Group 8 – High-Level Design Specification - Software Engineering
Spring 2014

Software Engineering

High-Level Design Specification

74120

105957

108069

108252

109195

University of Sussex

Table of Contents

1 Introduction

The high level design document specifies the overall structure of how the game shall be implemented. By translating the requirements specification, this document will provide the design details of the game's basic structure.

The design pattern used for the game will be the Model-View-Controller (MVC) design pattern, in which the each of these three components are abstracted from each other as much as possible. Java will be the language used to implement the ant game; Java's excellent threading and synchronization capabilities are ideal for this visualised simulation game.

The high level design will be primarily described with aid from system context diagrams. The diagrams, along with supportive text, will give a clear understanding of the relationships between the components, and provide sufficient information to produce the low level design document.

2 Architectural Design

2.1 Coding Style

The 'Google Java' coding style[1] will be used throughout the production of source code. This choice has been made as the Google convention follows a well-known coding practice, as well as being a complete and up to date source for the team to refer to.

2.2 Interaction Models

The design pattern to be used when programming the Ant Game is the Model-View-Controller. The way in which in the MVC pattern works is quite simple and an intuitive choice to make when creating GUIs. Its basic workings are accurately represented in figure 2.2.1 below.

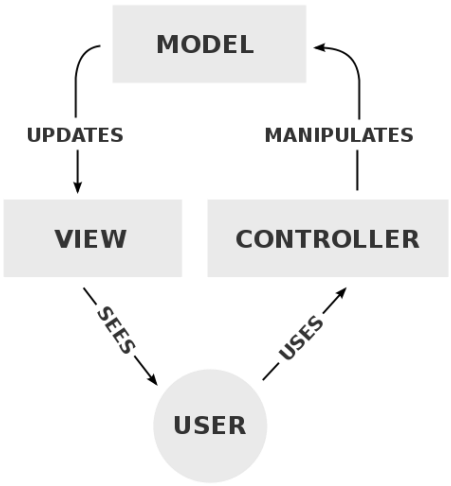

Figure 2.2.1. Model-View-Controller[2]

[1] http://google-styleguide.googlecode.com/svn/trunk/javaguide.html
[2] http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

Now being specific to the ant-game, the overall MVC structure is shown in figure 2.2.2 below.

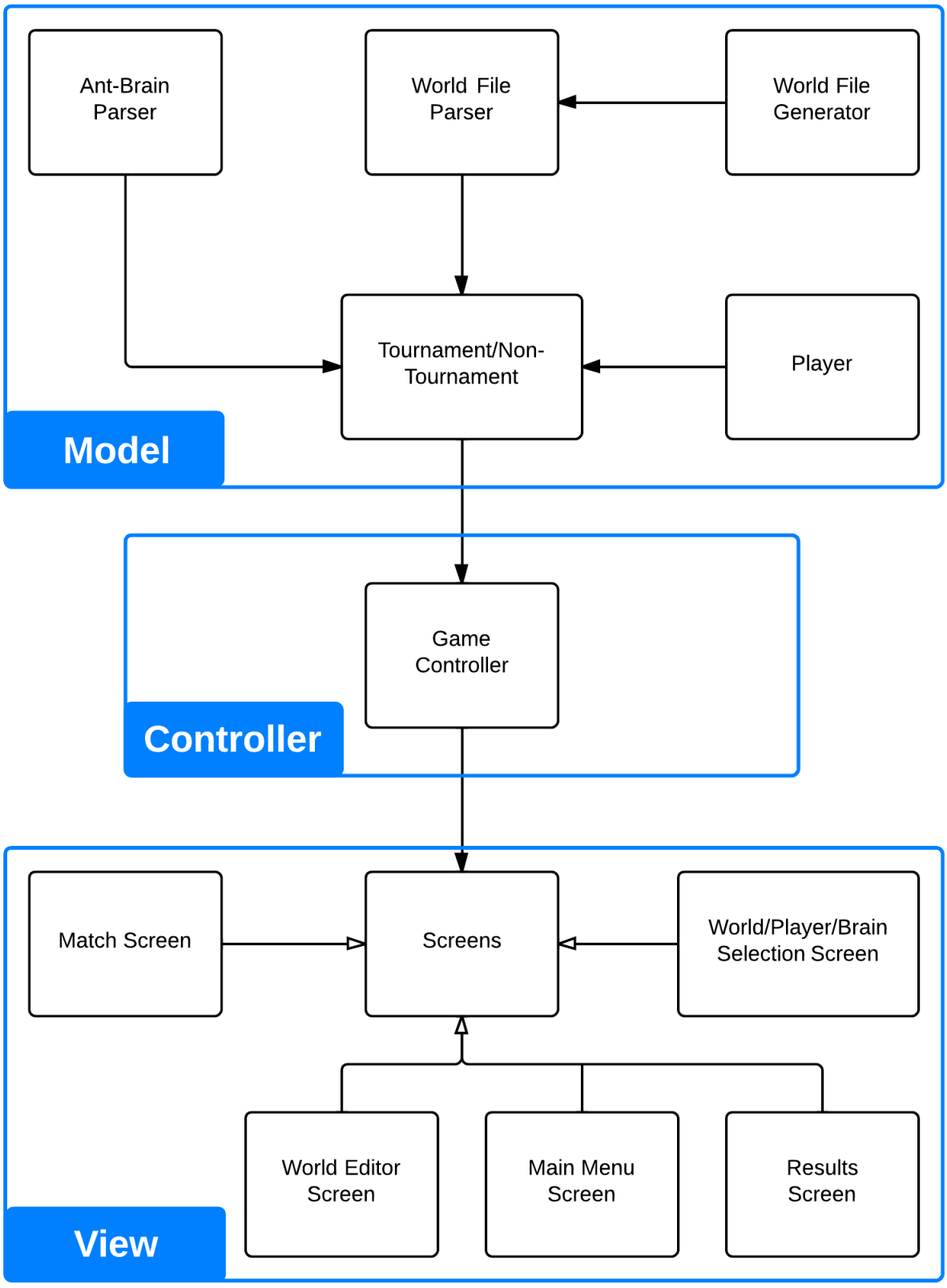

Figure 2.2.2. The overall MVC design pattern for the ant game

Some screens need to have buttons that interact with the underlying model, for this to happen each screen will have access to the game controller in which the game can invoke actions on the model, and finally the screens can update their display based on the game controller retrieves. This MVC implementation is both active and passive; it is passive because in some of the screens in the program (e.g. world creation screen) there is a need to request updates from the model, and at the same time it is active, as some other screens (e.g. match panel) are notified of any ongoing updates happening in the model.

Taking into account the structure of the MVC design pattern shown in figure 2.2.2, a diagrammatic representation of the overall game structure can be seen in figure 2.3.3 below.

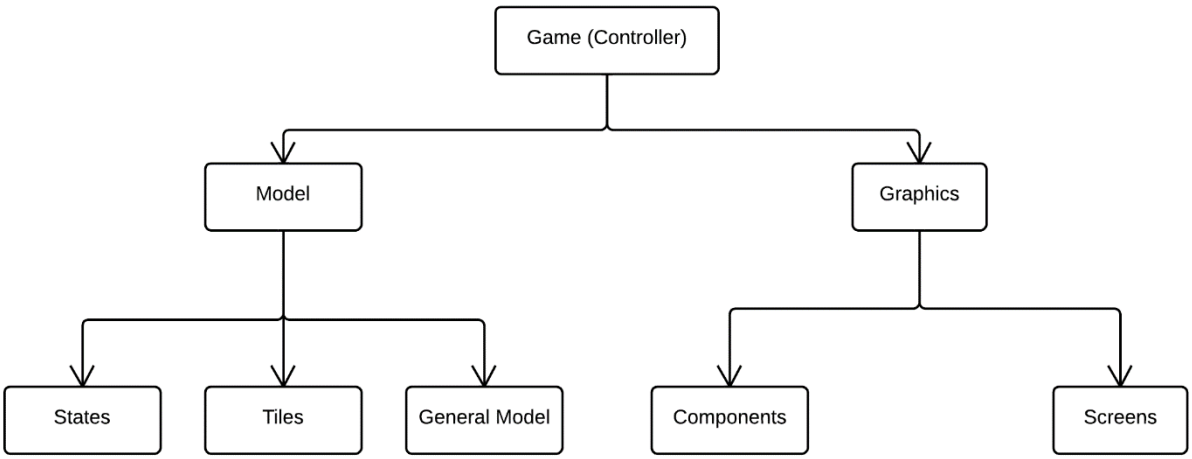

Figure 2.2.3. The overall structure of the ant game

2.3 System Context

In this section, the system context for each component of the ant-game is presented. For a single match to run, it requires each player to upload an ant-brain, a total of two players is needed and a single world is needed to play the match on; this is shown in figure 2.3.1 below.

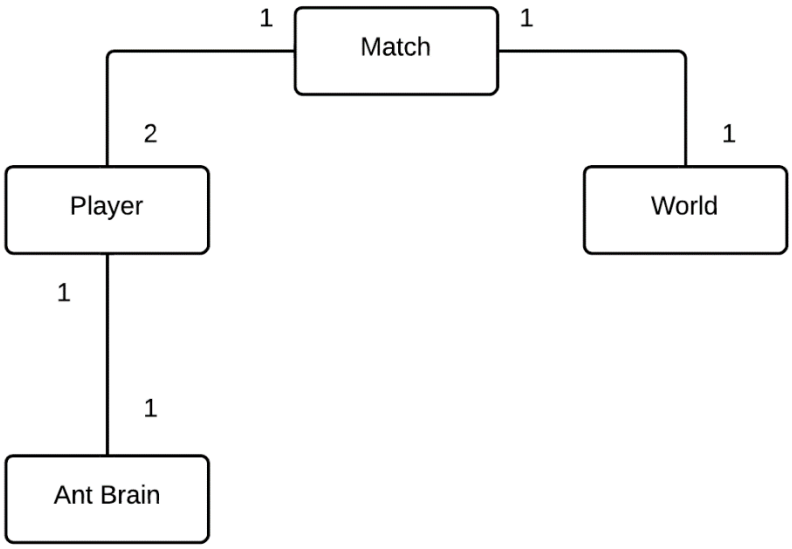

Figure 2.3.1. The system context for a single match

A tournament is simply a collection of generated matches, its system context is shown in figure 2.3.2 below.

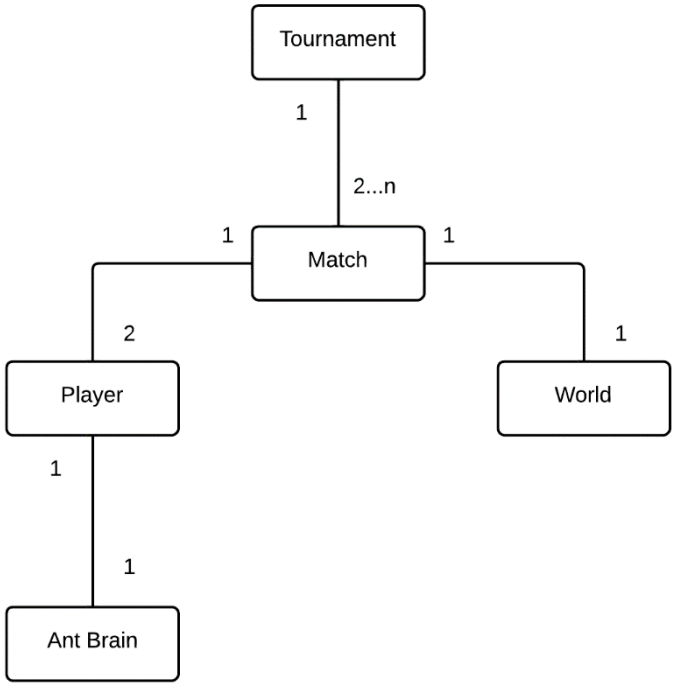

Figure 2.3.2. The system context for a tournament

Taking into account the minimum world dimensions is 30x30, the world component's system context is shown in figure 2.3.3 below.

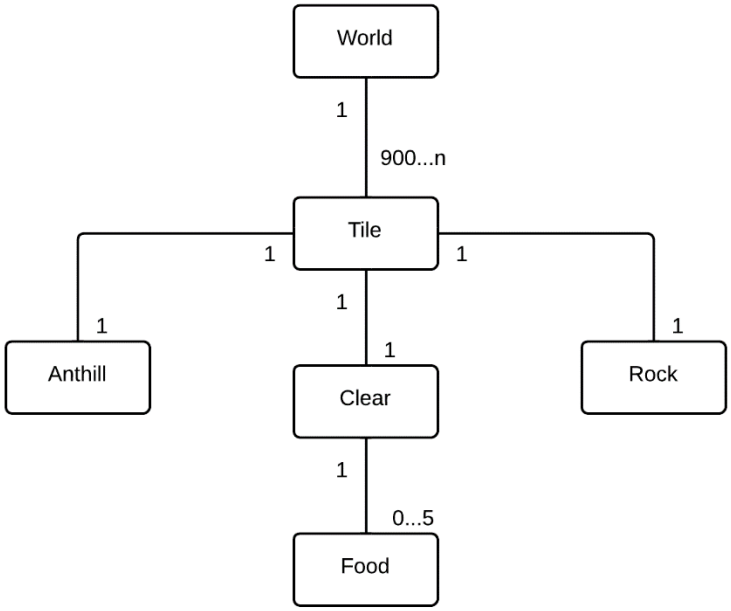

Figure 2.3.3. The system context for an ant-world

There is no need for a diagram to represent the ant-brain's system context, as an ant-brain is simply made up of multiple states.

3 Common Tactical Policies

3.1 Memory Management

One of the main advantages of using Java, is not having to have to manually place memory management mechanisms into our code. This is because Java performs automatic garbage collection and hence it assures the JVM's memory stack always has available memory (unless there is a bug in the code, however testing will ensure such bugs do not exist).

3.2 Overall Error and Input File Error Handling

General errors will be handled using Java's standard error handling methods, i.e., using and handling exceptions with *try*, *catch* and *throws*. Thorough testing will help to prevent fatal errors from being present in the final product, and in turn this will help to ensure the safety of the user's system. However in the case of a run-time error being present, the error will be handled in either the Model/Controller part of the ant-game (depending on where it occurs) and afterwards will be presented to the user through the use of the View (GUI) component.

Syntax errors in the files input by the user, such as the ant-brain or a world file, will be handled in the Model part of the MVC, by the respective parser. If a syntax error is present in the input, the user will be presented with an appropriate error message via the View.

3.3 Concurrency

Threading will be achieved using Java's standard Thread library. The game will consist of two main threads. When a match is running, the first thread will periodically update the Model, and the second thread will periodically update the display, doing this will allow the players to view the current state of the Model and prevent the GUI from becoming unresponsive while the simulation is running.

4 Index