Coursework Report

Sam Serrels
40082367@napier.ac.uk
Edinburgh Napier University
Software Development 3 (SET09101)

1 Introduction

The aim of this project was to implemented a 2D game in Java, using specific Design patterns. The game consists of spaceships moving randomly around on a grid, when the player ships moves into the same square as two enemy ships the game is over. IF the player is in an ”aggressive” mode then it takes three ships to kill the player. This project used a web page as the interface for the game, with the game logic being handed by a Java server. The web page connects via the network to the server, and the server can support multiple clients simultaneously.

2 Threads

Threads During a turn, the enemy ships choose a location to move to, this is done in parallel as this does not require modifying any data other than the ship itself. A reference to the player ship is passed to each thread, so the ship can also check for combat. This is all handled by threads, each thread process the move and combat functions, then passes the resulting data into the current gamesate command. Once all the threads have finished, the players health is checked to see if the game is over and a new enemy ship may be created.

3 Design Patterns

Command A ”Gamerstate” object is used to store all the procedures called each turn. This consists of any moves any ships make, and if any ships are created or deleted. This is enough information to successfully ”Undo” from any sate back to any previous state. A ”Redo” function is not implemented as this would require substantial additional logic to deal with ship creation and as the ships move randomly, a repeatable random method would have to be implemented.

Factory When a Gamestate is executed, if it requires a new ship to be created, the type is passed as a text string to a factory function. This function creates and returns the a new enemy ship of the requested type.

Strategy There are two uses of the strategy pattern in this project, one for ship movement and one for ship combat. The Ship base class has a reference to a strategy of type ”MoveMethod” and ”CombatMode”, these are interfaces for each strategy. The ship has methods to swap the strategy in use, this is used on the Player as it swaps between aggressive and passive combat modes. All ships move in the same manor so only one movement strategy is implemented, however the code could easily support more.

4 Networking and Webgui

Websockets The main java project runs as a server, with no user interface. When a client connects via the network, a new game is created for them. This exists in memory as long as the client is connected. The client can issue commands to the server such as ”Turn-Aggressive”, ”Turn-Passive”, ”Undo” and ”Restart”. The networking is handled by the websockets protocol which is supported natively in most modern browsers and has a multitude of available libraries for most platforms and languages. Websockets simplify the task of networking down to a few function calls, such as connect, disconnect, and message and error handlers. Messages are sent as received as plain text, so parsing is achieved with a simple switch statement.

Javascript web interface The interface for this games is accessed from a web browser, the code is written in Javascript and does not require any additional plugins or modifications to the browser. As Websockets are now implemented as a standard using them from within JavaScript is even easier than from in Java. The JavaScript code handles all the animations and button states, the server sends a ”State” message to the browser when the game state changes. This contains a JSON encoded list of all the ships and their position, the web page uses this along with a stored previous state to interpolate the ship positions and animate them.

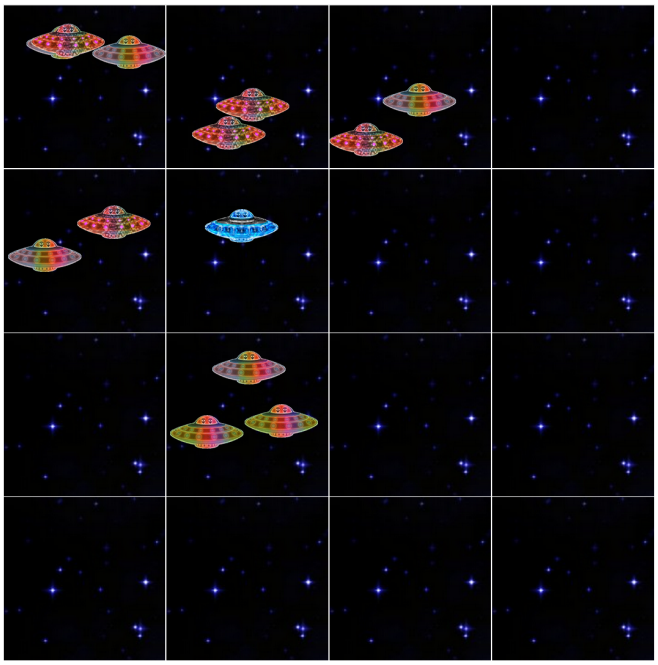

Figure 1: *Screenshot of game board*

Figure 2: *screenshot of full interface*

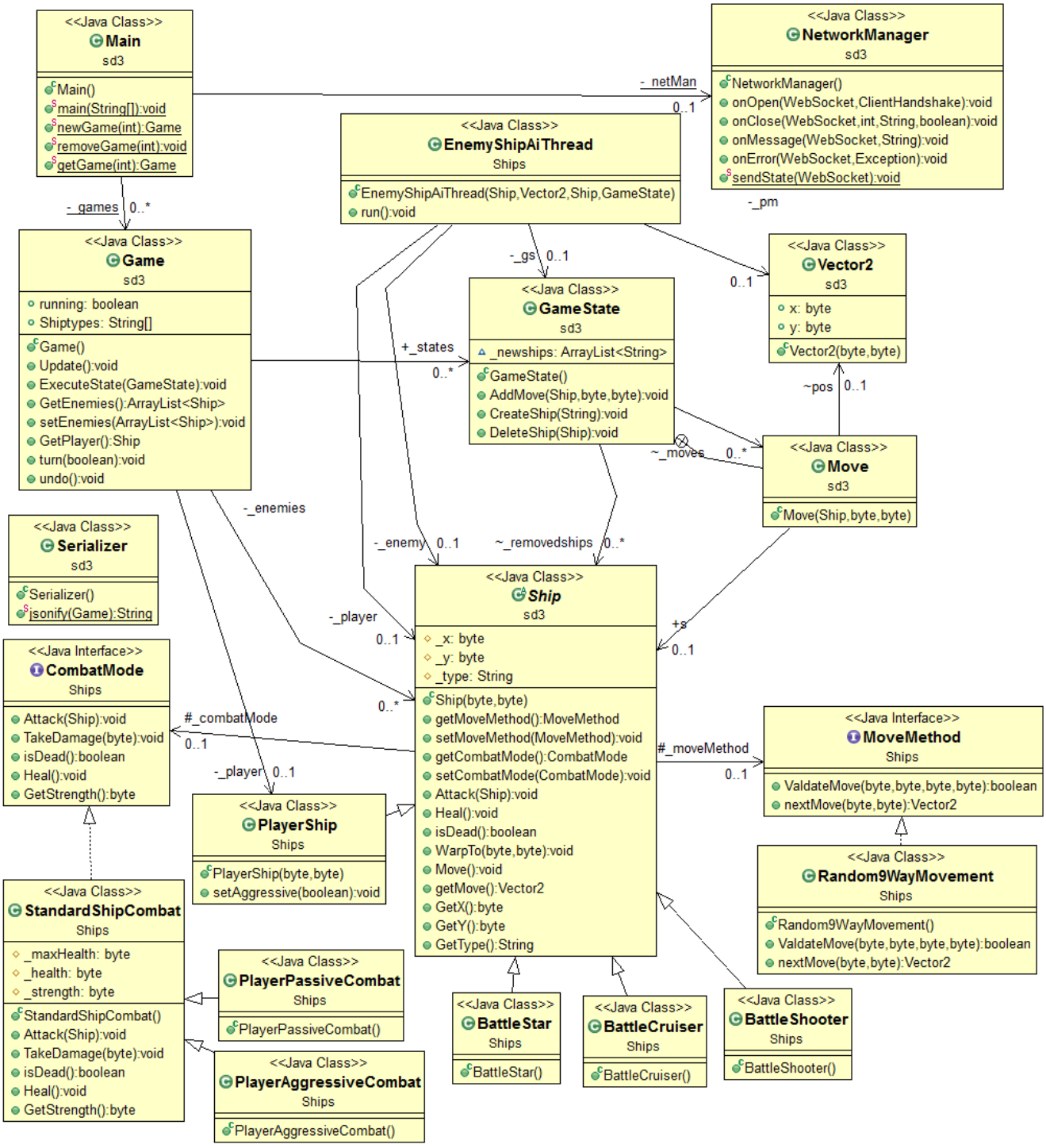

Figure 3: *Class diagram for enitre Server Project*