Test Plan for Assessment 2

Test Considerations

Introduction

Testing is a critical part of any software release. This Test Plan document details any testing we have done whether functional or nonfunctional. Generally have used unit tests on critical high risk classes and their methods, black-box testing on the game as a whole and other specifically designed tests where relevant. As we have used *AGILE* Development, we have tested the relevant places at the end of each of our sprints.

Test Coverage

System critical methods, particularly ones of where a large amount of computation occurs will be tested thoroughly.

Third-party libraries and in-built Java functionality will assumed effectively stable and therefore will not be tested. We have deliberately chosen third-party libraries that are very widely used in the hope that they will have very few bugs, and none that are critical to system functionality.

Basic methods, i.e. accessors/mutators and those which simply return a value, will be given a low precedence in testing, this is because they are simple to understand straight from the source code and are therefore unlikely to harbour any serious bugs that will affect critical system functionality.

Testing Environments

Java is a flexible programming language and can be run on a very wide range of systems. One of our project requirements is that the project runs on the computers in the Department Labs, therefore all whole system tests- such as black-box testing on the whole project will be run on these to ensure correct/expected functionality when run on these computers

Other testing, such as unit testing, that examines source code may be run on other systems, as the core reason for these tests is to ensure the correct code at low-level rather than functionality of the system as a whole.

Testing Methods

We have used a wide variety of tests and testing methods within our software testing to ensure our game is free from errors. These testing methods have included:

Unit Testing

We isolated the whole program into separate testable units to see if they were fit for use and whether they corresponded to the design specifications. We built test classes which instantiated the objects with specific input data and called the various methods belonging to the unit class and printed the outputs. By verifying that the output of each method corresponded to the expected value, we were able to prove that our class units worked as intended.

Black Box Testing

Brief example of how black box testing has been used- forming a range of test cases, testing them and the results.

System Testing

We will use system testing to test the system works as a whole. This will be measured against the start

Usability Testing

We put the game in front of a completely new user with the game manual to test if they could easily understand it. We set some initial test instructions for the user and some test conditions to decide how we would measure if the test was passed.

Acceptance Testing

We haven't run any acceptance testing for the project as yet, this is because the project implementation is still only in it's early stages, and we'd like to have more of the features from the brief before testing with the client.

Test Design

Test 1: Unit Test for Train Class

Test Conditions

The test shall be run on a laptop running windows 7. The unit test will be run on eclipse with a custom written test class.

Method

A test class will be written that instantiates an instance of a train at junction 0, moves it to junction 37, upgrades the train, breaks the train, and repairs the train. In between each of these actions it outputs the results of these changes.

Test

Test	Expected Result
Engine types remain throughout trains life	Remain constant despite game progression
New train can be created	Train is instantiated at junction
Upgrade a train's tier	Train tier changed from 1 to 2

Test 2: Black Box Test for Train Class

Test Conditions

The game will be run from the executable .jar file on a standard department lab PC running Windows 7.

Method

The game as a whole will be run, and we will test to see that the player's train is instantiated and displayed by the GUI.

Test

Test	Expected Result
Train is instantiated by the Game Engine when game starts.	Train appears on map at starting station.

Test 3: Unit Test for Player Class

Test Conditions

The test will be run on a laptop running Windows 7, in Eclipse with JDK 7.

Method

A brief test class is written that instantiates 2 players and calls several of its methods.

The Methods of the Player class that will be tested by our test class are –

- Player(int playerID, int homeStationID) //CONSTRUCTOR
- getPlayerScore()
- increasePlayerScore(int numPoints)
- getPlayerWealth()
- increasePlayerWealth(int numWealth)
- getPlayerTrains()
- buyNewTrain(int cost, int engineType, int ownerID, int trainID, int faultRate)

Test

Test	Expected Result
Instantiate 2 players	2 Players are instantiated
Get a player's current score	An integer representing a player's score (0 on instantiation)
Get a player's current wealth	An integer representing a player's wealth (0 on instantiation)
Increase a Player's score (for instance, by 2000)	The method is successfully executed and shows the desired increase (increase in score is tested by executing g*etPlayerScore method)*
Increase a Player's wealth (for instance, by 2000)	The method is successfully executed and shows the desired increase (increase in wealth is tested by executing g*etPlayerScore method)*
Buy a new train of cost less than player's wealth	The method is successfully executed and shows the desired changes. (Train is successfully added to player's trainlist

	and a consequent decrease in player's wealth is reflected)
Get a player's current list of trains	A list representing the player's list of train objects
Buy a train with cost more than a player's wealth	An error message should be displayed an no train should be added to player's train list and his wealth should remain the same

Test 4: Black Box Test for Player Class

Test Conditions

The game will be run from the executable .jar file on a standard department lab PC running Windows 7.

Method

The game as a whole will be run and we will test for the player's score and wealth at the start of the game and the player's score and wealth after the completion of a goal. This also shows that the GoalEngine class can recognise completed goals.

Test

Test	Expected Result
A Player should not have any wealth or any score at the start of the game.	The wealth and score tabs of the Player info section is both be set to 0 when the players are initialized.
Player's wealth and score should increase after completion of a goal	The wealth and score awarded by the goal is added to the player's wealth and score

Test 5: Black Box Test for MapGraph

Test Conditions

The test will be run on a laptop running Ubuntu 14.04 using the executable .jar file.

Method

To ensure that the methods in MapGraph work as intended, we have written a brief test class that instantiates an instance of MapGraph and calls several of it's methods with some test data, and prints the outputs.

The methods that will be tested by our test class are as follows:

- MapGraph.CreateMapArray() (tested through correct instantiation of the class)
- MapGraph.GetJunctionList(String file) (tested through correct instantiation of the class)
- MapGraph.AddTrain(int trainID, int location)
- MapGraph.MoveTrain(int trainID, int location, int destination)
- Junction.GetConnectedJunctions()
- Junction.GetTrains()
- Junction.IsPresent(Integer TrainID)
- Junction.FindNext(int destination)

Test

Test	Expected Result
Instantiate MapGraph	MapGraph object is successfully instantiated
Add train (ID 0) to junction (ID 0) on map	Train successfully added to junction
Get the next stop from junction 0 with a destination of juncton 1	The ID 37 should be returned
Get all connected junctions and stations from junction 0	The IDs 1 and 2 should be returned
Get list of trains at junction 0	The ID 0 should be returned
Move train from junction 0 towards junction 1	Train moved from junction 0 to junction 37 (towards junction 1)
Remove train (ID 0) from junction (ID 37) on map	Train 0 successfully removed from junction 37

Test 6: Black Box Test for GoalEngine

Test Conditions

The test will be run on a laptop running Ubuntu 14.04 using the executable .jar file.

Method

The game will be played as a whole and we will test for the goals being displayed properly upon the game opening. We will also test for the removal of a goal at the end of a turn, and it's subsequent replacement. If each goal is replaced after it is removed, then the game will always have 3 active goals.

Test

Test	Expected Result
Goal replacement after a removal	New goal is generated at the end of a turn to replace an existing one. New goal is also generated after one is completed
Goal descriptions are displayed correctly	Goal descriptions are displayed in the format: "Goal _num_: Get to _Station_ for $_num_ and _num_ exp."

Test 7: Black Box Test for Train Movement

Test Conditions

The game will be run from the executable .jar file on a standard department lab PC running Windows 7.

Method

We will run the game as a whole and we will test that the train movement works as intended, this will be achieved by attempting to move the train via a variety of valid and invalid methods and ensuring that the game deals with them in the intended way.

Test

Test	Expected Result
Attempt to move train further than it can move in one turn (Currently 2 spaces)	Train moves along track as far as it is capable and stops
Attempt to move train to a station closer than it's maximum movement allowance (Currently 2 spaces)	Train moves to the destination and stops without using up the rest of it's allowance
Attempt to move train without selecting a train	Prompt appears asking the player to select a train to move
Attempt to move train without selecting a destination	Prompt appears asking the player to select a destination
Attempt to move a train to a destination that passes a junction or station	Prompt appears informing the player that an invalid move was attempted
Attempt to move a train to a checkpoint	Prompt appears informing the player that an invalid move was attempted

Test 8: System Testing for Whole Game

Test Conditions

The test will be run on the standard lab setup in the department labs as one of our requirements is for the system to work on these machines. The machines will be booted into Windows 7 and the game run from the executable file.

Method

The game will be run as a whole from the executable .jar file, we will test that stations and junctions turn blue to indicate that they have been selected. We will then test the selection of checkpoints to ensure that they cannot be selected as destinations. Trains will be selected to ensure that a black border is added to them, we will ensure that the train's icon is moved upon the train itself being moved.

Test

Test	Expected Result
The GUI is expected to display an image of the map, with junctions, goals, and player information all displayed in their initial state.	The GUI will display an image of the map, with junctions, goals, and player information all displayed in their initial state.
Stations and junctions should be selectable.	Stations and junctions should turn blue when they are clicked on, to represent them being selected.
Checkpoints (junctions which have only two connected points) should not be selectable.	Checkpoints should not change colour when they are clicked on.
Trains should be selectable.	Trains will gain a black border upon being clicked to represent being selected.
When given a valid move for a train, the GUI should move the train icon.	Train icon moves.

Test 9: Usability Testing for the Whole System

Test Conditions

The test will be run on the standard lab setup in the department labs as one of our requirements is for the system to work on these machines. Developers will not be present in order to ensure that the participants are not given hints either deliberately or inadvertently. All efforts will be made to ensure that the users are acting independently at all times.

Method

1. Give a pair of new users the game manual to read.
2. Open the game for the users- we are testing usability of the game, not the users' abilities to open an executable file.
3. Ask the users to race to complete a goal.
4. Tell users they may ask for help if needed.
5. Once a goal has been completed, ask the users to answer a few simple questions to suggest what difficulties they came across.

Test

Test	Expected Result
Users manage to move a train by reading the game manual. The test is passed if no external help is required.	No external help is required.
Users manage to swap places at the end of player 1's turn. The test is passed if no external help is required.	No external help is required.
One user manages to complete a goal. The test is passed if no external help is required.	No external help is required.

Test Results

Test 1: Unit Test for Train Class

Results

Test	Result	Proof of Result
Engine types remain throughout trains life	Values remained correct and valid	See Fig. 1 Using constants ensures validity of such values
New train can be created	Train was instantiated at the correct junction	See Fig. 2 - 3
Upgrade a train's tier	Train tier changed from 1 to 2	See Fig. 2 - 3

Screenshot Proofs

```
Create a new train: in Lisbon (Junction ID 0)
Before move, train is at: 0
After move, (to adjacent checkpoint (Junction ID 37)) train is at: 37
Train tier:1
Train tier after upgrade: 2
Faulty is:
false
true
false
```

Fig. 1

```
5  public class Train {
6      //INSTANCE CONSTANTS
7      //ENGINE_TYPE: 1 -Electric, 2 -Diesel, 3 -Flying
8      private final int ENGINE_TYPE;
9
10     private final int OWNER_ID;
11     private final int TRAIN_ID;
12
```

Fig. 2

```
5
6   public static void main(String[] args) {
7       // TODO Auto-generated method stub
8       System.out.println("Create a new train: in Lisbon (Junction ID 0)");
9       Train testTrain = new Train(1,1,3,0,0);
10      System.out.print("Before move, train is at: ");
11      System.out.println(testTrain.getCurrentJunction());
12
13      testTrain.moveTrain(37);
14      System.out.print("After move, (to adjacent checkpoint (Junction ID 37)) train is at: ");
15      System.out.println(testTrain.getCurrentJunction());
16      //Will move train to any junction ID this is validated by Mapgraph, not locally.
17
18      System.out.print("Train tier:");
19      System.out.println(testTrain.getTier());
20      testTrain.upgradeTrain();
21      System.out.print("Train tier after upgrade: ");
22      System.out.println(testTrain.getTier());
23      // Any train can be upgraded, be it Diesel, Electric or Flying
24
25      System.out.println("Faulty is: ");
26      System.out.println(testTrain.isFaulty());
27      testTrain.breakTrain();
28      System.out.println(testTrain.isFaulty());
29      testTrain.repairTrain();
30      System.out.println(testTrain.isFaulty());
31      // Breaks and repairs train
```

Fig. 3

Test 2: Black Box Test for Train Class

Results

Test	Result	Proof of Result
Train is instantiated when game starts	Train appeared on map at correct station.	See Fig. 4

Screenshot Proofs

Fig. 4: Train is visible on Kiev Station.

Test 3: Unit Test for Player Class

Results

Test	Result	Proof of Result
Instantiate 2 players	2 Players are instantiated	The absence of errors on building and the ability to call the method *getPlayerScore* of the two player objects indicates that the both objects were instantiated correctly See (Fig. 5)
Get a player's current score	An integer representing a player's score (0 on instantiation)	As shown in (Fig. 5), a player's score is an integer (on instantiation it is 0)
Get a player's current wealth	An integer representing a player's wealth (0 on instantiation)	As shown in (Fig. 5), a player's wealth is an integer (on instantiation it is 0)
Increase a Player's score (for instance, by 2000)	The method is successfully executed and shows the desired increase (increase in score is tested by executing g*etPlayerScore method)*	As shown in (Fig. 5), the
Increase a Player's wealth (for instance, by 2000)	The method is successfully executed and shows the desired increase (increase in wealth is tested by executing g*etPlayerScore method)*	As shown in (Fig. 5), the increased wealth is 2000
Buy a new train of cost less than player's wealth	The method is successfully executed and shows the desired changes. (Train is successfully added to	As shown in (Fig. 5), Player 1 bought 3 trains of cost 1000,500 and 250 and the trains are added

	player's trainlist and a consequent decrease in player's wealth is reflected)	to his Trainlist and his resulting wealth is 250.
Get a player's current list of trains	A list representing the player's list of train objects	As shown in (Fig. 5), the player's list contains 3 trains that were bought by him
Buy a train with cost more than a player's wealth	An error message should be displayed an no train should be added to player's train list and his wealth should remain the same	As shown in (Fig. 5 & 6), a message "You don't have enough money to buy this train!" was displayed and player's current trainlist and wealth remained the same

Screenshot Proofs

Fig. 5 - Output from Unit Test

Console | PlayerUnitTest.java

```
Initialize Player 1 & Player 2
Player 1's Score : 0
Player 2's Score : 0

Player 1's Wealth : 0
Player 2's Wealth : 0
After Increase -
Player 1's Score : 2000
Player 1's Wealth : 2000

After buying 3 trains -
Player 1's Train List : [com.SEPR.game.Train@1db9742, com.SEPR.game.Train@106d69c, com.SEPR.game.Train@52e922]
Player 1's Wealth : 250
```

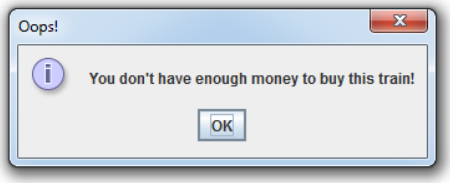


Fig. 6 - Error message from buying a train without required wealth.

Console | PlayerUnitTest.java

```
1  package com.SEPR.game;
2
3  public class PlayerUnitTest {
4      public static void main(String[] args) {
5          //Test 1 - Instantiate 2 Players
6          System.out.println("Initialize Player 1 & Player 2");
7          Player player1 = new Player(1,1);
8          Player player2 = new Player(2,1);
9
10         System.out.println("Player 1's Score : " + player1.getPlayerScore());
11         System.out.println("Player 2's Score : " + player2.getPlayerScore() + "\n" );
12
13         System.out.println("Player 1's Wealth : " + player1.getPlayerWealth());
14         System.out.println("Player 2's Wealth : " + player2.getPlayerWealth());
15
16         player1.increasePlayerScore(2000);
17         player1.increasePlayerWealth(2000);
18
19         System.out.println("After Increase -");
20         System.out.println("Player 1's Score : " + player1.getPlayerScore());
21         System.out.println("Player 1's Wealth : " + player1.getPlayerWealth() + "\n" );
22
23         //Player 1 with Wealth 2000 buys 3 trains of cost 1000,500,250
24         player1.buyNewTrain(1000, 1, 1, 0, 0);
25         player1.buyNewTrain(500, 1, 1, 0, 0);
26         player1.buyNewTrain(250, 1, 1, 0, 0);
27
28         System.out.println("After buying 3 trains - ");
29         System.out.println("Player 1's Train List : " + player1.getPlayerTrains());
30         System.out.println("Player 1's Wealth : " + player1.getPlayerWealth() + "\n");
31
32         //Player 1 with Wealth 250 buys a train of cost 1000
33         player1.buyNewTrain(1000, 1, 1, 0, 0);
34         System.out.println("After failing to buy a train -");
35         System.out.println("Player 1's Train List : " + player1.getPlayerTrains());
36         System.out.println("Player 1's Wealth : " + player1.getPlayerWealth());
37     }
38 }
```

Fig. 7 - The Unit Test File for this test

Test 4: Black Box Test for Player Class

Results

Test	Result	Proof of Result
A Player should not have any wealth or any score at the start of the game.	The wealth and score tabs of the Player info section are both be set to 0 when the players are initialized.	As shown in Fig. 1 and 2, wealth and money are set to 0 at start of the game.
Player's wealth and score should increase after completion of a goal	The wealth and score awarded by the goal is added to the player's wealth and score	As shown in Fig. 3 & Fig. 4, Player 2 completed the goal of taking his train to Madrid and the desired increase in his wealth and score is reflected.

Screenshot Proofs

Fig. 8

Fig. 9

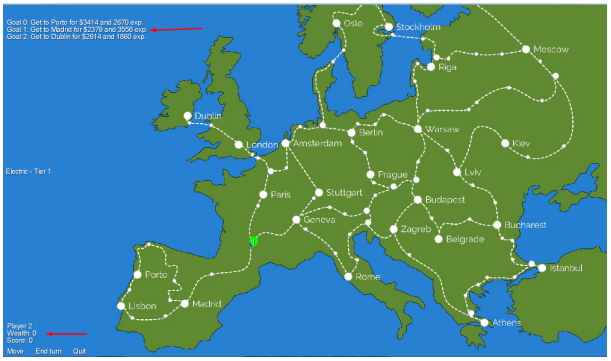


Fig. 10

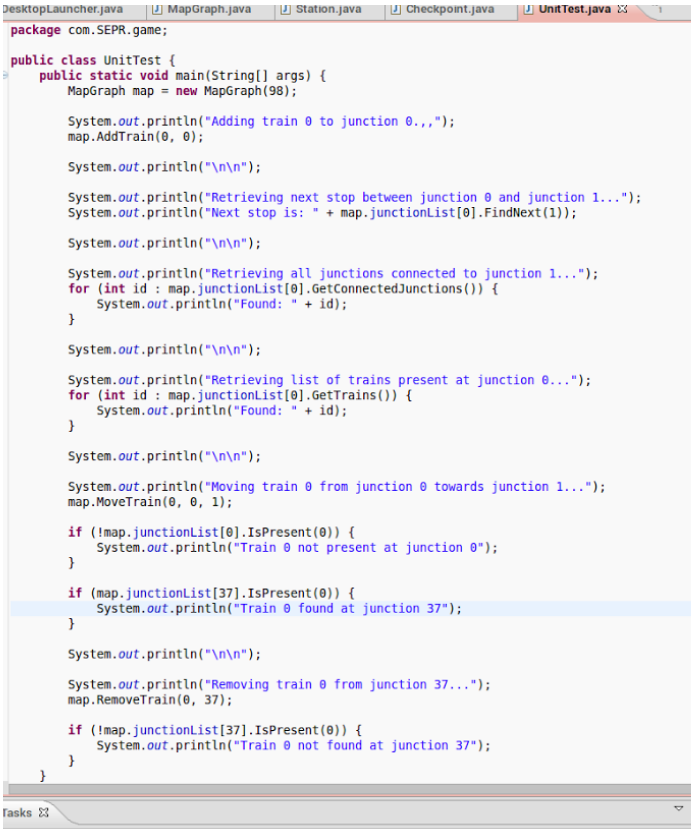
Fig. 11

Test 5: Unit Test for MapGraph

Results

Test	Result	Proof of Result
Instantiate MapGraph	MapGraph object is successfully instantiated	The absence of errors on building and executing, and the ability to call methods from the instantiated method indicated that the object was instantiated correctly
Add train (ID 0) to junction (ID 0) on map	Train was successfully added to junction	The output of junction.GetTrains() indicates that the train was added correctly
Get the next stop from junction 0 with a destination of juncton 1	The ID 37 was returned	See output of test class
Get all connected junctions and stations from junction 0	The IDs 1 and 2 were returned	See output of test class
Get list of trains at junction 0	The ID 0 was returned	See output of test class
Move train from junction 0 towards junction 1	The train was moved successfully from junction 0 to junction 37	The output of the second junction.GetTrains() indicates that the train was moved out of junction 0, and the output of the third junction.GetTrains() indicates that the train was moved to junction 37
Remove train (ID 0) from junction (ID 37) on map	The train 0 was successfully removed from junction 37	The output of junction.IsPresent(0) indicates that the train was removed correctly

Screenshot Proofs




```java
package com.SEPR.game;

public class UnitTest {
    public static void main(String[] args) {
        MapGraph map = new MapGraph(98);

        System.out.println("Adding train 0 to junction 0.,,");
        map.AddTrain(0, 0);

        System.out.println("\n\n");

        System.out.println("Retrieving next stop between junction 0 and junction 1...");
        System.out.println("Next stop is: " + map.junctionList[0].FindNext(1));

        System.out.println("\n\n");

        System.out.println("Retrieving all junctions connected to junction 1...");
        for (int id : map.junctionList[0].GetConnectedJunctions()) {
            System.out.println("Found: " + id);
        }

        System.out.println("\n\n");

        System.out.println("Retrieving list of trains present at junction 0...");
        for (int id : map.junctionList[0].GetTrains()) {
            System.out.println("Found: " + id);
        }

        System.out.println("\n\n");

        System.out.println("Moving train 0 from junction 0 towards junction 1...");
        map.MoveTrain(0, 0, 1);

        if (!map.junctionList[0].IsPresent(0)) {
            System.out.println("Train 0 not present at junction 0");
        }

        if (map.junctionList[37].IsPresent(0)) {
            System.out.println("Train 0 found at junction 37");
        }

        System.out.println("\n\n");

        System.out.println("Removing train 0 from junction 37...");
        map.RemoveTrain(0, 37);

        if (!map.junctionList[37].IsPresent(0)) {
            System.out.println("Train 0 not found at junction 37");
        }
    }
```

Fig. 12

Fig. 13

Test 6: Black Box Test for GoalEngine

Result

Test	Result	Proof of Result
Goal replacement after a removal	New goal is generated at the end of a turn to replace the existing one. A new goal is also generated after one is completed	See Fig. 14 & 15 for before replacement and after replacement respectively
Goal descriptions are displayed correctly	Goal descriptions are displayed in the format: "Goal _num_: Get to _Station_ for $_num_ and _num_ exp."	See Fig. 14 for goal descriptions being displayed

Screenshot Proofs

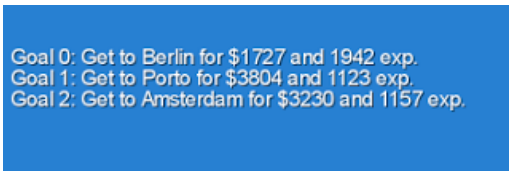


Fig. 14

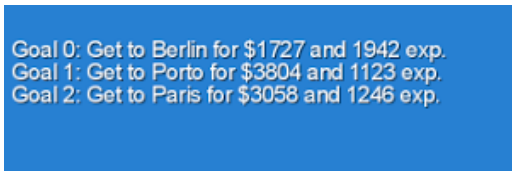


Fig. 15

Test 7: Black Box Test for Train Movement

Result

Test	Result	Proof of Result
Attempt to move train further than it can move in one turn (Currently 2 spaces)	Train only moves along the path as far as it is capable	See: Fig. 1&2, before move and after move respectively
Attempt to move train to a station closer than it's maximum movement allowance (Currently 2 spaces)	Train only moves to the destination and stops without using up the rest of it's allowance	See: Fig. 3&4, before move and after move respectively
Attempt to move train without selecting a train	Prompt appears asking the player to select a train to move	See: Fig. 5&6, before attempt and after attempt respectively
Attempt to move train without selecting a destination	Prompt appears informing the player that an invalid move was attempted	See: Fig. 7&8, before attempt and after attempt respectively
Attempt to move a train to a destination that passes a junction or station	Prompt appears informing the player that an invalid move was attempted	See: Fig. 9&10, before attempt and after attempt respectively
Attempt to move a train to a checkpoint	Prompt appears informing the player that an invalid move was attempted	See: Fig. 10&11, before attempt and after attempt respectively

Screenshot Proofs

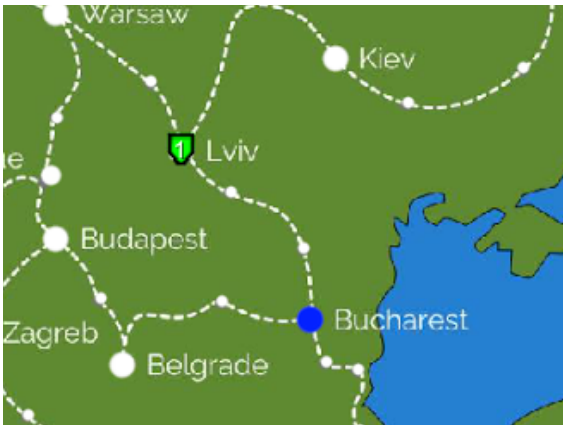


Fig. 16

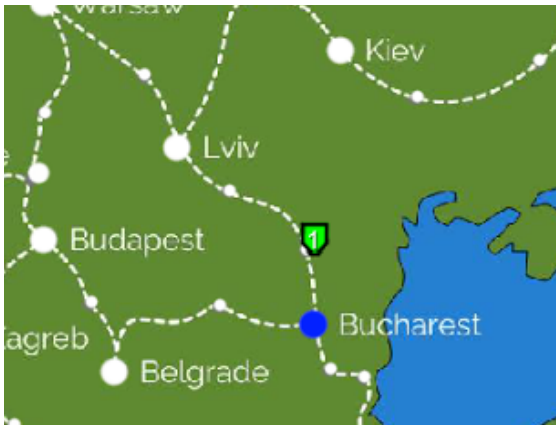


Fig. 17

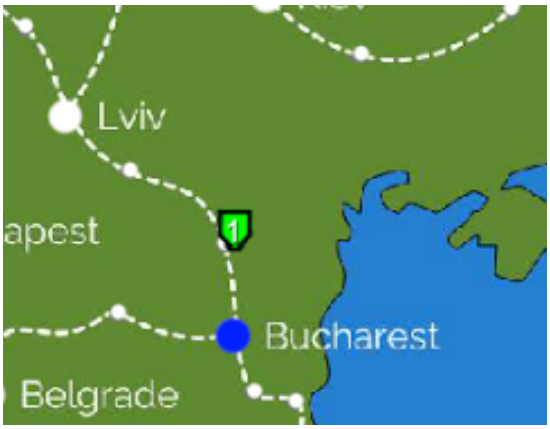


Fig. 18

Fig. 19

Fig. 20

Fig. 21

Fig. 22

Fig. 23

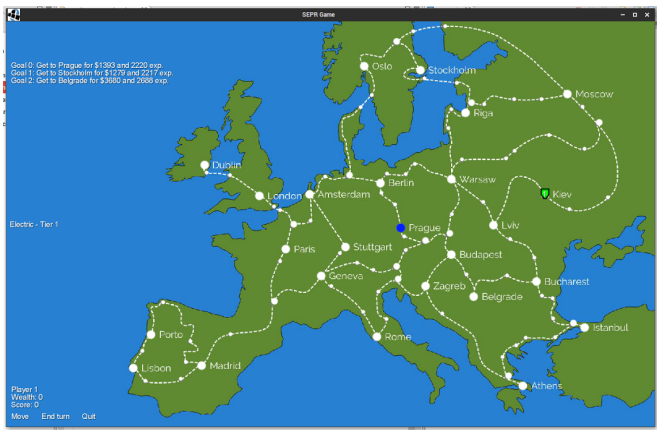


Fig. 24

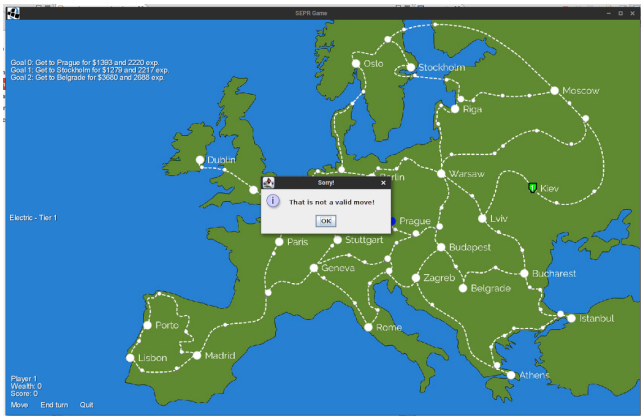


Fig. 25

Fig. 26

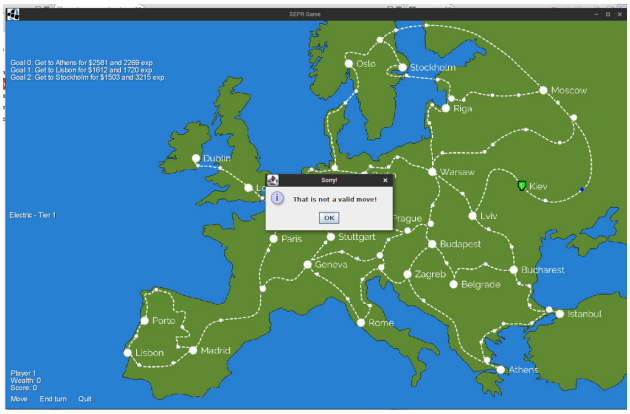


Fig. 27

Test 8: System Testing for Whole Game

Results

Test	Result	Proof
The GUI is expected to display an image of the map, with junctions, goals, and player information all displayed in their initial state.	The test matches the intended appearance of the map from our testing predictions. Working as intended.	See Fig. 28
Stations and junctions should be selectable.	Stations and junctions both turned blue when clicked. Working as intended.	See Fig. 29 & 30
Checkpoints (junctions which have only two connected points) should not be selectable.	The checkpoint was selectable and turned blue when clicked.	See Fig. 31
Trains should be selectable.	The train gained a black border around when it was clicked. Working as intended.	See Fig. 32
When given a valid move for a train, the GUI should move the train icon.	The train moves on the screen.	See Fig .32 & 33

Screenshot Proofs

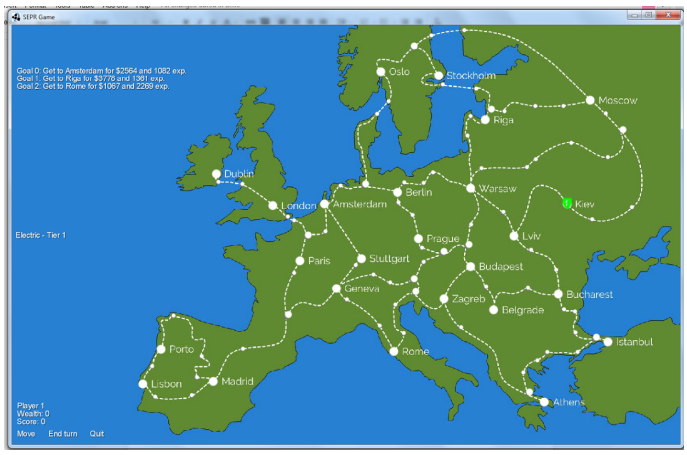


Fig. 28

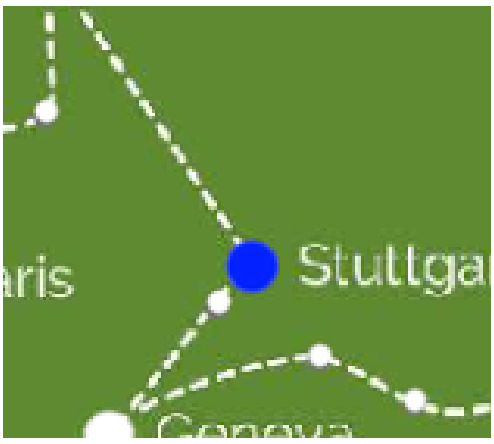


Fig. 29

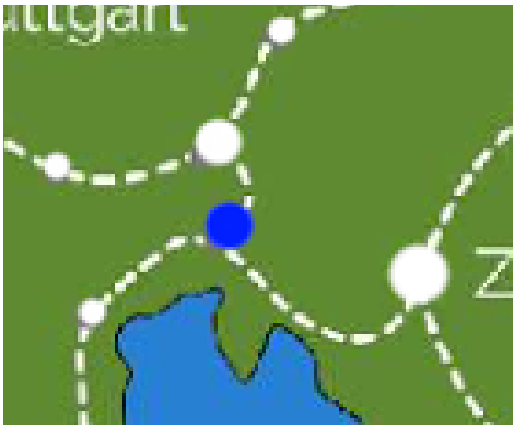


Fig. 30

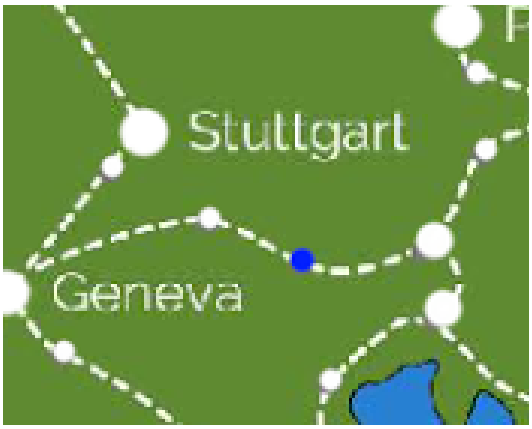


Fig. 31

Fig. 32

Fig. 33

Bug Fixes

Due to the implementation of the Checkpoints in the GUI, it was excessively complex to remove the event listeners from them without affecting the other Junctions. We have instead opted to include a check within the GameEngine class to prevent Checkpoints from being used as destinations, a prompt appears to the player informing them of an invalid move when a Checkpoint is attempted to be used as a destination.

Test 9: Usability Testing for the Whole System

Results

Test	Result	Proof of Result
Users manage to move a train by reading the game manual. The test is passed if no external help is required.	Pass.	No external help was required.
Users manage to swap places at the end of player 1's turn. The test is passed if no external help is required.	Pass.	No external help was required.
One user manages to complete a goal. The test is passed if no external help is required.	Pass.	No external help was required.

Notes

Participants asked whether they are meant to play the game together. It was clarified to them that they were meant to play against each other. As this was a question about the test rather than the game, it was not deemed that this was a significant enough query to cause the failure of the test.

Additional Comments from Participants

Rules seem more complicated than actually doing it- perhaps the user manual is slightly too formal.

Summary at the bottom of the user manual?

Goals change perhaps too quickly!

Buttons in the bottom are easy to use, but perhaps difficult to find without the user manual.