High-Level Design Specification

Software Engineering Group 6

12/3/2012: High-Level Design Specification, v1.0

March 2012 - Second Deliverable

Actions: Configuration Management applied, first check through for finalisation. QA.

Contents **Page no:**

Introduction

This document is a brief overview of how the analysis model is to be translated into a feature-complete design specification. It will provide details of the basic structure of the program, along with information regarding other implementation details like coding style.

In brief, the program will use the Model-View-Controller design pattern as a template for simplifying modular communications. Between the Model and Controller, and between the View and Controller, there will be only one route of communication.

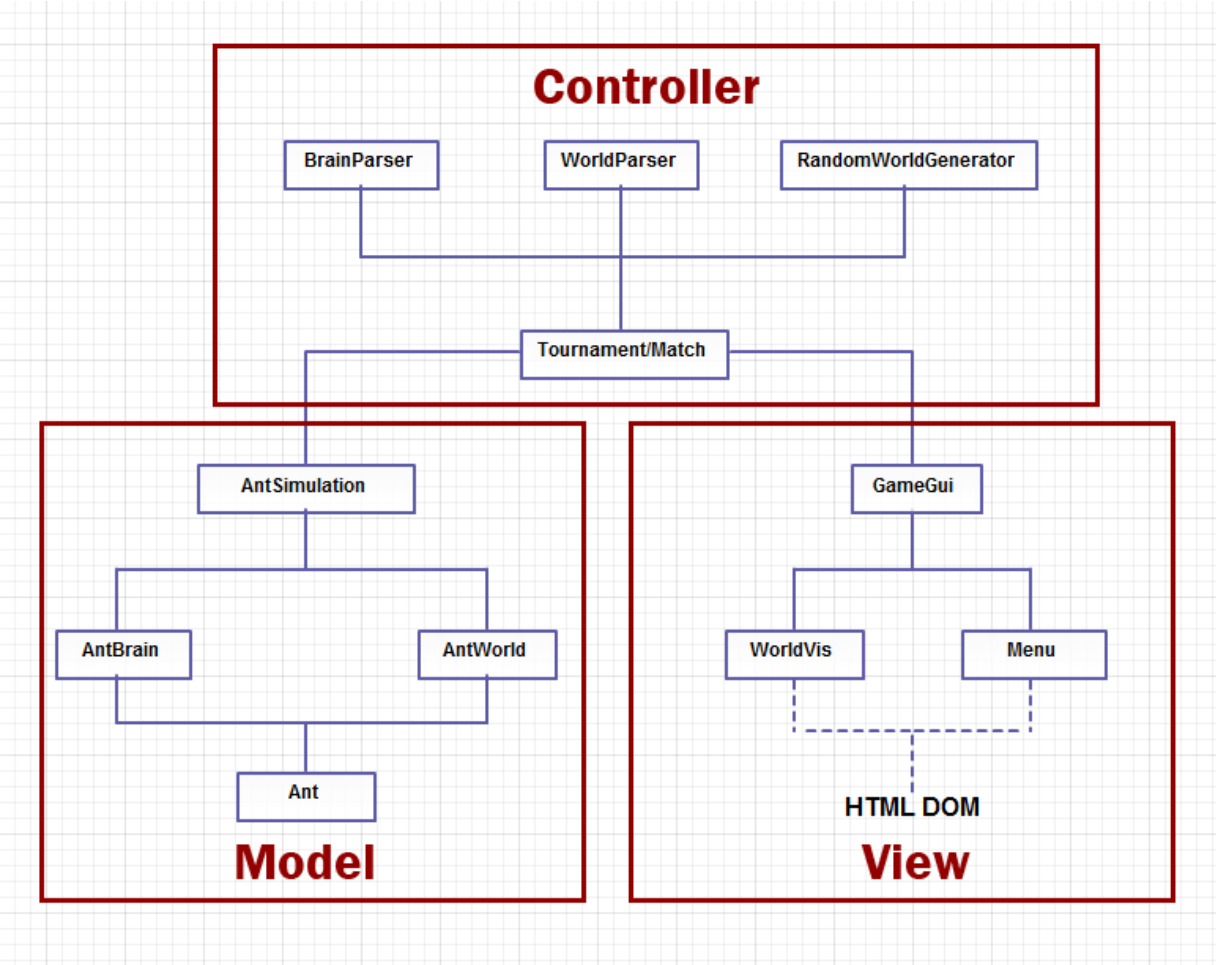
It is important to note that we will be using Javascript as the primary implementation language. Javascript lacks the traditional Class-based object-orientation of languages like Java. Luckily, it has first order functions, closures, dynamic typing, and other features which significantly reduce the complexity of implementing many common patterns which we will use (e.g. dependency injection, the observer pattern, etc). In lieu of a normalised modelling language for these latter features, we will describe the program in terms of classes using UML notation without limiting ourselves to directly mapping these classes to modules or Javascript prototype definitions. A proper description of the way we will utilise JavaScript's more expressive features will be given in the detailed design specification to follow.

Architectural Design

Design Patterns

Model-View-Controller architectural pattern for the program.

Here is a rough illustration of the division of class responsibilities with regard to MVC.



State Pattern for ants.

Rather than traversing a complex series of if/then/else statements when deciding what to do in a given round, an ant should have access to a *state* object which should contain a method that manipulates the ant according to the brain specification with minimum conditional-branching overhead.

Observer Pattern for view updates

The Model part of the application should have hooks for listeners which tell the View and/or the Controller about state changes which require some update in display.

Singleton Pattern for various classes

It makes no sense to have more than one instance of various types of objects. The classes we have identified as such are:

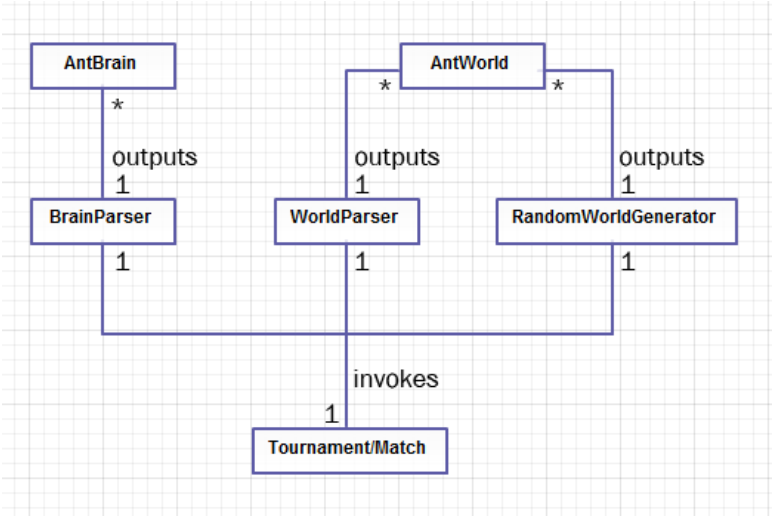
- BrainParser
- WorldParser
- RandomWorldGenerator
- GameGui

Concurrency

Javascript has no facility for concurrency. At best, time-sharing can be accomplished using the `setTimeout` and `setInterval` methods. This will be required when executing lengthy computations (primarily the simulation itself) to avoid locking up the browser and/or DOM.

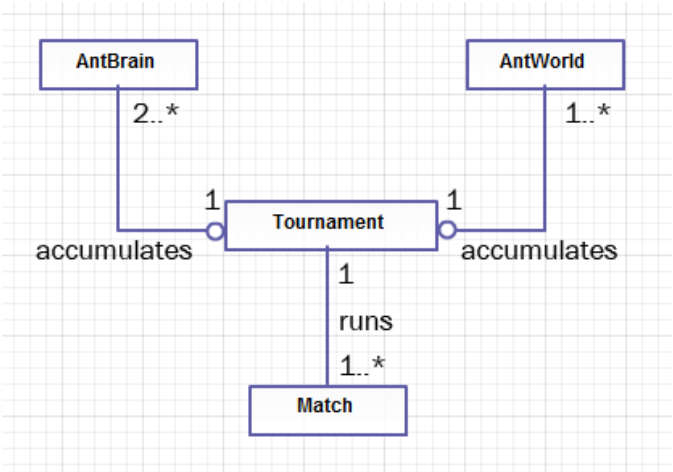
Physical Organisation of the System

Controller



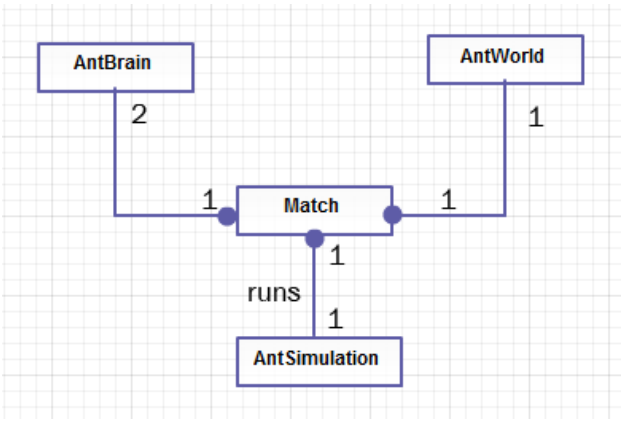
A tournament or a match compiles and arranges game components (according to user actions) using parsers and generators.

Tournament



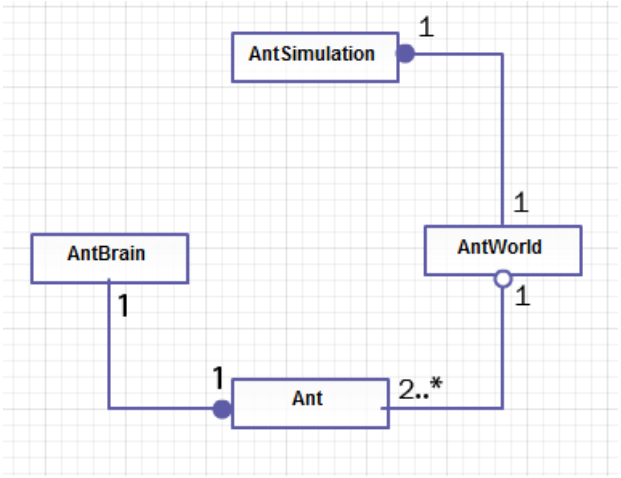
A tournament accumulates brains and worlds, and then runs a series of matches.

Single Match



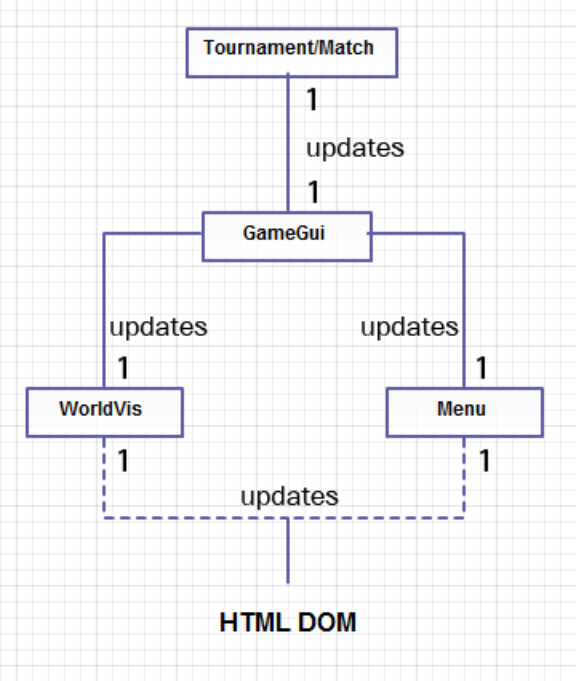
A single match is composed of two brains, one world, and a simulation.

Ants



Each ant has one brain; a world is populated by ants, and all of the ants in the world are iterated over by the simulation in each round.

View



A tournament or match updates the GameGui, which cascades updates down all the way to the DOM.

Javascript Coding Style

There are two methods by which we will maintain coding style.

- We will follow Google's Javascript style guide[1].
- There exist several style-checking tools which are widely used by Javascript developers such as Crockford's JSLint[2], JSHint[3], and various others. We will be using JSHint as it can be incorporated into the build process, giving automatic notification of badly-formatted code.

There exist some cases where Google's style guide and JSHint disagree. JSHint can and will be configured to ignore such cases.

Common Tactical Policies

Memory Management

In most setups, memory management is a serious issue. However, our project is being developed in JavaScript, and being run in a browser. Because of this, memory management is not an issue for our implementation.

Data Management

Data management needs to be applied to ensure that future modifications are as easy to perform as possible. In order to prevent confusion within the file system, a uniform file structure will be applied. The core data files will be stored within the "js" folder, all images will be stored within the "img" folder, all sound will be stored within the "sound" folder and the style sheet files will be stored within the "css" folder. This will prevent confusion within the programming team with regards to file storage locations, and ensure that the entire data structure is kept neat and tidy.

General Error Handling

General Errors will be handled in a standardised way, both for the benefit of testers and the end user in the unlikely event that there are serious errors in the final program. The program will output useful data to the GUI to assist the programming team to find and fix the problem(s). If there is an error in a specific part of the program the GUI would attempt to display a new window display the error message or code, e.g. "Failure in AntWorld" or "Error 999". This is to both help programmers to make bug fixing easier and alert the user if the problem is with their end. A good example of this would be if the end user had not got the correct version of Javascript installed, prompting them to download the appropriate version and therefore prevent the "Bug" being incorrectly reported due to a fault on the User's part.

Syntax Errors in Input Files

Syntax errors are likely within user-generated brain files, and they may cause the application to crash or run incorrectly. To prevent this, the parsers will throw an error if incorrect syntax is detected. In this case, the UI will display an error message which will inform the user as to what the error is, and which line it is on.

Requirements Cross Reference

Requirement in Analysis Document	Requirement in High Level Design
Ant	Ant
AntWorld	AntWorld
AntBrain	AntBrain
WorldCells	Dealt with in AntWorld
Game	AntSimulation
Tournaments	Tournament
AntHill	Death with in AntWorld

RandomWorldGenerator	RandomWorldGenerator
Gui	The elements contained within GameGui, Menu and WorldVis together cover the responsibilities given to “Gui” in the analysis document.
WorldParser	WorldParser

References

[1] http://google-styleguide.googlecode.com/svn/trunk/javascriptguide.xml
[2] http://www.jslint.com/
[3] http://www.jshint.com/