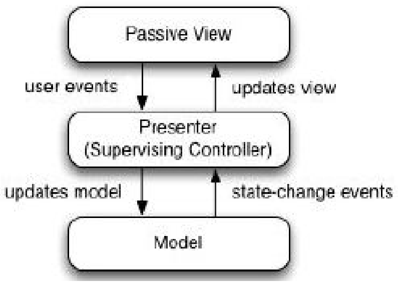
Tang Siu Cheong 1155127379
Fong Hok Yin 1155108613

IERG3080
Project Part II

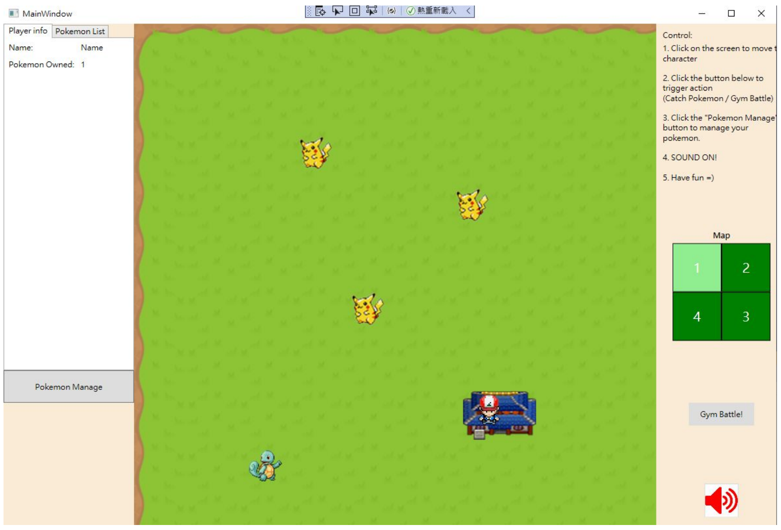
Pokemon GO!

Part I.
Brief of the Game Design Pattern
For the design pattern that we are trying to apply on the whole game development is the Model View Presenter (MVP) pattern. The reason why we have applied this pattern is that this pattern has achieved the best result of separating the Interface and the Logic behind the game. Compared to the Model-View-Controller (MVC) pattern, the MVP pattern can make sure the classes don't have to handle the interface. We want to have the classes behind handle data only while the presenters make use of the data to output meaningful information on the interface for the user.



Apart from the MVP pattern, our team has tried to follow the S.O.L.I.D principles, especially the Single Responsibility Principle, the Open/Close Principle and the Dependency Inversion principle. Even though this is a very first time for both of us to cooperate in writing a game with OOP languages and we may not have followed the principles perfectly, we have seriously discussed the importance of keeping the code simple and open for future extensions before starting the development.

For the interface part of the game, we made use of the Composite pattern to group up interfaces that are related to each other. From the MainWindow, we have combined the Player status, Pokemon list, Tutorial, Map, Buttons, and the game canvas together on one window. Interfaces with major functions, like Gym Battle, Catch Pokemon and Pokemon Manage, are separated from the MainWindow. This provided a good classification of different parts of the game and also a good basis for our group to divide our work, which we will mention about it in the below.



Control:
1. Click on the screen to move t
character

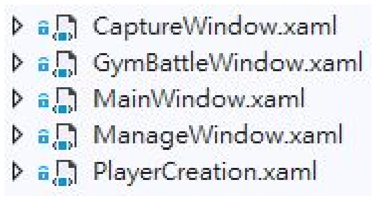
2. Click the button below to
trigger action
(Catch Pokemon / Gym Battle)

3. Click the "Pokemon Manage"
button to manage your
pokemon.

4. SOUND ON!

5. Have fun =)

Fig. MainWindow



CaptureWindow.xaml
GymBattleWindow.xaml
MainWindow.xaml
ManageWindow.xaml
PlayerCreation.xaml

Fig. Different Windows

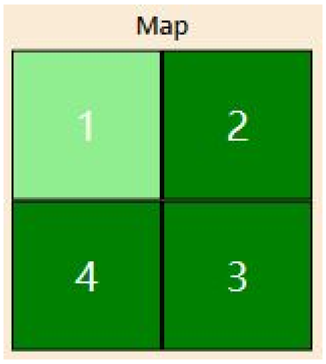
Part II.
Division of Labour (Alex + Albert ^^)

Alex:
- Player Creation
- Main Window
- World + Map Generation
- Grid + Location class
- BGM + sfx Player class
- Game Session class + Base Notification class
- Images + Animation
- BGM + sfx

Albert:
- Pokemon class + different pokemon type
- Pokemon Creation + Enemy / Pokemon Factory
- Pokemon Skills
- Gym Battle Window
- Gym Battle System
- Pokemon Capture Window
- Pokemon Manage Window

Part III.
Class Design and Reuse

Acknowledged the basic pillar of OOP, which is the Encapsulation, Inheritance, Polymorphism and Abstraction, we have made use of different creational patterns and structural design patterns in the class and program design, which we will mention in the below.



World, Map, World Factory and Grid

The world class is a static class that contains the World content of the game, which is the content of the map. As to the Map class, it is a class that is used for navigating the direction of the map. The

world factory class is a static class that stores factory methods for generating the world and map.
In the factory method, Grid class is used to store the content of the Map. In the game, it consists of 4 parts of the Map. Every part of the map is made up of 10 x 10 grids. The presenter will retrieve details of the grid, including Pokemon / GymBattle, and add them to the canvas.

<u>GameSession and Location</u>
This class played a crucial role in the whole game. It is responsible for recording the data throughout the game. The data in the class is directly linked to the interface as DataContext. The data in the class is presented to the interface with the use of WPF. On top of that, the Game Session class is also responsible for recording the World content and the player's location. There is only one and the only instance of this class in the whole game, as only one recorder is needed. Different windows use this class to share the data in the game.
As to the Location class, it is a class that used to record the current location of the player, which also records if that grid has any GymBattle or Pokemon available. The presenter can directly access the Location class to get to know the content of the current Grid, without accessing the 3D Grid array.

<u>BGMPlayer</u>
It is a static class that consists of two static MediaPlayer, which are the bgmPlayer and the sound effect player. Just like the name of the player, the bgm player is responsible for the background music player and the sound effect player is responsible for the sound effect, like attack sound or game lost sound. In this class, I have realized that the delegate variable is actually very useful for storing methods for later access. (Sorry for not realizing this fact from the lab haha =PP) The delegate variable is used to store my current playlist of music, and access the stored playlist through the delegate variable again after the music ended. The BGMPlayer class is being used in different windows, including the Main Window, Gym Battle and Catch Pokemon. The presenter is in charge of handling the music player.

<u>PlayerModel</u>
It is a class that is used to store the player's data, which is the name and collected pokemon. In this class, I tried to keep the class as neat as possible, since I expect there will be extensions in the future.
In the class, you may notice that the CollectPokemon uses ObservableCollection instead of Dictionary or List. It is because I found that only the ObservableCollection will allow the update to the MainWindow of the variable's changes, like Pokemon add or abandon.

<u>BaseNotificationClass</u>
It is a class that inherits from the INotifyPropertyChanged class. This class acted as a base class for many classes, like the Player class, Pokemon class and so. You may notice that most of the variables in the derived class of the BaseNotificationClass may call the method of OnPropertyChanged() in the setter (right after the variable value is being set). The reason why is that this class serves as a purpose of noticing the change of property and the presenter can therefore make real-time changes to the binded value on the interface.

PokemonModel

It is an abstract class which indicates the framework of all Pokemon appeared in our game. It defines all parameters and methods that a Pokemon should have.
In detail, there are 5 status-related parameters with read-only properties such as Level, HP, Attack, Defense, Speed. There are also a set of parameters and methods related to evolution and level up of Pokemon read-only parameters. For example, there is a parameter called EXP and a method called GainEXP. When the pokemon has gained enough EXP, there will be a method abstract LevelUpEvent to handle the growth of those status-related parameters. Moreover, a parameter NickName and ChangeName method are also defined to handle the naming of Pokemon.
In order to increase the reusability of the class and provide more room for extensions to the class user, the PokemonModel class has defined the parameters only. There is no assignment carried out in PokemonModel class.

Bulbasaur, Charmander, Pidgey, Pikachu, Squirtle

Having high reusability, PokemonModel has been reused frequently by inheriting to be a wide variety of sub-classes. In those sub-classes, the non-assigned parameters defined in PokemonModel will be assigned well based on the characteristics of different Pokemon.

For example, in one of the sub-classes Pikachu, the basic parameters like HP, Attack, Defense, Speed will be assigned as 30, 10, 10, 10 respectively. The abstract LevelUpEvent will have a corresponding override too. After the overriding, Pikachu class will have its own unique status parameters growth and the evolution of Pikachu class will also be defined. On the other hand, the normal attack of Pikachu will be assigned as Thunderbolt while the special attack will be assigned as Thunder. Finally, the parameters related to the evolution of Pikachu will also be assigned.

PokemonFactory

It is a static class implemented to create new Pokemon instances when the game is started or the gym battle is started. Due to the characteristics of the Factory Method pattern, the logic behind the creation of Pokemon instances will not be exposed to the class users. Besides, the class user cannot modify the creation of Pokemon instances which can decrease the risk of having errors in creating Pokemon instances.

Skill

It is a class indicating the attack method of a Pokemon. In this class, two parameters and a LearnSkill method are defined. One of the parameters is the name of the skill while another one is the power of the skill. For the LearnSkill method, an integer will be passed to the method and the Pokemon will learn the corresponding skill. For example, if LearnSkill(1) is called, the skill Thunderbolt with the power of 10 will be assigned to the Pokemon.

Part IV.
Challenges overcome

One of the biggest challenges that we faced in the implementation process is that we are not familiar with WPF. Therefore, when we have finished the implementation of the window that we are responsible to implement, we find that it is hard to combine them into a single product because we implement our own window separately. In order to overcome this challenge, we came up with creating a class called GameSession which stores almost all data and progress of our game. Whenever we want to access some data related to the player or the collected Pokemon of the player, we can access through using the GameSession. As a result, we can ensure that the data between windows and windows are synchronized.

Another huge challenge is teamwork. I still deeply remember Professor Hui has once said that “Every programmer thinks their program is the best” and so do I. Initially, I do have such thoughts and felt a bit struggling to make use of my teammate’s code. Soon I realized that there is never a so-called “perfect” program and you can always learn something from other programs. Blindly editing all the code to “fit” your flavour is greatly opposing the development progress and not helping the project. Let go of such a mindset is the first step to teamwork and improvement.

Last but not least, the animation part is also a huge challenge for us. In the past few years of study in the programming courses, we have never learnt anything about the animation of objects, not to mention that it is our very first-time experience of using OOP. It took us quite a long time to study the movement of the player object and the animation of the pokemon in gym battles. Huge thanks to online resources and tutorials, I am glad that the animation now looks fine.

Part V.
Reference

Didn’t use this way of movement eventually.
https://www.mooict.com/wpf-c-tutorial-move-an-object-with-keyboard-and-timer-in-visual-studio/

This website inspired us a lot with the very detailed tutorial! Strongly recommend this to the others! I am planning to tip this “Online teacher” a coffee! =P
https://scottlilly.com/build-a-cwpf-rpg/

Inspired the game’s way of movement
https://stackoverflow.com/questions/46380682/wpf-move-element-with-mouse

Very useful assets!
https://kenney.nl/assets?t=rpg

Life savior! Many pokemon images
https://pokemondb.net/sprites