CS443 Final Project Report

Keep the Light On

Team member: Kun Mo, Yongqi Zhang

Instructor: Sheng Bo

1. Project Statement

Our final project is to develop a graveyard-themed survival shooting game that can play through Android phones. This app is designed for relaxation and entertainment for all-aged group. It is an advertisement-free and wifi-free mobile game. Our user will play the role of graveyard keeper whose job is to keep the graveyard from zombies attack. As he needs the lamp to see through the dark, he has to protect the lamp also. The light inside the lamp will attract zombies from very far away. Even the graveyard keeper is armed with a weapon, he has to protect himself too. The zombies will get outraged if they are attacked by the graveyard keeper. They will turn to our graveyard keeper and want to have a taste of blood.

This app can be played on android phone with API level higher than 23.

2. Application Design

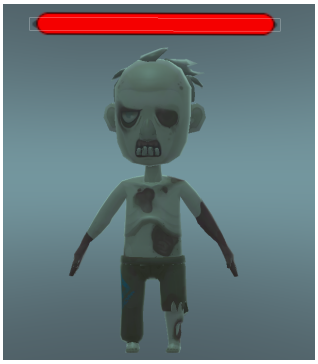
1) There are three major components (or game object) in the game described as following:
    1. Graveyard Keeper
        a. Our main character in this game, his goal is fighting with infinite waves of zombies to keep the light in the lamp on.
        b. Controlled by the virtual controller
        c. Armed with a gun which automatically fired 0.2 second
        d. Attained certain HP value; when it's zero, it indicates gameover condition
    2. The lamp
        a. Attained certain HP value; when it's zero, it indicate gameover condition
        b. Flash every second, given a realistic feeling of dark, windy night
    3. Zombies
        a. Their goals is to destroy the lamp
        b. Follow and attack anyone who harm them
        c. Attained certain HP value

d. Spawn in waves every 10 seconds

2) Other components:

4. Wall and Graves
    a. Zombies and graveyard keeper cannot go through the fence
    b. Zombies cannot walk on tiles and tombs
    c. Weapons can't fire through the wall and tombs
5. UI design
    a. Main Menu -start & exit the game

    b. Scoring system - display current score our user get during and after the game.
    c. Health Bar - indicate current HP



6. Music

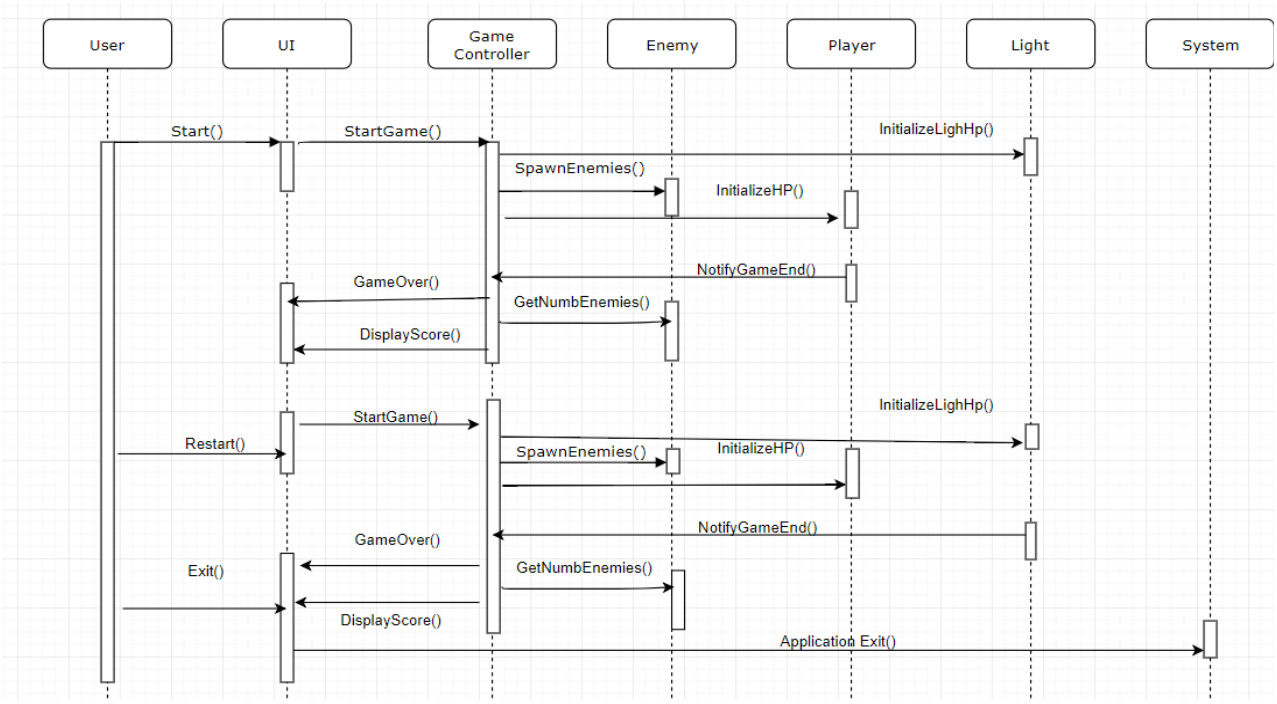
a. One background music as the user enter our game
b. Gunshot fired sound
c. Sound effect for the dead of zombie
d. Sound effect for the zombie when they get hurt

3)Game Logic

- The lamp will be placed in the center of the game world
- Graveyard Keeper will be placed near the lamp at the beginning of the game.

- Zombies will be generated every certain period of time; each of them has different HP values.
- GameOver condition: Graveyard Keeper is dead or the light in the lamp is off.

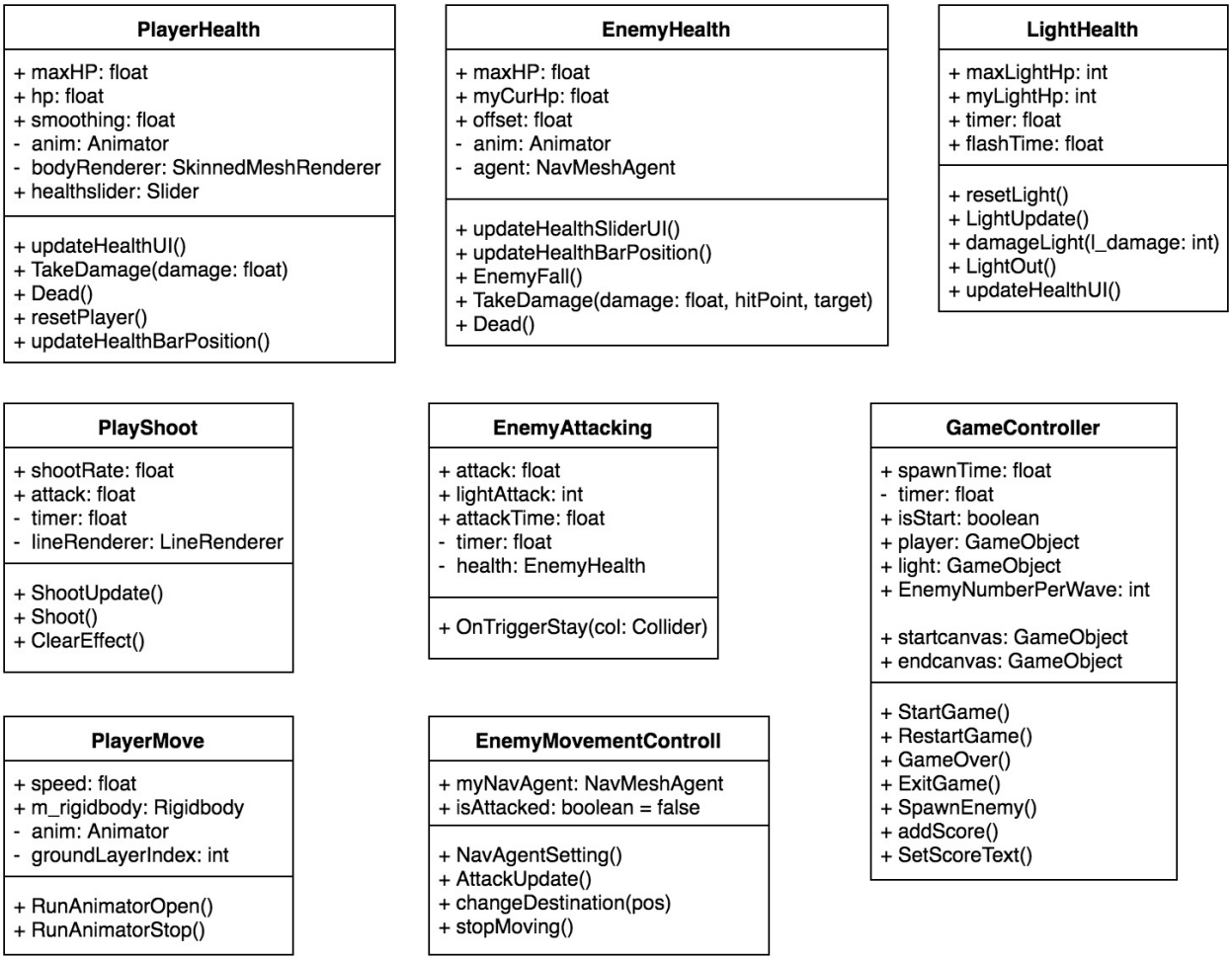
4)The flow of our game



When the app is open, it shows a start menu with two buttons. When the user clicks on the start button, the corresponding game controller method will notify to start the game. The game controller controls the initialize necessary parameters such as player's health, initialization of the wave of enemies, the light's health and so on to start the game. It also constantly detect whether the game is over, and display gameover menu and total score to on the screen.

3. Application Implementation and Evaluation

1) Class Diagram:



We defined eight classes in the game:

"PlayerHealth" class is to manage health condition of the player including taking damage from zombies. It also manages the change of skin color(when he gets hurts), the animation of different player state(e.g. Walking state,dead state and standing) , and the sinking effect after death.

"PlayerMove" class is to manage the movement of the player and the move animation play.

"PlayerShoot" class is to manage the shooting rate, shooting effects and shooting damage of player's gun.

"EnemyHealth" class is to manage the health condition of the enemy, the walking and standing animation and the transition to dead state.

“EnemyMovementControll” class is to manage movement targets of enemies in different situations e.g. before the attack and after the attack.

“EnemyAttacking” class is to manage the frequency of the attack, attack range and damages of each attack the enemy could bring.

“LightHealth” class is to manage the health condition of light.

“GameController” class is to manage the start and end menu, initialization/ rest of game objects, and the score system.

These classes are interacted with each other. In “PlayerHealth” class, “PlayerMove” and “PlayerShoot” classes are disabled when it detects player health is equal or below zero. Then it will call the dead method and call the “GameOver” method in “GameControll” class to notify the game is end. After that, the “GameControll” will pop out the gameover menu. Similarly, In “LightHealth” class, the GameOver method in “GameController” class will be called when the light health is equal or below zero. The lamp and player will be reset when the restart method is called in the “GameController” class.

The “PlayerShoot” class will call the TakeDamage method in “EnemyHealth” when the bullet hit the enemy. This is how the enemies can be hurt by the player. In “EnemyHealth” class, “EnemyMovementControll” and “EnemyAttacking” classes will be disabled when enemy health is equal or below to zero. When this happens, it will call dead method. Then addScore method in “GameController” class will be called by the “EnemyHealth”. This is how the enemies will stop moving and attacking if they are dead, and how the score get incremented. In “EnemyAttacking” class, it utilizes the hp in the “EnemyHealth” class to determine whether or not the enemies can attack others. The “EnemyAttacking” class will call the damageLight method in “LightHealth” class or the TakeDamage method in “PlayerHealth”class if either the lamp or player is near the enemy.

2)Testing

We used Android Virtual Device in Android Studio for testing. We didn't have enough time to program a virtual controller directly for Android devices, so we used a third party plugin to integrate with our program. However, some problems happened when we used plugin of virtual controller. The first problem is the plugin of virtual controller conflict with keyboard control input. In order to test it on PC, we have to comment out the piece of code we program for PC control. The second problem we encounter was that the movement animation cannot be played while the player is moving. As a result, we wrote two methods in "PlayerMove" script in order to incorporate the plugin with the player movement animation. In addition, we found that the control of Android phone is different from computers. Controlling the movement, rotation and shooting of player at the same time is very difficult and inconvenience on the phone. So for a better gameplay experience, we make the gun fired automatically.

We also tried to create a foggy and dark environment to make our game more realistic to the users. When we use the real-time light generation in unity, it renders what we want. However, when we export it to the phone, it does not render the same environment as we see on the PC because the hardware does not support the real-time light generation functionality. We have to find another way to achieve what we want. By reading the documentation and testing it on the android virtual device, we finally come out with using baked light functionality to achieve the goal.

When we first make our health UI for our player, we simply put it under our player object. When we test our game, we found out the UI not always facing the camera since our player could turn around. Eventually, we were able to solve the problem by the program it to rotate back to original place.

4. References

Zombie and player models and their animations are from unity asset store.

Virtual controllers are plug-in from unity asset store.

5. Experiences and Thoughts

The project offers us a great opportunity to learn Unity and start building a small game project that can play on PC and android phone platform. By the time, the game we develop only has one gaming mode, which is an endless fighting mode. It would be nice if we have more time to develop a variety of difficulty levels that allow users to select and play. Generate a level with specific difficulty could be done using some optimization techniques such as the genetic algorithm. We can define a game level difficulty as a function, which is consists of the following elements: number of waves of enemies, enemy health, the duration between each wave and number of enemy in a wave, player health and so on. By entering the level difficulty we want, the game system can simultaneously generate a new level with such difficulty.

Due to time limitation, we can't create different types of weapons. If we have enough time, we will find models of weapons and set up their animations in our player. We will put and hide all weapon models in player's hand first. Then we will randomly place "weapon" cards in the scene. Depend on what type of weapon card the player picks up, we will enable that type of weapon in the player's hand. We will use "GameController" class to manage the generation of weapon cards. Finally, adding more methods in PlayerShoot class for different type of weapons so that we can control the gun fire animations and its power.