System Architecture Style

The MVC (Model View Controller) is used in our software architecture design. In the usage scenario of users using VR glass to play game, every VR glasses are responsible for displaying the information of game to every users as the view. The model process the storage of game’s data and the central data of the system. The Controller Handles the information entered by the user. Being responsible for reading data from the view, controlling user input, and sending data to the model, which is the part of the application that handles user interaction. It is responsible for managing user interaction control.
The Layered architecture is used in our software architecture design. System is divided into several horizontal layers with clear roles and divisions. In our system, there are 4 layers, Mobile Edge System Layer, Mobile Edge Host Layer, MEC User Equipment and Cloud Center. Layers communicate with each other through the interfaces. Each layer provides a higher level of service to the next. The key to the layering pattern is to determine dependencies: by layering, we can limit dependencies between sub systems, makes the system more loosely coupled and easier to maintain. The purpose of the layered architecture is to attain the idea of high cohesion and low coupling.
We also use Client-server to design our software architecture. In the model, every VR equipment is a client, every MEC user equipment is a server. When the game is running, we are using VR glasses to do some operations, then the VR equipment will send messages to a MEC server to get graphics and information processing. Finally, the MEC will calculate the date and send back some graphic and information. There are some reasons we choose the style. Frist, it is mobility, on other words, it easily supports client mobility. We all know that it is important to allow players to play the game in lots of places. If we move VR glasses to other areas, then they are offline, it will be a bad experience. Second, the security typically controlled at server, also possible at application/business layer. It means it is very hard to destroy the game, because we have lots of MEC server, if one is damaged, we could connect to another which is close to us. Third, it is easy to add new servers or upgrade existing servers. It is useful because we have to create lots of MEC servers and update them every once in a while. Finally, it allows sharing of data between multiple users which could support our game running. We have 50 players in a game.

System Architecture Principles

The main six principles of system architect are all used in our system.

1. Open-closed principle
   Our system is open to add new functions but is closed to change past functions. You can see from maps in the games. If game adds new maps, we just need to overwrite the constructor of Games. Different layer implements the same interface but have different functions can also show this principle.
2. Dependence Inversion Principle
   There are also lots of examples of this principle. For example, if we need to add a new skill2 to a hero. W e don’t need to change the skill’s code in Heroes class. We only need to add a new Skills, and use it in Heroes. This decrease the dependence of each class.
3. Simple Responsibility Principle

This can be seen in Games and MainWindow. We didn't put them into a class because if we want to change the login frame or wait frame, we didn't need to change them in a class with game frame which may cause some bugs. Each class is only responsible for one function of reality.

4. Interface isolation principle
   In our system, Heroes only calls HeroesI, skillsI, and the GraphicsI interface, which covers the hero's skills, properties, and type functions, skillsI for the hero's skills, and the GraphicsI interface for the model. These are the interfaces that are closely related to Heroes, but some of the unrelated interfaces will not be called. The interface isolation principle uses the design idea of high cohesion and low coupling, which makes the classes more readable, extensible, and maintainable.
5. Demeter's rule
   In our system, there are three layers for data processing, transmission and storage, and the Upload and download functions are directly called in different layers to realize, which makes the objects keep the least understanding of other objects, thus reducing the coupling between classes.
6. The principle of substitution
   There are some combinations of subclasses and super classes. Subclasses can implement abstract methods of the parent class, but cannot override non-abstract methods of the parent class. Subclasses can add their own methods. When a subclass overloads a method of a parent class, the input parameters of the method are looser than the input parameters of the parent class. When a subclass's method implements a superclass's method (overrides, overloads, implements abstract methods), the output, return value of the method is more stringent or equal than the output, return value of the superclass.

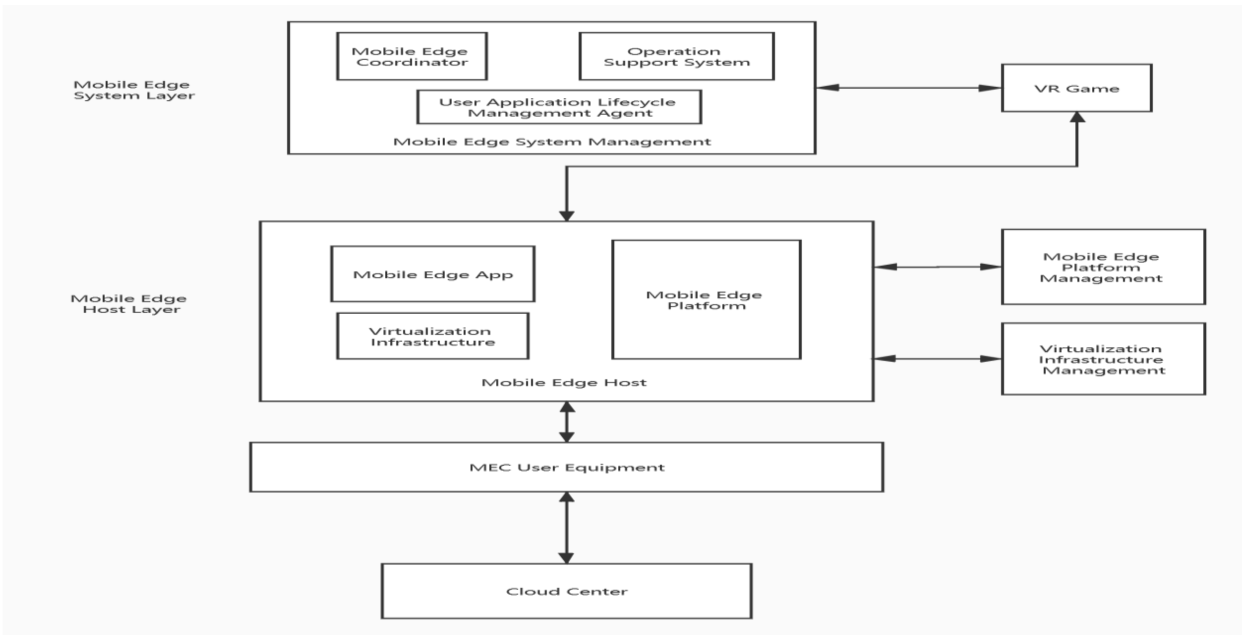
System Architecture Design Pattern

Skills Class: Factory Method pattern, Composite pattern
The reason for using Factory Method pattern to design 'Skills' Class is that this pattern has high expansibility which can increase the number of skills conveniently; It is easy to create objects; It don't need to care about the implementation of the class, only to care about the interface.
The reason for using Composite pattern to design 'Skills' Class is that this class uses both Graphics and Skills Interface, and using composite pattern enables code to work consistently with individual and composite objects; It is easier to add new objects to the composition.
Map Class: Singleton pattern
The reason for using Singleton pattern to design 'Map' Class is that this class only need to create one entity, and by using this pattern, we can avoid report the error of generating multiple maps in the game.
Hero Class: Factory Method pattern, Composite pattern
The reason for using Factory Method pattern to design 'Hero' Class is that this class need to create multiple objects, and this pattern has high expansibility to do this.
The reason for using Composite pattern to design 'Hero' Class is that this class uses many objects of other classes, so composite pattern enables code to work consistently with composite objects, and it is easier to add new objects to the composition.

Game Class: Factory Method pattern, Composite pattern, State pattern, Command pattern
The reason for using Factory Method pattern to design 'Game' Class is that in this class there are multiple objects and each game need to create a new entity. Using Factory Method pattern we can create different game for different group of players simultaneously.
The reason for using Composite pattern to design 'Game' Class is that this class uses objects of 'Hero' and 'Map' classes, which means that composite pattern enables code to work consistently with composite objects, and it is easier to add new objects (heroes operated by players and terrains in the map) to the composition.
The reason for using State pattern to design 'Game' Class is that in the game we need to determine the state of each player. We need to calculate the health value of each hero and then judge that wether he is alive or dead. Also we need to confirm which player wins the game (the player who comes to the peak first), by using state pattern we can judge this circumstance.
The reason for using Command pattern to design 'Game' Class is that in the game players need to input order (like up, down, left, right) to operate heroes. By using command pattern, users can operate heroes with the command object, which facilitates the storage, delivery, invocation, increase and management of the command object.
User Class: Factory Method pattern
The reason for using Factory Method pattern to design 'User' Class is that there will be many user accounts, and each user accounts is a different object. Factory Method pattern is proper for this kind of user system.
MainWindow Class: Facade pattern
The reason for using Facade pattern to design 'MainWindow' Class is that in our main window there will be different functions, and the main window is the interface of these functions, where Facade pattern is a pattern that makes multiple complex subsystems more accessible by providing a consistent interface.

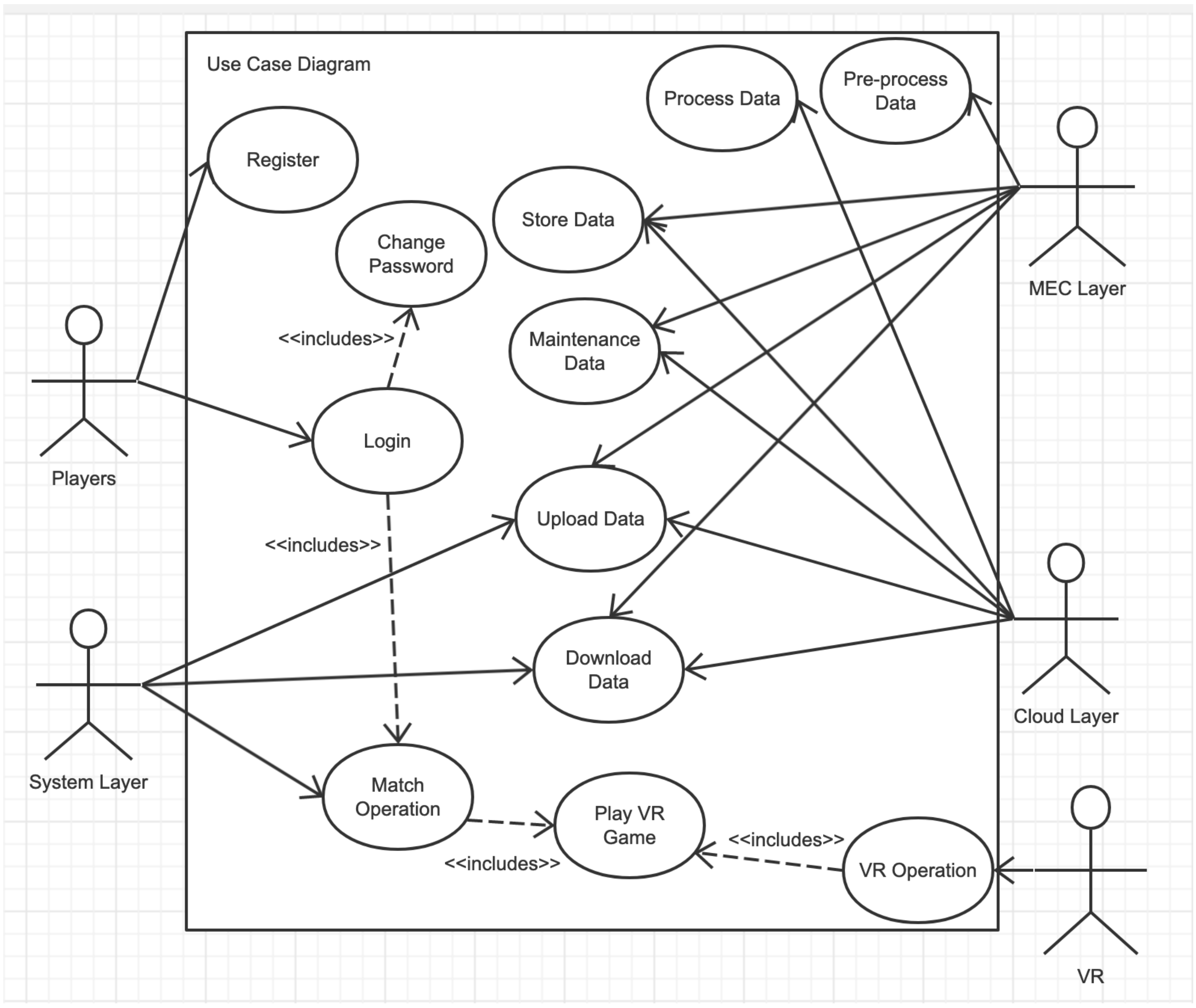
System Architecture Design Documentation

Deployment Diagram



In the deployment diagram, there are 4 layers, Mobile Edge System Layer, Mobile Edge Host Layer, MEC User Equipment and Cloud Center. In the Mobile Edge Host Layer, Mobile Edge System Management is supporting VR game. In the Mobile Edge System Management, there are Mobile Edge Coordinator, Operation Support System and User Application Lifecycle Management Agent. In the Mobile Edge Host Layer, Mobile Edge Platform Management and Virtualization Infrastructure Management are supporting Mobile Edge Host. In the Mobile Edge Host, there are Mobile Edge App, Virtualization Infrastructure and Mobile Edge Platform. The data will transfer to MEC User Equipment and the MEC server will calculate some of the data and send it back. Others without unnecessary will transfer to Cloud Center which will calculate some of the data and send it back.

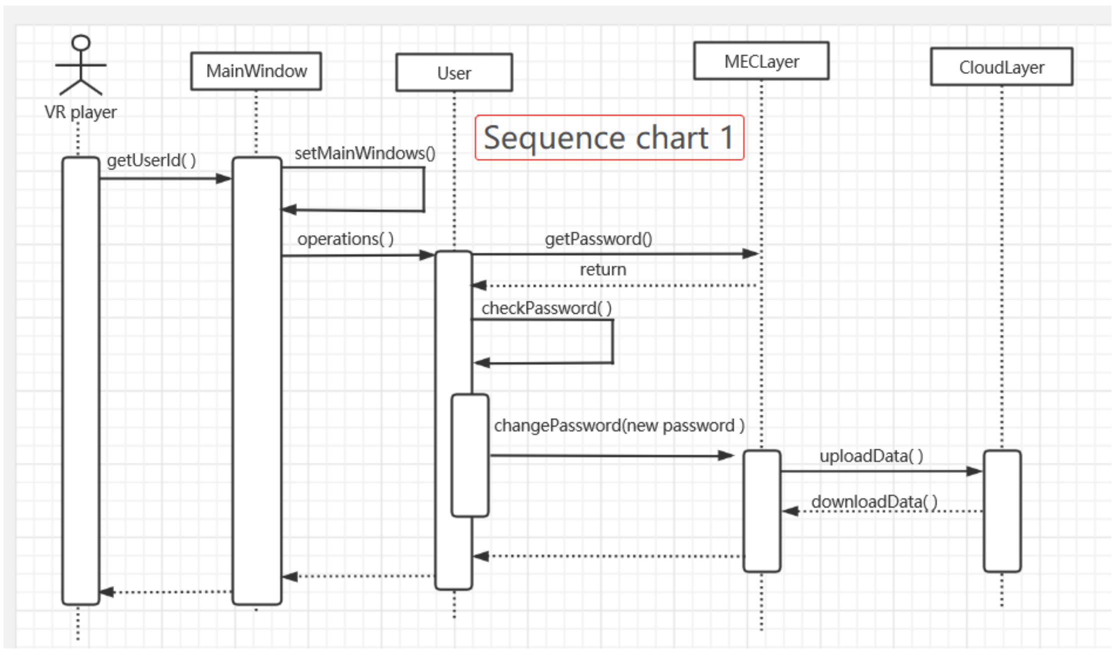
Use Case Diagram



The players can register for the game;
The players can login the game with their accounts;
The players can change the password after logging in;
The players can enter the matching queue after logging in;
The players can play VR games after matching;

The system layer can match games for players;
The system layer can upload the data to database;
The system layer can download the data from database;
The MEC layer can upload the data to database;
The MEC layer can download the data from database;
The MEC layer can store the data;
The MEC layer can maintain the data;
The MEC layer can preprocess the data;
The cloud layer can upload the data to database;
The cloud layer can download the data from database;
The cloud layer can store the data;
The cloud layer can maintain the data;
The cloud layer can process the data;
The VR can provide the VR operation to the game.

Sequence Chart

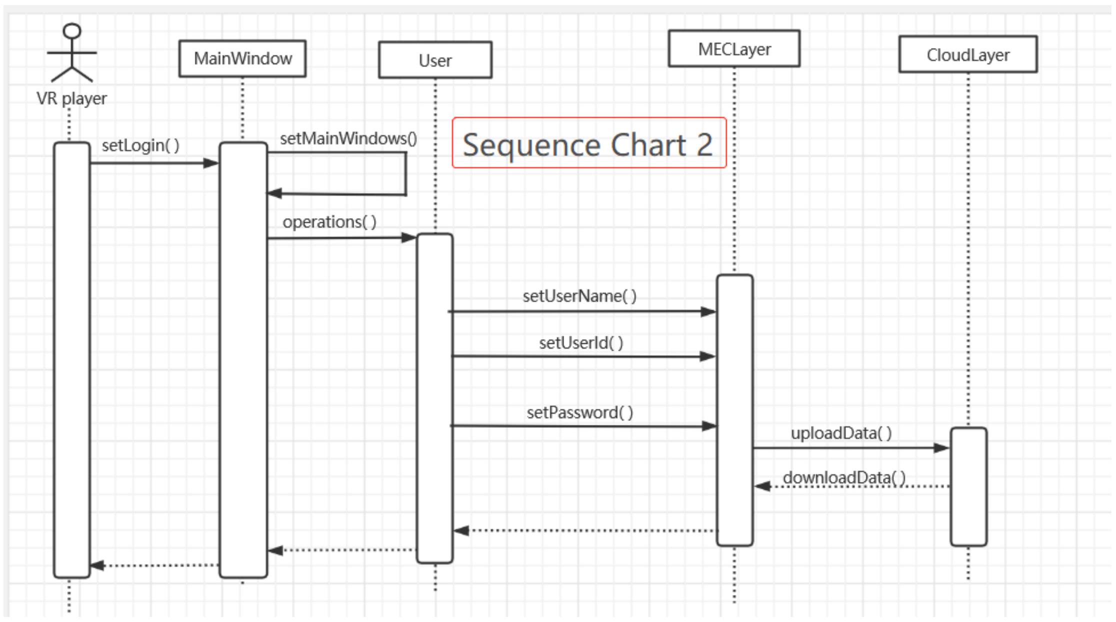


Change password:

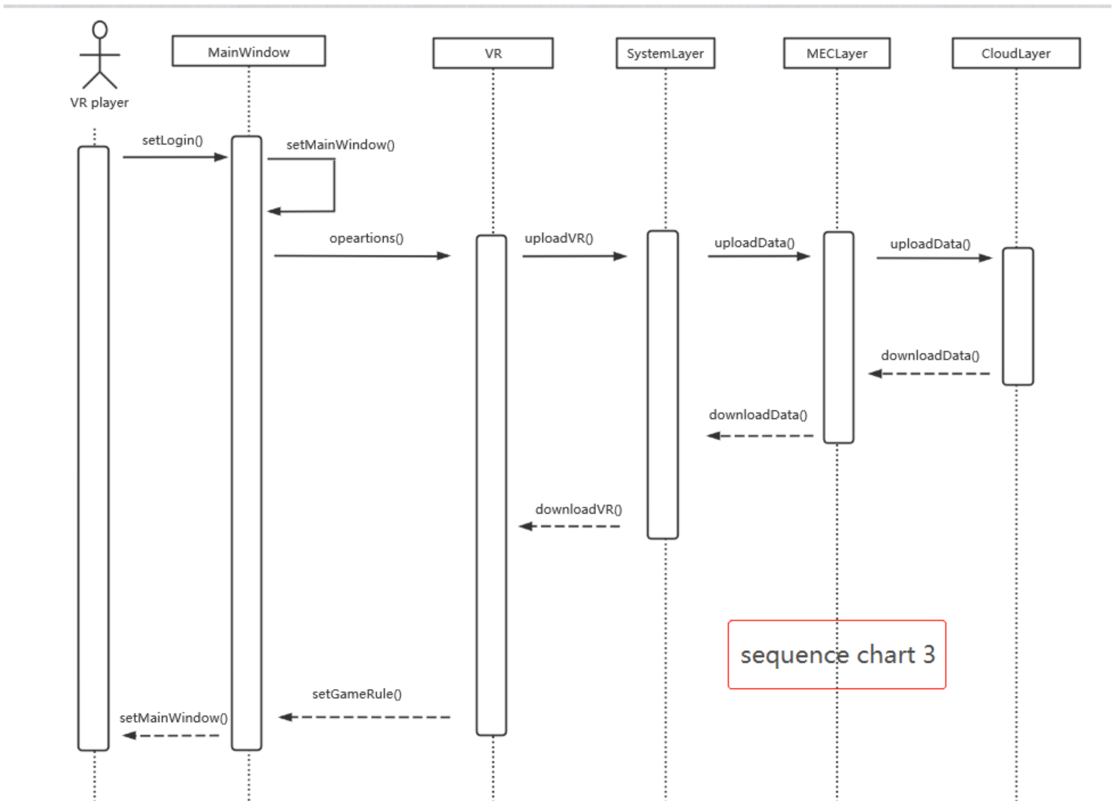
First, we will get the user input ID, then activate and enter the main interface, the system through the user input ID from the database to get the user's original password.

It then detects that the password entered by the user matches the original password, and if the match changes the original password in the database with the new password.

First, the data is packaged through MECLayer and sent to CloudLayer..

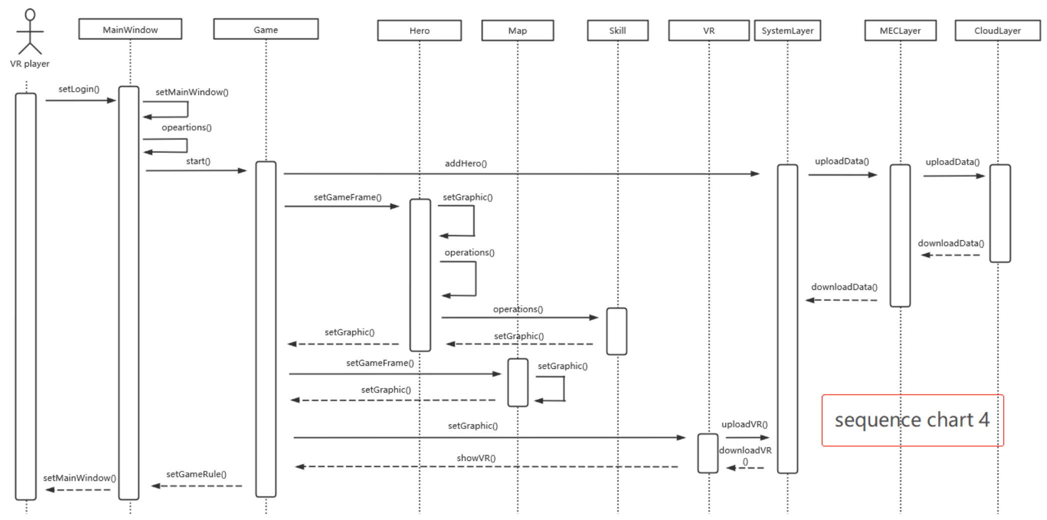


Register:
First, activate the user login window. The user can select registration in the login window and enter the registration page. Then, like user request ID, UserName and Password, the data is transferred to MECLayer, which is packaged by MECLayer and finally transferred to CloudLayer for database modification.



View the Game Rule:
VR player enters the main window after successful login and click to check the rules of the game. The game rules stored in the cloud database will be returned to the MEC database and

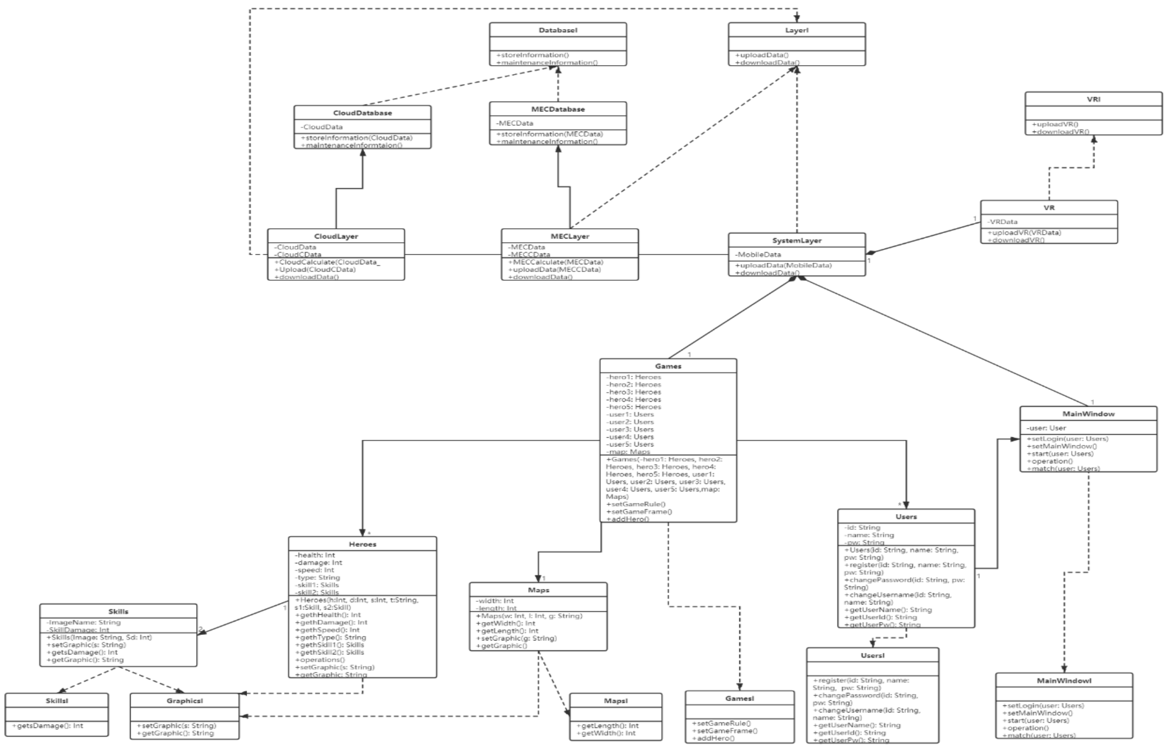
then returned to the game.



Play the VR Game:

After the VR player logs in successfully, the main window is loaded. Click "start game" to match the game. After selecting the hero, add the selected hero in the system layer. The data is updated to MEC layer. MEC returns the processed data and loads the game screen. The game screen corresponds to the loading of map screen, hero screen and skill screen respectively. The loading of game screen is displayed in VR glasses. During the game, data interaction occurs with MEC layer at any time (MEC terminal obtains some data from the cloud). The game is played according th the rules of the game until the game is decide to end. When game is over, VR player returns to the main window.

Class Diagram



The GraphicsI is an interface that let other classes call graphics. It has 2 functions. setGraphic() is to set a graphic. getGraphic() is to output a graphic.

The SkillsI is an interface that other classes could implement it to describe skills. It has 1 functions. getsDamage() is to let others know how much damage a skill has.

The Skills is a class that describes how a skill relates with heroes. It has 2 attributes and 3 functions. ImageName is name of the image which the hero will act when he/she use the skill. SkillDamage defines how much damage a skill has. setGraphics() is to set a graphic when a hero uses a skill. getsDamage() is to let others know how much damage a skill has. We use getGraphic() to output a graphic when a hero uses a skill.

The MapsI is an interface to define what functions Maps class has.

The Maps is a class to describe maps in the game. It has 2 attributes and 5 functions. width and length define the size of a map. getWidth() and getLength() could return width and length of a map. setGraphics() is to set a graphic of a map, getGraphic() is to output a graphic of a map.

The Heroes is a class to describe the state of a hero. It has 6 attributes and 9 functions. health, damage, speed is to define the health, aggressivity and speed of a hero. type defines what occupation the hero is, an assassin or a master? skill1 and skill2 are two skills that a hero has. The first 6 functions are to get health, aggressivity, speed, type and two skills of a hero. We use operations() to control actions of a hero. setGraphics() is to set a graphic of a hero, getGraphic() is to output a graphic of a hero.

The UserI is an interface to define what functions Users class has.

The Users is a class to record information of a user. It has 3 attributes and 6 functions. The id is a unique key different with others, name is the name of a user in the game and pw is the password of the user. register(), changePassword(), changeUsername() let users register, change password and change username. The let 3 functions could get name, id , pw of users.

The GamesI is an interface to define what functions Games class has.

The Games is a class to describe how the game will run. It has 11 attributes and 3 functions. A team should have 5 users connecting to 5 heroes. The map is to define what map the game uses. setGameRule() is to set rules of the game. We use setGameFrame() to show the game to users. addHero is to increase heroes to the game.

The MainWindowI is an interface to define what functions MainWindow class has.

The MainWindow is a class to let users play the game. It has 1 attribute and 5 functions. The user means who play the game. setLogin() allows user to login the main window. start() allows the game start. operation() allows interface interactive. match() allows matching teammates and opponents.

The VRI is an interface to define what functions VR class has.

The VR is a class represent VR glasses. It has data VRData. uploadVR() is to give data to other classes, downloadVR() is to get data from other classes.

The LayerI is an interface to define what functions other layer classes has.

The SystemLayer is a class represent the service portal of UE app and customer facing service in mobile edge system level. It has data MobileData. uploadData() is to give data to other classes, downloadData() is to get data from other classes.

The MECLayer is a class represent the platform running on the edge of mobile network. It has data MECData and data after computing MECCData. uploadData() is to give data to other classes, downloadData() is to get data from other classes, MECCalculate() is to calculate data to become MECCData.

The DatabaseI is an interface to define what functions other database classes has.

The MECDatabase includes all the data MECData in MECLayer. The functions are using to store and maintenance data.

The CloudLayer is a class represent Cloud Center. It has data CloudData and data after computing CloudCData. uploadData() is to give data to other classes, downloadData() is to get data from other classes, CloudCalculate() is to calculate data to become CloudCData.

The CloudDatabase includes all the data CloudData in CloudLayer. The functions are using to store and maintenance data.

You can see the HD diagrams in the Pictures category!!!!!!!!