Test Plan for Assessment 3

Test Considerations

Introduction

Testing is a critical part of any software release. This Test Plan document details any testing we have done for this assessment, whether functional or nonfunctional. Generally we have used unit tests on new classes, class extensions and their methods, and black-box testing on the game as a whole and other specifically designed tests where relevant. As we have used a form of test-driven *AGILE* Development, we have tested all relevant places throughout the implementation, and we have ensured that each stage of the project is fully working as intended at the end of each of our sprints.

Test Coverage

Clearly the original parts of the project have been well tested in the previous assessment by Team HEC as detailed by their testing document. Any requirements that have passed their tests will be assumed as working at the start of this assessment. To utilise time as efficiently as possible we will therefore not **re-test** any of these requirements, unless we are modifying the classes or methods that are specifically fulfilling these requirements.

All Classes will be **thoroughly** tested using both Unit Tests and black box testing to ensure that they are functioning correctly and robustly.

Accessors and mutators will be given a **low precedence** in testing are unlikely to harbour any serious bugs that will affect critical system functionality.

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.

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 using JUnit which instantiated the objects with specific input data and called all relevant methods belonging to the unit class and tested their outputs against expected outputs using JUnit assertions. We have decided to make use of JUnit due to it's wide use in testing applications, it allows us to make many assertions about the functionality of a class and check to ensure that the assertions remain true. By verifying that the output of each method corresponded to the expected value (through the use of assertions), we were able to show that our class units worked as intended. These tests were run near the end of this development cycle to ensure that the code was all working in its finished state.

To run the JUnit tests, the author of each section of code wrote tests and ran them using eclipse's built in JUnit functionality as it gave useful feedback about not only which tests failed but also why they failed

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 performed by the developers playing the game, they will press all buttons and ensure that

everything works as intended and the system is robust enough to handle unexpected button clicks and events.

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

Due to time constraints in this very short assessment run, we haven't been able to run any acceptance testing for the project as yet. This is not a major issue as we do not yet have a final product anyway.

Test Design & Results

Unit Tests for CargoGoal Class

A brief test class is written using JUnit that instantiates a Special Cargo Goal and calls several of its methods and the expected outputs were matched against real time outputs. All relevant methods and instance data members associated with CargoGoal Class have been tested via JUnit so coverage is extensive.

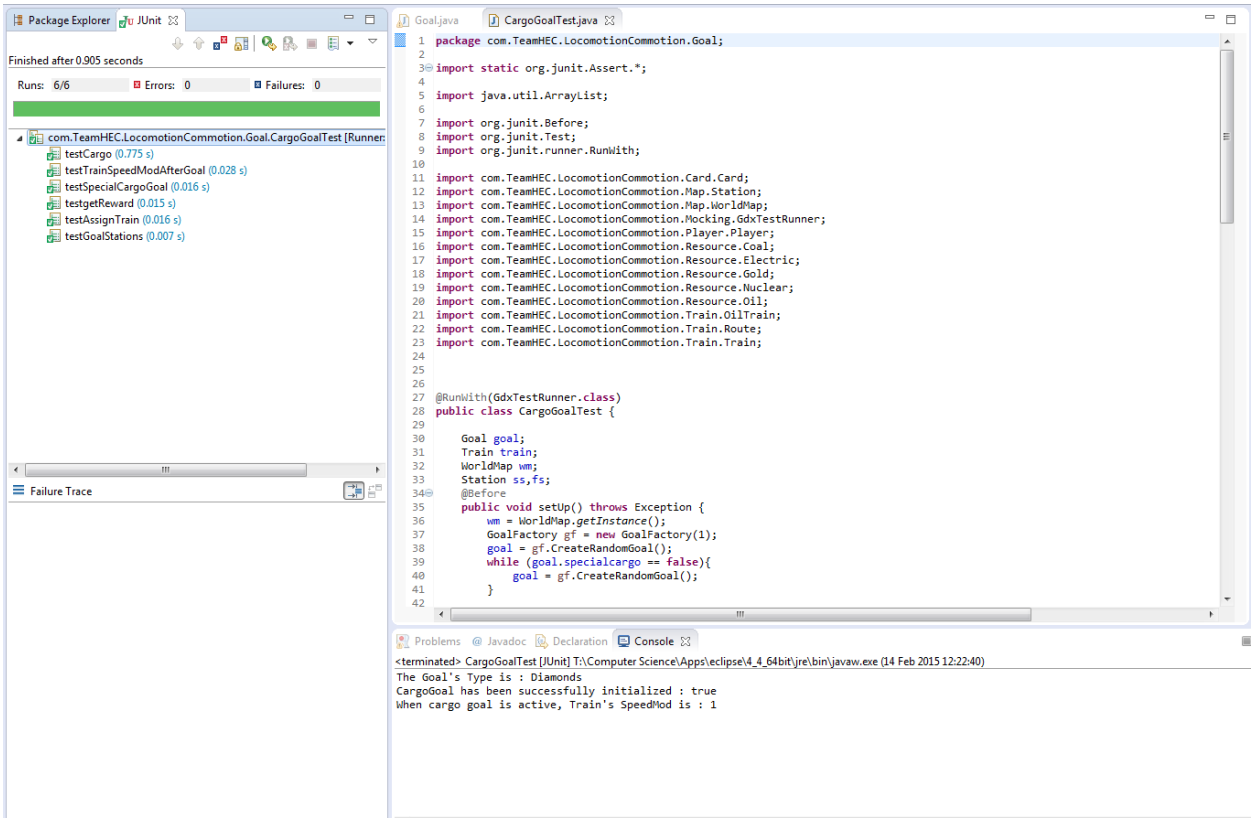
The Methods & Data members of the CargoGoal class that we tested are –

- boolean **specialcargo**
- String **getCargo()**
- String **getSStation()**
- String **getFStation()**
- void **assignTrain()** & Train **getTrain()**
- int **getReward()**
- void **goalComplete()**

Note: We found an error/bug where the train's speedmod won't reset to the original value after Special Cargo Goal Completion and fixed it.

Test Description	**Expected Result**	**Result**	**Proof of Result**	**Status**
Instantiate Special Cargo Goal and test the value of *'specialcargo'*	Special Cargo Goal is successfully instantiated and *'specialcargo'* is set to true to confirm it	Special Cargo Goal is successfully instantiated and *'specialcargo'* is set to true to confirm it	The absence of errors on building and the assertion *'specialcargo'* being true via JUnit test [testSpecialCargo Goal()]	Pass
Get the cargo type for Special Cargo Goal	A string 'Diamonds' is returned	A string 'Diamonds' is returned	Assertion of the string 'Diamonds' via JUnit test [testCargo()]	Pass
Check if the Special Cargo Goal has a start station	True	True	Assertion via Junit test [testGoalStations( )]	Pass

Check if the Special Cargo Goal has a finish station	True	True	Assertion via JUnit test [testGoalStations( )]	Pass
Get the reward for the Special Cargo Goal	An integer greater than zero	An integer greater than zero	Assertion of a reward being returned which greater than zero (via Junit test)	Pass
Check if the CargoGoal has successfully been assigned to a player's train	The selected train should be assigned to the goal and there is a decrease in the SpeedMod of the assigned train	The selected train should be assigned to the goal and there is a decrease in the SpeedMod of the assigned train	Assertion via Junit test [testAssignTrain()] , the ability to call getTrain() method of the goal and the decreased speedmod value, as shown in Fig 1	Pass
Check if a special cargo goal can successfully be completed	Execution of method goalComplete() should be successful	Execution of method goalComplete() should be successful	The absence of errors on calling the method and assertion via JUnit test [testTrainSpeedModAfterGoal()]	Pass
Check if the speedMod of the player's train is restored back after cargo goal completion	SpeedMod is back to original value	SpeedMod is back to original value	Assertion via JUnit test [testTrainSpeedModAfterGoal()]	Pass



(Fig 1 - Running of JUnit Tests for CargoGoalTest class and some outputs)

Unit Test for GoalFactory Class

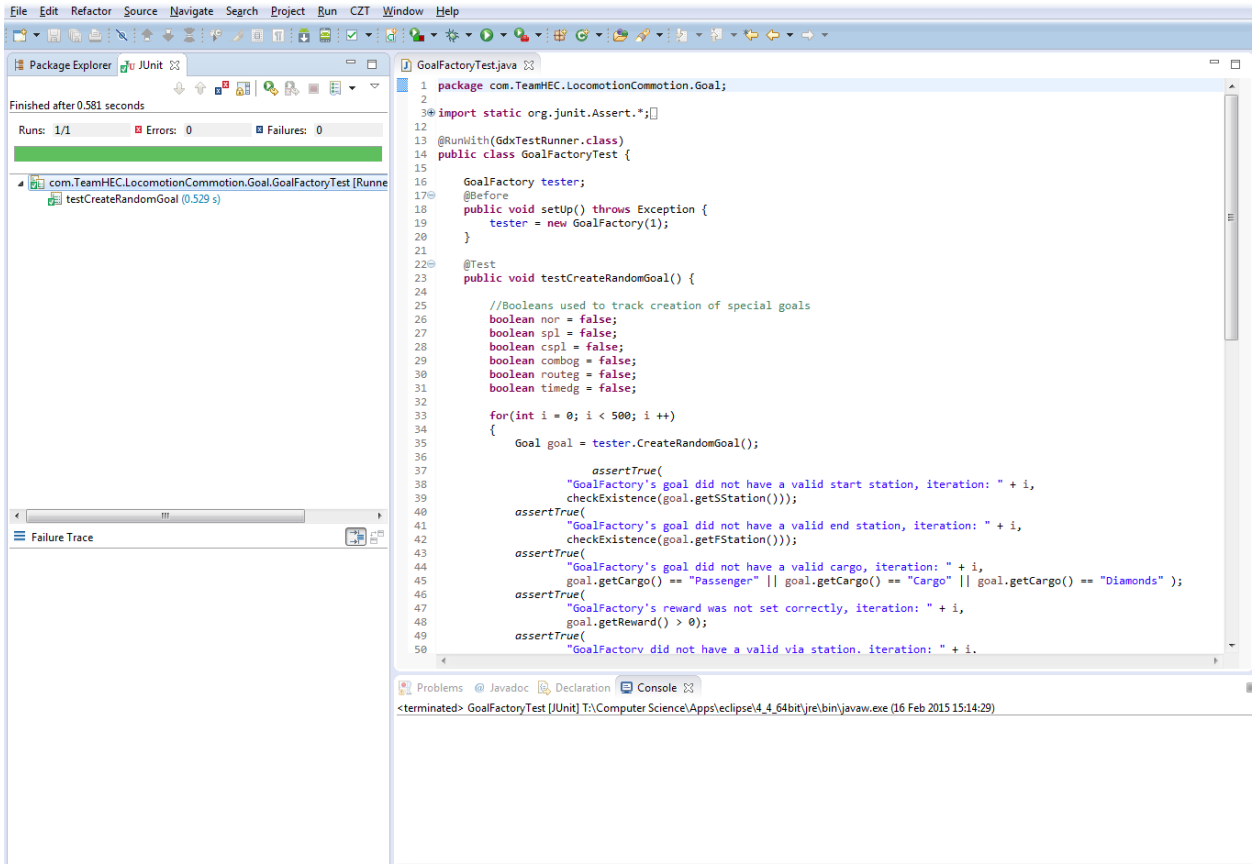
To ensure that the extended methods in Goal Factory class worked as intended, the JUnit test class for Goal Factory Class was extended and the extended methods were called with test data, and the outputs were verified via JUnit.

A series of 500 random goals were created via iteration in the JUnit test class and the following extended functionalities were tested.

Test Description	Expected Result	Result	Proof of Result	Status
Ability to create one or more non-special goals **(Func.Sys.2.2) (Func.Sys.2.3)**	One or more non-special goals are created by the GoalFactory	One or more non-special goals are created by the GoalFactory	Assertion via JUnit test [testCreateRandomGoal()]	Pass
Ability to create one or more Special goals **(Func.Sys.2.2) (Func.Sys.2.3)**	One or more non-special goals are created by the GoalFactory	One or more Special goals are created by the GoalFactory	Assertion via JUnit test [testCreateRandomGoal()]	Pass
Ability to create one or more Special CargoGoals **(Func.Sys.2.2) (Func.Sys.2.3)**	One or more non-special goals are created by the GoalFactory	One or more Special Cargo Goals are created by the GoalFactory	Assertion via JUnit test [testCreateRandomGoal()]	Pass
Ability to create one or more Route based goals **(Func.Sys.2.2) (Func.Sys.2.3)**	One or more Route based goals are created by the GoalFactory	One or more Route based goals are created by the GoalFactory	Assertion via JUnit test [testCreateRandomGoal()]	Pass
Ability to create one or more Timed based goals **(Func.Sys.2.2) (Func.Sys.2.3)**	One or more Timed based goals are created by the GoalFactory	One or more Timed based goals are created by the GoalFactory	Assertion via JUnit test [testCreateRandomGoal()]	Pass
Ability to create one or more	One or more Special Combo goals are created	One or more Special Combo goals are created	Assertion via JUnit test	Pass

Special Combo goals **(Func.Sys.2.2)**	by the GoalFactory	by the GoalFactory	[testCreateRandomGoal()]	
Test each goal to ensure a station isn't included that is currently unfixable (making the goal impossible to complete) **(User.GP.2.5)** **(Func.Sys.2.1)** **(Func.Sys.2.2)** **(Func.Sys.2.3)**	No goals are created with an unfixable station as either the start, via, or final stations.	No goals were created with an unfixable station as either the start, via, or final stations.	Assertion via JUnit test [testCreateRandomGoal()]	Pass

Note : All the above tests have been performed before 'score' class was added to the game design



(Fig 2 - Running of JUnit Tests for GoalFactoryTest class)

Unit Tests for TimedGoal Class

A small test class was written for the new TimedGoal class, that generates an instance of TimedGoal and tests each of its methods to ensure that they are operating as intended. All new methods relevant to TimedGoal were tested and their outputs were asserted using expected outputs through JUnit. All of the methods tested in the original GoalTest class were also implemented to extensive testing has been upheld for this class.

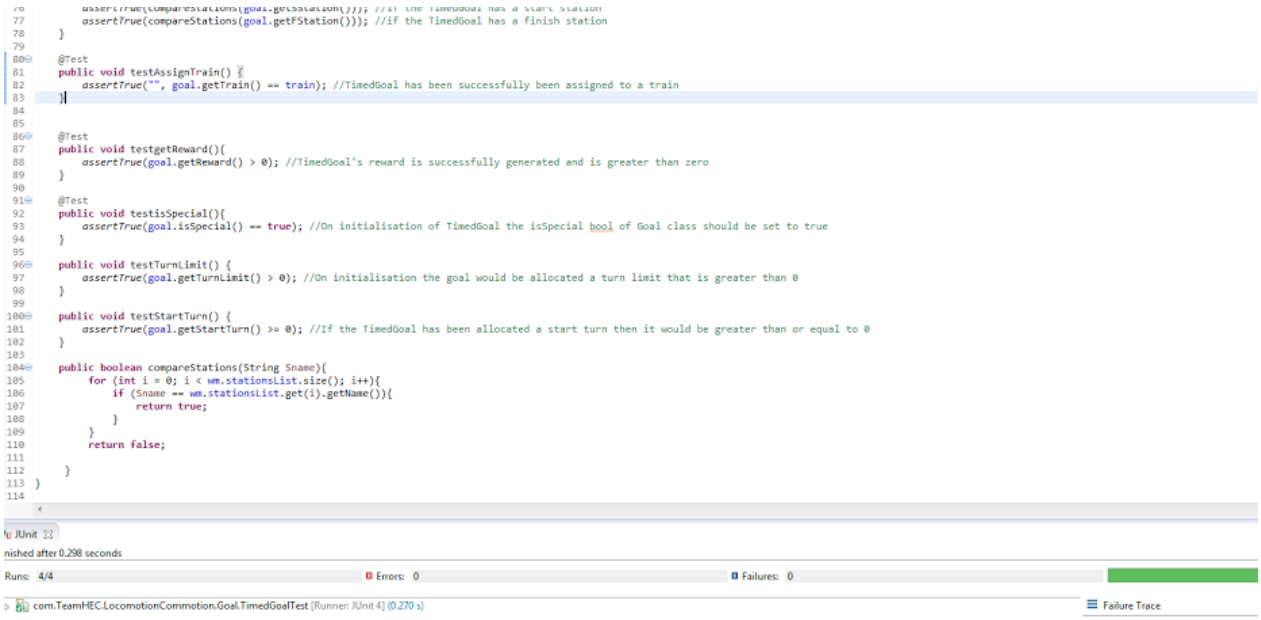
The Methods and Attributes of the Timed class that we tested are –

- Station **sStation**
- Station **fStation**
- Train **train**
- int **reward**
- boolean **isSpecial**
- int **turnLimit**
- int **startTurn**
- Station **getSStation()**
- Station **getFStation()**
- Train **getTrain()**
- int **getReward()**
- boolean **isSpecial()**
- int **getTurnLimit()**
- int **getStartTurn()**

Test Description	**Expected Result**	**Result**	**Proof of Result**	**Status**
Generates an instance of TimedGoal and test the value of isSpecial	isSpecial returns true	isSpecial returned true	The absence of errors upon instantiating TimedGoal and the assertion via JUnit	Pass

			[testIsSpecial()]	
Test the value of sStation	compareStations returns true	compareStations returned true	Assertion via JUnit [testGoalStations()]	Pass
Test the value of fStation	compareStations returns true	compareStations returned true	Assertion via JUnit [testGoalStations()]	Pass
Test the value of train	Train is correctly assigned to TimedGoal	Train was correctly assigned to TimedGoal	Assertion via JUnit [testAssignTrain()	Pass
Test the value of reward	reward should be greater than 0	reward was greater than 0	Assertion via JUnit [testgetReward()]	Pass
Test the value of turnLimit	turnLimit should be greater than 0	turnLimit was greater than 0	Assertion via JUnit [testTurnLimit()]	Pass
Test the value of startTurn	startTurn should be greater than or equal to 0	startTurn was greater than or equal to 0	Assertion via JUnit [testStartTurn()]	Pass

Note: After this test startTurn was changed to be 0 at the goal's creation and later changed to reflect the current turn counter once a train was assigned to it, at all points the startTurn is greater than or equal to 0.



Unit Tests for RouteGoalClass

A small test class was written for the new RouteGoal class, that generates an instance of RouteGoal and tests each of its methods to ensure that they are operating as intended. All new methods relevant to RouteGoal were tested and their outputs were asserted using expected outputs through JUnit. All of the methods tested in the original GoalTest class were also implemented to extensive testing has been upheld for this class.

The Methods and Attributes of the Timed class that we tested are –

- Station **sStation**
- Station **viaStation**
- Station **fStation**
- Train **train**
- int **reward**
- boolean **isSpecial**
- Station **getSStation()**
- Station **getViaStation()**
- Station **getFStation()**
- Train **getTrain()**
- int **getReward()**
- boolean **isSpecial()**

Test Description	Expected Result	Result	Proof of Result	Status
Generates an instance of RouteGoal and test the value of isSpecial	isSpecial returns true	isSpecial returned true	The absence of errors upon instantiating TimedGoal and the assertion via JUnit [testIsSpecial()]	Pass
Test the value of sStation	compareStations returns true	compareStations returned true	Assertion via JUnit [testGoalStations()]	Pass
Test the value of viaStation	compareStations returns true	compareStations returned true	Assertion via JUnit [testGoalStations()]	Pass
Test the value of fStation	compareStations returns true	compareStations returned true	Assertion via JUnit [testGoalStations()]	Pass
Test the value of train	Train is correctly assigned to TimedGoal	Train was correctly assigned to TimedGoal	Assertion via JUnit [testAssignTrain()	Pass

Test the value of reward	reward should be greater than 0	reward was greater than 0	Assertion via JUnit [testgetReward()]	Pass

Unit Tests for ComboGoal Class

A small test class was written for the new ComboGoal class, that generates an instance of ComboGoal and tests each of its methods to ensure that they are operating as intended. All new methods relevant to ComboGoal were tested and their outputs were asserted using expected outputs through JUnit. All of the methods tested in the original GoalTest class were also implemented to extensive testing has been upheld for this class.

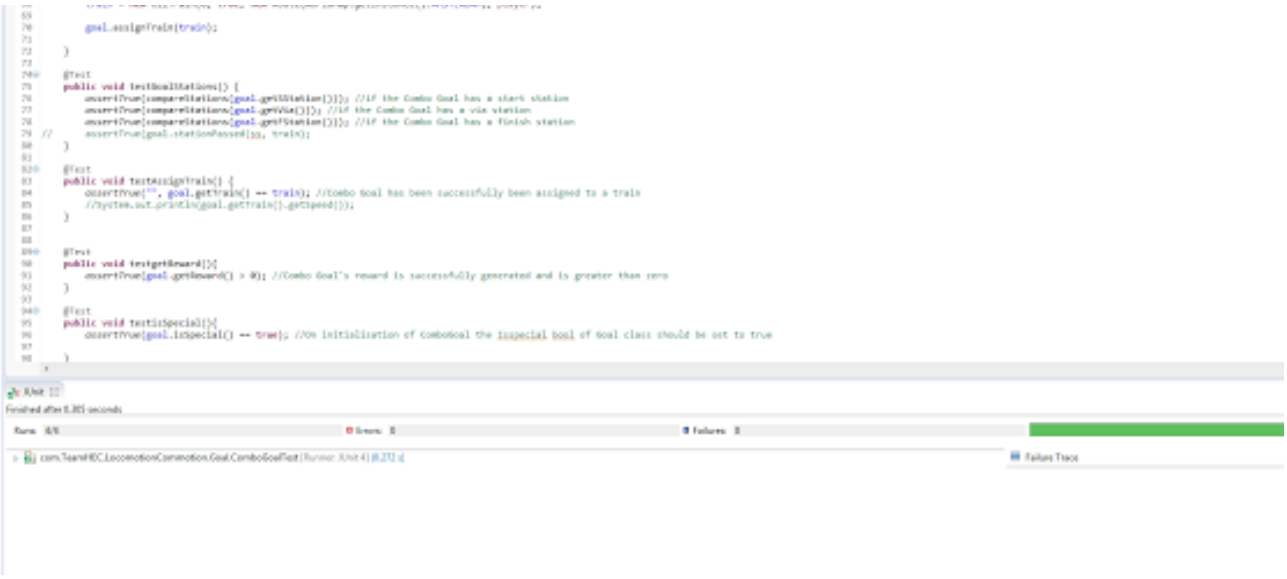
The Methods and Attributes of the Timed class that we tested are –

- Station **sStation**
- Station **viaStation**
- Station **fStation**
- Train **train**
- int **reward**
- boolean **isSpecial**
- int **turnLimit**
- int **startTurn**
- Station **getSStation()**
- Station **getViaStation()**
- Station **getFStation()**
- Train **getTrain()**
- int **getReward()**
- boolean **isSpecial()**
- int **getTurnLimit()**
- int **getStartTurn()**

Test Description	Expected Result	Result	Proof of Result	Status
Generates an instance of RouteGoal and test the value of isSpecial	isSpecial returns true	isSpecial returned true	The absence of errors upon instantiating TimedGoal and the assertion via JUnit [testIsSpecial()]	Pass
Test the value of sStation	compareStations returns true	compareStations returned true	Assertion via JUnit [testGoalStations()]	Pass
Test the value of viaStation	compareStations returns true	compareStations returned true	Assertion via JUnit [testGoalStations()]	Pass

Test the value of fStation	compareStations returns true	compareStations returned true	Assertion via JUnit [testGoalStations()]	Pass
Test the value of train	Train is correctly assigned to TimedGoal	Train was correctly assigned to TimedGoal	Assertion via JUnit [testAssignTrain()	Pass
Test the value of reward	reward should be greater than 0	reward was greater than 0	Assertion via JUnit [testgetReward()]	Pass
Test the value of turnLimit	turnLimit should be greater than 0	turnLimit was greater than 0	Assertion via JUnit [testTurnLimit()]	Pass
Test the value of startTurn	startTurn should be greater than or equal to 0	startTurn was greater than or equal to 0	Assertion via JUnit [testStartTurn()]	Pass

Note: After this test startTurn was changed to be 0 at the goal's creation and later changed to reflect the current turn counter once a train was assigned to it, at all points the startTurn is greater than or equal to 0.

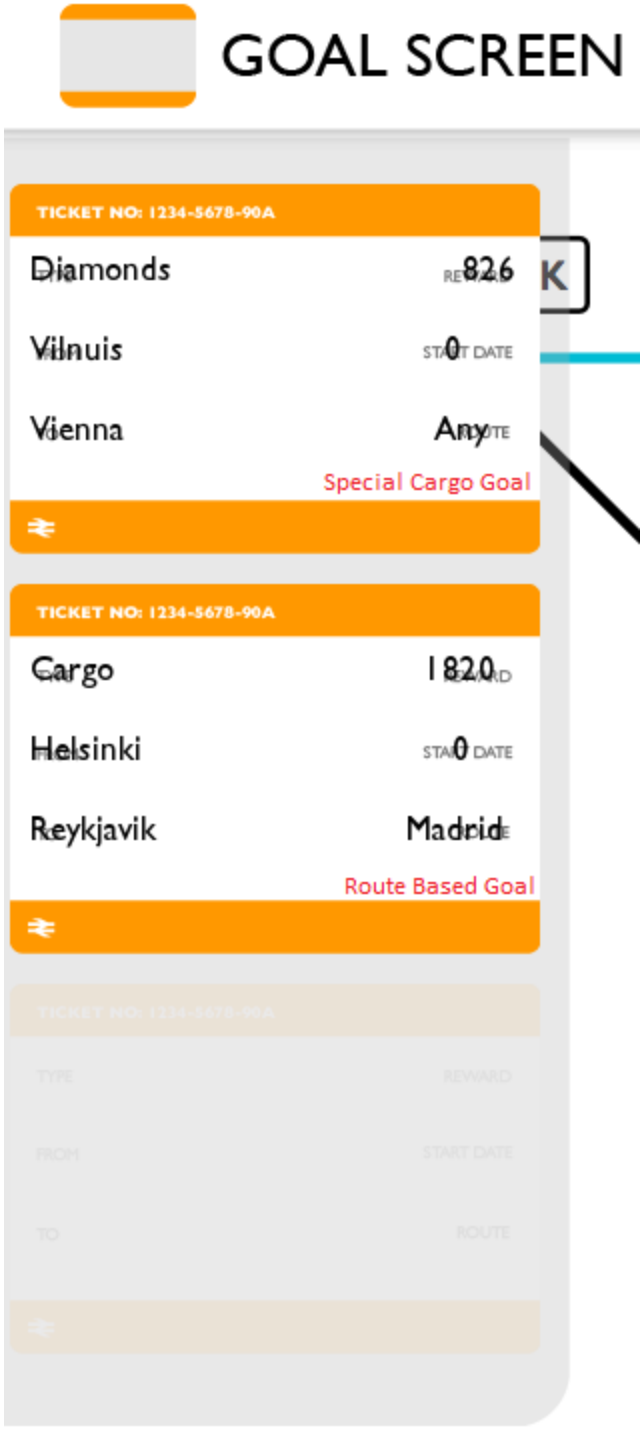


Black Box Testing (Goal Extensions)

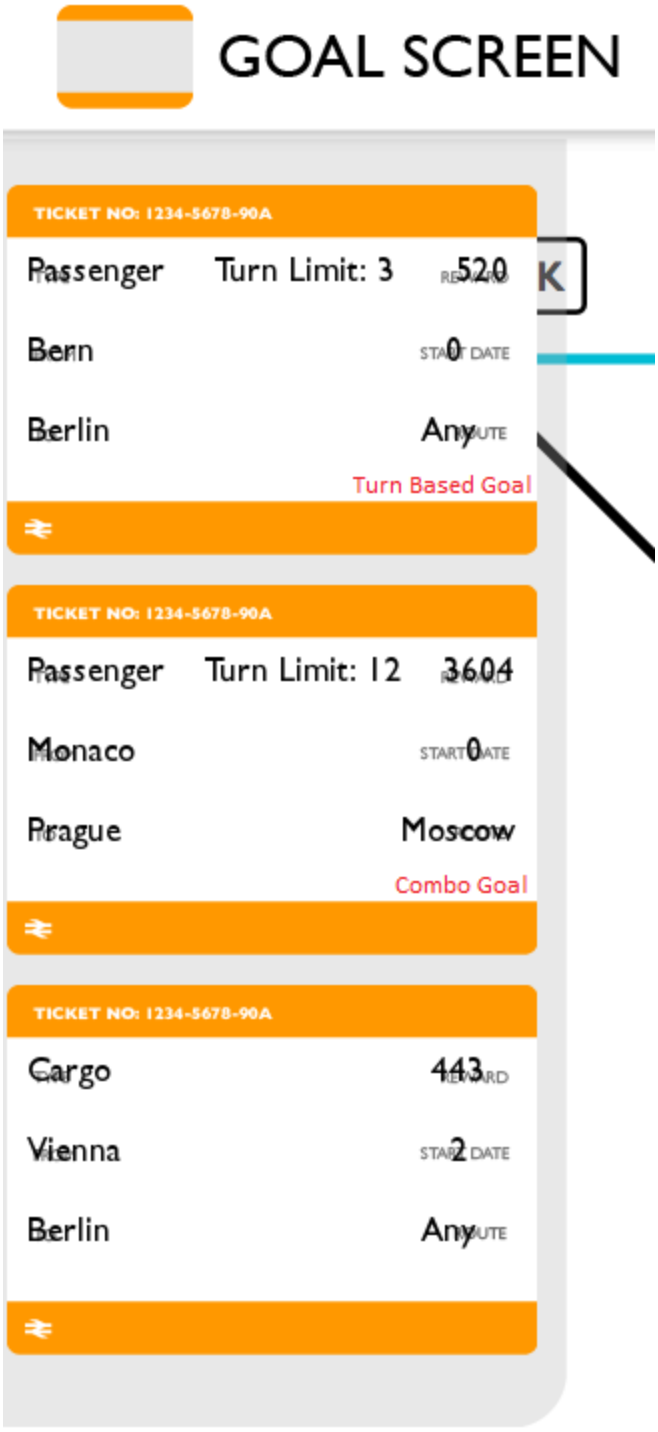
Test Description	Expected Result	Result	Proof of Result	Status
For Special Cargo Goals, Cargo should be set to Diamonds and a player should be able to choose such a goal	'Diamonds' is displayed on the card for Special Cargo Goal and a player is able to add that goal to his list of goals	'Diamonds' is displayed on the card for Special Cargo Goal and a player is able to add that goal to his list of goals	As shown in Fig. 1, 'Diamonds' is displayed on the Special Cargo Goal and it has been added to player's list of chosen goals	Pass
For Special route based goals, a via/route station should be specified and a player should be able to choose such a goal	A route station is specified for Special route based goal and a player is able to add that goal to his list of goals	A route station is specified for Special route based goal and a player is able to add that goal to his list of goals	As shown in Fig. 1, a route/via station 'Madrid' is displayed on the Route Based Goal and it has been added to player's list of chosen goals	Pass
For Special Timed based goals, a turn limit should be specified and a player should be able to choose such a goal	A turn limit is specified for Special Time based goal and a player is able to add that goal to his list of goals	A turn limit is specified for Special Time based goal and a player is able to add that goal to his list of goals	As shown in Fig. 2, a Turn Limit of '3' is displayed on a Special Route Goal and it has been added to player's list of chosen goals	Pass
For Special Combo Goals, a turn limit and a route station should be specified and a player should be able to choose such a goal	Both turn limit and route station are specified on the the goal card and a player is able to add that goal to his list of goals	Both turn limit and route station are specified on the the goal card and a player is able to add that goal to his list of goals	As shown in Fig. 2, a Turn Limit of '12' and a route station 'Moscow' are displayed on the Combo Goal and it has been added to player's list of chosen goals	Pass
Attempt to complete a standard (non-special) goal **(Func.SYS.2.4)**	The non-special goal is completed and the player is rewarded for the completion of the goal	The non-special goal is completed and the player is rewarded for the completion of the goal	As shown in Fig 4, the non-special goal is completed and the player is rewarded and the Goal Completion message is displayed.	Pass
Attempt to complete a Special Cargo Goal **(Func.SYS.2.4)**	The Special Cargo Goal is completed and the player is	The Special Cargo Goal is completed and the player is	There is no screenshot of this test due to it	Pass

	rewarded for the completion of the goal	rewarded for the completion of the goal	displaying the same message and using the same method as a standard goal.	
Attempt to complete a Special Route based goal **(Func.SYS.2.4)**	The Special Route Goal is completed and the player is rewarded for the completion of the goal	The Special Route Goal is completed and the player is rewarded for the completion of the goal	There is no screenshot of this test due to it displaying the same message and using the same method as a standard goal.	Pass
Attempt to complete a Special timed goal **(Func.SYS.2.4)**	The Special Timed Goal is completed and the player is rewarded for the completion of the goal	The Special Timed Goal is completed and the player is rewarded for the completion of the goal	There is no screenshot of this test due to it displaying the same message and using the same method as a standard goal.	Pass
Attempt to complete a Combo Goal **(Func.SYS.2.4)**	The Special Combo Goal is completed and the player is rewarded for the completion of the goal	The Special Combo Goal is completed and the player is rewarded for the completion of the goal	There is no screenshot of this test due to it displaying the same message and using the same method as a standard goal.	Pass
When a player fails to complete a goal in certain time limit after assigning the goal to his train, Goal Failed Message should be displayed. **(Func.SYS.2.4)**	Goal Failed Message is displayed whenever the turn limit is exceeded after assigning the goal to a train	- On exceeding the turn limit, Goal Failed message is displayed when a train has been to the start station, otherwise it is not. (Bug)	As shown in Fig. 3, a message was displayed to the player about him not being able to complete the goal in the required time limit	Pass (Fail if train hasn't been to start station)

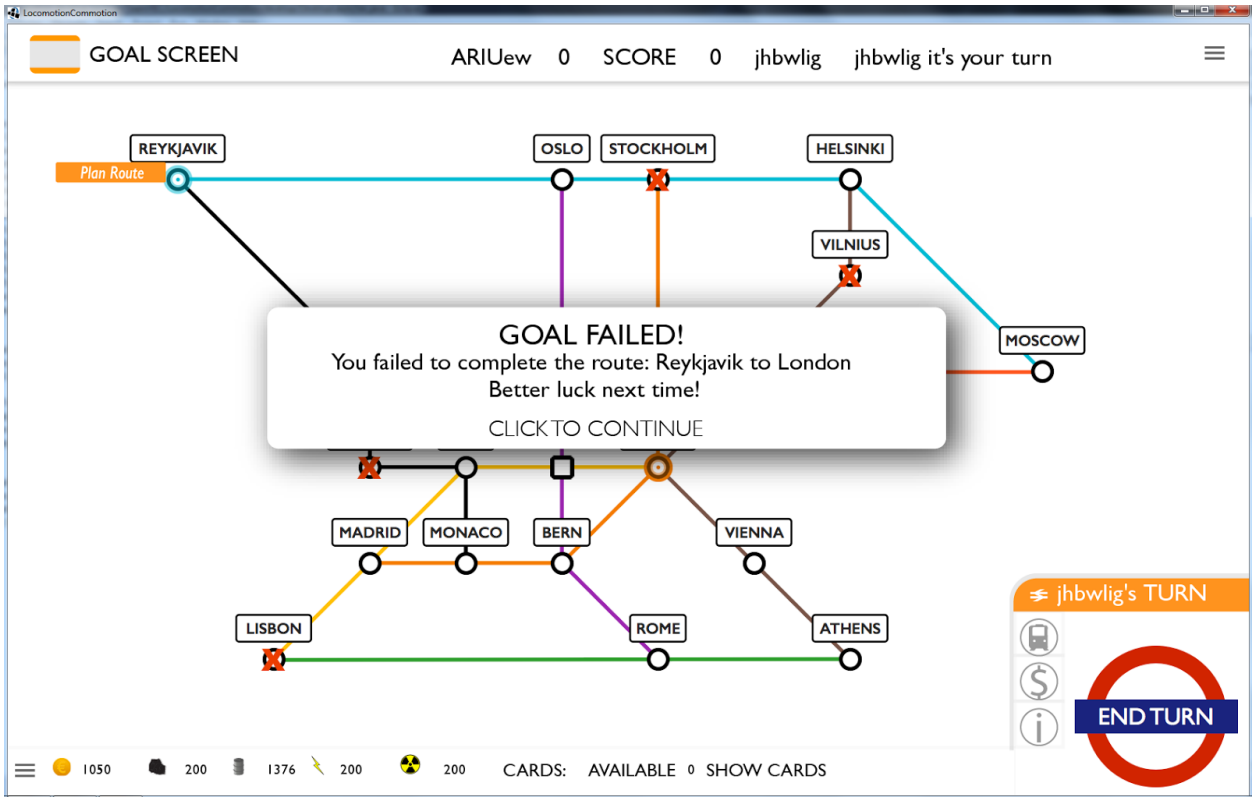
Note : All the above tests have been performed before 'score' class was added to the game design



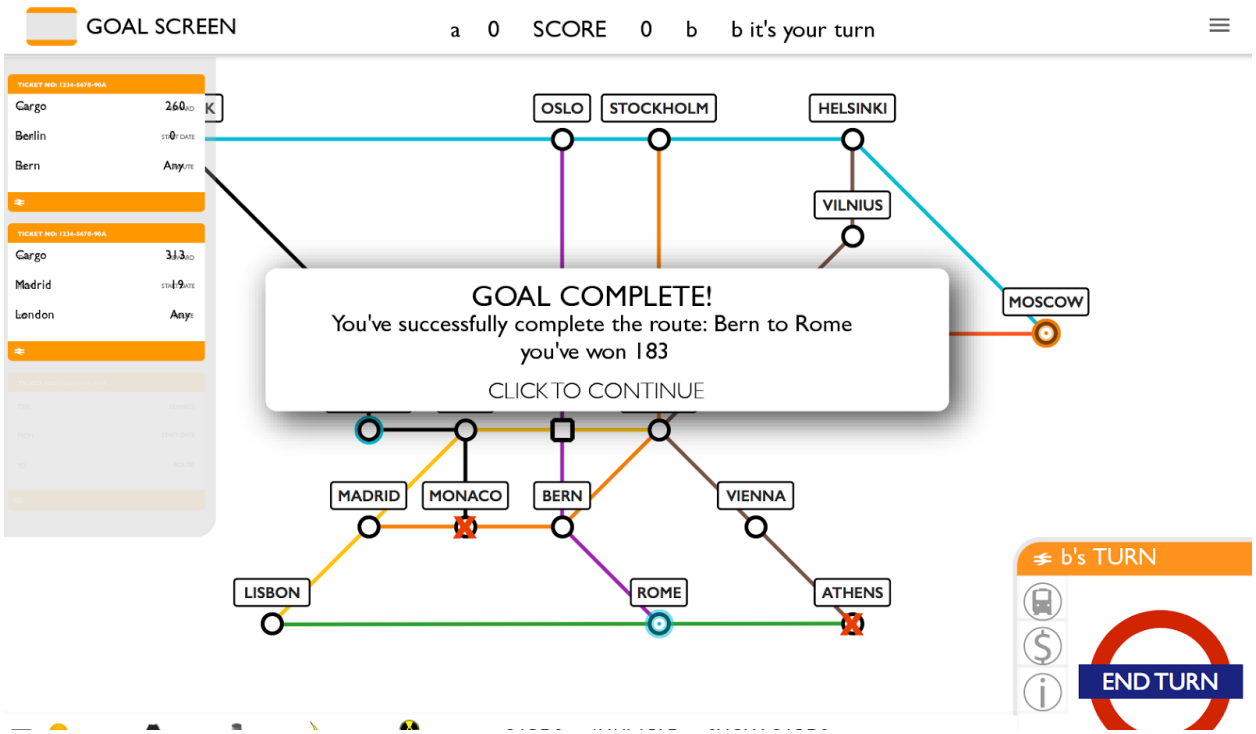
(Figure 1)



(Figure 2)



(Figure 3 - Message displayed on failing a goal)



(Figure 4 - Message displayed on completing a goal successfully)

Unit testing for Shop

In order to ensure that the methods added to the Shop class, the shopTest was extended to test the newly added buyTrain function. All acceptable inputs for the buyTrain method were tested and their outcomes were asserted through expected outcomes in JUinit. The methods originally implemented in the Shop were also tested to ensure extensive testing has been upheld in this class.

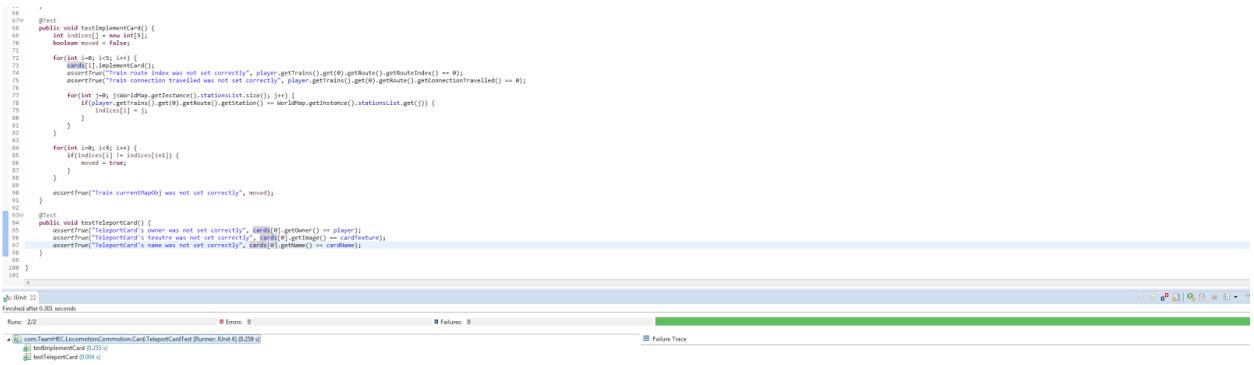
Test Description	Expected Result	Result	Proof of Result	Status
The buyTrain function is called with "Coal" as the parameter [**Func.SYS.4.9**]	The player's gold is reduced by the price of the train and the train is added to their list of trains	The player's gold was reduced by the cost of the train and the train was added to their list of trains	The JUnit test for the size of the players list of trains increasing by one and their gold reducing by the cost of the train	Pass
The buyTrain function is called with "Oil" as the parameter [**Func.SYS.4.9**]	The player's gold is reduced by the price of the train and the train is added to their list of trains	The player's gold was reduced by the cost of the train and the train was added to their list of trains	The JUnit test for the size of the players list of trains increasing by one and their gold	Pass

			reducing by the cost of the train	
The buyTrain function is called with "Electric" as the parameter [**Func.SYS.4.9**]	The player's gold is reduced by the price of the train and the train is added to their list of trains	The player's gold was reduced by the cost of the train and the train was added to their list of trains	The JUnit test for the size of the players list of trains increasing by one and their gold reducing by the cost of the train	Pass
The buyTrain function is called with "Nuclear" as the parameter [**Func.SYS.4.9**]	The player's gold is reduced by the price of the train and the train is added to their list of trains	The player's gold was reduced by the cost of the train and the train was added to their list of trains	The JUnit test for the size of the players list of trains increasing by one and their gold reducing by the cost of the train	Pass
The repairStation function was called on a test station	The players gold is decremented by 300 and the station is repaired	the players gold was decremented by 300 and the station was repaired	The JUnit test for the station isFaulty becoming true and the players gold reducing by 300	Pass
the repair station function was called on a test station when the player had no gold	the station remains faulty and the players gold remains the same	the station remained faulty and the players gold did not change	the JUnit test for the station isFaulty remaining true and the players gold not changing	Pass
the upgrade station function was called on a test station	The station level increasing by one and the players gold reducing by 400	the station level increased by 1 and the players gold was reduced by 400	the JUnit test for the station level increasing by one the the player's gold reducing by 400	Pass
The upgrade station function was called on a test station when the player had no gold	the station level and the player's gold should both remain unchanged	the stations level and the players gold both remained unchanged	The JUnit test for the players gold remaining the same and the station's level not changing	Pass

Unit test for TeleportCard

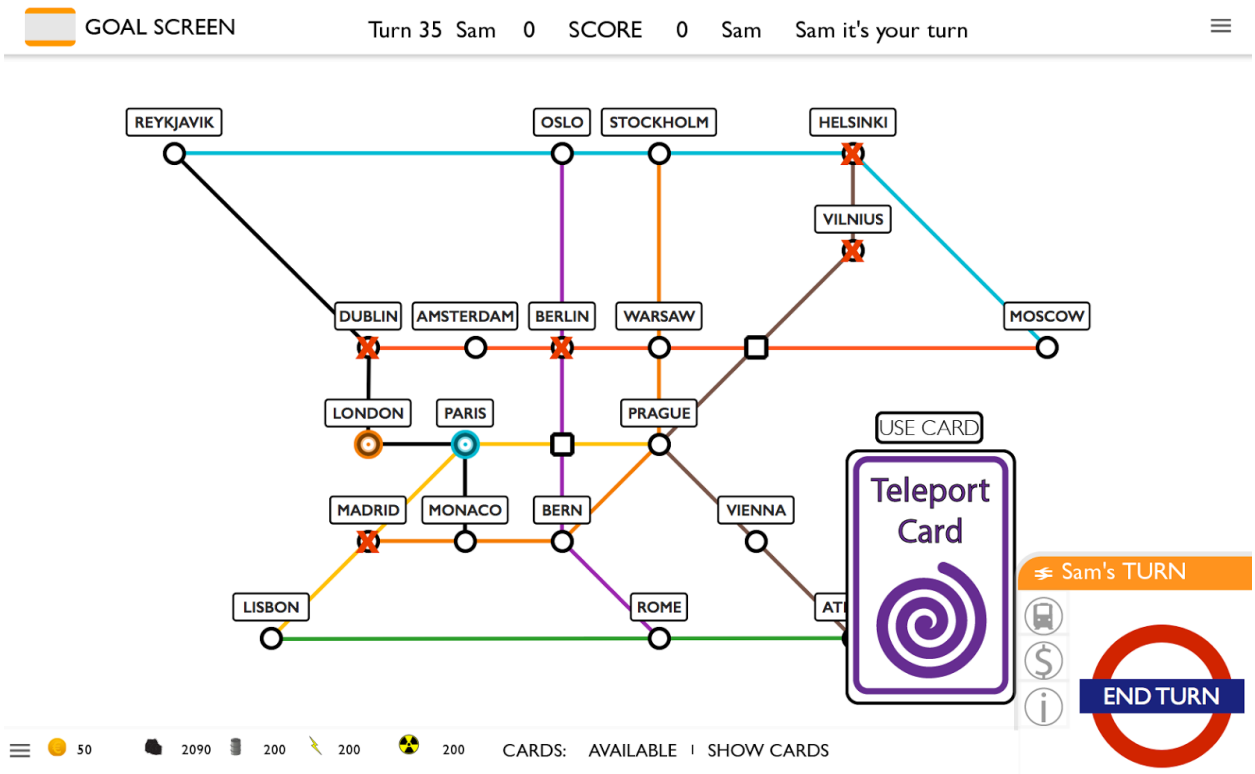
Originally the Teleportation WildCard did not function as intended, upon activation it would teleport the player's first train to a single pre-set station (London). This has now been altered to teleport a random train (owned by the player) to a random station on the map. A small extension was made to the original JUnit test class in order to ensure this new functionality was implemented correctly and worked as intended. The test table below only includes the extensions to the JUnit test as all original functionality and unit testing will have been performed by the previous team.

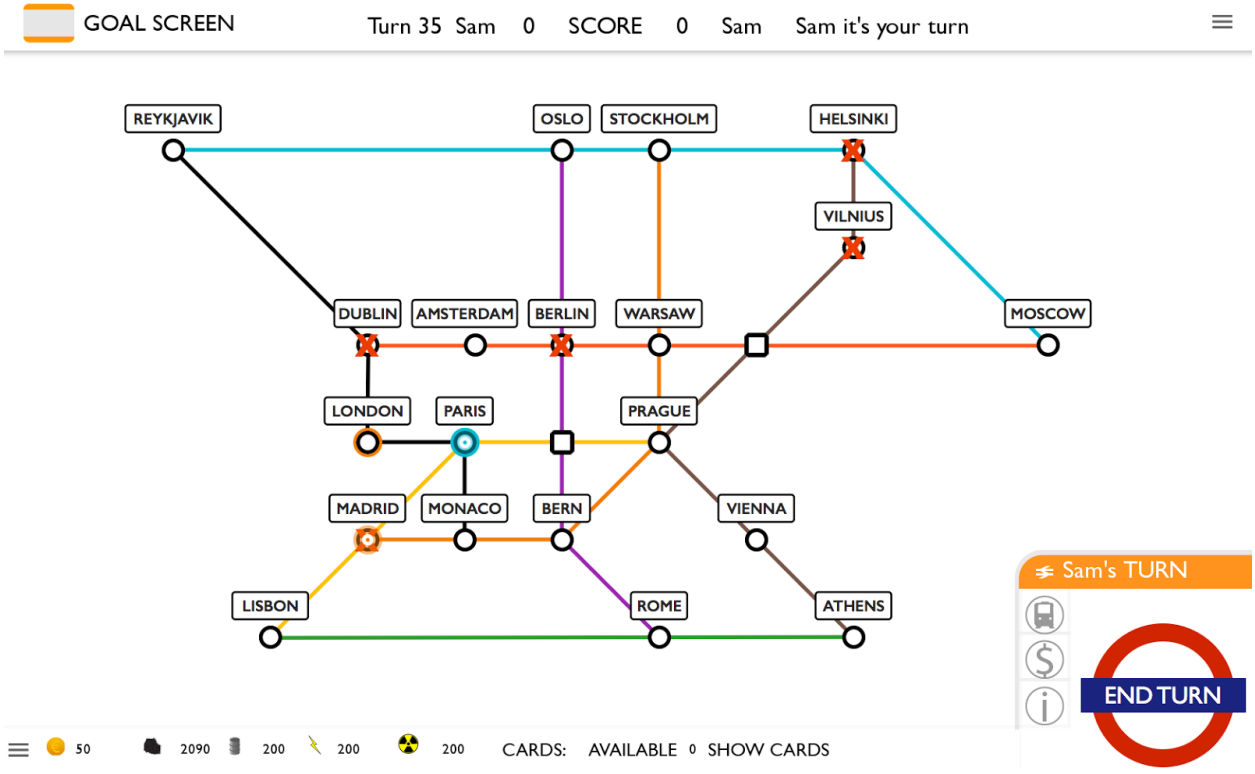
Test Description	Expected Result	Result	Proof of Result	Status
Activating the card moves a random train (owned by the player) to a random location. **(Func.SYS.4.8)**	A random train is moved to a random station.	A random train has moved to a random station.	Assertion via JUnit [testImplementCard()]	Pass



Black box testing for TeleportCard

Test Description	Expected Result	Result	Proof of Result	Status
Activating the card moves a random train (owned by the player) to a random location. **(Func.SYS.4.8)**	A random train is moved to a random station.	A random train has moved to a random station.	See before activation and after activation screenshots below. The Orange train has moved from London to Madrid.	Pass


GOAL SCREEN
Turn 35 Sam 0 SCORE 0 Sam Sam it's your turn
REYKJAVIK
OSLO
STOCKHOLM
HELSINKI
VILNIUS
DUBLIN
AMSTERDAM
BERLIN
WARSAW
MOSCOW
LONDON
PARIS
PRAGUE
USE CARD
Teleport Card
MADRID
MONACO
BERN
VIENNA
LISBON
ROME
Sam's TURN
END TURN
50
2090
200
200
200
CARDS: AVAILABLE 1 SHOW CARDS


GOAL SCREEN
Turn 35 Sam 0 SCORE 0 Sam Sam it's your turn
REYKJAVIK
OSLO
STOCKHOLM
HELSINKI
VILNIUS
DUBLIN
AMSTERDAM
BERLIN
WARSAW
MOSCOW
LONDON
PARIS
PRAGUE
MADRID
MONACO
BERN
VIENNA
LISBON
ROME
ATHENS
Sam's TURN
END TURN
50
2090
200
200
200
CARDS: AVAILABLE 0 SHOW CARDS

Unit Testing for Faults

A short test class was written for testing faults within the game. Although the 'Faults' implementation spans several several classes, we have tested it as one system.

The Methods and Attributes of we have tested for Faults are-

- WorldMap Class
  - generateFaults()
- Station Class
  - isFaulty()
  - isRepairable()
  - makeFaulty()
  - fixFault()
  - getStationLevel()
  - upgradeStation()
  - getFaultRate()

The faults section of the project was intended to fulfil the following Requirements:

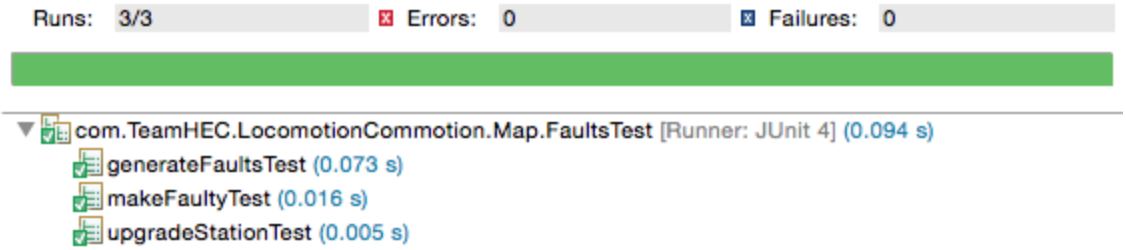
User.GP.6.3 : There **MUST** be at least two obstacles in the game.

The requirement **User.UI.10** also refers to faults, however this is about the GUI which is to be tested using a black box test, later in this document.

Test Description	Expected Result	Result	Proof of Result	Status
Station start level is 0	Station initialises at level 0	Station initialises at level 0	Assertion via JUnit passes	Pass
Station start fault rate is 0.1.	Station fault rate returns 0.1 when at level 0.	Station fault rate returns 0.1 when at level 0.	Assertion via JUnit passes	Pass
Station can be upgraded	Station level increases when upgraded	Station level increases when upgraded	Assertion via JUnit passes	Pass
Station fault rate is lower at higher levels	Station fault rate decreases when upgraded	Station fault rate decreases when upgraded	Assertion via JUnit passes	Pass
Station is initially not faulty.	Station isFaulty() method returns false.	Station isFaulty() method returns false.	Assertion via JUnit passes	Pass
Station is initially repairable.	Station isRepairable() method returns true.	Station isRepairable() method returns true.	Assertion via JUnit passes	Pass
Station can be made faulty	Station isFaulty() returns true,	Station isFaulty() returns true,	Assertion via JUnit passes	Pass

	when station is broken.	when station is broken.		
Station can be fixed if not permanently damaged	Station isFaulty() returns false again after fixFault() is called	Station isFaulty() returns false again after fixFault() is called	Assertion via JUnit passes	Pass

```java
package com.TeamHEC.LocomotionCommotion.Map;

import static org.junit.Assert.*;

@RunWith(GdxTestRunner.class)
public class FaultsTest {

    Station testStation;

    @Before
    public void setup(){
        testStation = WorldMap.getInstance().stationsList.get(0);
    }

    @Test //test upgradeStation()
    public void upgradeStationTest(){
        assertTrue("Station initialises as Level 0", testStation.getStationLevel() == 0);
        assertTrue("Station fault rate is initially 0.1%", testStation.getFaultRate() == 0.1);
        for(int i = 1; i < 5; i++){
            testStation.upgradeStation();
            assertTrue("Station can be upgraded", testStation.getStationLevel() == i);
            System.out.println(testStation.getFaultRate());
            assertTrue("Station fault rate decreases when upgraded", testStation.getFaultRate() <= 0.1);
        }
    }

    @Test //test isFaulty(), makeFaulty(), fixFault()
    public void makeFaultyTest(){
        assertFalse("Station initialises as not-faulty", testStation.isFaulty());
        assertTrue("Station initialises as repairable", testStation.isRepairable());
        testStation.makeFaulty();
        assertTrue("Station can be made faulty", testStation.isFaulty());
        if(testStation.isRepairable()){
            testStation.fixFault();
            assertFalse("Station can be successfully fixed", testStation.isFaulty());
        }

    }

    @Test //test generateFaults()
    public void generateFaultsTest() {
        for(int i = 0; i < 500; i++){
            WorldMap.getInstance().generateFaults();
        }

        Boolean flag = false; //creates a flag to determine if any of the stations in newMap are faulty

        for(int i = 0; i < WorldMap.getInstance().stationsList.size(); i++){
            if(WorldMap.getInstance().stationsList.get(i).isFaulty()) {
                flag = true;
                WorldMap.getInstance().stationsList.get(i).fixFault();
            }
        }
        assertTrue("Some faults are successfully generated at random.", flag);
    }


}
```



Black box testing for Faults

The faults section of our extension to the HEC project newly fulfils the requirements:

Func.OD.4.2 Game **SHOULD** alert players when a random event occurs.

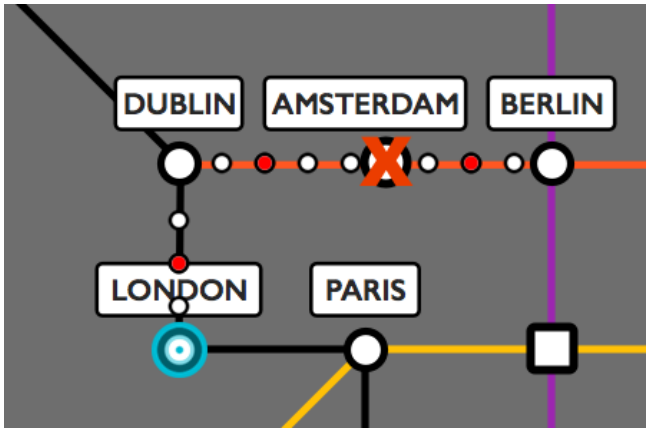
USER.GP.6.3 There **MUST** be at least two obstacles in the game.

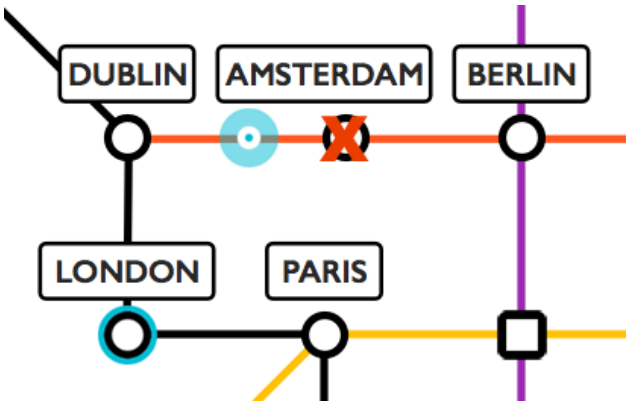
USER.UI.10 **MUST** display hazards on screen.

Test/Scenario	Expected Result	Result	Proof of Result	Status
Run a game to see faults appear at random on the map.	Faults will randomly occur throughout the course of the game.	As expected, several faults appear on the map.	See the screenshots below this table.	Pass
Attempt to move to a faulty station	The train cannot move to that station and is returned to the previous station. A warning message is fired.	The train cannot move to that station and is returned to the previous station. A warning message is fired.	See the screenshots below this table.	Pass
Attempt to leave a faulty station	The train cannot leave the station. A warning message is fired.	The train cannot leave the station. A warning message is fired.	See the screenshots below this table.	Pass
Repair a station by clicking the "repair" button on the station info panel	The station is no longer faulty and the station icon goes to the standard one	The station is no longer faulty and the station icon goes to the standard one	As shown in the screenshot below, the station was faulty and is then repaired	Pass
Attempt to repair a non-repairable faulty station	Warning message fires to prompt user of the illegal move	Warning message fires to prompt user of the illegal move	See the screenshots below this table.	Pass

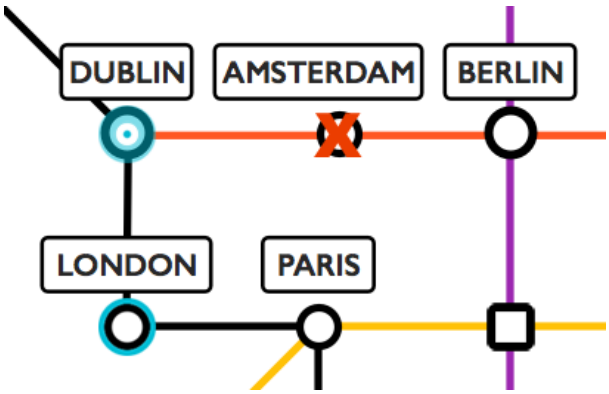
Test 1:

Test 2:


DUBLIN
AMSTERDAM
BERLIN
LONDON
PARIS


DUBLIN
AMSTERDAM
BERLIN
LONDON
PARIS

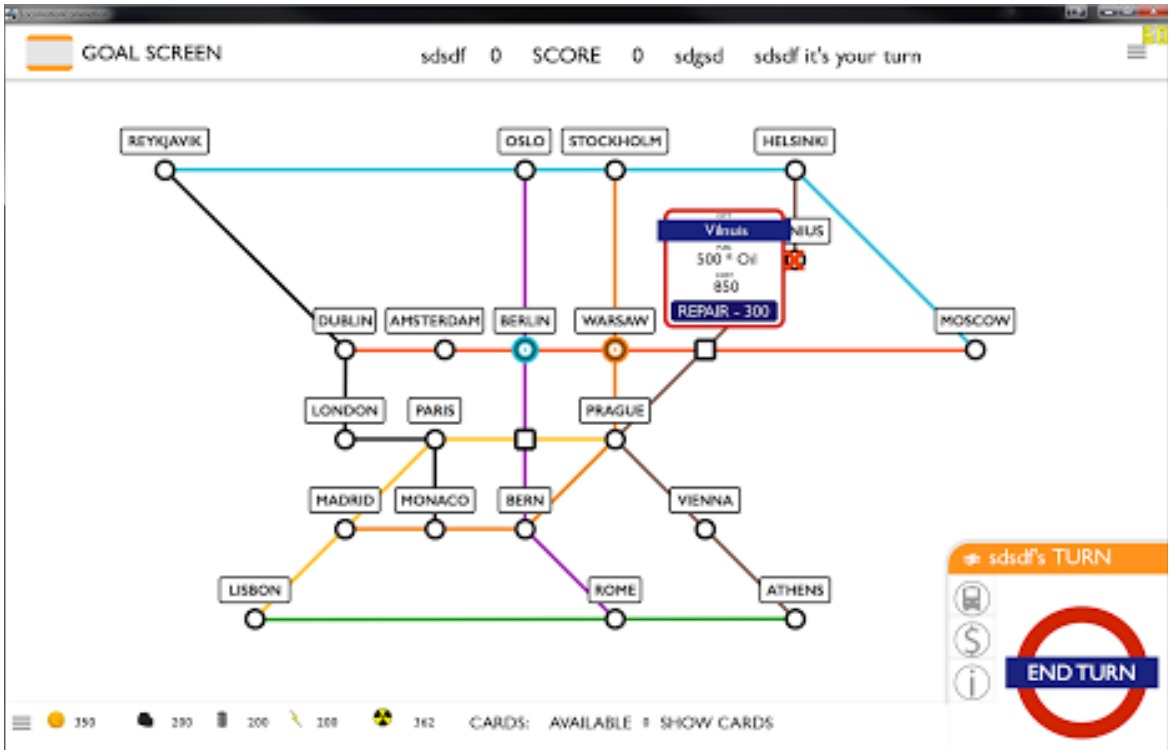

Sorry
The station is faulty. You must repair it to continue!
CLICK TO CONTINUE

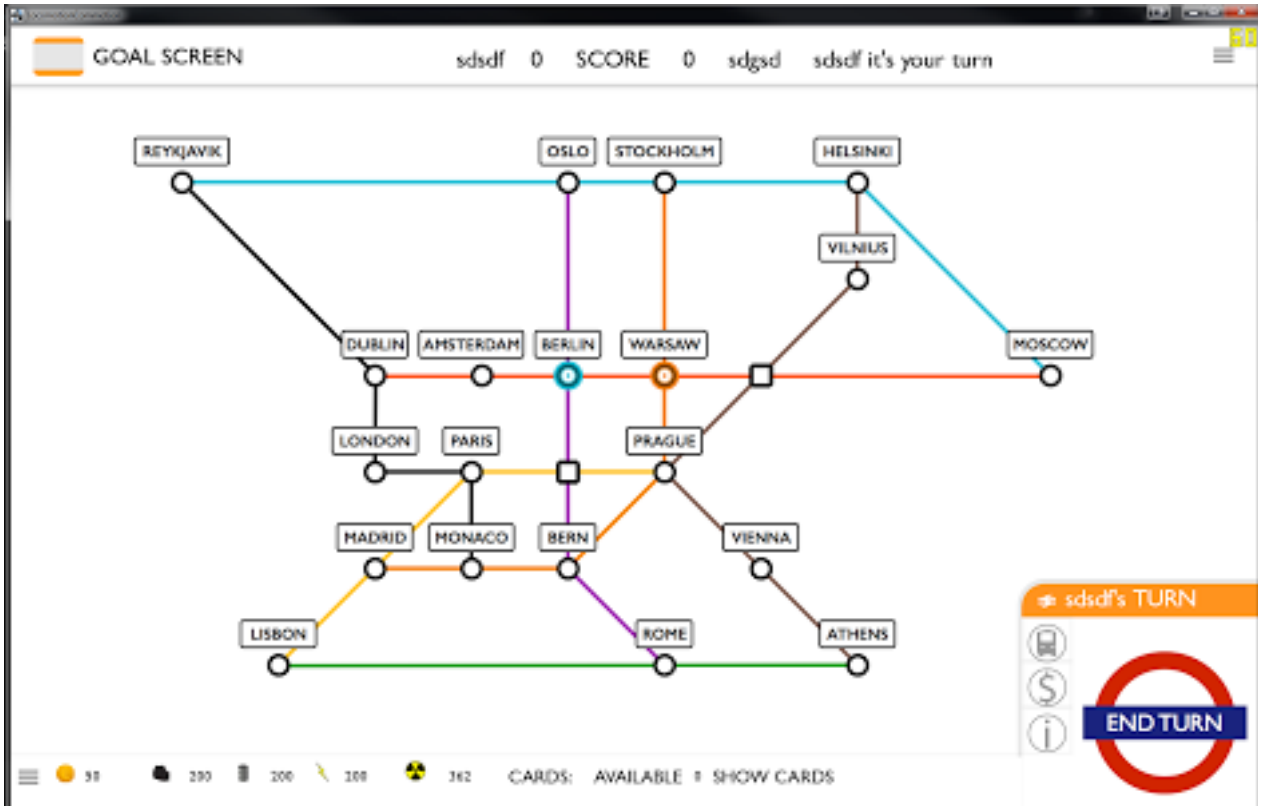

DUBLIN
AMSTERDAM
BERLIN
LONDON
PARIS

Test 3:


VILNIUS
VILNIUS
Sorry
The station is faulty. You must repair it to continue!
CLICK TO CONTINUE

Test 4:


GOAL SCREEN
sdsdf 0 SCORE 0 sdgsd sdsdf it's your turn
REYKJAVIK
OSLO
STOCKHOLM
HELSINKI
Vilnius
500
850
REPAIR - 300
DUBLIN
AMSTERDAM
BERLIN
WARSAW
MOSCOW
LONDON
PARIS
PRAGUE
MADRID
MONACO
BERN
VIENNA
LISBON
ROME
ATHENS
sdsdf's TURN
END TURN
CARDS: AVAILABLE SHOW CARDS


GOAL SCREEN
sdsdf 0 SCORE 0 sdgsd sdsdf it's your turn
REYKJAVIK
OSLO
STOCKHOLM
HELSINKI
VILNIUS
DUBLIN
AMSTERDAM
BERLIN
WARSAW
MOSCOW
LONDON
PARIS
PRAGUE
MADRID
MONACO
BERN
VIENNA
LISBON
ROME
ATHENS
sdsdf's TURN
END TURN
CARDS: AVAILABLE SHOW CARDS

Test 5


Sorry!
This station is beyond repair.
CLICK TO CONTINUE

Black box testing for Score

Most of the work in creating a score system was in pulling apart the "gold" and "points" systems that was implemented when we received the project from the previous group. The actual implementation of Score when finished is quite low level, with a mutator in Player, a Score class extending Resource and a method in goal that adds score.

Func.SYS.1 : System **must** keep both players' score

Func.SYS.8.1 : System **must** be able to add points to a players score

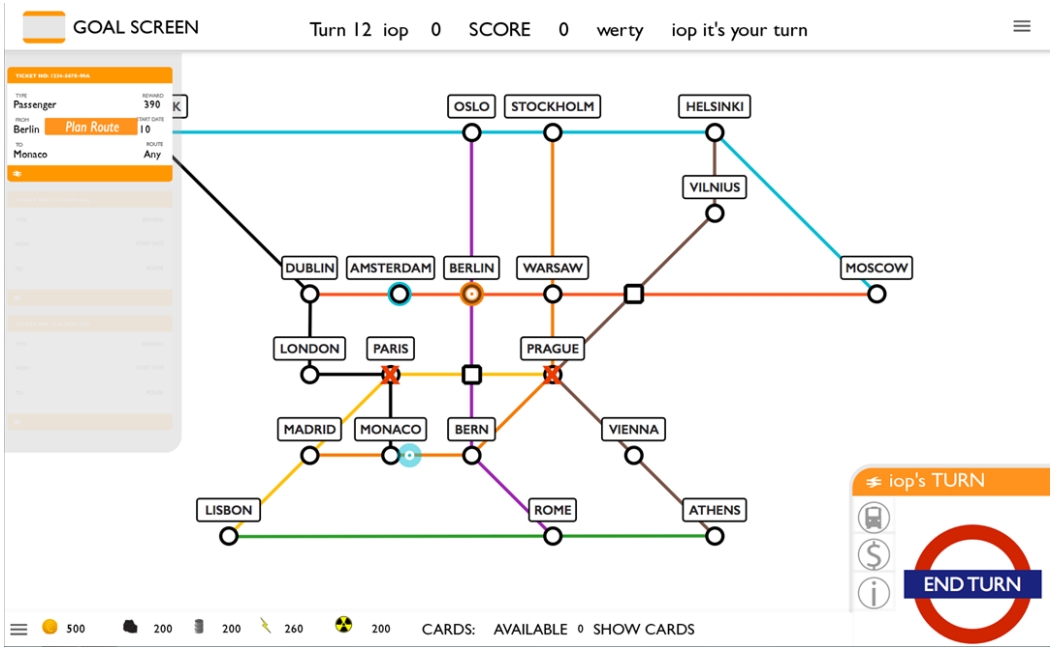
Func.Sys.8.2 : System **must** be able to assign points to a randomly generated goals.

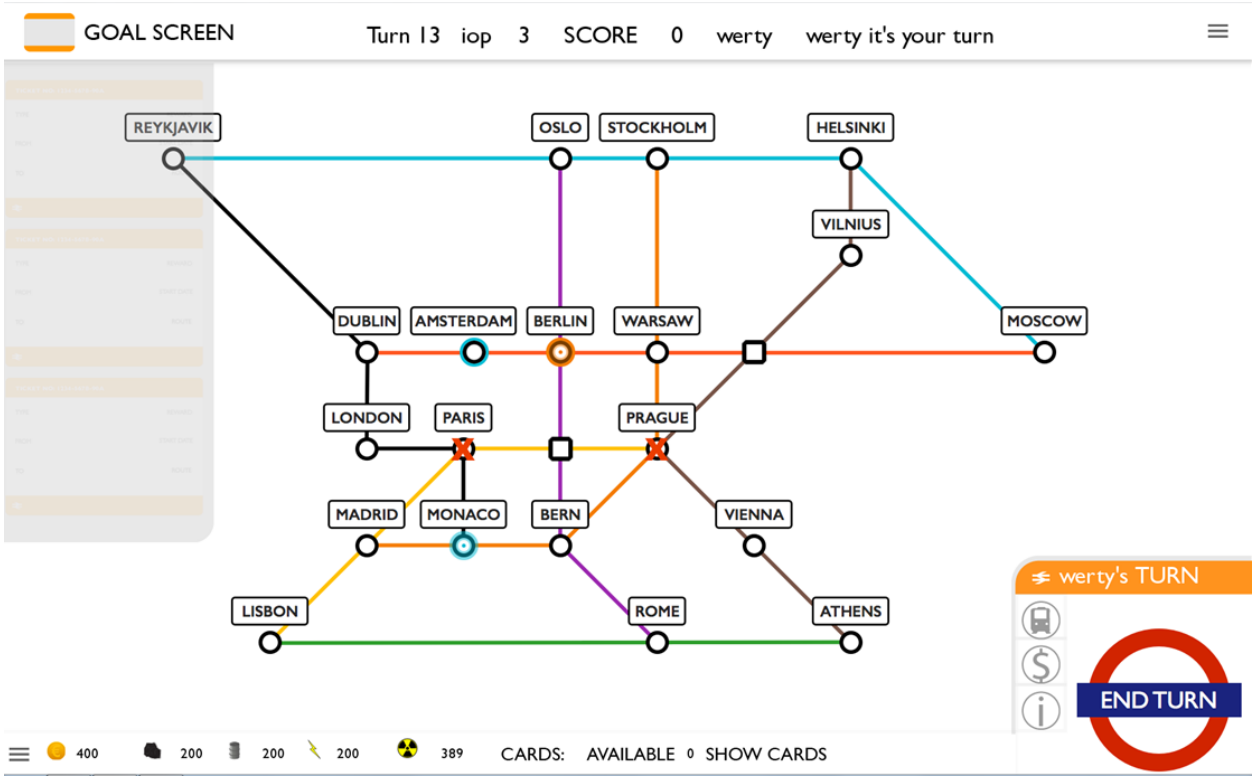
Func.Sys.4.1 : System **must** track of players resources in real time

It was decided that none of these components needed testing using JUnit, as any tests of mutators etc are trivial. Black box testing however would show the successful awarding of points to a player and their presence on screen.

The only time a player may receive points is when they complete a goal:

A goal from Berlin to Monaco was completed by player 1 with a reward of 390 gold and 3 Score.



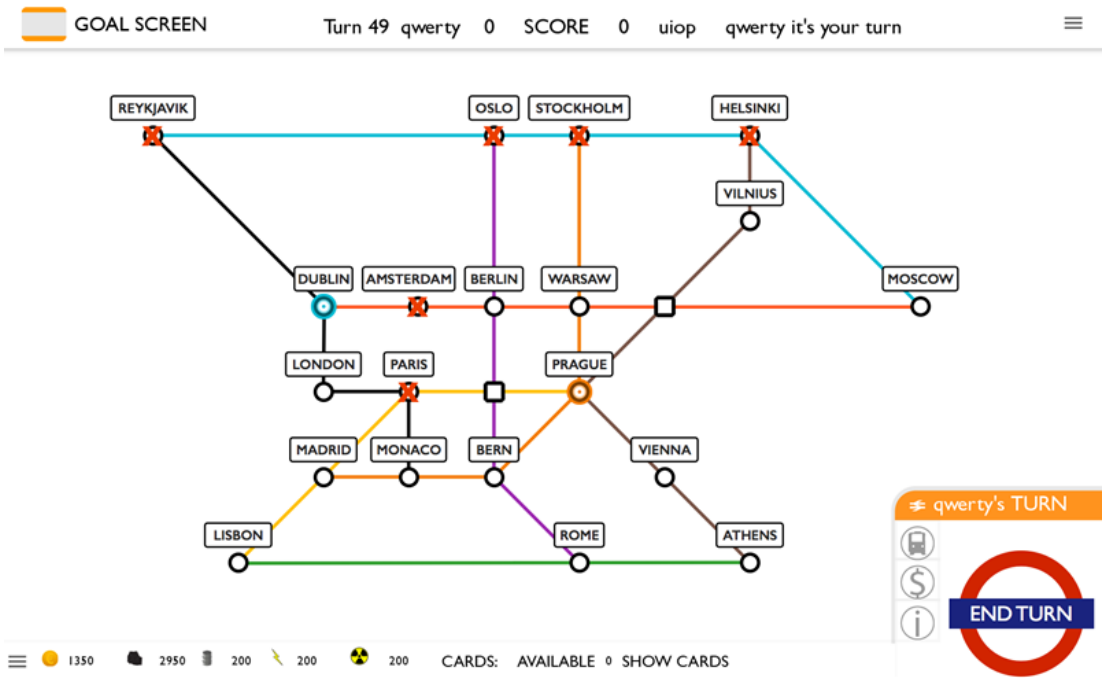


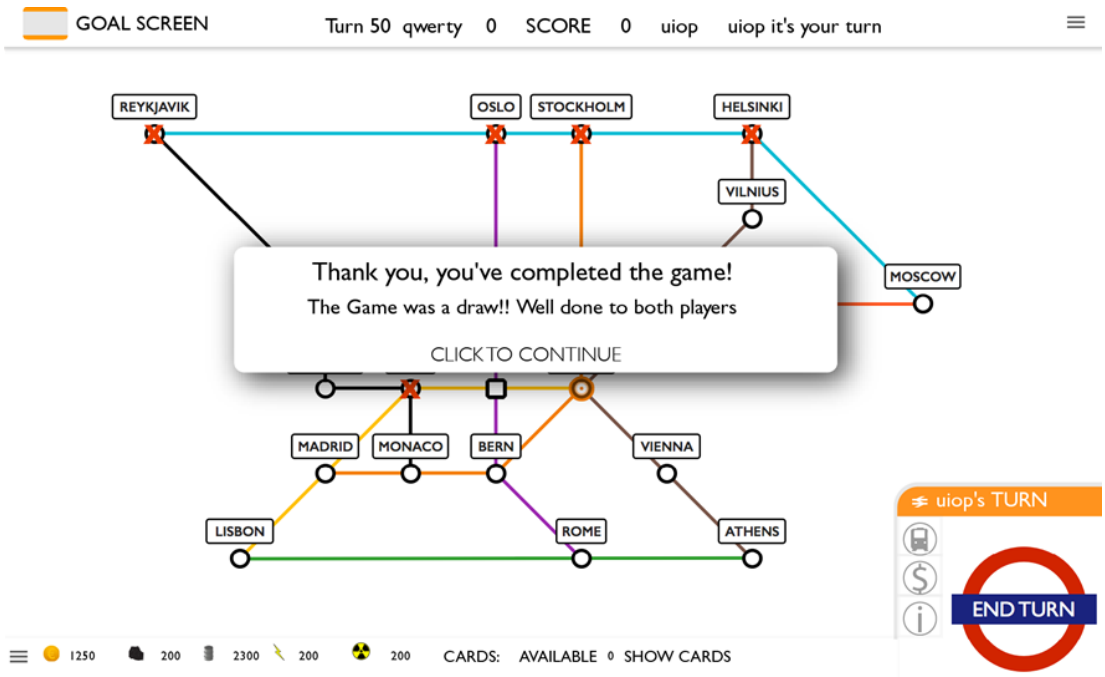
Black box testing for ending the game

The ability to end the game and have the program close after declaring the winner was implemented, this was a simple function in core game that the previous developers had named but not implemented. This functionality was black box tested:

Func.Sys.3.5 : System **must** be able to declare an end to the game once the game end condition has been reached.

Func.Sys.13: System **must** be able to terminate itself safely.


GOAL SCREEN
Turn 49 qwerty 0 SCORE 0 uiop qwerty it's your turn
REYKJAVIK
OSLO
STOCKHOLM
HELSINKI
VILNIUS
DUBLIN
AMSTERDAM
BERLIN
WARSAW
MOSCOW
LONDON
PARIS
PRAGUE
MADRID
MONACO
BERN
VIENNA
LISBON
ROME
ATHENS
qwerty's TURN
END TURN
1350 2950 200 200 200 CARDS: AVAILABLE 0 SHOW CARDS


GOAL SCREEN
Turn 50 qwerty 0 SCORE 0 uiop uiop it's your turn
REYKJAVIK
OSLO
STOCKHOLM
HELSINKI
VILNIUS
Thank you, you've completed the game!
The Game was a draw!! Well done to both players
CLICK TO CONTINUE
MOSCOW
MADRID
MONACO
BERN
VIENNA
LISBON
ROME
ATHENS
uiop's TURN
END TURN
1250 200 2300 200 200 CARDS: AVAILABLE 0 SHOW CARDS

Usability Testing

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.

The participants will be picked by being anyone passing by in the corridor outside the Software Laboratories in the Department of Computer Science, University of York. However, we will ensure that none of the participants have played any version of Locomotion Commotion before.

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 complete each task specified in the table below.
4. After each task is complete, *immediately* ask the users to rate how easy it was to understand and, where relevant, how challenging it was to complete.
5. Ease of understanding will be marked on a 5 point scale, where Very Easy (5) is our optimum result
6. Level of Challenge will be marked on a 9 point scale where 1 is too easy, 5 is just right and 9 is too hard. 5 is our optimum result again. The actual score for this will be the difference between 5 and the result recorded.
7. If the marks for the tests average out as 3.5 out of 5 or higher, the test passes.

Participants

The participants were all aged 18-20, which, although it is a limited range, fits the demographic of our expected audience for the game. There were 4 participants, of which 1 was female and 3 were male.

Results

Test/Scenario	**Expected Result**	**(Proof of) Result**	**Status**
Start a game (50 turn limit)	Will be measured by Ease of Understanding only as this is not an aspect of gameplay. The score will be 3.5 or greater.	Scored an average of 5.0 for Ease of Understanding.	Pass.

Select a goal	Will be measured by Ease of Understanding only as this is not an aspect of gameplay. The score will be 3.5 or greater.	Scored an average of 4.5 for Ease of Understanding.	Pass.
Assign a goal to a train	Will be measured by Ease of Understanding only as this is not an aspect of gameplay. The score will be 3.5 or greater.	Scored an average of 4.0 for Ease of Understanding.	Pass.
Complete a goal	Will be measured on both scales. These scores will each be 3.5 or greater.	Scored an average of 4.5 for Ease of Understanding and 4.0 for Level of Challenge.	Pass.
Complete a further two goals	Will be measured on both scales. These scores will each be 3.5 or greater.	Scored an average of 4.5 for Ease of Understanding and 3.5 for Level of Challenge.	Pass.
Finish the game	Will be measured on both scales. These scores will each be 3.5 or greater.	Scored an average of 4.0 for Ease of Understanding and 4.0 for Level of Challenge.	Pass.

Additional Notes

The participants praised the comprehensiveness of the user manual as well as the on-screen prompts.

The Level of Challenge became an average of 3.5 for Level of Challenge for completing a total of three goals, this is a near miss. It was thought that the Level of Challenge was slightly too hard, particularly for turn-limited goals. Some attention should be paid to this in further development and further tests should be undertaken in the next phase.

Trivial Requirements :

We felt that some of the system requirements, that had been traced by HEC already in the previous assessment didn't need re-testing as the code hasn't been modified. The following are those requirements **Func.Sys.2.3, Func.Sys.3.1, Func.Sys.3.2, Func.Sys.3.3, Func.Sys.4.3, Func.Sys.4.5, Func.Sys.4.6,, Func.Sys.4.8, Func.Sys.4.9, Func.Sys.4.10, Func.Sys.5.1, Func.Sys.6.1, Func.Sys.6.2, Func.Sys.6.3, Func.Sys.7.2, Func.Sys.10, Func.Sys.11.4, Func.Sys.12.2, Func.Sys.15.**

Some of the optional System Requirements were left out due to time constraints :

Func.Sys.2.5 : System **should** have special goals which provide Wildcards as reward

Func.Sys.4.5 : Stations **Could** randomly generate extra resources at the end of a turn.