Final Year Project Report

Full Unit – Final Report

Concurrency based game environment

Gurbir Singh

A report submitted in part fulfilment of the degree of

BSc (Hons) in Computer Science

Supervisor: Luo Zhaohui

Department of Computer Science
Royal Holloway, University of London

Declaration

This report has been prepared based on my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count: 12710(Excluding table of contents, Bibliography, Acronyms, Appendix: Diary and Notebook)

Student Name: Gurbir Singh

Date of Submission: 05/04/2024

Signature: Gurbir Singh

Table of Contents:

Link To Project Demo: https://youtu.be/gyr-04cxWj4

Abstract

A concurrency-based game environment is an interesting project that combines gaming and ever-evolving computing technology. In this plan, I will explore the exciting topic of concurrency, which allows simultaneous development of various game elements and interactions which will allow multiple games to be played at the same time in a single environment. This plan contains research information on various topics all linking to my project and eventually allowing me to successfully complete this project.

As the number of processors increases, using concurrency effectively is becoming more of a requirement. Threads will be an assured feature that will be used in my project as it will allow the development of my complex system to become more simplified, make better use of multiprocessor systems, and help make the concurrency environment more efficient and quick. Taking concurrency back to its history, in the past operating systems did not exist in computers. The computers had to execute a single program from start to end. This approach was inefficient as it didn't make optimal use of computer resources. As operating systems evolved it enabled operating systems to run multiple programs concurrently, allocating resources like memory and file handles [2].

The term thread is essential as it will be used in my project. It is a sequence of instructions given to the CPU to execute. The more threads the CPU executes the more tasks it can complete. It is essential as it can increase the speed and efficiency of multitasking [1]. Each thread has its program counter, register, and stack which keeps track of the instruction it's executing. All variables are unique to their threads and each thread cannot access other threads' contents. Allowing multiple threads to run simultaneously is also called multithreading [1] [2]. This allows large-scale applications to run much faster than they would run sequentially. If executed correctly in my project it will make playing multiple games at the same time possible via concurrency.

1. Introduction

1.1 The Problem

When developing a concurrency-based game environment , several challenges and considerations must be considered to ensure a seamless, captivating and engaging user experience. In my game, Stratego, in order to make sure that the game runs concurrently i.e. multiple tasks are being done at the same time such as timer, movement of pieces or placing pieces. This can and will be achieved through threads and using multithreading where each game will run on its own thread and each instance of the game will have its own thread such as the timer thread which will not interfere with the other respective games running at the same time simultaneously.[4]

Errors such as deadlocking, race conditions or even logic errors are known as silent errors. Silent errors are errors that may not be visible but could cause concurrent issues that might lead to immediate failure and could significantly impact the user's gaming experience. These errors can cause unexpected behaviour or synchronisation issues without displaying any error messages. To address the challenge of silent errors, I will need to build and integrate robust error handling framework along with implementing a logging framework. I will also need to build a responsive error reporting system which thoroughly logs each error thrown which will improve the overall reliability and stability of the game environment.[4][7]

Providing a seamless, consistent, aesthetically/Visually pleasing GUI(Graphical User Interface) for my game(Stratego) will be an essential and challenging part of building the game. In order to create a good GUI, I will be using JavaFX and Scene Builder. JavaFX is a framework used for building interfaces for Java-based games, it contains a set of graphic libraries and APIs which can be used to create a visually appealing user interface. Scene Builder is a visual layout tool that is used to design JavaFX games/applications, it uses a drag-and-drop system to design and modify the interface.[3]

Another important aspect of the game is user interaction and feedback where the user can have a seamless and engaging gaming experience, in order to achieve this I will need to ensure that the game checks that each independent move while concurrently running other activities in the game, this can be done by having a good user interface system along with providing real-time updates. Keeping in track with checking the moves I have to ensure that the other threads such as the timer thread, piece placement thread, movement threads are running concurrently to the move checking thread in order to maintain consistency and stability of the game environment which in turn provides better User interface experience for the user playing the game. To manage and synchronise all the different threads, I will need to put and code them in a single controller class where they can be linked with the backend(Java) and frontend(JavaFx and Scene Builder) which would make it easier to handle as opposed to having them in different classes which is not a good programming practice and also consumes time and effort.

In order to make the game interesting it will require additional interface features such as a history section which includes the history of previous moves played by the user, the history section does not only mean that the user can see the pieces captured/past moves but use those to gain strategic insights which the user can use against the AI enriching the learning experience. To enhance the competitiveness and adaptability of the AI, I would need to incorporate a dynamic difficulty adjustment system that tailors the challenge level to the player's skill. The AI will adapt to the user's actions and moves of moving/attacking a particular piece and play its moves accordingly, This system will analyse the player's past moves and overall game performance, then it will make the decisions accordingly. The user will be able to play a competitive game against the AI and try to outsmart the AI using their tactics.[8]

Along with the history section, a small section including the rules that states how each piece of a certain rank performs against another piece of the same or different rank for example a Marshal(highest ranked piece) against a General(rank below marshal i.e. marshal wins.).I will try to add and incorporate several additional features to enhance user interaction and experience whilst also having a clear feedback mechanism on the outcomes.

1.2 Aims and Goals of the Project

Project Goals

By the end of the project, I aim to have created a Java application that follows appropriate/correct concurrent practices/methods and also an application that is user-friendly. The application should run efficiently i.e all the threads should work concurrently without interfering with another thread providing a seamless user interface.The application/game should also have a great GUI(Graphical User Interface) to provide for an even better user experience, I aim to create an interface that is visually pleasing yet easy to navigate through which accommodates users at all skill levels. The game should also support concurrent gameplay allowing users to play multiple games simultaneously with each game instance not interfering with the other games that are being played. The gameplay in the application should be responsive providing a gaming experience to the players through code optimisation. Seamless switching between each game is one of the priorities of the game to ensure a smooth transition and gameplay experience for the user without losing any progress while switching.

When developing the game through JavaFX and Scene Builder I would like to have various sections which would make the game look more realistic such as a section that might contain amount of pieces there are per rank, for example:- a marshal piecetype only has one piece on the board as compared to a bomb piecetype which has 6 pieces placed on the board that can be placed as the user would like them to.Along with the section mentioned above I would also like add a section where all the 40 pieces are placed over one or two rows and the user/player can drag and drop the pieces onto the board themselves however they like. I also aim to create dedicated buttons that can place the pieces for the AI randomly as well as for the user but if the user does not like the placement of the pieces that have randomly placed the user can drag and drop the pieces themselves. I would also like to incorporate the drag and drop feature with placing player pieces randomly as when the user has placed a couple pieces by themselves the place player randomly should fill out the remaining places on the board with all the remaining pieces randomly. I would also like to include buttons such as the reset button where the game is reseted back to its original settings(no pieces on the board and game is completely reset) and also a start button that is used to initiate the game where the pieces can no longer be altered once clicked i.e. the game starts players can attack each other. I would also aim to include a timer that runs on its own thread and it only begins once the start button has been clicked. The timer would go back to 00:00:00 when the reset button is clicked and can begin again once the start button is clicked.Since the game is a single player game, I would like to have an AI that adapts to the players behaviour throughout the game and alters the difficulty accordingly.

Additionally, I hope to include a bunch of additional features such as a history of previous moves that have been performed by the users/player or the AI. Another feature or functionality that I would like to add is that when a piece of the player or the AI is killed, it is moved onto a section where all the pieces that have been killed are displayed so that the user/player can see what pieces have been eliminated from the game and plan their next move(s) accordingly.

To help me manage the game better I would like to add each move and play that is being done displayed in the terminal on my end as it would provide me with the correct details whether a play is being done correctly or there is an error. If there is an error it is easy to identify as every single play has been displayed in the terminal along with a message stating which piece killed which piece for example a message displaying:- Blue Marshal has survived and Red General has died. Another example could be when the flag has been captured a message displaying in the terminal stating Blue or Red team has captured the flag or Blue or Red team has won might be ideal. There are two blocks(Water) in the game where the user should not be able to move their pieces to or even select the area.

In order to make the game more user friendly I would like to add a menu bar where the rules can be stated for new players. In order to learn more about the rules the players can refer to the official rulebook of stratego.[6]

I also aim to have an appropriate error report sent if the game crashes due to any errors such as logic errors or concurrent issues such as deadlocking, race conditions.I will try to minimise the scope of developing an error by using the TDD approach and carrying out through and regular testing for the functionalities that are being developed.I also aim to use design patterns to make my code neater and efficient.The design patterns I aim to use are Singleton and MVC design pattern.

These are all the aims I hope to achieve by the end of the project along with having a few extras mentioned above.

Personal Goals

By developing this application/game I hope to have a better understanding of game development, developing Java applications using JavaFX and Scene Builder. I also hope to expand my knowledge on important concepts learnt during the project like threads, multithreading, and concurrency which will help me in the future to become a better developer. I also hope to learn and follow good coding standards i.e. avoiding code smells, testing the code that I have written to check if it works or not making me a better programmer.

As the project has progressed, my ambitions have also expanded. Developing a complex game like Stratego that consists of many functionalities should enhance my coding and complex problem solving skills. While developing the game I hope to gain experience on how to develop threads, I would also like to learn how to use SceneBuilder using the drag and drop method but also learn about FXML code. I aim to learn how to link the backend(Java code) to the frontend(JavaFX and Scene Builder) which would allow the game to operate how it's supposed to. I also hope that my documentation skills will improve which would result in having an organised report throughout the year starting from the project plan, moving onto the interim report and finally the final year report. I will use the feedback that I have received from markers and my supervisor to improve my report writing and documentation skills. From the feedback received about my documentation and after having a meeting with my supervisor I then decided to include a CHANGLOG.md that documents all the changes that have been carried out in the project consisting of several versions. Along with the CHANGLOG.md document I regularly keep a notebook which consists of all the knowledge ,information and also includes my research of all the topics relating to my game. I have also kept a diary of all the work that has been done after receiving the feedback which has improved my organisational and documentation skills which has helped me to stay on top of my work. I will use all these techniques and learn more which will make me a better programmer and also help me in my daily life to stay organised. I will use the feedback that I have received from markers and my supervisor to improve my report writing and documentation skills. This project will significantly enhance my game development capabilities, pushing me beyond basic game production to master complex concurrent approaches and methodologies.This project should also enhance my knowledge of game development, using IDE such as Eclipse, writing reports, and TDD.

Additionally, I aim to learn on how to use a gitlab repository effectively and efficiently by creating different functionalities and branches. By having multiple branches, It will allow me to work on different features and functionalities at the same time without interfering with other functionalities. By following coding conventions I delete the branches that have been completed and are not being used after merging the code with the main branch. Following all these coding standards along with a lot more that I have learnt and will still learn will help me communicate my skills Along with learning how to add meaningful commit messages. I also hope to learn how to manage project timeline by managing my time efficiently whilst creating an adequate work-life balance and not overworking myself. Working on Gitlab by myself also has given me an essential skill of how multiple functionalities can be developed at the same time which will help me a lot while working in a team.All of this should help me in my professional career to become a better software engineer.

1.3 Survey of Related Literature

1. **The Structured Phase of Concurrency. - Artem Polyvyanyy & Christoph Bussler**

   This abstract looks in a smart way to structure a problem using concurrency and goes into detail about what processes and other terms related to concurrency are. I will be using this paper as a reference through the game development to help me understand how to structure a problem in a better way.

2. **Java concurrency in practice. Pearson Education. - Brian Goetz**

   This is a book that I have used since the start of the development phase that has allowed me to learn alot about thread safety, multithreading and concurrency design patterns. The book focuses on the best practices associated with concurrent programming in java.

3. **Mastering the game of Stratego with model-free multiagent reinforcement learning - Julien Perolat, Bart de Vylder, Daniel Hennes, Eugene Tarassov, Florian Strub, Vincent de Boer, Paul Muller, Jerome T. Connor, Neil Burch, Thomas Anthony, Stephen McAleer..**

   This research paper helped me understand the game of stratego, it showcases advancements in strategic decision-making and artificial intelligence.I have used this paper since the development phase. This paper has been referenced throughout my report.

4. **https://www.nordicgames.is/wp-content/uploads/2015/06/Stratego_Travel_Rules_EN.pdf - Nordic Games**

   This pdf file includes all the rules and information about the game. It states the roles of all the pieces along with all the information which I used to gain knowledge about the game and also it helped me in implementing the game logic of my game.

5. **JavaFX Developer's Guide. Pearson Education. - Kim Topley**
   The JavaFx developer’s guide book has been used throughout my project. The book contains all of the materials needed to develop the Graphical User Interface for my Stratego application. This book has helped me learn about JavaFX utilities, exception handling, event listeners, and other features. This has allowed me to create a strong, multi-threaded game environment with a visually beautiful and intuitive interface that consists of a menu bar, buttons, panels, Hbox, Vbox, Scroll pane and many more containers. Using the book, I was able to produce an aesthetically and visually pleasing application that not only works effectively and efficiently but also provides a seamless and engaging user experience, which again was one of my projects main

objectives as it increases user satisfaction.Beyond JavaFX resources, the book contains other incredible resources; I have used it to gain insights into efficient code structuring and optimisation techniques, which have been critical in writing clean, maintainable, and high-performing FXML code for my Stratego application, thereby improving the overall quality of my application. This has helped me to lay a solid basis for learning JavaFX principles and it works along with providing my application with a balance of visual appeal and technical competence.

6. **Pro JavaFX 9: A Definitive Guide to Building Desktop, Mobile, and Embedded Java Clients - Johan Vos, Stephen Chin, Weiqi Gao, James Weaver, Dean Iverson, and Johan Vos**

   The guide book has been used in my project for building an aesthetically and visually pleasing GUI. I have mainly researched around chapter 9 of this book as it contains the main section that I required for my project that is Scene Builder. This book gave me a great layout on how to create an application using different components such as the use of containers(Anchor pane, Grid pane, Vbox, Hbox, Stack pane etc.). Containers allowed me to organise and hold UI components within the Stratego application. Along with containers it gave me knowledge on how to use and operate various controls such as buttons, Labels, TextField, CheckBox etc.. Controls allowed me in multiple ways, with the use of controls I was able to create buttons to place the pieces randomly for the player and the AI. This book has helped me improve my GUI development skills and I will refer to it in the future to learn new concepts.

7. **Concurrent programming in Java: design principles and patterns - Doug Lea**

   The book ‘Concurrent programming in Java: design principles and patterns.’ written by Doug Lea delves into an in depth discussion on concurrent programming in Java. The book provided me with valuable insights into Java concurrency API and also helped me learn on how to implement and create a robust multithreaded application. This book also helped me to gain and enhance my skills on design patterns and techniques. Through this book I also understood how to optimise concurrent programs which is essential for games and avoid significant pitfalls such as race conditions and deadlocks, which can have an effect on the gaming experience. This book laid a solid foundation for implementing the game logic.

8. **Refactoring: improving the design of existing code. Addison-Wesley Professional.- Martin Fowler**

   The book on refactoring allowed me to learn how to refactor my code efficiently which improved the project's overall readability, maintainability whilst reducing the complexity of the code making it easier to understand and read the code.

1.4 Objectives- Milestones Summary

This section of the report includes the summary of all the objectives /milestones that I wanted to achieve and their current progress.

Objectives/Milestones:

1. The main objective/milestone of this project was to create a concurrency-based game environment which can handle multiple tasks simultaneously in a single game instance. I have successfully achieved this objective by implementing a Stratego game that can run concurrently. This took a lot of research and practice by following tutorials along with implementing the code.

2. Create an aesthetically pleasing GUI with the use of JavaFX and Scene Builder. I have successfully achieved this objective/milestone by implementing a visually pleasing GUI by having a consistent colour scheme throughout, adding extra features such as a history of moves, amount of pieces a player has per pieceType and many more.This took a lot of research and practice by following tutorials along with implementing the code.

3. Implement a robust error-handling framework which provides real-time updates as the game progresses and tells the user if there is any error. I have successfully implemented this objective/milestone by using a lot of expectation handling and other error-handling constraints throughout the code. If there is any error in the game whilst playing it, the error is instantly shown in the terminal and shows what kind of error has been thrown which makes it easier to find the area where the error has occurred and use debugging to fix it.This took a lot of research and practice by following tutorials along with implementing the code.

4. Another objective that I wanted to achieve was having advanced gameplay features such as having an AI and playing against it instead of just playing a two-player game. I successfully achieved this by implementing an AI that the player can play against. The pieces of AI are also placed randomly and concurrently as the player is placing

their pieces.This took a lot of research and practice by following tutorials along with implementing the code.

5. Adding JavaDoc and comments to make the code easier to read. This objective/milestone was successfully achieved as I have added JavaDoc and comments in almost every single class of the code. JavaDoc makes it easier for the reader to know what the class does and how the methods affect the class which reduces confusion and reduces complexity.

1.5 Rationale

This project is centred around creating a concurrency-based game environment, which from what I understand means having many tasks running concurrently(simultaneously) to each other in a single game instance on different threads without interfering with each other and without interfering with the main application. The use of threads is essential in implementing a concurrency-based game environment as it allows many tasks to run on its own thread such as a timer thread runs concurrently to AI and player movement threads without interfering with each other. The purpose of this project was to delve into the realm of concurrent gaming environments and explore its key principles and the reason that concurrency is needed in today's world.

I used these key principles to implement a visually appealing GUI. Along with implementing GUI, I focused on creating a robust error handling framework which would make it easier to handle errors and debug them along with having advanced gameplay features such as an AI. All these decisions were inspired from having a great user experience i.e. the user has a nice experience exploring my Stratego game and found it engaging.

Moreover, this project gave me the opportunity to add significant growth professionally as well as personally.I gained a lot of knowledge of various new topics and computer science practices such as using JavaDocs, using agile methodology and also different programming approaches. Finally, I would like to add that this project allowed me to push myself outside my comfort zone and allowed me to explore new challenges technically in the realm of concurrency.

2. Concurrency-Based Game Environment Frameworks

2.1 States-of-the-art game development

In this section of the report, I will be delving into how concurrency is used in the modern game environment and the reason it is essential to have concurrency.

Developing games is a very challenging task as it involves a lot of different features and functionalities. Therefore, the game needs to run these features simultaneously to other features given the game scenarios to increase user interaction and user satisfaction. This can be done efficiently through the use of concurrency as in concurrency multithreading can be used to run these tasks simultaneously, multithreading is when a lot of threads work concurrently(simultaneously) without interrupting other threads. Each thread can be used to work on a single feature such as movement or timer. By using concurrency running tasks simultaneously effectively and efficiently[4]. All modern games use concurrency in some way or another. Some of the big-name games that use concurrency are: Fortnite, Street Fighter V and Battlefield V. This just goes to show how important concurrency is in order to make the games run efficiently to maximise user interaction.

2.2 Architectural paradigms and design patterns

In this section of the report, I will be discussing the architectural paradigms and design pattern(s) that I have used to develop my game.

In the game Stratego, I have used the separation of concerns paradigm by implementing the MVC(Model View Controller) design pattern. The separation of concerns paradigm is essentially a design principle that is achieved by 'separating a computer program into distinct sections' [13]. Just like how the MVC design pattern works, the problem is split into three sections. The model, the view and the controller. In my project, the model consists of all the important classes that are required for the game Stratego. I have used a StrategoController.java class as a controller to create a link between the view and the model essentially linking the view and model together and making sure that the game logic is correctly displayed. The view consists of the primary.fxml document which contains the FXML code which displays the game on the window. I would have also liked to implement a singleton design pattern but due to time constraints and also having little knowledge of the pattern I was unable to implement that to make the game run even more efficiently.

3. Software Engineering

3.1 Methodology

Throughout the development of the project I have followed a mix of extreme, multithreading and parallel development/programming approaches. Extreme programming allowed me to use the TDD(Test driven development) approach which I have been taught and have great knowledge of during second year. By using the extreme programming approach I was able to structure my code well and also it allows me to refactor the code that I had written. Refactoring is essentially the process of restructuring the code by changing its internal structure but the behaviour of the code remains the same. It improves the readability of the code and reduces complexity as well as enhancing maintainability [12]. Multithreading allowed me to use threads which helped me in achieving concurrency.

TDD(Test Driven Development)

I have been using the same thought process throughout the development of this project, especially when implementing the game logic. In TDD we write tests first and then write the code to make the test pass[5].The TDD cycle consists of 5 parts, we start with writing a test , then check if the test fails, if it does fail then we write the necessary code to make the test pass and lastly check if the test passes and if it does pass repeat from the beginning to write a new test however if the test fails we need to go back and write or correct the code.[5] This helps reduce confusion, increases efficiency and allows the programmer/developer to develop good coding standards whilst maintaining good code quality. Testing the code multiple times also reduces the chances of error in the program/application which would ensure an enhanced and uninterrupted user experience. It was important to follow the TDD approach as it allowed me to test my code and check if it was correct which saved me a lot of time and energy in not wasting my time just developing and then finding a bug and going back to the start and looking for the error/bug in my code. The TDD approach particularly helped me in implement the movement and placement of the pieces on the game board as I tried to write a bit of code without testing which got me stuck for days and then used the TDD approach which helped me implement the movement by firstly just writing a test in the BoardTest.java class and then writing the code. This approach provided me with a solid foundation for my project as I focused on implementing the game logic first which is the most essential part of the game. I then used that game logic to develop the GUI for the project during the second half of the term.

Multithreading

I have used multithreading to implement concurrency in my project. Multithreading(MT) is the process of simultaneous execution of multiple threads which allows for parallel processing. As I implemented various threads in a single instance of the game it was important to have many functionalities running at the same time without one functionality interfering with another. Therefore, Using multithreading allowed me to achieve this as I was able to combine and run all the threads that I had implemented on a single game instance. The project consists of various threads. Some of the threads that I have implemented are:

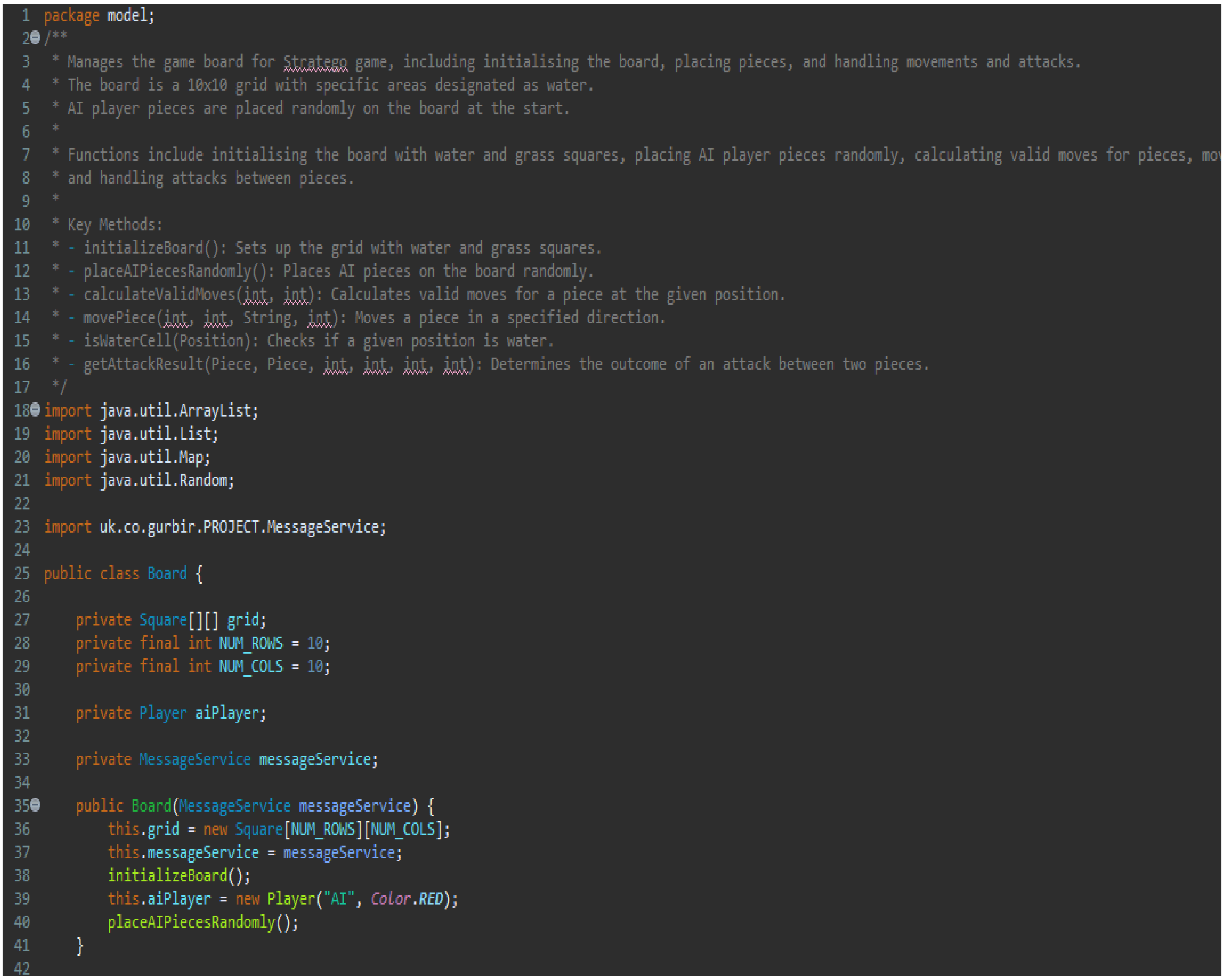
- A timer thread which runs in HH:MM:SS format once the game starts and it concurrently is updated as the game carries on every second. The thread starts when the start button is clicked
- A Player placement thread for the player themselves and the AI. I used two separate threads, one for allowing the player to place their pieces on the board and another thread that enables the AI to place the pieces on the board randomly. The player has two options to place the pieces either by drag-and-drop method or using the place piece randomly button. However, the player can also drag-and-drop a few pieces onto the board and then select the button to place the pieces and the remaining spaces will be filled. The AI pieces however are placed only using the button otherwise the player would know when the pieces have been placed and can cheat. These two threads run concurrently to each other as the player is placing pieces on the board, the AI is also placing the pieces randomly. Therefore, running concurrently to each other.
- A move piece thread. I decided to have a thread for moving the pieces on the board. This thread was essential in the player movement as it runs concurrently to the timer threads as well as other threads.

Overall, I believe that following these methodologies which consisted of extreme programming and Multithreading. It allowed me to achieve most of the objectives that I had set to achieve given the time frame of the project whilst learning many new topics and practices that I had never used before.

3.2 Code Snippets

This part of the software engineering section will include code snippets of the important and interesting methods that I have implemented and coded in my project. The snippets are organised and added in order of classes. If a class name has been mentioned above then all the methods mentioned below it are of the same class. If there is change in class that means that a new class code snippet is being added. This will allow me to maintain an organised approach rather than just including snippets randomly from different classes.

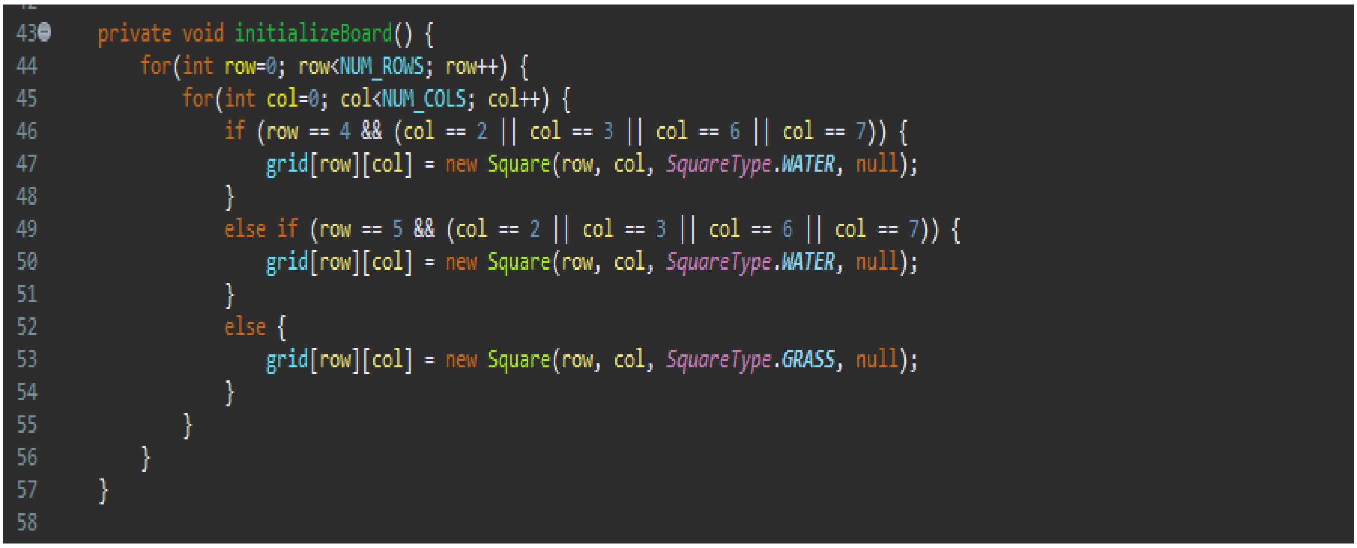
Board Class:



```java
package model;
/**
 * Manages the game board for Stratego game, including initialising the board, placing pieces, and handling movements and attacks.
 * The board is a 10x10 grid with specific areas designated as water.
 * AI player pieces are placed randomly on the board at the start.
 *
 * Functions include initialising the board with water and grass squares, placing AI player pieces randomly, calculating valid moves for pieces, mov
 * and handling attacks between pieces.
 *
 * Key Methods:
 * - initializeBoard(): Sets up the grid with water and grass squares.
 * - placeAIPiecesRandomly(): Places AI pieces on the board randomly.
 * - calculateValidMoves(int, int): Calculates valid moves for a piece at the given position.
 * - movePiece(int, int, String, int): Moves a piece in a specified direction.
 * - isWaterCell(Position): Checks if a given position is water.
 * - getAttackResult(Piece, Piece, int, int, int, int): Determines the outcome of an attack between two pieces.
 */
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.Random;

import uk.co.gurbir.PROJECT.MessageService;

public class Board {

    private Square[][] grid;
    private final int NUM_ROWS = 10;
    private final int NUM_COLS = 10;

    private Player aiPlayer;

    private MessageService messageService;

    public Board(MessageService messageService) {
        this.grid = new Square[NUM_ROWS][NUM_COLS];
        this.messageService = messageService;
        initializeBoard();
        this.aiPlayer = new Player("AI", Color.RED);
        placeAIPiecesRandomly();
    }

```

The board class is one of the core classes used to implement the game. As it is used to display the board that will be used to place the pieces on.The board has a 10x10 grid consisting of 10 rows and 10 columns. The board class also oversees the ongoing game states and also game actions such as placement of a piece, movement and even attack are all overseen in this class.

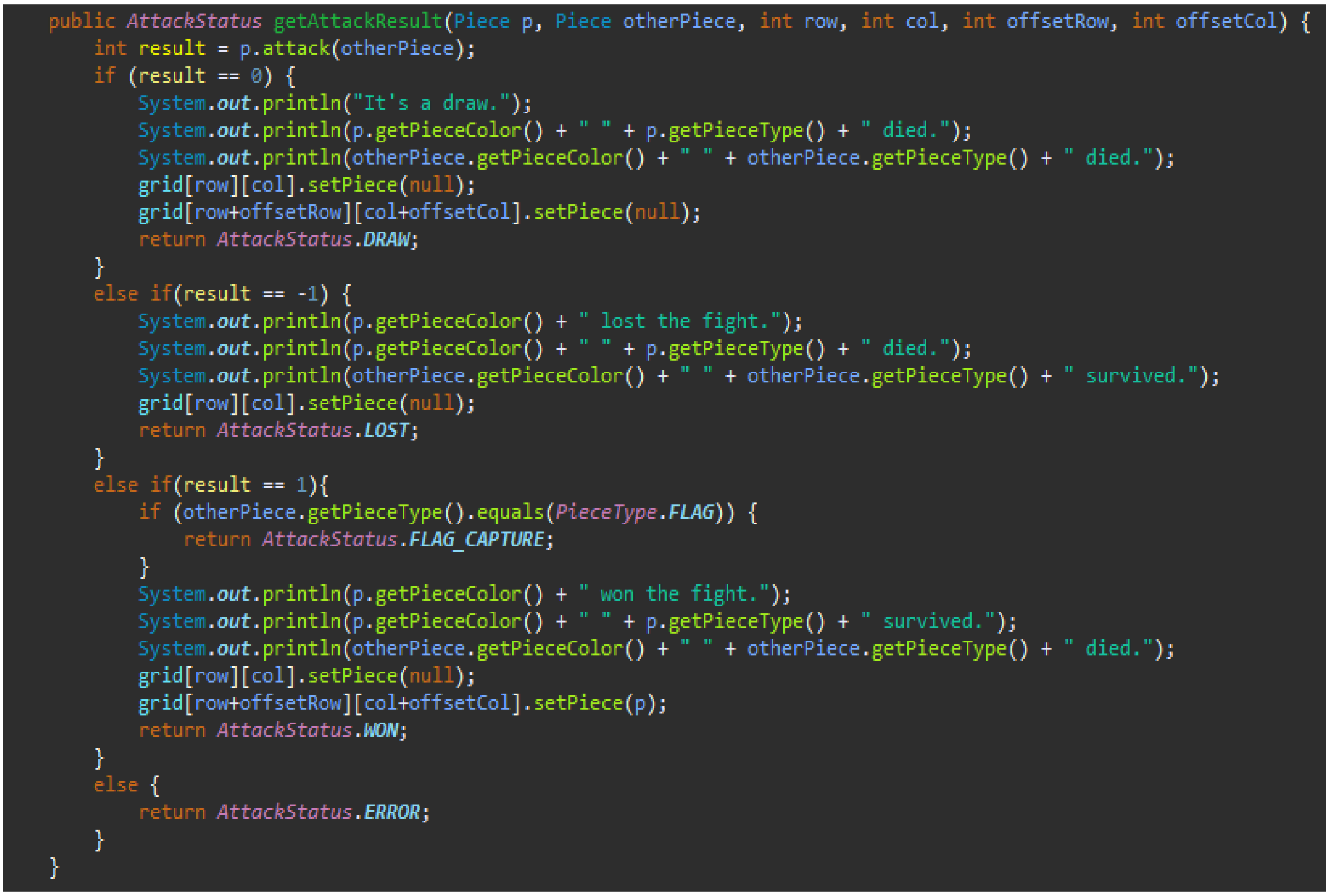
initializeBoard method:



```
    private void initializeBoard() {
        for(int row=0; row<NUM_ROWS; row++) {
            for(int col=0; col<NUM_COLS; col++) {
                if (row == 4 && (col == 2 || col == 3 || col == 6 || col == 7)) {
                    grid[row][col] = new Square(row, col, SquareType.WATER, null);
                }
                else if (row == 5 && (col == 2 || col == 3 || col == 6 || col == 7)) {
                    grid[row][col] = new Square(row, col, SquareType.WATER, null);
                }
                else {
                    grid[row][col] = new Square(row, col, SquareType.GRASS, null);
                }
            }
        }
    }
```

The intializeBoard() method used to initialise the board and in this method it can be seen that on the board there are some parts that consist of water where the pieces cannot be placed on.

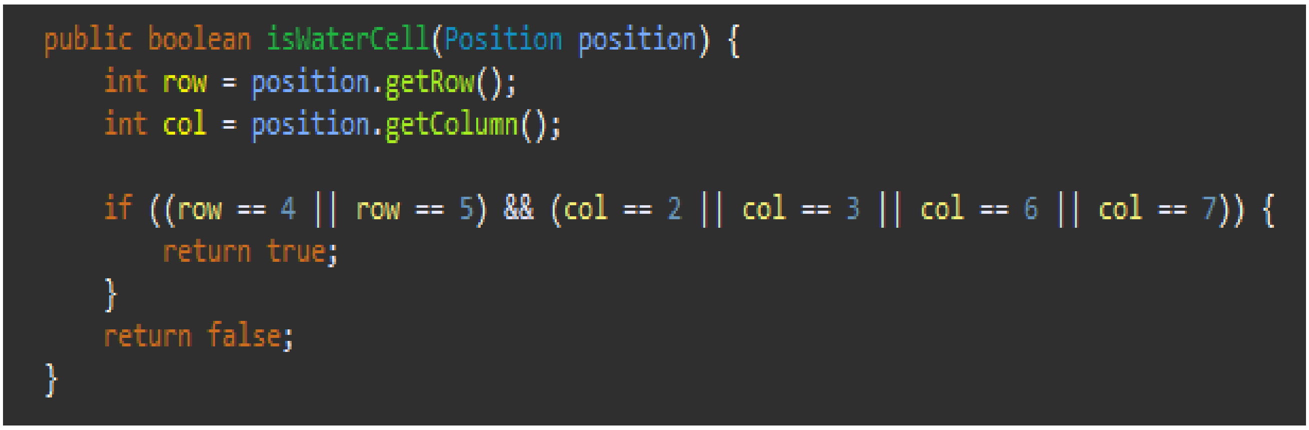
getAttackResult method:



```
    public AttackStatus getAttackResult(Piece p, Piece otherPiece, int row, int col, int offsetRow, int offsetCol) {
        int result = p.attack(otherPiece);
        if (result == 0) {
            System.out.println("It's a draw.");
            System.out.println(p.getPieceColor() + " " + p.getPieceType() + " died.");
            System.out.println(otherPiece.getPieceColor() + " " + otherPiece.getPieceType() + " died.");
            grid[row][col].setPiece(null);
            grid[row+offsetRow][col+offsetCol].setPiece(null);
            return AttackStatus.DRAW;
        }
        else if(result == -1) {
            System.out.println(p.getPieceColor() + " lost the fight.");
            System.out.println(p.getPieceColor() + " " + p.getPieceType() + " died.");
            System.out.println(otherPiece.getPieceColor() + " " + otherPiece.getPieceType() + " survived.");
            grid[row][col].setPiece(null);
            return AttackStatus.LOST;
        }
        else if(result == 1){
            if (otherPiece.getPieceType().equals(PieceType.FLAG)) {
                return AttackStatus.FLAG_CAPTURE;
            }
            System.out.println(p.getPieceColor() + " won the fight.");
            System.out.println(p.getPieceColor() + " " + p.getPieceType() + " survived.");
            System.out.println(otherPiece.getPieceColor() + " " + otherPiece.getPieceType() + " died.");
            grid[row][col].setPiece(null);
            grid[row+offsetRow][col+offsetCol].setPiece(p);
            return AttackStatus.WON;
        }
        else {
            return AttackStatus.ERROR;
        }
    }
```

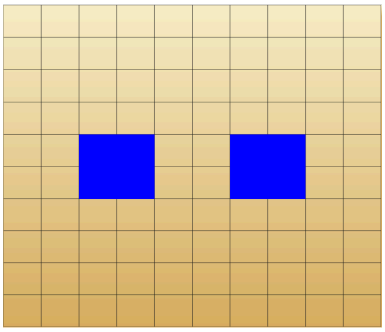
The getAttackResult method determines the outcome of an attack that happens between two pieces. For example: Marshal vs General. Marshal would win as it has a higher rank. The method handles the logic of the attack in the game. Attacking and defending of pieces are both taken into consideration in this method and the method uses the AttackStatus enum in order to categorise the outcome of the attack. The outcome is categorised into 5 parts: WON, LOST, DRAW, FLAG_CAPTURE and ERROR. The WON outcome is when the attack is successful and the piece captures the other piece it was attacking. The LOST outcome is when the attacking piece has lost in battle against the piece it was attacking due to having a smaller rank or maybe even a special piece. The FLAG_CAPTURE outcome is when the flag has been captured, this is where the game ends when the flag has been captured. The ERROR outcome is when there has been an unexpected error in the game.

isWaterCell method:



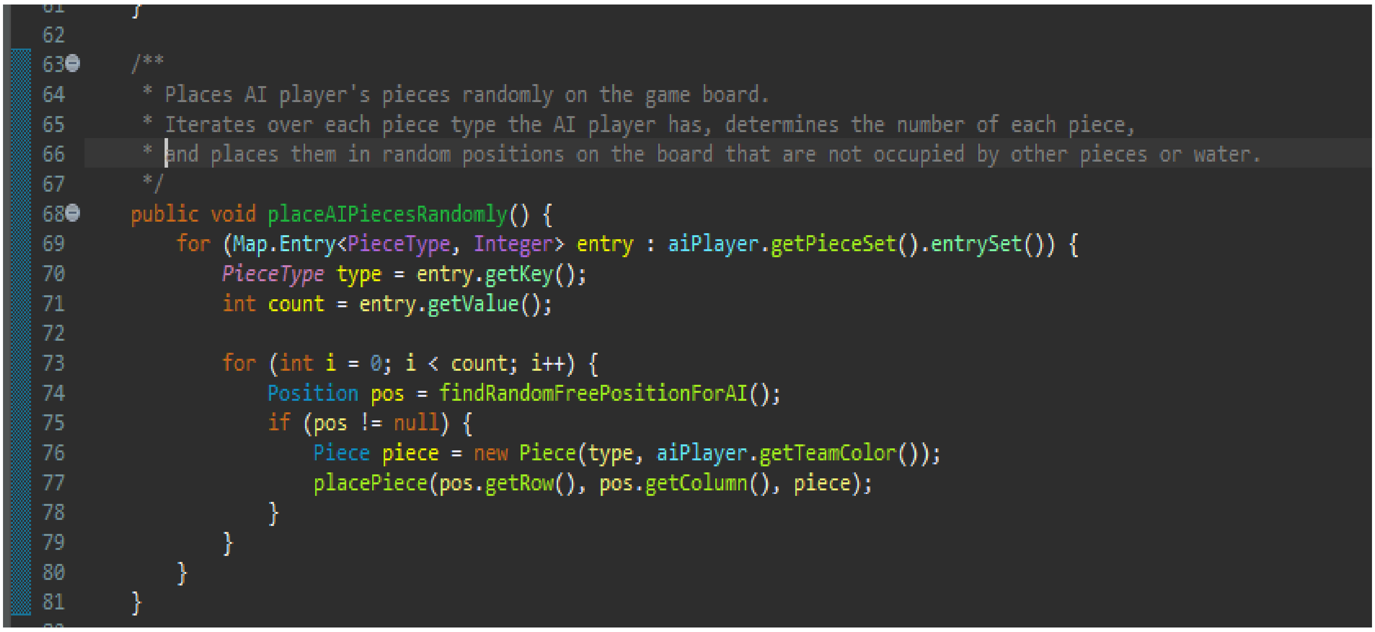
```
public boolean isWaterCell(Position position) {
    int row = position.getRow();
    int col = position.getColumn();

    if ((row == 4 || row == 5) && (col == 2 || col == 3 || col == 6 || col == 7)) {
        return true;
    }
    return false;
}
```

The isWaterCell method in board class is used to determine whether a position is water or not. As if it is a water cell, that area of the game board can not be selected by the player/user as it is a non-traversable region in the game. The water cells are usually found in the middle of the game grid, it can be seen here that if it row is 4 or row 5 and if its column 2 or column 3 column 6 or column 7 then true is returned which means it is a water cell in that area on the game grid otherwise false is returned. The board in execution look like the one below:



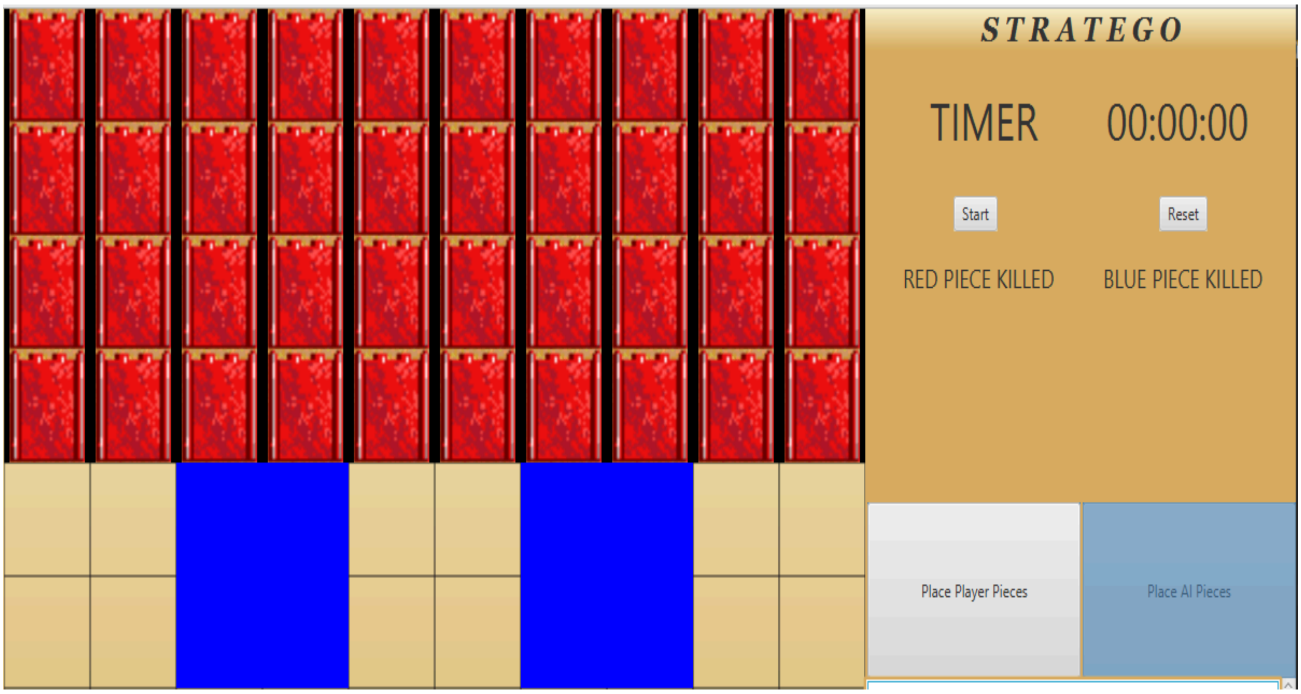
(water cells have been filled at the position mentioned above)

placeAIPiecesRandomly method:



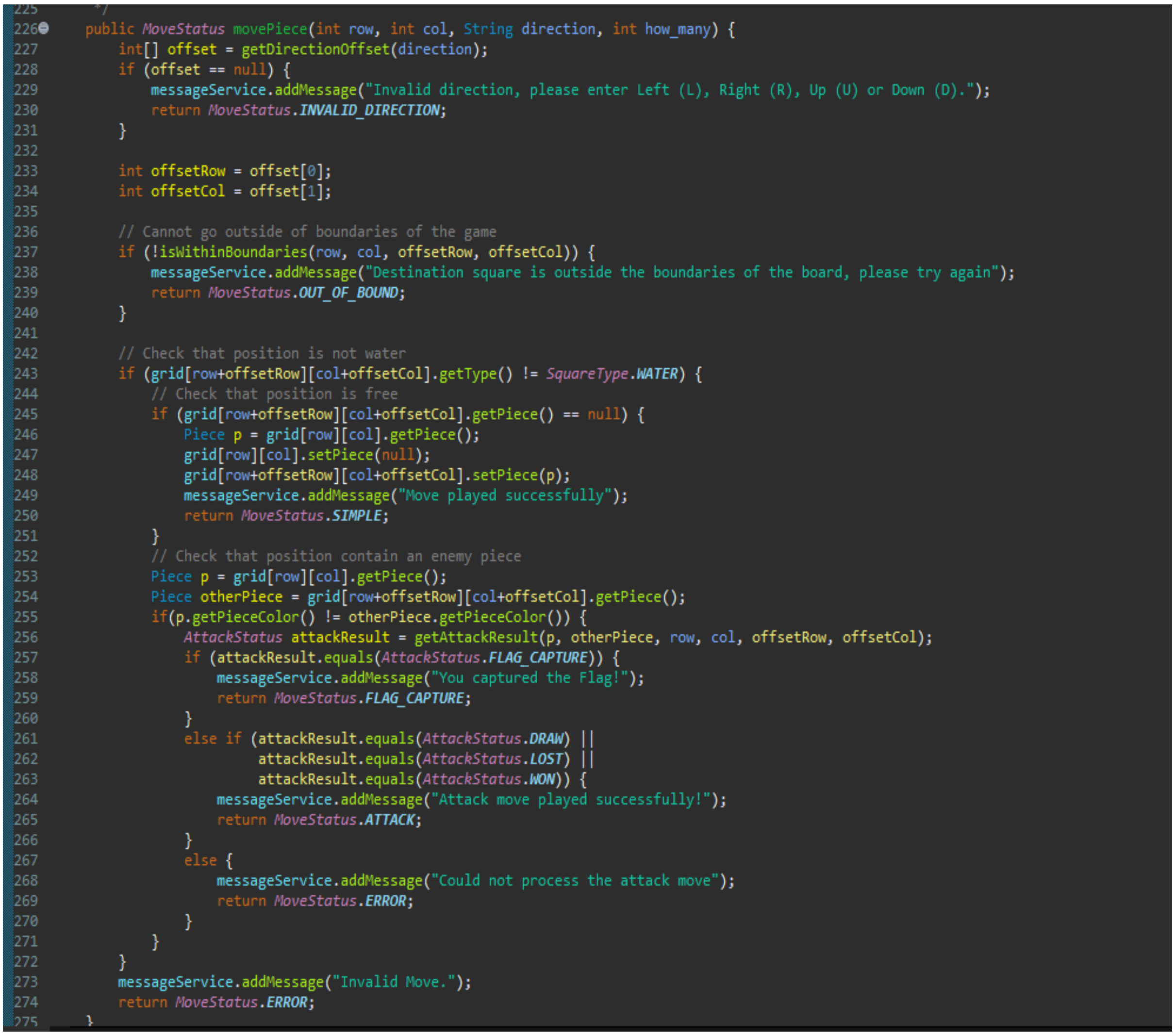
```
    }

    /**
     * Places AI player's pieces randomly on the game board.
     * Iterates over each piece type the AI player has, determines the number of each piece,
     * and places them in random positions on the board that are not occupied by other pieces or water.
     */
    public void placeAIPiecesRandomly() {
        for (Map.Entry<PieceType, Integer> entry : aiPlayer.getPieceSet().entrySet()) {
            PieceType type = entry.getKey();
            int count = entry.getValue();

            for (int i = 0; i < count; i++) {
                Position pos = findRandomFreePositionForAI();
                if (pos != null) {
                    Piece piece = new Piece(type, aiPlayer.getTeamColor());
                    placePiece(pos.getRow(), pos.getColumn(), piece);
                }
            }
        }
    }
```

The placeAIPiecesRandomly method is an essential as this method places the AI player pieces randomly on random positions on the board whilst ensuring that the pieces have not been overlapped on each other, not placed on square that have already been occupied by other pieces and also ensuring that the pieces have not been placed on water. The pieces will be covered so that the player is not able to see them otherwise they can cheat the game. The AI pieces that will be randomly placed look like the image shown below:



It can be seen in the above image that the AI piece button has been pressed, the pieces have been laid out on the board and been placed randomly for the AI to play the game.

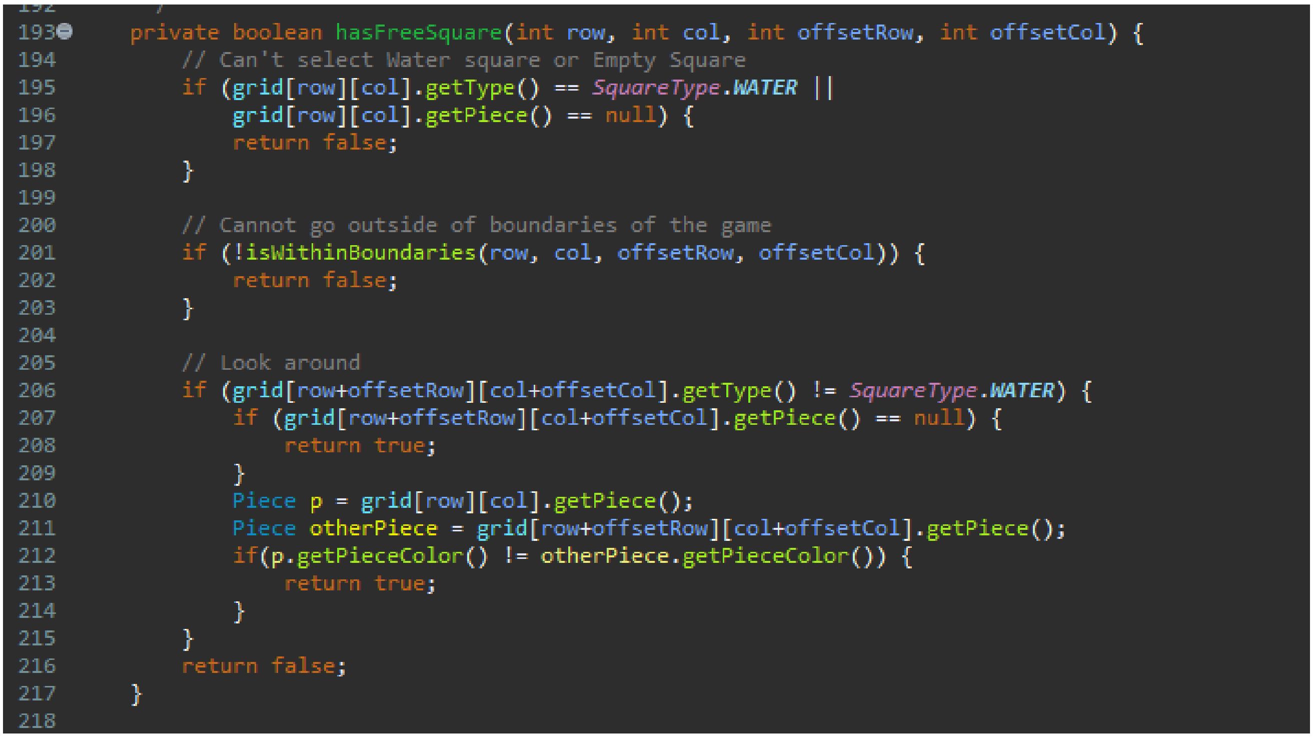
movePiece method:



```
     */
    public MoveStatus movePiece(int row, int col, String direction, int how_many) {
        int[] offset = getDirectionOffset(direction);
        if (offset == null) {
            messageService.addMessage("Invalid direction, please enter Left (L), Right (R), Up (U) or Down (D).");
            return MoveStatus.INVALID_DIRECTION;
        }

        int offsetRow = offset[0];
        int offsetCol = offset[1];

        // Cannot go outside of boundaries of the game
        if (!isWithinBoundaries(row, col, offsetRow, offsetCol)) {
            messageService.addMessage("Destination square is outside the boundaries of the board, please try again");
            return MoveStatus.OUT_OF_BOUND;
        }

        // Check that position is not water
        if (grid[row+offsetRow][col+offsetCol].getType() != SquareType.WATER) {
            // Check that position is free
            if (grid[row+offsetRow][col+offsetCol].getPiece() == null) {
                Piece p = grid[row][col].getPiece();
                grid[row][col].setPiece(null);
                grid[row+offsetRow][col+offsetCol].setPiece(p);
                messageService.addMessage("Move played successfully");
                return MoveStatus.SIMPLE;
            }
            // Check that position contain an enemy piece
            Piece p = grid[row][col].getPiece();
            Piece otherPiece = grid[row+offsetRow][col+offsetCol].getPiece();
            if(p.getPieceColor() != otherPiece.getPieceColor()) {
                AttackStatus attackResult = getAttackResult(p, otherPiece, row, col, offsetRow, offsetCol);
                if (attackResult.equals(AttackStatus.FLAG_CAPTURE)) {
                    messageService.addMessage("You captured the Flag!");
                    return MoveStatus.FLAG_CAPTURE;
                }
                else if (attackResult.equals(AttackStatus.DRAW) ||
                         attackResult.equals(AttackStatus.LOST) ||
                         attackResult.equals(AttackStatus.WON)) {
                    messageService.addMessage("Attack move played successfully!");
                    return MoveStatus.ATTACK;
                }
                else {
                    messageService.addMessage("Could not process the attack move");
                    return MoveStatus.ERROR;
                }
            }
        }
        messageService.addMessage("Invalid Move.");
        return MoveStatus.ERROR;
    }
```

The movePiece method is a core part of the code as it handles all the logic operations that are needed to move a piece on the board. It ensures that the move that is being carried out is valid. It also handles attacks and keeps updating the board accordingly. The movePiece method uses MoveStatus which is an enum to return an instance of MoveStatus in order to indicate the result of an action(WIN, LOST, DRAW).

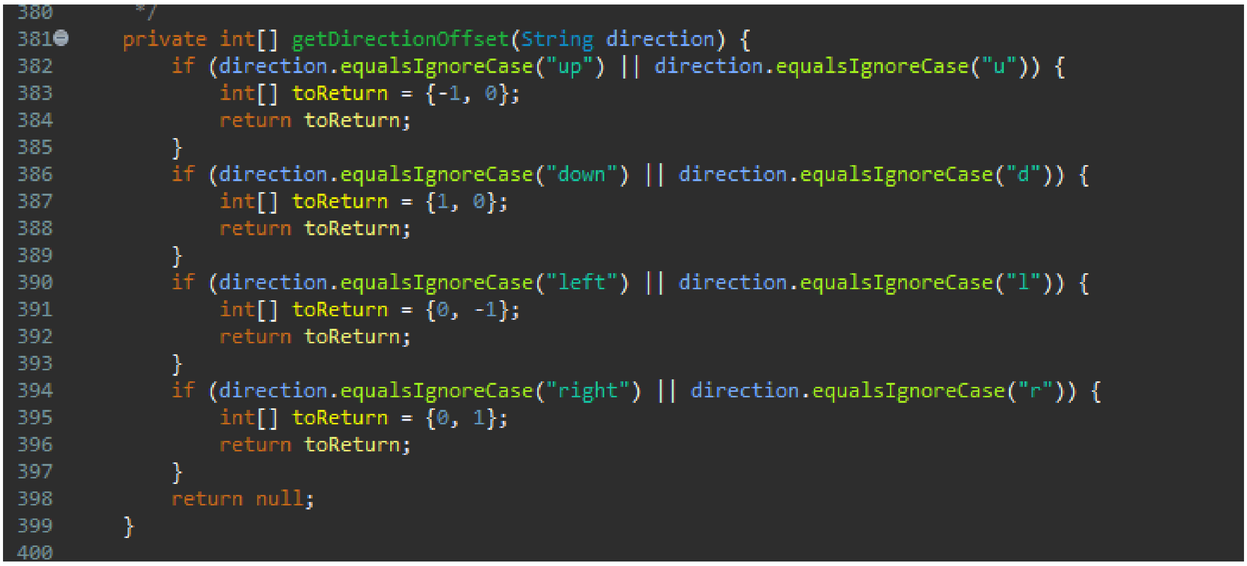
hasFreeSquare method:



```java
    private boolean hasFreeSquare(int row, int col, int offsetRow, int offsetCol) {
        // Can't select Water square or Empty Square
        if (grid[row][col].getType() == SquareType.WATER ||
            grid[row][col].getPiece() == null) {
            return false;
        }

        // Cannot go outside of boundaries of the game
        if (!isWithinBoundaries(row, col, offsetRow, offsetCol)) {
            return false;
        }

        // Look around
        if (grid[row+offsetRow][col+offsetCol].getType() != SquareType.WATER) {
            if (grid[row+offsetRow][col+offsetCol].getPiece() == null) {
                return true;
            }
            Piece p = grid[row][col].getPiece();
            Piece otherPiece = grid[row+offsetRow][col+offsetCol].getPiece();
            if(p.getPieceColor() != otherPiece.getPieceColor()) {
                return true;
            }
        }
        return false;
    }
```

The hasFreeSquare method is a boolean type method, it checks if there is a free square(free space) available of a square adjacent to a given position on the boards so that a piece can move into that position and potentially attack the enemy piece. The method makes sure that the space that has been selected is within the board boundaries and also the square is not a water cell. If the position does exist and is valid then true is returned otherwise, false is returned.

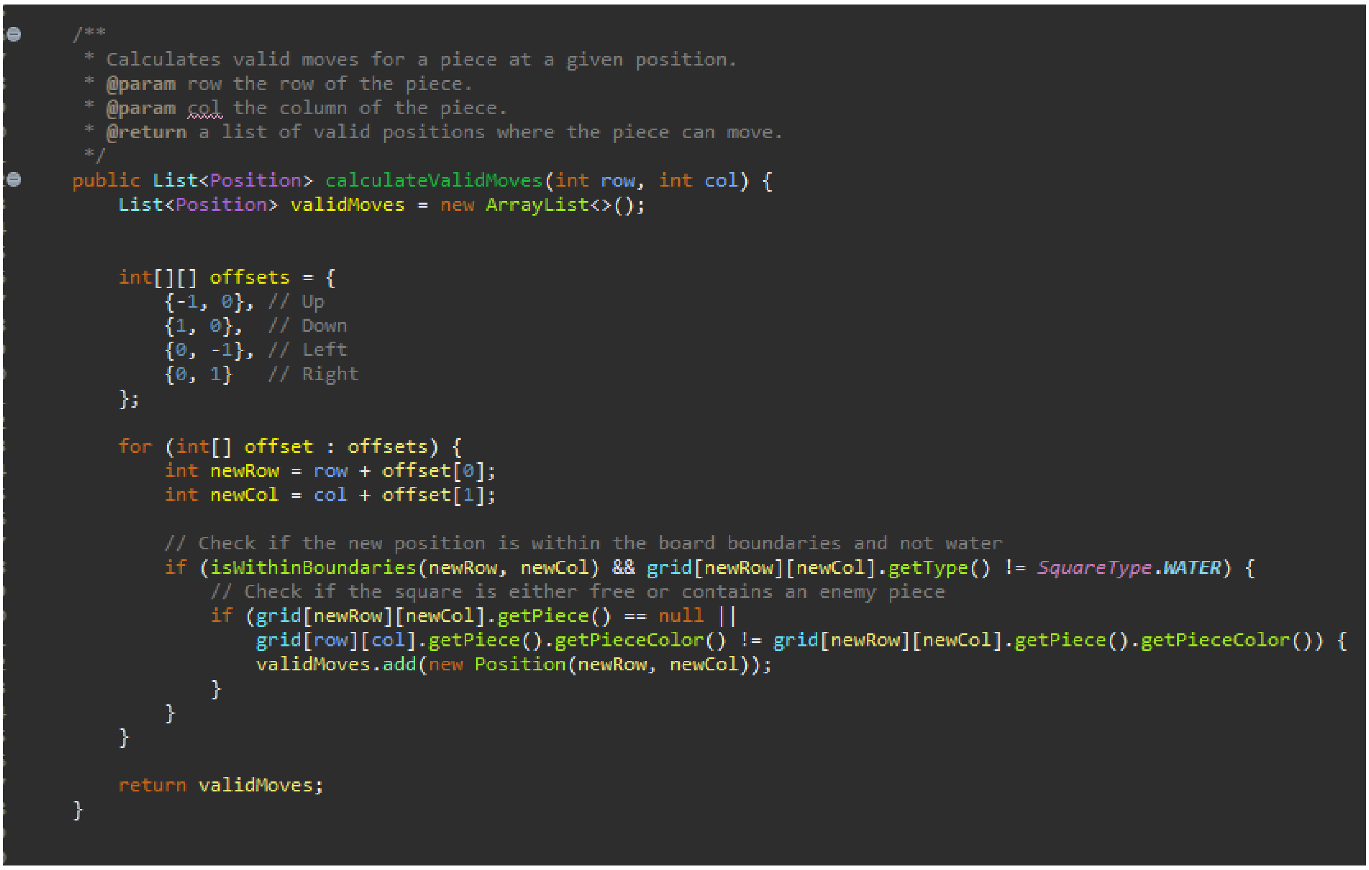
getDirectionOffset method:



```
     */
    private int[] getDirectionOffset(String direction) {
        if (direction.equalsIgnoreCase("up") || direction.equalsIgnoreCase("u")) {
            int[] toReturn = {-1, 0};
            return toReturn;
        }
        if (direction.equalsIgnoreCase("down") || direction.equalsIgnoreCase("d")) {
            int[] toReturn = {1, 0};
            return toReturn;
        }
        if (direction.equalsIgnoreCase("left") || direction.equalsIgnoreCase("l")) {
            int[] toReturn = {0, -1};
            return toReturn;
        }
        if (direction.equalsIgnoreCase("right") || direction.equalsIgnoreCase("r")) {
            int[] toReturn = {0, 1};
            return toReturn;
        }
        return null;
    }
```

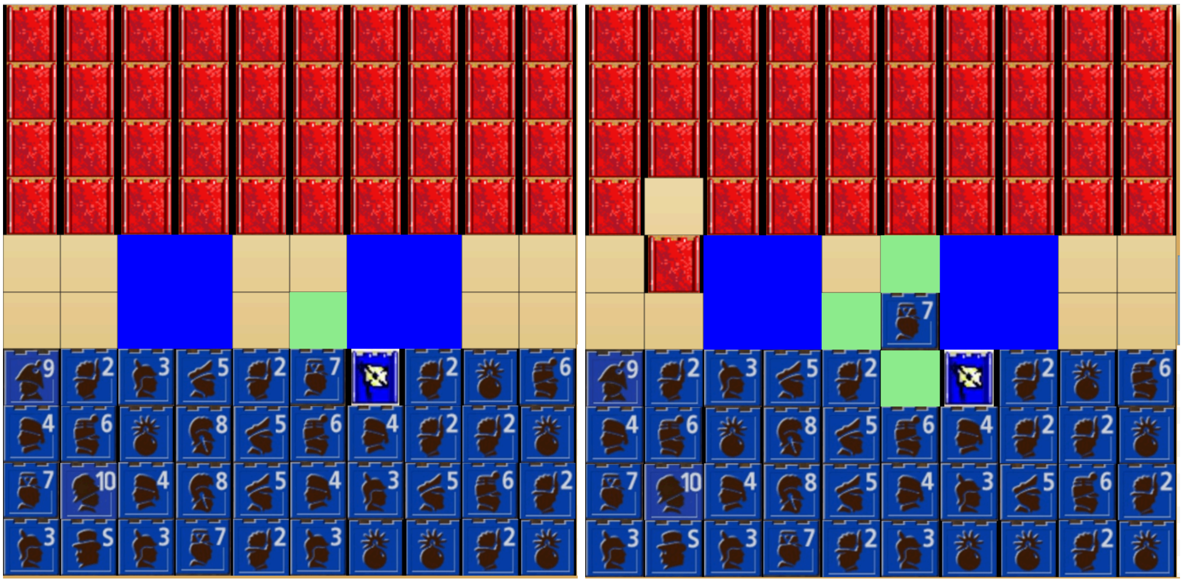
The getDirectionOffset method is a private method that returns an array which represents the change in direction in row and column. The direction Up is corresponded by {-1,0}, Down {1, 0} , Left is corresponded by {0, -1}, and Right is corresponded by {0, 1}.

calculateValidMoves method:



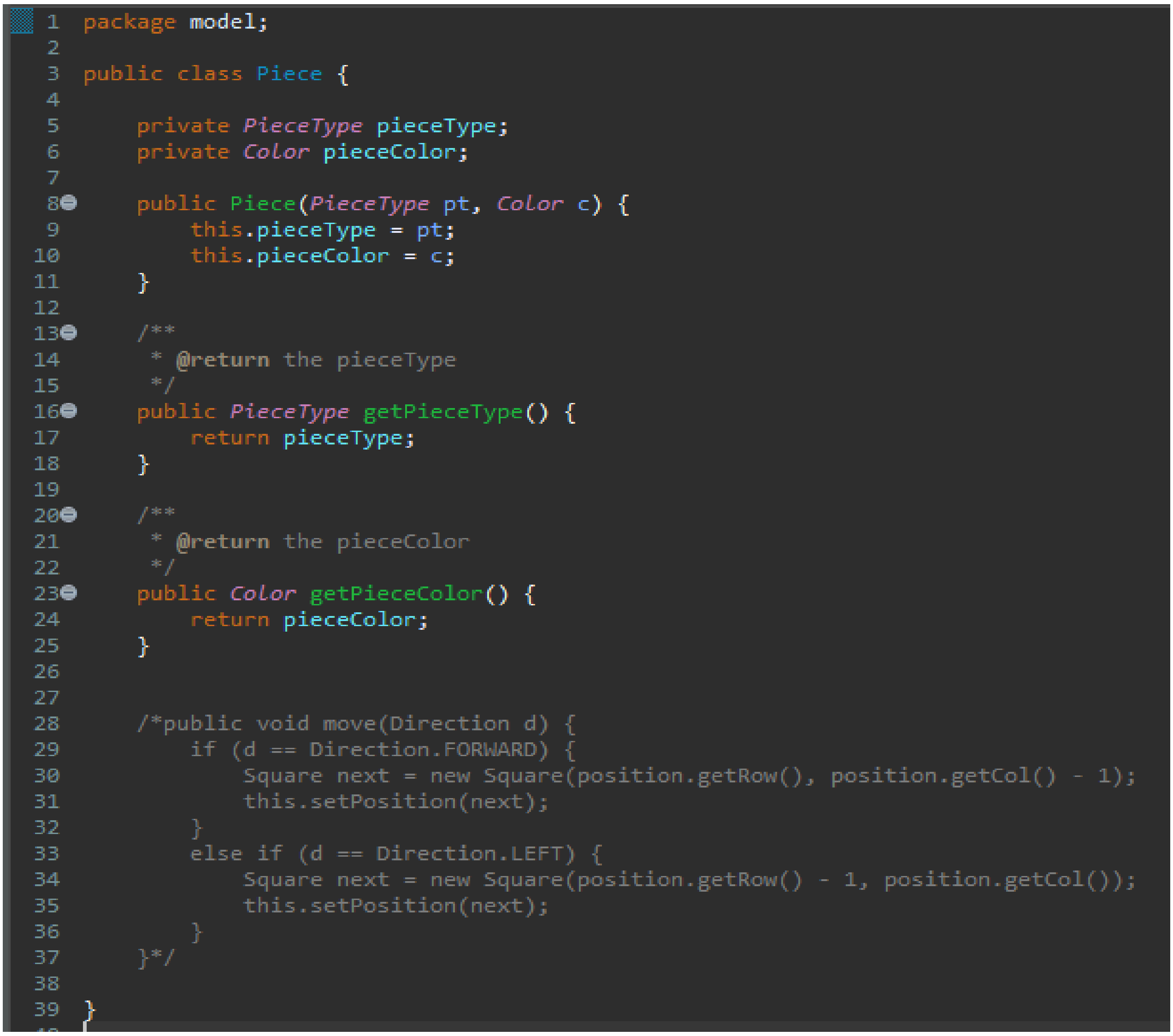
```
    /**
     * Calculates valid moves for a piece at a given position.
     * @param row the row of the piece.
     * @param col the column of the piece.
     * @return a list of valid positions where the piece can move.
     */
    public List<Position> calculateValidMoves(int row, int col) {
        List<Position> validMoves = new ArrayList<>();


        int[][] offsets = {
            {-1, 0}, // Up
            {1, 0},  // Down
            {0, -1}, // Left
            {0, 1}   // Right
        };

        for (int[] offset : offsets) {
            int newRow = row + offset[0];
            int newCol = col + offset[1];

            // Check if the new position is within the board boundaries and not water
            if (isWithinBoundaries(newRow, newCol) && grid[newRow][newCol].getType() != SquareType.WATER) {
                // Check if the square is either free or contains an enemy piece
                if (grid[newRow][newCol].getPiece() == null ||
                    grid[row][col].getPiece().getPieceColor() != grid[newRow][newCol].getPiece().getPieceColor()) {
                    validMoves.add(new Position(newRow, newCol));
                }
            }
        }

        return validMoves;
    }
```

The calculateValidMoves method returns and determines a list of all places/positions the selected piece can move to. The method checks all the four directions which are allowed (up, down, right, left) and if a position is available in the GUI it is displayed by green square(s) based on how many directions are available. This method is essential for the game logic as it enables calculation of every possible move a piece can make at any given point of the game, keeping in the rules and regulation of the game. The GUI representation can be seen in the image(s) below:



It can be seen in the images above when rank 7(major) of the blue player is selected all the possible moves have been displayed by a green/sea-green colour. In the image on the left it can be seen that since all the sides have been occupied and there is not any place other than in front of the major piece. In the image on the right hand side it can be seen that once the major piece moves up a square, all the available spaces/positions it can move to have been displayed, it is also important to notice that since there is a water cell on the right hand side, the piece cannot move in that direction.

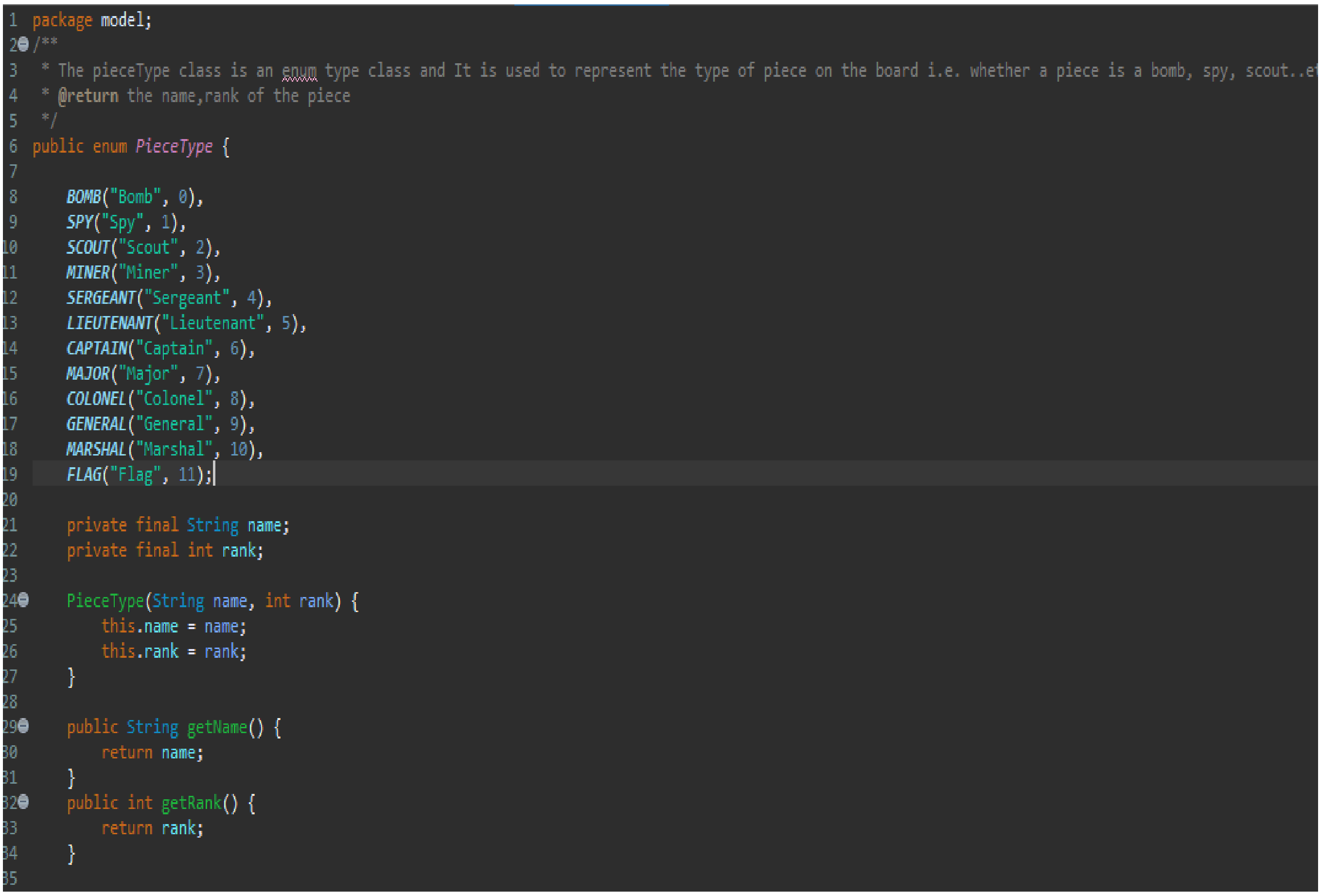
Piece class:



```
package model;

public class Piece {

    private PieceType pieceType;
    private Color pieceColor;

    public Piece(PieceType pt, Color c) {
        this.pieceType = pt;
        this.pieceColor = c;
    }

    /**
     * @return the pieceType
     */
    public PieceType getPieceType() {
        return pieceType;
    }

    /**
     * @return the pieceColor
     */
    public Color getPieceColor() {
        return pieceColor;
    }


    /*public void move(Direction d) {
        if (d == Direction.FORWARD) {
            Square next = new Square(position.getRow(), position.getCol() - 1);
            this.setPosition(next);
        }
        else if (d == Direction.LEFT) {
            Square next = new Square(position.getRow() - 1, position.getCol());
            this.setPosition(next);
        }
    }*/

}
```

The piece class is an important class as it used to represent the properties of a piece for the game. It can be seen that the piece consists of two properties that is PieceType(spy,bomb,marshal etc.) and of a colour which is associated with the player's team.

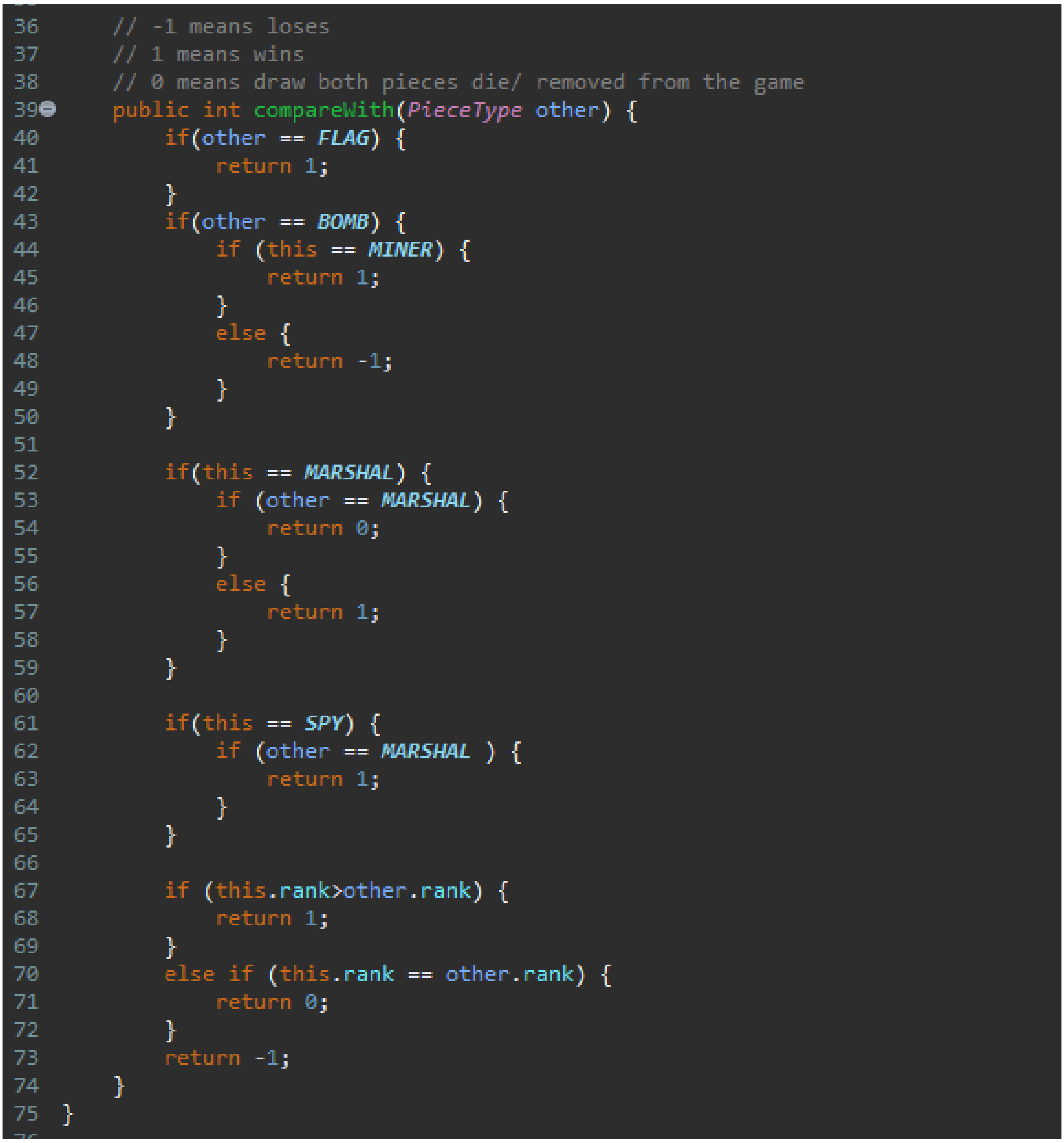
PieceType class:



```java
package model;
/**
 * The pieceType class is an enum type class and It is used to represent the type of piece on the board i.e. whether a piece is a bomb, spy, scout..et
 * @return the name,rank of the piece
 */
public enum PieceType {

    BOMB("Bomb", 0),
    SPY("Spy", 1),
    SCOUT("Scout", 2),
    MINER("Miner", 3),
    SERGEANT("Sergeant", 4),
    LIEUTENANT("Lieutenant", 5),
    CAPTAIN("Captain", 6),
    MAJOR("Major", 7),
    COLONEL("Colonel", 8),
    GENERAL("General", 9),
    MARSHAL("Marshal", 10),
    FLAG("Flag", 11);

    private final String name;
    private final int rank;

    PieceType(String name, int rank) {
        this.name = name;
        this.rank = rank;
    }

    public String getName() {
        return name;
    }
    public int getRank() {
        return rank;
    }

```

The pieceType class is an important class of this project as it is used to represent the pieces type such as Bomb, Spy etc . The class is an enum type and there are a limited number of pieces in the game given to each player. A pieceType consists of its name and a rank.

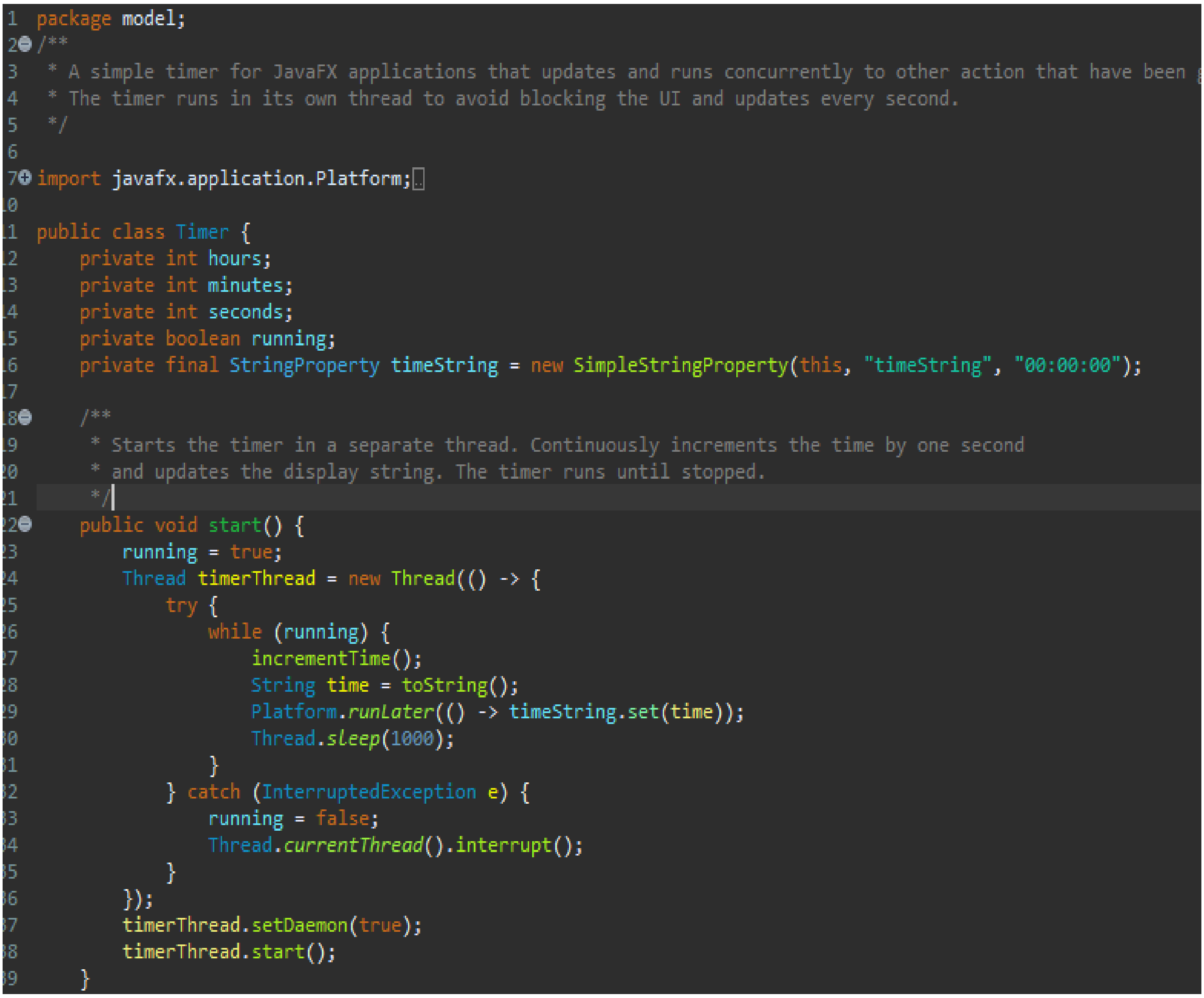
compareWith method:



```
    // -1 means loses
    // 1 means wins
    // 0 means draw both pieces die/ removed from the game
    public int compareWith(PieceType other) {
        if(other == FLAG) {
            return 1;
        }
        if(other == BOMB) {
            if (this == MINER) {
                return 1;
            }
            else {
                return -1;
            }
        }

        if(this == MARSHAL) {
            if (other == MARSHAL) {
                return 0;
            }
            else {
                return 1;
            }
        }

        if(this == SPY) {
            if (other == MARSHAL ) {
                return 1;
            }
        }

        if (this.rank>other.rank) {
            return 1;
        }
        else if (this.rank == other.rank) {
            return 0;
        }
        return -1;
    }
}
```

The compareWith method is used to compare the ranks of the pieces.It can be seen in the code above that if a piece captures a flag it means that the game is over and the player that captured the flag has won. The bomb kills all other pieces apart from the miner as it has special abilities. The marshal is the highest rank of all pieces and defeats all pieces apart from bombs and a spy. The method compares the rank of each piece and returns 1 if the piece has won, return -1 if the piece loses and return 0 if it's a draw, in case of a draw both pieces are eliminated from the board.

Timer Class:



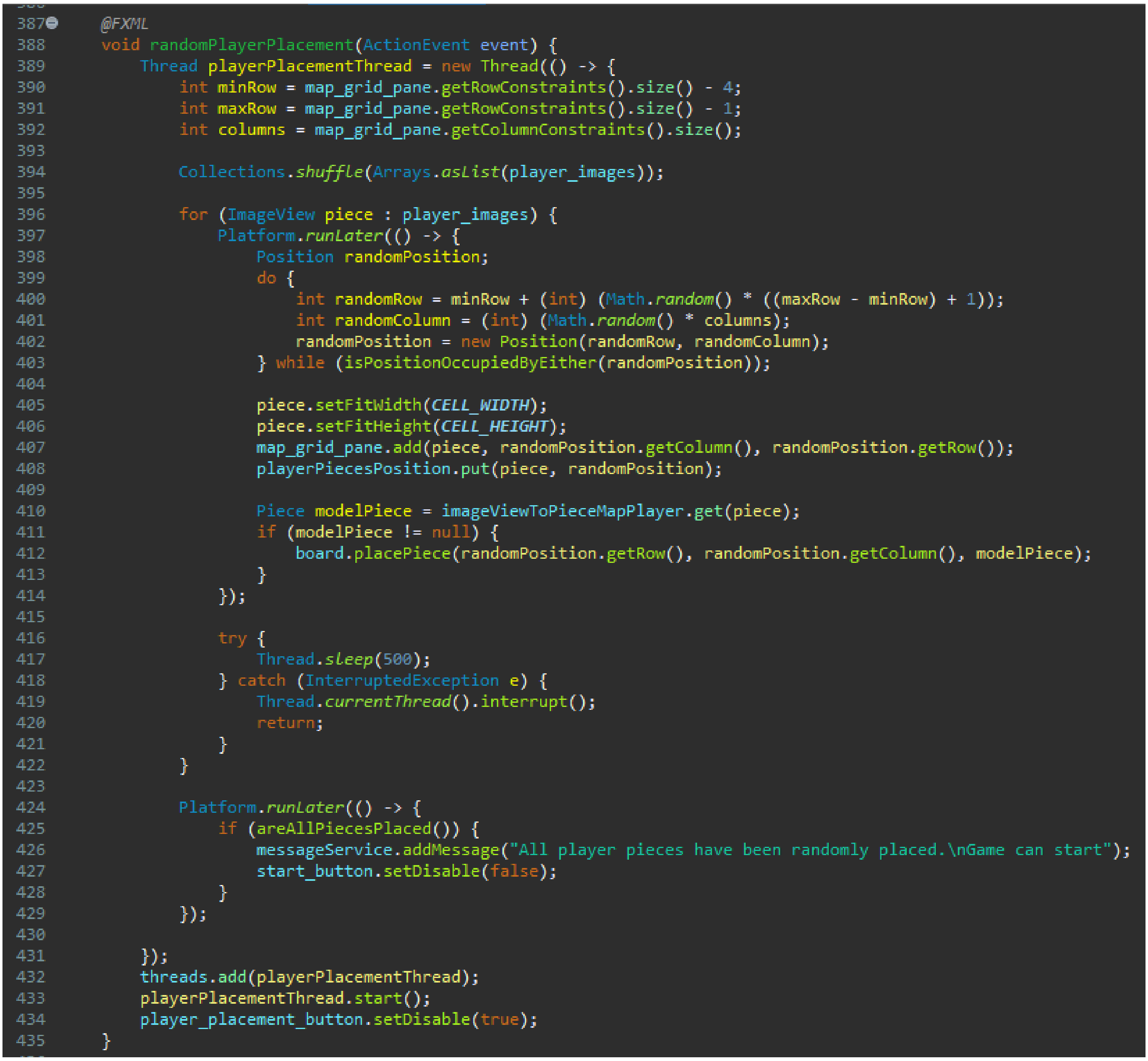
```java
package model;
/**
 * A simple timer for JavaFX applications that updates and runs concurrently to other action that have been
 * The timer runs in its own thread to avoid blocking the UI and updates every second.
 */

import javafx.application.Platform;

public class Timer {
    private int hours;
    private int minutes;
    private int seconds;
    private boolean running;
    private final StringProperty timeString = new SimpleStringProperty(this, "timeString", "00:00:00");

    /**
     * Starts the timer in a separate thread. Continuously increments the time by one second
     * and updates the display string. The timer runs until stopped.
     */
    public void start() {
        running = true;
        Thread timerThread = new Thread(() -> {
            try {
                while (running) {
                    incrementTime();
                    String time = toString();
                    Platform.runLater(() -> timeString.set(time));
                    Thread.sleep(1000);
                }
            } catch (InterruptedException e) {
                running = false;
                Thread.currentThread().interrupt();
            }
        });
        timerThread.setDaemon(true);
        timerThread.start();
    }
```

The timer class is an important class in the project as it allows the player to keep track of time and keep track of how the game session has been going on. The time in the game is represented by HH:MM:SS format. The timer class is important as it contains a thread in the start method and the thread is responsible for running the logic of the timer. The time is incremented every second of the game that is being played which means its running concurrently to other threads without interfering with other actions in the game and also without interfering with the main application thread.

StrategoController class:

The snippets below are from the StrategoController class which is one of the most important class in the project as Controls the gameplay logic for a Stratego game in a JavaFX application and manages the game state, including player and AI piece placement, turn handling, and user interactions through a graphical interface.

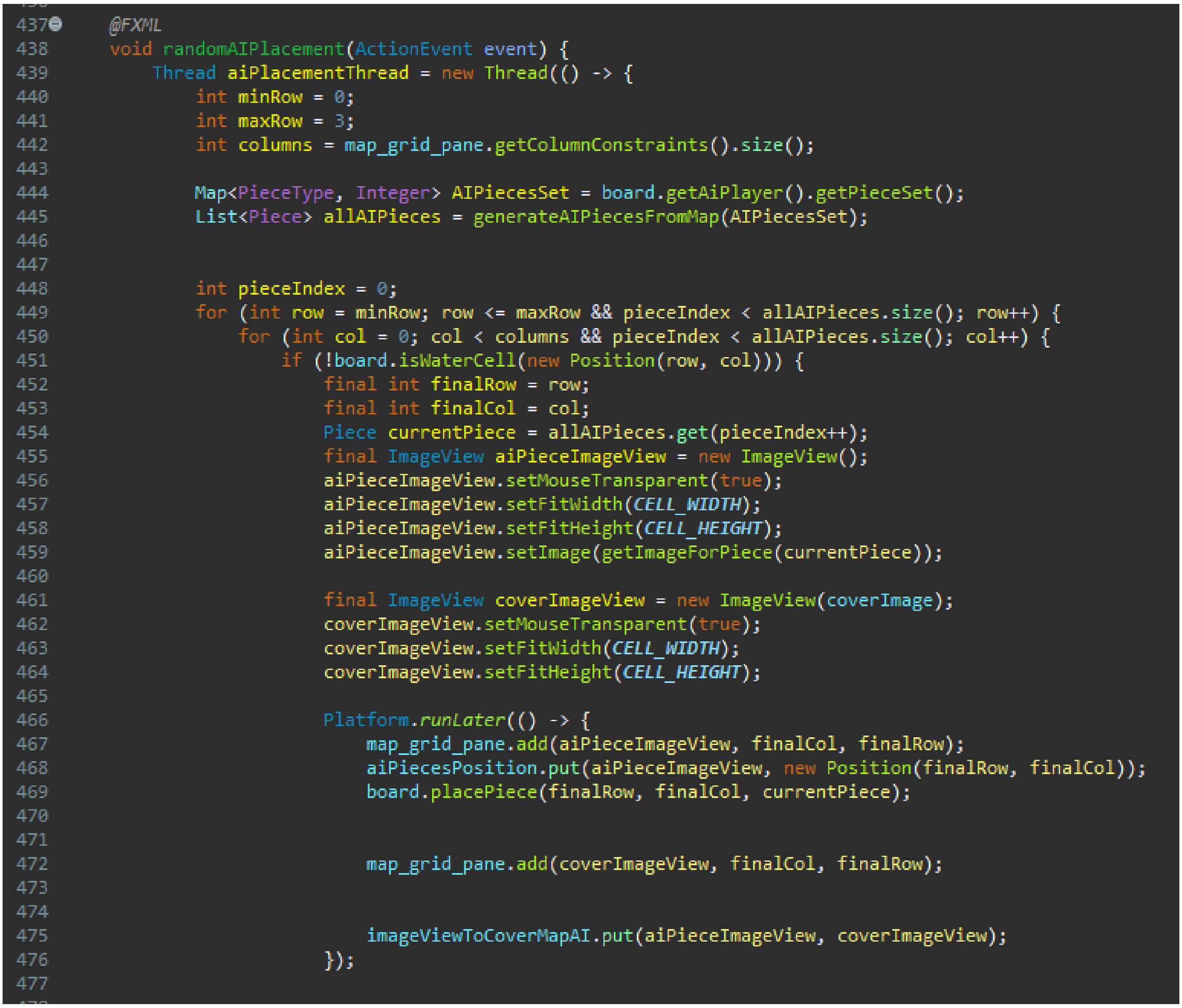
randomPlayerPlacement method:



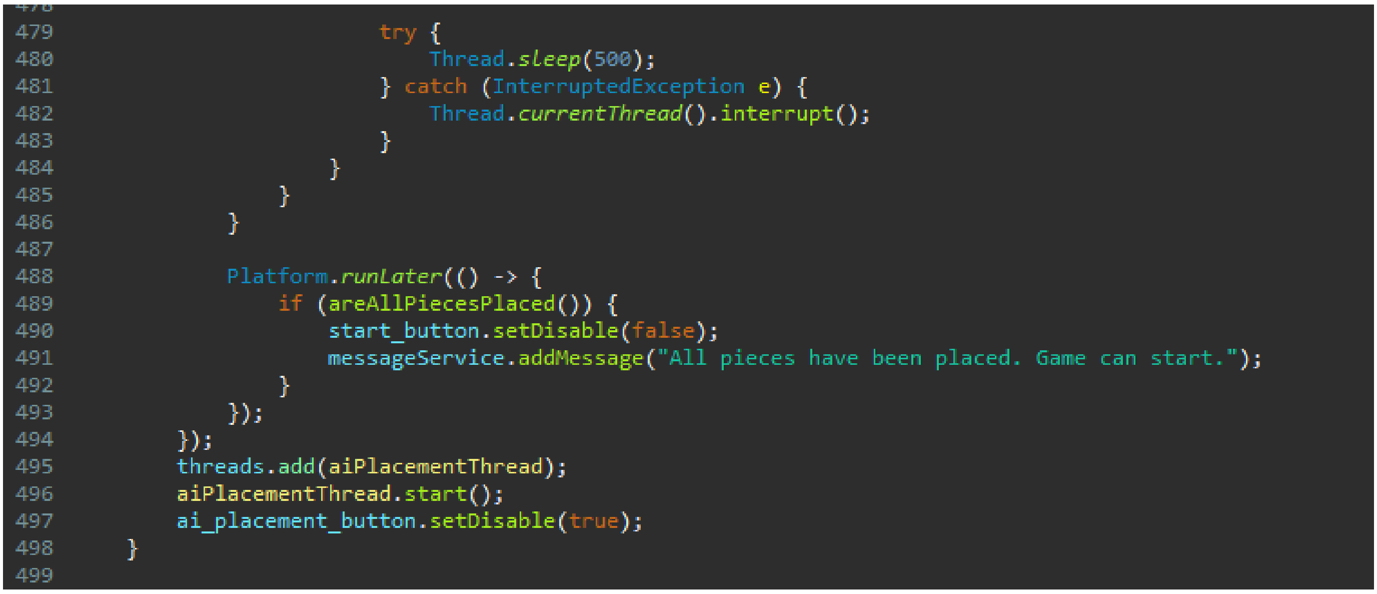
```
    @FXML
    void randomPlayerPlacement(ActionEvent event) {
        Thread playerPlacementThread = new Thread(() -> {
            int minRow = map_grid_pane.getRowConstraints().size() - 4;
            int maxRow = map_grid_pane.getRowConstraints().size() - 1;
            int columns = map_grid_pane.getColumnConstraints().size();

            Collections.shuffle(Arrays.asList(player_images));

            for (ImageView piece : player_images) {
                Platform.runLater(() -> {
                    Position randomPosition;
                    do {
                        int randomRow = minRow + (int) (Math.random() * ((maxRow - minRow) + 1));
                        int randomColumn = (int) (Math.random() * columns);
                        randomPosition = new Position(randomRow, randomColumn);
                    } while (isPositionOccupiedByEither(randomPosition));

                    piece.setFitWidth(CELL_WIDTH);
                    piece.setFitHeight(CELL_HEIGHT);
                    map_grid_pane.add(piece, randomPosition.getColumn(), randomPosition.getRow());
                    playerPiecesPosition.put(piece, randomPosition);

                    Piece modelPiece = imageViewToPieceMapPlayer.get(piece);
                    if (modelPiece != null) {
                        board.placePiece(randomPosition.getRow(), randomPosition.getColumn(), modelPiece);
                    }
                });

                try {
                    Thread.sleep(500);
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                    return;
                }
            }

            Platform.runLater(() -> {
                if (areAllPiecesPlaced()) {
                    messageService.addMessage("All player pieces have been randomly placed.\nGame can start");
                    start_button.setDisable(false);
                }
            });

        });
        threads.add(playerPlacementThread);
        playerPlacementThread.start();
        player_placement_button.setDisable(true);
    }
```

The randomPlayerPlacement method uses threading in order to place the player's pieces randomly on the board for the player without interfering with other ongoing game actions and most importantly without interfering or freezing the main application. By using threading in this method it allows the user interaction and other game actions to run smoothly while the pieces for the player randomly. This is why it was important for random player piece placement to run on a separate thread on its own. The method also delays when placing the pieces randomly piece by piece creating an aesthetic visual.

randomAIPlacement method:



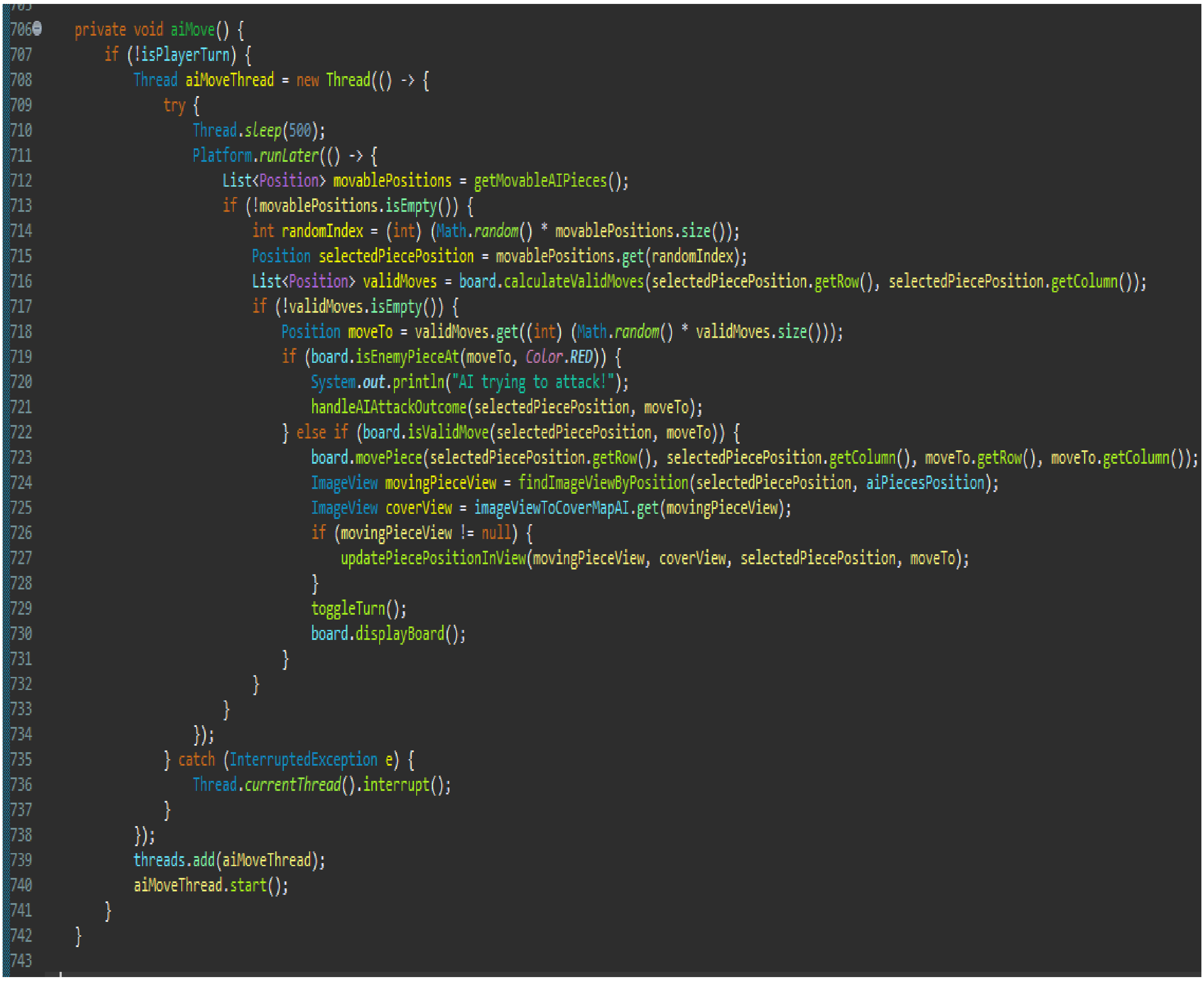
```
    @FXML
    void randomAIPlacement(ActionEvent event) {
        Thread aiPlacementThread = new Thread(() -> {
            int minRow = 0;
            int maxRow = 3;
            int columns = map_grid_pane.getColumnConstraints().size();

            Map<PieceType, Integer> AIPiecesSet = board.getAiPlayer().getPieceSet();
            List<Piece> allAIPieces = generateAIPiecesFromMap(AIPiecesSet);


            int pieceIndex = 0;
            for (int row = minRow; row <= maxRow && pieceIndex < allAIPieces.size(); row++) {
                for (int col = 0; col < columns && pieceIndex < allAIPieces.size(); col++) {
                    if (!board.isWaterCell(new Position(row, col))) {
                        final int finalRow = row;
                        final int finalCol = col;
                        Piece currentPiece = allAIPieces.get(pieceIndex++);
                        final ImageView aiPieceImageView = new ImageView();
                        aiPieceImageView.setMouseTransparent(true);
                        aiPieceImageView.setFitWidth(CELL_WIDTH);
                        aiPieceImageView.setFitHeight(CELL_HEIGHT);
                        aiPieceImageView.setImage(getImageForPiece(currentPiece));

                        final ImageView coverImageView = new ImageView(coverImage);
                        coverImageView.setMouseTransparent(true);
                        coverImageView.setFitWidth(CELL_WIDTH);
                        coverImageView.setFitHeight(CELL_HEIGHT);

                        Platform.runLater(() -> {
                            map_grid_pane.add(aiPieceImageView, finalCol, finalRow);
                            aiPiecesPosition.put(aiPieceImageView, new Position(finalRow, finalCol));
                            board.placePiece(finalRow, finalCol, currentPiece);


                            map_grid_pane.add(coverImageView, finalCol, finalRow);


                            imageViewToCoverMapAI.put(aiPieceImageView, coverImageView);
                        });

```



```
                        try {
                            Thread.sleep(500);
                        } catch (InterruptedException e) {
                            Thread.currentThread().interrupt();
                        }
                    }
                }
            }

            Platform.runLater(() -> {
                if (areAllPiecesPlaced()) {
                    start_button.setDisable(false);
                    messageService.addMessage("All pieces have been placed. Game can start.");
                }
            });
        });
        threads.add(aiPlacementThread);
        aiPlacementThread.start();
        ai_placement_button.setDisable(true);
    }
```

The randomAIPlacement method uses threading in order to place the AI's pieces randomly on the board for the player without interfering with other ongoing game actions and most importantly without interfering or freezing the main application. By using threading in this method it allows the user interaction and other game actions to run smoothly while the pieces for the AI randomly. This is why it was important for random player piece placement to run on a separate thread on its own. The method also delays when placing the pieces randomly piece by piece creating an aesthetic visual.

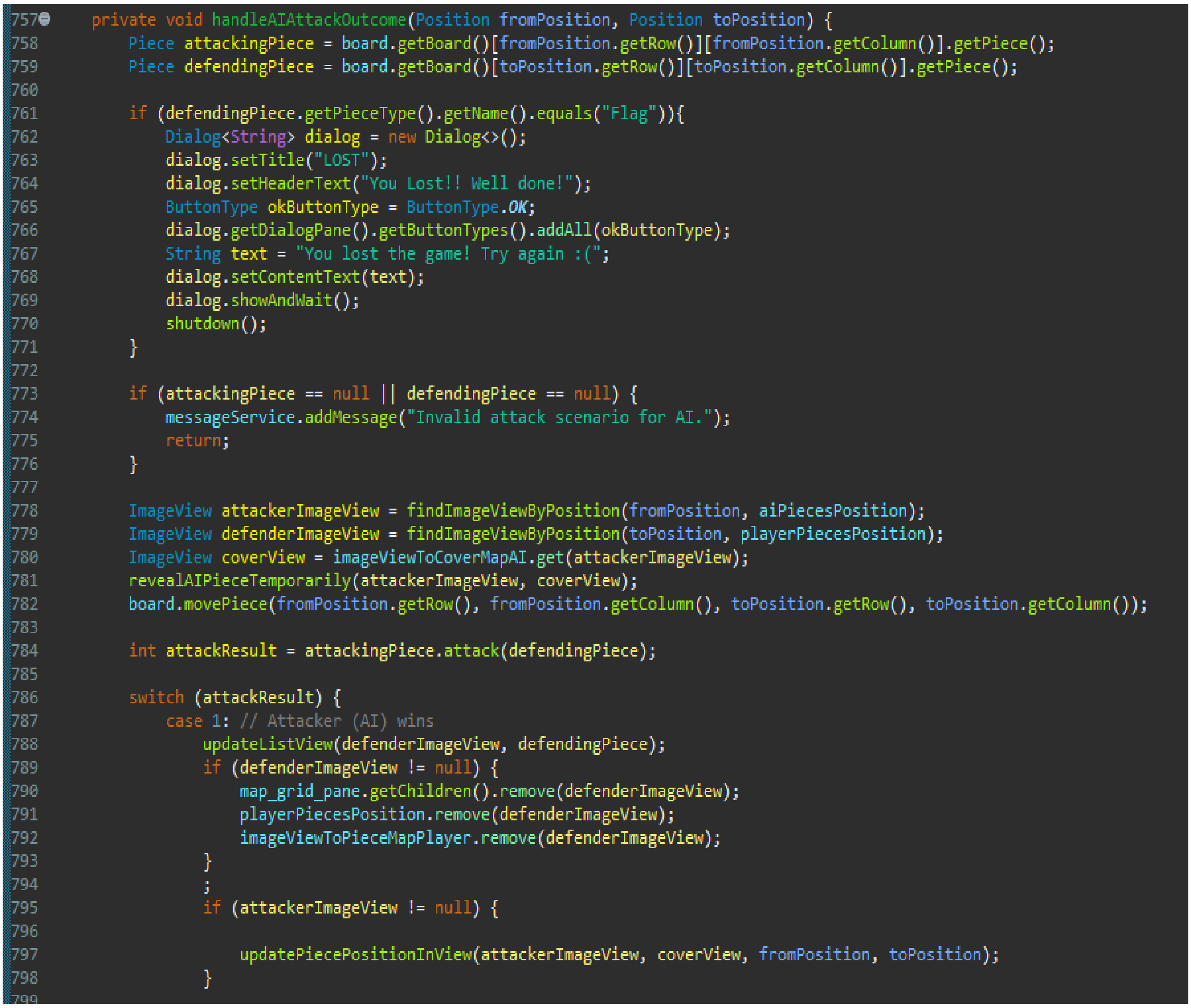
aiMove method:



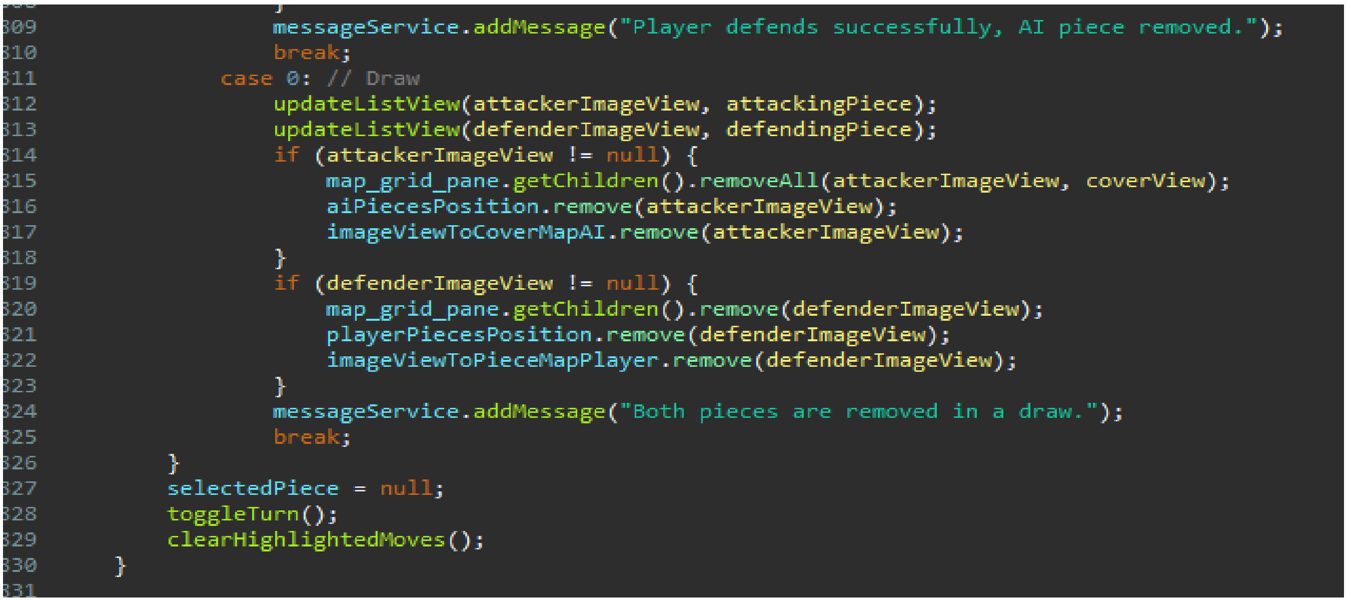
```
private void aiMove() {
    if (!isPlayerTurn) {
        Thread aiMoveThread = new Thread(() -> {
            try {
                Thread.sleep(500);
                Platform.runLater(() -> {
                    List<Position> movablePositions = getMovableAIPieces();
                    if (!movablePositions.isEmpty()) {
                        int randomIndex = (int) (Math.random() * movablePositions.size());
                        Position selectedPiecePosition = movablePositions.get(randomIndex);
                        List<Position> validMoves = board.calculateValidMoves(selectedPiecePosition.getRow(), selectedPiecePosition.getColumn());
                        if (!validMoves.isEmpty()) {
                            Position moveTo = validMoves.get((int) (Math.random() * validMoves.size()));
                            if (board.isEnemyPieceAt(moveTo, Color.RED)) {
                                System.out.println("AI trying to attack!");
                                handleAIAttackOutcome(selectedPiecePosition, moveTo);
                            } else if (board.isValidMove(selectedPiecePosition, moveTo)) {
                                board.movePiece(selectedPiecePosition.getRow(), selectedPiecePosition.getColumn(), moveTo.getRow(), moveTo.getColumn());
                                ImageView movingPieceView = findImageViewByPosition(selectedPiecePosition, aiPiecesPosition);
                                ImageView coverView = imageViewToCoverMapAI.get(movingPieceView);
                                if (movingPieceView != null) {
                                    updatePiecePositionInView(movingPieceView, coverView, selectedPiecePosition, moveTo);
                                }
                                toggleTurn();
                                board.displayBoard();
                            }
                        }
                    }
                });
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        });
        threads.add(aiMoveThread);
        aiMoveThread.start();
    }
}
```

The aiMove method is one of the most important methods as it handles the AI's gameplay by taking the AI's turn, the AI movement is done by selecting on the pieces available still to the AI in the game and either moving or attacking the enemy's piece. The AI runs on a separate thread on its own, it simulates a small pause which simulates that the AI is thinking before any decision is taken which makes the gameplay feel more human-like and natural. The method also ensures that the game play is smooth and flows nicely by alternating between the player and the AI turns.

handleAIAttackOutcome method:



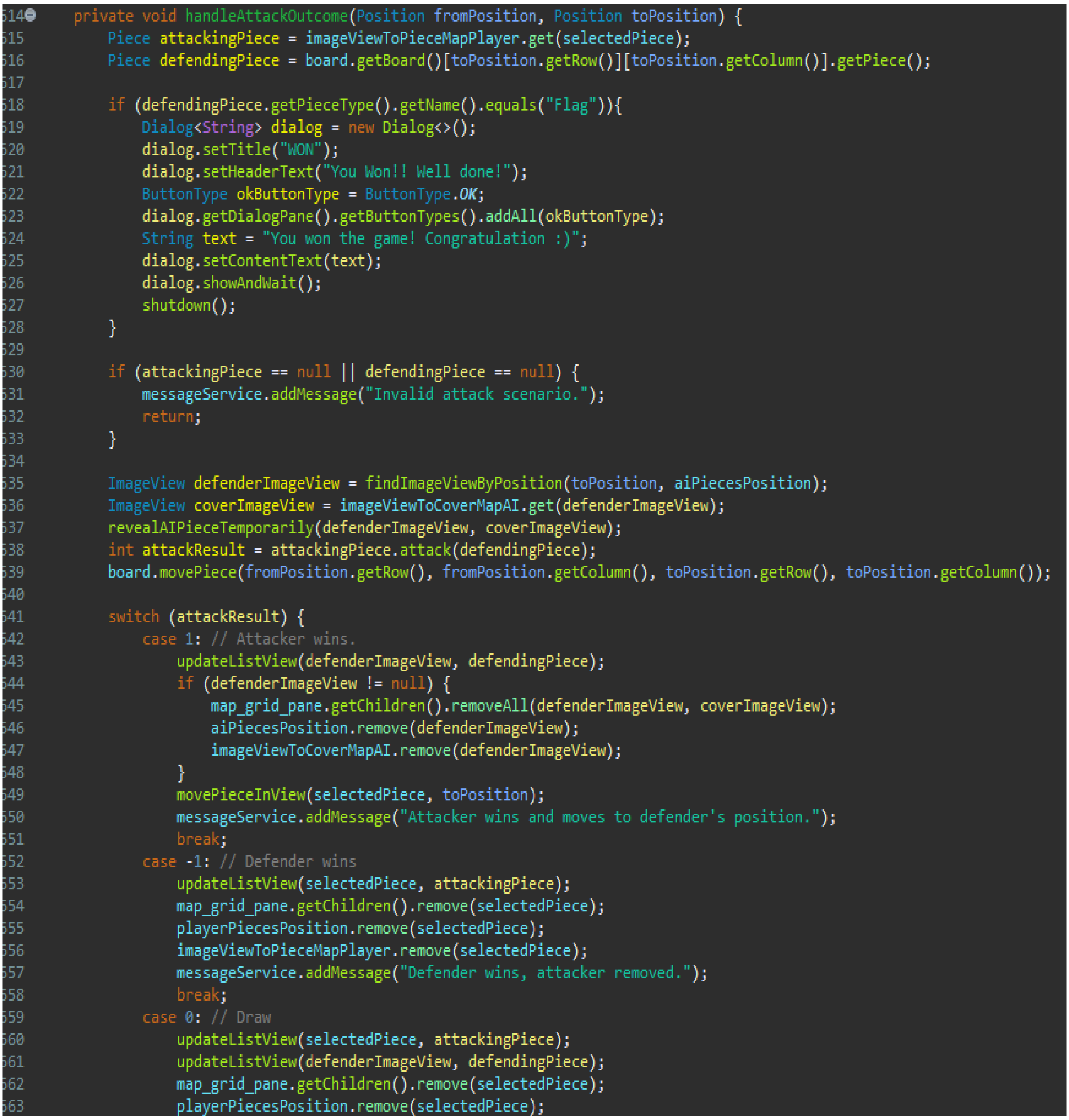
```
    private void handleAIAttackOutcome(Position fromPosition, Position toPosition) {
        Piece attackingPiece = board.getBoard()[fromPosition.getRow()][fromPosition.getColumn()].getPiece();
        Piece defendingPiece = board.getBoard()[toPosition.getRow()][toPosition.getColumn()].getPiece();

        if (defendingPiece.getPieceType().getName().equals("Flag")){
            Dialog<String> dialog = new Dialog<>();
            dialog.setTitle("LOST");
            dialog.setHeaderText("You Lost!! Well done!");
            ButtonType okButtonType = ButtonType.OK;
            dialog.getDialogPane().getButtonTypes().addAll(okButtonType);
            String text = "You lost the game! Try again :(";
            dialog.setContentText(text);
            dialog.showAndWait();
            shutdown();
        }

        if (attackingPiece == null || defendingPiece == null) {
            messageService.addMessage("Invalid attack scenario for AI.");
            return;
        }

        ImageView attackerImageView = findImageViewByPosition(fromPosition, aiPiecesPosition);
        ImageView defenderImageView = findImageViewByPosition(toPosition, playerPiecesPosition);
        ImageView coverView = imageViewToCoverMapAI.get(attackerImageView);
        revealAIPieceTemporarily(attackerImageView, coverView);
        board.movePiece(fromPosition.getRow(), fromPosition.getColumn(), toPosition.getRow(), toPosition.getColumn());

        int attackResult = attackingPiece.attack(defendingPiece);

        switch (attackResult) {
            case 1: // Attacker (AI) wins
                updateListView(defenderImageView, defendingPiece);
                if (defenderImageView != null) {
                    map_grid_pane.getChildren().remove(defenderImageView);
                    playerPiecesPosition.remove(defenderImageView);
                    imageViewToPieceMapPlayer.remove(defenderImageView);
                }
                ;
                if (attackerImageView != null) {

                    updatePiecePositionInView(attackerImageView, coverView, fromPosition, toPosition);
                }

```



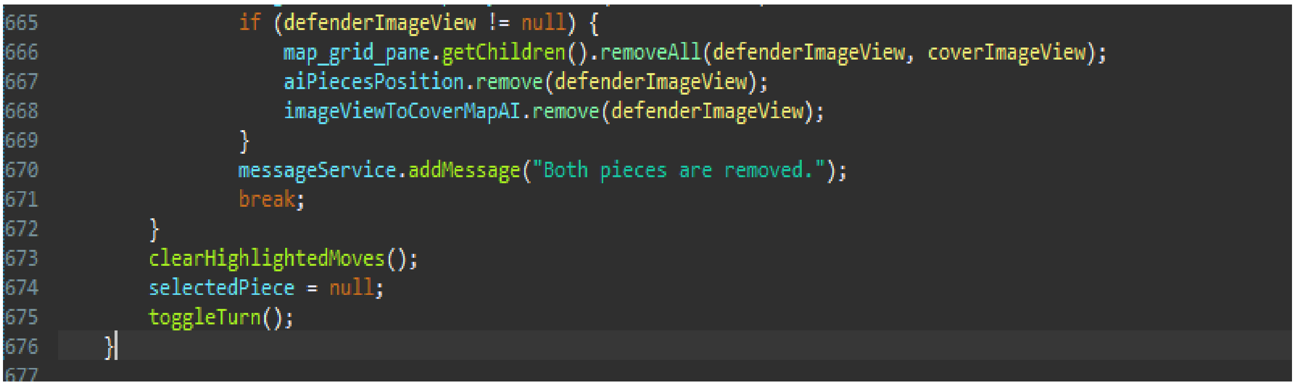
```
                }
                messageService.addMessage("Player defends successfully, AI piece removed.");
                break;
            case 0: // Draw
                updateListView(attackerImageView, attackingPiece);
                updateListView(defenderImageView, defendingPiece);
                if (attackerImageView != null) {
                    map_grid_pane.getChildren().removeAll(attackerImageView, coverView);
                    aiPiecesPosition.remove(attackerImageView);
                    imageViewToCoverMapAI.remove(attackerImageView);
                }
                if (defenderImageView != null) {
                    map_grid_pane.getChildren().remove(defenderImageView);
                    playerPiecesPosition.remove(defenderImageView);
                    imageViewToPieceMapPlayer.remove(defenderImageView);
                }
                messageService.addMessage("Both pieces are removed in a draw.");
                break;
        }
        selectedPiece = null;
        toggleTurn();
        clearHighlightedMoves();
    }

```

The handleAIAttackOutcome method decides the result of an attack between an AI piece and a player piece from the AI's perspective. When the AI’s piece attack’s the player piece or when the AI defends against the player, this method calculates the result of the battle, whether the AI has won, lost or drawn. The method also regularly keeps updating the game board once a piece has been killed/captured by the opponent. The method also checks that the game is being played according to the rules which maintains fairness and the integrity of the game.

handleAttackOutcome method:



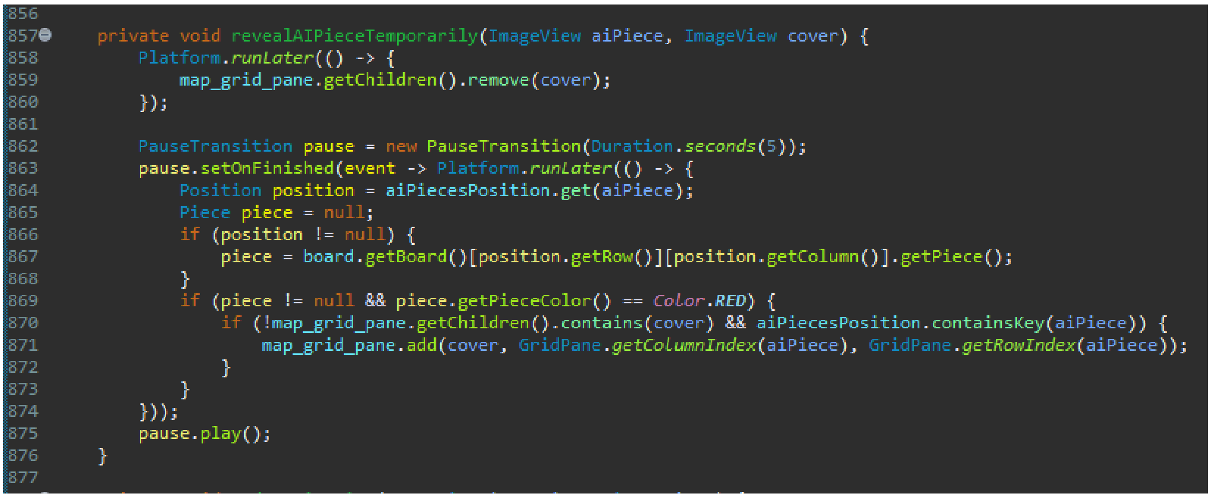
```
    private void handleAttackOutcome(Position fromPosition, Position toPosition) {
        Piece attackingPiece = imageViewToPieceMapPlayer.get(selectedPiece);
        Piece defendingPiece = board.getBoard()[toPosition.getRow()][toPosition.getColumn()].getPiece();

        if (defendingPiece.getPieceType().getName().equals("Flag")){
            Dialog<String> dialog = new Dialog<>();
            dialog.setTitle("WON");
            dialog.setHeaderText("You Won!! Well done!");
            ButtonType okButtonType = ButtonType.OK;
            dialog.getDialogPane().getButtonTypes().addAll(okButtonType);
            String text = "You won the game! Congratulation :)";
            dialog.setContentText(text);
            dialog.showAndWait();
            shutdown();
        }

        if (attackingPiece == null || defendingPiece == null) {
            messageService.addMessage("Invalid attack scenario.");
            return;
        }

        ImageView defenderImageView = findImageViewByPosition(toPosition, aiPiecesPosition);
        ImageView coverImageView = imageViewToCoverMapAI.get(defenderImageView);
        revealAIPieceTemporarily(defenderImageView, coverImageView);
        int attackResult = attackingPiece.attack(defendingPiece);
        board.movePiece(fromPosition.getRow(), fromPosition.getColumn(), toPosition.getRow(), toPosition.getColumn());

        switch (attackResult) {
            case 1: // Attacker wins.
                updateListView(defenderImageView, defendingPiece);
                if (defenderImageView != null) {
                    map_grid_pane.getChildren().removeAll(defenderImageView, coverImageView);
                    aiPiecesPosition.remove(defenderImageView);
                    imageViewToCoverMapAI.remove(defenderImageView);
                }
                movePieceInView(selectedPiece, toPosition);
                messageService.addMessage("Attacker wins and moves to defender's position.");
                break;
            case -1: // Defender wins
                updateListView(selectedPiece, attackingPiece);
                map_grid_pane.getChildren().remove(selectedPiece);
                playerPiecesPosition.remove(selectedPiece);
                imageViewToPieceMapPlayer.remove(selectedPiece);
                messageService.addMessage("Defender wins, attacker removed.");
                break;
            case 0: // Draw
                updateListView(selectedPiece, attackingPiece);
                updateListView(defenderImageView, defendingPiece);
                map_grid_pane.getChildren().remove(selectedPiece);
                playerPiecesPosition.remove(selectedPiece);
```



```
                if (defenderImageView != null) {
                    map_grid_pane.getChildren().removeAll(defenderImageView, coverImageView);
                    aiPiecesPosition.remove(defenderImageView);
                    imageViewToCoverMapAI.remove(defenderImageView);
                }
                messageService.addMessage("Both pieces are removed.");
                break;
        }
        clearHighlightedMoves();
        selectedPiece = null;
        toggleTurn();
    }
```

The handleAttackOutcome method decides the result of an attack between an AI piece and a player piece from the player's perspective. When the player’s piece attack’s the AI piece or when the player defends against the AI, this method calculates the result of the battle, whether the player has won, lost or drawn. The method also regularly keeps updating the game board once a piece has been killed/captured by the opponent. The method also checks that the game is being played according to the rules which maintains fairness and the integrity of the game.

revealAIPieceTemporarily method:

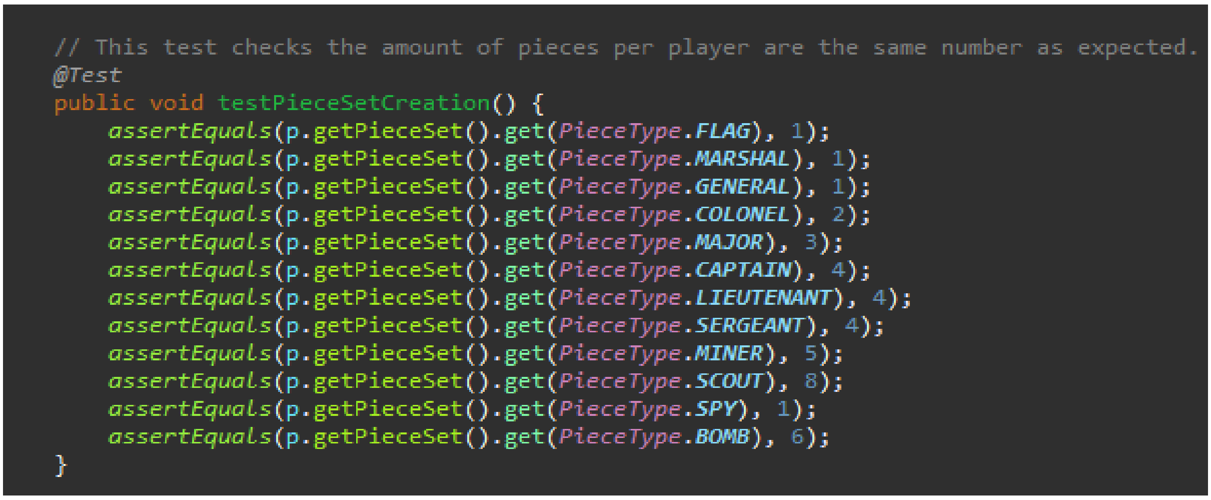


```java
    private void revealAIPieceTemporarily(ImageView aiPiece, ImageView cover) {
        Platform.runLater(() -> {
            map_grid_pane.getChildren().remove(cover);
        });

        PauseTransition pause = new PauseTransition(Duration.seconds(5));
        pause.setOnFinished(event -> Platform.runLater(() -> {
            Position position = aiPiecesPosition.get(aiPiece);
            Piece piece = null;
            if (position != null) {
                piece = board.getBoard()[position.getRow()][position.getColumn()].getPiece();
            }
            if (piece != null && piece.getPieceColor() == Color.RED) {
                if (!map_grid_pane.getChildren().contains(cover) && aiPiecesPosition.containsKey(aiPiece)) {
                    map_grid_pane.add(cover, GridPane.getColumnIndex(aiPiece), GridPane.getRowIndex(aiPiece));
                }
            }
        }));
        pause.play();
    }
```

The revealAIPieceTemporarily method temporarily uncovers the AI’s piece so that the player can see what piece it was killed by. This only occurs when the AI attacks and kills the player’s piece. The piece that has killed the player is unveiled for 5 seconds before hiding itself again. This element of the game adds a sense of mystery as the player cannot see the AI’s piece.

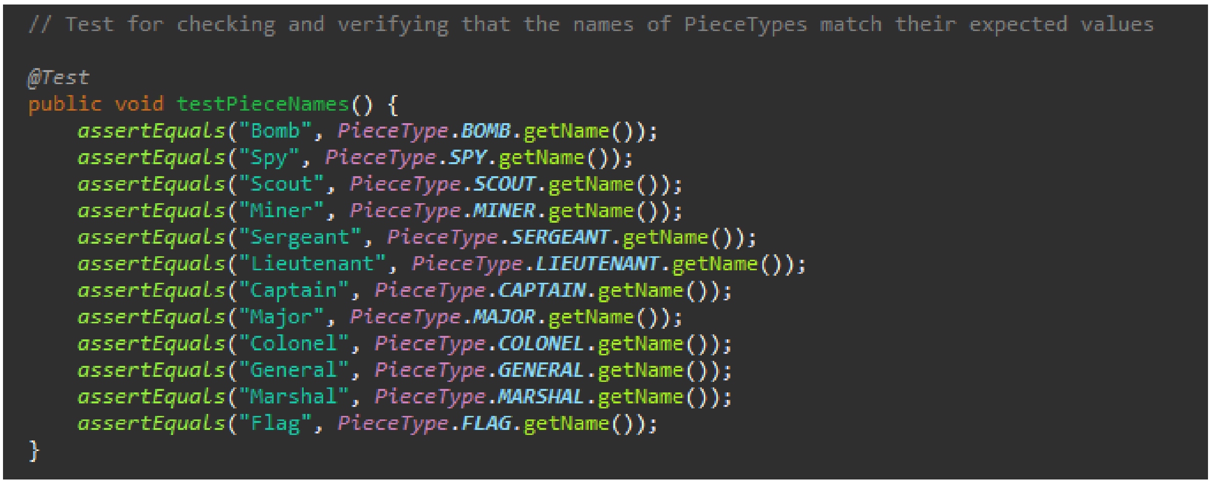
3.3 Testing

Testing is an essential part of this project as it's important to ensure that the functionalities added work as intended and in order to make sure that they work as intended thorough testing needs to be carried for each functionality developed.This section of the report contains snippets from test cases that were used to test the game logic and other functionalities.



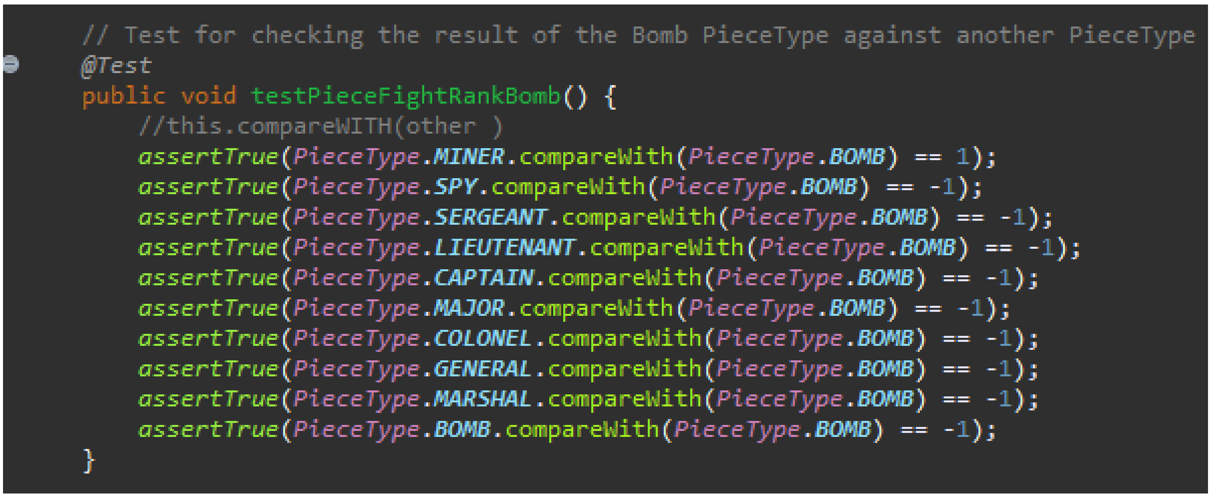
```
// This test checks the amount of pieces per player are the same number as expected.
@Test
public void testPieceSetCreation() {
    assertEquals(p.getPieceSet().get(PieceType.FLAG), 1);
    assertEquals(p.getPieceSet().get(PieceType.MARSHAL), 1);
    assertEquals(p.getPieceSet().get(PieceType.GENERAL), 1);
    assertEquals(p.getPieceSet().get(PieceType.COLONEL), 2);
    assertEquals(p.getPieceSet().get(PieceType.MAJOR), 3);
    assertEquals(p.getPieceSet().get(PieceType.CAPTAIN), 4);
    assertEquals(p.getPieceSet().get(PieceType.LIEUTENANT), 4);
    assertEquals(p.getPieceSet().get(PieceType.SERGEANT), 4);
    assertEquals(p.getPieceSet().get(PieceType.MINER), 5);
    assertEquals(p.getPieceSet().get(PieceType.SCOUT), 8);
    assertEquals(p.getPieceSet().get(PieceType.SPY), 1);
    assertEquals(p.getPieceSet().get(PieceType.BOMB), 6);
}
```

This test case checks that each piece set contains the expected amount of piece types per player. The ‘assertEquals’ statement in this test case is used to verify that the amount of piece types in ‘PieceSet’ class matches the expected amount of pieces per player.



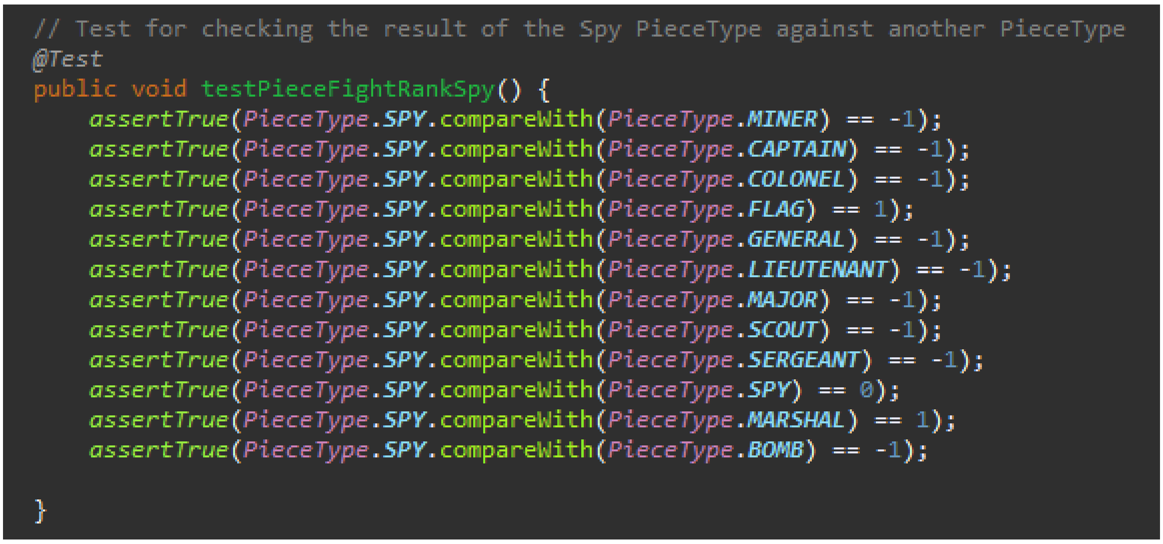
```
// Test for checking and verifying that the names of PieceTypes match their expected values

@Test
public void testPieceNames() {
    assertEquals("Bomb", PieceType.BOMB.getName());
    assertEquals("Spy", PieceType.SPY.getName());
    assertEquals("Scout", PieceType.SCOUT.getName());
    assertEquals("Miner", PieceType.MINER.getName());
    assertEquals("Sergeant", PieceType.SERGEANT.getName());
    assertEquals("Lieutenant", PieceType.LIEUTENANT.getName());
    assertEquals("Captain", PieceType.CAPTAIN.getName());
    assertEquals("Major", PieceType.MAJOR.getName());
    assertEquals("Colonel", PieceType.COLONEL.getName());
    assertEquals("General", PieceType.GENERAL.getName());
    assertEquals("Marshal", PieceType.MARSHAL.getName());
    assertEquals("Flag", PieceType.FLAG.getName());
}
```

This test case checks and verifies that the expected value of the PieceType matches the actual value using the getName() method. The ‘assertEquals’ statement in this test case is used to verify that the name of piece types in ‘PieceType’ matches the expected name.



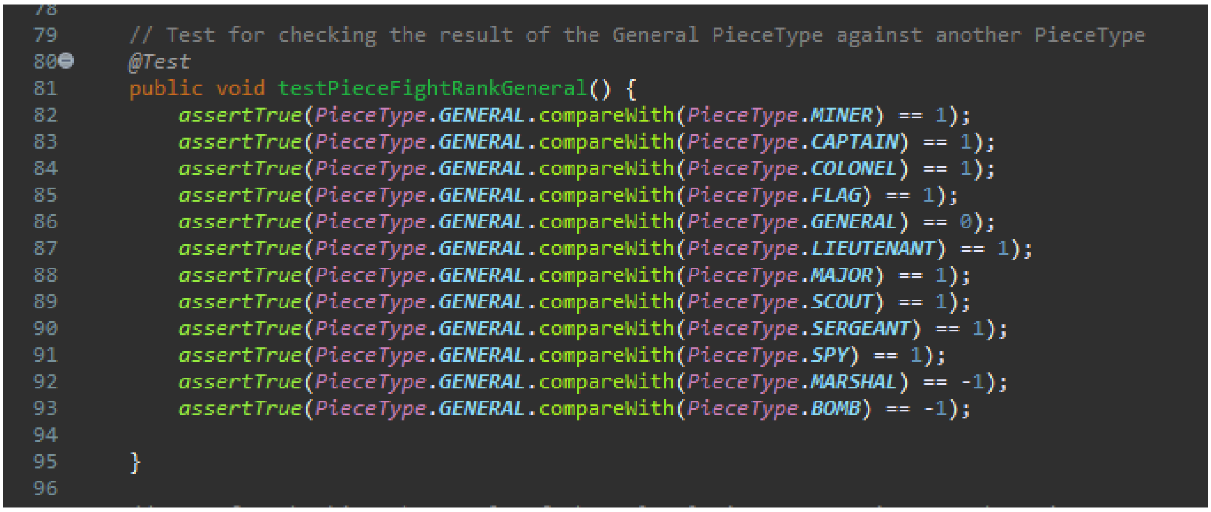
```
    // Test for checking the result of the Bomb PieceType against another PieceType
    @Test
    public void testPieceFightRankBomb() {
        //this.compareWITH(other )
        assertTrue(PieceType.MINER.compareWith(PieceType.BOMB) == 1);
        assertTrue(PieceType.SPY.compareWith(PieceType.BOMB) == -1);
        assertTrue(PieceType.SERGEANT.compareWith(PieceType.BOMB) == -1);
        assertTrue(PieceType.LIEUTENANT.compareWith(PieceType.BOMB) == -1);
        assertTrue(PieceType.CAPTAIN.compareWith(PieceType.BOMB) == -1);
        assertTrue(PieceType.MAJOR.compareWith(PieceType.BOMB) == -1);
        assertTrue(PieceType.COLONEL.compareWith(PieceType.BOMB) == -1);
        assertTrue(PieceType.GENERAL.compareWith(PieceType.BOMB) == -1);
        assertTrue(PieceType.MARSHAL.compareWith(PieceType.BOMB) == -1);
        assertTrue(PieceType.BOMB.compareWith(PieceType.BOMB) == -1);
    }
```

This test case checks the result of the ‘bomb’ PieceType against other piecetype as it can be seen that the bomb is only defeated if a miner attacks it as the miner is a special piece in the game and has the ability to defeat the bomb. The bomb defeats all other pieces, as the bomb is a static piece it therefore does not move. The assertTrue is used to verify and compare that the result of the ‘bomb’ piecetype against other pieces is as expected.



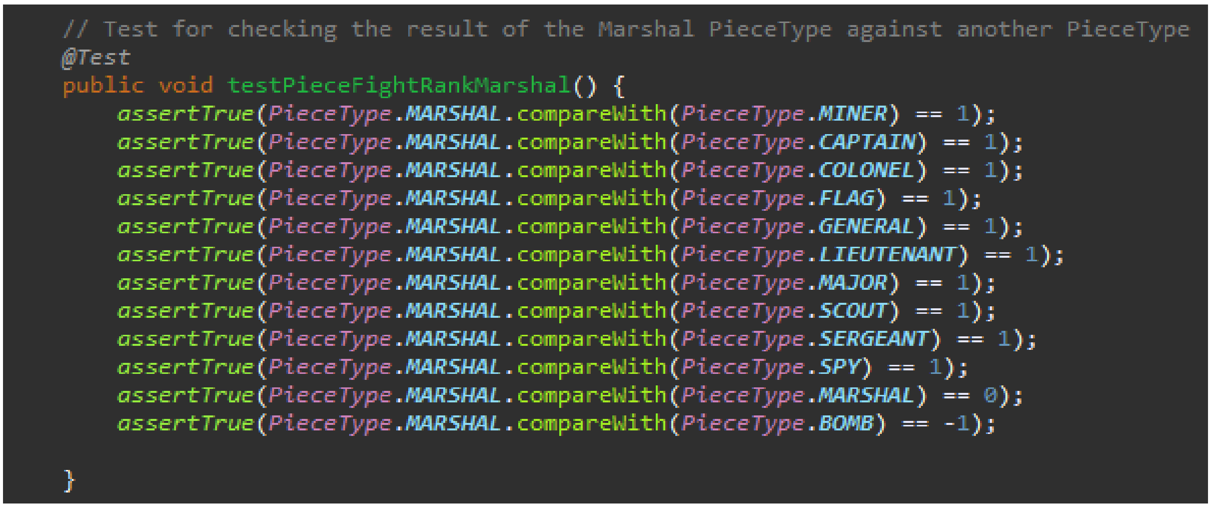
```
// Test for checking the result of the Spy PieceType against another PieceType
@Test
public void testPieceFightRankSpy() {
    assertTrue(PieceType.SPY.compareWith(PieceType.MINER) == -1);
    assertTrue(PieceType.SPY.compareWith(PieceType.CAPTAIN) == -1);
    assertTrue(PieceType.SPY.compareWith(PieceType.COLONEL) == -1);
    assertTrue(PieceType.SPY.compareWith(PieceType.FLAG) == 1);
    assertTrue(PieceType.SPY.compareWith(PieceType.GENERAL) == -1);
    assertTrue(PieceType.SPY.compareWith(PieceType.LIEUTENANT) == -1);
    assertTrue(PieceType.SPY.compareWith(PieceType.MAJOR) == -1);
    assertTrue(PieceType.SPY.compareWith(PieceType.SCOUT) == -1);
    assertTrue(PieceType.SPY.compareWith(PieceType.SERGEANT) == -1);
    assertTrue(PieceType.SPY.compareWith(PieceType.SPY) == 0);
    assertTrue(PieceType.SPY.compareWith(PieceType.MARSHAL) == 1);
    assertTrue(PieceType.SPY.compareWith(PieceType.BOMB) == -1);

}
```

This test case checks the result of the ‘Spy’ PieceType against other piecetype as it can be seen that the Spy is defeated by every piece apart from when it attacks the marshal but if a marshal attacks the spy and the spy is defeated. The assertTrue is used to verify and compare that the result of the ‘Spy’ piecetype against other pieces is as expected.



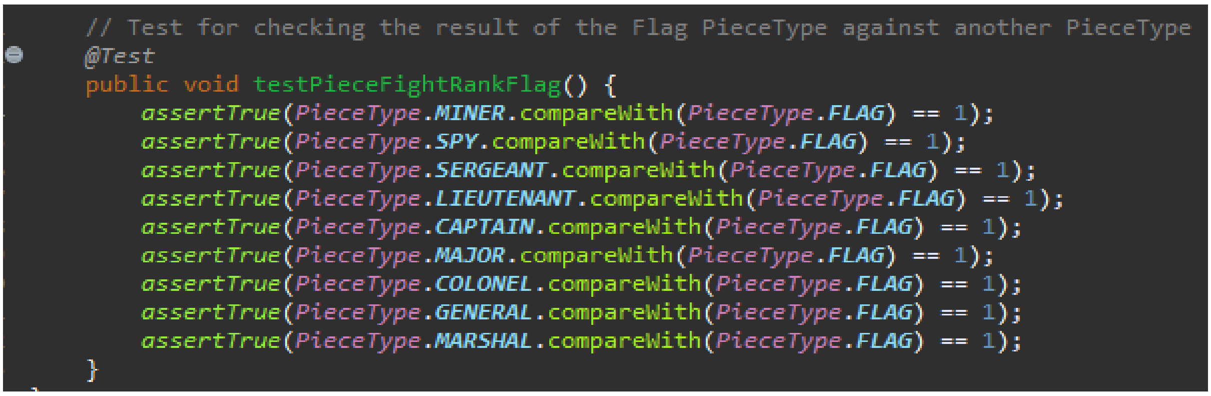
```
    // Test for checking the result of the General PieceType against another PieceType
    @Test
    public void testPieceFightRankGeneral() {
        assertTrue(PieceType.GENERAL.compareWith(PieceType.MINER) == 1);
        assertTrue(PieceType.GENERAL.compareWith(PieceType.CAPTAIN) == 1);
        assertTrue(PieceType.GENERAL.compareWith(PieceType.COLONEL) == 1);
        assertTrue(PieceType.GENERAL.compareWith(PieceType.FLAG) == 1);
        assertTrue(PieceType.GENERAL.compareWith(PieceType.GENERAL) == 0);
        assertTrue(PieceType.GENERAL.compareWith(PieceType.LIEUTENANT) == 1);
        assertTrue(PieceType.GENERAL.compareWith(PieceType.MAJOR) == 1);
        assertTrue(PieceType.GENERAL.compareWith(PieceType.SCOUT) == 1);
        assertTrue(PieceType.GENERAL.compareWith(PieceType.SERGEANT) == 1);
        assertTrue(PieceType.GENERAL.compareWith(PieceType.SPY) == 1);
        assertTrue(PieceType.GENERAL.compareWith(PieceType.MARSHAL) == -1);
        assertTrue(PieceType.GENERAL.compareWith(PieceType.BOMB) == -1);

    }
```

This test case checks the result of the ‘General’ PieceType against other piecetype as it can be seen that the General is only defeated if a Marshal attacks it as the Marshal piece type has a higher rank in the game and has the ability to defeat the every other piece or it can only be defeated when attacking a bomb. The General defeats all other pieces, The assertTrue is used to verify and compare that the result of the ‘General’ piecetype against other pieces is as expected.



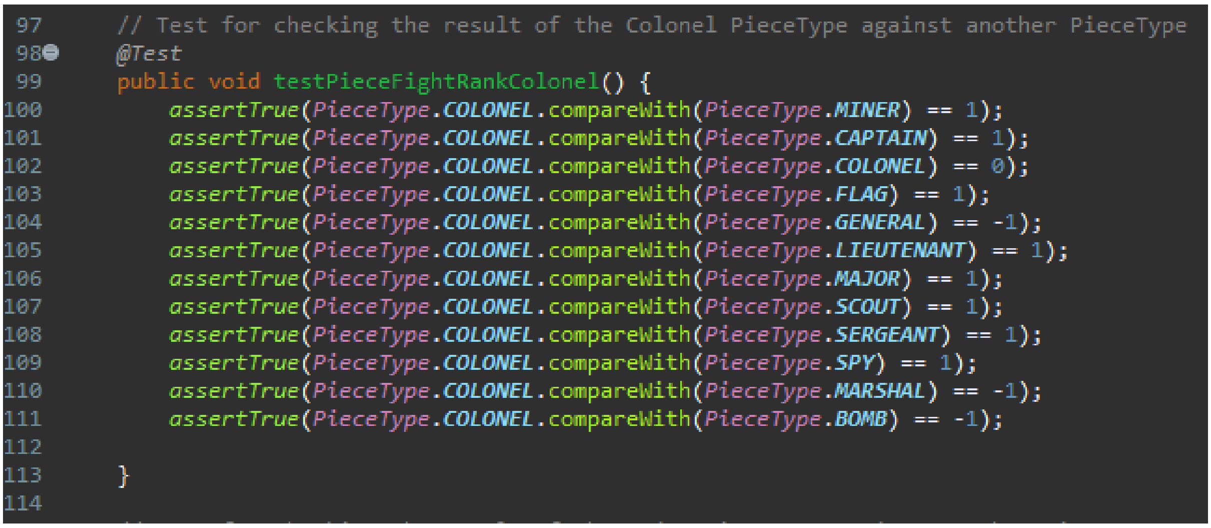
```
// Test for checking the result of the Marshal PieceType against another PieceType
@Test
public void testPieceFightRankMarshal() {
    assertTrue(PieceType.MARSHAL.compareWith(PieceType.MINER) == 1);
    assertTrue(PieceType.MARSHAL.compareWith(PieceType.CAPTAIN) == 1);
    assertTrue(PieceType.MARSHAL.compareWith(PieceType.COLONEL) == 1);
    assertTrue(PieceType.MARSHAL.compareWith(PieceType.FLAG) == 1);
    assertTrue(PieceType.MARSHAL.compareWith(PieceType.GENERAL) == 1);
    assertTrue(PieceType.MARSHAL.compareWith(PieceType.LIEUTENANT) == 1);
    assertTrue(PieceType.MARSHAL.compareWith(PieceType.MAJOR) == 1);
    assertTrue(PieceType.MARSHAL.compareWith(PieceType.SCOUT) == 1);
    assertTrue(PieceType.MARSHAL.compareWith(PieceType.SERGEANT) == 1);
    assertTrue(PieceType.MARSHAL.compareWith(PieceType.SPY) == 1);
    assertTrue(PieceType.MARSHAL.compareWith(PieceType.MARSHAL) == 0);
    assertTrue(PieceType.MARSHAL.compareWith(PieceType.BOMB) == -1);

}
```

This test case checks the result of the ‘Marshal’ PieceType against other piecetype as it can be seen that the Marshal is only defeated if a spy attacks it as the spy is a special piece in the game or it can only be defeated when attacking a bomb. The Marshal defeats all other pieces, The assertTrue is used to verify and compare that the result of the ‘Marshal’ piecetype against other pieces is as expected.



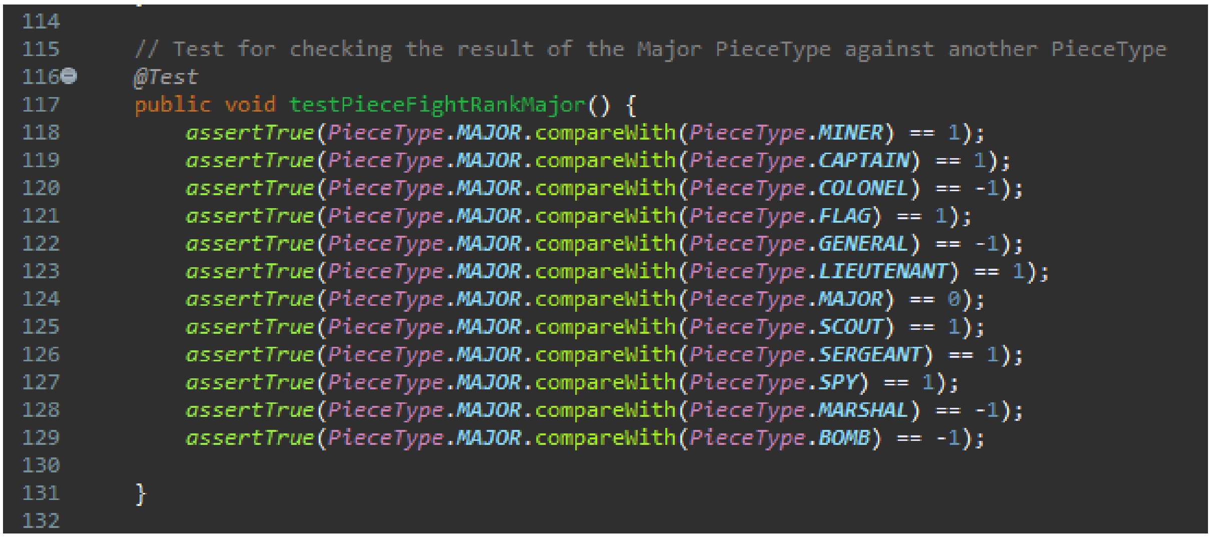
```
    // Test for checking the result of the Flag PieceType against another PieceType
    @Test
    public void testPieceFightRankFlag() {
        assertTrue(PieceType.MINER.compareWith(PieceType.FLAG) == 1);
        assertTrue(PieceType.SPY.compareWith(PieceType.FLAG) == 1);
        assertTrue(PieceType.SERGEANT.compareWith(PieceType.FLAG) == 1);
        assertTrue(PieceType.LIEUTENANT.compareWith(PieceType.FLAG) == 1);
        assertTrue(PieceType.CAPTAIN.compareWith(PieceType.FLAG) == 1);
        assertTrue(PieceType.MAJOR.compareWith(PieceType.FLAG) == 1);
        assertTrue(PieceType.COLONEL.compareWith(PieceType.FLAG) == 1);
        assertTrue(PieceType.GENERAL.compareWith(PieceType.FLAG) == 1);
        assertTrue(PieceType.MARSHAL.compareWith(PieceType.FLAG) == 1);
    }
```

This test case checks the result of the ‘Flag’ PieceType against other piecetype as it can be seen that any piece that captures the flag wins the game and when the flag is captured by one team the game ends.The Flag is a static piece therefore, it does not move. The assertTrue is used to verify and compare that the result of the ‘bomb’ piecetype against other pieces is as expected.



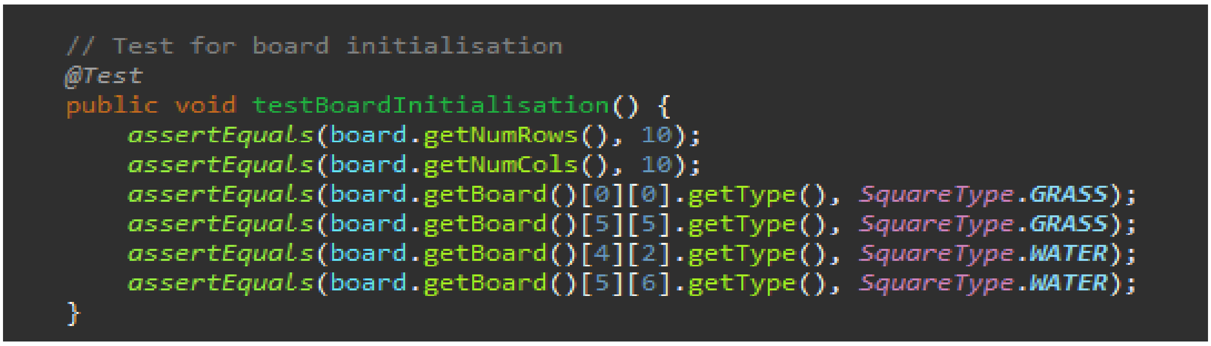
```
    // Test for checking the result of the Colonel PieceType against another PieceType
    @Test
    public void testPieceFightRankColonel() {
        assertTrue(PieceType.COLONEL.compareWith(PieceType.MINER) == 1);
        assertTrue(PieceType.COLONEL.compareWith(PieceType.CAPTAIN) == 1);
        assertTrue(PieceType.COLONEL.compareWith(PieceType.COLONEL) == 0);
        assertTrue(PieceType.COLONEL.compareWith(PieceType.FLAG) == 1);
        assertTrue(PieceType.COLONEL.compareWith(PieceType.GENERAL) == -1);
        assertTrue(PieceType.COLONEL.compareWith(PieceType.LIEUTENANT) == 1);
        assertTrue(PieceType.COLONEL.compareWith(PieceType.MAJOR) == 1);
        assertTrue(PieceType.COLONEL.compareWith(PieceType.SCOUT) == 1);
        assertTrue(PieceType.COLONEL.compareWith(PieceType.SERGEANT) == 1);
        assertTrue(PieceType.COLONEL.compareWith(PieceType.SPY) == 1);
        assertTrue(PieceType.COLONEL.compareWith(PieceType.MARSHAL) == -1);
        assertTrue(PieceType.COLONEL.compareWith(PieceType.BOMB) == -1);

    }

```

This test case checks the result of the ‘Colonel’ PieceType against other piecetype as it can be seen that the Colonel is only defeated if a Marshal or General attacks it as the Marshal and General piece types have a higher rank in the game or it can only be defeated when attacking a bomb. The Colonel defeats all other pieces, The assertTrue is used to verify and compare that the result of the ‘Colonel’ piecetype against other pieces is as expected.



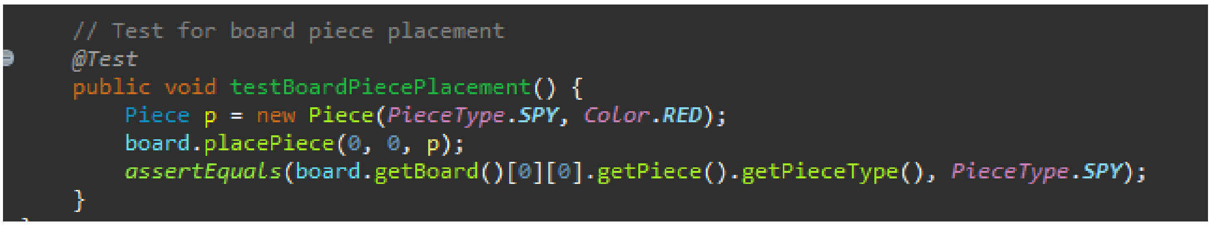
```

    // Test for checking the result of the Major PieceType against another PieceType
    @Test
    public void testPieceFightRankMajor() {
        assertTrue(PieceType.MAJOR.compareWith(PieceType.MINER) == 1);
        assertTrue(PieceType.MAJOR.compareWith(PieceType.CAPTAIN) == 1);
        assertTrue(PieceType.MAJOR.compareWith(PieceType.COLONEL) == -1);
        assertTrue(PieceType.MAJOR.compareWith(PieceType.FLAG) == 1);
        assertTrue(PieceType.MAJOR.compareWith(PieceType.GENERAL) == -1);
        assertTrue(PieceType.MAJOR.compareWith(PieceType.LIEUTENANT) == 1);
        assertTrue(PieceType.MAJOR.compareWith(PieceType.MAJOR) == 0);
        assertTrue(PieceType.MAJOR.compareWith(PieceType.SCOUT) == 1);
        assertTrue(PieceType.MAJOR.compareWith(PieceType.SERGEANT) == 1);
        assertTrue(PieceType.MAJOR.compareWith(PieceType.SPY) == 1);
        assertTrue(PieceType.MAJOR.compareWith(PieceType.MARSHAL) == -1);
        assertTrue(PieceType.MAJOR.compareWith(PieceType.BOMB) == -1);

    }

```

This test case checks the result of the ‘Major’ PieceType against other piecetype as it can be seen that the Major is only defeated if a Marshal or General or Colonel attacks it as the Marshal ,General and Colonel piece types have a higher rank in the game or it can only be defeated when attacking a bomb. The Major defeats all other pieces, The assertTrue is used to verify and compare that the result of the ‘Major’ piecetype against other pieces is as expected.



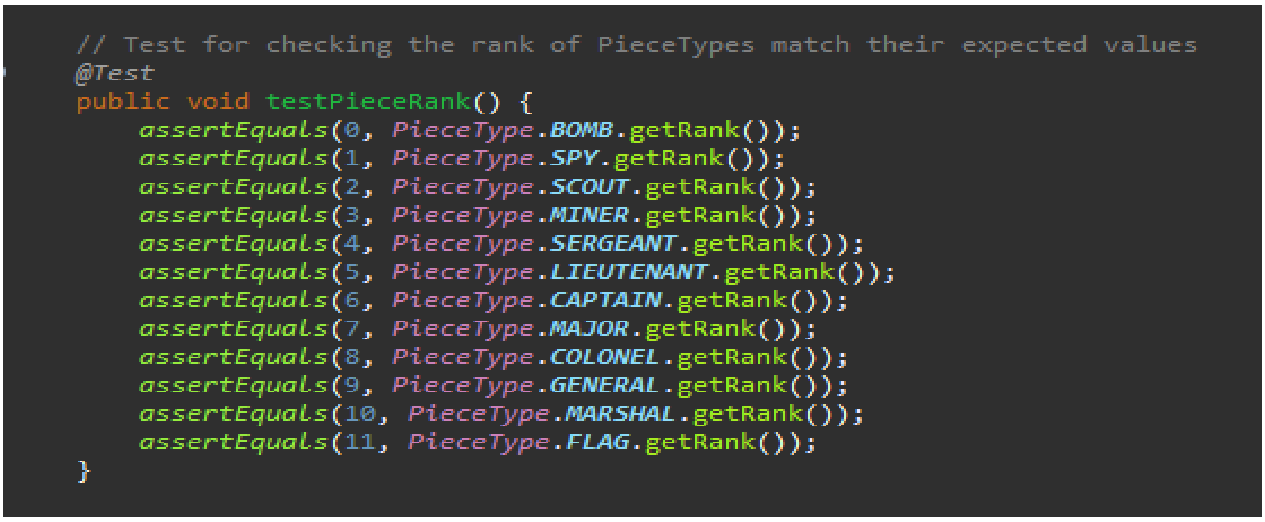
```
// Test for board initialisation
@Test
public void testBoardInitialisation() {
    assertEquals(board.getNumRows(), 10);
    assertEquals(board.getNumCols(), 10);
    assertEquals(board.getBoard()[0][0].getType(), SquareType.GRASS);
    assertEquals(board.getBoard()[5][5].getType(), SquareType.GRASS);
    assertEquals(board.getBoard()[4][2].getType(), SquareType.WATER);
    assertEquals(board.getBoard()[5][6].getType(), SquareType.WATER);
}
```

This test case is used for board initialisation, it checks that the number of rows and columns are each equal to 10 as the board grid is 10 x 10. It also then checks that there are squares that a piece cannot be placed on which consist of water and the rest consist of grass on which a piece can be placed on.The assertTrue is used to verify that the board initialises as expected.



```java
    // Test for board piece placement
    @Test
    public void testBoardPiecePlacement() {
        Piece p = new Piece(PieceType.SPY, Color.RED);
        board.placePiece(0, 0, p);
        assertEquals(board.getBoard()[0][0].getPiece().getPieceType(), PieceType.SPY);
    }
```

This test case is used for the placement of pieces onto the board. In this particular test, the spy piece is being placed on the red team at the grid position 0,0. The assertTrue is used to verify that the piece has been placed at the specific position as expected.



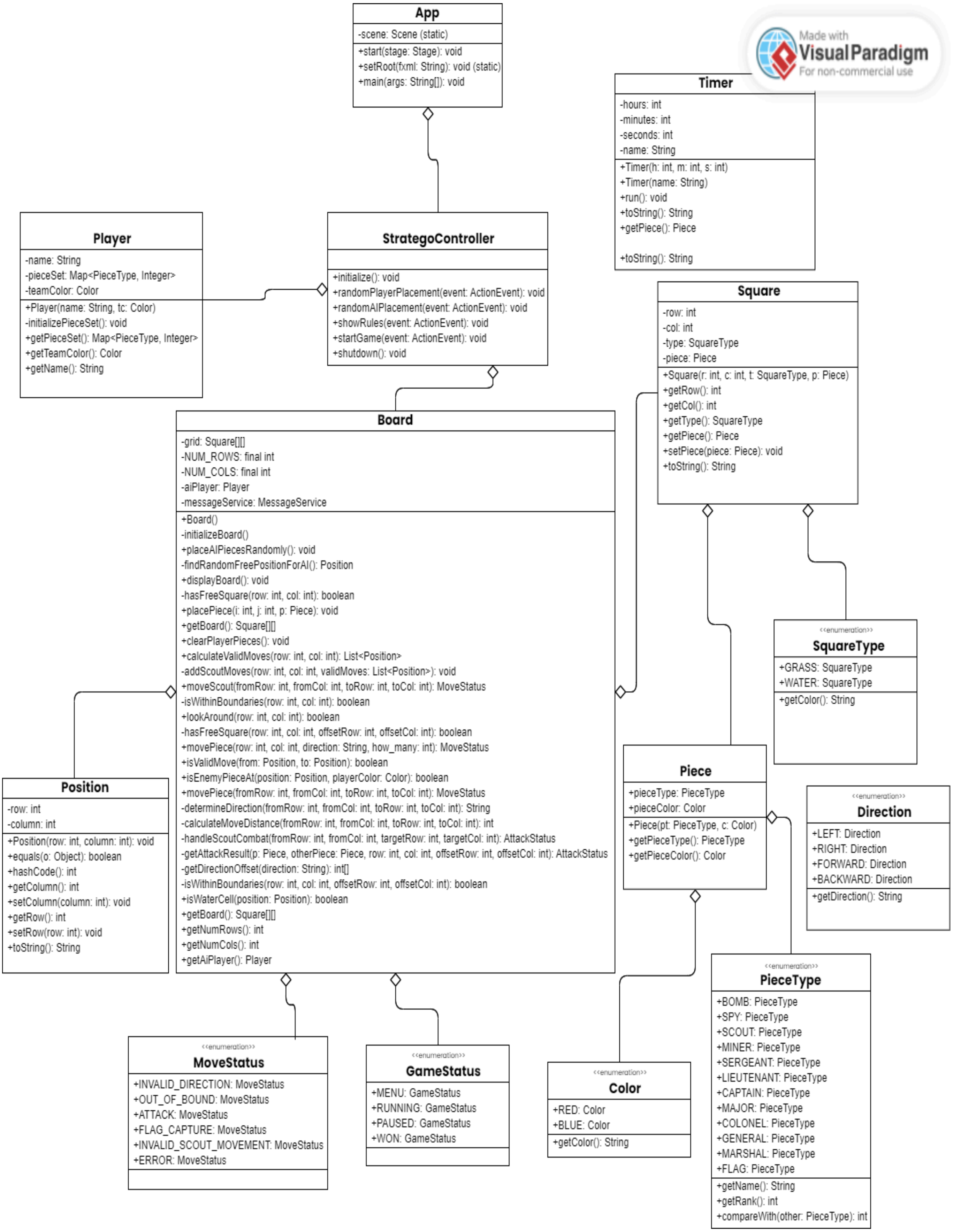
```java
    // Test for checking the rank of PieceTypes match their expected values
    @Test
    public void testPieceRank() {
        assertEquals(0, PieceType.BOMB.getRank());
        assertEquals(1, PieceType.SPY.getRank());
        assertEquals(2, PieceType.SCOUT.getRank());
        assertEquals(3, PieceType.MINER.getRank());
        assertEquals(4, PieceType.SERGEANT.getRank());
        assertEquals(5, PieceType.LIEUTENANT.getRank());
        assertEquals(6, PieceType.CAPTAIN.getRank());
        assertEquals(7, PieceType.MAJOR.getRank());
        assertEquals(8, PieceType.COLONEL.getRank());
        assertEquals(9, PieceType.GENERAL.getRank());
        assertEquals(10, PieceType.MARSHAL.getRank());
        assertEquals(11, PieceType.FLAG.getRank());
    }
```

This test case checks and verifies that the expected rank of the PieceType matches the actual value using the getName() method. The ‘assertEquals’ statement in this test case is used to verify that the rank of piece types in ‘PieceType’ matches the expected rank.

Above are some code snippets of a few test cases that I have coded to test my code and check that the particular functionalities work as intended.

3.4 Class Diagrams and User cases

UML Class Diagram: Below is the UML class diagram for this project


App
-scene: Scene (static)
+start(stage: Stage): void
+setRoot(fxml: String): void (static)
+main(args: String[]): void
Made with Visual Paradigm For non-commercial use
Timer
-hours: int
-minutes: int
-seconds: int
-name: String
+Timer(h: int, m: int, s: int)
+Timer(name: String)
+run(): void
+toString(): String
+getPiece(): Piece
+toString(): String
Player
-name: String
-pieceSet: Map<PieceType, Integer>
-teamColor: Color
+Player(name: String, tc: Color)
-initializePieceSet(): void
+getPieceSet(): Map<PieceType, Integer>
+getTeamColor(): Color
+getName(): String
StrategoController
+initialize(): void
+randomPlayerPlacement(event: ActionEvent): void
+randomAIPlacement(event: ActionEvent): void
+showRules(event: ActionEvent): void
+startGame(event: ActionEvent): void
+shutdown(): void
Square
-row: int
-col: int
-type: SquareType
-piece: Piece
+Square(r: int, c: int, t: SquareType, p: Piece)
+getRow(): int
+getCol(): int
+getType(): SquareType
+getPiece(): Piece
+setPiece(piece: Piece): void
+toString(): String
Board
-grid: Square[][]
-NUM_ROWS: final int
-NUM_COLS: final int
-aiPlayer: Player
-messageService: MessageService
+Board()
-initializeBoard()
+placeAIPiecesRandomly(): void
-findRandomFreePositionForAI(): Position
+displayBoard(): void
-hasFreeSquare(row: int, col: int): boolean
+placePiece(i: int, j: int, p: Piece): void
+getBoard(): Square[][]
+clearPlayerPieces(): void
+calculateValidMoves(row: int, col: int): List<Position>
-addScoutMoves(row: int, col: int, validMoves: List<Position>): void
+moveScout(fromRow: int, fromCol: int, toRow: int, toCol: int): MoveStatus
-isWithinBoundaries(row: int, col: int): boolean
+lookAround(row: int, col: int): boolean
-hasFreeSquare(row: int, col: int, offsetRow: int, offsetCol: int): boolean
+movePiece(row: int, col: int, direction: String, how_many: int): MoveStatus
+isValidMove(from: Position, to: Position): boolean
+isEnemyPieceAt(position: Position, playerColor: Color): boolean
+movePiece(fromRow: int, fromCol: int, toRow: int, toCol: int): MoveStatus
-determineDirection(fromRow: int, fromCol: int, toRow: int, toCol: int): String
-calculateMoveDistance(fromRow: int, fromCol: int, toRow: int, toCol: int): int
-handleScoutCombat(fromRow: int, fromCol: int, targetRow: int, targetCol: int): AttackStatus
-getAttackResult(p: Piece, otherPiece: Piece, row: int, col: int, offsetRow: int, offsetCol: int): AttackStatus
-getDirectionOffset(direction: String): int[]
-isWithinBoundaries(row: int, col: int, offsetRow: int, offsetCol: int): boolean
+isWaterCell(position: Position): boolean
+getBoard(): Square[][]
+getNumRows(): int
+getNumCols(): int
+getAiPlayer(): Player
<<enumeration>>
SquareType
+GRASS: SquareType
+WATER: SquareType
+getColor(): String
Position
-row: int
-column: int
+Position(row: int, column: int): void
+equals(o: Object): boolean
+hashCode(): int
+getColumn(): int
+setColumn(column: int): void
+getRow(): int
+setRow(row: int): void
+toString(): String
Piece
+pieceType: PieceType
+pieceColor: Color
+Piece(pt: PieceType, c: Color)
+getPieceType(): PieceType
+getPieceColor(): Color
<<enumeration>>
Direction
+LEFT: Direction
+RIGHT: Direction
+FORWARD: Direction
+BACKWARD: Direction
+getDirection(): String
<<enumeration>>
PieceType
+BOMB: PieceType
+SPY: PieceType
+SCOUT: PieceType
+MINER: PieceType
+SERGEANT: PieceType
+LIEUTENANT: PieceType
+CAPTAIN: PieceType
+MAJOR: PieceType
+COLONEL: PieceType
+GENERAL: PieceType
+MARSHAL: PieceType
+FLAG: PieceType
+getName(): String
+getRank(): int
+compareWith(other: PieceType): int
<<enumeration>>
MoveStatus
+INVALID_DIRECTION: MoveStatus
+OUT_OF_BOUND: MoveStatus
+ATTACK: MoveStatus
+FLAG_CAPTURE: MoveStatus
+INVALID_SCOUT_MOVEMENT: MoveStatus
+ERROR: MoveStatus
<<enumeration>>
GameStatus
+MENU: GameStatus
+RUNNING: GameStatus
+PAUSED: GameStatus
+WON: GameStatus
<<enumeration>>
Color
+RED: Color
+BLUE: Color
+getColor(): String

As the UML class diagram might be a bit confusing to read so I have also tried to split the UML class diagram into images in parts which might provide a better viewing.

Image 1:

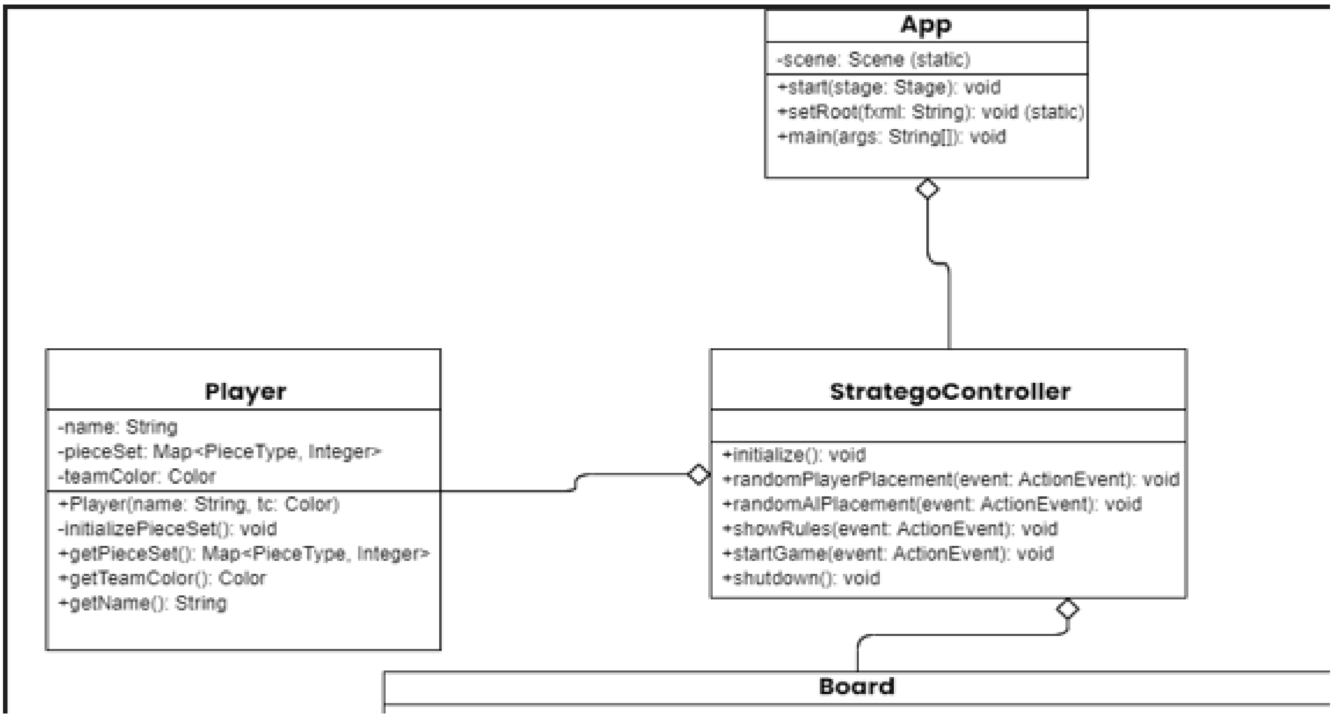

App
-scene: Scene (static)
+start(stage: Stage): void
+setRoot(fxml: String): void (static)
+main(args: String[]): void
Player
-name: String
-pieceSet: Map<PieceType, Integer>
-teamColor: Color
+Player(name: String, tc: Color)
-initializePieceSet(): void
+getPieceSet(): Map<PieceType, Integer>
+getTeamColor(): Color
+getName(): String
StrategoController
+initialize(): void
+randomPlayerPlacement(event: ActionEvent): void
+randomAIPlacement(event: ActionEvent): void
+showRules(event: ActionEvent): void
+startGame(event: ActionEvent): void
+shutdown(): void
Board

Image 2:

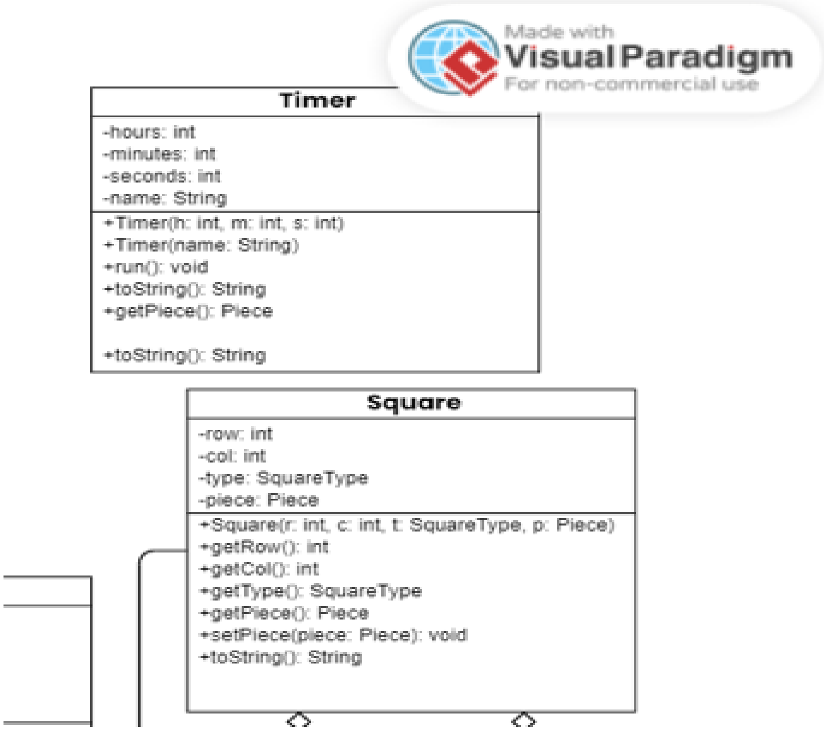


Image 3:

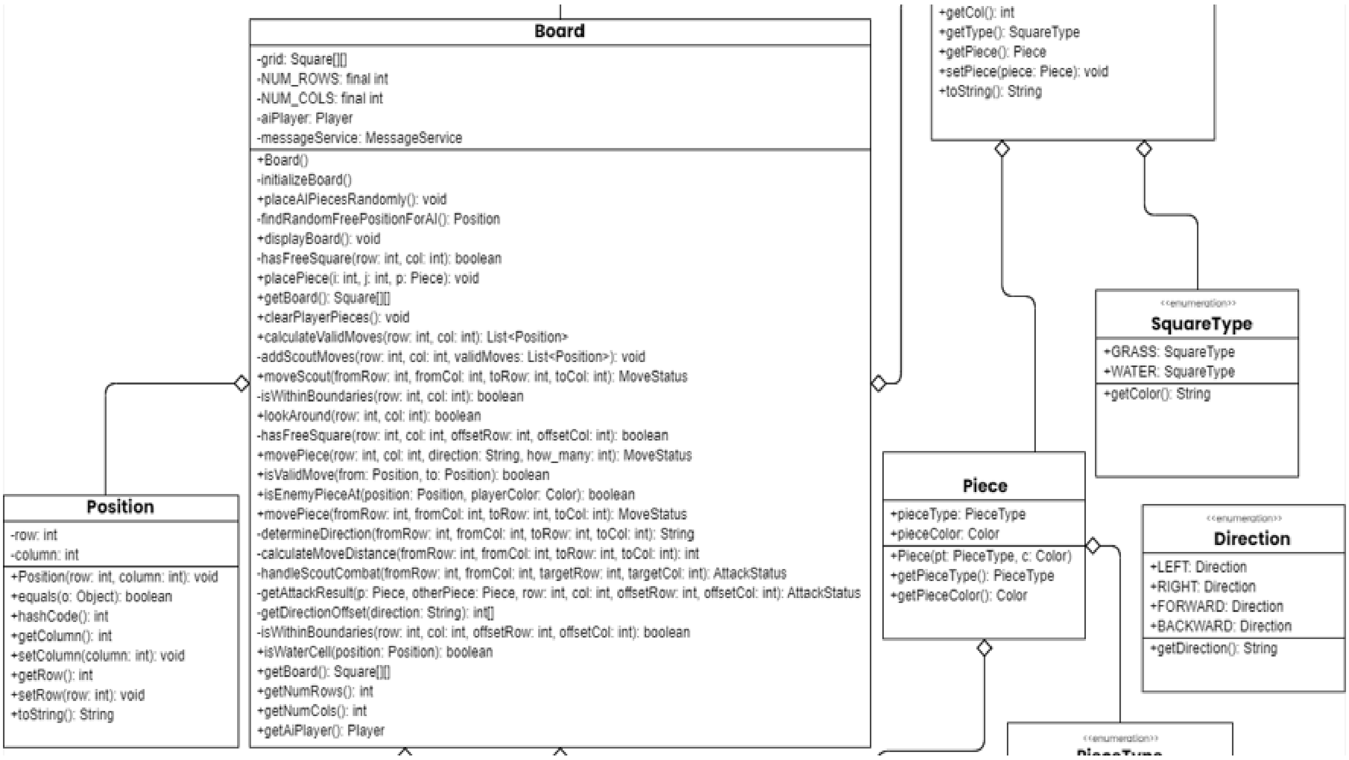
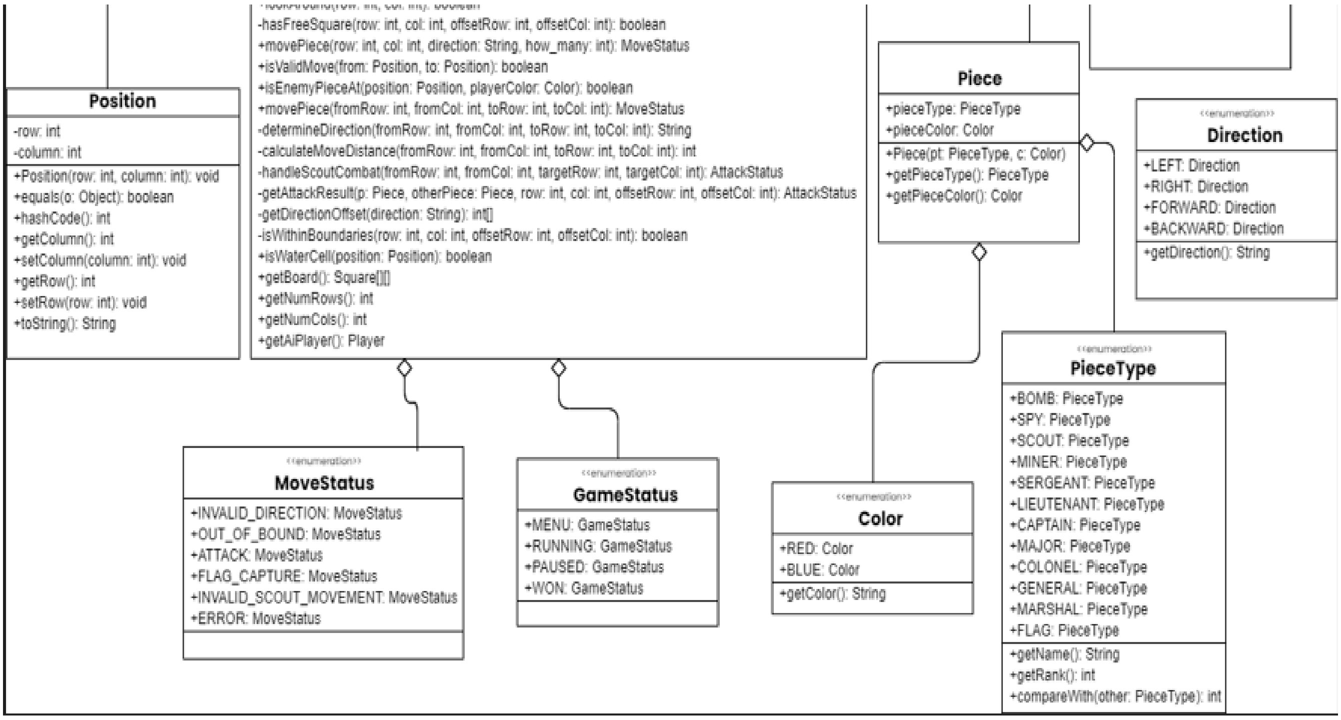

+getCol(): int
+getType(): SquareType
+getPiece(): Piece
+setPiece(piece: Piece): void
+toString(): String
Board
-grid: Square[][]
-NUM_ROWS: final int
-NUM_COLS: final int
-aiPlayer: Player
-messageService: MessageService
+Board()
-initializeBoard()
+placeAIPiecesRandomly(): void
-findRandomFreePositionForAI(): Position
+displayBoard(): void
-hasFreeSquare(row: int, col: int): boolean
+placePiece(i: int, j: int, p: Piece): void
+getBoard(): Square[][]
+clearPlayerPieces(): void
+calculateValidMoves(row: int, col: int): List<Position>
-addScoutMoves(row: int, col: int, validMoves: List<Position>): void
+moveScout(fromRow: int, fromCol: int, toRow: int, toCol: int): MoveStatus
-isWithinBoundaries(row: int, col: int): boolean
+lookAround(row: int, col: int): boolean
-hasFreeSquare(row: int, col: int, offsetRow: int, offsetCol: int): boolean
+movePiece(row: int, col: int, direction: String, how_many: int): MoveStatus
+isValidMove(from: Position, to: Position): boolean
+isEnemyPieceAt(position: Position, playerColor: Color): boolean
+movePiece(fromRow: int, fromCol: int, toRow: int, toCol: int): MoveStatus
-determineDirection(fromRow: int, fromCol: int, toRow: int, toCol: int): String
-calculateMoveDistance(fromRow: int, fromCol: int, toRow: int, toCol: int): int
-handleScoutCombat(fromRow: int, fromCol: int, targetRow: int, targetCol: int): AttackStatus
-getAttackResult(p: Piece, otherPiece: Piece, row: int, col: int, offsetRow: int, offsetCol: int): AttackStatus
-getDirectionOffset(direction: String): int[]
-isWithinBoundaries(row: int, col: int, offsetRow: int, offsetCol: int): boolean
+isWaterCell(position: Position): boolean
+getBoard(): Square[][]
+getNumRows(): int
+getNumCols(): int
+getAiPlayer(): Player
Position
-row: int
-column: int
+Position(row: int, column: int): void
+equals(o: Object): boolean
+hashCode(): int
+getColumn(): int
+setColumn(column: int): void
+getRow(): int
+setRow(row: int): void
+toString(): String
<<enumeration>>
SquareType
+GRASS: SquareType
+WATER: SquareType
+getColor(): String
Piece
+pieceType: PieceType
+pieceColor: Color
+Piece(pt: PieceType, c: Color)
+getPieceType(): PieceType
+getPieceColor(): Color
<<enumeration>>
Direction
+LEFT: Direction
+RIGHT: Direction
+FORWARD: Direction
+BACKWARD: Direction
+getDirection(): String
<<enumeration>>
PieceType

Image 4:


+lookAround(row: int, col: int): boolean
-hasFreeSquare(row: int, col: int, offsetRow: int, offsetCol: int): boolean
+movePiece(row: int, col: int, direction: String, how_many: int): MoveStatus
+isValidMove(from: Position, to: Position): boolean
+isEnemyPieceAt(position: Position, playerColor: Color): boolean
+movePiece(fromRow: int, fromCol: int, toRow: int, toCol: int): MoveStatus
-determineDirection(fromRow: int, fromCol: int, toRow: int, toCol: int): String
-calculateMoveDistance(fromRow: int, fromCol: int, toRow: int, toCol: int): int
-handleScoutCombat(fromRow: int, fromCol: int, targetRow: int, targetCol: int): AttackStatus
-getAttackResult(p: Piece, otherPiece: Piece, row: int, col: int, offsetRow: int, offsetCol: int): AttackStatus
-getDirectionOffset(direction: String): int[]
-isWithinBoundaries(row: int, col: int, offsetRow: int, offsetCol: int): boolean
+isWaterCell(position: Position): boolean
+getBoard(): Square[][]
+getNumRows(): int
+getNumCols(): int
+getAiPlayer(): Player
Position
-row: int
-column: int
+Position(row: int, column: int): void
+equals(o: Object): boolean
+hashCode(): int
+getColumn(): int
+setColumn(column: int): void
+getRow(): int
+setRow(row: int): void
+toString(): String
Piece
+pieceType: PieceType
+pieceColor: Color
+Piece(pt: PieceType, c: Color)
+getPieceType(): PieceType
+getPieceColor(): Color
<<enumeration>>
Direction
+LEFT: Direction
+RIGHT: Direction
+FORWARD: Direction
+BACKWARD: Direction
+getDirection(): String
<<enumeration>>
MoveStatus
+INVALID_DIRECTION: MoveStatus
+OUT_OF_BOUND: MoveStatus
+ATTACK: MoveStatus
+FLAG_CAPTURE: MoveStatus
+INVALID_SCOUT_MOVEMENT: MoveStatus
+ERROR: MoveStatus
<<enumeration>>
GameStatus
+MENU: GameStatus
+RUNNING: GameStatus
+PAUSED: GameStatus
+WON: GameStatus
<<enumeration>>
Color
+RED: Color
+BLUE: Color
+getColor(): String
<<enumeration>>
PieceType
+BOMB: PieceType
+SPY: PieceType
+SCOUT: PieceType
+MINER: PieceType
+SERGEANT: PieceType
+LIEUTENANT: PieceType
+CAPTAIN: PieceType
+MAJOR: PieceType
+COLONEL: PieceType
+GENERAL: PieceType
+MARSHAL: PieceType
+FLAG: PieceType
+getName(): String
+getRank(): int
+compareWith(other: PieceType): int

User case stories:

- **As a player I want to be able to initialise the game in order to start the game.**
  - In the current stage of the game, the player can initialise the game and start the game. The player will be placed on the blue team. I did not have the time to make the selection random. Therefore, the user is placed on the blue team and the AI is placed on the red team.
- **As a player I want to be able to see my pieces and place them on the board.**
  - After the player has started the game, the game board is generated in a 10x10 grid with the pieces being placed on it using a button that places the pieces randomly or the player can use drag-and-drop method to manually place the pieces.
- **As a player I want to be able to move the pieces.**
  - The player can move the pieces however they like, given there is space around to move the piece otherwise the piece doesn’t move.The player can move the piece up,down,right,left by clicking the selected piece given the restrictions the piece has i.e. some pieces have no restrictions such as the scout and that piecetype can move however pieces they want to move but all the other pieces are only allowed to move one step in any direction. All the directions that the piece can move in are highlighted in green colour when the piece is clicked.
- **As a player I want to be able to attack the opponent's piece**
  - The player can successfully attack the opponent’s piece and when attacked the piece that wins replaces the place of the other piece. For eg. A spy attacks a marshal, Spy WINS.
- **As a player I want to be able to see my progression and see the history of moves.**
  - The player can see their progression and view the history of moves in a dedicated section on the game board. The player can scroll through the list to view past history as well. In addition to the history of moves, the player can see at what time of game a player’s piece was killed and which piece was eliminated from the game.
- **As a player I want to be able to save and load the game.**
  - I was unable to implement this feature this term due to lack of time and knowledge on how to load and save a game.

- **As a player I want to be able to view and read the rules of the game.**
  - The Player can view the rules by going to the help section in the menu bar of the game and under the help section, they can click on the rules option from the drop-down list and view the rules of the game. A pop up window is shown with all the rules written in it. The player can press the 'ok' button to exit the rules section.

- **As a player I want to view the amount of pieces there are per pieceType.**
  - The player can view the amount of pieces there are per pieceType via a dedicated section on the left hand side of the game panel under the 'Amount: Piece Number' section.

- **As a player I want to be able to view the pieces that have been killed.**
  - The player can view the amount and the name of pieces that have been killed/captured in a list view, where they can even see the number of pieces that have been killed per pieceType. The list keeps updating as the game proceeds.

- **As a player I want to place the AI pieces randomly and be unable to see them.**
  - The player can place the AI pieces randomly with a 'Place AI pieces' button and the pieces are covered as they are placed which does not allow the player to see the AI pieces and where they have been placed.

- **As a player I want to be able to quit the game.**
  - The player can quit the game at any point they like by just simply clicking the 'quit' button.

- **As a player I want to be able to win the game.**
  - The player can win the game by simply capturing the opponent's flag as possible. The player that captures the opponent's flag first wins the game. A pop up message is displayed when the flag is captured stating 'You Won!!! Congratulation :)' and the game window closes after ok is clicked when the pop-up message appears.

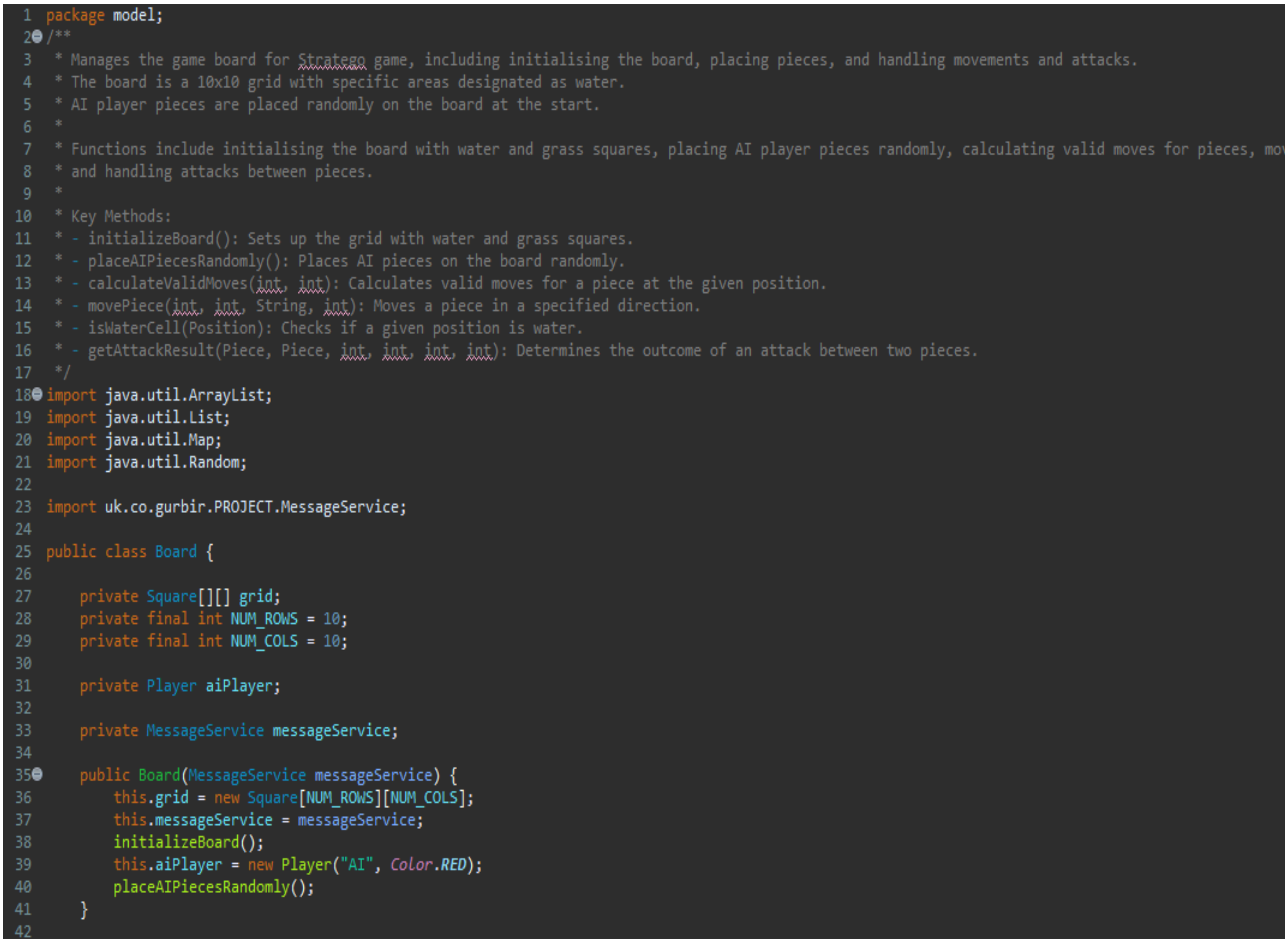
- ➢ **As a player I want to be able to have access to a few feedback mechanisms.**
  - I was not able to achieve this user-case of having few feedback mechanisms such as audio cues when a player is selected or an audio cue when a piece has been killed due to lack of time, resources and knowledge.
- ➢ **As a player I want to be able to view how long it took to complete the game.**
  - The player can view how long the game has been going on for as there is a timer included in the game which runs on its own thread concurrently.
- ➢ **As a player I want to have accessibility options such as a board that helps people with colorblindness.**
  - I was not able to achieve this use-case of having accessibility options such as a board that helps people with colorblindness, due to lack of time, resources and knowledge.
- ➢ **As a player I want to be able to play with my friends or players online.**
  - The game currently is only a single-player game against the AI. I was unable to achieve this use-case due to lack of time to make the game multiplayer or even play on a server online.

3.5 Documentation

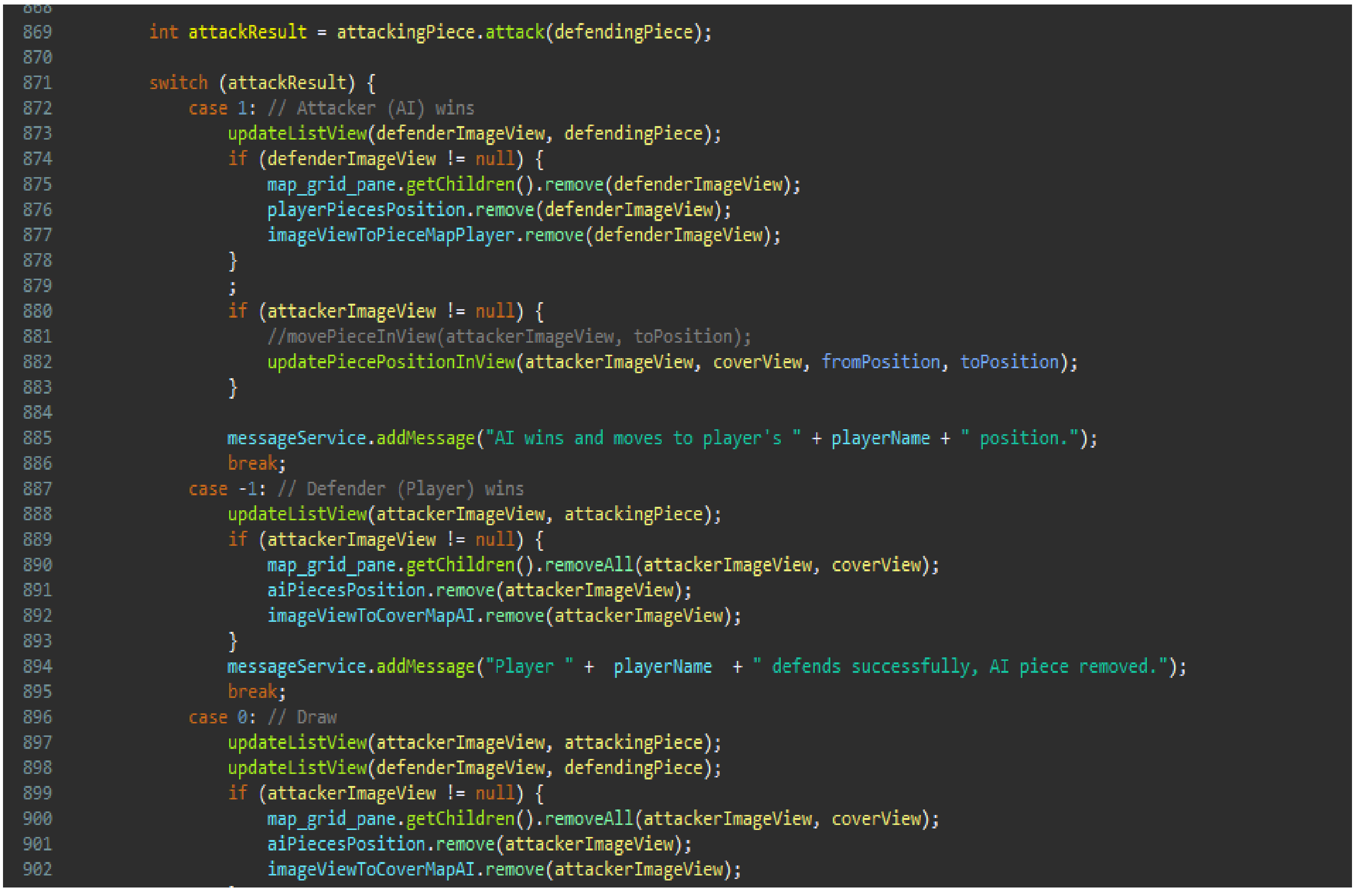
In my project, I have maintained clarity and maintainability by implementing a thorough documentation approach. I used this approach by using two main strategies: Use of inline comments and use of external documentation.

Inline comments:

I used JavaDoc and comments in order to have a clear structure to my code. I used JavaDoc in order to annotate my class and the methods underneath them. The purpose of using JavaDoc was to improve the project's overall readability, maintainability whilst reducing the complexity of the code making it easier to understand and read the code. This approach made it easier for readers to navigate and understand parameters, return values that are in the methods. Using JavaDoc and comments also helped me remove redundant code. Few examples of JavaDoc and comments being used in the code can be found below:



```
package model;
/**
 * Manages the game board for Stratego game, including initialising the board, placing pieces, and handling movements and attacks.
 * The board is a 10x10 grid with specific areas designated as water.
 * AI player pieces are placed randomly on the board at the start.
 *
 * Functions include initialising the board with water and grass squares, placing AI player pieces randomly, calculating valid moves for pieces, mov
 * and handling attacks between pieces.
 *
 * Key Methods:
 * - initializeBoard(): Sets up the grid with water and grass squares.
 * - placeAIPiecesRandomly(): Places AI pieces on the board randomly.
 * - calculateValidMoves(int, int): Calculates valid moves for a piece at the given position.
 * - movePiece(int, int, String, int): Moves a piece in a specified direction.
 * - isWaterCell(Position): Checks if a given position is water.
 * - getAttackResult(Piece, Piece, int, int, int, int): Determines the outcome of an attack between two pieces.
 */
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.Random;

import uk.co.gurbir.PROJECT.MessageService;

public class Board {

    private Square[][] grid;
    private final int NUM_ROWS = 10;
    private final int NUM_COLS = 10;

    private Player aiPlayer;

    private MessageService messageService;

    public Board(MessageService messageService) {
        this.grid = new Square[NUM_ROWS][NUM_COLS];
        this.messageService = messageService;
        initializeBoard();
        this.aiPlayer = new Player("AI", Color.RED);
        placeAIPiecesRandomly();
    }

```



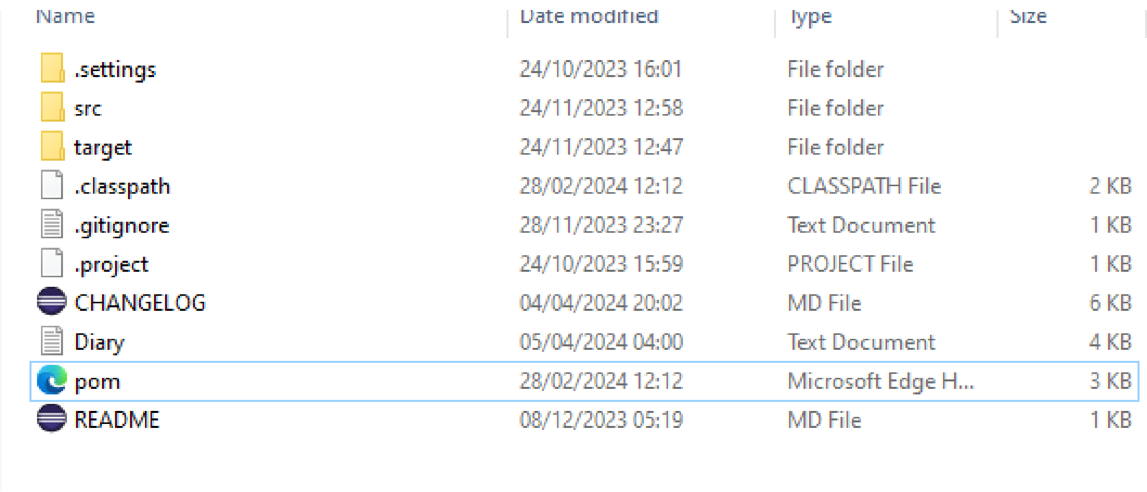
```
int attackResult = attackingPiece.attack(defendingPiece);

switch (attackResult) {
    case 1: // Attacker (AI) wins
        updateListView(defenderImageView, defendingPiece);
        if (defenderImageView != null) {
            map_grid_pane.getChildren().remove(defenderImageView);
            playerPiecesPosition.remove(defenderImageView);
            imageViewToPieceMapPlayer.remove(defenderImageView);
        }
        ;
        if (attackerImageView != null) {
            //movePieceInView(attackerImageView, toPosition);
            updatePiecePositionInView(attackerImageView, coverView, fromPosition, toPosition);
        }

        messageService.addMessage("AI wins and moves to player's " + playerName + " position.");
        break;
    case -1: // Defender (Player) wins
        updateListView(attackerImageView, attackingPiece);
        if (attackerImageView != null) {
            map_grid_pane.getChildren().removeAll(attackerImageView, coverView);
            aiPiecesPosition.remove(attackerImageView);
            imageViewToCoverMapAI.remove(attackerImageView);
        }
        messageService.addMessage("Player " +  playerName  + " defends successfully, AI piece removed.");
        break;
    case 0: // Draw
        updateListView(attackerImageView, attackingPiece);
        updateListView(defenderImageView, defendingPiece);
        if (attackerImageView != null) {
            map_grid_pane.getChildren().removeAll(attackerImageView, coverView);
            aiPiecesPosition.remove(attackerImageView);
            imageViewToCoverMapAI.remove(attackerImageView);
```

External documentation:

Alongside the use of inline comments, the use of external documentation was essential in order to maintain code stability, structure. The documentation was also used to gather complete project information, setup instructions and version history. The external documentation section consist of the following documents:

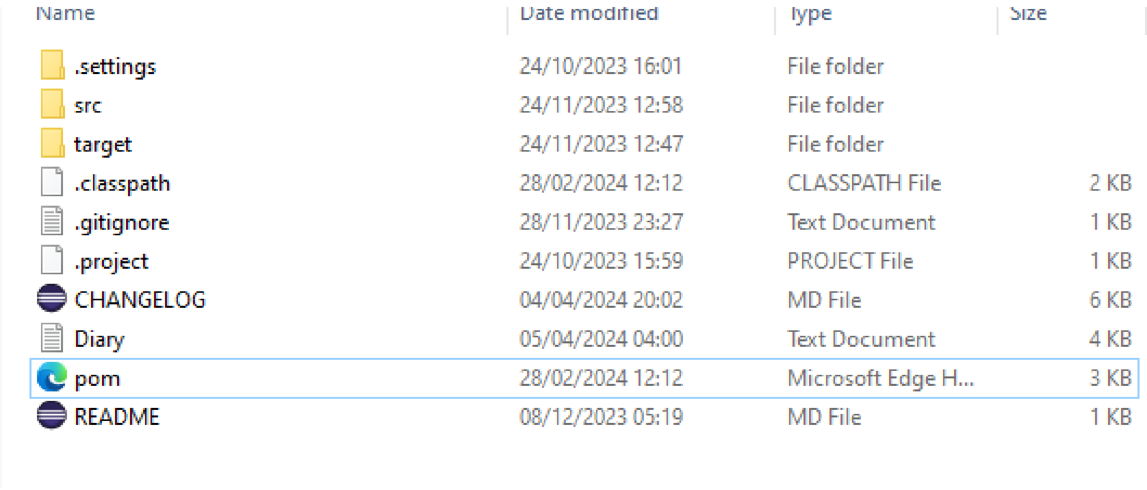
- Readme.md
- diary.txt
- CHANGELOG.md
- Notebook

Readme.md: The readme markdown file includes the instructions on how to run the application correctly. This file is essential as it will make sure the reader can install and run the game on their end with ease. It can be seen below how readme.md has been in the project folder so that the user can follow the instructions easily.



Name	Date modified	Type	Size
.settings	24/10/2023 16:01	File folder	
src	24/11/2023 12:58	File folder	
target	24/11/2023 12:47	File folder	
.classpath	28/02/2024 12:12	CLASSPATH File	2 KB
.gitignore	28/11/2023 23:27	Text Document	1 KB
.project	24/10/2023 15:59	PROJECT File	1 KB
CHANGELOG	04/04/2024 20:02	MD File	6 KB
Diary	05/04/2024 04:00	Text Document	4 KB
pom	28/02/2024 12:12	Microsoft Edge H...	3 KB
README	08/12/2023 05:19	MD File	1 KB

Diary.txt: Include a work log of what has been done between on what dates, keeping overall structure to the code whilst keeping a history of work that has been done throughout the course of the project. It also highlights the progress that has been made in the project from the beginning to the end.It can be seen below how Diary.txt has been in the project folder so that the reader can follow the log of progress made.



Name	Date modified	Type	Size
.settings	24/10/2023 16:01	File folder	
src	24/11/2023 12:58	File folder	
target	24/11/2023 12:47	File folder	
.classpath	28/02/2024 12:12	CLASSPATH File	2 KB
.gitignore	28/11/2023 23:27	Text Document	1 KB
.project	24/10/2023 15:59	PROJECT File	1 KB
CHANGELOG	04/04/2024 20:02	MD File	6 KB
Diary	05/04/2024 04:00	Text Document	4 KB
pom	28/02/2024 12:12	Microsoft Edge H...	3 KB
README	08/12/2023 05:19	MD File	1 KB

CHANGELOG.md: The CHANGELOG markdown file includes a log of all the version history and the significant changes that have been implemented in this project. The CHANGELOG file consists of the version history, under the version history is the added section where we state all the additions made in that version of code. There are also deleted and modification sections. The deleted section includes all the features/functionalities that have been deleted in that version of that file. The modification section includes all the modifications made to existing classes or methods.

Notebook: The notebook consists of the notes that I have taken during my discussion with my supervisor during our meeting regarding the project. It contains key points that I discussed with my supervisor during our meeting. The notebook has been placed in a section of its own near the end of the report.

4. End Game Development

4.1 Architecture of End System

The final architecture of the Stratego game leverages an MVC(Model View Controller) design pattern and the use of concurrency via multithreading; this was done in order to have an organised system that includes the visuals/GUI, the game logic or even the rules of the game. The MVC pattern allowed me to update and code different parts of the code easily. The heart of the MVC pattern is at the StrategoController.java class which acts as a controller and essentially links the models to the view. The model in the game is a package that includes all the classes required to develop the game. Some of the classes that are in the model package are the board class, the piece class, the pieceType class, the square class and many more. The model package contains the game logic which is important to ensure that the move being played is valid, the movement that a piece is doing is valid and the result of a duel between two pieces. The view section of the MVC pattern contains FXML code which is used to display the game board to the user and the controller essentially creates a link between the model and view ensuring that the activities being carried out in the view are logical.

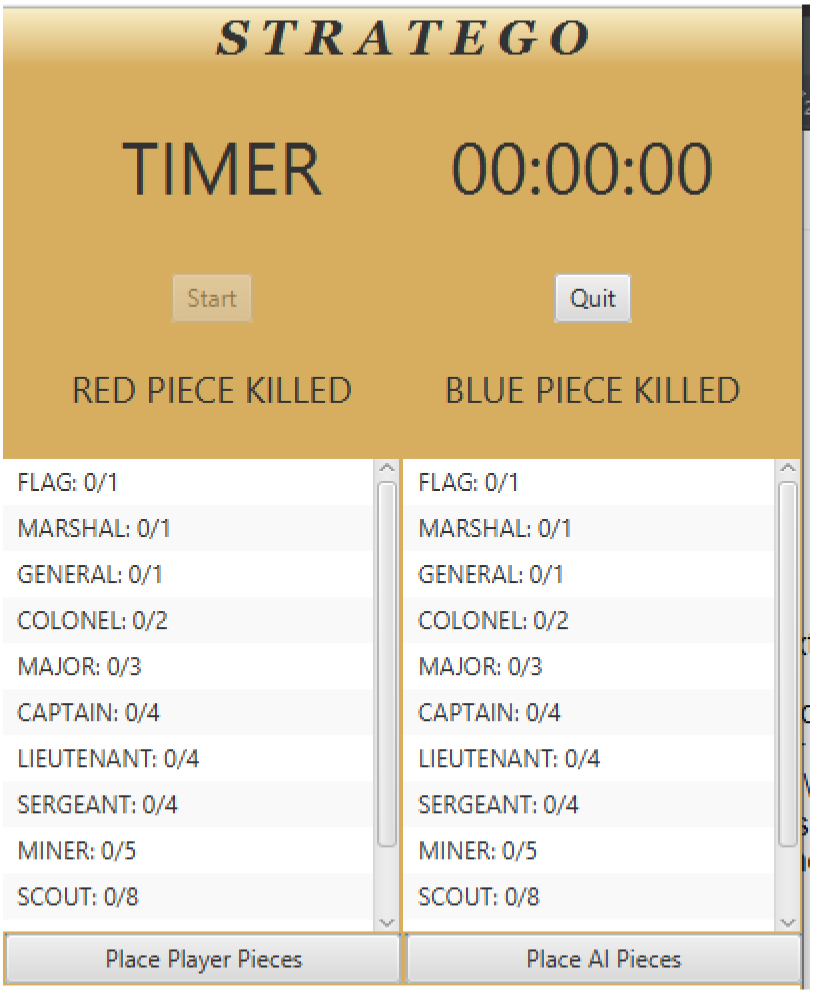
To boost game performance and maximise user experience multithreading was used. Multithreading is essentially when a lot of threads work simultaneously to each other without interfering with the main application and avoid causing any delays. Multithreading allowed me to advance gameplay features such as playing against an AI. An example of multithreading is moving AI and player pieces(both running on there on separate threads) along with the timer continuously updating as the game progresses. This approach allowed me to make a game that displays concurrency, has a visually pleasing GUI and a robust error-handling framework.

4.2 Features of the End System

This section of the report is about all the interesting and key features of the Stratego game which make the game special.

The AI interaction is one of the most vital features in this game as the game is single-player and it is essential that the player has someone to play against. The AI interaction starts by firstly placing the AI pieces randomly through a place AI piece button. Once the button is pressed or clicked the button is disabled and the AI pieces start being placed randomly on the board. The pieces are covered so that the player is unable to see them. However, the AI piece cover does come off for 5 seconds only and only when the player attacks the AI piece and the AI piece has a higher rank or is a special piece. The piece is uncovered so that the player can remember what the piece was and try to kill it with a more powerful piece. This makes the gameplay feel much more natural. In the images below some of the points talked about above can be seen.

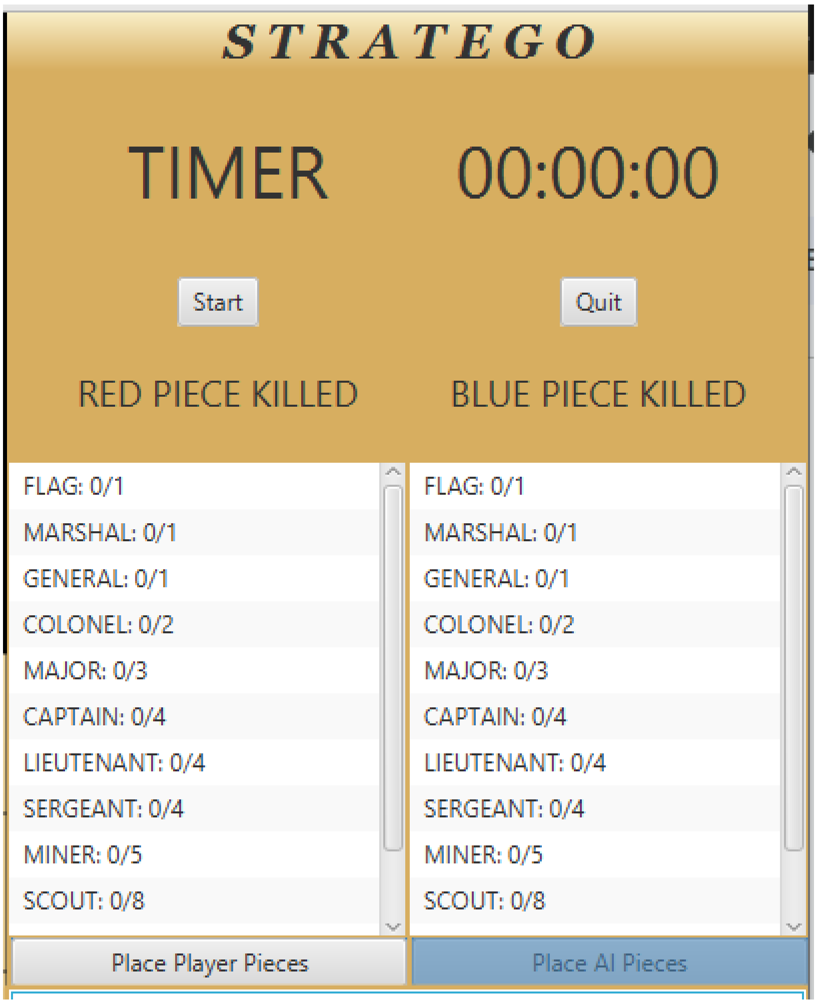
Image 1 :



TIMER 00:00:00

RED PIECE KILLED	BLUE PIECE KILLED
FLAG: 0/1	FLAG: 0/1
MARSHAL: 0/1	MARSHAL: 0/1
GENERAL: 0/1	GENERAL: 0/1
COLONEL: 0/2	COLONEL: 0/2
MAJOR: 0/3	MAJOR: 0/3
CAPTAIN: 0/4	CAPTAIN: 0/4
LIEUTENANT: 0/4	LIEUTENANT: 0/4
SERGEANT: 0/4	SERGEANT: 0/4
MINER: 0/5	MINER: 0/5
SCOUT: 0/8	SCOUT: 0/8

Image 2:



RED PIECE KILLED	BLUE PIECE KILLED
FLAG: 0/1	FLAG: 0/1
MARSHAL: 0/1	MARSHAL: 0/1
GENERAL: 0/1	GENERAL: 0/1
COLONEL: 0/2	COLONEL: 0/2
MAJOR: 0/3	MAJOR: 0/3
CAPTAIN: 0/4	CAPTAIN: 0/4
LIEUTENANT: 0/4	LIEUTENANT: 0/4
SERGEANT: 0/4	SERGEANT: 0/4
MINER: 0/5	MINER: 0/5
SCOUT: 0/8	SCOUT: 0/8

Image 3:

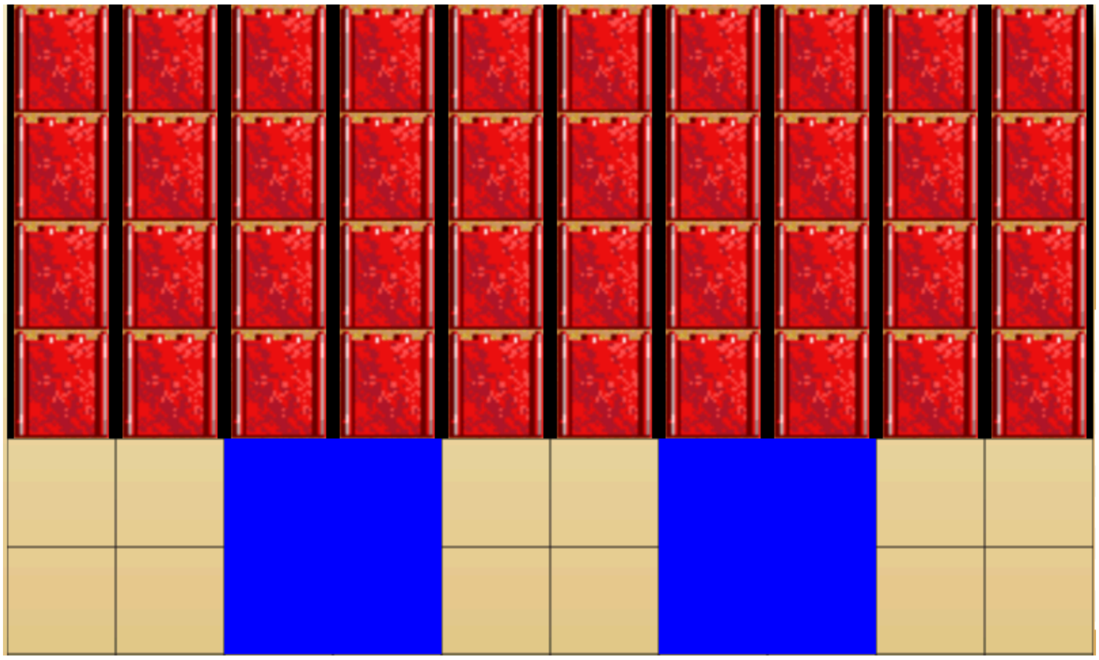
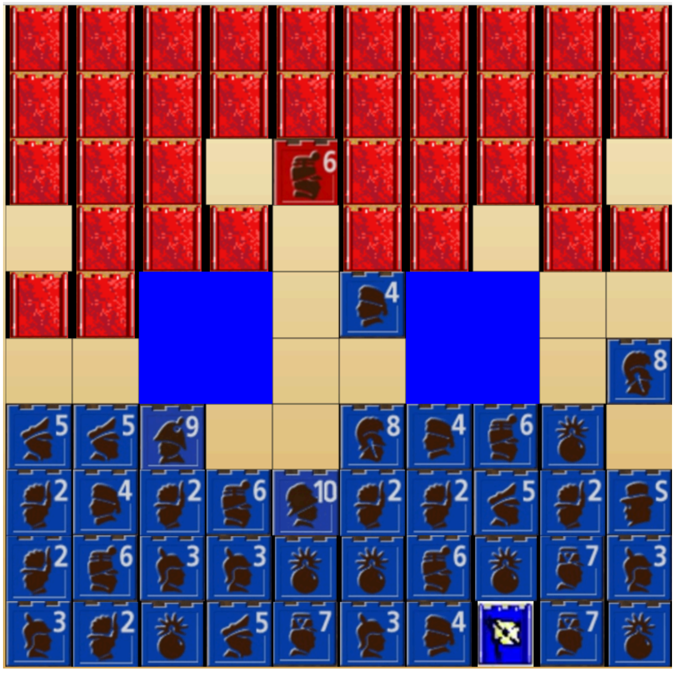


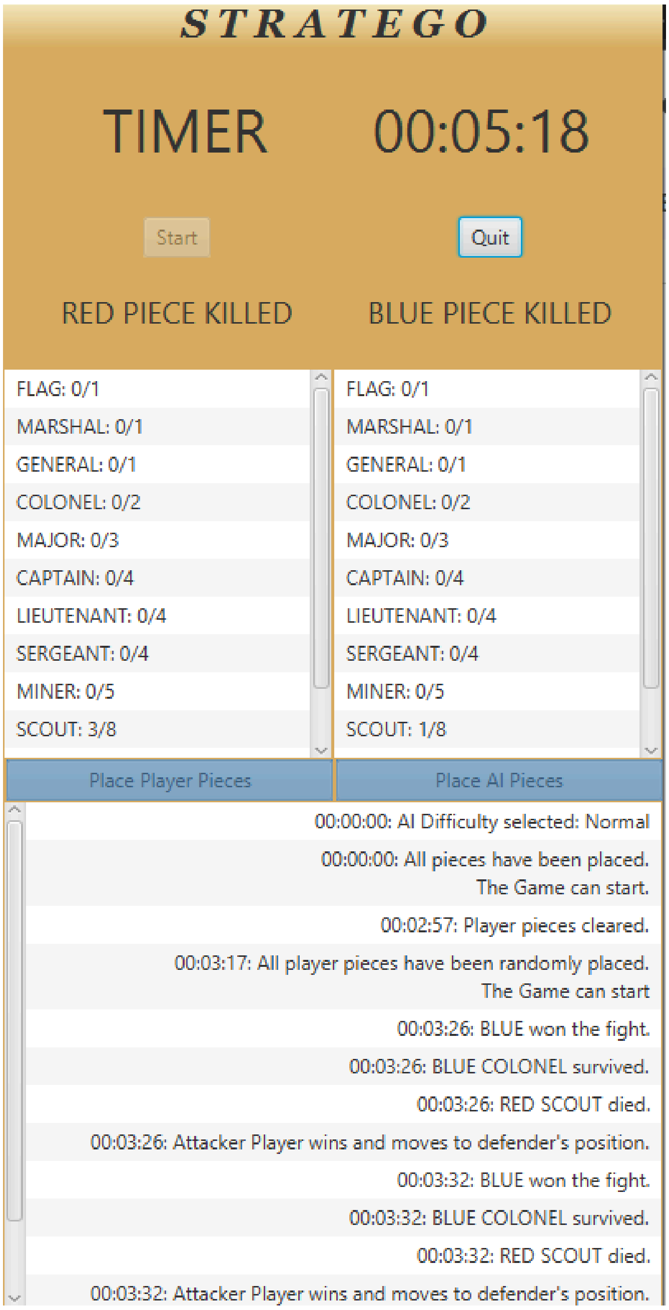
Image 4:



It can be seen in image 1 where the button is to place the AI pieces. In image 2 we can see how the button is disabled once clicked/pressed and in image 3, we can see how the 40 AI pieces have been placed on the board whilst being covered. In image 4, it can be seen that the blue player has a smaller rank as compared to the red player therefore, the red piece has been uncovered.

The history of moves and Red/Blue piece killed features give an interesting look to the game which gives a better look to the game. The history of moves display all the moves that are being done by the player or the AI against each other. The Red/Blue piece killed is a list view just like history of moves which displays how many pieces have been killed of the particular piece type. Both of these features keep updating as the game progresses. Both the features have been displayed below: It can be seen in image 1 below how the history of moves and pieces that have been killed keep updating throughout the game.

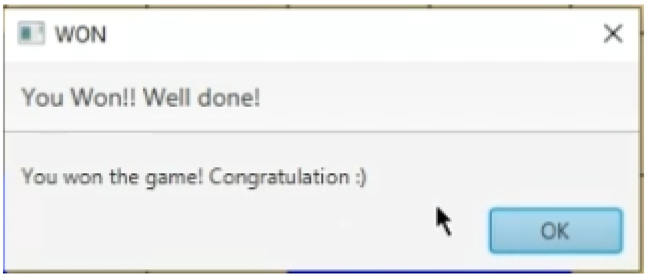
Image 1:



RED PIECE KILLED	BLUE PIECE KILLED
FLAG: 0/1	FLAG: 0/1
MARSHAL: 0/1	MARSHAL: 0/1
GENERAL: 0/1	GENERAL: 0/1
COLONEL: 0/2	COLONEL: 0/2
MAJOR: 0/3	MAJOR: 0/3
CAPTAIN: 0/4	CAPTAIN: 0/4
LIEUTENANT: 0/4	LIEUTENANT: 0/4
SERGEANT: 0/4	SERGEANT: 0/4
MINER: 0/5	MINER: 0/5
SCOUT: 3/8	SCOUT: 1/8

```
00:00:00: AI Difficulty selected: Normal
00:00:00: All pieces have been placed.
The Game can start.
00:02:57: Player pieces cleared.
00:03:17: All player pieces have been randomly placed.
The Game can start
00:03:26: BLUE won the fight.
00:03:26: BLUE COLONEL survived.
00:03:26: RED SCOUT died.
00:03:26: Attacker Player wins and moves to defender's position.
00:03:32: BLUE won the fight.
00:03:32: BLUE COLONEL survived.
00:03:32: RED SCOUT died.
00:03:32: Attacker Player wins and moves to defender's position.
```

Another important feature is the flag capture feature. As we know in the game the objective is to catch the opponent's flag as soon as possible. When the flag is captured by the player or the AI a message window pop’s up with a message inside. It can be seen below in the image:



You Won!! Well done!

You won the game! Congratulation :)

It can be seen in the image above the pop up message once the flag has been captured. When the ok button is clicked the instance of the game shuts down as the game has now been completed.

Overall, I managed to implement a lot of important features that make the user interaction even better. I add the key features that I felt are needed to be displayed in this section. All the features and functionalities have been covered in the demo video. Please watch that to find out everything about the game.

4.3 Running the application

In order to run the application please follow the given steps:

Click on PROJECT folder which is the main folder

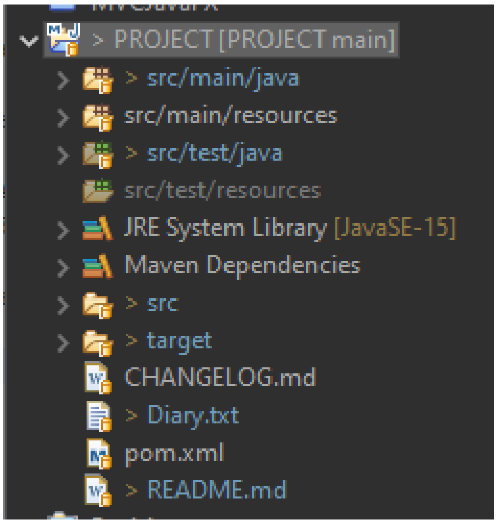
1. Under PROJECT select src/main/java folder
2. Under the src/main/java folder please select the uk.co.gurbir.PROJECT package
3. Right click on App.java class
4. From the given options select Run as
5. From Run as, Select Java application
6. The Game window should pop up which includes the board and all the pieces along with the extra features displayed.

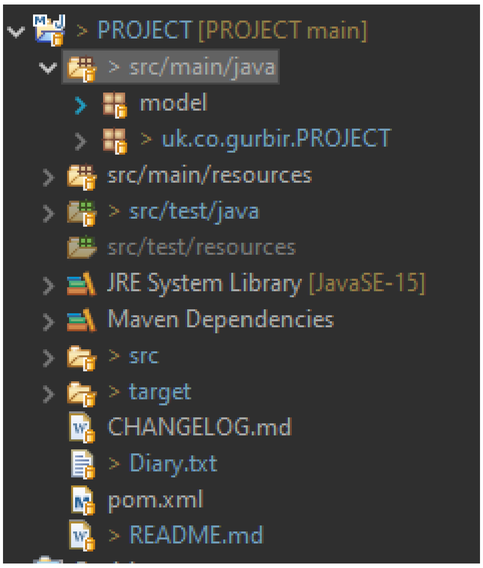
// Please ensure that you have installed the pom.xml file correctly. If the code does not work as intended.

Please select the Main Project Folder, Right click on it and From Runs As please select Run Configurations and Under the arguments section, go to VM arguments and try and replace or add the following line: --module-path

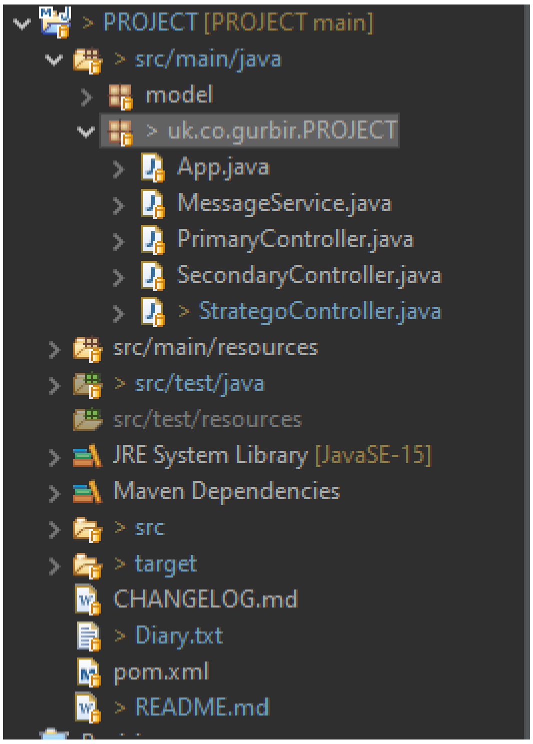
"C:\Users\Gurman\Documents\JAVALibraries\openjfx-17.0.10_windows-x64_bin-sdk\javafx-sdk-17.0.10\lib" --add-modules javafx.controls,javafx.fxml" Please make the changes to the line accordingly where you Java libraries are stored. Please ensure that if the code doesn't work on just downloading the folder. Try to have the same sdk version in order to make it run.

A visual step by step:

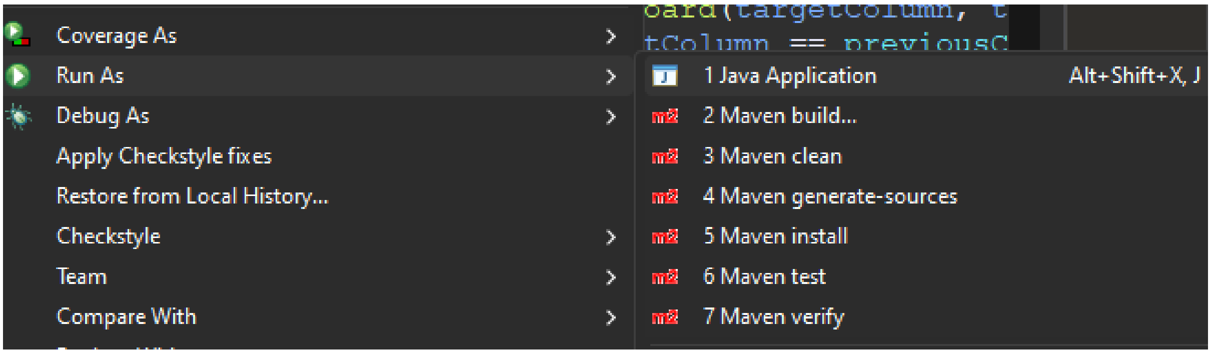
1. 

2. 

```
> PROJECT [PROJECT main]
  > src/main/java
    model
    > uk.co.gurbir.PROJECT
  src/main/resources
  > src/test/java
  src/test/resources
  JRE System Library [JavaSE-15]
  Maven Dependencies
  > src
  > target
  CHANGELOG.md
  > Diary.txt
  pom.xml
  > README.md
```

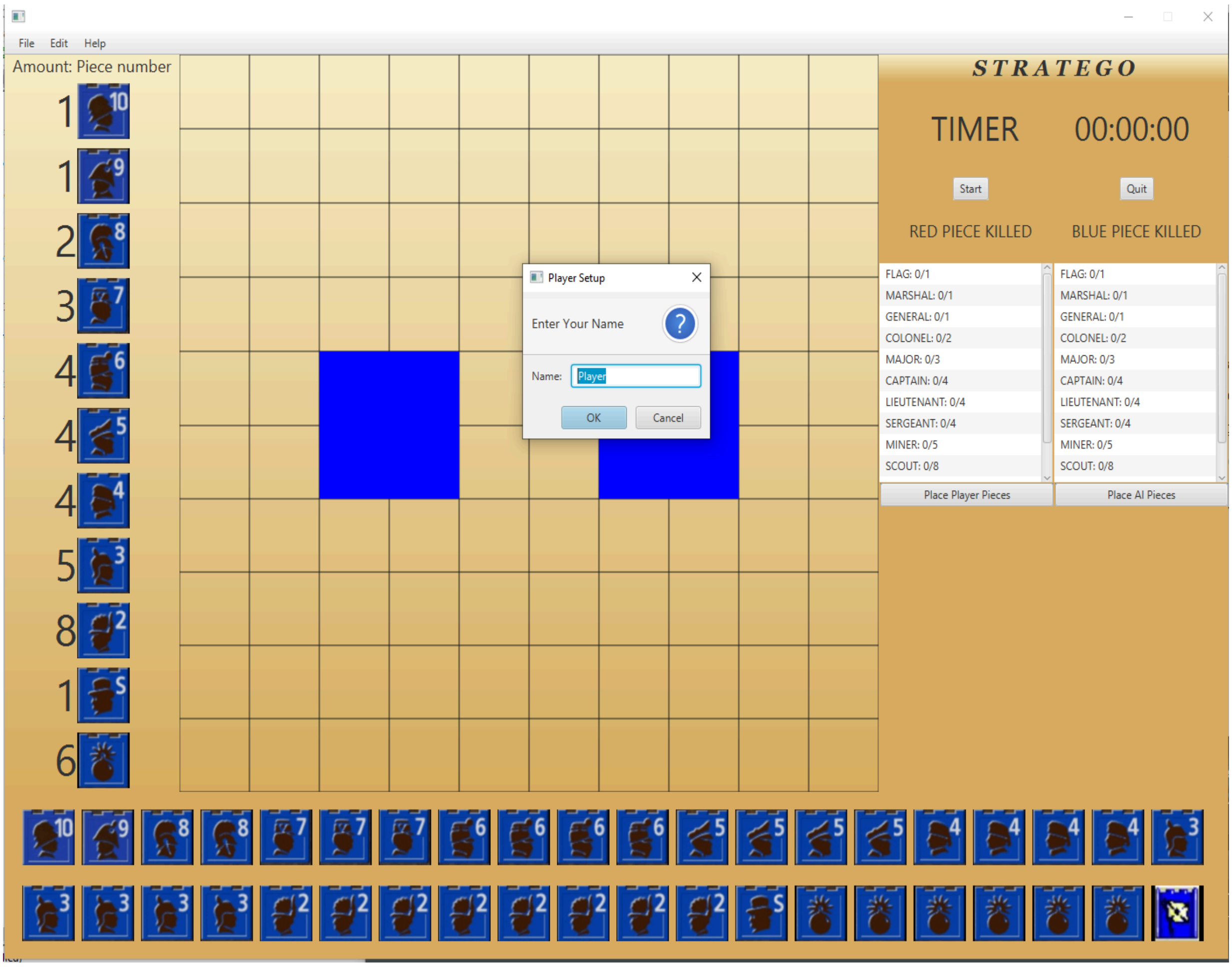
3. 

```
> PROJECT [PROJECT main]
  > src/main/java
    model
    > uk.co.gurbir.PROJECT
      App.java
      MessageService.java
      PrimaryController.java
      SecondaryController.java
      > StrategoController.java
  src/main/resources
  > src/test/java
  src/test/resources
  JRE System Library [JavaSE-15]
  Maven Dependencies
  > src
  > target
  CHANGELOG.md
  > Diary.txt
  pom.xml
  > README.md
```

4. 

```
oard(targetColumn, t
tColumn == previousC
```

5.



Amount: Piece number	
1	10
1	9
2	8
3	7
4	6
4	5
4	4
5	3
8	2
1	S
6	

Player Setup

Enter Your Name

Name: Player

RED PIECE KILLED	BLUE PIECE KILLED
FLAG: 0/1	FLAG: 0/1
MARSHAL: 0/1	MARSHAL: 0/1
GENERAL: 0/1	GENERAL: 0/1
COLONEL: 0/2	COLONEL: 0/2
MAJOR: 0/3	MAJOR: 0/3
CAPTAIN: 0/4	CAPTAIN: 0/4
LIEUTENANT: 0/4	LIEUTENANT: 0/4
SERGEANT: 0/4	SERGEANT: 0/4
MINER: 0/5	MINER: 0/5
SCOUT: 0/8	SCOUT: 0/8

4.4 Potential Future Enhancement

Given exposure to new resources, I would like to make a few future enhancements that I was not able to make this term due to either lack of time, knowledge or resources.

Firstly, I would like to add a load and save game option that allows the player to load their previous progress and also allow the player to save their current game which can be loaded later to continue. Secondly, I would like to add a homescreen that shows the badge of the game with nice wallpaper and option to start or load a saved game.

Additionally, I would like to add sound cues, essentially the theme song of the original stratego game that plays when the player is on the homescreen along with having sound cues for selecting and attacking pieces. Thirdly, I would like to code an AI that can adapt to the situation of the game and can change difficulty accordingly, basically a dynamic AI but due to lack of knowledge and practice in developing AI, I was not able to achieve it this term. Fourthly, I would like to have a button that creates multiple instances of the game and all of them running at once. I tried to implement this but it crashed my game repeatedly. I would have also liked to include an executable jar file which would make running the game much easier for the player.

Above is a list of all the future enchantments I would like to make and achieve in order to make the game even more fun and interesting.

5. Assessment

5.1 Risks and Mitigation:

All projects, including mine, have risks and challenges. It is important to identify these risks during early stages such as the development stage and make sure that a proper mitigation plan has been implemented. This section of the project plan discusses the general risks related to the project(concurrency-based game environment) and provides the optimal solutions.

5.1 Concurrency-related bugs

Risk: Concurrency-related bugs include the game crash due to overheating or the threads not being executed simultaneously.
To mitigate this risk: Code smells should be followed, and every stage should be tested using TDD where these bugs can be reviewed and fixed.

5.2 Performance Issues

Risk: The game can become laggy, slow, or less responsive due to it being inefficient concurrently. This could be caused by a lack of parallelism or excessive creation of threads.
To mitigate this risk: Specialised tools such as profiling, monitoring tools, etc. can be used to optimise and make the game more responsive and smooth.

5.3 Project Overexertion

Risk: By putting excessive effort and time on the project it can lead to a burnout which means becoming less productive, it could also raise health problems if the amount of sleep required is less than before.
To mitigate this risk: Take breaks, Divide the workload into sections and plan ahead.

5.4 Skill Gap

Risk: Having insufficient knowledge and expertise regarding a certain topic might not bring out the best results.
To mitigate this risk: Do extensive and thorough research on the topic and proper planning and setting deadlines for oneself.

5.5 Hardware failure

Risk: Important data could be lost if the main hardware device fails or gets any issue.
To mitigate this risk: Commit regularly to the git repository after completing each assigned task.

5.6 Software failure

Risk: Having a software failure could lead to race conditions.
To mitigate this risk: Synchronisation mechanisms can be used to prevent race conditions where only a particular thread should be allowed to access the shared data at the time.

5.7 Code maintenance

Risk: Due to the amount of code being a lot i.e. having many lines of code, it is easy for the developer to get confused if the code is not organised and easy to read.
To mitigate this risk: Avoid code smells,follow the programming standards,reuse code, review and refactor the code regularly.

5.8 Inadequate Documentation

Risk: Not having well written reports,instructions, explanations which would make it harder to develop the game or even give the users a hard time trying to figure out how the system works.
To mitigate this risk: Have a clear project/documentation plan of how the system works,the instructions,explanations. Reviewing the documentation again and again to see if any updates are required.

5.9 Testing complexity

Risk: Testing multiple simultaneous actions/situations can be tricky in a concurrent environment which could lead to not being able to identify errors or bugs
To mitigate this risk: Use TDD and create tests for each action to minimise the chance of having a bug and errors. If any errors occur, have an error report sent to the developer.

5.2 Professional Issues

During the development of my game Stratego, I ran into several professional issues involving ethical programming practices and use of resources, respecting intellectual property, project management issues, inclusivity in game design and more that will be discussed in this section of the report.Ethical programming practises and other issues became a professional issue through various avenues about my game Stratego:

Firstly, I had to be aware that any third-party libraries or third-party code snippets that I had used were utilised properly and responsibly while complying with proper licensing. I had to ensure that all the resources/references that I had used in my project were given proper credit which was important to avoid any further complication and potentially get plagiarised for not properly mentioning the source.

Secondly, I had to ensure that I did not change the original logic of the game, by coding it differently. So, I had to refer to a strict set of instructions to avoid facing possible legal breaches regarding the development of the game given that Stratego is a very popular and old game with existing copyrights.

Thirdly, following ethical programming practices I had to ensure that my game quality was high and the game was reliable. This meant I had to make sure that the game was free of bugs in all the instances I coded. I used a robust error handling framework which allowed me to ensure that if there was a bug, it would be caught easily and should be fixed easily as well. To make sure the methods did not include any redundant code and only contained code that was required, I followed the TDD approach and did a lot of testing, especially for the game logic.

Fourthly, I had to make sure that when implementing the AI or the player movement and other important algorithms, I had to ensure that my systems did not promote any sort of bias or discrimination. This is directly relevant to the AI, therefore I had to code the AI very carefully such that it performs the same given all scenarios.

Another professional issue that I encountered was about being transparent and honest which meant that I needed to communicate all the actions, features, how the game operates, how threads run concurrently in the game, how many threads there are in the game and more actions with utmost honesty and transparency in keeping with ethical standards. Communicating builds a sense of clarity and trust with the user/players of the game which can increase engagement thus, leading to a better user experience.

Lastly, protecting user data and privacy was important to gain the trust of users and avoid any complications such as further legal actions. I had to give careful consideration to this issue as the user's name is entered at the beginning of the game. Therefore, I decided to delete the player's name once they closed the game. I would like to store the user data in a secure server but due to lack of time, I was unable to do this.

5.3 Self evaluation

Since the beginning of the project I have gained and improved on various skills whilst learning a lot of new practices such as how to follow agile methodology and more. From the project and personal goal, I was able to achieve most of my goals. Some of the goals were the following:

- Creating a game that runs concurrently, I was successfully able to achieve this goal as the game that I have developed runs concurrently via the help of threads and also managed to make the game user friendly.

- Building a visually pleasing GUI with a lot of features such as history of moves, Randomly placing pieces. I was able to achieve this goal successfully as the game had most of the features that I wanted to implement. I was not able to implement a couple things such as creating multiple instances of the same game but I will learn about this more in the future and try to implement this in the next project I have.

- Adding documentation like JavaDoc and comments to appropriate methods and classes. I achieved this goal successfully by adding JavaDoc in almost all the classes that required JavaDoc along with adding comments where necessary.

- Being able to drag-and-drop pieces manually. I successfully achieved this feature as the player can manually place their 40 pieces on the board however they like but in addition to this the user can also place their pieces using a place player piece button which places the pieces randomly.

Above are some of the goals that I set out to achieve which I thought were significant in the development of the project but as well as developing my coding skills. Throughout this project I learnt how to thoroughly research topics and find references that are relevant to my topic. I also learned to manage my time better by dedicating time to different things and not only one. Overall, I would say that I really enjoyed making this project whilst learning key skills that I will use not only in my professional life but also in my personal life.

6. Bibliography:

[1]Polyvyanyy, A. and Bussler, C. (2013). "The Structured Phase of Concurrency." In: Bubenko, J., Krogstie, J., Pastor, O., Pernici, B., Rolland, C., and Sølvberg, A. (eds) Seminal Contributions to Information Systems Engineering. Springer, Berlin, Heidelberg. Available at: https://doi.org/10.1007/978-3-642-36926-1_20.

[2]Perolat, J., De Vylder, B., Hennes, D., Tarassov, E., Strub, F., de Boer, V., Muller, P., Connor, J.T., Burch, N., Anthony, T. and McAleer, S., 2022. Mastering the game of Stratego with model-free multiagent reinforcement learning. Science, 378(6623), pp.990-996.

[3] Oracle.,(2023). JavaFX Scene Builder. Oracle. Available at: https://www.oracle.com/java/technologies/javase/javafxscenebuilder-info.html

[4] Goetz, B., (2006). Java concurrency in practice. Pearson Education.

[5] D. Cohen. (2022). CS2800: Software Engineering(Test Driven Development)

[6] Nordic Games. (2015). Available at:

https://www.nordicgames.is/wp-content/uploads/2015/06/Stratego_Travel_Rules_EN.pdf

[7] Bloch, J. (2017). Effective java. Addison-Wesley Professional.

[8] Togelius, J. (2019). Playing smart: On games, intelligence, and artificial intelligence. MIT Press.

[9] Topley, K. (2010). JavaFX Developer's Guide. Pearson Education.

[10]Vos, J., Chin, S., Gao, W., Weaver, J., Iverson, D., Vos, J., ... & Iverson, D. (2018). Using scene builder to create a user interface. Pro JavaFX 9: A Definitive Guide to Building Desktop, Mobile, and Embedded Java Clients, 129-191.

[11] Lea, D. (2000). Concurrent programming in Java: design principles and patterns. Addison-Wesley Professional.

[12]Fowler, M. (2018). *Refactoring: improving the design of existing code*. Addison-Wesley Professional.

[13] Wikipedia contributors. (n.d.). Separation of concerns., from https://en.wikipedia.org/wiki/Separation_of_concerns

7. Planning and time-scale

In my first and second term I made considerable progress with my project and achieved most of my targets that I had set myself. I implemented majority of the game logic and GUI where the player can fully play the game with the AI.My timeline certainly changed a bit as I focused more on implementing the game logic and GUI both as they are the most essential part of the game, having a solid foundation and implementing the game logic gave me a strong setup to complete the game in the this term by implementing an aesthetically pleasing GUI.

Term 1:

- Studied and researched Java concurrency mechanisms, threads, and synchronisation.
- Understood and documented the rules and mechanics of Stratego.[6]
- Developed a comprehensive project plan, including refined abstract, timeline, and initial risk assessment.
- Created a prototype of Stratego using pseudocode, flowcharts, and user-cases.
- Set up the project in Java using Eclipse and began development, with JUnit tests to follow TDD.
- Implemented basic game features for a single instance of Stratego (board setup).
- Made preparations for presentation about the project
- Conducted comprehensive testing and debugging (using JUnit)to ensure smooth operation of implemented game system functions
- Game Logic implementation : Successfully implemented game logic for the project using threads.
- Developed fundamental features essential for game functionality, such as checking how each piece would react when fought against another piece. For eg :- Marshal vs Scout, Marshal wins
- Ensured thread safety and data consistency across game instances.
- Implemented attack features for the pieces.
- Created Use-case stories and UML diagrams.

Term 2:

- Created JUnit tests ensuring GUI elements correspond accurately to game states.
- Developed a simple GUI that interfaces with the single instance game
- Design patterns extraction to optimise the code. Code smell removal. Done this term
- Implement advanced game rules, piece movements
and win conditions for Stratego.

- Finished implementing additional threaded features, such as score tracking, piece movement, and event handling within game instances.
- Ensured all threaded operations are synchronised and data is consistent.
- Worked on improving the GUI, focusing on user experience and interactivity.
- Considered adding features like a game history viewer, interactive help.
- Added additional JUnit tests for new functionalities and ensure all tests pass.
- Evaluated code for potential optimisations and refactor where necessary.
- Addressed any code smells and refined code with design patterns.
- Debugged and refined code according to feedback and test results.
- Compiled all reports, ensuring thorough documentation of the design, implementation, testing, and any challenges encountered.
- Included a user manual for the game environment.
- Implemented a game that can run concurrently
- Implemented a basic AI that can attack, defend and do more action.

Acronyms:

RTI	Real-Time Interactions
OO	Object-Oriented
SE	Simultaneous execution
TDD	Test-driven development
MT	Multithreading
MVC	Model-View-Controller
IDE	Integrated Development Environment
GUI	Graphic user interface
AI	Artificial Intelligence

Notebook

The Notebook section of the report consists of the key points discussed and talked about with my supervisor in the meetings. The notebook has helped me keep a log of all the important topics that my supervisor discussed with me including reviews on the interim report and discussions about the project.

Meeting dates:

2nd October 2023

26th October 2023

23rd November 2023

25th January 2024

7th March 2024

Notes:
2nd October 2023: -

- Discussion of what project I will be creating around the topic of concurrency based game environments. The supervisor explained to me exactly what was expected from the project along with the meaning of concurrency: Concurrency needed to be in a single game instance and not many instances of a game.
- Discussion done about how the project plan was going as it neared the submission date.

26th October 2023:

- Discussion about Gitlab and commit frequency.
- During the meeting progress of the project so far was discussed in relation to the setup of Java using eclipse with the correct plugins.
- Also discussion on Interim report done.

23rd November 2023:

- Discussion of current progress. The progress made in the project coding wise.
- Discussion of progress of interim report
- The explanation of the deliverables explained by the supervisor.
- Missed the meeting due to technical issues but discussed these over email.

25th January 2024:

- Discussion of current progress and how the JavaFx and GUI is being implemented.
- Discussed on how to do code management better and also the importance of JavaDoc.
- The supervisor gave tips on how to manage my time better with the project.

7th March 2024:

- Discussion of the current progress on GUI and JavaFX
- Discussion regarding Final Year Report
- Discussion on the final deliverables and what needs to be added in them.
- Discussion regarding references.

Appendix: Diary

01/10/2023:
- Read and Research the rules of Stratego and watch a game online (YouTube).

02/10/2023:
I tried to play my own stratego game to get used to the rules as well as the layout and characters movement.

03/10/2023:
- Further improved my understanding of Stratego by reading some tips and tools about how to play it well and
put it into practice again.

04/10/2023:
- Set up my coding environment by downloading the latest version of Eclipse (IDE) and Java SDK as well as
testing it on a small helloworld.java project.

05/10/2023:
- Read online about Javafx and how to use it. Watched a video of someone using it to try to understand how it works.

06/10/2023:
- Set up Javafx on my Eclipse and attempted to create a small Javafx project following a tutorial.

07/10/2023:
- Further improved my understanding of JavaFX with the installation of scene builder as a helping tool.

08/10/2023:
- Day off

09/10/2023:
- Attempted a small helloworld.java project using Scene Builder and JavaFX.

10/10/2023:
- Read about Maven as a build automation tool.

15/10/2023:
- Followed a Scenebuilder tutorial online to create a small project.

16/10/2023:
- Familiarise me with the interface of scenebuilder.

17/10/2023:
- First UML class diagram draft.

18/10/2023:
- User cases draft.

19/10/2023 to 25/10/2023:
- Drafting an initial design document, outlining expected functionalities and UI expectations.

26/10/2023 to 1/11/2023:
- Started working on tutorials on how to implement threads in Java.

2/11/2023/ to 3/11/2023:
- Day off (not overworking myself)

4/11/2023 to 10/11/2023:
- Did a small project on threads and made progress on the report.

11/11/2023 to 13/11/2023:
I did thread testing and did an important commit on thread testing by learning how to create a timer for my game.

14/11/2023 to 16/11/2023:
- Started writing JUnit tests using my basic classes from the UML Class Diagram (PieceType mostly).

17/11/2023 to 27/11/2023:
- Did more research about my project and how to make it as efficient as possible using concurrency whilst making progress on the game logic implementation (Board, Square, Player, Piece).

28/11/2023 to 01/12/2023:
- Improved the game logic with the use of more advanced OOP concepts such as enums to replace some of my classes (pieceType, SquareType, …) and also did thorough testing for these classes for my game logic.

02/12/2023 to 04/12/2023:
- Made progress on the interim report and finished off my presentation for the project.

05/12/2023 to 07/12/2023:

- Implement movement for the pieces on the board.

25/01/2024 to 05/02/2024:
- Added CHANGELOG.md whilst working on implementing code which allowed the player to only move their respective piece

07/02/2024 to 05/03/2024:
- Merging of code done. A major issue/bug developed which did not allow me to work on my project for a very long time. Fixed after a lot of research.

11/03/2024 to 15/03/2024:
- Research Scene Builder and JavaFX and create initial instances of the game board whilst adding other functionalities such size panels, grid pane etc.

18/03/2024 to 26/03/2024:
- Worked on setting up the controller and linking it with JavaFX. By this point, the game board has been fully set up such as adding pieces, and rules on the game board.

27/03/2024 to 29/03/2024:
- Added code that enabled it to be drag-and-dropped on the board. Continued with implementing the link between JavaFx and game logic.

30/03/2024 to 31/03/2024:
- Implemented code which allowed the player to randomly place pieces on the game board.

01/04/2024 to 03/04/2024:
- Implement most of JavaFx whilst creating the AI and AI movement.

04/04/2024 to 05/04/2024:
- Final touches and completion of report.