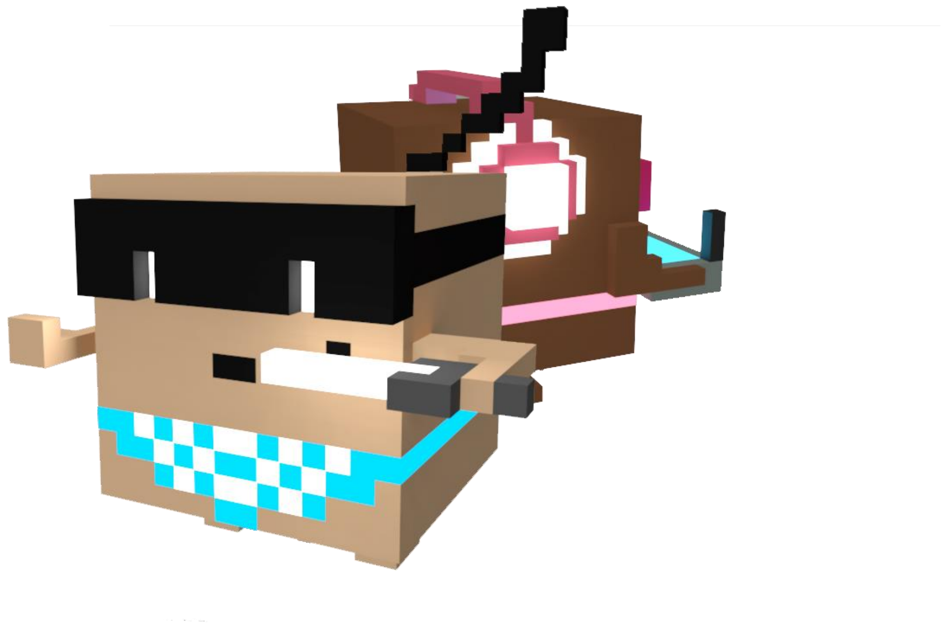
Notorious Duo

Software Engineering Project 2

2017/5/31



Xuanhui Xu

Group 6 2XL

NAME

XU XUANHUI

LIU YUXUAN

LIU YUHANG

XU CHEN

Software Engineering

Table of Contents

Chapter 1 – Project Overview

1.1 Project Background

Some people prefer a peaceful and quiet life, but Mr. X and his best partner, Ms. L may beg to differ. This notorious duo leads such a bold and wild life that they are always on the top-wanted list. And now a new-built vault has drawn their attention…

1.2 Project Introduction

Our project "The Notorious Duo" is a PC game featuring dual-player system. The game is split into two sides, one player interacts with the 3D game scene (R5) and the other interacts with the 2D game scene(R6). 3D player controls Mr. X who is a really talent burglar to get into the vault, breaking the maze and collecting information. 2D player controls Ms. L who is skilled at tablets and network. She has the power to hack into nearly any system. So this time, players get all the message they want, solving the quiz, figuring out the password, helping Mr. X from the other side of the world.

1.3 Project Goals and Setting Out Plan

The 3D game scene was designed to be a huge maze with a "vault" at the bottom, which serves as the goal of the game. Therefore, the player on 3D (denoted by Mr. X) side will have to try to find his way to the vault with the help from the player on 2D side (denoted by Ms. L). And how it actually works?

In order to solve this problem, photon networking framework was introduced into our project, which serves as an information/data carrier between two game scenes and plays a role as the backbone of our game. The 2D game scene was originally a mini-map which is the bird-view of the 3D game scene separated into 3 different floors. And with the help of the photon networking framework, Ms. L is now able to see the live position of Mr. X. In addition, there are some game objects in 3D game scene with which Mr.

X can interact and trigger a mini-game on 2D game scene. And Ms. L is supposed to win the mini-game so that Mr. X can use that exact game object to continue his journey to the vault.

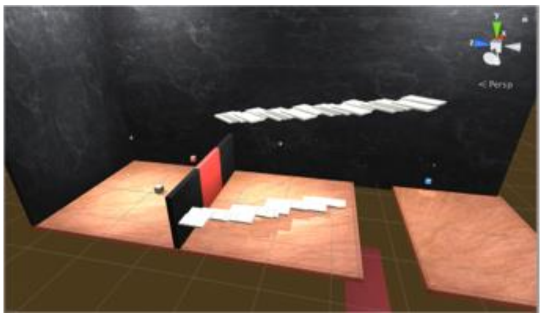
Now the architecture of our project is quite clear, as follows:

- 3D game scene and 2D game scene
- Photon Networking Framework that connects scenes together
- Storage(database) that stores user data

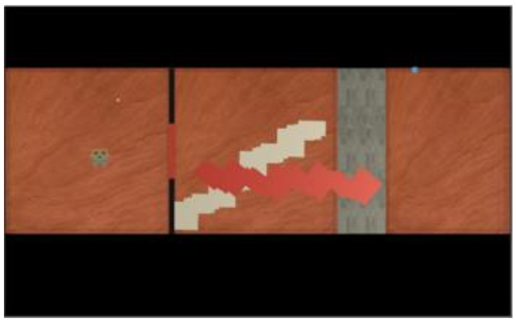
In order to achieve goals of our game, there was much to learn. As for the game scene work packages, we went through a couple of tutorials on Unity official website. And when it came to photon networking, we read a long list of introductions and tutorial documents which explain how this whole framework works. Last but not least, some supplementary files were checked out during our try to connect MYSQL with Unity.

1.3.1 Front End Interface

The implementation of 3D game scene (O1) and 2D game scene (O2) started with very limited functions.



O1. Basic 3D Interface



O2. Basic 2D Interface

1.3.2 Networking

The implementation of photon networking started with primitive text transmitting between two devices based on a client-server

1.3.3 Storage

The implementation of storage started with designing the database

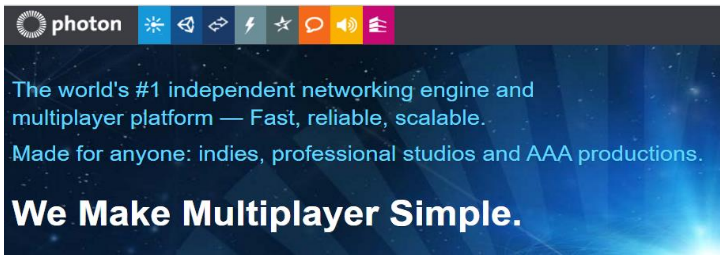
Chapter 2 - Project Requirement

2.1 Specific Requirements

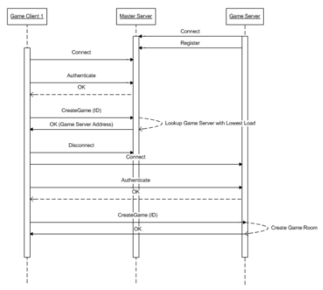
2.1.1 Networking

All projects must feature a networking component. Data must be shared between a minimum of two networked devices (laptops, mobiles, tablets, etc.). These devices do not have to be different in specification or type – for instance two or more laptops is OK. Proposals should specify the networking architecture that is to be used (peer to peer, client/server, etc.).

Our game is a two-player-game, so it is really easy to meet this net working requirement. Two players use their own computers or laptops (no matter is OS X or Windows OS) to play the game, connected by network. And the networking architecture is client/server (R2), which uses photon network (R1). The server create a room and clients are added in.


photon
The world's #1 independent networking engine and multiplayer platform — Fast, reliable, scalable.
Made for anyone: indies, professional studios and AAA productions.
We Make Multiplayer Simple.

R1. Photon Networking



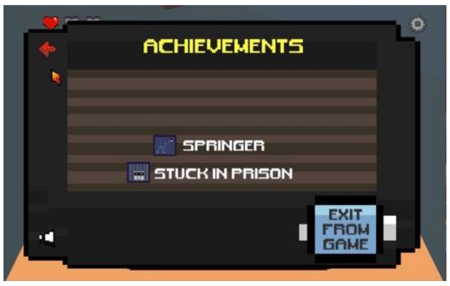
R2. Client/Server

2.1.2 Storage

Data that is shared between devices must be stored and retrievable by all devices. It is common (but not absolutely required) for a database to be employed. Specifics of the storage mechanism(s) should be provided and justified in the proposal.

Our game needs to login through an account, so database is needed for register and login checking. What's more, our game has achievements system, the data of which is stored in MySQL database (R3) related to users account. When a player reaches the condition of an achievement, the data would be inserted in the database. And then when player click the achievement board, system would get the data from database and show the information in front interface (R4).

R3. MySQL

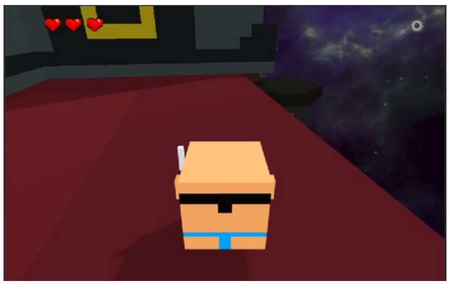


R4. Achievements Tablet

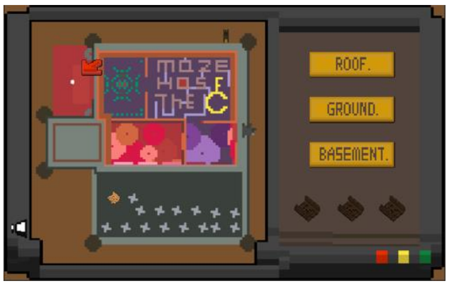
2.1.3 Front end interface

All projects must feature one or more front end interfaces which allow users to access functions and features of the final application. In some cases, a front end may not be required on all devices. For instance, in a client/server architecture it is possible that only client devices will need front end interfaces. Note however that an interface for the server should still be considered for developers, etc.

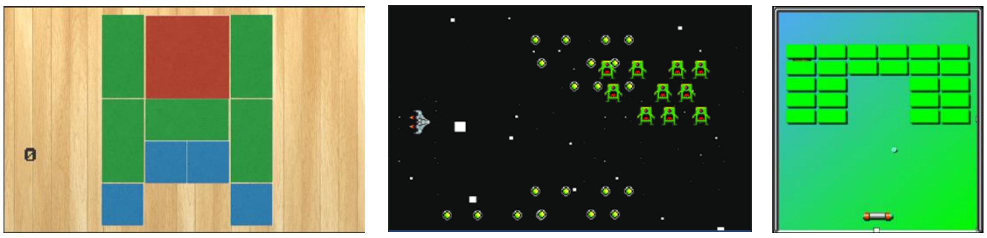
Because our game is a two-player-game that is based on unbalance information for two players, forcing them to cooperate and communicate, so there's totally different interfaces for two devices. The 3D game player would have a 3D game interface (R5) and control Mr. X as a third person view. Also, 3D player can check on the achievements tables (R4). On the contrary, 2D game player would have a 2D interface (R6) to operate and many mini games (R7) to play with. However, there's still some interfaces are designed for both 3D and 2D players, such as Login/Register interface, Room Choose/Create interface and Character Choose interface (R8). There's no specific interface designed for server as we used photon network, which we just need to ask for rooms.



R5. 3D Interface



R6. 2D Interface

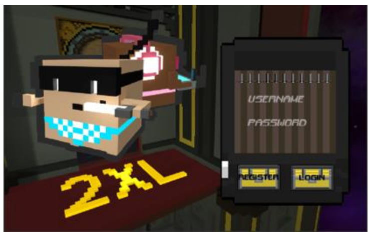


R7. Mini Games Interfaces

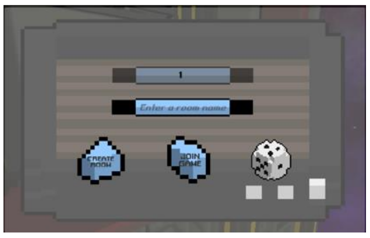
2.2 Additional Requirement

2.2.1 Additional gameplay feature (networking, front end)

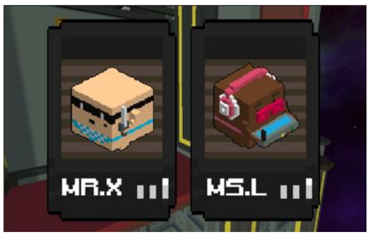
We added voice system to let two players talk to each other. And also, room system is added for making players have the abilities to create/join room (R8).



R8. Login/Register interface



Room Choose/Create interface



Character Choose interface

Chapter 3

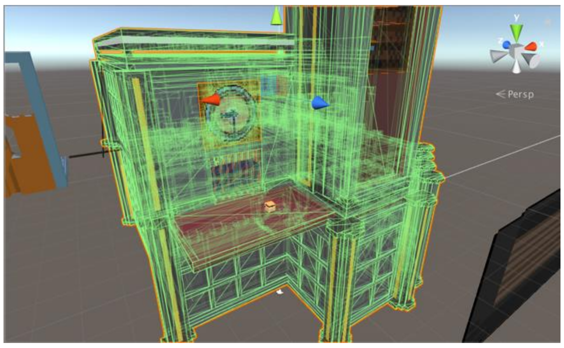
– System Design and Implementation

Since the game is a two-player cooperation game, at the very beginning of our project, the game was designed to satisfy 4 necessary parts. They were network, GUI, database, and the tricky elements to make it like a game. The first demo was designed to be a very simple game that a 3D player in a simple scene and it movement will be showed on a 2D map. The details of how we achieve the demo and how other features were added in are as follow.

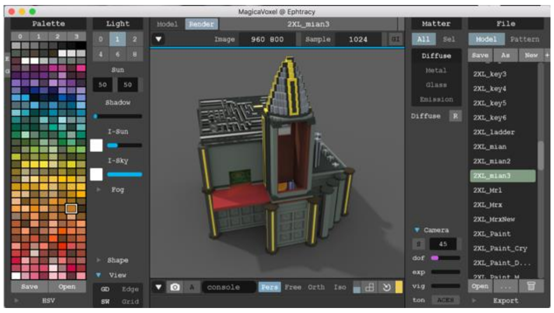
3.1 GUI Implementation and Design

3.1.1 Basic Environment Design

The challenge here is just by using cube or cylinder to build up scene costs plenty of time and hard to be artistic. In order to solve this problem, a software called MagicaVoxel is used for creating pixel models. It can generate .obj file of the models that can be imported into Unity 3D directly with a fully functional mesh collider. Although beautiful environment scenes are created by MagicaVoxel (D2) quickly and fluently, it also creates a lot of unnecessary faces for mesh colliders (D1), which would take much more inner storage. However, Isaac who is in charge of GUI bought a Plug-in Gear called ‘MagicaVoxel Qubicle to Unity’ (I1) to generate prefabs for Unity to save storage.



D1. Mesh Collider



D2. MagicaVoxel

3.1.2 Items Design

All the little items (D3) in games are made by MagicaVoxel, too. With the help of the Plug-in Gear, 3D features can transform in to 8-bit sketch (D4) easily.

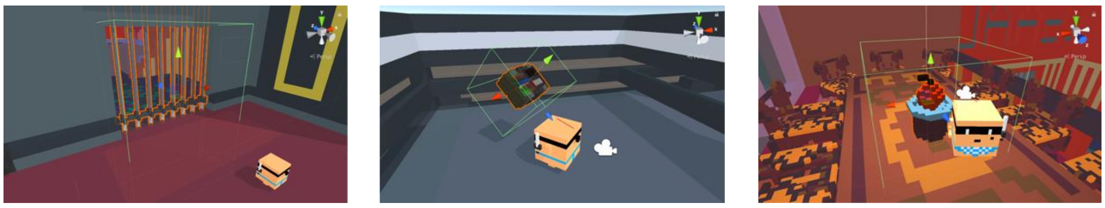
D4. Self-made Sketch

D3. Self-made Pixel Model

3.2 Trick Implementation and Design

3.2.1 Colliders

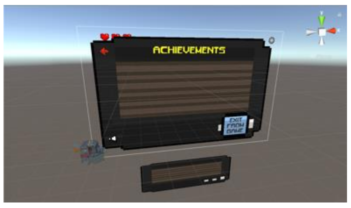
By using colliders (D5), plenty of tricks can be achieved. Like auto gate, C4 picked up, button, maps… After 'is Trigger' is turned on, the colliders would become a trigger to judge collision.

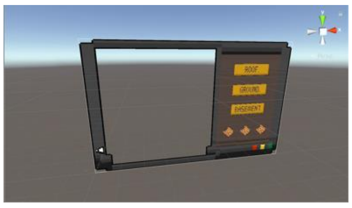


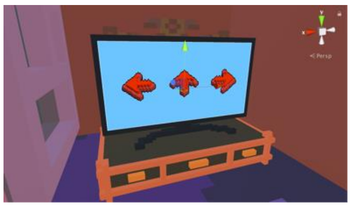
D5. 'Is Trigger' Colliders

3.2.2 Canvas

The information flow on the screen in 3D interface (D6) and 2D interface (D7) are made by canvas. Especially for the TV trick (D8), canvas is put into the world coordinate, which makes player use cursor to input answer.


D6. 3D Canvas


D7. 2D Canvas


D8. 3D Canvas - TV

3.2.3 C4 Follow and Free Camera

C4 follow script and free camera are modified from the follow script from standard assets. [1]

3.3 Game Controller Implementation and Design

3.3.1 Character Move

Make the character move, calculate the speed direction when the character jump and allow a little operation for correction, which makes Mr. X having the ability do the tricky jump like climbing on the wall. At the beginning, just changing position by update rates could make the character climb the wall. In order to get through this, character controller is used.

3.3.2 Health Manager

In charge of reducing health by falling from high, and show the health state on the canvas in 3D interface. Did the math about the time our character flying in the sky for judging, but there seems having some bugs about .isGround() function.

3.3.3 Achievements Tablet

Set different achievements' condition. Show the achievement tablet when the condition is achieved.

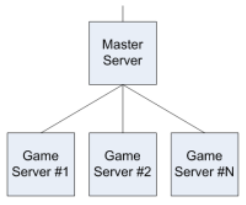
3.4 Network Implementation

3.4.1 Connection

In the network part, the photon server was used as our game server. And LoadBanlancing Application is our server logic. The basic set up is always simple: There is always one Master Server and 1...N Game Servers.

The Master Server handles these tasks:

- keep track of the games that are currently open on the Game Servers.
- keep track of the workload of the connected Game Servers and assign peers to the appropriate Game Servers.



Photon Server Concept: LoadBalancing Setup

- keep and update a list of available rooms for clients in the "Lobby".
- find rooms (random or by name) for clients and forward the game server address to them.

The Game Servers has these tasks:

- host the game rooms. They run a slightly modified version of the Lite Application for this purpose.
- regularly report their current work load and the list of their games to the Master Server. [6]

Since our game doesn’t need too much game server, we have only one game server. When the client connects to the master server they will be arranged to the game server. And then they can create or join a room.

3.4.2 Position synchronization

In this part, we give the two objects we want to synchronize a same script called photon view and set a same ID to them. Then we record the position information as a vector 3D and transfer it as a stream. Then 2D scene gets it and updates position of the object which has the same ID in 3D scene. Here we got two challenges. The first was how to transfer the 3D position information to a 2D map. And we solved it with Isaac’s good idea. We changed the 2D map to a complete 3D scene, just change the view perspective so people who look at it will just see a 2D map. Then the position could be correct and didn’t need to change anything. The second challenge was the photon view had a problem that the people who join the room first will own the Mr. X (our game main character) which has photon view. So if a player creates a room and chooses to play 2D scene. The other player cannot move Mr. X because he doesn’t own it. The problem stuck me quite a long time and finally solved by adding a Boolean to set if it is a 3D character. When client choose a scene to play, we check if it is 3D scene and if it is we ask for the ownership of the character. Whether the character was owned by 3D or 2D it will be transferred to 3D player.

3.4.3 RPC

In order to make our game more interesting, we need to create more interaction between two scenes. And Remote Procedure Calls were used to achieve this part. Remote Procedure Calls are exactly what the name implies: Method-calls on remote clients (in the same room). For instance, the 3D character push a button in 3D scene. It will raise an event and send it to the client in the same room. And the 2D scene can get the event with the method on events. According to the different event codes, 2D or 3D scene will do different tasks.

3.5 Database Design and Implementation

There are two tables in our database system, the first is for player to register and login. In this table, we have username, password and player name. And the username is also unique because we will use it as the primary key in the second table. The other table is achievement table, as I said the primary key is username. When player with a username got an achievement I this table, the value will change from null to number one. And the main challenge in this section actually is how to record the achievements. We have not only one achievement and at first we haven't decided what achievement we will get. So give achievement an alone table might be a good choose. And just simply change the value can be easy to implement in the code.

3.6 Mini Games Design and Implementation

When it comes to the implementation of mini-games, things went rather well. The first game and the third game are quite similar. Both of them are based on rigid body movement and collision detection, that is to say, of the game objects should be set to rigid body with collider so that they can affect each other in a physical way. But the second game was a different story and it’s will be talked about last.

3.6.1 Space Shooter

As for the first game, the main challenge was to make the game run smoothly. This shouldn’t be a problem at the beginning, but as more bullet-shooting enemies added into the scene, the number of game objects increased rapidly, which severely damaged the smoothness of the game. Fortunately, the solution to this problem was simple — object destroy functions were used to eliminate previous enemy bullets right after when new bullets came out. This worked well to maintain the overall number of game objects on the screen and improved the smoothness. Referenced website [2]

3.6.2 Brick Buster

Although the third game had different game mechanics from the first one, the core techniques were almost the same. What makes it different is the physics material used to make sure the ball moves in a physical pattern. Luckily this can be done quite easily through Unity, which offers such a mechanism by default. Referenced website [3]

3.6.3 Hua Rong Dao

The second game is the trickiest. It’s based on mouse dragging scripts [4] combined with some collision detection. The first challenge is to decide which is the better way to move a block. Two mechanisms were taken into consideration — key controlling and mouse dragging. The former one provides more controlled block movements while the latter one can be done without selecting which object the player would like to deal with. And after a series of testing, the latter scheme was proved to be a better choice. The original mouse dragging script was acquired online, and it had been modified to fit in our own game. Everything seemed fine, until not long after there was a bug coming up. The mouse dragging script worked by re-positioning the game object to the position of the cursor, which leaves a problem that there would be no collision detection during the re-positioning process until the block is finally landed. And as a result, the player was able to drag the block to wherever he wants regardless of surrounding blocks. In other words, this mini-game would be totally meaningless before this bug is handled. It’s an intelligent game after all, but there was no such thing as "intelligence" if the user could win the game within a second.

Considering that this annoying bug was created by the natural property of mouse dragging script, it didn't seem like a smart move to modify the script itself. Therefore, the only reasonable way other than turning to another moving mechanism is to change the collision properties of block objects. And luckily it worked to some extent — now the blocks would never be placed one above another like before. But that dragging bug remained unsolved. This troublesome bug had been haunting for a long time, until a mathematical method was applied to have it handled. It’s been mathematically proved that it takes at least 81 moves to solve this puzzle, which indicated that the target block should not reach the exit area before 81 moves. And with the help of this piece of information, a click counter was added to the game, which can stop the player from illegal winnings. This mechanism was proved useful, and added additional fun to it.

Chapter 4 – Testing

There are many various models of software development processes in a project. In our project, to eliminate the risks associated with our project, we chose to use spiral model. The spiral model views development of software as processing in a series of iterations, repeatedly visiting the same stages and activities. Therefore, when we finished one little iteration, we would do a software testing to make sure the project can work out. The software testing methods we used are as follows:

4.1 Black-box Testing

"Black-box testing treats the software as a "black box", examining functionality without any knowledge of internal implementation, without seeing the source code. The testers are only aware of what the software is supposed to do, not how it does it." [5]

Input
Blackbox
Output

T1. Black Box Diagram

4.1.1 UI Testing

In GUI part, we ran the whole program to check if all items in space were showed in correct position. When combining the building with small items, we ran the program and checked if there was a split joint issue.

4.1.2 Logic Function Testing

In database part, to test if log-in function and register function could work well, we created a new client and added information in the front-interface. After that, we checked if the information was inserted into correct table of database.

In networking part, to test whether data in one computer could be transmitted to the other computer in real time, we used two computers. When one user controlled the character in 3D interface, we checked if the other user in 2D interface could trace the character in 3D.

4.2 White-box Testing

"White-box testing tests internal structures or workings of a program, as opposed to the functionality exposed to the end-user. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs." [2]

When using white-box testing, we looked through the whole code part instead of running the program. During this testing, we checked if there were syntax errors and wrong functions. For example, when using C# to insert data to database, there might be syntax error query.

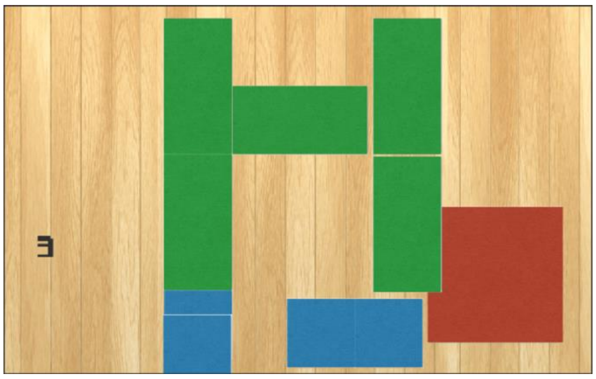
4.3 Dynamic Testing

Dynamic testing is actually executing programmed code with a given set of test cases. During the mid-term of our project, we combined four members' work together to a basic demo. We ran the demo to check if it could work and if it met our requirements.

4.4 Major bugs we discovered and overcame

4.4.1 Hua Rong Dao

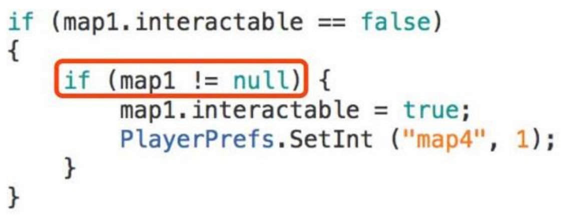
In this 2D game, player should use remaining space to move the red square to the bottom. However, in practice, player can drag the red square out fiercely. (T2) Therefore, Liu Yuhang changed this game to a trick, if user click 81 times, he/she would pass the game.



T2. Fiercely Drag in Hua Rong Dao

4.4.2 Self-destroyed Button

During the live test, we met a problem. When 3D player picked up the map, the inactive map button in 2D didn't become active. Every line of code was under logic, but it always showed the button had been destroyed. Thankfully, we found a solution online by Google (T3). Shows that it is a common bug in Unity 3D.



```
if (map1.interactable == false)
{
    if (map1 != null) {
        map1.interactable = true;
        PlayerPrefs.SetInt ("map4", 1);
    }
}
```

T3. Judge the button whether exist before use it

4.4.3 Shared Achievement Tablets

There would be nothing wrong if the character only gets one achievement for a period of time. However, when we did the UI test, if the character got two achievements really closely, one of the achievement would not show up as they shared one tablets. So specific tablets for each achievement are created, which solved this bug.

Chapter5 – Individual Contributions

5.1 Game controller, GUI, Trick Design – XU XUANHUI

I am Isaac, taking care of Game controller, GUI and Trick Design three parts in this project. And I got a lot cooperation with my group member during the process of building up our game.

5.1.1 Game Controller

The foundation of the game. It's a control system that makes players having the ability to control character and get interaction with the environment.

- Character move: Make the character move, calculate the speed direction when the character jump and allow a little operation for correction, which makes Mr. X having the ability do the tricky jump like climbing on the wall.
- Health Manager: In charge of reducing health by falling from high, and show the health state on the canvas in 3D interface. I did the math about the time our character flying in the sky for judging, but there seems having some bugs about .isGround() function. The calculation is updated by Frank.
- Achievements Tablet: Set different achievements' condition. Show the achievement tablet when the condition is achieved.
- Collision: Create colliders for objects needed.

5.1.2 GUI

All models showed in game (3D and 2D) is made by myself, mostly using MagicaVoxel. Importing models by .obj directly will slow down the speed of running the game, because this process would create a lot of unnecessary model's faces for the mash collider. In order to solve this problem, I bought a plug-in gear (10$) (I1) to generate prefabs, which saves some storage. I have also take care of the interface for UI such as login and Icon. Making the interaction more vivid for our player. Icon is designed by myself while the cursor is download from the internet which is in Dota2's theme.

I1. Plug-in Gear

5.1.3 Trick Design

One of the most active part of the game, including gates, map, trick…

- Gate & Button: There's two type of gates showing in this game. One is the type of rising up automatically when the character getting close. (Such as the main gate and the gate in front of elevator) The other one is opened by a button. (Roof gate)
- Trick & Button: There's also some tricks in the game you need to finish to get to the aim. (Like TV with button to lift the ladders, the final quiz before opening the chest or the switch to activate the elevator, C4 can be picked up and explode the wall. etc. etc.)
- Map: And there's some information paper for 3D player to pick up. After that, 2D player can read the information used for passing the game.
- In 2D scene: 2D player can click the items showing on the map and get the details.

5.1.4 INTERATE THIS WORK INTO THE OVERALL GROUP PROJECT

Because my part of work is the core and usually has a lot of changes, no matter for models or the control system, I gathered other's work packages and put them together. After this process, I sent the latest version to my group members and add functions.

At the beginning, Frank combined my origin version with the photon network. Frank clarified the code and sent it to me in the next. As for Jason's part, he packed his work packages in to a sdk file, then I imported his file in to my package directly. Cherry's code could be pasted directly with some other .dll files.

Author: XU XUANHUI

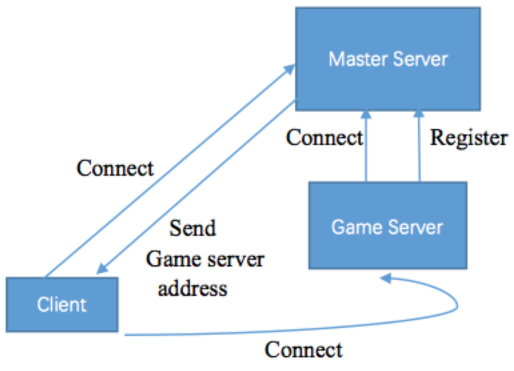
5.2 Networking – LIU YUXUAN

I am Frank who in charge of all of the network part in our project. My main task is to set up the server and let the players can transform information to each other. And I will describe how I achieve my work through three parts.

5.2.1 Server

We choose photon server as our game server, because it uses client-server model and it is a game server so it has already written a lot of useful function for our game. At first, I wanted to write the server part myself, but after a few weeks reading their document. I found that one of their sample server can satisfied all the function we need, so I decided to use this server directly. Then is the client part, photon also provide a package called photon unity networking (PUN) which is basically the rewrite of unity networking but much more convenient to use. Based on that, I write a simple client connect demo which can connect to the server on my computer by local IP address.

Here is the basic logic our server uses (I2):



I2. Server Logic

5.2.2 Lobby&&Room&&Position transform

We use lobby and room structure in our game in order to give players a choice to choose the partner. After every player connect to the server, they will atomically join a lobby. And in this lobby, they can choose to create their own room or join a room. In addition, the room property has been set to support 2 players max since our game is for two players. So when a room has two players, other player can't join this room. Player can choose whether to play 2D part or 3D part after they join a room. However, that brought a problem that I need to make sure they won't choose same side. I will talk about it in the next part. Then, I use photon view (a script to replace the network view in Unity) in 2D and 3D scenes. By setting a same view ID so they can know which object need to be synchronized. A big problem occurred in this section, because photon view and the transform function was written for synchronization in same scene and our game was in different scenes. Thank to Isaac's help, and finally we change the data which send by 3D scene to make it right in 2D scene.

5.2.3 RPC&&Additional Function

After the position transform, we also need to transform information in game in order to let the player can cooperate to the other player. To achieve that, I used RPC (Remote Procedure Calls). The function RaiseEvent can send data to the player in the same room. And OnEvent function can get the event and do different thing according to the data it received. That also solved the problem if two player want to play same side. When the first player joins the room and choose the side, it will send an event to the room and stored in room cache. Other player joins the room they will receive the cache event, and they will know the previous player choice and this player cannot choose the same side. And all of other interaction between 2D and 3D also use the way to achieve.

After I finished these things, we test our game and find that it was really difficult and confused if 2D player cannot communicate with 3D player. So I use photon voice package to achieve this function in our project.

5.2.4 INTERATE THIS WORK INTO THE OVERALL GROUP PROJECT

Because network is one of the most important part in our game. I need to make sure it will work well in our game instead of just in my demo. So I always test if it will work well in Isaac's main game. We change our code together to make that suitable for our game and Isaac also help me a lot in the position synchronization.

Author: LIU YUXUAN

5.3 2D Games, Sound Effect – LIU YUHANG

My main work covers all three mini-games which are planted into our project. The way they work requires the communication between 3D game scene and 2D game scene. When the 3D player triggers a button/lever in the 3D game scene, there should be a mini-game popping up (by using loadLevel() functions in scripts) in 2D game scene. And the 2D player are ought to pass the game in order to help the player in 3D game scene. And this is done by providing clues which are useful to solve puzzles to the 3D player or by triggering hidden game objects with which the 3D player can interact.

5.3.1 Space Shooter

The first game is a simple and classic space shooter game. The user controls the spaceship with W/A/S/D or arrow keys and shoots with ctrl/left mouse button to eliminate all the enemies. The enemies also shoot bullets and are moving towards the player, which leaves limited time for the user to destroy them all.

5.3.2 Hua Rong Dao

The second game is a traditional Chinese intelligent game called Hua Rong Dao with hundreds of years of age. The goal of the game is to move the target block to the exit area by moving all the blocking blocks away. Frankly speaking, this game might be too challenging to play as a sole puzzle game in our project. And that’s why I built a "backdoor" to it. Considering it has been mathematically proved that it requires at least 81 moves to win, I set a click counter in it which records how many times the player has clicked, which can be seen as how many moves the player has already made. And here comes the tricky part: the player can drag the target block straight to the exit area after 81 clicks, and this cannot be done before making 81 clicks.

5.3.3 Brick Buster

The third game is also a classic one which might take you back to your childhood — it’s a brick buster game! And almost everyone knows how it’s played! The player simply needs to move the racket (term for that panel-like stuff that bounces the ball) by left/right arrow keys and bounce the ball in order to destroy the bricks. A important clue which can lead the 3D player to the final vault is hidden underneath the bricks. Mr. X is counting on Ms. L now!

In addition to the mini-games, I’m in charge of the sound effects /background music of our game as well. And we’re really proud to say that all the sound effects are made by ourselves — Isaac and I did some dubbing work, to be exact. But unlike the sound effects, the background music pieces were found on some online music resource sites.

5.3.4 INTERATE THIS WORK INTO THE OVERALL GROUP PROJECT

What I did to these music pieces was to simply add them into our game scenes. (The games imported process is in Isaac's document above)

Author: LIU YUHANG

5.4 Storage – XU CHEN

There are several different mechanisms, such as cloud, local, shared and things like that, used in application. In our game, we chose to use shared mechanism.

Notorious Duo is a 2-player game, which means transmitting data in real time is vital importance. It is quite complicated to duplicate files to each user computer. Therefore, by using shared mechanism, only one file server exists containing all the data, backup and archiving processes are streamlined, and the risk of conflicting backups or archives is eliminated.

There are two main tasks that I am in charge of, one of which is creating database, and the other is implement data input and output.

5.4.1 Database Create

For database part (I3), I used MySQL to create database. In our game, we only need two tables, which are users table and achievements table.

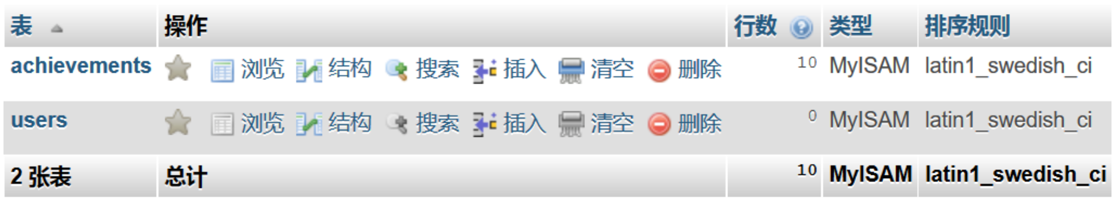
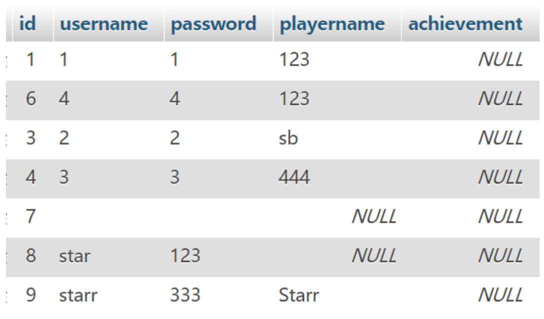


表	操作	行数	类型	排序规则
achievements	浏览 结构 搜索 插入 清空 删除	10	MyISAM	latin1_swedish_ci
users	浏览 结构 搜索 插入 清空 删除	0	MyISAM	latin1_swedish_ci
2 张表	**总计**	**10**	**MyISAM**	**latin1_swedish_ci**

I3. Database Structure

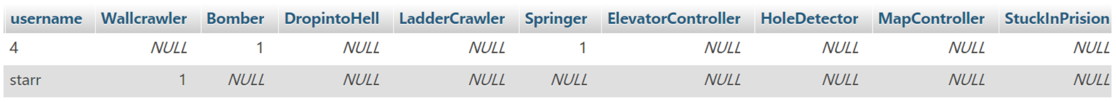
In users table (I4), it stores users' information in detail, which are id, username, password, player name and achievement. Id is the primary key for this table and it will be increased by one when a new user register.



id	username	password	playername	achievement
1	1	1	123	*NULL*
6	4	4	123	*NULL*
3	2	2	sb	*NULL*
4	3	3	444	*NULL*
7			*NULL*	*NULL*
8	star	123	*NULL*	*NULL*
9	starr	333	Starr	*NULL*

I4. User Table

In achievements table (I5), it stores which user get which achievements. When user get a new achievement, the data in this table will be changed in '1'. Otherwise, it will remain 'NULL'. Username is the primary key in this table.



username	Wallcrawler	Bomber	DropintoHell	LadderCrawler	Springer	ElevatorController	HoleDetector	MapController	StuckInPrision
4	*NULL*	1	*NULL*	*NULL*	1	*NULL*	*NULL*	*NULL*	*NULL*
starr	1	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*

I5. Achievements Table

5.4.2 Database Implementation

As for implementing data part, I used C# to implement all classes that are needed in Unity-3D. There are two classes in this project. In CMYsql class, firstly, I connected our database with Unity-3D. Then I added register and log in method in this method to make sure that users' information can be inserted into database. Besides, players can also add their nickname to database. In AchievementController class, the find() method is used to insert data into achievements table. Therefore, player can easily check which achievement they get.

5.4.3 INTERATE THIS WORK INTO THE OVERALL GROUP PROJECT

At the very beginning of our project, we could not completely foresee how much work do we need, so I only made a basic framework of storage part, which were database structure, how many tables did we need, which kind of method did we need. After Liu Yuxuan made a basic networking structure, we had a discussion and decided to use shared storage mechanism, so I deleted the localhost command. With his help, I solved the problem that users in LAN could not have access to database successfully.

Creating database is quite easy, but if it cannot be connected with the project, it will make no sense for our game. Therefore, I cooperated connection part with Xu Xuanhui who was in charge of Front-end interface. With Xu's help, I successfully connected out database with Unity-3D. After that, we discussed how and in which way the achievement interface showed up. He designed all GUI items and I implemented all methods.

Author: XU CHEN

Chapter 6 - Conclusion

In conclusion, what we did was to build one game that runs on two different PCs at the same time based on Unity game engine with photon networking framework. Each game round runs on two separate computers, one was featuring 3D and the other was 2D. But the server was able to support more players. For each game round, players in both game screens should cooperate with the other one in order to achieve the final goal of the game. The player in 3D game scene was positioned in a huge maze, while the player in 2D are able to give him directions because the 2D game scene provided bird views of the entire maze. Also, some interactions between the character and game objects would cause mini-games popping up in the 2D game scene. And the player on 2D side had to pass the game so that the player on 3D side could continue his journey. Achievements systems were also available; the player would be awarded achievements by doing some certain things.

the biggest challenge during our implementing process was to get the photon networking framework working, which servers as the backbone of our game. It really was a time-costing process to figure out how it exactly works due to its complexity. Another impressive challenge was setting the camera view in 3D game scene. Apparently the camera will have to stick to the character in the game scene and follow him wherever he goes to provide a good third-person view. But there was another problem. When the player was trying to move his mouse to look around, player’s view turned but the camera had to maintain a reasonable distance to the character so the game scene would not be blocked by the character. The player would like to see the game scene, not the butt of the character he’s controlling. Luckily, we finally managed to find proper solutions on the internet. And only after this problem was solved when our game looked like a real game.

Finding out problems and trying to fix them was a painful job, but it was these problems that made our work precious. We’re now confident to say that our game supports in-time communication, and it’s more than just sending game information. In the final version of "The Notorious Duo", players on each side are even able to talk to each other through a voice system. In addition, we’re more than proud to announce that all game objects, including game scenes were designed by ourselves. It cost a lot of time, but it was all worthy when we finally saw what we had accomplished.

Of course the project is still far from perfect. There are dozens of cool stuffs we’d like to try to add to our project if we had more time. For example, we were thinking about adding enemies(guards) attempting to attack the character in 3D game scene while the player on 2D side is trying to crack the puzzle or trying to win a mini-game. And the 3D player could be given more available animations and movements in addition to running and jumping. What's more, we’d like to design a longer storyline with more complex gameplay. And these are just two very primitive ideas about how to improve our game among dozens of creative thoughts!

Developing "The Notorious Duo" as a group project was an amazing experience for all of us. It offered us a great opportunity to have a deeper and further understanding of software developing process. Besides, the importance of teamwork was emphasized like never before, due to the great complexity of the project. And this could be much more beneficial to us than simply writing lines of codes in the long run.

We really appreciate your patience and support! Your help was most valuable and precious, and we’re more than grateful to have your guide during our development.

References

[1] Standard Assets: https://www.assetstore.unity3d.com/cn/#!/content/32351

[2] Space Shooter: http://pixelnest.io/tutorials/2d-game-unity

[3] Brick Buster: https://noobtuts.com/unity/2d-arkanoid-game

[4] Hua Rong Dao – Mouse Drag: http://blog.csdn.net/tc1hen/article/details/8459700

[5] Black Box Text: https://en.wikipedia.org/wiki/Software_testing#Black-box_testing

[6] Master Server:

https://doc.photonengine.com/enus/onpremise/current/applications/loadbalancing/application

Appendices

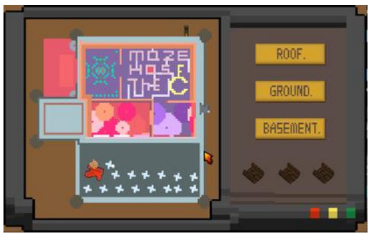
Game Instructions

3D player:

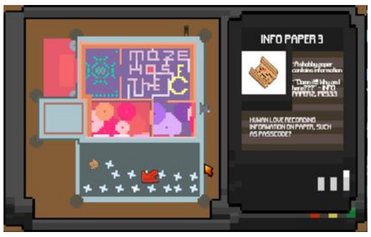
- W, A, S, D to move Mr. X.
- Move mouse to change the face direction.
- E to pick up maps or switch on button.
- Space to jump.
- Click the button on right top to check achievements. (R5)

2D player:

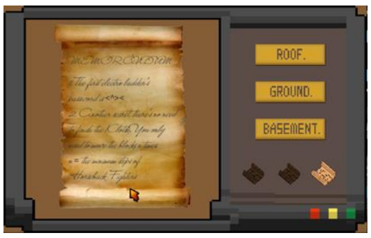
- Click big yellow buttons to change the view point of the castle. (A1)
- Click the items on the screen to see the details of it. (A2)
- Check the paper's information after Mr. X collected it. (A3)



A1



A2



A3