Android in order to test the games performance on the Oculus Quest 2.

The amount of work is too large to be compressed into a single diagram. Some smaller features are left out of the project plan, and can be used to fill empty time slots. These are functionalities that are not considered necessary, or high in priority, yet could be added if time allows. All features can be found in the vision document (see Appendix A).

User tests will be conducted after iteration 4. These are described further in Chapter 3.5: *Evaluation*. These user tests will consist of gameplay tests conducted by HVL Media Lab.

3.4.3 Risk assessment

There are several events that could end up hurting the development process. Time is of the essence, and being able to foresee and mitigate risks could prove to be crucial. Table 3.2 highlights some of the risks associated with the development process, the probability and consequence and what measures are in place to prevent them.

Table 3.2 *Risk assessment table. The risk product is the product of the probability of the event, and its consequence should it happen. A higher number, or warmer color displays a higher total risk, and needs to be addressed appropriately.*

	Event / Risk	Cause	Probability	Consequence	Risk product	Measures
1	Outdated technology	New and better technology is taking over	Low (2)	Medium (3)	6	Make sure we use the most up-to-date tools, and avoid using tools with bad support
2	Technical issues	Unforeseen problems with project and assets	High (4)	Medium (3)	12	Use the resources available to us for troubleshooting. Factor in time for research and bug fixing when time scheduling
3	Too high absence	Illness, obligatory assignments in other courses and other events	Low (2)	Low (2)	4	Good project management, and options to work from home
4	Uneven and overlapping workload	Poor communication and cooperation.	Medium (3)	Medium (3)	9	Actively using our project management tools, like teams, and teamgantt to mark which task you are working on.
5	Project is not finished in time	Too big ambitions, poor time management or inefficient project management	Medium (3)	Very high (5)	15	Using an iterative process to always have a playable version ready. Prioritize important functionality.

3.5 Evaluation plan

3.5.1 Performance Tests

Performance will be continuously monitored as features are added or removed. For a typical software project, this might be a bit excessive, but for this project it is important. The game in its current state is most likely not playable on any current generation mobile VR headset. The user would benefit greatly if the game was able to run on mobile VR as it would eliminate the need for a powerful gaming machine. This would significantly reduce the cost for users, and expand the game's reach.

The goal of the early performance tests is to determine whether or not it is possible to achieve this, and what it would take to make it happen. The performance tests will also shape the team's priorities. Should it become clear that a well running mobile VR build is improbable, the team can prioritize other features, like higher visual fidelity.

Tools for performance monitoring include the Unity Profiler, Frame Debugger and Memory Profiler. Monitoring the VR headsets metrics is done through the Oculus Developer Hub. The device logs are also available through this application, which makes it useful for identifying device specific errors.

3.5.2 Functional Tests

Features are internally tested by the development team during the week. These are to verify that the features work as the developer intends. At the weekly client meetings, the features worked on that week will be displayed with the client present. This allows the client to approve, or suggest improvements to the added features. These tests are mainly to ensure the team, and client are on the same page in terms of the game's functionality. It also helps confirm that the clients requirements are still fulfilled. Other topics of the meeting are what problems the team are facing and next steps going forward.

3.5.3 Alpha Tests

Full gameplay testing will be done by the team, and the client primarily. It will start with a system test, verifying that the software works on a variety of different machines. Then it will be followed by an acceptance test. This will be a complete playthrough of the game, including all added features. This is done before iteration 4, giving the team time to fix potential bugs or missing features.

3.5.4 Beta Tests

When development is complete, the team and client will conduct beta tests. These tests will test the educational program as a whole. Subjects will be external testers, in different age groups and with different levels of VR experience. The subjects will first conduct the onboarding, The onboarding consists of a video prepared by Media Lab. They then get to play the game for a period of time. After or during the playthrough they will be asked a list of questions as shown in Appendix E. The questionnaire is primarily focused on the user experience, but users can also express their thoughts on the story as these comments could benefit the client. The goal is to identify flaws in the execution of the system as a whole, and to identify potential aspects of which the VR application fails to provide a good experience.

4. Design and Development

This chapter will describe the finished solution in an appropriate amount of detail. As this project expands beyond a system development project, it will also contain a brief rundown on some of the workflows used.

4.1 Architecture

This subchapter aims to explain the overall architecture of the game, in reasonable detail. In order to understand the architecture, it is recommended to have a basic understanding of how Unity works.

4.1.1 Basics of Unity

A unity project consists of one or more scenes. A scene is a 3D space in which objects, terrain, lights, audio sources etc. are placed to make an environment.

Every single entity within a scene is a GameObject. These GameObjects act as containers for different components. GameObjects can be nested within other GameObjects, making it a child of the root GameObject. All GameObjects come with a Transform component that controls the GameObjects position in world space, or in relation to its parent. As mentioned previously, every single entity is a GameObject, which means that even global scripts running in the background, have to be attached to a GameObject, which is placed somewhere within the scene.

Components are classes that can be added to a GameObject. Unity comes with a wide variety of predefined components that makes development easier. Examples are Mesh Filter to define a GameObjects mesh, a Mesh Renderer for adding materials to the mesh, colliders, audio sources and more.

Custom components are the main way of adding custom functionality to the game. Components are C# classes. Essentially all components inherit from the MonoBehaviour class. This gives access to engine specific functionality and allows the component to be attached to a GameObject.

Components can interact with other components, even on other GameObjects. For example, a component on the player object, can change the color of another object, as long as you can obtain a reference to that other object.

4.1.2 Architecture Overview

When explaining the game’s architecture it is mainly referring to the elements that dictate the game's flow. Due to the nature of the story being linear, players must speak to characters in the right order, as some dialogues build upon information gained in previous interactions. The solution for this was to track the player's progress, and to both restrict and guide the player to the correct character. This subchapter will briefly explain the key components and how they interact. The different system will be explained in more detail in chapter 4.2.

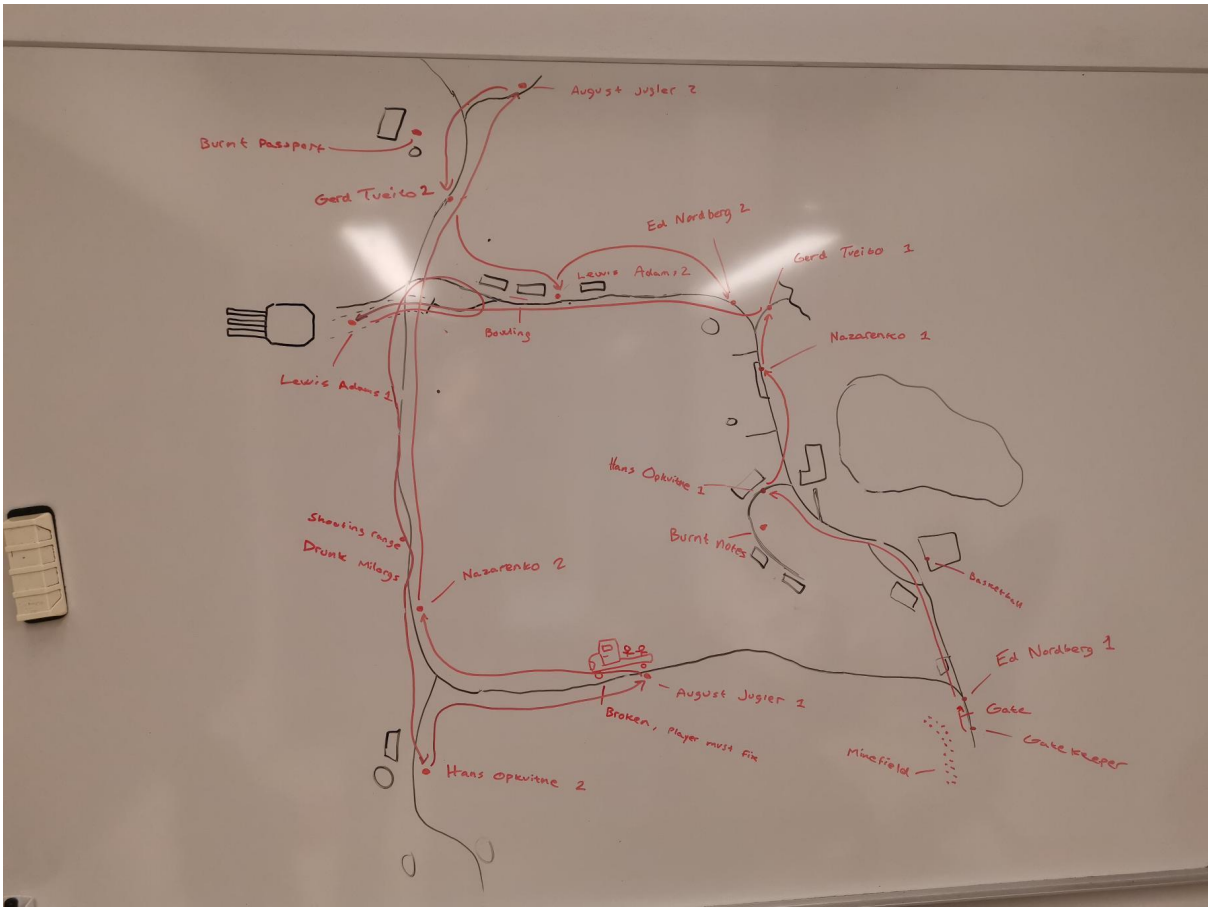

Figure 4.1: Planned sequence of character interactions, as well as placements of key objects.

The **Event Manager** controls the game's state. The event manager's state can be accessed and changed by other components. The event manager will notify other components that are subscribed to it that the state has changed, and what the new state is. Other components can declare a function to be run in the event of a state change. A component can also check the current state without waiting for an update.

The **NodeParser** is responsible for handling interactions with NPCs. Every dialogue interaction has a NodeParser instance. The **PlayerSensor** component activates the notebook, and starts a selected NodeParser's dialogue when the player is near an interactable NPC. A dialogue can only be started if the game is in the correct game state. There are in total 14 game states.

The **notebook** is a GameObject that contains the UI elements for the dialogue system, including text, buttons and visual effects. These are controlled through the NodeParser components.

The **MapHandler** component listens for changes in game state and updates the map GameObject accordingly. The map always points to the next interactable NPC.

The **NPCHandler** component is the script primarily responsible for hiding and showing NPCs depending on the game's state. Every character could have their individual event listeners for game state changes, and react accordingly. However having it all in one script makes it easier to manage, and lowers the amount of subscribers to the event system. Over the course of development, the NPC handler was extended to handle more objects than just NPCs.

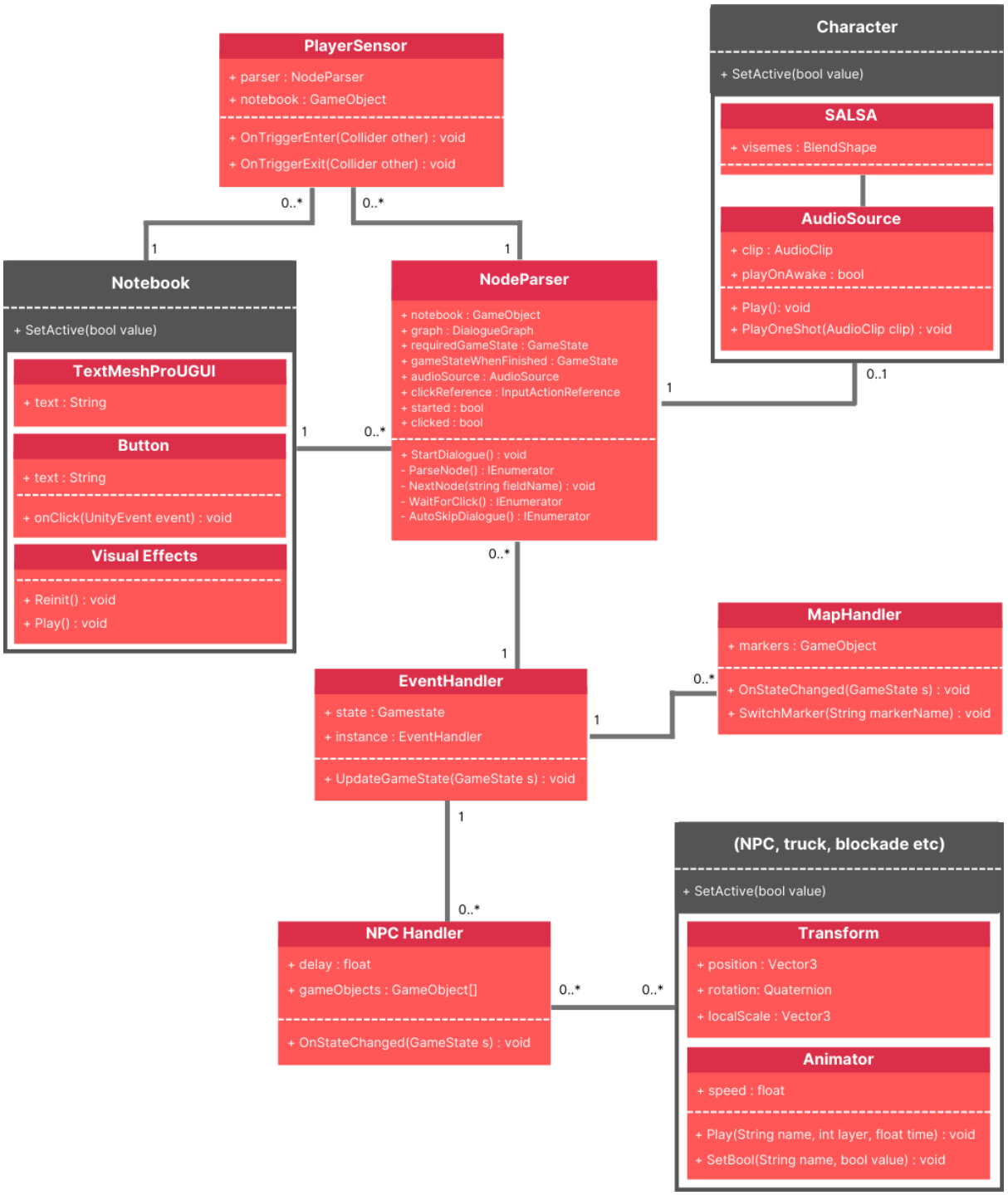

Figure 4.2: Modified class diagram of the most central features.

The class diagram displayed in figure 4.2, shows the connections between the most important components. GameObjects are in the diagram displayed as gray containers with components within them. All components are attached to GameObjects, but only components where GameObject methods are used are displayed in the diagram. A common use of these methods are to activate and deactivate the object.

4.2 Gamification

4.2.1 Dialogue System

The inherited project featured a simple dialogue system created with VIDE Dialogues (Version 2.2.2, Albazcythe, 2018). This dialogue system was remade from scratch using the xNode framework for Unity. The xNode framework makes it easier to create custom node based systems for Unity. Developers can make their own custom nodes, and connect them in a node graph. A script can then parse through the nodes one by one, processing the data of each node. xNode does not come with any premade nodes or parsers, but supplies developers with tools that make creating these easier.

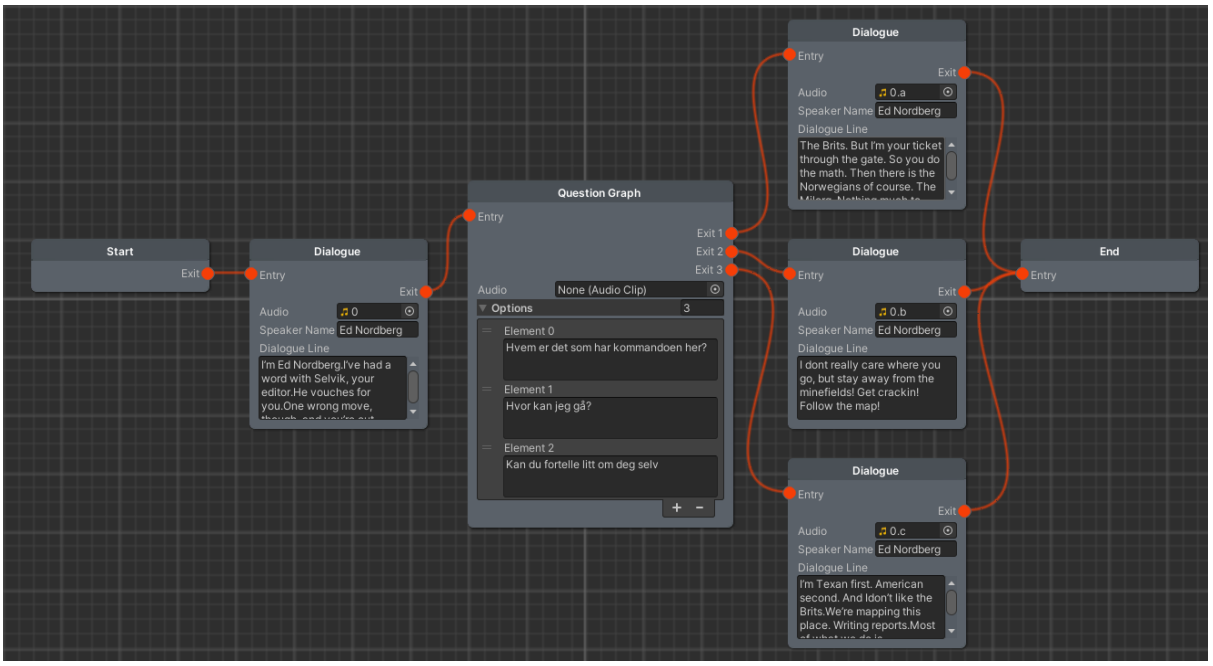

Figure 4.1: An example dialogue graph. The graphs used in the game are typically much bigger

The new dialogue system consists of one dialogue graph per dialogue interaction. Every dialogue and every combination of questions within one interaction is located within one graph. In the game you meet each NPC two times, this means there are in total 12 dialogue graphs in the game. These graphs contain a start node, followed by a combination of dialogue nodes, question nodes and ending with an end node. New nodes can be added easily by right clicking the graph and selecting the desired node. Nodes can be connected by dragging the output port towards an input port creating an edge between them.

A dialogue node features an audio clip field for the dialogue audio, and a field for the text version of the dialogue. In the script these are represented as public variables. It contains a single input and output port.

Question nodes contain a list of questions. The amount of questions is decided in the node graph. The node does not have an upper limit to the number of questions, but the current parser script will only display as many questions as can fit on the UI.

The node parser script is responsible for cycling through the nodes and processing their data. It starts by finding the node labeled start, and uses that as the starting node. The name of the

nodes determine the action to be performed. When a question node is parsed, the parser tells the UI to show the buttons, and hide every other UI element. The buttons are then populated so that each button represents one question. Each button is linked to a specific output port, and fires an event when clicked, telling the parser to follow that particular output. The buttons are then hidden.

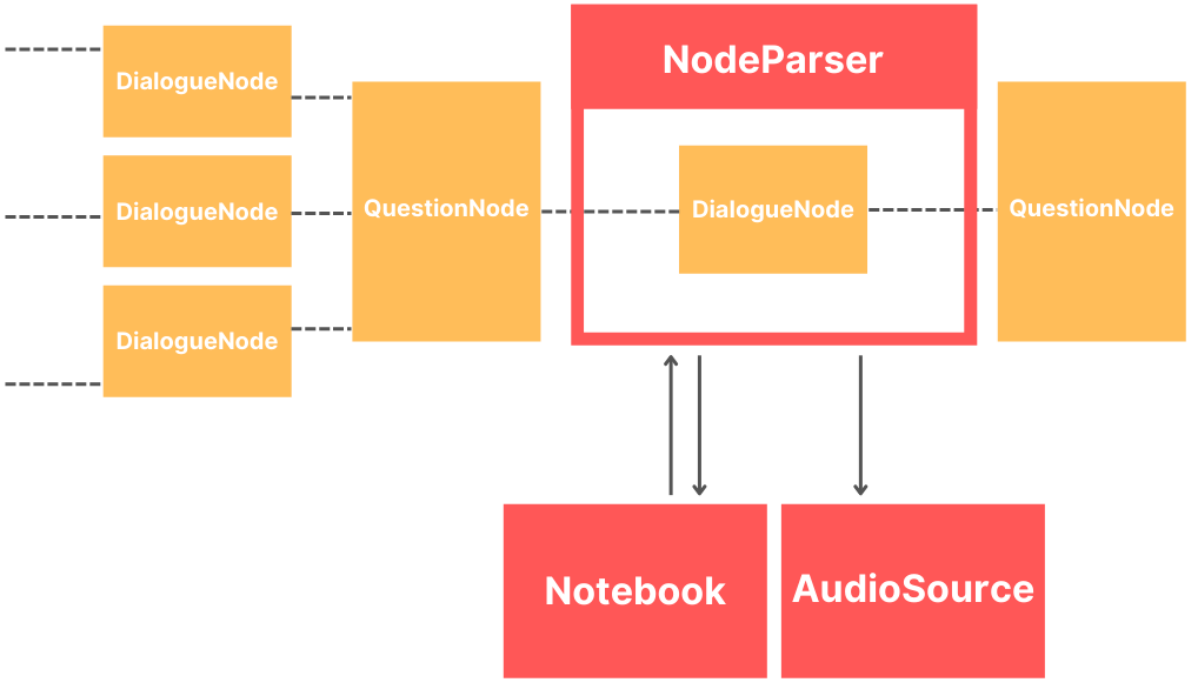

Figure 4.2: Node Parser visualization

The dialogue UI takes place on a physical object that appears in your hand when in proximity of an interactable character. The notebook is locked in the character's hand to ensure the player does not accidentally lose it, as seen in Figure 4.3 (p. 30). This was also done as a measure to prevent grip fatigue during long interactions. The layout of the notebook varies depending on which node is being played. In the event of a dialogue node, the notebook will display two fields: a header with the speaker name, and a text field for the dialogue itself. In the events of a question node, it will hide the speaker and dialogue fields and instead display up to three buttons. The number of buttons corresponds to the number of questions in the node, with an upper limit of three questions.

Figure 4.3: The notebook when a question node is being parsed.

4.2.2 Event System

In order to determine the game flow and behavior from user input and interactions, the group built an Event System/State Machine. In Fjell Festning, each state changes after the player has finished a dialogue. There are 12 dialogue interactions, therefore 12 states were added, as well as the “StartGame” and “EndGame” state. When a player enters a new state, it gives the team the opportunity to change the game's behavior according to the state the player is in. When the player is conversing with the various characters in the game, the state will change after each dialogue.

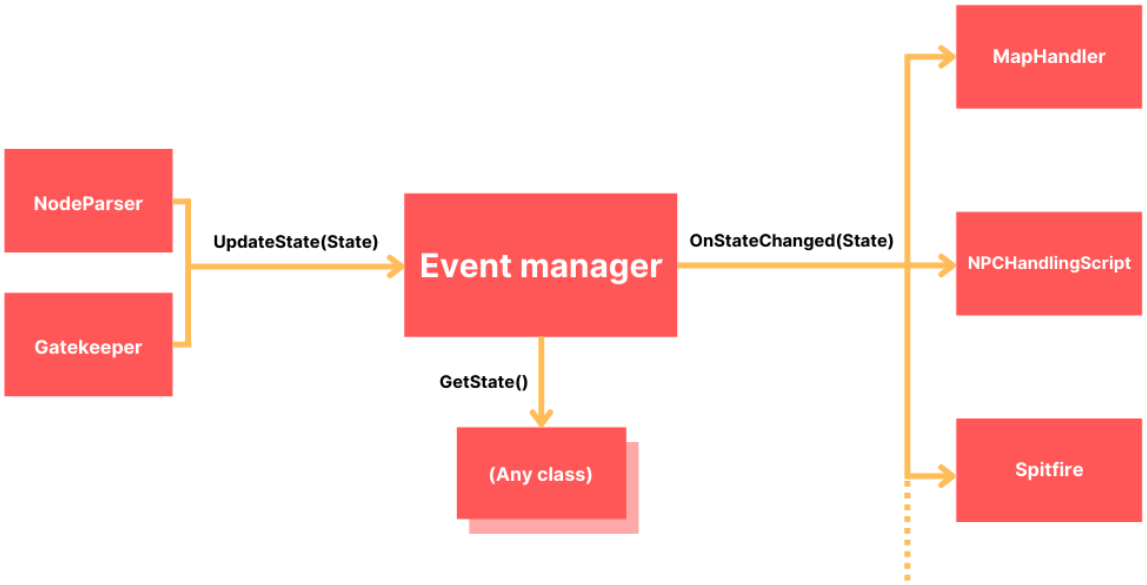

Figure 4.4: Structure of the event system

The state machine also helped structure the story’s narrative. A player cannot interact with a character if it hasn’t interacted with a required character beforehand. For example, the player

can not start a conversation with Gerd if the player didn't converse with Nazarenko. This allowed the group to have a set of rules and constraints for what the player can and can not do during runtime. Letting the player interact with all the characters in the scene without having these constraints based on the state the game is in would have ruined the story line and confused the player. Figure 4.4 illustrates some of the GameObjects that publish and subscribe to the Event Manager, where objects to the left publish a new state and the objects to the left subscribe to the Event Manager.

4.2.3 XR Rig, Interactors and Interactables

Interactions with the environment is a key feature in any virtual reality experience. VR has three distinct characteristics: Interaction, immersion and imagination (Li, 2019). In order to make the game as immersive as possible, interactions with objects and characters is essential.

The XR interaction toolkit package is a high level, component based interaction system for creating VR and AR experiences (Unity, 2022). The group used the toolkit's premade XR Rig in order to quickly create a movement system. The XR Rig acts as the player's eyes and ears in the virtual world (Unity Technologies, 2021). The rig also features two Ray Interactors in the position of each hand, allowing the player to grab objects by pointing at them and hitting the grab button. The group also added a feature called snap turn, allowing the player to turn using the joystick instead of having to physically turn around.

Figure 4.5: Two interactable objects, the pistols and beer bottles

The XR Interaction Toolkit contains the XR Grab Interactable component, which when attached to an object in the scene allows it to be grabbed and held. Grab Interactables can be grabbed by any Interactor, like for example the player's hands. For realism and functionality, only some objects should be grabbable. For example it would not make sense for the player to grab and hold a car.

For example, Figure 4.5 has multiple interactable objects, two pistols and several bottles. The pistols can be used with a max limit of 7 bullets per weapon and lets the player shoot down a gun range with targets. When the magazine is empty, an audio clip will play indicating that the pistol is empty. The bottles can also be grabbed. These were added to emphasize how chaotic Fjell Festning was at the time, where weapons were laid out in the open, and guards were drunk on duty.

4.2.4 Inventory

Having somewhere to store your items is extremely helpful as certain interactables are necessary for traversing through the game, and constantly grabbing them with your hand becomes very tiresome after a while. To improve the user experience, the team decided to create an inventory system. A UI based inventory was considered, but instead the team landed on a physically based inventory system as it increased realism and was also more satisfying to use.

The inventory system consists of a belt attached to the player. The belt follows the position of the camera, with a custom script component attached that calculates the belt's rotation to always point with the camera. Without the script, the belt would stand still facing one direction independent of the camera's position and direction, ruining the immersion of wearing a belt.

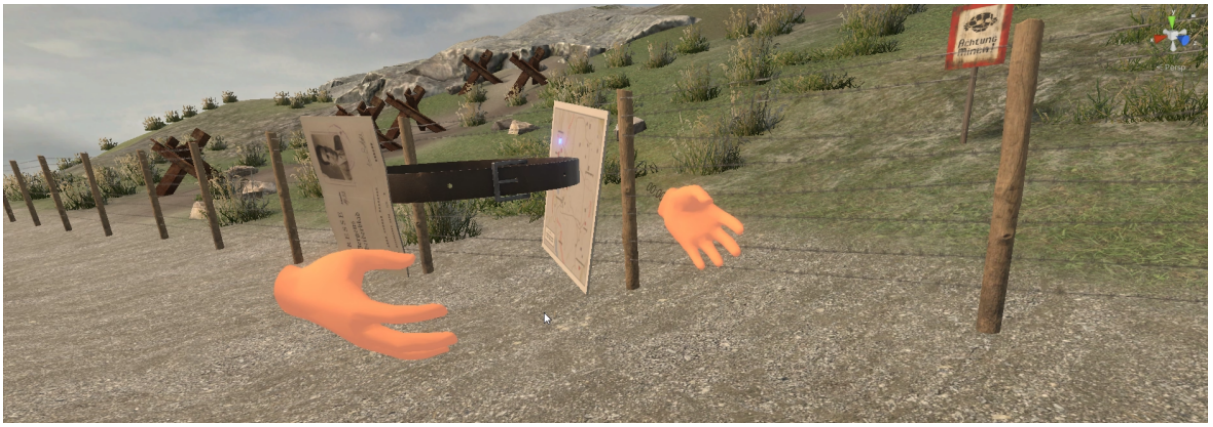

Figure 4.6: Inventory

The belt itself consists of two Socket Interactors from the XR toolkit that can hold Interactable GameObjects (Unity Learn, 2021). Any object that contains a grab interactable component can be placed inside of a socket. The sockets will hold the item in place and as such will act as the player's inventory system. The group chose to have two Sockets for the inventory, one for the map and another for the player's identification papers. Altering the pivot point of the Socket made it easy to position the objects when they snap onto the Sockets.

The map is an interactable gameobject, helping the player navigate through the story. It has 12 markers each pointing to a different character interaction. The map's state is based on the current game state, using it to enable / disable markers. When the player starts the game, only one marker is lit up, indicating that there is an interaction. When the state changes, a new marker is lit up. Figure 4.6 shows the first interaction with only one blue marker glowing on the map.

The group was slightly worried that the players won't understand the whole concept of the inventory system. A gatekeeper was then added to familiarize the use of the belt before the story takes place, introducing VR object interactions. Here, the gatekeeper walks towards the player and asks for identification papers by playing an audio clip made by the group. The player here must find the papers and give them to the security guard. When identified, the gatekeeper will give the player a map, which then opens the gate for further exploring the game.

Figure 4.7: The gatekeeper checking the journalists identification papers.

The gatekeeper was created entirely from scratch by members of the group, using Blender to model and Mixamo for animation and rigging. Creating this character and the associated interaction is perhaps the most complex task completed by the group, as the interaction is fundamentally different from the rest. This little interaction plays a big part in helping the player learn the controls and familiarize themselves with the inventory system and map.

The character was created with reusability in mind. The character is performance optimized, cycles between idle, walk and run animations based on movement speed and even has mouth expressions that can be controlled through a lip sync plugin. The character also moves his head to look at the player.

4.2.5 Animations

Unity has great animation features that make it possible to implement animations with a simple set of tools. The group took advantage of this and created animations for characters, both hands and vehicles to make the game feel more lively.

Animations were made by animating the targeted object with recorded keyframes for each movement. In order to instantiate an animation, an Animation Controller must be added to the object. An Animation Controller arranges and maintains a set of Animation Clips (animations) and Animation Transitions (Unity, 2017). This helps determine when an animation should start or stop. The group can also trigger an animation by the state the game is in or by adding trigger objects around the scene. Figure 4.7 shows the initial state of the object, which is set to “Idle”. When a trigger or state changes during run - time, a script starts the “SpitfireAnimation”. Being able to change these animation clips helped game development since at any given time the animations can be set.

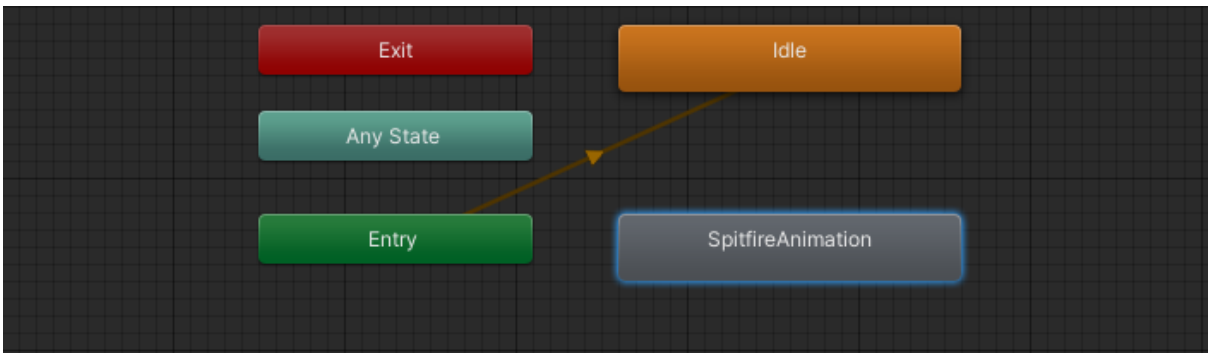

Figure 4.8: Animation Controller

Some animations were made by the in - house animation tool. An example of this is the Spitfire fighter planes that fly past the player at a certain point in the game. The reason for the plane being a Spitfire is because it was a detailed free asset in the Unity Asset Store. There were also British soldiers roaming Fjell Festning, so realistically having a Spitfire flying over the compound could have occurred during 1945 after the war.

For example, the propellers for the Spitfires are animated per frame and are then looped. This creates the illusion that the animation is running continuously. This method was also used when animating the wheels for the German trucks. The other animation clip for the plane focuses on the Spitfires world space coordinates and is animated in a straight line by moving the object's axis from point A to B in a single keyframe. Unity's animation tool lets the transition of moving an object to the desired location seamless and smooth. Combining these two animations make up a realistic airplane flyby. These methods were also used for other dynamic objects in the scene, not only the Spitfire.

Figure 4.9: Spitfire Animation per rotation keyframe

Other animations were imported from Mixamo (Mixamo, n.d.), an Adobe website that specifically offers free professional character animations and character rigging that corresponds to the character's humanoid figure. All animations for the characters are from Mixamo, the team would have used countless hours if these humanoid figures had to be animated by hand.

4.2.6 NPC Navigation

The first interaction the player encounters is a gatekeeper wanting the player's identification before gaining entry into the fortress. Problems occurred based on the gatekeeper's movement

and how he will move towards the player according to the player's position. Animating the gatekeeper's movement would take a lot of time and experience to complete since every keyframe must move a part of the gatekeepers' joints.

A simple solution was adding a navmesh to the terrain and using the navmesh agent component on the gatekeeper. A navmesh is a flat plane showing where an agent can walk. The mesh is generated by creating a mesh above the terrain. The areas immediately around meshes that are marked as static, are subtracted from the mesh if their angle in relation to the rest of the NavMesh is steep enough. This creates a navigation mesh that estimates a walkable surface of the scene (Unity Documentation, n.d.). The navmesh agent is a component you add to a GameObject that uses the baked navmesh information, allowing the agent to navigate through the scene. Through a script you can give the agent a destination, and using the navmesh, it will calculate the fastest route. It can also determine when the destination has been reached.

In the script, the gatekeeper's destination is the player's position. The destination will be updated each frame, since the player can be in a different position depending on where the player decides to position themselves for the interaction. The blue area in Figure 4.10 visualizes where a NavMesh Agent can navigate with the help of baking the terrain.

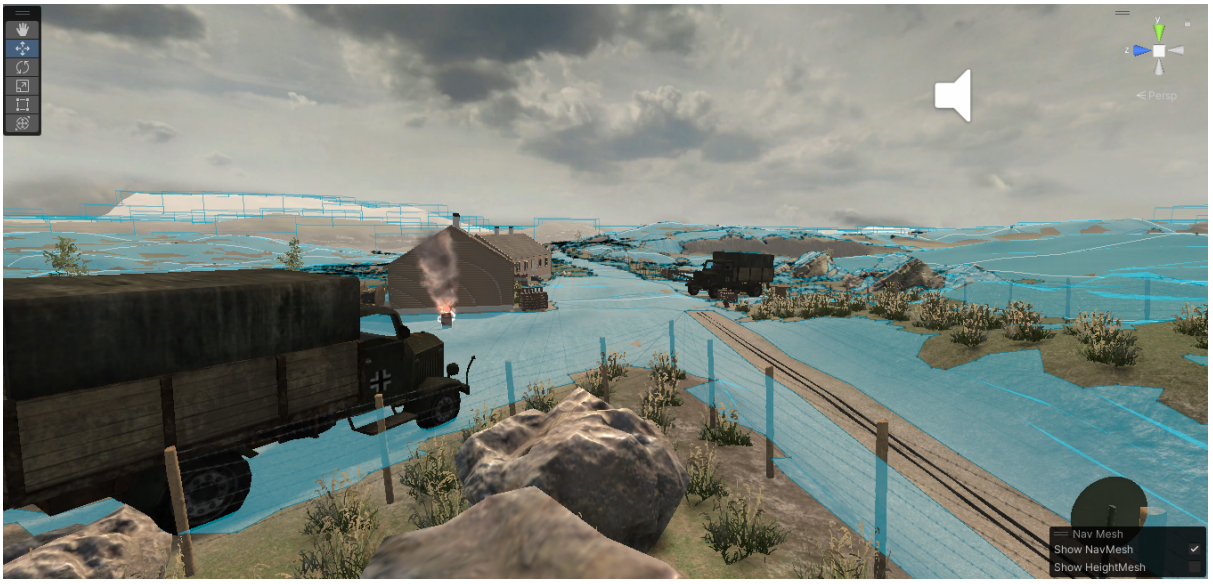

Figure 4.10: Baked NavMesh. Walkable areas are highlighted blue.

The NavMesh will only consider static elements when baking the NavMesh. Since the NavMesh is baked, it means it is pre computed. For dynamic objects the team wished the agent to avoid, they added a Navmesh Obstacle component to it. Every object with this component will allow the agent to identify the moving object. This feature became useful for moving objects like cars etc.

4.2.7 Visual Effects

The smoke, flames and sparks in the game were created with the help of Unity's particle system with the downloadable Unity asset VFX Graph. The asset enabled the use of visual effects with the assistance of a visual node - based logic (Unity Learn, 2021). The use of this tool made the smoke implementation easy and efficient.

Having a visual real time scene window of the particles while developing the smoke made it easy to both visualize and edit the smoke effect while the game is running. Without the VFX Graph, it would be difficult to estimate the scale and light interactions the smoke will have in the end result. The VFX Graph also renders particles on the graphical processing unit (GPU) which lets the developer render far more particles than you otherwise could. The drawback is that the particles do not react to the environment, for example sparks bouncing off the terrain surface and other objects.

Figure 4.11: Visual effect of a burning barrel.

4.3 Environment and Design

4.3.1 Lighting

4.3.1.1 Directional Lighting

Lighting is a uniquely powerful tool when it comes to improving a game's visual fidelity. Directional lighting is a useful light component for creating sunlight in a scene (Unity Documentation, n.d.). In the scene, the directional light behaves as the sun (the main light source) and gives off light wherever the directional light is positioned in the scene. The rotation of the light can be used to determine the time of day.

The lighting was configured in such a way that it matches the time, color and sun position of May, which is the month the story takes place. The initial project's directional light had a completely white color, but now the color is warmer, reminiscent of a morning in May.

4.3.1.2 Spot Lighting

A spotlight is a lighting source that projects light within a specific location and range. The light gives off a cone - shaped area of illumination (Unity Documentation, n.d.)d. In one of the bunkers the player can enter, there are several spotlight objects directing light from the roof of the bunker. Before the spotlights were added, the cave was dull and lacking excitement, with little to no atmosphere. With just a couple spotlights, it helped make this part of the game more ambient and intimidating to enter.

Figure 4.12: No spotlights

Figure 4.13: Spotlights

4.3.2 Nature recreation

In regards to the 3D environment, the goal is to have it look as close as possible to what it did in 1945. The first group did a great job of photo scanning the terrain and added rocks and trees to make it look more realistic. Unfortunately, that group did not have enough time to further improve upon this. The asset pack used for foliage and various nature objects is Meadow Environment - Dynamic Nature. This pack includes trees, bushes, grass, rocks, cliffs, terrain textures, etc.

The team's first goal in recreating the environment was to replace the already existing grass with a more realistic and detailed foliage without losing much performance. GPU instancing helped the game increase the amount of grass around the terrain, making the scene feel more realistic. GPU instancing is a draw call optimization method that renders multiple copies of a mesh with the same material in a single draw call (Unity, n.d.). A wind shader from the previous group was added on both the grass and trees from the Meadow Environment pack. The project client commented on the wind and wanted it removed in the finished product. Performance

wise, this worked well since the game engine won't be doing any calculations for these during runtime.

Previously, the cliff prefab in the asset pack was only used in certain areas of the scene. By scaling these up, these cliff objects could occupy a large area of terrain while only being one object, limiting the amount of objects in the game.

Figure 4.14: Grass and Cliffs

The terrain has multiple textures that blend together creating a realistic template to add objects and scenery to. These textures also focus on creating depth, bumps, roughness and the illusion of lighting. Certain textures such as gravel, have normal maps. These maps help a texture create the illusion of light absorption and roughness. The benefit of including a normal map is that it does not lower the game's performance, even though the object has more detail.

4.3.3 Map/Game design

When filling in the scenes with objects, the main focus is comparing the photos from Fjell with the map design of the game. Using assets that correlate with WWII helps create a realistic environment when compared with old photos from Fjell. Every aspect of the main scene is based off of real pictures from Fjell during the war, this includes buildings, fencing, gates, rivers, etc. These images were given to the team with the camera positions of where soldiers were believed to be taking the picture from. Having this information helped estimate the position of the barracks and where the terrain is rough or smooth.

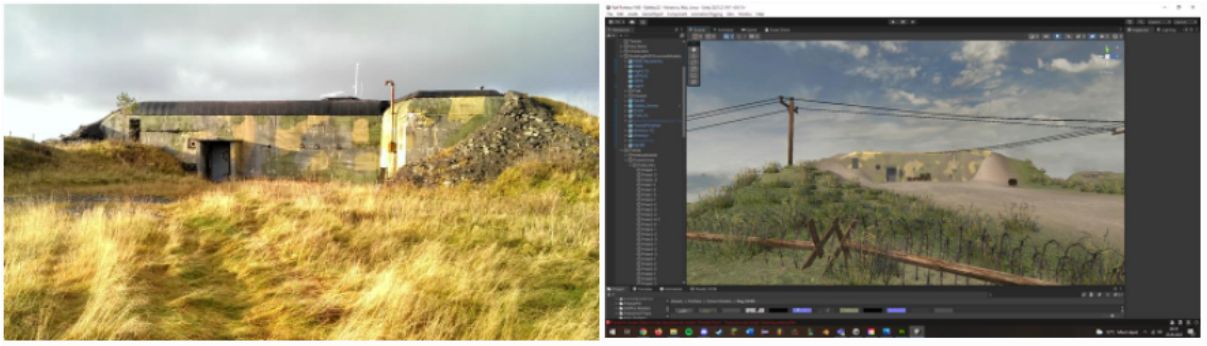

Figure 4.15: Map design with the help of reference images. Reference image is on the left believe it or not (Museum Vest, n.d.)

The flow of the game (Game Design) also needs to be consistent with the map design. Restricting the player to walk in certain areas must match with the fencing and building positions. The aim is to restrict player movement so the player will have to follow the questline, and not move to areas where they get stuck etc. The creation of these barriers requires strategic placement of fencing and other objects to avoid having invisible walls.

Other areas will also open up when a state changes after a dialog. For example, a German truck blocks the player's path before one of the bunker entrances. After entering the bunker and talking to one of the characters, the truck will move and drive off by triggering the truck's Animation Controller, leaving the blocked area open for the player. Restricting pathways and navigating them through map design made it easier for the developer to control the game flow, knowing when and where a player can move at certain points of the game.

Figure 4.16: Truck blocking a path, the truck drives off after a new state

New areas have also been added to resemble the state Fjell was in after the war by examining photos from Fjell in the 1940's. This includes more fencing, a barracks near the canon and a medical bunker. As well as roughness in the terrain with more dirt and gravel. The terrain was photo scanned which visualizes how the terrain at Fjell looks like today, however the terrain in 1945 had more flaus with creators created by mines and dirt like areas where there may have been construction.

When designing the map, it was important to have multiple points of interests (POI's). These locations will present new valuable information for the player or let the players interact with new objects. The previous group had 6 dialogue interactions, whereas this project has 12. This made it possible to create more locations the player will walk through. Some of the new locations are the gun range, a revamped tunnel, the barracks outside the cannon, a truck explosion site, the barracks near the gun range and more.

4.3.4 Scenes

The game consists of four scenes, an intro, taxi, main game and an outro scene. All four scenes make out the whole game. A scene in Unity is where a developer can make content for the

game. A game can have one or multiple scenes, in Fjell Festning there are four. Three out of the four scenes are for visual purposes only, where the player won't be able to move, only absorb information by studying their environments. This sub - chapter will focus on the three other scenes included in the game.

Intro

When booting up the game, the player is introduced to the intro scene. Here, the player is placed in an environment similar to the main scene. The sole purpose here is introducing the player to the story and background of Fjell. A canvas is placed in the scene which displays a video given from Media Lab that explains who the player is and what they will be doing throughout the game. After the video is done, a fade in screen will appear which makes the transition for each scene seamless for the player.

Figure 4.17: Intro scene

Taxi drive

After the transition from the intro scene, a new fade out screen will appear. Every scene has implemented this concept for making transitions comfortable for the player. It consists of making the scene visible by first altering the alpha value from dark to transparent and vice versa. The taxi scene is a 22 second long taxi drive into the gated area of Fjell Festning.

The player is placed in the back of a car animated to drive towards the gate, with a taxi driver in the front seat driving the vehicle. Here, the taxi driver has an audio clip explaining what recently happened that day which builds upon the storyline. It also acts as a bridge introducing the player to the base game.

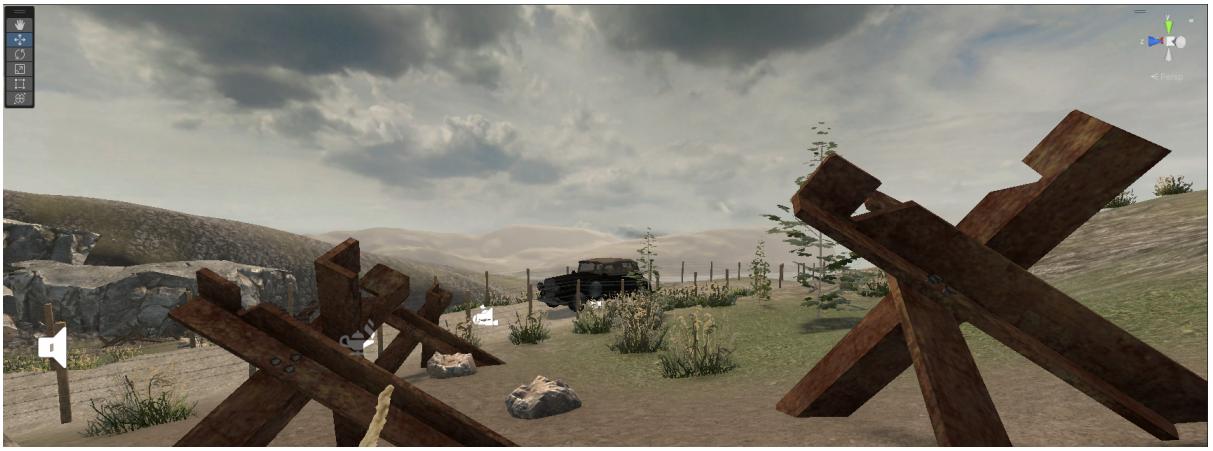

Figure 4.18: Taxi drive scene

Outro
The outro is very similar to the intro scene, in regards to its content. After the player has finished the last dialogue in the main scene, a taxi will be seen driving towards the player. An audio clip of a car honk will also be played, helping the player look for the taxi before the fade script is triggered and a new scene is rendered in.

The player will then spawn in a theater with a canvas, here the player won't be able to move, only rotate. The canvas will display a video given to the team from Media Lab. When the video finishes, it will exit the application.

Figure 4.19: Theater scene

4.3.5 Photogrammetry

Photogrammetry is the science of creating 3D models from photographs. The process consists of taking multiple overlapping photos of an object to then convert and render them into a digital, three dimensional object. The preceding bachelor group created 3D models of various buildings from Fjell festning, such as RegelBau bunkers and the underbelly of the Gneisnau cannon.

Photogrammetry is an exciting idea in theory, but much harder to execute in practice. The challenge is that scans have to be near perfect and in well lit areas to get good results. This is

especially difficult in areas with uneven lighting such as bunkers or even just outside with natural lighting. Less than perfect scans will create unwanted artifacts, this happens as the program tries to fill in blank spaces where the scan is subpar. This in turn creates an abundance of extra vertices, which in turn is tremendously destructive towards the performance of the game.

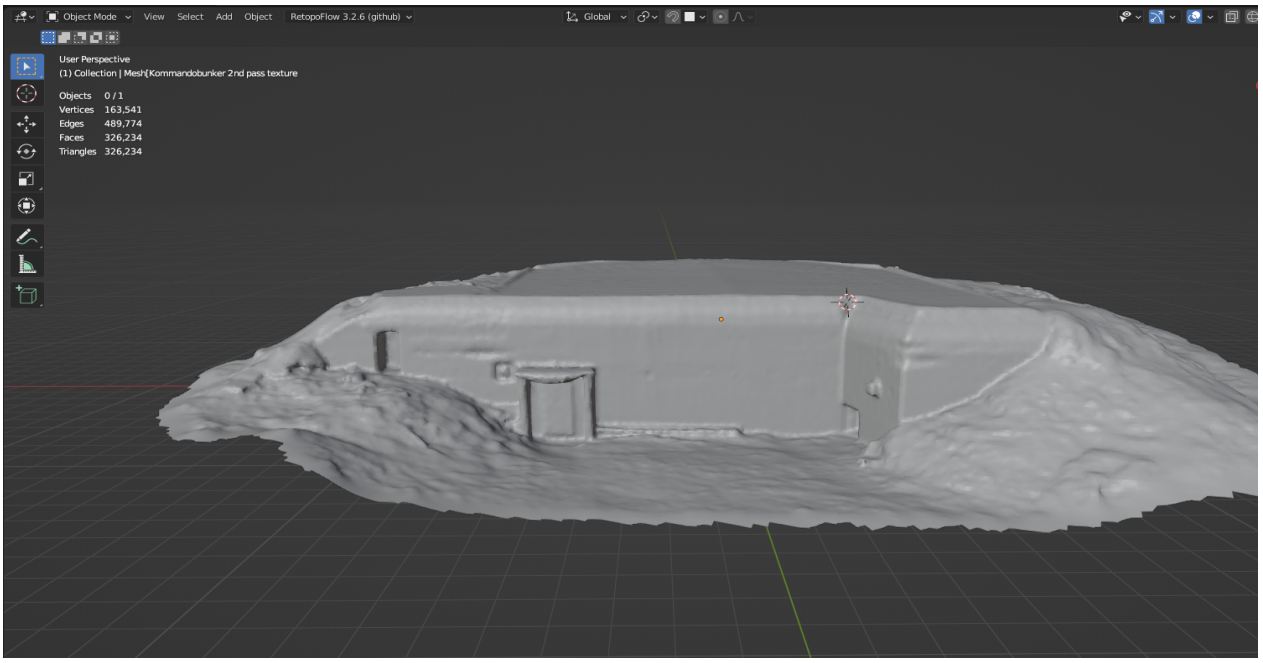

Figure 4.20: Photo scanned commando bunker

Figure 4.20 shows the original photo scanned commando bunker. As shown in the photo the vertex count for this model is a total of 163 541 vertices. A very high count for a model of its size. The solution to this problem was to recreate the model manually in Blender. Tedious work that yielded great results, as shown in Figure 4.21. The total vertex count went down to 1403 vertices, which is an immense improvement of the photo scanned model. The team made a total of three new models replacing the older photos scanned models.

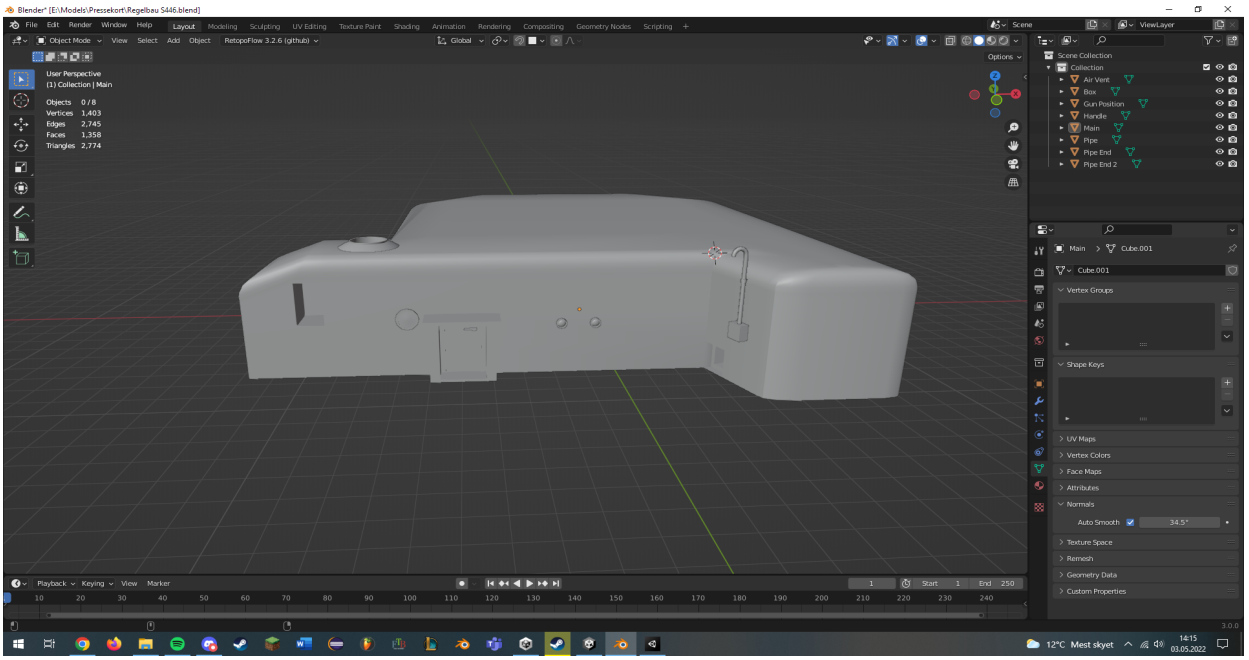

Figure 4.21: Recreated commando bunker

This is not to say that photogrammetry cannot work in VR development. Unfortunately, VR games must have the lowest amount of vertices rendered per frame for it to run smoothly. It is also possible to put the photo scanned models into Blender and use the decimate function. The decimate function allows you to reduce the vertex count of a mesh with minimal shape changes (Blender, 2022). One could increase the performance further by having different levels of decimation, for different levels of details (LOD) depending on how far away the player is from the mesh. In sum, with better technology, photogrammetry could have been a viable option for creating 3D models in this project.

4.4 Performance

4.4.1 What dictates good performance?

Several aspects of the project determine the game's performance. By analyzing the game through developer testing and analyzation tools, the project group discovered multiple areas of performance drops as well as an increase of draw calls and vertices.

Visible objects in a scene are sent to the GPU for it to be drawn on the screen. What determines the frame rate, is the amount of draw calls the GPU needs to render (Unity Support, 2021). The GPU is fast, however if the central processing unit (CPU) has too many instructions to send to the GPU, it will lower the amount of frames per second (FPS). One needs to also take into consideration the other tasks the game needs to perform, for example scripts and animations.

4.4.2 Unity Profiler

In many cases it is hard to identify cases of performance drops by trough gameplay testing. Unity has a Profiler tool that gathers and displays data on the performance of a running application in areas like the CPU, memory, renderer and scripts (Unity Documentation, 2022). It's a tool that can identify where the application needs performance improvements.

When running the game, it allows the team to play and pause the application to iterate over areas with bad performance. It is then possible to point out areas such as assets, camera rendering and code with a visual representation as a chart (Unity Documentation, 2022). While in Play Mode with the profiler on, the team can study the individual frames to get a better sense of the underlying issue. Using the Unity Profiler, the team was able to identify that the game is currently GPU bound. Unfortunately the profiler cannot explicitly tell which objects are causing the most GPU load. The methods used to fix these issues will be further discussed in the following chapters.

4.4.3 Culling

When rendering a scene, the main focus is to render objects that are in view of the camera/player. Rendering unnecessary geometry not visible to the player can drastically lower the game's performance.

By default, the Unity game engine is equipped with a culling method called Frustum Culling. This method is used to filter out all objects that lie outside of the camera's eye by removing them out of the scene. Frustum culling helps the GPU limit the amount of draw calls it needs to calculate per frame. However, there is still a problem when rendering out unwanted geometry inside the frustum area.

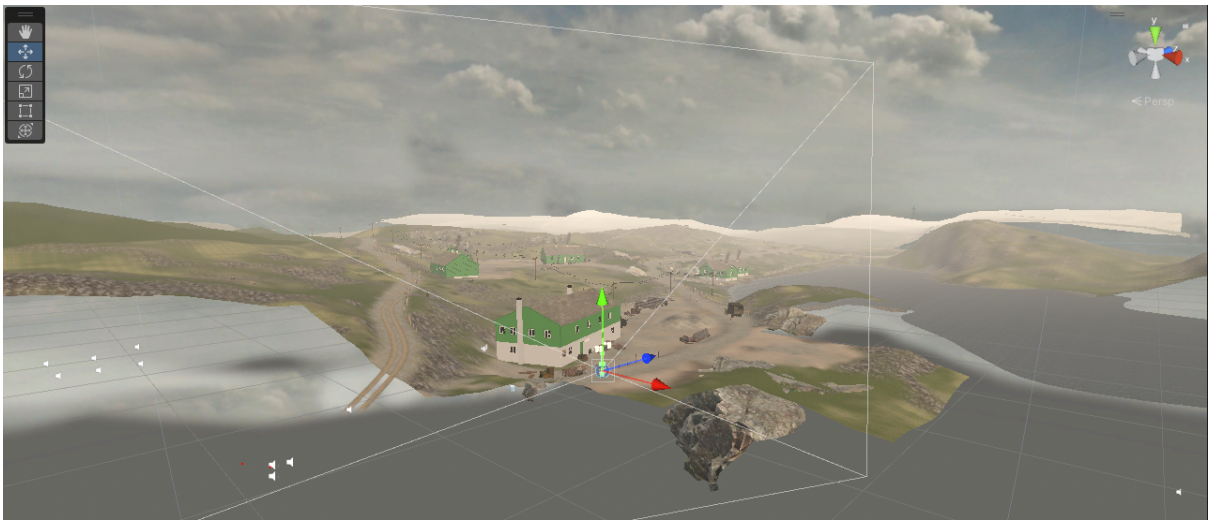

Figure 4.22: Frustum Culling on

While bug testing, the project group made notice of FPS drops and a high vertice count in the smallest of areas where the map had little to no objects visible to the player. The problem was that frustum culling does not cull objects that lie behind other objects, these were still being rendered in, adding unnecessary GPU draw calls.

Occlusion Culling was then added to prevent this from happening, further improving the game's performance. This method occludes geometry that lies behind other geometry. The data for culling in Unity is composed of cells, these are generated when baking in the occlusion culling (Unity Documentation, n.d.). The data/cells are used to help the camera specify what GameObject should be rendered in the scene and what should be occluded (Unity Documentation, n.d.). For this process to work, all objects in the scene should be marked as static. GameObjects marked as static tells the Unity Engine that the light and gravity

calculations on the object do not need to be performed. The occlusion culling process already has a solution to this, but keeping the amount of dynamic objects to a minimum will help the performance of the game as well.

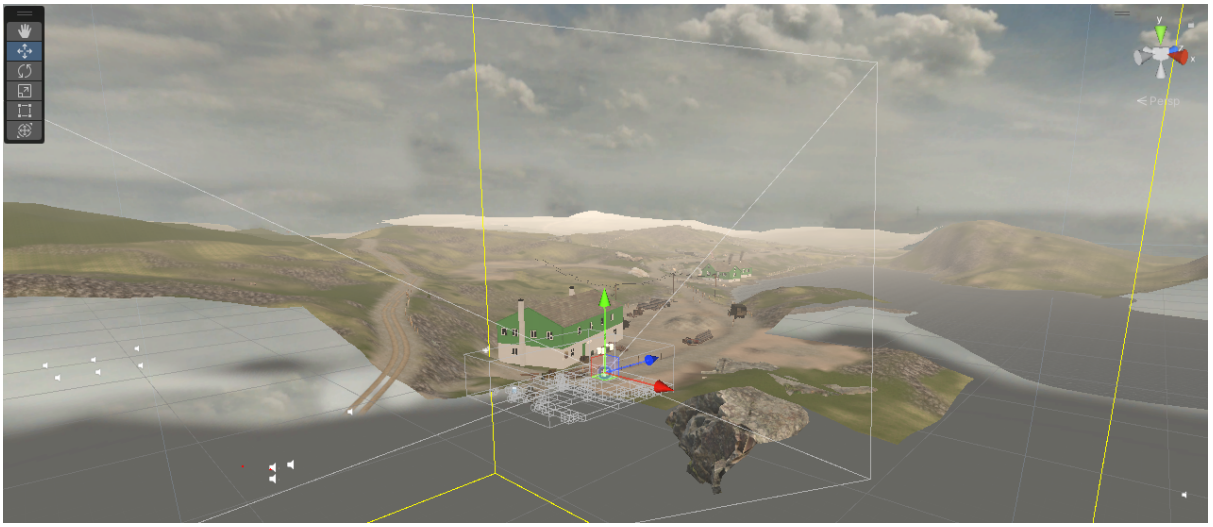

Figure 4.23: Frustum and Occlusion Culling on

4.4.4 Light Baking

Light Baking is a common performance technique that precomputes lighting on static GameObjects in a scene. The precomputed light data is saved in a texture, commonly called a light map. These light maps store how much shade each pixel will have corresponding to the position of the light source. Any game can benefit from this since it calculates the lighting beforehand rather than calculating the lighting during runtime. In Fjell Festning, the light source never translates or rotates, this decreases the load on the CPU substantially.

Unfortunately, the team encountered several issues when trying to bake lighting into the scene. Overlapping UV maps are believed to be the main problem. Overlapping UVs happen when two or more parts of a model share the same area of a texture. If the front and the back of a building share that texture space, when baking the lighting, the texture might simultaneously be overlayed by sunlight and shadow. This is likely to produce artifacts when lightmap baking.

Figure 4.24: The scene currently when baking lighting. Bad lightmap UVs are believed to be the problem

Auto-generating lightmap UVs through Unity sometimes helps, but in this case it did not. It would require manually setting the UVs of the objects within the modeling software so that they have no overlap.

4.4.5 Models

Each model in a scene has a vertex count which determines the complexity of how detailed the geometry of an object should be. A big problem in game development is how one should render each model depending on the player's position. The main goal here is to make rendering more efficient and increase the game's framerate, especially if the game is in virtual reality.

Level of Detail (LOD) is a technique that renders models distant from the player with a lower poly count which reduces the number GPU operations (Unity Documentation, 2022). The concept is determining how important objects are from the camera. Fjell Festning uses discrete LODs which are often used in games where each model has several versions of itself that descend in quality from where the player is standing. The Unity Game Engine has an inbuilt LOD system that does this automatically for the developer. The system takes in all versions of a model and the developer can adjust the value of when a new version of the model will be rendered in.

The Meadow Environment - Dynamic Nature asset pack has three LOD models for each object in the pack. Grass, rock and cliff models were frequently used in the scene. Having these LODs significantly helped the games performance, especially when culling is activated in the background.

4.4.6 Materials and Textures

When working with textures it is important to keep the texture resolution as low as possible without affecting the visuals. The smaller the object, the smaller the texture size required to make it look good. In order to ensure there are no textures larger than 2048×2048 pixels, the

project is set up to automatically lower the resolution of imported textures if they exceed that threshold.

Every material on an object requires a separate draw call. If you for an instance have a matte concrete building with reflective glass windows, you would need two draw calls for that single object. To optimize performance, our custom models contain as few separate meshes and materials as possible.

Sharing materials between objects is a good way of improving performance. It allows unity to batch the different object meshes into the same draw call. There are two methods Unity uses to accomplish this: static batching, and GPU instancing, each with their own advantages and requirements.

4.4.7 Batching and instancing

Instancing as previously mentioned is a method to optimize draw calls, so that it renders multiple copies of a mesh instead of creating new ones (Unity, n.d.). This helps performance immensely, especially when used in relation to forming realistic nature. If the machine had to render each blade of grass or rock it would cause a performance bottleneck. This is because rendering an object requires the GPU to find the right buffer, and the right vertex attributes which have to be transferred from the CPU (De Vries, 2017).

Static batching requires the object to be marked as static. A static object is an object that is guaranteed not to move. GPU instancing requires that the objects share the same mesh, and material, but allows for moving objects. GPU instancing can be enabled on the material. GPU instancing is not very effective on meshes with few vertices. Unity recommends using GPU instancing on models with more than 256 vertices.

This was used on some of the objects scattered around the scene. More specifically interactable objects like glass bottles that the player could pick up and move. Objects with less vertices like grass used static batching. This was possible as the Client did not want the grass to interact with the wind.

5. Results

5.1 Evaluation method

Evaluation was executed in accordance with the project evaluation plan. This included regular functional and performance tests during the course of development, followed by alpha and beta tests towards the end.

The alpha tests were conducted on the 21st of April. The test took place on 3 different computers, allowing the team to test the build on different hardware, and to check whether or not the setup process worked as expected. The client had originally expressed willingness to perform the playthrough, however due to complications with the setup process, the playthrough was performed by the group. The game was projected on a canvas for the client to monitor. The goal was to identify compatibility issues, and an overall acceptance test used to validate that the product is as the client expects. The alpha tests were conducted early enough so that potential issues or missing features could be implemented before beta tests and the projects end.

Figure 5.1: Beta tests with external users.

Beta tests were conducted on May 6th. Corrections in the setup process based on the knowledge gained from the setup difficulties in the alpha tests allowed for a quicker start. 7 subjects were confirmed to join, with many more suspected to join. Testers would arrive in pairs of two to avoid downtime if any testers are delayed, or if testers have to quit mid session due

to nausea etc. Testers would complete the first two steps in Media Labs planned education program: watching the onboarding video, followed by a playthrough of the game. The group and client would monitor the playthrough, taking notes and asking questions underway. The testers were encouraged to point out difficulties, issues, improvements and wanted features.

5.2 Evaluation result

5.2.1 Performance Test

Performance tests early in development signaled that a version running purely on mobile VR hardware was unlikely. To get the game to launch on the Oculus Quest 2 headset it required the texture resolution to be lowered to 1/4th the original size as the game crashed due to memory shortage. The game then ran at a shocking 6 frames per second.

The original build ran at only 28 frames per second on PC. The decision was made to focus on making the PC experience as good as possible instead of potentially wasting time on performance improvements that were not going to make a difference.

In the course of development, several causes for the low performance were identified. Profiling the game using the built in Unity Profiler showed the game to be mainly GPU bottlenecked. This is a situation where the CPU has to wait for the GPU to finish before proceeding to the next frame, resulting in lower frame rate. Unity does not have access to the inner workings of the GPU. Because of this, the graphically intensive objects were manually identified by disabling groups objects in the hierarchy while monitoring the performance.

After figuring out the causes for the low performance, the group actively fixed or removed the origins. As of today the game runs at most locations a smooth 72 frames and in some areas even higher. The load times also subsequently decreased during the course of development, some assets caused the load time to be considerably longer. Performance is at a level which, with a few, minor tweaks to the terrain and models can run on an Oculus Quest 2 unit. Something which seemed improbable earlier in the development phase.

5.2.2 Alpha Tests

During the alpha tests it became clear that there were some compatibility issues between certain machines and the application. While the exact cause was unknown, it was likely an issue with the machine's standard XR runtime being SteamVR. Switching the XR runtime to OpenXR through the Oculus app was for some reason locked on these machines. However, using SteamVR we were able to get the game running on these machines as well.

The client stated that he was happy with what had been achieved in the three months of development. However, the tests did highlight some areas on which improvements could be made. A common theme being the fact that the area feels a bit empty, considering there was likely a couple of hundred people present at the fortress at that time according to the client. Table 5.1 contains a list of features and issues in addition to a priority

Table 5.1: *Feedback table with priority level*

Feature	Priority

Add the outro scene, including a taxi that picks up the player, and a scene in a theater showing a clip of the miners risking their lives as the story might not do a good enough job at explaining this aspect of the story.	1
Add interactable objects like notes, journals and personal items that the player can inspect to get a deeper understanding of the situation on the fortress.	2
Look into adding more characters to the scene. Identify what performance impact this would have and how to mitigate it. Characters on the crash site have higher priority as they are mentioned in the story.	3
Add german propaganda posters to the compound, minefield signs and potentially wanted posters showing the wanted fugitive mentioned in the story.	4
Add a radio broadcast to the radio next to the barracks.	5
Work on the sound design, mainly more ambient sounds. Also includes modifying the footstep sounds and fixing the audio of the planes flying by.	6
Tweak certain objects. Some objects are seemingly too large. Some seem to disappear at certain camera angles likely due to occlusion culling.	7

5.2.3 Beta Test

During the beta tests the group got a lot of valuable input from the users in the form of real time verbal reactions and points. The feedback the team got through testing was very similar to other players, almost every test subject gave the same feedback as the previous tester.

As shown in Figure 5.2, 6 out of 10 users experienced some form of nauseousness when playing the game. This was due to factors like varying collisions with the terrain, and continuous movement with the joystick and head turning. A potential fix would be to implement locomotion by teleportation as an option to continuous movement.

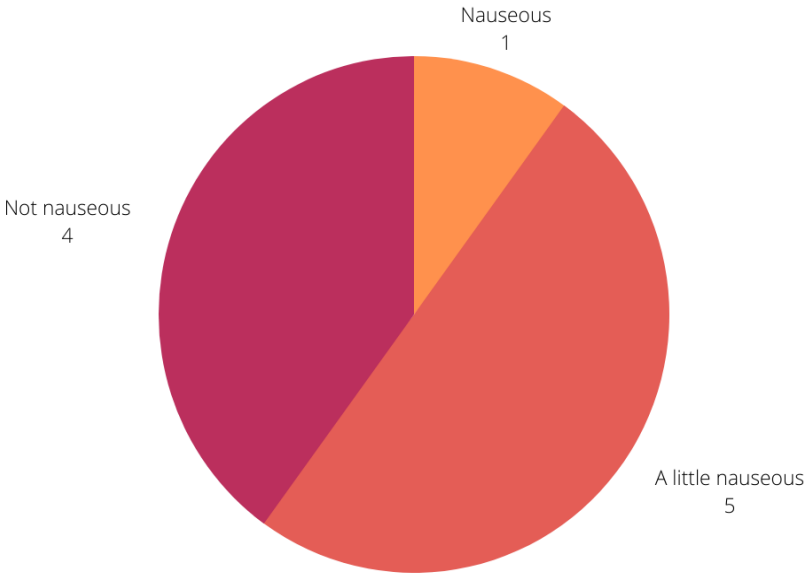

Figure 5.2: Pie chart displaying 6 out of the 10 testers experienced some form of nausea

A lot of users struggled to understand how to control the player and how to interact with the world around them. This included a difficulty understanding how to move the character,

grabbing objects, interacting with characters and how to use the inventory system. When explained, most testers were able to complete the playthrough without additional explanation. This indicates that the system works, but needs to be explained better within the game. This was the initial plan with the gatekeeper interaction, but it is clear it still needs some work.

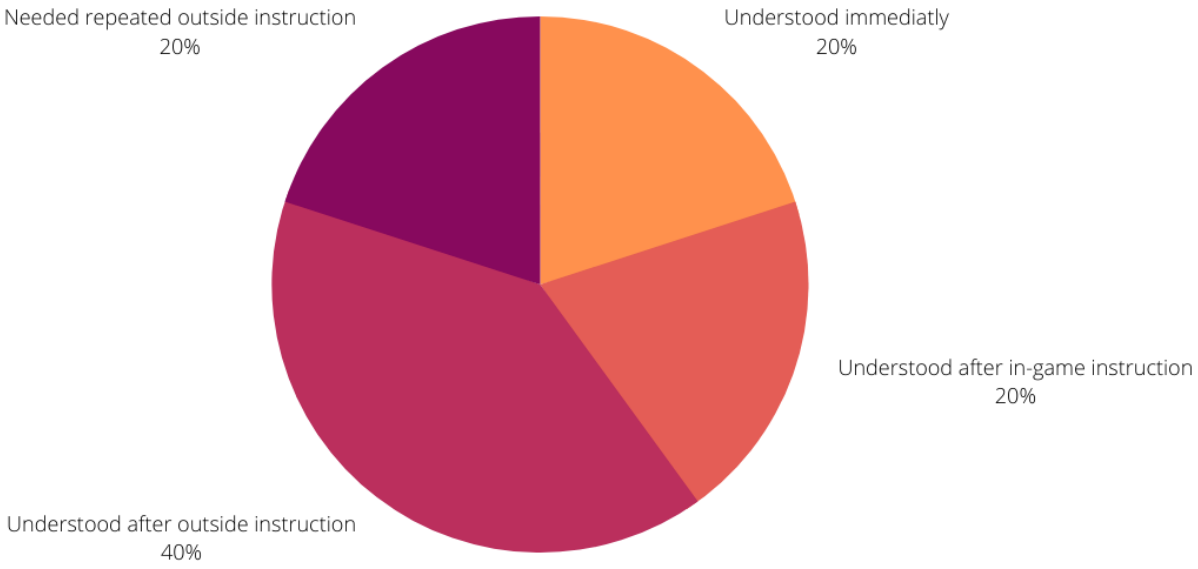

Figure 5.3: Pie chart displaying the difficulty testers had understanding the controls.

Most testers experienced some level of difficulty regarding navigating the game with the majority of testers expressing the difficulty of navigation to be somewhere between moderate and difficult (see Figure 5.4). When asked about how they thought the map worked, 4 out of 10 answered that the blue dot represented the players position and not the target position.

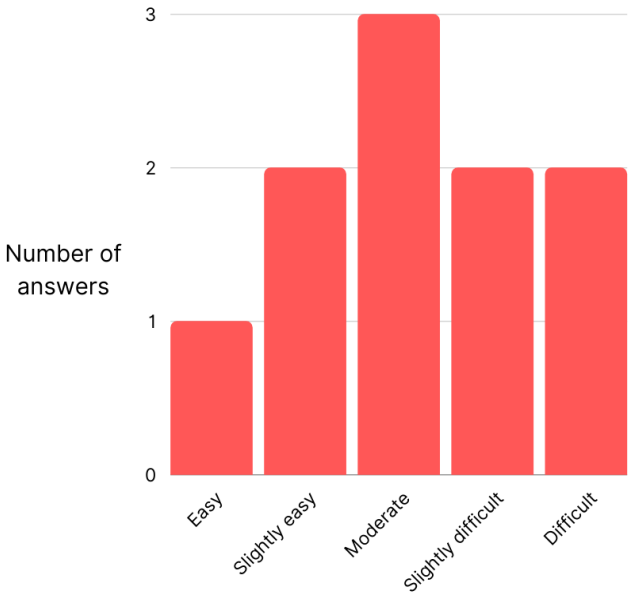

Figure 5.4: Bar chart displaying the testers experienced difficulty of navigation.

It became clear that the wish of having players orient themselves according to the map and their surroundings was at the expense of the user experience. On the other hand, some testers experienced no issues navigating using the map, indicating that the map, and environment is accurate enough for navigation to be possible using it. Testers also experienced some difficulty identifying which NPCs were interactable, and some spent a significant amount of time trying to interact with non-interactable NPCs.

Multiple testers dropped or lost the map during their playthrough making it very difficult to understand where to go. A quick fix would be for the map to automatically return to the players inventory should they step too far away from it.

The team was mostly interested in constructive feedback, but that does not mean all feedback was negative. A large number of testers were impressed by the game's visual fidelity and map design. The ambient sounds of birds/wind and footsteps while moving through the game made the game more realistic, and seeing objects move around the scene, for example the German trucks and airplanes made the world feel alive. Other aspects of the gameplay design like interacting with the gatekeeper and using the notebook for conversing was generally well received by the testers, especially being able to pick up random objects in the scene, even if some struggled with actually grabbing them.

5.3 Project Result

The project result is evaluated in regards to the goals the group set at the start of the project phase. The original project goals were to create an entertaining and pedagogical virtual reality video game, by further developing a Unity project. Additionally the other goal was to optimize the project by reducing the file size and gaining performance.

The game is now in a finished state as in regard to the original project goals set by the group and Media Lab. The solution contains a game with multiple scenes, user interactions, non playable characters with voice lines and moving entities like trucks and planes. The game's storyline is now fully playable from start to finish.

Moreover, the game itself now has a substantially decreased file size compared to the original. In addition to this, the game's performance has increased, now having a much higher average frame rate. This means the game is more optimized and can potentially be run on lighter machines, such as a Quest 2 unit.

Some tasks from Media Lab were not completed, such as certain interactable events and more characters moving around in the scene. This was not included both due to time and mostly because of performance constraints. However, the fundamental parts of the game are complete, as well as other significant gamification and scenery additions.

5.4 Project Execution

Figure 5.5 (p. 52) displays the actual project execution compared to the plan. Beige represents time spent on a feature outside the original plan. Red represents the original plan where no actual work was done in that time frame. Orange represents the timeframes where the feature was developed in accordance with the original plan.

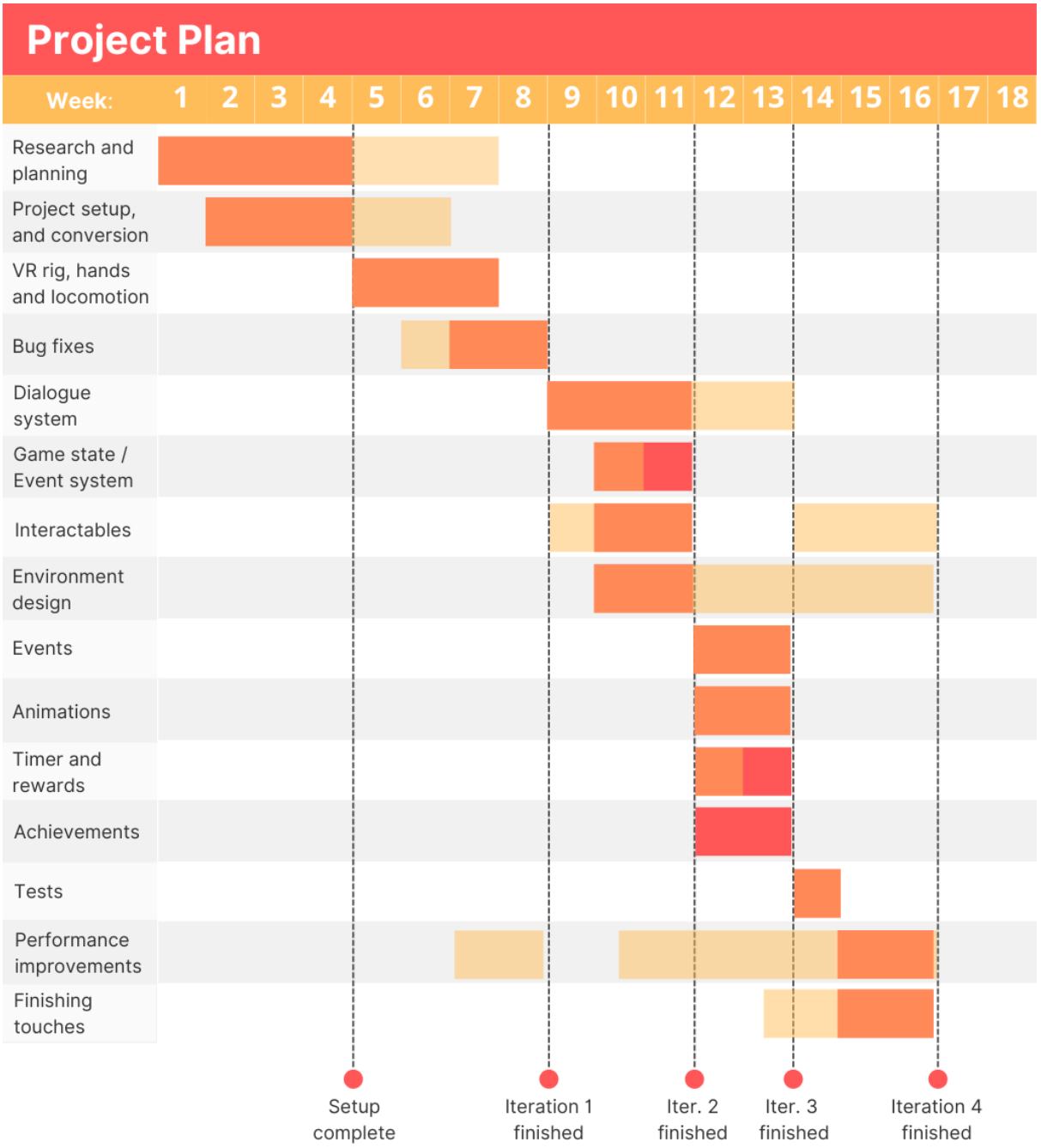

Figure 5.5: Actual project execution.

In most areas the group was able to keep up with the project plan. Most deviations were not due to setbacks but rather due to shifts in priority. Due to the chosen project methodology, this was not an issue. While the team are saddened by not having time to add achievements and rewards, they were able to add many other features that were not part of the original plan, like the map, inventory and the gatekeeper. Due to the minimal viable product approach, the game was for the most part always in a playable state and had little risk of

The weekly meetings with the client were a big reason for the project's success. It allowed the group to quickly get feedback on both the planned features, and the features under development.

6. Discussion

At the project's end, the team delivered on the initial requirements of finishing the project, however, the end product could be better. Mainly, the team believes the game is not as fun, and engaging as it could be. This is mainly a product of the short timeframe that was provided, but there are still things that could have been done differently.

In this chapter we will reflect on what went well, and what could have been done better.

6.1 Collaboration

We believed the initial project description was a little vague. The client did not initially have many specific requirements. Thankfully we had close contact through the entire development process and was able to get a better sense of the direction the client wanted us to go. We do however believe it might have been beneficial to set some of the requirements in stone earlier in development, as we would continue to get new requirements up until two weeks before the deadline.

Collaboration within the team was for the most part very good. The team would meet up multiple times per week, either physically or through zoom / discord, to work on, plan or discuss the project. We tried to keep each other accountable for their work by declaring what each member would work on. At the end of the week we would explain the work done that week. A problem with this is that it is easy to lose track of what has been done by yourself, and what other team members were supposed to do. It is also difficult to confront other team members in such a small team when they don't do their fair share. A better approach might be to use a kanban board to more easily visualize what team members did and did not do.

At the start of the project we used Unity Collaborate as our source control system. This worked well for the first couple of months before we started experiencing some issues. Mainly when multiple people worked in the scene, it would result in a file conflict that could not be resolved by merging. This in spite of working on different parts of the scene. We have to this date not been able to identify the reason for this, we even contacted the Unity cloud service support team, but they were unable to identify the issue.

This issue meant that multiple people could not work in the scene simultaneously. This severely impacted the speed of development and meant we had to plan out in detail who would do what when. We later were forced by Unity to migrate from Unity Collaborate to PlasticSCM source control. This process took some time and the same issue as before persisted, indicating that the issue lies within Unity or the structure of the scene. Since PlasticSCM had the same issue, we determined it would not be worth the risk to switch to for example GitHub as it is likely to not fix the issue, and the integration with Unity is worse than the current option.

6.2 Project Setup

As discussed in 3.2, we decided to update the project to URP and to Unity 2021.2. Setting up the project took several attempts, as the project got errors and crashes that were believed to be caused by the engine or by one of the frameworks. These were very difficult to troubleshoot, so a strategy that seemed to work best was to create a brand new project in Unity 2021.2 using the URP template. Then installing all necessary plugins and creating a

simple test scene for testing basic XR integrations. Scene includes a player controlled rig, a flat ground plane and a grabbable cube. This scene was then built and deployed on Oculus Quest 2 headsets, verifying that android support was working as intended.

The scene from the initial project was then imported to the project. The scene looked almost completely empty as we had not yet added the assets. The assets were added one by one to the assets folder, until the scene was filled. This was done to lower the project's file size, as the project was taking almost 15 minutes to launch, as well as allowing us to identify the assets that were causing errors and crashes. At the end of this process we had a clean scene with only working assets that we were expected to use.

Although it took a while to do this entire process it gave us a far better basis to work from. We were ensured that an android build was theoretically possible, and any issues with this were likely to be performance based and not asset or project based. The file size was severely reduced and the project launch and build speed was noticeably quicker. It also gave us a much deeper understanding of the project, the scene and the assets we had access to.

6.3 Code Practices

Working independently with so many different features often resulted in a lot of scripts with very specific functionalities. In some instances this was unavoidable, but we could have done a better job at generalizing some of the scripts which would increase the reusability.

Another bad practice was not documenting or planning the functions and methods properly from the start. This was not a problem early on, but as the methods grew more complex they also became difficult to read and follow. If we had time we would like to restructure a couple of these scripts, mainly the NodeParser and NPCHandler.

The NodeParser has steadily increased its functionality over time, but has resulted in a very large script with a number of different functions. It might be difficult for new developers to understand how things are connected. We have however a few things we have agreed to fix in the time frame between the report submission and the EXPO on the 15th of June, including documentation.

Originally the NPCHandler was strictly for controlling the interactable characters in the game. This includes disabling and enabling them as the story develops. However during the course of development this script was expanded to cover more than just that. This was a quick and easy way of implementing certain features, but made the script unnecessarily large. It also makes it more vulnerable to crashes, as an error with one object or character can cause other events to not fire. There is however a positive side to this. It allows us to quickly discover issues. Having every event and character controlled through one script also helps to visualize the flow of the game, and makes it quick and easy to alter the chain of events.

The documentation could definitely be better at this stage. Currently, many of the scripts do not have documentation, other than the one provided in the system documentation (See Appendix C). We plan to document the most relevant scripts using XML documentation comments, and Unity inspector customization like we did in NodeParser.

6.4 Unaccomplished Goals

One of the big goals of the project was to have the game running on the Oculus Quest 2 hardware. There were from the beginning some questions regarding whether or not this goal was possible. After the initial performance tests we saw it as improbable that we would be able to make this a reality, however as changes were made to the project, the performance increased significantly. The game is currently in a state where with just some more performance improvements could run on the mobile VR platform. Some of the potential improvements are listed in *Chapter 7.3: Further work*

According to the vision documents list of features (See Appendix A) There were 3 unimplemented features. We were unfortunately unable to add achievements and scores, two features we believe would make the game significantly more fun if implemented correctly. The reason we did not is because of lack of time. We believe that this feature had to be implemented well in order to have the desired effect and we simply did not have the time to do that. Another example was to include multiple characters that moved about to fill the scene, this was not possible since the characters given to the team were too performance heavy.

7. Conclusions and Further Work

This final chapter will present the conclusion of the project in relation to the discussion in chapter 6. Additionally the project result will be compared to the original goals set by the group and Media Lab, which were presented in chapter 1.

7.1 Goals and Results

The original project goal was to create a historically correct, enjoyable learning experience for pupils by gamifying the previous bachelor project, as well as increase the performance and playability of the game.

The goal was reached by further developing the previous solution to a working VR game which includes gamification elements as well as other improvements to create an immersive, enjoyable learning environment. The project owner is satisfied with the results with all initial expectations being fulfilled. As mentioned in chapter 5, the beta testers had a lot of positive feedback regarding the state of the game, and how immersive the experience was. Much of the feedback came in relation to moveable objects, interactions and moving entities in the scene, such as planes and trucks. The new dialogue system is also completed, letting any developer easily expand the system.

Figure 7.1: Before project start.

Figure 7.2: The game in its current state.

Furthermore, the goal was also achieved by improving the game's FPS. The average frame rate from the previous project was below 28 (on a GTX 1070 GPU), which is nowhere near the recommended 72 FPS benchmark for Oculus VR games. The group finds the results satisfactory, seeing as the FPS now is at a stable level, even above the 72 frames per second in certain areas. The file size is also smaller due to removing unnecessary assets which will considerably reduce the installation time.

Additionally, as mentioned in chapter 6.4, a goal was to create a version of the game which would be runnable on a stand alone Oculus Quest 2 unit, without the need for a robust gaming computer. With this version, the game could be used in classrooms for students who do not have the possibility to travel to the actual fortress. The end result here is as expected. The team had communicated early that reaching this goal would be improbable, due to the bad performance at the current time. As of today, the project requires a gaming computer as well as

an Oculus Link cable, to run to its full capability. However the performance optimizations the group has done, has increased the performance to a point where a mobile VR version is within reach, but it still needs just a little work.

The mine and timer features were implemented and were also functional objects in the game. As of today, these still lie in the projects assets folder. These features were not included in the final product since the project team had no more time to integrate these correctly with a respawn / score system. Features such as the achievement system were also not implemented, this will be further discussed in Chapter 7.3.

During development, the team developed features outside the original plan that made the user experience more enjoyable and interactive. These were features such as the inventory system, map, pistol, identification papers and the gatekeeper. More technical tools like the Event Manager, were not part of the original plan, but were an addition that helped structure our code. These were prioritized because the team determined these features to be more effective at improving the user experience than timers or mines.

7.2 Relevance of the project

Virtual reality games for educational purposes is by no means a new phenomenon, yet it is still an exciting field for technology. With newer and better technologies and solutions being rapidly produced, VR has remained a core part of the future.

This project could have relevance for other museums who would like to create similar solutions, either to create video games or digital recreations of historic buildings and areas. A lot of museums are looking to add this type of technology to their locations as mentioned by Øyvind Fosse during one of the project meetings.

Furthermore, the project does not only have relevance for historical purposes. The project could also tie in to other projects in relation to pedagogical games for schools. Other subjects may be just as relevant as history, such as religion, ethics and science.

7.3 Further Work

The group recommends replacing or modifying the characters. The characters currently have a severe impact on performance. Certain areas of the game run at less than half the frame rate with characters enabled. We believe the reason to be the fact that the characters comprise of 8 different meshes. Animated characters use Skinned Mesh Renderers for their meshes as they need to deform. Skinned Mesh Renderers are very taxing and should be limited to one per character. We recommend either combining the meshes of the characters or replacing them. The gatekeeper model has better performance, so it can be kept.

Given the last recommendation, the second recommendation would be to add more non playable characters in the scene. One could take advantage of the better performing characters, and add more of them to the scene. This will in turn increase immersion in the scene by having a greater quantity of NPC's moving about the virtual fortress.

Some of the models are making lightmap baking impossible. We recommend taking a look at these and fixing their overlapping UVs. The main problems are the barracks and the

surrounding buildings. Unity is unable to automatically generate good lightmap UVs for these, so we recommend manually setting these and avoid any overlap. Using this in conjunction with light probes could severely improve the CPU load should it prove to be a problem.

The game currently does not have a main menu. We recommend adding one that allows the player to start the game from several stages. This is to make it possible for players to skip the intro, or to resume the game from a state close to where they left off. This is especially helpful in the event of a crash. Other options in the menu could be sound settings and preferred locomotion method, where players could choose between teleportation and continuous movement. This feature should ideally be available from within the main scene as well.

Achievements and scores are two features we unfortunately did not get the time to add. We believe this would greatly improve the user experience as it would give the player a sense of progression and rewards for doing so. Achievements could be to find a hidden object or room, or to perform a specific action. Score could be tied up to the amount of information you were able to gather. The final score could be displayed at the end of the game, which would in turn increase replayability, as players would be encouraged to increase their total score, and gather more information in the next playthrough.

8. References

Aber, J. S., Marzolff, I., & Ries, J. (2010). *Small-Format Aerial Photography: Principles, Techniques and Geoscience Applications*. Elsevier Science. https://doi.org/10.1016/B978-0-444-53260-2.10003-1

Beck, K., Beedle, M., van Bennekum, A., & Cockburn, A. (2001, 1 1). *Manifesto for Agile Software Development*. Manifesto for Agile Software Development. Retrieved March 12, 2022, from https://agilemanifesto.org/

Blender. (2022, 02 05). *Decimate Modifier*. Retrieved 05 14, 2022, from https://docs.blender.org/manual/en/latest/modeling/modifiers/generate/decimate.html

Brigsted, T. (2021). *xNode* [Node editor for Unity]. Github/xNode. https://github.com/Siccity/xNode

Crazy Minnow Studio. (2022, Feb 04). *SALSA LipSync Suite | Animation Tools*. Unity Asset Store. Retrieved May 10, 2022, from https://assetstore.unity.com/packages/tools/animation/salsa-lipsync-suite-148442#description

De Vries, J. (2017, 10 26). *LearnOpenGL - Instancing*. Learn OpenGL. Retrieved May 15, 2022, from https://learnopengl.com/Advanced-OpenGL/Instancing

Epic Games. (n.d.). *Digital Humans | MetaHuman Creator*. Unreal Engine. Retrieved May 10, 2022, from https://www.unrealengine.com/en-US/digital-humans

Epic Games. (n.d.). *Nanite Virtualized Geometry in Unreal Engine | Unreal Engine Documentation*. Unreal Engine 5 Documentation. Retrieved May 10, 2022, from https://docs.unrealengine.com/5.0/en-US/nanite-virtualized-geometry-in-unreal-engine/

Facebook Technologies. (n.d.). *Oculus Lipsync for Unity Development: Unity*. Oculus Developer Center. Retrieved May 10, 2022, from https://developer.oculus.com/documentation/unity/audio-ovrlipsync-unity/

Immersion VR. (2022, jan 01). *VR for Education - The Future of Education*. Immersion VR. Retrieved March 12, 2022, from https://immersionvr.co.uk/about-360vr/vr-for-education/

Insko, B. E. (n.d.). *OpenXR Overview - The Khronos Group Inc*. Khronos Group. Retrieved May 4, 2022, from https://www.khronos.org/openxr/

Khronos Group. (n.d.). Retrieved May 9th, 2022, from https://www.khronos.org/openxr/

Khronos Group. (2019, July 29). *OpenXR Overview - The Khronos Group Inc*. Khronos Group. Retrieved May 4, 2022, from https://www.khronos.org/openxr/

Knowles, R. (2020, January 8). *What is Agile Research and Why Should Insights Teams Care?* LinkedIn. Retrieved May 22, 2022, from https://www.linkedin.com/pulse/what-agile-research-why-should-insights-teams-care-roddy-knowles

Krita Foundation. (n.d.). *History*. Krita. Retrieved May 10, 2022, from https://krita.org/en/about/history/

Li, Y. (2019, February). Gesture interaction in virtual reality. *Virtual Reality & Intelligent Hardware*, *1*(1), 28. ScienceDirect.com. 10.3724/sp.j.2096-5796.2018.0006

Mixamo. (n.d.). *Mixamo*. Mixamo. Retrieved April 22, 2022, from https://www.mixamo.com/#/

Museum Vest. (n.d.). *Korleis kjem du deg til Fjell festning?* Fjell Festning. Retrieved May 22, 2022, from https://fjellfestning.museumvest.no/norsk/praktisk-informasjon/korleis-kjem-du-deg-til-fjell-festning/

Qualcomm. (n.d.). *Extended Reality XR | Immersive VR*. Qualcomm. Retrieved May 20, 2022, from https://www.qualcomm.com/research/extended-reality

Qualcomm. (n.d.). *Virtual Reality*. Qualcomm. Retrieved May 20, 2022, from https://www.qualcomm.com/research/extended-reality/virtual-reality

Ridley, J. (2022, February 8). *The Quest 2 is now the headset of choice for 46% of Steam VR users*. PC Gamer. Retrieved May 22, 2022, from https://www.pcgamer.com/the-quest-2-is-now-the-headset-of-choice-for-46-of-steam-vr-users/

Unity. (n.d.). *Manual: GPU instancing*. Unity - Manual. Retrieved April 29, 2022, from https://docs.unity3d.com/Manual/GPUInstancing.html

Unity. (2017, 11 21). *Manual: Animator Controller*. Unity - Manual. Retrieved April 22, 2022, from https://docs.unity3d.com/Manual/class-AnimatorController.html

Unity. (2021, August 31). *Universal Render Pipeline overview | Universal RP | 11.0.0*. Unity - Manual. Retrieved May 10, 2022, from https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@11.0/manual/index.html

Unity. (2022, May 05). *Choosing and configuring a render pipeline and lighting solution*. Unity - Manual. Retrieved May 9, 2022, from https://docs.unity3d.com/Manual/BestPracticeLightingPipelines.html

Unity. (2022, May 5). *Manual: Render pipelines*. Unity - Manual. Retrieved May 9, 2022, from https://docs.unity3d.com/Manual/render-pipelines.html

Unity. (2022, May 5). *Manual: Using the High Definition Render Pipeline*. Unity - Manual. Retrieved May 9, 2022, from https://docs.unity3d.com/Manual/high-definition-render-pipeline.html

Unity Documentation. (n.d.). *Manual: Types of light*. Unity - Manual. Retrieved May 1, 2022, from https://docs.unity3d.com/Manual/Lighting.html

Unity Documentation. (n.d.). *Occlusion Culling*. Unity - Manual. Retrieved May 2, 2022, from https://docs.unity3d.com/560/Documentation/Manual/OcclusionCulling.html

Unity Documentation. (n.d.). *Scripting API: NavMeshAgent*. Unity - Manual. Retrieved May 1, 2022, from https://docs.unity3d.com/ScriptReference/AI.NavMeshAgent.html

Unity Documentation. (2022, May 5). *Manual: Profiler overview*. Unity - Manual. Retrieved May 15, 2022, from https://docs.unity3d.com/Manual/Profiler.html

Unity Documentation. (2022, May 7). *Manual: Level of Detail (LOD) for meshes*. Unity - Manual. Retrieved May 14, 2022, from https://docs.unity3d.com/Manual/LevelOfDetail.html

Unity Learn. (2021, February 11). *Introduction to Particle Systems*. Unity Learn. Retrieved April 22, 2022, from https://learn.unity.com/tutorial/introduction-to-particle-systems

Unity Learn. (2021, March 7). *XR Interaction Toolkit: Working with the Socket Interactor*. Unity Learn. Retrieved May 4, 2022, from https://learn.unity.com/tutorial/xr-interaction-toolkit-working-with-the-socket-interactor#

Unity Support. (2021, August 27). *Why are my batches (draw calls) so high? What does that mean?* Unity Support. Retrieved May 5, 2022, from https://support.unity.com/hc/en-us/articles/207061413-Why-are-my-batches-draw-calls-so-high-What-does-that-mean-

Unity Technologies. (2021, February 22). *Configuring an XR Rig with the XR Interaction Toolkit*. Unity Learn. Retrieved May 20, 2022, from https://learn.unity.com/tutorial/configuring-an-xr-rig-with-the-xr-interaction-toolkit#

Valve Corporation. (2021, February 24). *SteamVR Plugin | Integration*. Unity Asset Store. Retrieved May 4, 2022, from https://assetstore.unity.com/packages/tools/integration/steamvr-plugin-32647#description

VIDE Dialogues. (2018, November 05). *VIDE Dialogues*. VIDE Dialogues. Retrieved April 26, 2022, from https://videdialogues.wordpress.com/

Western Norway University of Applied Sciences. (2020, August 19). *Læringslab*. Høgskulen på Vestlandet. Retrieved May 22, 2022, from https://www.hvl.no/alu/medielab/laringslab/

Western Norway University of Applied Sciences. (2020, October 15). *HVL*. Media Lab - Western Norway University of Applied Sciences. Retrieved May 21, 2022, from https://www.hvl.no/en/alu/media-lab/

Appendix E: Test Questions

Fjell Fortress 1945 User Test

Tests performed friday 6th. May. 2022

[redacted] (not shared) Switch account

* Required

Did you feel any physical discomfort during the playthrough?

- ◯ None
- ◯ Headache
- ◯ Nausea
- ◯ Other: _____

How easy was it to understand the controls? (Moving the player character, picking up objects, interacting with characters etc.)

	1	2	3	4	5	
Difficult	◯	◯	◯	◯	◯	Easy

How intuitive was the inventory system?

	1	2	3	4	5	
Difficult	◯	◯	◯	◯	◯	Intuitive

How easy was it to navigate through the game? *

	1	2	3	4	5	
Hard	◯	◯	◯	◯	◯	Easy

How immersive was the game experience?

	1	2	3	4	5	
Felt completely artificial	◯	◯	◯	◯	◯	I cannot seperate the game from reality

To what extent did the surroundings reflect World War II?

	1	2	3	4	5	
Not very realistic	◯	◯	◯	◯	◯	Very releastic