Contents

[General Game Design 1](#_Toc60262627)

[Class Diagram 1](#_Toc60262628)

[Software Patterns used in this project 2](#_Toc60262629)

[Special Highlights 3](#_Toc60262630)

[Special design in special situation 4](#_Toc60262631)

[Special debug and testing tools 4](#_Toc60262632)

[Division of Tasks 5](#_Toc60262633)

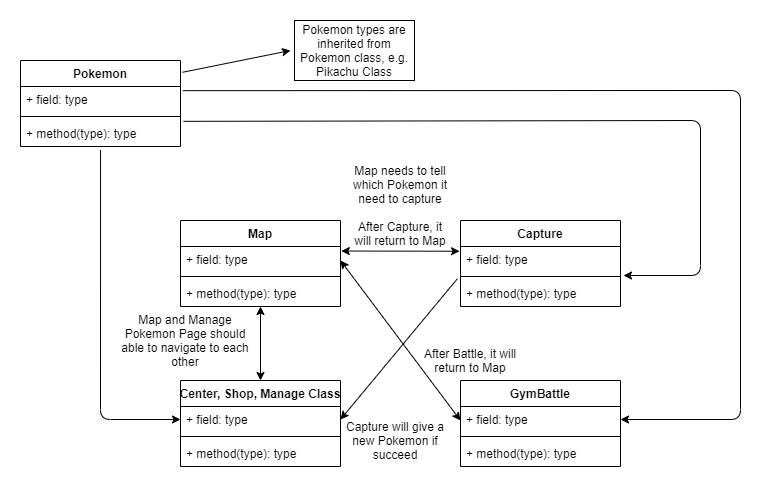
## General Game Design

When the player first enters the game, they will see a starting page. Pressing a left mouse key will enter the map. In this map, players can use the keyboard direction key to navigate. When near an object (houses and counters on map) that can interact with, the screen will display an enter key icon to notify player can use the keyboard enter key to interact.

On the map, the player can do various tasks:

* Pokémon centre: This is a place for the player to see their Pokémon collection and to select the favourite Pokémon for a gym battle
* Shop: This is for the player to buy powerup to level up their Pokémon
* Capture: Players can play a simple mini-game to capture the Pokémon. If they succeed, the Pokémon will add to their collection
* Gym Battle: This is a game mode where the player can choose a Pokémon in their collection to battle with a random Pokémon. If the player wins, they will get some coins as a reward.

## Class Diagram



## Software Patterns used in this project

In the following, it will highlight considerations when we design the game in terms of reusability and design patterns.

There are few software patterns are used frequently inside this project, including:

* Model-View Separation
  + In GymBattle.xaml.cs, you may find two classes namely GymBattle (View) and Battle (Model). In the Shop.xaml.cs, Manage.xaml.cs and other parts, we apply Model-view frequently so that even if the GUI changed, the model may not need to change.
* Singleton
  + Multiple locations used the singleton method, for example, in MainWindow.xaml.cs ManagePokemon and map\_info class, we used singleton to perform some effects similar to the global variable that can pass between pages.
* Factory Method
  + We created methods like RandomPokemon in Pokémon class and Pokémon Evolve() in Eevee.cs, they are factory methods that allow us to hide some details to randomly pick a Pokémon. For example, the user doesn't know which kinds of Eevee they can get from the evolution, this may include picking different Pokémon that may have different probabilities. This simplifies a lot when other places (e.g. capture) need to select a Pokémon often.
* Strategy
  + In GymBattle.xaml.cs, when the Pokémon attacks, it uses a delegate to select which attacks the player chosen. This simplifies a lot and makes the code more readable.
* Template Method
  + In Pokémon class, I have used a lot of abstract keyword in methods, the reason behind is to avoid future programmer that wants to inherit the parent class but forget to implement some of the important methods. As using abstract keyword, the compiler forces all the subclass need to include the same method signature, this ensures the program can run properly.
* Façade
  + In Capture and GymBattle game mode, I only expose the constructor to allow other programmers to enter the game. In this way, I hide all the detail about the GUI and game logic, to have high reusability and portability. Therefore, if another programmer wants to use it, it does not require much effort to understand and use it.
* Observer
  + In the GymBattle, I attempt to use an observer pattern to observe some variables (e.g. finished, isWin). In this way, I can separate model and view more. Although I know in real life won’t check the variable in every 1ms as it consumes a lot of system resources, I don’t know how to implement it in this project. However, I keep in mind I am trying to do an observer pattern when I create this part.

Also, when designing the solution, we keep SOLID principles in mind.

* Single Responsibility Principle
  + In our project, each class and method are named according to their usage and we tried our best to keep one method/classes are only doing one thing (e.g. create animation, check winning condition, etc.)
* Open/Close Principle
  + Although we made the best effort to make it is extendable while closed for modification, I found it is very difficult to achieve. However, we made some codes that are reaching this principle.
  + For example, in the Capture class, it only needs to pass a Pokémon object to initialize the game. This allows future updates that Pokémon capture can start in a different location on map/Pokémon without modifying the current code.
* Liskov Substitution
  + In the solution, we are use inheritance carefully as it can create a complex relationship between classes if overuse. Therefore, in our project, only Pokémon variations (e.g. Pikachu) are inherited from Pokémon classes as it needs some properties from its parent class.
  + Also, when writing methods, the parameter can set to “Pokémon” data type using inheritance and polymorphism. if not, each Pokémon type will need a method signature and it will become too complex to make.
* Interface Segregation Principle
  + Inside the classes, if the method does not necessary to expose to other classes, it will set it to private methods as it prevents some other teammate from misuse and creates unnecessary bugs.
* Dependency Inversion Principle
  + Inside the game, we tried our best to create as few dependencies between classes as possible. Even if it has dependencies between them, it will still try to make it one-way communication for easier to debug and expand in the future

## Special Highlights

In the project, it has a class called CustomTimer. The code inside is modified from Project Hint, but this class allows the timer to reuse and I created 4 overloaded constructors to ensure it can be used in most situations.

* public CustomTimer(int interval, Func<bool> procedure)
* public CustomTimer(int interval, Action End)
* public CustomTimer(int interval, Func<bool> procedure, Action End)
* public CustomTimer(int interval, Action Initialize, Func<bool> procedure, Action End)

In this code, it allows passing a function into a parameter (initialize, procedure and End are functions). Below is the skeleton of the class, you can see how this class reuse this timer.

private void StartTimer\_Click(int interval)

{

Initialize();

}

private void timer\_Tick(object sender, EventArgs e)

{

procedure() //If true, it will call Stop()

}

public void Stop()

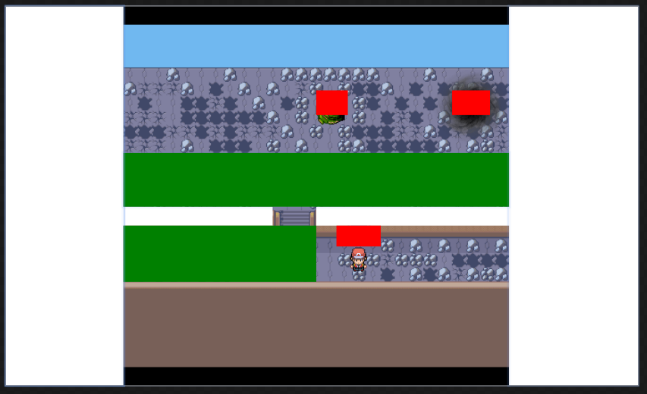
{

End();

}

## Special design in special situation

In all maps, (Including Map, Wild, tower, Top, Market, Center), we use a special custom design,



(The look of the map when invisible blocks are not hidden) (We use colour to distinguish what type of object they are)

(Solid blocks – white, portal to other pages - red, region that will meet pokemon – green)

No matter how the map objects are designed for each map, we just need add all these xaml items into a list of FrameworkElement, then we use dispatcherTimer to check coillision and interception from time to time, finally stop user from walking through this solid block objects. So, we can reuse the codes with only small changes for each map, even the code is reusable but we didnt make it into a reusable class, as the RoutedEventArgs, FrameworkElement and some xaml components are difficult to pass between pages and classes, so we decide to reuse the code safely for each map by just copying the code, in order to avoid unknown xaml bugs. But we still maintain the model-view separation and we can still reuse the code very easily for each map.

Inside these codes, we have Canvas\_KeyDown and KeyUp to check user keyboard input, and check\_intersect() to check if character has interception with other objects, top, bot, left, right\_collision() to check which sides of the object the player collides with, and finally we can stop user from walking through that side of the solid blocks. With this complex logic code, finally we can simulate solid block collision in the WPF.

## Special debug and testing tools



I make these special tools for teaching the user how to play the game and make some convenient tools to let me and my teammate can get access to the main game function directly for testing and debugging. Save development time.

## Division of Tasks

|  |  |
| --- | --- |
| Parts that are done by Po Wai Yin (Jason)  Job: Model part and some view part, making reusable base class, design the Pokémon class | GymBattle (Both xaml and c# code)  Capture (Both xaml and c# code)  Pokemon.cs  CustomTimer.cs  Munchlax.cs  Pikachu.cs  Rayquaza.cs  Charmeleon.cs  Mainwindow (ManagePokemon class) |
| Parts that are done by To Yi Him (Wilson)  Job: View part and some model part, try to combine the use of the class and connect the view and model part | StartPage (Both xaml and c# code)  All maps (Include Map, Wild, tower, Top, Market, Centre) (Both xaml and c# code)  Shop (For upgrade Pokémon)  (Both xaml and c# code)  Manage (For manage Pokémon collection)  (Both xaml and c# code)  Mainwindow (map\_info class)  Eevee.cs and some other Pokémon classes  Testing and Tutorial mode |
| Parts that both of us do together | GetPicture.cs |