InTouch Project Development Document (Programming Part)

Michael Luan & Jeremy Liu

1. Setting up two Oculus devices.
2. Building the Project.
3. Setting up the basic synchronization of two players’ movement.
4. Setting up the basic synchronization of two Oculus devices.
5. Spawn Points of two players.
6. Basic setting for turn-based actions.
7. Systems that resolves synchronization problem
   1. Delay system
   2. Dual variable system
   3. Rabbits
8. Item
   1. Property of item
   2. Send item
   3. Bag collection system
   4. Use item
   5. Collect item
9. Turn system
10. Determining winning/losing

# **Setting Up Two Oculus Devices**

**This is a two-player game and requires two set of Oculus devices linking together.**

**For the First step, we linked two Oculus devices including Rift and Touch to two separate test computer, and use our computer to remote control the test computer. However, we found that when using remote mode to control the test computer to test our game, the Oculus Runtime SDK could not work properly. I have tried different methods such as re-install the runtime SDK and used administration account, but it still not works. Finally, we decided to use our computer for development, and use GitHub to manage the version of our project, then use a test computer to pull from Git and test.**

# **Building the Project**

**After configuration and download the needed oculus plugin and GitHub environment, I build our project called InTouch and initial the GitHub project. Here is our project GitHub address:**

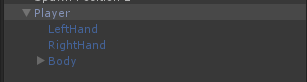
<https://github.com/luanhaoqing/InTouch>

# **Setting up basic synchronize of two player movement**

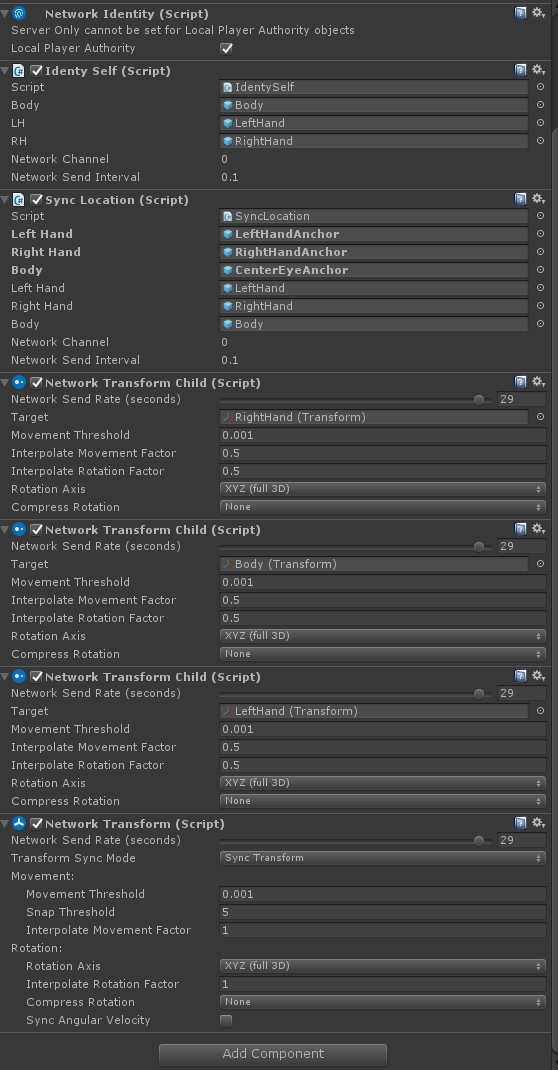
**We went through the official tutorial of unity network functions, and built a basic scene including synchronizing two player’s movement using *NetworkManager*, *NetworkIdentity*, and *NetworkTransform*. Then the two players could see each other’s movement in unity window. This unity package of the network is based on the model of one player host the server, and the other could join the game by entering the host’s IP address.**

# **Setting up basic synchronization of two Oculus device.**

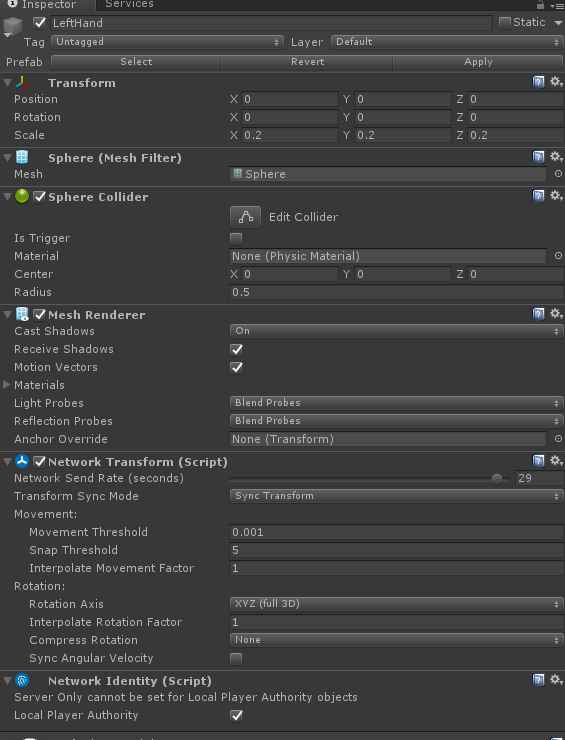
**I met some difficulties in setting up the two Oculus devices. At first, I did the same thing as I did before for the two Oculus devices. Each time when a player joined the game, it will automatically generate an Oculus prefab. However, I found it works very wired; it would blink, and also control the other’s movement. After I had tried different possibilities, I found the reason is that the built-in function of Oculus would try to control both of the prefabs of Oculus. So I tried to modify codes in the Oculus controller function, while it still not works well.**

**Finally, I figured out a way to solve this problem without modifying codes of Oculus. I use the idea of proxy to create a new set of objects to reflect the movement of the player.** 

**I use codes which attached on the Player object to get the movement of the Oculus and touch and then pass them to the objects which will be synchronized with the other. After several tries, I finally find it should use Network Transform Child function instead of Network Transform function. So the configuration of Player is like this:**



**And each part of the objects such as Left Hand, Right Hand has a network transform attached.**



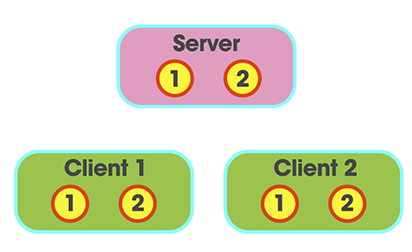
# Spawn Points of players

**To allow each player to start in their designated position once they login, I use Network Start Position function of Unity to realize the basic function. Firstly, I added two Spawn points, and the two player prefabs would be generated there. However, I find that because of the player would be synchronized to the Oculus, so that in start function, I set the Oculus position to Spawn position.**

# **Basic setting for turn-based actions**

**In order to make anything work in the game, we need to build a basis of turn-based actions. That is, saying “your turn”, and “my turn”.**

**To set the basic turn base mechanic, I did a lot of research. For the newest version of Unity network function, for a two player game, there are totally six same scripts running at the same time:**



**And the player who hosts the game is a client as well as a server.**

**Therefore, I divide the code into two parts: Server parts and Client parts, and count the time in the server, and sync to the client.**

# **Systems that resolves synchronization problem**

Synchronization between two clients within the framework of Unity Network Manager is tricky. We have developed a few methods to make sure that it works smoothly.

## Delay system

Because there are two players acting on both clients, delay in transmitting messages may cause serious problem in condition judgements. For example, if this piece of code is to run on both client:

If (bool == true)

Do something

Set bool = false

Then when this run on the hosting computer, the global Boolean bool would be set to false. The other client may run this script 10~50ms later, and by then because the Boolean has already been changed, it never goes inside the if-loop on the client.

To solve this, we forced a 0.5-second wait time before these changes to ensure that the script will run on both computers:

If (bool == true)

Do something

Wait(0.5seconds)

Set bool = false

## Dual variable system

Dual variable is also used in certain cases to ensure that the action on both computers are recorded and will not interfere with the server.

In this case, a synchronized global variable determines if something *should* run on both client. Then on each client, a local variable records if the said thing *has* run on local client.

## Rabbits

Rabbits are used to synchronize the control input of the other player to local client. The problem we are trying to solve are:

1. Control input cannot be synchronized.
2. Each client’s control is local and the other client cannot acquire any information of it.

This means that if Player 1 goes from Move mode into Send mode, then on Player 2’s computer they cannot see it happen.

Then we utilize rabbits to achieve our goal because the movement of objects can be synchronized. Every time a player changes control mode, the rabbit is moved to a different position. And the other client knows the state change and can reflect it locally.

# Item

Item system is the most complex system in the game. Players can:

* Get,
* Use, or
* Send

these items as they play the game.

## Properties of an item

Every item is a prefab saved in the assets folder.

They have the following properties:

* Name (String)
* Could send (Boolean)

They share the same tag “ITEM” in Unity Editor.

## Sending items

Sending item uses a “rabbit”. After the send command is given, the “rabbit” is then moved to different places, so that both clients know that this player is going into send mode.

If the couldSend Boolean in item property is true, then whenever the item collides with player’s hand, it will initiate the sending process.

The sending process goes as follows:

1. Finding the two players – they are instantiated after each connect to the server, and are found though the Player tag.
2. Identifying player – on each client, identify the sender and receiver.
3. Move the item form sender’s inventory to the receiver’s.

## Bag collection system

This inventory system stores the player’s items. Using this system, the player can achieve these actions with items:

Get();

Trade();

Use()

The inventory maintains an array with the length same as inventory slots. It also keeps track if the item number and the rune number in the inventory.

After each change to the inventory, a rearrange() function is used to tidy things up.

## Using items

Now the only usable item is the Healing Crystal.

Again, we use a rabbit to determine whether a player is in Use Mode. Once confirmed, when the item OnTriggerEnter() with the player’s hand, it will then start the using.

The Healing gem is within the Player prefab so its movement is synchronized. Player then move the gem over the tiles to determine which one to heal.

## Collecting items

The logic for collecting items is simple.

If an item comes in contact with the player character on board, it will check if the inventory is full.

If the item is a key, then check in to bag collection system if it have three runes. If it does, get() the key.

If the item is not a key and the inventory is not full, it goes to the bag collection system to get() this item.

# Turn system

**The game is a turn-based game. The turn system only runs on server and a variable called** count is synched.

Within every turn, count goes form 3 to 0 as players use up their actions. When it reaches 0 the turn will end.

# Determining winning/losing

At the moment there is only one way to trigger the losing process:

Every time a tile’s health reaches 0, detect if it has the player character on it. If yes, the game is lost.

For winning, a sequence of events needs to happen:

1. Two players both have the key, then,
2. The gate shows, then,
3. Two players both reach the gate, then
4. Players win.