Detailed Design Specification

Software Engineering Group 6

01/05/2012: Project Plan, v1.0a

May 2012: Third Deliverable

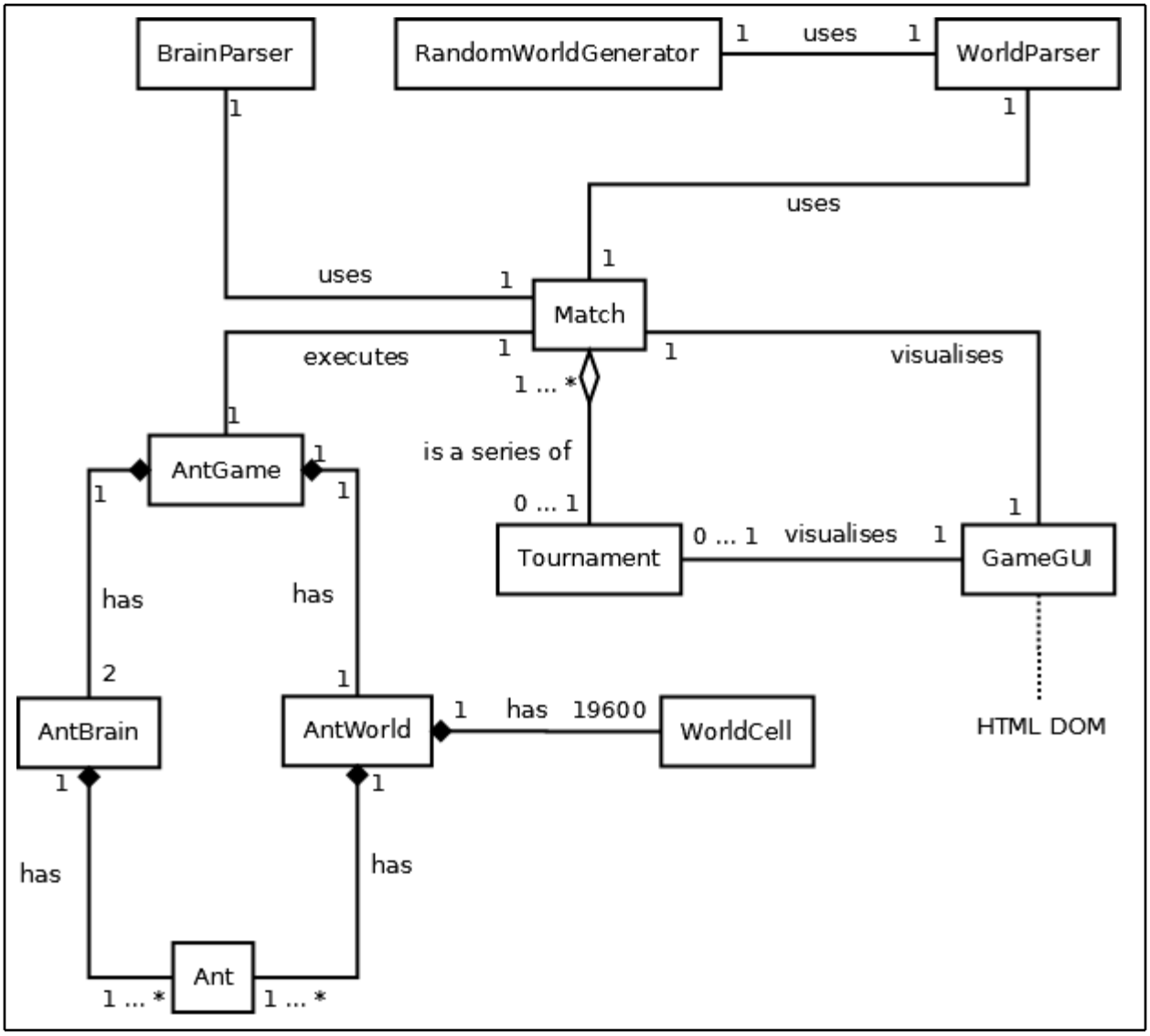
Contents: **Page No:**

Introduction

The Detailed Design Specification has been created to provide more detail and context when used with the High level Design Specification, this document goes into additional detail about the specific layout of the programming and includes any revisions that have been made between the creation of the High level Design and this Detailed Design Specification. This document will provide greater detail about the components making up the program and specify specific methods that the program may use. This document will be used by the Programming team alongside the Customer Requirements documentation and the Acceptance Criteria.

Architectural Overview

The architectural design has evolved since the High level Design Specification, and the specific ways certain classes interact have been subtly altered.



Changes in Architecture

The main changes when compared to the High level Design are as follows:

The way the RandomWorldGenerator interacts with the rest of the program has been changed, rather than directly interacting with the match class, it has been redirected to pass information to the WorldParser, This way the program has a uniform and consistent way of calling the input for the World into the game.

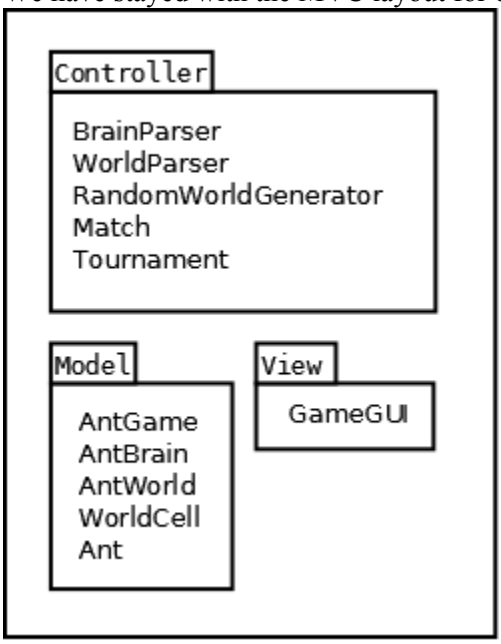
A new class called WorldCell has been introduced. This is will make the system simpler to understand, and easier to modify in future should the customer decide they want to add features to the program. A good example would be if they wanted to add an additional type of cell into the world.

The WorldVis, Menu and GameGui classes have been condensed into a single GUI class, this is due to the fact that the implementation of a GUI in JavaScript is simpler than if you were coding in an alternative language such as Java as the browser performs the majority of the functions.

AntSimulation has been renamed to AntGame as this better reflects the classes functions.

Logical Groupings

We have stayed with the MVC layout for our program, logical groupings as follows:



Model Overview

Controller Overview

RandomWorldGenerator
-seed: int
+RandomWorldGenerator()
+getRandomWorld(): AntWorld
WorldParser
+WorldParser(source)
+getAntWorld(source): AntWorld
BrainParser
+BrainParser(source)
+getAntBrain(source): AntBrain
Match
-brain1: AntBrain
-brain1Score: int
-brain2: AntBrain
-brain2Score: int
-world: AntWorld
-currentStep: int
-steps: int
+Match(brain1:AntBrain,brain2:AntBrain,world:AntWorld)
+run()
+getViewInfo()
Tournament
-brains: Array<AntBrain>
-brainsScores: Array<int>
-worlds: Array<AntWorld>
-currentMatch: int
-Matches: int
+Tournament(brains:Array<AntBrain>,worlds:Array<AntWorld>)
+getCurrentMatch()
+getScore()
uses
uses
uses
is a series of
1
1
1
1
1
1
1
1

Ant

The Ant class is responsible for simulating the behaviour of the ant using the information fed in from the AntBrain class and feeds this simulated behaviour back into the AntWorld class. Important features of the Ant class include a unique integer ID; an AntBrain object; a team colour and a pair of integers representing the Ants' position on the grid.

Method	Description
Constructor(antBrain,posX,posY,team)	An ant is created using an ant brain, while also being given a position and a team.
Step()	Increments to the next Step.
getState()	Returns the current state the ant is in.
getTeam()	Returns the team the ant is on.
isResting()	Checks to see if the ant is resting.
hasFood()	Checks if the ant has food.
checkForDeath()	Checks if this ant should be killed.
checkForAdjacenDeaths()	Checks to see if surrounding ants should be killed.
getAdjacentCell()	Returns which cell the ant is facing.
getCurrentCell()	Returns Which cell the ant is in.

WorldCell

The WorldCell class helps the AntWorld and AntGame classes distinguish between the different types of the cell that are being implemented in the game. The key parts of the WorldCell class are the different classes that the cell could take one (Ant,Empty,Rock,Food,Anthill) and the location that the cell is in the

world. A design decision was made to extract the WorldCell class from the rest of the AntWorld class due to the fact it will make future modification of the game easier. For example new types of cell would be easier to introduce using this layout within the program.

Method	Description
WorldCell(type)	When the WorldCell is created it is fed a string that designates the cells type.
addMarker()	Adds a marker to the WorldCell.
removeMarker()	Removes a marker from the WorldCell.
addFood()	Adds food to the WorldCell.
getLocation()	Returns the WorldCell's location within the world.

BrainParser

The BrainParser class is a static Utility class with one public method. Its job is to compile source code written for an ant brain into an intermediate representation.

Method	Description
getAntBrain(source)	Compiles source code, returns intermediate representation or undefined if syntax error.

WorldParser

The WorldParser class is a static utility class with one public method. Its job is to compile text representations of game worlds into an intermediate representation.

Method	Description
getAntWorld(source)	Compiles source code, returns intermediate representation or undefined if syntax error.

RandomWorldGenerator

The RandomWorldGenerator is a static utility class with one public method. Its job is to randomly generate Ant World source files suitable for use in contests.

Method	Description
getRandomWorld()	Returns a randomly generated source file to be fed into the WorldParser class.

Match

The Match class represents a pairing of two ant brains to compete against each other in a particular world. It keeps track of scores and other game statistics. It also provides hooks for the GUI to receive updates on the game's progress.

Method	Description
Constructor(brain1,brain2,world)	Creates an instance of the Match class
run()	Starts the match simulation
getViewInfo()	Returns information for the GUI to update itself

Tournament

Instances of the Tournament class construct a list of Matches to be run according to contest rules. They keep track of multiple ant brains and their overall scores from matches on multiple worlds.

Method	Description
Contructor(brains,worlds)	Creates an instance of the Tournament Class
getCurrentMatch()	Returns the current/next match object running/to be run
getScores()	Returns a list of brains and their respective scores within the tournament.

AntGame

The AntGame class coordinates the running of the simulation. It populates the world with ants and controls the speed of the simulation.

Methods	Description
Constructor(redBrain,blackBrain,world)	Creates an instance of the AntGame class
run()	Starts the simulation
setSpeed()	Sets the simulation speed.

AntBrain

The AntBrain class describes a list of functions which act upon an ant in accordance with states described in ant brain source code.

Methods	Description
Constructor(intermediateCode)	Takes the intermediate representation of ant brain code and blindly converts it into functions.

GameGui

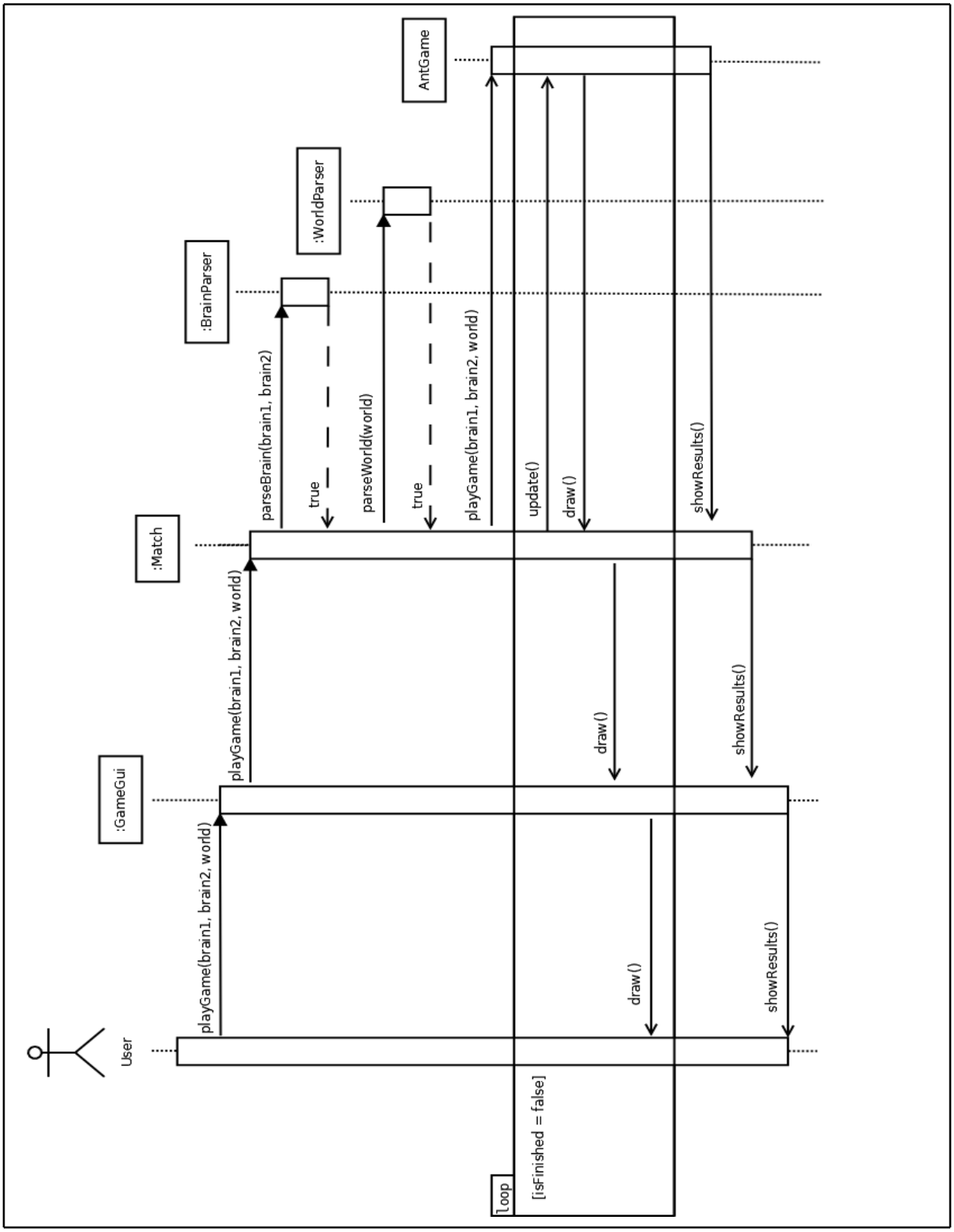
The GameGui class controls the functional properties and visibility state of HTML DOM components in accordance with the state of the application. It has no public methods.

AntWorld

The AntWorld class describes a hexagonal grid of WorldCell objects.

Methods	Description
getAdjacentCell(x,y,dir)	Returns the cell adjacent to the cell at (x,y) in the direction dir.
getAllAdjacentCells(x,y)	Returns a list of all cells adjacent to the cell at (x,y)
getCell(x,y)	Returns the cell at (x,y)

Sequence Diagram


User
:GameGui
:Match
:BrainParser
:WorldParser
AntGame
playGame(brain1, brain2, world)
playGame(brain1, brain2, world)
parseBrain(brain1, brain2)
true
parseWorld(world)
true
playGame(brain1, brain2, world)
update()
draw()
showResults()
draw()
showResults()
draw()
showResults()
loop
[isFinished = false]

Sign Off

Project Manager:

Design Team:

Programming Team:

QA Team:

Analysis Team: