TABLE TOPPINGS – ENGINE FOR ANY TABLETOP GAME SYSTEM.

Miguel Bermudo Bayo
Estudiante de Máster, diseño y programación de videojuegos.
Nuevas tecnologías e investigación

Main professor – Jordi Duch Gavaldà
Tenured professors – Joan Arnedo Moreno, Helio Tejero Navarro.

02/01/2022

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Table Toppings - engine for any tabletop game system.*
Nombre del autor:	*Miguel Bermudo Bayo*
Nombre del consultor/a:	*Jordi Duch Gavaldà*
Nombre del PRA:	*Joan Arnedo Moreno*
Fecha de entrega (mm/aaaa):	01/2022
Titulación:	*Máster en diseño y programación de videojuegos.*
Área del Trabajo Final:	*Nuevas tecnologías e investigación*
Idioma del trabajo:	*Inglés*
Palabras clave	*Server, Tabletop, Engine*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.*

Este trabajo se centra en intentar ofrecer una solución tridimensional y moderna a los jugadores de rol de mesa, así como al mismo tiempo a los desarrolladores de elementos 3D que desean compartir su talento con los fans del género.

Hay multitud de sistemas para poder jugar una infinidad de juegos de rol, generalmente son herramientas diseñadas con un sistema en mente y con la compatibilidad en segundo plano.

Por este motivo se plantea TableToppings, esta herramienta aspira a ofrecer mediante arquitectura cliente-servidor, un entorno de juego tridimensional para aumentar la inmersión de los jugadores en sus partidas y customizar de cualquier manera que se desee la experiencia de juego.

La herramienta integra un creador de escenas donde se pueden crear, terrenos con diferentes elevaciones, añadir objetos obtenidos de la pagina web, pintar texturas sobre el terreno, etc.

Asimismo, también se ofrece la posibilidad de Self-Hosting, donde será el usuario final el que ofrezca su potencia de computación a sus compañeros y de esta manera podrán jugar todos conectados entre sí.

Usando el lenguaje GO orientado al bajo nivel y el multihilo se consigue una buena eficiencia para mantener un buen numero de equipos conectados sin una sobrecarga masiva del equipo en cuestión.

Además, al ser Unity la plataforma elegida para el desarrollo se puede conseguir la compatibilidad de múltiples plataformas para que cualquier persona con cualquier dispositivo pueda jugar con sus amig@s.

Abstract (in English, 250 words or less):

This works main focus is to offer a modern a tridimensional solution to tabletop gaming, as well as to offer 3D developers the chance to share their talents with the tabletop community as a whole.

Multitude of systems already exist but they do lack the power of 3D environments like what unity offers as well as being incompatible for mobile devices. some of them do offer their communities the chance to build upon what they have already developed, but the assets for these types of games are so widespread (as they are basically pngs) that you do not see any market for them.

Therefore, I envisioned TableToppings, this tool aspires to offer a 3D tabletop gaming environment via a server-client architecture, to help players get more immersed in their stories. You can customize the games feel and experience in any way you want by using our API (in development). functions that try to cover basically all the functionality you could want.

The engine integrates a scene creator where you can develop either 2D (WIP) or 3d environments to play your games. For now, it's planned to support terrain detailing, texture painting, terrain elevation and a Liquid layer, the latter being already fully supported, the roadmap also contemplates volumetric fog.

There is also the fact that the system has been developed with headless hosting in mind, this means that any player with a decent enough computer could be able to host several clients without much issue. But in case they want 100% uptime, they could just use our own platform for hosting (this is also in our roadmap)

I've decided to use GO which is a low-level oriented language focused on threading, to be able to increase the performance of the server, by using GO every client has a dedicated TCP socket for themselves, making it very optimized for large number of them.

Finally, since the main client is developed in Unity, we can develop a multi-platform system, so a scenario where players could be connected via IOS, Andoroid, Windows, Linux, and Mac, could be a reality.

Table of Contents.

Table of figures

1. Introduction

1.1 Context and reasons for development

The project aims to offer Tabletop players a platform to play their favorite systems with as many tools as possible to create a visually attractive experience, moves towards the classical ways where the minis reigned supreme, but now instead of buying very expensive 3D printed minis (figurines that represent characters in a game), you can but way cheaper animated 3D models for anything that an artist has chosen to create for our platform.

No tool aims to create a community of collaboration in this sense, even though some do support the addition of features to them, there isn't any incentive for the users to post their own creations monetary or otherwise.

Due to the recent increase in popularity of the genre there are a lot of people with very creative minds and interests that would probably like to be able to make monetary gain in return for their skills, and players always love to have very pretty visuals.

1.2 Objectives

Since this project is meant to be an actual application, the scope of it goes much further than a typical TFM, so most of the objectives are incomplete, they are the following

- Develop an application that can be used as a server for the players, it must provide:
    - File synchronization – complete
    - Player authorization – complete
    - Object/Player Scene Synchronization – incomplete
    - Scene Permission handling – incomplete
    - Audio Synchronization – incomplete
    - Animation Synchronization – in progress.
    - Etc…
- Develop a client that can interpret the server's operation and offer good performance, as well as the following
    - File synchronization with server – complete.
    - Scene creation – In progress
    - Menus from SXML & XCSS templates – In progress.
    - User Objects Synchronization – In progress
    - Reflection Lua compiled functions – incomplete.
    - Etc…
- Develop a website that allows user to sell the assets they've developed for the game.
- Publish some instructional tutorials on how to add compatibility.

1.3 Focus and methodology followed

We've followed the modern approach to software development called agile, of course since I am just one person developing the software right now, it is obvious that we cannot perform the daily and weekly meetings, so we just followed the Kanban board organizational ideology.

The focus for the project was to evaluate feasibility. I aimed to create at least the ability to send messages to and from the server, which was accomplished, and to create a synced play scene and a map creation scene, both of them ended up incomplete due to time constraints but they are likely to be easily doable.

1.4 Planning
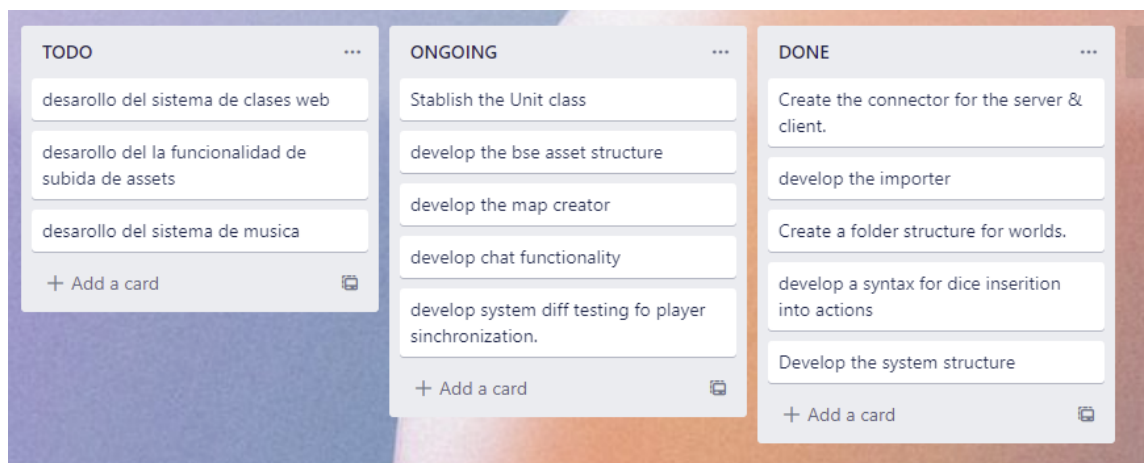
Figure 1: trello board final state.

We've used a simple trello board to enumerate the tasks that comprise the project, this is the current state of that board, as you can see most of the tasks are either completed or ongoing. This is because as we mentioned we are evaluating if the project is viable, hence the need to cover as much ground as possible. The left-out tasks are the one that I've previously made in other projects, so I know they are viable or in the case of the music system we didn't get around to doing.

The first PEC was idea development, so we didn't need the Kanban board by then, no tasks on them were completed.

The focus of the second PEC was to create the server connectivity, just to accomplish the connection between the GO server and the unity client, which was satisfactory completed.

The third PEC was the one in which we developed the play scene, in this one we tried to add some flair to the game, so we transitioned to URP and went for some environmental effects with underwater post processing effects as well as external ones, it is planned to add volumetric fog to the game as well.

The last PEC was the one where we added some polish to the game, we developed the file evaluation system where the client tells the server which files, they have, and the server updates them accordingly.

It was also the PEC where we made a very basic map creator that showcases the ideas, we do have for it.

1.5 Summary of resulting products

We ended up with a functional server that can send and receive any kind of message and process them accordingly, and an equally functional receiving client that can communicate with the server to ask for information and files.

The client also has the initial map creation tooling as well as a basic and for now disconnected play scene that host the concepts for the future connected scene.

1.6 Summary of topics.

Data structure – Here we will cover how the information related to the game will be stored, it is important since we have to make it easy for users to add knowledge to the game in an understandable way.

Server – We will be covering the underlying server functionality and how it was implemented to work in tandem with clients.

Client – How the client was made and all the functions that are there and still need to be implemented into it.

Costs – The estimated costs of the project if it went into full development.

User Manual- the steps to take to be able to launch the game successfully.

Conclusions – What can be improved and what we have learned while doing the project1.7

Glossary of terms.

TCP - "Transmission Control Protocol" – "TCP enables two hosts to establish a connection and exchange streams of data. TCP guarantees delivery of data and also guarantees that packets will be delivered in the same order in which they were sent" *

Asset – refers to an object that can be used in any way by a client, it can be a 3D model, a music track, some json files, etc.

System – refers to a series of rules that players generally have to follow to play a certain setting for their game, the items enemies, etc that they could have.

Tabletop – the term that indicates a type of game that is generally played on a table with a group of people, this is a very broad term that encapsulates a lot of different games.

Handlers – A type of function meant to control a very specific subset of the whole process, for example connectivity between server and client.

Messages – the way in which we refer to the information being sent through the TCP channel.

*Stouffer, K., Stouffer, K., Pillitteri, V., Lightman, S., Abrams, M. and Hahn, A., 2015. *Guide to industrial control systems (ICS) security*. USA: U.S. Department of Commerce, p.135.

2. Data structure

This is essentially how we organize the files inside the server, we chose Json as the data storage method, this is because we expect the users to create content for the game and due to this, we cannot offer a convoluted experience further than reading plain and structured text files, which json offers very conveniently.

2.1. Assets

The assets are in essence, every PNG, JSON, FBX, etc we are going to end up using in the game. This is what users will be able to create and sell to other users in our marketplace.

We will have an info.json file which duty is to assign an ID to every asset that there is in those folders as well as get their path from that position (to reduce server strain), this is necessary as to coordinate multiple clients we need to know which id does the item have in order to interact with it later on.

The assets also contain effects that can be used by different systems by referencing them. In their actions, the thing about this is that some systems may require certain asset packs to function and for that reason we might need to re-thing how these action-linked systems work.

2.2. Systems

Also, in the Assets folder we can find the Systems, this are the large set of rules and structures that define the different games you could play, they are the backbone of the whole project.

These files contain different class structures and any number of objects that adhere to these classes, we will have to define a very minimal number of mandatory classes for a system to work, we need a class for the **actions** which are what user-controlled agent can perform, an **actor** class, that defines the information that needs to be held by the actor object in order for the game to display it and modify it, and finally there is the **effects** class, that indicate possible states that the user can find themselves in, for example laying down or being stronger in some regard, these effects are planned to modify certain action results.

System must also have pages definitions for any of the assets they provide if they are modifiable, pages use the XML extension of UIElements provided by unity to create their menus dynamically, so the users can tie function execution to buttons and use a special syntax "{{action.id.attribute}}" or similar to provide the user with interactivity.

It is also planned to include conditionals by using <% if condition %> and even some custom code support by adding a LUA language parser to the page templates.

2.3. Worlds

Worlds are high level concepts, they are basically containers for the adventure, they use a certain system to tell their story, a player can join a world by connecting to the IP of the deployed server that contains it and await the worlds activation.

On login if the player has administrative permissions, he would be able to start up the worlds he owns. Every world will have an information file that contains which assets they are using, who owns it, and which players are allowed to connect to those worlds.

2.4. Scenes

Scenes are what worlds are comprised of smaller units that have the map information that the owner has created as well as every object that must be synchronized with an id assigned to it.

They are what the players will be ultimately interacting with, they must also store every change made by the players when interacting with them.

They are the way to neatly store all of the games information and communicate it to the server, via the messages we have

3. Server

3.1. Configuration, authentication & connection establishment

The server operates via TCP socket and threads to handle them.
First, the server reads the configuration file and loads the user and world information intro a structure it can read.

Once this information has been gathered, we forward the port defined by the configuration file, for this we use UPnP a protocol made for easy port forwarding, there is a problem with this protocol though, it's generally not enabled by default in some older routers, for that we will investigate another way to establish the connection via UDP NAT overpass. Which is a technique that can forward data from every NAT on the way.

In case there is no way that we can overcome this problem is where the dedicated servers come into play. We will provide dedicated servers for our users, for a price they will always have their server active.

After the forwarding, we can create a TCP socket that listens for incoming connections on the now open port, this listener runs on a while loop and Pass the forward to the connection handler as a gorrutine.

GOrrutines are a way the language handles threads. By typing go in front of a certain function this function is now threaded and safe, so it can communicate in any way with the other variables and don't risk concurrency in any case.

3.2. Handlers and Sub-Handler

These functions are the ones that deal with incoming data.
The **main handler** simply receives the connection from the listener and then starts the authentication subroutine, this routine waits for a message from the client containing the login information, evaluates it and determines whether it should or not be logged in.

It would be probably smart to add a limiter to the amount of tries a user could make before breaking the connection to the server.

Whenever login is successful, we save the users information in a structure as well as the connection so we can link petitions to the server to the corresponding users.

At this point we enter the infinite loop part of the application where we read the messages that come from the client and process them, go handles TCP sockets pretty badly so we just empty the buffer until we reach the end of it and then process that, the intended use is that you define a delimiter for the messages which is a terrible idea since you don't know what your messages contain in most instances.

If we received a stop flag, we will logout the user and interrupt the connection.

In any other case we will forward the message to the **message Handler.** This handler interprets the bytes from the message and sends them downwards into the corresponding subhandler, at the moment the only existing subhandler is the connection subhandler since those are the only operations made by the client right now.

These subhandler manages the following petitions currently:
- Sending world information to the client (so the admin can activate the desired world).
- Loading the world in the server and keeping it open for update petitions to be sent.
- File Synchronization (this will be explained later)

3.3. Message structure

We have been referencing messages, but what are they exactly and how are they handled?

It's nothing more than a constant stream of bytes, but internally they have a structure, so we know which part of the code to send them. This structure follows a pattern of the type: "[0XX, 1XX, 2XX…]" where the first number references the handler that will take care of it and the other 2 numbers are the actions that will have to take place in that handler to process the message correctly

The rest of the message chages depending on the operation to perform, for example, *"001:accepted;ID,type,username"* is the message that the server sends back to the client as an indication for an accepted login. Containing the user's information as part of the answer.

3.4. File synchronization

File synchronization is by far the hardest operation that has been implemented and forced a change in the way the server handles messages.

Since file are no more than a lot of bytes in a certain order we needed to send massive amounts of information and beforehand we didn't consider what would happen if we went over the buffers capacity.

So, to send files we ended up setting up a 16k buffer that right padded the messages and frontloaded a bit of information per file (which ended up being applied to every single message)

So, the new structure of any message now is:
"paddingbytes:msgID:msgpart[t];001:accepted;ID,type,username[…]"

Where *paddingbytes* is the number of bytes that must be deleted from the message before processing, *msgID* is the unique id of the message being sent, *msgpart* indicates which part of the message this is and the t indicates if this is the last part of the message.

The message is followed by as many padding bytes as necessary to fill the 16384-byte array indicated by the […]

The handling of the file reconstruction and message processing is made simple thanks to this structure but determining the necessities of this system was a tremendously painful undertaking.

4. Client

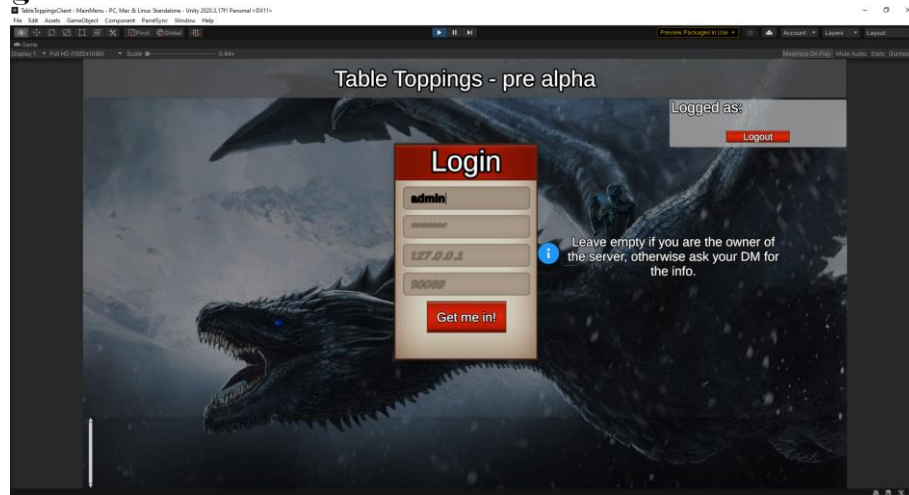## 4.1. Login Screen and Main Menu

Figure 2: Login Screen

The first screen we see when logging into the game, it has the necessary fields to be able to connect to the server and has the added functionality that it can change the focus of the elements with the tab button, a nice detail that surely the users can appreciate.
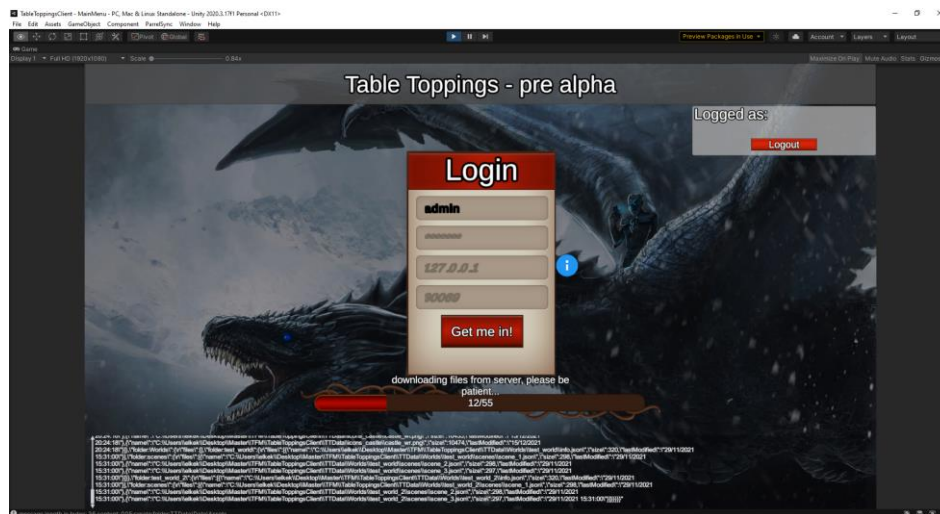

Figure 3: downloading files.

When we successfully login we proceed to see how the file download is going via a simple loading bar with some flairs. Whenever the download is complete, we will write the files in our local memory, and we can see the progress in the same progress bar.


Figure 4: logged user.

We can also see the user that we are logged as and its role whenever the file synchronization is finished.

Figure 5: Launch a world menu.

The client will also display to the admin what world are currently available to them to launch this is only displayed for the admin, the clients will be prompted to wait for the world to be selected by the admin if there isn't one already loaded or will be sent to the play scene.

Figure 6: Options menu.
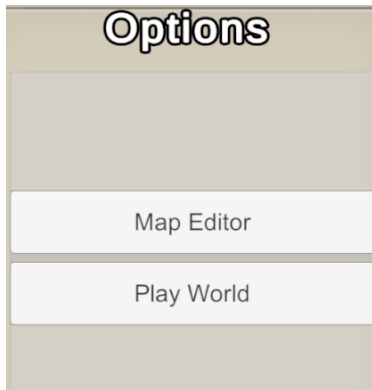
After selecting which world to load, the options to either go to the map editor or go to the play world scene will be displayed for the admin only, the players would be directly redirected to the Play Scene.

4.2. File synchronization

For the client-side file synchronization and message reception I have determined that the best way to do it is to store the messages into a double map structure of the type: "*Dictionary<int, Dictionary<int, byte[]>>*"
This structure first integer key is corresponding to the message unique ID, a way to retrieve it when it's time to rebuild it.

The second key is to order the messages via the order number sent by the server. the expected question about this would be why not use a list or an array? well since we don't know how many messages are going to arrive nor the order of arrival we would need to dynamically adjust the list for every message we get, which is not a cheap operation, and adding keys to the dictionary is simpler, because you can then retrieve them in order by simply calling then by their index.

Before we can store the messages though, we need to interpret them to prune the padding bytes as well as determine their ID and order number.

4.3. Map creation

Map creation is at the core of these kinds of games, since players are ultimately the people who would judge the game for how it looks it's a crucial part of them
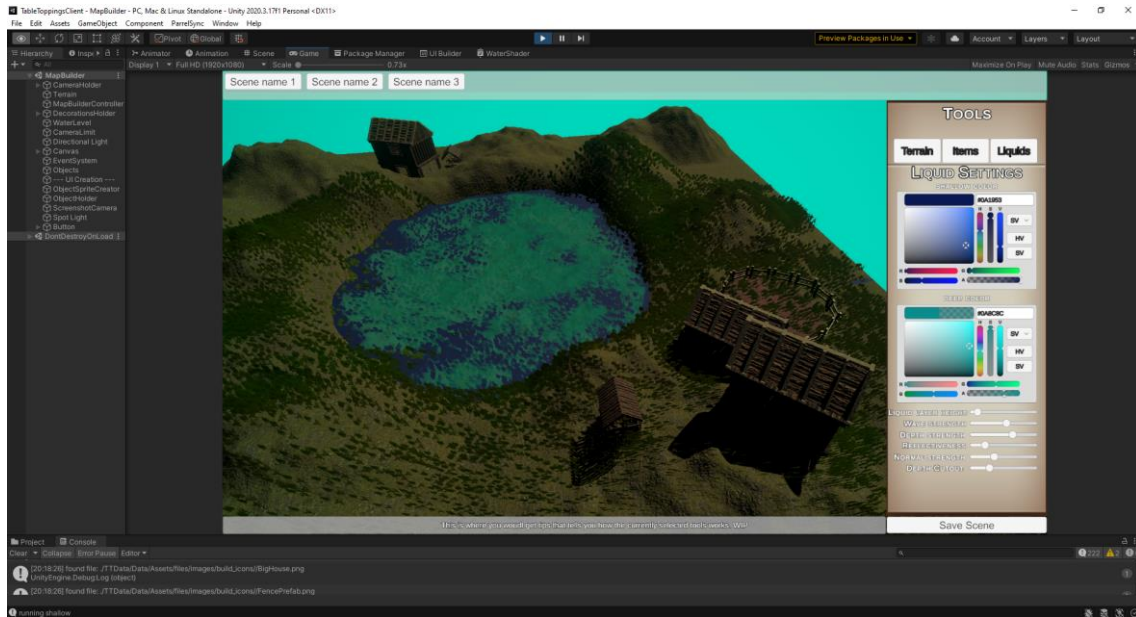

Figure 7:Map creator Scene

Figure 8: Tools for map creation tabs

The tools we offer for map creation for now are, terrain manipulation (global) and fluid layer manipulation. We've also setup the basis for item and detail placement. We are planning to add terrain operations that manipulate the grid on a local scale, for the players to be able to create the map they want.

And we need to create a grid placement option for elements like floors, walls etc.

4.3.1. Terrain manipulation

For now, we can adjust the maximum height of the map (terrain wise), the width and the length of it and the minimum height.

This last operation raises every terrain to that size, so you need to be careful when using it.

*Figure 9:terrain global tools*

4.3.2. Liquid layer

Every terrain comes with a liquid layer integrated in it at height zero, if so desired you can raise the terrain to be over it and not use it but in case you do want to, you can make use the menu to change the deep and shallow colors, as well as the height, the waviness, the depth sharpness, and the normal strength, also you can set how much height is it required for the deep end to begin.

Figure 10:liquid layer visible

Figure 11: liquid layer settings

4.3.3. Detailing, texturing and Item placement.

Figure 12: texture and detail painting menu.
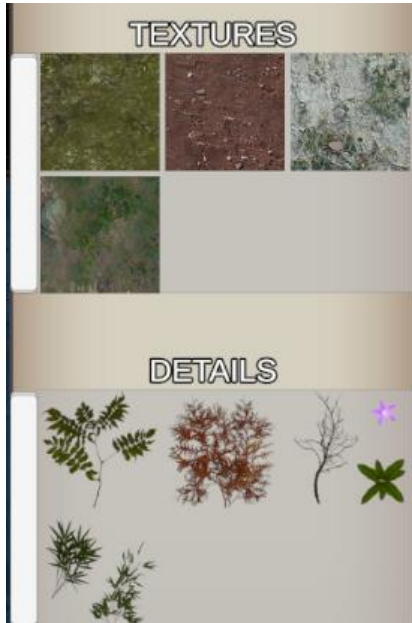
Texture painting is a very important aspect since the floor will need to be different kinds of terrains for different settings.

Also, you need to distinguish mountains and peaks, right now there is map generation tool we implemented for the play scene that paints the terrain based on elevation and we do want to import that tool to the map creator scene, but for now it is only meant to do painting by hand.

Details make use of the terrain integration for them to spawn them as a cutout instead of a mesh as it is way less computationally expensive to deform them if the are planes.

4.4 Play Scene

Figure 13: all units in the scene

4.4.1. Map Loading

Maps are loaded via a heightmap image, and the texturing right now happens based on the height of the terrain the information is transferred into the splatmap for that terrain which can be stored as an image.

We also load a NavMesh for the units to use for land-based movement. And a grid that adapts to the terrain marking all 1-unit squares that are on that terrain.

14

4.4.1. Unit control

All units in the scene have an outline that shows blue if you own it and yellow if you don't. this is so you can see them even if they are obstructed by something since the outline renders in front of everything else as seen in figure 13.

When clicked on, the units become selected and now glow green, in this mode you can control where to move them as well as raise them up or lower them back down with spacebar and shift respectively.

You can also multi-select them by pressing control while clicking on a second one. While in multi select mode a red ball will appear on scene, this shows where the center of the selection is, and when clicking on a new destination, the units will move towards it based on the position from the ball to the new position.

4.4.2. Dice rolling & chat

Since we are playing tabletop games, they all come down to dice rolling, for this reason we have decided to let players link buttons in the modular menus with rolling functionality. We've made a parser for dice rolls and determine the results by checking the actual physical state of the dice.
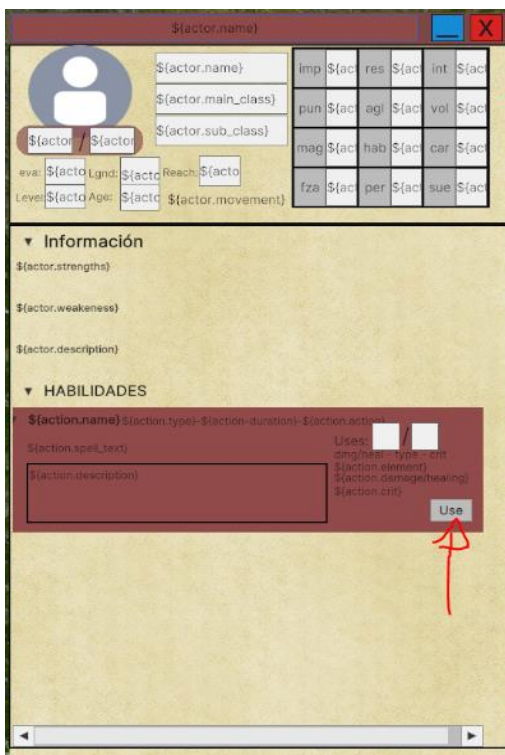

In this example we've designed a basic character sheet that contains the information for a default system called role up, it is all the information an actor from that system can have and we've linked this action with rolling 12 6-sided die.
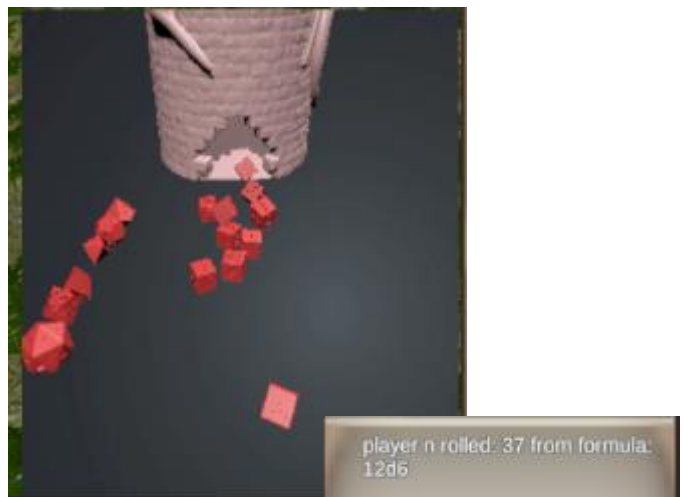

Figure 14: Character information sheet.

Figure 15:Dice being rolled and the result being added to chat.

15

4.4.3. Post processing and miscellaneous.

We've setup some embellishing elements for the game, like the use of URPS post processing volumes and camera effects to make the game shine more. Bloom, camera color adjustments, vignettes, contrasts, etc.

We are also planning on adding volumetric lighting and fog but since it does not come out of the box with URP as it does with HDRP we've decided not to do so right now.

We also plan to include lighting control for the players, this is so they can make day and night scenes, caves etc without much hassle.

5. Costs

To continue the implementation of this project it would most likely require a team, dedicated servers, for web and player server hosting, here we will be the costs for a month and a year for each of the elements.

Elements	Monthly cost (€)	Yearly cost (€)
Web design	600-6000 (depending on development stage)	23400
Game programming	12000-24000 (depending on the team's size)	144000
Game art	10000 (flat)	10000
Server hardware	15000 (flat)	15000
Server maintenance (maintainer + electricity)	1500	18000
Domain name	-	15

For a total of 185.415€ yearly + 25.000€ flat.

6. User Manual

To launch the project, you need to execute the dndserver.exe, it will tell you if it could successfully open the ports and you will then be able to connect to it.

To open the client, you do it like every other unity game, and then login with the admin credentials which are just admin in the username slot and nothing else.
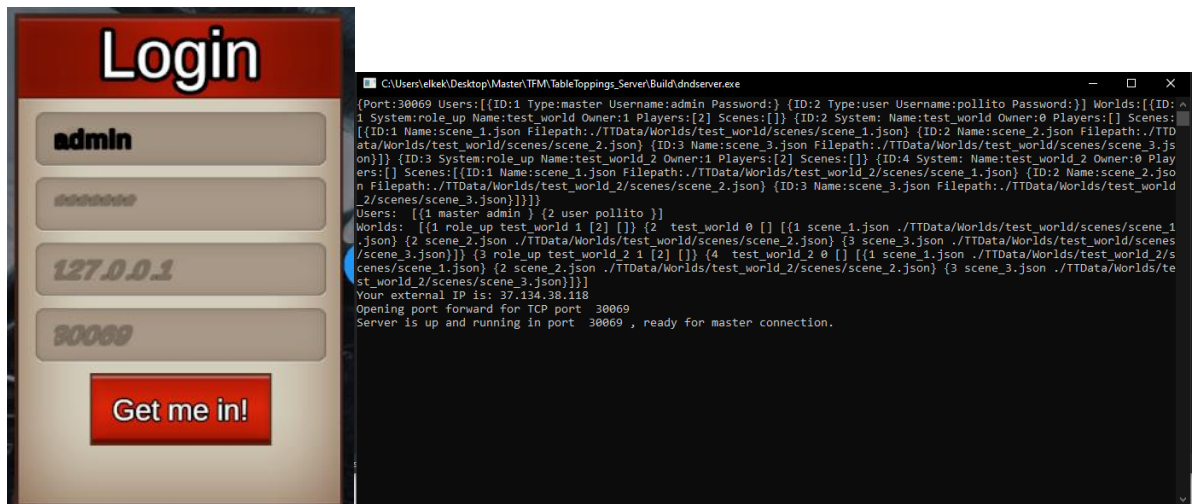

Figure 16: the client credentials and the server ready for connection.

From then on just click on any loaded world and either "play scene" to view the play scene contents or the "Map builder" to access the early map builder tools.

7. Conclusions

Lessons learned: These types of projects are not adequate for an academic environment as they have no real end in sight, lots of shortcuts had to be taken to accommodate the type of environment that a project like this simply cannot afford.

Objectives reached: The project reached a state of minimum functional project, meaning it shows basically every system that the game will incorporate but almost none of them are in a completed state, some of them aren't even functional and just there as placeholders.

Nonetheless I am satisfied with the project result since I was able to evaluate the feasibility of what I am trying to accomplish for the future.

Planification: the planification was established looking forwards to the future, for this reason it was basically impossible to complete, this lead to a mid-project direction change where we weren't building from the ground up and just focusing on showing every idea the project had in a minimal state.

The methodology was followed quite loosely since I did time crunches instead of incremental week by week progress, since I do have a full time job that reduces compatibility a significant amount with this kind of improvement.

Future work: There is a lot of future work for this project, for one designing launching the product, the development of the website, the non-unity linked assets, a character builder and a long Etc.