Test Plan

by

SEPR Group GVA

Introduction

This test plan is based on a IEEE test plan outline, provided freely by Dublin City University[1].

This a master level plan to organise testing of our current game code. Similarly to our previous assessment we will discuss the: test items, features to be or not to be tested however we will also discuss known bugs.

As we have had to continue from another team's code we initially had to run and analyse the tests they had left behind; they all passed. Upon inspection of the tests, they were clearly very in depth, this gave us confidence that any future errors would be due to our code interacting poorly with theirs. Therefore we've had to create new tests for our modifications and extensions, to maintain the functionality of the game.

Test items

Test items are object classes that are to be individually tested for code correctness and integration. As the code we took from team HEC met the set of requirements we may assume that, as long as we do not make any changes that detract from the requirements, our final product will meet the requirements as well.
Test items that share properties because of the object oriented programming paradigm won't each need full explicit testing as those properties will have already been tested in other classes. For example fix rail card and break rail card are so similar that they can be treated as one kind of card. GoFasterStripes is much like a resource card in it's implementation it probably won't need a test of it's own.

Test item criteria will simply be: *the code has been modified by us and it is an object that the player can interact with.*

Thusly the test item list is: Teleport card and Break/Fix rail cards, (we extended) GoFasterStripes card.

Testing methods

There will be situations where code requires interaction from a player and it's simpler to test "by mouse" rather than writing code to call methods otherwise called by such clicks. So we will execute a test algorithm and record the results, this is a form of black box testing. Otherwise use white box testing with JUnit code; explicitly referencing and explaining the test code and it's results.

Features to be tested or not to be tested

We will test features that we've added or extended, these are: broken rail not being traversable and getting points for gold being earned.

[1] http://www.computing.dcu.ie/~davids/courses/CA267/ieee829mtp.pdf

All features previously implemented or ones that are subsets of tested features do not need to be explicitly tested. In our case by testing the functionality of the break/fix rail cards we also test the functionality of broken rail.

If features were previously implemented then they have already passed their JUnit tests, they have also passed their black box tests from us playing the game and not noticing any problems, aside from the known bugs mentioned in the extension report on page 2. However we modified how trains collide so that will need to be tested

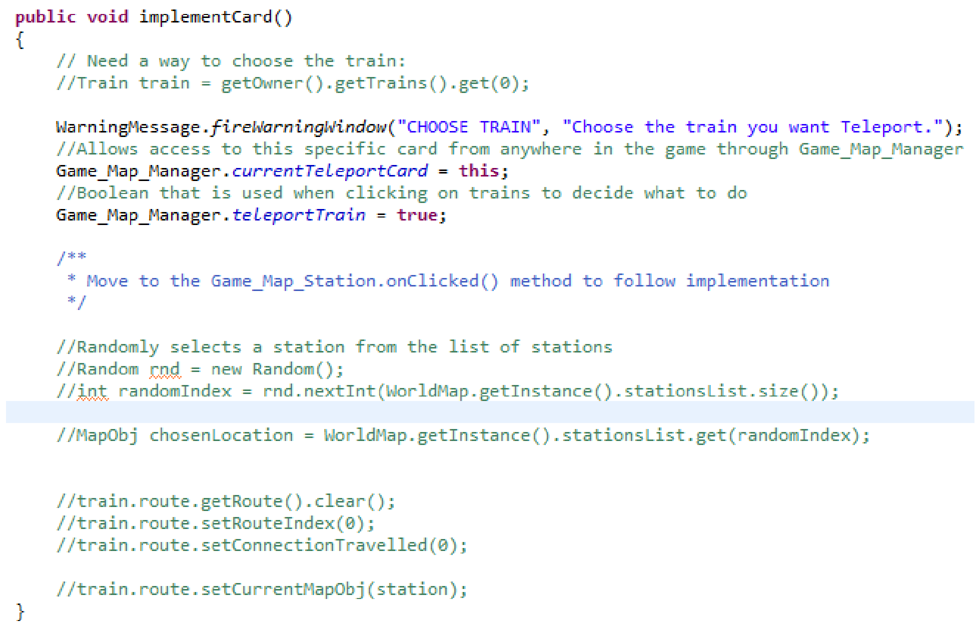
However rail being breakable is something that we have extended because of a change in the user requirements.

Features to be tested: getting points for gold being earned, train collision

N.B Formatting note - the discussion of tests will follow the structure: describe/explain code/modifications, description of test and pass criteria, description of results and then the resolutions made from the results.

Teleport card

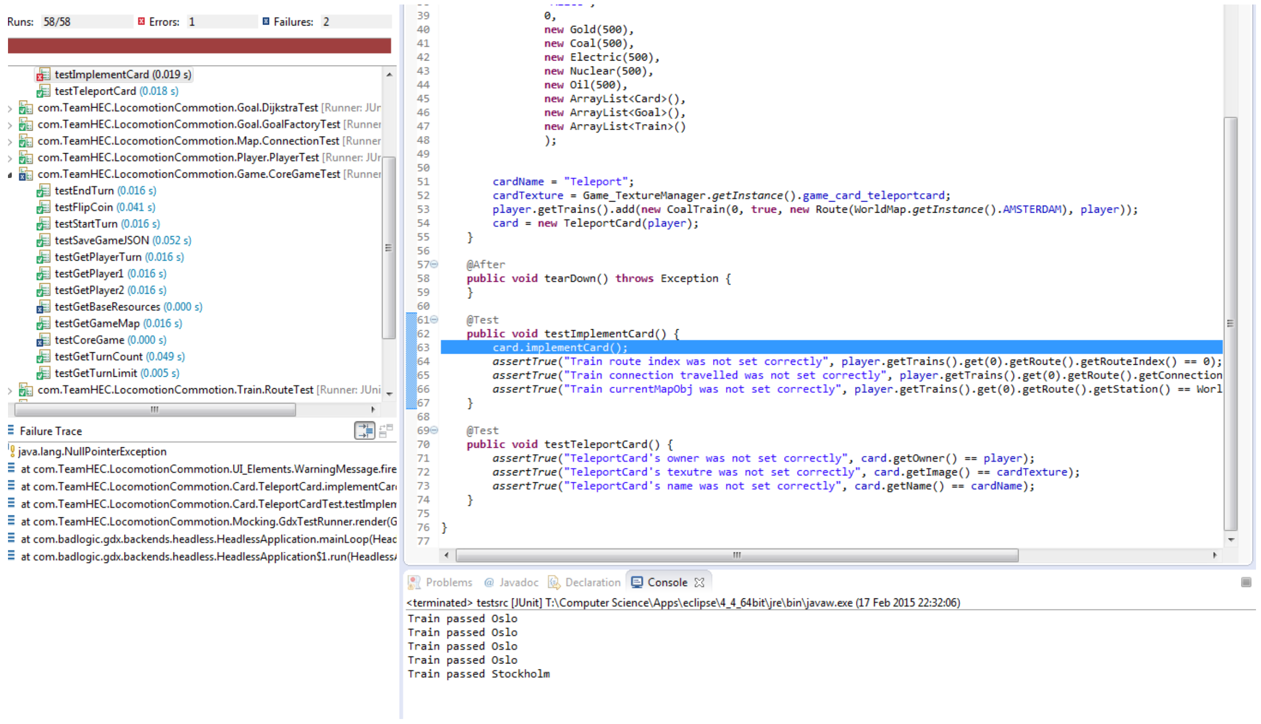
Originally the teleport card took the first train in a player's array list of trains and set its current location to a random station that exists on the map (assuming all stations are in the station list). Thus "teleporting" the train to a random station. This can be observed in the comments within the following screen snip of the TeleportCard.java class (snipping tool used).



```
public void implementCard()
{
    // Need a way to choose the train:
    //Train train = getOwner().getTrains().get(0);

    WarningMessage.fireWarningWindow("CHOOSE TRAIN", "Choose the train you want Teleport.");
    //Allows access to this specific card from anywhere in the game through Game_Map_Manager
    Game_Map_Manager.currentTeleportCard = this;
    //Boolean that is used when clicking on trains to decide what to do
    Game_Map_Manager.teleportTrain = true;

    /**
     * Move to the Game_Map_Station.onClicked() method to follow implementation
     */

    //Randomly selects a station from the list of stations
    //Random rnd = new Random();
    //int randomIndex = rnd.nextInt(WorldMap.getInstance().stationsList.size());

    //MapObj chosenLocation = WorldMap.getInstance().stationsList.get(randomIndex);


    //train.route.getRoute().clear();
    //train.route.setRouteIndex(0);
    //train.route.setConnectionTravelled(0);

    //train.route.setCurrentMapObj(station);
}
```

However Jack modified the teleport card so that upon use, in-game, it requests the player to choose a train to teleport and a target station to teleport to. This change actually created a

NullPointerException in a JUnit test, because the a test left by team HEC called the function card.implementCard(), but there's nothing to implement.



```
Runs: 58/58    Errors: 1    Failures: 2

    testImplementCard (0.019 s)
    testTeleportCard (0.018 s)
> com.TeamHEC.LocomotionCommotion.Goal.DijkstraTest [Runner: JUr
> com.TeamHEC.LocomotionCommotion.Goal.GoalFactoryTest [Runner
> com.TeamHEC.LocomotionCommotion.Map.ConnectionTest [Runner
> com.TeamHEC.LocomotionCommotion.Player.PlayerTest [Runner: JUr
  com.TeamHEC.LocomotionCommotion.Game.CoreGameTest [Runner
    testEndTurn (0.016 s)
    testFlipCoin (0.041 s)
    testStartTurn (0.016 s)
    testSaveGameJSON (0.052 s)
    testGetPlayerTurn (0.016 s)
    testGetPlayer1 (0.016 s)
    testGetPlayer2 (0.016 s)
    testGetBaseResources (0.000 s)
    testGetGameMap (0.016 s)
    testCoreGame (0.000 s)
    testGetTurnCount (0.049 s)
    testGetTurnLimit (0.005 s)
> com.TeamHEC.LocomotionCommotion.Train.RouteTest [Runner: JUni

Failure Trace
java.lang.NullPointerException
at com.TeamHEC.LocomotionCommotion.UI_Elements.WarningMessage.fire
at com.TeamHEC.LocomotionCommotion.Card.TeleportCard.implementCar
at com.TeamHEC.LocomotionCommotion.Card.TeleportCardTest.testImplem
at com.TeamHEC.LocomotionCommotion.Mocking.GdxTestRunner.render(G
at com.badlogic.gdx.backends.headless.HeadlessApplication.mainLoop(Heac
at com.badlogic.gdx.backends.headless.HeadlessApplication$1.run(Headless
```

```java
                0,
                new Gold(500),
                new Coal(500),
                new Electric(500),
                new Nuclear(500),
                new Oil(500),
                new ArrayList<Card>(),
                new ArrayList<Goal>(),
                new ArrayList<Train>()
                );


        cardName = "Teleport";
        cardTexture = Game_TextureManager.getInstance().game_card_teleportcard;
        player.getTrains().add(new CoalTrain(0, true, new Route(WorldMap.getInstance().AMSTERDAM), player));
        card = new TeleportCard(player);
    }

    @After
    public void tearDown() throws Exception {
    }

    @Test
    public void testImplementCard() {
        card.implementCard();
        assertTrue("Train route index was not set correctly", player.getTrains().get(0).getRoute().getRouteIndex() == 0);
        assertTrue("Train connection travelled was not set correctly", player.getTrains().get(0).getRoute().getConnection
        assertTrue("Train currentMapObj was not set correctly", player.getTrains().get(0).getRoute().getStation() == Worl
    }

    @Test
    public void testTeleportCard() {
        assertTrue("TeleportCard's owner was not set correctly", card.getOwner() == player);
        assertTrue("TeleportCard's texutre was not set correctly", card.getImage() == cardTexture);
        assertTrue("TeleportCard's name was not set correctly", card.getName() == cardName);
    }

}
```

Problems @ Javadoc Declaration Console

```
<terminated> testsrc [JUnit] T:\Computer Science\Apps\eclipse\4_4_64bit\jre\bin\javaw.exe (17 Feb 2015 22:32:06)
Train passed Oslo
Train passed Oslo
Train passed Oslo
Train passed Oslo
Train passed Stockholm
```

Furthermore to save time in this test Michael modified CardFactory.java and createAnyCard() so that purchasing a card always gives you a teleport card rather than hoping that the card factory will generate a teleport card[2].

```java
magicCardList = new ArrayList<Card>();
magicCardList.add(teleport);
magicCardList.add(goFaster);
magicCardList.add(breakRail);
magicCardList.add(fixRail);
```

This design choice allows us select cards to generate should we want to, such as now.

```java
public Card createAnyCard()
{
    ArrayList<Card> cardList = new ArrayList<Card>(magicCardList);
    cardList.addAll(resourceCardList);

    //return cardList.get(random.nextInt(cardList.size()));
    return cardList.get(0);
}
```

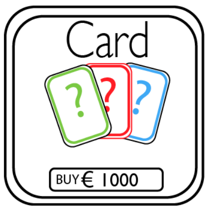
Test criteria

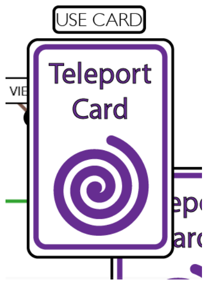
We extended the game with the teleport card by team HEC's request, and thus we interpreted the requirements of the card as: *the card must be able to instantaneously take a train from one station to another.*

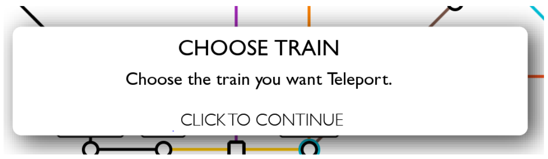
[2] N.B - this change is not in the final code; it just made testing more efficient

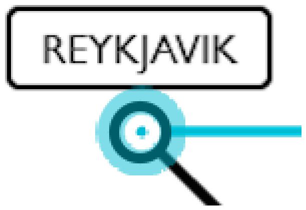
My algorithm for this black box test is as follows:
1.Purchase teleport cards
2.Select teleport card
3.Select a train
4.Select a city

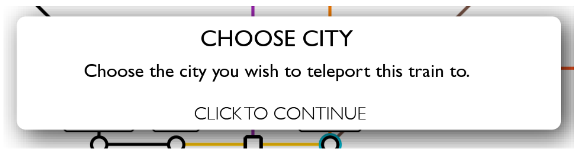
Snips of each step (from left to right):

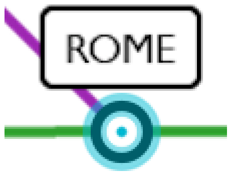





CHOOSE TRAIN
Choose the train you want Teleport.
CLICK TO CONTINUE



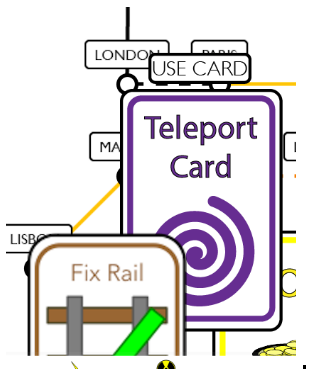

CHOOSE CITY
Choose the city you wish to teleport this train to.
CLICK TO CONTINUE

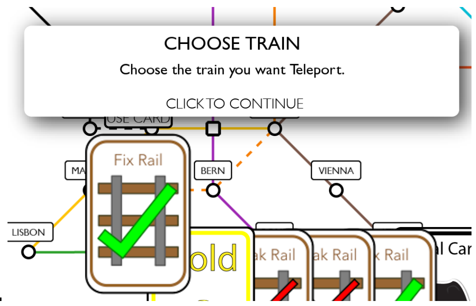


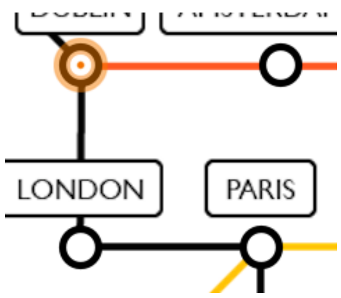
Results
The after applying the teleport card's magic to a train, the train appeared at the target station, had no current planned route and maintained functionality thereafter. Therefore the test was a success.

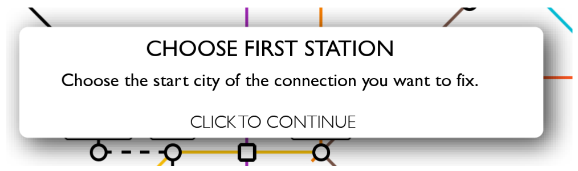
Further testing
Upon (or after closing) the prompt "CHOOSE TRAIN", a player may use another card (the card may be of any type) successfully, yet if it's use requires selection of a city then the teleport effectively disappears. This was even tested with GoFasterStripes card.

Here there was an orange train in Dublin (above London); it was selected for teleportation but then the fix rail was selected and used between London and Paris, the teleport was then no longer available.



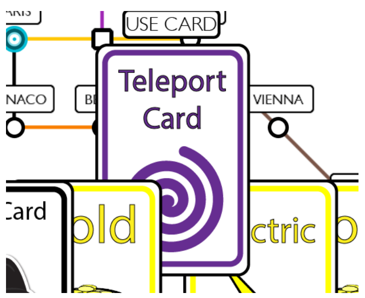

CHOOSE TRAIN
Choose the train you want Teleport.
CLICK TO CONTINUE

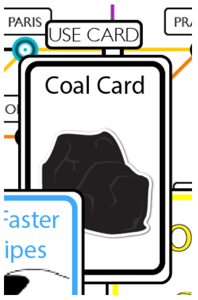



CHOOSE FIRST STATION
Choose the start city of the connection you want to fix.
CLICK TO CONTINUE

It may have been worthwhile to change this however the team felt that it was part of the strategy of the game that the player follows the prompts correctly.

However if the second card does not select a city then you maintain the ability to teleport, even though the player has ignored the prompt. This is a known bug, discovered by this test. Similarly a player can string together use of teleport cards yet only the last one works.





However by using GoFasterStripes card in a similar test we have also confirmed that GoFasterStripes card also works as intended.

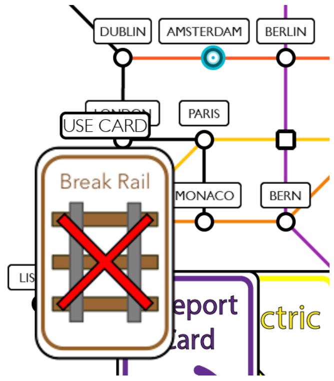
Fix/Break rail cards
show a break/fix card being used.
N.B Formatting note - the discussion of tests will follow the structure: describe/explain code/modifications, description of test and pass criteria, description of results and then the resolutions made from the results.

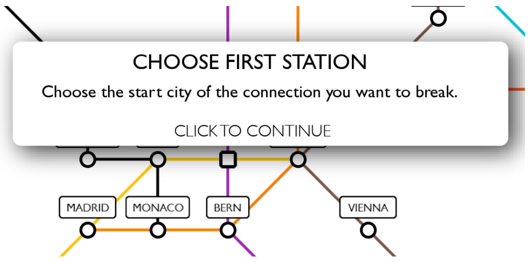
Test criteria

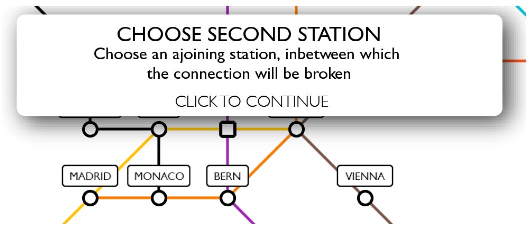
We had to implement this obstacles because of a change in requirements and we brought broken rail into the game by the existence of these cards, and thus we interpreted the requirements of the card: *the card must be able to make rail between two stations, untraversable.*

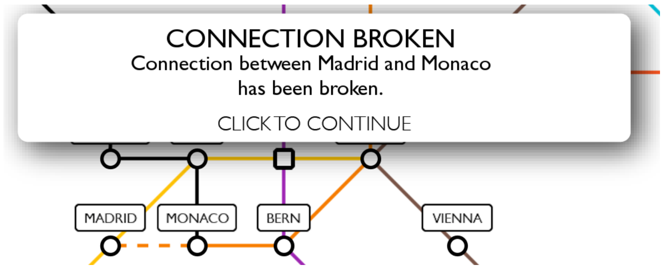
My algorithm for this black box test is as follows:
1.Purchase fix/break cards
2.Select card
3.Select a first city
4.Select a second adjacent city
5.Attempt to plan a route across the broken rail
6.Fix the broken rail
7.Traverse the fixed rail

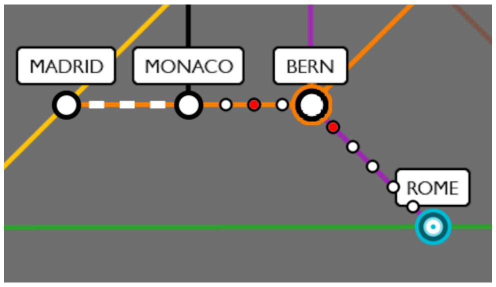
Here the rail between Madrid and Monaco was made to be broken, when selecting a train route it is not possible to select Monaco - Madrid

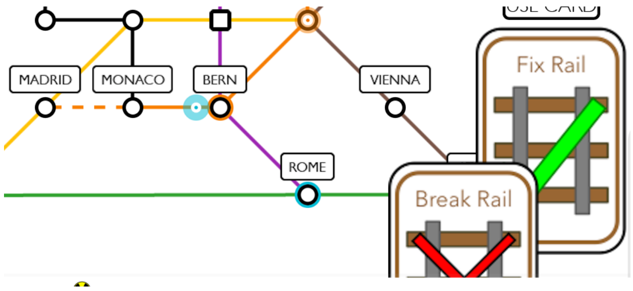


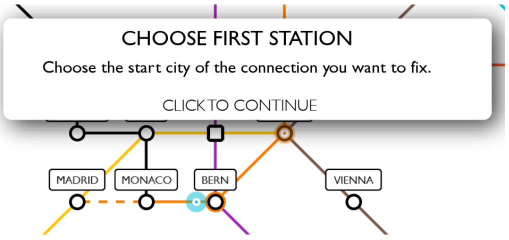

CHOOSE FIRST STATION
Choose the start city of the connection you want to break.
CLICK TO CONTINUE


CHOOSE SECOND STATION
Choose an ajoining station, inbetween which the connection will be broken
CLICK TO CONTINUE


CONNECTION BROKEN
Connection between Madrid and Monaco has been broken.
CLICK TO CONTINUE

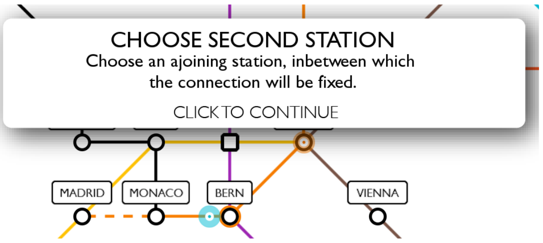





CHOOSE FIRST STATION

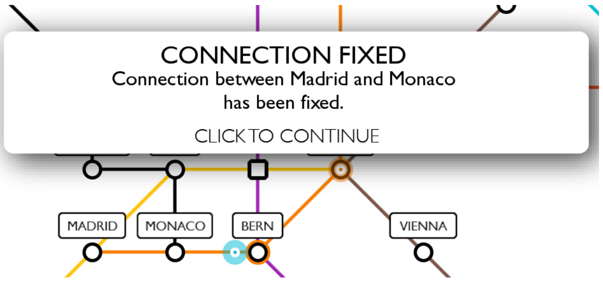
Choose the start city of the connection you want to fix.

CLICK TO CONTINUE


CHOOSE SECOND STATION

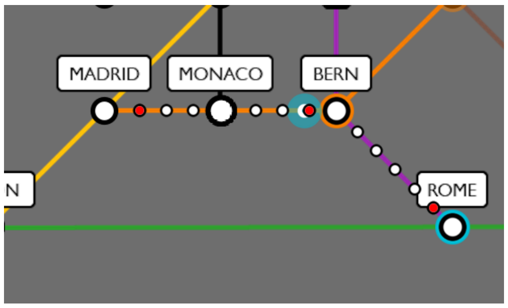
Choose an ajoining station, inbetween which the connection will be fixed.

CLICK TO CONTINUE


CONNECTION FIXED

Connection between Madrid and Monaco has been fixed.

CLICK TO CONTINUE



After using the fix rail card the rail Monaco-Madrid may be selected for routing by the blue train as normal

Results

The rail became untraversable after using the break rail card and then traversable after using a fix rail card, this meets both requirements of both cards and also successfully tests the broken rail feature. All three tests were successful.

It is also possible that a player selects a non adjacent city when prompted to to select an adjacent one, this does nothing.

Further testing

However upon testing .Also when breaking between a city and a junction you may select ‘backwards’ and effectively break the same rail twice. This causes the game to crash when attempting to fix it. But this is unlikely as a player wants to break as much rail as possible to maximise use of the break rail card so would rather break two different lengths of rail. With more time we could tackle this problem further and possible make it a game feature.

For cards in general it would usually be sensible to see what happens if a player was to be idle for a huge amount of time, for this we could write a JUnit test that simulates a large number of time steps however in our game there is a turn timer that forces a player to end their turn after 60 seconds so this does not need to be considered.

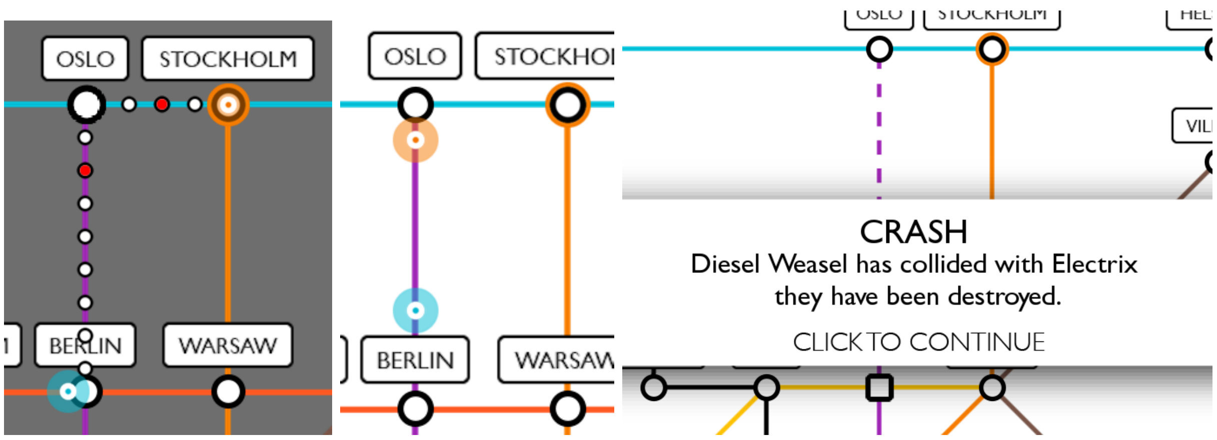
Although it was difficult to implement unit tests on the fix and break rail cards, it is straightforward to test that the breakConnection() and repairConnection(). This is done in two tests, the first one tests the breakConnection() method by iterating through all the cities on the map and calling breakConnection() on all of the connections. We then iterate through all the connections a second time and *assertTrue* that they are *untraversable.* We then set all the connections back to *traversable* using the repairConnection() method so that this test doesn't mess up any other tests.
The other unit test is on the repairConnection() method, we first break all of the connections using the breakConnection() method, which is already tested and working, and then fix all of them again using repairConnection(). We then *assertTrue* that all the connections are *traversable* again.

Train collision
Trains collide by connection their hit boxes if and only if they are, on the same piece of rail or one is in a city and one is on a connecting piece of rail.

I will test these conditions via in-game black box tests. One test where a train will be inside a station with the other approaching the station, and another test where they are on the same rail and collide head on. The direction of collision shouldn't matter as it is the connection of hitboxes that denotes a collision so a test for collision where one train catches up to another is unnecessary, also this is unlikely as neither player wants to lose their trains so would not try and intentionally plan a collision course which this kind of collision is likely to require.

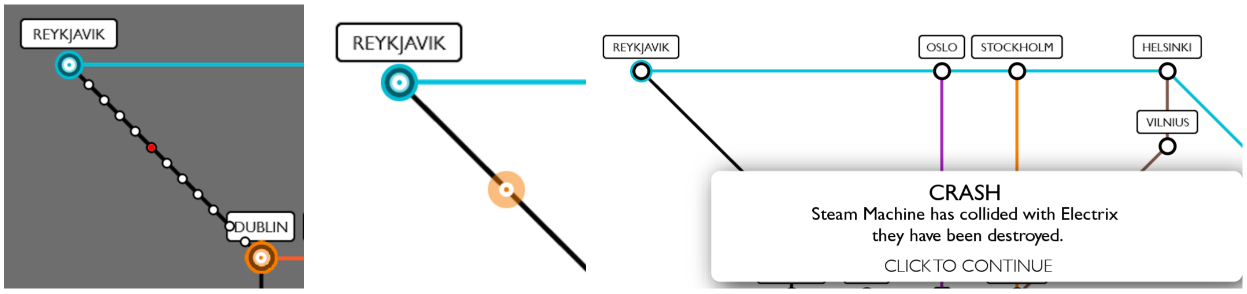
Test criteria
If trains on the same piece of track connect, they are both destroyed. If one train is inside a city and another train approaches from a connecting piece of rail they are both destroyed. If two trains come into collision range of each other yet are on different pieces of rail, this could be due to corners being awkwardly close together because of the map's design, then they do not collide.

Here the blue train is being sent to Oslo and the orange to Berlin, during orange's turn they collide; breaking the rail they're on.



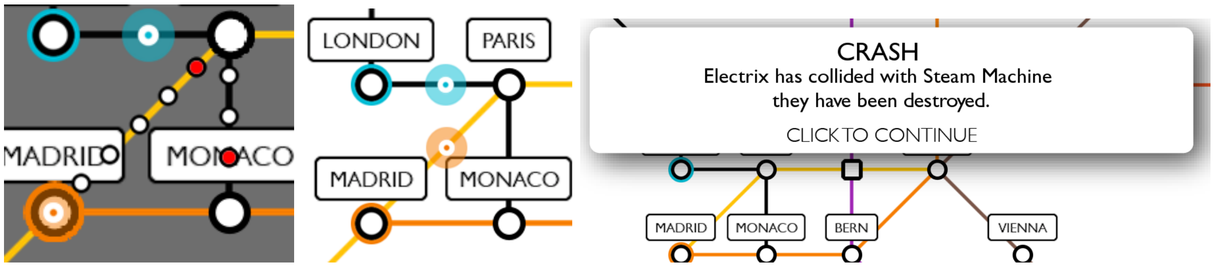
CRASH
Diesel Weasel has collided with Electrix
they have been destroyed.

Here the orange train is set to head into Reykjavik to collide with the blue train. Upon reaching collision range they crash.



CRASH
Steam Machine has collided with Electrix
they have been destroyed.

The orange train is set to collide with blue while neither train are actually on the same piece of rail. While clicking the end turn button fast enough to get simultaneous movement; it is difficult to tell whether they are colliding outside of the city or not.



CRASH
Electrix has collided with Steam Machine
they have been destroyed.

Results

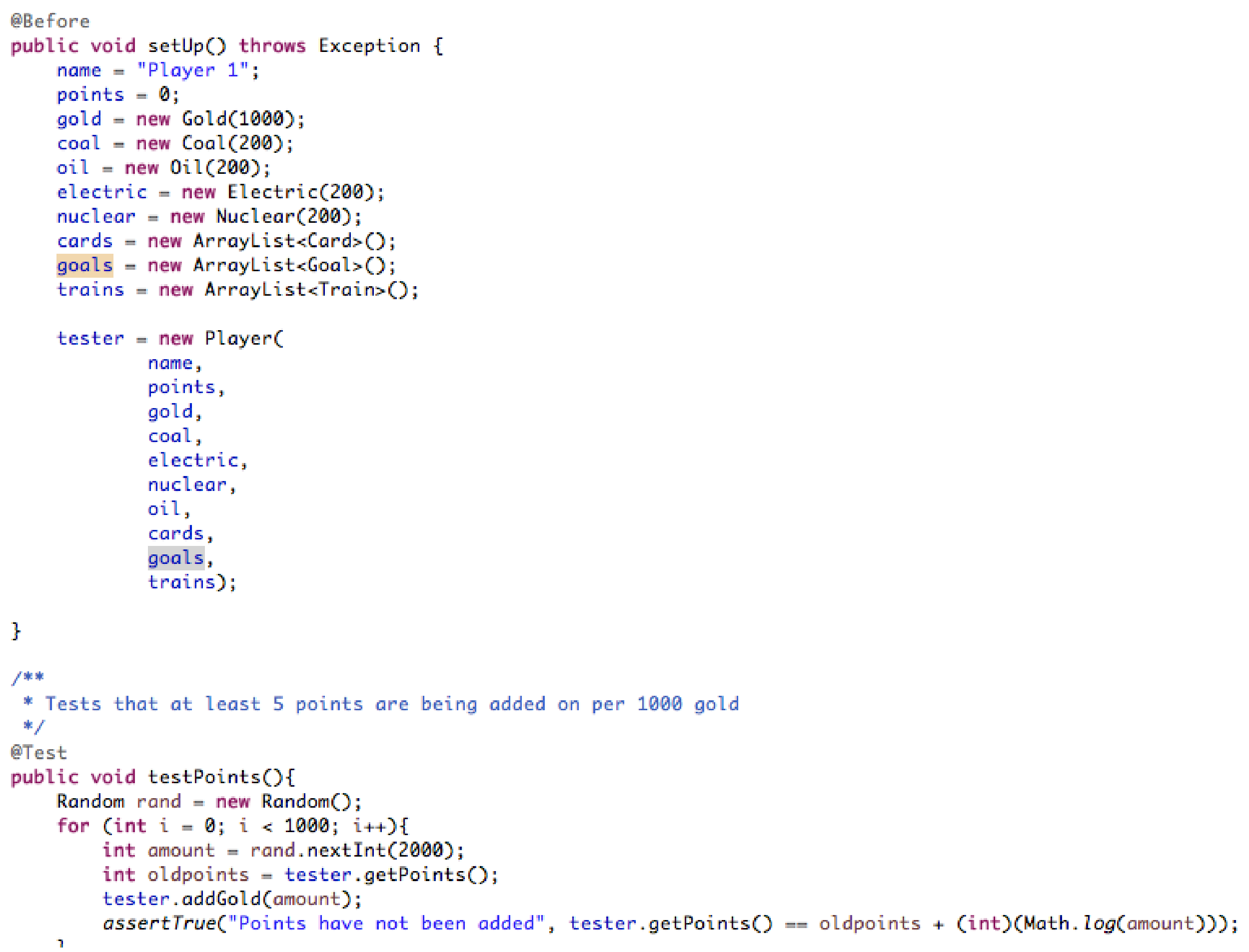
The head on collisions worked as planned; all trains were destroyed and rail was broken where intended. However it is clear that white box tests need to be performed to set two trains near each other as it is too difficult and inaccurate to do it by hand.

Further testing

When performing the head on collision while quickly clicking the end turn button for both players (making the trains move simultaneously) the collision still occurs and properly interrupts the turn with the prompt.

Points from earning gold

This was tested with a unit test that creates a new Player and then repeatedly adds between 0 and 2000 gold to that players gold attribute. This is done 1000 times and every time this happens there is an *assertTrue* statement to check whether the points are as they should be, which is the old number of points added to $\log_e$ of the amount of gold given.



```java
@Before
public void setUp() throws Exception {
    name = "Player 1";
    points = 0;
    gold = new Gold(1000);
    coal = new Coal(200);
    oil = new Oil(200);
    electric = new Electric(200);
    nuclear = new Nuclear(200);
    cards = new ArrayList<Card>();
    goals = new ArrayList<Goal>();
    trains = new ArrayList<Train>();

    tester = new Player(
            name,
            points,
            gold,
            coal,
            electric,
            nuclear,
            oil,
            cards,
            goals,
            trains);

}

/**
 * Tests that at least 5 points are being added on per 1000 gold
 */
@Test
public void testPoints(){
    Random rand = new Random();
    for (int i = 0; i < 1000; i++){
        int amount = rand.nextInt(2000);
        int oldpoints = tester.getPoints();
        tester.addGold(amount);
        assertTrue("Points have not been added", tester.getPoints() == oldpoints + (int)(Math.log(amount)));
    }
```