

Project Summary

1 GAME OVERVIEW

1.1 Background and Story

This project is to design an RPG (Role-playing game) by utilising Java and Javafx as main developing languages. The name of the game is **Pokémon Battle**. As with traditional RPGs, players have the freedom to control and play as different Pokémon characters with different skills to explore the maze, as well as battling with enemies in different levels. We designed an extensive skill pool, complex battle system, and strict upgrading system to help players get better experience.

The story of this game is about a tribe of Pokémon living on a Far East Island who use their own power to fight against foreign invaders. In this game, the player will perform one of the Pokémon, and different Pokémon will have different combat skills and initial capabilities. Each maze map is one of the four zones on the island. A lot of enemies lurk in various areas of the island and vary in level and combat power. There are also a number of different treasures to help the player improve combat skills and get through the island. Players will need to assess their own strengths and the skills of the corresponding enemies and allocate the use of props and routes of action accordingly.

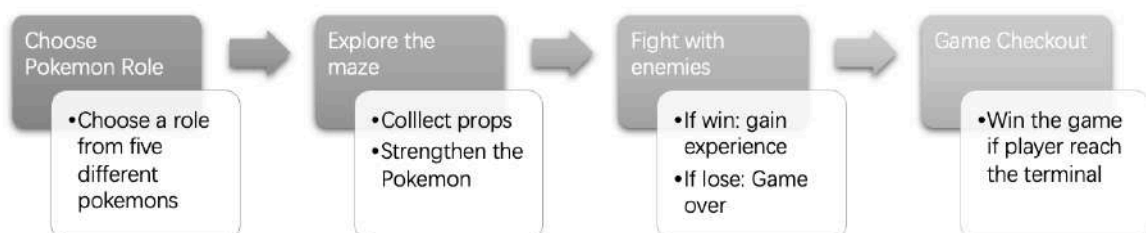
1.2 Character Design

There are five different Pokémon in the game, each with different basic information. The basic information includes level, skills, Health point, Mana point (stands for the energy to use a magic skill), defence and attack abilities, experience, and the abilities to cross grass/water/stones. Players can choose a Pokémon to continue at the start of the game.



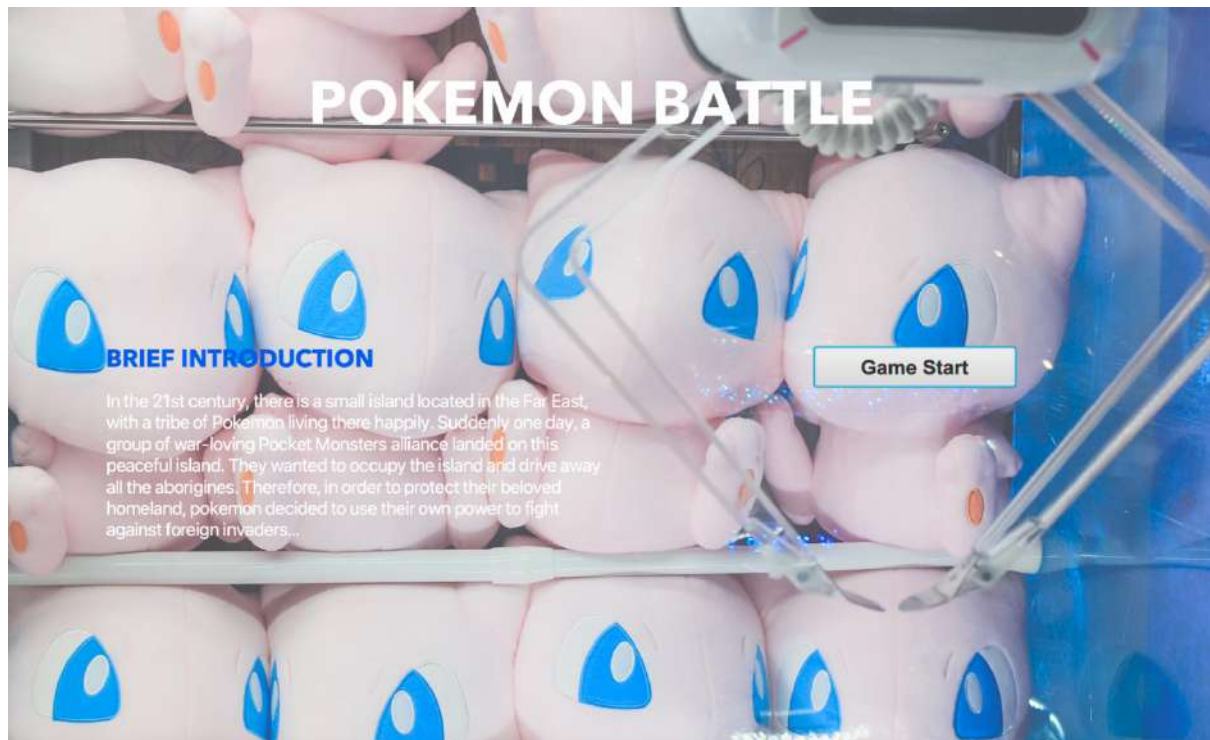
1.3 Event Flow

The game progress is demonstrated in the following flow:

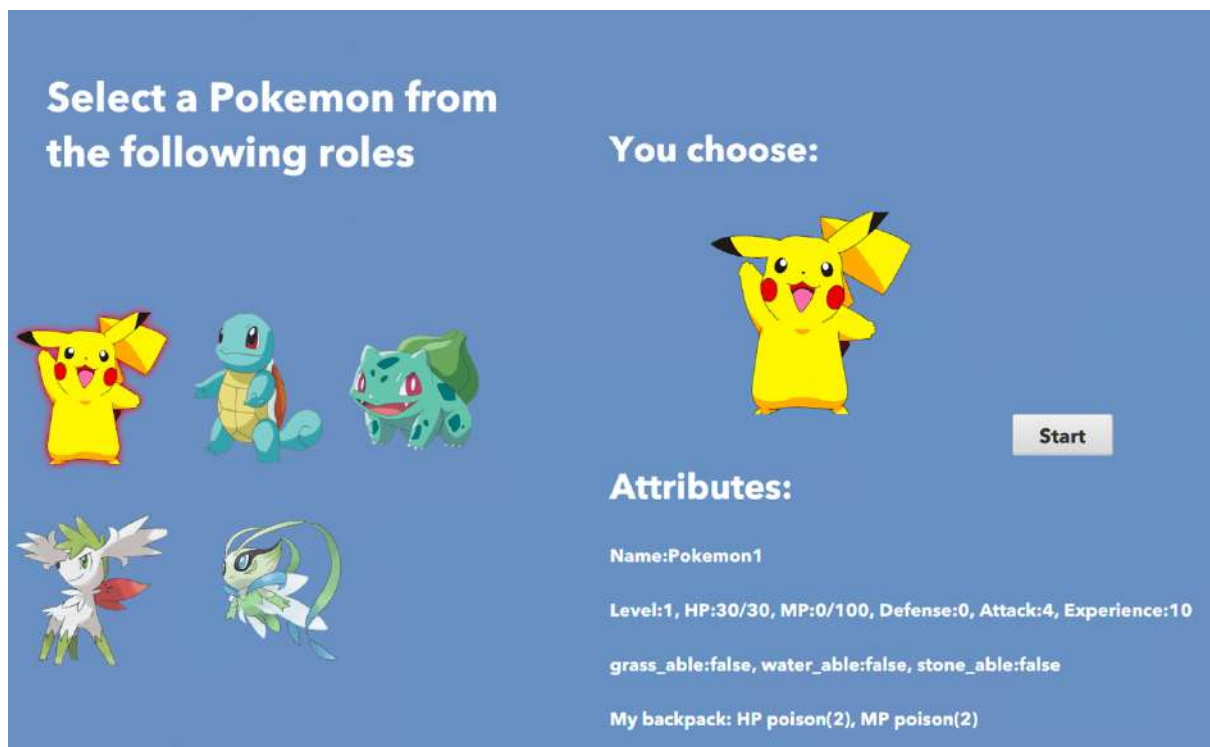


Screenshots of game progress

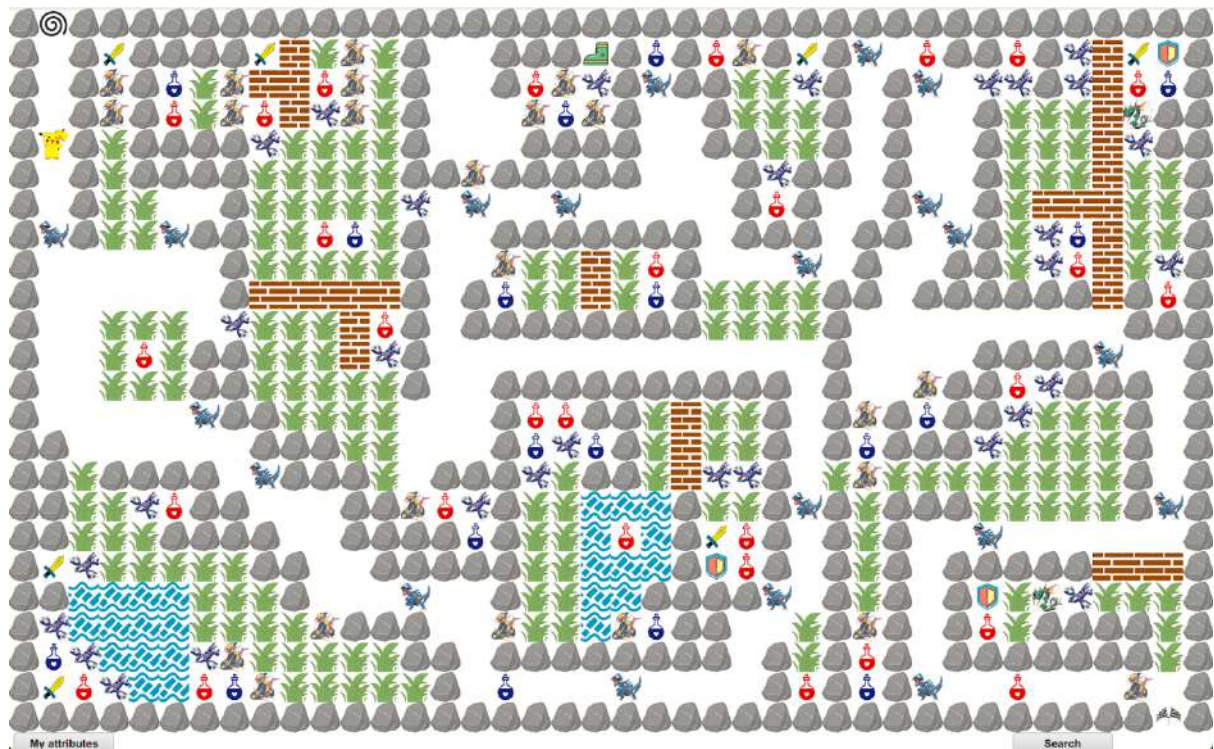
The starting page:



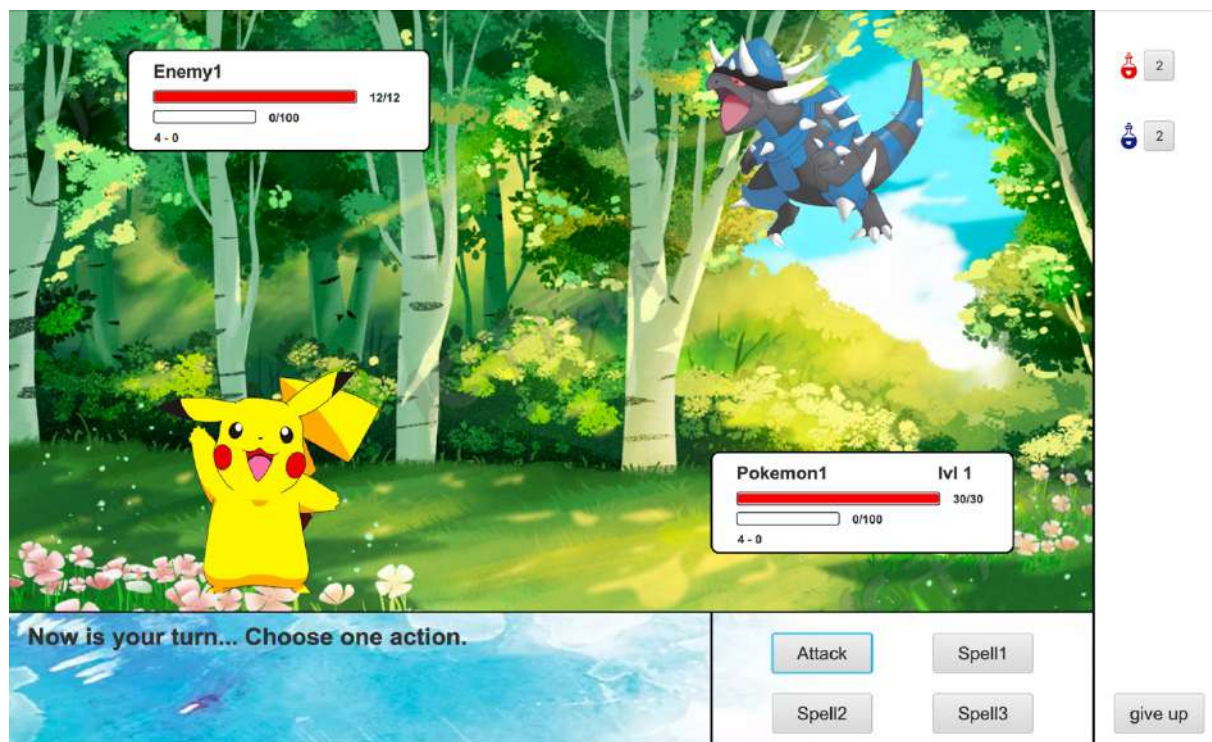
Select a role to continue:



Once selected role, the player needs to walk through the maze:



The battle page when the Pokémon encountered an enemy:



Win / Lose:

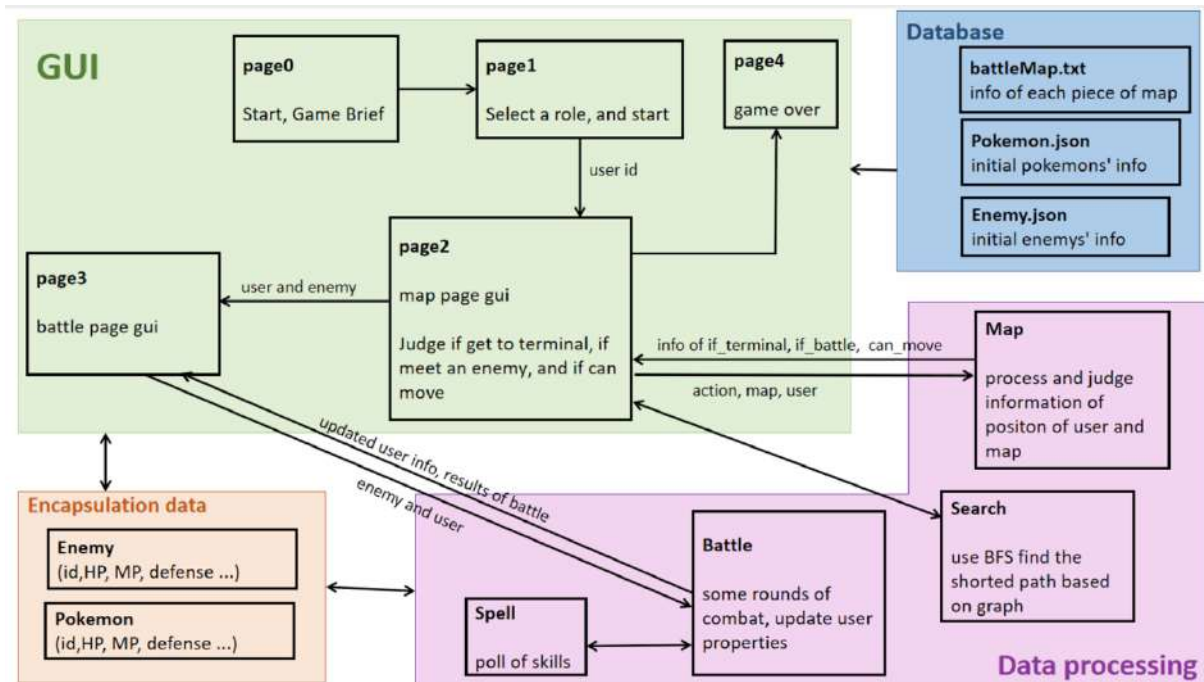
Congratulations!! You win!

Game over! You lost!

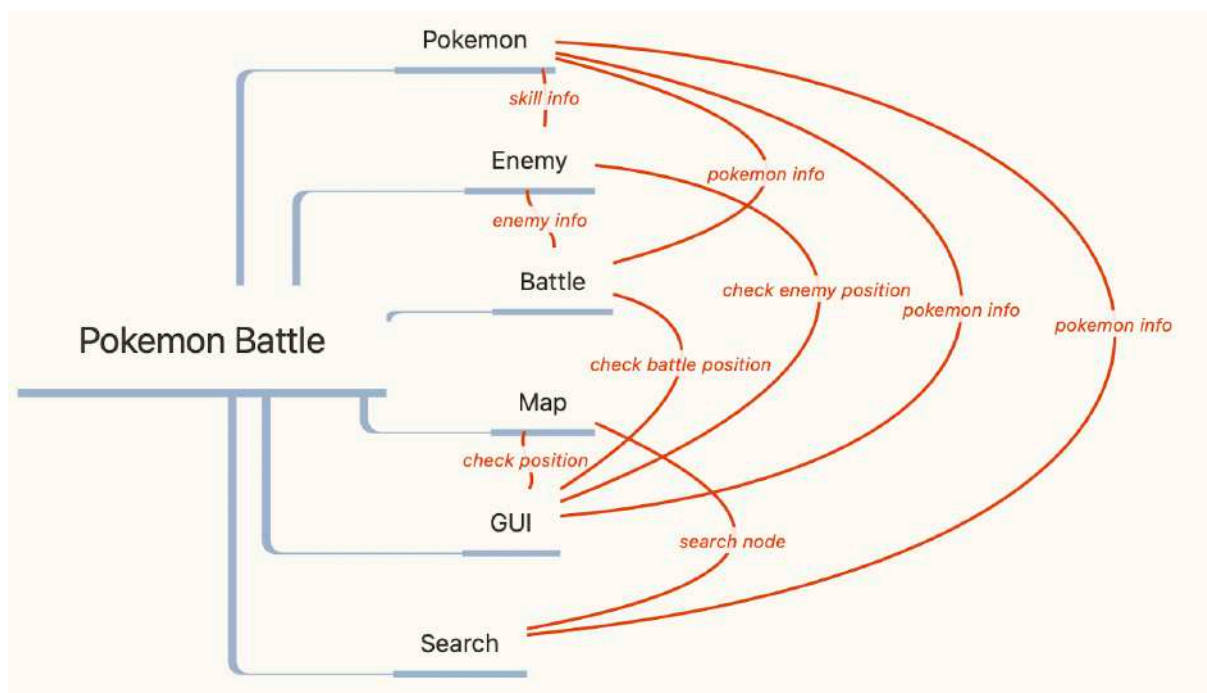
2 GAME DESIGN

2.1 Architecture

The GUI skeleton of the project is as below:



The back-end skeleton of the project is as below:



2.2 Introduction to Functions and Menus

The summary of each class's main function is as below:

Class Name	Main Function
Pokémon	Use Json file to record each Pokémon's initial basic information. Methods to record changes of base information, and the players position in the map during the whole game.
Enemy	Use Json file to record each enemy's initial basic information. Methods to record changes of base information during each battle.
Battle	Use AtomicBoolean and AI (the method helps selecting the best way to win), as well as Javafx to update the changes of basic information both for Pokémon and the enemy and update the front desk animations at the same time. Record the result of battle (win or gameover) and Pokémon's level upgrading results.
Map	Use Bespoke and text file to load map information. Methods to decide map situations, including enemies in different types, props, obstacles, starting and ending of the map, map switching, etc.
GUI	The main front-end display by using Javafx, including six screens from beginning to the end of the game.
Search	Use BFS (breadth first search) algorithm to find the shortest path to a nearby accessible target.

2.3 Game Operations

2.3.1 Map System

Movement & Obstacle Determination & Props

Players can control the Pokémon to move by pressing arrow keys on the keyboard. *ArrowLeft* stands for moving left, and *ArrowRight* stands for moving right, *ArrowUp* stands for moving up, *ArrowDown* stands for moving down.

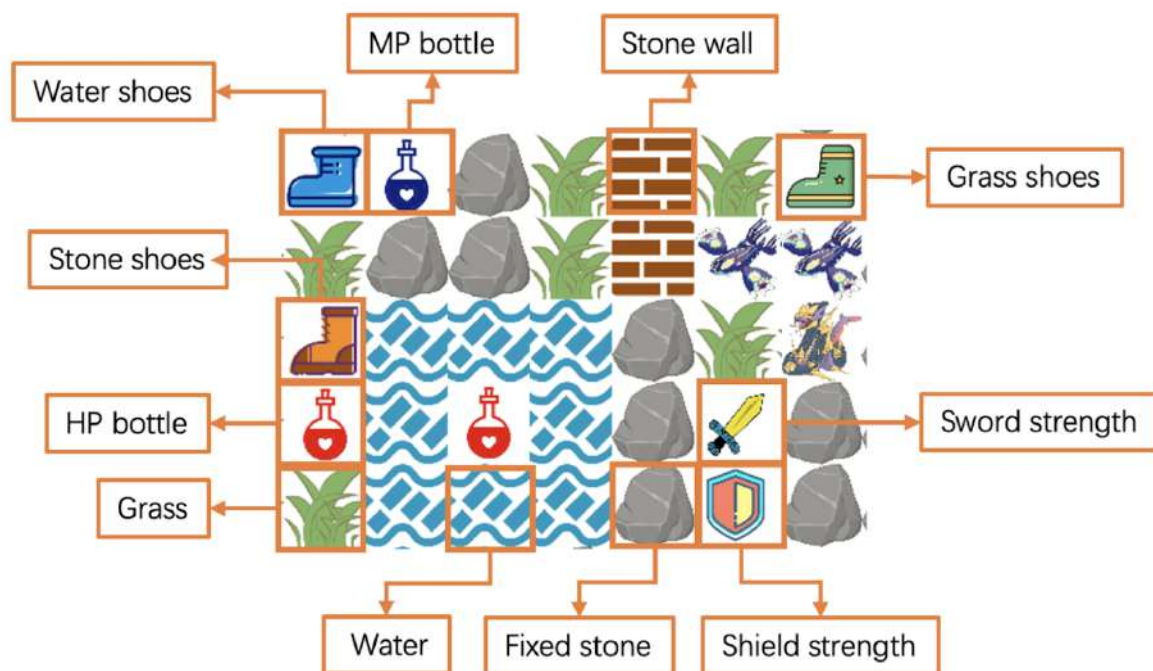
There is 1 obstacle, *Fixed stone*, on the map that can never be crossed, and 3 obstacles, *Grass*, *Water*, *Stone wall*, that can only be crossed if the conditions are met. If players move the Pokémon to the position of those obstacles, the system will refuse to finish the movement.

There are 3 props, *Grass shoes*, *Water shoes*, *Wall shoes*, helping players to gain extra skills to cross the *Grass*, *Water*, *Stone wall* respectively. If players move to the position of those props, the corresponding abilities will become available.

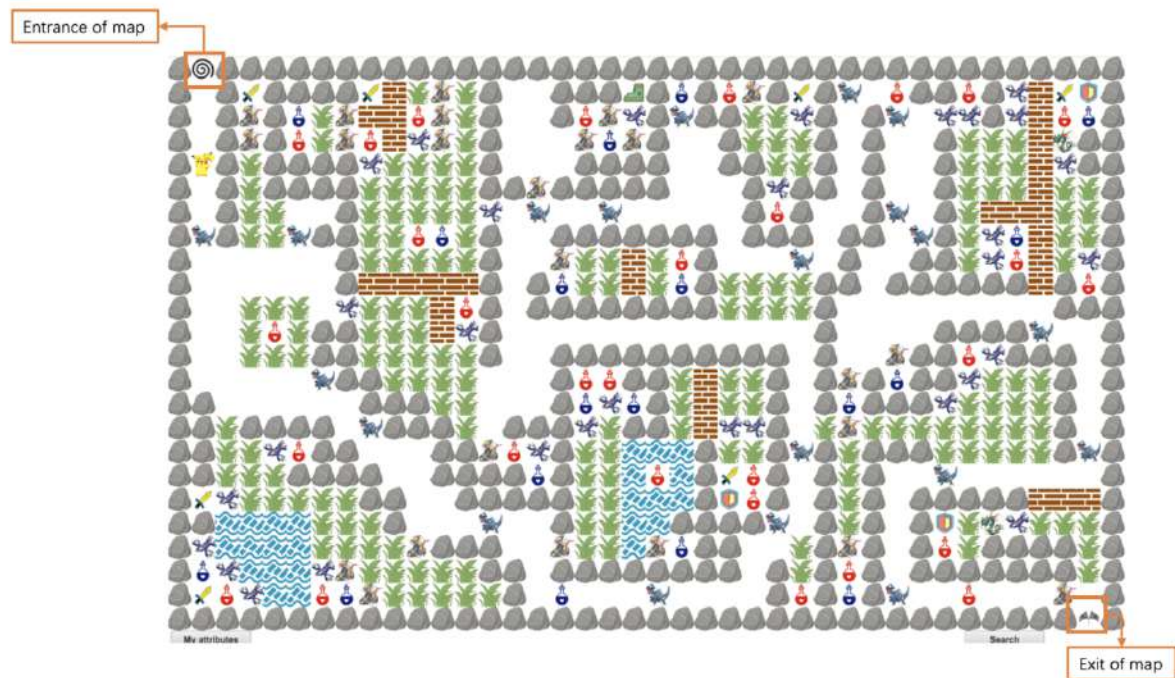
There are 4 props, *HP bottle*, *MP bottle*, *Sword strengthen*, *Shield strengthen*, helping players to upgrade health points, mana points, the number of attacks, and the number of defence respectively. If players move to the position of *HP bottle* or *MP bottle*, the bottles will be added to players' backpacks. If players move to the position of *Sword strengthen* or *Shield strengthen*, the attack or defence ability will be increased immediately.

Map Switching

There are totally 4 different maps in this game. Each map has enemies in different levels and props in different usage. In each map, players will always start at the entrance or the exit of the map. If players reach the end of the map, they can continue to the next map if they move the Pokémon to the correct direction, e.g., if the player stands in the exit of the following map and continue moving downward, then the interface will switch to the next map:



However, if the current map is already the fourth map (the last one), then if the player reaches the end of the fourth map, the game will finish, and the interface will switch to the checkout screen of the game.



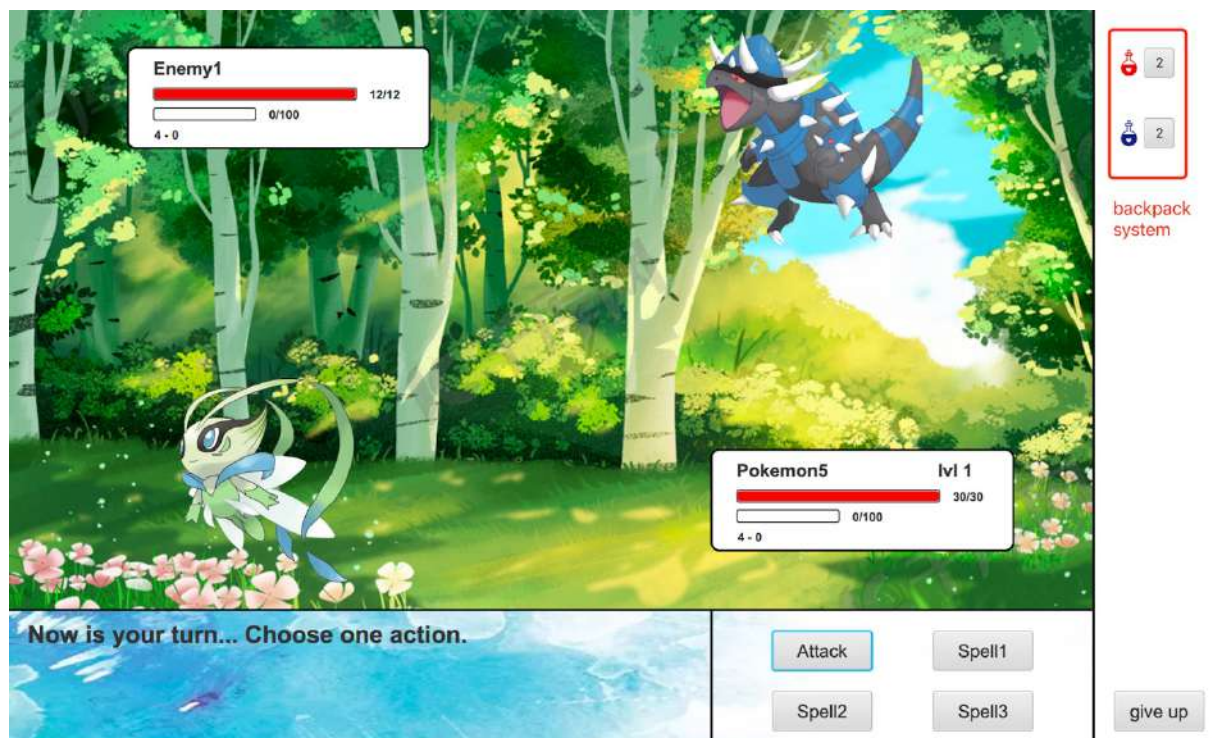
Conversely, if the player reaches the entrance of the map and wants to switch to the last map, he /she can control the Pokémon to move backwards, and the interface will switch to the last map. If the current map is the first map, it will not respond to the illustrated action above.

2.3.2 Backpack System

The game allows players to store HP bottles and MP bottles into the backpack. Players will be initially assigned two HP bottles and two MP bottles. If players encounter a new HP bottle or MP bottle in the map, the bottle will be automatically stored into the backpack. Players can check their props inside the backpack by clicking "My attributes" button in the bottom left corner of the interface:

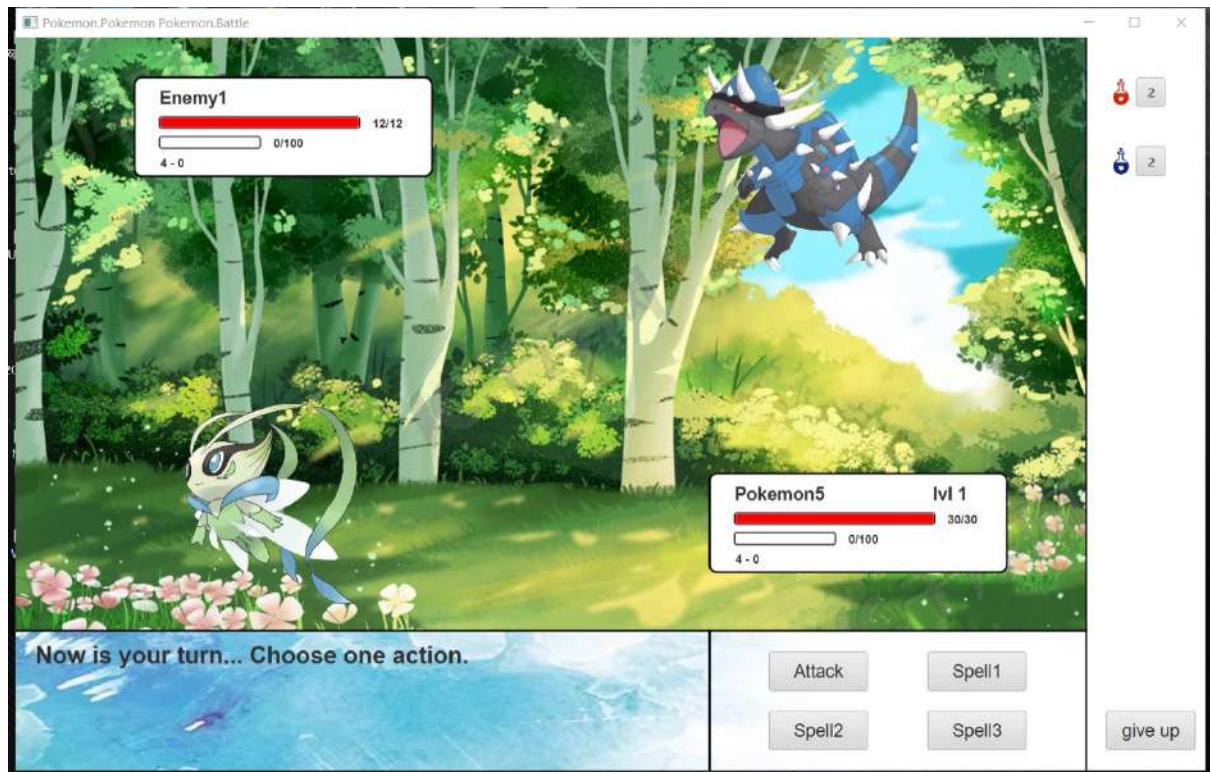


During each battle, players can use the bottles inside the backpack in turn by clicking the number of bottles on the right side of the interface:



2.3.3 Combat System

The GUI is implemented using Javafx. The example is as followed:



GUI

Top left: Enemy info bar (Name, HP bar, MP bar, attack - defence)

Top right: Enemy avatar

Bottom right: User info bar (Name, HP bar, MP bar, attack - defence)

Bottom left: User avatar

Bottom: Information of turn system.

Control

Right: HP poison (60% HP recover), MP poison (40 mana recover), leave the game.

User control: Attack, Spell 1(20 mana cost), Spell 2(20 mana cost), Spell 3(80 mana cost).

If mana is not enough for spell, we block this button, same mechanism as poisons.

Turn System

Project uses **Turn** system: User turn -> Enemy turn -> User turn -> Enemy turn ... (Until User/Enemy loses all HP)

If the user loses all HP, it will redirect to the Game Over page.

If the user wins the battle, it will redirect to the **Map**.

Skills Setting (All skills have animations)

Attack: Use (Attack - Defence) and deal damage. No mana cost.

Spell 1: 20 mana cost.

Spell 2: 20 mana cost.

Spell 3: 80 mana cost (more powerful).

2.3.4 Enemy System

Enemy 1-4 can only attack the user.

Enemy 5-6 can have two random spells. When it reaches 30 mana, it will use a random spell.

Enemy 7(Advanced): predict whether use Spell_3 is optimal (die if use Spell_3). /Combo spell on 3 & 8.

If user_attack > enemy_attack: use Spell_3 to increase attack, else use Spell_8 to deal with more damage.

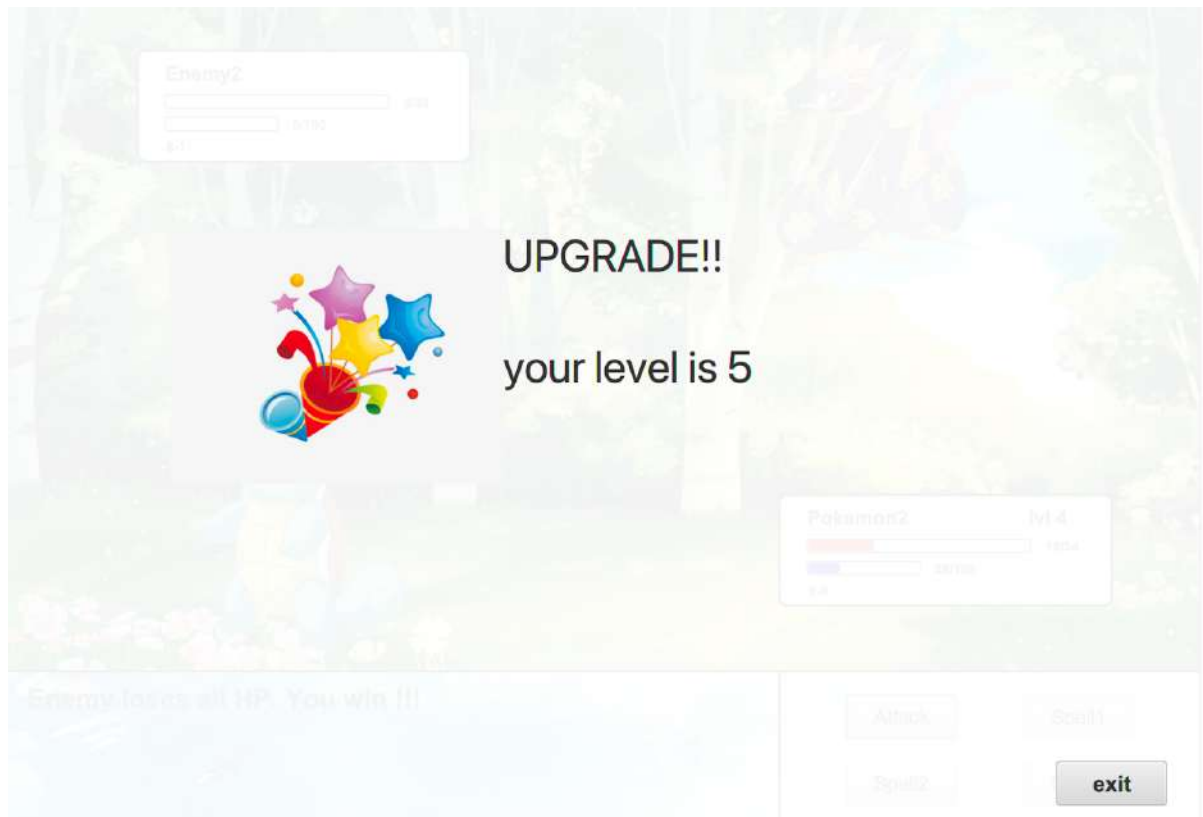
Enemy 8(Advanced): predict optimal from using Spell_3(check the HP spend on and the damage gain).

If current HP is low, use Spell_5 to recover HP. Otherwise, use Spell_9 to reduce use attack (permanent).

Enemy 7 & 8 will think of using saved mana for Spell_3.

2.3.5 Player Upgrading System

After each battle, if players win, they can gain some experience (depending on the enemy's level). And if the experience grows to a certain level, players can upgrade their Pokémons correspondingly:



Different levels will have different requirements of the number of experiences to upgrade. The relationship between different levels and the required number of experiences:

Current Level Region	Number of experiences needed to upgrade
Level 1 - 10	10 per level
Level 10 - 20	100 per level
Level 20 - 30	500 per level
Level 30 - 40	2000 per level

Different level will bring extra abilities for the Pokémon (except obtaining props to unlock the ability ahead):

Level	Ability to cross grass	Ability to cross water	Ability to cross stone wall
Above 10	True	False	False
Above 20	True	True	False
Above 30	True	True	True

3 DEVELOPMENT TOOLS AND PLATFORMS

3.1 Programming language

This project uses Java as the main back-end programming language, and Javafx as the main front-end programming language.

3.2 Tools

Our team is united in using IntelliJ IDEA as the IDE tool and using GitLab as the platform for code commit and git version control.

4 GAME TESTER

In this project, we will use JUnit4 as our main testing package. Since the operation in our game is mainly based on the players' keyboard control on GUI interface, we prepared a series of separated testing classes and methods to verify the different classes inside our game's back-end coding instead of creating an auto tester to cover the whole process. We have totally created five kinds of test classes for the whole game's test work.

4.1 testPokémon

Test Purpose

This test class aims to test the creation and the change of Pokémon.

Related Class and Methods

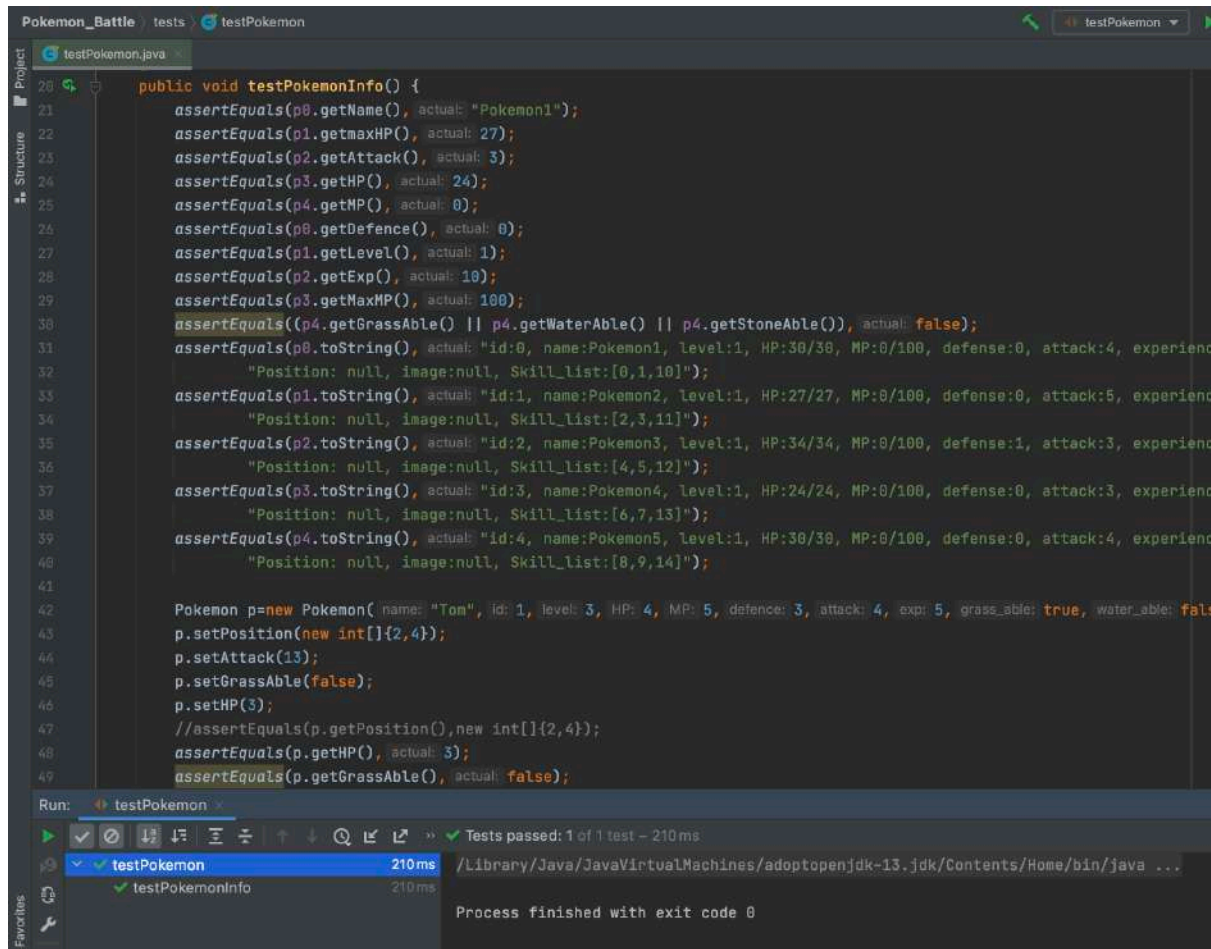
In our game, the class 'Pokémon' is used for creating and storing Pokémon information. There are five different Pokémon with different initial basic information, which are loaded from the json file 'Pokémon.json'. The basic information includes Pokémon's 'id', 'name', 'level', 'HP', 'MP', 'attack', 'defence', 'exp', 'grass_able', 'water_able', 'stone_able', 'max_HP', and 'skill_list'. The accuracy of the creation and change of Pokémon's information will influence the first and second interface of the game.

Test Procedure

As the basic information of each Pokémon is loaded from the same json file, we created five different instances of Pokémon in the test class by using the method 'GUI.PokémonLoadFromJson()', covering all kinds of Pokémons in this game. For each instance, we randomly chose one to two attributes and used the assertEquals method to verify the accuracy of values in each attribute. Those tests also verified the accuracy of loading information from the json file at the same time. Meanwhile, in order to verify the changes of values of attributes which should happen during the game (by using a series of GET and SET methods), we created another Pokémon instance by using the Constructor inside the Pokémon class and manually set the attributes, and then randomly verified the values of Pokémon's attributes.

Test Results

The test cases all passed and confirmed the correct usage of loading the json file, initializing basic information of each attribute, as well as storing the changes of Pokémon's attributes.



```
20 public void testPokemonInfo() {
21     assertEquals(p0.getName(), actual: "Pokemon1");
22     assertEquals(p1.getMaxHP(), actual: 27);
23     assertEquals(p2.getAttack(), actual: 3);
24     assertEquals(p3.getHP(), actual: 24);
25     assertEquals(p4.getMP(), actual: 0);
26     assertEquals(p0.getDefence(), actual: 0);
27     assertEquals(p1.getLevel(), actual: 1);
28     assertEquals(p2.getExp(), actual: 10);
29     assertEquals(p3.getMaxMP(), actual: 100);
30     assertEquals((p4.getGrassAble() || p4.getWaterAble() || p4.getStoneAble()), actual: false);
31     assertEquals(p0.toString(), actual: "id:0, name:Pokemon1, level:1, HP:30/30, MP:0/100, defense:0, attack:4, experience:0, position:null, image:null, skill_list:[0,1,10]");
32     assertEquals(p1.toString(), actual: "id:1, name:Pokemon2, level:1, HP:27/27, MP:0/100, defense:0, attack:5, experience:0, position:null, image:null, skill_list:[2,3,11]");
33     assertEquals(p2.toString(), actual: "id:2, name:Pokemon3, level:1, HP:34/34, MP:0/100, defense:1, attack:3, experience:0, position:null, image:null, skill_list:[4,5,12]");
34     assertEquals(p3.toString(), actual: "id:3, name:Pokemon4, level:1, HP:24/24, MP:0/100, defense:0, attack:3, experience:0, position:null, image:null, skill_list:[6,7,13]");
35     assertEquals(p4.toString(), actual: "id:4, name:Pokemon5, level:1, HP:30/30, MP:0/100, defense:0, attack:4, experience:0, position:null, image:null, skill_list:[8,9,14]");
36
37     Pokemon p=new Pokemon( names: "Tom", id: 1, level: 3, HP: 4, MP: 5, defence: 3, attack: 4, exp: 5, grass_able: true, water_able: false);
38     p.setPosition(new int[]{2,4});
39     p.setAttack(13);
40     p.setGrassAble(false);
41     p.setHP(3);
42     //assertEquals(p.getPosition(),new int[]{2,4});
43     assertEquals(p.getHP(), actual: 3);
44     assertEquals(p.getGrassAble(), actual: false);
45 }
```

Run: testPokemon x

Tests passed: 1 of 1 test - 210 ms

testPokemon 210ms /Library/Java/JavaVirtualMachines/adoptopenjdk-13.jdk/Contents/Home/bin/java ...

testPokemonInfo 210ms

Process finished with exit code 0

4.2 testEnemy

Test Purpose

This test class aims to test the creation and the change of enemy.

Related Class and Methods

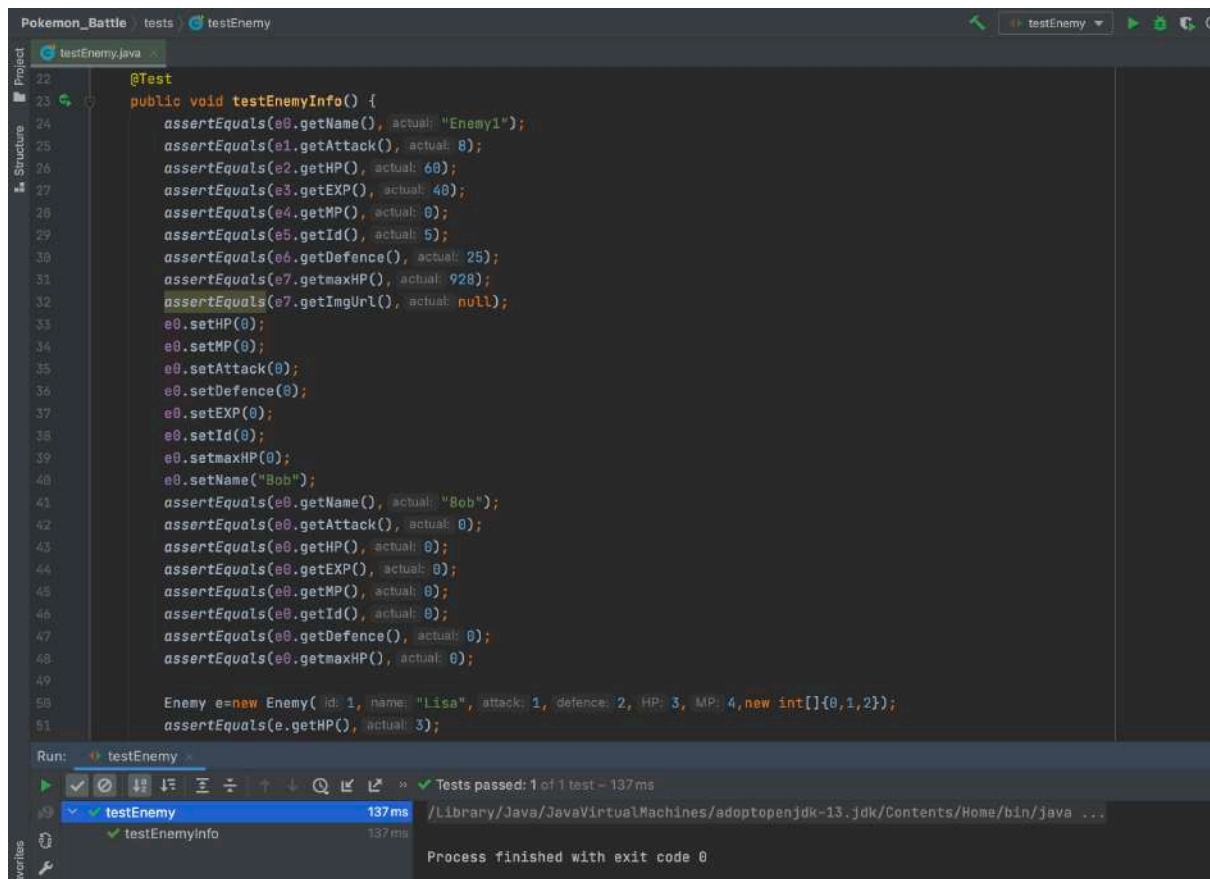
In our game, the class 'Enemy' is used for creating and storing enemy's information. There are eight different enemies with different initial basic information, which are loaded from the json file 'Enemy.json'. The basic information includes enemy's 'id', 'name', 'HP', 'MP', 'attack', 'defence', 'maxHP', 'EXP' and 'skill_list'. The accuracy of the creation and change of enemies' information will influence the battle interface of the game.

Test Procedure

As the basic information of each enemy is loaded from the same json file, we created eight different instances of enemies in the test class by using the method 'GUI.enemy_loading()', covering all kinds of enemies in this game. For each instance, we randomly chose one to two attributes and used the assertEquals method to verify the accuracy of values in each attribute. Those tests also verified the accuracy of loading information from the json file at the same time. Meanwhile, in order to verify the changes of values of attributes which should happen during the game (by using a series of GET and SET methods), we created another enemy instance by using the Constructor inside the Enemy class and manually set the attributes, and then randomly verified the values of enemy's attributes.

Test Results

The test cases all passed and confirmed the correct usage of loading the json file, initializing basic information of each attribute, as well as storing the changes of the enemy's attributes.



```
22  @Test
23  public void testEnemyInfo() {
24      assertEquals(e0.getName(), actual: "Enemy1");
25      assertEquals(e1.getAttack(), actual: 8);
26      assertEquals(e2.getHP(), actual: 60);
27      assertEquals(e3.getEXP(), actual: 40);
28      assertEquals(e4.getMP(), actual: 0);
29      assertEquals(e5.getId(), actual: 5);
30      assertEquals(e6.getDefence(), actual: 25);
31      assertEquals(e7.getMaxHP(), actual: 928);
32      assertEquals(e7.getImgUrl(), actual: null);
33      e0.setHP(0);
34      e0.setMP(0);
35      e0.setAttack(0);
36      e0.setDefence(0);
37      e0.setEXP(0);
38      e0.setId(0);
39      e0.setMaxHP(0);
40      e0.setName("Bob");
41      assertEquals(e0.getName(), actual: "Bob");
42      assertEquals(e0.getAttack(), actual: 0);
43      assertEquals(e0.getHP(), actual: 0);
44      assertEquals(e0.getEXP(), actual: 0);
45      assertEquals(e0.getMP(), actual: 0);
46      assertEquals(e0.getId(), actual: 0);
47      assertEquals(e0.getDefence(), actual: 0);
48      assertEquals(e0.getMaxHP(), actual: 0);
49
50      Enemy e=new Enemy( id: 1, name: "Lisa", attack: 1, defence: 2, HP: 3, MP: 4, new int[]{0,1,2});
51      assertEquals(e.getHP(), actual: 3);
```

Run: testEnemy x

Tests passed: 1 of 1 test - 137 ms

testEnemy 137 ms /Library/Java/JavaVirtualMachines/adoptopenjdk-13.jdk/Contents/Home/bin/java ...

testEnemyInfo 137 ms

Process finished with exit code 0

4.3 testMap

Test Purpose

This test class aims to test the creation and the change of map.

Related Class and Methods

In our game, the class 'Map' is used for creating and storing map information. There are 23 different elements on the map standing for different things in our game, including eight kinds of enemies, seven kinds of props, four kinds of obstacles, the entrance and exit of the map, and the blank place which allow users to move. All the map elements are loaded from the text file 'battleMap.txt' with a bespoke method. And those map elements will change when players move onto related elements and execute the activities including battle or picking up props. The accuracy of the creation and change of maps will influence the maze exploration, battle, and the upgrading of players during the game.

Test Procedure

As the basic information of each map is loaded from a text file, we created an extra text file with virtual situations and loaded the text file into the test class by using the method 'initialMap()'. This test map covered all kinds of elements in this game.

Firstly, we chose one of the places which is the exit or the terminal of the map and used the assertTrue and assertFalse method to verify the judgement if the player has reached the end of the map. Those tests also verified the accuracy of loading information from the json file at the same time.

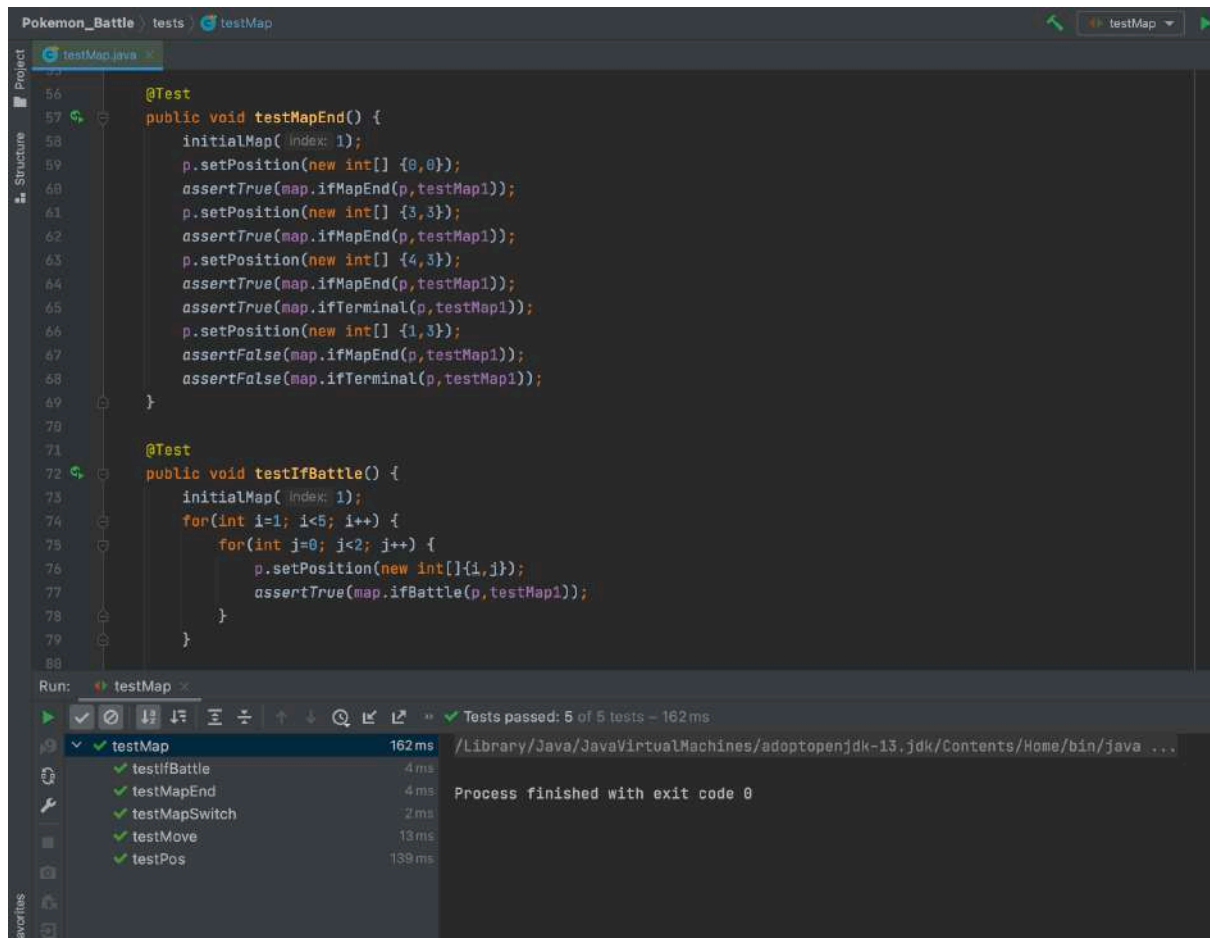
Secondly, we use assertTrue and assertFalse method to verify all the elements in this test map and the judgement of whether the player met an enemy and needed to start fighting.

Thirdly, in order to verify the judgement of movement for the player, we manually set the position and action of the player and use assertTrue and assertFalse method to verify if the player can move to the target direction. Furthermore, we set the situation that the player's action is to change to the next/last map, to verify the judgement of map switching.

Finally, we used assertEquals method to verify the accuracy of each element in the map.

Test Results

The test cases all passed and confirmed the correct usage of loading the text file, initializing map elements, as well as storing the changes of the map.



4.4 testSearch

Test Purpose

This test class aims to test the accuracy of using BFS algorithm in a map.

Related Class and Methods

In our game, the class 'Search' and its method is used for players to search the place of a certain target which is the nearest to players by using breadth first search algorithm. if the target does not exist on the map, or players are just standing on the target, the method will return null, otherwise it will return a list consisting of the steps for players to reach the target.

Test Procedure

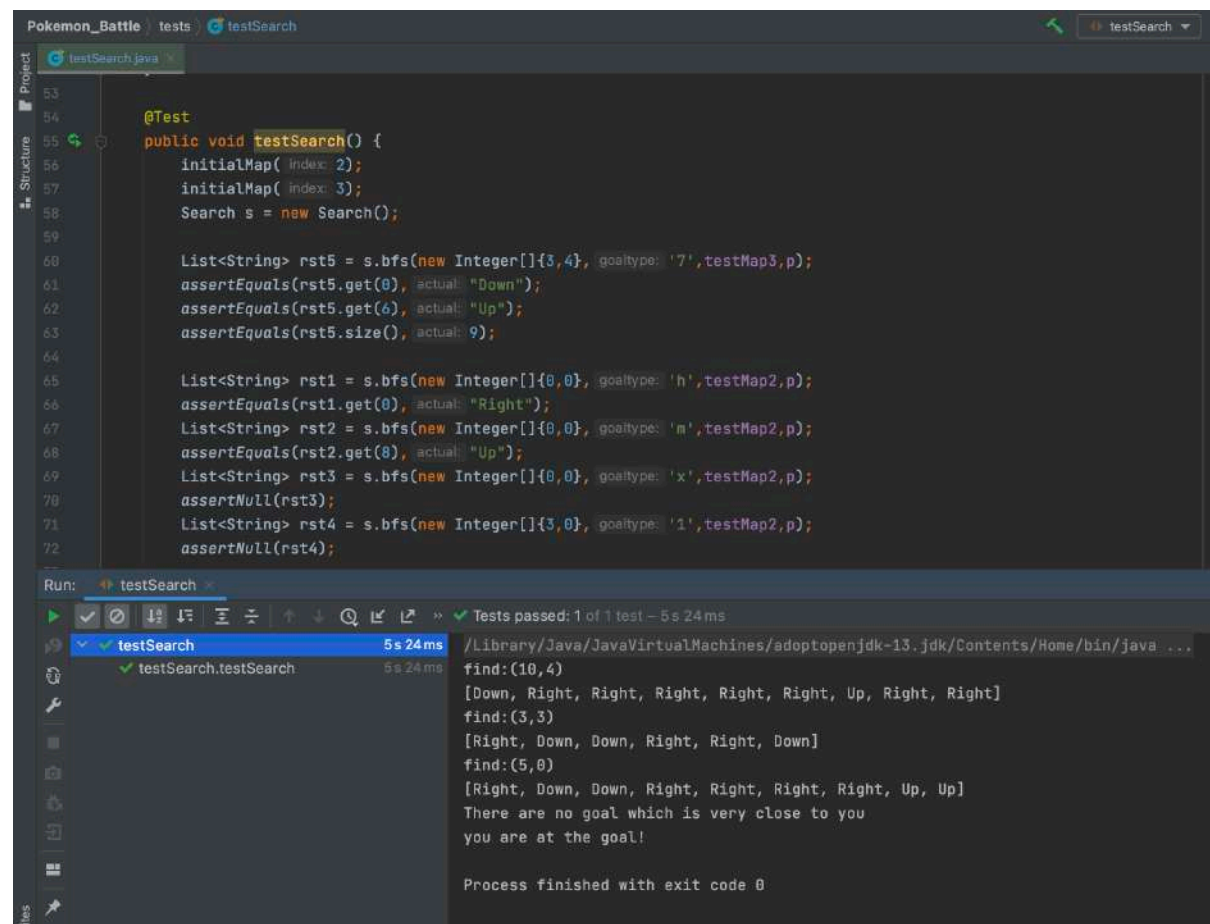
Similar to the test procedure in testMap, we first created two extra text files with virtual situations and loaded the text files into the test class by using the method 'initialMap()'. These two test maps covered different situations for searching certain targets.

The first test scenario is for players to get around the obstacles and reach the target. We manually set the current position of the player and use the method 'bfs()', and then use assertEquals method to verify if the algorithm found the correct route.

The second test scenario covers the search for targets which do not exist on the map, the search for targets which are unable to reach (no accessible route exists), and the search for simple routes. We manually set the current position of the player and use the method 'bfs()', and then use assertNull method to verify if the algorithm found that no possible route exists, as well as using assertEquals method to verify if the algorithm found the correct routes.

Test Results

The test cases all passed and confirmed the correct usage of the searching algorithm.



The screenshot shows an IDE window titled 'Pokemon_Battle' with a sub-window 'tests' containing 'testSearch'. The main editor displays the code for 'testSearch.java'. The code includes a '@Test' annotation and a 'public void testSearch()' method. Inside the method, it initializes a map, creates a 'Search' object, and performs several 'bfs' (Breadth-First Search) operations. It uses 'assertEquals' to verify the results of the searches and 'assertNull' to verify that no route exists for certain goals. The 'Run' button is highlighted, and the output window shows the results of the test execution.

```
@Test
public void testSearch() {
    initialMap( index: 2);
    initialMap( index: 3);
    Search s = new Search();

    List<String> rst5 = s.bfs(new Integer[] {3,4}, '7', testMap3, p);
    assertEquals(rst5.get(0), actual: "Down");
    assertEquals(rst5.get(4), actual: "Up");
    assertEquals(rst5.size(), actual: 9);

    List<String> rst1 = s.bfs(new Integer[] {0,0}, 'h', testMap2, p);
    assertEquals(rst1.get(0), actual: "Right");
    List<String> rst2 = s.bfs(new Integer[] {0,0}, 'm', testMap2, p);
    assertEquals(rst2.get(0), actual: "Up");
    List<String> rst3 = s.bfs(new Integer[] {0,0}, 'x', testMap2, p);
    assertNull(rst3);
    List<String> rst4 = s.bfs(new Integer[] {3,0}, '1', testMap2, p);
    assertNull(rst4);
}
```

Run: testSearch

Tests passed: 1 of 1 test - 5 s 24 ms

testSearch 5 s 24 ms

testSearch.testSearch 5 s 24 ms

/Library/Java/JavaVirtualMachines/adoptopenjdk-13.jdk/Contents/Home/bin/java ...

find:(10,4)

[Down, Right, Right, Right, Right, Right, Up, Right, Right]

find:(3,3)

[Right, Down, Down, Right, Right, Down]

find:(5,0)

[Right, Down, Down, Right, Right, Right, Right, Up, Up]

There are no goal which is very close to you

you are at the goal!

Process finished with exit code 0

4.5 testBattle

Test Purpose

This test class aims to test the change of Pokémon's or enemy's basic information, the battle result, and the usage of intelligent judgement methods during the game's battle.

Related Class and Methods

In our game, the class 'Battle' is used for controlling the changes of Pokémon's or enemy's basic information and the battle results related. If players win the battle, there will also be some changes in experience value and level, which also influence the player's basic information and abilities.

Meanwhile, we have an intelligent judgement method in class 'Battle' for a certain kind of enemy to decide which is the best way (better for winning battles) to attack the player, in order to increase the difficulty of the game.

Test Procedure

Firstly, we created five different Pokémon in order to cover all the situations with change of level and abilities (different Pokémon have different award settings in upgrading). We also created two different enemies, because a certain enemy (id=6) will be used for testing the intelligent judgement method. Then we created the Battle instances by using the constructor of Battle class.

For the battle results, we manually changed the HP value of Pokémon and enemy and used the `assertTrue` method to verify the results. For the intelligent judgement method, we designed six different scenarios to cover all the branches inside the method '`ai()`', and use `assertEquals` method to verify the result of each scenario.

As for the changes of three abilities (grass ability, water ability, and stone ability) due to an upgrade after the battle, we manually changed the level of a Pokémon to the required level and use `assertTrue` method to verify if the change of three different abilities is accurate as designed. And for the changes of basic information due to an upgrade, we manually changed the level of five kinds of Pokémons to the required level and tested the change of attack or defense value of each Pokémon after the upgrade respectively.

Test Results

The test cases all passed and confirmed the change of Pokémon's or enemy's basic information and the battle result, as well as the correct usage of the intelligent judgement method.

Pokemon_Battle tests testBattle

testBattle.java

```
25 @Test
26 public void testBattleResults() {
27     p0.setHP(0);
28     assertTrue(b0.gameover_test());
29     p0.setHP(30);
30     e6.setHP(0);
31     assertTrue(b0.win_test());
32 }
33
34
35 @Test
36 public void testAI() {
37     p0.setAttack(150);
38     e6.setMP(80);
39     e6.setHP(500);
40     assertEquals(b0.ai(), actual: 2);
41     e6.setMP(40);
42     assertEquals(b0.ai(), actual: -1);
43     e6.setMP(30);
44     assertEquals(b0.ai(), actual: -1);
45     e6.setMP(10);
```

Run: testBattle

Tests passed: 4 of 4 tests - 170 ms

testBattle	170 ms	/Library/Java/JavaVirtualMachines/adoptopenjdk-13.jdk/Contents/Home/bin/java ...
testAbility	142 ms	38
testAI	12 ms	54
testBattleResults	8 ms	86
testCal	8 ms	4.8000000000000001
		42.0
		4.8000000000000001
		42.0
		4.8000000000000001
		42.0

Process finished with exit code 0