Generic smartphone game controller

Bachelor thesis

Degree programme:	Computer Science
Author:	Quentin Flückiger
Thesis advisor:	Prof. Marcus Hudritsch
Expert:	Eric Dubuis
Date:	11 March 2019

Management Summary

Our first job was to learn about the architecture and possibility of Networking within the Unity3D API as it was the one chosen because of its “simplicity” and its cost.

Content

1 Introduction

Games, something that has traveled throughout history and is part of many cultures. They are even found in nature, as most animals learn by playing when they are newly born. It is a very old pedagogic way to teach, one that doesn't even need a common language. Humanity pushed this process even further by developing board games, games that can sharpen the mind or which are there only to spend some time. The oldest board game, supposed to be the oldest, is the "Senet". A game from Ancient Egypt created approximatively in 3'500 BCE. The Senet isn't played anymore, as the rules were lost at some point in history, but another very old game is still played in our time. The "Go", a game who saw birth in Ancient China as far back as 2'000 BCE and is still widely played and popular. The more commonly known game of "Chess" is only approximatively 1'400 years old.

With the technology produced by humankind, it allowed the human to develop other ways to play games, on the computer, console and now smartphone as well, but we never stopped producing board games and inventing new one either. We even created Pen and Paper games, which are stories told by one person where others, the players, interact by giving vocal input. And the popularity of games has been increasing incredibly, even more with a smartphone as they allow people to play it very easily. Just a download and then hop it's in the pocket, they can play it everywhere. "There are approximately 2.2 billion gamers in the world. Out of the estimated 7.6 billion people living on earth, as of July 2018, that means almost a third of people on this planet are gamers." (Gaimin, 2018) Those numbers are crazy, and one can see the market opportunity in this field. But what hasn't been made yet? What could be something new? Something interesting? Isn't it possible to take benefits from both board games and video games?

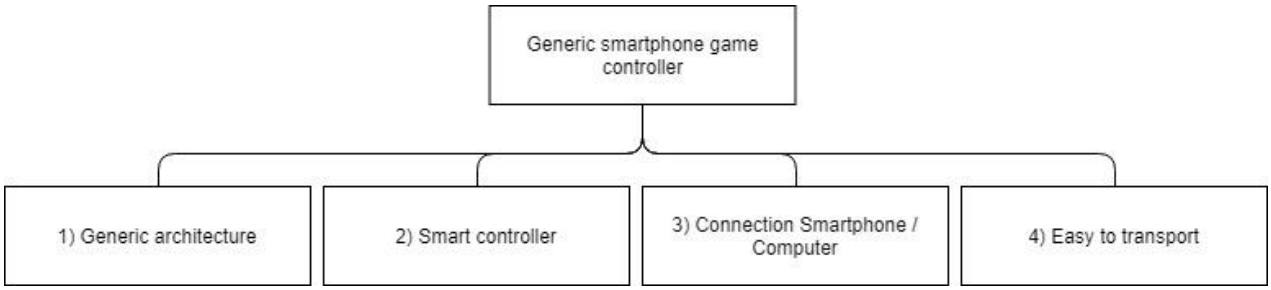
Those were the questions we asked our self and that led to the idea of this project.

Project Management

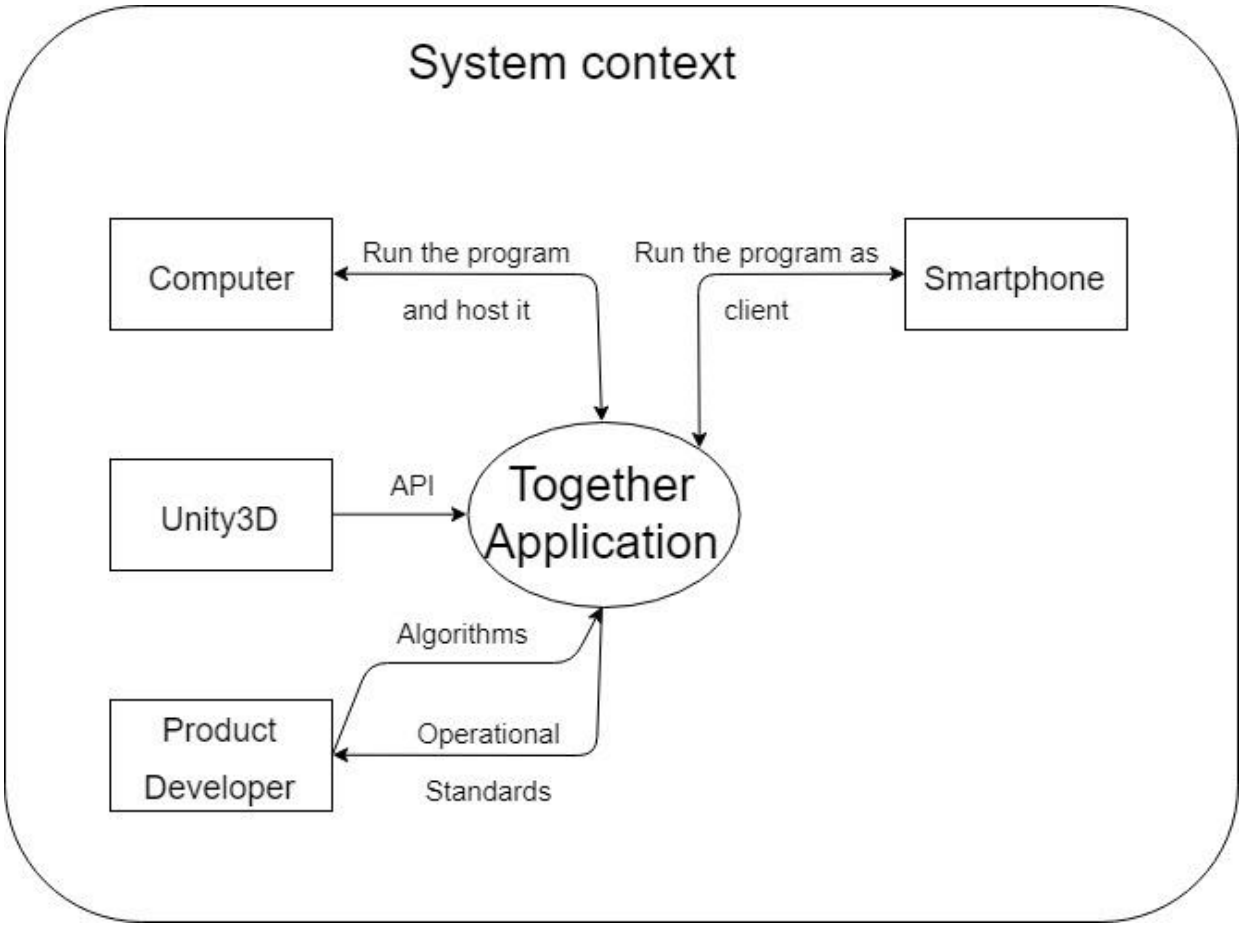
1.1 Vision

Creating a generic gaming software/platform/library which will use a computer/tv screen as the view and smartphone as a controller to allow people to play together at video games without having to carry, or buy, a heavy controller. Most of the people have a smartphone which can be used as a substitute and can even be made “smart”. This library will allow games to be easily built with its component (network, controller, …).

1.2 Goals



1.3 System Context



1.4 Requirements

1.4.1 Legend and additional information:

Table columns definition

- Nr := identification number
- P := Priority (3 levels expressed in with numbers: 1;2;3)
- V := Variability (3 levels expressed in with numbers: 1;2;3)
- C := Complexity (3 levels expressed in with numbers: 1;2;3)
- R := Risk calculated from {P;V;C} (numerical value)
- Status := planned (will be done) / stand by (nice to have) / done
- Goals := goals reference(s)

1.4.2 Functional Requirements

Number		Description	Status	P	V	C	R	Goals
1		Connection						
	1.1	Connection from smartphone to computer.	Approved	3	1	2	1.83	3
2		Build generic						
	2.1	The system must be designed to allow for optimal extensibility and reusability.	Approved	3	3	3	3	1
3		Extendable						
	3.1	Dynamic HUB which accept new games	Approved	2	1	3	2.17	1
4		Smartphone possibility						
	4.1	Exploit the components of the smartphone to deepen the singularity of the concept.	Approved	3	1	1	1.33	2
5		User Interface (UI)						
	5.1	Design and implement a UI that allows the user to interact with the system on smartphone using touch technology.	Approved	3	1	1	1.33	2
	5.2	Design and implement a UI that allows the user to interact with the system on computer.	Approved	3	1	1	1.33	-

1.4.3 Technical Requirements

Number	Description	Status	P	V	C	R	Goals
1	Software						

	1.1	C#	Approved	3	-	-	-	-
	1.2	Unity3D 2018.3.7f1 or lower 2018 version.	Approved	3	-	-	-	-

1.4.4 Quality Requirements

Number		Description	Status	P	V	C	R	Goals
1		Platform compatibility						
	1.1	Compatibility with Android / Apple phones	Approved	2	1	2	1.83	-

2 Understanding Unity3D Network

The first step we took in our project was to decide which network technology we should use to achieve our goals. There were mainly two options, Photon and UNeT. As we never had any experience with Photon we decided to use UNeT. UNeT is the built-in technology from Unity, one can find the documentation and tutorials on the official Unity website. This technology is separated into two categories: High Level API (HLAPI) and Low Level API (LLAPI).

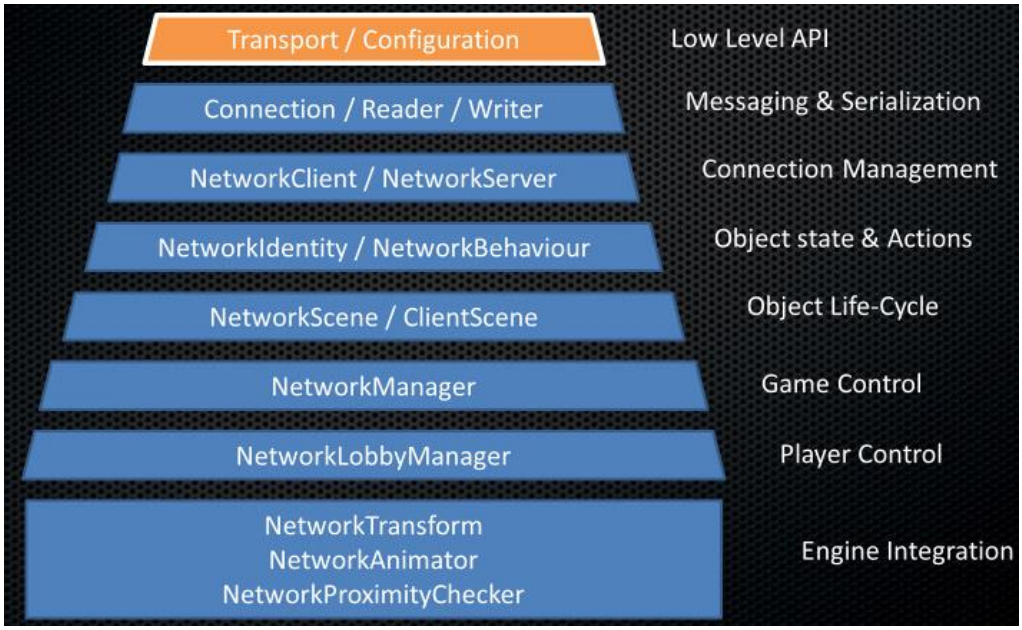


Figure 1: Unity3D Network Layers

The LLAPI, or Transport Layer API, lets developers built their own networking system. It is best used when we are already experienced in network programming and as well when we need a very specific network. As it requires to be built from the very base.
The HLAPI is built on top of this layer and others which adds functionality. It is a set of networking commands which can be found in the namespace: UnityEngine.Networking. It allows the developer the same ease of use as the rest of Unity's component system as they can be added the same way as another component would be.
Between the two possibilities, we choose to use the HLAPI because it would fasten the development of our software. If we were doing this project as our project two as well, we would have chosen the LLAPI as it would have suited us more to have our own made networking system with our own messages. Another reason why working with LLAPI would have been better is because of UNeT being deprecated in the near future and the only information Unity has been given is that the HLAPI would not be compatible but the new system would be close to the LLAPI allowing to go from the latest to the newest. But as we tried at the beginning of the project to develop a little example with the LLAPI we acknowledged that it would take us too much time.

For this example, we created two different projects, a server, and a client, established a connection between a given port and sent some messages from one to the other. Bellow, you will see the architecture of the initialization of the server.

```
public void Init()
{
        NetworkTransport.Init();

        ConnectionConfig cc = new ConnectionConfig();
        reliableChannel = cc.AddChannel(QosType.Reliable);

        HostTopology topo = new HostTopology(cc, MAX_USER);
```

```
        hostId = NetworkTransport.AddHost(topo, PORT, null);
        webHostId = NetworkTransport.AddWebsocketHost(topo, WEB_PORT, null);

        Debug.Log(string.Format("Opening connections on port {0} and web port {1}.", PORT,
        WEB_PORT));
        isStarted = true;
}
```

Afterward, on each update a method “UpdateMessagePump” would be called and its duty was to listen to any networking event and act accordingly.

```
switch (type)
{

            case NetworkEventType.Nothing:
                break;

            case NetworkEventType.ConnectEvent:
                        Debug.Log(string.Format("User {0} has connected through host {1}.",
      connectionId, recHostId));
                break;

            case NetworkEventType.DisconnectEvent:
                Debug.Log(string.Format("User {0} has disconnected.", connectionId));
                break;

            case NetworkEventType.DataEvent:
                BinaryFormatter formatter = new BinaryFormatter();
                MemoryStream ms = new MemoryStream(recBuffer);
                NetMsg msg = (NetMsg)formatter.Deserialize(ms);
                OnData(connectionId, channelId, recHostId, msg);
                break;

            default:
            case NetworkEventType.BroadcastEvent:
                Debug.Log("Unexpected network event type.");
                break;

}
```

We created our own network message abstract class called “NetMsg” with which we would create our personal messages such as LoginRequest. But as we would have had to build all of it, it would have been too time-consuming for this project, so we decided to go with the easier and faster to use HLAPI.

After choosing the technology we would be using for our project we started to browse forums, youtube videos, Udemy courses and documentation on the Unity website to acquire knowledge and example material which would be used later on in the project.

Following tutorials, we created a very simple 2D prototype using the network manager and its HUD. It was a very simple one where the player was a cube which could move around 2 axes and fire at other player or boxes. The health and transform position would be synchronized. At this stage, we could connect with the smartphone, but we would all see the same thing, be it on the computer screen or the smartphone one.

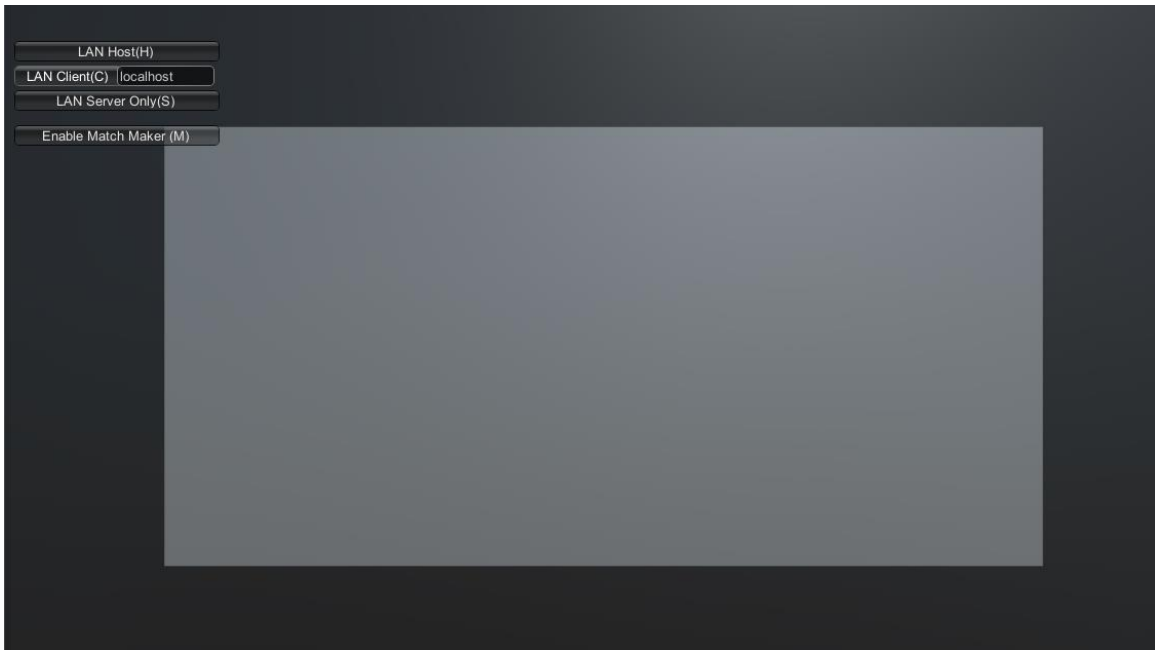


Figure 2: First try multiplayer lobby

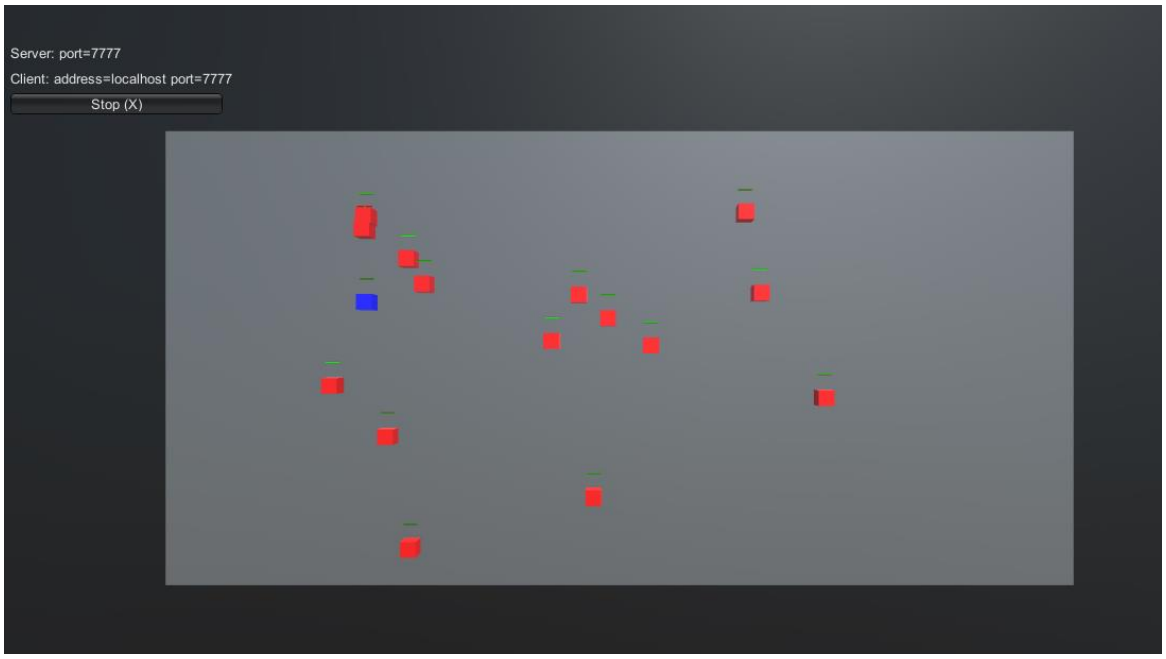


Figure 3: First try in game

As we started to understand how the HLAPI of Unity UNeT was working, or we thought, we decided to move onto the next step and implement our first game. A snake game.

3 The snake

There are multiple reasons why we chose a snake game as our first game. Firstly, because the logic of the game is simple. You are a snake and your possible actions are to move in three directions (four minus the current one), to eat an apple on collision or to die upon collision with anything else than an apple (be it walls, tails or another snake). Secondly, this game only needs four inputs, so the smartphone controller UI would be very simple as well. And the data sent with the network would be insignificant. For the sake of simplicity, we built the snake for computers, while having in mind the will to enhance it later for smartphones, just to be able to test it.

We started by looking at what had already been done as a multiplayer snake game in Unity and found a few ideas, resources from which we learned. We stumbled upon a very interesting example from Rober Esbjörnsson that we followed in the beginning and gave us the following results.

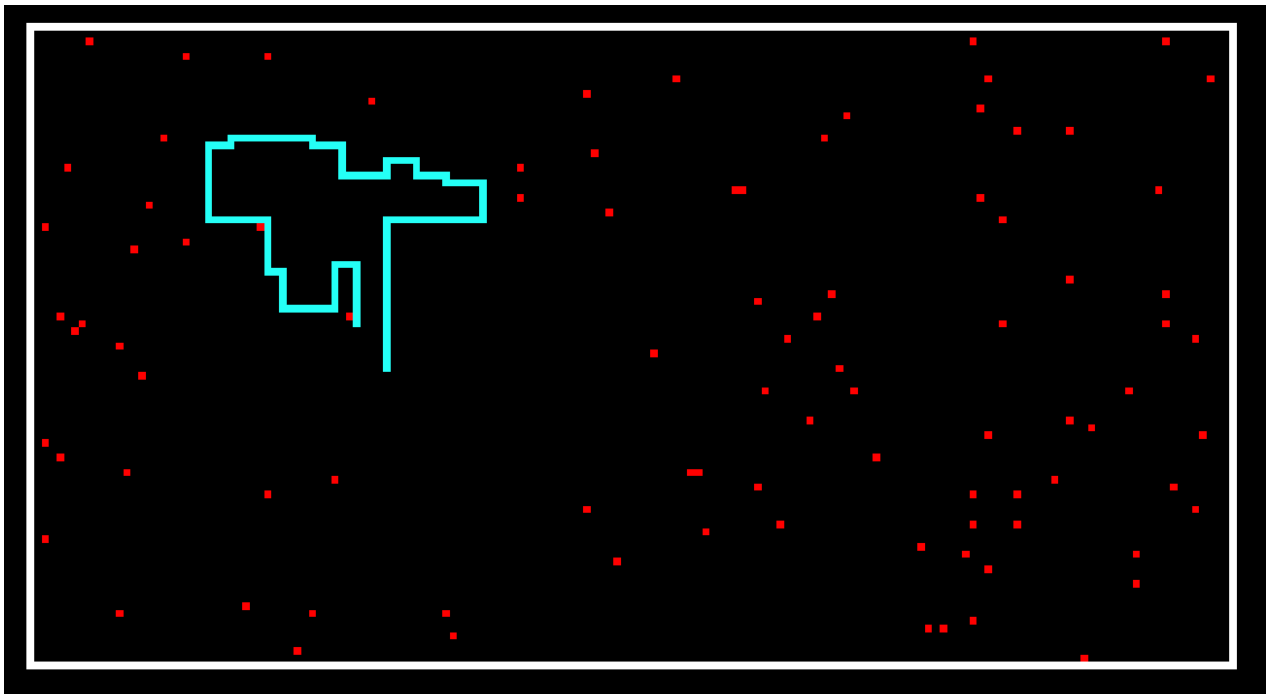


Figure 4: Esbjörnsson Snake

Thus, we continued in that direction and implemented the networking part of the existing project. We created a lobby scene, synchronized the movements of the snakes, made the apple spawner spawn apple only on the server and then synchronize their positions to the other clients. We added the possibility to add a player name which would as well be synchronized over the network.

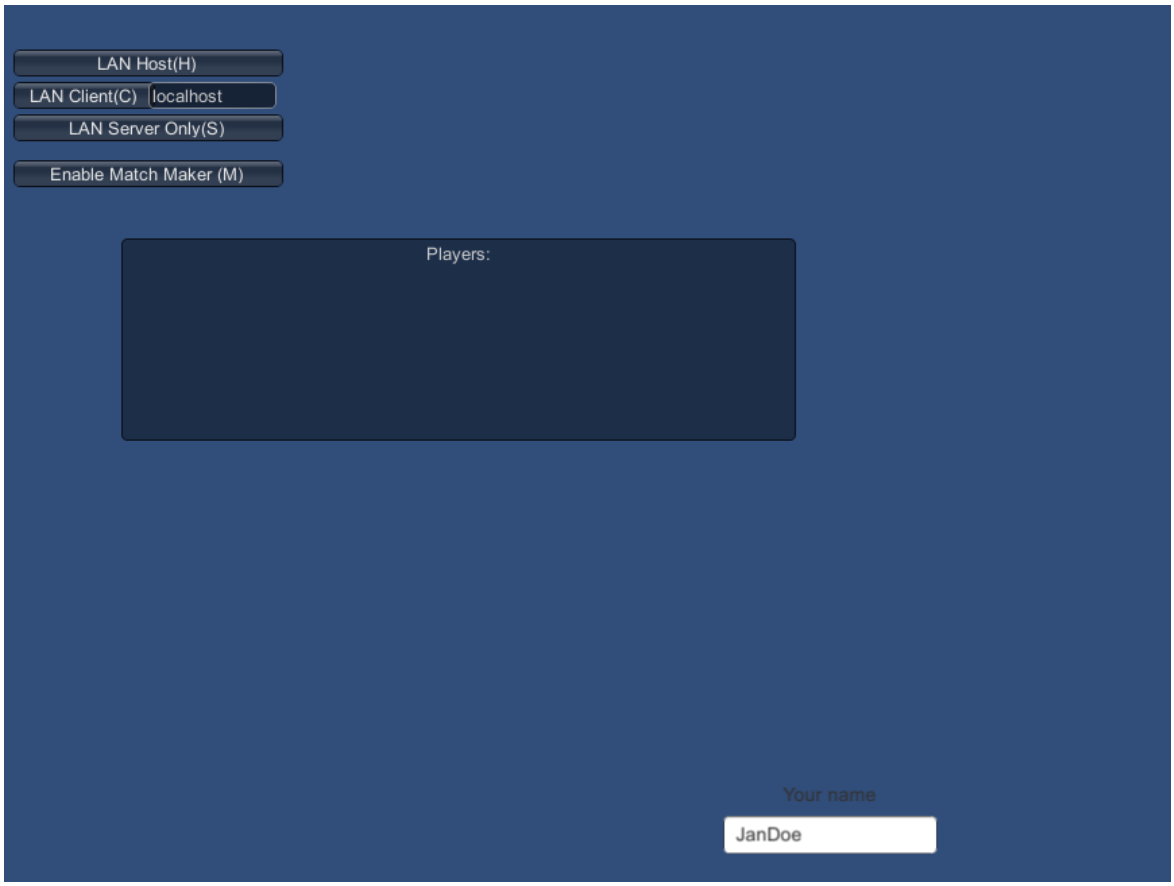


Figure 5: Esbjörnsson Snake Network Manager

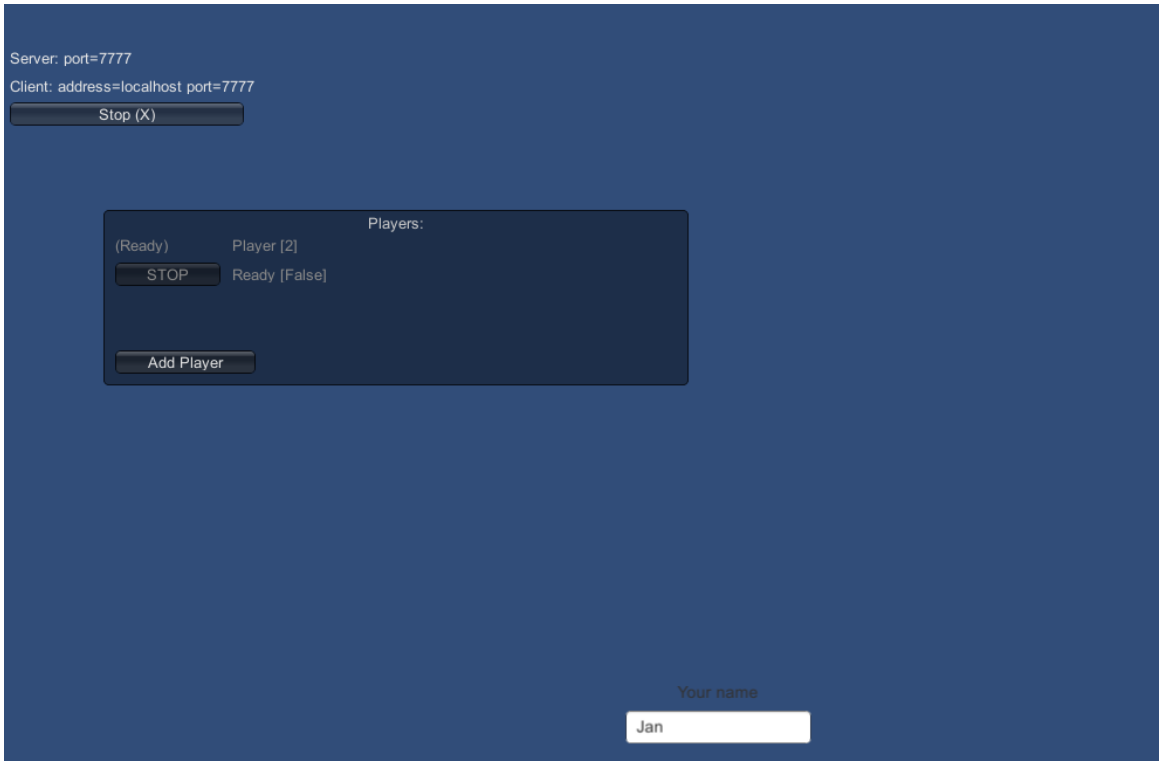


Figure 6: Esbjörnsson Snake Lobby

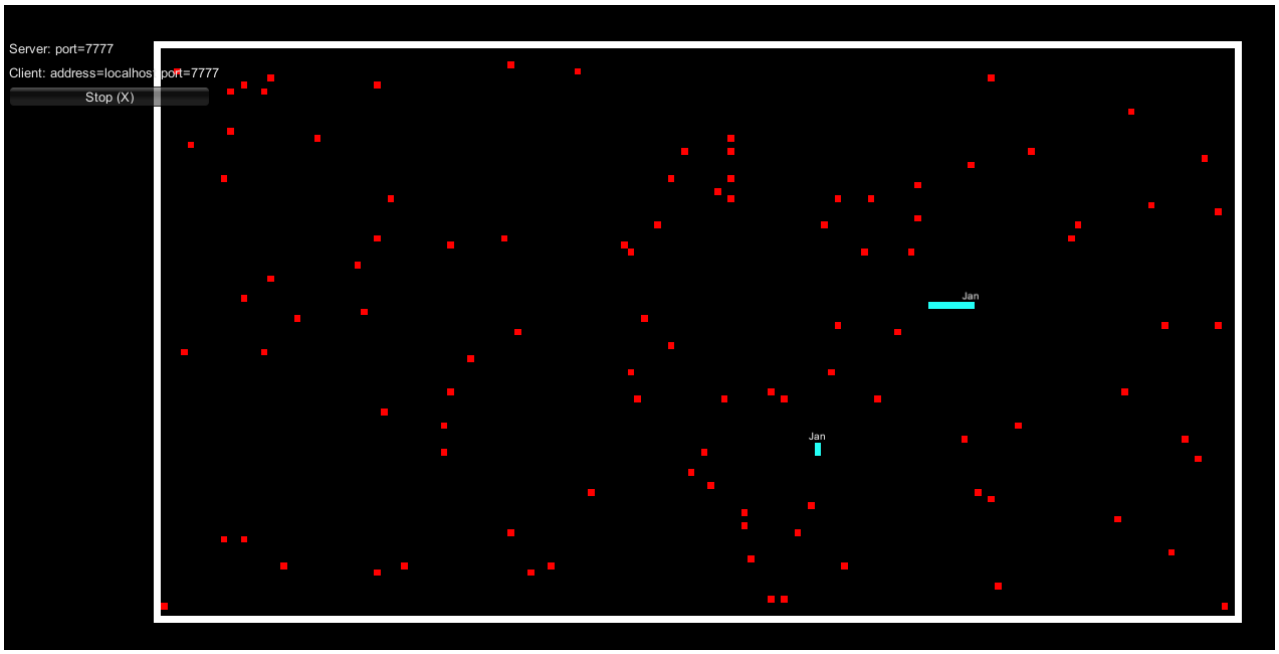


Figure 7: Esbjörnsson Snake Multiplayer

But there were multiple problems such as the names not synching correctly, and the transform positions neither so we decided to build our own snake, using some of the knowledge we gathered.

With all the information and experience we acquired we developed the final snake game. Every snake invokes a method repetitively as a command on the server, this method moves them in the direction they are currently facing. To make the game a little more challenging we introduced the ability to speed the game every given amount of time. But doing this would let the tail of the snake behind every time such a speed up would occur. This was a problem where some methods weren't called in the right order and at the right time. We had our snake game working on standalone version, using Unity editor as the server, and we could control the snake with the arrow keys. Now was the time to port it to smartphone, we decided to port it to Android as it was more convenient for us not having a mac. We used some platform dependent compilation in order to choose which UI to show to the user depending on their platform. We encountered a few problems, the first one was with the method assignment of the buttons. At first, they were assigned at runtime, supposed to, but they would not to it. So we had to add the canvas with buttons as a child of the player prefab and spawn them with it. This would work half fine, as it would solve our last problem but create a new one. The controller would work perfectly if it was the only one to be spawned, but as soon as another player would join their controller would be the only one working and the same would happen when a new one arrives. This problem is actually a known problem in Unity and the solution was to disable the canvas component of the player and only enables it if the player is the local player. The final snake game controller looks as follow.

Figure 8: Final Snake Game Controller

And for the snake game in itself looks like this.

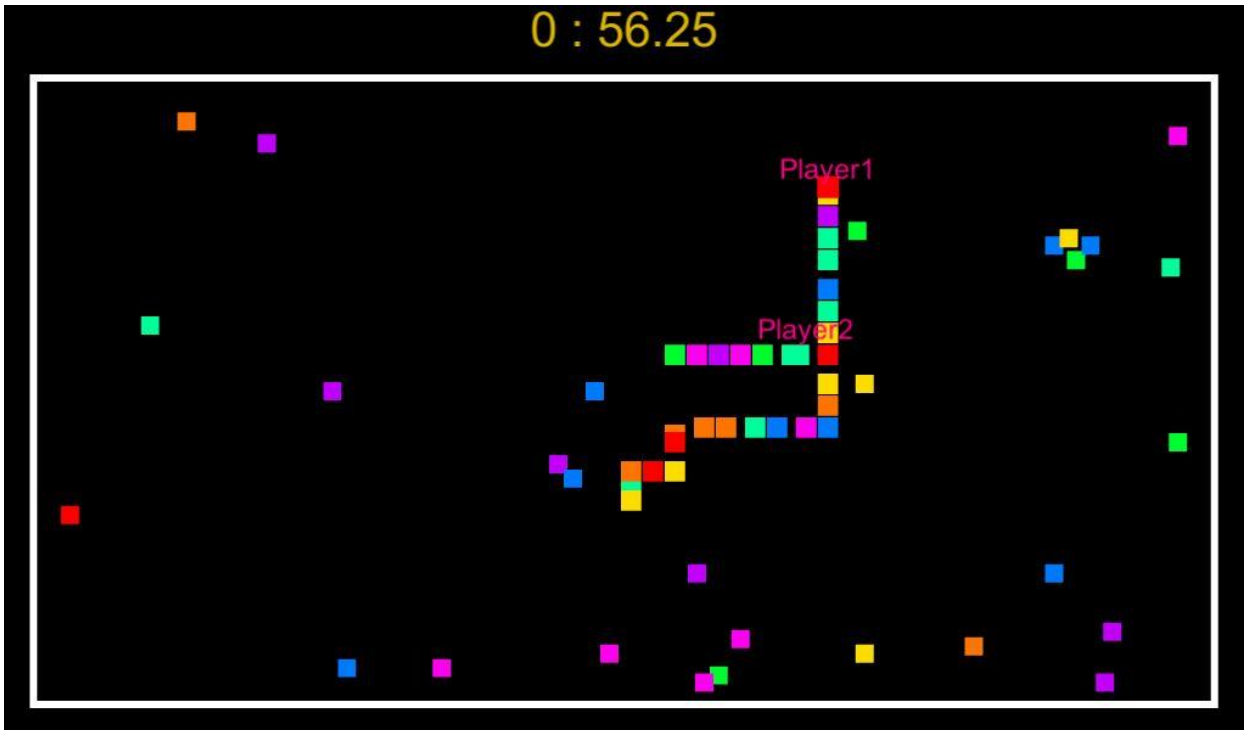


Figure 9: Final Snake Game

We then started to create our own connection buttons, which would override the network manager. Even if this would work perfectly fine on standalone, it would often produce a connection issue on a smartphone. The error message was the following.

a) Attempt to send to not connected connection {1}
UnityEngine.Networking.NetworkIdentity:UNetStaticUpdate()
b) Failed to send internal buffer channel:1 bytesToSend:26
UnityEngine.Networking.NetworkIdentity:UNetStaticUpdate()
c) Send Error: WrongConnection channel:1 bytesToSend:26
UnityEngine.Networking.NetworkIdentity:UNetStaticUpdate()

As our current UI was very bad and inappropriate for smartphones, we were using the old OnGui() from Unity which doesn't give a lot of freedom, we decided to drastically change the lobby and connection system.

4 Lobby

Using what we learned with the Udemy course “Unity Networking From Scratch for (Unity 5 to Unity 2018.2)” (Byl, 2019) we started to work towards our lobby manager which implement the namespace Prototype.Networking. We disabled the old network manager and network manager hud to replace it with the newly found lobby manager. As the lobby manager comes with a lot of unused feature for this project, such as match making, we changed it and added our own main panel. This main panel is composed by two sub-panel. One that will be active on the server and one on the client.

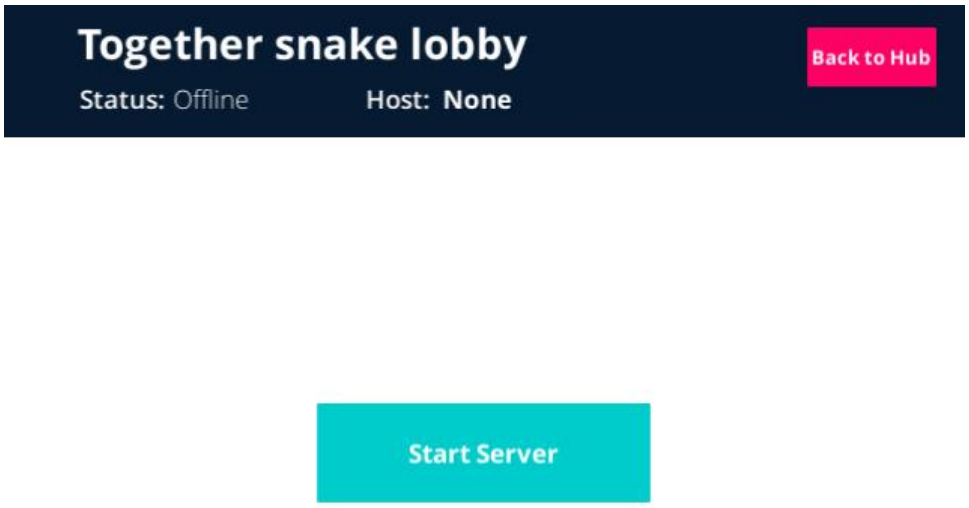


Figure 10: Snake Lobby Server

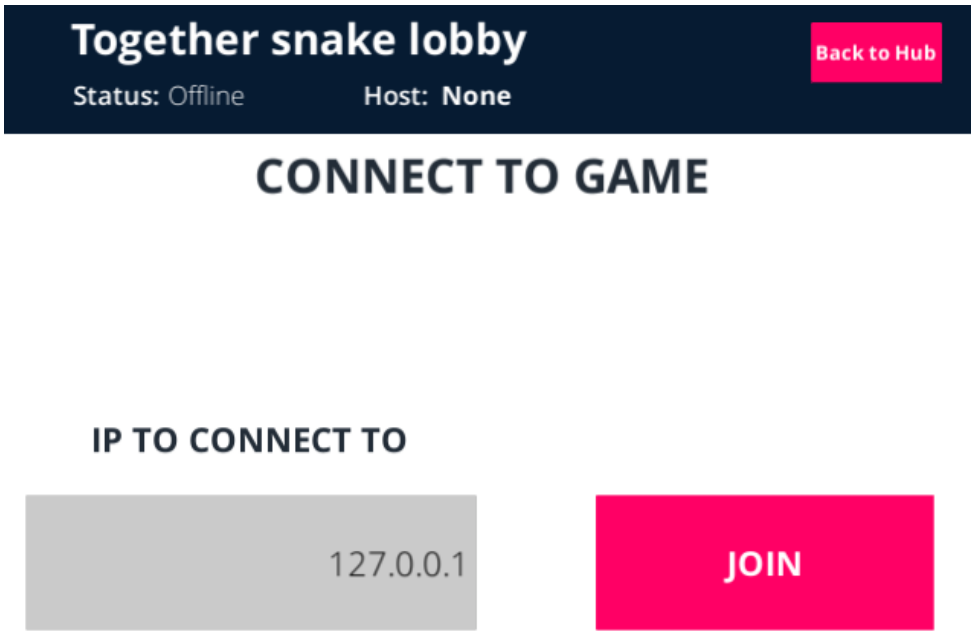


Figure 11: Snake Lobby Client

Both panels would use the same script, LobbyMenu.cs, which contains mainly callback called by the UI, such as:

```
public void OnClickJoin()
{
        lobbyManager.ChangeTo(lobbyPanel);

        lobbyManager.networkAddress = ipInput.text;
        lobbyManager.StartClient();

        lobbyManager.backDelegate = lobbyManager.StopClientClbk;
        lobbyManager.DisplayIsConnecting();

        lobbyManager.SetServerInfo("Connecting...", lobbyManager.networkAddress);

        mainPanel.SetActive(false);
}
```

To get the current IP address of the server we use the System.Net namespace in the “LocalIPAdresse” function as follow:

```
public string LocalIPAddress()
{
        IPHostEntry host;
        string localIP = "";
        host = Dns.GetHostEntry(Dns.GetHostName());
        foreach (IPAddress ip in host.AddressList)
        {
            if (ip.AddressFamily == System.Net.Sockets.AddressFamily.InterNetwork)
            {
                localIP = ip.ToString();
                break;
            }
        }
        return localIP;
}
```

We need the IP address because it is currently the way to connect to the server as we didn’t implement a matchmaking system. Once connected the clients would arrive in the waiting lobby, where they can personalize a few options, very basic for now. For the snake, they can choose their player name and the color of their future snake. That information is shared from the lobby to the game with the help of the LobbyHook.cs file. For example with the snake the hook looks like the following:

```
public class SnakeLobbyHook : LobbyHook
{
    public override void OnLobbyServerSceneLoadedForPlayer(NetworkManager manager,
    GameObject lobbyPlayer, GameObject gamePlayer)
    {
        LobbyPlayer lobby = lobbyPlayer.GetComponent<LobbyPlayer>();
        SnakeSetUpPlayer snake = gamePlayer.GetComponent<SnakeSetUpPlayer>();

        snake.playerName = lobby.playerName;
        snake.playerColour = lobby.playerColor;
    }
}
```

And then need to be correctly set up in the SnakeSetUpPlayer.cs fil.

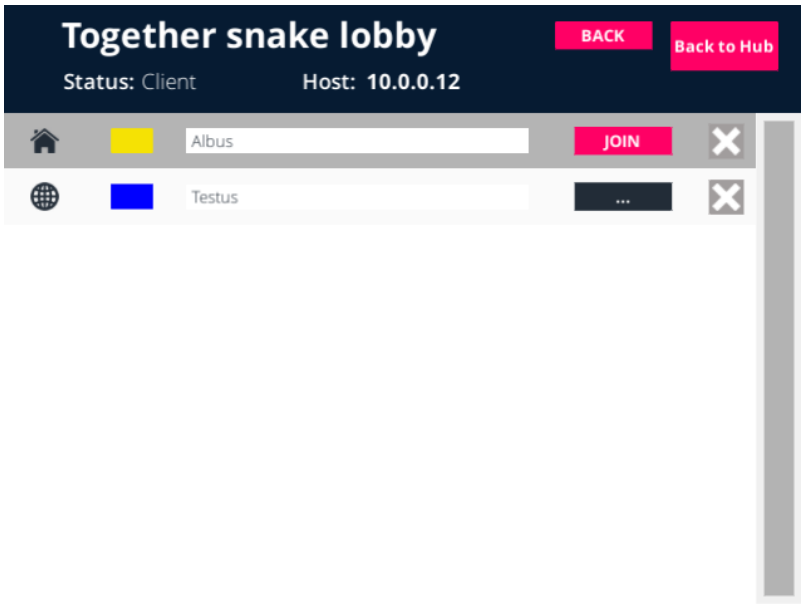


Figure 12: Snake Waiting Lobby

At this point, we encountered an interesting problem. As we were testing the application on old tablets, Android 4.4, some UI elements were not rendered. The PlayerInfo prefab, it is the whole grey rectangle from the figure above, which holds the customizable information and the button to notify that this player is ready would not appear on our tablets. But Pressing at the position the button “Join” should be would still trigger it as if it was there. Therefore we assumed it was a problem of version compatibility as it is working perfectly on standalone and on a newer tablet. When we were over with the lobby we decided to develop our second game using the newly developed lobby.

5 Domino Game

This second game had to be different from the other one to let us explore other possibilities. Therefore, we chose a game where the controller would be way more at the center of it compared to the snake. A domino game is basically a card game, where each card has two values, every player starts with seven dominos in hands and has a goal to empty its hand. We started by creating our own object domino which derivate from ScriptableObject and added the possibility to create them within the create asset menu from Unity.

```
[CreateAssetMenu(fileName = "NewDomino", menuName ="Domino")]
public class DominoCard : ScriptableObject
{
    public Sprite artwork;
    public int upperValue;
    public int lowerValue;

    private bool isUpperUsed = false;
    private bool isLowerUsed = false;
    private bool isUsed = false;

    #region Getters/Setters
    #endregion
}
```

We then created our twenty-eight dominos and needed a way to display those objects as scriptable objects are data containers. Thus, we created the DominoDisplay.cs to handle the visual representation of the data as well as the OnClick listener. We used the same logic of platform dependent compilation as for the snake game to differentiate the server and the client. Whenever a player is registered it calls the DominoGameManager to receive a hand. Unfortunately, for now, the DominoGameManager are not synchronized over the network as we discovered later that they were independent of each other. Once the play has its hand it will represent it by instantiating domino from the domino prefab with the data furnished by the serializable domino and position them one after the other on the controller.

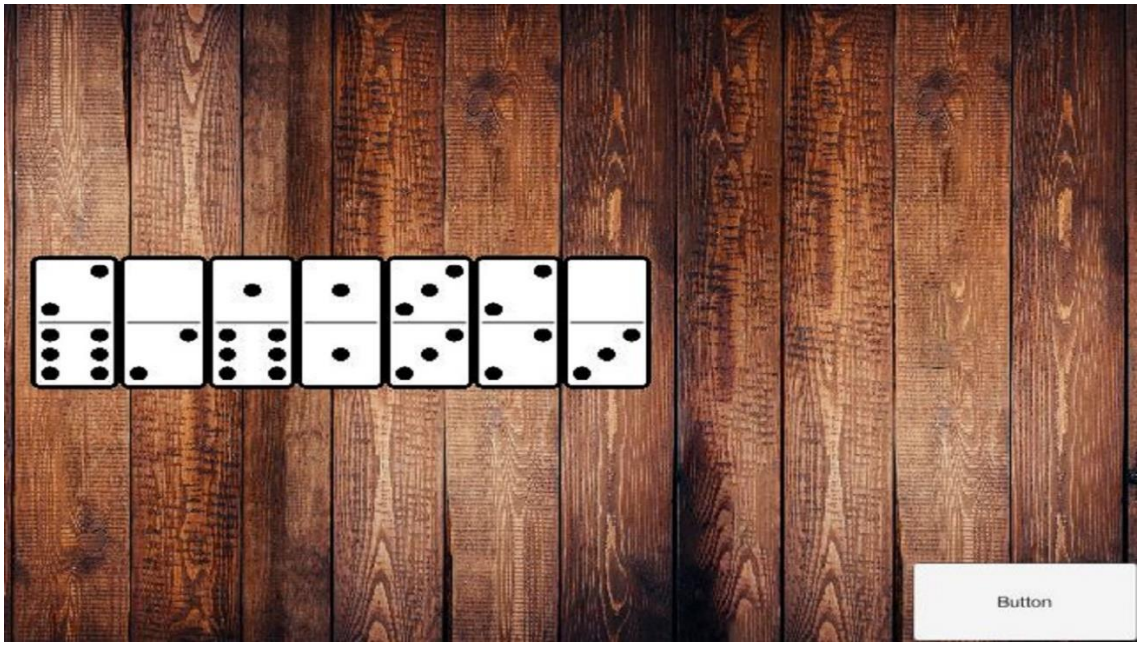


Figure 13: Final Domino Controller

Unfortunately, we encountered a lot of problem with the synchronisation of hands between all the controllers and the main game, therefore we were not able to finish the implementation of the domino game.

We passed the user name from the lobby and used it on the game screen as turn indicator.

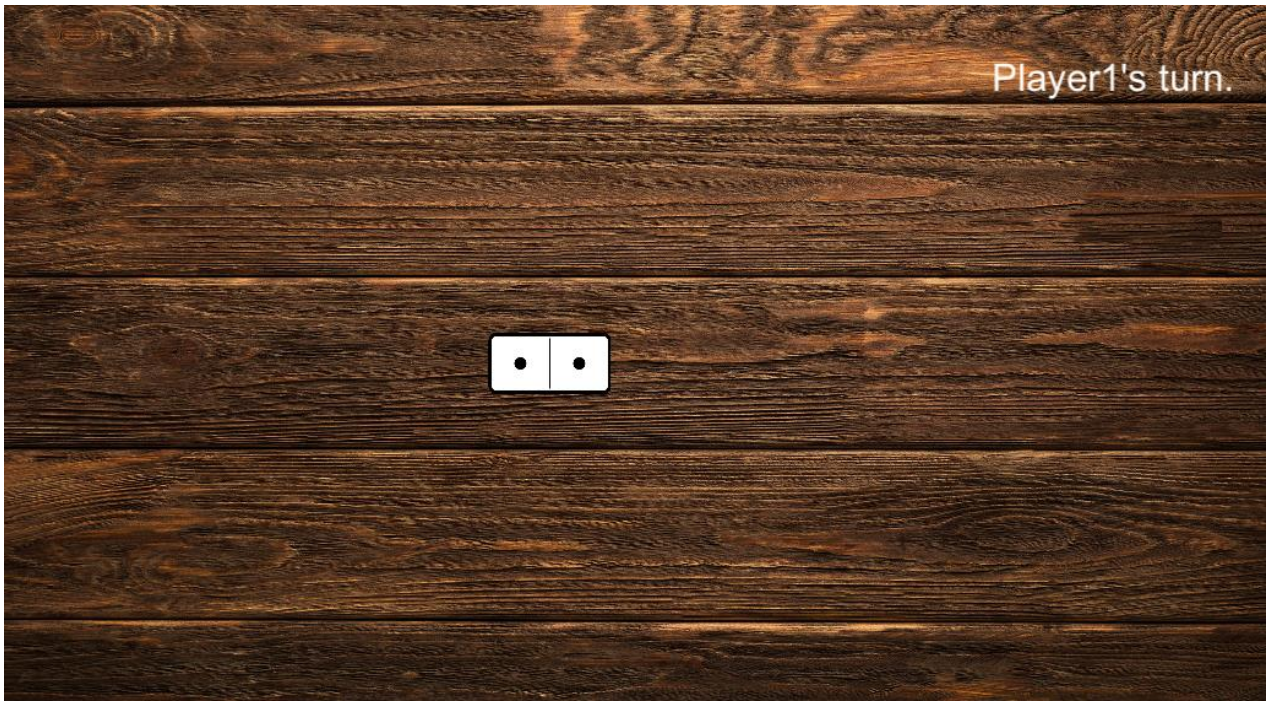


Figure 14: Final Domino

6 Conclusion and future work

6.1 Results

6.2 Improvements / Future work

7 List of illustrations

8 Bibliography

Byl, P. d. (2019, Mars). *Unity Networking From Scratch for (Unity 5 to Unity 2018.2)*. Recovered on Udemy: https://www.udemy.com/unet_intro/

Esbjörnsson, R. (2017, 04 23). Recovered on https://www.youtube.com/watch?v=XKlOUQ45tYc

Gaimin. (2018, 07 20). *Gaimin*. Recovered on gaimin.io: https://gaimin.io/how-many-gamers-are-there/

9 Erklärung der Diplomandinnen und Diplomanden / Déclaration des diplômant-e-s

Selbständige Arbeit / *Travail autonome*

Ich bestätige mit meiner Unterschrift, dass ich meine vorliegende Bachelor-Thesis selbständig durchgeführt habe. Alle Informationsquellen (Fachliteratur, Besprechungen mit Fachleuten, usw.) und anderen Hilfsmittel, die wesentlich zu meiner Arbeit beigetragen haben, sind in meinem Arbeitsbericht im Anhang vollständig aufgeführt. Sämtliche Inhalte, die nicht von mir stammen, sind mit dem genauen Hinweis auf ihre Quelle gekennzeichnet.

Par ma signature, je confirme avoir effectué ma présente thèse de bachelor de manière autonome. Toutes les sources d'information (littérature spécialisée, discussions avec spécialistes etc.) et autres ressources qui m'ont fortement aidé-e dans mon travail sont intégralement mentionnées dans l'annexe de ma thèse. Tous les contenus non rédigés par mes soins sont dûment référencés avec indication précise de leur provenance.

Name/*Nom*, Vorname/*Prénom* .....

Datum/*Date* .....

Unterschrift/*Signature* .....

Dieses Formular ist dem Bericht zur Bachelor-Thesis beizulegen.
Ce formulaire doit être joint au rapport de la thèse de bachelor.