Testing Documentation

Software Engineering Group 6

11/6/2012: Testing Documentation, v2.0

May 2012 - Fourth Deliverable

Actions: Finalised

Contents: **Page No:**

Introduction

Testing is an essential and important part of Software Engineering as it may be the difference between a working and bug free product to an incomplete non-executable program. Within this document, a series of documentation tests for how the source code for the Ant Game has been produced will be recorded in a series of categorical testing.

There will be test phases that the source code will have to go through and both the Quality Assurance Team and Programming Team will be working collaboratively to document all of the necessary and relevant tests that would be advisable and relevant to the overall Ant Game JavaScript program.

Throughout the testing, an account of what was said in the first deliverable on the basis of testing shall be applied to this document. Therefore, the classifications of each type of testing shall be integrated and accounted for i.e. White Box Testing and Black Box Testing will be mentioned within this fourth deliverable.

The main purpose of this deliverable is to ensure that the program is refined and full proofed with quality tests from aesthetics, functionality and perhaps fail safes have been implemented into the source code. This testing document shall be evidence of the working and final version of the Ant Game developed by Team Good, group 6 of the Software Engineering course.

Testing Scope

The series of tests that are to be applied to the source code will be of a thorough and critical assessment of the entire Ant Game developed by Team Good. The examination of its functionality, performance, aesthetics and efficiency as a program will be looked through where an analysis of its major achievements and drawbacks shall be highlighted throughout the progression of testing. In doing so, what has been specified within the Customer Specification (the first deliverable) shall be correlated to what will be said throughout this document. Hopefully, all testing specifications that have been pointed out will be documented and recorded within this to fully gain an understanding of how well the Ant Game has been developed in the specific high level language of JavaScript code.

We must firstly take into consideration that the Programming Team has already developed tests throughout the process of programming the Ant Game. Therefore the testing document also has the purpose of representing those tests in a formal and user friendly manner. As the development of the tests has been designed through the development of the source code, the Quality Assurance Team needs to ensure that these tests are valid and relevant and actually serve a purpose for testing all three elements of functionality, aesthetics and performance.

As the tests have been developed throughout the programming, every aspect and process of development has been critically looked over for its contribution to the final development of the Ant Game. Overall, this testing shall aid both the Programming Team and Quality Assurance team that no errors have been overlooked and a final analysis of the software is assured of an error free state for submission.

Key Criteria to be tested are:

- The correctness of the parsers
- The suitability of randomly generated worlds
- The correctness of the simulation (i.e. do the ants do everything that they should?)
- The functionality of the GUI (i.e. does it allow the user to do everything they need to do with respect to the customer requirements.)
- The suitability of the GUI (e.g. how responsive is it? Does it work on all modern browsers?)

Test Plan

For this section of the Testing deliverable, all of the specified types of tests shall be broken down into a hierarchical level of phases that can be applied to the source code. These phases will describe the tasks and overhead software that incorporates the types of testing such as: ‘White Box Testing’ and ‘Black Box Testing’.

Firstly, the description of each variable element needs to be looked at, these variables will be presented and briefly described for each phase of the testing. The variables will be placed in an organised fashion within a table. These elements:

- Test Phases: A description of the phases and tasks involved with each test.
- Overhead Software: A phase may require overhead software.
- Test Procedures: A description of what needs to be carried out within the test.

Moreover, the data and the formalities that shall be recorded will include:

- Test Descriptions: A description of what tests are to be carried out in reference to the phases.
- Overhead Software Description: A description of the software that aids the test shall be described.
- Expected Results: A description of the successful criteria of the test, met or not met.
- Test Case Data: A description (state of name) of data files used for the specific test based on the phases accounted for.

Testing is split into three distinct phases:

- Unit Testing
- Integration Testing
- Acceptance Testing

Each of these may involve the following strategies:

- White Box Testing

  Ensuring that all possible branches of computation have been executed.

- Black Box Testing

  Ensuring that functions/modules behave correctly

Phase 1. Unit Testing

Unit testing is carried out using the *Nodeunit* testing framework for node.js.

Phase 1.1. The Brain Parser

Function to Unit Test	Description	Required Scaffolding
_parseInt	Converts strings representing integers into numbers.	none
_parseLine	Parses a single instruction	Sample instructions, and expected outputs for deep comparison.
parseAntBrain	Parses an ant source file	Sample source files, and expected outputs for deep comparison

Phase 1.2 The World Parser

Function to Unit Test	Description	Required Scaffolding
_parseGridLine	Parses a single grid line from world source file.	Sample grid lines, and expected outputs for deep comparison.
_isSurroundedByRock	Checks whether a given grid has a solid border of rocks	A sample grid
_gridContains	Searches a grid for a specific cell type	A sample grid
_getElements	Returns a list of 2D arrays which represent the shape of elements of some specified target type. An ‘element’ here is a contiguous region of one particular cell type	A sample grid, and expected outputs for deep comparison
_getElementBox	Takes a list of coordinates and places them in a 2D boolean array (“box”), bouding them in the process.	Some coordinates and expected outputs for deep comparison.
_getElementCoords	Returns a list of elements in the format of coordinates which comprise them.	A sample grid and expected outputs for deep comparison
_getAdjacentCoord	Gets the coordinates of a grid	None (just numbers)

	cell which is adjacent to a given cell in a specified direction.	
_containsLegalFoodBlobs	Checks whether a given box represents shapes which are legal food blobs	Sample box.
_attemptBoxIntersection	Tries to match a shape in a given box, and returns the box sans the shape on success.	Sample box, sample shape, expected output.
_cloneBox	Clones a box	A sample box.
_cropBox	Crops a box (removes blank padding)	A sample box, expected output
_isLegalHill	Checks whether the shape in a box represents a legal hill.	Sample boxes.

Phase 1.3. Random World Generator

Function to Unit Test	**Description**	**Required Scaffolding**
_superimpose	attempts to overlay a shape onto a grid.	Sample grid, sample shapes.
generateRandomWorld	Generates random worlds	parseAntWorld to check validity of generated worlds

Phase 2. Integration Testing

Integration testing is also carried out using *Nodeunit*.

Phase 2.1. The Simulation

Goal of the test:

> To make sure that the behavioural mechanics of the game function in accordance with the customer's requirements, and to supplement the unit testing of the core game components. This latter goal is included because unit testing the core game components would be an extremely protracted process, and we can quite safely assume that the individual components are correct if this test passes.

Required Scaffolding:

> The test requires running the game using the sample ant brain and tiny world provided by the customer. The output of the game is checked against the dump file also provided by the customer. In order to achieve this, a function to read the dump file state-by-state is required, and the ability to output the state of the the running simulation at each iteration in the same format to check against the data in the dump file is required. Also, an implementation of the customer's pseudo-random number generator is required.

Phase 3. Acceptance Testing

Acceptance Testing is carried out manually by the Quality Assurance team. Each test requires a build of the game.

GUI Attributes to Test

1. Root menu navigation
2. Single Match setup
    a. Picking Brains
    b. Picking a World
3. Contest Setup
    a. Picking Brains
    b. Picking Worlds
4. Adding/Modifying Brains
5. Adding/Modifying worlds during single match setup
6. Adding/Modifying worlds during contest setup
7. Generating random worlds
8. Running a single match with graphics
9. Running a single match without graphics
10. Running a contest with graphics
11. Running a contest without graphics

Test Procedures

Phase 1.1. The Brain Parser

note: these tests can be found in the file `test/model/AntBrainParser-test.js`

White Box Tests

1. **Function**

   `_parseInt`

   Description

   Checking that preceding zeroes are stripped correctly.

   Scaffolding

   none

   Expected Results

   ```
   _parseInt("00003") == 3
   _parseInt("00030") == 30
   _parseInt("000") == 0
   ```

2. **Function**

   `parseAntBrain`

   Description

   Checking that the parser works with windows newlines (`CRLF`).

   Scaffolding

   A syntactically correct brain with windows newlines.

   An expected result for deep comparison.

   Expected Results

   The parsed brain should be exactly the same as the expected result.

3. **Function**

   `parseAntBrain`

   Description

   Checking that the parser works with mac newlines (`CR`).

   Scaffolding

   A syntactically correct brain with mac newlines.

   An expected result for deep comparison.

   Expected Results

The parsed brain should be exactly the same as the expected result.

4. **Function**

   `parseAntBrain`

   Description

   Checking that the parser works with unix newlines (`LF`).

   Scaffolding

   A syntactically correct brain with unix newlines.

   An expected result for deep comparison.

   Expected Results

   The parsed brain should be exactly the same as the expected result.

5. **Function**

   `parseAntBrain`

   Description

   Test that an error is thrown when the brain has no states

   Scaffolding

   A file with a couple of lines but nothing on them. i.e. `"\n\n"`

   Expected Results

   An error is thrown.

6. **Function**

   `parseAntBrain`

   Description

   Test that an error is thrown when the brain has a syntax error

   Scaffolding

   A brain with a syntax error. See the code for details.

   Expected Results

   An error is thrown.

7. **Function**

   `parseAntBrain`

   Description

   Test that an error is thrown when a nonexistent state is pointed to

   Scaffolding

   A brain in which a nonexistent state is pointed. See the code for details.

   Expected Results

   An error is thrown.

8. **Function**

   `parseAntBrain`

   Description

   Test that an error is thrown when a marker number is not in the range 0-5

   Scaffolding

   A brain in which an illegal marker number is given.

   Expected Results

   An error is thrown.

Phase 1.2. The World Parser

note: these tests can be found in the file `test/model/AntWorldParser-test.js`

White Box Tests

1. **Function**

   `_parseGridLine`

   Description

   Check that it works for valid even lines

   Scaffolding

   valid even line "`# 1 5 . # 9 - + `"

   expected output for deep comparison

   Expected Results

   deep comparison should return true

2. **Function**

   `_parseGridLine`

Description

Check that it works for valid odd lines

Scaffolding

valid odd line " 1 # . + -"

expected output for deep comparison

Expected Results

deep comparison should return true

3. **Function**

`_parseGridLine`

Description

Check that it throws an error if odd and even lines are swapped

Scaffolding

valid odd and even lines from tests 1 & 2

Expected Results

An error should be thrown in each case

4. **Function**

`_parseGridLine`

Description

Check that it throws an error for invalid characters

Scaffolding

invalid odd line " 3 y . +"

Expected Results

An error should be thrown

5. **Function**

`_parseGridLine`

Description

Check that it throws an error when seeing an odd line with too many spaces at the beginning

Scaffolding

invalid odd line " 3 . # +"

Expected Results

An error should be thrown

6. **Function**

`_isSurroundedByRocks`

Description

Check that it returns true for a grid that is surrounded by rocks

Scaffolding

mock grid

Expected Results

It should return true

7. **Function**

   `_isSurroundedByRocks`

   Description

   Check that it returns false for a grid that is not surrounded by rocks

   Scaffolding

   mock grid

   Expected Results

   It should return false

8. **Function**

   `_gridContains`

   Description

   Check that it returns true if the grid contains the target cell type and false otherwise (6 tests)

   Scaffolding

   mock grid

   Expected Results

   5 true, one false

9. **Function**

   `_getAdjacentCoord`

   Description

   Check that it returns the correct coordinates for all directions on both odd and even rows. (12 tests)

   Scaffolding

   expected coordinates returned

   Expected Results

   The expected coordinates should match the returned coordinates

10. **Function**

    `_getElementCoords`

    Description

    Check that it returns the correct coordinates for the target element (7 tests)

    Scaffolding

    expected coordinates returned

    Expected Results

    The expected coordinates should match the returned coordinates

11. **Function**

    `_getElementBox`

    Description

    Check that it returns the correct box for the given coordinates

    Scaffolding

    a set of coordinates and an expected box for deep comparison

    Expected Results

    The returned box should match the expected box

12. **Function**

    `_getElements`

    Description

    Check that it returns elements correctly

    Scaffolding

    mock grid

    expected element boxes

    Expected Results

    The returned element boxes should match the expected ones

13. **Function**

    `_cloneBox`

    Description

    Check that a box is cloned correctly, and is not the same object

    Scaffolding

    mock box

    Expected Results

    The returned box should be deeply equal to the mock box, but have a different memory location

14. **Function**

    `_cropBox`

    Description

    Check that a box is cropped correctly, and that blank boxes should become empty when cropped. (2 tests)

    Scaffolding

    mock box

    expected cropped version of mock box

    mock blank box

    Expected Results

    mock box should be deeply equal to the expected cropped version

    blank box should become empty when cropped

15. **Function**

    `_attemptBoxIntersection`

    Description

    Check that when an intersection is found, the correctly modified box is returned, and the topRow attribute is modified correctly.

    Scaffolding

    mock box

    expected output version of mock box

    expected topRow

    Expected Results

    mock box should be deeply equal to the expected output

    topRow should be as expected

16. **Function**

    `_containsLegalFoodBlobs`

    Description

    Check that it returns true when given a grid with legal food blobs

    Scaffolding

    mock boxes containing legal food blobs (3 for different configurations/shapes)

    Expected Results

    Should return true for all three mock boxes.

17. **Function**

    `_containsLegalFoodBlobs`

    Description

    Check that it returns false when given a grid with illegal food blobs

    Scaffolding

    mock boxes from test 16 modified to be illegal

    Expected Results

    Should return false for all three mock boxes.

18. **Function**

    `_isLegalHill`

    Description

    Check that returns true for legal hills which start on both odd and even rows

    Scaffolding

    two legal hills boxes (one for odd, one for even)

    Expected Results

    Should return true for both mock boxes.

19. **Function**

    `_isLegalHill`

    Description

    Check that returns false for illegal hills which start on both odd and even rows

    Scaffolding

    hills from Test 18 modified to be illegal

    Expected Results

    Should return false for both mock boxes.

Black Box Tests

1. **Function**

   `parseAntBrain`

   Description

   Check that works correctly for contest-legal world

   Required Data Files

   `test/model/maps/goodMap.dat`

   Expected Results

   No error is thrown

2. **Function**

   `parseAntBrain`

   Description

   Check that errors are thrown for contest-illegal worlds

   Required Data Files

   `test/model/maps/badMap-foodNum.dat`

   `test/model/maps/badMap-foodNum2.dat`

   `test/model/maps/badMap-foodShape.dat`

`test/model/maps/badMap-hillShape.dat`

`test/model/maps/badMap-rockNum.dat`

`test/model/maps/badMap-rockTouchingHill.dat`

`test/model/maps/badMap-hillTouchingHill.dat`

Expected Results

Errors are thrown for each badMap.

Phase 1.3. The World Generator

note: these tests can be found in the file `test/model/RandomWorldGenerator-test.js`

White Box Tests

1. **Function**

   `_superimpose`

   Description

   Check that it returns false when a superimposition is not possible

   Scaffolding

   mock grid

   Expected Results

   should return false

2. **Function**

   `_superimpose`

   Description

   Check that it returns true when a superimposition is possible, and that the given grid now contains the result of the superimposition.

   Scaffolding

   mock grid

   expected grid after superimposition

   Expected Results

   should return false

   mock grid should be deeply equal to expected grid.

Black Box Tests

1. **Function**

   `generateRandomWorld`

Description

Check that it generates contest-legal worlds (5 tests)

Scaffolding

World Parser required to check legality

Expected Results

each world should be parsed without an error being thrown

Phase 2. Integration Test

note: this test can be found in the file `test/model/AntGame-test.js`

Description

The customer requirements specify a sample ant brain, a small sample world, an algorithm for generating pseudo-random numbers, and a dump file containing information regarding the first 10,000 iterations of a game which pits the sample ant brain against itself on the small sample world, using the specified pseudo-random number generator for the flip instructions. For each iteration, the full state of the world is given.

This test uses the code I have written to run a simulation with the same parameters as the game used to generate the dump file. The goal is to check that the state of the world in my simulation matches the corresponding state in the dump file exactly for each iteration.

Scaffolding

Pseudo-random numbers

I used a arbitrary-precision integer library written by Matthew Crumley, John Tobey, and Vitaly Magerya called BigInteger. See http://silentmatt.com/biginteger/ for details.

This was necessary because JavaScript does not support integer overflow, and attempting to simulate integer overflow without the BigInteger library proved impossible, due to the way JavaScript handles number types (all numbers in JavaScript are double-precision floats. Even the ones that look like integers).

Eventually I got a working version of the specified algorithm. See /src/model/debug/DebugRNG.js for details.

File Reader

I had to write a small function to read in states from the dump file. It reads a kilobyte at a time onto a buffer, and whenever it detects a full world state has been read in, it returns the state. This needed to be done in parallel to the running of the algorithm,

because taking a 52mb file and splitting it on a regex, then doing comparisons etc etc is not a fast way to do things.

Required Data Files

test/model/debug/dump.all

test/model/debug/sample.ant

test/model/debug/tiny.world

Expected Results

The output of the program should match the data in the dump file

Phase 3. Acceptance Tests

Phase 3.1. Root Menu Navigation

Preconditions

none

Tests

1. **Description**

   Checking that the game loads into the root menu correctly

   Actions

   Navigate to the game page in a web browser

   Conditions for passing

   The game loads and the root menu is shown.

2. **Description**

   Checking that clicking the link to the single match setup screen functions correctly.

   Actions

   Load the game

   Click the “Single Match” button

   Conditions for passing

   The single match setup screen is shown.

3. **Description**

   Checking that clicking the link to the contest setup screen functions correctly.

   Actions

   Load the game

   Click the “Contest” button

   Conditions for passing

   The contest setup screen is shown.

Phase 3.2. Single Match Setup

Preconditions

The user has navigated to the single match setup screen

Tests

1. **Description**

   Check that clicking the link to the main menu works

   Actions

   Click the “Main Menu” breadcrumb link

   Conditions for passing

   The root menu is shown

2. **Description**

   Check that toggling graphics on/off works

   Actions

   Click the graphics toggle button.

   Click the graphics toggle button again.

   Conditions for passing

   The button changes from “with” to “without” and back to “with” again, with appropriate color changes.

3. **Description**

   Check that the number of digits in the ‘rounds’ input field cannot exceed six.

   Actions

   Type 7 or more numbers in the field.

   Conditions for passing

   The field stops accepting new digits after 6 are present

4. **Description**

   Check that any non-numeric characters typed into the ‘rounds’ field disappear when the user clicks away.

   Actions

   Type some non-numeric characters into the field

   Click away from the field

   Conditions for passing

   The non-numeric characters are removed

5. **Description**

   Check that clicking the link to pick the red brain works

   Actions

   Click the ‘pick’ button under the ‘Red Brain’ heading.

   Conditions for passing

   The Brain List screen is shown, and the currently selected red brain is highlighted with its source code shown on the right hand side of the screen.

6. **Description**

   Check that clicking the link to pick the black brain works

   Actions

   Click the ‘pick’ button under the ‘Black Brain’ heading.

   Conditions for passing

   The Brain List screen is shown, and the currently selected black brain is highlighted with its source code shown on the right hand side of the screen.

7. **Description**

   Check that clicking the link to pick the world works

   Actions

   Click the ‘pick’ button under the ‘World’ heading.

   Conditions for passing

   The World List screen is shown, and the currently selected world is highlighted with its thumbnail shown on the right hand side of the screen.

Phase 3.2.a Single Match Picking Brains

Preconditions

The user has navigated to the single match setup screen

Tests

1. **Description**

   Check that clicking the link to the main menu works while picking a red brain

   Actions

   Click the ‘Pick’ button under the ‘Red Brain’ header

   Click the ‘Main Menu’ breadcrumb link

   Conditions for passing

   The user has been taken back to the main menu

2. **Description**

   Check that clicking the link to the main menu works while picking a black brain

   Actions

   Click the ‘Pick’ button under the ‘Black Brain’ header

   Click the ‘Main Menu’ breadcrumb link

   Conditions for passing

   The user has been taken back to the main menu

3. **Description**

   Check that clicking the link to the single match setup works while picking a red brain

   Actions

   Click the ‘Pick’ button under the ‘Red Brain’ header

   Click the ‘Single Match Setup’ breadcrumb link

   Conditions for passing

   The user has been taken back to the single match setup screen, and no changes have been made to the selected components.

4. **Description**

   Check that clicking the link to the single match setup works while picking a black brain

   Actions

   Click the ‘Pick’ button under the ‘Black Brain’ header

   Click the ‘Single Match Setup’ breadcrumb link

   Conditions for passing

   The user has been taken back to the single match setup screen, and no changes have been made to the selected components.

5. **Description**

   Checking that a red brain can be picked properly

   Actions

   Make a note of the currently selected red brain name

   Click the ‘Pick’ button under the ‘Red Brain’ header

   Hover the mouse over a brain which is not the current red brain

   Click the green ‘use’ button that appears

   Conditions for passing

The user has been taken back to the single match setup screen and the name of the currently selected red brain has changed to that of the one chosen.

6. **Description**

   Checking that a black brain can be picked properly

   Actions

   Make a note of the currently selected black brain name

   Click the ‘Pick’ button under the ‘Black Brain’ header

   Hover the mouse over a brain which is not the current black brain

   Click the green ‘use’ button that appears

   Conditions for passing

   The user has been taken back to the single match setup screen and the name of the currently selected black brain has changed to that of the one chosen.

Phase 3.2.b Single Match Picking World

Preconditions

The user has navigated to the single match setup screen

Tests

1. **Description**

   Check that clicking the link to go to the main menu works

   Actions

   Click the ‘Pick’ button under the ‘World’ header

   Click the ‘Main Menu’ breadcrumb link

   Conditions for passing

   The user has been taken back to the main menu

2. **Description**

   Check that clicking the link to go back to the single match setup screen works

   Actions

   Click the ‘Pick’ button under the ‘World’ header

   Click the ‘Single Match Setup’ breadcrumb link

   Conditions for passing

   The user has been taken back to the single match setup screen

3. **Description**

Check that the world can be picked properly

Actions

Make a note of the currently selected world name

Click the ‘Pick’ button under the ‘World’ header

Hover the mouse over a world which is not the currently selected one

Click the green ‘use’ button that appears

Conditions for passing

The user has been taken back to the single match setup screen and the name of the currently selected world has changed to that of the one chosen.

Phase 3.3. Contest Setup

Preconditions

The user has navigated to the contest setup screen

Tests

1. **Description**

   Check that clicking the link to the main menu works

   Actions

   Click the “Main Menu” breadcrumb link

   Conditions for passing

   The root menu is shown

2. **Description**

   Check that clicking the link to select brains works

   Actions

   Click the ‘Select’ button above the ‘Brains’ list.

   Conditions for passing

   The Brain List is shown

3. **Description**

   Check that clicking the link to select worlds works

   Actions

   Click the ‘Select’ button above the ‘Worlds’ list.

   Conditions for passing

   The World List is shown

Phase 3.3.a Contest Setup Picking Brains

Preconditions

The user has navigated to the brains list through the contest setup screen

Tests

1. **Description**

   Check that clicking the link to the main menu works

   Actions

   Click the “Main Menu” breadcrumb link

   Conditions for passing

   The root menu is shown

2. **Description**

   Check that clicking the link to the contest setup screen works

   Actions

   Click the “Contest Setup” breadcrumb link

   Conditions for passing

   The contest setup screen is shown

3. **Description**

   Check that adding a brain causes it to disappear from the list

   Actions

   Click the ‘use’ button when hovering over a particular brain

   Conditions for passing

   The brain has disappeared from the list

4. **Description**

   Check that an added brain appears in the list of selected brains on the contest setup screen

   Actions

   Click the ‘use’ button when hovering over a particular brain

   If there are still brains in the list, click the “Contest Setup” breadcrumb link

   Conditions for passing

   The contest setup screen is shown, and the chosen brain is in the list of selected brains.

5. **Description**

   Check that if all brains in the list are selected, then the user is automatically taken back to the contest setups screen

   Actions

   Click the ‘use’ button on each brain in the list

   Conditions for passing

   The contest setup screen is shown, and all brains are present in the selected brains list.

6. **Description**

   Check that dismissed brains disappear from the selected brains list

   Actions

   Ensure that at least one brain has been chosen.

   Navigate to the contest setup screen.

   Hover over a brain and click the ‘dismiss’ button that appears

   Conditions for passing

   The Brain has disappeared from the list

7. **Description**

   Check that dismissed brains reappear in the main brains list

   Actions

   Ensure that at least one brain has been chosen.

   Navigate to the contest setup screen.

   Hover over a brain and click the ‘dismiss’ button that appears

   Click the ‘Select’ button above the ‘Brains’ list.

   Conditions for passing

   The dismissed brain is shown in the main brains list.

Phase 3.3.b Contest Setup Picking Worlds

Preconditions

The user has navigated to the worlds list through the contest setup screen

Tests

1. **Description**

   Check that clicking the link to the main menu works

   Actions

   Click the “Main Menu” breadcrumb link

Conditions for passing

The root menu is shown

2. **Description**

   Check that clicking the link to the contest setup screen works

   Actions

   Click the “Contest Setup” breadcrumb link

   Conditions for passing

   The contest setup screen is shown

3. **Description**

   Check that adding a world causes it to disappear from the list

   Actions

   Click the ‘use’ button when hovering over a particular world

   Conditions for passing

   The world has disappeared from the list

4. **Description**

   Check that an added world appears in the list of selected brains on the contest setup screen

   Actions

   Click the ‘use’ button when hovering over a particular world

   If there are still worlds in the list, click the “Contest Setup” breadcrumb link

   Conditions for passing

   The contest setup screen is shown, and the chosen world is in the list of selected worlds.

5. **Description**

   Check that if all worlds in the list are selected, then the user is automatically taken back to the contest setup screen

   Actions

   Click the ‘use’ button on each world in the list

   Conditions for passing

   The contest setup screen is shown, and all worlds are present in the selected worlds list.

6. **Description**

   Check that dismissed worlds disappear from the selected worlds list

Actions

Ensure that at least one world has been chosen.

Navigate to the contest setup screen.

Hover over a world and click the ‘dismiss’ button that appears

Conditions for passing

The world has disappeared from the list

7. **Description**

   Check that dismissed worlds reappear in the main worlds list

   Actions

   Ensure that at least one world has been chosen.

   Navigate to the contest setup screen.

   Hover over a world and click the ‘dismiss’ button that appears

   Click the ‘Select’ button above the ‘Worlds’ list.

   Conditions for passing

   The dismissed world is shown in the main brains list.

8. **Description**

   Check that contest-illegal worlds are not shown

   Conditions for passing

   There is no world in the list called “Tiny World”.

Phase 3.4 Adding/Modifying Brains

Preconditions

The user has navigated to the brains list through either the single match setup screen or the contest setup screen.

Tests

1. **Description**

   Check that clicking on a particular brain highlights the brain and shows its source code in the box on the right hand side of the screen

   Actions

   Click a brain other than the one currently highlighted

   Conditions for passing

The clicked brain is now highlighted and its source code is shown in the box on the right hand side of the screen.

2. **Description**

   Check that clicking the button to add a brain opens the editor dialog.

   Actions

   Click the button marked “add+”;

   Conditions for passing

   The edit dialog is shown with the header “Add New Brain” and blank fields.

3. **Description**

   Check that clicking the cancel button in the edit dialog causes it to disappear

   Actions

   Click the button marked “add+”

   Click the button marked ‘cancel’

   Conditions for passing

   The edit dialog has disappeared

4. **Description**

   Check that clicking the close button in the edit dialog causes it to disappear

   Actions

   Click the button marked “add+”

   Click the button marked ‘×’ at the top-right of the screen

   Conditions for passing

   The edit dialog has disappeared

5. **Description**

   Check that clicking the darkened background while the edit dialog is open causes it to disappear.

   Actions

   Click the button marked “add+”

   Click somewhere on the darkened background

   Conditions for passing

   The edit dialog has disappeared

6. **Description**

   Check that attempting to compile a malformed brain opens an alert with a description of the error caught.

Actions

Click the button marked “add+”

Copy the given resource text into the source field.

Click the button marked “compile”

Conditions for passing

An alert showing the message “Malformed Instruction: ‘turn ahead 2’ at line 2” is displayed.

Resources

```
flip 1 0 1 ; this brain is not legal
turn ahead 2
```

7. **Description**

   Check that attempting to compile a well formed brain without giving a name creates and highlights a new brain called “Untitled Brain”.

   Actions

   Click the button marked “add+”

   Copy the given resource text into the source field.

   Click the button marked “compile”

   Conditions for passing

   The edit dialog disappears and a new brain appears at the top of the list called “Untitled Brain”. It is highlighted and its source code is visible in the box on the right hand side of the screen.

   Resources

```
flip 2 1 0 ; this is an untitled brain
turn left 0
```

8. **Description**

   Check that attempting to compile a well formed brain with a custom name works properly.

   Actions

   Click the button marked “add+”

   Type a name into the name field

   Copy the given resource text into the source field.

   Click the button marked “compile”

   Conditions for passing

   The edit dialog disappears and a new brain appears at the top of the list with the given name. It is highlighted and its source code is visible in the box on the right hand side of the screen.

   Resources

```
flip 2 1 0 ; this brain is named
turn left 0
```

9. **Description**

   Check that attempting to edit a brain by making it illegal doesn’t change it.

   Actions

   Ensure that there is a custom brain in the list

   Hover over the custom brain and click the edit button (yellow with pencil icon)

   Change the name of the brain

   Change the source of the brain such that it becomes illegal

Click the button marked “compile”

Click OK on the error alert box

Close the edit dialog

Conditions for passing

Neither the name or the source of the brain has changed.

10. **Description**

Check that attempting to edit a brain legally works properly.

Actions

Ensure that there is a custom brain in the list

Hover over the custom brain and click the edit button (yellow with pencil icon)

Change the name of the brain

Change the source of the brain such that it remains legal

Click the button marked “compile”

Conditions for passing

The edit dialog disappears and both the name of the brain and the source code have changed in accordance with the modifications made.

Phase 3.5 Adding/Modifying Worlds during single match setup

Preconditions

The user has navigated to the worlds list through the single match setup screen.

Tests

1. **Description**

Check that clicking on a particular world highlights the world and shows its thumbnail on the right hand side of the screen

Actions

Click a world other than the one currently highlighted

Conditions for passing

The clicked world is now highlighted and its thumbnail is shown in the box on the right hand side of the screen.

2. **Description**

Check that clicking the button to add a world opens the editor dialog.

Actions

Click the button marked “add+”;

Conditions for passing

The edit dialog is shown with the header “Add New World” and blank fields.

3. **Description**

   Check that clicking the cancel button in the edit dialog causes it to disappear

   Actions

   Click the button marked “add+”

   Click the button marked ‘cancel’

   Conditions for passing

   The edit dialog has disappeared

4. **Description**

   Check that clicking the close button in the edit dialog causes it to disappear

   Actions

   Click the button marked “add+”

   Click the button marked ‘×’ at the top-right of the screen

   Conditions for passing

   The edit dialog has disappeared

5. **Description**

   Check that clicking the darkened background while the edit dialog is open causes it to disappear.

   Actions

   Click the button marked “add+”

   Click somewhere on the darkened background

   Conditions for passing

   The edit dialog has disappeared

6. **Description**

   Check that attempting to compile a malformed world opens an alert with a description of the error caught.

   Actions

   Click the button marked “add+”

   Copy the given resource text into the source field.

   Click the button marked “compile”

   Conditions for passing

   An alert showing the message “The ant world must contain at least one source of food” is displayed.

   Resources

```
5
5
# # # # #
 # . # - #
# . # + #
 # . # . #
# # # # #
```

7. **Description**

   Check that attempting to compile a well formed world without giving a name creates and highlights a new world called “Untitled World”.

   Actions

   Click the button marked “add+”

   Copy the given resource text into the source field.

   Click the button marked “compile”

   Conditions for passing

   The edit dialog disappears and a new world appears at the top of the list called “Untitled World”. It is highlighted and its thumbnail is visible in the box on the right hand side of the screen.

   Resources

```
5
5
# # # # #
 # . # - #
# . # + #
 # . . 5 #
# # # # #
```

8. **Description**

   Check that attempting to compile a well formed world with a custom name works properly.

   Actions

   Click the button marked “add+”

   Type a name into the name field

   Copy the given resource text into the source field.

   Click the button marked “compile”

   Conditions for passing

   The edit dialog disappears and a new world appears at the top of the list with the given name. It is highlighted and its thumbnail is visible in the box on the right hand side of the screen.

   Resources

```
5
5
# # # # #
 # . # - #
# . # + #
 # . . 5 #
# # # # #
```

9. **Description**

   Check that attempting to edit a world by making it illegal doesn’t change it.

   Actions

   Ensure that there is a custom world in the list

   Hover over the custom world and click the edit button (yellow with pencil icon)

   Change the name of the world

   Change the source of the world such that it becomes illegal

   Click the button marked “compile”

   Click OK on the error alert box

   Close the edit dialog

   Conditions for passing

   Neither the name nor the source of the world has changed.

10. **Description**

Check that attempting to edit a world legally works properly.

Actions

Ensure that there is a custom world in the list

Hover over the custom world and click the edit button (yellow with pencil icon)

Change the name of the world

Change the source of the world such that it remains legal

Click the button marked “compile”

Conditions for passing

The edit dialog disappears and both the name of the world and the source code have changed in accordance with the modifications made.

Phase 3.6 Adding/Modifying Worlds during contest setup

Preconditions

The user has navigated to the worlds list through the contest setup screen.

Tests

1. **Description**

   Check that attempting to compile a well-formed but contest-illegal world opens an alert with a description of the error caught.

   Actions

   Click the button marked “add+”

   Copy the given resource text into the source field.

   Click the button marked “compile”

   Conditions for passing

   An alert showing the message “Too few lines” is displayed.

   Resources

```
5
5
# # # # #
 # . # - #
# . # + #
 # . # 5 #
# # # # #
```

Phase 3.7 Generating Random Worlds

Preconditions

The user has navigated to the worlds list through either the single match setup screen or the contest setup screen.

Tests

1. **Description**

   Check that generating random worlds works

   Actions

   Click the button marked “generate+”

   Conditions for passing

   A new world called “Random World ^{n}”, where $n \in N$, is shown in the worlds list. It is also highlighted and its thumbnail is shown on the right hand side of the screen.

Phase 3.8 Running a Single Match with Graphics

Preconditions

The user has navigated to the single match setup screen

Tests

1. **Description**

   Check that the match starts.

   Actions

   Click the button marked “Go”

   Conditions for passing

   The selected world is shown.

   The team names are shown in the top bar

   The breadcrumb navigation disappears.

   The speed is shown, along with buttons marked ‘+’ and ‘-’

   A button marked ‘cancel’ is shown.

   The ants are moving/doing whatever their brain dictates.

2. **Description**

   Check that the match finishes properly

   Actions

   Click the button marked “Go”

   Wait for a while

   Conditions for passing

   A results dialog is shown, indicating the number of food gathered by each

team, along with the number of deaths experienced by each team.

3. **Description**

   Check that the results dialog closes properly

   Actions

   Click the button marked “Go”

   Wait for the results dialog to appear

   Click the button marked ‘x’ at the top-right of the dialog.

   Conditions for passing

   The results dialog closes and the user is shown the single match setup screen.

4. **Description**

   Check that increasing the speed has the desired effect

   Actions

   Click the button marked “Go”

   Ensure that the number indicating speed is less than 10.

   Click the button marked “+”

   Conditions for passing

   The number indicating the speed of the game increases by 1.

   The ants do their thing slightly faster than before.

5. **Description**

   Check that decreasing the speed has the desired effect

   Actions

   Click the button marked “Go”

   Ensure that the number indicating speed is greater than 1.

   Click the button marked “-”

   Conditions for passing

   The number indicating the speed of the game decreases by 1.

   The ants do their thing slightly slower than before.

6. **Description**

   Check that cancelling the game has the desired effect

   Actions

   Click the button marked “Go”

   Click the button marked “Cancel”

   Conditions for passing

The user is returned to the single match setup screen.

Phase 3.9 Running a Single Match without Graphics

Preconditions

The user has navigated to the single match setup screen

Tests

1. **Description**

   Check that the match starts.

   Actions

   Click the button marked “Go”

   Conditions for passing

   The team names and the name of the world are displayed

   The navigation disappears.

   A button marked ‘cancel’ is shown.

   A progress bar fills up

2. **Description**

   Check that the match finishes properly

   Actions

   Click the button marked “Go”

   Wait for a while

   Conditions for passing

   A results dialog is shown, indicating the number of food gathered by each team, along with the number of deaths experienced by each team.

3. **Description**

   Check that the results dialog closes properly

   Actions

   Click the button marked “Go”

   Wait for the results dialog to appear

   Click the button marked ‘x’ at the top-right of the dialog.

   Conditions for passing

   The results dialog closes and the user is shown the single match setup screen.

   The navigation reappears.

4. **Description**

Check that cancelling the game has the desired effect

Actions

Click the button marked “Go”

Click the button marked “Cancel”

Conditions for passing

The user is returned to the single match setup screen.

The navigation reappears.

Phase 3.10 Starting a Contest

Preconditions

The user has navigated to the contest setup screen.

Tests

1. **Description**

   Check that an error alert is displayed when the user attempts to start a contest with no worlds

   Actions

   Choose some brains for the contest

   Click the button marked “Go”.

   Conditions for passing

   An alert box appears with the message, “At least one world must be chosen”

2. **Description**

   Check that an error alert is displayed when the user attempts to start a contest with fewer than two brains

   Actions

   Choose one or more worlds.

   Choose one or no brains.

   Click the button marked “Go”.

   Conditions for passing

   An alert box appears with the message, “At least two brains must be chosen”

3. **Description**

   Check that when sufficient brains and worlds are chosen, the contest begins.

   Actions

   Choose one or more worlds.

   Choose two or more brains.

Click the button marked “Go”.

Conditions for passing

The contest fixtures/rankings screen is shown.

The fixtures are such that each brain plays against each other brain on every world twice; once for each color.

Clicking the ‘played’ link shows an empty table.

Phase 3.11 Running a Contest

Preconditions

The user has navigated to the contest results/fixtures screen.

Tests

1. **Description**

   Check that playing all matches works properly

   Actions

   Click the button marked “Play All”

   Conditions for passing

   Games are played sequentially. When one finishes, the next one starts.

   When all games are finished, the user is returned to the fixtures/rankings screen.

2. **Description**

   Check that cancelling a match during a Play All session preserves the results of any fixtures which have been played.

   Actions

   Click the button marked “Play All”

   Wait until the second match starts.

   Click the button marked “cancel”.

   Conditions for passing

   The user is taken back to the fixtures/rankings screen.

   The fixture that was played is shown in the Played Fixtures list, and not in the remaining fixtures list.

   Two of the ants have played one fixture.

3. **Description**

   Check that playing individual fixtures works

Actions

Hover over a fixture

Click the button marked ‘play’ that appears

Conditions for passing

The game is played and finishes.

The fixture that was played is shown in the Played Fixtures list, and not in the remaining fixtures list.

Two of the ants have played one fixture.

Test Results

The test results are an important and essential part of this document as they will record the results of running each of the test phases within this deliverable and their resultant values for the source code. For this section of the document, a record of the results for running each of the test phases shall be described in accordance to the respective phases the results will be derived from.

These are the following data that shall also be recorded:

- Results: A description/statement of the actual result of a test.

This will be a simple description of the resultant outcome of a test, it can be a basic evaluation of the resultant output or input of a test but will serve a good deal for providing informative text.

- Status: A statement based on the success or failure of a test.

The status is important in respect to the state of a test as it will give informative value on the current situation a test is in. It may be in a critical status e.g. a failure where it could then be passed onto as required for an Action to see if a corrective solution is available to provide a successful test outcome.

- Action: If a test has failed, the testing phase could result for a corrective action to be taken to try and provide a successful solution for the test.

The action part of the test results is an essential requirement as it will define the overall outcome of a test. However, the Quality Assurance Team and the Programming Team will have the final judgement on the actions for a test therefore preliminary evaluations on the actions can be changed as the test develops. There must always be an action be it not available or available the action will determine the overall justification for the final test result(s).

Ideally, a testing framework (software) should be used to conduct some or most of the tests in respect to the source code while some tests such as UI (User Interface) would be tested on Internet browsers (major i.e.: Internet Explorer, Safari, Mozilla FireFox, Opera, Chrome). This will therefore allow for an in depth outlook on the functionalities and the visual capabilities of the game in its target working environment.

Nodeunit is the testing framework that has been chosen to aid most of the testing for the source code of the Ant Game. It utilises simple syntax and powerful tools to allow for the easy async unit testing that is used by JavaScript (Github, 2012.) Installation of Nodeunit is required for its utilities to be used accordingly for the source code at hand (the Ant Game). A test runner is used to analyse the criteria for testing and provides a legitimate presentation of the tests carried out.

Unit Test Results

Brain Parser

#	Function	Description	Status	Action
1.1 w.1	`_parseInt`	Checking that preceding zeroes are stripped correctly	PASS	N/A
1.1 w.2	`parseAntBrain`	Checking that the parser works with windows newlines	PASS	N/A
1.1 w.3	`parseAntBrain`	Checking that the parser works with mac newlines	PASS	N/A
1.1 w.4	`parseAntBrain`	Checking that the parser works with unix newlines	PASS	N/A
1.1 w.5	`parseAntBrain`	Test that an error is thrown when the brain has no states	PASS	N/A
1.1 w.6	`parseAntBrain`	Test that an error is thrown when the brain has a syntax error	PASS	N/A
1.1 w.7	`parseAntBrain`	Test that an error is thrown when a nonexistent state is pointed to	PASS	N/A
1.1 w.8	`parseAntBrain`	Test that an error is thrown when a marker number is not in the range 0-5	PASS	N/A

World Parser

#	Function	Description	Status	Action
1.2 w.1	`_parseGridLine`	Check that it works for valid even lines	PASS	N/A
1.2 w.2	`_parseGridLine`	Check that it works for valid odd lines	PASS	N/A
1.2 w.3	`_parseGridLine`	Check that it throws an error if odd and even lines are swapped	PASS	N/A
1.2 w.4	`_parseGridLine`	Check that it throws an error for invalid characters	PASS	N/A
1.2 w.5	`_parseGridLine`	Check that it throws an error when seeing an odd line with too many spaces at the beginning	PASS	N/A
1.2 w.6	`_isSurroundedByRocks`	Check that it returns true for a grid that is surrounded by rocks	PASS	N/A

1.2 w.7	`_isSurrounded ByRocks`	Check that it returns false for a grid that is not surrounded by rocks	PASS	N/A
1.2 w.8	`_gridContains`	Check that it returns true if the grid contains the target cell type and false otherwise (6 tests)	PASS	N/A
1.2 w.9	`_getAdjacentC oord`	Check that it returns the correct coordinates for all directions on both odd and even rows. (12 tests)	PASS	N/A
1.2 w.10	`_getElementCo ords`	Check that it returns the correct coordinates for the target element (7 tests)	PASS	N/A
1.2 w.11	`_getElementBo x`	Check that it returns the correct box for the given coordinates	PASS	N/A
1.2 w.12	`_getElements`	Check that it returns elements correctly	PASS	N/A
1.2 w.13	`_cloneBox`	Check that a box is cloned correctly, and is not the same object	PASS	N/A
1.2 w.14	`_cropBox`	Check that a box is cropped correctly, and that blank boxes should become empty when cropped. (2 tests)	PASS	N/A
1.2 w.15	`_attemptBoxIn tersection`	Check that when an intersection is found, the correctly modified box is returned, and the topRow attribute is modified correctly.	PASS	N/A
1.2 w.16	`_containsLega lFoodBlobs`	Check that it returns true when given a grid with legal food blobs	PASS	N/A
1.2 w.17	`_containsLega lFoodBlobs`	Check that it returns false when given a grid with illegal food blobs	PASS	N/A
1.2 w.18	`_isLegalHill`	Check that returns true for legal hills which start on both odd and even rows	PASS	N/A
1.2 w.19	`_isLegalHill`	Check that returns false for illegal hills which start on both odd and even rows	PASS	N/A
1.2 b.1	`parseAntBrain`	Check that works correctly for contest-legal world	PASS	N/A
1.2 b.2	`parseAntBrain`	Check that errors are thrown for contest-illegal worlds	PASS	N/A

World Generator

#	Function	Description	Status	Action
1.3 w.1	`_superimpose`	Check that it returns false when a superimposition is not possible	PASS	N/A

1.3 w.2	`_superimpose`	Check that it returns true when a superimposition is possible, and that the given grid now contains the result of the superimposition.	PASS	N/A
1.3 b.1	`generateRandomWorld`	Check that it generates contest-legal worlds (5 tests)	PASS	N/A

Integration Test

#	Description	Status	Action
2	Check output of program against dump file	PASS	N/A

Acceptance Tests Results Table

#	Description	Chrome	Firefox	IE9	Opera	Safari	Action
3.1	Checking that the game loads into the root menu correctly	PASS	PASS	PASS	PASS	PASS	N/A
3.1.2	Checking that clicking the link to the Single Match setup screen function correctly	PASS	PASS	PASS	PASS	PASS	N/A
3.1.3	Checking that clicking the link to the Contest setup screen functions correctly	PASS	PASS	PASS	PASS	PASS	N/A
3.2.1	Check that clicking the link to the Main Menu works	PASS	PASS	PASS	PASS	PASS	N/A
3.2.2	Check that toggling graphics on/off works	PASS	PASS	PASS	PASS	PASS	N/A
3.2.3	Check that the number of digits in the rounds input field cannot exceed 6 digits	PASS	PASS	PASS	PASS	PASS	N/A
3.2.4	Check that any non-numeric characters typed into the ‘rounds’ field disappear when the uses clicks away.	PASS	PASS	PASS	PASS	PASS	N/A
3.2.5	Check that clicking the link to the pick the Red Brain works	PASS	PASS	PASS	PASS	PASS	N/A

3.2.6	Check that clicking the link to the pick the Black Brain works	PASS	PASS	PASS	PASS	PASS	N/A
3.2.7	Check that clicking the link to the World works	PASS	PASS	PASS	PASS	PASS	N/A
3.2.a.1	Check that clicking the link to the Main Menu works while picking the Red Brain	PASS	PASS	PASS	PASS	PASS	N/A
3.2.a.2	Check that clicking the link to the Main Menu works while picking the Black Brain	PASS	PASS	PASS	PASS	PASS	N/A
3.2.a.3	Check that clicking the link to the Single Match setup works while picking Red Brain	PASS	PASS	PASS	PASS	PASS	N/A
3.2.a.4	Check that clicking the link to the Single Match set up works while picking Black Brain	PASS	PASS	PASS	PASS	PASS	N/A
3.2.a.5	Checking that a Red Brain can be picked properly	PASS	PASS	PASS	PASS	PASS	N/A
3.2.a.6	Checking that a Black Brain can be picked properly	PASS	PASS	PASS	PASS	PASS	N/A
3.2.b.1	Check that clicking the link to go to the Main Menu works	PASS	PASS	PASS	PASS	PASS	N/A
3.2.b.2	Check that clicking the link to go back to the Single Match set up screen works	PASS	PASS	PASS	PASS	PASS	N/A
3.2.b.3	Check that the World can be picked properly	PASS	PASS	PASS	PASS	PASS	N/A
3.3.1	Check that clicking the link to the Main Menu works	PASS	PASS	PASS	PASS	PASS	N/A
3.3.2	Check that clicking the link to the Select Brains works	PASS	PASS	PASS	PASS	PASS	N/A

3.3.3	Check that clicking the link to select the World works	PASS	PASS	PASS	PASS	PASS	N/A
3.3.a.1	Check that clicking the link to the Main Menu works	PASS	PASS	PASS	PASS	PASS	N/A
3.3.a.2	Check that clicking the link to the Contest set up screen works	PASS	PASS	PASS	PASS	PASS	N/A
3.3.a.3	Check that adding a Brain causes it to disappear from the list	PASS	PASS	PASS	PASS	PASS	N/A
3.3.a.4	Check that an added Brain appears in the list of Selected Brains in the Contest set up screen	PASS	PASS	PASS	PASS	PASS	N/A
3.3.a.5	Check that all Brains in the list are selected, then the user is automatically taken back to the Contest set up screen	PASS	PASS	PASS	PASS	PASS	N/A
3.3.a.6	Check that dismissed Brains disappear from the selected Brains list	PASS	PASS	PASS	PASS	PASS	N/A
3.3.a.7	Check that dismissed brains reappear in the main brains list	PASS	PASS	PASS	PASS	PASS	N/A
3.3.b.1	Check that clicking the link to the Main Menu works	PASS	PASS	PASS	PASS	PASS	N/A
3.3.b.2	Check that clicking the link to the Contest setup screen works.	PASS	PASS	PASS	PASS	PASS	N/A
3.3.b.3	Check that adding a World causes it to disappear from the list	PASS	PASS	PASS	PASS	PASS	N/A
3.3.b.4	Check that an added World appears in the list of selected Brains on the Contest setup screen	PASS	PASS	PASS	PASS	PASS	N/A

3.3.b.5	Check that if all worlds in the list are selected, then the user is automatically taken back to the Contest setup screen	PASS	PASS	PASS	PASS	PASS	N/A
3.3.b.6	Check that dismissed Worlds disappear from the selected Worlds list	PASS	PASS	PASS	PASS	PASS	N/A
3.3.b.7	Check that dismissed Worlds reappear in the main Worlds list	PASS	PASS	PASS	PASS	PASS	N/A
3.3.b.8	Check that contest-illegal Worlds are not shown	PASS	PASS	PASS	PASS	PASS	N/A
3.4.1	Check that clicking on a particular Brain highlights the Brain and shows its source code in the box on the right hand side of the screen	PASS	PASS	PASS	PASS	PASS	N/A
3.4.2	Check that clicking the button to add a Brain opens the editor dialog	PASS	PASS	PASS	PASS	PASS	N/A
3.4.3	Check that clicking the cancel button in the edit dialog causes it to disappear	PASS	PASS	PASS	PASS	PASS	N/A
3.4.4	Check that clicking the close button in the edit dialog causes it to disappear	PASS	PASS	PASS	PASS	PASS	N/A
3.4.5	Check that clicking the darkened background while the edit dialog is open causes it to disappear	PASS	PASS	PASS	PASS	PASS	N/A
3.4.6	Check that attempting to compile a malformed Brain opens an alert with a description of the error caught	PASS	PASS	PASS	PASS	PASS	N/A

3.4.7	Check that attempting to compile a well formed Brain without giving a name creates and highlights a new Brain called “Untitled Brain”.	PASS	PASS	PASS	PASS	PASS	N/A
3.4.8	Check that attempting to compile a well formed Brain with a custom name works properly	PASS	PASS	PASS	PASS	PASS	N/A
3.4.9	Check that attempting to edit a Brain by making it illegal doesn’t change it.	PASS	PASS	PASS	PASS	PASS	N/A
3.4.10	Check that attempting to edit a Brain legally works properly.	PASS	PASS	PASS	PASS	PASS	N/A
3.5.1	Check that clicking on a particular worlds list through the single match setup screen.	PASS	PASS	PASS	PASS	PASS	N/A
3.5.2	Check that clicking the button to add a world opens the editor dialog.	PASS	PASS	PASS	PASS	PASS	N/A
3.5.3	Check that clicking the cancel button in the edit dialog causes it to disappear.	PASS	PASS	PASS	PASS	PASS	N/A
3.5.4	Check that clicking the close button in the edit dialog causes it to disappear.	PASS	PASS	PASS	PASS	PASS	N/A
3.5.5	Check that clicking the darkened background while the edit dialog is open causes it to disappear.	PASS	PASS	PASS	PASS	PASS	N/A
3.5.6	Check that attempting to compile a malformed world opens an alert with a description of the error caught.	PASS	PASS	PASS	PASS	PASS	N/A

3.5.7	Check that attempting to compile a well formed world without giving a name creates and highlights a new world called "Untiled World".	PASS	PASS	PASS	PASS	PASS	N/A
3.5.8	Check that attempting to compile a well formed world with a custom name works properly.	PASS	PASS	PASS	PASS	PASS	N/A
3.5.9	Check that attempting to edit a world by making it illegal doesn't change it.	PASS	PASS	PASS	PASS	PASS	N/A
3.5.10	Check that attempting to edit a world legally works properly.	PASS	PASS	PASS	PASS	PASS	N/A
3.6.1	Check that attempting to compile a well-formed but contest- illegal world opens an alert with a description of the error caught.	PASS	PASS	PASS	PASS	PASS	N/A
3.7.1	Check that generating random world works.	PASS	PASS	PASS	PASS	PASS	N/A
3.8.1	Check that the match starts.	PASS	PASS	PASS	PASS	PASS	N/A
3.8.2	Check that the match finishes properly.	PASS	PASS	PASS	PASS	PASS	N/A
3.8.3	Check that the results dialog closes properly.	PASS	PASS	PASS	PASS	PASS	N/A
3.8.4	Check that increasing the speed has the desired effect.	PASS	PASS	PASS	PASS	PASS	N/A
3.8.5	Check that decreasing the speed has desired effect.	PASS	PASS	PASS	PASS	PASS	N/A
3.8.6	Check that cancelling the game has the desired effect.	PASS	PASS	PASS	PASS	PASS	N/A
3.9.1	Check that the match starts.	PASS	PASS	PASS	PASS	PASS	N/A
3.9.2	Check that the match finishes properly.	PASS	PASS	PASS	PASS	PASS	N/A

3.9.3	Check that the results dialog closes properly.	PASS	PASS	PASS	PASS	PASS	N/A
3**.9.4**	Check that cancelling the game has the desired effect.	PASS	PASS	PASS	PASS	PASS	N/A
3.10.1	Check that an error alert is displayed when the user attempts to start a contest with no words.	PASS	PASS	PASS	PASS	PASS	N/A
3.10.2	Check that an error alert is displayed when the user attempts to start a contest with fewer than two brains.	PASS	PASS	PASS	PASS	PASS	N/A
3.10.3	Check that when sufficient brains and worlds are chosen, the contest begins.	PASS	PASS	PASS	PASS	PASS	N/A
3.11.1	Check that playing all matches works properly.	PASS	PASS	PASS	PASS	PASS	N/A
3.11.2	Check that cancelling a match during a Play All session preserves the results of any fixtures which have been played.	PASS	PASS	PASS	PASS	PASS	N/A
3.11.3	Check that playing individual fixtures works.	PASS	PASS	PASS	PASS	PASS	N/A

Overview

Finally, having fully analysed the criteria that was specified by the deliverable specifications from the first and second deliverables, a justification on the success of the source code and overall game can be concluded. By providing a Testing Specification the source code has a solid background that provides as proof that it has working and rectified elements along with the detection of prominent bugs/errors within the Ant Game.

This document was able to record and provide an outlook to all three types of testing that were critical as well as the compulsory types of testing that were described in the previous deliverables i.e.: White Box Testing and Black Box Testing. There are multiple variations of presenting the testing data found within this document thus a clear overview of the Testing for the developed Ant Game has been thoroughly invoked, designed and presented in a reasonably organised fashion. Moreover, providing a descending order of phases that are important to the testing phase has been recorded within this document as the team believed that these specific factors of the source code require attention as these features and elements serve the most critical parts to the programming of the Ant Game thus all of the phases which the team thought relevant and essential to test are found here within this testing document.

With the integration and documentation of all relevant and available tests of the source code, it is with confidence that Group 6: Team Good can provide an overall satisfactory result of success to the creation and overall finishing of the Software Engineering Course, as a final game - the Ant Game - has been produced, developed and finalised.

The series of tests provided the necessary evidence that the source code is up to par with all testing specifications that were needed to be met. A variety of variables i.e. Overhead Software, Test Descriptions etc. has been used to effectively portray the different test phases the source code had to go through along with Test Results that efficiently highlight the key facts about the test have also been formalised within this deliverable.

References & Bibliography

Introduction to Software Engineering, Software Engineering Course website. [online] Available at: <https://studydirect.sussex.ac.uk/course/view.php?id=14546&rel=home> [Accessed 6/6/2012].

Nodeunit, 2012. *Github.* [online] Available at: <https://github.com/caolan/nodeunit> [Accessed 6 June 2012].

Sign Off

- Analysis Team

- Design Team

- Programming Team

- Quality Assurance Team