The Chinese University of Hong Kong, Shenzhen

CSC4001: Software Engineering

School of Data Science

Final Report Document (Version 6): *Foredawn*

Group 4

Author:	*Student Number:*
Hang Yu	119010406
Bodong Yan	119010369
Ziang Liu	119010211
Zexu Han	119010091

May 13, 2022

Contents

1 Introduction

1.1 Overview

Our project is to design and implement a web game platform, where users can not only experience the main plot of the game story but also have fun with the build-in mini-games. This software will be a combination of traditional web mini-game, adventure game, and online user platform. We are aimed to design an exciting main plot for users to explore and a lightweight platform for several mini-games.

1.2 Objective

The design idea of this software refers to a game called Pummel Party. It is a multiplayer party game composed of monopoly games, a number of mini-games, communication system and achievement list. It provides a great platform for players to play with friends. Then we combine this kind of design with our intention to create an adventure game to build a themed and interesting game platform.

1.3 Highlight

The project is mainly composed of four parts, back-end, ADV game as main body, a mini-game implemented by original JavaScript and another mini-game by game engine: pixi. The users can maintain their save-data in their account, so that they can continue the game from any of their save points next time they log in. Also, the choices the users make in the game can lead to different game experiences and endings. In addition, the two mini-games are used to present some plots in the main story. Specifically speaking, there's a scene that the users' avatar is escaping from mutant zombie by a helicopter, so the first mini-game will let the users control the helicopter by themselves to get to a safe place. And the second mini-game is in order to present the material collection task of the main game.

2 Architecture Design

This part describes the system architecture of our system. Architecture diagram and the DFD of the whole system will be presented below with the corresponding text explanations. All these descriptions and diagrams are used to explain the system more accurately and comprehensively from different aspects.

2.1 System Architecture

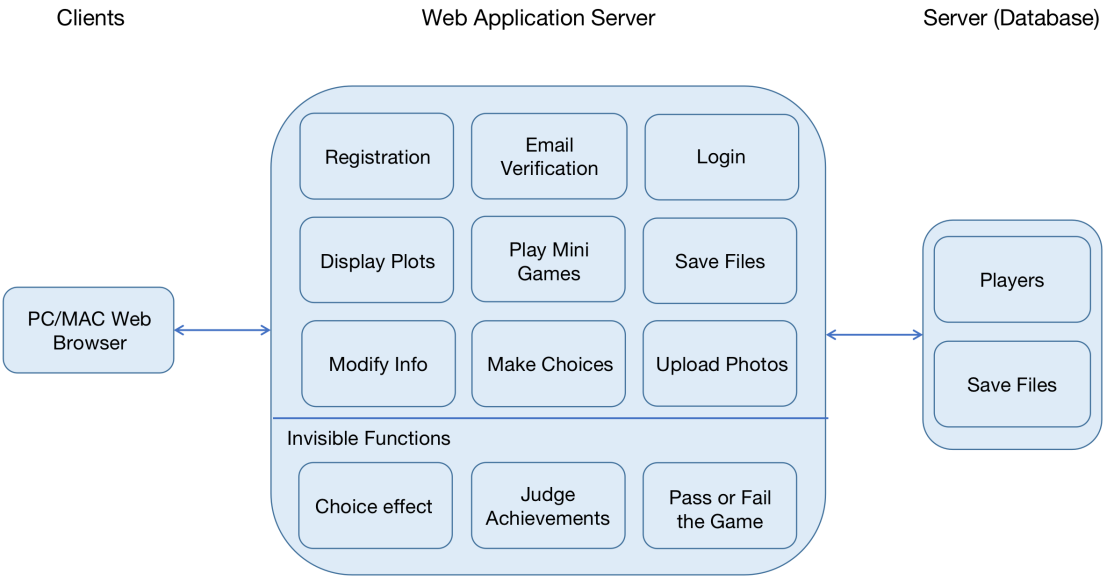

Figure 1: Architecture Diagram

The above figure describes the overall architecture design of our system. In the frontend part, users can access our system though a web browser. Through the user interface, the player will be able to use functions like registration, login, email verification, uploading photos, reading stories, playing mini games, save the process of game, and modifying personal information. The administrators have the ability to access all the users and modify their passwords. Also, some functions are hidden from users, including choices effect, judging conditions for achievements, and determining whether the player passes or fails the mini games. For the server side, there are two databases to store the information of players and save files of the game in our system.

2.2 Data Flow Diagram

Our system can be divided into two main parts: Player module and Game module. This part of the report shows the data flow diagrams (DFD) and the description on the design specification of data flow and behavior of these two parts.

2.2.1 Player Module

Before entering the game, players need to create an account if they don't have one. To do so, they are required to enter their email address and its format will be checked. An email with an activation link will be sent to that address. After players click the link, they

should be able to set a password and re-enter it. Both the email address and the password will be stored in a database named Players. A success message will be shown.

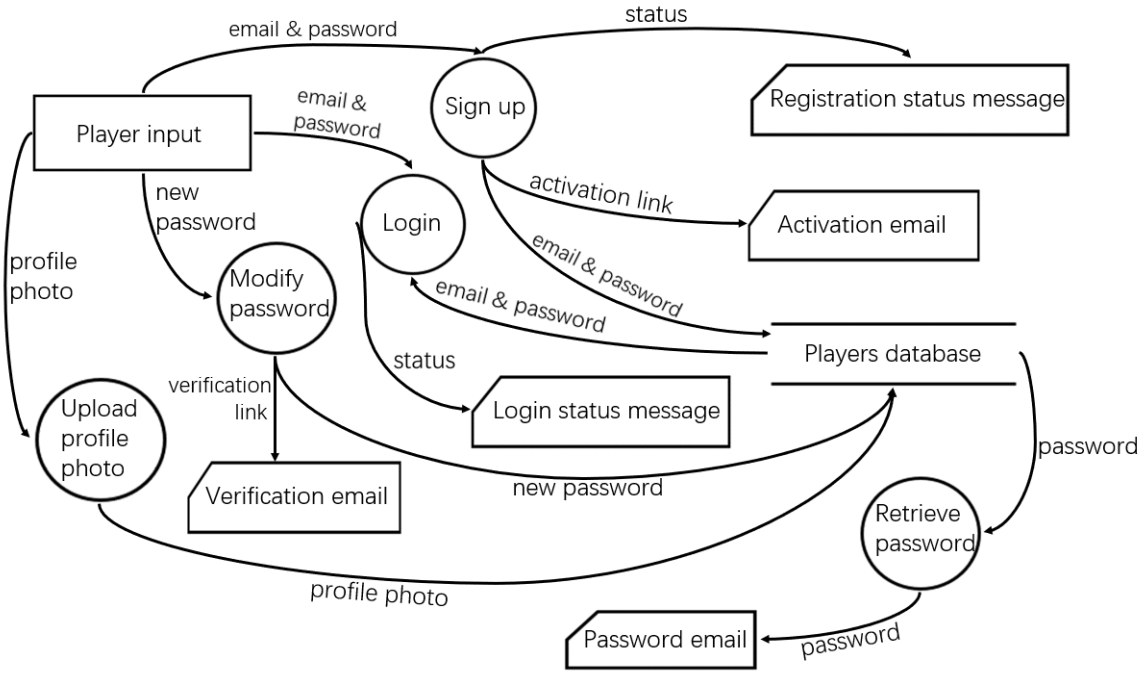

Figure 2: Data Flow Diagram: Player Module

Then players may login the game by entering their email address and password correctly. If players forget the password, they can retrieve the password by receiving an email where their password is in. After login, players can customize their home page, for example, upload a profile photo. All these information will be stored in database. Players can also modify the password by clicking a verification link sent to their email address and entering a new password. Correspondingly, data in the Player database will be updated.

2.2.2 Game Module

Basically, our game is a visual novel with two mini games. Materials of the whole story including texts, images, mini games, and a branching choice system are all stored in js files. The branching choice system, which is a graph structure, is vital to our game because the plot development relies entirely on it. We refer to the game progress of a player as a "location", a "process", or a "record". They all have the same meaning, indicating the player's location in the branching choice system (or "plot tree").

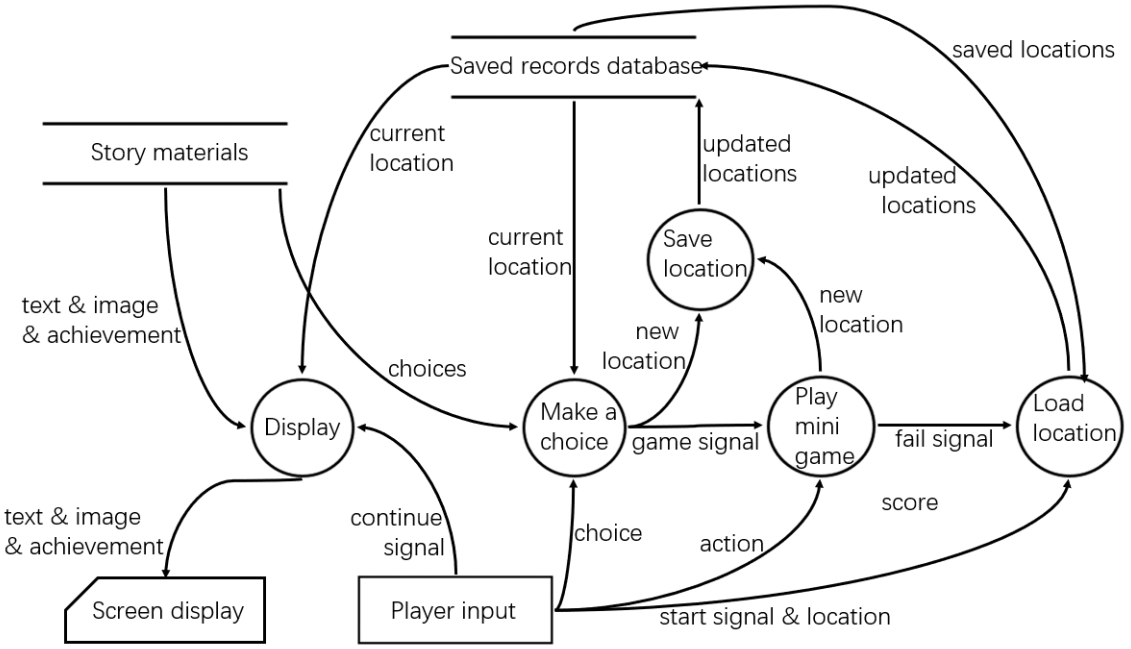

Figure 3: Data Flow Diagram: Game Module

A player's current location as well as the locations where he has ever been are stored in a database called Save files, which performs semantically. Every time the player changes his location by making a choice or playing a mini game, his current location will be updated automatically. The player can load a saved location in Save files at any time, that is, update his current location into a previous one. The screen displays texts and images in real time, according to the player's current location. In this way, the player can "go back" to branch into new scenarios.

3 Detailed Description of Components

3.1 UMLs

Class Diagram Figure 4 shows all the main components of our software. Class *Users* consists of two subclasses *Players* (ordinary users) and *Admin Users*. *Players* and *Admin Users* both can get access to the other three parts but with different qualifications.

1. For *Players*, its attributes include any information when a player participates in *Friends System*, *Games* and *Community* such as *Friends˙List* which should record who are his/her friends and if who are in game or offline.

2. And *Admin Users* equips with more permission to take care of the environment of the game.

3. Saving and loading game progress are key functions in *Games*. Also, achievements are now available for those who can not wait to show their wonderful performance in game.

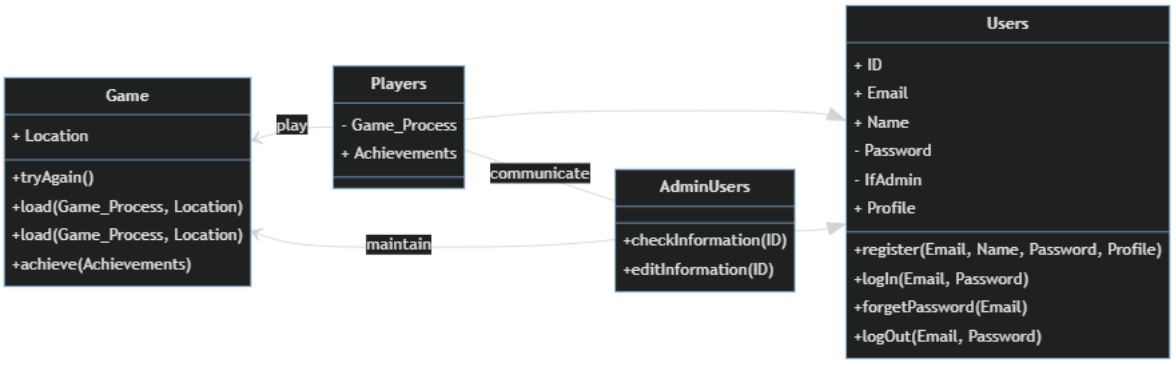

Figure 4: Class Diagram

Use Case Diagram Figure 5 shows all the functions that will be involved in our project and their relations with our players and administrator. These functions linked with users will be directly displayed in the user interface, and the other function will be shown during the game process. As is shown in the diagram, normal players and administrators have quite different functions, but some of their functions are dependent with each other. For example, administrator can only change the password after the user have registered. The users can also change password, but the difference is that normal user can only change his/her own password, while administrator can change any password

Sequence Diagram Figure 6 displays the whole process of a player's game experience. Since the main content of the system is an visual novel plot game mixed with some mini games, different essential parts like plots, choices, and games (representing mini games) are separated here to make the procedure more clearly.

After successful registration, the player can start game to enter the story. When encountering some essential plots, the player will be asked to make a choice which will impact the further story. After several choices, the player will confront a choice triggering a mini game. Different plots will trigger different game. If the player win the game, the story continues. If not, the player can try again until overcome the challenge or load the saved files to make a different choice. If the steps of the plots or the scores of the mini games satisfy the achievement conditions, players will get the corresponding titles and achievements. After repeating the above processes several time, the player will go to one of the ending plots and the game is over.

After ending the game, Players can restart to unlock other plots and endings. When the player is out of the game, he can go to personal center to change his information or upload photos. Administrators can access to any common users and modify their passwords.

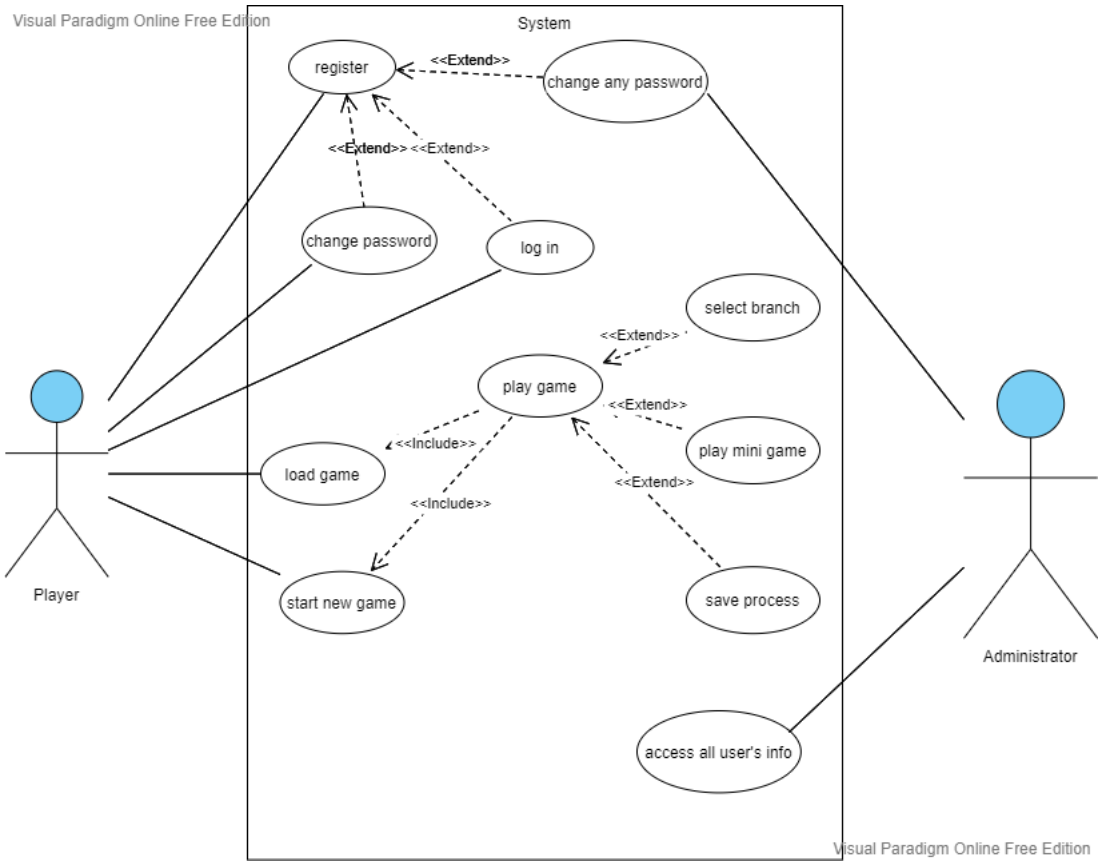

Figure 5: Use Case Diagram

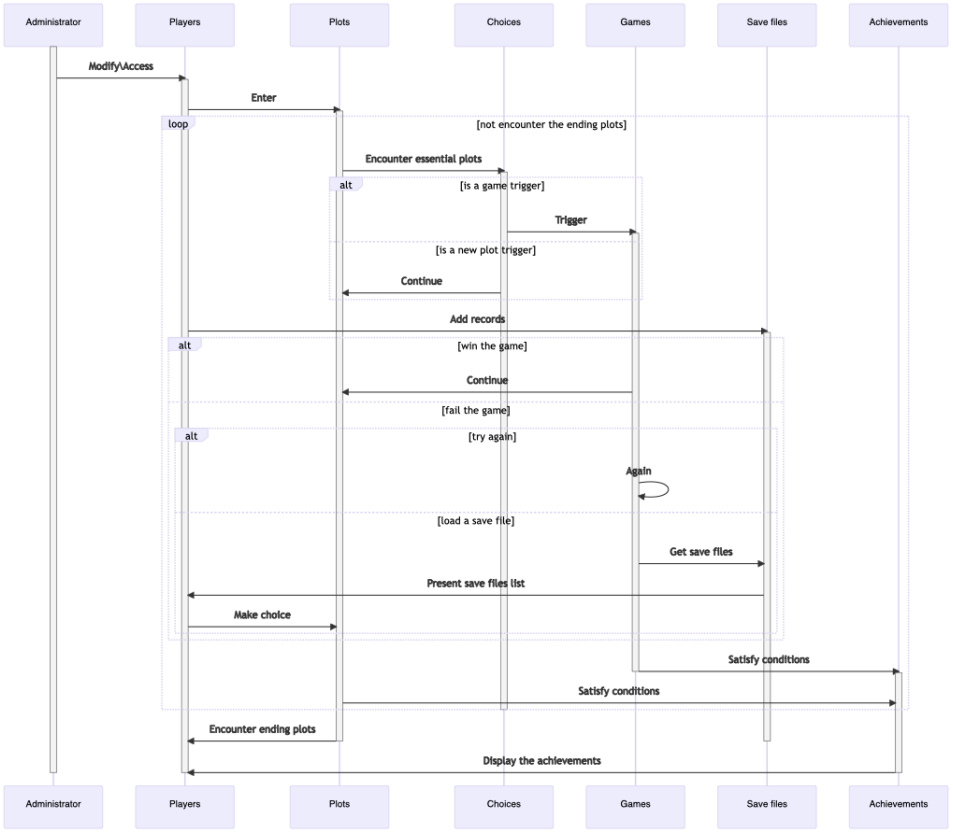

Figure 6: Sequence Diagram

3.2 Functionality

Sign up In the sign up part, the main functionalities are checking the input, sending email, and saving the information into database. For checking inputs, we check :

- whether the username or the password is empty
- whether the username has been used
- whether the password is longer than 8 bits
- whether the email is valid

Then, we send the identifying code to the email and the button will be disabled in one minute. After connecting with the back-end, we check the correctness of the identifying code. Only if all above constraints are satisfied, the user can sign up successfully. Then, we insert the user information into the database.

Login In the login page, the user can sign in with the correct information. For checking inputs, we check :

- whether the username or the password is empty
- whether the username is exist in the database
- whether the password is correct
- whether the login person is an administrator

Only if the information is correct, the user can login successfully. If he is a user, he will see the main game page after login. If he is an administrator, he will see the administrator page with the user list. If the user never registered before, he can click the 'sign in' button and turn to the sign in page.

Personal Center Click the 'Personal Center' button in the game page, the user can enter his personal center. The page displays username and email. User can change password in this page, still we check:

- whether the old password is correct
- whether the new password is longer than 8 bits
- whether the input of verification is the same as new password

Then, we change the password in the database. The user can also upload photos in this page. The image should be less than 4M. After uploading successfully, the information will be stored in the database. Next time the user opens the personal center page, the photo will be displayed. If the user logout, the page will turn to login page.

Administrator The administrator can see all the users in the database. All the user information except password will be displayed in the table list. Administrator can change the user's password with the username. We will check:

- whether the username is exist
- whether the new password is longer than 8 bits
- whether the input of verification is the same as new password

Then, we change the password of that user in the database. Administrator can click the 'update' button to see the change. If the administrator logout, the page will turn to login page.

Main Body: Visual Novel Like any other visual novel, the player can move the story forward by either 1. clicking the dialog box to continue or 2. clicking the choice button to make a choice. The player may 3. acquire some achievements by making certain choices.

What's more, the player can 4. save his/her current story progress or 5. load a progress saved before. The player can also 6. go back to the main menu and 7. start a new game, 8. load game, or 9. go to the personal center.

Mini Game 1: *Escape by Helicopter* In this mini-game, the users shall control the helicopter to avoid colliding with the obstacles and try to make it to a safe point to escape from the mutant zombie.

- **Helicopter Control** Use the mouse click to control the raising or dropping of the helicopter. Passing through one obstacle will make the user gain one point.
- **Winning State** After getting a certain points, the helicopter will get to a safe point and escape successfully, meanwhile the monster that is chasing behind will fade out. After that, the users can continue to the main game story.
- **Losing State** If the helicopter collide with the ground or the obstacles, it will fall and be caught by the zombie. In this case, the users fail the game.
- **Restart the Game** After failing the game, the users can click on the 'OK' button to restart the game and try again.

Mini Game 2: *Treasure Hunter* The core task in *Treasure Hunter* is to avoid zombies' attack, take the treasure chest and escape from the dungeon. Main functions in this mini game are listed below.

- **Game State Control**: *Treasure Hunter* contains two states (Running and Paused). Player is able to switch game state at any time. Detailed manual about the game will be printed on the screen for reference if game is paused.
- **Movement Control**: This control unit consists of two components: IO embedded movement control of adventurer and auto control of zombies. Player can guide the adventurer according to manual while zombies are always moving in their path.
- **Collision Detection**: There are two missions for this detection unit: preventing the adventurer from moving out of screen and deducting HP when zombies hit adventurer.
- **Dynamic Health Bar**: To help player have full awareness of game information, a real-time HP bar is provided. Game logic checks 60 times a second whether a zombie is damaging the adventurer. If so, the damage reflects immediately on this HP bar. During the next 0.5 second, program turns the adventurer into semi-transparent and free of any extra damage.

3.3 Procedure

Sign up The implementation of sign up is contained two parts, the front-end part in sign_up.js and the back-end part in server.js. In the sign_up.js, we implement the following

things mainly:

- Check whether the username or the password is empty

```
//username and password should not be empty
if (!username || !password) {
    alert('Please enter both username and password.');
```

Figure 7: Empty username or password

- Check whether the password is longer than 8 bits

```
// check whether the password is less than 8 bits
if (password.length < 8) {
    alert('The password is not strong enough.');
    return
```

Figure 8: Password should longer than 8 bits

- Check whether the email is valid

```
function isEmail(strEmail) {
    if (strEmail.search(/^\w+((-\w+)|(\.\w+))*\@[A-Za-z0-9]+((\.|-)[A-Za-z0-9]+)*\.[A-Za-z0
    else return false;
}
```

Figure 9: Check whether its a valid email

- Enable the button after 1 min

```
var myajax = $.ajax({
    url: "http://127.0.0.1:8080/verify",
    data: {
```

Figure 10: Requirement to verify identifying code

- Send the requirement to the back-end for sending email

```
// send the username and email to the back-end
$.get('http://127.0.0.1:8080/email', {
    email: email,
```

Figure 11: Requirement to send email

- Send the requirement to the back-end for verifying the identifying code

```
var myajax = $.ajax({
    url: "http://127.0.0.1:8080/verify",
    data: {
```

Figure 12: Requirement to verify identifying code

In the server.js, the sign up part is mainly about how will the server deal with the requirements about sending email and verifying code.

```
// send email
app.get('/email', (req, res) => {
    my_v_code = createSixNum();
```

Figure 13: Send email

```
// verify the identification code
app.get('/verify', (req, res) => {

    // the query to check whether the username ha
    const sqlStr1 = 'select * from user where use
```

Figure 14: Verify identifying code

Login In the login part, we mainly have two parts: the front-end part in index.js and the back-end part in server.js. In the index.js, we mainly implement:

- Check whether the username or the password is empty

```
//username和password不能为空
if (!username || !password) {
    alert('Please enter both username and password.');
    return
```

Figure 15: Empty username or password

- Click the 'sign up' button can turn to the sign up page

```
function sign_up() {
    setTimeout("window.location = 'sign_up.html'", 500);
}
```

Figure 16: Jump to sign up page

- Send the requirement to the back-end for login

```
var myajax = $.ajax({
    url: "http://127.0.0.1:8080/login",
```

Figure 17: Requirement to login

In the server.js, we mainly implement:

- Deal with the requirement from the front-end

```
// login
app.get('/login', (req, res) => {
```

Figure 18: Deal with login requirement

- Check whether the username is exist

```
// the query to find whether the username is exist in the DB
const sqlStr1 = 'select * from user where username = ?';
var param = [req.query.username];
```

Figure 19: Check whether the user exist

- Check whether the password is correct

```
// Then, check whether the password is correct
const sqlStr2 = 'select password from user where username = ?';
var param = [req.query.username];
```

Figure 20: Check whether the password is correct

- Check whether the user is an administrator

```
// Then, check whether it is an administrator
// Send the flags to the front-end, deciding which page he will
const sqlStr3 = 'select isAdmin from user where username = ?';
var param = [req.query.username];
```

Figure 21: Check whether the user is an administrator

Personal Center In the personal center part, we mainly have two parts: the front-end part in my.js and the back-end part in server.js. In the my.js, we mainly implement:

- Display the information of the user

```
// display the user information as soon as openning the page
window.onload = function() {
    var status = false;
```

Figure 22: Display the information of the user

- Modify password

```
// modify the password with old password and valid ne
function modify_pwd() {
    old_pwd = document.getElementById('old').value;
```

Figure 23: Modify password

- Upload photos

```
//upload photo
$(function() {
    // ajax
    function update(url, formdata, callback) {
        $.ajax({
```

Figure 24: Upload photos

- Requirement to modify password and get the information of the user

```
    var myajax = $.ajax({
        url: "http://127.0.0.1:8080/my",
        data: {
```

Figure 25: Requirement to get the information or modify the password

- Requirement to upload photos

```
var url = "http://127.0.0.1:8080/picture"
var imgFiles = $("#pic")[0].files[0]
```

Figure 26: Requirement to upload photos

In the server.js, we mainly implement:

- Send the user information

```
app.get('/my', (req, res) => {
    if (req.query.flag == 0) {
        res.send({
            current_user: temp_user,
```

Figure 27: Send the user information

- Modify the password in the database

```
// modify the password
const sqlStr6 = 'update user set password = ? where username = ?';
var param = [new_pwd, req.query.current_user.username];
```

Figure 28: Modify the password

- Upload photos and save in the database

```
// upload photos
app.post('/picture', upload.single('imges'), (req, res) => {
    var imges = req.file
```

Figure 29: Upload photos

Administrator In the administrator part, we mainly have two parts: the front-end part in admin.js and the back-end part in server.js. In the admin.js, we mainly implement:

- Display the user list

```
// display user list when open the page
window.onload = function() {
    var status = false;
```

Figure 30: Display the user list

- Modify password according to username

```
// modify password according to the username
function modify_pwd() {
```

Figure 31: Modify password

- Update information after changing the password

```
// update infomation
function update_info() {
    var status = false;
```

Figure 32: Update information

- Requirement to do all the things above

```
// ajax send application to the back-end
var myajax = $.ajax({
    url: "http://127.0.0.1:8080/admin",
    data: {
```

Figure 33: Requirement send to the back-end

In the server.js, we mainly implement:

- Deal with the requirement from front-end

```
// administrator
app.get('/admin', (req, res) => {
    if (req.query.command == 'show_all_user') {
```

Figure 34: Deal with the requirement from front-end

- Display the user list

```
// display all users' information
const sqlStr1 = 'select * from user';
var param = [];
```

Figure 35: Display the user list

- Modify the password in the database

```
// modify a user's password according to the username
const sqlStr5 = 'update user set password = ? where username = ?';
var param = [req.query.new, req.query.username];
```

Figure 36: Modify the password in the database

Visual Novel All functions involved in this part are in game.js.

- Display a scene

```
// display the nth scene
function display(n) {
    // display achievement
    if (someScenes.indexOf(n) != -1) {
```

Figure 37: Display a scene

- Start a new game

```
// New Game button
starter.children[0].addEventListener('click', function() {
    starter.style.display = 'none';
```

Figure 38: Start a new game

- Load game

```
// Load Game button
starter.children[1].addEventListener('click', function() {
    starter.style.display = 'none';
```

Figure 39: Load game

- Ask the player for a name

```
// ask player for a name of the main character
function enter_name() {
    clearScreen();
```

Figure 40: Ask the player for a name

- Adjust the name given by the player

```
// used to adjust the name the player gives the main character
button4.onclick = function() {
    var yourName = document.getElementById('yourName');
```

Figure 41: Adjust the name given by the player

- Continue the dialog

```
// click the conversation box to continue
textBox.onclick = function() {
    if (typeof graph[currentScene] == "number") {
```

Figure 42: Continue the dialog

- Make a choice

```
// click choice button to respond
button1.onclick = function() {
button2.onclick = function() {
button3.onclick = function() {
```

Figure 43: Make a choice

- Save or load a record

```
// save or load a specific record, there are totally 10 locations
location1.onclick = function() {
    if (mode == 'save') {
        var today = new Date();
```

Figure 44: Save or load a record (e.g. the top one)

- Return to the main menu

```
// go back to the main menu
backButton.onclick = function() {
    clearScreen();
```

Figure 45: Return to the main menu

Mini Game 1: *Escape by Helicopter* The implementation of the functions in this mini-game is mainly composed of 4 parts: helicopter's animation and control, monster's animation and control, obstacles generation and movement, collision and game state monitor,

- **Helicopter's Animation and Control** The control function of this part is all in file bird.js. After the user click the start button, the copter will be inserted in the screen by function showbird(). And the animation of the copter is controlled by wingwave() function which is based on the frame-by-frame animation. Meanwhile, the flying and the dropping of the helicopter is controlled by flybird() function. If the falling speed of the copter is larger than 0, the bird will fall down, but the user can click to minus falling speed by 8 to raise the copter.
- **Monster's Animation and Control** The control function of this part is in file copter.js. After the game starts, the monster will be flying on the left upper corner of the screen. The waving animation of the monster is controlled by function monster_w() which is also based on frame-by-frame animation. The monster can only catch the helicopter when the copter collides or falls down, the catching function of it is controlled by function monster_catch(). And it will fade out of the screen if the player wins, this animation is controlled by function monster_lose().
- **Obstacles Generation and Movement** The generation of the block obstacles is controlled by function Block() in file block.js. The position of the blocks and the interval distances between two obstacles are generated randomly. The movement of the obstacles are implemented in function landRun() in copter.js. It will move the blocks and the grass land towards the copter to make the user feel the copter is moving forward.
- **Collision and Game State Monitor** The collision judgement directly determine the game's state. It's implemented in rectangleCrashExamine() function of file baseObj.js. It will judge if two rectangles collide with each other, so that it can be used to detect the collision between the helicopter and the land or the obstacles. The game state is also judged in function landRun() of file copter.js based on both the collision monitor and the score of user.

Mini Game 2: Treasure Hunter As is described in section *Functionality*, *Treasure Hunter* is composed of 4 major components. Specific implementation of these components are as follows.

- **Game State Control**: The game is refreshed 60 times per second by pixi engine application *ticker*. In this case, starting game turns into switching the visibility of

```
// update argument and wait for retry() to restart game
function driver(){
  startScene.visible = false;   // switch to game scene
  app.ticker.start();           // active ticker and start gameloop
  ifPause = 0;
}
```

Figure 46: Activator Function: driver()

```
// loop through all the zombies' movement
let damage = false, sin, cos;
zombies.forEach(function(zombie, index) {
  // one zombie is always chasing the adventurer
  if (index == 3){
    sin = (player.y - zombie.y) / Math.sqrt(Math.pow((player.x -
    cos = (player.x - zombie.x) / Math.sqrt(Math.pow((player.x -
    zombie.vx = ZOMBIE_SPEED * cos;
    zombie.vy = ZOMBIE_SPEED * sin;
  }
```

Figure 47: Auto Movement Control

corresponding messages and stopping ticker, as shown in Figure 46. Similar tricks are imported to pause game.

- **Movement Control**:

  Adventurer's movement and zombies' movement are both implemented in game logic (function *play(delta)* in source file). Computer loops through every zombie and defines their movement. Refer to Figure 47 for one of zombies' movement.

- **Collision Detection**:

  Collision Detection is used when adventurer meets the boundaries, when he gets treasure chest and when zombies do damage to him. Trivial instructions in detect function *hitTestRectangle(r1, r2)* are presented in Figure 48.

```
// Check for a collision on the x axis
if (Math.abs(vx) < combinedHalfWidths) {
  // A collision might be occuring. Check for a collision on the y axis
  if (Math.abs(vy) < combinedHalfHeights) {
    hit = true;
  } else {
    hit = false;
```

Figure 48: Collision Detection

```
// damage event
if(damage) {
  // player becomes semi-transparent
  player.alpha = 0.5;

  // reduce HP
  healthBar.outer.width -= DAMAGE;
```

Figure 49: Damage

- **Dynamic Health Bar**: Dynamic Health Bar is made of two rectangle (one is black and the other is red). When being hurt, red one reduces its width (Figure 49).

4 User Interface Design

4.1 Description

Bar The name of our website is Foredawn which means the dawn will come soon. And the logo of our website is a png of fire standing for hope and warmth.

Login page Each time a user opens our website, the login page will be seen. The login page takes two pictures which are soldier and ruins as background. The head indicates our logo, user label, and the name of our game "Foredawn" in a proper font. The footer displays our group information. If the user has been signed up, he can enter the correct information in the black bar and click the sign in button to login. If the user never signed up before, he can click the sign up button to jump to the sign up page to register. The other robust settings have been explained in the functionality part.

Sign up page The basic design of the sign up page is the same as login page. The main difference of it is input bars and buttons. The user could enter username, password (longer than 8 bits), and valid email. Then click the "Send identifying code" to send the identifying code through email. The "Send identifying code" will turn to "Please wait for x seconds" to countdown 60 seconds before the button be enable again. If the user enter the correct identifying code, he can click the sign up button to register. The other robust settings have been explained in the functionality part.

Personal center page The personal center page chooses a lighter and hopeful image as background, which indicates the player is the hope of this game. The header and footer is similar to the login and sign up pages with the difference of a "back to game" button which will jump to the game page. The username, photo, and email will be displayed according to the login information. The user can select photos on the computer to upload by clicking "upload". After entering the correct old password, valid new password, and verification, the user can click "modify" to change password. If the user clicks "logout", the page will jump to login page. If the user tries to access this page through hyperlink

without login, the page will be blank. The other robust settings have been explained in the functionality part.

Administrator page The administrator page has similar design as personal center page. The main part is the information list table of all users. The administrator can modify the users' password in the black input bar and click "modify" button, according to the username (with the rule of unique). Then click the "update" button, to see the change. If the administrator clicks "logout", the page will jump to login page. If someone tries to access this page through hyperlink without login, the page will be blank. The other robust settings have been explained in the functionality part.

Main menu page The main menu page consists of a background image, the "New Game" button, the "Load Game" button, and the "Personal Center" button.

Visual novel page The visual novel page consists of a background image and an interactive box. The interactive box consists of the speaker's information (name photo image), the words said by the speaker, the choice buttons (if there's any), the "Back" button, the "Save" button, the "Load" button, and the achievement message (if there's one).

Saved records page The saved records page consist of a window with ten records (because we limit the maximum number of records to be ten) and a cross button. The non-empty record shows the name given by the player, the scene name, and the time when this record was saved.

4.2 Screen Images

Figure 50: Bar

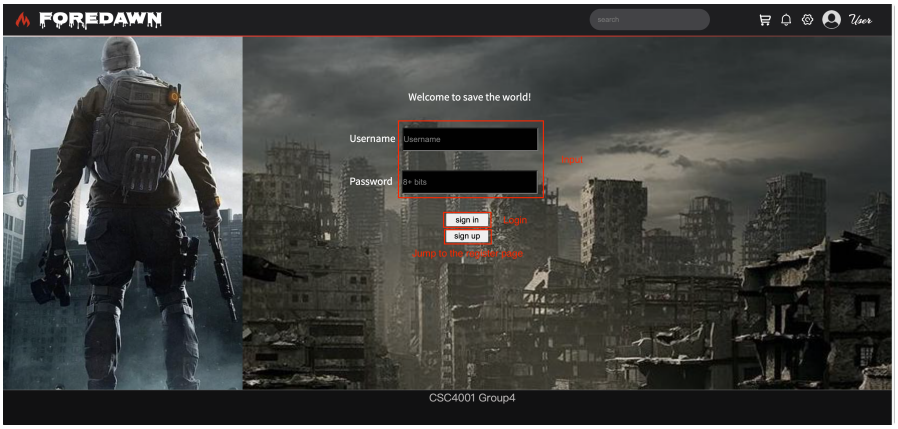

Figure 51: Login page

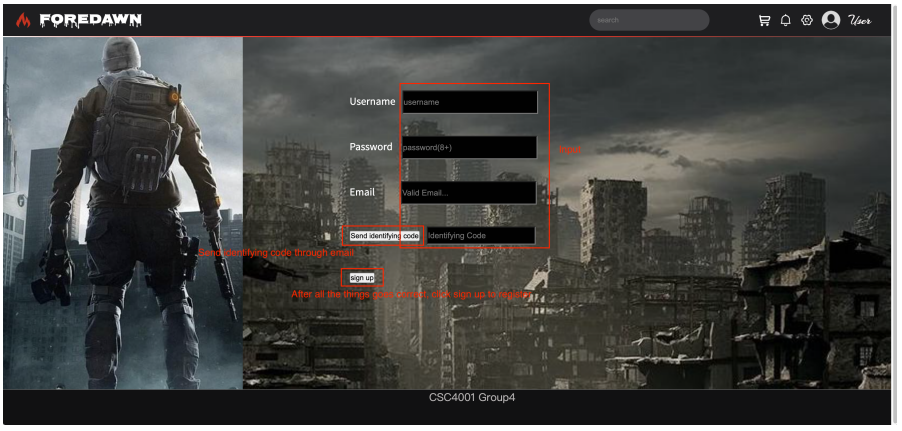

Figure 52: Sign up page

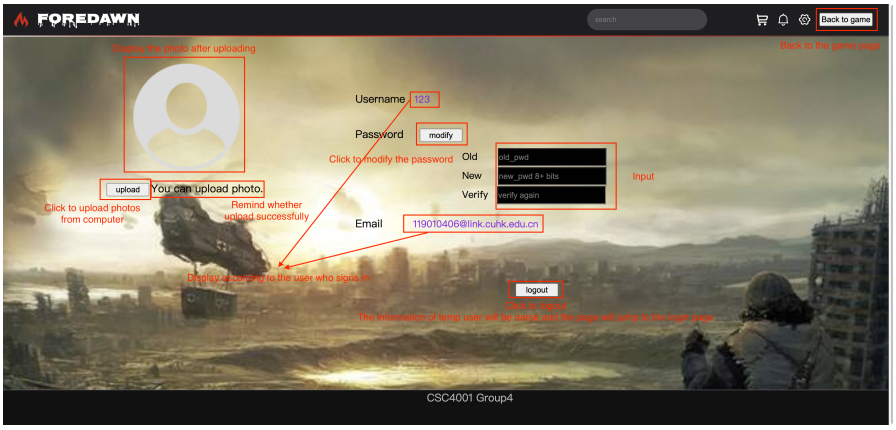

Figure 53: Personal center page

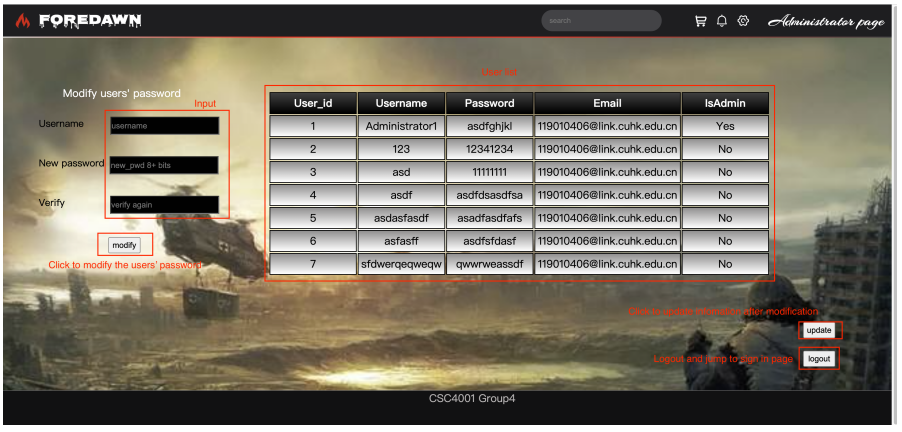

Figure 54: Administrator page

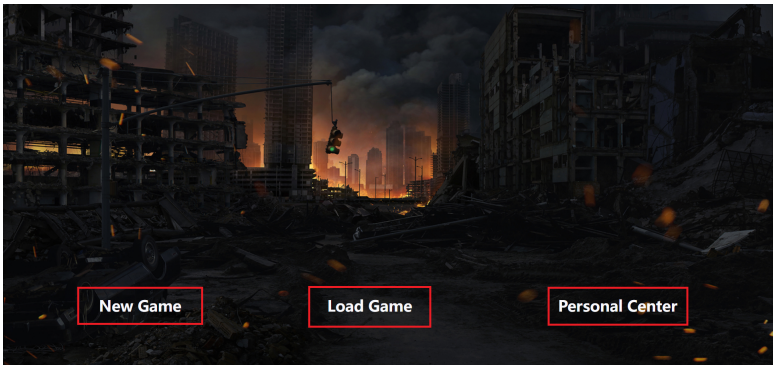

Figure 55: Main menu page

Figure 56: Visual novel without choices

Figure 57: Visual novel with choices

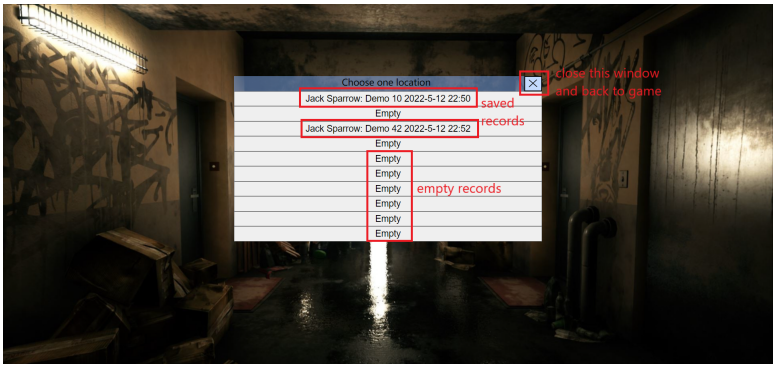

Figure 58: Saved records page

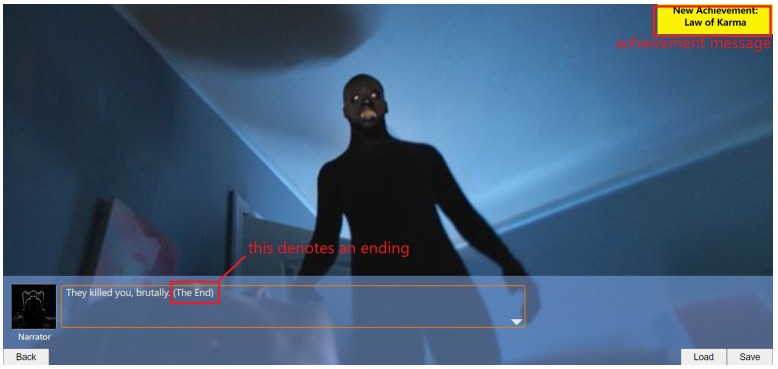

Figure 59: Achievement and ending message

4.3 Objects and Actions

4.3.1 Administrator

For administrators, the only two interface he will access are the login page and the administrator page.

The administrator can login through login page with the correct information. Then, he can see the administrator page. The administrator page automatically display the user list once the administrator login. Administrator can change the users' password with the corresponding username (with the rule of being unique) in the left three input bars. Click the "modify" button can apply the modification and then click the "update" button can see the change. If he click the "logout" button, the account will be logout and the page will turn to the login page.

4.3.2 Player

For a player (user) of this website, he can access almost all web pages except the administrator page, including sign up page, login page, game menu page, game content, and

personal center page.

First, the player will see the login page. If he has signed up, he can login with the correct information. If he never signed up before, he can click the "sign up" button to turn to the sign up page. After registering successfully, he can login with the username and password. Then, he will enter the game main menu page. If he clicks the "Personal Center" button, he will see the personal center page. He can upload profile photos, change password, or go back to game by clicking "Back to game" button.

If he plays this game the first time, he can click the "New Game" button to start. If he has saved some game processes before, he can click the "Load Game" button to load game and continue with a game process saved before. In the game process, the player can click the dialog box to see the following story. When he encounters choices, he can choose a provided button. When he encounters mini games, he should play them. At any time in the game, the player can save the current process by clicking the "Save" button. If he chooses a location where a process has already been saved, the current process will replace the old one. After making some crucial choices, the player will receive an achievement on the right up corner of the screen. If the player wants to stop the game before reaching one ending, he should first save the current process and then he can click the "Back" button and return to the main menu page. However, if the player forgot to save his process and he just went to the main menu, don't worry, he can return his recent process by clicking the "Load Game" button and the cross button without choosing any saved process.

5 Test

5.1 Test Plan

The test plan mainly contains four parts, the front and back end interaction, behaviours of web ADV game, test on functions of mini-game 1, and mini-game 2.

5.2 Case-n input and expected output

Front and back end interaction Figure 60 to figure 64 will list the test cases for registration page, login page, personal center page(including password change and upload photo profile), and administrator page.

For the test case of administrator page, there are already three users registered: Administrator1 with password asdfghjkl, User1 with password 123abcdefg, Strv with password 123123123.

Test on functions of mini-game 1 We test the mini-game on these game states:

- **Start Game**
- **Helicopter Control**
- **Collide with ground or block obstacles**

Case Number	Username	Password	Email	Identify code	Expected output
1	User1	00	123	None	Wrong Password
2		12	1779598903@qq.com	None	Wrong Username
3	User1	1a2b3c4d5	1234	None	Wrong Email
4	User1	12345678	1779598903@qq.com	Correct code	Registration success
5	User1	12345678	3391456114@qq.com	Correct code	Duplicated Username
6	User2	123456asd	3391456114@qq.com	Wrong code	Wrong identify code

Figure 60: registration test cases

Case Number	Username	Password	Expected output
1	User1	12345678	Login successfully
2	datutu	asdfghjkl	No such user
3	User1	abcdefgh	Wrong password
4	User1		Alert empty password
5		12345678	Alert empty username
6			Alert empty username and password

Figure 61: login test cases

Using account User1 to test password change

Case Number	Old password	New password	Verify password	Expected output
1	12345678	asdfghjkl	abc	Please verify new password again
2	123abcdef	asdfghjkl	asdfghjkl	Wrong old password
3		123abcdefg	123abcdefg	Wrong old password
4		123abcdefg	12345678	Please verify new password again
5	12345678	123abcdefg	123abcdefg	Change successfully

Figure 62: password change cases

Using account User1 to test photo upload

Case Number	Upload photo	Expected output
1	PNG photo with size < 4MB	Upload success
2	PNG photo with size>4MB	Alert the size problem, fail
3	JPG photo with size<4MB	Upload success
4	JPG photo with size>4MB	Alert the size problem, fail

Figure 63: upload photo cases

Using administrator account: Administrator1 to test administrator's changing password

Case Number	Input username	New password	Verify password	Expected output
1	User_nobody	123123123	123123123	Fail to change password
2	User_nobody2	12341234	abcd1234	Alert wrong verify password
3	User1	1a2b	1a2b	Alert password too short
4	Strv	1234abcd	1234abcd	Change successfully
5	User1	abcd1234	abcd1234	Change successfully
6	Administrator1	asdfghjkl1234	asdfghjkl1234	Change successfully

Figure 64: password change of administrator

input	expected output
w	move up
a	move left
s	move down
d	move right

Table 1: Case 1: Movement Control

input	expected output
space	game pauses
click "go"	game resumes
click "try again"	game restarts
click "next"	game quits

Table 2: Case 2: Game State Control

- **Restart game**
- **Winning Game**

Mini Game 2: Treasure Hunter Refer to Table 1 and 2.

5.2.1 Purpose of Test Cases

The purpose of the test cases above are listed here.

Front and back end interaction The purpose of the front and back end interaction test cases above are listed as following.

Test cases purpose for the registration page:

- **Case 1** To test whether the program can identify the registration password that is less than 8 bits and alert the user.
- **Case 2** To test whether the empty registration user name can be identified.
- **Case 3** To test recognition of invalid email format.
- **Case 4** Test valid registration information.
- **Case 5** Test if the program can detect duplicated username and alert the user.
- **Case 6** Test whether the program can detect incorrect identify code and raise alert.

Test cases purpose for the login page:

- **Case 1** Test for valid login information
- **Case 2** Test for detecting unregistered user
- **Case 3** Test for registered user login with wrong password
- **Case 4** Test for handling empty password
- **Case 5** Test for handling empty username
- **Case 6** Test for handling empty password and username

Test cases purpose for the password change, note that the original code for this account is 12345678.

- **Case 1** Test for handling the case when new password and verify password is different.
- **Case 2** Test for the case when the old password is not entered correctly.
- **Case 3** Test for empty old password.
- **Case 4** Test for invalid old password while the new password and verify password is different.
- **Case 5** Test for valid information.

Test cases purpose for the upload photo profile:

- **Case 1** Test for uploading png file with size smaller than the maximum size
- **Case 2** Test for png file whose size exceed the maximum size.
- **Case 3** Test for uploading jpg file with size smaller than the maximum size
- **Case 4** Test for jpg file whose size exceed the maximum size.

Test cases purpose for password change of administrator:

- **Case 1** Test for changing password of an unregistered user, expecting a fail alert
- **Case 2** Test for the case when the input password and verify password is different.
- **Case 3** Detection for changing the password to a new password with length less than 8 bits.
- **Case 4** Test for changing valid normal user with valid new password.
- **Case 5** Test for changing valid normal user with valid new password.
- **Case 6** Test for changing valid administrator user with valid new password.

input	output
w	move up
a	move left
s	move down
d	move right

Table 3: Case 1: Movement Control

input	output
space	game pauses
click "go"	game resumes
click "try again"	game restarts
click "next"	game quits

Table 4: Case 2: Game State Control

5.2.2 Test Outputs Pass/Fail

Front and back end interaction test output We use the test cases above to help to debug, and the last version of our back-end program pass all the tests.

Test output of functions of mini-game 1 figure 65 to figure 69

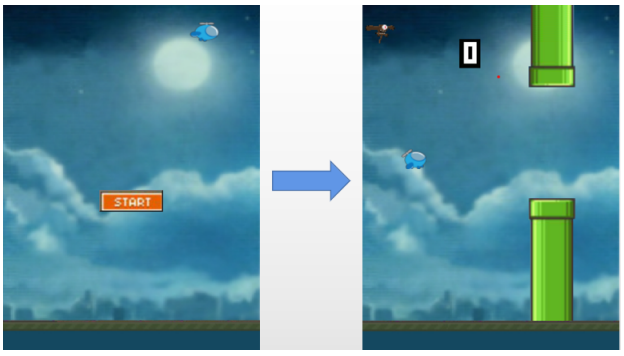

Figure 65: start mini-game 1

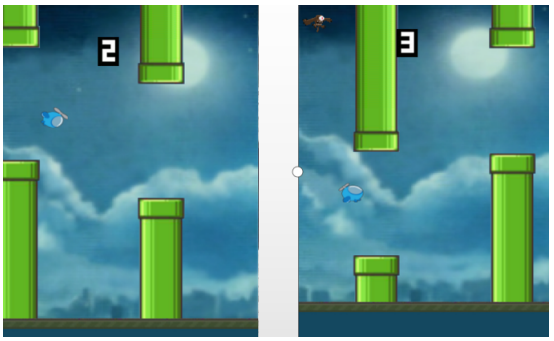

Figure 66: fall and raise control of helicopter

Mini Game 2: Treasure Hunter Refer to Table 3 and 4.

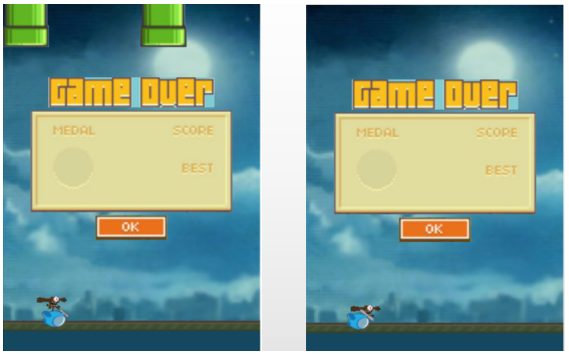

Figure 67: collide with ground or obstacles

Figure 68: restart game test

Figure 69: player win, monster fade and winning info display

6 Learning Outcomes

6.1 Procedure and Test

Drawing the UML diagram gave us a goog chance to get familar with the UML diagrams, and it helps a lot when designing the structure of our software. Also, We have learned the method to set test cases for our program which covers most of the possible cases.

6.2 New technology

We have learned the front and back end interaction using node.js and jQuery's Ajax. The mysql database applied in practice. Design a mini game using original JavaScript and the web game engine: pixi.

7 References

https://en.wikipedia.org/wiki/Visual˙novel#RPG˙hybrids

https://en.wikipedia.org/wiki/Resident_Evil

https://www.nationalgeographic.com/culture/article/chernobyl-disaster

https://nuke.fas.org/guide/japan/bw/