DAWN

Requirement Define

Version 1.0

**By:**

Dawn Group

2019-04

**Group Member:**

Zihan Xu

Yi Kuang

Chenyu Yang

Yuting Lan

Jianzhen Cao

**Document Language:**

English

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 2019-4-2 | 1.0 | Finish the 1st edition of feasibility analysis report | Zihan Xu, Yi Kuang, Chenyu Yang, Yuting Lan, Jianzhen Cao |
| 2019-4-14 | 2.0 | Unified the name of some class | Zihan Xu, Yi Kuang, Chenyu Yang, Yuting Lan, Jianzhen Cao |
| 2019-4-20 | 2.1 | Modified some use case specification. | Zihan Xu |
|  |  |  |  |
|  |  |  |  |

目录

[1 Introduction 1](#_Toc23653)

[1.1 Purpose 1](#_Toc22591)

[1.2 Domain 1](#_Toc30178)

[1.3 Definition 1](#_Toc13331)

[1.4 Reference 1](#_Toc5161)

[1.5 Overview 1](#_Toc17430)

[2 Existing System 1](#_Toc9004)

[3 Proposed System 2](#_Toc23300)

[3.1 Introduction 2](#_Toc3871)

[3.2 Functional Requirement 2](#_Toc3863)

[3.2.1 <Functional Requirement 1: Platform > 2](#_Toc27843)

[3.2.2 <Functional Requirement 2: Gameplay> 2](#_Toc25528)

[3.2.3 <Functional Requirement 3: Sociability> 3](#_Toc5342)

[3.3 Non-functional requirements 3](#_Toc11659)

[3.3.1 Usability 3](#_Toc21656)

[3.3.2 Reliablity 4](#_Toc27620)

[3.3.3 Performance 4](#_Toc8598)

[3.3.4 Supportability 5](#_Toc27376)

[3.3.5 Design Constrains 6](#_Toc30723)

[3.3.6 Interface 6](#_Toc23571)

[3.3.7 Legal, copyright and other notices 7](#_Toc31221)

[3.3.8 Applicable standard 8](#_Toc15948)

[3.4 System Model 8](#_Toc25283)

[3.4.1 Senario 8](#_Toc32411)

[3.4.2 Usecase Specification 11](#_Toc13899)

[3.4.3 Objective Model 19](#_Toc14529)

[3.4.4 Active Model 24](#_Toc4928)

[3.4.5 User Interface 37](#_Toc720)

Requirement define

# 1 Introduction

## Purpose

This is a requirement specification document. In this document, we will define most of the system requirement, so that all the develop team member can have a clear picture of the whole system.

## 1.2 Domain

The whole game system.

## 1.3 Definition

See more details in glossary document.

## 1.4 Reference

*Feasibility Analysis Report*

*Object-Oriented Software Engineering--Using UNL, Patterns, and Java (Third Edition)*

## 1.5 Overview

This document is divided into two parts, one of which is existing system and the other is proposed system. In the existing system part, we will show something about battle royal game. Both functional and non-functional requirement will be defined in the proposed system and system models that contain Scenes, use case information, object models and dynamic models are also included.

# 2 Existing System

Now there are many battle royal games both in personal computers and mobile phones. The most famous one is PLAYEREUNKNOWN’S BATTLEGROUNDS in personal computers, which now has its pocket edition. It has a game model that asks player to parachute, search, eliminate other players and survive to be the last one. Its popularity leads numbers of game makers making similar game with the game model and different character model. The game model is great, but just copy it can’t get respect from players, which means the game makers need to add their unique elements. That’s why Fortnite and APEX stand out in these games. The Fortnite combine the battle royal model and sandbox model, which gives player much more playability. The APEX add skills into the existing game system, which makes the battle more unpredictable and exciting. In our game, we change the game model slightly and add some more elements. More details about our game will be shown in other documents.

# 3 Proposed System

## 3.1 Introduction

DAWN is a game based on local area network which is often played together with friends, classmates or companions in parties.

## 3.2 Functional Requirement

### 3.2.1 <Functional Requirement 1: Platform >

DAWN wants to provide a platform on which in a party some people can start a game conveniently and quickly, which means if there are some people sitting together, that is to say, they are not very far from each other, then they can create a local area network and they all join into the network to get into a shared game.

For example, one of them can create a hot spot and all others can join into the local area network. In this way the game can start in a short term, no network delay and no matching time and so on.

### 3.2.2 <Functional Requirement 2: Gameplay>

DAWN provides a vast occupation system and a great game setting. DAWN provides 4 occupations which has different quality and skills to enrich the game experience. In this game the four occupations have no absolute advantages but is relatively more powerful than others in certain aspects so players can enjoy different occupations fairly. Besides that, we design a rich game setting including stages and properties. Players can use different properties to realize different effects and different stages the game has different settings.

### 3.2.3 <Functional Requirement 3: Sociability>

DAWN provides a chance for players to find like-minded players. In the near future, we will develop a complete system for social activity. People can enjoy the game itself and at the same time making friends with each other.

## 3.3 Non-functional requirements

### 3.3.1 Usability

Many factors can affect the usability of our project, for instance, (1) time for a user to learn this game (2) how we can develop our project according to the existing system (3) Requirements for compliance with recognized usability standards such as IBM's CUA standard and Microsoft's GUI standard.

#### 3.3.1.1 <Usablity 1 >

The time for a user to learn this game can largely determine whether this game will be welcomed by players. We hope to make a breif guidance, just like many game do, to train our users to be proficient. Besides, we will need to make the buttons more explict to help our users better understand the rule of our game.

#### 3.3.1.2 <Usablity 2 >

That we can develop our project based on the existing system can determine the efficiency of our project development. If we can reuse some existing codes or system, the whole developing process can become much more easier. Becides, this way of developing can make our code more easy to understand, thus more convenient to maintain.

#### 3.3.1.3 <Usablity 3 >

Requirements for compliance with recognized usability standards such as IBM's CUA standard and Microsoft's GUI standard is esesential for our development.

### 3.3.2 Reliablity

The reliability of our system should include: (1) system opening hours and operating permissions. (2) mean time between failures, mean time to repair, and accuracy requirements. (3) the highest code error rate, usually expressed as bugs per thousand lines of code (bugs/KLOC) or bugs per function point (bugs/function-point) (4)error rate, classified by small error, large error and serious error. "Serious" errors must be defined in the requirements, such as complete loss of data or complete inability to use some part of the system.

#### 3.3.2.1 < Reliablity 1 >

Our system opening hours should be 7x24 and the operating permissions should be attributed to all team members, so that all members can maintain the system.

#### 3.3.2.2 < Reliablity 2 >

Mean time between failures: 1000 hours;

Mean time to repair: 1 hour;

#### 3.3.2.3 < Reliablity 3 >

Highest code error rate, usually expressed as bugs per thousand lines of code (bugs/KLOC) or bugs per function point (bugs/function-point): 30b/KLOC

#### 3.3.2.4 < Reliablity 4 >

Small error: 20 b/KLOC

Large error: 5 b/KLOC

Serious error: 1 b/KLOC

### 3.3.3 Performance

The performance of our system should include: (1) response time to transactions (average, maximum) (2) throughput, such as the number of transactions per second (3) capacity, such as the number of customers or transactions the system can hold (4)degraded mode (acceptable mode of operation when the system is degraded in some form) (5) resource utilization, such as memory, disk, communications, etc

#### 3.3.3.1 < Performance 1 >

The average response time of the system is 0. 5s is appropriate, and the maximum response time should not exceed 2s

#### 3.3.3.2 < Performance 2 >

This platform is designed for anyone with Android device. Considering the actual situation, the number of requests processed per second is less than 100. Therefore, this system throughput is not big, the concurrency number request is not high

#### 3.3.3.3 < Performance 3 >

For the scope of application of the system, the number of all games is 10000, the number of online users is 100, and the number of registered users is not more than 1000.

#### 3.3.3.4 < Performance 4 >

When the system can not work in the best performance mode due to some conditions, the need to enter the degraded mode. In the de-escalation mode, we consider the system to be able to carry 1/10 of the original design load and still be acceptable. At this time, the demand for hardware and communication is greatly reduced, and the most basic demand can be met.

#### 3.3.3.5 < Performance 5 >

The initial prediction of the system load is not large, is not expected to exceed 10000 records, memory occupation is not more than 100MB. Disk usage is not expected to exceed 1GB. For network requests, we assume that the average page request is 10Kb and 20 requests per second, so we expect to need about 0.5mbp s of bandwidth.

### 3.3.4 Supportability

This section should list all the requirements that will improve the supportability or maintainability of the built system, including coding standards, naming conventions, class libraries, maintenance access, and maintenance utilities.

#### 3.3.4.1 < Supportability 1 >

Coding standards and naming conventions. See the code and programming standards guide to programming and coding styles.

#### 3.3.4.2 < Supportability 2 >

Maintain access rights and utilities. Maintenance access is restricted to maintenance personnel only, and users shall not increase their permissions without authorization. The maintenance utility program shall be saved and used by the maintenance personnel and shall not be distributed at will.

### 3.3.5 Design Constrains

This section should list all design constraints for the system being built. Design constraints represent design decisions that have been approved and must be followed. These include software languages, software process requirements, specification of development tools, architectural and design constraints, purchased artifacts, libraries, and so on

#### 3.3.5.1 < Design Constrains 1>

Considering the applicable population, the software should support English, simplified Chinese and traditional Chinese languages. This software development language is JAVA.

#### 3.3.5.2 < Design Constrains 2>

Software process requirement constraints. Analyze software requirements according to requirements management plan.

#### 3.3.5.3 < Design Constrains 3>

Architecture and design constraints. The system architecture follows the C/S architecture, the design of each module as far as possible to achieve the separation, modular development, but also pay attention to the reusability and portability.

#### 3.3.5.4 < Design Constrains 4>

Class library, etc. Rongyun IMKi t instant messaging cloud service.

### 3.3.6 Interface

This section specifies the interfaces/interfaces that the application must support. It should be very specific, including protocols, ports, logical addresses, etc., so that the software can be developed and validated according to interface/interface requirements.

) user interface. The user interface of this system is the mobile terminal And r o I dApp. The interface was developed by ma t e r I a l

D e s I gn style, in order to achieve a beautiful and concise effect. (2) hardware interface. The system is C/S architecture, the need for hardware to be able to run the browser device. Considering the demand of mobile computing, devices equipped with And r o I d operating system are required for App operation. (3) software interface. The software used in this system is as follows: a. MySQLSe r v e r. B. And r o I d. (4) communication interface. The communication protocol used in this system is as follows: a. TCP/IP protocol.

#### 3.3.6.1 User Interface

The user interface of this system is the mobile terminal And r o I dApp. The interface was developed by Material Design style, in order to achieve a beautiful and concise effect.

#### 3.3.6.2 Hardware Interface

The system is C/S architecture, the need for hardware to be able to run the browser device. Considering the demand of mobile computing, devices equipped with And r o I d operating system are required for App operation.

#### 3.3.6.3 Software Interface

The main softwares used in this system are: MySQL Server and Android.

#### 3.3.6.4 Communication Interface

The communication protocol used in this system is as follows:

1. TCP/IP protocol.
2. UDP protocal
3. HTTP protocal

### 3.3.7 Legal, copyright and other notices

This system uses the individual preparation and the open source code constitution, uses the open source part to abide by the agreement which the open source code uses, the non-open source part copyright belongs to this development group. The final interpretation right of this system (including but not limited to software, use, etc.) belongs to this development team.

### 3.3.8 Applicable standard

Computer software development specification GB8566 -- 88;

Computer software unit test guide GB/T15532 -- 95;

Software maintenance guide GB/T14079 -- 93;

Computer software reliability and maintainability management GB/t14394-93.

## 3.4 System Model

In this section, we will be talking and defining the models used in the system.

### 3.4.1 Senario

<Senario 1>:

**Scenario Name**: Register

**Participating actor instance**: User Tom

**Mainstream**:

1. Tom clicks the "Register" button on the page.
2. The system receives Tom's request, jumps to the registration page and displays the registration form for filling in.
3. Tom fills in the registration form's user name, login password, confirmation login password, mobile phone number, e-mail, and clicks the "Confirm Registration" button after filling in.
4. Systematic identification of the validity of form information.
5. The system prompts for successful registration and jumps to the login interface.

<Senario 2>:

**Scenario Name**: Login

**Participating actor instance**: User Tom

**Mainstream**:

1. Tom clicks the login button on the page.

2. The system receives Tom's request, jumps to the login page, and displays the login form for filling in.

3. Tom fills in the user name and login password of the login form, and then clicks the "login" button.

4. Systematic identification of the validity of form information.

5. The system prompts for successful login and jumps to the game selection page.

<Senario 3>:

**Scenario Name**: CreateRoom

**Participating actor instance**: User Tom

**Mainstream**:

1. Tom clicks the "CreateRoom" button on the boundary interface.
2. The system receives Tom's request, jumps to the CreateRoom page and displays required choices for Tom，for example, how many people the room contain.
3. Tom completes the required choices and presses "Confirm CreateRoom" button after that.
4. Systematic identification of the validity .
5. The system prompts for successful CreateRoom and jumps to the room interface.

<Senario 4>:

**Scenario** **Name**: JoinRoom

**Participating actor instance:** User Tom

**Mainstream:**

1. Tom clicks the "JoinRoom" button on the page.

2. The system receives Tom's request, jumps to the room list page, searched the rooms that have been created and displays the list of rooms for users to choose.

3. Tom chooses a room, and clicks "join" button to join the room.

4. The system adds Tom to the room and jumps to the room page.

<Senario 5>:

**Scenario Name**: SelectRole

**Participating actor instance**: User Tom

**Mainstream**:

1. Tom clicks the box of himself on the page.
2. The system pops up a role select page that is less than the room page, which contains all the roles that Tom owns.
3. Tom clicks a role to select a role, clicks “Select” button to confirm the choice.
4. The system hides the role select page and changes the role information of Tom on the room page.

<Senario 6>:

**Scenario Name**: Prepare

**Participating actor instance**: User Tom

**Mainstream**:

1. Tom clicks “Prepare” button to claim that he is prepared.
2. The “Prepare” button turns to “Cancel” button. Tom can click it to cancel prepare state.
3. When the system is prepared, the game starts.

<Senario 7>:

**Scenario Name**: Move

**Participating actor instance**: User Tom

**Mainstream**:

1. Tom clicks the direction button in the game.
2. System receives Tom's request.
3. System identifies the direction for moving and judge whether this move is legal.
4. System gives the permission for moving.
5. System sends the updated data for moving.

<Senario 8>:

**Scenario Name:** Attack

**Participating actor instance:** User Tom, User Jim

**Mainstream**:

1. Tom touches the "attack" button on the screen.
2. The system receives Tom's request and shows the character movement according to his weapon.
3. The system sends the message to server.
4. The server judges whether other players exist in the attack range.If not, the action will finish.If other players exist(suppose the target player is User Jim’s character), then go to 5.
5. Server calculates the damage based on Tom’s weapon and attributes, and reduces life value of Jim’s character.
6. Tom completes attack and begins another movement.

<Senario 9>:

**Scenario Name**: UseProps

**Participating actor instance**: User Tom

**Mainstream**:

1.Tom opens the bag.

2. Tom chooses the prop he wants to use .

3.The system subtracts the prop by 1 and if the prop is used up, the icon will be hidden.4.The system changes the attribute value of Tom according to the type of prop.

5.Tom closes the bag.

<Senario 10>:

**Scenario Name**: ShowResults

**Main participants**: User Tom

**Mainstream**:

1. Ending condition has been triggered or the game time has ended.
2. System send the results to Tom.
3. Game shows the results in the interface.
4. Tom clicks the acknowledge button.
5. Display ended, get back to the enter interface.

<Senario 11>:

**Scenario** **Name**: EndOfGame

**Participating actor instance**: User Tom

**Mainstream**:

1. The game logic for end of game is triggered.
2. Ending information is sent to Tom.
3. Tom views the information and gets informed of the result.
4. Tom presses the ‘exit’ button and gets back to the enter interference.

### 3.4.2 Usecase Specification

1. All the participants：

|  |  |
| --- | --- |
| Participant | Interpretation for the participant |
| User | A user who have installed our game, he/she can login and join a game. |
| Player | A login user who is playing our game. |

1. All the usecases in our system

|  |  |  |
| --- | --- | --- |
| Usecase | Usecase Level | Interpretation |
| Register | User Goal | Register account |
| Login | Sub-function | Login to the system |
| CreateRoom | User Goal | Create a room |
| JoinRoom | User Goal | Join a room |
| SelectRole | Sub-function | Select a role |
| Prepare | Sub-function | Prepare for the game |
| GameProcess | User Goal | The progress of game |
| Move | Sub-function | Move a role |
| Attack | Sub-function | Operation attack |
| UseProp | Sub-function | Use a prop |
| EndOfGame | Sub-function | End of game |
| ShowResults | User Goal | Show the final result |

1. Detailed usecase specification.
2. Register

**Use Case Name**: Register

**Scope**: System Use Case

**Level**: User Goal

**Main participants**: User

**Stakeholders and users whose concerns:** User not yet registered: successfully registered account

**Prerequisites**: Users enter the registeration interface.

**Postcondition**: User registration is successful.

**Mainstream**:

1. users click the "Register" button on the page.

2.The system receives the user's request, jumps to the registration page and displays the registration form for filling in.

3. Users fill in the registration form's user name, login password, confirmation login password, mobile phone number, e-mail, and click the "Confirm Registration" button after filling in.

4. Systematic identification of the validity of form information.

5. The system prompts for successful registration and jumps to the login interface.

**Expansion process**

Registration form information is invalid

(1) In step 3, the login password filled in by the user is illegal. The system prompts the user to fill in the login password too short, too long or too simple, and asks the user to modify a legitimate password and submit it it again.

(2) In the third step, the user fills in the confirmation password and the login password are inconsistent.

(3) In step 3, the user name filled out by the user has been registered.

(a). The system prompts the user that the username has been registered, and requests the user to revise it, submit it it again and try again.

(b)The system prompts users to fill in inconsistent login passwords twice. Please check the changes and submit them again.

(4) In step 3, the number of the mobile phone that the user filled in has been registered. The system prompts the user that the mobile phone number has been registered. Please check the modification and submit it again.

(5) In step 3, the user's e-mail has been registered. The system prompts the user that the e-mail has been registered, and asks the user to check the modification and resubmit it.

**Special Needs**: None.

**Frequency:** It may happen at any time, but in general, a user will only register once, so the rate will not be too high.

1. Login

**Scope**: System Use Case

**Level**: Subfunctionl

**Main participants**: User

**Stakeholders and users whose concerns:** User who had registered: login for playing the game.

**Prerequisites**: Users enter the login interface.

**Postcondition**: the log of the user is successful.

**Mainstream**:

1. The user clicks the login button on the page.

2. The system receives the user's request, jumps to the login page, and displays the login form for filling in.

3. The user fills in the user name and login password of the login form, and then clicks the "login" button.

4. Systematic identification of the validity of form information.

5. The system prompts for successful login and jumps to the game selection page.

**Expansion process**

The login form information is invalid:

(1) In step 3, the username and password filled in by the user are illegal.

(2) In step 3, the username entered by the user does not exist. The system prompts the user that the username does not exist. Please ask the user check the changes and resubmit them.

(3) In step 3, the login password of the user is incorrect. User name or login prompted by the system.The password is incorrect. Please check the changes and resubmit them

**Special Needs**: None.

**Frequency:** It may happen at any time and with high frequency.

1. CreateRoom

**Use Case Name**: CreateRoom

**Scope**: System Use Case

**Level**: User Goal

**Main participants**: User

**Stakeholders and users whose concerns:** User who has logged in and is preparing to set up a game: successfully set up a game ,in another word ,successfully create a gameroom.

**Prerequisites**: Users enter the game boundary interface.

**Postcondition**: The game room has been set up successfully.

**Mainstream**:

1. users click the "CreateRoom" button on the boundary interface.

2.The system receives the user's request, jumps to the CreateRoom page and displays required choices for users，such how many people the room contain.

3. Users complete the required choices and press "Confirm CreateRoom" button after that.

4. Systematic identification of the validity .

5. The system prompts for successful CreateRoom and jumps to the room interface.

**Expansion process**

CreateRoom choices is illegal

(1) In step 3, the fundamental information set by the user is illegal. The system prompts the user to the number of the users the room can contain is beyond the limit and asks the user to modify a legitimate password and submit it it again.

**Special Needs**: None.

**Frequency:** It may happen at any time. It is unavoidable to create a Room when a game begins.

1. JoinRoom

**Use Case Name**: JoinRoon

**Scope**: System Use Case

**Level**: User Goal

**Main participants**: User

**Stakeholders and users whose concerns:** User who had logged in: join an existed room to play a game.

**Prerequisites**: Users login their account.

**Postcondition**: User join room successfully.

**Mainstream**:

1. Users click the "JoinRoom" button on the page.

2. The system receives the user's request, jumps to the room list page, searched the rooms that have been created and displays the list of rooms for users to choose.

3. Users choose a room, and click "join" button to join the room.

4. The system adds users to the room and jumps to the room page.

**Expansion process**

No room existed:

1. In step 2, no room has been created. The system prompts the user to create a room.

Failed to join room:

1. In step 4, an error occurs such as the internet instability when the user join the room. The System prompts the user the failure of join room, prompts the user to try again or choose another room and reload the reloads the list of rooms.

**Special Needs**: None.

**Frequency:** It may happen at any time, but in general, the expectation of times that a user join a room per game is less than 1, so the rate will not be too high.

1. SelectRole

**Use Case Name**: SelectRole

**Scope**: System Use Case

**Level**: Subfunction

**Main participants**: User

**Stakeholders and users whose concerns:** User who had joined a room: select a role that the user will use in the next game.

**Prerequisites**: Users join an existed room successfully.

**Postcondition**: The user selects a role successfully.

**Mainstream**:

1. The user clicks the box of himself or herself on the page.
2. The system pops up a role list that is less than the room page, which contains all the roles that the user owns.
3. The user clicks a role to select a role, click “Select” button to confirm the choice.
4. The system hides the role select page and change the role information of the user on the room page.

**Expansion process**

The user doesn’t select a role:

1. In step 1, the user doesn’t select a role and then click the “Prepare” button. After the game begins, the system will assign a random role to the user that the user owns.

**Special Needs**: None.

**Frequency:** It may happen at any time after it that the user joins a room and before the user prepares..

1. Prepare

**Use Case Name**: Prepare

**Scope**: System Use Case

**Level**: Subfunction

**Main participants**: User

**Stakeholders and users whose concerns:** User who has joined a room: claim that the user has prepared to start a game.

**Prerequisites**: Users join a room.

**Postcondition**: The user prepares successfully.

**Mainstream**:

1. The user clicks “Prepare” button to claim that he or she is prepared.
2. The “Prepare” button turns to “Cancel” button. Users can click it to cancel prepare state.
3. When all the users are prepared, the game start.

**Expansion process**: None.

**Special Needs**: None.

**Frequency:** It may happen at any time after it that the user joins a room and before the game start.

1. GameProcess

**Use Case Name**: GameProcess

**Scope**: System Use Case

**Level**: User goal

**Main participants**: Player

**Stakeholders and users whose concerns:** Players who are in the same room

**Prerequisites**: All players are ready.

**Postcondition**: All players return to the room or account page.

**Mainstream**:

1. The server create the map generate the props and initial location of each role in the map.
2. The system create the role object for each player.
3. Players explore the map (included sub-functions: Move, Attack and UseProps) and update their data to the server continuously.
4. The server send life value, location of each player back timely.
5. When one player fulfill the end condition, the system call the EndOfGame and ShowResult for him.
6. After all the players have ended, the server destroy the map object, and players choose returning to room or account page.

**Expansion process**: None

**Special Needs**: None.

**Frequency:** It happens only once, but always lasts for 10-15 minutes.

1. Move

**Use Case Name**: Move

**Scope**: System Use Case

**Level**: Sub function

**Main participants**: User

**Stakeholders and users whose concerns:** User not yet registered: successfully registered account

**Prerequisites**: Player presses the moving button.

**Post-condition**: The role finished moving.

**Mainstream**:

1. Players click the direction button in the game.

2. System receives the user's request.

3. System identifies the direction for moving and judge whether this move is legal.

4. System gives the permission for moving.

5. System sends the updated data for moving.

**Expansion process**

The move is identified to be illegal (moving against obstacles or moving outside the map).

1. In step 1, player pick the illegal direction for movement;
2. In step 3, the system finds out that multiple players are seeking request to move into the same block.

**Special Needs**: None.

**Frequency:** It may happen at any time in a game, the precise frequency depends on the frequency of game server and network status.

1. Attack

**Use Case Name**: Attack

**Scope**: System Use Case

**Level**: Subfunction

**Main participants**: Player

**Stakeholders and users whose concerns:** User who are in the game process

**Prerequisites**: Players have entered the game.

**Postcondition**: Player finishes the action or is defeated.

**Mainstream**:

1. Players touch the "attack" button on the screen.

2.The system receives the player's request and shows the charactor movement according to his weapon.

3.The system sends the message to server.

4. The server judges whether other players exist in the attack range.If no one hurts,the action will finish.

5. Server calculate the damage based on weapon and role attributes, and reduce life value of the target being attacked.

6. The role completed attack and begin another movement.

**Expansion process**

1. In step 5, the target’s life value is reduced to zero.

(2) That target player will be defeated and call the EndOfGame and ShowResult. The server will send the message to other players left.

(3) The player making attacks will get one score.

**Special Needs**: None.

**Frequency:** It may happen at any time during the game process but when one attack action has not finished yet, there won’t be another.

1. UseProp

**Use Case Name**: UseProps

**Scope**: System Use Case

**Level**: Subfunction

**Main participants**: Player

**Stakeholders and users whose concerns:** User who are in the game process

**Prerequisites**: Player have entered the game.

**Postcondition**: Player finishes this action or is defeated.

**Mainstream**:

1.Player opens the bag.

2. Player chooses the prop he wants to use .

3.The system subtract the prop by 1 and if the prop is used up, the icon will be hidden.

4.The system changes the attribute value of role according to the type of prop.

5.Player closes the bag.

**Expansion process**

Wear equipments

1. In step 2, player chooses an equipment.
2. The system judges whether the role is wearing an equipmeng in the same place. If yes, the previous equipment will be off and back in the bag.
3. The system minus the attribute value by the value of the provious equipment.

Change life value

1. In step 4, the prop changes the life value of role.
2. The system will update the new life value to the server.
3. The server changes the life value recorded.

**Special Needs**: None.

**Frequency:** It may happen at any time during the game process as long as the player has props. But sometimes when player has just used one, he don’t need to close and open the bag again to use another.

1. EndOfGame

**Use Case Name**: EndOfGame

**Scope**: System Use Case

**Level**: Sub-function

**Main participants**: User

**Stakeholders and users whose concerns:** Users who are still alive when the game ended.

**Prerequisites**: the game is ended.

**Post-condition**: users press the ‘exit’ button.

**Mainstream**:

1. The game logic for end of game is triggered.
2. Ending information is sent to every user.
3. Users view the information and get informed of the result.
4. All users press the ‘exit’ button and get back to the enter interference.

**Expansion process**

None

**Special Needs**: None.

**Frequency:** It happens when the game ends, so the frequency is one time for a game.

1. ShowResults

**Use Case Name**: ShowResults

**Scope**: System Use Case

**Level**: User Goal

**Main participants**: User

**Stakeholders and users whose concerns:** Users in the same game that has ended.

**Prerequisites**: Users trigger the ending condition or the game time is over.

**Post-condition**: All players have click the acknowledge button.

**Mainstream**:

1. Players have triggered the ending condition or the game time has ended.
2. System send the results to players.
3. Game show the results in the interface.
4. Players click the acknowledge button.
5. Display ended, get back to the enter interface.

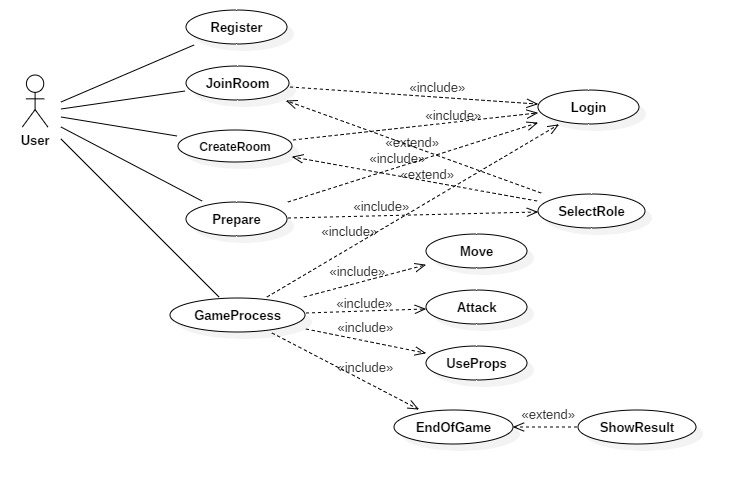
**Expansion process**

None.

**Special Needs**: None.

**Frequency:** It happens when a member triggers the ending condition or the game time ends.

1. The complete use case model:



### 3.4.3 Objective Model

Entity Class

|  |  |  |  |
| --- | --- | --- | --- |
| Entity class name | Attributes | Association Class | Definition |
| Account | userID:int  username:string  password:string  phonenumber:string  e-mail:string  rankrecord:string | Player | User information class, save username, password, Mobile number, mailbox and user ID |
| Room | roomID:int  roomname:string  roomcapacity:int  preparestate:vector<bool> | Player  Role | A container for many players |
| Role | roleName:string  roleid:int  roleapperance:pictureposition  roledress:pictureposition  speed:int  attack:int  weapon:Prop  shoe:Prop  bag:vector<Prop>  lifevalue:int  vision:int | Role\_Simple  Room | The game role that it is operated by the user in the local. |
| Role\_Simple | id:int  lifevalue:int  position:vector<int> | Role | The role information interact in the server |
| Map | proplist:vector<Prop>  rank:vector<int>  livingrole:vector<Role\_simple>  totalpicture:pictureposition  wall:pictureposition | Role\_Simple  Role  Account | The area where the role can move on. |
| Player | id:int  name:string  roletype:int | account  Room | Someone who operates the room |
| Prop | id:int  type:int  value:int  PropWork:function  proppic: pictureposition | Map | The equipment or Drugs which role uses in the game |

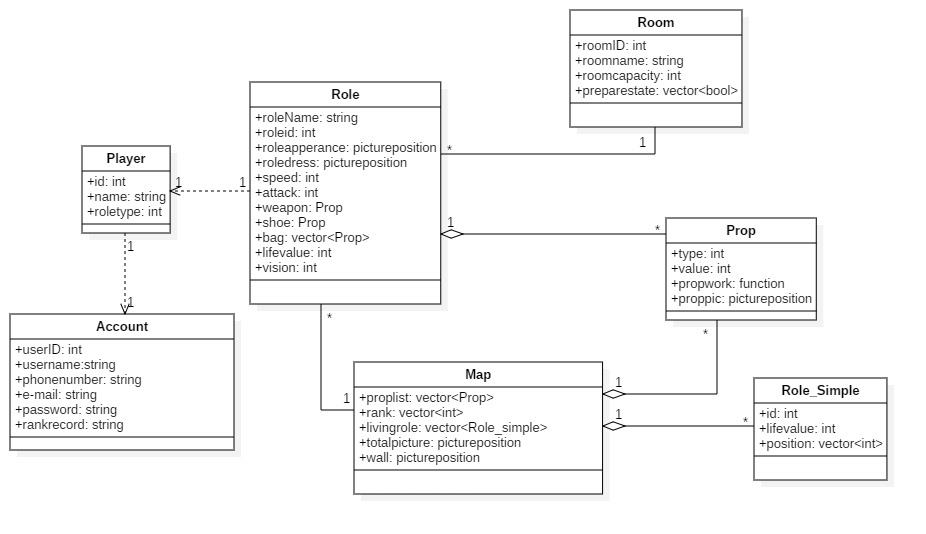
Boundary Class

|  |  |
| --- | --- |
| Boundary class name | Definition |
| RegisterPage | Page for user to submit register information. |
| LogInPage | Page for users to login. |
| AccountPage | The page to show user’s information |
| CreateRoomPage | Page for users to create room. |
| ChooseRoomPage | Page for users to search and choose room. |
| RoomList | The page for showing the room list. |
| RoomPage | The page for users waiting for beginning. |
| RoleList | Page for users to select role. |
| GameProcessPage | The page for users to play games. |
| ResultPage | Page for the display of final result. |

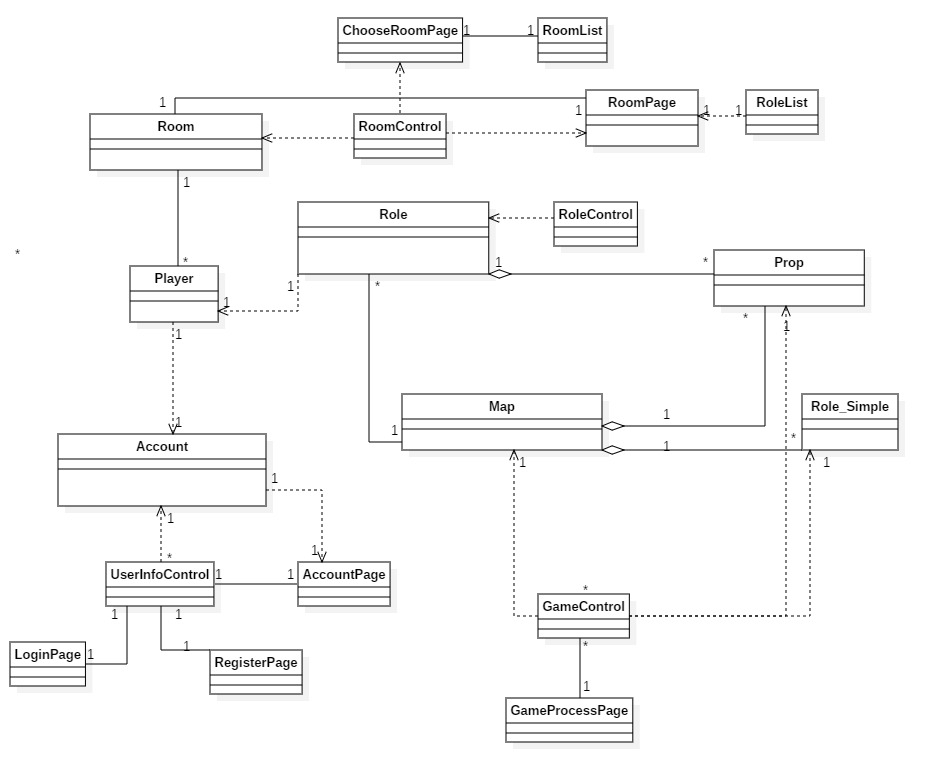
Control Class

|  |  |
| --- | --- |
| Control Class Name | Definition |
| GameControl | Control the game logic |
| RoomControl | Control and manage all the existing rooms |
| UserInfoControl | Store and manage the users’ information |
| RoleControl | Store the property of roles, which attributes the preperty to the users’ roles. |

Entity Class Diagram



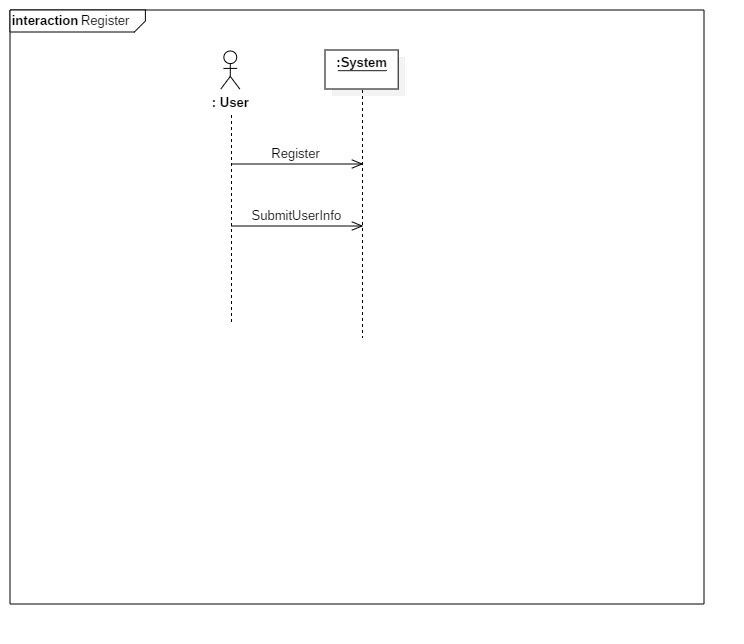
All class diagram



### 3.4.4 Active Model

1. System Sequence Diagram and Operation Contract

**Use case: register**



**Contract co1: SubmitUserInfo**

Operation :submitUserinfo (username:Username , password:Password, email:Email. userID:UserID, phonenumber:Phonenumber)

Cross reference:Register

Precondition:The player enters any page

Postcondition:

Create the instance U of User;

U. userID assignment;

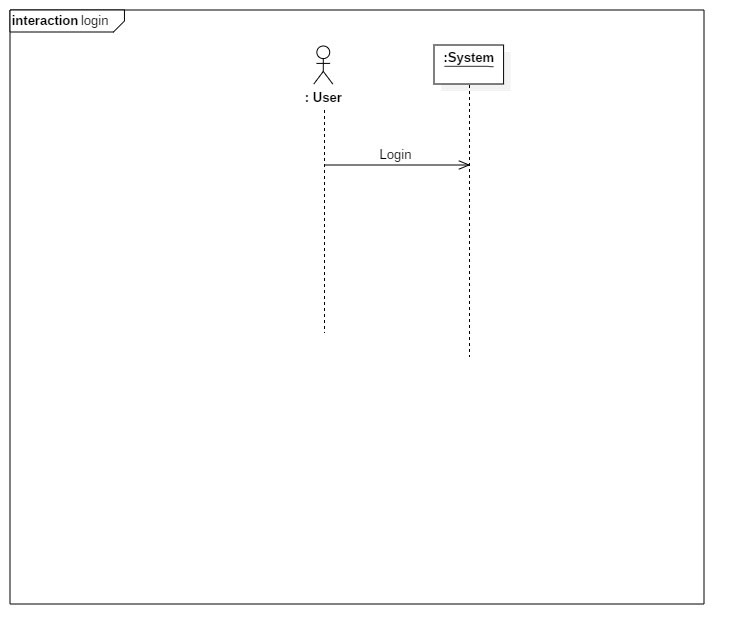
U. userName assigns username;

U. password is assigned to password;

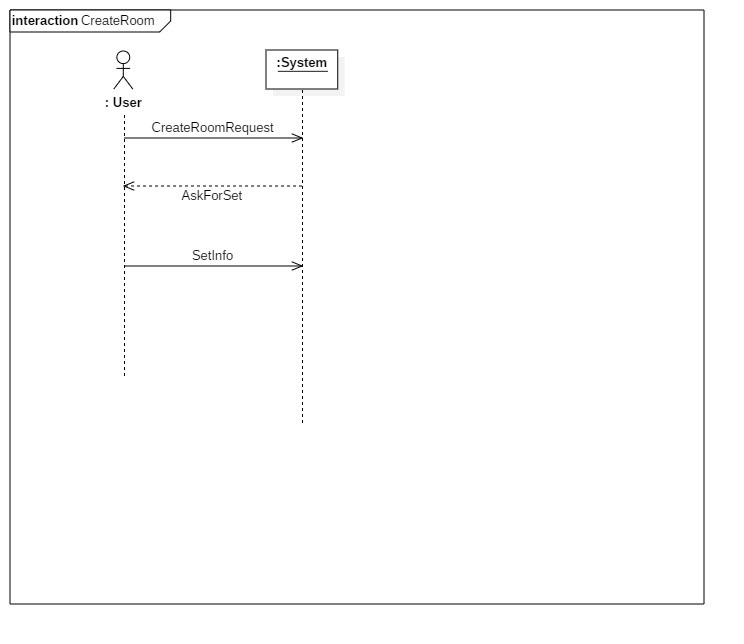
U. phonenumber is assigned phonenumber;

U.e-mail is assigned e-mail.

**Use case: Login**



**Use case: CreateRoom**



**Contract co2:SetRoomInfo**

Operation:setRoominfo(roomName:RoomName ,roomID:RoomID,roomcapacity:RoomCapacity )

Cross-reference: SetRoom

Pre-condition:The player prepares to select the Room.

Postconditions:

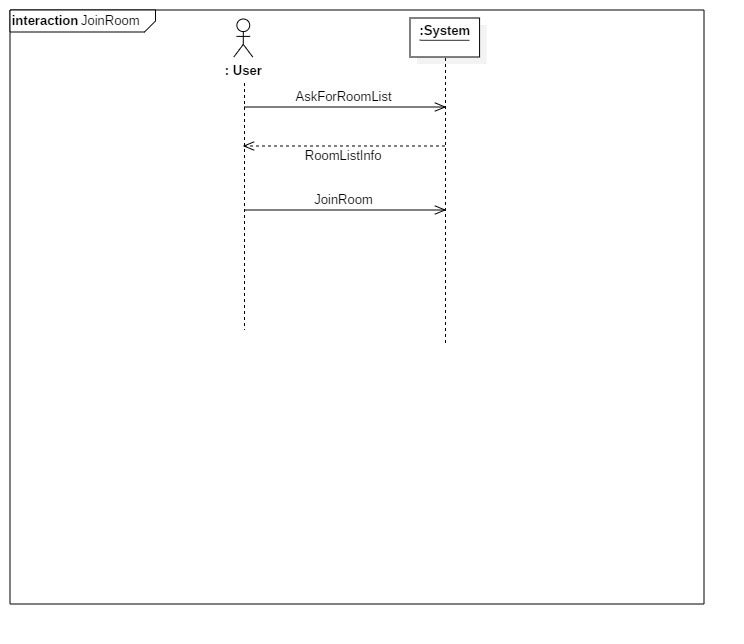
The player establishes the room.

R.roomName assignment.

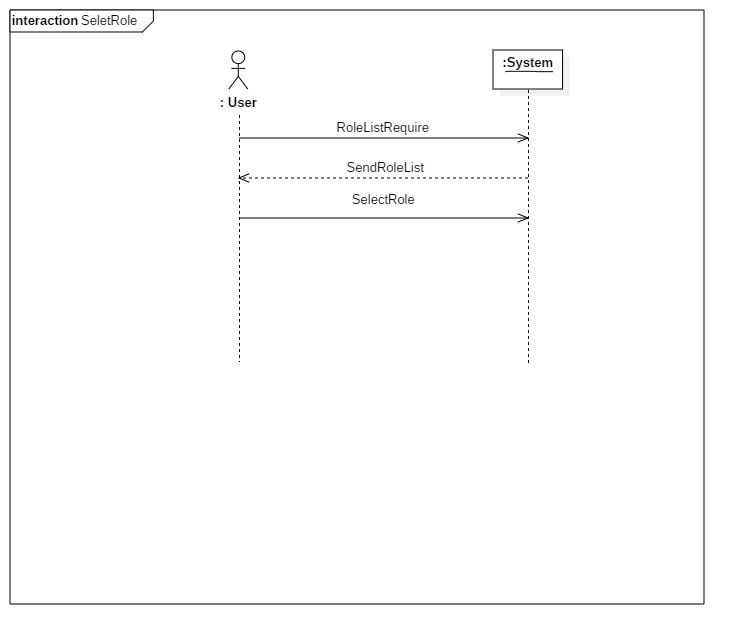
R.roomID is assigned to RoomID.

R.roomCapacity is assigned to RoomCapacity.

**Use case: JoinRoom**



**Use case: SelectRole**



**Contract co3: RoleInfo**

Operation: selectRole (roleName:roleName , roleAppearance:RoleAppearance, roledress:RoleDress )

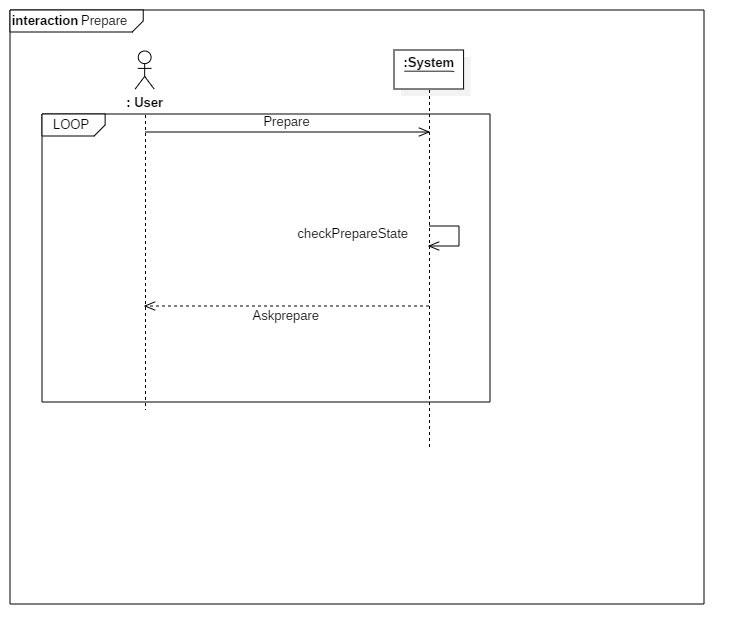
Cross-reference: SelectRole

Pre-condition: The player prepares to select the role.

Postconditions:

1. The player select the role successfully.
2. Ro.roleName assignment.
3. Ro.apperance is assigned to RoleAppearance.
4. Ro.roledress is assigned to RoleDress

**Use case: Prepare**



**Use case: GameProcess**



**Contract CO4: GenerateMap**

function: GenerateMap(r:Room)

cross-reference: Prepare, GameProcess.

precondition: All players are ready.

postcondition:

1. Create the instance of Map, m.
2. Create instances of Prop and generate their locations randomly.
3. Create an instance of Vector<Prop>, assign it to m.proplist.
4. For each ID in r.IDlist, create an instance of Role\_simple rs,assign ID to rs.id.
5. For each rs, generate a location vector, assign it to rs.location.
6. For each rs, according to the element in r.roletype at the same place, generate a life value, assgn it to rs.HP.
7. Create an instance of Vector<Role\_simple>, add all instances of Role\_simple into it, assign it to m.livingrole. Create an instance of Vector<int>,assign it to m.rank.

**Contract CO5: Initialize**

function: Initialize(m: Map, roleID: int, roletype: int)

cross-reference: GameProcess.

precondition: The map is generated in server.

postcondition:

1. Make a copy of the instance m, as m\_local.
2. Create an instance of Role as r.
3. assign roleID to r.roleID.
4. According to roletype, assign different values to r.HP, r.speed, r.vision.
5. Disttribute spaces for r.bag (an instance of Vector<Prop>).

**Contract CO6: RenewInfo**

function: RenewInfo(livingrole: Vector<Prop>,proplist :Vector<Prop>)

cross-reference: GameProcess, EndOfGame.

precondition: In the game process.

postcondition:

1. Update the the intance of m\_local, assign livingrole to m\_local.livingrole.
2. Update the the intance of m\_local, assign proplist to m\_local.proplist.

**Use case: Attack**



**Contract CO7: CauseDamage**

function: CauseDamage(damage: int, targetID: int)

cross-reference: GameProcess, Attack.

precondition: One user hit another.

postcondition:

1. Compare targetID with elements in m.livingrole, find the Role\_simple rs, which has the same re.id.
2. Reduce rs.HP by damage.
3. If rs.HP is no more than 0, remove it from m.livingrole.

**Use case: UseProp**



**Contract CO8: ChooseProp**

function: ChooseProp(p: Prop, r:Role)

cross-reference: GameProcess, UseProps.

precondition: User opens the bag.

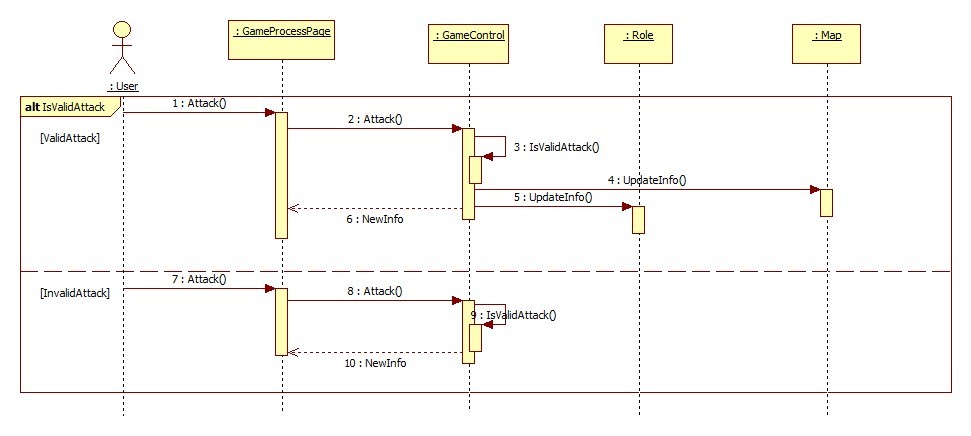
postcondition:

1. Remove the prop from Role instance r.bag.
2. If p.type is shoe, add p.value to r.speed ,add r.shoe in r.bag and set r.shoe with p.
3. If p.type is weapon, add p.value to r.attack, add r.weapon in r.bag and set r.weapon with p.
4. If p.type is medicine, call the function p.PropWork(r: Role).

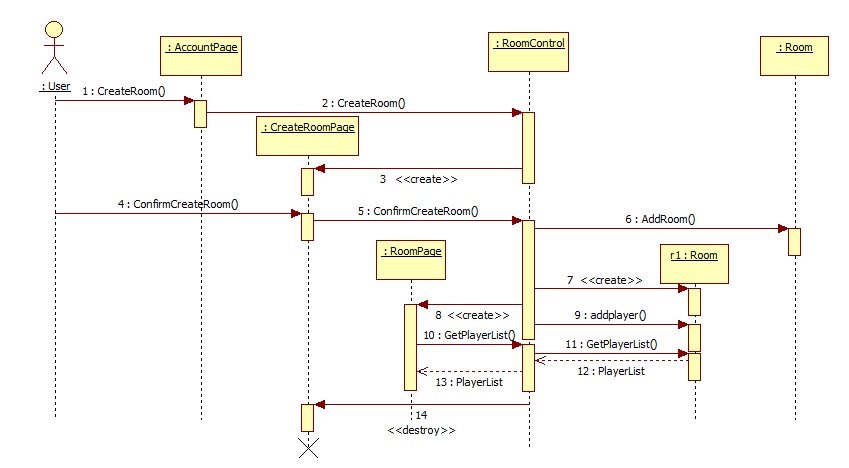
**Use case: EndOfGame Use case: ShowResult**



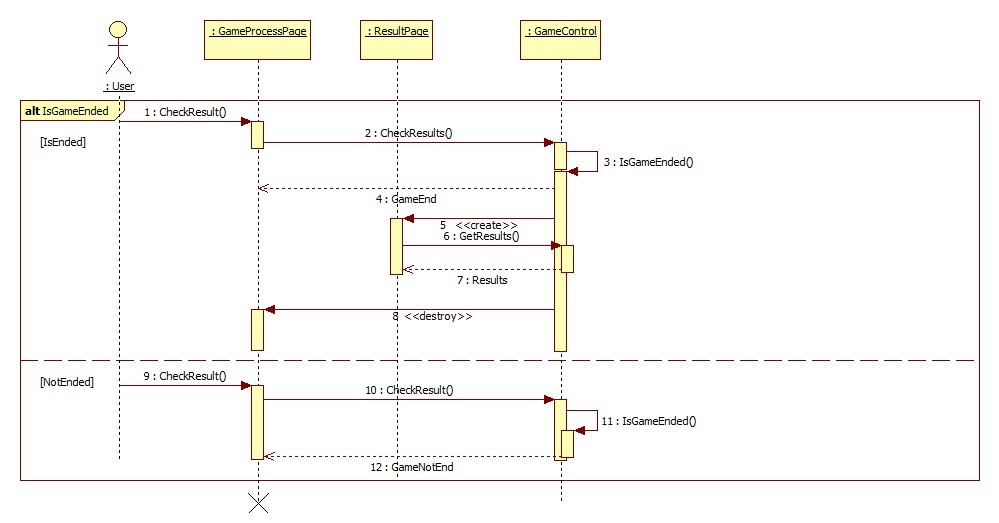
1. System Message Sequence Diagram
   1. Attack



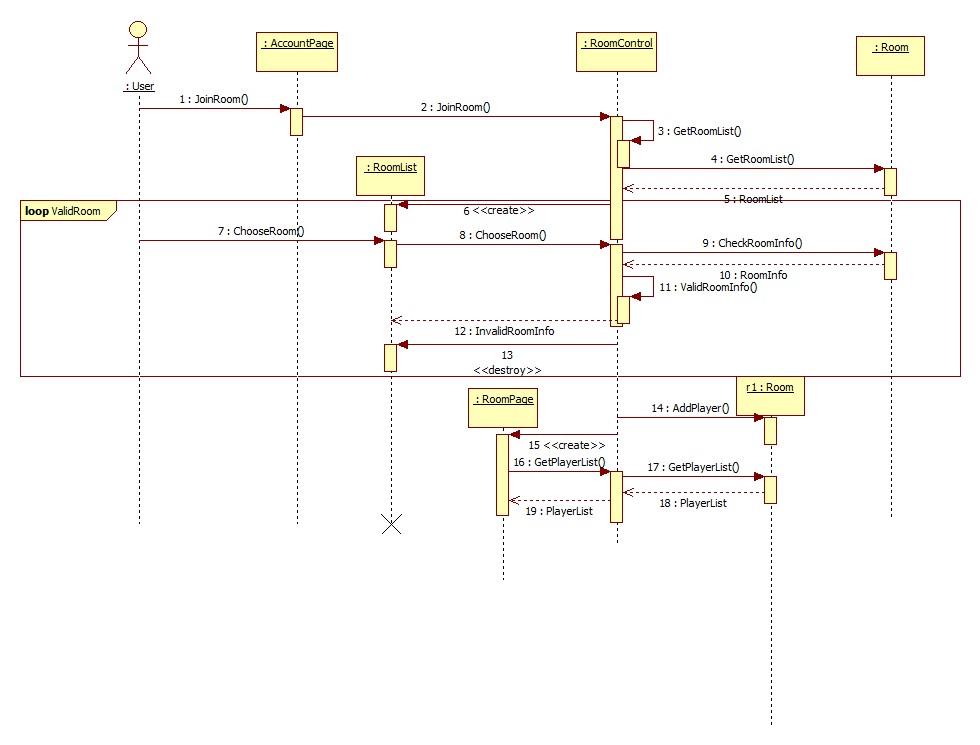
* 1. CreateRoom



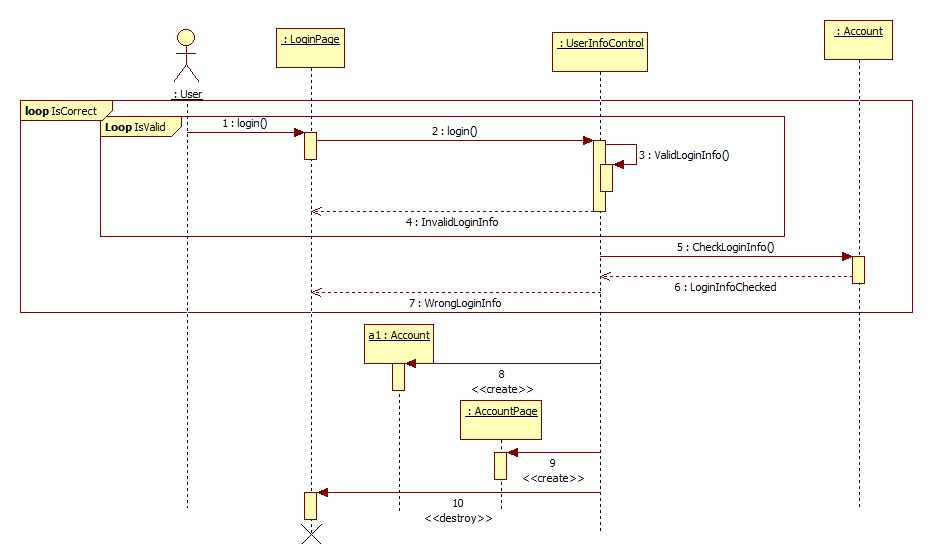
* 1. End&Result



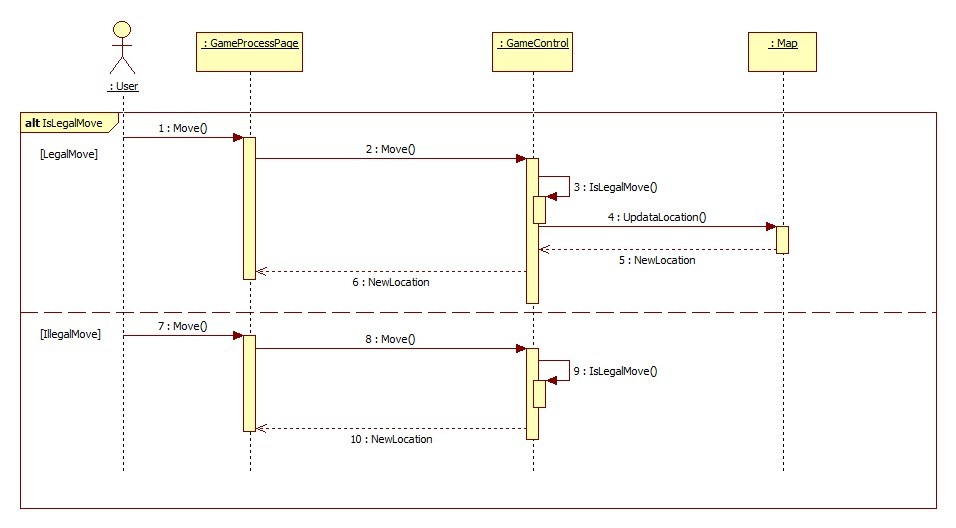
* 1. JoinRoom



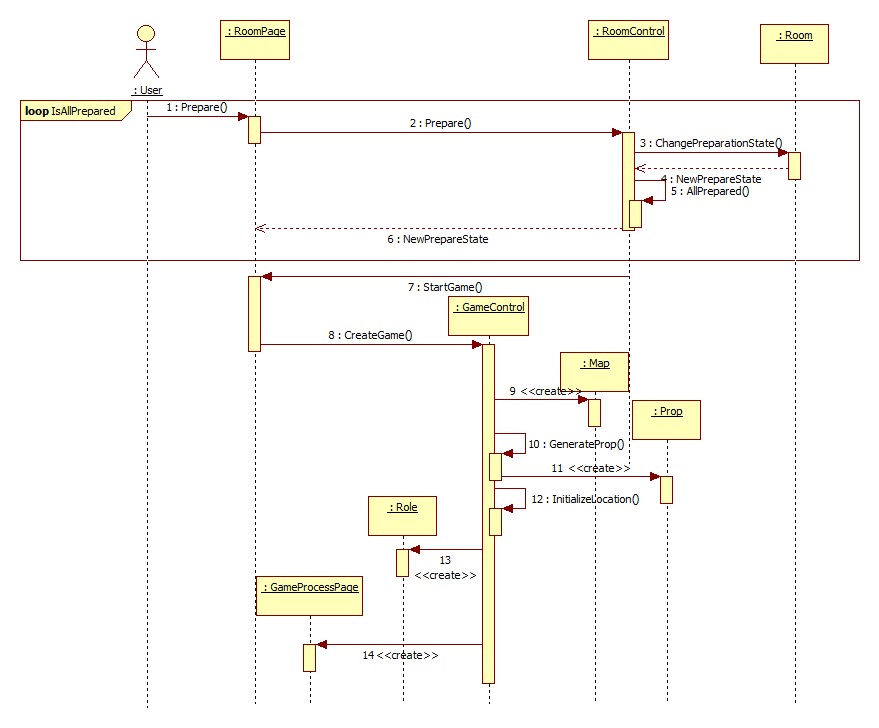
* 1. Login



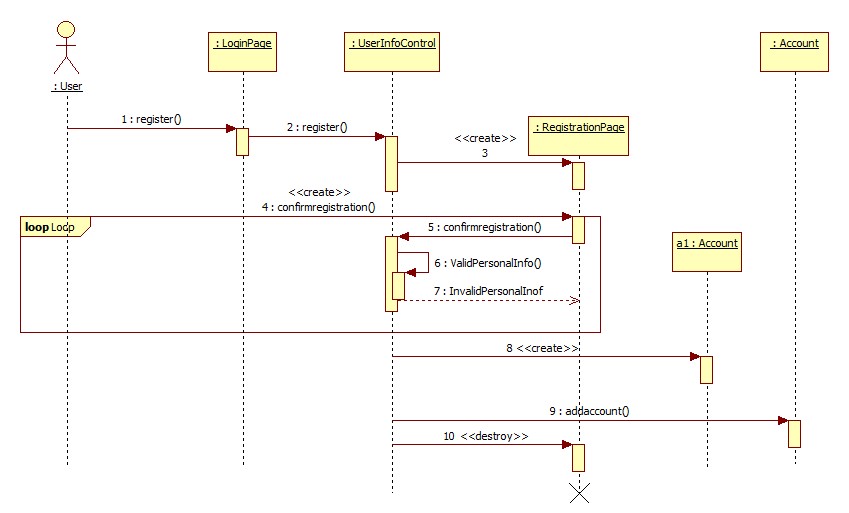
* 1. Move



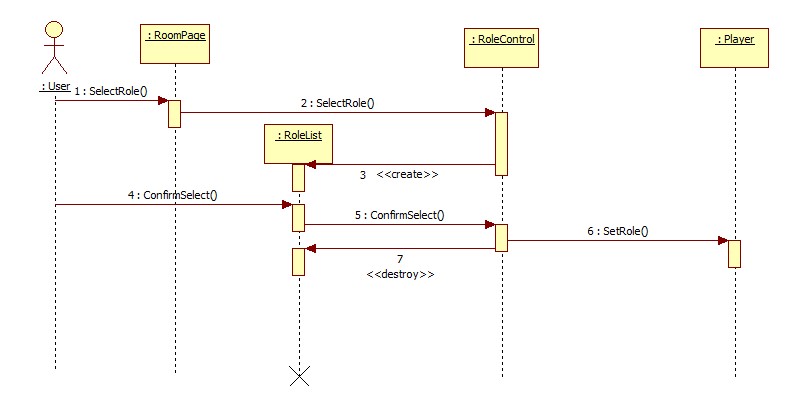
* 1. Prepare&StartGame



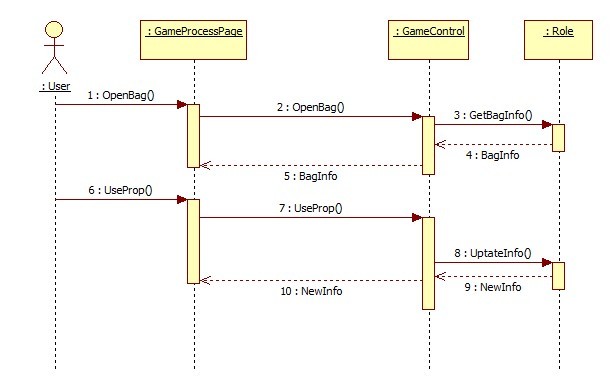
* 1. Register



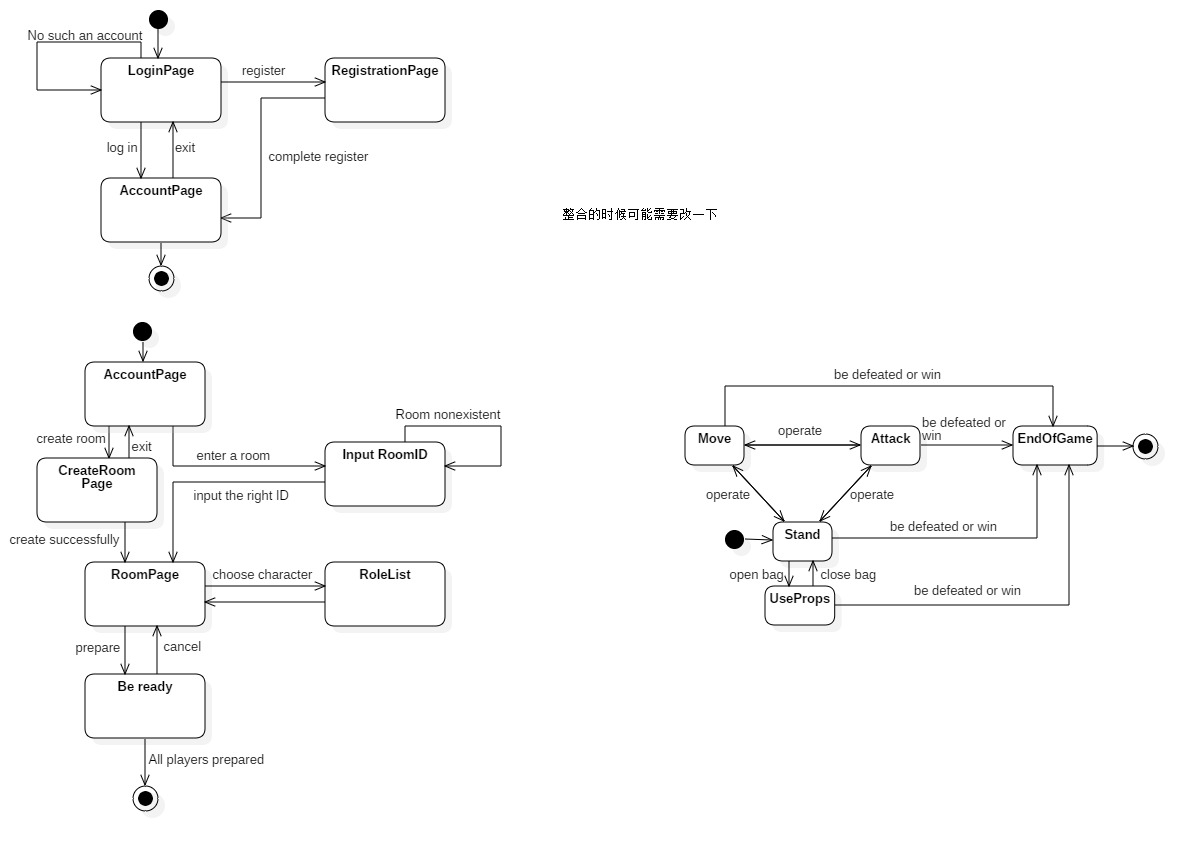
* 1. SelectRole



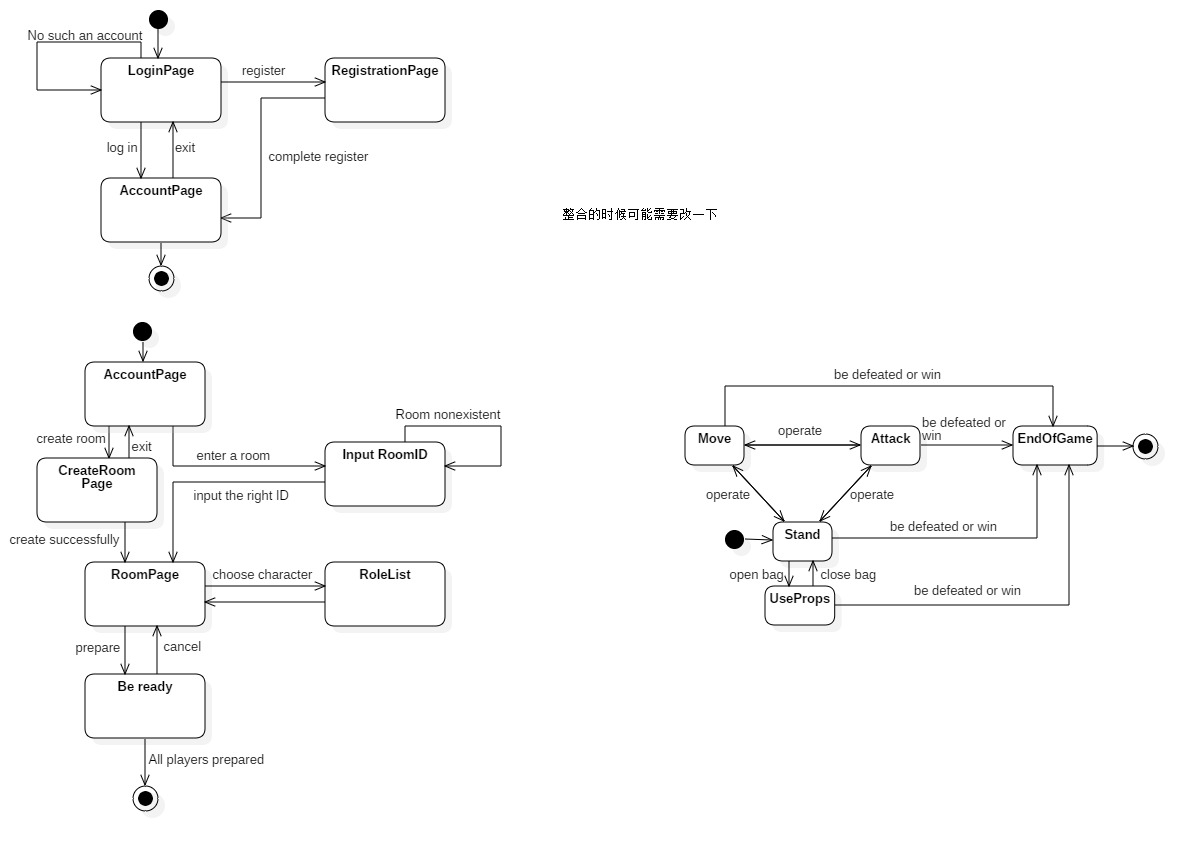
* 1. Use



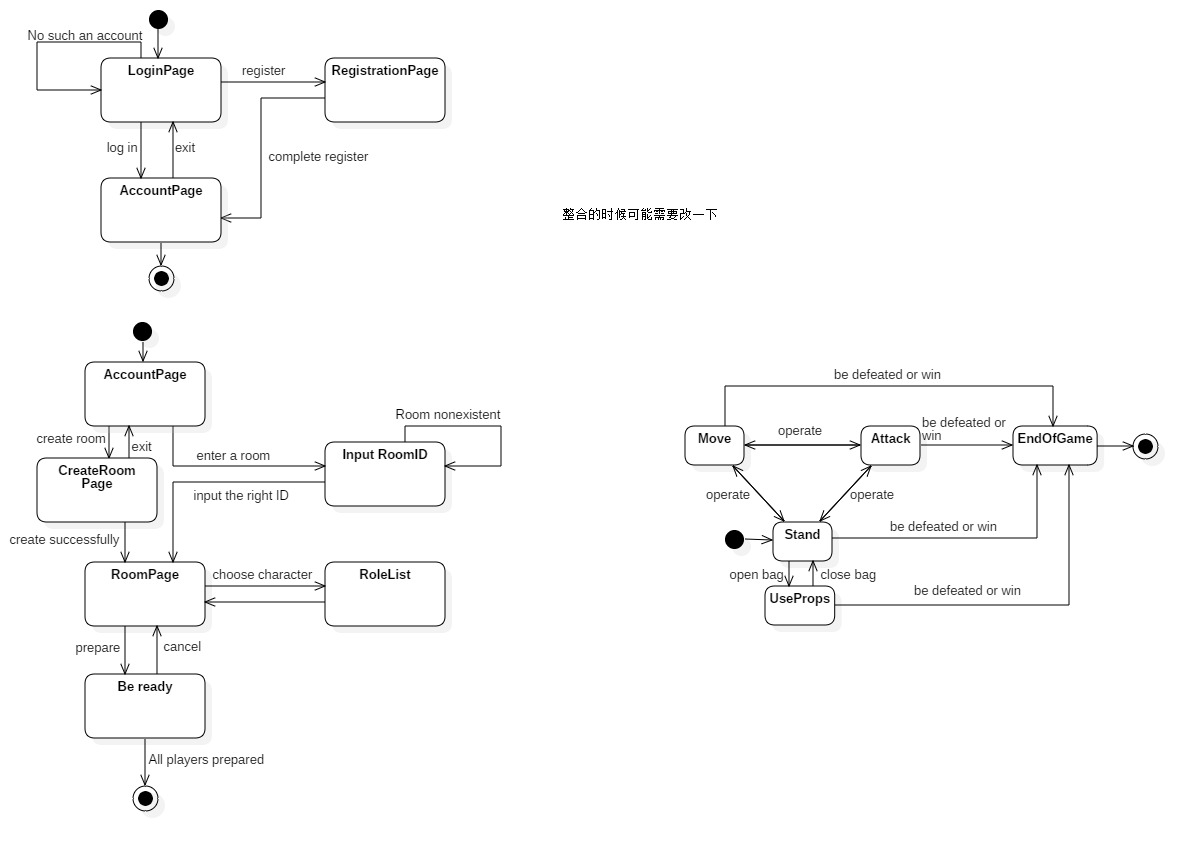
1. State Diagram
   1. Login Class



* 1. Room Class



* 1. GameProgress Class



### 3.4.5 User Interface

a) RegisterPage

In the central part of the register interface there are four bars from top to bottom: user account, user name, user password, user e-mail.A user needs to fill in to accomplish registering.

b) LoginPage

As above there are two bars:user account, user password.A user needs fill a password matching a account to log in.

c) AccountPage

There is a column in the left in which a user’s name and level bars, records and settings buttons are listed.In the bottom center there is a ‘enter’ button.

On clicking into the records interface, which has a form to show every time and result record of a game.And the setting interface has volume bar and Frame rate button listed vertically.

d) CreateRoomPage

There is an input text box and some option box, and users can set the name and property of a room. Then a room will be created.

d) ChooseRoomPage

There is bar and a button centrally shown to fill a room number in to search for the room.If the room exists, enter the room

e) RoomPage

Horizontally the users who play together are shown, and in the center bottom there is a ‘prepare’ button.

f) GameProcessPage

On the bottom left a rocker is shown, bag and attack button are right.On the top right there is a map.

e) ResultPage

Every player’s result are shown vertically, and ‘again’, ‘quit’ buttons are listed in the bottom center.