|  |
| --- |
| Edinburgh Napier University |
| Software Development 3 – Coursework Report |
|  |
|  |
| **40056161** |
|  |

|  |
| --- |
|  |

Contents

[Introduction 2](#_Toc404603024)

[GUI 3](#_Toc404603025)

[Threads 4](#_Toc404603026)

[Design Patterns 5](#_Toc404603027)

[Factory 5](#_Toc404603028)

[Observer 5](#_Toc404603029)

[Strategy 5](#_Toc404603030)

[Command 5](#_Toc404603031)

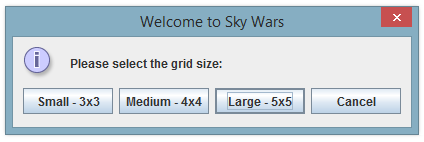
[Programming Techniques Not Taught 6](#_Toc404603032)

# Introduction

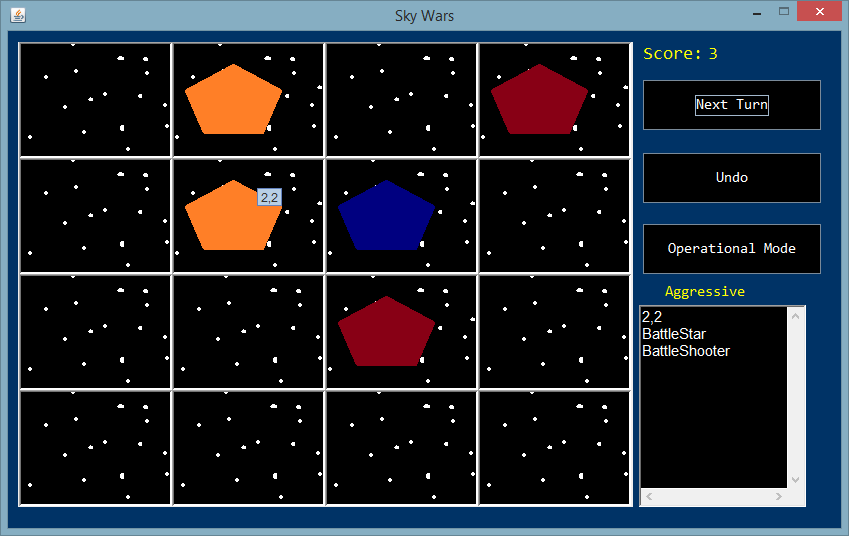
This report documents the software development techniques used to complete the coursework. It will discuss design patterns, use of threads and the main features of the GUI.

Sky Wars is a turned based game which takes place on a grid. The player spaceship moves randomly throughout the grid, it can only move to nearby squares. There is a one in three chance that an enemy ship will enter the grid via the top left hand square. Enemy ships move in a similar fashion to the player ship. If the player lands on the same square as an enemy (or multiple enemies) a battle event occurs. Depending on a number of factors the player will either win or lose the game. The user can change the operational mode of the player ship. Each turn takes place when the user clicks the ‘next turn’ button. Turns can also be undone with the ‘Undo’ button. Every time the player survives a battle their score is incremented.

# GUI

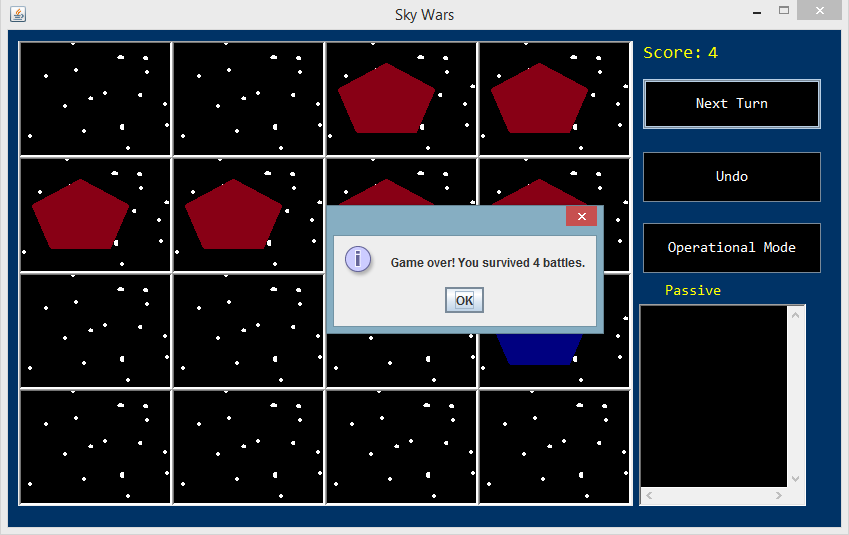
When the game begins the user is given the option to select the size of the grid. Originally I had the option for the user to pick any grid size they desired. However, some grid sizes would make the game unplayable (i.e. 1x7 grid). To circumnavigate this problem I presented the user with three grid sizes to choose from.

The game grid itself is built using JButtons. To the right of the grid there are 3 buttons, ‘Next Turn’, ‘Undo’ and ‘Operational Mode’. The score and the operational mode are displayed using labels.

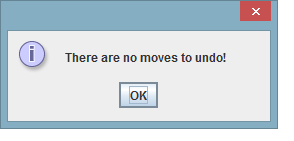


The player ship is shown as blue, whilst a single enemy is shown as red. Orange signifies that there is more than one ship in the square. Hovering over the square shows how many ships are in the text area below the buttons.

When the player is destroyed an alert box appears showing the player’s score.

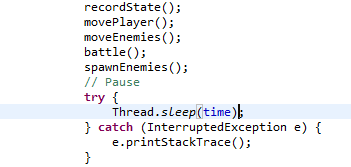


If the player tries to undo past the beginning of the game an appropriate message is displayed.



# Threads

Threads are used to show a pause between the game’s turns. Every time the ‘Next Turn’ button is used, a brief delay occurs before the ships move around the grid . This is also useful as it stops the user from abusing the button which would previously carry out multiple turns all at once.

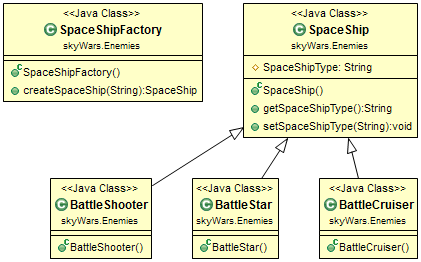


Threads are also used when implementing the sound effects and music. This is discussed later on in *Programming Techniques Not Taught*.

# Design Patterns

## Factory

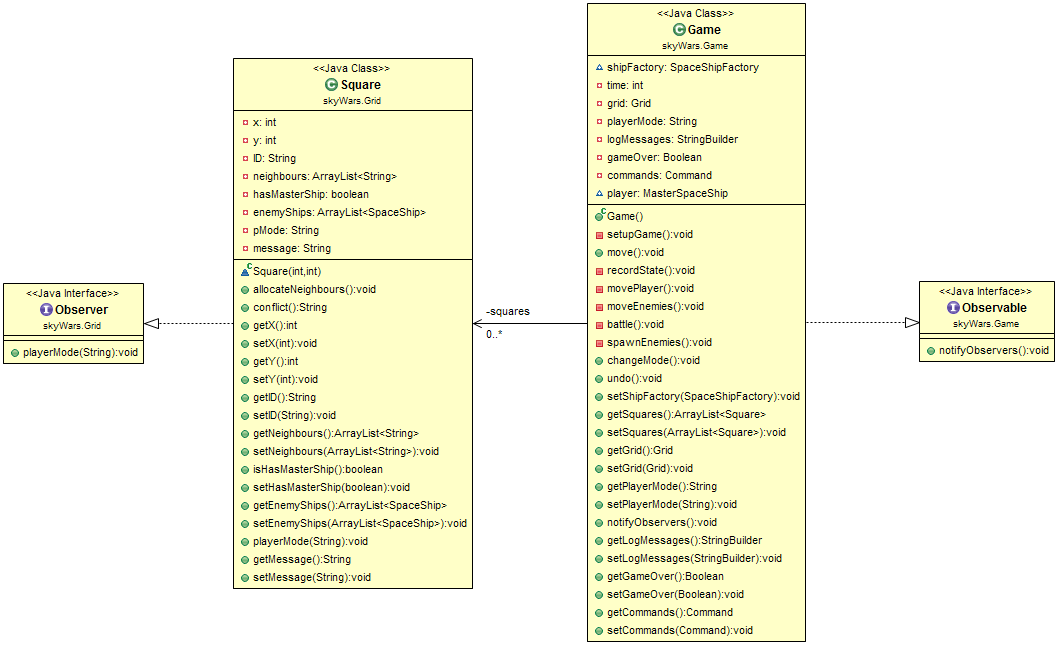
The factory pattern is utilised for the creation of the enemy spaceships. It is designed so that it is easy to create a new type of space ship. For the purposes of the coursework, three types of spaceships are specified. Without a factory pattern it would not be as simple to add a new type of ship.



## 

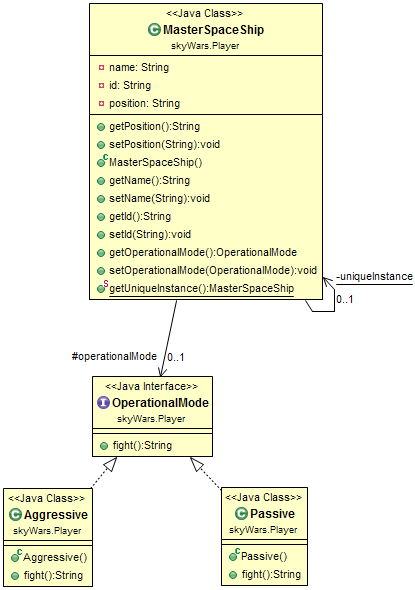
## Observer

When a square contains the player and any number of enemy ships a battle takes place. In order to determine the outcome of the battle the square needs to know the operational mode of the player. To implement this, the observer pattern is utilised. Where each square observes the game object to retrieve the player’s operational mode. The diagramshows that the square object implements the ‘Observer’ interface whilst the game has the interface ‘Observable’.



## Strategy

The user can toggle the player’s operational mode between passive and aggressive at any point in the game. These are behaviours unique to the player class. The Strategy pattern is used to easily swap between player behaviours. Without this pattern changing behaviours would involve accessing the class directly and over-riding methods.



F

## Command

The undo method returns the grid to its previous state. The Command design pattern is used to implement this. Before each turn is processed the Command class records the player position and the enemy spaceships’ positions for each square. The enemy position is a class that stores the SpaceShip object and the square in which it resides. These details are pushed to their respective stacks. When the player click’s the undo button, the Game object invokes the methods returnPlayer() and returnEnemies(). These methods pop the player and list of enemy ships off of the stack and the GUI redraws the game grid to account for the new changes.

Here is an excerpt from the code showing how the stack is used to return the player object:

**public** **void** addPlayer(Square s) {

player.push(s);

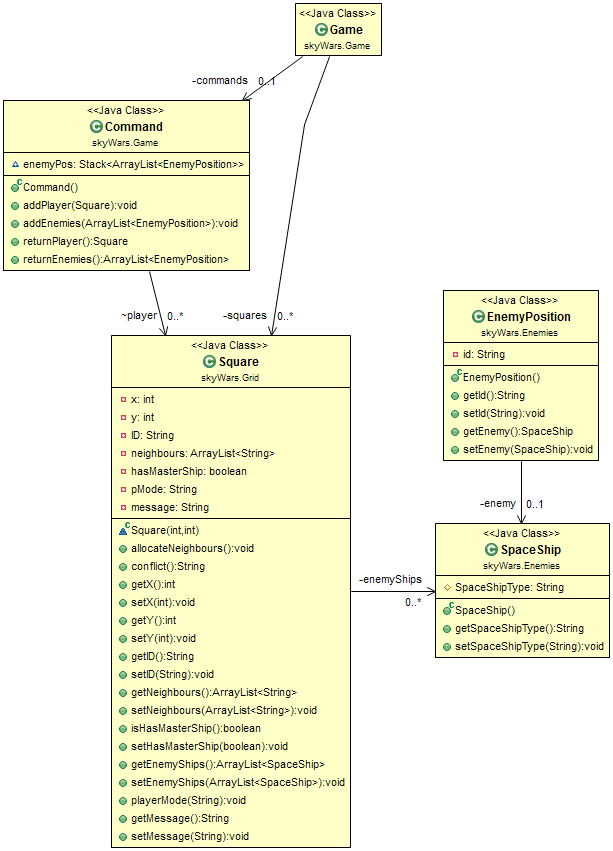
}

**public** Square returnPlayer(){

Square last = player.pop();

**return** last;

}



# Programming Techniques Not Taught

## Sound

The game uses a class called PlaySound to play the music and sound effects. This is called simply by specifying the filename of the .wav file like so:

PlaySound.*play*("laser.wav");

The class starts a new thread which utilises the built-in java audio system. Once the user selects the grid size they wish to play on the music begins. When a battle occurs, the player loses or the operational mode is changed a separate sound effect is played respectively.

Here is the thread being created when PlaySound is called:

**new** Thread(**new** Runnable() {

**public** **void** run() {

**try** {

String file = "Sound Files/"+fileName;

Clip clip = AudioSystem.*getClip*();

AudioInputStream inputStream = AudioSystem.*getAudioInputStream*(**new** File(file));

clip.open(inputStream);

clip.start();

} **catch** (Exception e) {

e.printStackTrace();

}

}

}).start();

As you can see the class uses a ‘Clip’ to play the .wav file, the thread is then started. The code used for sound was inspired from this online example: <http://alvinalexander.com/java/java-audio-example-java-au-play-sound>.

All sound were created from scratch using a MicroKorg XL and Audacity.