**City University of Hong Kong**

Department of Computer Science

CS3343 Software Engineering Practice

2022-23 Semester A

Test Report

Project Title: River Crossing Game

(with customizable puzzle and solver)

Team Members:

Chau Tsun Yu (56625222)

Chau Wai Tong (56660328)

Chin Man Chong (56701200)

Fung King Shun (56659723)

Lam Kin Hei (56618753)

**Table on Contents**

[1. Test Report 4](#_Toc121087344)

[2. Methodology 5](#_Toc121087345)

[3. Coverage Analysis 6](#_Toc121087346)

[4. Testing details (RuleSet.java) 7](#_Toc121087347)

[4.1. Testing function isValidMove() 7](#_Toc121087348)

[4.1.1. Unit testing – getNextState() 8](#_Toc121087349)

[4.1.2. Unit testing – isTravelable() 9](#_Toc121087350)

[4.1.3. Integration Test – isValidMove() 10](#_Toc121087351)

[4.2. Testing function isWin() 14](#_Toc121087352)

[4.2.1. Unit testing – isIdentical() 14](#_Toc121087353)

[4.2.2. Integration Test – isWin() 16](#_Toc121087354)

[5. Testing details (Solver.java) 17](#_Toc121087355)

[5.1. Testing function getLegalMove() 18](#_Toc121087356)

[5.1.1. Unit testing – getTravelable() 18](#_Toc121087357)

[5.1.2. Integration Test – getLegalMoves() 21](#_Toc121087358)

[5.2. Integration test of solve() 23](#_Toc121087359)

[6. Testing Player Mode Undo/Redo Command 25](#_Toc121087360)

[7. System Test 27](#_Toc121087361)

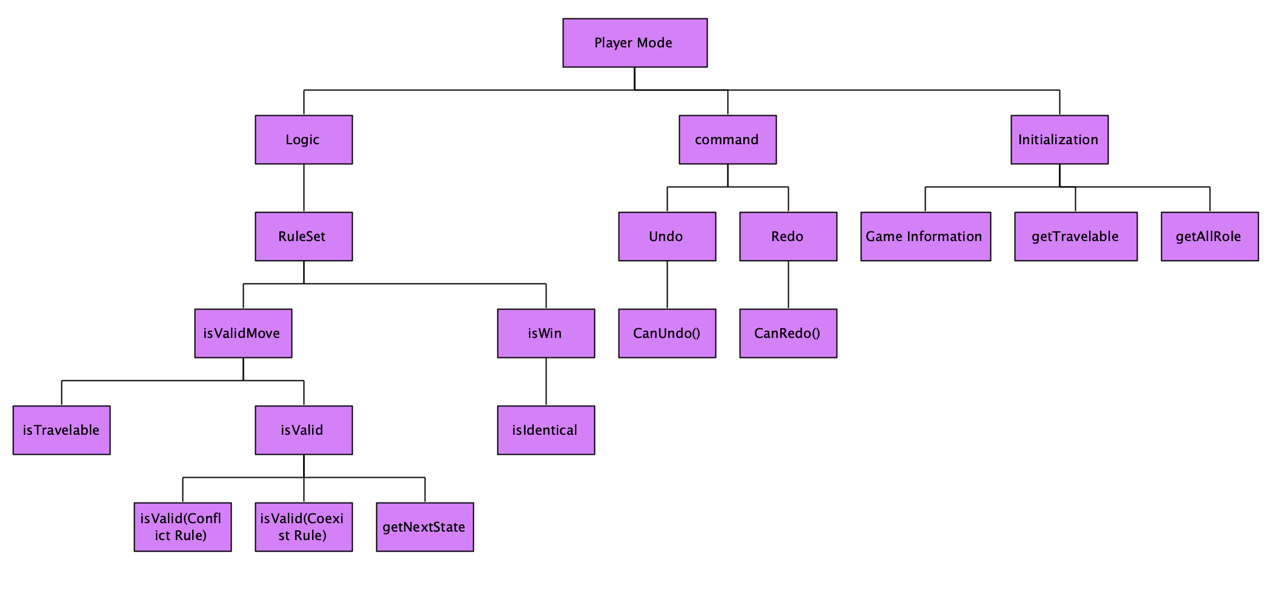
[7.1. Solver Mode 27](#_Toc121087362)

[7.2. Player Mode 28](#_Toc121087363)

[8. User acceptance testing (UAT) 29](#_Toc121087364)

# Test Report

Game Mode A (Player Mode) Hierarchy Diagram



Game Mode B (Solver Mode) Hierarchy Diagram



# Methodology

Our group has chosen Bottom-up testing strategy as our testing method. This is a kind of integration testing where low-level modules are tested first, then followed by high level module. It involves taking integrated code and test those code together, before testing a whole system. The reason why we chosen this testing method is that it is easy to develop test conditions. Besides, Disjoint subsystems can be also tested at the same time. Therefore, it can ensure all the modules inside the system are tested as a single unit. In our design code, our function output depends on different sub-function, especially the logical part in our design. To test every function thoroughly, Bottom-up testing strategy is chosen by our group.

# Coverage Analysis

一張含有 桌 的圖片

自動產生的描述

The overall coverage in our testing is 77.4% (**95% if ignoring the UI**). The main reason why the coverage is under 90% is that the Graphic User interface is hard to test since it mainly depends on user’s input and print out some colorful graphic such as the boat, river, and role. For example, in DrawGraphic.java, The function is only print out some symbol. It does not relate to the logical part and algorithm in our design. The purpose of implementing UI is only for better visual effect. Without considering the user Interface part, the overall coverage is above **95%** if we focus on the logic and algorithm part in our design.

Total number of test Case: 55

一張含有 文字 的圖片

自動產生的描述

# Testing details (RuleSet.java)

一張含有 桌 的圖片

自動產生的描述

## Testing function isValidMove()

一張含有 文字 的圖片

自動產生的描述The main function in RuleSet.java is function - isValidMove(PuzzleState, Move). This function returns a response whether the move is valid or not. The return value depends on different components and functions. They are isTravelable(), isValid() and getNextState(). Following the principle of Bottom-up testing, we should first test these three functions separately first, then perform a integration testing on function isValidMove().

### Unit testing – getNextState()

This function accept a move from user and act as a parameter for function isValid().

The testing method for this function is testing all the combination.

The combination is as following:

1. Two roles cross the river from land A to land B.
2. Only one role crosses the river from land A to land B.
3. No role crosses the river
4. Two roles cross the river from land B to land A.
5. Only one role crosses the river from land A to land B.

Example:

This is a test case simulating two roles “farmer” and “sheep” cross the river from land A and land B successfully.

一張含有 文字 的圖片

自動產生的描述

Coverage:

一張含有 文字 的圖片

自動產生的描述

### Unit testing – isTravelable()

This function accepts a move from user and check whether roles in this move is travelable or not. The testing method for this function is testing all the combination. 一張含有 文字 的圖片

自動產生的描述

The combination is as following:

1. The move involves at least one traveler
2. The move involves no traveler
3. The move is not valid

Example:

This is a test case simulating at least one role “farmer” is on the travelable list, so the function returns true.

一張含有 文字 的圖片

自動產生的描述

Coverage:

一張含有 文字 的圖片

自動產生的描述

### Integration Test – isValidMove()

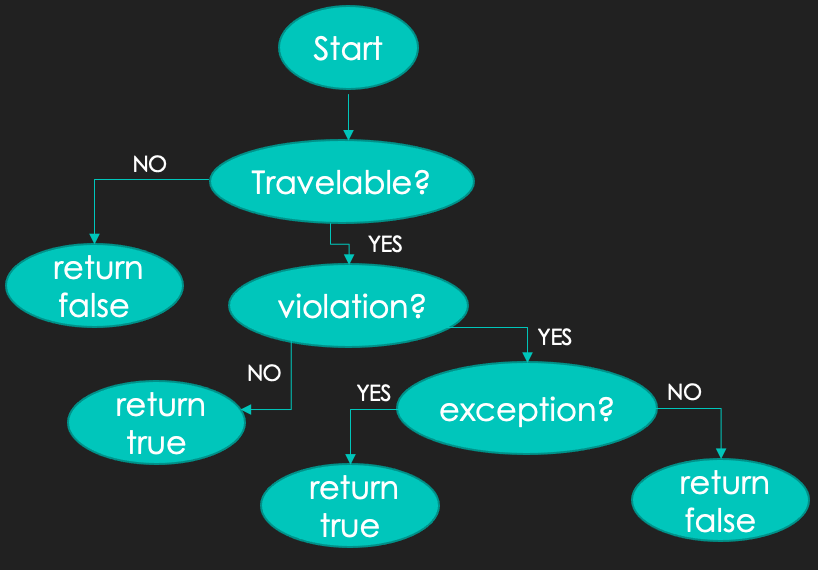
In isValidMove() function, there are total of four path. Our test case has covered all the paths.

Path 1: Start -> Travelable? -> False

Path 2: Start -> Travelable? -> Violation? -> True

Path 3: Start -> Travelable? -> Violation? -> Exception? -> True

Path 3: Start -> Travelable? -> Violation? -> Exception? -> False



Since we have tested isTravelable() and getNextState(), the last step we test is isValid(). isValid() test whether the state violate the conflict or coexist rule. The game rule we decided is as following: If the move violate conflict or coexist rule with exception, then it will return true. Otherwise, it returns false. Therefore, we can design the test case with the truth table below.

Predicate testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | Travelable? | Valid State? | Exception? | Result |
| 1 | True | False  (Violate conflict rule) | False  (Conflict rule exception) | False |
| 2 | True | False  (Violate conflict rule) | True  (Conflict rule exception) | True |
| 3 | True | False  (Violate coexist rule) | False  (Conflict rule exception) | False |
| 4 | True | False  (Violate coexist rule) | True  (Conflict rule exception) | True |
| 5 | True | False  (Violate conflict rule) | False  (Coexist rule exception) | False |
| 6 | True | False  (Violate conflict rule) | True  (Coexist rule exception) | True |
| 7 | True | False  (Violate coexist rule) | False  (Coexist rule exception) | False |
| 8 | True | False  (Violate coexist rule) | True  (Coexist rule exception) | True |
| 9 | False | - | - | False |
| 10 | True | True  (No rule violation) | - | True |

Example:

This is a test case simulating there is violation of conflict rule but with exception of coexist rule.

一張含有 文字 的圖片

自動產生的描述

Integration test coverage

一張含有 桌 的圖片

自動產生的描述

## Testing function isWin()

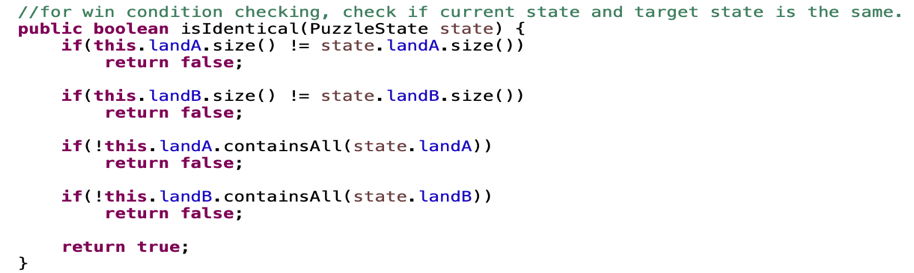
The Win() function is depend on the return value of getTargetState() function inside other class (PuzzleState.java). Therefore, we will first test the function isIdentical() inside PuzzleState.java.

一張含有 文字 的圖片

自動產生的描述

### Unit testing – isIdentical()

The function accepts a parameter-PuzzleState and return true if current land A is equal to final land A and current land B is equal to final land B. Otherwise, it returns false. In our test case, we will consider every if statement inside the isIdentical() function.



There are total of 6 combinations. They are as following:

1. Final land B is equal to current land B but Final land A is not equal to current land A

2. Final land A is equal to current land A but Final land B is not equal to current land B

3. Final land A is not equal to current land A and Final land B is not equal to current land B

4. Final land B size is not equal to current land B

5. Final land A size is not equal to current land A

6. Final land A and land B is equal to current land A and land B

Besides, we have also considered the case of empty puzzleState which mean there is no roles exist in land A and land B respectively.

Example:

This is a test case simulating final land B is not identical to current land B so the return value is false.

一張含有 文字 的圖片

自動產生的描述

Coverage:

一張含有 文字 的圖片

自動產生的描述

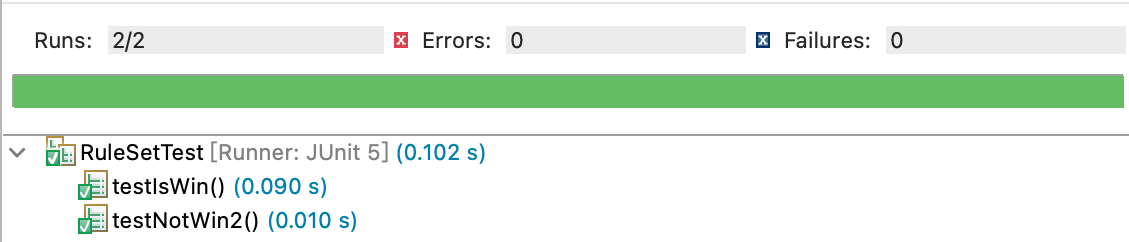
### Integration Test – isWin()

Since the isWin() function is only depend on isIdentical() so we need only two test cases (is win case / not win case) to test the function.

一張含有 文字 的圖片

自動產生的描述

Coverage:



# Testing details (Solver.java)

一張含有 桌 的圖片

自動產生的描述

The main function inside Solver.java is solve () function which returns a solution of river crossing game with input customized role and rule. The solution is mainly depend on the getLegalMove() function since it records every moves occur in a solution.

一張含有 文字 的圖片

自動產生的描述

## Testing function getLegalMove()

Inside the function, it mainly depends on two functions which are getTravelable() and isValidMove(). Since isValidMove() is tested in RuleSet.java so we only need to perform unit testing on getTravelable() function and then perform integration testing of getLegalMoves().

### Unit testing – getTravelable()

Since getTravelable() is only to get the travelable role in the json file. It means it highly depend on the input by players. We don’t know what players will input inside the json file. Therefore, we have tested all possible combination and description of every test case is as following:

Test case 1:

Testing there is only one travelable character.

Json testing file:

一張含有 文字 的圖片

自動產生的描述

Test case 2:

Testing no travelable character.

Json testing file:

一張含有 文字 的圖片

自動產生的描述

Test case 3:

Testing all character is travelable character.

Json testing file:

一張含有 文字, 螢幕, 螢幕擷取畫面 的圖片

自動產生的描述

Test case 4:

Testing a non-exist travelable character.

Json testing file:

一張含有 文字 的圖片

自動產生的描述

Test case 5:

Testing a duplicated travelable character.

Json testing file:

一張含有 文字, 螢幕 的圖片

自動產生的描述

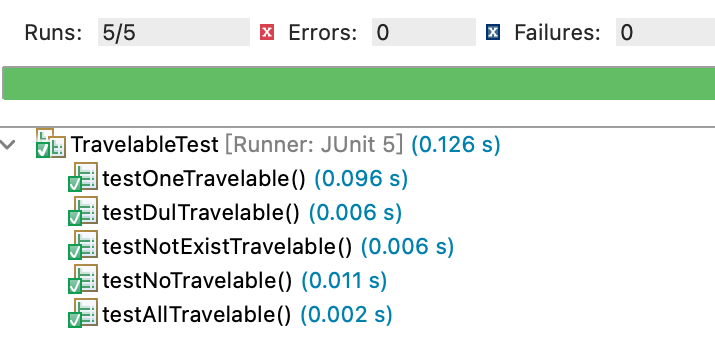
Example:

This is a test case simulating there are only one travelable character.

一張含有 文字 的圖片

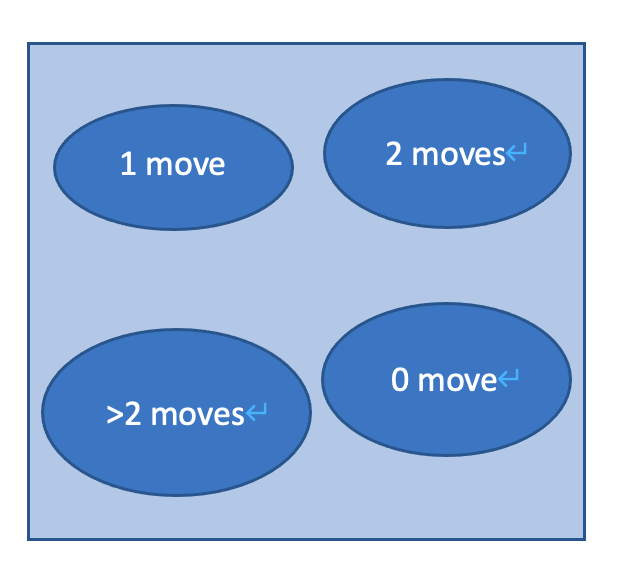
自動產生的描述

Coverage:



### Integration Test – getLegalMoves()

The limitation of this test case is that we might not cover all the possible case. For example, there could be 10 legal moves, 100 legal moves and 1000 legal moves. In our case, we adopt partition testing approach to test every representative from subdomain. Therefore, we just assume that >2 legal moves are inside the testing domain. In other words, “>2 legal moves” is the representative of from the subdomain (e.g. 10, 100, 1000……).



Our test cases cover 8 situations

1. There is only one legal move from land A to land B

2. There is only one legal move from land B to land A

3. There is two legal moves from land B to land A

4. There is two legal moves from land A to land B

5. There is >2 legal moves from land A to land B

6. There is >2 legal moves from land B to land A

7. There is no legal move from land A to land B

8. There is no legal move from land B to land A

Example:

This is a test case simulating there are >2 legal move from land B to land A.

一張含有 文字 的圖片

自動產生的描述

Coverage:

一張含有 桌 的圖片

自動產生的描述

## Integration test of solve()

Since we have performed integration testing of the main function inside solve() which is getLegalMove(). Then, we can try to test the function to test the solution provided by this function.

There will be two combinations:

1. Return a valid solution
2. Return no solution

Example:

This is a test case simulating there is no solution.

一張含有 文字 的圖片

自動產生的描述

Coverage:

一張含有 文字 的圖片

自動產生的描述

(\* We have also test the different difficulty to see if it function well)

# Testing Player Mode Undo/Redo Command

In player mode, player can perform undo & redo command respectively. For testing the functionality of those commands, we will consider the following test case:

1. Testing an allowed Undo command followed by a move

2. Testing an allowed Undo command followed by a redo command

3. Testing a disallowed Undo command without a move or a redo command

4. Testing an allowed Redo command followed by an undo command

5. Testing a disallowed Redo command without previous undo command

Idea behind:

1. Undo can only be performed if there is previous move and redo.
2. Redo can only be performed if there is previous undo.

一張含有 文字 的圖片

自動產生的描述一張含有 文字 的圖片

自動產生的描述

Since undo () and redo() function is depend of the return value of canUndo() and canRedo() respectively. Therefore, we will test the value returned by those functions.

Example:

This is a test case simulating a Redo command is allowed followed by an undo command.

一張含有 文字 的圖片

自動產生的描述

Coverage:

一張含有 文字 的圖片

自動產生的描述

# **System Test**

## Solver Mode

Test Case for Solver Mode System Testing:

一張含有 文字 的圖片

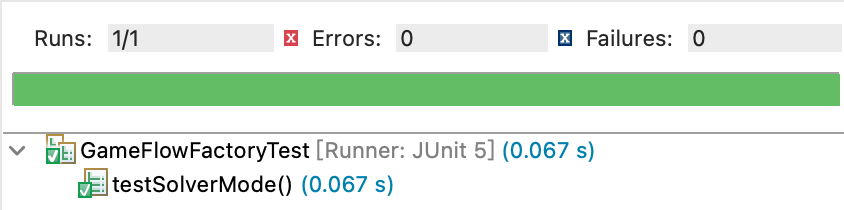
自動產生的描述

Actual output in console:

一張含有 文字, 收據 的圖片

自動產生的描述

Coverage:



## Player Mode

Since player mode is difficult to perform system testing since it must keep accepting player’s input and command in the process. However, the logic behind is same as solver mode. The only difference is that solver mode generate solution automatically, while player mode is manually generated solution by user. As we have tested the logical part by performing unit testing and integration testing separately above. Therefore, we assume solver mode testing is enough to be our system testing.

# User acceptance testing (UAT)

In UAT, we tested our program on different operation system. The result are as

follows:

Linux platform:

一張含有 文字 的圖片

自動產生的描述

Window platform:

一張含有 文字 的圖片

自動產生的描述

Ios platform:

一張含有 桌 的圖片

自動產生的描述