Team-Project Test Report

Team ***, University of Birmingham 2014

Abstract

This is the test report for Team *** at the University of Birmingham team-project module 2014. It states all aspects of the testing that were completed over the course of the project. It contains a test plan, test cases, acceptance testing and unit testing.

Contents

1 Test Plan

1.1 Introduction

This test report documents the testing of the game developed for team project. It is used to make sure the team test the functionality of the game regularly, hence to ensure the game is playable and meets the high-level user requirements of the game throughout the course of the project. The tests refer to the software specification requirement of the game. This document documents testing of all the functionalities of the game, including the menus for setting up the game and networking, the tutorial, the running of entire game in singleplayer and multiplayer mode, the networking and the AI elements implemented in the game.

1.2 Test Items

The items to be tested is based on the user's point of view, the function and features that the game should provide. The following describes all the items that have to be done:

Test Item	Test Description	Test Date	Tester
Map	The test ensures that the user is able to click on certain buttons and change aspects of the game	13/02/2014	*****
Main Menu	The test ensures the user enters the game properly and has access to the other game menus.	26/02/2014	*****
Map Menu	The test ensures players can place towers on the map during the game.	20/03/2014	***** *****
Map Editor	The test ensures the user can create their own map.	17/03/2014	*****
Single player Menu	The test ensures the user can choose their role in the game.	26/02/2014	*****
Multiplayer menu	The test ensures the user can choose to host or join the game in a multi-player game.	26/02/2014	*****
Option Menu	The test ensures the user can customise the game.	30/03/2014	*****
Lobby Menu	The test ensures users can join the game and display properly in the lobby menu	24/02/2014	*****
Units	The test ensures the user can buy units.	26/02/2014	*****
Tower	The test ensures the user can upgrade and sell towers.	13/03/2014	*****
Chat	The test ensures that the chat client is working .	25/02/2014	*****
Tutorial	The test ensures the tutorial runs properly.	19/03/2014	*****

1.3 Approach

The game testing and JUnit testing were performed on each component as they were developed. The code for a specific function was executed to ensure the function returned the expected result.

The testing was managed both by the team member who wrote the code and tested by the tester to ensure it worked as expected. All the tests are fully documented in the test cases (see Appendix A). If the test failed, the tester reported the failure to the members who were involved in that function of the game. After the bug was fixed, it was tested again by the tester.

1.4 Item Pass/Fail Criteria

For each test case the team stated the expected outcome (what should happen). The pass/fail criteria is based on whether the test cases met this criteria.

2 Test Cases

Throughout the course of the project, the team conducted numerous amounts of test cases for each package and feature of the game. These were then documented in spreadsheet format as test cases. These test cases were there to make sure that all possible errors that might occur in the run time of the game were found, and that they were dealt with appropriately. 12 spreadsheets full of test cases were produced, which can be found in the appendices. Below is an example of one of the spreadsheets of testing that was conducted.

2.1 Example Test Cases

Test Case ID	Input	Expected Output	Output	Pass?	Date tested
TC_ME_01	Mouse click in the bounds of the grass tile button. Then clicking on the map over a grass tile.	Nothing will show up as the two tiles are the same.	Nothing happened.	y	17/03/2014
TC_ME_02	Mouse click in the bounds of the grass tile button. Then clicking on the map over a different tile type.	The grass will overwrite the other tile type and show up under the mouse.	Grass tile overwrote the path tile.	y	17/03/2014
TC_ME_03	Mouse click in the bounds of the path tile button. Then clicking on the map over a different tile type.	The path will overwrite the other tile type and show up under the mouse.	Path tile overwrote the grass tile.	y	17/03/2014

TC_ME_04	Mouse click within the bounds of the open button. While there are no custom maps in the Maps folder.	The list of custom maps will show up with no items in the list.	No items were shown in the list.	y	17/03/2014
TC_ME_05	Mouse click within the bounds of the open button. While there are 3 different custom maps in the Maps folder.	The three custom maps will show up in the list of maps to be chosen.	The list of three maps showed up.	y	17/03/2014
TC_ME_06	Selecting a custom map to be loaded from the list.	The map will be read and be rendered on the map screen.	The selected map was rendered onto the map screen.	y	17/03/2014
TC_ME_07	Clicking the cancel button instead of loading a map.	The confirm_dialog will close and show the mapCreator.	Map creator was shown.	y	17/03/2014
TC_ME_08	Mouse click in the bounds of the save button. With an invalid map.	A error message should show asking the user to create a valid map before saving.	Error dialog was shown.	y	17/03/2014
TC_ME_09	Mouse click in the bounds of the save button. With a valid map.	Input box should show up and allow the user to type in their name for their custom made map.	Input box dialog was shown.	y	17/03/2014
TC_ME_10	Mouse click in the bounds of the undo button once a path tile has been placed on the map.	The path tile should dissapear.	Path tile was removed.	y	17/03/2014
TC_ME_11	Mouse click in the bounds of the redo button once a path tile has been drawn and then undone.	The path tile should reappear.	Path tile was replaced.	y	17/03/2014

TC_ME_12	Mouse click in the bounds of the clear button.	A confirm dialog should show up asking the user to confirm the clearing of the map.	Confirm dialog was shown.	y	17/03/2014
TC_ME_13	Clicking the confirm button inside the confirm dialog of the clear map button.	The map should be cleared and be placed back to the default look with just grass tiles and two HQs.	Map was reset.	y	17/03/2014
TC_ME_14	Once selected a path tile type, clicking and dragging the mouse along the sides of the map screen.	Nothing should show up as the user cannot place path onto the sides of the map.	Nothing showed up.	y	17/03/2014
TC_ME_15	Mouse click in the bounds of the valid map button with an invalid map on screen.	A message dialog should appear stating to the user that the map on screen is not a valid map.	Correct message dialog was shown.	y	17/03/2014
TC_ME_16	Mouse click in the bounds of the valid map button with a valid map on screen.	A message dialog should appear stating to the user that the map on screen is a valid map.	Correct message dialog was shown.	y	17/03/2014
TC_ME_17	Mouse click in the bounds of the help button.	A help message should be shown, prompting the user to hover over buttons to see what they do.	Message was shown to the user shown next to the help button.	y	17/03/2014
TC_ME_18	Once the help button had been clicked moving the mouse over each individual button.	Message dialogs should show up next to the button being hovered over.	Message was shown to the user shown next to the corresponding button.	y	17/03/2014

TC_ME_19	Once the help button had been clicked. Clicking on it again and then moving the mouse over buttons.	The message dialogs should no longer show up.	No message dialogs were shown when the mouse hovered over the buttons.	y	17/03/2014

2.2 Test Case Statistics

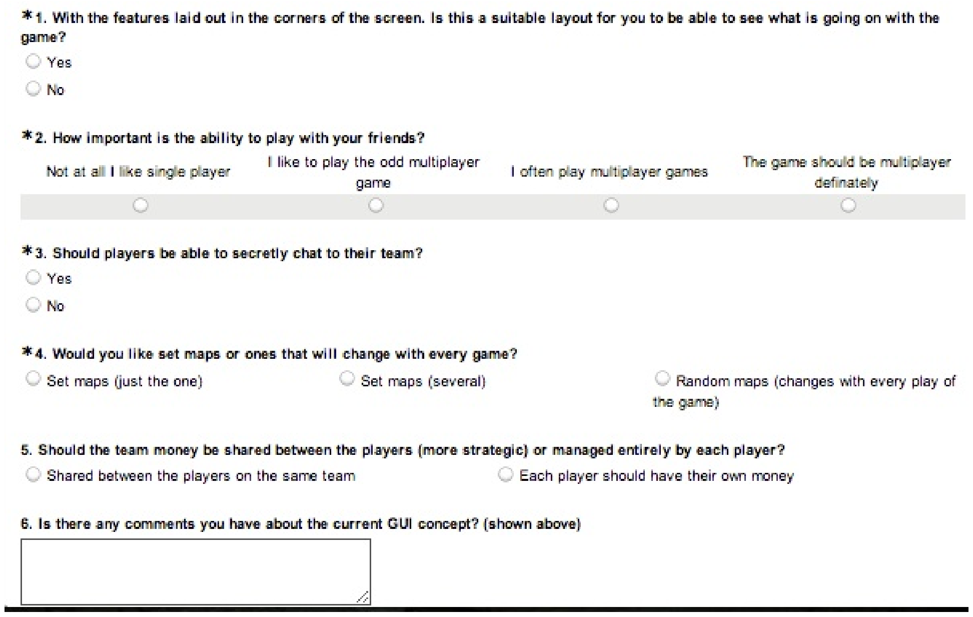
A collection of all test cases conducted are available in the appendix. Please see appendix A in the Appendix.pdf document.

- 12 Spreadsheets full of test cases
- Over 150 different test cases covering the whole project spread
- All test cases passed

3 Usability Testing

The team all agreed that usability testing would be a key mechanism to ensure that the target audience found the game enjoyable and easy to use. As such, a consistent usability testing scheme throughout the duration of the project was decided, allowing constant review of the game to improve the overall quality.

An early example of this can be seen once the team had solidified the GUI concept and some of the initial plans for the game. To ensure the game was easy and pleasant to use, the team compiled the final GUI concept and other relevant questions regarding the gameplay into an online questionnaire which was circulated to the appropriate target audience. Some of the results of this questionnaire are displayed in the figure below.



*1. With the features laid out in the corners of the screen. Is this a suitable layout for you to be able to see what is going on with the game?

◯ Yes
◯ No

*2. How important is the ability to play with your friends?

Not at all I like single player	I like to play the odd multiplayer game	I often play multiplayer games	The game should be multiplayer definately
◯	◯	◯	◯

*3. Should players be able to secretly chat to their team?

◯ Yes
◯ No

*4. Would you like set maps or ones that will change with every game?

◯ Set maps (just the one) ◯ Set maps (several) ◯ Random maps (changes with every play of the game)

5. Should the team money be shared between the players (more strategic) or managed entirely by each player?

◯ Shared between the players on the same team ◯ Each player should have their own money

6. Is there any comments you have about the current GUI concept? (shown above)

Figure 1: GUI User Acceptance Survey

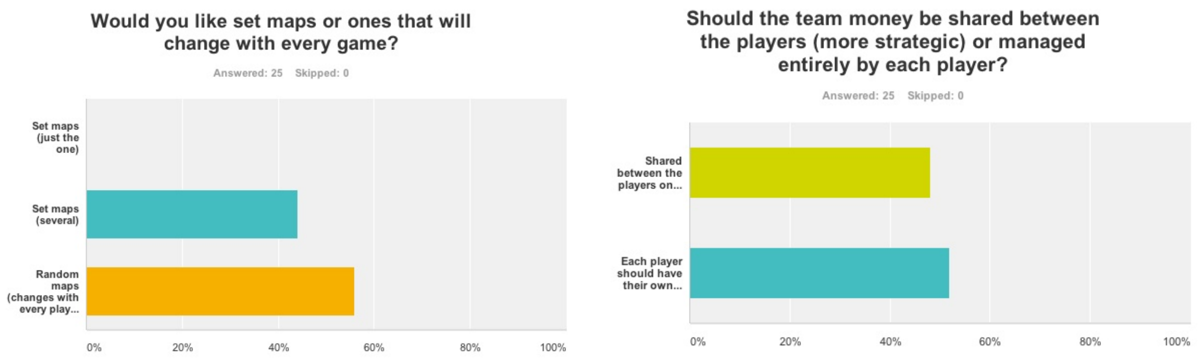


Figure 2: Analysis of User Acceptance Survey

Following the results from this questionnaire, the team reviewed the feedback and made any modifications if necessary. For example, the questionnaire revealed that all who completed it wanted at least multiple maps to play on. However, there was a fairly even split between having set maps and random maps. As such, the team agreed that both functionality was important to include so adapted the work breakdown structure appropriately to include both these features.

Other results such as whether money should be shared between players on a team revealed quite a surprising result to the team. More people wanted each player to have their own gold as opposed to it being shared between the players on a team. However after careful consideration, it was agreed that sharing gold between players on a team greatly improved the cooperative aspect to the game and as such decided to continue with the initial plan of having gold shared, despite the result of the acceptance test.

During the core development of the game, usability testing was put on hold to allow complete development to complete as a priority. However, once the game was in a playable state usability testing was reintroduced to allow review of the features implemented.

For example, once the game could be played in a single player fashion, team members allowed their peers to play the game and review their experience with it. This provided brilliant insight as to any changes that might be necessary. One thing that was revealed is that the game was fairly unbalanced at the beginning. Buying certain units or towers often guaranteed a win for the team, no matter what the opponent tried to do. As a result, we reviewed the balancing issues, made some changes to the power of units and towers and restarted usability testing to see if this made the game more suitable for the target audience. This process was continued up until almost completion of the game which ensured ever improving quality of gameplay for the end user.

This testing method also greatly helped to implement the AI functionality, since how human players actually played the game was noted and reviewed so that the AI could try to emulate the same strategies.

The usability testing the team employed throughout the duration of the project turned out to be very successful in reviewing how the game covered the five aspects of usability.

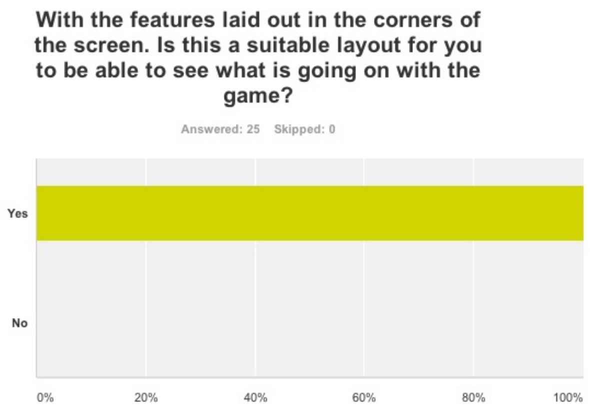
Learnability was covered through testing how the tutorial allowed the player to learn the game, along with how easy the game was to pick up after playing a couple of games. Updates were made according to any issues users had with learning the game, such as improving the information provided in the tutorial or ‘help’ areas within the program.

Efficiency was tested by watching users play the game. It was noticed that after players had got used to the game that clicking on individual buttons for units and towers became too repetitive and so hotkeys were added to allow more advanced users to play the game quickly and easily.

In terms of memorability, it was reviewed how easy new users to the game could navigate the menu systems and play games. Due to the easy to use menu system as a result of prioritising HCI, there were no issues with users trying to navigate the menus or play games, even in the case of intermittent use by casual/new users.

Checking for errors was essentially done by our own testing techniques when developing the actual game. Any functionality that was put into the game would be rigorously tested to ensure stability of the new feature. However the usability testing to test previous aspects also covered any errors which might occur as they were documented and prioritised if a user experienced an issue.

The satisfaction of the game was under constant review during these usability tests as well and actual statistics can be seen in the questionnaire asked at the start. In terms of actually recording complete statistics for the satisfaction with the game after completion, due to time constraints we did not compile actual statistics however in the demos we gave to our target audience, all users agreed that the game was fun to play, the graphics were brilliant and the sound effects made a fantastic addition.



4 Unit Testing

JUnit tests can be found in the svn repository under src/jUnitTests. Unit tests were written for critical parts of the game, and cover all major classes:

- Entity
- Game
- GameServer
- Map
- MapFileReader
- RandomMapGenerator
- Towers
- Projectile
- TutorialBox

An interface was also written so that custom tests could be run through GameContainer in the main game loop. This gave us greater freedom in the nature of tests we could easily write. All OpenGL-related code was also modified so that rendering could be switched off before a test was run, allowing classes not related to the game loop to be run without an OpenGL context.