Edinburgh Napier University

SET11103: Software Development 2

Coursework Report

‘Sky Wars’

40111780

28/03/2018

1 Introduction

The aim of this project was to implement the game *Sky Wars*. *Sky Wars* is a game in which one good Master ship randomly enters a four by four grid and proceeds to move randomly around the grid. On every move, there is a one in three chance of an enemy ship spawning and entering the grid in the top left hand corner of the grid. An enemy ship can either be a BattleStar, a BattleCruiser or a BattleShooter. The type of enemy ship spawned is also random. Once enemy ships enter the board, if the Master ship lands on the same grid square as one enemy ship the Master ship kills that enemy ship. If the Master ship lands on the same grid square as two or more enemy ships, the enemy ships kill the Master ship and the game ends. There is also an offensive mode for the game that requires three or more enemy ships on the same grid square as the Master ship to kill the Master ship.

2 Project Overview

The design for the grid in this project consists of a Square class and a Grid class. The Square class contains an X and Y coordinate, with a toString() override to print out the Square position. The grid class consists of a 2D array of Squares, with each array index corresponding to the Square X and Y coordinate. For example, position [0][0] in the 2D grid array would consist of a Square with coordinate X = 0 and Y = 0.

To keep track of the Master ship, it is only necessary to keep track of one Square. Therefore, the Master ship position is an element of the Grid class. However, the enemy ships can vary and exist in their own object classes. The classes BattleShooter, BattleStar and BattleCruiser inherit from the abstract superclass Ship. The Factory pattern determines which enemy spawns during the game.

The game is visualised through the GUI class, which contains JButtons, JLabels and ImageIcons. The GUI class implements the Observer pattern to receive updates from the Grid class. The Observer pattern requires two interfaces, a Subject (Observable) interface and an Observer interface. The GUI class is an instance of the Observer and subscribes to the Observable Grid class to receive notifications.

3 Design Patterns

As mentioned above, this project implements the Observer pattern and the Factory pattern. The Observer pattern is an advanced feature that automatically updates the GUI of any changes to the Grid class. Upon GUI construction, the GUI registers as an observer of the Grid class. The Grid method notifyObservers() is then called at the following times:

- when the Master ship randomly enters the Grid
- when an enemy ship enters the grid
- when all ships move
- when the game mode (offensive or defensive) is changed
- when an enemy dies
- when the Master ship dies

The Factory pattern is an example of a programming technique not taught on the course module. The Factory pattern uses polymorphism to determine the random enemy spawned in the game. The benefit of the factory pattern is that the code that returns a random enemy is removed from the main code and thus cannot be altered.

4 Polymorphic Programming

The inheritance used in the enemy ship classes is an example of polymorphism. The three enemy ship classes, BattleStar, BattleShooter and BattleCruiser, are subclasses of the abstract class Ship. This allows implementation of the Factory pattern and instantiation of the enemy ship hashmap in the Grid class that records each enemy in game and that enemy's position (HashMap<Ship, Square>). Polymorphism allows the hashmap keys to have varying ship types, enabling the locations of all different enemy ships to be recorded in one hashmap.

5 GUI: Main Features

As mentioned above, the GUI consists of JButtons, JLabels and ImageIcons. The JLabels create the visual representation of the grid, as well as displaying the current game mode (offensive or defensive), the number of enemies in game, and any other updates such as ‘Enemy enters!’ or ‘Game Over!’. The ‘Enter Master’ button calls the randomStartPosition() method in the Grid class, generating the Master ship’s first Square. The ‘Move’ button calls moveMaster(), moveEnemies(), enemyEnter() and collide(). The collide() method checks for any collisions between the Master ship and any enemies, and depending on the current game mode set by the ‘Defensive/Offensive’ button, executes either collisionDefense() or collisionOffense() in the Grid class. The ships used in the GUI are resized png files set to ImageIcons.

6 Other Advanced Features

Besides the Observer pattern implemented in the game, this project also includes offensive and defensive game modes as described above and save game status functionality. The save game function implements serialisation. Whenever any change occurs to the Grid class, as well as notifying all observers (the GUI class), the changes are serialised, so the current game state is always saved in memory. When a player closes the game window, the game is still stored in memory and can be reloaded. Upon programme execution, the player is asked whether they wish to reload the previous game. Should they click yes, an instance of the Grid class is created, passing in the appropriate Boolean value that orders the Grid constructor to deserialise the previous game. This Grid class instance is then passed to a GUI object, which calls the notifyObservers() method in the GUI constructor to reset the ship image icons, game mode and enemy count to the correct values, so the game may continue where it left off.

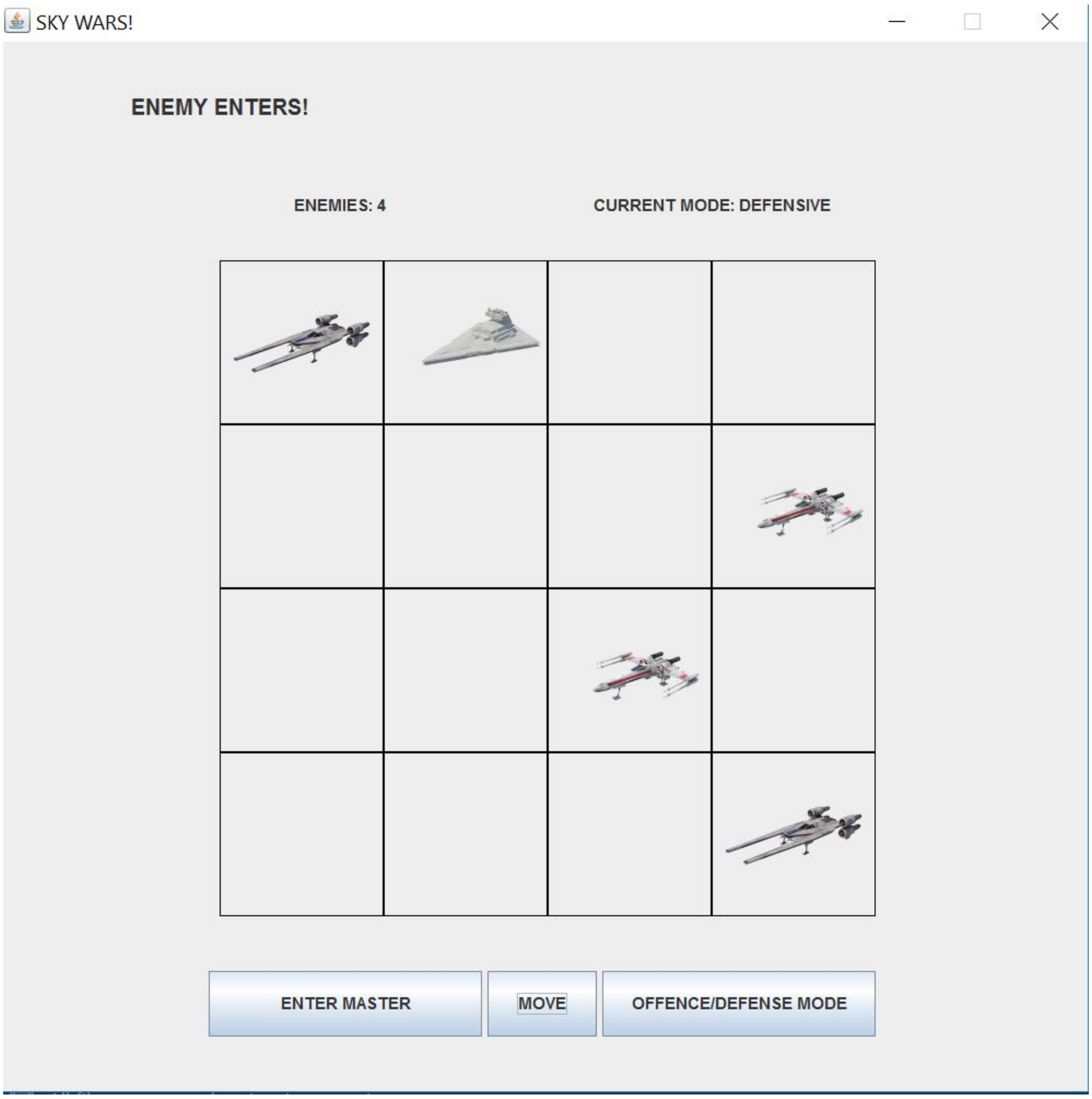

Figure 1: Sky Wars GUI

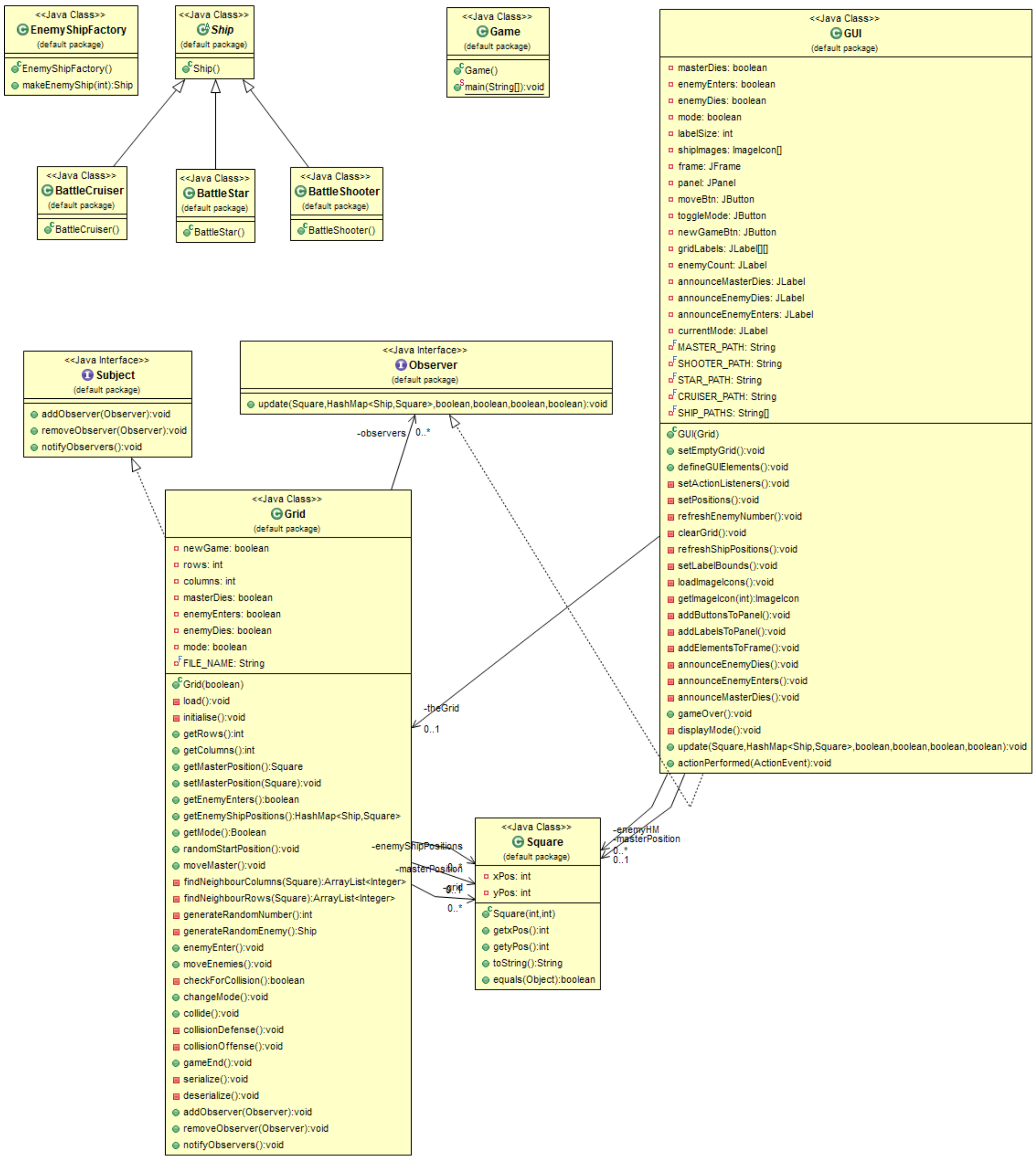

Figure 2: ObjectAid Class Diagram