Bachelor's project

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Genti Dautaj
Oscar Vagle
Viktor Jarl Palmason
Ole Kristian Lund Lysø

Video Game Modding's Potential for Teaching Programming

Developing a modding Integration for a Turn-Based Strategy Game

Bachelor's project in Programming
Supervisor: Tom Røise

May 2021

NTNU
Kunnskap for en bedre verden

Genti Dautaj
Oscar Vagle
Viktor Jarl Palmason
Ole Kristian Lund Lysø

Video Game Modding's Potential for Teaching Programming

Developing a modding Integration for a Turn-Based Strategy Game

Bachelor's project in Programming
Supervisor: Tom Røise
May 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Kunnskap for en bedre verden

Abstract

Programming is starting to be a larger part of the school's curriculum. This is because it is a valuable skill to learn in a world where computers are getting a larger impact in our society. Video game modding, which is about changing the game's assets and features, can be used as a medium to learn programming. Progress Interactive, an independent video game studio, will explore how modding can be used to learn programming for upper secondary school students. Therefore, Progress Interactive has guided us on developing a small game which includes basic video game modding possibilities. This has been developed using the Unity game engine, Turn-Based Strategy framework and Moonsharp, a Lua interpreter written in C#, for executing Lua scripts from modders. This report will describe the development history and explanation of our self-made game as well as reflections upon the potential modding has for teaching programming.

Sammendrag

Programming begynner å bli en større del av skoleundervisningen. Dette er fordi programmering er en viktig egenskap å lære i et samfunn der datamaskiner får en større og større rolle. Videospillmodifikasjon, som handler om å endre på teksturer, spillogikk og andre elementer av videospill, kan bli brukt for å lære bort programmering. Det uavhengige spillselskapet, Progress Interactive, vil utforske hvordan spillmodifikasjon kan bli brukt for å lære bort programmering for videregående skoler. På bakgrunn av dette har Progress Interactive veiledet oss gjennom utviklingen av et enkelt spill som kan modifiseres. Dette har blitt utviklet med spillmotoren Unity, et Turbasert strategirammeverk og Moonsharp, en Lua interpreter utviklet i C# for å eksekvere Lua skripts fra spillmodifikasjonsprogrammerere. Denne bacheloroppgaven vil beskrive utviklingen av spillet samt refleksjoner om potensialet spillmodifikasjon har for opplæring i programmering.

Table of Contents

Acronyms

1. TBS = Turn-based strategy Framework
2. Unity = Unity game engine
3. StreamingAssets = acronym for Streaming Assets folder from Unity

Explanation of words and synonyms

1. Modding implementation and Lua integration are synonyms. (TBS game)
2. To mod = modify something. In this context related to video games.
3. Moddable = modifiable related to video game modding
4. Video game modders, modders = people who modify games.
5. Tile = cell, a hexagon shaped tile which makes up the game map in the TBS game.
6. Client = product owner. Our contact person at Progress Interactive is referred to either as client or product owner.
7. Cellgrid = game map

Introduction

Project Backgrounds

Game modification is a broad and expansive world that covers just about any game and has existed ever since the 80's *[1]*. Video game modifications (commonly abbreviated as ”game mods”, or simply ”mods”) are alterations to a game's assets and features with the intention of changing the game's behaviour and presentation, as well as the practice of users implementing their own content into an existing game. Mods are primarily developed and implemented by consumers of the game, rather than the original developers, and are not restricted to programmers as mods are not necessarily just behavioural changes but also visual and auditory. The ”Steam Workshop“, provided by Valve themselves on their digital game distribution platform Steam, is the prime example of a community-driven hub that provides user-created mods for practically any game hosted on their platform *[2]*.

Modders (the common term for people modifying games) can express their ideas and creativity by implementing features into their games of liking, further enhancing the gameplay experience for both themselves and other fans alike. A mod can range from something as simple as changing the colour of a character's shirt, to adding multiplayer functionality to an originally single-player game, for example the world-renowned Counter-Strike mod for Half-Life *[3]*. A user can add a sword they designed with a particular sound effect into their favourite fantasy game and share it with the world. Perhaps just for fun, they would love to replace all the cars in a racing game with various fish. Modders can also alter the game in ways to make them more accessible, e.g., visual changes meant to aid colour-blind players, which in turn extends the game's reach out to a broader audience.

In addition, video game modding can also be used to learn programming for upper secondary schools which is Progress' motivation for this project. Exploring the potential modding has for teaching programming will be discussed in this report.

The first reason for using video game modding to learn programming is because it involves adding code to an already existing codebase. This is a relevant and a valuable skill to have as a software developer *[4]*. When students are modding they are learning to adapt their code to the game and to read documentation in order to teach themselves what they need in order to create their mod.
Secondly, it opens the opportunity to create something more interesting or teach something more related to what students expect out of learning programming. From the master thesis “CodeFlip” many students had expectations of creating big complex games when they started with programming *[5]*.

Moreover, using video game modding to help students learn programming is relevant for use in upper-secondary schools. This is because there exists IT elective courses which has a goal to teach programming and more specifically, use others code *[6]*. The latter is relevant for video game modding since this is, as previously mentioned, an act of adapting your own code to already existing software.

It is a sign that programming will be a larger part of the school's curriculum since the Norwegian Directorate for Education and Training has proposed making programming a mandatory course in primary school and lower secondary school *[7]*.

There are many different tools and approaches to successfully “modding” a game, and Lua scripting is an example of such a tool which we used for our project. Through Lua scripting, students will be able to modify the game to implement their own ideas with the scripts they create, resulting in a diverse collection of projects between students.

In addition to Lua scripts, we used these tools to add modding support:

1. Moonsharp - a C# implementation of the Lua interpreter
2. C# - the primary programming language for the Unity game engine
3. Unity - the game engine serving as the basis for our modding interface
4. Turn-Based Strategy Framework

What is Lua and why use it?
Lua is an interpreted scripting language which easily can be embedded into other programming languages. Lua is also heavily used in video game modding since it is lightweight and fast *[8]*. The latter is especially useful for video games which can be computer intensive. In addition, a scripting language like Lua is useful because this enables video game modders to edit code during runtime without having to reboot the game that they are modding *[9]*. And last, but not least, the simple syntax is useful for people with no prior programming experience such as high school students. This advantage helps the students to focus more on the programming rather than remembering the correct syntax.

Project Description

The task was mainly about implementing a Lua integration for Progress’ upcoming Real-Time Strategy (RTS) game. However, during the project we had not been given access to any codebase related to Progress’ game. Thus, we have implemented the Lua integration on top of a self-made turn-based game using the Turn Based Strategy Framework *[10]*. Our Lua integration will be compatible with Progress’ upcoming game since they also will be using the mentioned framework to develop their game. This Lua integration will give users the ability to learn basic programming principles such as…

1. Defining, accessing, and using meaningful variables.
2. Writing own functions, and/or editing pre-defined functions to a new purpose.
3. Creating useful Objects and attribute values to these Objects.
4. Have multiple scripts communicating through events.

In addition, the Lua integration should give users access to use more advanced programming features and principles such as…

5. Improve knowledge about Objects and introduce encapsulation to make code cleaner and more secure.
6. Teach reusability and lessening duplicate code with inheritance.
7. Give a thorough rundown of multitasking through coroutines.

Other tasks, which were also part of our project were to...

1. Make sure that mods are registered and working in the game.
2. Make sure that sound and images can be used with mods.

Our modding implementation

Our modding implementation can be described by two parameters. The first is the amount of game components which are modifiable. The combat, movement, sound effects, faction images, and terrain are components in the game which can be modified through our modding implementation. The second parameter is how much each game component can be modded. Generally, this is limited for each game component. This will be further discussed in the conclusion section.

Targeted audience

Audience for using the Lua integration

The primary audience for the Lua integration is high school students with an interest in programming or modding. Other potential audiences are people with the same interest or people who will become invested in Progress' game and have a desire to expand the game.

Audience for the report

The primary audience for the report is people with a background in programming and with an interest in video game modding or modification of video games as an approach for teaching programming. Secondary audience are teachers and education departments. This Lua integration could be interesting because there does not exist many video games with modding support with the intention of teaching programming. Teachers and education departments could monitor the effect of teaching programming through modding and compare it to traditional programming classes or video games with programming as its gameplay.

Game companies or game developers are also aimed for this report. The effect of modding a game could be of interest to them because modding support can lead to a video game with a longer life span *[11]*.

Our background & competence

The group consisted of a mix of application-focused and game programming-focused students, however we shared the same common subjects as we had gone through the same bachelor program.

There were certain differences in experience between us since we had taken different courses, or that some of us had taken less courses than the others. The experiences that we had in common is a thorough understanding with object-oriented programming languages, namely languages such as C#, C++, Java and more. However, our experience differed related to experience with the game engine Unity. Some of us have much experience with Unity, and others have less.

We also shared subjects pertaining various aspects of software development, including designing and testing software, system development process, software security, and more. We also learned the importance of software methodologies and how they are implemented in the software development process. We believed this knowledge set a solid foundation for tackling this project.

What we had to learn

The concept of scripting within the context of developing video games or providing a modifiable interface was an unfamiliar territory of programming in our group, making it a completely new concept for all of us to dive into. The closest some of us had come to scripting was in C# for Unity, but the video game modification aspect of it was something we all had to familiarize ourselves with for this project. Being our first hands-on experience with a proper scripting language intended to be used as a moddable interface for a game engine meant potentially coding in a way not like what we were used to.

Although most of us had heard about the Lua language, particularly in the context of video games, none of us had any actual prior experiences with the language. As this was to be the main tool for game modification in this project, it was therefore important for our group to prioritize familiarizing ourselves with what the language offers. Thankfully, the experience and knowledge obtained throughout the course of this bachelor program made learning Lua feel like a natural extension of our established programming knowledge.

Aside from Lua we were to use the Unity game engine in our development and have the Lua scripts work in conjunction with it. Since some group members were lacking experience in Unity in order to complete the project task, we set off time, especially during the first weeks of this semester, to learn more about the core features of Unity. This included learning about game objects, components, making prefabs, learning how to write and attach scripts to game objects in order to modify game objects and their components.

We could not simply just connect Lua scripts directly into Unity, however, as Unity primarily uses and is built around C# scripts. In order to make Unity recognize our Lua script files via C#, we had to learn how to use the tool "Moonsharp". This acted as an interpreter for Lua and written entirely in C#, making it compatible with the game engine. Due to the lack of extensive tutorials and information online on how to properly use the tool especially in a Unity setting, this quickly became arguably the hardest utility for us to learn.

Development Framework

Why we chose Scrum

Requirements are unknown up front and they can change *[12]*

We initially chose Scrum because we believed requirements would likely change during the project. This was because creating a modding implementation for Progress' upcoming game depends on what features Progress' game will include. Since our modding implementation is dependent on a video game currently in development then that means that changes can appear which again means that requirements can change. As previously mentioned, we did not add modding support for Progress' game, but we added it for a game we developed ourselves. Therefore, our initial reason for choosing Scrum did not apply anymore.

However, we still believe that choosing Scrum was a good choice because video game development and development related to games such as modding are suitable for agile methodologies. It is hard to determine if game features or modding features works as expected before seeing it in action when it gets implemented.

Retrospective meetings are helpful tools for newcomers *[13]*

Since we have never applied any development methodologies in a real-life scenario before it is valuable for us to have reflection incorporated into the development methodology. Retrospective meetings can help us to reflect over what went well and what went bad from the previous sprints and make adjustments to the next sprint backlog in order to learn from mistakes.

Why we did not use Extreme Programming or Kanban

Since Scrum has sprints, we had a plan for upcoming tasks for the upcoming sprint weeks. This would help us better estimate how long certain tasks will take and be better prepared to reach our deadlines for our development, and more importantly, the bachelor report. Kanban and Extreme Programming does not have sprints in the same sense as Scrum and are therefore methodologies which are harder to plan ahead of time with *[14]*.

How we implemented Scrum

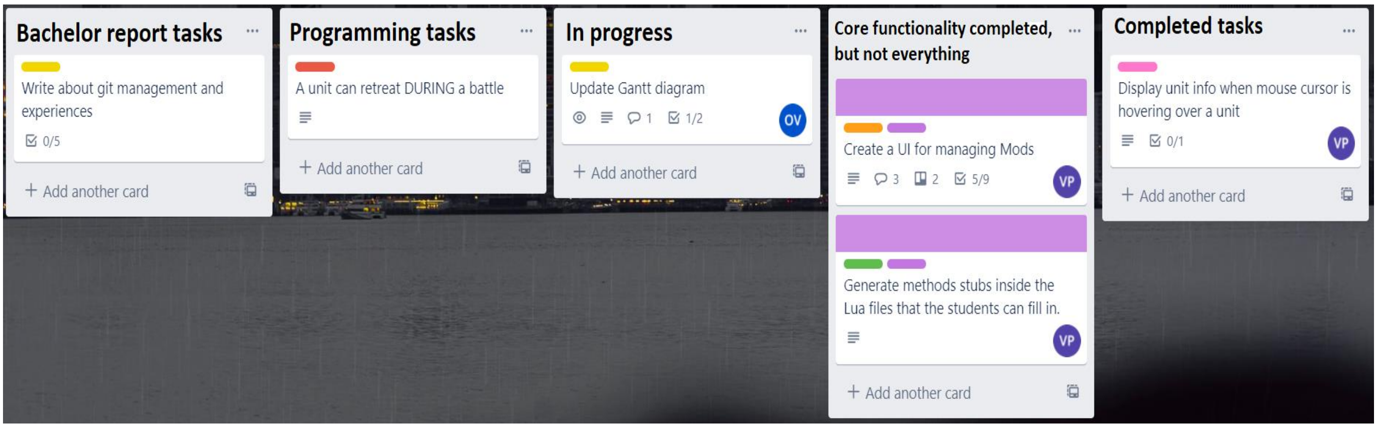
The following section will describe how we used sprints, Trello, to organize our sprint backlog, various scrum events and more.

Sprint

During this project we have mostly had 2 weeks sprints. A sprint length of one week could be too short to get a long-term plan of our work. Therefore was 2 weeks sprint a fitting sprint length for us. In addition, this sprint length is short enough to keep us on our toes with regards to frequent deadlines.

Sprint backlog

When we created the sprint backlog, we focused on having a clear definition of when a task is done. We think that having a clear definition of a task makes it easier for us to be productive when we have clear tangible tasks that need to be done. Lastly, the sprint backlog was easily accessible on our Trello board.



Bachelor report tasks
- Write about git management and experiences ☑ 0/5

Programming tasks
- A unit can retreat DURING a battle

In progress
- Update Gantt diagram 1 ☑ 1/2 OV

Core functionality completed, but not everything
- Create a UI for managing Mods 3 2 ☑ 5/9 VP
- Generate methods stubs inside the Lua files that the students can fill in. VP

Completed tasks
- Display unit info when mouse cursor is hovering over a unit ☑ 0/1 VP

Sprint review- and planning meeting

In our project we merged the review- and planning meeting. We did this because it felt natural to plan the next sprint after we showcased what we had accomplished in the previous sprint. For the most part our client gave us some tasks to do for the next sprint and then we got started on the new tasks. However, we let our client know if we were unsure if we were able to complete the tasks given in the sprint. This resulted often in our Product owner to further specify the priorities of the tasks given.

Specialist vs generalist

This is an important note to address since Scrum defines generally that team members should be more generalist than specialist. Through the project we have focused on being generalists, but we have been open for individual team members to have some level of expertise in certain areas. What we do not want to do is to be too dependent on a skill which one team member has.

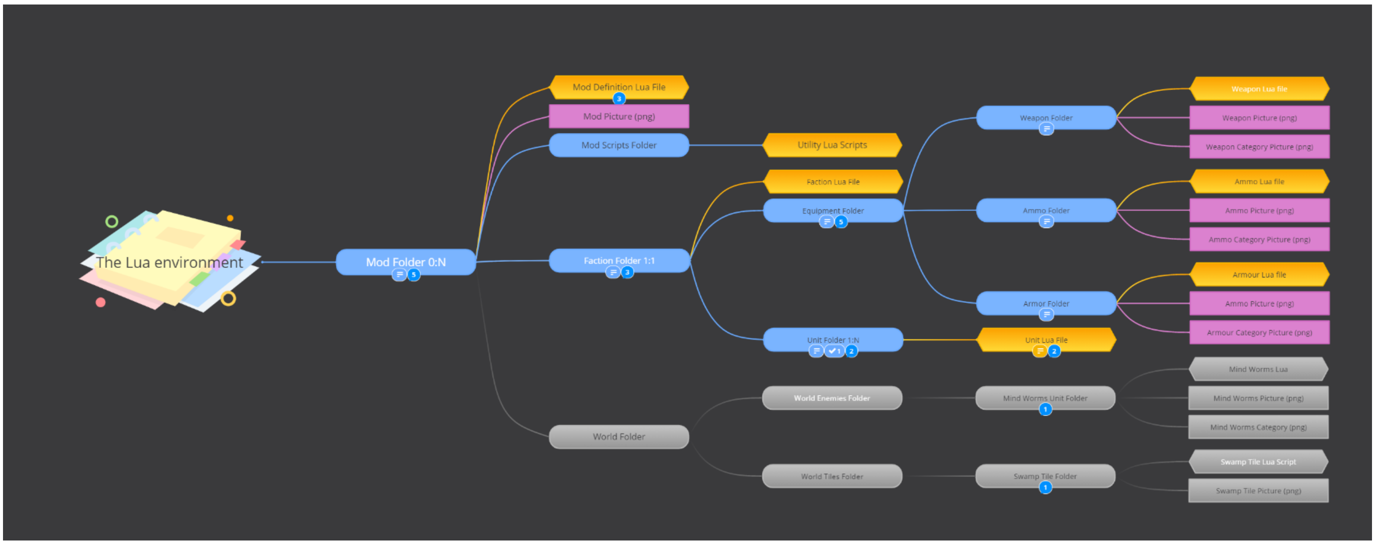
Boundaries

A few boundaries were decided by our product owner during the early planning stage of development. These boundaries include:

- **Windows** support by default, multi-platform as a secondary priority.
- We are to use the **git** repo provided by Progress.
- Development should be done in **Unity 2019 LTS** version.
- We must use the TBS framework asset in our development.
- The modding must be done through Lua.

Workflow

The project had different modules that could be worked on independently, so we split responsibility for these modules very evenly among our group members based on their preference and previous experiences.
To keep track of our work, we used **Trello**. And to time track the individual tasks in our respective modules, we decided to use **Toggl**.
Working separately like this could easily have made things harder for us than necessary, but by using an **Ayoa** mind map we could at any time see what the end-result should be.



Git and branches

Throughout the project we have mainly used branches in a way such that each team member has their own branch. For the sprints in the middle of April where we implemented UI, prototype, real-time movement and combat, we started having branches dedicated to different features.

Struggles with git

At the beginning of April, we had three branches that had many differences from each other. The process of merging the branches together took many hours because of merge conflicts which arose. After this painful experience we were more focused on merging at least once a week in order to avoid making the code from the branches too different.

Meetings

To keep our working tree clean, we would have meetings through a private discord server to discuss work that had been done that week. We also spent this time resolving merge conflicts between our feature branches, if any.

Every Wednesday we had meetings with our supervisor, Tom Røise. Tom guided us throughout the project with our focus on implementation and its relevance to the thesis, along with any other questions we might have.

Richard Barlow was the project owner and representative for Progress. We met with Richard every other week at the end of each sprint. He helped us with what he wanted the final product to be and gave us examples from previous experiences that he thought could be helpful for us.

Limitations

Our client decided that we should use the TBS Framework. This meant that we were limited by only using code which was compatible with this framework. Moreover, our TBS game might not represent a close version to what Progress is developing since the framework as the name implies is turn based while Progress' upcoming game will be real-time.
In addition, we had to use the 2019 version of Unity for our project to be compatible with what Progress uses themselves. This makes it easier for Progress to use parts of our Unity project for their upcoming game if they want to.

Group roles

Name: Assigned Role	Responsibilities
Viktor: Project leader Titles: Unity consultant	Project leader: Ensures that the goals of the project are always clear for the group and clarifies any misunderstandings or confusions about the project. Unity lead consultant: To be the primary person to help with the Unity engine and with tasks related to it.
Oscar: Social coordinator, scheduler, Notetaker	Social coordinator: Arranges social events for the group where we can hang out and do something besides work. Scheduler: Arranges meetings with the project owner and the group supervisor. This role is also the contact person for supervisor and client for the group. Notetaker: Take notes during each meeting and then make a summary of what was said and/or decided during the meeting.
Genti: Arbitrator/monitor, encourager	Arbitrator/monitor: Observes over the state of the group and its members and regularly brings up group climate and process during meetings and discussions, especially if he or she senses tension or conflict brewing. Encourager: Give moral support to the group, be active in praising good work and to keep the group in high spirits.
Ole: Devil's advocate/ Quality Controller, Security supervisor.	Devil's advocate/ Quality Controller: Remains on guard against "groupthink" scenarios. Ensures that all arguments have been heard, and looks for holes in the group's decision-making process, in case there is something overlooked. Security supervisor:

	Make sure that the group follows a security standard to ensure a secure development. He is not responsible for implementing security, just the rules the group must follow when developing.
All: MoonSharp/Lua consultant, Report supervisor.	MoonSharp/Lua consultant: To be familiar enough with MoonSharp and Lua to be able to assist in tasks related to them. Report Supervisor: Observes the state of the Bachelor report and has the group regularly work on the report.

Specification

Goals

- The modding integration should be supported on multiple platforms.
- The users must have the power to modify game components to improve gameplay.
- The modding integration should be a suitable environment to learn programming for unexperienced users.

Functional requirements

- Mods shall be edited during runtime.
- The folder structure for a new mod shall be generated through a UI in the game.
- The Lua integration shall be able to mod factions, units, terrain tiles and equipment.
    - Damage and defense calculations in units shall be modifiable.
    - The unit’s sound effects shall be modifiable.
    - Gameplay events related to when units move into a tile and exits a tile shall be modifiable.
    - Faction Icons shall be customizable.
    - Units and equipment stats shall be modifiable.
    - New terrains can be created, and existing ones shall be modifiable.
- All mods and factions must have unique names.
- All syntax, runtime and naming errors related to mods shall be shown to the end user.

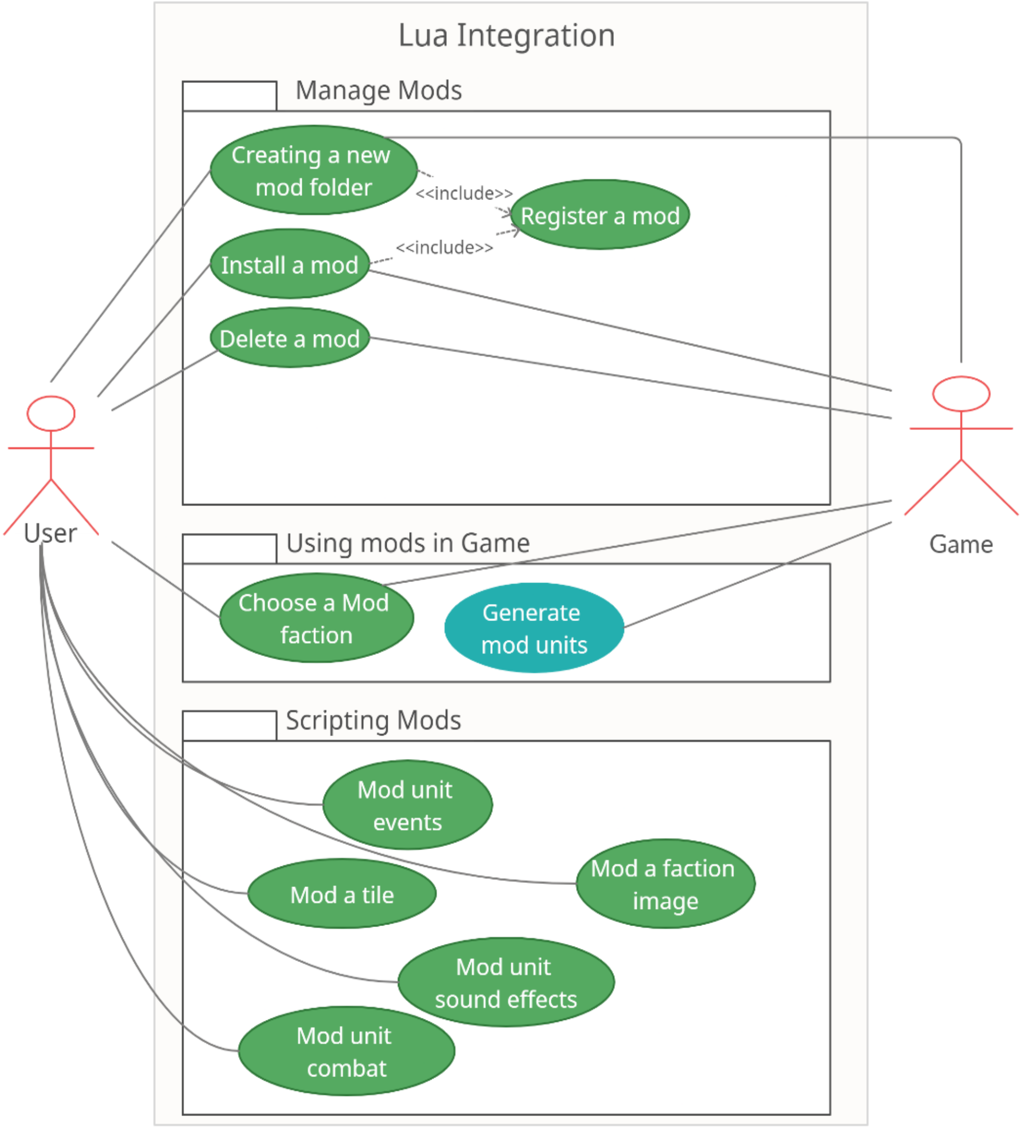
Non-functional requirements

- The system shall not decrease its frame rate below 40 fps when Lua files are executed.
- Lua files shall not use more than 50MB of memory.
- The system should be supported on Windows 10, mac OS 11 (Big Sur) and the latest LTS version of Ubuntu 20.04.2.0 (Focal Fossa).

Security requirement

1. The Lua files shall not have permission to read, write and/or delete files on disk.
2. The Lua file shall not be able to use operating system level functions. This includes executing operating system shell commands.
3. Mod folders are to be scanned for malicious content before being uploaded to Progress' database of mods.

Use Case Diagram



This use case diagram has been split up into three packages related to different use case categories. The actor "user" represents every user in the game including a game modder. The "game" actor represents the TBS game with the Lua integration.

Use Case Description

Use case	Creating a new mod folder
Actor	Student, game
Goal	Generating a new mod folder in the Unity's StreamingAssets folder.
Precondition	The user has opened the game and arrived at the main menu.
Normal flow	• Student clicks on "Mods". • Student clicks on "Create Mod". • Student fills in mod-, faction- and unit names in text fields. • Student clicks "Create new mod". • The game will generate the mod folder structure inside StreamingAssets.
Alternate path	If either the inputted Mod name or faction name are already taken, the mod folder does not get generated and a message stating that the names are already taken is displayed to the user.
Postcondition	A new mod folder structure is generated inside the StreamingAssets folder, and the mod, faction and unit folder are set to the names that the student has inputted.

Use case	Delete a mod
Actor	Student, game
Goal	Deleting a local mod folder
Precondition	A local mod folder exists and is installed.
Normal flow	The following actions are performed: 1. Student clicks on "Mods". 2. Student clicks on "Uninstall mod". 3. Student selects the mod or mods that they want to uninstall (from a list of installed mods). 4. Student clicks on uninstall. 5. A message appears indicating that the action has been completed.
Postcondition	The mod folder and its Mod Name and Faction registry are deleted.

Use case	Install a mod
Actor	Student
Goal	That the student can install a mod created by another user.
Precondition	The user has opened the game.
Normal flow	1. Student clicks on "Mods". 2. Student clicks on "Show available mods online" 3. Student picks a mod to install and clicks on "download and install". 4. The game gives a message indicating that the mod has been installed.

Alternate path	The student moves the already decompressed mod file into the Streaming Assets. Thereafter, the student continues from step 1.
Postcondition	The mod has been installed which means that it will alter the behaviour of the game.

Use case	Choose a mod faction
Actor	Student
Goal	A faction is chosen, and the units is assigned to a unit type belonging to the faction.
Precondition	The user has opened the game and arrived at the main menu.
Normal flow	1. Student clicks on “Play”. 2. The system lists all factions within the game and the factions from the available mods. 3. Student chooses faction for a player. 4. Student clicks on the “Start” button.
Postcondition	The tile-based map is loaded, and the units has been registered to a unit type belonging the chosen faction.

Use case	Generate mod units
Actor	Game
Goal	Units are registered in a faction
Precondition	A player has chosen a faction and started the game.
Normal flow	• The game spawns the units onto the tile-based map. • The game makes sure that Lua files of the chosen factions alters the behavior of its units.
Postcondition	Units from the chosen faction are spawn onto the map.

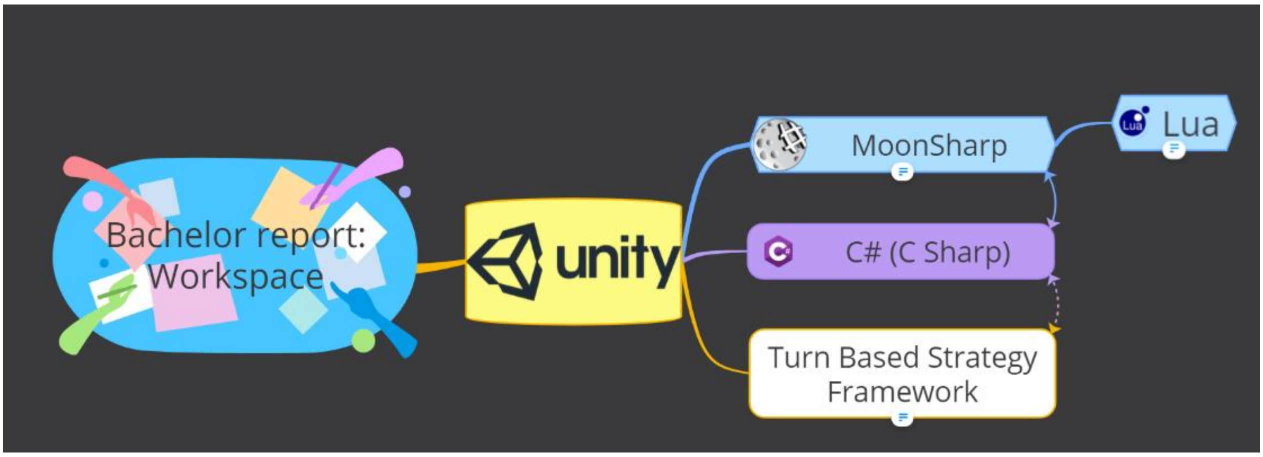
Use case	Mod unit events, mod unit sound effects and mod unit combat (modifiable elements are used as unit events, -sound effects and –combat).
Actor	Student
Goal	Change the behavior of the modifiable elements in between play sessions or during gameplay.
Precondition	A mod has been created and the student has opened the unit Lua file inside the newly created mod.
Alternate path #1: mod unit events	1. The student edits the Lua functions related to unit movement.
Alternate path #2: mod unit sound effects	• The student specifies the sound file that they want to use as sound effect in the unit Lua file.
Alternate path #3: mod unit combat	• The student alters the combat calculations by modifying the “dealDamage()” and/or “defend()” function in the unit Lua file.
Postcondition	The game loads in the Lua files including the new changes from the student.

Use case	Mod a faction image
Actor	Student
Goal	Successfully use a custom image for the student’s faction
Precondition	A faction image exists inside the faction folder
Normal flow	1. Create a variable for the faction image in the faction Lua file. 2. Assign the variable the image file name.
Postcondition	The game loads in the faction Lua file with the new changes.

Use case	Mod a tile
Actor	Student
Goal	A tile which is used in the map has been modified.
Precondition	A mod has been created and the student has opened the file related to terrains.
Normal flow	• The student changes different attributes related to tile. The different attributes are... • The colour of the tile. • The name of the tile. • A defence number which gives points to units which are standing on the modified tile while defending. If the defence number is 2, then the unit has two additional points in their defence attribute. This will decrease the damage that they will take when another unit attacks it. • A variable which determines if the tile is walkable or not. Mountains is an example of a terrain tile which could be used as a non-walkable tile.
Postcondition	The game loads in the Lua files including the new changes from the student.

Technologies

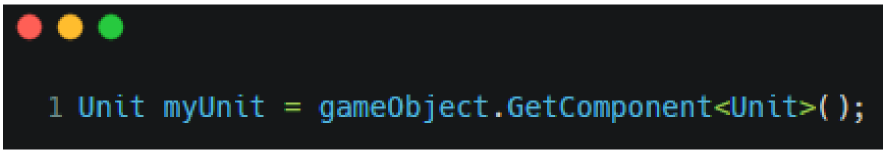
The mind map below gives an overview of the different tools and technologies which have been used in this project.



This mind map shows that the game engine Unity is the main tool we used. All other tools interact with Unity in order to create the Lua integration.

Unity

Unity is a game engine which is popular among indie developers *[15]*, like Progress and supports many platforms. In Unity almost every object that can be modified is of type “Game Object” *[16]*. Objects such as units and hexagonal tiles from TBS framework is of type “Game Object”. One common operation to do with “Game Objects” is to retrieve components from it. Components define the behaviour of “Game Objects” *[17]*. They can alter the position, the physics or the colour of them. In many code examples under “Implementation” a script will be retrieved from a “Game Object”. An example is retrieving a script component from an object of type Unit:



```
Unit myUnit = gameObject.GetComponent<Unit>();
```

Here a script component is retrieved from a “GameObject” using the function “GetComponent<>”.

C# and .NET

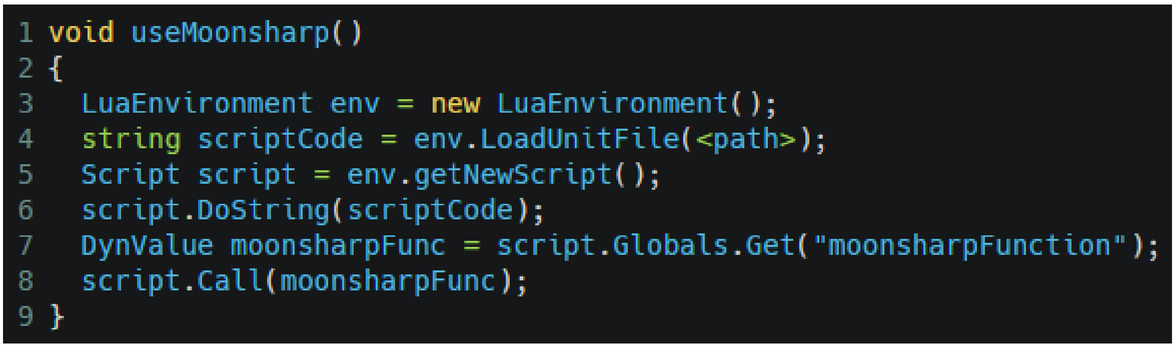
C# is a general-purpose object-oriented programming language which can be used with the .NET framework. The mentioned framework is cross-platform, and it can be used to develop applications for desktop, mobile and games. There have not been many issues with learning the language for our group since the syntax is similar to C++, which we are familiar with.

Moonsharp

Moonsharp is C# implementation of the Lua interpreter. This tool has been mainly developed by Marco Maostropaolo with some contributions from other users *[18] [19]*. The features that we have used includes...

1. Expose C# functions to Lua which enables Lua scripts to call functions from C#.
2. Expose Lua functions to Progress’ game. This way functions written by modders can be called in the game.
3. Expose C# classes to Lua
4. Sandbox the Lua environment. The purpose behind this is to protect Progress’ upcoming game against malicious use of the Lua integration.

Common operations that will be done using Moonsharp is saving a function from a Lua script into a variable in C#. The other common operation is to use the “.Call” method to execute Lua scripts. The code example below is an example of these two operations.:



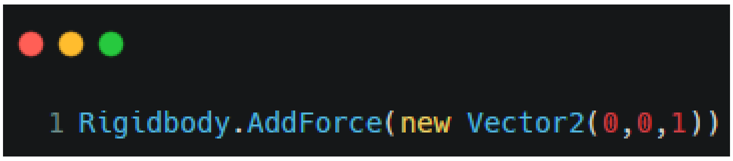
```
1 void useMoonsharp()
2 {
3   LuaEnvironment env = new LuaEnvironment();
4   string scriptCode = env.LoadUnitFile(<path>);
5   Script script = env.getNewScript();
6   script.DoString(scriptCode);
7   DynValue moonsharpFunc = script.Globals.Get("moonsharpFunction");
8   script.Call(moonsharpFunc);
9 }
```

Firstly, the script is retrieved in the “scriptCode” variable on line 4. Then, a Script object is created using “LuaEnvironment”. This class is responsible for configuring Moonsharp and further explanation for it can be found in the security section. Thereafter, a function from a Lua file is saved in the variable “moonsharpFunc” on line 7. Finally, the Script object executes the Lua function with the “.Call” method.

The level of control between C# and Lua

It was hard to get a high-level overview of how a modding implementation would look like. One of the problems with getting a high-level overview are to decide how much Lua or C# controls.

An example could be a game where a character jumps in a 3D space. The Lua integration for this game should make a character jump. A component called “Rigidbody” is needed for the player to achieve this in Unity. Below is one simplified way of implementing this in Unity.



```
Rigidbody.AddForce(new Vector2(0,0,1))
```

Here a vector is added as a force to the Rigidbody component to the player. The third parameter is positive since it represents the z-axis. One way of implementing this in Lua is to expose an object which represents the player and add a “jump()” method which calls “AddForce” on the Rigidbody. The code for Lua will then look like this:

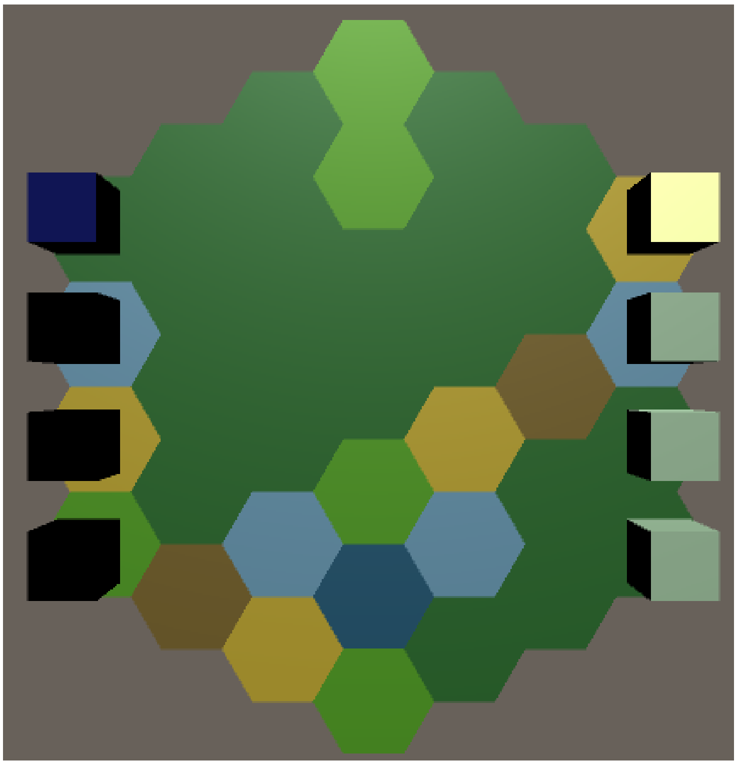
```
Player.jump()
```

This implementation gives little control to the Lua code. Another way of implementing this could be to expose an object which represents the Rigidbody component from Unity. This will give Lua code more freedom, but it increases the chances of Lua code breaking the game logic in some way because of all the possibilities it opens for the modding scripts.

Throughout the development of the Lua integration, we ended up with the approach which gave the Lua code little freedom. This was done to ensure that the game logic did not break.

TBS Framework

Turn Based Strategy Framework, TBS for short, is a highly customizable framework for developing turn-based strategy games and is the key asset in this project as we used it to create a game to use in our development of the Lua integration. It allows us to create custom shaped maps, place objects like units or obstacles on it and play games with both human and AI players.



The main elements in TBS that we modified was the combat system and the cell grid, or game map as it is also called. We modified the combat system to use Lua files and we modified the cell grid to use custom tiles.

In TBS, the game is controlled with the CellGrid script. It keeps track of the game, stores cells, units, and players objects. It starts the game and makes turn transition, it reacts to user interacting with units and cells, and raises events related to game progress. The script is attached to the CellGrid game object which parents all the cells that the game map consists of.

The combat system, as well as unit selection and movement, is handled by sub classes of the CellGridState class. There are three subclasses: CellGridStateWaitingForInput, CellGridStateUnitSelected, CellGridStateBlockInput.

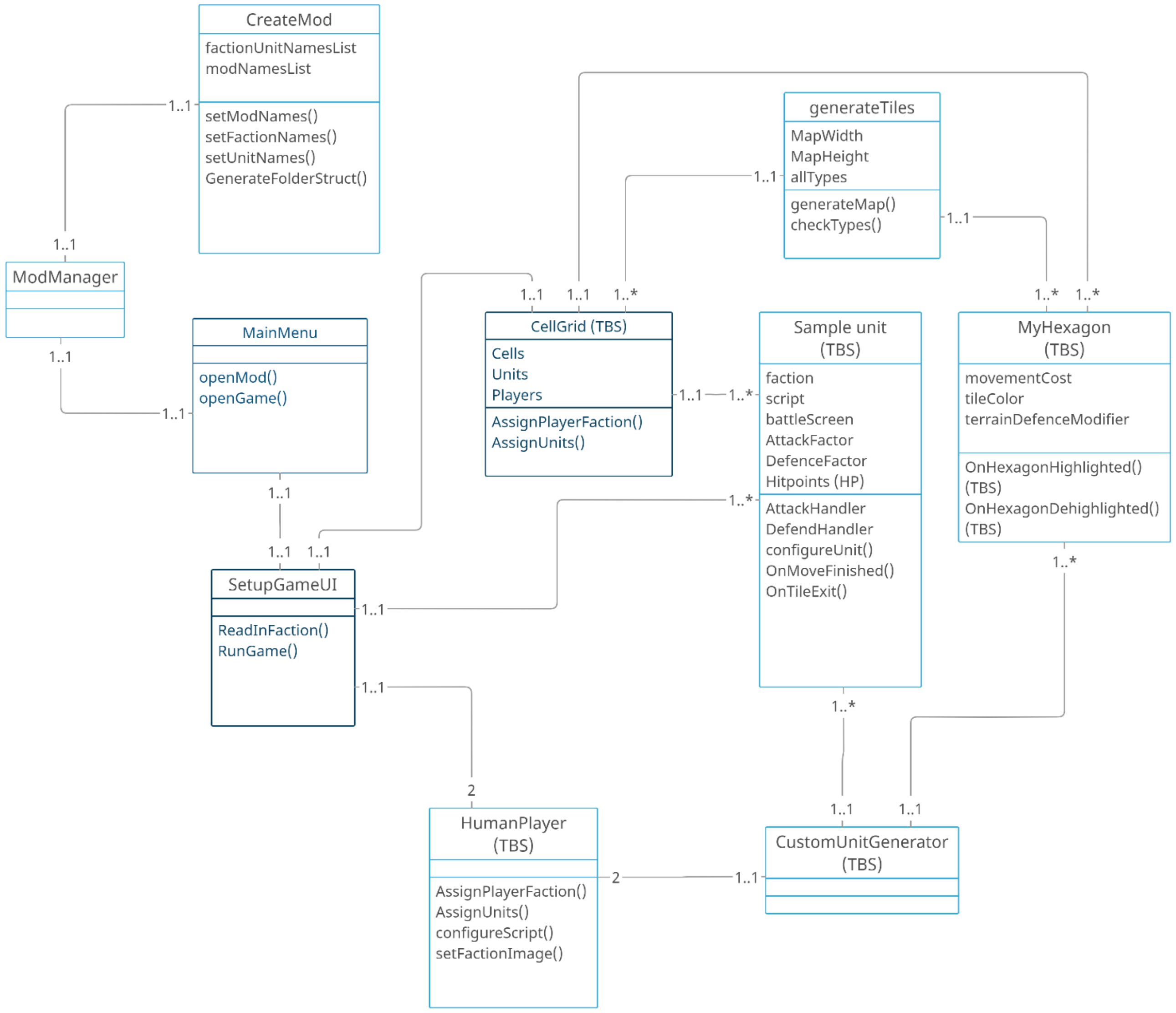
When a unit attacks another unit the attacking unit calls its Attack Handler to calculate the damage. The calculated damage subtracts the health, or the health points, of the defending unit. This happens in the defend handler in the defending unit.

Game Map/CellGrid

As mentioned before the CellGrid game object parents all the cells that the game map consists of. TBS works initially with two shapes, squares and hexagons, and TBS has scripts to implement these two shapes. The square and hexagon scripts each contain abstract classes. So, to create a custom square or hexagon type you would need to create a class that inherits from the square or hexagon abstract class. For our game we created a custom hexagon cell type.

Technical Design

UML



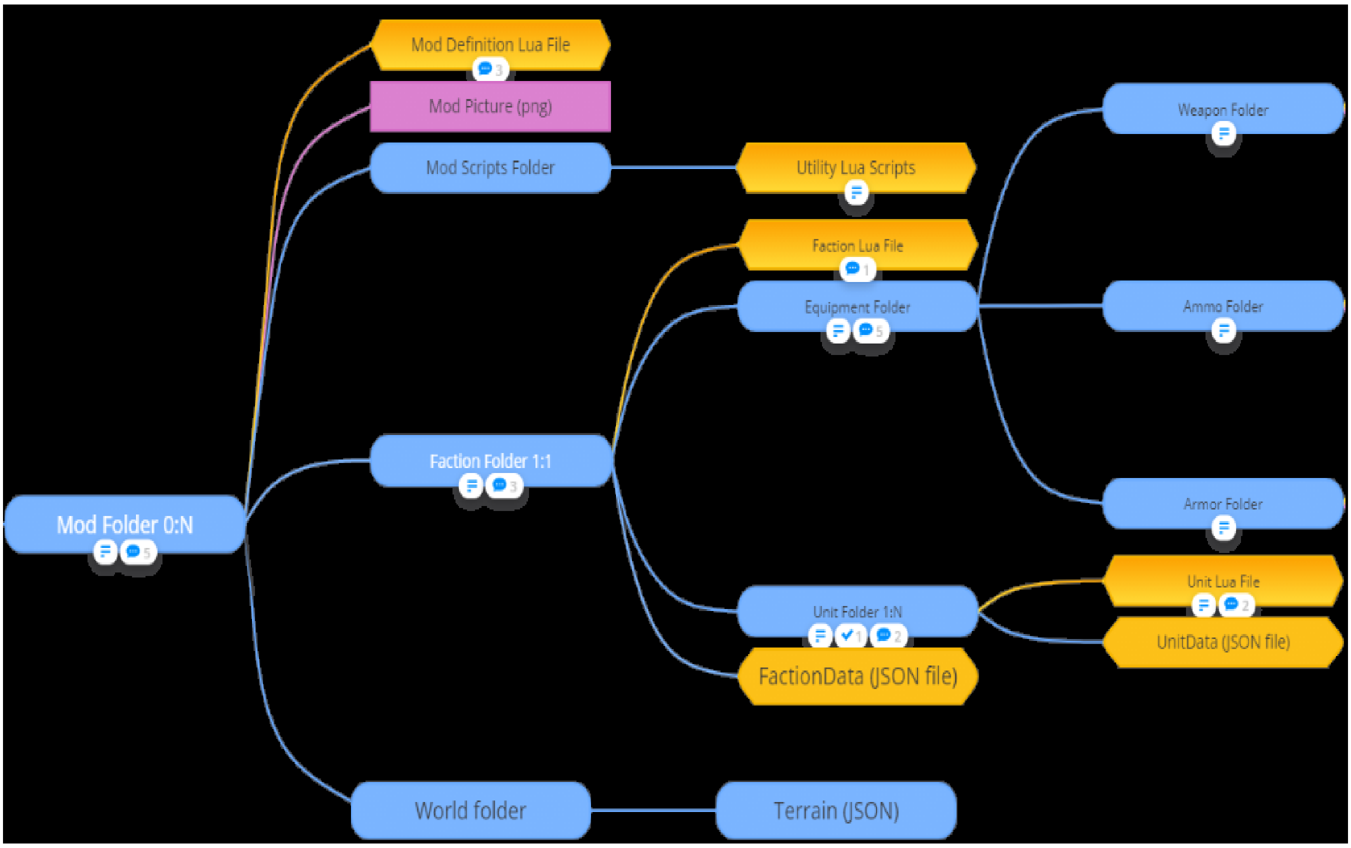
All classes which are marked with “TBS” are classes which comes from the TBS framework. We have modified this framework by either inheriting from TBS classes or overriding virtual methods. An example of this is “Sample Unit” which inherits from “Unit” class. Many classes here will be mentioned later in the report such as “CreateMod”, “generateTiles” related to randomly generated terrain, “Sample Unit” related to combat etc.

Choice related to mod management

Streaming assets

For students to load mods during runtime, as well as being able to create their own mod files, the mod files are handled through the Streaming Assets folder (simply called “StreamingAssets”). Since students will be able to share their mods with fellow students, the game will need to be able to identify the location of these mods, which is made possible via the StreamingAssets folder. Lua scripts are also not a native part of Unity, meaning that the game must load these scripts via StreamingAssets so that they can run within the game *[20]*.

File structure of a mod



The given picture shows the file structure of a mod. Most of the modifications made by the user is done through the Unit Lua file inside the unit folder. Modifications to this file can change combat, movement and sound effects used by units. Mod a tile is done in “terrain.json” file inside the “World folder”. Folders which are not used in the Lua integration includes “weapon”, “armour” and “ammo” folders. These were planned to be developed but they did not get implemented.

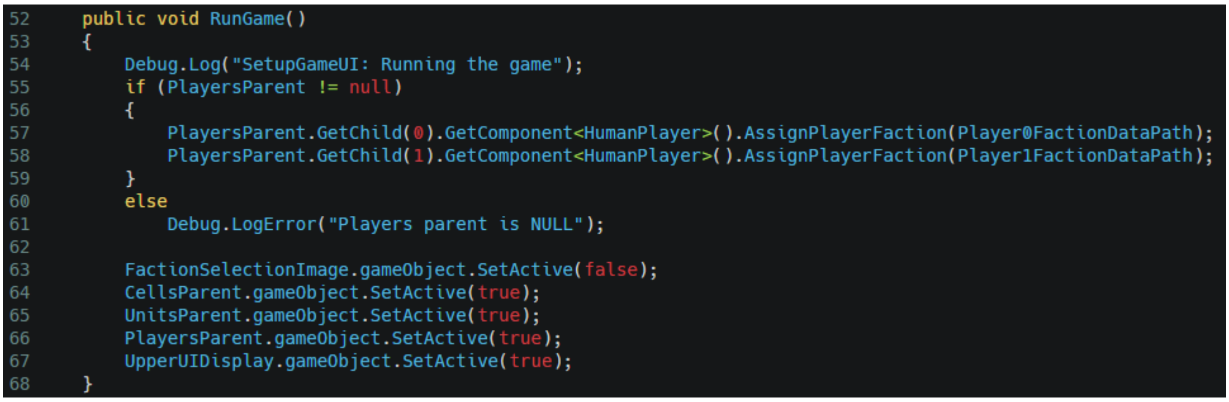
Proxy pattern

Proxy pattern is about making a placeholder for a given class. An example of this pattern is shown if a Lua script directly calls functions from the Cell class, the class in TBS Framework which handles one tile. If this is the case, then the Lua script can call every public function from the Cell class which might break the game. A proxy pattern solution is to create a placeholder class for the Cell class. This proxy only includes the necessary functions and it provide a clean interface with the Lua scripts and the game logic.

Implementation

Configuring a mod

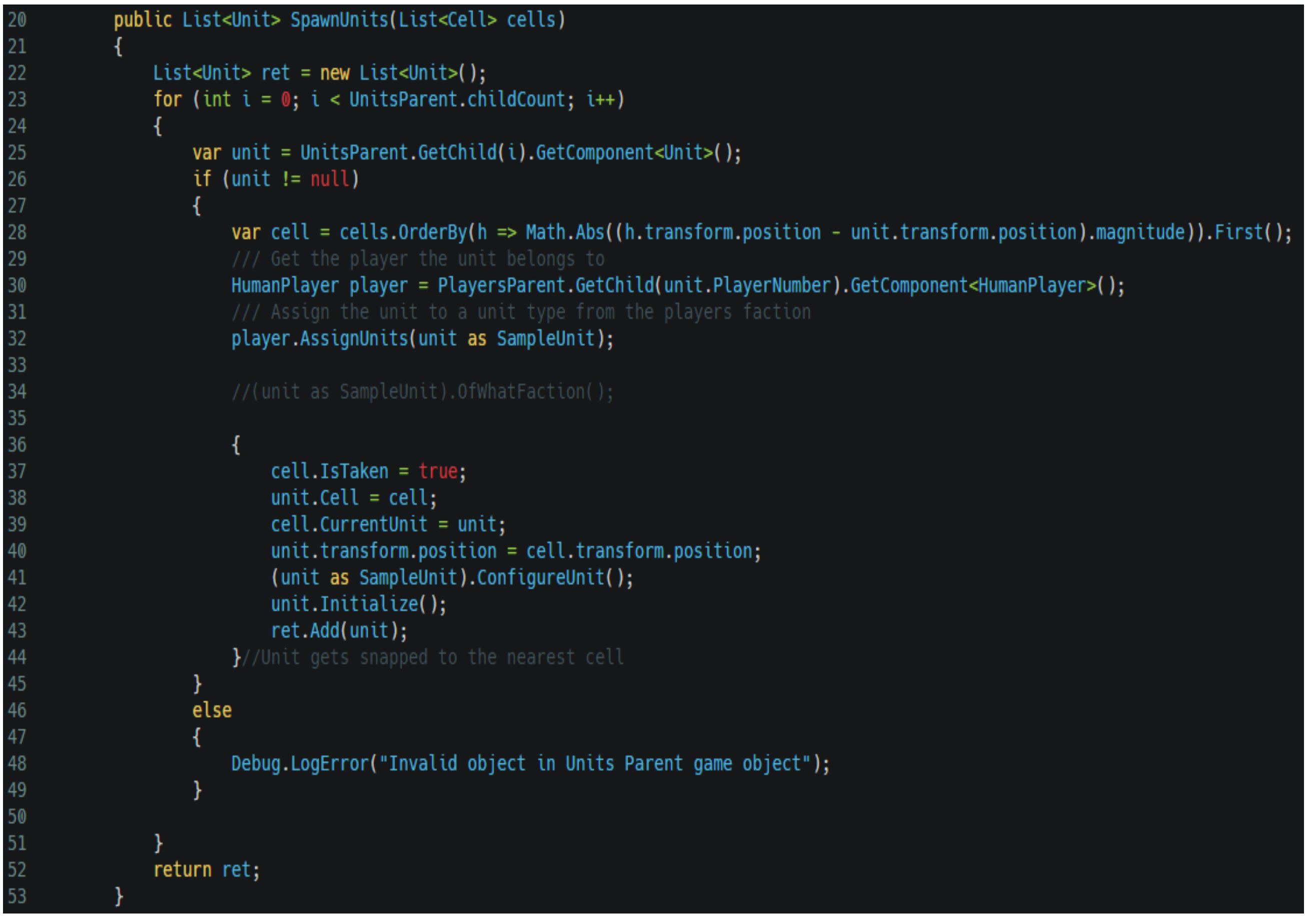
After the player selects the desired units for each side and starts the match, the game loads all the necessary data related to those units. To start off this process, the script “SetupGameUI.cs” starts running on match start-up and executes the function “RunGame()”.



```
    public void RunGame()
    {
        Debug.Log("SetupGameUI: Running the game");
        if (PlayersParent != null)
        {
            PlayersParent.GetChild(0).GetComponent<HumanPlayer>().AssignPlayerFaction(Player0FactionDataPath);
            PlayersParent.GetChild(1).GetComponent<HumanPlayer>().AssignPlayerFaction(Player1FactionDataPath);
        }
        else
            Debug.LogError("Players parent is NULL");

        FactionSelectionImage.gameObject.SetActive(false);
        CellsParent.gameObject.SetActive(true);
        UnitsParent.gameObject.SetActive(true);
        PlayersParent.gameObject.SetActive(true);
        UpperUIDisplay.gameObject.SetActive(true);
    }
```

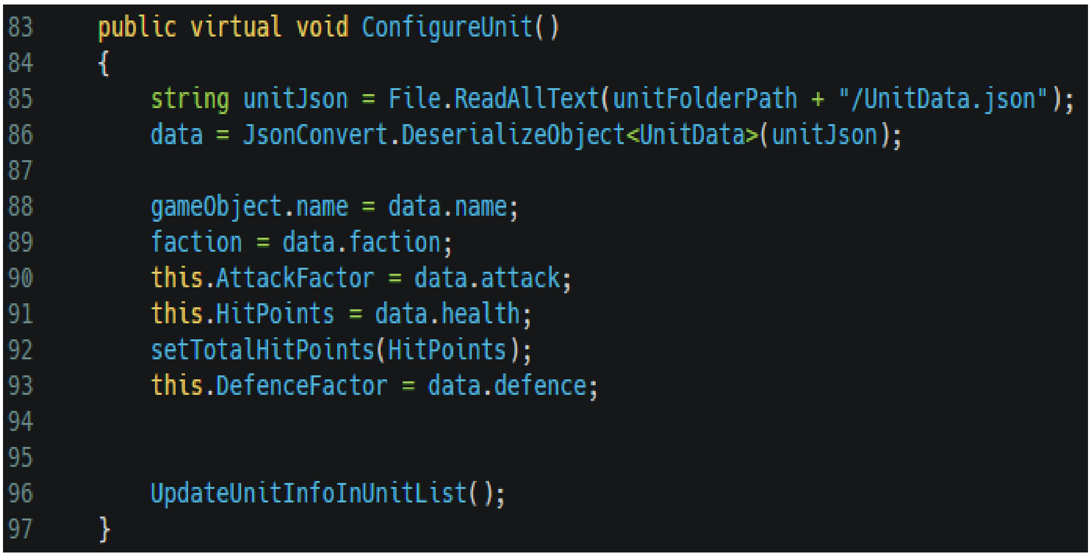
In this function, each player component is retrieved from the HumanPlayer class by utilizing Unity’s “GetComponent” function, and the selected factions are assigned to each player. The faction data for each respective player is identified through the “PlayerxFactionDataPath” variable which directs to the folder path of the relevant faction data.

After each player has been assigned their respective factions, various game objects are set to active at runtime, but the most important for the mod configuration is “CellsParent”, which is in fact “CellGrid” in the TBS framework whose task is to generate the map. When this object is set to active, the script “CustomUnitGenerator.cs” starts running, which will generate the list of units that are to spawn in the game.



```
    public List<Unit> SpawnUnits(List<Cell> cells)
    {
        List<Unit> ret = new List<Unit>();
        for (int i = 0; i < UnitsParent.childCount; i++)
        {
            var unit = UnitsParent.GetChild(i).GetComponent<Unit>();
            if (unit != null)
            {
                var cell = cells.OrderBy(h => Math.Abs((h.transform.position - unit.transform.position).magnitude)).First();
                /// Get the player the unit belongs to
                HumanPlayer player = PlayersParent.GetChild(unit.PlayerNumber).GetComponent<HumanPlayer>();
                /// Assign the unit to a unit type from the players faction
                player.AssignUnits(unit as SampleUnit);

                //(unit as SampleUnit).OfWhatFaction();

                {
                    cell.IsTaken = true;
                    unit.Cell = cell;
                    cell.CurrentUnit = unit;
                    unit.transform.position = cell.transform.position;
                    (unit as SampleUnit).ConfigureUnit();
                    unit.Initialize();
                    ret.Add(unit);
                }//Unit gets snapped to the nearest cell
            }
            else
            {
                Debug.LogError("Invalid object in Units Parent game object");
            }

        }
        return ret;
    }
```

The function goes through the entire list of units that are to be assigned values based on the unit type (as dictated by the comments in the code). Once a unit has been assigned to a player, they are then assigned a cell on the battlefield, followed by attributing the various aspects of the unit through the “ConfigureUnit()” function. This function can be found in the “SampleUnit.cs” script.

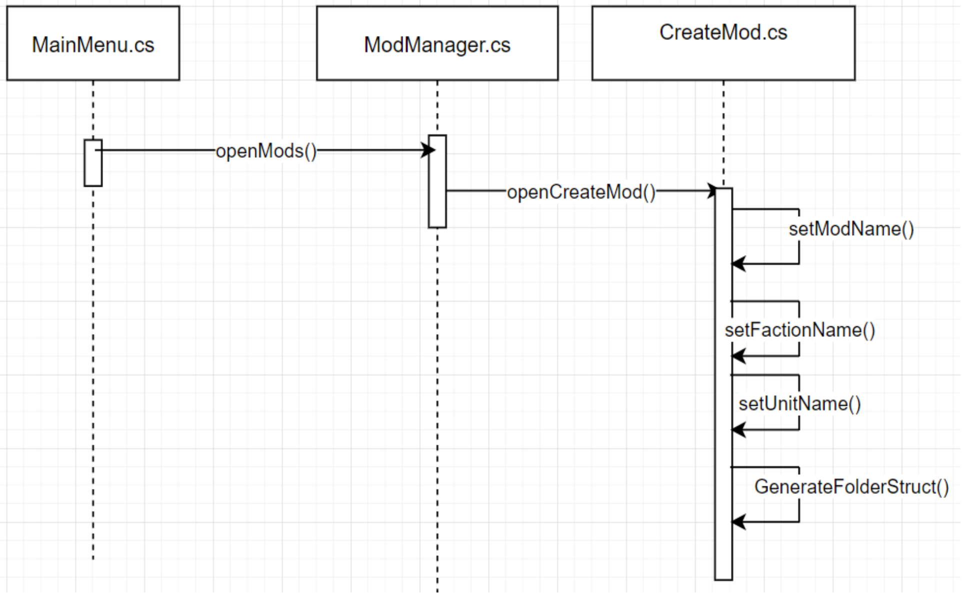


```
    public virtual void ConfigureUnit()
    {
        string unitJson = File.ReadAllText(unitFolderPath + "/UnitData.json");
        data = JsonConvert.DeserializeObject<UnitData>(unitJson);

        gameObject.name = data.name;
        faction = data.faction;
        this.AttackFactor = data.attack;
        this.HitPoints = data.health;
        setTotalHitPoints(HitPoints);
        this.DefenceFactor = data.defence;


        UpdateUnitInfoInUnitList();
    }
```

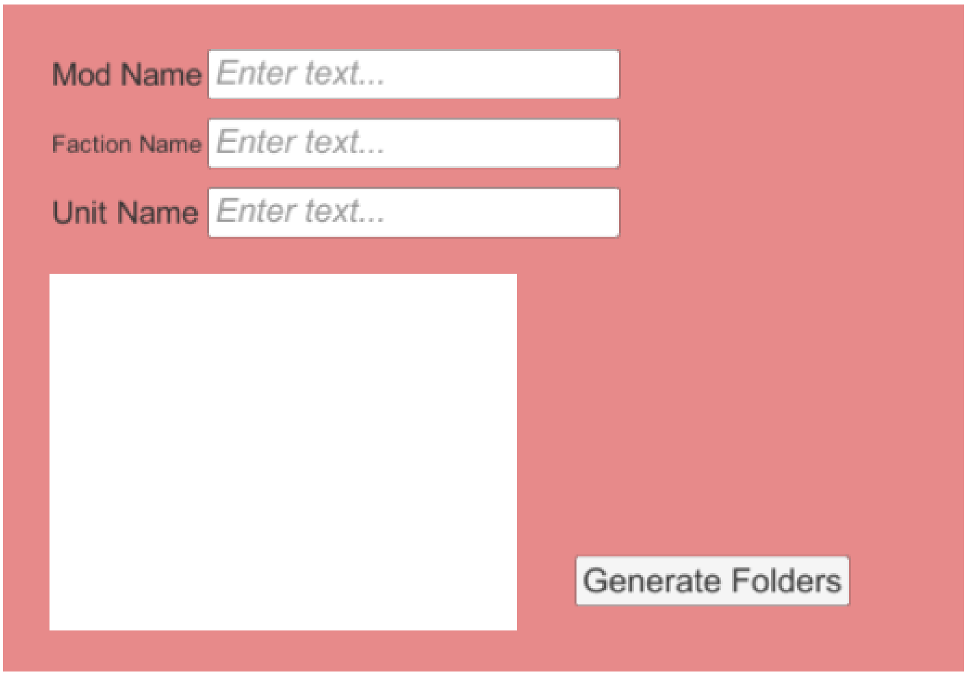
The unit obtains its attributes via this function call, where it will get its name, what faction it belongs to, attack points, health points, and defence factor. The data is read from a JSON file called “UnitData.json” which belongs to the unit in question.

Create a mod

When the user needs to create a mod, they click first on “Mods” button, followed by “Create new mod” button. The high-level code execution is shown in the following sequence diagram.

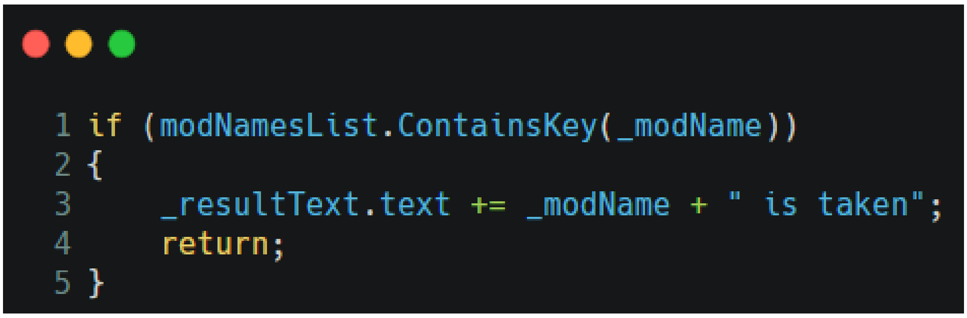


This menu below shows up when the user clicks the “Mods” button on main menu and thereafter “Create new mod”.



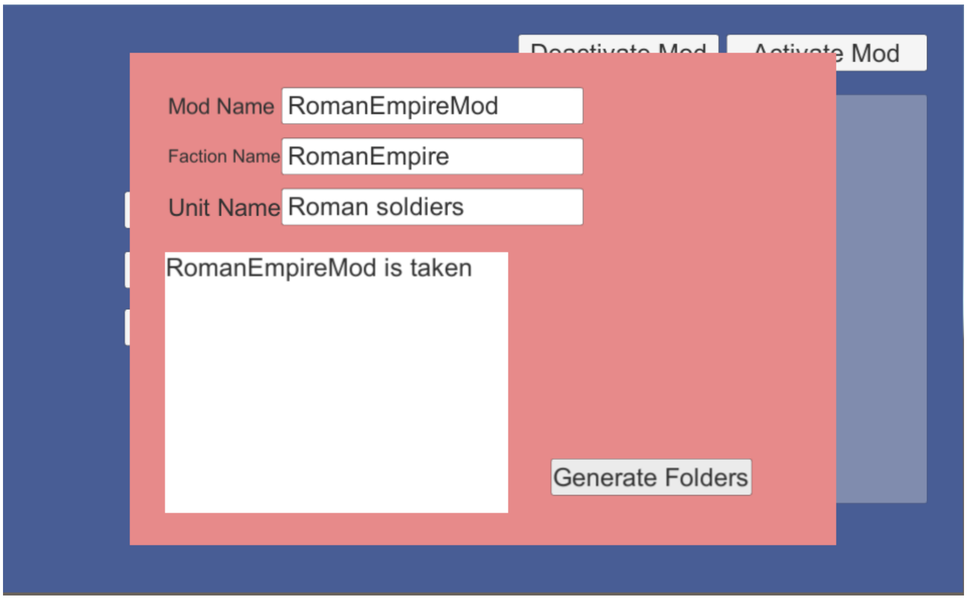
The user then fills in the name for their mod, faction and unit. An example could be a “RomanEmpireMod” with “RomanEmpire” as the faction’s name and “Roman soldiers” as the unit’s name.

One functional requirement states that “No mods shall be created with identical names”. Therefore, the “GenerateFolderStruct()” function checks if the mod- or faction names has already been taken. The code below checks this for the mod names.



```
if (modNamesList.ContainsKey(_modName))
{
    _resultText.text += _modName + " is taken";
    return;
}
```

The picture below shows what happens when a user tries to create a new mod with an already used mod name.



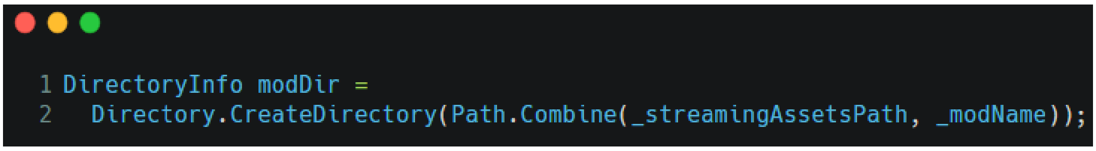
Mod Name RomanEmpireMod

Faction Name RomanEmpire

Unit Name Roman soldiers

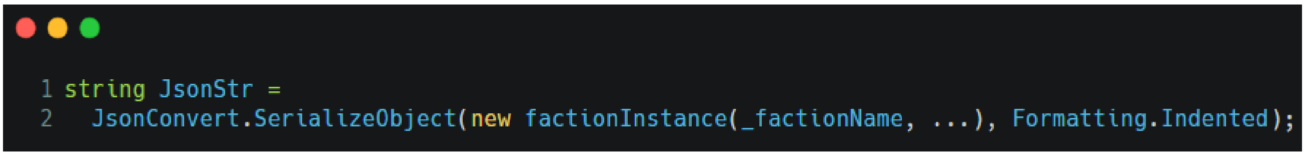
RomanEmpireMod is taken

In addition, it creates folders for the newly created mod. The code below creates the top-level directory for the mod.



```
DirectoryInfo modDir =
  Directory.CreateDirectory(Path.Combine(_streamingAssetsPath, _modName));
```

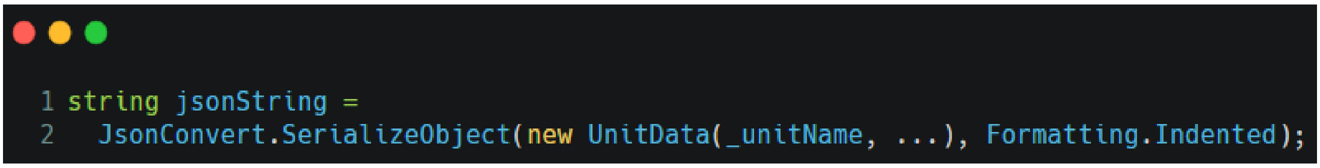
While “GenerateFolderStruct()” creates the directories it also adds Lua and JSON files in the mod directory. The code below serializes a faction object and saves it in the faction directory.



```
string JsonStr =
  JsonConvert.SerializeObject(new factionInstance(_factionName, ...), Formatting.Indented);
```

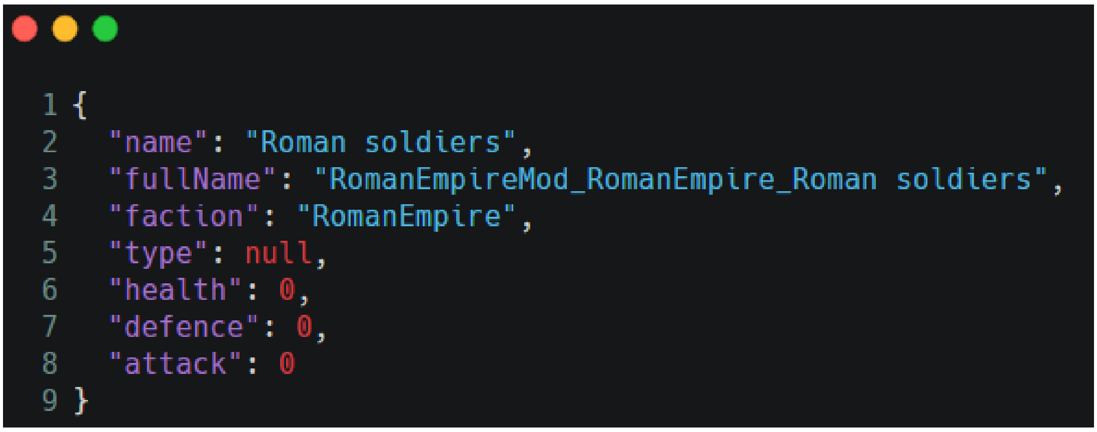
Newtonsoft JSON framework is used here to serialize the faction object.

Moreover, the JSON file for the units are created.



```
string jsonString =
  JsonConvert.SerializeObject(new UnitData(_unitName, ...), Formatting.Indented);
```

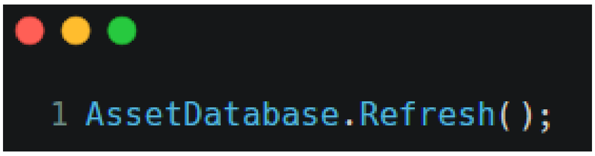
The unit JSON file looks like this:



```
{
  "name": "Roman soldiers",
  "fullName": "RomanEmpireMod_RomanEmpire_Roman soldiers",
  "faction": "RomanEmpire",
  "type": null,
  "health": 0,
  "defence": 0,
  "attack": 0
}
```

This is also the same file produced when you configure a unit. The defence, health and attack attributes can be changed by the modder. This will be relevant for the combat section later in this report.

The last code line in “GenerateFolderStruct” ensures that mods are ready to be used after they have been created.



```
AssetDatabase.Refresh();
```

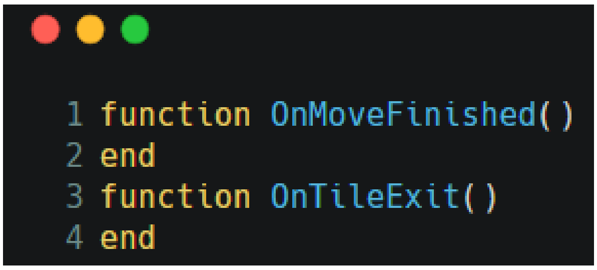
The Asset Database refreshes and updates the game about the new mod files that have been created in the Streaming Assets folder *[21]*.

Use “OnMoveFinished” event in a mod

This implementation shows how we followed our requirement: “Gameplay events related to when units move into a tile and exits a tile shall be modifiable”. This will be shown by using an example mod which colours the tile red when OnMoveFinished event is called. The event is called whenever a unit moves from tile A to tile B.

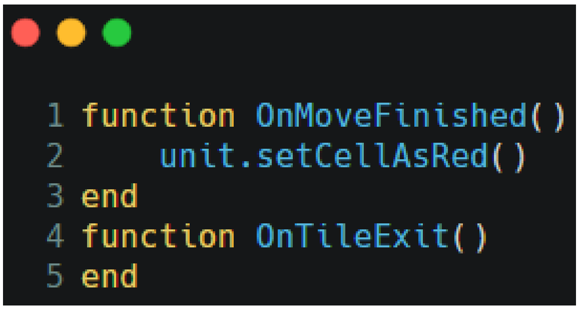
From the modders point of view

First the modder needs to edit the unit Lua file which is in the StreamingAssets folder. The picture below shows the unit Lua file.



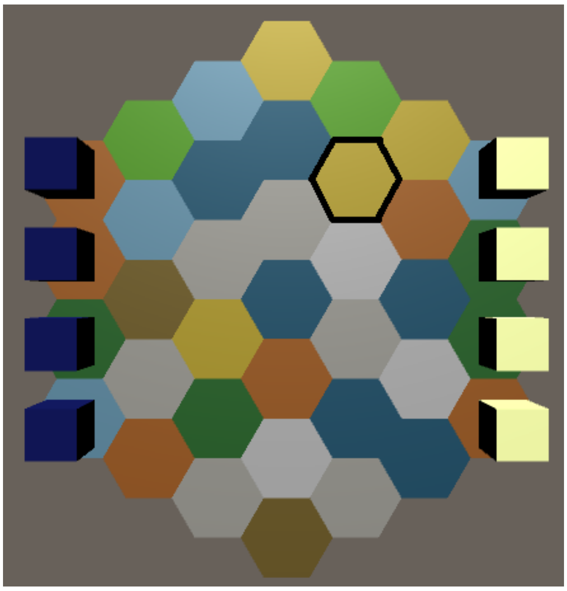
```
function OnMoveFinished()
end
function OnTileExit()
end
```

To change the colour of the tile the modder uses the variable “unit” like this:



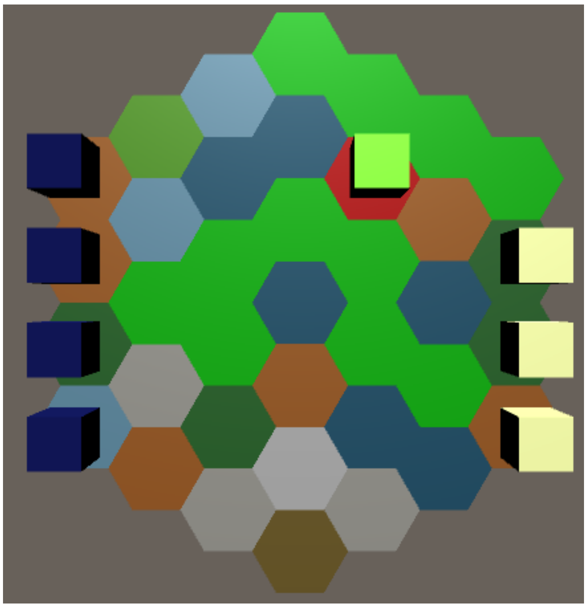
```
function OnMoveFinished()
    unit.setCellAsRed()
end
function OnTileExit()
end
```

Here is a picture before the unit is moved:



The tile which has black borders is the tile the unit is going to move into.

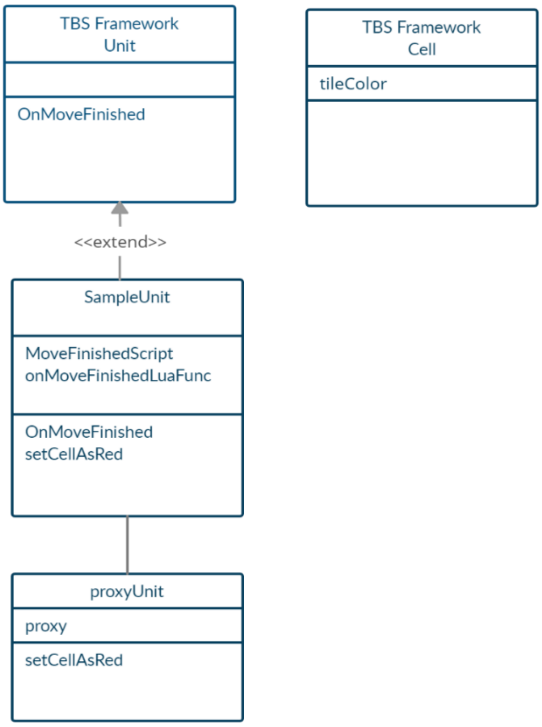
And here is a picture after the unit has moved:



The tile under the unit, which is marked green, has changed the colour into red.

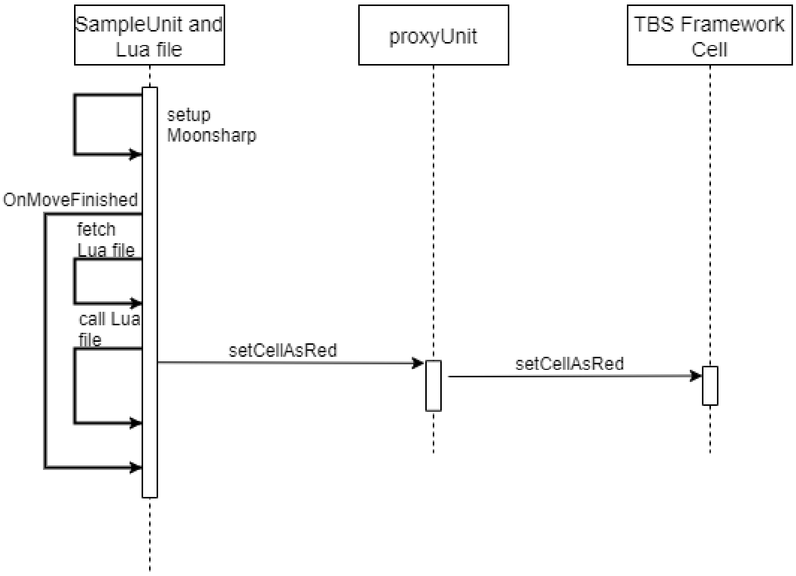
From the Lua integration point of view

As previously mentioned, the “OnMoveFinished” event has been used to achieve this. This event exists from the TBS framework. The UML diagram below will show how our classes are integrated with this framework for this example:



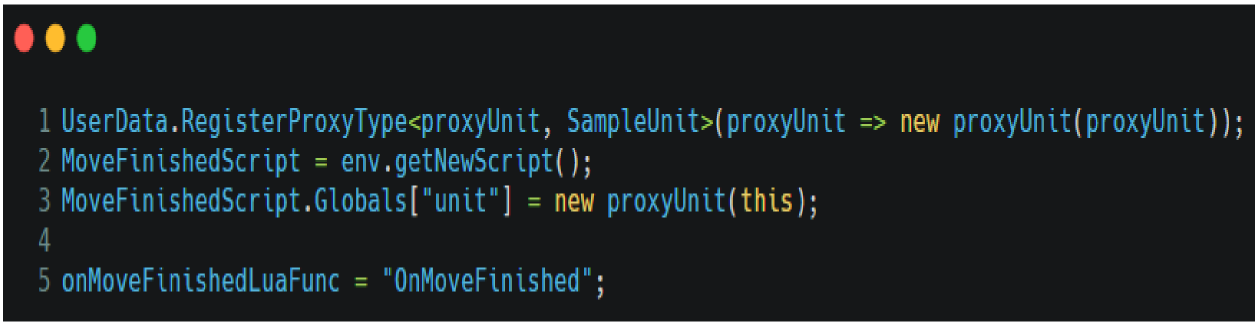
The custom class “SampleUnit” inherits the “OnMoveFinished” event from “TBS Framework Unit”. “SampleUnit” also aggregates the “proxyUnit” class. This is used as a placeholder for unit for the Lua file.

The sequence diagram below shows the high-level data flow.



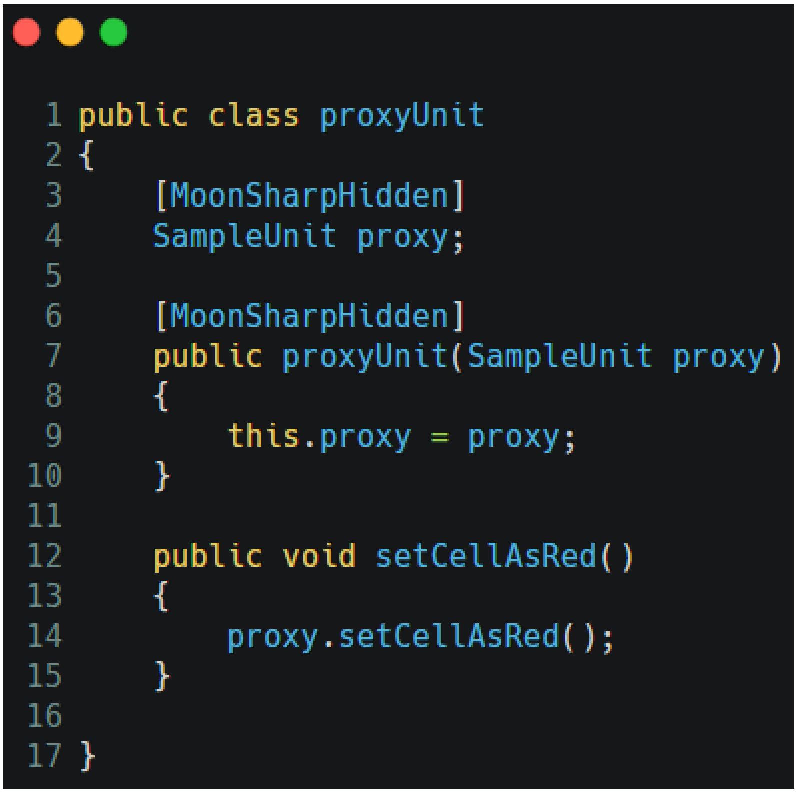
C# code behind OnMoveFinished

The code below shows how Moonsharp is configured before the “OnMoveFinished” event is called:



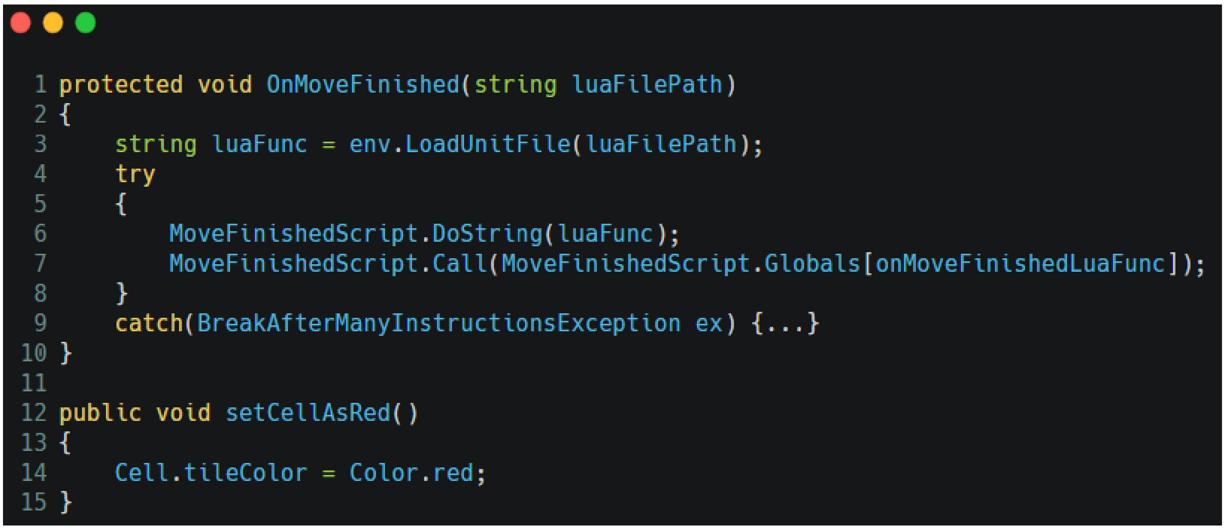
```
1 UserData.RegisterProxyType<proxyUnit, SampleUnit>(proxyUnit => new proxyUnit(proxyUnit));
2 MoveFinishedScript = env.getNewScript();
3 MoveFinishedScript.Globals["unit"] = new proxyUnit(this);
4
5 onMoveFinishedLuaFunc = "OnMoveFinished";
```

The first line shows that a proxy type is registered. This enables all Moonsharp Script objects to recognize this type. The proxy type is set as the keyword “unit” in the Lua code on line 3. Since a proxy type is exposed to Lua then that means that the Lua code only interacts with the proxy instead of the real “SampleUnit” type.



```
public class proxyUnit
{
    [MoonSharpHidden]
    SampleUnit proxy;

    [MoonSharpHidden]
    public proxyUnit(SampleUnit proxy)
    {
        this.proxy = proxy;
    }

    public void setCellAsRed()
    {
        proxy.setCellAsRed();
    }

}
```

This class has a reference to the real “SampleUnit” file. This makes sure that when the modders write “unit.setCellAsRed()” then the same function is called on the real “SampleUnit” type. In addition, the “[MoonSharpHidden]” attributes make sure that the reference and the constructor is unavailable to the Lua script. This is important to enforce that only the proxy type is used and not the actual “SampleUnit” type in the Lua scripts.



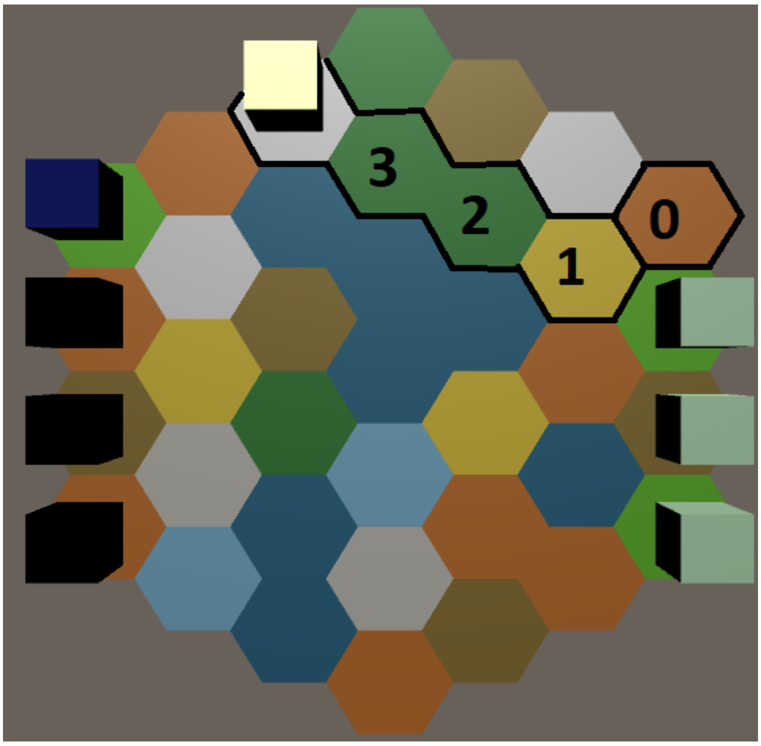
```
 1 protected void OnMoveFinished(string luaFilePath)
 2 {
 3     string luaFunc = env.LoadUnitFile(luaFilePath);
 4     try
 5     {
 6         MoveFinishedScript.DoString(luaFunc);
 7         MoveFinishedScript.Call(MoveFinishedScript.Globals[onMoveFinishedLuaFunc]);
 8     }
 9     catch(BreakAfterManyInstructionsException ex) {...}
10 }
11
12 public void setCellAsRed()
13 {
14     Cell.tileColor = Color.red;
15 }
```

Firstly, the “luaFunc” variable is reassigned with Lua code every time the “OnMoveFinished” event occurs on line 3. This is to load in new changes during runtime which is one of the functional requirements. Lastly, the “MoveFinishedScript” calls the “OnMoveFinished” function from “luaFunc”.

The function definition used in the Lua file “setCellAsRed()” is found at the end of the code snippet. Finally, the code tests if an exception occurs. This is related to a non-functional requirement which will be explained in the testing section.

Considered implementation

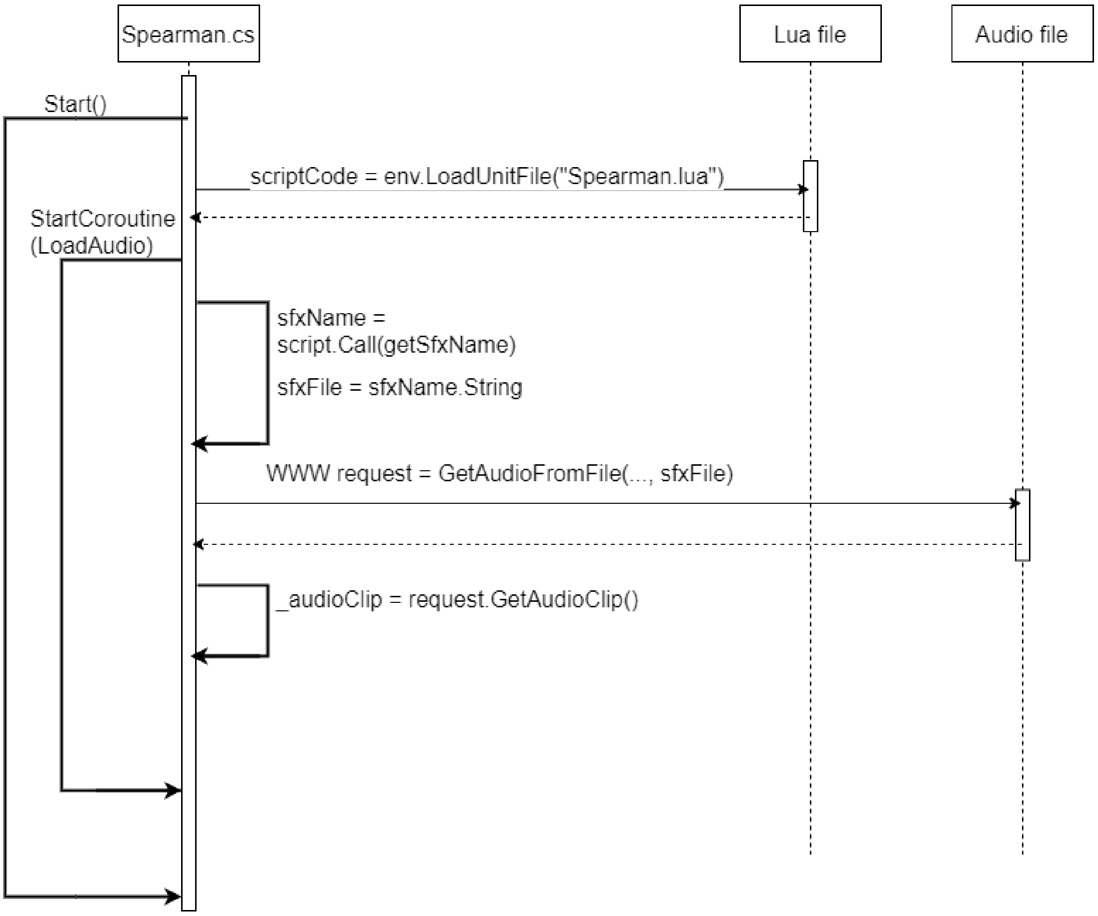
The considered implementation was about giving the Lua modders the opportunity to modify all the tiles in the path which the unit travels. This can be explained using this picture:



Here the unit travels from tile 0 to the tile where it is currently standing. The idea is to expose a list of tiles from tile 1 to the destination tile to Lua. A possible modding idea is to create an ice unit. This unit could turn every tile in its path into ice tiles. These ice tiles could give a small damage to players which were standing on them.

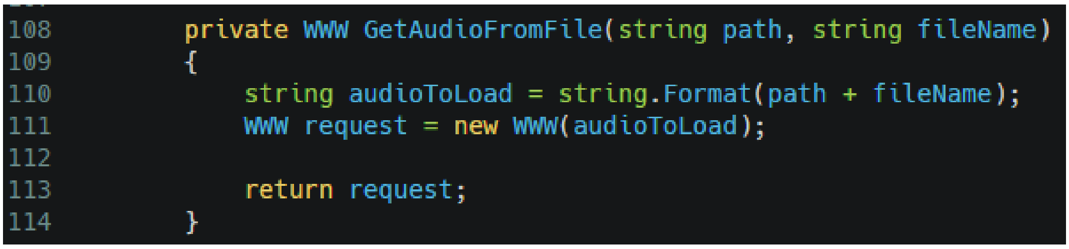
Audio

The sequence diagram below gives a high-level overview of the code execution related to unit audio.



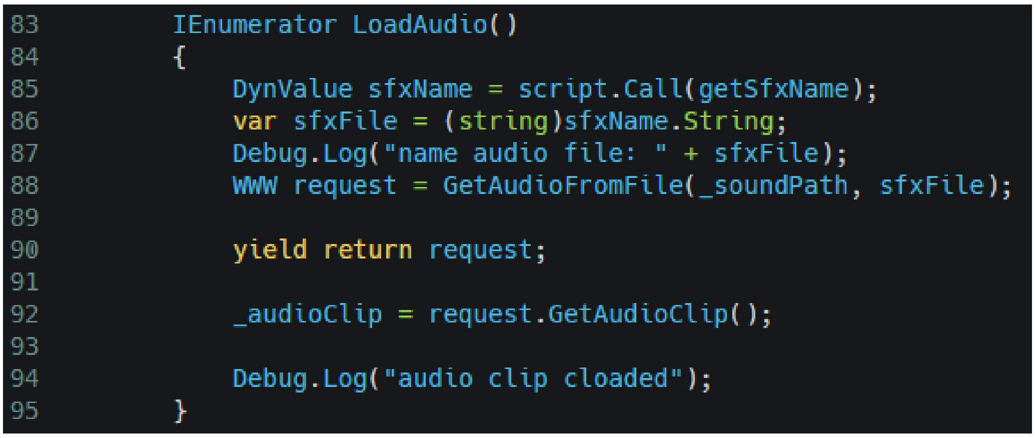
Spearman.cs
Lua file
Audio file
Start()
scriptCode = env.LoadUnitFile("Spearman.lua")
StartCoroutine
(LoadAudio)
sfxName =
script.Call(getSfxName)
sfxFile = sfxName.String
WWW request = GetAudioFromFile(..., sfxFile)
_audioClip = request.GetAudioClip()

To make loading audio files via the streaming assets folder possible, our best options were to use the WWW or UnityWebRequest libraries offered by Unity. These libraries essentially access online resources via a URL-path, but also offers loading files locally via local file paths, which is exactly what we wanted to achieve. Although WWW is considered “obsolete” with UnityWebRequest being its replacement *[22]*, most of the tutorials published by various users offered on this scenario utilized WWW. Since the two libraries would accomplish the same for our project, we decided to rely on WWW as it was easier to find examples online fitting what we wanted to accomplish.



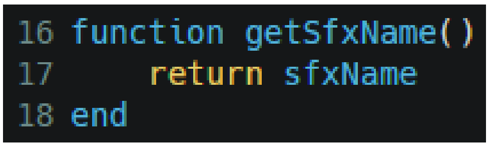
```
        private WWW GetAudioFromFile(string path, string fileName)
        {
            string audioToLoad = string.Format(path + fileName);
            WWW request = new WWW(audioToLoad);

            return request;
        }
```

First the program must locate the file that the user wants to load through a function. We made the function return the location and name of the file as a WWW object in order to later manipulate it in the next function so that the audio from the object gets extracted.



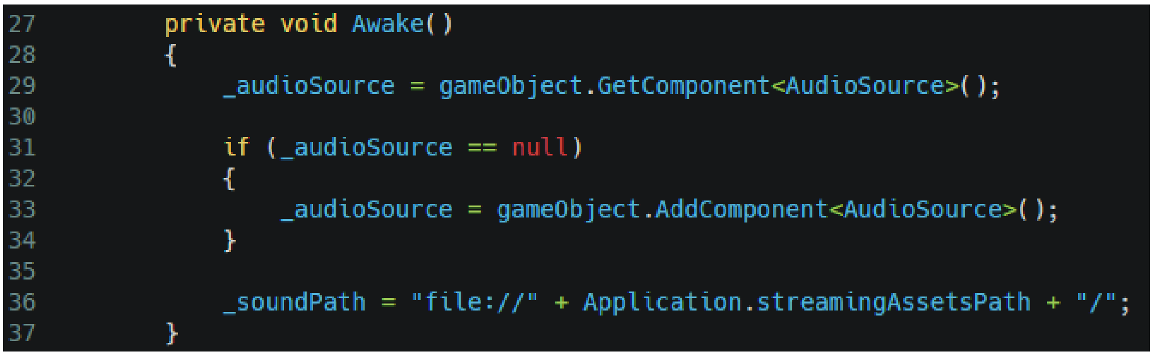
```
        IEnumerator LoadAudio()
        {
            DynValue sfxName = script.Call(getSfxName);
            var sfxFile = (string)sfxName.String;
            Debug.Log("name audio file: " + sfxFile);
            WWW request = GetAudioFromFile(_soundPath, sfxFile);

            yield return request;

            _audioClip = request.GetAudioClip();

            Debug.Log("audio clip cloaded");
        }
```

The above function’s task is to load the audio file from computer and adding it to the unit in the game logic when the game starts. The Lua function “getSfxName” is called which returns the value of the audio file name variable defined by the user in the Lua script (pictured below) and store it in a variable for use in the next step.



```
function getSfxName()
    return sfxName
end
```

The function then creates a WWW object (“request”) which will store the value returned by “GetAudioFromFile” whose “_soundPath” (which is the file path) is defined in the Awake() function of the unit, and “sfxFile” being the value of the variable from sfxName.

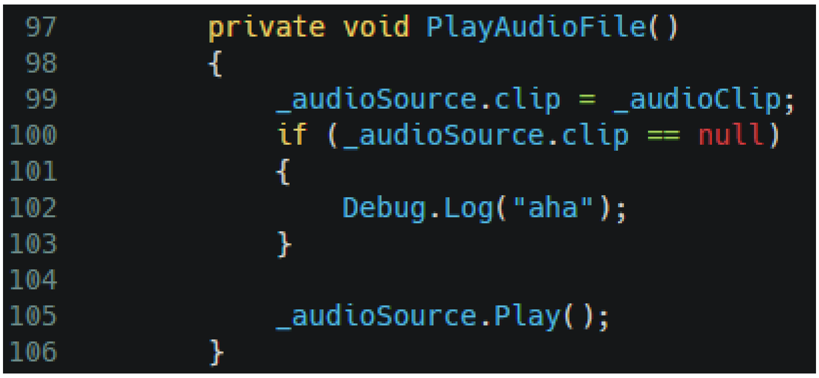


```
        private void Awake()
        {
            _audioSource = gameObject.GetComponent<AudioSource>();

            if (_audioSource == null)
            {
                _audioSource = gameObject.AddComponent<AudioSource>();
            }

            _soundPath = "file://" + Application.streamingAssetsPath + "/";
        }
```

Various attempts were made to make this function run properly during runtime, and through trial and error we found out that simply making LoadAudio() a void function led to unexpected behaviour in the game. Not all audio files would load properly (or at all), and sometimes the audio clips would sometimes play randomly at game start-up or closing. This turned out to be the fault of WWW running asynchronous to the game logic, leading to the audio file not having enough time to load in time, leading to these unexpected behaviours *[23]*.

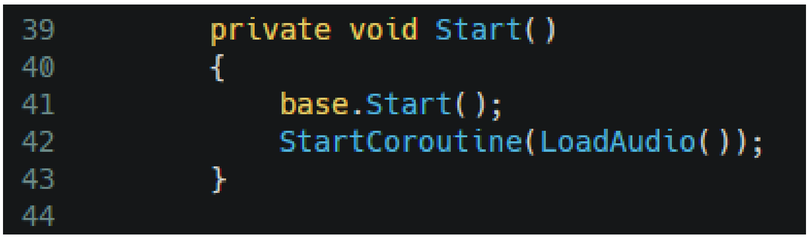
That is why we turned the LoadAudio() function into an IEnumerator which let us use the “yield return” tool. This tool let us inform Unity that the WWW request must go through and finish before setting the audio file, giving the function enough time to fully load the file, thus providing the result we wanted.

“_audioClip” is a variable of type AudioClip which is where the audio file’s data is stored in the unit’s AudioClip component. Because we have the audio file’s data stored in the WWW object “request”, this allowed us to call a WWW-specific function “GetAudioClip” which returns an audio clip that can be stored in an AudioClip object.



```
        private void PlayAudioFile()
        {
            _audioSource.clip = _audioClip;
            if (_audioSource.clip == null)
            {
                Debug.Log("aha");
            }

            _audioSource.Play();
        }
```

We also created a function that simply plays the audio when called (“PlayAudioFile”). It adds the audio clip into the AudioSource component of the unit by storing the data into the variable “_audioSource” which is an object of the type of the same name. This makes the audio playback more organized and open for editing if there needs to be a change in for example variables.



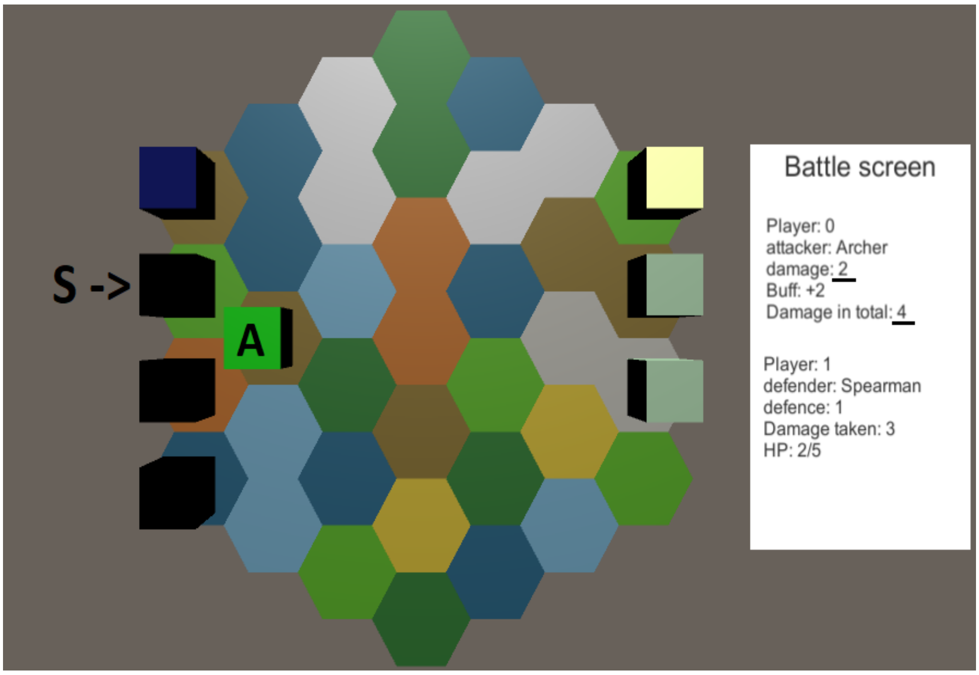
```
        private void Start()
        {
            base.Start();
            StartCoroutine(LoadAudio());
        }

```

The coroutine is simply started when the game begins so that the audio file(s) load on startup and are ready for playback whenever/wherever PlayAudioFile() is called in the code.

Combat

This section will be structured with first showcasing combat without any modifications. Then, combat will be shown including modifications. Lastly, the C# code which loads in the Lua code and how the game interacts with TBS Framework, will be shown.

Combat without modding



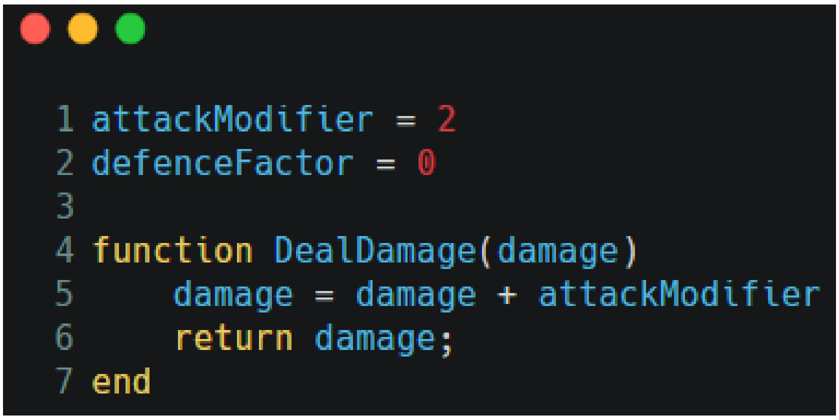
Battle screen

Player: 0
attacker: Archer
damage: 2
Buff: +2
Damage in total: 4

Player: 1
defender: Spearman
defence: 1
Damage taken: 3
HP: 2/5

The unit marked with “A” is a unit of type archer and the unit with an “S” is of type Spearman. An archer attacks a spearman unit with 2 as its damage. The damage in total is 4 since archer deals double damage against spearman.

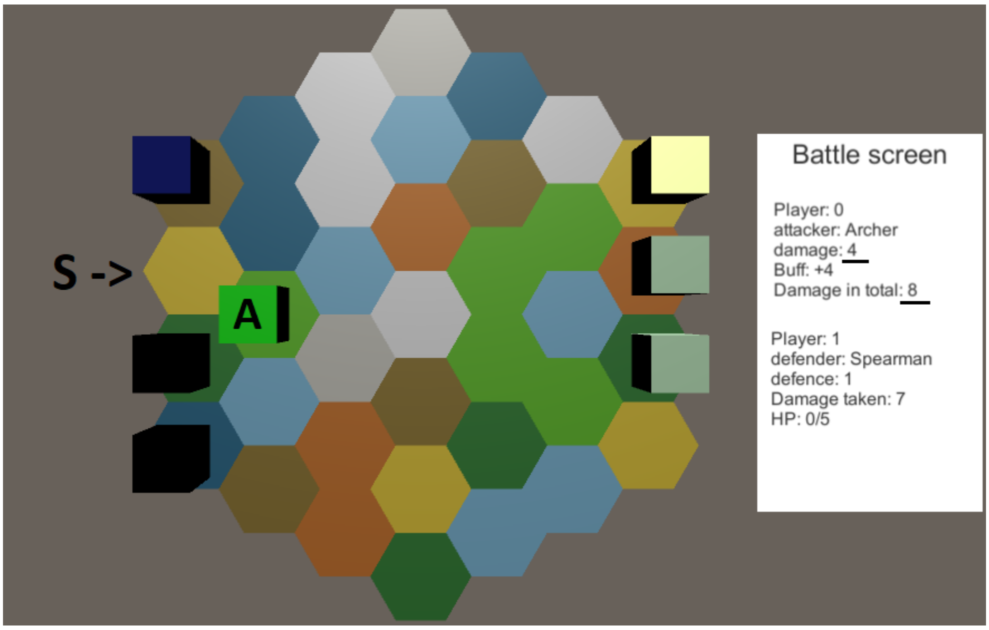
The modder can edit the “Archer.lua” file if they want to increase the damage attack. The damage calculation inside the “Archer.lua” in the mod folder looks like this:



```
attackModifier = 2
defenceFactor = 0

function DealDamage(damage)
    damage = damage + attackModifier
    return damage;
end
```

Combat with modding

The next picture shows the effect of this simple modification.



Battle screen

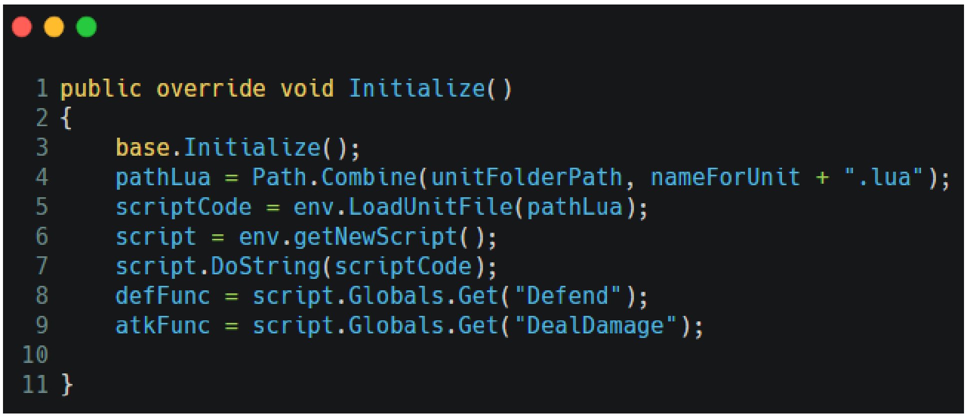
Player: 0
attacker: Archer
damage: 4
Buff: +4
Damage in total: 8

Player: 1
defender: Spearman
defence: 1
Damage taken: 7
HP: 0/5

Now the attack is 4 instead of 2 and the total damage becomes 8. The spearman unit is destroyed because of the damage increase.

C# code behind the combat

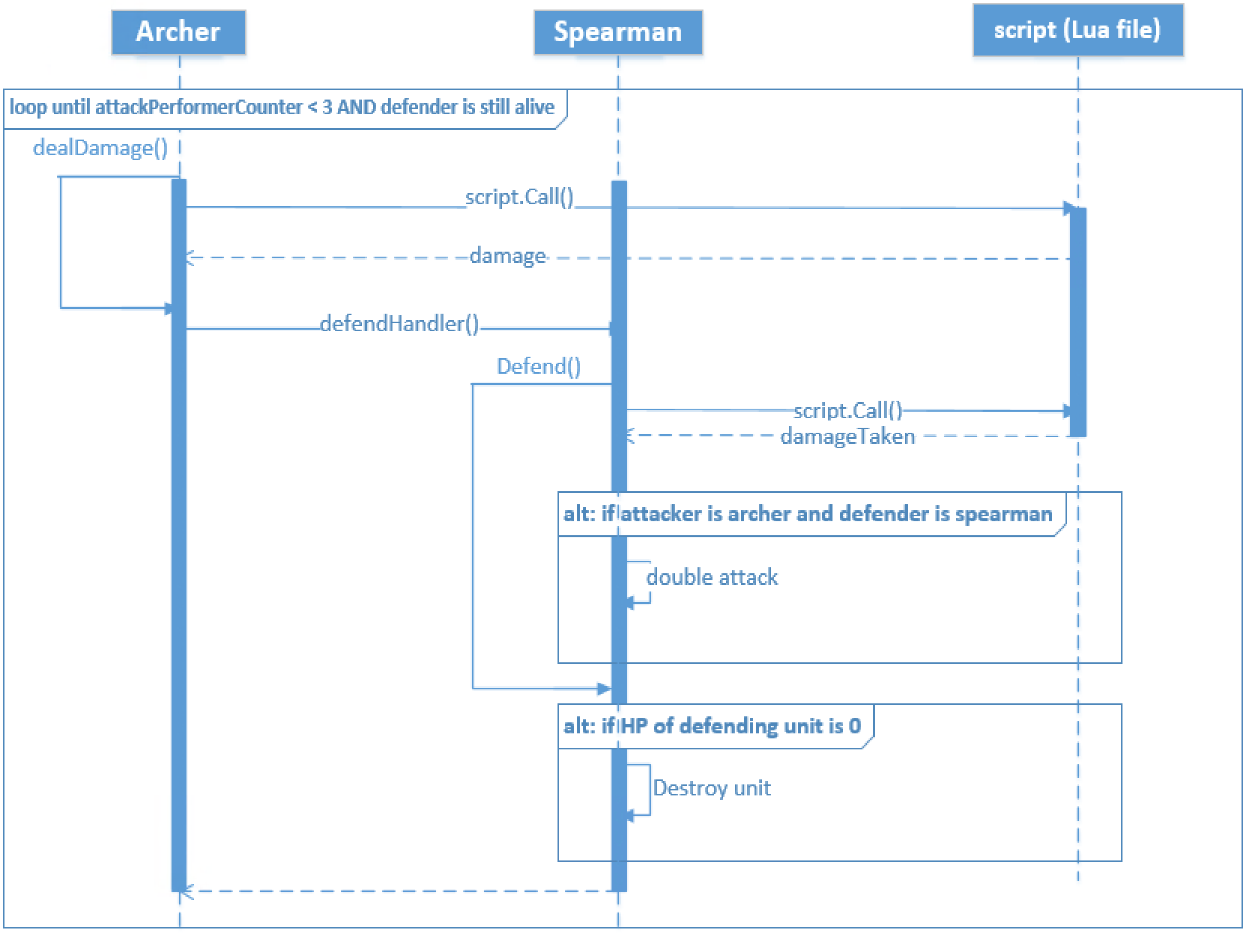
Moonsharp needs to be configured before the combat begins. The configuration includes creating the necessary Moonsharp variables and fetching code from a Lua file. Below shows an “Initialize()” function which configures the Lua file for the archer to be used in-game.

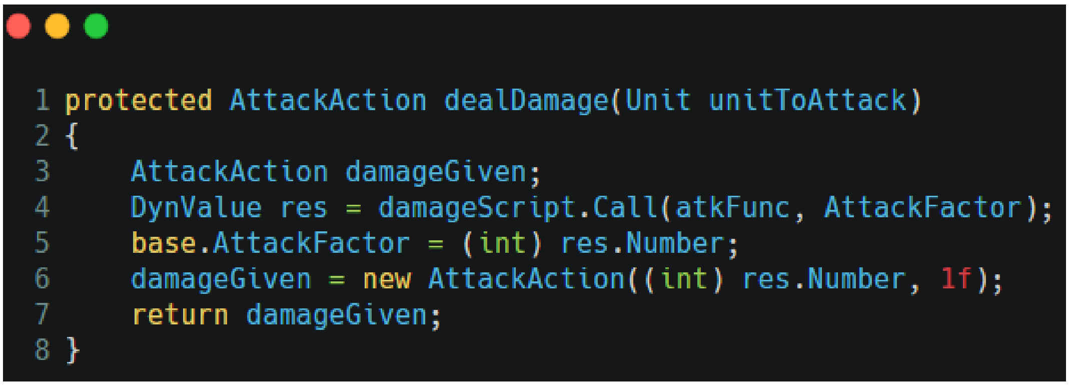


```
 1 public override void Initialize()
 2 {
 3     base.Initialize();
 4     pathLua = Path.Combine(unitFolderPath, nameForUnit + ".lua");
 5     scriptCode = env.LoadUnitFile(pathLua);
 6     script = env.getNewScript();
 7     script.DoString(scriptCode);
 8     defFunc = script.Globals.Get("Defend");
 9     atkFunc = script.Globals.Get("DealDamage");
10
11 }
```

The Lua code is retrieved from the Archer Lua file inside the mod folder on line 5. Then **defFunc** is loaded with the “Defend()” function from the Lua file.

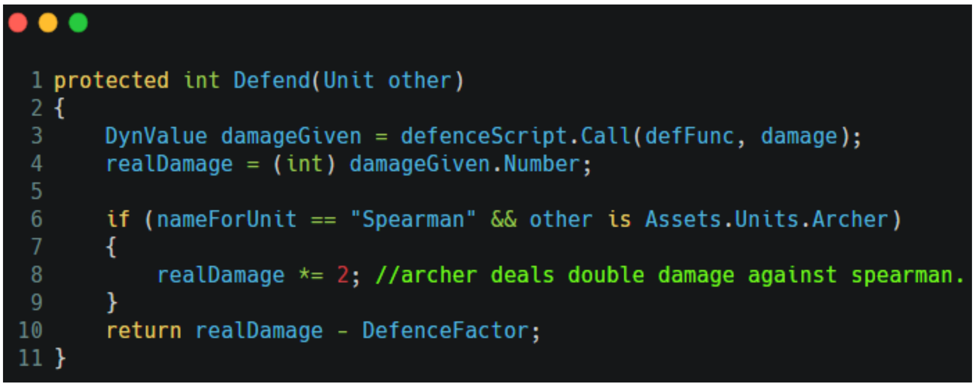
Below is a sequence diagram which shows the high-level code execution when combat begins.


Archer
Spearman
script (Lua file)
loop until attackPerformerCounter < 3 AND defender is still alive
dealDamage()
script.Call()
damage
defendHandler()
Defend()
script.Call()
damageTaken
alt: if attacker is archer and defender is spearman
double attack
alt: if HP of defending unit is 0
Destroy unit



```
1 protected AttackAction dealDamage(Unit unitToAttack)
2 {
3     AttackAction damageGiven;
4     DynValue res = damageScript.Call(atkFunc, AttackFactor);
5     base.AttackFactor = (int) res.Number;
6     damageGiven = new AttackAction((int) res.Number, 1f);
7     return damageGiven;
8 }
```

On line 4 the DealDamage() function in the unit Lua file is called by passing in the **atkFunc** and **AttackFactor** which is the base damage the unit can do to a unit. The Call() function will return the modified damage after the **AttackFactor** gets modified by the **attackModifier** that exists in the unit Lua file.



```
 1 protected int Defend(Unit other)
 2 {
 3     DynValue damageGiven = defenceScript.Call(defFunc, damage);
 4     realDamage = (int) damageGiven.Number;
 5
 6     if (nameForUnit == "Spearman" && other is Assets.Units.Archer)
 7     {
 8         realDamage *= 2; //archer deals double damage against spearman.
 9     }
10     return realDamage - DefenceFactor;
11 }
```

The “Defend()” function works in a very similar way like “DealDamage()” works. Again, a number is retrieved from the Lua function by passing the **defFunc** variable to “script.Call()” and the initial damage. The damage then gets reduced by the units’ DefenceFactor in the unit Lua file and it gets reduced again by the DefenceFactor for the unit on line 10.

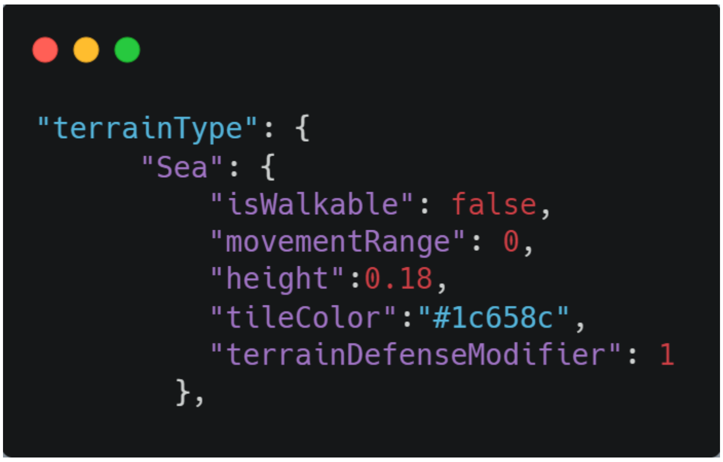
Terrain Generation

Early on Progress stated that they wanted a terrain system that would be moddable. Using procedurally generated heightmaps with Perlin Noise, we were able to create a system that is both editable to achieve what the user wants but also always generates a playable and fun environment.

Perlin Noise

In 1983 Ken Perlin invented a function to generate procedural textures for computer-generated effects *[24]*. This Perlin Noise is very appealing because it creates a very smooth sequence of pseudo-random values, which is perfect when you think of effects that require very natural gradual qualities and transitions – like terrain.

As shown in the figure above, Perlin Noise is represented as just a map of pseudo-random black and white gradients. At every white and black gradient, a pseudo-random Perlin-value is assigned. As we are using this for terrain, we have chosen to call it the ‘height’.
Exactly how we used this map to generate dynamic terrain is explained below:

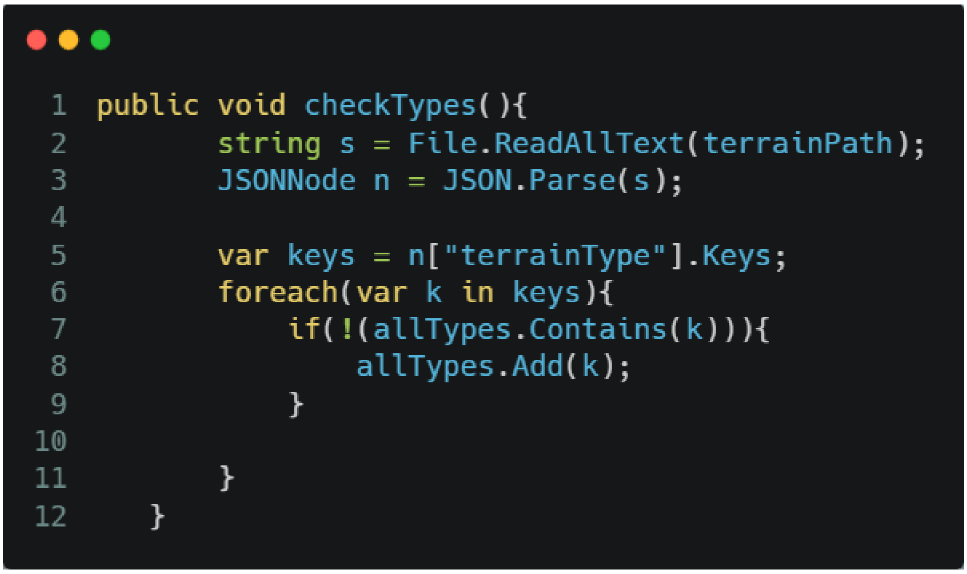


```
"terrainType": {
      "Sea": {
          "isWalkable": false,
          "movementRange": 0,
          "height":0.18,
          "tileColor":"#1c658c",
          "terrainDefenseModifier": 1
        },
```

To make sure that students can modify the terrain, but also understand exactly what they are modifying, the different types of terrain have their own attributes contained in a .json file as seen above.

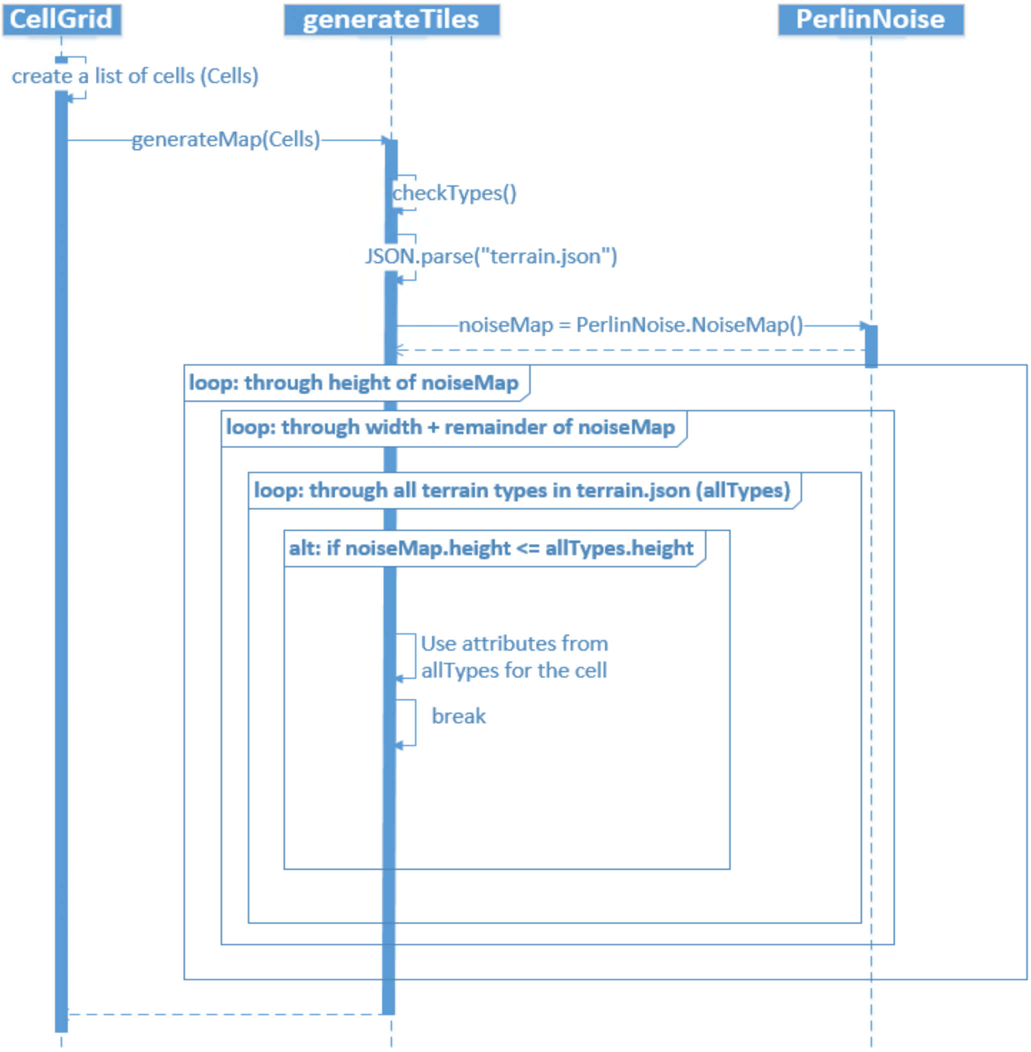
The students can add their own terrain-type, and assign different attributes:

- IsWalkable: Can you walk across this tile?
- MovementRange: When standing on this tile, how far can you go further?
- Height: Controls the procedural generation, which is explained below.
- TileColor: Hexadecimal value for the color that this tile should have.
- TerrainDefenseModifier: How much strategic defensiveness does this tile add to combat?

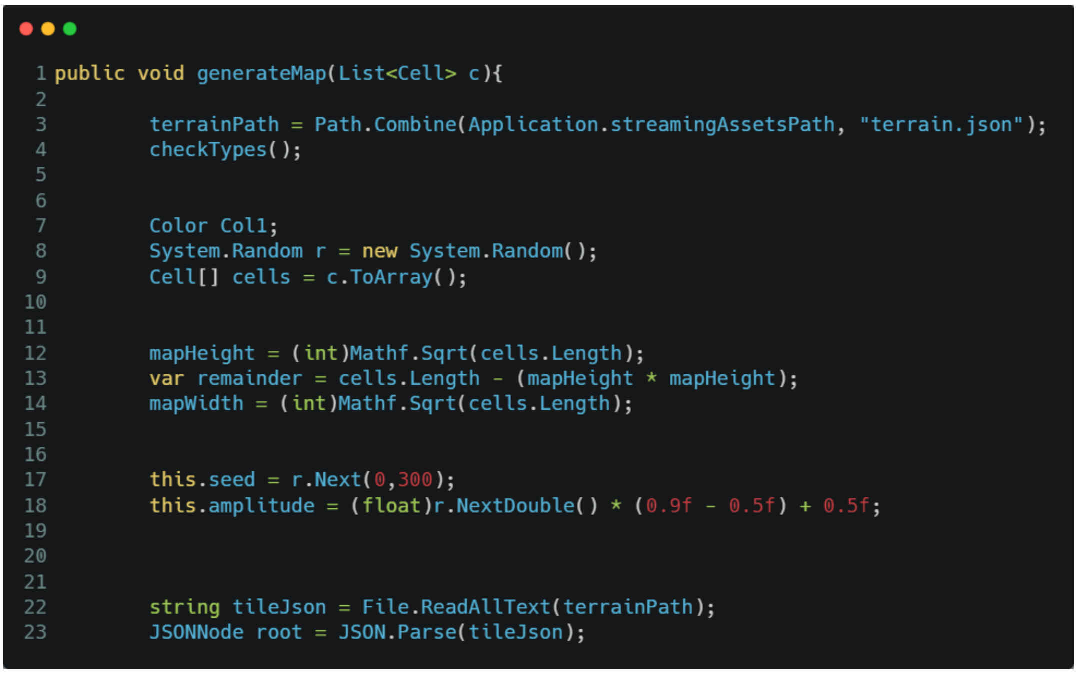


```
public void checkTypes(){
       string s = File.ReadAllText(terrainPath);
       JSONNode n = JSON.Parse(s);

       var keys = n["terrainType"].Keys;
       foreach(var k in keys){
           if(!(allTypes.Contains(k))){
               allTypes.Add(k);
           }

       }
   }
```

There is no set max-number of types when it comes to terrain. Therefore, checkTypes() runs on every game start-up, loops through every unique type of terrain and adds it to the allTypes list.


CellGrid
generateTiles
PerlinNoise
create a list of cells (Cells)
generateMap(Cells)
checkTypes()
JSON.parse("terrain.json")
noiseMap = PerlinNoise.NoiseMap()
loop: through height of noiseMap
loop: through width + remainder of noiseMap
loop: through all terrain types in terrain.json (allTypes)
alt: if noiseMap.height <= allTypes.height
Use attributes from allTypes for the cell
break

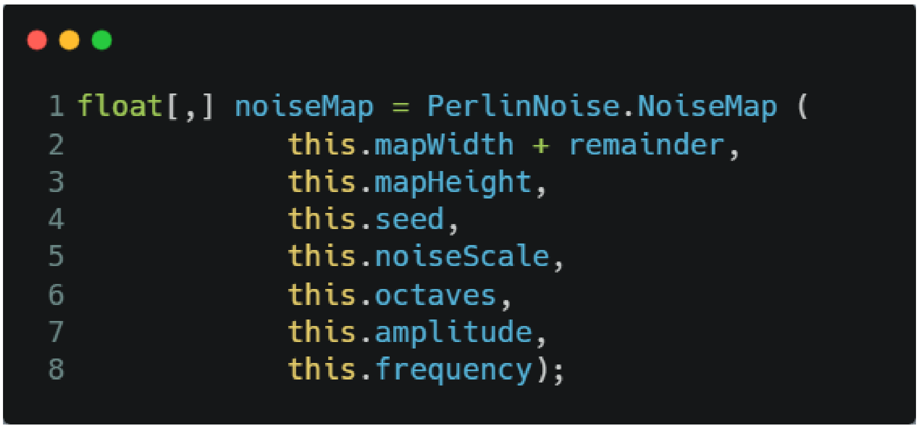
The Sequence Diagram shows the process in which the in-game tiles are made from a CellGrid after being assigned Perlin values and a Cell-type



```
public void generateMap(List<Cell> c){

        terrainPath = Path.Combine(Application.streamingAssetsPath, "terrain.json");
        checkTypes();


        Color Col1;
        System.Random r = new System.Random();
        Cell[] cells = c.ToArray();


        mapHeight = (int)Mathf.Sqrt(cells.Length);
        var remainder = cells.Length - (mapHeight * mapHeight);
        mapWidth = (int)Mathf.Sqrt(cells.Length);


        this.seed = r.Next(0,300);
        this.amplitude = (float)r.NextDouble() * (0.9f - 0.5f) + 0.5f;



        string tileJson = File.ReadAllText(terrainPath);
        JSONNode root = JSON.Parse(tileJson);
```

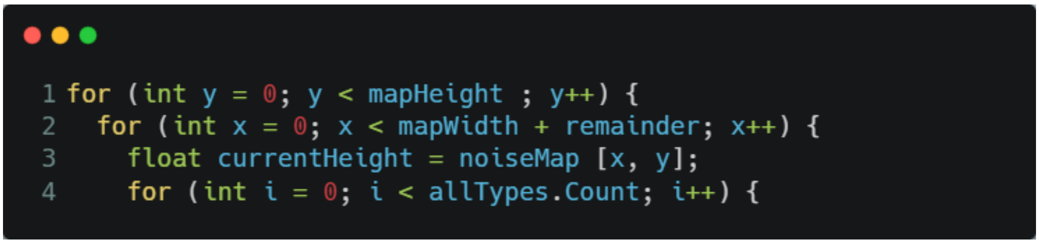
The list of all initiated Cells is sent to generateMap() before these Cells are transformed into the ‘Hexagon’-type tiles visible in-game. On our part this is a way of adapting to the order that TBS create useable tiles:
As mentioned in the ‘Technologies’ section, in TBS a Cell is being assigned as a Hexagon-type, it is then created with all the attributes it inherits from itself as a Cell – This means that by the time it is being set as a Hexagon all attributes must already be assigned.

Perlin Noise is generated as a square image of noise. It uses a height and width value to loop through itself. Accordingly, to properly project the Perlin-values on-to our tiles, the dimensions of the noise map should be the same as a list, array, or grid of our tiles.
Unfortunately, only Hexagon-type tiles are assigned to the TBS Grid-class containing all in-game tiles. Meaning that in this instance it is too early to access the width() and height() of the grid. A workaround is using the square root of the number of cells – and then taking the square root remainder into account later.
An example explaining how the ‘remainder’ variable works when generating:
Our default set of tiles has a 6x7 size. Because the Perlin map is being generated as a perfect square – which would be 6x6 – the remaining set of tiles would have to be considered, otherwise they would not be painted at all.



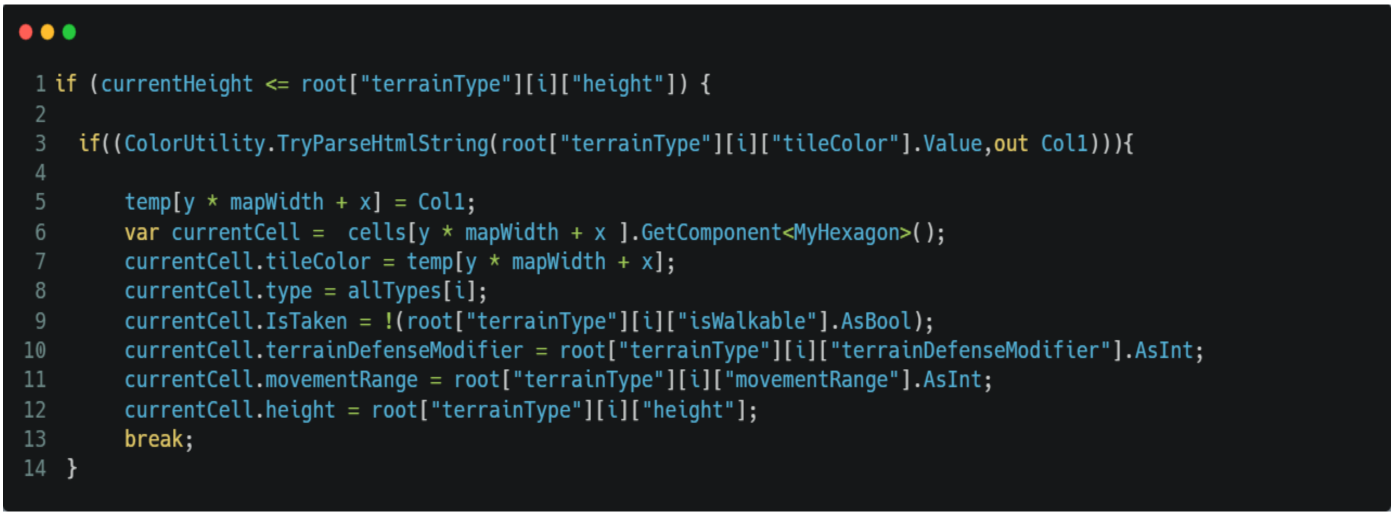
```
float[,] noiseMap = PerlinNoise.NoiseMap (
            this.mapWidth + remainder,
            this.mapHeight,
            this.seed,
            this.noiseScale,
            this.octaves,
            this.amplitude,
            this.frequency);
```

A noise map of values, like the figure shown above in ‘Perlin Noise’, is then generated to be used for height-comparison *[25] [26] [27]*. The code from the previous figure is applied to make sure that the noise map being generated is exactly the size of the number of cells we are trying to paint terrain on. The other values such as amplitude, frequency, octaves and noiseScale are all values to play around with to get a desired type of noise.
We have decided to add slight randomization for the seed and amplitude, so the map has some variation between games.



```
for (int y = 0; y < mapHeight ; y++) {
  for (int x = 0; x < mapWidth + remainder; x++) {
    float currentHeight = noiseMap [x, y];
    for (int i = 0; i < allTypes.Count; i++) {
```

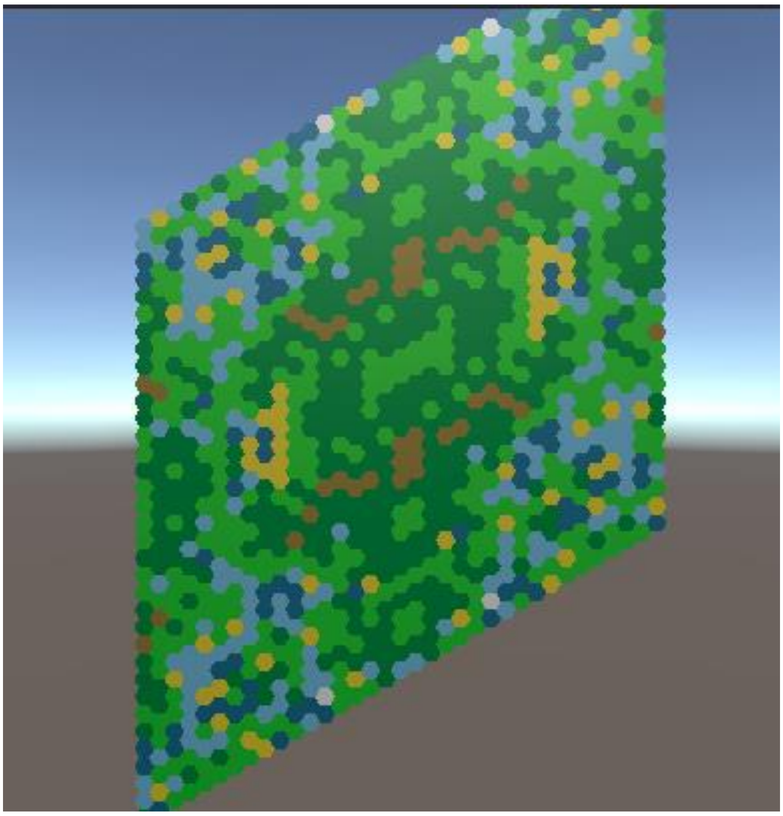
We are then looping through every height and width in the noise map and comparing them to our own specified terrains in our .json file.



```
if (currentHeight <= root["terrainType"][i]["height"]) {

  if((ColorUtility.TryParseHtmlString(root["terrainType"][i]["tileColor"].Value,out Col1))){

      temp[y * mapWidth + x] = Col1;
      var currentCell =  cells[y * mapWidth + x ].GetComponent<MyHexagon>();
      currentCell.tileColor = temp[y * mapWidth + x];
      currentCell.type = allTypes[i];
      currentCell.IsTaken = !(root["terrainType"][i]["isWalkable"].AsBool);
      currentCell.terrainDefenseModifier = root["terrainType"][i]["terrainDefenseModifier"].AsInt;
      currentCell.movementRange = root["terrainType"][i]["movementRange"].AsInt;
      currentCell.height = root["terrainType"][i]["height"];
      break;
 }
```

It all comes together in the first line of the figure above as it is considering both the procedural generation of Perlin Noise and attributes specified by modders. If the ‘height’ Perlin-value in the **current location on the noise map** is less than the height of a type of **terrain from terrain.json specified by a student**, the tile in this specific location is assigned the attributes and colour of the correlating terrain.json-type.

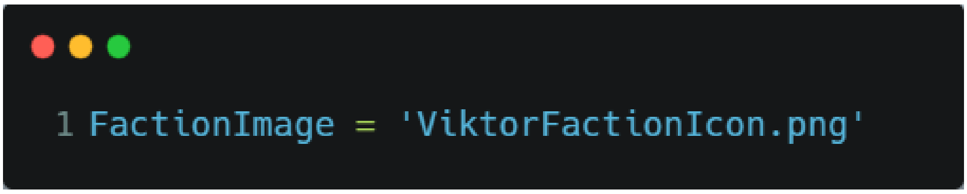
The result, if the heights are properly assigned in terrain.json, will be a procedurally painted terrain with smooth transitions between types like seas, rivers, beaches, and forests for instance.



This specific figure shows a 35x35 grid of tiles, and unfortunately Perlin Noise works better with large noise maps – So on smaller maps like the 6x7 shown earlier in the document the results might not look as delicate. Think of this effect as a zoom-feature on a camera. The more zoomed-in you are, the less clear the image.

Changing the faction icon image to a custom made one

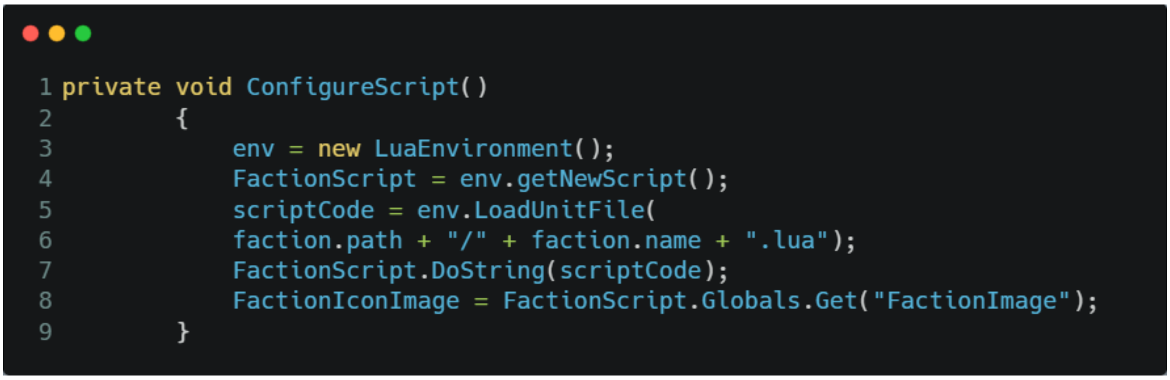
A modification you can do to the faction is changing the faction icon to an image of your choice, it can be an image with a logo you created or something you found online. The modification is done through the Faction Lua file.



```lua
FactionImage = 'ViktorFactionIcon.png'
```

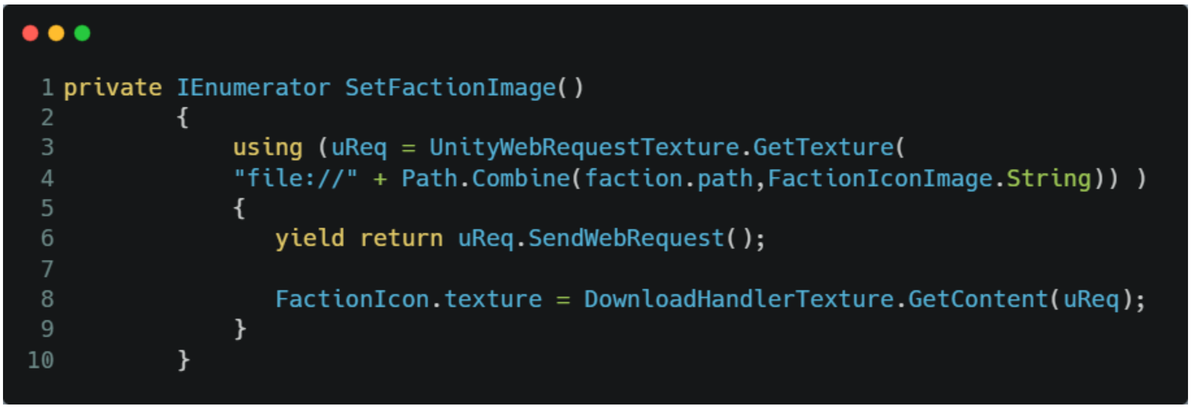
The modification is simple. You must define a variable with the name ‘FactionImage’ and assign it the name of the image, and the image must exist within the faction folder.

Then in the HumanPlayer script the image name is fetched from the Lua file on line 8.



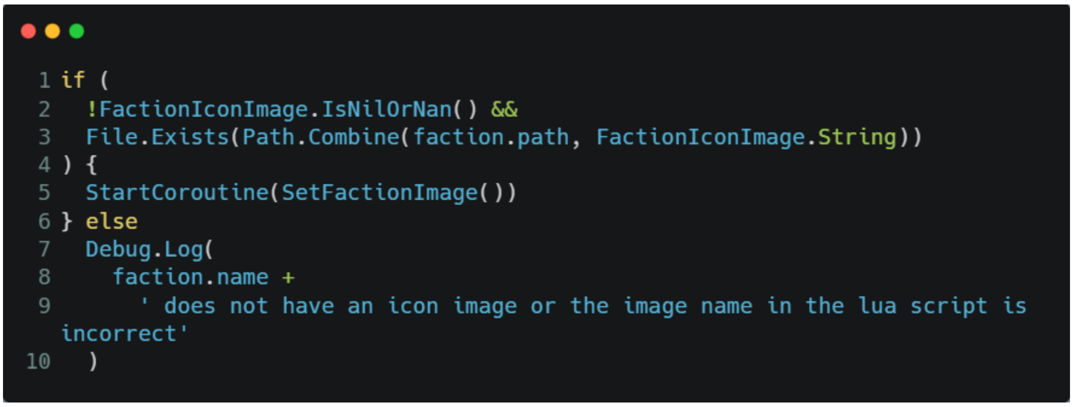
```
1 private void ConfigureScript()
2         {
3             env = new LuaEnvironment();
4             FactionScript = env.getNewScript();
5             scriptCode = env.LoadUnitFile(
6             faction.path + "/" + faction.name + ".lua");
7             FactionScript.DoString(scriptCode);
8             FactionIconImage = FactionScript.Globals.Get("FactionImage");
9         }
```

Then it gets the image from the faction folder and assigns the FactionIcon object in the Unity Scene to it on line 8.



```
 1 private IEnumerator SetFactionImage()
 2         {
 3             using (uReq = UnityWebRequestTexture.GetTexture(
 4             "file://" + Path.Combine(faction.path,FactionIconImage.String)) )
 5             {
 6                yield return uReq.SendWebRequest();
 7 
 8                FactionIcon.texture = DownloadHandlerTexture.GetContent(uReq);
 9             }
10         }
```

There is an if statement checking if the ‘FactionImage’ variable exists in the Lua file and if the faction image itself exists in the faction folder.

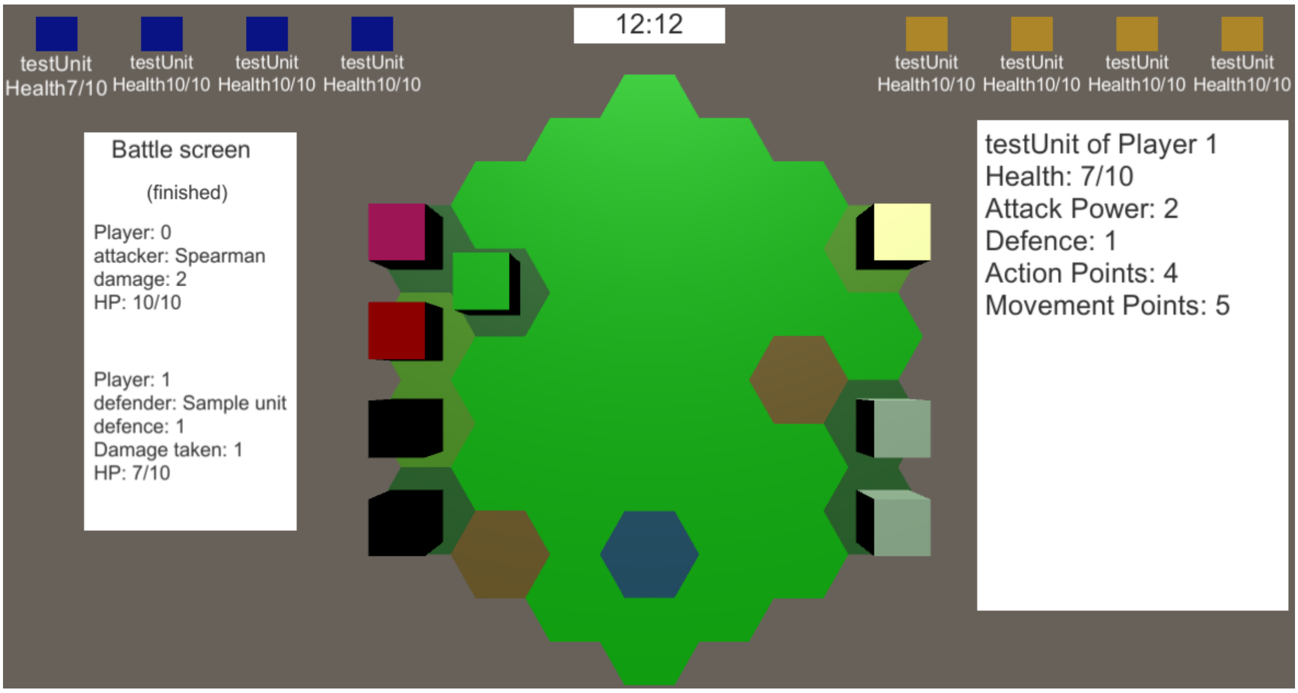


```
 1 if (
 2   !FactionIconImage.IsNilOrNan() &&
 3   File.Exists(Path.Combine(faction.path, FactionIconImage.String))
 4 ) {
 5   StartCoroutine(SetFactionImage())
 6 } else
 7   Debug.Log(
 8     faction.name +
 9       ' does not have an icon image or the image name in the lua script is
incorrect'
10   )
```

Non-modifiable game features

The features that were non-modifiable were UI and real-time movement. The latter is implemented to give the prototype game a more real-time feel. This is done to make the prototype similar to what Progress' upcoming game will look like.

The UIs were added to communicate more clearly what happens in the game to the end users. This is also helpful for modders to see the new effects in action.



12:12

testUnit Health7/10
testUnit Health10/10
testUnit Health10/10
testUnit Health10/10

testUnit Health10/10
testUnit Health10/10
testUnit Health10/10
testUnit Health10/10

Battle screen

(finished)

Player: 0
attacker: Spearman
damage: 2
HP: 10/10

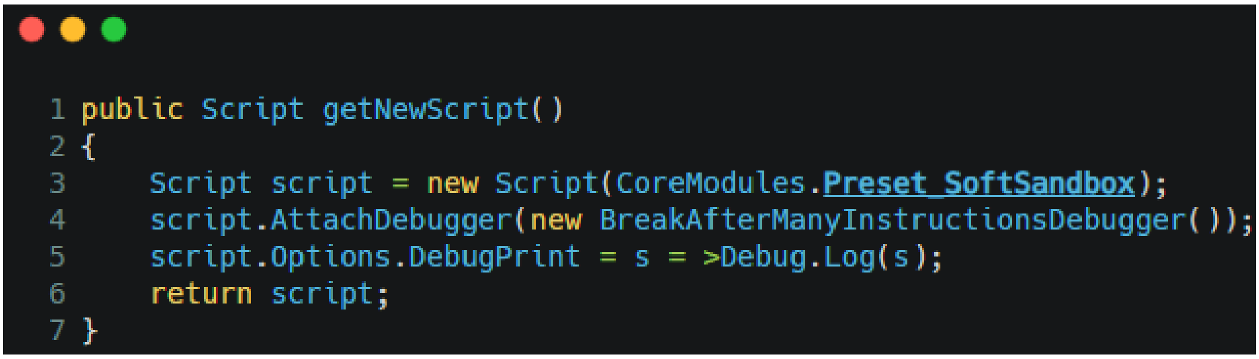
Player: 1
defender: Sample unit
defence: 1
Damage taken: 1
HP: 7/10

testUnit of Player 1
Health: 7/10
Attack Power: 2
Defence: 1
Action Points: 4
Movement Points: 5

Picture of battle screen and UI containing info about units on the right side and at the top.

Security

We have followed our security requirement related to restricting the modder to read, write or delete files and restricting them from executing operating system shell commands. We did this by configuring a "Soft Sandbox" through Moonsharp *[28]*. This sandbox prevents Lua files from accessing the "io" and "os" library. These libraries are used to read and write files on disk and execute operating system shell commands *[29]*. The soft sandbox is configured with the "getNewScript()" function in the "LuaEnvironment" class when a Script object is created on line 3.



```
1 public Script getNewScript()
2 {
3     Script script = new Script(CoreModules.Preset_SoftSandbox);
4     script.AttachDebugger(new BreakAfterManyInstructionsDebugger());
5     script.Options.DebugPrint = s = >Debug.Log(s);
6     return script;
7 }
```

This function is used to create every Script object in the prototype game to ensure that all Lua files have the same level of security.

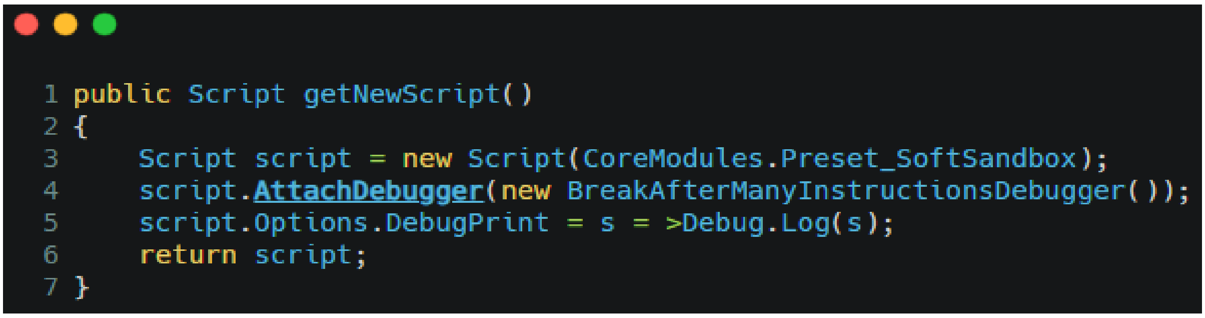
Testing

Verifying non-functional requirements

Fps requirement for Lua file execution

In the non-functional requirement we have written that a Lua file when executed shall not decrease the frame rate below 40 fps. To follow this requirement, we configured Moonsharp to count the number of instructions a Lua file executes. This can stop the Lua execution if a student has written code with too many instructions. An example of code with too many instructions is an infinite running loop.

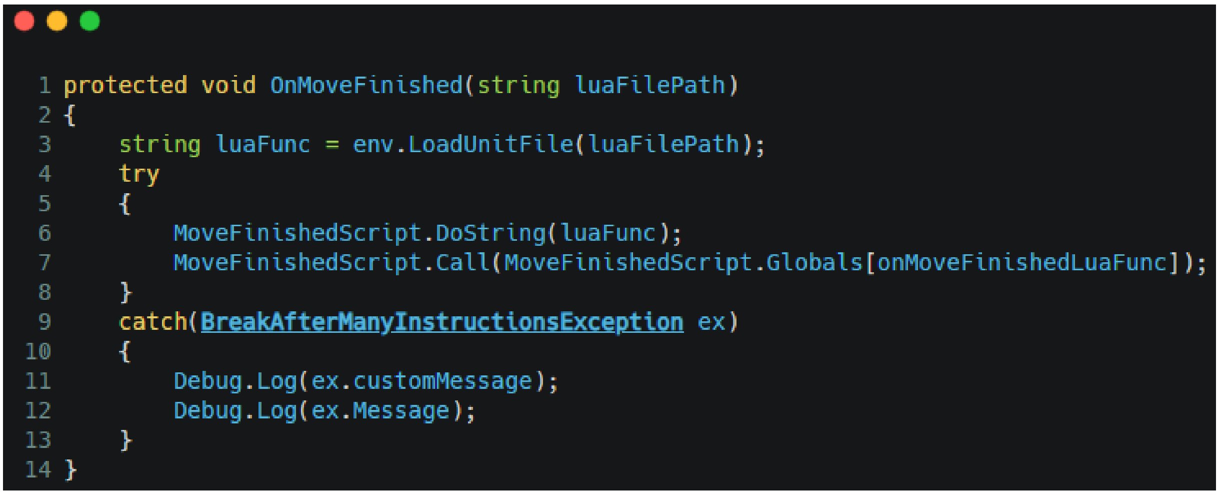
Counting the number of instructions when executing Lua files is possible by attaching a debugger class to the Moonsharp Script object which executes the Lua files:



```
1 public Script getNewScript()
2 {
3     Script script = new Script(CoreModules.Preset_SoftSandbox);
4     script.AttachDebugger(new BreakAfterManyInstructionsDebugger());
5     script.Options.DebugPrint = s = >Debug.Log(s);
6     return script;
7 }
```

On line 4, a debugger is attaching the class “BreakAfterManyInstructionsDebugger”. This class counts the instructions executed in a Lua file and it raises an exception when a limit has been reached.

The exception is caught on line 9 in the “OnMoveFinished” event function down below.



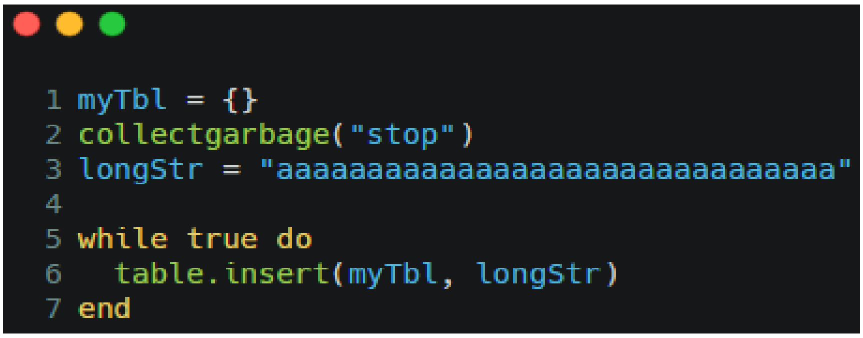
```
 1 protected void OnMoveFinished(string luaFilePath)
 2 {
 3     string luaFunc = env.LoadUnitFile(luaFilePath);
 4     try
 5     {
 6         MoveFinishedScript.DoString(luaFunc);
 7         MoveFinishedScript.Call(MoveFinishedScript.Globals[onMoveFinishedLuaFunc]);
 8     }
 9     catch(BreakAfterManyInstructionsException ex)
10     {
11         Debug.Log(ex.customMessage);
12         Debug.Log(ex.Message);
13     }
14 }
```

Result of fps requirement of Lua file execution

The prototype game did not follow the requirement. The frame rate dropped down to approximately 2 to 3 fps when a Lua file with an infinite loop was executed. This happened both on the Linux and Windows build. Due to time constraints, we did not have time to improve the performance. We could have executed the Lua files on a different thread than the main thread. This would decrease the load of the main thread which handled the rest of the game logic.

Memory usage of executing Lua files

This requirement is about making sure that the system does not crash because of memory usage when executing Lua files. This requirement did not get tested because of time constraints. However, we do think that the system is robust against too much memory usage through Lua files. Consider this Lua code example which will result in a crash because of memory usage:



```lua
1 myTbl = {}
2 collectgarbage("stop")
3 longStr = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"
4
5 while true do
6   table.insert(myTbl, longStr)
7 end
```

This code stops the garbage collector on line 2 which is possible to use in the “Soft Sandbox” which was mentioned in the previous security section. Afterwards, the code adds a long string in an infinite loop into the data structure table “myTbl”. The prototype game could stop this from happening since the game already counts the number of instructions a Lua file executes. A test which counts the instructions of Lua files does not necessary solve all kinds of memory usage problems. This depends on, among other things, if the Lua script can allocate enough memory to crash the game before the “BreakAfterManyInstructionsException” is raised.

Compatibility testing

The prototype was tested for 64-bit versions of Ubuntu 20.04.2 LTS and Windows 10 OS build 19041.928. *[30] [31]*. The testing focused on making sure that tiles, combat, unit and sound effect could be modified. In addition, we focused on checking if the UI worked correctly and reflected upon the usefulness of the UIs. Functionality related to testing real-time movement was not tested. We did not prioritize it because this feature was not directly related to modding. Real-time movement is a feature in the game which cannot be modified.

We did not manage to run a build for mac OS 11 (Big Sur) as we had planned because the testing for this platform started too late in the project.

What got tested

The use-cases related to modding were prioritized in the compatibility testing. This included modding tiles, combat, sound effect and units. These features worked on both the Windows and Linux build. However, there were some usability issues that were found related to combat and sound effects. The battle screen did not always communicate clearly if a damage- or defence increase had been added in a Lua file. Moreover, no message was given, neither in game nor in a log file, when the specified sound path did not exist in a Lua file.

What got added because of testing

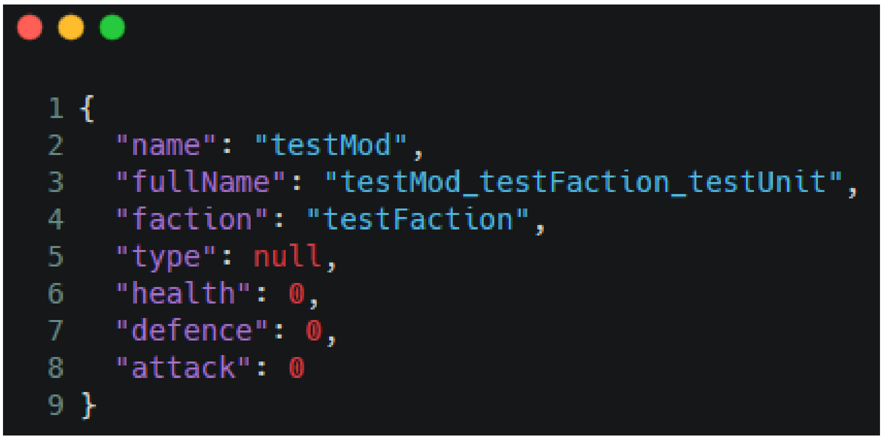
The ability to toggle between full screen and windowed mode was added. In addition, code was added to print out messages to a log file. This will be useful later when newer builds are tested.

User testing

There was no user testing with upper secondary school students because of time constraints. As an alternative we arranged a user test with our client instead. This gave us both technical feedback and feedback related to user experience.

Technical feedback

There were issues related to combat with the battle-screen and the health points for the units. Firstly, the battle screen did not appear in the first attack in the play session. However, the battle screen appears when the player attacks for the second time. Secondly, units have 0 in health as their default value when a new mod is created. This should be changed to a positive number in the “UnitData.json” file which stores stats for the unit such as health, attack and defence. If a unit has 0 health point, then all attacks end in the same way – the defending unit gets killed. In addition, our client advised to make a check in the game if the health is either 0 or negative in the “UnitData.json” file. This will prevent modders from creating units with health stats which do not make sense gameplay wise.

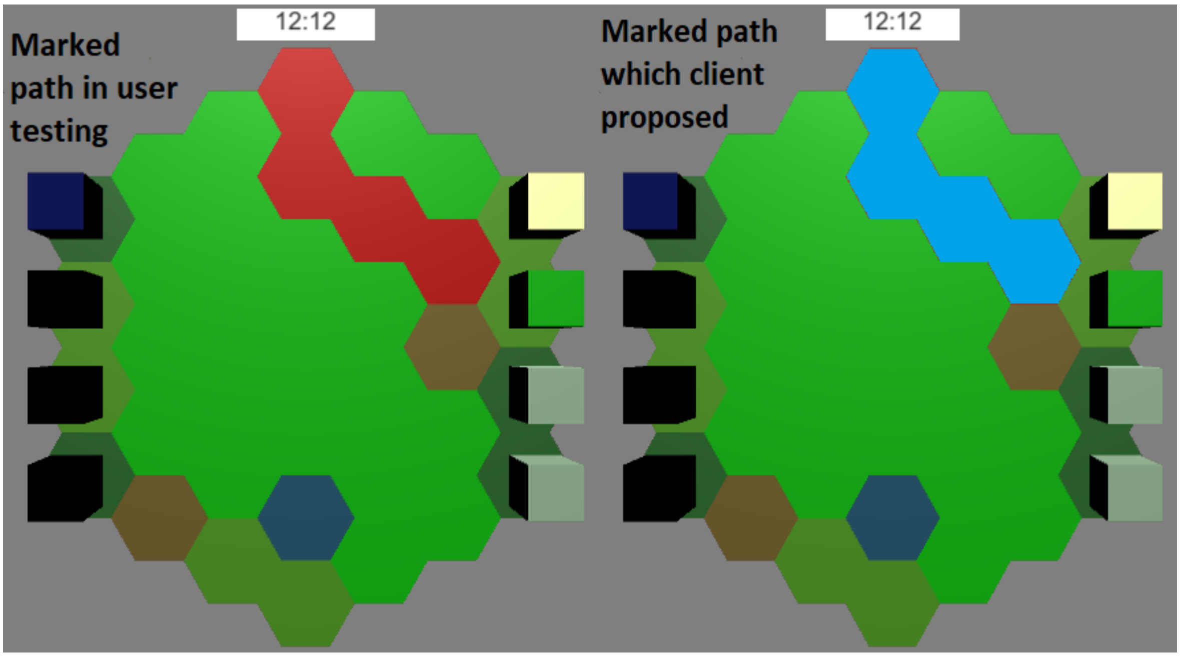


```json
{
  "name": "testMod",
  "fullName": "testMod_testFaction_testUnit",
  "faction": "testFaction",
  "type": null,
  "health": 0,
  "defence": 0,
  "attack": 0
}
```

UnitData.json with 0 as its default value for health.

User experience feedback

The feedback was related to communicating more clearly what options a unit has when it moves and give a clear message if a tile is non-traversable.
A path marked in red will be shown before the player chooses to move a unit like this:


Marked path in user testing
12:12
Marked path which client proposed
12:12

The green-coloured unit has a path marked in red.

Our client said that red should not be used since it normally means that something is wrong, like an error. He proposed using blue as the colour for the marked path instead. In addition, this is also more helpful for people with colour blindness, especially for those in the category protanopia which has difficulties seeing the difference between red and green *[32]*.

Another feedback we received was to use yellow as colour instead of red to mark an enemy unit which can be attacked. The reason to use yellow is because it is often perceived as a warning. The warning in this case is that a unit is about to attack an enemy.

Deployment

We have made successfully working builds for both Windows 10 and Ubuntu 20.04.02 LTS. As previously stated, there were problems with the macOS build and testing for this platform started too late in the project. Therefore, we chose to not use that much time on fixing build issues with the mac build.

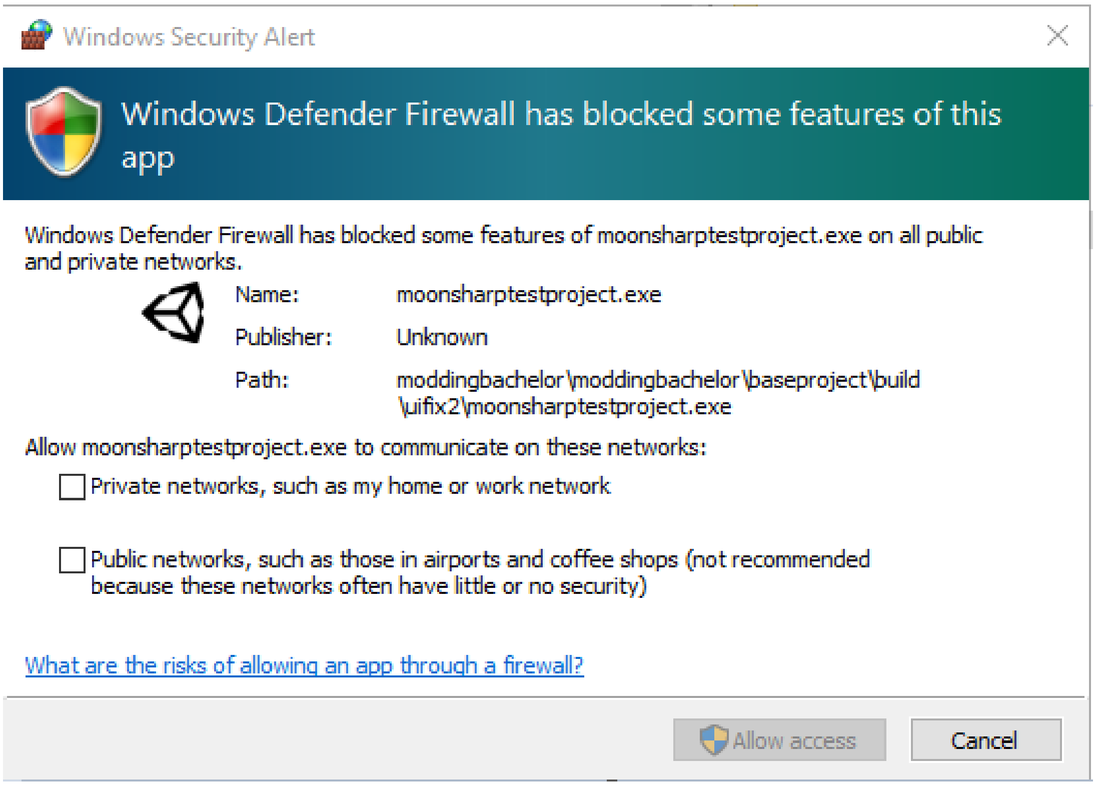
Problems with builds

The first builds of the prototype could not create or load in mods. This was solved by configuring Unity to use a newer version of .Net. This framework is used since C# is used with Unity. The new version was changed to .NET 4 instead of the old .NET 2. This fixed many issues related to reading from JSON files which was the reason why mods could not be read or created.

There is one disadvantage with using a newer .NET version. .NET version 4 does not have as good cross-platform support as .NET 2*[33]*. However, this did not pose a problem when testing our prototype on Windows or Linux.

Practical info about the builds

The only inconvenience with the Windows build is to specify a few simple settings regarding the firewall when project has been opened.



Windows Security Alert

Windows Defender Firewall has blocked some features of this app

Windows Defender Firewall has blocked some features of moonsharptestproject.exe on all public and private networks.

Name: moonsharptestproject.exe

Publisher: Unknown

Path: moddingbachelor\moddingbachelor\baseproject\build\uifix2\moonsharptestproject.exe

Allow moonsharptestproject.exe to communicate on these networks:

□ Private networks, such as my home or work network

□ Public networks, such as those in airports and coffee shops (not recommended because these networks often have little or no security)

What are the risks of allowing an app through a firewall?

The firewall settings windows which appear when opening the Windows build.

The Linux build is also simple to use. Here it is important to explicitly make the game, or more specifically the .86_64 file, executable. This is done for security purposes.

Conclusion and Discussion

Result

Our solution largely reflects what we had originally envisioned our TBS prototype game to look like. While most of our intended features had been implemented, some aspects did not make the cut, either due to time constraints, or simply because we did not deem them fitting for our project. Our client, being happy with the results, approved of our latest build and decided to use it as the basis for their prototype for conducting user testing with high school students.

When it comes to the mod aspect, users are able to create their own mod files through the mod creator interface, specifying the name of the mod, the faction, and its designated unit. The mod files generated allow users to edit the various attributes of a unit through Lua and JSON file editing. These attributes include attack points, health points, defense points, unit type, sound effects upon attacking, and faction image.

In terms of gameplay, we successfully adjusted the framework to allow generating maps with random tiles. There are various tile types that have different impacts on unit movement, a feature which Progress can further develop in order to add more effects if they so desire. The game features UI in the form of pop-up windows for units, tiles, and combat, offering visual feedback to players on not just the basic info involved in the gameplay but also the effects of their modified units' attributes.

One of the planned features included modifiable weapons, armors, and ammos. The idea was for users to equip the various units with different equipment altering units' interaction with the game world, whether it be for combat or terrain movement. These alterations could vary from equipping an archer with a weapon more effective against certain other unit types, to perhaps armor that ignored certain terrain types (e.g., crossing lava), and more.

Another proposed feature was to integrate some sort of connection to an external storage (e.g., a database) in order to store and make mods accessible online. This idea was scrapped relatively early into the project's development, partly due to a lack of push from our client, but also in favor of focusing on gameplay and modding-related features of our project. As we realized that a database might be outside the scope of our project task, we ultimately decided to omit this feature.

Challenges in regulating freedom in modding

Throughout the development process, we started discussing just how much freedom a user should have in altering the game via mods. There comes a point where the software might be at risk of misuse if someone is allowed too much freedom in modding. As the freedom of modding increases, more system-sensitive actions such as reading from files, writing new files, loading 3rd-party modules etc., become available to modders. How much freedom should a modder have before it is not considered unacceptable? Too much freedom could potentially allow modders to break the game logic, which might pose a security threat, but also deviate too far from how the game originally was built to be played.

Related to that discussion, however, we also discussed how video game mods are meant to be a creative expression of one’s ideas, and how limiting freedom could also lead to less variety in expressing mod ideas. In an educational setting, one would have to determine just how much of the game’s base a student would be allowed to mod, while also taking into consideration how much expressive freedom a student should be allowed to integrate into the game.

Despite this, we still believe that mods are a viable platform for teaching students the principles of programming. The creative aspect of mods would allow students to learn by trying to implement an idea they find interesting, rather than being bound to a specific task provided by the teacher. By motivating students to solve a task by using their creativity and implementing their own solutions, it will motivate them to learn the necessary programming principles needed in order to bring their idea into fruition while simultaneously solving the task at hand.

Further work

There are a lot of possibilities to improve the modding integration, but the capabilities of modding integration is dependent on the game features that exist in the game and how open they are to modification. The game we created to implement the Lua integration with has almost all its components open for modding. That said, the components themselves are not highly customizable because there are not a lot of “data” from the components exposed to the Lua scripts to be used to create a mod for it. Therefore, we would like to expose more meaningful variables and functions that can be modified and create more if necessary.
This should provide more alternatives when making mods and give more control in modifying mechanics.

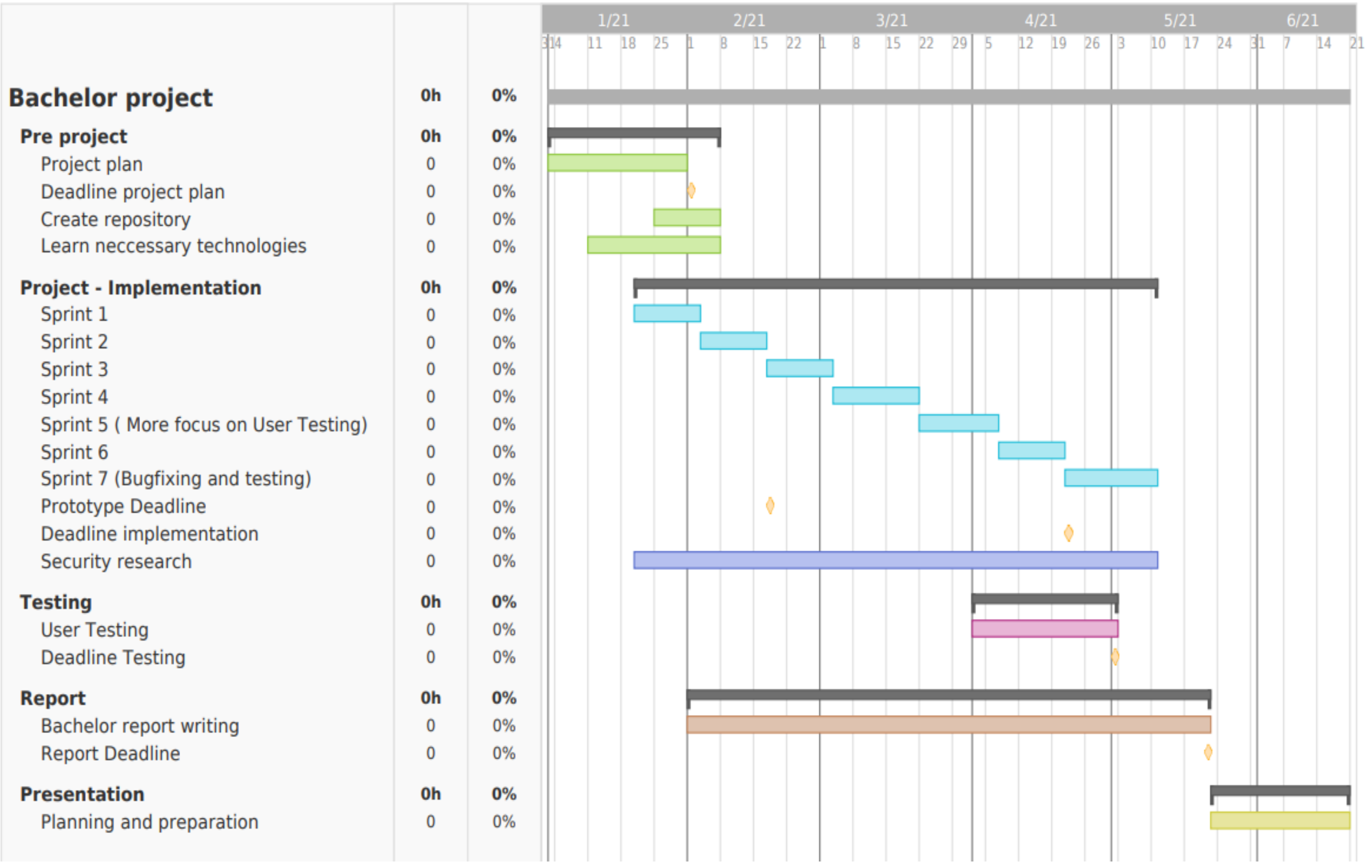
A further step is to allow creating custom functions within the Lua files that can be used in the C# code. To achieve this a naming standard would have to be established to fetch and be used in the C# code. For example, all custom functions that are supposed to be used when a unit attacks must start with something like “UnitAtkFunc_....”. And these standards can be improved to accommodate more unique functions.

The modding integration lacks error handling. If a modder writes code which results in an error, then the modder should be explained through error messages what the problem is. This would include reporting errors coming from the game, and errors if they are creating mods not following our mod schema. Which can be writing Lua files that don't adhere to our Lua file schema or adding objects to the JSON files that don't have the correct structure or data.

Group work evaluation

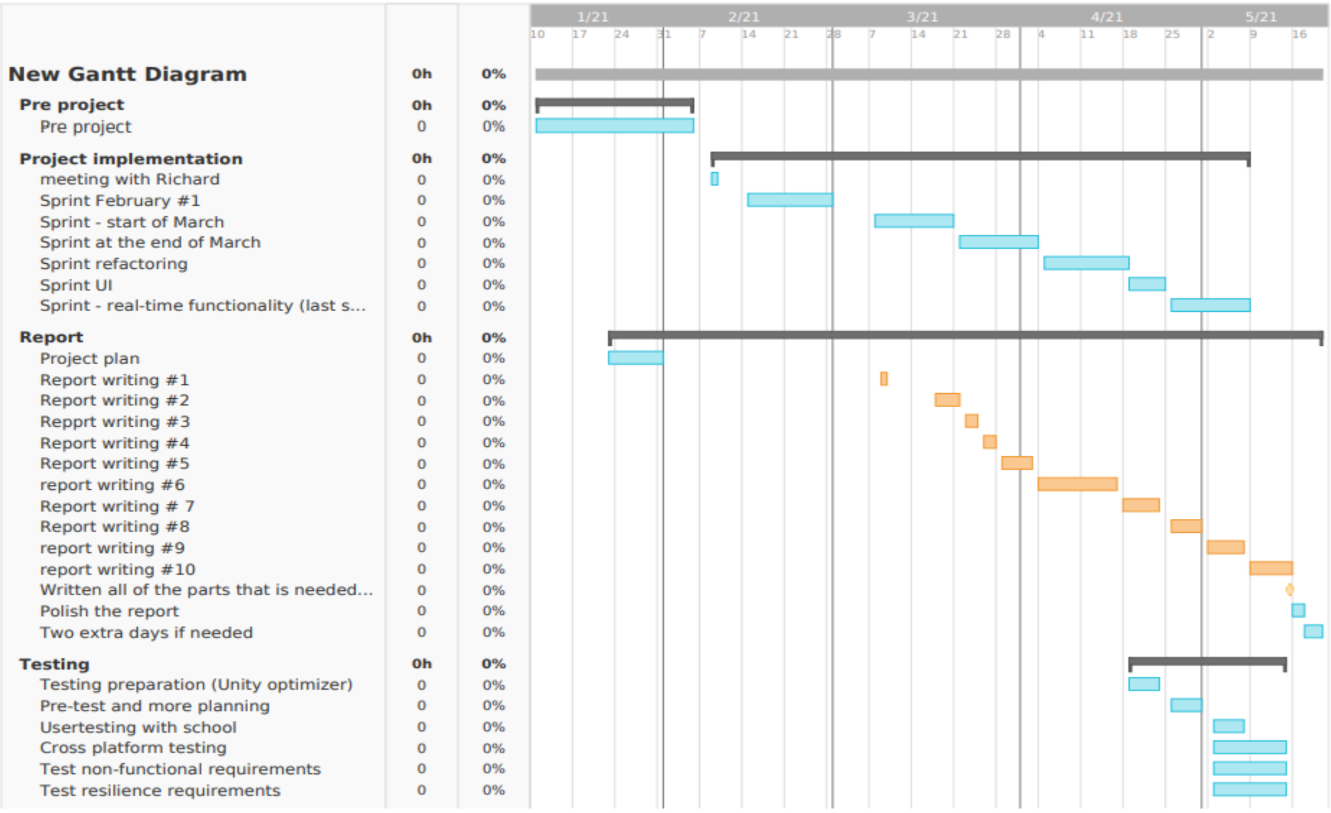
What we learned from planning - Gantt

Down below is our Gantt diagram from our project plan which was made at the end of January.



Bachelor project	**0h**	**0%**
Pre project	**0h**	**0%**
Project plan	0	0%
Deadline project plan	0	0%
Create repository	0	0%
Learn neccessary technologies	0	0%
Project - Implementation	**0h**	**0%**
Sprint 1	0	0%
Sprint 2	0	0%
Sprint 3	0	0%
Sprint 4	0	0%
Sprint 5 (More focus on User Testing)	0	0%
Sprint 6	0	0%
Sprint 7 (Bugfixing and testing)	0	0%
Prototype Deadline	0	0%
Deadline implementation	0	0%
Security research	0	0%
Testing	**0h**	**0%**
User Testing	0	0%
Deadline Testing	0	0%
Report	**0h**	**0%**
Bachelor report writing	0	0%
Report Deadline	0	0%
Presentation	**0h**	**0%**
Planning and preparation	0	0%

And here is the Gantt diagram over what happened during the project:



New Gantt Diagram	**0h**	**0%**
Pre project	**0h**	**0%**
Pre project	0	0%
Project implementation	**0h**	**0%**
meeting with Richard	0	0%
Sprint February #1	0	0%
Sprint - start of March	0	0%
Sprint at the end of March	0	0%
Sprint refactoring	0	0%
Sprint UI	0	0%
Sprint - real-time functionality (last s...	0	0%
Report	**0h**	**0%**
Project plan	0	0%
Report writing #1	0	0%
Report writing #2	0	0%
Repprt writing #3	0	0%
Report writing #4	0	0%
Report writing #5	0	0%
report writing #6	0	0%
Report writing # 7	0	0%
Report writing #8	0	0%
report writing #9	0	0%
report writing #10	0	0%
Written all of the parts that is needed...	0	0%
Polish the report	0	0%
Two extra days if needed	0	0%
Testing	**0h**	**0%**
Testing preparation (Unity optimizer)	0	0%
Pre-test and more planning	0	0%
Usertesting with school	0	0%
Cross platform testing	0	0%
Test non-functional requirements	0	0%
Test resilience requirements	0	0%

By viewing both Gantt diagrams we realize that we did not follow the plan for report writing or testing. Firstly, report writing stopped completely after the project plan was delivered and the writing started again approximately in the middle of March. This can be seen by viewing the orange tasks in the last diagram. Secondly, testing was planned to start in April, but it started in May instead.

On the other hand, when we realized that we had not followed our initial Gantt diagram we decided to make a new one in the middle of April. We were more determined to follow the new Gantt diagram since we did not follow the last one.

To ensure that we worked enough and efficiently with our report we started doing weekly feedback sessions of the report with our supervisor. This gave us a weekly deadline which helped us to write on the report regularly. In addition, to avoid using too much time on programming rather than report writing we communicated clearly what features we realistically could implement to our product owner.

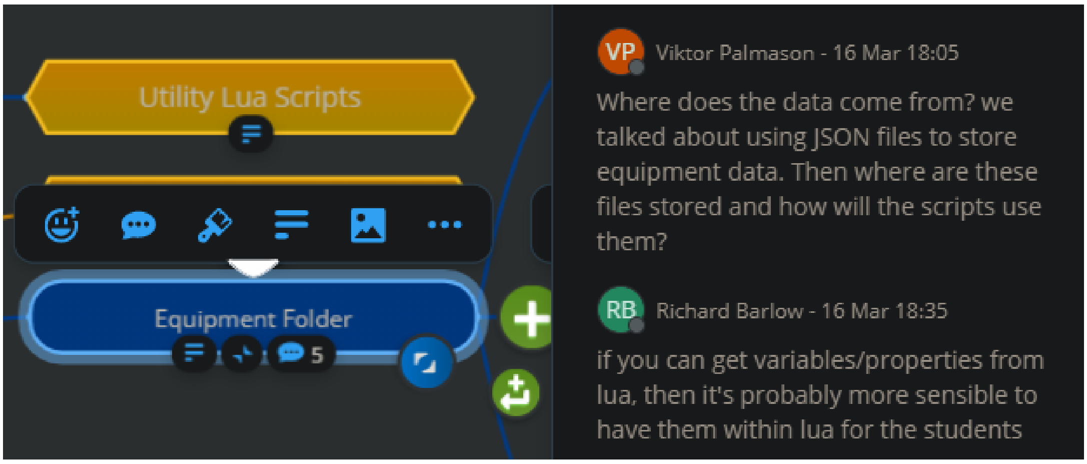
Use more time for specification

We think it would have been easier for us to work on this project if we had set aside more time to define precisely what we were going to do. At the start we knew we would make a Lua integration using Moonsharp for an upcoming game from Progress. We knew that the game would be a turn-based strategy game which uses tiles for its world. We did not know any more details about the game than that. In addition, we did not know that we were going to use a Turn Based Framework for the development for the game before 10th of February. If we had known this at the beginning then we could have used more time learning the framework in addition to Moonsharp and Lua during the first month of the project.

We could have been stricter with our product owner and encouraged him to set aside more time to specify the project. However, game development benefits from agile development since game development is a creative task which requires trial and error to be successful. On the other hand, we think that it would have been possible to do more planning since our client knew that the game was going to include tiles and be strategy oriented. Therefore, choosing the TBS framework would have been possible early on in January.

Communication struggles

Throughout the bachelor project there had been times where our product owner could not attend a scheduled meeting which led to a rescheduling of those meetings. We tried to cover the lack of meetings with emails in-between rescheduled meetings instead when this occurred. This worked well enough to get a high-level understanding of what we were supposed to do. However, we did not get enough information to properly define tasks and describe exactly when a given task was done.



Viktor Palmason - 16 Mar 18:05

Where does the data come from? we talked about using JSON files to store equipment data. Then where are these files stored and how will the scripts use them?

Richard Barlow - 16 Mar 18:35

if you can get variables/properties from lua, then it's probably more sensible to have them within lua for the students

Comments attached to an object in the Ayoa diagram.

The measures that we did to improve our communication was to use Ayoa and Discord. The former has a feature that allows commenting the different parts of the diagram, which opened the possibility to attach questions related to specific parts of the folder structure diagram. This helped us to get more detailed info regarding the project.
Discord was used to have a low threshold way for communication with our product owner. We first used this with our client on the beginning of April.

Experiences from learning the necessary technologies

During the first 4 weeks of the bachelor project we had a difficult time learning Lua, Moonsharp and more importantly how a modding support could have been implemented. When we look back at this time we have realized that we should have asked for more help from our product owner, fellow students or from NTNUs teachers. They could have provided knowledge, tutorials or other learning resources about this topic. This would have helped us since the only tutorials we knew was one Moonsharp tutorial *[34]*. This gave us an overview over the functionalities of Moonsharp, but it did not give us a thorough view into the different ways into adding modding support.

In addition, we should have learned Lua more properly before we learned Moonsharp. This is because it is easier to get a high-level understanding of ways of implementing modding support when the programmer knows the possibilities and constraints of Lua.

Conclusion

When we look back at our TBS game, we are proud of what we have accomplished. This project has given us insight into how valuable modding is both in terms of a potential way of learning programming, and for improving a video game. Moreover, this project has given us insight into the indie video game landscape in Norway when receiving guidance from Progress. Lastly, we would like to thank our client for choosing us for this bachelor project and our supervisor for assisting us during development.

References

[1] *A. Dyer, "PC Game Mods - From Smurfs to Counter-Strike and Beyond!," 18 March 2016. [Online]. Available: https://www.nvidia.com/en-us/geforce/news/history-of-pc-game-mods/. [Accessed 31 March 2021].*

[2] *Steam, "Steam Workshop about," [Online]. Available: https://steamcommunity.com/workshop/workshopsubmitinfo/. [Accessed 27 April 2021].*

[3] *Redbull, "The history of Counter-Strike," 6 August 2020. [Online]. Available: https://www.redbull.com/se-en/history-of-counterstrike. [Accessed 06 April 2021].*

[4] *M. S. El-Nasr and B. K. Smith, "Learning through game modding," p. 21, January 2005.*

[5] *NTNU, "CodeFlip," CodeFlip, p. s.36, 2019.*

[6] *Norwegian Directorate of Education and Training, "udir.no," [Online]. Available: https://www.udir.no/kl06/INF1-01/Hele/Kompetansemaal/informasjonsteknologi-2. [Accessed 23 April 2021].*

[7] *Norwegian Directorate of Education and Training, "udir.no," 29 September 2016. [Online]. Available: https://www.udir.no/tall-og-forskning/finn-forskning/rapporter/teknologi-og-programmering-for-alle/. [Accessed 23 April 2021].*

[8] *Lua website, "Lua: about," [Online]. Available: https://www.lua.org/about.html. [Accessed 5 April 2021].*

[9] *GameDev StackExchange, "Why are games using interpreted languages instead of compiling code into libraries and calling them at runtime?," 26 September 2016. [Online]. Available: https://gamedev.stackexchange.com/questions/130531/why-are-games-using-interpreted-languages-instead-of-compiling-code-into-librari. [Accessed 5 April 2021].*

[10] *Unity, "Unity Asset Store," 7 December 2015. [Online]. Available: https://assetstore.unity.com/packages/templates/systems/turn-based-strategy-framework-50282. [Accessed 18 May 2021].*

[11] *P. Hyman, "Video game companies encourage 'modders'," 9 April 2004. [Online]. Available: https://web.archive.org/web/20080506004712/http://www.hollywoodreporter.com/hr/search/article_display.jsp?vnu_content_id=1000484956. [Accessed 31 March 2021].*

[12] *T. Mubarik, "Why choose Scrum for you project?," 17 December 2020. [Online]. Available: https://www.linkedin.com/pulse/why-choose-scrum-your-project-talha-mubarik. [Accessed 21 March 2021].*

[13] *P. G. D. E. S. R. A. H. S. and L. B. , "Onboarding: How Newcomers Integrate into an Agile Project Team," p. s.12, 2020.*

[14] *Productplan, "Kanban vs. Scrum," [Online]. Available: https://www.productplan.com/learn/kanban-scrum/. [Accessed 21 March 2021].*

[15] *M. Dealessandri, "gameindustry.biz," 16 January 2020. [Online]. Available: https://www.gamesindustry.biz/articles/2020-01-16-what-is-the-best-game-engine-is-unity-the-right-game-engine-for-you. [Accessed 3 May 2021].*

[16] *Unity, "Unity Manual," 8 August 2017. [Online]. Available: https://docs.unity3d.com/Manual/GameObjects.html. [Accessed 3 May 2021].*

[17] *Unity, "Unity Manual," 18 September 2018. [Online]. Available: https://docs.unity3d.com/Manual/UsingComponents.html. [Accessed 3 May 2021].*

[18] *Github, "Moonsharp Github Page," [Online]. Available: https://github.com/moonsharp-devs/moonsharp/graphs/contributors. [Accessed 13 April 2021].*

[19] *Github, "Maostropaolo's User Page on Github," [Online]. Available: https://github.com/xanathar. [Accessed 13 April 2021].*

[20] *Unity , "Unity manual 2019.4," 14 May 2021. [Online]. Available: https://docs.unity3d.com/2019.4/Documentation/Manual/StreamingAssets.html. [Accessed 19 May 2021].*

[21] *Unity, "Unity Manual," 19 April 2021. [Online]. Available: https://docs.unity3d.com/Manual/AssetDatabaseRefreshing.html. [Accessed 22 April 2021].*

[22] *Unity Scripting API, "Unity Documentation," [Online]. Available: https://docs.unity3d.com/ScriptReference/WWW.html. [Accessed 29 April 2021].*

[23] *Unity Scripting API, "Unity Scripting API ((Wayback Machine)," 12 August 2016. [Online]. Available: http://web.archive.org/web/20160812184333/http://docs.unity3d.com/ScriptReference/WWW-ctor.html. [Accessed 23 April 2021].*

[24] *Wikipedia, "Perlin noise," 15 April 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Perlin_noise&oldid=1017951949. [Accessed May 2021].*

[25] *S. Lague, "Youtube," 6 February 2016. [Online]. Available: https://www.youtube.com/watch?v=WP-Bm65Q-1Y. [Accessed 14 April 2021].*

[26] *S. Lague, "Github," 2017. [Online]. Available: https://github.com/SebLague/Procedural-Landmass-Generation. [Accessed 14 April 2021].*

[27] *R. Oliveira, "GameDev Academy," 15 October 2019. [Online]. Available: https://gamedevacademy.org/complete-guide-to-procedural-level-generation-in-unity-part-1/. [Accessed May 2021].*

[28] *Moonsharp, "Moonsharp.org," [Online]. Available: https://www.moonsharp.org/sandbox.html. [Accessed 13 May 2021].*

[29] *[Online]. Available: https://www.moonsharp.org/sandbox.html.*

[30] *Microsoft, "support.microsoft.com," 13 April 2021. [Online]. Available: https://support.microsoft.com/en-us/topic/april-13-2021-kb5001330-os-builds-19041-928-and-19042-928-cead30cd-f284-4115-a42f-d67fec538490. [Accessed 11 May 2021].*

[31] *Ubuntu, "wiki.ubuntu.com," 11 February 2021. [Online]. Available: https://wiki.ubuntu.com/FocalFossa/ReleaseNotes/ChangeSummary/20.04.2. [Accessed 11 May 2021].*

[32] *D. Nichols, "DavicMathLogic," [Online]. Available: https://davidmathlogic.com/colorblind/#%231B69D8-%2344E51E-%23FFC107-%23004D40. [Accessed 13 May 2021].*

[33] *Unity, "Unity documentation," [Online]. Available: https://docs.unity3d.com/Manual/overview-of-dot-net-in-unity.html. [Accessed 11 May 2021].*

[34] *MoonSharp, "Getting Started - A quick guide to your first MoonSharp project," MoonSharp, [Online]. Available: https://www.moonsharp.org/getting_started.html. [Accessed 19 05 2021].*

[35] *A. Holst and M. Magnussen, "Code Flip: A Game for the Norwegian Elective Course of Programming," p. s.13 (section 2.3), 2019.*

[36] J. L. Jones, "Lua is Cool," 10 january 2016. [Online]. Available: https://www.lua.org/pil/contents.html. [Accessed 15 February 2021].

[37] R. Ierusalimschy, L. H. d. Figueiredo and W. C. Filho, "Lua – an extensible extension language," 1996. [Online]. Available: https://www.lua.org/spe.html. [Accessed 29 March 2021].

[38] I. Yucel, J. Zupko and M. S. El-Nasr, "IT education, girls and game modding," p. 14, 2 May 2006.

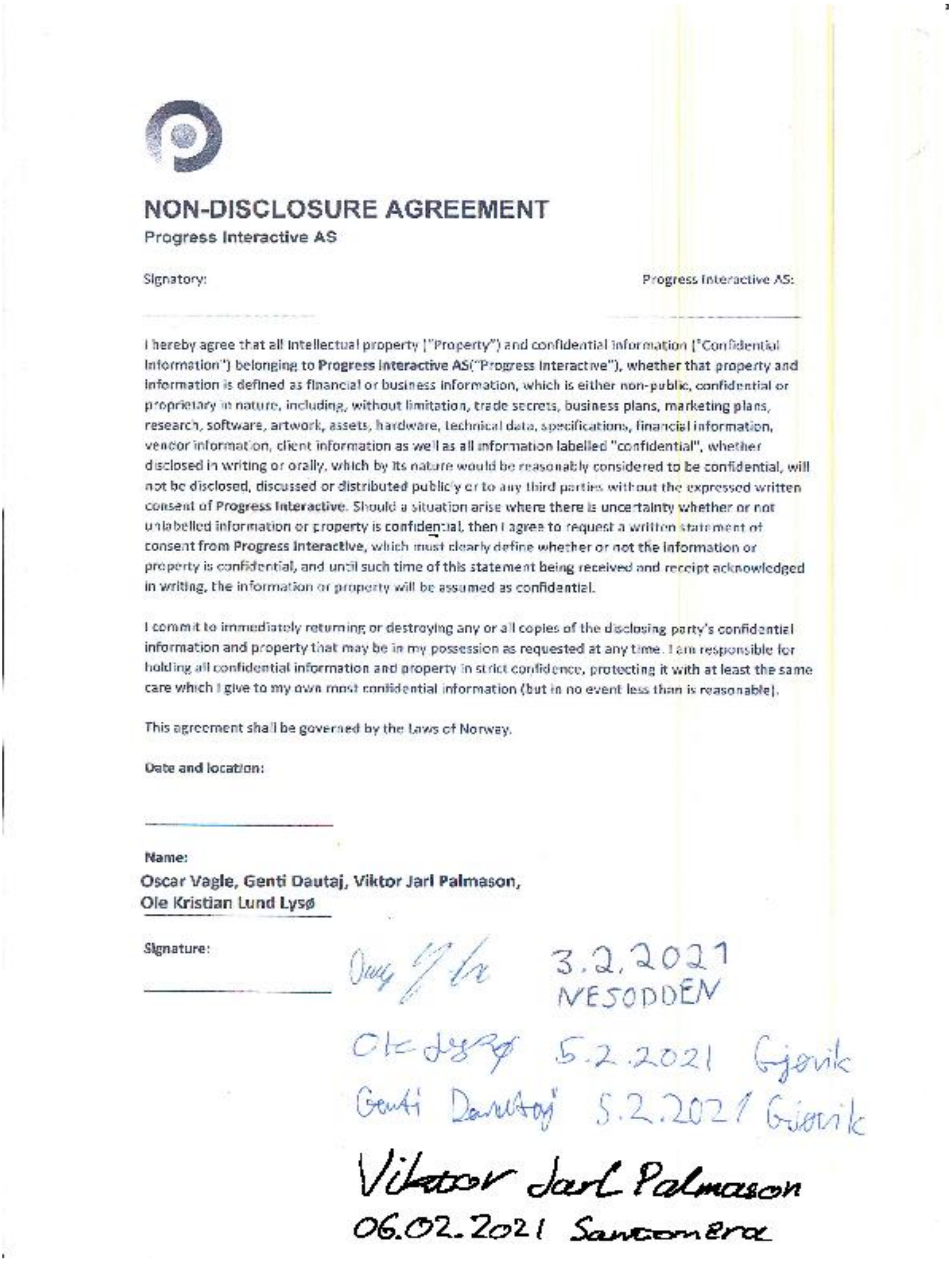
[39] Progress Interactive Wordpress Page, "Progress Interactive Blog," [Online]. Available: https://progressinteractiveblog.wordpress.com/. [Accessed 26 April 2021].

[40] Linkedin, "Progress Interactive AS - Linkedin," [Online]. Available: https://www.linkedin.com/company/progress-interactive/about/. [Accessed 26 April 2021].

[41] Lua-users, "Lua-users.org," 18 September 2010. [Online]. Available: http://lua-users.org/wiki/IoLibraryTutorial. [Accessed 13 May 2021].

Appendix

NDA



NON-DISCLOSURE AGREEMENT

Progress Interactive AS

Signatory: Progress Interactive AS:

I hereby agree that all intellectual property ("Property") and confidential information ("Confidential Information") belonging to **Progress Interactive AS**("Progress Interactive"), whether that property and information is defined as financial or business information, which is either non-public, confidential or proprietary in nature, including, without limitation, trade secrets, business plans, marketing plans, research, software, artwork, assets, hardware, technical data, specifications, financial information, vendor information, client information as well as all information labelled "confidential", whether disclosed in writing or orally, which by its nature would be reasonably considered to be confidential, will not be disclosed, discussed or distributed publicly or to any third parties without the expressed written consent of **Progress Interactive**. Should a situation arise where there is uncertainty whether or not unlabelled information or property is confidential, then I agree to request a written statement of consent from **Progress Interactive**, which must clearly define whether or not the information or property is confidential, and until such time of this statement being received and receipt acknowledged in writing, the information or property will be assumed as confidential.

I commit to immediately returning or destroying any or all copies of the disclosing party's confidential information and property that may be in my possession as requested at any time. I am responsible for holding all confidential information and property in strict confidence, protecting it with at least the same care which I give to my own most confidential information (but in no event less than is reasonable).

This agreement shall be governed by the Laws of Norway.

Date and location:

Name:

Oscar Vagle, Genti Dautaj, Viktor Jarl Palmason,
Ole Kristian Lund Lysø

Signature:

3.2.2021
NESODDEN

Ole Lysø 5.2.2021 Gjøvik

Genti Dautaj 5.2.2021 Gjøvik

Viktor Jarl Palmason
06.02.2021 Santomera

Project plan

Background

Progress Interactive AS is an independent video game development company situated in Hamar, Norway. Their interests extend to developing games as a product for business to business, business to consumer, and business to education, with the idea to create games for a positive change in society.

Programming taught at high school usually involves making students develop something with a predetermined result from scratch and thus opens up very little room for freedom for the students to explore and apply their own creative ideas, making the process dull for students. For this project, Progress wants to develop a game intended for high school students to learn the basics of programming concepts and practices. They want these skills to come into fruition through the application of Lua script integration via Unity in order to modify the game to their own liking. By allowing students to modify an already-existing game it opens up for a more creative process, as well as allowing students to express themselves more freely through their programming work while simultaneously learning the fundamentals of programming.

Project goals

Product goal:

A Lua script integration for Progress's game that gives players the ability to make mods to customize the game. The integration will also act as a learning platform for students to learn about basic programming. Hence it should be accessible by users with no programming knowledge and for experienced programmers.

Impact goal:

There are two impact goals:

- Make Progress's game more engaging by making it customisable because players can then change and probably enhance the gameplay by adding their creative creations and ideas, and it also gives players the ability to express themselves.
- Providing a purposeful/meaningful and rewarding learning experience when acting as a learning platform for programming. By learning programming and how to write the scripts the students will be able to make mods for the game to change the gameplay and enhance it as well. And so as they continue to learn and keep getting better, they will be able to make more creative mods. Therefore, when the students learn, they are rewarded with the ability to enhance their gameplay experience by creating mods for the game.

Learning Goals:

- Learn to write scripts in Lua to make mods for other video games in the future.
- Get a deeper understanding of how game development works in a real-life scenario
- Learn about the effects of having a game being open mods.
- Learn about the modding community.
- Learn about education through video games

    - Get better at using the Unity engine
    - Get better at time tracking work
    - Get better at working in a group
    - Gain experience in developing software under an agile development framework (in this case, Scrum)

Project task

The task is to implement a Lua integration for Progress' upcoming Real-Time Strategy (RTS) game. This mod platform will give users the ability to learn some basic programming principles such as…

- Defining, accessing and using meaningful variables.
- Writing own functions, and/or editing pre-defined functions to a new purpose.
- Creating useful Objects and attribute values to these Objects.
- Have multiple scripts communicating through events.

In addition the Lua integration should give users access to use more advanced programming features and principles such as…

- Improve knowledge about Objects and introduce encapsulation to make code cleaner and more secure.
- Teach reusability and lessening duplicate code with inheritance.
- Give a thorough rundown of multitasking through coroutines.

Scope

Game modification is a broad and expansive world that covers just about any game and has existed ever since the 80's. Developers can express their ideas and creativity by implementing features into their games of liking, further enhancing the gameplay experience for both themselves and other fans alike. Mods can range from simple cosmetic changes to new mechanics, entire cities, or in some cases even the option for multiplayer.

Mods come in many shapes and forms, and one of the most popular mods through video game history is the Counter-Strike mod for the first-person shooter game Half-Life. The mod would use the assets present in the game and introduce a terrorists vs. counter-terrorists game mode between players through a multiplayer component that was also added by the developers of the mod. It became a different game altogether and gained so much popularity that Valve, the developers of Half-Life, decided to fully develop Counter-Strike into a standalone game that has now become a global success as a franchise.

There are many different tools and approaches to successfully "modding" a game, and Lua scripting is an example of such a tool which we will use for our project. Through Lua scripting, students will be able to modify the game to implement their own ideas with the scripts they create, resulting in a diverse collection of projects between students. Progress is in charge of developing the game and its base features while we will develop the mod component working as a scripting interface.

In addition to Lua scripts, we will also utilize various tools such as:

- Moonsharp - a C# implementation of the Lua interpreter
- C#
- Unity
- Zip / Gzip Multiplatform Native Plugin for Unity (runtimeZipPlugin)

Group Infrastructure

Group roles:

Name: Assigned Role	Responsibilities
Viktor: Project leader, Report supervisor*, Scrum master Titles: Unity consultant, RuntimeZip consultant.	Project leader: Ensures that the goals of the project are always clear for the group and clarifies any misunderstandings or confusions about the project. Be vigilant over the state of the Project and keep the group focused on the tasks and oriented towards the goals. Starts each meeting and introduces each issue that must be addressed during the meeting. In addition, if the group is going to make a decision about something and there is an equal number of votes for both sides, then the project leader has a double vote. Unity lead consultant: To be the primary person to help with the Unity engine and with tasks related to it. RuntimeZip lead consultant: To be the primary person to help with the RuntimeZip and with tasks related to it. Report supervisor: Observes the state of the Bachelor report and has the group regularly work on the report.
Oscar: Social coordinator, scheduler, Notetaker	Social coordinator: Arranges social events for the group where we can hang out and do something besides work. Scheduler: Arranges meetings with the project owner and the group supervisor. This role is also the contact person for supervisor and client for the group.

	Notetaker: Take notes during each meeting and then make a summary of what was said and/or decided during the meeting. That includes for example the conclusions, questions, tasks given, planned meetings.
Genti: Arbitrator/monitor, encourager	Arbitrator/monitor: Observes over the state of the group and its members and regularly brings up group climate and process during meetings and discussions, especially if he or she senses tension or conflict brewing. Should settle arguments and conflicts within the group by proposing a suggestion to resolve the dispute. Should make sure that everyone in the group feels they are a part of the group and project. Encourager: Give moral support to the group, be active in praising good work and to keep the group in high spirits.
Ole: Devil's advocate/ Quality Controller, Security supervisor.	Devil's advocate/ Quality Controller: Remains on guard against "groupthink" scenarios (i.e., when the pressure to reach the group goal is so great that the individual members surrender their own opinions to avoid conflict and view issues solely from the group's perspective). Ensures that all arguments have been heard, and looks for holes in the group's decision-making process, in case there is something overlooked. Keeps his or her mind open to problems, possibilities, and opposing ideas. Serves as a quality-control person who double-checks every detail to make sure errors have not been made and searches for aspects of the work that need more attention. Keeps an eye out for mistakes, especially those that may fall between the responsibilities of two group members. Security supervisor: Make sure that the group follows a security standard to ensure a secure development. He or she is not

	responsible for implementing security, just the rules the group must follow when developing.
All: MoonSharp/Lua consultant, Report supervisor.	MoonSharp/Lua consultant: To be familiar enough with MoonSharp and Lua to be able to assist in tasks related to them. Report Supervisor: Observes the state of the Bachelor report and has the group regularly work on the report.

These roles are based on the group roles explained here: https://uwaterloo.ca/centre-for-teaching-excellence/teaching-resources/teaching-tips/developing-assignments/group-work/group-roles-maximizing-group-performance

Group Rules:

1. Group members must respect each group member and their opinions and ideas.
2. Group members should be allowed to speak their mind and not be interrupted.
3. No group member can have more than three roles.
4. No group member can have more than three tasks assigned.
5. A group member must report if he/she is going to be absent from work and or meetings.
6. Team members will put in 25 hours of work each week.
7. A group member may have the responsibility to carry out a task, but the others have the responsibility to make sure that the assigned group member finishes the task and provide help if needed.
8. If a group member is not reachable through main communication channels, Discord or mail, then their personal phone number will be used.
9. Group members must meet on time for each meeting and be flexible on meeting times.

Group routines:

Every Friday there is an internal group meeting where we can discuss technical problems we might get stuck with, how the work has been going and plan what tasks we are going to do next week.
In addition, Wednesday is used as an additional internal group meeting if needed.

Development method

Why we chose Scrum

Requirements are unknown up front and they can change
Since we are making a Lua integration for an upcoming game then that fact alone will make sure that requirements can change. This is because what our Lua integration can offer is dependent on what the game offers. If for example the game has story events, then it is possible for us to expose the appropriate functions, objects and etc. to the Lua interface for

modders to customize. If not, then it is out of our reach. Scrum is suitable for this project because it leaves room for changing sprints in between of them which will probably happen.

In the project we are using unknown technology which in turn makes it difficult for us to estimate correctly how long tasks will take. Since we can change the sprint backlog in between the sprints, then the effect of estimating incorrectly is minimized.

Retrospective meetings are helpful tools for newcomers

Since we have never applied any development methodologies in a real-life scenario before it is valuable for us to have reflection incorporated into the development methodology. Retrospective meetings can help us to reflect over what went well and what went bad from the previous sprints and make adjustments to the next sprint backlog in order to learn from mistakes.

Why we did not use Extreme Programming or Kanban

Since Scrum has sprints then we have a plan for upcoming tasks for the next 2 to 4 weeks. This can help us better estimate how long certain tasks will take and be better prepared to reach our deadlines for our development, and last but not least the bachelor report. Kanban and Extreme Programming does not have sprints in the same sense as Scrum and are therefore methodologies which are harder to plan ahead of time with.

How we implement Scrum

When applying development methodologies in a real-life scenario it is important to understand the underlying motivations for our chosen agile framework, which is Scrum. In other words, we will avoid getting hung up about following exactly every Scrum activity we can think of. However, we should not change too much of the methodology resulting in making a completely different methodology. We are keeping the sprints, the roles with product owner and scrum master, as well as working towards having an independent team.

Sprint

We have chosen to have two weeks sprints in order to have more frequent small deadlines throughout the project. However, this can change if we find it necessary to do so.

Sprint backlog

When we create the sprint backlog we are going to focus on having a clear definition of when a task is done. We think that having a clear definition of a task makes it easier for us to be productive when we have clear tangible tasks that need to be done. Last but not least, the sprint backlog is going to be easily accessible on our Trello board.

During the sprint review and -retrospective meeting we are going to prioritize showing the running functioning code that we have worked on. This will motivate us for actually finishing what we are working on, or at least showing something which works. However, if our client is unable to show up to a meeting then we should upload the code to our repo instead.

Specialist vs generalist

This is an important note to address since Scrum defines generally that team members should be more generalist than specialist. We in the team have decided that we are mainly focusing on having generalists on our team but open up for individual team members to have some more level of expertise in certain areas. What we do not want to do is to be too dependent on a skill which one team member has. This will make our group more vulnerable if for some reason one key team member is unable to work.

Quality Assurance

List of documentation used during the project:

- MoonSharp/Lua: https://www.moonsharp.org/
- Unity Engine: https://unity.com/
- Mirror multiplayer library from Unity: https://mirror-networking.com/
- RuntimeZip plugin: https://assetstore.unity.com/packages/tools/input-management/zip-gzip-multiplatform-native-plugin-39411?_ga=2.55146280.249719791.1611159132-1516186184.1600533260
- Programming language: C#
- Visual Studio.

Implementation

Source Code & Documents

- Project code will be stored in a repository of its own.
- Assets handed out from the client will be retrieved from a different repository shared between us and the client.
- The report will be written in Google Docs.
- Documents that are report, code or product-related will be shared in a Discord channel.

Development-routines

- Every task will be created and sorted in Trello based on its current status.
- Time spent on respective tasks will be time-logged in Toggl.
- Classes and functions will be properly commented.
- Variables, classes, and functions should have meaningful and informative names.
- Every commit should have a descriptive comment.

Internationalization

The project is required to support both English and Norwegian as language options.

Risk Assessment

In the following table there are the different risks which might occur. They include mitigations, consequence and probability for the risk to occur. Number one means low and number three means high probability.

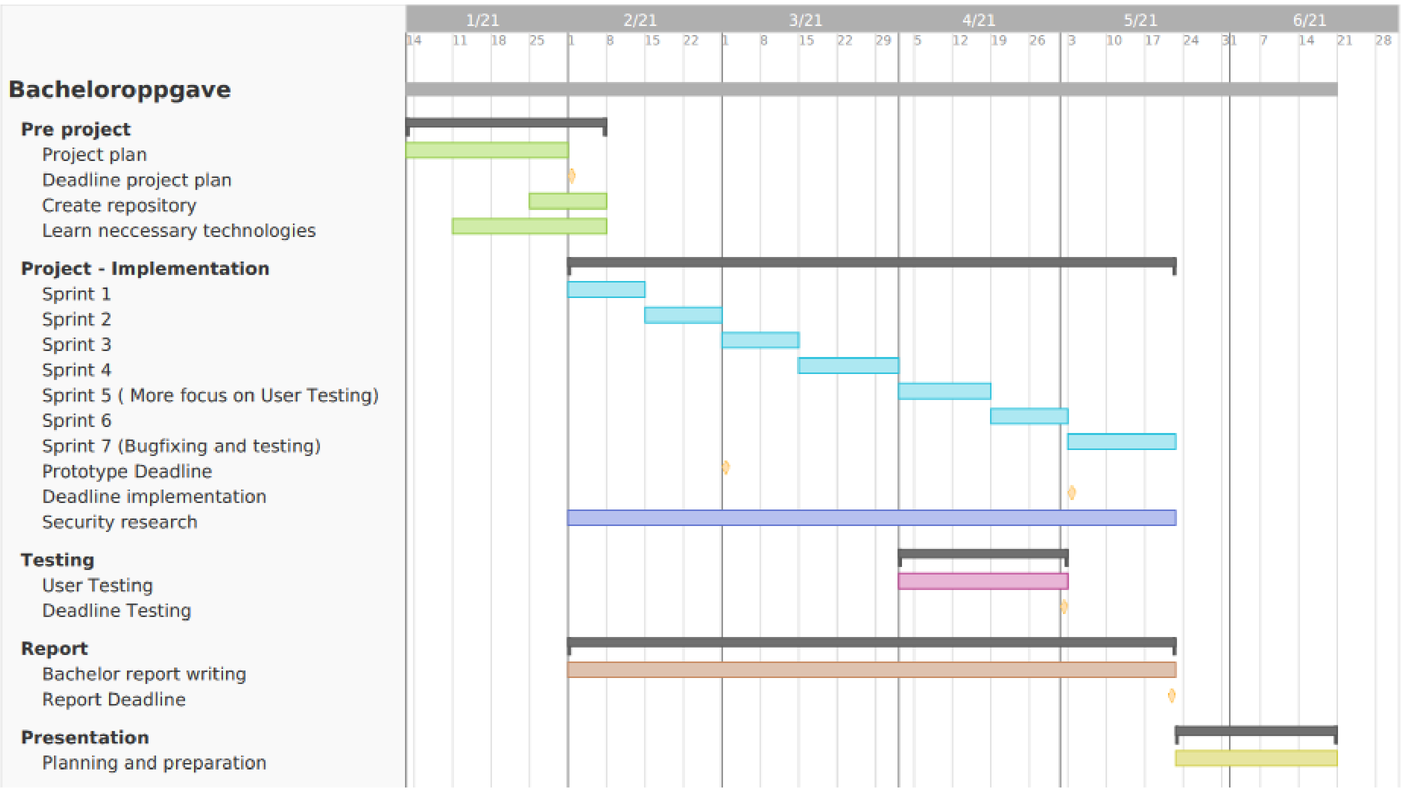
Description of risk	Mitigation	Level of consequence	Level of probability
The game development in Progress goes slower than expected. This delays our work since we have less assets and game features to make a Lua integration with.	If this happens then we must still try to learn the technologies that have been given. We will do this by making a Lua integration with one of Viktor’s video games which he has made through his study.	high	1
Group members get infected with Covid-19 which results in less progress from our group.	If one of the group members gets infected then we reassign tasks for that person to someone else in the group which is healthy. If the infected group member is still able to do work, then they can do tasks which require less focus and effort while the more demanding tasks are assigned to group members which are able to work.	medium	2
Our client gets infected with Covid-19.	We will request another employee from Progress to be our new client temporarily. In addition, we will ask our supervisor about other possible solutions.	medium	1
Security exploits are found which can violate	We are going to mitigate this by sandboxing the Lua	high	3

the confidentiality, integrity or availability principles.	integration. By implementing this we can disable certain functions from the player in order to limit what they are able to do.		
A Group member loses interests and starts to contribute less to the project.	Through meetings we will let each member have the chance to discuss issues with the project and as a group we will work on fixing those issues.	medium	1
RuntimeZipPlugin is too slow to load in a mod zip file at runtime into Progress' game. A mod zip file is in this case a zip file containing scripts and/or pictures which alter the game.	We will discuss with our client about finding another solution. Another solution could be to decompress files and place them in the folders from Progress' game before it launches.	medium	1
Development stops because of internal conflicts in the group.	We are mitigating this by trying to be as honest as possible with each other. If one of us for example is unfocused or lacks motivation we then try to help each other out. In addition, we are planning to spend some time together outside of bachelor project work in order to get to know each other better. This will change our view from working with strangers from working with friends.	high	2
Client shows little interest or does not	We bring up the issue to the client and have a	medium	1

provide us with enough help or guidance throughout the project.	discussion to find a solution.		
One group member is unable to work resulting in losing valuable skill in the group.	As mentioned in the paragraph “generalists vs specialists” we will make sure that everybody has a level of competence in the areas we need in order to complete our tasks.	medium	1

Resource Plan

In the Gantt-chart below we’ve created a plan regarding how we would preferably delegate time and resources.



(* ”Necessary technologies” defined as Lua,Moonsharp,runtimeZip, Unity)

Milestones

Throughout the project we have milestones to complete on-time.

- 1st Milestone [February 1st]: **Deadline Project Plan**

- Planning phase ended, developing a prototype next.

- 2nd Milestone [March 1st]: Deadline Prototype at the end of February
  - Prototype completed, allows for user-testing to begin.

- 3rd Milestone [April 30th]: **Deadline Testing**
  - User testing complete.
  - Consider what changes have to be made from test-analysis.
  - Start final stages of development (bug fixes and alike).

- 4th Milestone [May 3rd]: **Deadline Implementation**
  - Product should be 100% complete by this point.
  - Moving focus on-to finishing our report.

- 5th Milestone [May 20th]: **Deadline report**

NTNU
Norwegian University of
Science and Technology

PROJECT AGREEMENT

between NTNU Faculty of Information Technology and Electrical Engineering (IE) at Gjøvik (education institution), and

Richard Barlow (employer), and

Oscar Vagle, Genti Dautaj, Viktor Jarl Palmason, Ole Kristian Lund Lysø (students)

The agreement specifies obligations of the contracting parties concerning the completion of the project and the rights to use the results that the project produces:

1. The student(s) shall complete the project in the period from 11.01.2021 to 15.06.2021.

The students shall in this period follow a set schedule where NTNU gives academic supervision. The employer contributes with project assistance as agreed upon at set times. The employer puts knowledge and materials at disposal necessary to complete the project. It is assumed that given problems in the project are adapted to a suitable level for the students' academic knowledge. It is the employer's duty to evaluate the project for free on enquiry from NTNU.

2. The costs of completion of the project are covered as follows:
- Employer covers completion of the project such as materials, phone/fax, travelling and necessary accommodation on places far from NTNU. Students cover the expenses for printing and completion of the written assignment of the project.
- The right of ownership to potential prototypes falls to those who have paid the components and materials and so on used to make the prototype. If it is necessary with larger or specific investments to complete the project, it has to be made an own agreement between parties about potential cost allocation and right of ownership.

3. NTNU is no guarantor that what employer has ordered works after intentions, nor that the project will be completed. The project must be considered as an exam related assignment that will be evaluated by lecturer/supervisor and examiner. Nevertheless it is an

obligation for the performer of the project to complete it according to specifications, function level and times as agreed upon.

4. All passed assignments will be registered and published in NTNU Open, which is NTNUs open archive.

This depends on that the students sign a separate agreement where they give the library rights to make their main project available both on print and on Internet (ck. The Copyright Act). Employer and supervisor accept this kind of disclosure when they sign this project agreement, and they must possibly give a written message to students and head of Department if they during the project period change view on this kind of disclosure.

The total assignment with drawings, models and apparatus as well as program listing, source codes and so on included as a part of or as an appendix to the assignment, is handed over as a copy to NTNU who free of charge can use it in lessons and in research purpose. The assignment or appendix cannot be used by NTNU for other purposes, and will not be handed over to an outsider without an agreement with the rest of the parties in this agreement. This applies as well to companies where employees at NTNU and/or students have interests.

5. The assignment's specifications and results can be used by the employer's own work. If the student(s) in its assignment or while working with it, makes a patentable invention, relations between employer and student(s) applies as described in Act respecting the right to employees' inventions of 17th of April 1970, §§ 4-10.

6. Beyond the publishing mentioned in item 4, the student(s) have no right to publish his/hers/theirs assignment, fully or partly or as a part of another work, without consensus from the employer. Equivalent consent must be made between student(s) and lecturer/supervisor regarding the material placed at disposal by the lecturer/supervisor.

7. The students shall hand in the assignment with attachments electronic (PDF) in NTNU's digital exam system. In addition the students shall hand in a copy to the employer.

8. This agreement is drawn up with one copy to each party. On behalf of NTNU it is the head of the Department/Group that approves the agreement.

9. In each case it is possible to enter separate agreement between employer, student(s) and NTNU who closer regulate conditions regarding issues such as ownership, further use, confidentiality, cost coverage, and economic utilization of the results.

If employer and student(s) wish an additional or new agreement, this will occur without NTNU as a partner.

10. When NTNU also act as employer, NTNU accede to the agreement both as education institution and as employer.

11. Possible disagreements concerning understanding of this agreement are solved by negotiations between the parties. If consensus is not achieved, the parties agree that the disagreement is solved by arbitration, according to provision in Civil Procedure Act of 13th of August 1915, no 6, chapter 32.

12. Participants by project implementation:

NTNUs supervisor (name): Tom Røise

Employers contact person (name): Richard Barlow

Student(s) (signature): Genti Dautaj date 15.01.2021
Viktor Jarl Palmason date 15.01.2021 Oscar Vagle date 15.01.2021
Ole Kristian Lund Lysø date 15.01.2021

DocuSigned by:
Richard J. Barlow
5BD2C2D109DB4F7...

Employer (signature): ____date *1/29/2021*
____

The Project Agreement is to be handed in in digital version in Blackboard. Digital approval by head of the Department/Group.

If a paper version of the Agreement is needed, is must be handed in at the Department in additional.

Head of Department/Group (signature): ___________________________________ date ________