



ANT GAME PROJECT

Software Engineering – Group 2

Team Food Thief

Version 1.0

118417, 118440, 118488, 119386, 122513

CONTENTS

Contents

REQUIREMENTS	3
1.1 Preface.....	4
1.2 Introduction.....	4
1.3 Best Practice	4
1.4 Timescale.....	5
1.5 Group Structure.....	5
1.6 Gantt Chart.....	5
2.1 Project Plan	6
2.2 Stakeholders.....	6
3.1 Project Constraints.....	6
4.1 System Architecture	7
5.1 Functional Requirements	8
5.2 Non-Functional Requirements	8
5.3 Acceptance Criteria	9
6.1 Project Issues.....	10
6.2 Risks.....	10
6.3 User Documentation & Training	10
6.4 Costs	11
DESIGN	12
7.1 System Design	13
8.1 Model-View-Controller	14
8.2 Class Diagram	15
8.3 Sequence Diagram	16
8.4 Use Case	17
8.5 Gameplay Diagram.....	18
8.6 Aggregation Association.....	20

IMPLEMENTATION	21
9.1 Implementation Strategy	22
9.2 Implementation Timescale.....	22
TESTING.....	23
10.1 Testing	24
10.2 Testing Scope	24
10.3 Travis CI	25
10.4 Unit Testing	26
10.5 System Testing	29
10.6 Acceptance Testing	30
REFLECTION.....	31
11.1 Changes from the Original Design.....	32
11.2 Future Improvements	33
11.3 Maintenance	34
11.4 Review of Waterfall.....	35
11.5 Project Management Reflection	35
11.6 Implementation Reflection	35
11.6.1 GitHub.....	35
11.6.2 General Implementation	36
11.7 Personal Reflection	37
11.7.1 Samuel Kirsten.....	37
11.7.2 Jacob Hughes	37
11.7.3 Junho Lee	38
11.7.4 Hao-lin Liang	38
11.7.5 Tina Wang.....	39
12.1 Appendices.....	41
12.1.1 Minutes.....	41
12.2 Gantt Chart.....	43
12.3 Critical Path.....	44



REQUIREMENTS

1.1 Preface

This document will read by the end users and developers of the software system.

Document Version: 1.0

1.2 Introduction

This document describes the requirements for a software system that implements and simulates ant brains in both randomly generated and user specified worlds, in a competition-based environment.

The multiplayer game simulates ant colonies from player specified brains. During a single game, two colonies will fight for survival in a world containing rocks, food, ants and colonies. The winning colony is the one with the most food remaining after all the rounds have ended. If enemy ants surround an ant, it dies and becomes food for the enemy. Ants can leave scents in parts of the world to communicate with their colony.

The game will be built in Java and use the Swing library to develop the GUI.

1.3 Best Practice

All meetings should be minuted with attendance noted from all contributors.

All communication when not in person is done through a Facebook group, and Google Docs is used for non-code collaboration.

Each member of the team who works on the code should use the same IDE for consistency. The chosen environment is JetBrains IntelliJ IDE. Git provides version control with code hosted on GitHub. In addition, a test driven approach to development will be used. The JUnit library will be responsible for this and continuous integration will be monitored by the use of Travis CI.

1.4 Timescale

- Initial Deadline for Requirements: 13/02/15
- Final Requirements Deadline: 22/02/15
- Initial Deadline for Design: 01/03/15
- Final Design Deadline: 08/03/15
- Implementation Deadline: 22/03/15
- Testing Deadline: 30/03/15
- Soft Deadline: 09/04/15
- Soft Project Deadline: 10/04/15
- Actual Deadline: 12/04/15

1.5 Group Structure

The development group will work without any formal structure with work distributed to people based on current workload and ability.

1.6 Gantt Chart

See [10.2 Appendix](#)

2.1 Project Plan

The management and planning for this project will adopt the waterfall process. The project will consist of 5 phases of development: Requirement specification, Design, Implementation, Testing and Review. The nature of the waterfall development process requires each section to be completed before commencing the next stage. The initial soft deadline for the group is 9th April 2015. All work should be completed by this date to allow for any changes, delays or corrections to be resolved before the final deadline.

2.2 Stakeholders

- University of Sussex Informatics Department
- Developers of the system – Software Engineering Group 2
- Any other people testing the system

3.1 Project Constraints

The project has a number of constraints:

- **Time:** The project in its entirety must be completed by Thursday 16th April 2015. A soft deadline for the group is set as the 9th of April 2015 to allow for any changes to be accounted for. Progress reports keep track of the progress at each meeting.
- **Collaborators:** There are 5 people allocated to the project, with no option to expand the team. In addition, people have different modules it is not practical to find working hours that suit everybody. Meetings between the team members are therefore limited to one seminar, plus one or two hours extra a week, with an exception to made for those working with pair programming. All communication is done either in person or through a Facebook group.
- **Software Limitations:** There are no significant software limitations for the project. The principle software this project will be developed in JetBrains IntelliJ Idea. This is free to download for educational use. GitHub & Git will be used for version control

and code hosting. Git is free, and GitHub offers private repositories free for education use. Lastly, Travis CI, which is to be used for continuous integration management of the project is available under the GitHub educational package. This connects to the private repository and so should not present any immediate limitations.

- **Cost Limitations:** The project has no financial cost. Staffing costs will not be applicable to this project due to team members participating for educational purposes. As previously mentioned, the software to be used is provided free for educational purposes resulting in zero development cost.

4.1 System Architecture

The ant game will consist of a set of different components, which work together to produce the overall game.

The system will be fundamentally built upon the Model View Controller pattern. The reasons for this are highly beneficial: It allows for separation between the UI and game logic, limiting the scope of possible bugs to their respective module by reducing the coupling of objects. This approach also allows for improvements upon the UI and game model to be carried out independently with the assertion that changes on one module will not negatively impact the other.

The main component will be either a text based or graphical user interface. This will consist of an area to load brains and maps as well as an area to launch the games in a tournament. This will tie in with the teams and the environment and allow a game to commence.

The world component will be launched by the previous module, and allow for a world for gameplay. The environment will provide the many issues that the ants will encounter during gameplay. This world will then be represented graphically by the main UI component, making sure it adheres to the MVC pattern.

Lastly the ant will be the final component that is initialised by the team component.

For the MVC diagram see [8.1 Model-View-Controller](#)

5.1 Functional Requirements

1. A player will be able to input a brain that they have created in an external file.
2. The system shall check if the brain is compatible using lexical analysis. If the brain is compatible with the software system it is passed into a colony.
3. The game will have a user interface that can launch the game and input ant brains. In addition this user interface must provide the view for the entire application, so must show the ant progress.
4. It should randomly generate a world for the ants to exist in. This world must contain food for the ants, rocks that an ant cannot pass, two colonies and the ants themselves.
5. It should define the boundaries of the game. In this instance the boundaries are rocks going around the entire map to stop the ants from trying to escape. In addition it must define a hexagonal grid.
6. There should be an ant colony that spawns ants, loads ant brains and manages ant movements. All instructions to the ants are provided by the colony using the brain the player loaded. A colony must also store ant food that is brought back.
7. The game itself should calculate the winner. This will be the colony with the most food at the end of the game after 300000 rounds.
8. The user should be able to import a custom world, if the random world is not sufficient or required.

5.2 Non-Functional Requirements

1. Reliability - The rate of failure occurrence of this game shall be lower than 3%. The game should not crash under normal operating circumstances, and if so any error should be handled and should not affect the overall gameplay of the system.
2. Robustness - During the event of a failure in the application, any errors or exceptions should be handled and not prevent the rest of the application from running.

3. Efficiency – There are significant performance and efficiency requirements to take into account with the application:
 - 3.1 User response time: After the user inputs the brain settings, the game interface should start up automatically in a maximum response time of 3 seconds.
 - 3.2 Event response time: movement of every ant should be computed within 10 milliseconds.
 - 3.3 Screen refresh time: the game interface should refresh at the rate of the monitor 50Hz. The game will compute far faster than this so ant progress will be shown as fast as it is possible for the display to refresh.
4. Storage – The application is likely to be no more than 20Mb in total, however it is heavily dependent on the Java Virtual Machine, which needs to be installed for this application to work. This can be up to 200Mb
5. I/O – The application should be playable with a GUI that will accept keyboard and mouse input. A keyboard is required for building the brain, and the mouse will be used to navigate the GUI.
6. Ease of use - This game shall be easy to learn and played by any user over 10 years old. The GUI will be written for the English language, but should be easily understandable for those who are not as proficient in the language. User documentation should be provided.
7. Portability – The game requires the use of the desktop Java Virtual Machine, so this will need this installed in order to operate the application. Once this is installed it will run on any desktop operating system, which can run the JVM.

5.3 Acceptance Criteria

The acceptance criteria will determine the point at which the user is able to accept the software from the developers and it relies on the following 4 requirements:

- The first is that the user is able to load a brain into the application and that the brain is then parsed and imported into a colony.
- The second is that the user is able to import a world into the application. While most maps are randomly generated, the option to import a custom map is there as well.

- Third is the ability to play a one on one game and have a winner.
- The last is the ability to play a tournament with lots of different players and for it to calculate a winner from that.

The results of the acceptance test is located [10.6 Acceptance Testing](#).

6.1 Project Issues

Project issues define the conditions and constraints on the project. As a small project with no financial consequences or safety of life issues involved there are very few issues and those that are present are low risk.

6.2 Risks

As there are no financial constraints or safety of life issues with the project, the risks associated are somewhat less than a normal software development project. However there are still some issues to consider:

- The project may not be delivered by the deadline.
- The project may be submitted incomplete or with missing supporting documentation.
- Low quality of the final product, where it meets the specification but may be inefficient, confusing or occupy a large file size.
- Requirements creep whereby the project has not strictly adhered to the original specification.
- Issues with low productivity, non-compliance or confusion amongst the development team due to inexperience or other commitments.

6.3 User Documentation & Training

User documentation will be provided with the project. Training is expected to not be required beyond that of the User Documentation as the application should be easy to use, as stated in the requirements.

6.4 Costs

As the project is for an educational purpose, no developers are paid and no money changes hands. All the software used is free for educational use.



DESIGN

7.1 System Design

The design specification is detailed below. It outlines the structure of the software system, and the interconnectivity of each of its components.

For the design of the system, a modular approach using the Model, View, Controller principle was deemed to be the best approach for building the system.

Each of the classes in the class diagram will be written as an interface before any code will be written, with test classes written to meet the interfaces. This approach is referred to as test driven development and it will mean that even though different pairs in the development group will be working on different parts of the application, it should still all work together when all the code is recombined.

Once these interfaces have been implemented and the objects interact correctly. The interfaces used during the development will be removed and the main classes refactored to simplify the source code and allowing for ease of modification in the future.

8.1 Model-View-Controller

The aforementioned MVC pattern, which the implementation of the project will use, is described in greater detail below. The game and tournament classes will be used to control the data in the world. In addition an interface will be built on top that will display game data, however the game will not be dependent on it for operation.

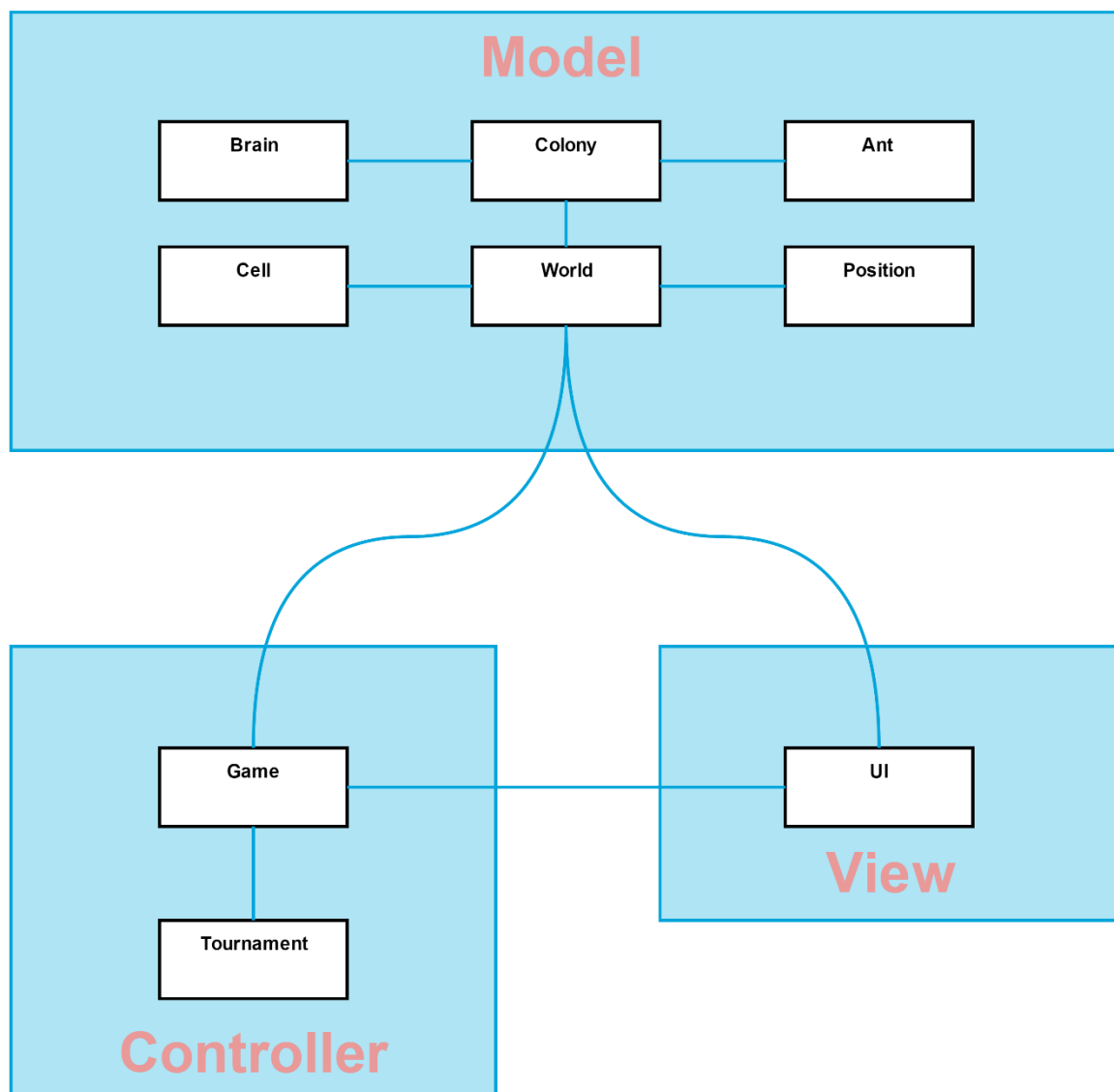


Figure 1 - MVC of the Application

8.2 Class Diagram

The class diagram for our project outlines the structure of the main classes that will be used to construct our application. Game logic is mostly provided by the colony class, which will use an imported ant brain to control the movement of its ants and manage moves and turns.

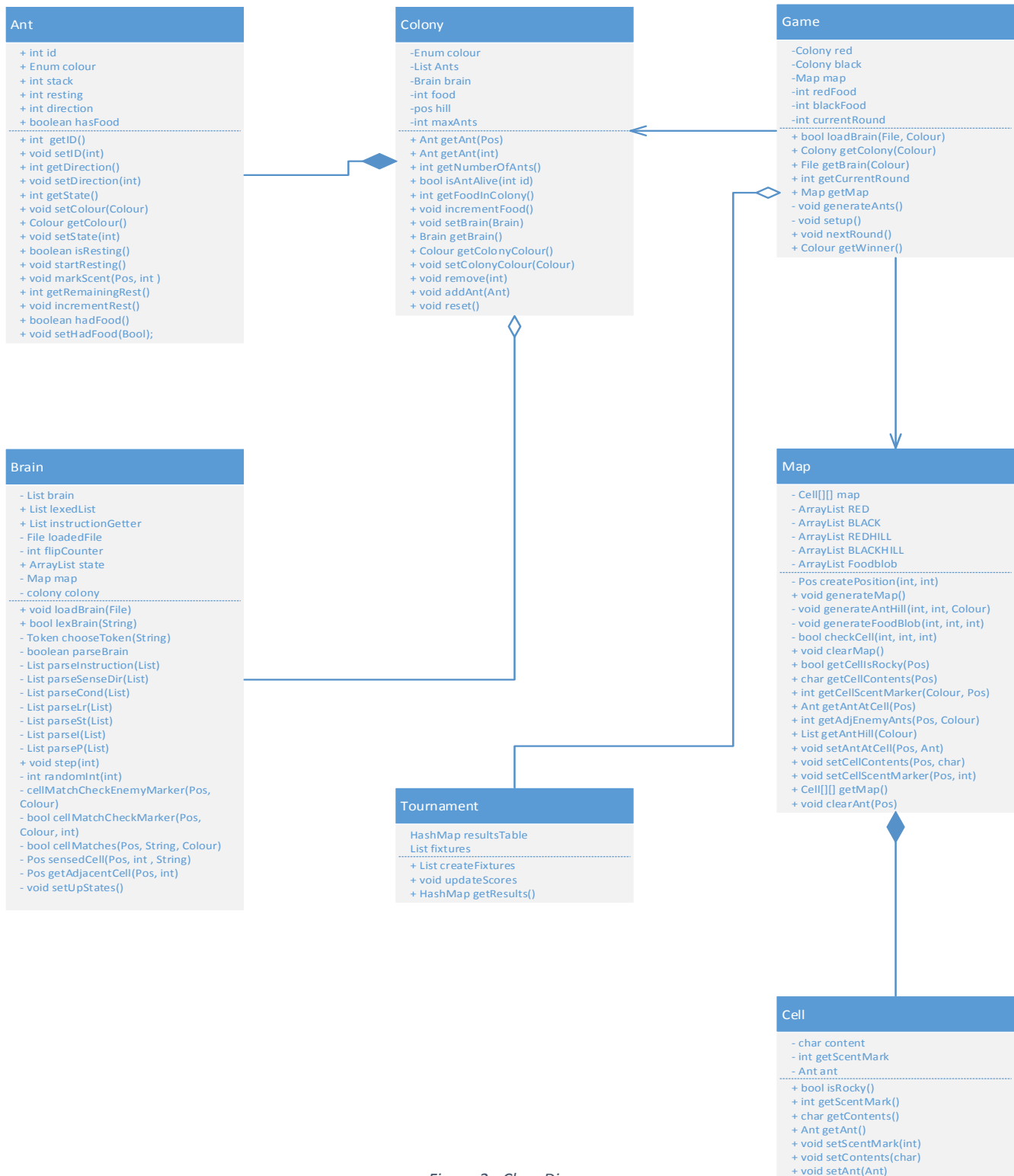


Figure 2 - Class Diagram

8.3 Sequence Diagram

The sequence diagram below outlines the user interaction with the software system and the interactivity of the different components. The user interacts with the system for a very short part of the game lifecycle, with the system running independently of the user once a game starts.

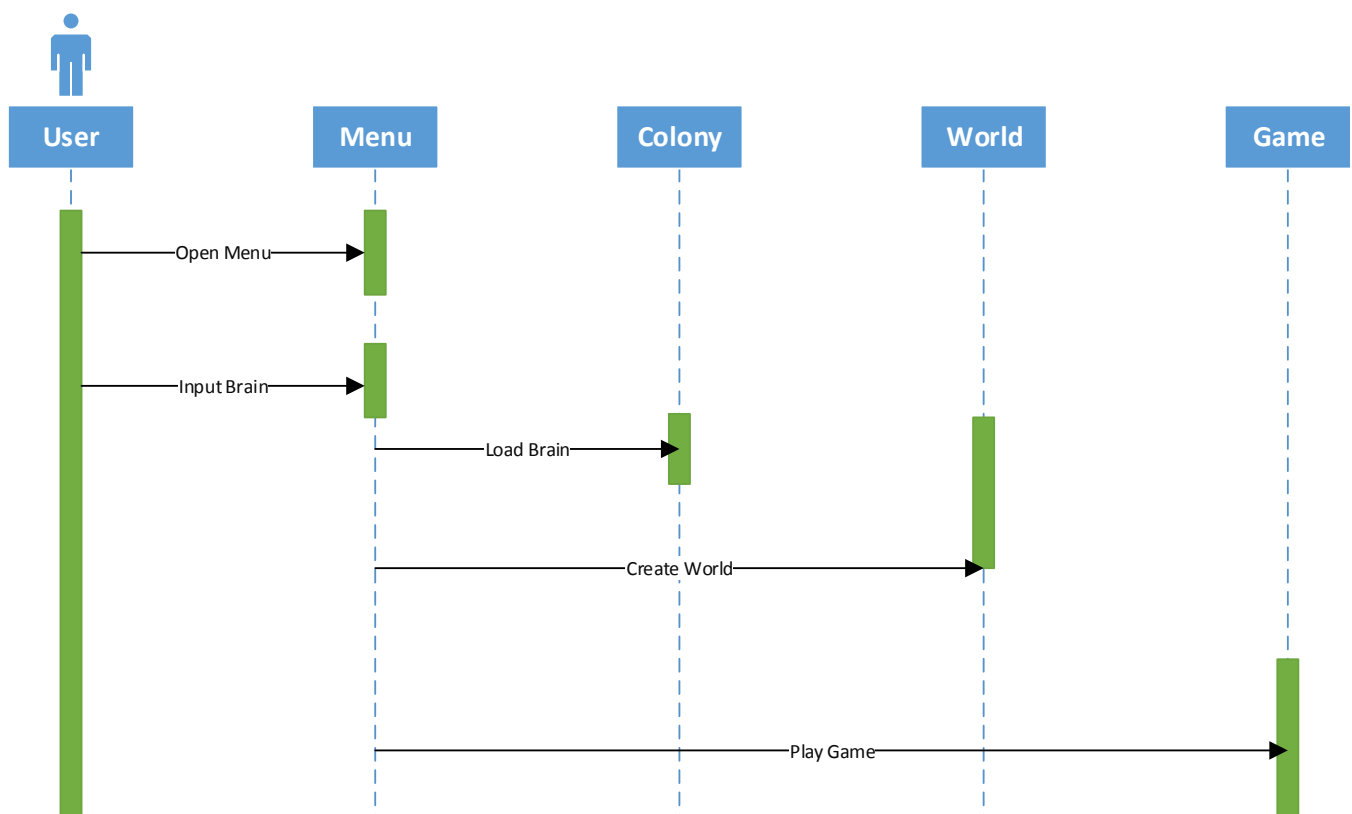


Figure 3 - Sequence Diagram

8.4 Use Case

There are 4 tasks that the user can do in the game, two of which are necessary to play a game. The user first will have to load a brain into the application and then has to option to play a one on one game or to play a tournament. An additional option is to load a custom world.

The diagram below outlines the interaction:

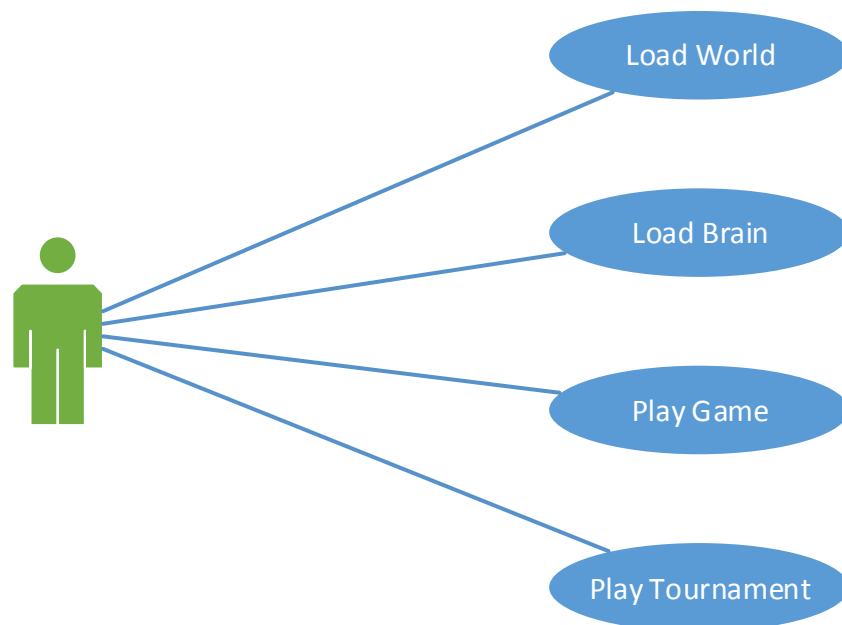


Figure 4- Use Case Diagram

8.5 Gameplay Diagram

The diagram below outlines the interaction between the components when launching a game.

The ant brains are imported into the respective colonies that are created by the game. The world is randomly generated and imported into the game. The game then creates the ants and allocates them to the colonies. After this a game is set up and the game is then free to run.

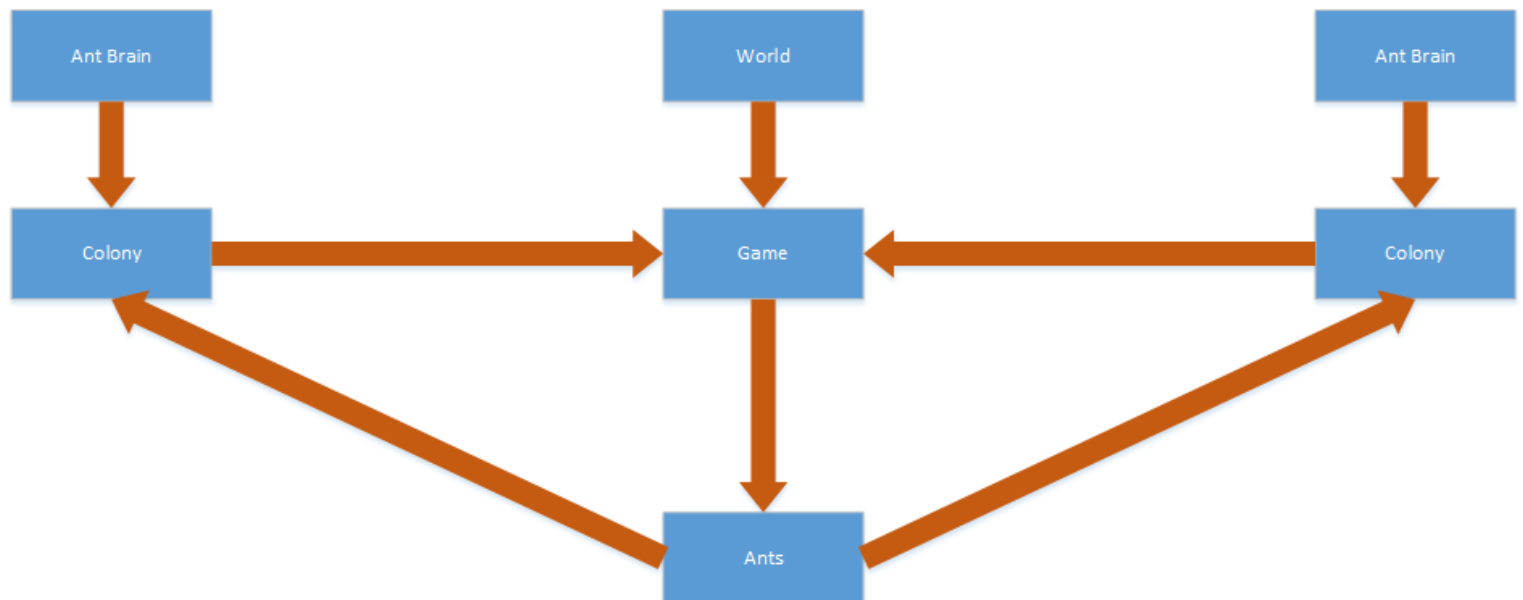


Figure 5 - Gameplay Diagram

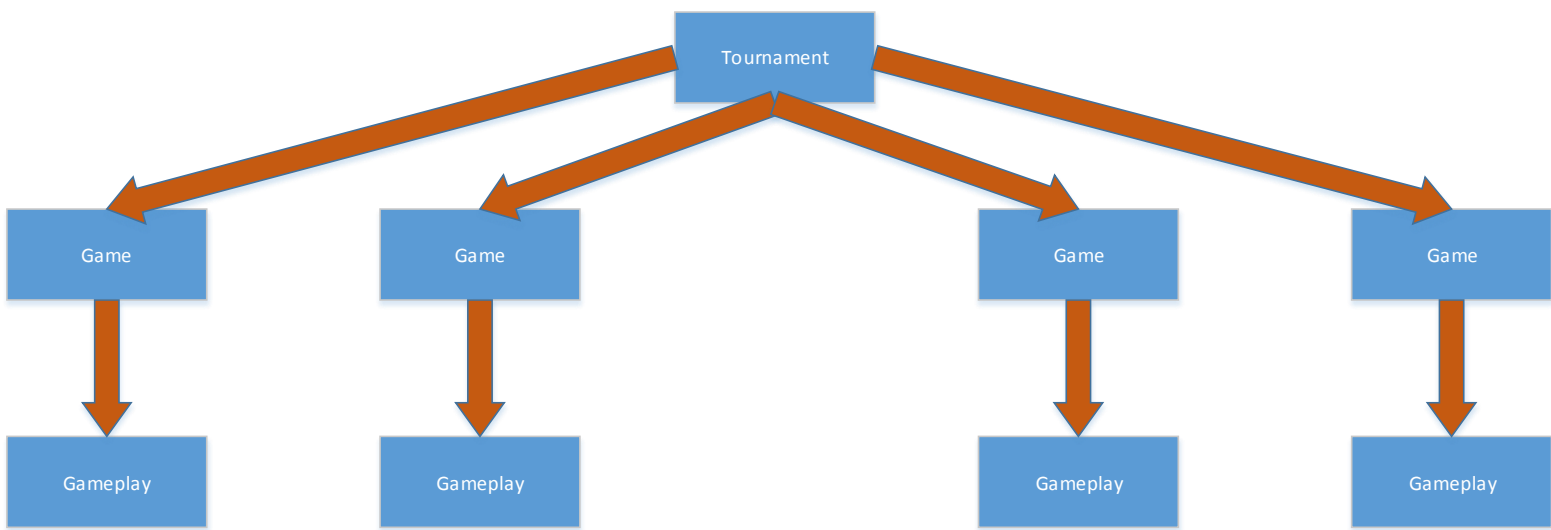


Figure 6 - Game Diagram for Tournament

The above diagram shows the gameplay for a tournament. The tournament can run multiple games, which is dependent on how many players there are. The box for gameplay includes the classes shown in Fig 5.

8.6 Aggregation Association

In standard gameplay the following classes will be in use:

- Many ants will be distributed between two colonies and a brain will be allocated to each colony.
- Each colony will then take part in a game, which is assigned a random map.
- A tournament will host multiple games
- Each map is assigned a random collection of many cells.

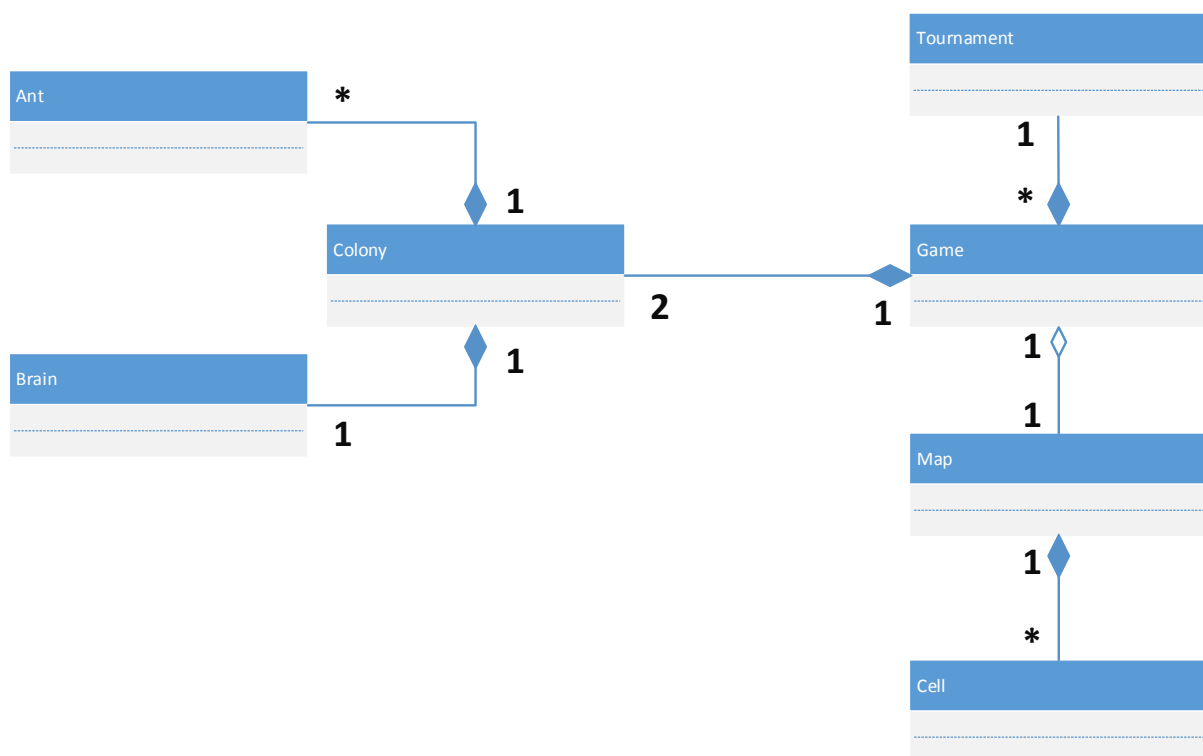


Figure 7 - Aggregation Association



IMPLEMENTATION

9.1 Implementation Strategy

The project will be implemented using a number of strategies. Java was selected for the project as its object-orientated nature is suited to this style of project and it was familiar to the team.

Interfaces are being written before the code is being written, and a test suite is being written to test the classes when they are implemented. Pair programming is being used to implement crucial interface methods, such as the functionality of the brain and the checking of loaded files.

All code is under version control, and is stored on GitHub. Continuous integration is being used and the code is built and tested on each commit using Ant and Travis CI.

9.2 Implementation Timescale

The Implementation should take 10 days from start to finish, and has an extra 4 days available for any overruns or features that were not factored into the design.

As test classes are already written, testing can be performed on the go to make sure methods work correctly, greatly speeding up the development progress.



TESTING

10.1 Testing

The project is using test driven development. Interfaces were written before any of the methods were coded and then tests were written to match the interfaces. This process makes it a lot easier and faster to develop later on. Unit testing is done primarily by the IDE and Travis CI handles integration testing.

Full system testing will be outlined below and this will test how the system works in addition to the GUI.

10.2 Testing Scope

The scope for the testing of the application will be split into four sections.

The first is Continuous Integration testing which performs JUnit tests on the entire application every time a build is pushed to the GitHub repository. This demonstrates how much of the application is working to spec, and gives an overview of how the different components are working together.

The second will be JUnit tests that will be performed by the developers inside the IDE. These were written as part of the design phase, and allow the developers to quickly check if the methods they are writing work.


The third is system testing. This will cover how the application performs as a whole, with specific emphasis on a GUI and menu system, which is not easily covered by unit test.

The last is acceptance testing, this will be done with a customer and will be based off our acceptance testing. If the application meets the criteria then the application is ready for the end user.


























10.3 Travis CI

Travis CI is continuous integration software that creates builds and tests on a software project when code is pushed to a remote repository, for this project that was GitHub.

It allowed us to test the builds we uploaded and have a centralised log of all those builds. It is useful in the sense that it allowed us to get an overview of how the project was progressing.

samkirsten/se  build failing

Current Branches Build History Pull Requests Settings

 master sad  Jun committed	# 172 failed 7b0c8d0	10 sec about 12 hours ago
 master gui update2  tina6397 committed	# 171 failed 964f30d	7 sec about 12 hours ago
 master gui update - customize map buttons added  tina6397 committed	# 170 failed 0fc1417	10 sec about 14 hours ago
 master gui update  tina6397 committed	# 169 failed 9fae1fd	7 sec about 14 hours ago
 master new stuff  alanliang12 committed	# 168 failed 8389876	8 sec about 14 hours ago
 master Merge remote-tracking branch 'origin/master'  tina6397 committed	# 167 failed b594abd	8 sec about 16 hours ago
 master stuff  alanliang12 committed	# 166 failed fdacf97	7 sec about 16 hours ago
 master stuff  alanliang12 committed	# 165 failed df78095	10 sec about 18 hours ago
 master Merge remote-tracking branch 'origin/master'  Jun committed	# 164 failed 40f6870	5 sec a day ago
 master fix the cnflict  alanliang12 committed	# 163 failed 416fd56	8 sec a day ago
 master Commit  Jun committed	# 162 failed 6b9ba60	6 sec 2 days ago
 master resolve  alanliang12 committed	# 161 failed 87f8a76	8 sec 6 days ago
 master Commit	# 160 failed	7 sec FEEDBACK & SUPPORT

10.4 Unit Testing

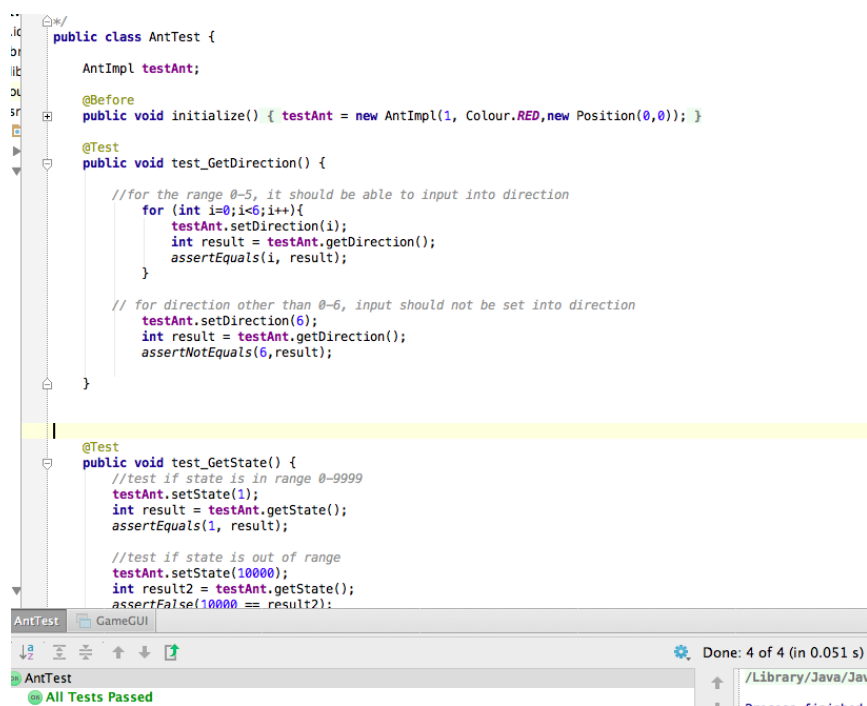
As test driven development was used in this project, a series of unit tests were created before any methods were written and all code that was implemented was tested against these tests.

Refer to the source code for the full unit tests.

There are some examples of unit tests below:

The Ant test has 4 test methods within these test methods they test behaviours multiple times such as the test getDirection method which has a for loop running through and testing each direction.

As shown in the screen shot all tests have passed with no errors.



```
public class AntTest {  
    AntImpl testAnt;  
  
    @Before  
    public void initialize() { testAnt = new AntImpl(1, Colour.RED, new Position(0,0)); }  
  
    @Test  
    public void test_GetDirection() {  
        //for the range 0-5, it should be able to input into direction  
        for (int i=0; i<6; i++){  
            testAnt.setDirection(i);  
            int result = testAnt.getDirection();  
            assertEquals(i, result);  
        }  
  
        // for direction other than 0-6, input should not be set into direction  
        testAnt.setDirection(6);  
        int result = testAnt.getDirection();  
        assertEquals(6, result);  
    }  
  
    @Test  
    public void test_GetState() {  
        //test if state is in range 0-9999  
        testAnt.setState(1);  
        int result = testAnt.getState();  
        assertEquals(1, result);  
  
        //test if state is out of range  
        testAnt.setState(10000);  
        int result2 = testAnt.getState();  
        assertEquals(10000, result2);  
    }  
}
```

AntTest GameGUI

Done: 4 of 4 (in 0.051 s)

AntTest

All Tests Passed

/Library/Java/Jav

Process finished

Figure 8 - Running Ant Class Tests

The brain test has 21 tests, and all passing with no errors, in most test methods each one will test multiple behaviours such as the one shown in the testFailTokens method which test for multiple syntactically wrong tokens used in the brain.

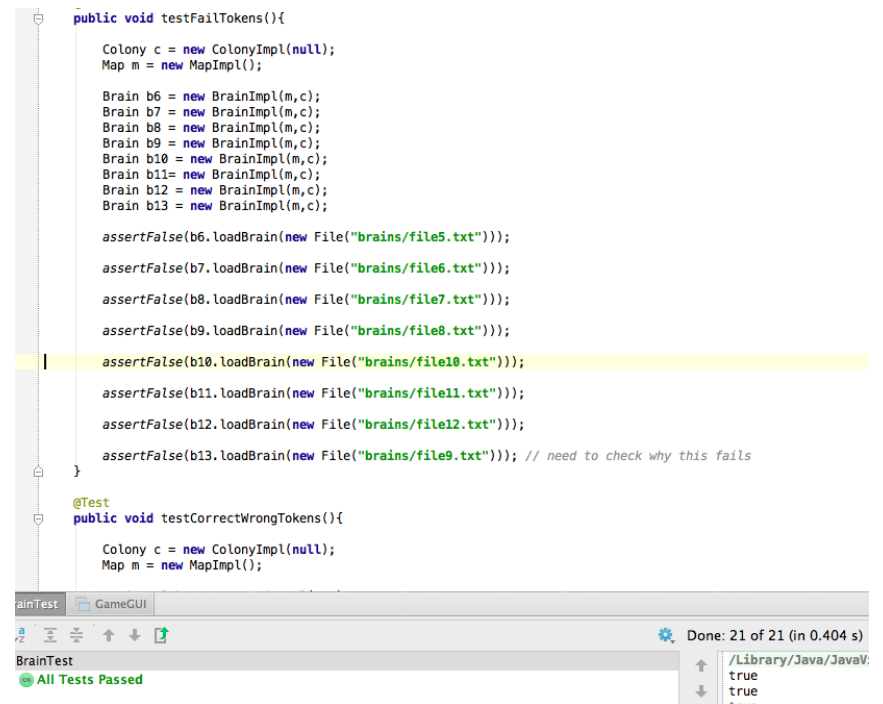
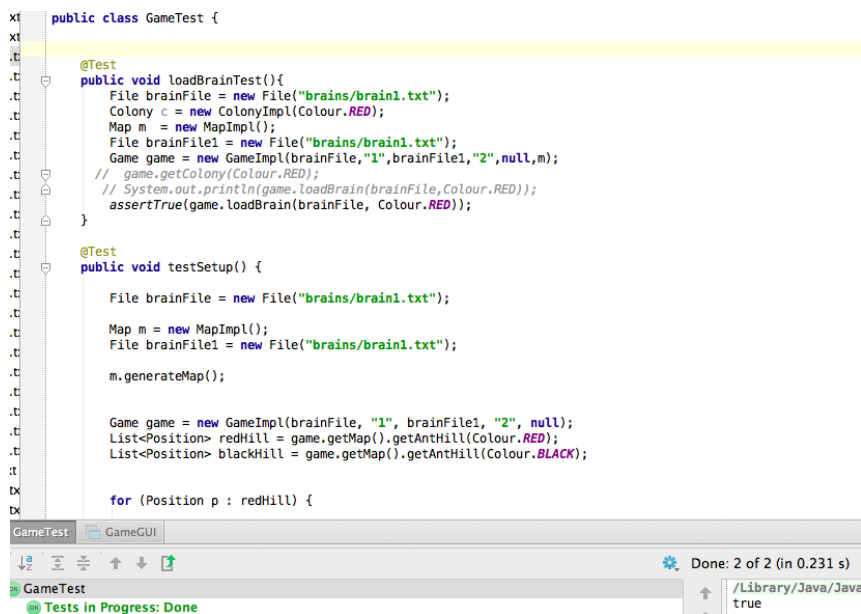


Figure 9 - Running Brain Tests



The game test doesn't have a lot of tests as there are only a few public methods and the rest are private and getters. Private methods are not tested as if the public methods work this shows the private methods work as well. The two tests run without fail.

Figure 10 - Running Game Tests

The map test has 7 test methods in total and again in each test method they test multiple behaviours such as checking for valid content material.

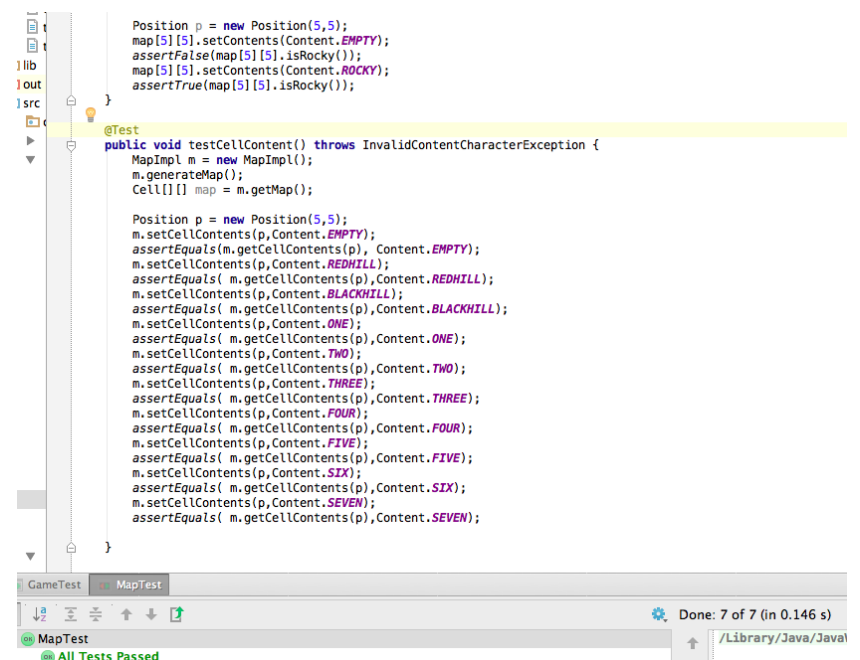


Figure 11 - Running Map Tests

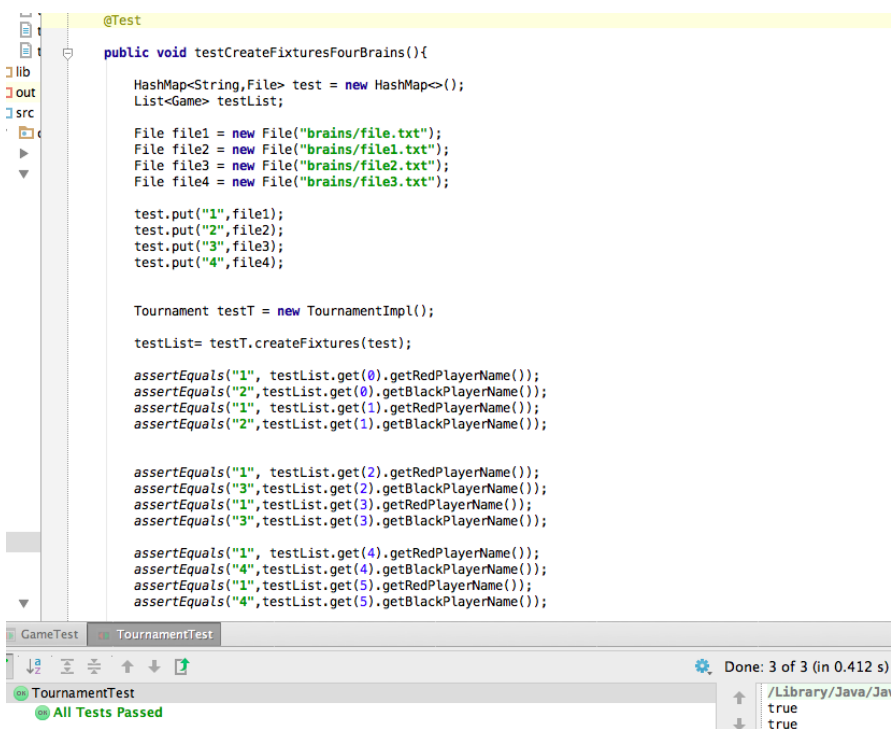


Figure 12 - Running Tournament Tests

The tournament test again has only a few tests as there are only a few public methods to be tested, also each test method has multiple assertion tests checking for multiple behaviours instead of spreading out all the tests to different test methods.

10.5 System Testing

For system testing once the GUI had been linked with the model, what was done first was to run the game to check if the model has linked correctly with the view, however this caused quite a few errors such as referencing different objects. Once these bugs were fixed, the functions within the game could be tested.

Firstly the load brain function was tested. Multiple brains were loaded in, some with correct syntax and some with incorrect syntax to check if the system would reject and accept the correct brain files. This function had no errors and produced the right results.

Secondly, the function to allow a custom world to be loaded in to play a single game was tested. This was tested to check if it only allows syntactically well-formed worlds to be loaded. This function also behaved correctly. The function to randomly generate ant worlds were also tested this was done by generating multiple worlds and checking if the items generated within the world met the requirements given.

The feature to allow a single game to be played on was also tested. This was done by loading two brain files into the game, and running it. This test was also done for a randomly generated world and also a custom world given by the user. This feature had given the correct behaviour, this also tested for if the game could visualise a given ant world.

The last feature to be tested was to check whether the game could allow a tournament to be played; this was done by loading multiple brain files in and checking whether all the players were able to play each other. This feature also gave no problems.

However a problem that had occurred when testing was that the GUI could not render the game image fast enough to keep up with the for loop which runs through the 300k rounds of one game. This was attempted multiple times as some methods, which had been attempted, had failed and other methods had to be used. A method that was tried was using a thread, which would sleep for half a second just before it updates the GUI every time however this did not work. The method used to fix this problem was to make the background behind the canvas to be white and make the cells within the world have no

colour so they would look white, and only repaint the cells, which have changed every time the GUI updates.

10.6 Acceptance Testing

Acceptance testing determines whether the software system is acceptable for the end user. The software system developed will be ran with the user present and if the tests pass the acceptance criteria listed in the requirements it is ready for the user.

Test	Pass/Fail?
Load a custom world	Pass
Load a brain	Pass
Play a game	Pass
Play a tournament	Pass

The acceptance testing was performed as a group with Anthony Trory, all tests passed and the application is deemed suitable for shipping to the end user.



REFLECTION

11.1 Changes from the Original Design

The project generally followed the specification laid out by the original design documents, however some changes, particularly those relating to the GUI, were significantly different.

The changes are as follows:

- The creation of a full Java Swing GUI to control and visualise the Ant game.
- Creation of many more classes, especially those to interface with the GUI and Enums.
- Vastly more methods were required than specified in the design. This is partly due to interfacing with the GUI, but also to account for additional features that weren't necessarily understood from the project specification.
- The MVC deviates slightly from that proposed in the design. Instead of the Game and Tournament forming the controller, they are now part of the model and a separate controller object was created to handle updates the view makes to the model. Although this is technically implicit using the Swing library. It was decided that it would be more beneficial to abstract it completely.
- Creating single games as well as a tournament fixture is all done within the tournament class, as creating fixtures was put in the tournament class it would be unnecessary to repeat the same code and have another method creating just a single game, so therefore we used this method to also create single games.

11.2 Future Improvements

Future Improvements

As has been recognised from user acceptance testing, the current distribution of the system works to satisfy all given feature requirements. However, future releases of the software will implement the following features:

- **Real time user control of game round speed:** Due to the modular separation of the GUI and game logic. It would not be too labour intensive to implement this feature. A buffer holding each game turn's outcome would be stored in the GUI and could then be played back at a user decided speed, adjustable by a graphical slider.
- **Option to run game from command line:** A separate distribution of the game, packaged with a simplistic command line interface would allow the game to be run without the need for the computer to utilise CPU intensive graphics processing for game simulation. This would also benefit users who wish to test their brain files and receive instant game results of its progress.
- **Multistage tournament to allow for more competing brains:** The current system organises tournaments by having every brain play each other, awarding 2 points for a win and 1 for a draw. This is very computationally expensive, and quickly ends in a high number of games to be played. This was the reason for the decision to limit the number of user loadable brains to 4 per tournament. To counter this problem, a future release will allow for a multistage tournament style – similar to the FIFA world cup, where fixtures are created which allow winning brains from each group to progress on to the next stages until a final stage.
- **The use of custom worlds in a tournament:** Currently, custom worlds can only be loaded and played on single games against two players. Future releases hope to expand upon this so that, using the new aforementioned tournament style, each stage of the tournament can be assigned a world for which the players will compete on.

11.3 Maintenance

Maintenance of the software system is not a great priority as this is designed for an educational project, and is unlikely to be used beyond this purpose.

However if the application was to be released publically then it bug tracking and reporting would have to be taken into account. As there are no critical bugs that affect safety or security, bugs could be collected, fixed and then the application could be released in versions, say one a month until most bugs are fixed.

The developers would be in charge of fixing the bugs, as a separate maintenance or QA team would be too much for a project as small as this.

11.4 Review of Waterfall

Waterfall is a useful way of approaching a software development project and for us who are unfamiliar with large software development projects it provided a good framework for us to follow.

It does however have some issues. There were significant changes in the design throughout the project and additional features had to be added and changed, which waterfall does not account for particularly well, and would be better suited to agile.

Overall, while it is not perfect, it was adequate for what we needed in this project.

11.5 Project Management Reflection

The project ran smoothly from a management standpoint. Issues with communication breakdown and team cohesion never arose and work generally ran smoothly. There were issues with overruns in the expected timeframes for each section, however as this is the first time a project of this nature has been attempted by us as a group, these issues were to be expected and didn't significantly affect the project.

We looked at producing a Gantt chart using Redmine however it seemed unnecessary for a project as small as ours, so we settled for Microsoft Project instead.

11.6 Implementation Reflection

11.6.1 GitHub

GitHub caused many problems for the project. This is due to complexities within the software and the lack of understanding as it was an area we hadn't learnt much about.

There were issues with syncing at university as the lab computers didn't author commits properly. In addition there was a DDOS attack on GitHub during our peak development phase that caused problems so syncing.

IDE specific files (.iml files) caused errors as people worked from different directories on their own laptops and when syncing would cause problems when opening the files in IntelliJ. This was resolved by adding .iml files to the .gitignore file.

There were significant sync conflicts as it would fail to merge code which has been written by two different people and would keep both copies causing failed builds and confusion. The GitHub desktop software didn't sync before a commit which caused this issue, the command line version will not allow a push without a pull from the server first so it gets around this issue.

11.6.2 General Implementation

As we made interfaces to work with, everyone generally worked from the same code. A few methods had to be added to the interfaces, as methods were needed for specific functions to work so the interfaces were updated while working on the implementation.

As we did not plan our GUI in a huge amount of detail in the design stage implementing the GUI was difficult:

- Firstly rendering the game image on the GUI caused problems as the repaint() method in java swing could not keep up with how fast the game was running so we had to try multiple methods to get round this issue.
- Secondly, flickering when rendering the image was a problem as half the rendered picture could not be seen as the flicker was very bad in early stages of making the GUI.

11.7 Personal Reflection

11.7.1 Samuel Kirsten

Personally, I am rather impressed with how smoothly the project ran compared to what could have gone wrong. It brought together 5 people who were unaccustomed to this sort of work and managed to produce a substantial project at the end of it.

The project management section did get off to a slow start, and there were some misunderstandings as to what the specification was asking for, especially with the requirements section. However these issues were quickly overcome, and we worked as a team to get through the remaining sections of the project.

The implementation ran smoothly. With the use of test driven development and interfaces the initial phases of the development went smoothly with few significant issues. Issues only arose later on when the project drifted away from what was specified in the design; this was due to us not foreseeing the work that was required.

Project tools such as GitHub, Travis-CI and IntelliJ IDEA were all greatly beneficial in creating a smooth workflow, and GitHub in particular was very useful in allowing code to be shared instantly when it was committed.

11.7.2 Jacob Hughes

Overall, the project was well managed and tasks were clearly defined and delegated between members. Meetings were organised in a way that during each session, a clear desired outcome was agreed upon in addition to the time for the next meeting. This was greatly beneficial to making sure each member understood what was tasks remained and was a key reason for why we did not experience any conflicts.

The speed of the implementation phase would have been greatly improved without technical errors out of our control, such as the GitHub DDOS attack, which led to many sync, conflicts and IntelliJ's build structure breaking as a result. This issue was overcome by

zipping project revisions and sharing them via google drive, although this process was extremely time consuming and pushed us over our implementation deadline.

I feel the pair programming was useful in providing logically sound code, as two members of the team were able to provide their expertise. However, it was very difficult to write code with another person looking over your shoulder. It could be quite tense at times and led to small disagreements amongst the best coding style.

11.7.3 Junho Lee

While the team started the design phase, I felt this section was quite difficult and I think we did not consider designing the GUI as an important factor. As we believed that the important part of the design phase is just the structure of Classes and its method but as we only looked into designing the GUI briefly this caused problems when we came to implement the GUI. If the design for the GUI within the design stage were improved upon, efficiency of implementing the GUI would have been better.

A problem we had faced with building the GUI was the game screen should be updated within very short period, however our game didn't (it took 0.4 seconds to update one frame. After facing this problem, we tried to fix this problem but there were so many possibilities of what was making the game slow e.g. Brain, Map, and game frame in GUI. However as a team we didn't have much real experience in optimizing a game, so we had spent a lot of time researching on this matter. I feel like if we was to do this project again we could spend more time on the GUI within the design phase.

11.7.4 Hao-lin Liang

Overall this project ran relatively smoothly. Although initially the project had started off slowly during the phase of writing up the requirements and design documents, the project became easier once people were more accustomed to working as a group and project roles had been established.

To make the implementation stage easier, Interfaces were created allowing modules to be written by separate people with the same method signatures to avoid clashes. Once the interfaces were completed, the process of the implementation stage ran with few errors and only slight alterations to the original interfaces. The implementation of the model ran with relative ease as a test driven approach to development was enforced. This made sure that all methods had the correct behaviour at the point of creation and allowed for simpler debugging in later stages by safely eliminating possible bug locations. Implementing the view and controller however was proved much more difficult. Interfaces had not been used for these two sections of the application. The creation of the GUI itself was not difficult; yet merging this with the model had revealed some small design flaws, which needed addressing. The unit testing of the model allowed these small bugs to be easily pinpointed to issues within the GUI regarding incorrect object referencing. Another major implementation issue experienced was refreshing the game displayed on the GUI when playing the Ant game; this had proven to be very difficult, as the GUI could not refresh the game content fast enough. The project on the whole went according to plan, however if I were to redo this project I would have preferred the requirements and design stage to be completed earlier, as the implementation and testing stages had exceeded our given deadline due to error testing. In addition, this task was not the only assignment I had and balancing it with solo projects had proved difficult.

11.7.5 Tina Wang

During this project we had encountered difficulties and underestimated the project itself, but we managed to overcome these problems and finished the game. There are a few things that I personally like about the project and a few that we could improve for our future projects.

The thing I like most about this project is how everyone in this team is dedicated to the team. Although each of us got different role and responsibilities, when one of us face any difficulties, other members tend to help and try to solve the problem together. Everyone working as a team made the whole process smoother.

However, things did not go well with the GUI implementation as we imagined before.

A problem we encountered during the implementation stage is that the GUI does not link well with the model we have coded. We tried many approaches to modify the controller and model to make the GUI refresh properly. It was lucky that we started the implementation earlier before the holidays so when we spotted the problem; we still have plenty of time to solve it and can finish up the documentations on time.

Although the project is not an easy work, I enjoyed this experience as working on a programming task in a team.

12.1 Appendices

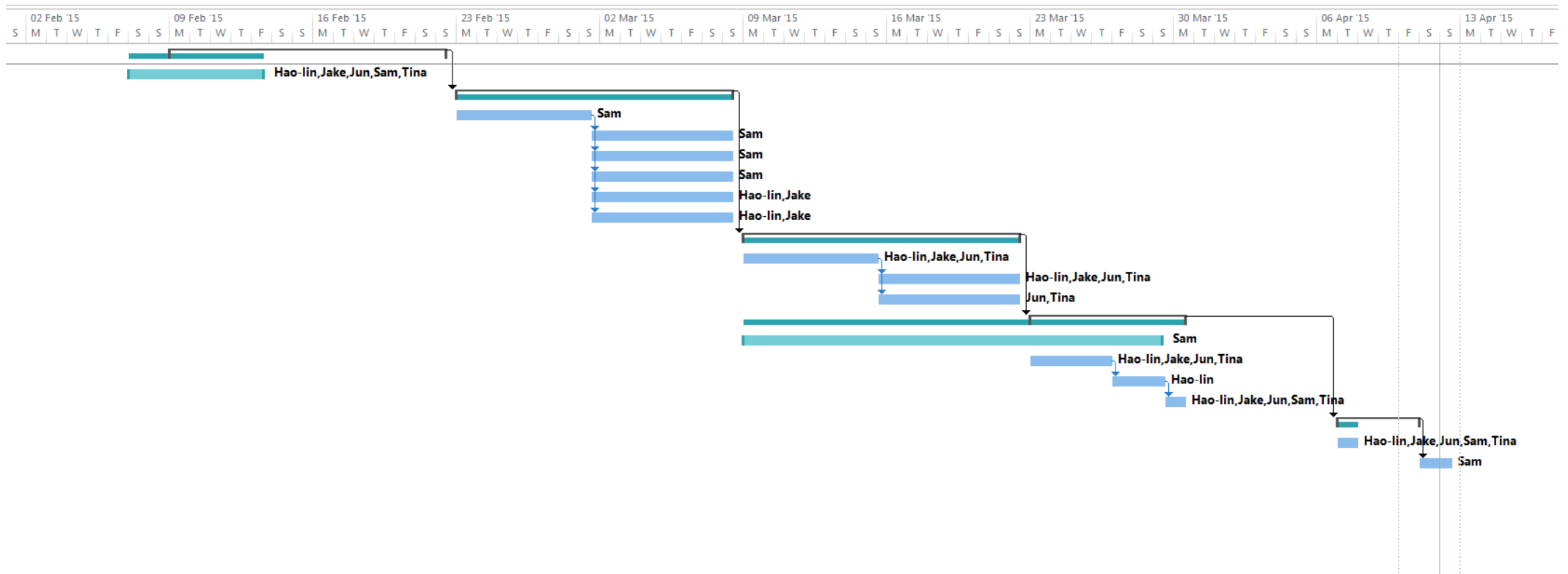
12.1.1 Minutes

Minutes were taken at every meeting. Outlines was who attended, the progress that had been made, the work to be done and any notes that we needed to bear in mind.

Date	Type	Who was present	Progress	For next meeting	Notes
16/02/15	Seminar	Everyone	Set Up GitHub Requirements Draft	Finish Requirements	Use Javadoc Travis CI TDD
19/02/15	Meeting	Everyone	Initial requirements done Discussed application structure	Requirements in one document	
23/02/15	Seminar	Everyone	Started Design Using Volare Spec	Requirements in one document	
06/03/15	Meeting	Everyone	Started interfaces Design diagrams	Finish design	
08/03/15	Seminar	Everyone	Discussed ant-cell paradox Discussed world creation	Generalisation Hierarchy	
09/03/15	Meeting	Everyone	Discussed brain structure and interfaces	Sam to write reports	To use DFS in brain?
16/03/15	Meeting	Everyone minus Sam	General update on progress	Finish all tests	
22/03/15	Meeting	Everyone minus Sam	Finished testing and interfaces	Plan to code	
26/03/15	Seminar	Everyone	Finished all	Improve	Pair

			code specified by design	reports Make a GUI	programming
30/03/15	Seminar	Everyone	Discussed progress Set soft deadline	Write own reflections Testing plan Review of waterfall	

12.2 Gantt Chart



12.3 Critical Path

