Table of contents

1.Introduction

1.1 Summary of the game

1.2 Where the game originated from

1.3 Purpose of the project

1.3.1 Objectives of the game.

2.Technologies that were used.

2.1 Game Development

2.2 What is a game engine.

2.3 Why Unity

2.3.1 Mirror

2.4 What are Databases and why Firebase

3.The Game

3.1 The player experience

3.1.1 Use case Diagram

3.1.2 Scenario Diagram

3.2 The parts of the game

3.2.1 Registration Scene

3.2.2 Main menu Scene

3.2.3 Battlegrounds Scene

3.3 The ruleset and the mechanics of the Duels

3.3.1 The Cards

3.3.2 The H.U.D. and the Drop zones

3.3.3 Mulligan

3.3.4 Turns

3.3.5 Adding clarity

3.3.6 Animations and sounds

4.The code that was used.

4.1 Registering and Signing In

4.2 Saving and Loading Decks

4.3 UIManager

4.4 The Cards

4.4.1 The Card’s movement

4.5 GameManager

4.6 Network Manager

5. Network infrastructure

5.1 Authentication

5.2 Realtime Database

5.2.1The structure

6. Conclusion

6.1 Problems faced during development

06.2 Improvements for the future

6.3 Personal fulfillment

Ευχαριστιες

Abstract

1. Introduction

1.1Summary of the Game

1.2 Where the game originated from

1.3 Purpose of the project

The point of my thesis is to build a 2d multiplayer card game in which players create their custom decks from an array of available cards and duel each other.

1.3.1 Objectives of the game.

The players can save their custom built decks and each player can have up to 8 custom decks. Before starting a duel each player has to select a single deck, from their available custom decks, to play with. Each duel takes place in separate instances and each instance has exactly two players. One of whom, acts as a host, meaning that takes both the role of the server and the client. Then he gives the second player his ip address in order for him to connect to the particular instance. The second player acts sole as a client and connects to the ip address that the first player gave him. Afterwards the game com mences.

The game is separated in phases. The first phase is called the mulligan phase. Each player draws 5 cards and decides if he wants to keep them as their starting hand or if they want to discard (mulligan) them, meaning that they are placed back to the deck and the deck is then shuffled. Then he draws 5 new cards which he must keep. After both players decided if they want to keep or mulligan their starting hand, the next phase begins. The main gameplay loop.

The main gameplay loop is separated by turns. Each player has a turn in which he has to take some actions. After one player is done with his turn he gives the other player the initiative, and waits for him to pass back the initiative. So the players change turns back and forth until the game ends. When a player takes the initiative, either from being first or after the other player has ended his turn, he automatically refills his mana crystals (Hereinafter referred to as mana, for simplicity’s sake). Mana is the resource each player has in order to play cards from their hands to the battleground. Each card has a mana cost in order to be played and after a player plays a card, the mana cost of the said card is withdrawn from the player’s mana reserves. If a card has a greater cost than the player’s current mana reserves then he cannot play this card. The mana for each player starts from 0 and each time that a player has the initiative he refills the previous mana and he gets +1 up to a maximum of 10. For example, *Player A has 2 mana and he plays a card that costs 1 mana. The 1 mana is withdrawn from the players 2 mana and his new mana is 1. Then he proceeds to give Player B the initiative. After player B finishes with his turn and give the initiative back to player A, Player’s A mana refills (so he goes back to 2 mana) and gains a +1 mana so it becomes 3. Each time its Players A turn he gets one more mana until he gets to 10, where his mana just refills each round.*

2. Technologies that were used.

2.1 Game Development

Game development, as the name suggests, is the art of creating games. Some of the development stages into game development are the design of said game, the production of it and finally its publication.

In the design stage, the game designers will plan out how the game will play. That includes every decision from the start to the end of the game. Moreover, they will try to answer questions such as for what audience the game appeals to and how long will it take for it to be actually developed. Finally, they will come up with some type of prototype of the game, to see if it works in practice. Typically, that takes about 1/5th of the total production time.

Production is the stage that takes the longest. In this stage the developers create the characters, the environment, the dialogs, the sounds, the effects, etc. They constantly try out everything to see if it fits together. A big chunk of their work usually does not even make it to the end, as they deem it does not fit the gameplay. They constantly change things up for it to feel more “fluid”.

Lastly, it’s the publication of the game. In this stage the game’s content is ready (or at the very least “almost” ready). Usually the AAA companies (companies with huge budget and number of employees) have a publisher. That means a company that specializes in the marketing and distribution in exchange for a portion of the profits. If the company is small and cannot afford a publisher, then they publish the game themselves.

2.2 What is a game engine?

A game engine is the core software necessary for a game to properly run[1]. Game engines usually provide a selection of development tools and reusable components that are needed for a game to be created. These engines constantly take care of things such as rendering the graphics, animations, artificial intelligence and many more things. Because its something so vital, its important for game developments to choose the right game engine that suits their needs. That’s why big developer studios create their own game engines from scratch. That enables them to choose exactly what features the engine will contain and how it will work. That’s a long and expensive process however. Smaller studios that don’t have the budget and/or time to create their own game engine, use pre-existing ones. Such engines are Unity and Unreal Engine.

2.3 Why Unity?

Unity is a cross-platform game engine[2]. It can be used to create 2D, 3D, virtual reality and augmented reality games such as simulations. Some of its benefits are its user friendly, it can support a vast array of platforms from desktop to mobiles, it has a great community with extra tools that are great and lastly its free.

Unreal Engine, another game engine such as Unity, also provides many benefits. Such benefits are, it can handle a wide variety of games, better inherently shooting and first person mechanics overall. It also support cross-platform games like Unity does and lastly it is also free.

For the purposes of this project I went with Unity as I am making a 2D card game and not a shooter. Moreover the friendliness of Unity’s UI gives her an advantage.

2.3.1 What is Mirror

Mirror is a high level Networking library for Unity, compatible with different low level Transports[3]. At its core, Mirror is a system that adds multiplayer capabilities for Unity games. It is built of the lower level transport real-time communication layer and handles many of the common tasks that are required for multiplayer games. Mirror is focused on ease of use and iterative development and provides useful features straightway.

2.4 What are Databases and why Firebase.

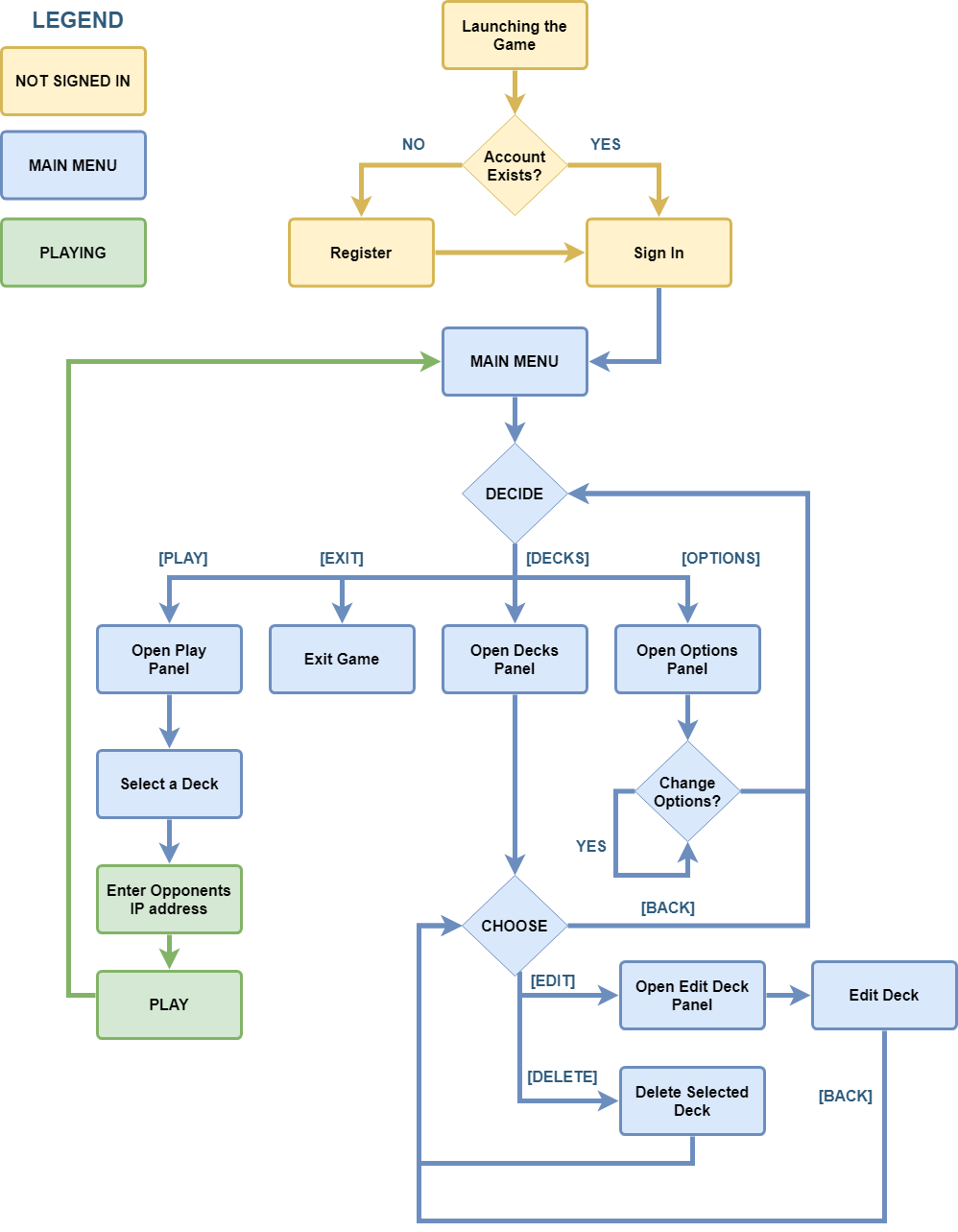
A database is an organized collection of data, generally stored and accessed electronically from a computer system[4].

A cloud database is a database that typically runs on a cloud computing platform and access to the database is provided as-a-service. There are two common deployment models: users can run databases on the cloud independently, using a virtual machine image, or they can purchase access to a database service, maintained by a cloud database provider. Database services take care of scalability and high availability of the database. Database services make the underlying software-stack transparent to the user[5]. Firebase’s Realtime Database is a service exactly like that.

Firebase is a service that provides developers with a vast array of tools. These tools provide solutions to common game development “issues” that occur during development stages, thus giving the developers the freedom to focus on the app experience itself. Tools that are included in the firebase ecosystem are cover areas of analytics, authentication, databases configuration, file storage etc.[6]

3 The Game

3.1 The Player Experience



3.2 The parts of the game

3.2.1 The registration scene

The game consists of three primary scenes. The first is the registration scene where the user would have to enter his credentials. In order to access the game an account is necessary. That’s because the game is online and each player can create different decks. That means we would have to do two essential things. Firstly we would have to store each player’s decks and secondly these decks must be accessible at any time across any platform or system. Therefore we solve both of these issues by implementing an account system.

When first opening the game, the player is presented with a choice to either sign in with his credentials or to register a new account. If the player chooses to register, then an email is required as well as a username and a password. After the player selects register he would have to go back to the sign in screen and connect using his credentials. Only his email and password are required for him to connect. After entering his credentials he will enter the main menu of the game.

3.2.2 The main menu scene

The main menu is the second of the three primary scenes in our project. The main menu consists of three buttons. The last of them is the Options button where a player can change the games volume.

The second button is the Decks button, where a player can create, edit or delete a deck. When the player opens the deck window he will be presented with all of his created decks up to 8. Those decks will be loaded directly from the database (which I will analyze in the next chapter). Selecting a deck will let the player either delete or edit the particular deck. If the player chooses to edit the deck or create a new one then the Deck Creation panel opens. There, the game we pre-load the already selected deck if the player chose the edit option. Otherwise, if the player had selected to create a new deck, then the game wouldn’t have pre-loaded a deck. In any case, the player can drag and drop any card he wishes to use in his deck.

The Deck Creation window is simple to use. On the right side the player is presented with every possible card the game has to offer, ordered by value of cost. On the left side the player can see in a more minimized view, his current deck. On the upper left corner displays the card deck’s name.

Each deck must have exactly 30 cards. A card count number exists on the bottom right corner to display this. The player can add or remove cards by simply dragging the card he wishes to add to his deck from the right side to the left and vice versa for removing them. Once the player is ready to save the deck he simply exits the window and a pop up message informs him that his deck will be saved.

Last but not least, the player can click on the play button and a Deck Selection window opens up. There the player must choose the deck he will be playing with and once he has, a play button will lighten up.

3.2.3 The battlegrounds scene

This is arguable the most important scene of the game. Here the players duel each other.

Firstly, a player takes the role of the host and the other player takes the role of a client. The client has to enter the IP address of the host. The client will connect and stay connected with him as long as the host exists. When both players hit the Ready button the game will begin. The players duel each other until either of their health points reaches 0. When this happens the player with the 0 hit points loses and the other player wins. After they finish their duel the players get returned to the main menu.

3.3 The ruleset and the mechanics of the Duels

In this part of the thesis, I will explain all the different mechanics and rules that take place during a duel between players.

3.3.1 The cards

The most important aspect in a card game, are of course the cards themselves. Each card displays on the upper left corner their cost in mana crystals, in the bottom left corner their attack points and in the bottom right corner their life points. The card’s name can also be seen in the top area and in the center area an artwork of the card exists. Bellow the artwork there is some space for a description (although I’m not using it yet). I purchased the cards artwork from the asset store, but I made the card borders and the icons myself.

3.3.2 The H.U.D. and the Drop zones

During the duel, the players H.U.D. (Heads Up Display) gets enabled. Each player can see their avatar on the bottom left corner and their opponent’s avatar on the upper right corner. Bellow each player’s avatar resides their remaining life HP (Hit Points). The player’s Mana Crystals are shown on the bottom right corner. Each time a player gets a turn their Mana Crystals are refunded, and their cap is increased by 1 up to 10. That means Players will start with 1 mana in turn 1 and have 10 mana by turn 10. Every player can see their deck above their Mana Crystals. Players can also see how many cards their opponent holds in their hand, albeit they cannot see what cards they are.

Drop zones are the area’s a player (or the game) can drop cards. Each player has their hand area and their respective drop zone. A player cannot drop a card in the opponents drop zone or hand.

3.3.3 Mulligan

When the duel begins, each player will draw 5 random cards from their deck. The players can either choose to keep them or to Mulligan. Mulligan means that they will discard their 5 cards back in their deck and draw 5 new random cards. Each player can only Mulligan once and only at that stage, meaning that after they mulligan they have to keep them.

3.3.4 Turns

For a player to take any action, it has to be his turn. Always the player that acts as the host starts first. At the start of each turn, the player’s Mana Crystals refill and they draw a card from their deck. During their turns player can drop cards to the battlefield and/or attack enemy cards or even the enemy player itself.

For a player to drop a new card to the battlefield, they must have equal or more mana crystals than the card they want to drop costs. Players gain Mana Crystals passively as the game progresses, up to 10. When the player drops a card, the cost of the card is withdrawn from their available Mana Crystals.

After a card is dropped the player must give his opponent the chance to “react”. That means that he cannot attack with his card unless a turn is passed since he played it. Once a turn is passed, the card is ready to attack. The player has to select it and then select a target.

A target can be anything from the opponents dropped cards to the opponent avatar itself.

Once a player is happy with his actions, he must press the “End Turn” button to pass the turn to the other player.

The game goes back and forth until a player loses all of his life Hit Points.

3.3.5 Adding Clarity

Visual clarity is one of the most important things a game developer can add to their game. Its imperative for a player to be able to see all of his available actions in a single glance. In order to help the players keep track which of their cards can attack I added a visual indication around them.

If a card is available to attack, it will be surrounded by green flames. When the player selects a card to attack with, the fire of that card will turn blue. Subsequently as the player hover with his cursor over the enemy cards a red flame appears to surround them indicating that they are a target. Moreover, the cursor turns from a hand to a sword.

3.3.6 Animations and sounds

To expand in the mater of clarity, animation and sound cues are also used to indicate some specific situations. In particular, when a card attacks another card an animation is played letting both players know what’s happening. Additionally when a card is destroyed it emits a special “dissolving” shader designating its destruction.

In the scope of sound, ….

4.The code implementation and the Classes that I used.

In this sector of my thesis, I will analyze and explain in depth the code that I used. I will delve deeper in order to explain the reasons behind my decisions.

4.1 Registering and Signing In

Let us start from the beginning.

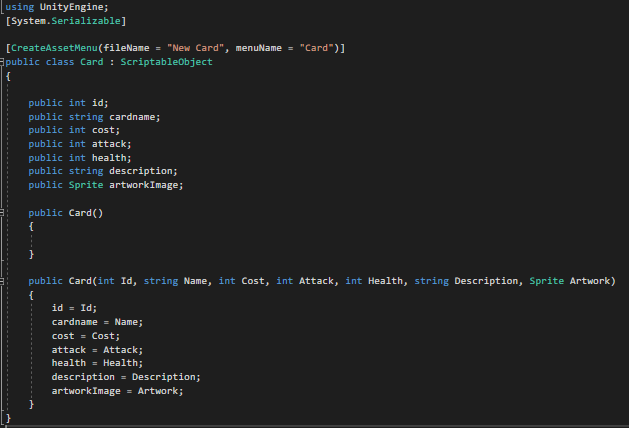
4.2 Saving and Loading Decks

4.3 UIManager

4.4 Scriptale Objects and Cards.

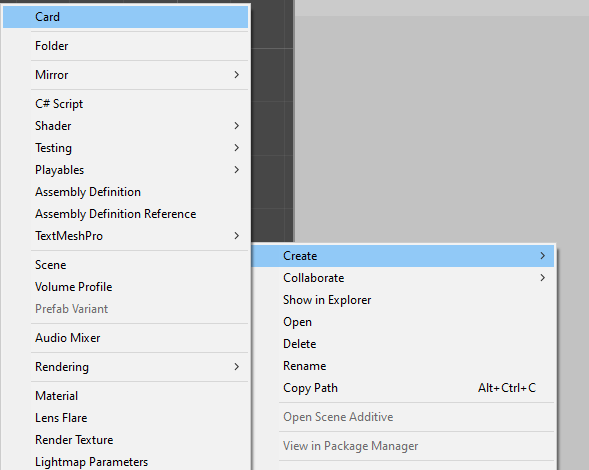
4.4.1 Cards Instantiation

It’s vital for the game to be scalable. For that reason I created the class “Card” which is inherited to all of our cards.

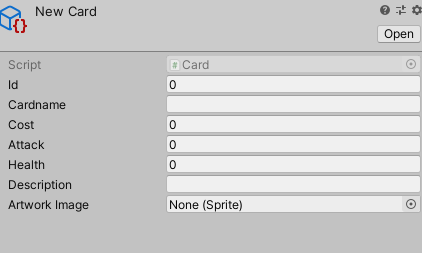


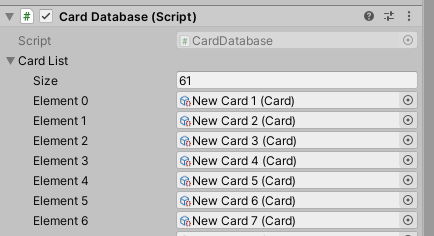
Each card saves some values like an id, its name, the cost, etc.

I added an option on the editor to create a new card, simply by right clicking.

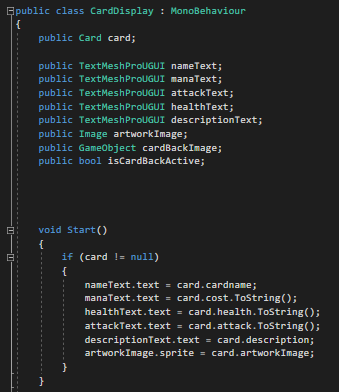


This option creates a scriptable object that represents our card. Next, I can edit the values that we want the card to have.



Lastly I drag the scriptable object to a card database that contains a list with all the cards I want to use. 

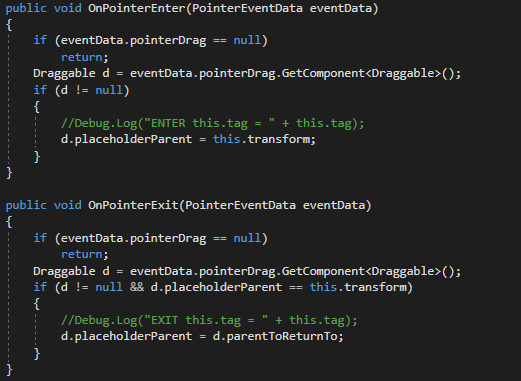
Now I just instantiate a card prefab over and over again and I simply assign it a scriptable object. Then I added a CardDisplay class which is responsible to load the values from the scriptable object to the actuall prefab.



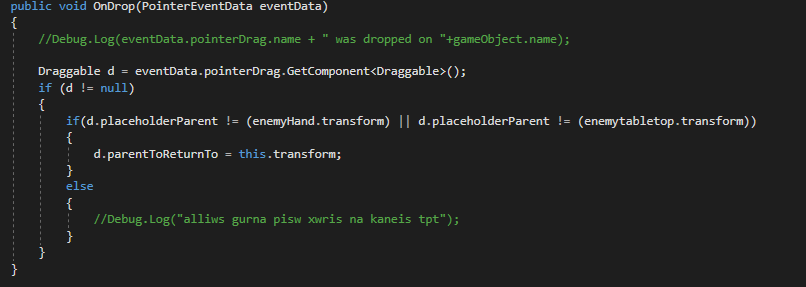
4.4.2 DropZones

Every Drop zone area in the game has a DropZone script attached to is. This scripts contains some event handlers that trigger when the mouse pointer has enter or exited an area. When this happens they check if we drag something (like a card) and if we do, they set their parent to the specific area.

Then as the cards exit the area, they set the card’s parent back to the previous area they were.



If however, the player drops the card inside the area, then they set the previous parent to the zone. (their current parent already changed when they entered the area, with the OnPointerEnter method)



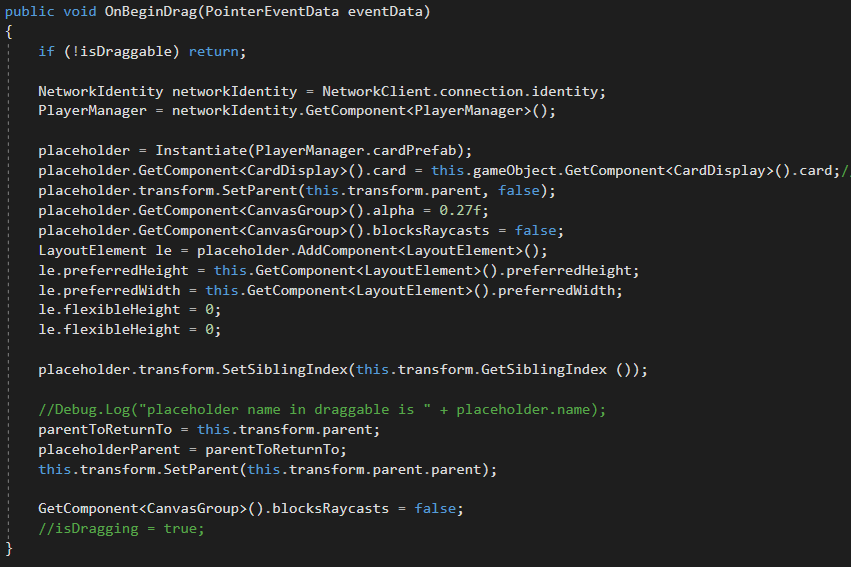
4.4.3 Card movement

Every card has a “draggable” script which allows the player to move the card. Inside the script there are drag handlers that trigger when the user begins to move a card, when it moves and when he ends to move it.

The OnBeginDrag takes as a parameter the pointer event that generates when the player clicks on a card. We get the network identity of the player (so we know who tries to move the card) and we store it. After that we instantiate the placeholder which is a copy of the card, and shows the player where his card is going to be placed.

I also store the container of this card as parentToReturn to, incase an invalid move takes place, so I will know where to return the card to. And I also save the current parent in placeholderParent.

Last but not least, while I move the card around I want my mouse to be able to RayCast what’s bellow the card. However, my card blocks any ray cast from the mouse cursor. So I temporarily set the blocksRaycasts to false.

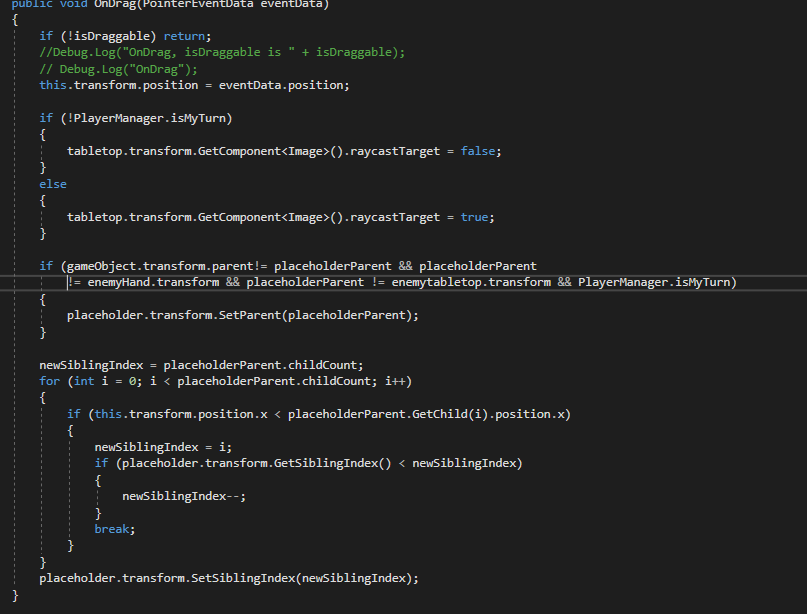


When the player finally starts to drag the card around the OnDrag method activates.

First we need to check if the card is draggable and if it’s the players turn to play.

There are some conditions in place that restrain the players from doing illegal moves such as dropping the cards on the opponent’s area etc. If all the conditions are true, then I set the placeholder parent to be equal to the latest placeholderParent value, which was changed within our DropZone script.

Lastly, I keep track the position of the card in the area it starts, so if it returns to the previous parent it will be placed in the same spot. I do that, by comparing the x value of the placeholder with the x value of the other cards and determining its sibling index.

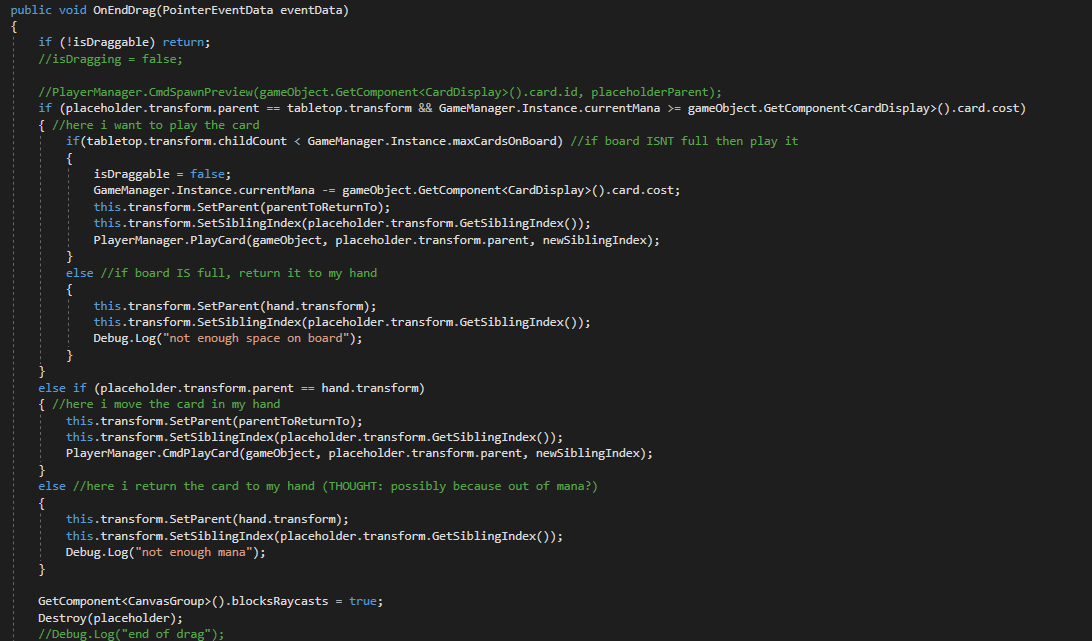


The OnEndDrag method is called when the player let go of the card.

Firstly I check if the card is played in the proper area, if the player has the required mana, and if the board is not full. Then if all these requirements are true, I withdraw the cost from the players mana pool and I change the parent to the board. Then with the command “PlayerManager.PlayCard” I send a command to the server that I played a card. With the command I also send some parameters as to what’s the card I played and where I played it too. If the board is full, then I place the card back to the hand.

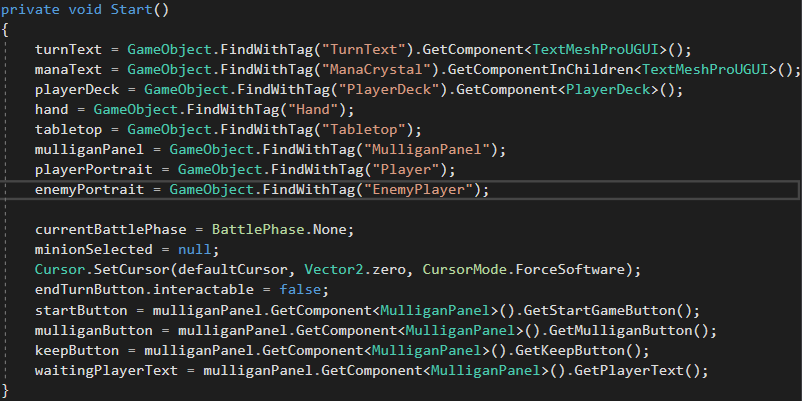
It should be noted that the player can change the card order of his hand anytime he wishes. That’s why I also check if the destination area is the hand. If it is I let him reorder the cards as he wishes.

Lastly, I set the blockRaycast back to true and I destroy the placeholder.



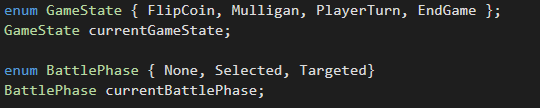
4.4 GameManager

The GameManager is a class responsible for the main functionality in the game. The GameManager acts like the “dealer” in a game of cards. Basically it takes care of providing the players with cards, managing their resources like mana, keeping track the phase the game is in, and changing the turns between the players. One of the main things that the GameManager does, is the initialization of some classes.



Most of the things in our scene like the texts and the drop zones are spawned dynamically when the players connect to each other. For that reason I cannot simply assign the variables from the editor. So, I assigned every important thing with the appropriate tag and used the FindWithTag command to assign them correctly. After that I initialized some of the variables to their first value.

I used two enum variables, one that keeps tracks of the phase each player is and the second keeps track of the battle phase the attacking player is.

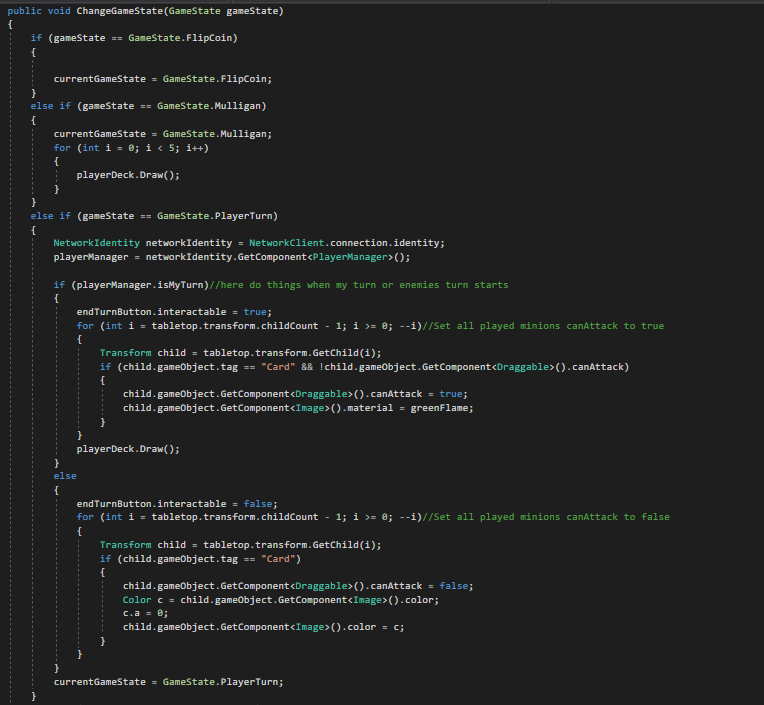


In the ChangeGameState method I pass as a parameter the gamestate enum. Than an if argument checks the stage of the game.

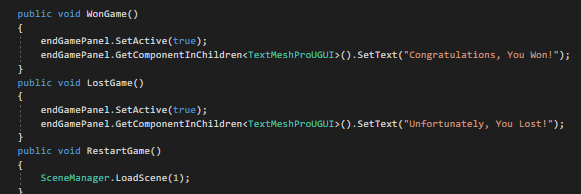
If the players are in the mulligan state then they draw 5 cards each from their decks.

If the players are in the PlayerTurn stage, a nested if checks if it’s the players turn. If it is, we set the canAttack of our minions that we have played to true. We also enable the green flames around them to indicate it to the player. Finally we draw a card.

If its not our Turn we set the canAttack of our minions to false and also we disable the green flames by changing the alpha channel of the color to 0.



IWe also manage the end game messages from the GameManager.



If the player winswe win or loses Ilose we display the appropriate message to the screen. IWe also have a RestartGame method that loads the main menu scene when the player clickswe click the Restart button.

4.5 PlayerManagerNetwork Manager

The script that is responsible for the communications between the players is the PlayerManager. Mirror needs a server in order for the players to be able to connect and play together. However, in Host mode a player can act both as a client and a server. The drawback of this method is that a client has to connect to another client and sometimes that is difficult due to security reasons like firewalls. So the client that acts as the host must port forward their rooter and allow incoming connections to their computer. A dedicated server would solve this problem, but it requires a monthly fee, and in the scope of this thesis I would like to keep the project free.

4.5.1 Authority

Authority is a way of deciding who owns an object and has control over it. Authority is divided in two parts. Server authority and Client authority.

Server authority means that the server has control of an object. Server has authority over an object by default. This means the server would manage and control of all collectible items, moving platforms, NPCs, and any other networked objects that aren't the player.

Client authority means that the client has control of an object.

When a client has authority over an object it means that they can call Commands and that the object will automatically be destroyed when the client disconnects.

Even if a client has authority over an object the server still controls SyncVar and control other serialization features. A component will need to use a Commands to update the state on the server in order for it to sync to other clients. [7]

4.5.2 Attributes

Networking attributes are added to member functions of NetworkBehaviour scripts, to make them run on either the client or server.

These attributes can be used for Unity game loop methods like Start or Update, as well as other implemented methods. [8]

[Server] and [Client] attributes means that the following method can only be called by the server or a client respectively.

[Command] is called from a client to run on the server. For example when a client wants to deal some damage on the opponents cards, he sends a command to the server with the damage he deals and then the server takes the damage value and applies it to the other client.

[TargetRPC] and [ClientRPC] indicate that method uses a Remote Procedure Call (RPC).

RPC’s can only be called by the server.

4.5.3 Remote Procedure Calls

RPCs are split in two parts, ClientRPC and TargetRPC. Their functionality is the same, but to whom they address is not.

ClientRpc calls are sent from objects on the server to objects on clients. Since the server has authority, then there no security issues with server objects being able to send these calls. Basically, when server takes an action and wants all the clients to know about it then an RPC is called. For example, if a player’s life points changes and we want every player that is connected to receive this change, we can do a ClientRPC call and pass the new life points.

TargetRPC is similar but it can only target one specific client. We call this when we don’t want every client to know a change.

The arguments passed to commands and ClientRpc calls are serialized and sent over the network. [9]

Πινακας περιεχομενων

Λιστα εικονων

Λιστα πινακων

Εισαγωγη

Η ενότητα αυτή περιλαμβάνει μια εισαγωγή στο περιεχόμενο και σκοπό της πτυχιακής εργασίας. Για παράδειγμα:

Περιλιψη

Η ενότητα αυτή περιλαμβάνει μια περίληψη της πτυχιακής εργασίας.

Κινητρο διεξαγωγης

Η ενότητα αυτή περιγράφει το κίνητρο για την εκπόνηση της πτυχιακή εργασίας, καθώς και τους στόχους που επιτεύχθηκαν

Bibliography

1

2 <https://en.wikipedia.org/wiki/Unity_(game_engine)>

3 <https://mirror-networking.gitbook.io/docs/general>

4 <https://en.wikipedia.org/wiki/Database>

5 <https://en.wikipedia.org/wiki/Cloud_database>

6 <https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0>

7 <https://mirror-networking.gitbook.io/docs/guides/authority>

8 <https://mirror-networking.gitbook.io/docs/guides/attributes>

9 <https://mirror-networking.gitbook.io/docs/guides/communications/remote-actions>