Test Plan Document

For

Pokémon Go Back

Prepared by:

Name	ID
Amitt Bhardwj	40037899
Gaganjot Kaur	40037933
Gunpreet Ahuja	40041311
Gurpreet Kaur Lotey	40038315
Gurpreet Raju	40013682
Harinder Kaur	40041313
Mandeep Singh	40040964
Navjot Singh	40014477
Navrang Pal Kaur Dhaliwal	40047555
Simranjit Singh Bachher	40040601

Instructor name: Stuart Thiel

Table of Content

1. TEST PLAN

1.1 SCOPE AND APPROACH

The scope of this document is to develop some testing strategies to analyze our system for bugs and errors. This document is based on IEEE standard 829. We have performed unit testing on the individual methods/functions of the classes of our project using 'JUnit'. We have also used JUnit to perform 'integration testing' on our code. Once we familiar ourselves with more testing strategies, we will develop test cases for 'system level' testing and also include more unit tests and integration tests for complete code coverage. Results will be recorded after performing the testing. Only when the classes have passed their corresponding test, they are considered acceptable.

1.2 RESOURCES

We have the following resources:

- Junit library
- Software Requirement Document
- Software Design Document
- Software Testing Plan
- Software Verification and Validation Plan
- Software Development Team
- Software Testing Team

1.3 ITEMS TO BE TESTED

We have found the following items that need to be tested:

- Software Requirements Document
- Software Design Document
- Source Code

1.4 FEATURES TO BE TESTED

Features are the functions that our system will provide to the users -:

Feature Name	**Corresponding Use Case**
Setup Game	UC1

Cards Dealt	UC2
Play Turn	UC3
Add Energy	UC4
Play Trainer	UC5
Evolve	UC6
Retreat	UC7
Attack	UC8
End Turn	UC9

1.5 Roles and Contribution

For the **roles and contribution** please have a look on **Software Design Specification document** [**Section 7**]

2. TEST DESIGN SPECIFICATION

TEST REFINEMENT APPROACH

For the refinement of testing, we will perform the informal reviews. We will perform casual meeting with our Software Quality Assurance team, both team members will discuss the testing approaches and find the most suitable approaches.

1.2 TESTING APPROACHES

We will follow the Strategic Testing approach. We will perform Unit testing, Acceptance Testing, Regression Testing, Integration testing, Validation testing and System Testing. The order of the testing is from Unit testing, Integration testing, Regression testing, Validation testing, System testing and Acceptance testing.

Unit Testing: For unit testing, we use White box and Black box testing. For White box testing we create test cases that can guarantee that all the independent paths have been executed at least once. Whereas for black box testing we create some test cases and verify the behavior from outputs received. Unit is performed by the developer.

Integration testing: After unit testing, we will collaborate the integrated modules and start integration testing to verify combined functionality. Integration testing will be done continuously as unit tests are executed. So, most of the time will be consumed in integration testing. Integration testing is performed by the developer.

Validation testing: After Integration Testing, Validation testing is performed by Independent Tester.

Regression testing: After Validation testing, the whole application is tested for the modification in any module or functionality. This testing is done to check if bugs are produced due to added functionality. Regression Testing is performed by Software Testing Team.

System testing: Entire system is tested as per the requirements. Black-box type testing that is based on overall requirements specifications, covers all combined parts of a system. System testing is performed by Independent Tester.

Acceptance testing: Acceptance testing is done to verify if system meets the user specified requirements. User does this testing to determine whether to accept application. It is the last step in testing. Acceptance Testing is performed by User. Although it is going to be performed by the respected evaluator but still we have done with our team members.

3. TEST CASE SPECIFICATION

The purpose of the Test Case is to formally identify and communicate the detailed specific conditions which will be validated to enable an evaluation of some particular aspects of the Target Test Items. Test Cases may be motivated by many things but will usually include a subset of both the Requirements (Use Cases, performance characteristics, etc.) and the risks the project is concerned with.

The Test Case is primarily used:

- To enumerate an adequate number of specific tests to ensure evaluation completeness.
- To identify and reason about required Test Scripts and drivers, both manual and automated.
- To provide an outline for the implementation of Test Scripts and drivers by providing a description of key points of observation and control, and any pre and post conditions.

3.1 Brief Outline

3.1.1 Test Case ID: It gives Test cases sequence number. It is a unique identification number for tests.

3.1.2 Test Case Summary: A summary of the purpose or objective of the test, the scope, and any preconditions of the test.

3.1.3. Prerequisites: Prerequisites describes the required state that the system should be in before the test begins. It is required for each execution condition.

3.1.4. Test Procedure: The Test procedure includes a list of specific steps to be applied during the test. It includes the fields or objects interacted with the specific data values entered when executing the test case.

3.1.5. Test Data: It includes the data required for executing the test case.

3.1.6. Expected Result: The resulting state which is expected as a result of the test having been executed. It may include positive and negative responses.

3.1.7. Actual Result: The result which we acquire after running the test case.

3.1.8. Created By: Name of Software Testing Team members, who created the test cases.

3.1.9. Date of Creation: The date, particular test case is created.

3.1.10. Test Environment: System and software required for executing test case successfully.

4. JUnit4 Framework

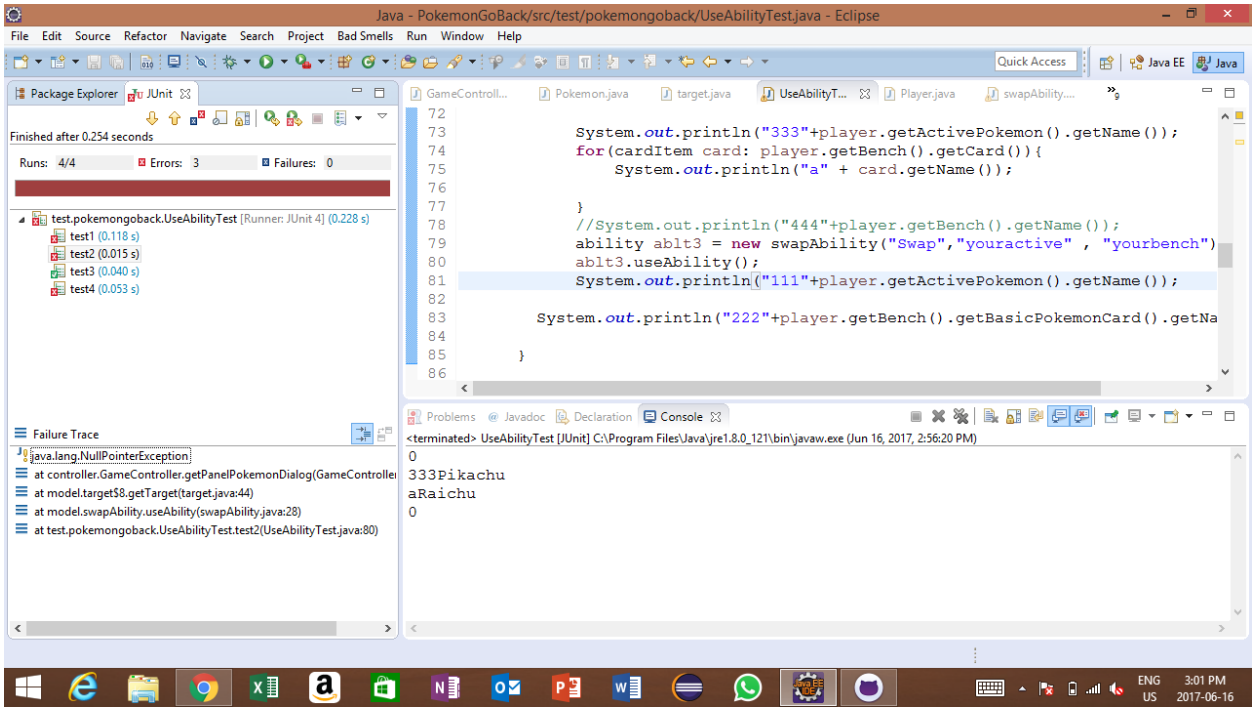
JUnit is a unit testing framework for the Java programming language. Unit has been important in the development of test-driven development, and is one of a family of unit testing frameworks which is collectively known as **xUnit** that originated with **SUnit.**

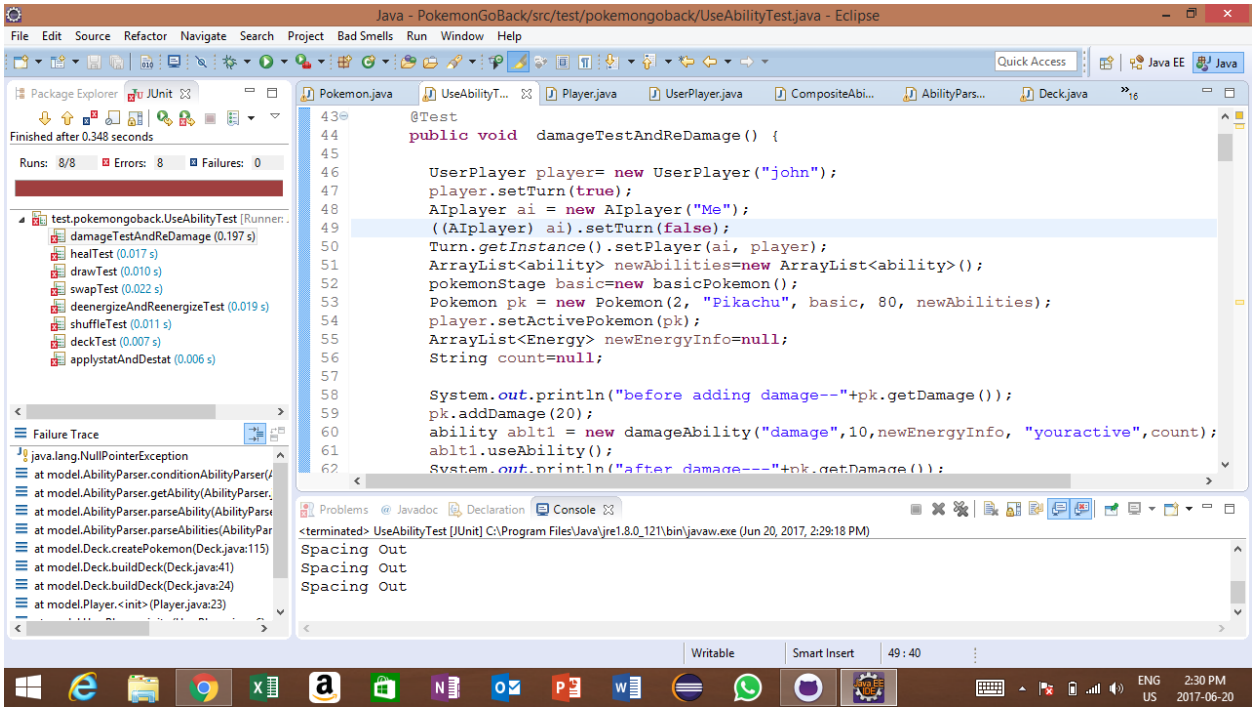
Code Example-:

```
package test.pokemongoback;
import static org.junit.Assert.*;
public class AddDamageTest
{
    ArrayList<ability> newAbilities=new ArrayList<ability>();
    pokemonStage basic=new basicPokemon();
    Pokemon pk = new Pokemon(2, "Pikachu", basic, 60, newAbilities, null);
    UserPlayer up= new UserPlayer("john");

    @Test
    public void test()
    {
        int expected=10;
        pk.addDamage(10);
        int actual=pk.getDamage();
        assertEquals(expected, actual);
        String expected2="deck";
        String expected1="knockedOut";
        up.setActivePokemon(pk);
        pk.addDamage(10);
        String actual2=pk.getState();
        assertEquals(expected2, actual2);
        pk.addDamage(60);
        String actual1=pk.getState();
        pk.addDamage(80);
        String actual3=pk.getState();
        assertEquals(expected1, actual3);
        assertEquals(expected1, actual1);
    }

}
```

5.Regression Testing

Here are some screenshots of the regression testing that shows how the changes in code or minor errors will lead to fail the test.

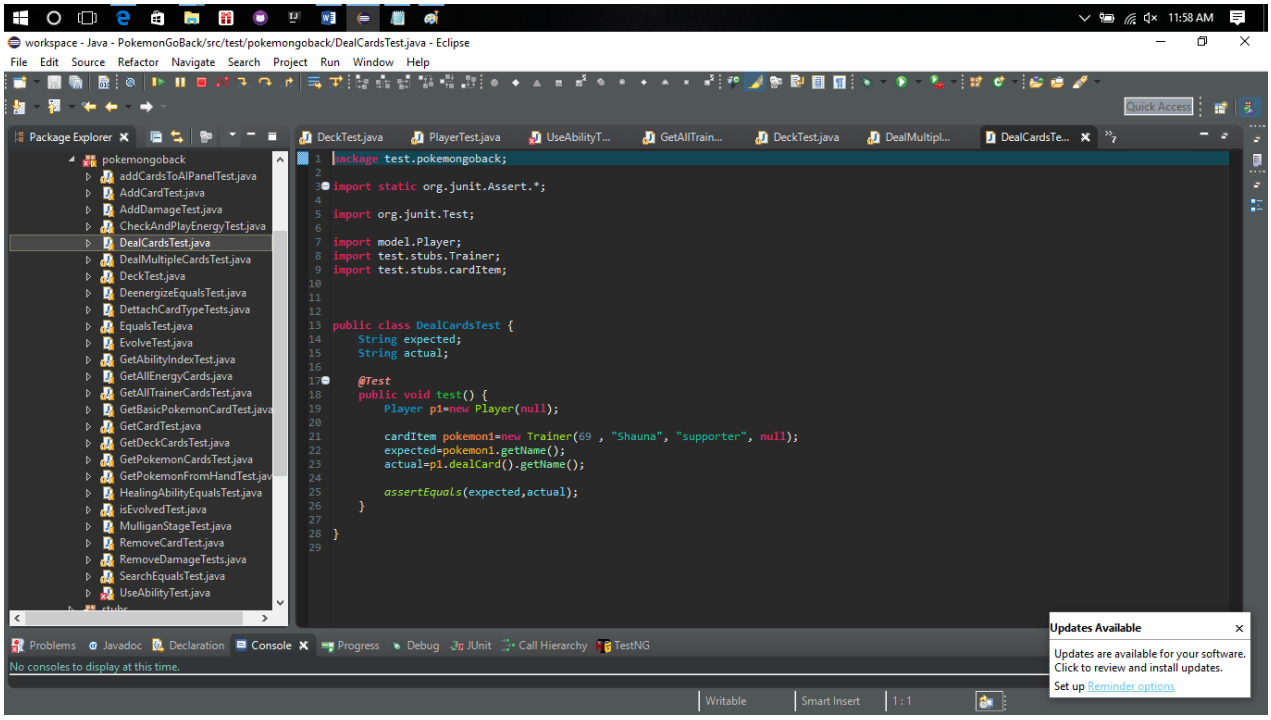



Java - PokemonGoBack/src/test/pokemongoback/UseAbilityTest.java - Eclipse
File Edit Source Refactor Navigate Search Project Bad Smells Run Window Help
Quick Access
Java EE
Java
Package Explorer
JUnit
Finished after 0.348 seconds
Runs: 8/8
Errors: 8
Failures: 0
test.pokemongoback.UseAbilityTest [Runner:
damageTestAndReDamage (0.197 s)
healTest (0.017 s)
drawTest (0.010 s)
swapTest (0.022 s)
deenergizeAndReenergizeTest (0.019 s)
shuffleTest (0.011 s)
deckTest (0.007 s)
applystatAndDestat (0.006 s)
Failure Trace
java.lang.NullPointerException
at model.AbilityParser.conditionAbilityParser(A
at model.AbilityParser.getAbility(AbilityParser.
at model.AbilityParser.parseAbility(AbilityPars
at model.AbilityParser.parseAbilities(AbilityPar
at model.Deck.createPokemon(Deck.java:115)
at model.Deck.buildDeck(Deck.java:41)
at model.Deck.buildDeck(Deck.java:24)
at model.Player.<init>(Player.java:23)
Pokemon.java
UseAbilityT...
Player.java
UserPlayer.java
CompositeAbi...
AbilityPars...
Deck.java
43 @Test
44 public void damageTestAndReDamage() {
45
46 UserPlayer player= new UserPlayer("john");
47 player.setTurn(true);
48 AIplayer ai = new AIplayer("Me");
49 ((AIplayer) ai).setTurn(false);
50 Turn.getInstance().setPlayer(ai, player);
51 ArrayList<ability> newAbilities=new ArrayList<ability>();
52 pokemonStage basic=new basicPokemon();
53 Pokemon pk = new Pokemon(2, "Pikachu", basic, 80, newAbilities);
54 player.setActivePokemon(pk);
55 ArrayList<Energy> newEnergyInfo=null;
56 String count=null;
57
58 System.out.println("before adding damage--"+pk.getDamage());
59 pk.addDamage(20);
60 ability ablt1 = new damageAbility("damage",10,newEnergyInfo, "youractive",count);
61 ablt1.useAbility();
62 System.out.println("after damage---"+pk.getDamage());
Problems
Javadoc
Declaration
Console
<terminated> UseAbilityTest [JUnit] C:\Program Files\Java\jre1.8.0_121\bin\javaw.exe (Jun 20, 2017, 2:29:18 PM)
Spacing Out
Spacing Out
Spacing Out
Writable
Smart Insert
49 : 40
ENG
US
2:30 PM
2017-06-20

6.Stubs

Test stubs are mainly used in incremental testing's top-down approach. Stubs are computer programs that act as temporary replacement for a called module and give the same output as the actual product or software.

Example-:



7.Test Cases

For the **test cases** please check the **PDF** file named as a **Test Cases** in **Document** folder.

Test Case	Type
TC_001	**Integration**
TC_002	**Integration**
TC_003	**Integration**
TC_004	**UNIT**
TC_005	**Integration**
TC_006	**Integration**
TC_007	**Integration**
TC_008	**UNIT**
TC_009	**Integration**
TC_010	**Integration**
TC_011	**Integration**
TC_012	**Integration**
TC_013	**Integration**
TC_014	**Integration**
TC_015	**UNIT**
TC_016	**UNIT**
TC_017	**Integration**

8.Conclusion

Test case document includes various methodologies, techniques as well as components of the system which is under testing. It describes the various testing ways like unit, regression, acceptance. Junit4 framework and stubs used for unit testing. Top-down approach used for integration testing.