

**Team LYS**

# **TaxE Game**

## **Software Quality Assurance Plan**

TABLE OF CONTENTS

1. INTRODUCTION..... 1

2. TEST ITEMS ..... 3

3. FEATURES TO BE TESTED ..... 3

4. FEATURES NOT TO BE TESTED..... 4

5. APPROACH..... 4

6. PASS / FAIL CRITERIA ..... 5

7. TESTING PROCESS ..... 5

8. ENVIRONMENTAL REQUIREMENTS..... 7

9. CHANGE MANAGEMENT PROCEDURES..... 8

10. PLAN APPROVALS ..... 8

# 1. INTRODUCTION

## 1.1 Objectives

The objective of this test plan is to ensure confidence in the correctness and usefulness of the project deliverables (namely the game product), where correctness responds to the functionality of the game components and usefulness refers to its meeting of the requirements.

This test plan has been written for the Trains Across Europe (TaxE) game that team LYS has adopted for this section of the assessment. The code provided by team FVS is assumed to be functioning correctly and well tested. All tests written and undertaken for this assessment will solely be for the sections of code written by team LYS.

## 1.2 Testing Strategy

As test driven development is going to be used during this project, tests will be written for every module, and then modified, until the module passes the tests requirement.

Unit tests will be created for all necessary functions; however some of the functions created can only be tested via system/interface testing. If unit tests have been created, it is likely that integration tests will also be needed to ensure that the functions interact correctly with the previously written code.

System/Interface testing will ensure that all necessary system requirements have been met through the modifications made to the code during this assessment.

Regression testing will also be used frequently due to this product undergoing test driven development. After every test has failed, the code will be edited and the test will be rerun to determine if the outcome has changed accordingly; this process will repeat until the anticipated outcome is achieved.

## 1.3 Scope

As the product is undergoing test driven development, unit tests will be written for every piece of code that affects some recordable value. These unit tests will be modified after every change made to the code.

Tests will also be written, in the form of test cases, for every piece of code that affects some part of the UI. These tests will need to be repeated after every modification of the code.

As such, testing of this product is a continuous procedure and will need monitoring regularly.

### 1.4 Reference Material

The user and system requirements, and game design concepts are all available to download at:

<http://www-users.york.ac.uk/~oeh503/fvs/>

This STP was created from a template made by adapting the IEEE standard for software testing.

IEEE Standard for Software Test Documentation, IEEE Standard 829-1998, (2009)

For formatting purposes, an example of a test plan was used; it is available at this address:

[1]

[https://www.google.co.uk/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0CEMQFjAA&url=http%3A%2F%2Fwww.se.rit.edu%2F~royalflush%2Fdocuments%2FTestPlan.doc&ei=jm-6VMjSHIGqUcyohLgN&usg=AFQjCNHClWmSWrhXdkBD7Glo\\_Si3HdjUzw&sig2=nDC4dZOMJAxqG0Kf8s4FPA](https://www.google.co.uk/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0CEMQFjAA&url=http%3A%2F%2Fwww.se.rit.edu%2F~royalflush%2Fdocuments%2FTestPlan.doc&ei=jm-6VMjSHIGqUcyohLgN&usg=AFQjCNHClWmSWrhXdkBD7Glo_Si3HdjUzw&sig2=nDC4dZOMJAxqG0Kf8s4FPA)

Several definitions in relation to testing and risks were taken from the following two books:

[2] R. S. Pressman, *Software Engineering A Practitioners Approach Seventh Edition*. Singapore: McGraw-Hill, 2010.

[3] I. Sommerville, *Software Engineering Ninth Edition*. Boston, MA: Pearson, 2011.

### 1.5 Definitions and Acronyms

STP refers to Software Test Plan

UI refers to User Interface

### 1.6 Template

A template has been used in the creation of this test plan so that it conforms to the standard IEEE format. As the creator of the test plan had no previous experience with creating such a document, this template allowed for all necessary inclusions to be made and restricted the amount of unnecessary information included.

## 2. TEST ITEMS

### 2.1 Program Modules

User Interface – Manual execution of a set of tests will be performed upon all aspects of the user interface to ensure that these aspects respond in the manner expected [1]. These tests will cover all aspects of the software requirements that have been met within this assessment. All previously met requirements are assumed to have been tested previously.

Game – A series of unit tests and integration tests will be written to test that all aspects of the Game module provide the results expected. The expected results for each test will be determined by the developers beforehand depending upon the inputs for said test. All previously written code is assumed to have been tested previously.

### 2.2 Job Control Procedures

As no job control procedures have been used within the project there are no testing plans in place.

### 2.3 User Procedures

The user documentation, i.e. the User Guide, will be reviewed to determine whether it is accurate in terms of the actual implementation of the game. The user documentation will also need reviewing to determine whether it is complete with regards to explaining all interactive components of the game.

### 2.4 Operator Procedures

The game needs to be run upon both a Linux and a Windows operating system to ensure that it can be run in the specified environments. It is not possible to test the product in all environments as every runtime environment is different depending upon software installed upon the hardware.

## 3. FEATURES TO BE TESTED

- |                       |                                               |
|-----------------------|-----------------------------------------------|
| 1. Obstacles          | -- related system requirements 4c, 5e, 6b     |
| 2. Scoring            | -- related system requirements 3a, 3d, 3g, 4b |
| 3. Quantifiable Goals | -- related system requirements 3b, 3c         |

### 4. FEATURES NOT TO BE TESTED

All other features within the system requirements were determined to have either been implemented in the previous iteration of the game, or to be unnecessary in the production of the game. Some features were determined to be unnecessary to implement, as the three features specified above were necessary for the completion of the assessment briefing, and the time limit provided prevented further features being implemented.

### 5. APPROACH

#### 5.1 Component Testing

Component testing, also known as unit testing, focuses upon the smallest unit of software design – the software component or module [2,p.456]. Each team member will write unit tests for the class, or module, that they are working upon. These tests must verify all possible states of the module (all possible input types) and check the value of all attributes that are associated with the module.

Most of the features to be tested only affect the GUI and not any internal records; as such, they cannot be tested using unit tests. Only the quantifiable goals will be tested using unit tests.

#### 5.2 Integration Testing

Integration tests need not be written for these features, as the only existing modules that use the quantifiable goal features, call them through a random probability system. Therefore it is not possible to test the modules.

#### 5.3 Interface Testing

As it has been agreed that the game will be downloaded and run as an executable file, not run within a web page, it is not necessary to perform interface testing in terms of interactions between the game and external interfaces.

The UI will be tested to compare the functionality of the game, with the system and user requirements, in order to ensure that these requirements have been met. These tests will be undertaken in a series of test cases which will have the related requirement stated, screenshots of the game will be provided (where possible) as evidence that the requirement has been met.

#### 5.4 Performance Testing

Performance testing will be used to test the resource usage of the game.

### 5.5 Regression Testing

Upon making modifications to the system, tests must be re-executed for all components that this modification is related to (directly or indirectly). These tests ensure that this modification has not adversely affected the purpose of the component [1].

Regression testing will be evidenced where a test has failed, code has had to be modified, and the retest has passed. The expected and actual outcomes will be stated for the cases in which the test fails.

### 5.6 Acceptance Testing

Acceptance testing must be undertaken by the customer, however, due to time constraints the system is not ready to be tested by the customer and as such must be submitted without this testing.

## 6. PASS / FAIL CRITERIA

### 6.1 Suspension Criteria

The execution of the tests will only be suspended if some critical failure is detected that will impede the ability to perform all associated tests [1].

### 6.2 Resumption Criteria

These tests will only be resumed if the developers believe that the problem that caused the critical failure has been resolved [1]. All tests will be re-executed to assure that the developers fix has not adversely affected the rest of the script components.

### 6.3 Approval Criteria

The test results will be approved if they meet the expected results in the test description.

## 7. TESTING PROCESS

### 7.1 Test Deliverables

A test report will be part of the test deliverables. This report will include a series of test cases which provide evidence of each type of testing that was undertaken upon the game script. These test cases will include a short description, a test type, the related test requirement if applicable and the status of pass or fail.

### 7.2 Testing Tasks

- Develop scripts for automated testing
- Write test cases for system/interface testing
- Execute tests
- Report defects
- Complete test report
- Manage all changes that are made to the tests, script and test plan


### 7.3 Responsibilities

The team members who have written sections of code will be responsible for creating the unit tests for their own code modifications. These team members will also run the game to test that their changes have been implemented correctly; they will make a record of these tests and any issues that they may have encountered. The team tester will run all system tests and compare them to the related system or user requirements.

### 7.4 Resources

There are very few resources that will be used in the testing of this project aside from the team members who will be manually testing the scripts and writing the script tests. One of these resources is JUnit which will be used to create the skeleton for the unit tests for each module. Other resources used include the libraries from both the Gradle and Libgdx frameworks which were employed to ease the implementation of the graphical aspects of the game.

### 7.5 Schedule

<i>Task</i>	<i>Deliverable</i>	<i>Week Performed</i>	Iterative Process 
Develop scripts for automated testing		Continuous (as programming is underway)	
Write test cases for system/interface testing		Continuous	
Execute tests		Continuous	
Report defects		Continuous	
Complete test report	Test Case Report	Final Week	



## 8. ENVIRONMENTAL REQUIREMENTS

### 8.1 Hardware

- No specific hardware requirements necessary

### 8.2 Software

- 1 – Java IDE with JUnit testing software
- 1 – Gradle build system to manage external libraries and set the test, source and assets correctly so that the IDEs can identify them
- 1 – Libgdx Java game development frame work

### 8.3 Security

There are no security requirements specified in the system requirements provided for this product.

### 8.4 Tools

JUnit will be used in order to write the unit tests for the system. The Gradle and Libgdx libraries will also be used in the creation of this system.

### 8.5 Publications

No publications have been used to support testing purposes.

### 8.6 Risks and Assumptions

As testing can only show the presence of errors in a program and not the absence of errors, the number of tests that can be run upon a program is infinite [3,p.231]. Whilst a vast number of tests should be efficient to encounter nearly all errors, due to a restricted amount of developers and time, the number of tests that can be run is less than would be potentially optimal. Therefore, there is always a risk of new errors that could affect the system at runtime. As a result, many test cases will be run using the GUI to try and detect any potential runtime errors.

The libraries from both the Gradle and Libgdx frameworks have been relied upon to ease the implementation of many of the graphical aspects of the game; as such we were unable to test details within these external libraries.

It was also assumed that the code provided to us by team FVS had been fully tested and proven to be working without errors.

### 9. CHANGE MANAGEMENT PROCEDURES

Updates can be made to the STP at any time however unscheduled changes must be proposed to the team tester either in person or via email. (The team tester is the team member who writes the STP and as such knows the format of the plan and where any changes should be made.)

The team tester will gather the team to discuss the proposal; this proposal will then be accepted, rejected or accepted with modifications.

If the proposal is accepted, it will be implemented, along with any modifications, and added to the new STP by the team tester.

Scheduled changes to the STP will occur: immediately after every product change during the assessment, half way through each assessment time frame and three quarters of the way through the assessment time frame.

After every change, the new version of the STP will be made available upon the game distribution site.

### 10. PLAN APPROVALS

<i>Name</i>	<i>Signature</i>	<i>Date</i>
Akeem Brown		
Arushi Aneja		
Cameron Miller		
Ed Brayshaw		
Holly Hendry		
Nick Michael		