|  |
| --- |
| **Group 9** |
| **Software Engineering Project** |
| **Requirements Analysis** |
|  |
|  |
|  |

|  |
| --- |
| Version 1.1 |

Table of Contents

[1.0 - Introduction 2](#_Toc349225088)

[1.1 - Purpose 2](#_Toc349225089)

[1.2 – Scope 2](#_Toc349225090)

[1.2.1 – Gameplay 2](#_Toc349225091)

[1.3 – Definitions, Acronyms and Abbreviations 3](#_Toc349225092)

[2.0 – Requirements Specification 4](#_Toc349225093)

[2.1 – Functional Requirements 4](#_Toc349225094)

[2.1.1 - Ant Brain 4](#_Toc349225095)

[2.1.2 - Ant Control 4](#_Toc349225096)

[2.1.3 - World Generation 5](#_Toc349225097)

[2.1.4 - World Visualisation 6](#_Toc349225098)

[2.1.5 - Gameplay System 6](#_Toc349225099)

[2.1.6 - Tournament System 6](#_Toc349225100)

[2.1.7 - Syntax Checker for Brains/World 7](#_Toc349225101)

[2.1.8 - Random Number Generator 7](#_Toc349225102)

[2.1.9 - Graphical User Interface 7](#_Toc349225103)

[2.2 – Non-Functional Requirements 8](#_Toc349225104)

[3.0 – UML Diagrams 9](#_Toc349225105)

[3.1 – Use Case Diagrams 9](#_Toc349225106)

[3.1.1 – System Overview 9](#_Toc349225107)

[3.1.2 – Ant Brain 9](#_Toc349225108)

[3.1.3 – World 10](#_Toc349225109)

[3.1.4 – Gameplay 10](#_Toc349225110)

[3.1.5 – Tournament 11](#_Toc349225111)

[3.2 – Use Case Scenarios 12](#_Toc349225112)

[3.3 – Sequence Diagrams 13](#_Toc349225113)

[3.3.1 – System Overview 13](#_Toc349225114)

[3.3.2 – Gameplay Sequence 14](#_Toc349225115)

[3.3.3 – Tournament Sequence 15](#_Toc349225116)

[4.0 - References 16](#_Toc349225117)

# 1.0 - Introduction

This section introduces the requirement specification document for the Ant Attack application which provides the purpose and scope of the application. All definitions and references are included along with an overview detailing the remaining requirements.

## 1.1 - Purpose

This document presents a comprehensive description of the Ant Attack application. It will clarify the main purpose and features of the application including what the system will do. This document will assess fundamental constraints of how the system will function; identifying all functional and non-functional requirements. Specifications well set out how the overall application reacts to specific stimulus, its responses and how users interact with the system via a GUI. This document is put forward for both stakeholders and developers of the application.

## 1.2 – Scope

The system planned is “Ant Attack”, which shall allow two-players to play a strategy game that allows users to design FSM ant brains. The system shall check to see if the brains supplied are syntactically well formed. The brains will be loaded from files into either a random generated ant world or a user-designed ant world. The user can then run a game that simulates the behaviour of two different ant species that belong to two different ant colonies. Both the random-generated and user-designed ant worlds should be syntactically well-formed.

The system will check that each random ant world meets the requirements used in tournaments and must allow an arbitrary number of players to upload ant brains to play against each other in a tournament. Each given ant world should contain elements including anthills, rocks and food and randomly-generated worlds must adhere to these requirements. The system will also provide a GUI that can present the ant world graphically and allow users to interact with the system.

The system will not allow more than two players to play a game unless it is a tournament and shall only reveal the winners of a tournament once 300000 rounds have been played. The winner shall be determined by the player who has collected the most food particles on their anthill. The system should not allow any food being carried by ants to be counted at the end of 300000 rounds and must not allow ants to be any other colours apart from black or red.

### 1.2.1 – Gameplay

The objective of the game is for each ant species to explore the world, collecting food. The ant hill with the most food wins the game. The ants communicate with other ants within their team by leaving trails. These trails are left by means of chemical markers and have limited sensing capabilities. This allows the ants to sense, but not modify the markers of the other species. There are various obstacles in the world that hinder the ant’s performance such as rocks and being attacked by the opponents species. An ant will die and become a food particle if it's completely surrounded in an attack of the opponents species.

## 1.3 – Definitions, Acronyms and Abbreviations

*FSM* – Finite State Machine is a mathematical abstraction used to design computer programs. It allows the computation structure of programs to be expressed in a specific, graphical manner.

*Java* – A portable object orientated computer language developed by Sun Microsystems used to implement applications.

*GUI* – Graphical User Interface that displays graphical elements onto the computer screen and allows users to interact with the system.

*Functional* *requirements* – defines the functions of the software system. Functions can be described as a set of inputs, the behaviour and outputs of the inputs.

*Non-functional requirements* – are identified by the operation of the overall system.

*UML* – Unified Modelling Language: Allows for diagrammatic representation of system behaviours, features and interactions.

# 2.0 – Requirements Specification

What follows is a list of the functional and non-functional requirements determined from the system brief provided by the customer. Upon implementation of the system these requirements will be used to confirm that the system works as promised and conforms to all specifications laid out herein.

## 2.1 – Functional Requirements

### ****2.1.1 - Ant Brain****

* **Representation:**
  + **FSM of x states exactly x lines long**
  + **Line number x = state**
  + **Each state contains command and arguments**
* **Functions:**
  + **Move – move forward one cell, change state**
  + **Turn – change direction left or right by one direction to face new direction, change state**
  + **Sense – change state based on condition of adjacent cell**
  + **Flip – generate random number to determine state change**
  + **Mark – set a mark of number i of ant’s colour in current cell, change state**
  + **UnMark – remove mark of number i of ant’s from current cell, change state**
  + **PickUp – pickup food from current cell, change state**
  + **Drop – drop food in current cell, change state**

### ****2.1.2 - Ant Control****

* **Ant Properties:**
  + **Colour – red or black identifying team colour**
  + **ID – unique per ant per team**
  + **State – state of FSM representing brain**
  + **Resting – rest period before next action legal**
  + **Direction – forward facing of ant in game world**
  + **HasFood - whether ant is carrying or not carrying food**
* **Movement capabilities:**
  + **Within the hexagonal layout constraints of the world, 6 directions**
  + **Move North East, East, South East, South West, West, North West**
  + **Movement based on direction, turn to face the next direction sequentially based on left/right**
  + **Directions:**
    - **0 – East**
    - **1 – South East**
    - **2 – South West**
    - **3 – West**
    - **4 – North West**
    - **5 – North East**
  + **Determine adjacent cell using direction and current position:**
    - **E, SE, SW, W, NW, NE**
* **Sensing capabilities:**
  + **Current cell**
  + **Ahead – directly ahead of ant**
  + **Left or right ahead – ahead plus left/right turn**
  + **Friend/foe identification**
  + **Rock/impassable terrain**
  + **Food/ant(friend/foe) with food**
  + **Markers (friendly/enemy)**
  + **Home position (friend/foe)**
* **Actions:**
  + **Move – as above**
  + **Mark/Unmark position/cell**
  + **Pickup/Drop food**
  + **Turn – as above**
  + **Flip – change state based on random number x**
  + **Set chemical marker 0-6**
  + **Sense friendly marker and marker value**
  + **Sense enemy marker (not value, just presence)**
  + **Set state of ant to s**
  + **Set resting value to a value r**
  + **Set direction to direction d**
  + **Set whether ant has food to true**

### ***2.1.3 - World Generation***

* **Hexagonal grid**
* **Co-ordinate system:**
  + **x,y – x = column, y = row**
  + **0,0 – Top left of world**
  + **Odd rows are offset by 1 to accommodate hexagonal grid shape**
* **Identifiers for:**
  + **Rocky cells**
  + **Clear cells**
  + **Anthills (red/black)**
  + **Food particle (number representing quantity in that cell)**
  + **Chemical markers per team**
  + **Ant is present**
* **Contest requirements:**
  + **150x150 cells**
  + **Rocky perimeters**
  + **2 anthills – sides length 7, 6-way symmetric (hexagonal)**
  + **14 rocks**
  + **11 cells of food – 5x5 rectangular layout each cell containing 5 food**
  + **Ants spawned in anthill cell facing direction 0 (East)**
* **Functions:**
  + **Is there an ant at cell p**
  + **Get ant at cell p**
  + **Set ant at cell p**
  + **Clear ant at cell p**
  + **Check ant is alive given id**
  + **Check ant position given id**
  + **Kill ant (clear ant at cell) give id**
  + **Check for food at cell p**
  + **Set food at cell p**
  + **Check for anthill at cell p and of colour c**

### ****2.1.4 - World Visualisation****

* Plain text file
* Line 1 – x dimension
* Line 2 – y dimension
* Rest of file = y lines of x characters
  + # Rocky
  + . Clear
  + + Red Anthill
  + - Black Anthill
  + 1-9 Food Particles in cell

### ****2.1.5 - Gameplay System****

* Load ant brains from ant brain definition
* Cells of anthills each populated by 1 ant facing direction 0 (East)
* Ants assigned IDs based on left-to-right, top-to-bottom scan of the world cells
* Execute 300,000 rounds
  + Per round iterate through ants
  + Execute ant brain instruction based on state of FSM and random number generation
  + Detect surrounded ants (5 or more adjacent enemy ants)
    - Kill ant if true
    - Turn into 3 food particles at cell
* Count food particles per anthill cell per team after round 300,000
  + Most total food particles = winning team

### 2.1.6 - Tournament System

* Match ant brains against another ant brain
* Allow n number of ant brains to compete
* Execute gameplay system with those two ant brains
* Determine winner
* Tally wins per brain
* Result of tournament = ant brain with largest number of wins

### 2.1.7 - Syntax Checker for Brains/World

* Analyse files representing world:
  + Check for dimensions in first two lines
  + Iterate through the rest of lines and check layout is correct (odd rows are offset +1)
  + Check world conditions meet specification
    - Dimensions
    - Number of elements
    - Bordering rocky cells
* Analyse files representing ant brains:
  + Check for erroneous tokens
  + Ensure tokens follow correct syntax (take correct number of arguments etc.)
  + File length <= 10000 lines
  + Comments laid out so ignored by game, using semicolon to denote comment

### 2.1.8 - Random Number Generator

* Return random number between 0 and n-1 when given number n
* Use seed to determine random number calculation
* Match generator to specification

### 2.1.9 - Graphical User Interface

* Print ant world to screen
* Allow user to interact with system with mouse and keyboard
* Generate statistics (ant food etc.) from game and display on screen

## 2.2 – Non-Functional Requirements

* Deadline – Thursday 18th April 2013
* Team members - 6
* Running entire game with everything active- response time
* Robustness, reliability, maintainability, scalability and usability
* The dimensions of the world must always be 150x150 cells when in tournament
* Game must support two players simultaneously
* Cells on the perimeter are always rocky
* Ant brain can contain up to 10,000 states
* There must always be at least one empty cell between non-food elements
* Every random world must contain exactly the same elements to compete in a tournament
* The position and orientation of the elements are chosen randomly
* Each blob must contain 5 food particles
* Capable of running for 300,000 rounds in a reasonable time
* Ease of construction of ant brains
* Well documented code and features
* Well tested for reasonable inputs and scenarios
* Maintainable code base that facilitates extension in the future (e.g. modularity, encapsulation)

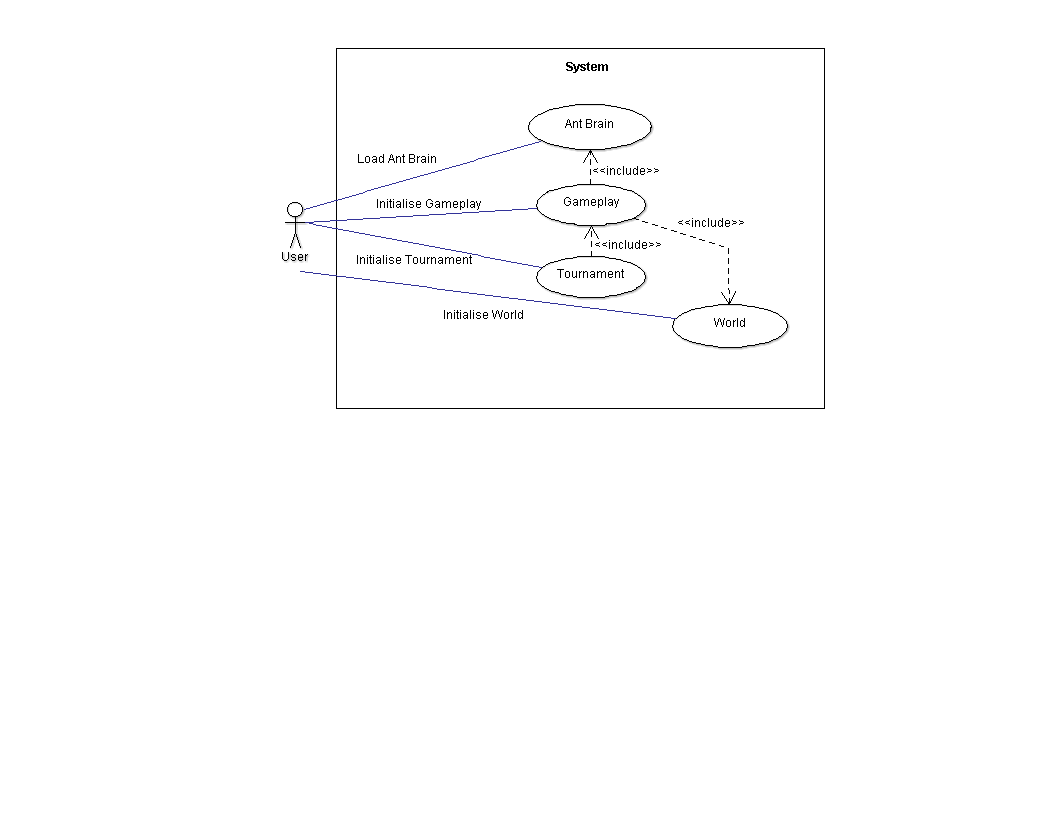
# 3.0 – UML Diagrams

This section presents a selection of initial UML diagrams developed from the information garnered from analysing the system requirements, specifically focusing on user and high-level system interactions within the system.

## 3.1 – Use Case Diagrams

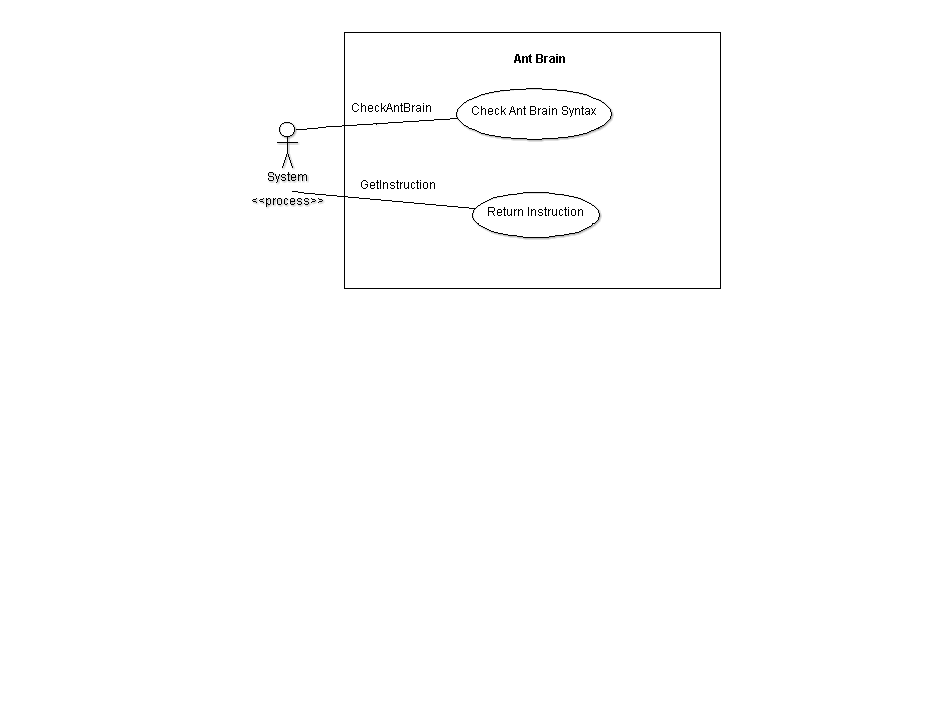
Use Case diagrams details interactions with the system from the perspective of users and elements within the system interacting with other elements.

### 3.1.1 – System Overview



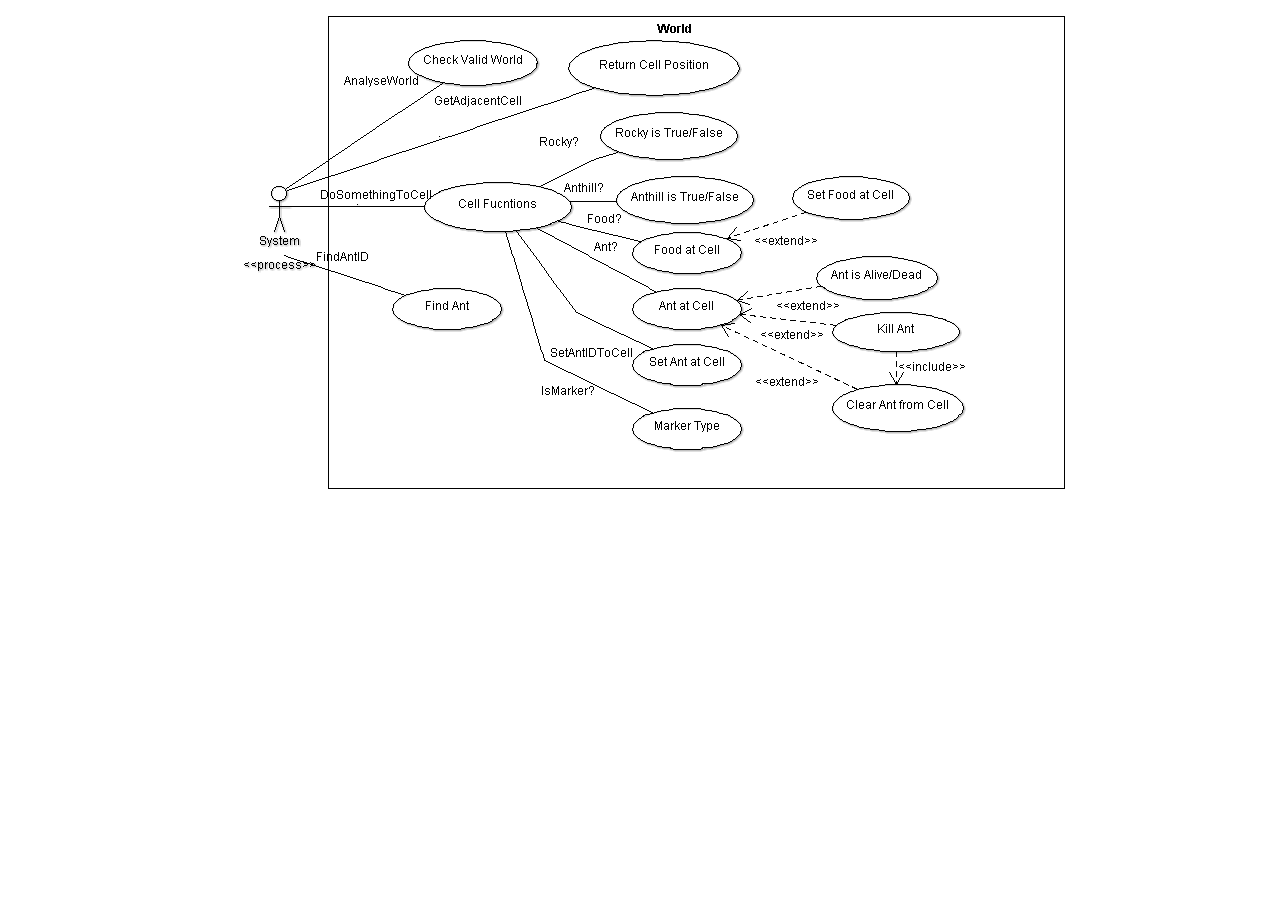
* User Initialises **tournament** and **gameplay** via a GUI.
* The user initialises world via a GUI.
* User loads **ant brain** to the system via a GUI.

### 3.1.2 – Ant Brain



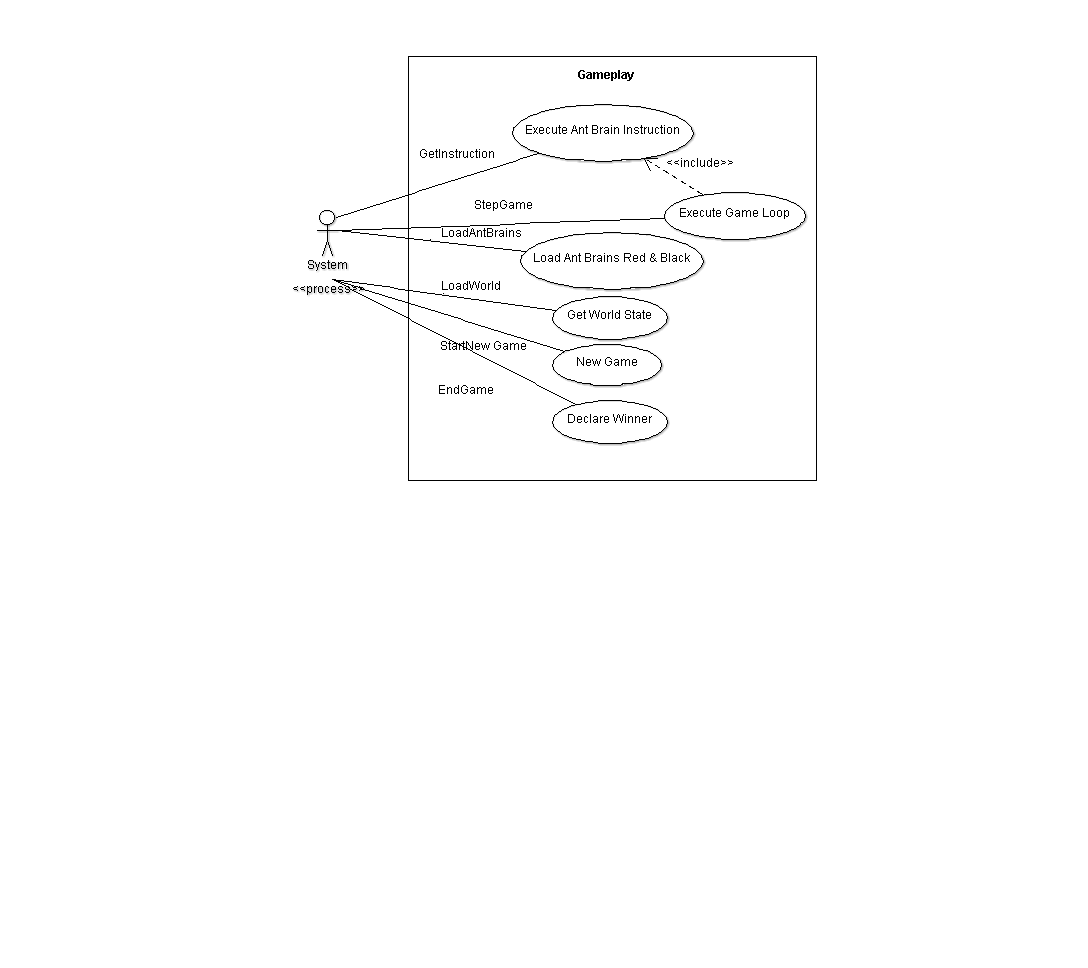
* System checks **ant brain** is syntactically correct.
* System gets and executes **ant brain** instruction.

### 3.1.3 – World



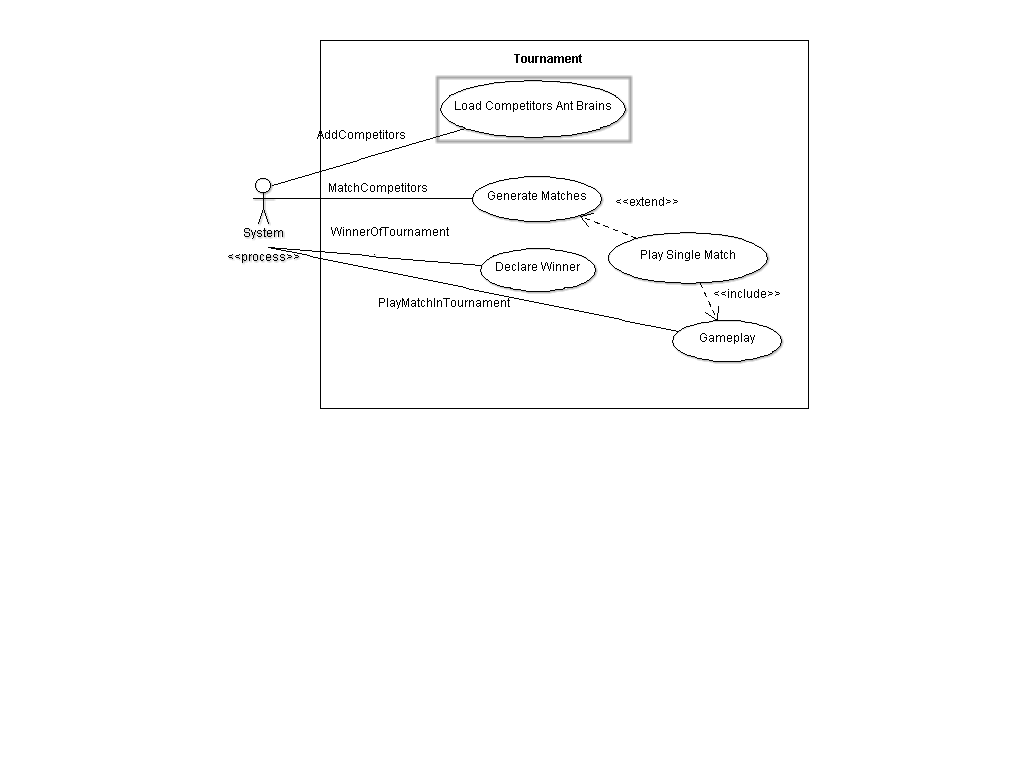
* System analysis **world** to check it is valid.
* System calls **get adjacent cell** function which returns **current cell** position.
* System calls **find ant ID** function and returns the ant with the corresponding **ant ID**.
* System calls the **do something to sell** function
  + Returns true of false if cell **is rocky**.
  + Returns true or false if cell **is anthill**.
  + If **food** is in cell, then set food to current cell.
  + If ant is in cell is **ant alive/dead**?
  + **Kill ant** and **clear ant** from cell.
  + Set **ant ID** at current cell.
  + Returns **marker** type if cell **is marked**.

### 3.1.4 – Gameplay



* System loads new **ant world** and checks it is syntactically correct.
* System checks red & black **ant brains** are syntactically correct and loads them to the **ant** **world**.
* System starts **new game**.
* System gets and executes **ant brain** instruction
* System activates the **step** function
* Game loop is executed.
* **Game ends** and system declares a winner after 300000 steps.

### 3.1.5 – Tournament

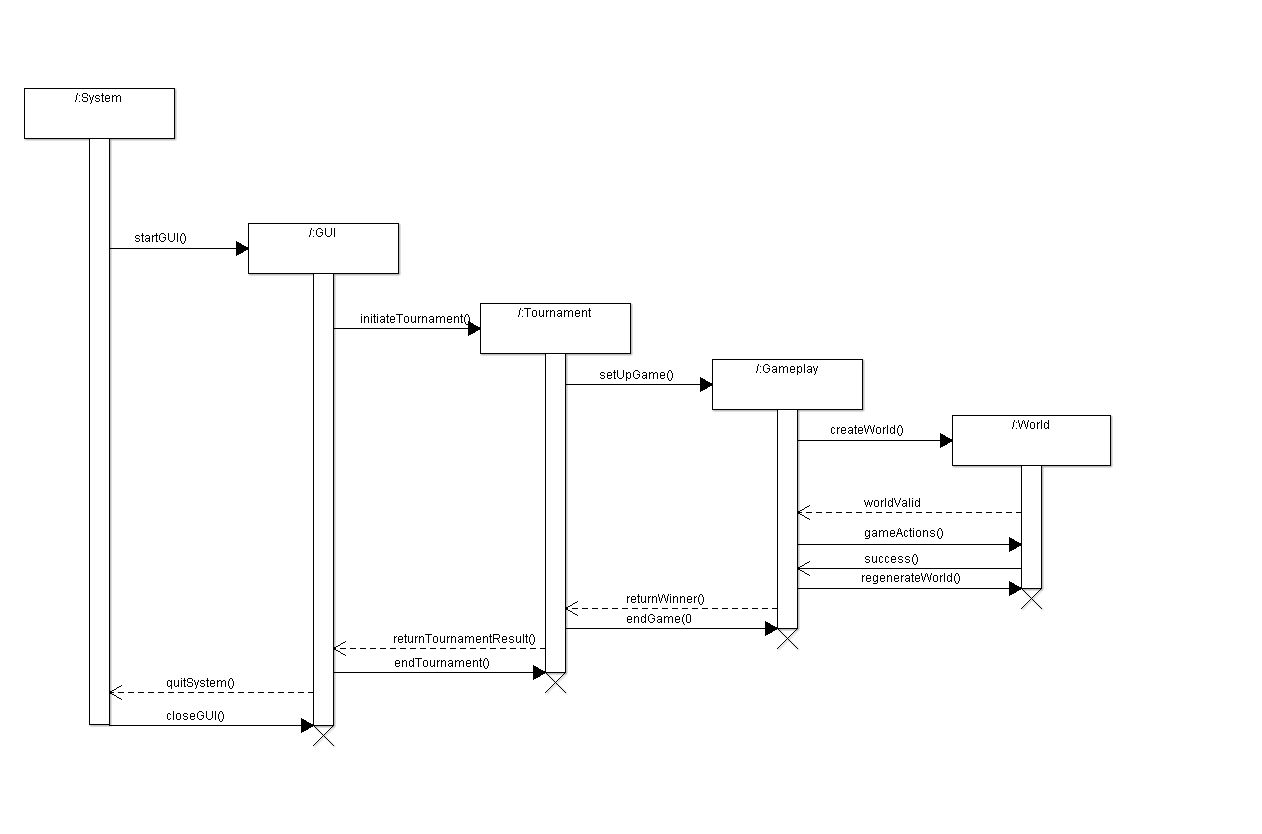


* System adds **competitors** to **tournament** and loads **ant brains**.
* System generates and **matches competitors.**
* System calls **play match in tournament** which initialises **gameplay**.
* System declares **winner** of **tournament**.

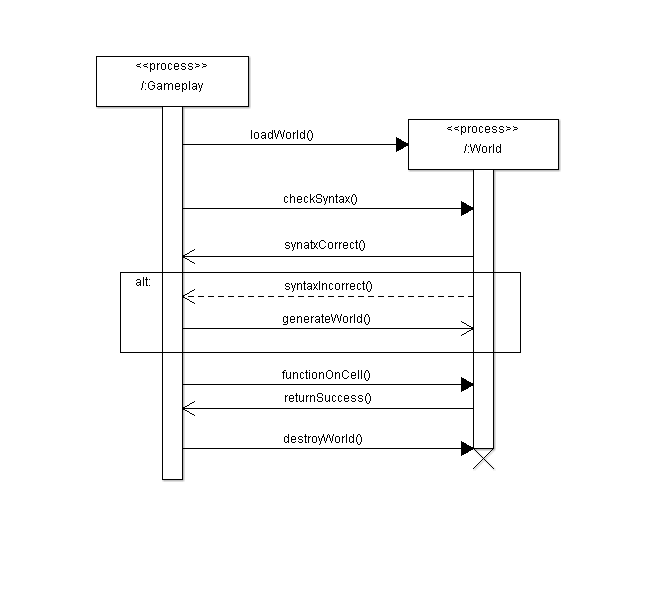
## 3.2 – Sequence Diagrams

Sequence diagrams are usually concerned with how processes within a system operate with one another with strong emphasis on ordering of processes.

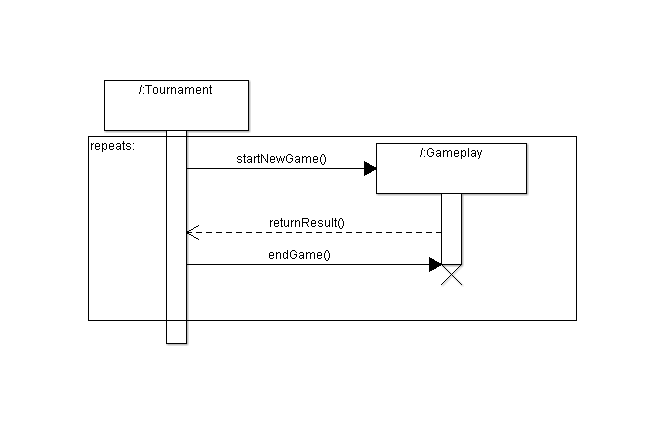
### 3.2.1 – System Overview



### 3.2.2 – Gameplay Sequence



### 3.2.3 – Tournament Sequence



# 4.0 - References