IERG 3080 Software Engineering Project Design Report

1. Class Design and Reuse

1.1. High Level Description

In high level, the Pokemon Go game design is divided into 2 layers. In lower layer, there is classes for general usage and basic model (e.g. Pokemon, item). In higher layer, there are 4 sub-program which are designed to fulfill all the requirements in the game. Higher layer classes can make use of lower layer classes through various interfaces, and the whole designs generally follows the Model-View Presenter pattern. Therefore, we can easily divide the class into 3 categories (model, view, presenter) according to their role. In this report, we will explain the design of classes one by one according to their applications and roles.

1.2. Classes for General Usage

1.2.1. Model Classes

Movespace.cs

This code file describes the model of attack moves, which enable a Pokemon to make damage in a battle. The file consists of a base class Move and there is a class inherited from Move for every move. This is to prevent the construction of some move that are not pre-defined. Also, for every Pokemon using the same move, the base damage and accuracy should be fixed. Although in this simple prototype, only damage and accuracy are defined for those classes, it can support additional features like battle effect for future development. The class in this code file will be reused in all battle-related applications, and there will be a List of Move for every Pokemon, which indicate all moves that the Pokemon can use.

PokemonSpace.cs

This code file describes the model of Pokemon in the game. Similar to MoveSpace.cs, the file consists of a base abstract class Pokemon and there is a class inherited from Pokemon for every hierarchy of Pokemon. For example, we use the same class to model Pikachu and its evolved version Raichu. However, we use two seperate classes to model Pikachu and Ditto. We design the classes in this manner according to 2 main reasons. The first reason is that when we evolve a Pokemon, some fields are fixed, it can be inconvenient if we need to create a new instance of another class. The second reason is that it can prevent class explosion (note that there are already about 900 species of Pokemon now) and at the same time an instance of class can only represent one species of Pokemon (e.g. an instance of Pikachu class represents Pikachu when its stage equals to 0, and represent Raichu when its stage equals to 1), this can still ensure that the class design satisfies the single responsibility principle. Since Pokemon is the basics of the game, it will be reused frequently. Besides the class aforementioned, there is also an interface called IPokemonGenerator that are used to generate Pokemon. In PokemonSpace.cs, we give an example class StandardPokemonGenerator that implement the interface. This class will be reused in application that require the need to generate pokemon (e.g. Pokemon encountered in catching game, Pokemon of gym master in gym battle). For future development, there can be more generator that generate different type of pokemon in different places.

ItemSpace.cs

This code file describes the model of items in the Pokemon Go game. The Item in this game can be classified into 4 types (Heal, Revive Item, Ball, Boost Item) according to their nature.

Therefore, the file consists of a base abstract class Item. In addition, there is an intermediate class inherited from Item for every types of item. Finally, we create a class for each item inherited from the intermediate class of its corresponding type. Item is also another important element in a Pokemon game. In particular, we can choose different type of balls to catch Pokemon, use potion to heal Pokemon during gym battle, use potion and revive item to heal your Pokemon, and use boost Item to Power up or Evolve your Pokemon. The classes in this code file will be reused in those applications. Similar to PokemonSpace.cs, we also have an interface called ItemFactory that are used to Generate Item. In ItemSpace.cs, we give an example class StandardItemGenerator that implement the interface. This class will be reused for item generation in the navigation part. For future development, there can also be more generator that generate different type of item in different places, or we can generate some rewards for player who win in the gym battle.

PokemonWorld.cs

This code file consists of only 1 class called Player, which describes the model of a player in this game. The Player class consist of a list of Pokemon as well as Items, which indicate all the Pokemons and items that the Player own. The overall game state (e.g. naviagation, catching pokemon, etc.) are determined by a field called gameMode of the player. Since this Pokemon Go game prototype is designed for single person, we adopt singleton pattern for this Player Class. This class is reused when we need to change the game state, or in the case that we need to update the Pokemon and item list.

1.2.2. View Classes

GeneralView.cs

This code file consists of a number of interfaces and implementations that allow the presenters to modify the game view. Basically, an interface is designed for one purpose, which satisfies the interface segregation principle. First of all, there are IGameElement that support display and hide of game elements, this is implemented by a base class GameElement. Every implementation of view interface is inherited from this class, so that their visibility can be controlled easily. For text block, text box and button, there are IGameText interface that control the text displayed, this is implemented by the GameText class. For combo box, there are IComboBox interface that control the element displayed, this is implemented by the GameComboBox class. The combo box used in this game has 2 main types, one is to display the player's Pokemon, another is to display the player's Item. Therefore, 2 additional class GeneralPokemonBox and GeneralitemBox are inherited from the GameComboBox class. For list box, there are IPokemonListBox interface that allow us to add, remove and get element in the box. For image box, there are IImageBox interface that control its image source, this is implemented by the GameImageBox class. For progress bar, there are IHPBar interface that that control the display of Pokemon's HP Bar, this is implemented by the GameHPBar class. Those classes will be reused by the presenter of different sub-programs. If any additional view is required, we can simply implement those interfaces in other way and then reuse them.

1.2.3. Presenter Classes

MainWindow.xaml.cs

This code file consists of only 1 class called MainWindow, which are used to handle the interaction between player and all UI elements in the game.

PokemonWorld.cs

This code file consists of a base presenter class called GamePresenter, which enable a presenter to show or hide all of its game element. All presenter classes of the sub-program in this Pokemon go game are inherited from the GamePresenter class.

1.3. Classes for Navigation

Navigation.cs

The navigation sub-program allows player avatar (a pokeball in this game) to walk around a map using the direction button on the keyboard. Events like “Gym Battle” and “Get Item” will be triggered when player walk near some pre-defined place indicated by a text block. When event is triggered during navigation mode, all game elements in Navigation.cs will be hided and that of its corresponding sub-program (e.g. BattleGame.cs) will be shown.

1.3.1. Model Classes

First of all, to model a spot for specific purpose (e.g. Catch Pokemon, Gym Battle), we have created a class Spot so that the x-/y- coordinate of a spot can be stored and shuffled whenever we want. And then, to model a map for navigation, there is a class MapModel that contains all Spot required in the Pokemon Go game, including the player’s position, the place to catch Pokemon, the place to participate gym battle and the place to get item.

1.3.2. View Classes

In navigation game, we use a canvas to display map view. Therefore, a class called MapCanvas is created so that we can change the position of player’s avatar and the text blocks representing different spot according to player actions. For other game view control, we can simply reuse the classes in GeneralView.cs .

1.3.3. Presenter Classes

For the presenter, we have a class called MapPresenter that observe player input (whether he/she pressed a direction button), detect the distance between player’s position and different game spot, as well as update the corresponding game spot after the user has triggered a game event.

1.4. Classes for Capture Pokemon

CatchGame.cs

The capture Pokemon sub-program allow player to encounter and catch Pokemon through a wheel of fortune game, which is reused from that in project part I.

1.4.1. Model Classes

To model the catching game, a class called CatchModel is designed, it can be accessed by the presenter through an interface called ICatchModel. The class is used to store the game state, the wheel state, the wheel angle, as well as the catch rate (probability that can catch the Pokemon). Also, the game state is evaluated in this class given the wheel angle and catch rate.

1.4.2. View Classes

To display a wheel of fortune, a class called WOFWheel is designed, it can be accessed by the presenter through an interface called IWOFWheel. There is another class called Category that store the percentage, title and color of different segment of wheel. To make the gameplay more convenient, a class called WOFItemBox is inherited from the GameComboBox class, so that only balls owned by the player are displayed in the item box. For other game view control, we can simply reuse the classes in GeneralView.cs.

1.4.3. Presenter Classes

For the presenter, we have a class called CatchPresenter that roll and stop the wheel, as well as changing the wheel when user change its ball used. For example, when user use an Ultra Ball instead of a PokeBall, the catch rate will increase, and the percentage of purple sector (catch success) will increase too.

1.5. Classes for Gym Battle

BattleGame.cs

The battle game sub-program allows player to participate in a gym battle with no more than 3 Pokemon. The level of gym master's Pokemon is fixed to be 7 (slightly higher than default level), and the species of Pokemon are generated randomly. For future development, the Pokemon and the level can be fixed or adjustable, and there can be gym master that own only Pokemon with a single type.

1.5.1. Model Classes

First of all, a class called BattleTurn is designed to model a single battle turn. A battle turn consists of 2 Pokemon and their operation (e.g. attack with a move/ heal) in that turn. It is always the first priority to execute a heal operation, and for attack operation, the Pokemon with higher speed will be done first. If the Pokemon HP is 0 before it attacks, the attack operation will be cancelled. The damage calculation generally follows the one in the official Pokemon game. Furthermore, a class called BattleModel is designed to model the gym battle, which can be accessed by the presenter through an interface called IBattleModel. A gym battle is simply a series of battle turns, the model will keep accept user input (pressing button to choose the attack/ heal operation) until all of player's Pokemon in the gym battle fainted (HP = 0), or all of gym master's Pokemon fainted. Gym master's action are randomized. However, for future development, it could be deterministic under some logic (e.g. choosing the move that makes the highest damage).

1.5.2. View Classes

A class called BattlePokemonBox is inherited from the GameComboBox class, in which only the Pokemon with HP > 0 are displayed in the Pokemon box. Another class called BattleItemBox is inherited from the GameComboBox class, in which only Heal item is displayed in the item box (that means we don't allow to revive a Pokemon and catch gym master's Pokemon during gym battle). Also, a class called SelectedItemBox that implements the IPokemonListBox interface is designed to show the Pokemon selected by the player to participate in a gym battle. For other game view control, we can simply reuse the classes in GeneralView.cs.

1.5.3. Presenter Classes

For the presenter, we have a class called BattlePresenter that control the game view among different game state (Pokemon selection/ gym battle), as well as updating the game model corresponds to player's action.

1.6. Classes for Manage Pokemon

ManagePokemon.cs

The manage Pokemon sub-program allow player to power up, evolve, name, heal and release his Pokemon.
To power up a Pokemon, the player needs to select a rare candy and press the "power up" button.
To evolve a Pokemon, the player needs to select an evolve stone and press the "evolve" button.

1.6.1. Model Classes

The model of this sub-program simply reuses the Player class in PokemonWorld.cs.

1.6.2. View Classes

The game view control of this sub-program simply reuses the classes in GeneralView.cs.

1.6.3. Presenter Classes

For the presenter, we have a class called ManagePresenter that update the game view according to player's selection of Pokemon, and update the model corresponds to player's action on his Pokemon.

2. Software Pattern Used

2.1. Software Architectural Pattern

Model-View-Presenter: used for the design of whole program.

2.2. Creational Patterns

Factory Method: IPokemonGenerator and ItemFactory interface and the classes implementing them.

Singleton: Player class since it is a single player game.

2.3. Design Patterns on Abstract Concepts

Observer: Event Handlers in MainWindow and ManagePresenter class.

Strategy: IPokemonGenerator and ItemFactory interface allow us to generate Pokemon/ item according to different strategy.

2.4. Structural Design Patterns

Adaptor: all the presenter classes in the game, connecting the model and view classes.

2.5. S.O.L.I.D Principle

Single Responsibility Principle: every class in this game can only has single purpose, as mentioned above.

Open/Close Principle: protected field in most classes allow extension and prevent modification.

Liskov Substitution: Inheritance is guided by Polymorphism. For example, from Item to Heal to Potion.

Interface Segregation Principle: interface is designed for only one purpose, but not general purpose.

Dependency Inversion Principle: presenter classes access the view classes through various interfaces.

3. Challenges Overcome

In this project, the biggest challenge that we have overcome is how can we get start on such a large project. According to the specification, it seems like we have to do a lot of thing in order to complete the project. To overcome this challenge, we first divide the whole game into small components, so that we can code it one by one, which is much easier than to coding the game as a whole.

4. Job Division

Project Component	**Responsible Person**
Model Classes (General Usage)	KEI Yat-long
View Classes (General Usage)	Chan Sum Yuet
Presenter Classes (General Usage)	KEI Yat-long
Model Classes (Navigation)	KEI Yat-long
View Classes (Navigation)	Chan Sum Yuet
Presenter Classes (Navigation)	KEI Yat-long
Model Classes (Capture Pokemon)	KEI Yat-long
View Classes (Capture Pokemon)	Chan Sum Yuet
Presenter Classes (Capture Pokemon)	Chan Sum Yuet
Model Classes (Gym Battle)	KEI Yat-long
View Classes (Gym Battle)	Chan Sum Yuet
Presenter Classes (Gym Battle)	Chan Sum Yuet
Presenter Classes (Manage Pokemon)	KEI Yat-long