IERG3080 Project Part 2 Report

Hung Man Kei (SID:1155127099)

Tsai Kwun Ki (SID: 1155126289)

**Work Division**

Hung Man Kei: Pokemon Class, MiniGame, Start, MainWindow

Tsai Kwun Ki: MainWindow, GymWindow, Bag Class, ManageWindow

**Class Design**

**MainWindow:**

 The Main Window class is actually the map of the game, and the player is controlled by WASD. In the beginning there will be initialization of all pokemons. The class will check player movement for every 20 millisecond and check if the player is entered area including GYM on the right hand side building, Home (pokemon management) on the left hand side building, and wild pokemon (mini-game) randomly move for every second in the grass area. (invisible just like the classic pokemon game).

Methods:

Canvas\_KeyDown: Trigger when keyboard key is pressed

Canvas\_KeyUp: Trigger when keyboard key is released

Pokemons\_Initialize: Pokemons initialization and bag initialize

miniGame\_Area\_Time\_Tick: Random set miniGame trigger area for every second

Time\_tick: Check player movement and check any event triggered

Check\_Event: check any event triggered

Movement\_reset: To reset movement to false since when entering the area the key is pressed down

randomPokemon: A function to random generate a pokemon

For details, comments are included in the code.

**StartWindow:**

 For start window, it is initial choice for player to choose one pokemon to start the game, the class will only be used once in the beginning and 3 buttons are included for choosing which pokemon

**GymWindow:**

For the gym window there’s mainly two areas for the UI, Enemy area and player area, then there’s two buttons, one is the attack button and the other button is the end button (for escape). And in the player and enemy area there will be two rectangle for pokemon’s image and their name and HP status.

For the code setUp() is for setting up initial things above (pokemons, HP etc.), and there is few method trigger by button which is Attack\_Click and End\_Click. For the battle, it’s just simply player attack first, and check enemy die or not, then enemy attack back, check player die or not. Simple.

Methods:

setUp() Initialize

End\_Click : When end button is clicked, end the gym battle

Attack\_Click : When attack button clicked, player attack first and enemy fight back

Check(): method to check any side has lower or equal to zero health

Gym\_End()  method to end the gym page

For details, comments are included in the code.

**MiniGameWindow:**

MiniGame have a wheel, 2 button and the pokemon, firstly, pass the random pokemon into the game, and pass the bag, when player catch successfully, it will add the Pokemon into the Player’s Bag. In the GameEngine, the game is playing a random draw, having a wheel and pointer, when it point to grey or blue mean success, otherwise fail. In the Wheel class, there will generate a random wheel, the Game engine will have method to check the result, and also, it will have the spinning methond to control the wheel the spin and stop as well. In the Window interface will have the button to control it.

**Pokemon:**

In Pokemon class design, there are two other classes have to know first, one is “Dictionary”, one is “SeriesDictionart”.” Dictionary” is a object that saving each Pokemon, and “SeriesDictionary” is save the Pokemon which is the same series, it has a integer field call “current”, use to save the index of the Pokemon. So the Pokemon Class is inherit “SeriesDictionary”, it have a lot of specific field like HP, epx, mp ,etc. Pokemon is really like a object and a Pokemon that Player really will get and save in player Bag. Also, in the Pokemon Class, there are some function that can use, like evolve, power up, and attack etc. These functions are do by the Pokemon, Also, SeriesDictionary is have save a current, which is saving the specific pokemon, and when current is the last of the list means that it cannot be evolve anymore.

About the constructor of the Pokemon Class, its parameter can initialized the value of the Pokemon, but the level is default as random, but the attack or the hp can be input or set random value when passing the parameter also.

**Bag:**

Located in the file of MainWindow.xaml.cs, it represent the inventory of player to store pokemons.

Method:

Add: Add pokemon to player's bag in minigame (wild pokemon catching)

Sell: For selling the pokemon in home (management page)

Check: For debugging, check every pokemon in bag.

**ManageWindow:**

Manage Window is the most messy class in the program, since there are too many buttons to handle.

UI is basically 6 grid and each grid represents one pokemon. And each grid has an area to show the pokemon's image,  and buttons for actions. Right bottom corner are the exit button to back to the map and the left hand side corner is the money that players have.

The function of manage window is to manage pokemon, players can rename, power up, evolve and sell their pokemon. ManageSetUp adds all UI components into lists for processing more conveniently.  So foreach pokemon then one grid, and other grids are default hidden. updateStatus is a method to update the data of the pokemon and money. updateImage is to update the image of the pokemon (in case that the pokemon evolve, not very commonly use.) ShowUI is a method to turn the grid into visible. And EndClick is to end the page when button is clicked, Manage\_End is to end this window and show back the map.  And for every button and textbox will have a method when action is done by player, the action will execute.

Methods

ManageSetUp()  add UI component into lists for processing, show the money, check the radiobutton of the selected pokemon. And initialization

updateStatus:  To update status when action is committed

updateImage: To update image after evolving

showUI: To make the component of one pokemon grid visible

End\_Click: When button back to map clicked, end this window

Manage\_End: method to end this window and show the map back

////////////////////////////////X below represent the index of the component/////////////////////////////////////

There’s total 6 set of these method. Button are trigger by clicked, textbox of nickname trigger by typing, selectX is trigger by checking the radiobutton of it

NicknameX\_TextChanged:  When text changed change the pokemon's nickname

EvolveX\_Click: When evolve button clicked, run evolve

PowerupX\_Click: When powerup button clicked, run powerup

SellX\_Click: When sell button clicked, run sell

HealX\_Click: When sell button clicked, check heal

SelectX\_Checked: When radio button of select checked ,change the pokemon selected to this pokemon

**Class Reuse**

The class reused the most is the Pokemon class, each new pokemon generated whether it is enemy, wild pokemon or a pokemon that player owned are using the class of Pokemon, which is very convenient since we have 6 kinds pokemons, I think it’s easy to even adding more different kinds of pokemons by using the pokemon class, for other classes we have keep passing the data in different window and classes.

However, the Dictionary and SeriesDictionary have reuse a few times when doing the initialize the Pokemon. Since Dictionary is save one Pokemon and SeriesDictionary is saving the Pokemon which is the same series, then in the Pokemon Class is inherited SeriesDictionary, but added some value of the Pokemon, not only the Dictionary. Then we do not need to import the Series Dictionary again in Pokemon, so it will reduce of complexity and also reduce the memory that have used.

**Software patterns used**

We used a Model-View-Presenter(MVP) pattern. We use it in every Window the Model is used for storing data such as Pokemon name, hp, level, exp. The View is used to display the route events for users such as mouse click. For example, in the MiniGame Window, we put the wheel and the game engine in another class, when we need to use it both, we call it in the Window, and put into the window, when we need to check the event or do some event(like spinning the wheel), it will used the GameEngine class method. But when we need to project the wheel, we need use the Wheel class method. The Presenter the Widow part include the button and canvas.

Also, we used Composite in the Pokemon class we have use composite, we use it to inherited the SeriesDictionary to reduce the redundancy of the class. Some of the method will be save in the Pokemon class instead of SeriesDictionary, it is because only the Pokemon that in Player’s bag can do those action.

Lastly, we used Iterator, we use it in saving the the Pokemon, since we cannot limit the number of pokemon (in bag or allPokemon), so, we save the Pokemon as List, then when we want to show all pokemon or find out the pokemon, the Iterator will be used in these parts.

**Challenges overcome**

We have faced a lot of challenges when we are doing this project. Firstly, the work division, I think doing the group project is very difficult in doing work division, how to make both of us have the fair work and also we can do it separately but can merge together, so, we tried to do the first MainWindow part, and do the basic class, like bag, Pokemon, first, and do the remaining part separately.

Secondly, another challenge is that we class design, because of the work division, we need to do the class design better to that we can use it in different window and classes. The way we overcome is we design the whole game first before we really doing, then we implement it, to make sure that all the classes and method can use it properly.

Thirdly, the most important part is that we have to merge our work into one, I think it is the most challenging part, since everyone have different style to write the code, when we tried to merge the code we found that some of them have collision and we have to discuss the what is the meaning of the code when we want to change or use other’s code, the way that we solve this problem is to write the comment in the code, then we can read the code by the comment instead of asking directly.