Software Engineering

Detailed Design Specification

74120

105957

108069

108252

109195

University of Sussex

Table of Contents

1 Introduction 3
2 Overview 4
3 Model 5
4 View 6
4.1 Overview 6
4.1.1 Graphical Components Outline 6
4.1.2 Screen Panels Outline 8
4.2 Components 12
4.3 View Screens 13
5 Controller 15
5.1 Threading 17
6 Index 18

1 Introduction

In the high level design document, the overall structure of the program was presented, along with the design pattern to be adopted when implementing the structure. It is now appropriate to take a closer look at the individual components of the game.

As Java is an object oriented programming language, the detailed design will be primarily in the form of object and class diagrams using UML. The diagrams, along with the supportive text, hopes to provide sufficient detail to allow the programmers implement the code quickly and efficiently.

The design decision was made in the high level design document that the best approach for this project is to use the MVC (Model-View-Controller) design pattern. As the understanding of the View and Controller are dependent on the workings of the Model, the document will describe the Model components of the program first, followed by the View, and finally, the Controller.

The ordering in which the components are described is an ideal ordering when it comes to developing the game. The main reason for this, is that if any significant changes need to occur in the Model, parts of the View may become redundant, or need further development. Therefore, developing the View and Controller after the Model should help to save effort which could have be focused elsewhere.

It is important to note that various changes have occurred between the high level and low level design specification documents. Although there are not many alterations, two crucial changes are made where concurrency is concerned, without the changes that have been described in section 5.1, the graphical user interface will not function correctly.

2 Overview

Figure 2.1 below represents the architectural overview of the program to be implemented.

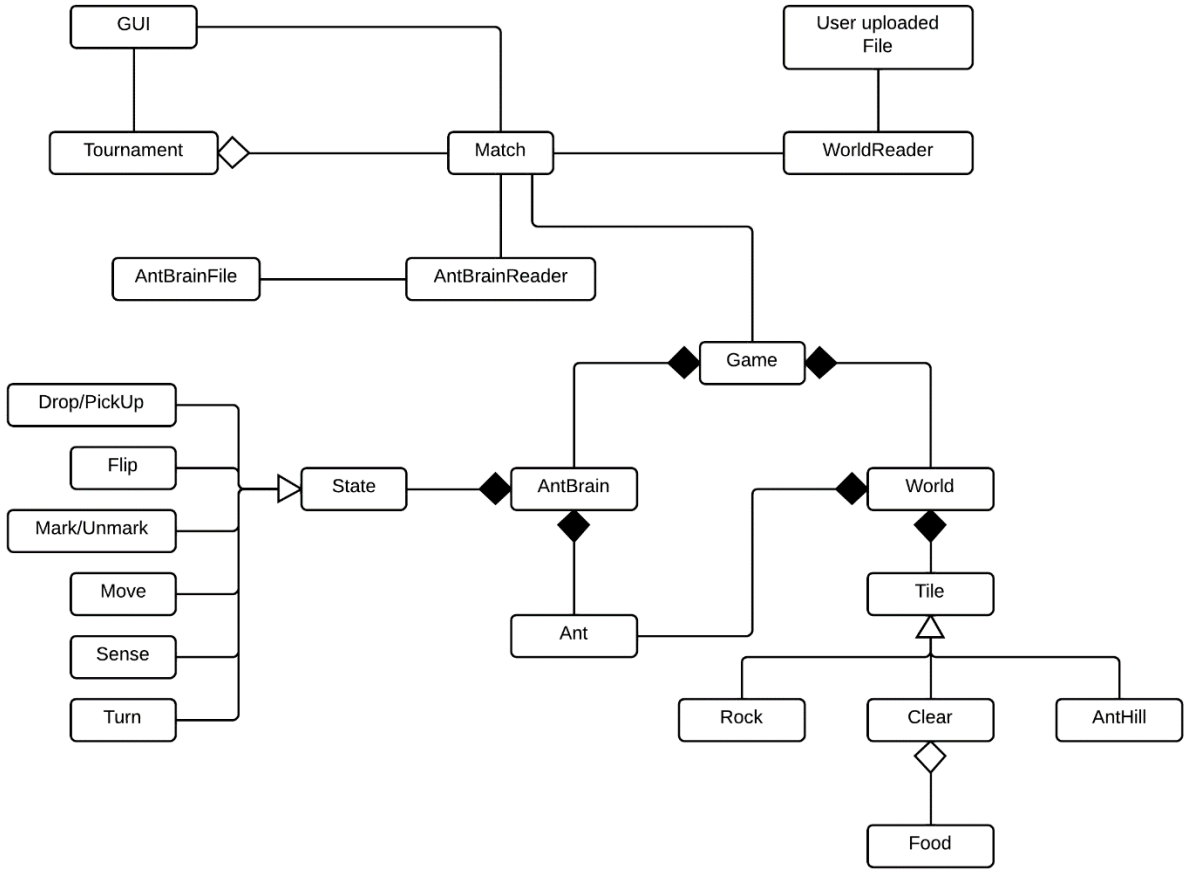

Figure 2.1. Architectural overview of the program

Taking into account the architectural overview presented in figure 2.1 above, it is clear that our program is organised into three fairly distinct sections of code. Each section represents a component from the design pattern we are following, MVC. The model is represented by all classes below 'Game', the view is represented by all classes above 'Game'. The controller is represented solely by 'Game' and the diagram shows how it links both other parts of the game together.

The distribution of the components is as follows:

- **Model**:
  - AntBrain - A list of states (Flip, Move, Sense, etc.).
  - World - A collection of tiles (Rock, Clear or Anthill tile).
  - Ant - An object generated in the game, which follows a given brain.
  - WorldReader - Parses and returns a World.
  - AntBrainReader - Parses and returns an AntBrain.
  - Tournament - A collection of Matches.
- **View**:
  - GUI - A collection of screens, which allows the players to use the model.
- **Controller**:
  - Game - A handler that passes information from the View and the Model and vice versa.

3 Model

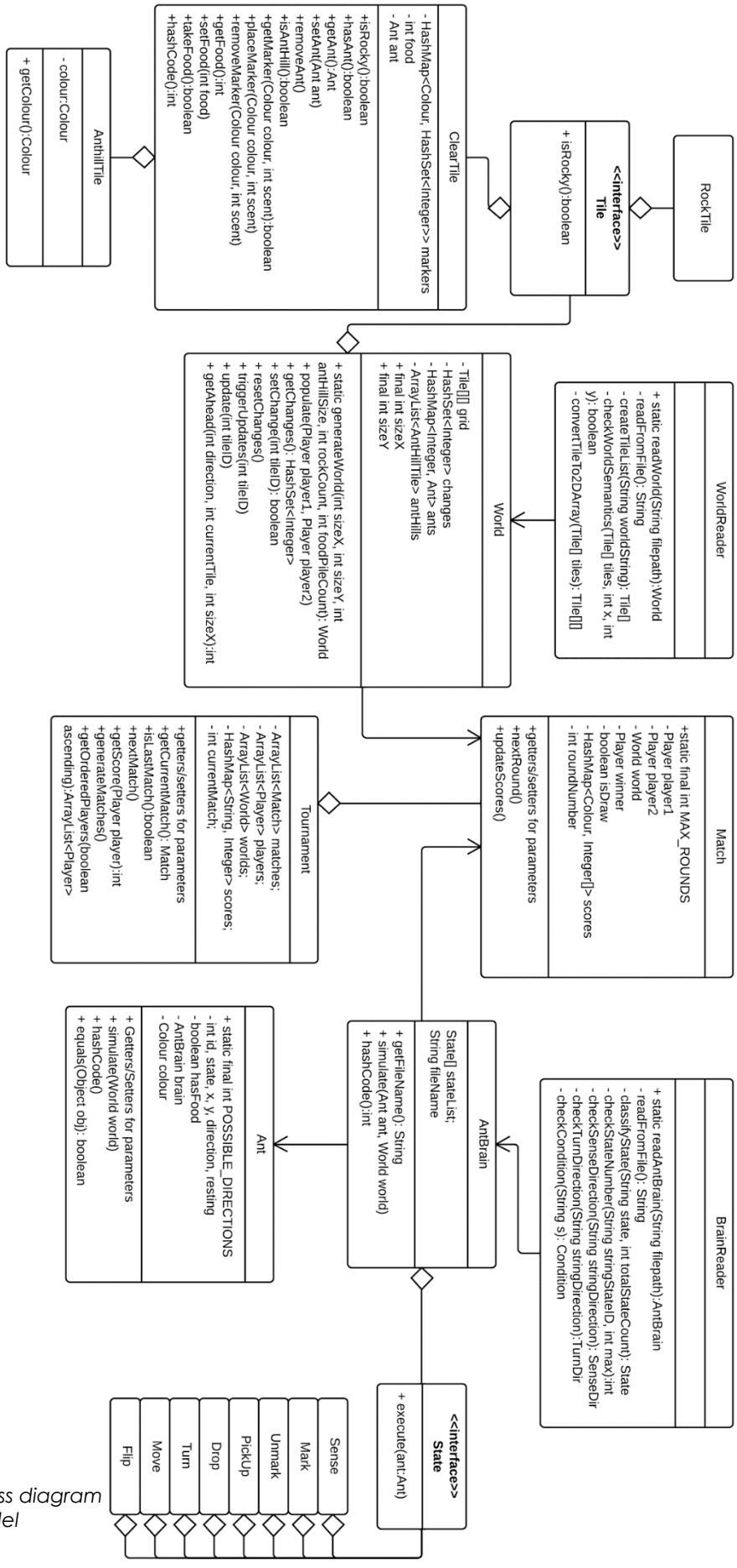

Figure 3.1. The class diagram of the overall Model

4 View

4.1 Overview

The GUI will be implemented with Java's native libraries, AWT and Swing. Using Swing (by default) does not provide the look and feel felt appropriate for the ant game, so a number of classes will be created to provide the desired appearance. These helper components will allow the ant game to display custom graphics, along with custom looking buttons with hover effects.

4.1.1 Graphical Components Outline

Component Name	Outline
ImagePanel	• Takes an image as its parameter. • A JPanel that paints an image to its background. • Its size should also match the image's width and height. • The JPanel should also be transparent, to allow PNGs to be displayed correctly.
DualImagePanel	• Takes two images as its parameters. • A JPanel that paints an image to its background. • Its size should also match the image's width and height. • The JPanel should be transparent, to allow PNGs to be displayed correctly. • Has a publicly available method to display the first image. • Has a publicly available method to display the first second.
ImageButton	• Takes two images as its parameters, the first is the standard image, the second is the hover image. • A JPanel that paints an image to its background. • Its size should also match the image's width and height. • The JPanel should be transparent, to allow PNGs to be displayed correctly. • When hovered over, the second image should be displayed, along with the hand mouse pointer. • When exited, the first image should be displayed, and default mouse pointer should be returned. • An abstract method that allows the actions taken when a mouse is clicked to be implemented.
Hexagon	• This will extend an AWT Polygon, and its shape will represent a regular hexagon. • Six points of the hexagon will be set in relation to each other, and the center point of the hexagon. • The Hexagon's size will be resizable, this will allow for variable zoom levels of the HexGrid to be achieved. • The Hexagon's colour will be variable; both the fill and outline colour can be set.

	• The Hexagon's stroke width will also be variable.
HexGrid	• This component will be made up of Hexagon objects stored in a 2D array, to make up a grid. • The HexGrid will be a JPanel displaying the grid described previously. • The HexGrid will be dynamically resizable, by resizing all hexagons in the grid individually. This can be done incrementally or by passing in an absolute value. • It will be possible to swap out any hexagon in the grid for a new one.

Table 1 Graphical Components Outline

4.1.2 Screen Panels Outline

The main window will be a JFrame and will hold a number of screens (being represented as JPanels). When a different screen is to be displayed to the players, the Controller will be told to switch the display from the current screen to the new screen. When a screen switched in by the Controller, the screen should be updated. The action of 'updating' the screen varies from screen to screen, it could mean that the screen is reset back to its default values, or it could mean that some values for the screen should be input (for example when displaying match results).

Screen Name	Outline
MainMenuPanel	• A JPanel that displays the title (as an ImagePanel), and two buttons (as ImageButtons). • When the tournament button is clicked, it requests the controller to switch to the TournamnetSelectionPanel. • When the single match button is clicked, it requests the controller to switch to the SingleMatchPlayerPanel. Actions when update() is called: • Set the current match in Game to null. • Set the current tournament in Game to null.
MatchPanel	• A JPanel that displays a HexGrid inside a JScrollPane. Along with stats about the two players, zoom in and out buttons, increase and decrease speed buttons, and a pause/resume button. Actions when update() is called: • Reset the game speed. • Reset the zoom level. • Set the nickname for player 1 • Set the nickname for player 2
MatchResultsPanel	• A JPanel that displays the results of the match. • Both players will have their stats displayed, using JLabels. The stats will include the number of food particles collected, ants alive and dead at the end of the game. • The screen will be decorated by ImagePanels displaying informative text, including the title and the winner declaration. • There will also be a main menu ImageButton to take you back to the menu, if it was a single match, or a next ImageButton to proceed to the next match. • The ImageButton specified will request the controller to switch the screen to the implied follow-up screen. Actions when update() is called: • Updates player names

	• Updates winner • Updates stats for each player

SingleMatchPlayerPanel	• A JPanel that retrieves all of the inputs from the user for use within the game core.
	• The screen will display two JPanels, one for each player. Each of these individual panels will have the following:
	  o A player JLabel, which will always stay as 'Player 1' and 'Player 2', to show the separate player inputs.
	  o For both players there will be the following:
	  o A nickname label, a place to input a nickname, and a DualImagePanel with tick/cross images loaded in, to indicate validity of the name.
	  o An ant-brain label, a FileChooser to upload an ant-brain file, and a DualImagePanel with tick/cross images loaded in, to indicate validity of the brain.
	• The screen will also have two ImageButtons to go back to the main menu, and to proceed. These buttons will request the controller to switch the screen either to the main menu, or to the SingleMatchWorldPanel screen, respectively.
	
	Actions when update() is called:
	• None

SingleMatchWorldPanel	• A JPanel that prepares and displays ant-worlds for the single match to be played.
	• It will have a HexGrid inside a JScrollPane for display of the map.
	• This panel will have three ImageButtons:
	  o The first will be for generating and displaying a random valid world, repeated clicks will repeatedly generate worlds and display them.
	  o The second will open a JFileChooser to upload an ant-world to the game. It will also display it.
	  o The last ImageButton will request the controller to switch screens to the WorldEditorPanel, so a world can be created for the game.
	• There is also a DualImagePanel with a JLabel to indicate whether the ant-world supplied is valid. The DualImagePanel holds BufferedImages of tick/cross to indicate this.
	
	Actions when update() is called:
	• The HexGrid is redrawn to whatever the newest map supplied to the game is.

WorldEditorPanel	• A JPanel that has informative ImagePanels guiding the user into inputting values into the world generator.
	  o The first ImagePanel will be for the title of the page
	  o The next four are for the inputs:
	  ▪ Number of rocks to be in the world
	  ▪ Amount of food to be in the world
	  ▪ Size of anthills on the map
	  ▪ World dimensions
	  o The first three in the above have a '-' ImageButton, and a '+' ImageButton to alter the amount to be used. These buttons will update a JLabel to show what input the user has chosen, from: low, medium, high.
	  o The world dimensions has two input fields to allow the user to specify a size in this format: columns x rows. This will have a validation DualImageButton to show whether the input sizes given are suitable.
	  ▪ A maximum size will be 300 x 300.
	  ▪ A minimum size will be 30 x 30.
	• There will also be two ImageButtons on the bottom of the page,
	  o One will go back a page, to wherever this screen was accessed from
	  o One will create the world and take you back to where this screen was accessed from but also supply the world generated.
	
	Actions when update() is called:
	• Resets selections back to the default values:
	  o 'Medium' for the amount of food, ant hill size, and number of rocks.
	  o 150 x 150 for the world dimensions.

TournamentResultsPanel	• A JPanel that has a ImagePanel on the top, for the title of the screen. • There will also be a large JScrollPane to hold a JPanel for each player. • Each of the JPanels for a player, will have their name and their scores. • The JScrollPane will put the players in the order of their placement in the tournament. • There will also be an ImageButton on the bottom to return to the main menu. Actions when update() is called: • Gets the players in order of placement

	• Adds the players to the JScrollPane • Removes all previous tournament results from the list.
TournamentSelectionPanel	• A JPanel that has multiple input points on the screen, a JScrollPane to hold players, ImagePanels for labels, and ImageButtons for navigation between screens. • A JPanel at the top of the screen will hold the title of the screen. • There will be a JLabel and a JTextField next to it, to notify the user where to put in the nickname for a player to be added. The JTextField will have a DualImagePanel next to it to indicate the validity of the currently typed name. • There will be another JLabel and a JFileChooser next to it, to allow the user to upload an ant-brain for the player. The JFileChooser will have a DualImagePanel next to it, to indicate the validity of the ant-brain that was chosen in the file chooser. • There will be a DualImageButton to add the player into the game, and to create a JPanel for the player in the JScrollPane. When clicked, this button should check both the name and the ant-brain that have been selected are valid, and if so, upload them to the game core. • The JScrollPane will hold a representation of each player ready to be in the game, each as a JPanel. The player JPanels will have JLabels with the player name, and brain name. There will also be a DualImageButton to remove the player from the game core and the JScrollPane. • A JLabel and a JTextField will also be present to allow the user to dictate how many ant worlds should be generated for use in the tournament. • Two DualImageButtons will be present to navigate back a page, and to start the tournament, Actions when update() is called: • Players list is reset to empty.

Table 2 Screen Panels Outline

4.2 Components

Presented below in figure 4.2.1 is the components package, this part of the View provides:

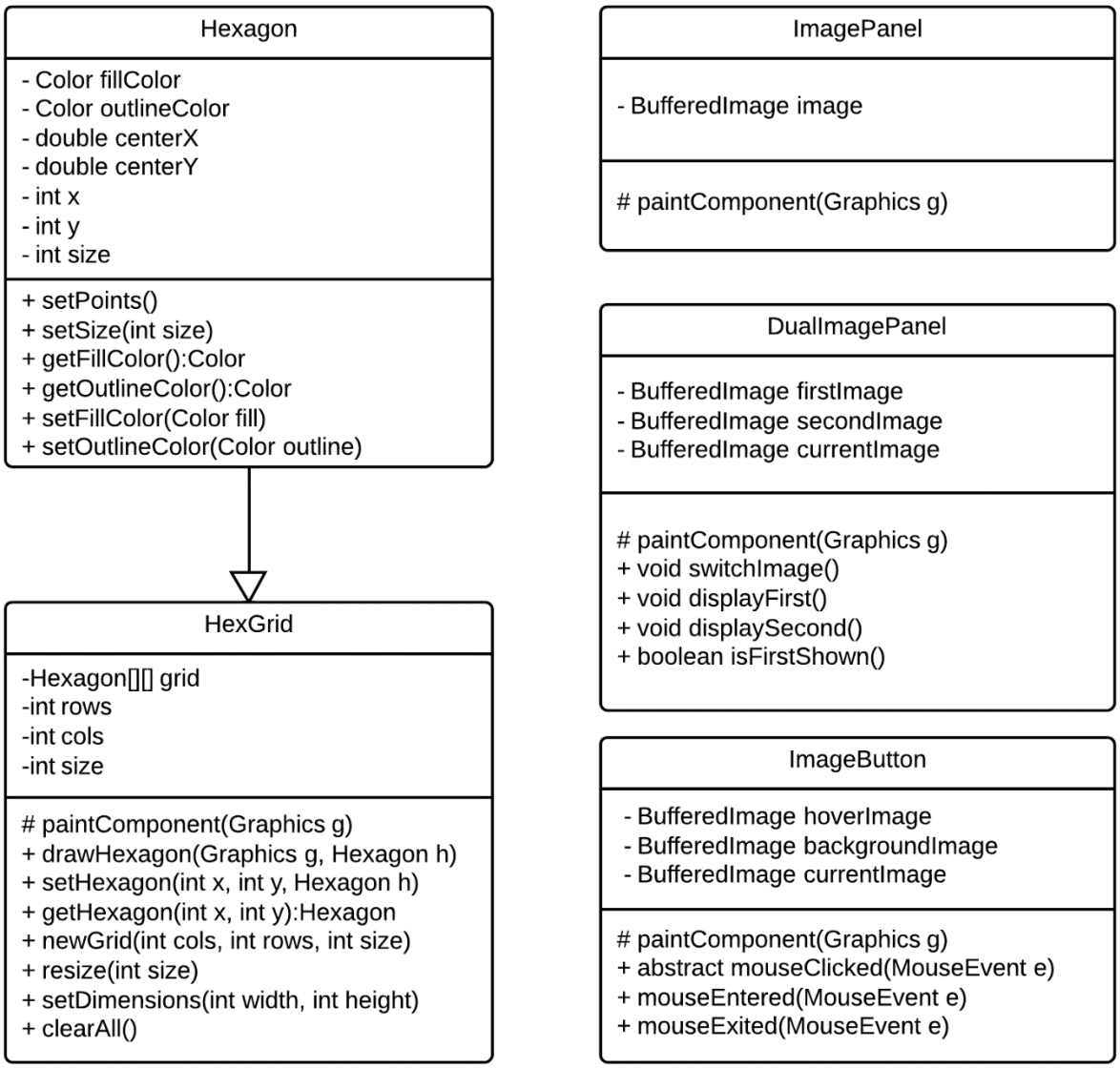

Figure 4.2.1. View - class diagram of the components package

By analysing the diagram shown in figure 4.2.1 above, it is clear that most classes are unrelated (as in, they don't have a relationship) to other classes; an exception being the generalization relations between the Hexagon and HexGrid classes and the ImagePanel and DualImagePanel classes. This lack of inter-relations between classes can be explained by the fact that most of the Controller classes are use used to manipulate the Model, which in turn updates the View.

Also, as the Ant-Game is by definition a game that is not very interactive, i.e. doesn't require a lot of user interaction, the Controller part of the program is, as a consequence, not very complex.

It is logical, to now illustrate the View (Screens package) part of the model. It simply consists of a number of screens that can switch between each other in order to present the user with the intended view.

4.3 View Screens

The class diagram for the screens used View is shown in figure 4.3.1 below:

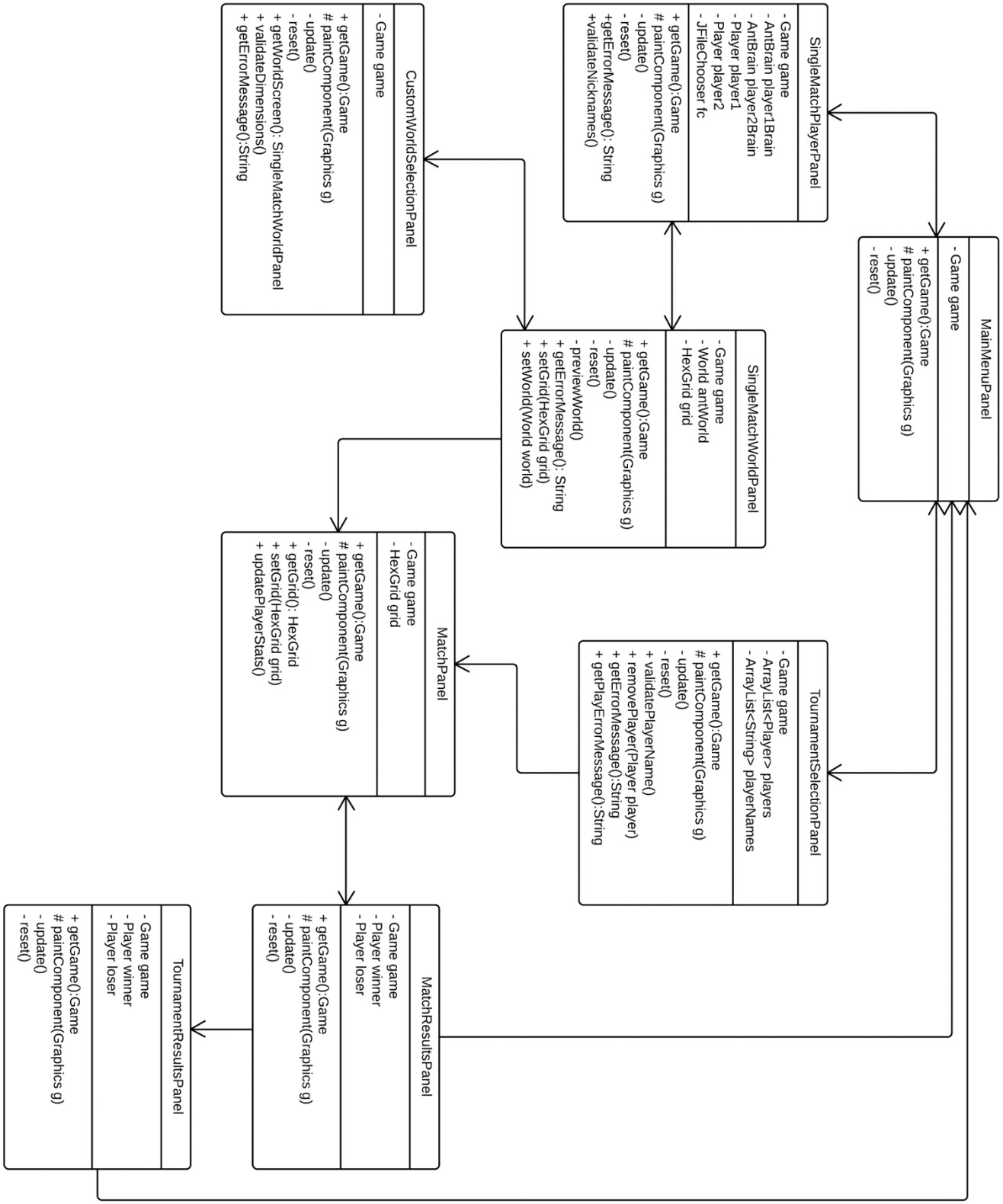

Figure 4.3.1. View - class diagrams for the screens package

A quick analysis of the diagram, shown in figure 4.3.1 above, reveals that most association relations between classes are multi-directional. The View is implemented this way in order to allow the players to go back and forth in most menus. This allows the players to be visualizing a certain screen, and if they choose to do so, they are permitted to go back to the previously presented screen, through the use of a "Back" button. However, not all screens allow the user to go back for logistic reasons.

It can also be seen here that the MainMenuPanel is a hub, many of the screens will direct into it. Being a game, this makes perfect sense, because it is where all interaction from the user will start, regardless of whether they want to play a tournament or non-tournament game. It will also be where interaction ends; once the results of a game have been shown to the user, either for a tournament or non-tournament match, they will be taken back to the main menu, essentially ending that cycle of user interaction. If they choose to play again, the user will have passed through the main menu to do so.

5 Controller

The controller is made up of a single class, Game.java. The UML diagram for the controller is as follows:

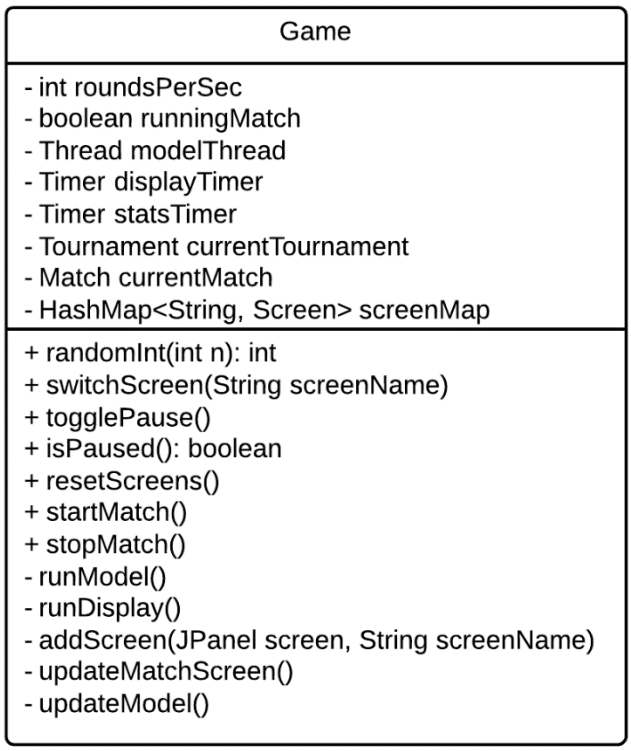

Figure 5.1: Controller - Class diagram for Game

Figure 5.1 above, shows how the Game class is structured. Because the ant game is a simulation, the controller part of the MVC design pattern we are going to follow only needs to be simple.

The Game will have straightforward methods such as switchScreen to quickly move to different parts of the game. The switchScreen method will be an essential part of making the game run smoothly, because the screens will all be created and ready to use upon loading the game, and only put on display when needed. This saves the need to create a page every time we want to access it, and instead, when we want to access a page, we will reset any values on it that could have changed, essentially re-initialising it (this will only be done when it makes sense, if returning to the SingleMatchPlayerPanel from the SingleMatchWorldPanel, it wouldn't make sense to erase the players that had been uploaded to the game.).

You can see our implementation plans to have ways of manipulating how fast the game runs. It is pushed from the View, through a GUI element, into the controller, roundsPerSec, which will then update how many times the model plays a round per second. This change will then be seen in the View, the ants should be moving faster and stats should be changing quicker.

Method Name	Method Function
randomInt(int n): int	Generates a pseudo-random number between 0 (inclusive) and n (exclusive).
switchScreen(String panelName)	Switch the screen that's being displayed to the specified screen.
addScreen(JPanel screen, String name)	Adds the screen to the game so that it can be swapped to when needed. Adds a mapping between a screen and its name
startMatch()	Start running the match that is currently loaded in. Start updating the HexGrid to display the match.
stopMatch()	Stop the current match.
runModel()	Runs the main game loop for the model.
runDisplay()	The main game loop for the graphics. Starts updating the hexagon grid to display the current state of the current match.
updateMatchScreen()	Updates the hexagon grid to be the current state of the current match.
updateModel()	Progresses the model onto the next round.
setTileColor(Hexagon hexagon, Tile tile)	Modifies the given Hexagon relative to the given normal tile to represent data about the tile such as is it a RockTile or AntHillTile or is there food in the tile.
togglePause()	Pauses the model updates if running and saves the current speed. If the game is not running, it sets the games speed to the previous speed.
resetScreens()	Calls the reset method on each screen.
isPaused()	Tells whether or not the game isn't updating the model.

Table 3 Method names with their corresponding functions

5.1 Threading

Threading is a necessity as there are three main actions being performed simultaneously, all of which vary in speed. It is also essential to ensure that the program is always responsive to user interaction, no matter what it's doing. However, it has been determined that altering AWT and Swing components from a standard Thread is not reliable! Instead, **the design will be changed to use Swing Timers instead of Java's Thread libraries when altering AWT and Swing components**, these are ideal as they safely executes on the event-dispatch thread.

After closer analysis, it was decided that the player statistics do not need to be updated as fast as the hexagon grid; as a result of this, **there are now three main simultaneous actions**. The first action updates the model, the second updates the grid, and **the third has been added to update the player's statistics during a match.**

Each of these actions is carried out in a specific Thread/Swing Timer:

1. The first action is in charge of periodically updating the model, the speed of this will vary depending on the speed that the players decide upon, but will be roughly in the range of once a second to 30,000 times a second. This action will be carried out using Threads provided in Java's standard Thread library.
2. The second action is carried out by a Swing Timer, which is simply in charge of updating the Match Panel's hexagon grid periodically. This timer is only required to work at a speed that guarantees the screen to be updated at 30 to 60 times per second.
3. The third action is also carried out by a Swing Timer. This Timer is responsible for keeping the game stats displayed to the players up-to-date for the current match in play. The speed that this should run at is around once per second.

6 Index

A

Architectural Overview 4

C

Components 12
Components package class diagram 12
Controller 4, 17
Controller - Class diagram for Game 17

G

Graphical Components Outline 6

I

Introduction 3

M

Method information 18
Model 4
Model-View-Controller 3

S

Screen Package class diagram 14
Screen Panels Outline 8

T

Threading 19

V

View 4
View Screens 14