Henry James Browne

Assisted Shogi

B.Sc. (Hons) Computer Science

25th March 2022

Abstract

Shogi is a two-player strategy board game also known as Japanese chess. For this project I create an assisted version of the game allowing beginner shogi players to use the program against another playing within the program using the same device or against players using a different application. For example, they can play online against a player, copying over the same moves made in the online game into the program. Furthermore, the program allows players to play the board game with all the shogi rules and fundamentals of the game, with added assisted features that allow beginner users to get better at the game, assisting them as they play with move hints and more assisted features. A repository of the complete program can be found within the files submitted; other documents used for the project such as the documents used to conduct testing for the program can be found at the end of the report. The structure of the project report can be found in the next section.

Table of Contents

1.0 Introduction

1.1 Project Aims

The aims for my project were to (1) create a user-friendly and easy-to-use graphical user interface that allows the user to play against a user on the same machine allowing them to play the shogi board game with all the rules and fundamentals of the game. (2) The program aims to allow for assisted features on-top of the base game that also assists a user on potential good moves and advising them about potential bad moves. Users should be able to play against opponents -in an online game for example- while copying the moves of the game into the application, using the application assisted features to beat users of the same level: beginner users. (3) The program's design aims to be for beginner users to help them get better and understand the game more. Because there are few assisted shogi applications, let alone assisted programs for beginners I thought this would be a useful program to create if the aims mentioned are met.

1.2 Project Scope

I first had to create the shogi game application that allows the user to play shogi within a user-friendly interface that had all the fundamental shogi features and rules: legal moves, promotions, captures, drops, legal drops, checks, checkmate along with other more subtle features of the game.

Once the base shogi game was created, I then had to create the assisted features that fundamentally help beginner shogi players understand and get better at the game: helping them beat opponents with the main feature advising the user of good and bad moves. I them had to develop the application further to bring all these features of the assisted shogi game into a user-friendly interface.

Finally, once the program was created an experiment is conducted. Participants are gathered to conduct an experiment to test the program. The results gathered will then be analysed and an evaluation of the results will be determined. Allowing for a conclusion to be determined as to if the program was successful in meeting the user requirements, the project aims and to determine if the project was successful.

1.3 Report Overview

This document will first give background information behind the project (*2.0 Background*); with a description of the shogi game and background research that has been conducted that allowed me to gather ideas and create the project.

Next the document will talk about the design of the program (*3.0 Design*) -with reference to the user requirements at the beginning (requirements created in the project proposal): this section covers each section of the program with a description of what the section does, and the code architecture used for the section: how the program works and inspiration behind ideas. This section will also talk about the graphical user interface used for sections and the reasoning behind the graphical user interface design ideas. This section also provides a UML diagram of the system.

The implementation section (*4.0 Implementation*) will then cover how the code implemented is different from the architecture mentioned and user requirements and why.

The system in operation section (*5.0 System in Operation*) covers the all the program features mentioned previously, demonstrating each section of the feature in action with screenshots and descriptions. The testing and evaluation.

The testing and evaluation section (*6.0 Testing and Evaluation*) will cover the experiment conducted to test the program, the results, and an evaluation of the results.

A conclusion (7.0 Conclusion) is then derived from the testing that determines how the overall project went. Finally references of sources used for the project can be found at the end of the document (*References*) along with the appendices (*Appendices*).

2.0 Background

2.1 Shogi Context

Shogi is the Japanese version of chess. Like chess if your king can be captured in your opponent's next turn, your king is in check and must move into safely in the next turn. However, if it cannot move into safety the king is in checkmate and the opponent wins this game (this is the aim of the game).

Furthermore, like chess pieces include:

The bishop: this piece can move any number of spaces diagonally only.

The rook: this piece can move any number of spaces vertically and horizontally.

The knight: this can move 2 spaces vertically then one space horizontally. It can also jump over pieces.

The king: this can move only one space in any direction (vertically, horizontally, or vertically).

The pawn: this piece can move one space forward.

Shogi has however the addition of pieces not also used in chess, as shown below:

The lance: The lance can move any number of spaces horizontally forward

The silver general: The silver general can move one space diagonally in any direction or one space forward.

The gold general: The gold general can move one space in any direction except diagonally backwards.

Shogi also has the addition of promoted pieces unlike chess, a piece can be promoted if it makes it to the furthest 1/3 of the board, the player can promote the piece at the end of their turn or choose not to, however if the piece makes it to the end of the opposing side of the board it must be promoted. Each promoted piece moves the same with extra additional moves.

The promoted bishop can now additionally move 1 space in any direction.

The promoted rook can now additionally move 1 space in a diagonal direction.

All other promoted pieces (silver general, knight, lance, pawn) can now additionally move the same as the gold general (one spaces in any direction except diagonally backwards).

The shogi board consists of a 9 by 9 grid of squares and the piece layout at the beginning of the game is shown in the screenshot below:

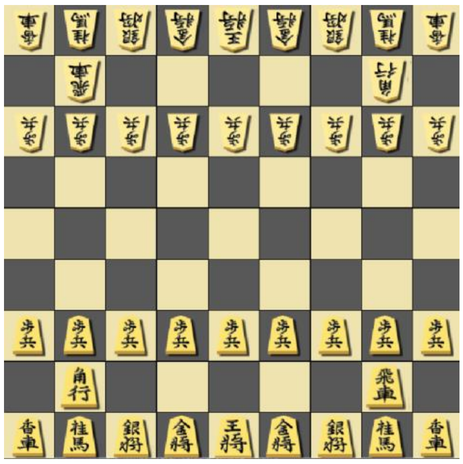


Figure 2.2-1: Screenshot showing default shogi board layout

Furthermore, in shogi pieces you have captured can be dropped anywhere on the board instead of moving a piece; the piece will be un-promoted. However, you cannot drop a pawn in the same row as another one of your un-promoted pawns. A pawn cannot be dropped to give immediate checkmate however other pieces can be dropped to achieve this. You cannot drop a piece into a space where it cannot move. If you drop a piece on a promoted zone, you cannot promote the piece until it moves on a future turn.

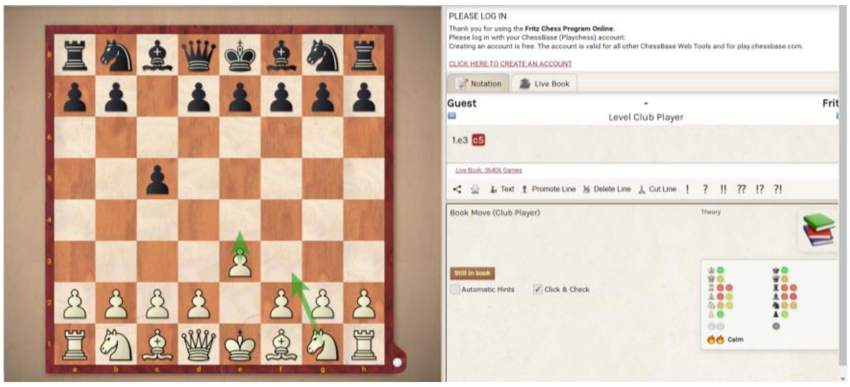
2.2 Background Scope

Unlike chess, few assisted shogi programs currently exist therefore I thought this would be a good project to create. However, because few assisted shogi programs exist currently there was not much background research I could do of existing programs to get ideas therefore to get ideas for the project I looked at existing assisted chess programs.

Having little experience playing Shogi myself but with a lot of chess experience I tried to apply my chess knowledge, along with the assisted chess program research, with a lot of research into the shogi game to get a greater understanding of the game (such as learning not only good moves but good drops also, allowing me to apply chess concepts to shogi to help me create the program.

2.3 Existing Programs

An example of an existing assisted chess program that I have found with my research is fritz chess. Fritz chess is a highly rated chess engine. The users can play a game within the application, for each move the program (if the user has the hints option turned on) using a machine learning algorithm calculates the next best moves and displays them on the screen with green arrows on the piece to the square the piece should move to. The screenshot below shows an example of fritz chess hints being displayed within the game.


PLEASE LOG IN

Thank you for using the **Fritz Chess Program Online**.
Please log in with your ChessBase (Playchess) account:
Creating an account is free. The account is valid for all other ChessBase Web Tools and for play.chessbase.com.

CLICK HERE TO CREATE AN ACCOUNT

Figure 2.3-1: Screenshot showing fritz chess example move hints

Another example of an existing assisted chess program is Lichess: this program allows users to play against users online or against AI. It also has training scenarios where users can play and the program will display good move hints when required, the move hints are displayed as arrows on the piece that point towards the square that is a good move (as shown below in Figure 2). Lichess also has a feature where you can customize a board allowing the users to create custom scenarios to play against the AI or online players.

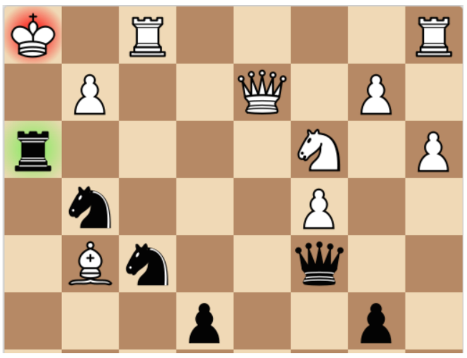


Figure 2.3-2 Screenshot showing Lichess graphical user interface

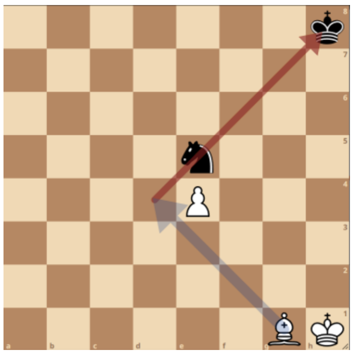


Figure 2.3-3: Screenshot showing Lichess example move hints

2.4 Existing Programs Common Features

A common GUI feature I have found from existing assisted chess programs is a very minimalist and simple graphical user interface with a common feature being used to display the hints: arrows located on the pieces to the square to represent a good move hint. Another common feature of the GUI is the ability to drag and drop pieces when making moves. Most assisted chess programs also give the option to request move hints rather than just instantaneously displaying these hints.

A lot of assisted chess programs also allow for moves to be reverted. Once the game ends, they commonly display an analysis of the game giving a rating of moves made.

From my research I have discovered a lot of widely used concepts within chess programs for example: a board representation concept used is a bit board representation: a bitboard is a 64-bit sequence of bits, a series of bitboards for each piece type, for each side, can be used to represent the board piece positions.

Moreover, another example of a widely used feature used to calculate how good moves are calculated is the concept of representing chess moves as a game tree and using tree evaluation to look moves in advance and calculate the best moves by removing the nodes of the tree that are evaluated as bad or poor moves.

3.0 Design

3.1 User Requirements

3.1.1 Game Board

- (Functional) It shall allow the user and opposing player (using the same GUI to play) to make valid/ legal moves.

- (Functional) It shall allow the user to drop captured pieces if the selected square doesn't break the rules.

- (Functional) It shall notify the user if they have been checked or checkmated, allowing them to restart the game.

- (Non-Functional) The program shall use a similar looking GUI as the one above, promoted pieces will be red to indicated they're promoted.

- (Non-Functional) It shall allow the player/players to click a piece and click the square the wish to move it to.

- (Non-Functional) It shall show users captured pieces on the left side of the board.

- (Non-Functional) It shall allow the user to click captured pieces and click a valid placement on the board to drop onto the board.

- (Non-Functional) It shall notify the user if they're in check with a pop-up message, if they're check mated an alternative pop-up message will appear and (*Design 3.8*) the user will be allowed to restart the game or quit the program.

(See Implementation 4.1 for where implementation does not meet the requirements and why)

3.1.2 Board Status

- (Functional) It shall change the state of pieces when the user tries to move it to a new square, variables that define the state of the piece include: if the piece is protected or not.

- (Functional) To detect if it is a bad move/ drop or not the program shall save the state of the board and contain an array of all the spaces the opposing players pieces can move to, (*Assisted Features 3.3*) if the square the user tries to move to is contained in the array and the piece is unprotected it will be detected as a bad
move/drop.

(See Implementation 4.2 for where implementation does not meet the requirements and why)

3.2.3 Assisted Features

- (Functional) To detect if it is a bad move / drop or not the program shall save the state of the board and contain an array of all the spaces the opposing players pieces can move to, if the square the user tries to move to is contained in the array and the piece is unprotected it will be detected as a bad move/drop.

-(Functional) To notify the user of a good move the program shall notify the user if there is an opponent's piece that is unprotected and can be captured; the array of all the spaces the opposing players pieces can move to could be used to detect if the players pieces are un-protected, if the variables that define the state of the piece suggest it is un-protected and it is contained in an array of all the possible squares the user can move to, it will be detected as a capturable piece.

- (Non-Functional) It shall assist the user by allowing them to make a move and if it is a bad move: the piece is unprotected and can be taken the user will get a prompt telling them it is a bad move and why, it will allow them to continue to make the move if they choose or (*Design 3.5*) they can cancel the move.

- (Non-Functional) It shall assist the user by showing textual messages on the right-hand side of the board, telling the user if and what piece is capturable.

(See Implementation 4.3 for where implementation does not meet the requirements and why)

3.2.4 Custom Board

(See Implementation 4.4 for where implementation does not meet the requirements and why)

3.2.5 Revert Last Move

-(Functional) If the user is notified of a bad move/ drop and uses the GUI to cancel the move, the program shall return the piece to the original state.

-(Non-Functional) It shall assist the user by allowing them to make a move and if it is a bad move: the piece is unprotected and / or can be taken the user will get a prompt telling them it is a bad move and why, it will allow them to continue to make the move if they choose or they can cancel the move.

(See Implementation 4.5 for where implementation does not meet requirements and why)

3.2.6 Menu

(See Implementation 4.6 for where implementation does not meet requirements and why)

3.2.7 In-Game Menu

- (Non-Functional) It shall notify the user if they're in check with a pop-up message, if they're check mated an alternative pop-up message will appear and the user will be allowed to restart the game or quit the program.

(See Implementation 4.7 for where implementation does not meet requirements and why)

3.2 Game Board

3.2.1 Description

When the user presses play on the menu the shogi board window is displayed. Within the game window the pieces are displayed on the board grid image with text on the instructive text on the right and a menu button top right. The user can play the game by clicking on a piece to select a piece and pressing a square on the board to move it to respectively. To unselect a piece the user can also re-click the same piece.

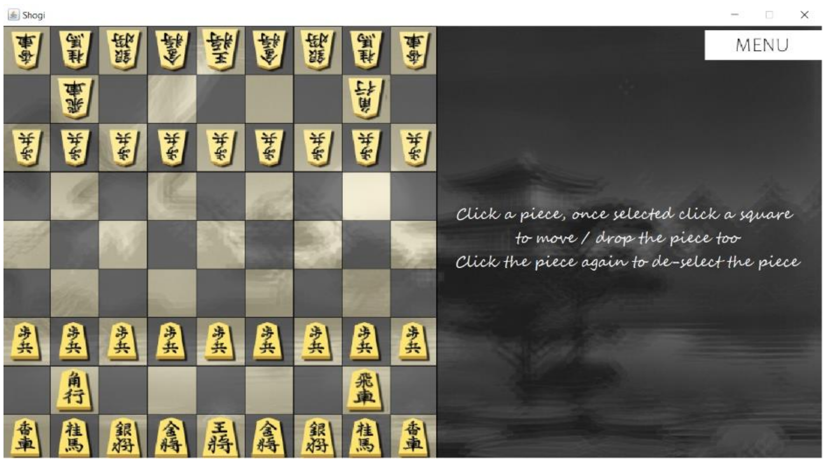
If a piece is captured on a move the piece’s colour is inverted and it is moved to the right-hand side of the board where it can be dropped, again by selecting it and clicking a valid square to drop it on.

The user cannot move to / drop on a square that is invalid; it is not a valid move for that piece if the square is not in that piece’s movement range (illegal move), it is not a valid move or drop if the square contains a piece of its colour. Pieces cannot be dropped to capture. The user also cannot move a piece that is blocking an enemy piece’s legal move toward the user colour king, as this would result in check; the king also cannot move or move to capture into check. The user cannot drop a pawn in the column of a pawn of the same colour and a pawn cannot be dropped to checkmate.

If a player is checked, the screen display goes red with a check indication being displayed on the right of the screen; the user cannot move unless that move results in the player not being in check. If the player cannot move anywhere that results in the player no longer being in check, the checkmate indication is displayed instead.

If a player moves a piece into or out of the promotion zone promotion screen / buttons are displayed where the user must choose to promote the piece or not promote the piece by clicking the corresponding button; if a player moves the piece out of the promotion zone the promotion screen is displayed.

If the menu button in the top right of the screen is pressed a menu window is displayed where the user can navigate the program (if menu is pressed the game is exited and cannot be resumed). If revert last move is pressed the last move made by the user is reverted and the board state is updated accordingly. Furthermore, if a player is checkmated, they can restart the game or quit the program using the in-game menu.



Click a piece, once selected click a square to move / drop the piece too
Click the piece again to de-select the piece

Figure 3.2-1: Screenshot of the game board screen at the beginning of a game

(See *System in Operating 5.1* for screen shots showing the Game Board features in action)

3.2.2 Code architecture

(Relevant classes: Shogigui, MenuFrame, JFrame, Menu, JComponent, MouseListener1, MouseAdapter, ImageFactory)

When the play button is pressed the game board window and user interface is generated using java swing. Upon every mouse click the board display is updated; accordingly, the array lists of image objects that make up the board interface are updated; an array list of all the pieces within the board are iterated through and the array list of the images corresponding to the images is updated dependent on the instance variables that make up the piece. For example, if a piece moves on the move mouse click the corresponding image for this piece within the array list of pieces is moved to the new x and y location of the piece. If a piece is clicked the active piece square is added to the array, to display the piece is selected.

The same update method (`drawboard`) is used to update array lists of images such as check, checkmate, promotion buttons, assisted feature images. Images within the array lists are generated using a factory class I have created that takes parameters for the image and returns the image that is stored in the arrays to be outputted. Once this method updates the array of all the board image array lists java swing is used to display the images in a new method.

Furthermore, using java – object-oriented programming made it easy to maintain lists of the board objects and the corresponding images. Moreover, using the image factory design pattern and piece objects rather than a bit representation made displaying the images a lot easier because the program is more understandable; a piece objects contains all the data used to generate the image on the screen using the image factory.

3.2.2.1 Board Class Description

Once a Board object has been initialized the board settings selected in the menu are saved as instance variables and other board variables / array lists are initialized, a grid (array list of square objects) is initialized consisting of 81 squares (9x9). Finally, the java swing (`JComponent`) constants are set to create the basis GUI and the pieces on the board grid are initialized.

Pieces and other objects are organised within the square grid using the `square_width` integer variable multiplied by the column (y) / row (x).
The board API consists of two array lists (`white_pieces` and `black_pieces`); if a custom board layout of pieces was not created from the menu (the `customPieces` array list passed as parameter is empty) the default shogi layout piece objects are initialized and added to the corresponding array list.

I used square object and piece objects to represent the board because I am using java to program the application, I took advantage of the object orientated programming features rather than using a "bit board" representation, using bits to represent squares and pieces rather than objects. Furthermore, although this approach is less optimal this made the board and pieces a lot easier to maintain and program the rest of the program, such as for the assisted features.

Mouse clicks within the board component are detected using java swing; a mouse click in the window is used to determine if a button, piece, or board square was pressed -by determining the location of the mouse click- and a response is generated.

The images that make up the board display once generated / updated (on each mouse click) and stored in array lists of image objects -as mentioned are displayed- using java swing (`paintComponent`$_1$).

Using a factory class design pattern and swing library made the program a lot more manageable and a lot easier to make the graphical user interface.

3.2.2.2 Square Class Description

The square class is contained within the board class. A `Square` object consists of instance variables `x` and `y` that define its locations within the board grid.

Squares are generally used to represent significant regions on the board for example if a piece is blocking check from an opposing piece this piece's movement range becomes limited to an array list of squares between the king and the king attacker (the piece can only move within these specified squares because if it moved to other squares this would result in the piece moving to check its own king). Furthermore, by making this class it was a lot easier to create specified regions of the board by simply creating array lists of squares.

3.2.2.3 Piece Class Description

(Relevant classes: Piece, Bishop, GoldenGeneral, King, Knight, Lance, Pawn, Rook, SilverGeneral)

A piece object is defined by its colour its location on the board grid on the x axis, its location on the board grid on the y axis, the file location of its image displayed on the grid, the board instance it is contained within, and its promoted status. By using object orientated programming because pieces in Shogi have so many defining variables it made it easy to maintain pieces throughout the program. Each piece is a parent of the piece class, however each piece variant within the shogi game has a separate child class. The inheritance here is used to diversify piece values. The value of each piece I determined using the Tanigawa (NHK 2006) piece values – I used this because from my research I found it is a widely used and accepted determinant for piece values.

The pieces legal move definitions, this is done using by method overriding the method that defines each pieces move set (`canMove`).

For example the rook has a specific piece value and ranged attack directions defined for each instance, and the `canMove` method defines a range of movement specific for the rook and promoted rook; the rook's value is 10 -unless promoted its value is 12- and can only make legal moves in a straight line: move destination must be on the same x and y axis unless promoted it can move one square in each direction as well as its unpromoted moves.

For each method that defines a piece's set movement range (`canMove`) in each `Piece` child class if the piece can legally move to the destination defined by its parameters (x and y) this method returns true, if not false.

If a piece instance Boolean variable (`is_captured`) is true, a method to determine a piece's drop range (`canBeDropped`) is used instead of the method used to determine legal drops instead of moves based on x and y parameters. This method is the same for every piece because captured pieces can legally be dropped anywhere unless it's on top of a piece or out of bounds. However, this method is overridden for the Pawn class however to make sure pawns cannot be dropped on the same column as another pawn and to make sure the pawn cannot be dropped to checkmate the king (checked using the `isCheckMateMove` method in Piece).

Using object-oriented programming as mentioned made representing pieces a lot easier and more manageable; here polymorphism is used to easily allow the creation of different pieces within the program to have different move and drop variations, as mentioned. Inheritance is also very useful here because it allows for pieces to share a lot of method used within the program, adding to the efficiency of the program. A lot of chess programs I have looked at use bit maps to represent pieces however because I am using object-oriented programming, I thought I would use array lists of objects to represent the board rather than a bit representation, although a bit board representation would make the program more optimized.

3.2.2.4 Piece Legal Moves

For each piece legal moves are determined by if the move is in the piece's movement set as explained previously and if the move is not out of bounds, not on top of a same colour piece and if the move does not check the same colour king, if all conditions are met the move is legal.

To check if the move does not check the same colour king a method is used to check if a piece moving to a specified position results in the king of its own colour being directly threatened / checked. Furthermore, if a piece is blocking check the piece can only move to the square between the piece it is blocking check from and the king.

This method is overridden within the king class and only returns false if the king does not move into any opposing pieces attacking squares or if the move to square contains a piece the piece must be unprotected for the king to be able to move their / capture the piece. Furthermore, to not allow the king to move into check / any squares that are dangerous. Moreover, here is another example of were using polymorphism to add variation to the king piece, without having to re-write code.

Moreover, the squares class created is fundamental for the features mentioned because array lists of squares are used to define the squares where pieces blocking check can legally move to and array lists of squares are used to determine what squares are safe for the king to move to.

3.2.2.5 Making a move and promotions

Mouse clicks are detected within the board window as mentioned; the location of each mouse click within the window is used to determine what piece is clicked, dependent on the sequence of mouse clicks a response is generated and the method that updates the display of board updates the display accordingly.

Every time an un-promoted piece is moved into the promotion zone the promotion button display is toggled by changing the value of a Boolean variable. Mouse clicks are now used to determine if the user chose to promote or not promote the piece, dependent on the button the response is generated and displayed -using the method mentioned. If a piece is promoted a Boolean variable for the piece instance is toggled to change the movement range and the promoted image corresponding to the piece is retrieved and displayed when the board is updated after the move -as mentioned. The piece remains in an array list until it moves out of the promotion zone, allowing the piece to still be promoted if it moves out of the zone.

A common feature within chess programs (*Background 2.3*) is a drag and drop option I did not use this feature because this is difficult to implement using java swing and because of the time constraint on the project I did not feel it to be necessary.

3.2.2.6 Drops and captures

If a player moves to a square that contains a piece, a method is called that sets that piece to captured: a Boolean instance variable is changed to true to represent this and it is moved to the left of the board in an organized row with the rest of the captured pieces. The movement range of this piece is now determined by a different method: canBeDropped rather than canMove. This method takes an x and y parameters and determines if a piece can legally be dropped to that square; this method is the same for every piece however is overridden for the pawn because pawns have additional drop rules to other pieces: pawns cannot be placed in the same column as same colour pawns, and they cannot be dropped to checkmate.

3.2.2.7 Check

To determine check the program checks for each piece (if it is not captured) if it can move legally to the opposing king's position the player who's playing as that piece's colour is in check. Additionally, piece cannot make moves that as a result would check the same colour king, this is checked in the method that checks for legal moves for each piece (called every time the user tries to make a move - in the `canMove` method), as mentioned previously.

3.2.2.8 Check mate

To determine check mate the program checks if there are no moves out of check and a player is check then check mate is determinded.

3.3 Board Status

3.3.1 Description

The board status updates the values within the board that are used to calculate legal moves and assisted features; for example, it updates the pieces that block to check, if a piece is blocking checking the user can only move this piece in a way that would still result in the piece blocking check or move in a way that removes the threat.

3.3.2 Code Architecture

Every time a move is made all the update methods to update all the instance variables (for each piece instance), board variables and array lists that make up the status of the board are called.

Every time a move is made the status of the board is updated: all the variables that make up the status of the board are updated / re-calculated. These variables include instance variables for each `Piece` and `Board` variables such as the array lists of attacking squares (`Square` objects) for each player.

I found it very useful maintaining board status variables throughout the program because these variables are frequently used throughout the program, meaning other features of the program do not have to re-calculate these variables and instead can just easily retrieve them for example:

The white and black attacking square array lists are used to make sure the king cannot move into check and the array lists containing the moves / drops out of check with another array list containing the corresponding pieces for each move and drop are used to only allow pieces to block check, if a king is checked and they're used to generate the best moves out of check in the assisted features.

3.4 Assisted Features

3.4.1 Description

If the tutorial setting was selected in the menu, text on the right of the screen is displayed when a piece is clicked to guide beginner users: showing them the (legal) movement range of that piece and the value of the piece (Low, Medium, or High).

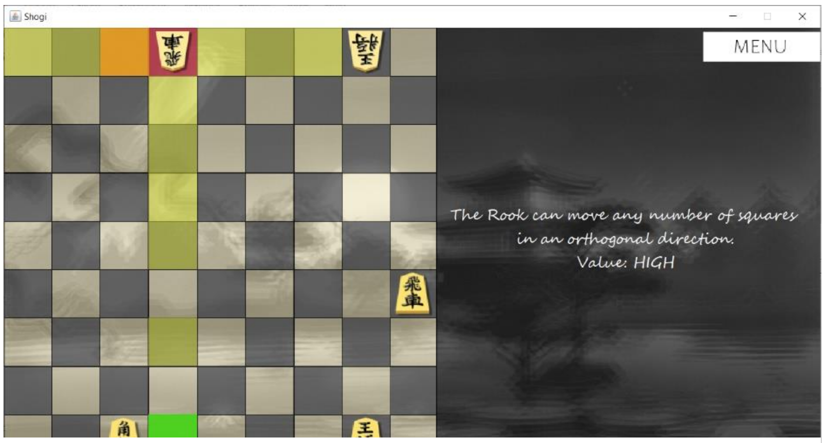
If the hints settings were selected in the menu, dependent on what colour pieces the user selected to play as in the menu:

If one of the user's pieces can capture an opposing piece the opposing pieces square will also be green (as this is a great move) it will also display a green arrow on the piece that can capture this piece guiding the user where to move to capture the enemy piece. The squares of the user's colour piece that can be captured by an opposing piece are displayed as red with a black arrow being display on top of the piece showing what direction to move the piece to safety.

If their colour piece is selected safe / good, great and amazing moves are calculated for the piece and displayed as yellow or green squares on the board: safe / good moves are defined as moves that would not threaten the piece being moved (it doesn't become capturable), good moves are defined as moves that threaten a piece (an opposing piece becomes capturable) and great moves are defined as moves that threaten one or more pieces (one or more opposing pieces become capturable), as this move results in a capture on the next move. Safe / good moves are represented by yellow squares, great moves are represented by orange squares and amazing moves are represented by green squares (similar to opposing capturable squares because amazing moves result in a capture).

If the user's-coloured pieces are checked good and great possible moves out of check are calculated with green arrows on pieces pointing towards greens squares indicating where to move for a great move out of check (capture of checking piece) and black arrows on pieces pointing towards yellow squares indicating where to move for a good move out of check (blocks that don't result in pieces becoming capturable or king moves) -if there are no good or great moves out of check nothing is displayed.

If there is a possible checkmate move somewhere on the board once the user makes a move, this is calculated, and a gold arrow is displayed on the piece / pieces that have a possible checkmate move and the square the piece should move to is displayed as gold. If the opposing player has a possible checkmate move against the user, this is also displayed similarly with a warning graphic square and red arrow from the piece to the square.



The Rook can move any number of squares in an orthogonal direction.
Value: HIGH

Figure 3.4-1: Screenshot showing an example of the assisted features move hints

(See *System in Operating 5.3* for screen shots showing the Assisted features in action)

3.4.2 Code Architecture

3.4.2.1 Assisted Features Fundamentals

(Relevant classes: AssistedFeatures, Arrow, ImageFactory)

The Assisted Features API works similarly to the Board API: once an instance is created within the board class, a method is called (`generateAssistedImages`) this method calls various methods that generate the assisted images, these images are then retrieved by this method stored in various array lists that can retrieved and displayed within the board class using an accessor method once generated using the method mentioned.

3.4.2.2 Tutorial Text

The tutorial text once updated displays the default tutorial image unless a piece is selected then the corresponding text is generated.

3.4.2.3 Capturable pieces

The pseudo code below explains the algorithm that is used to determine if a piece is capturable:

```
// check if piece is unprotected and can be captured
IF piece.getAttackers().size() > 0 AND piece.is_protected() = FALSE THEN
        capturablePieces.APPEND(piece)
        RETURN
END IF
// check if lower value piece can capture highter value piece (sacrafice)
IF piece.getAttackers().size() > 0 THEN
        FOR attacker IN piece.getAttackers():
                IF attacker.getValue() < piece.getValue() THEN
                        capturablePieces.APPEND(piece);
                        piece.addCaptureWith(attacker);
                        RETURN
                END IF
        END FOR
END IF
// calculate capturable pieces depending on exchange value calculated from the exchange resultant
// from if all the pieces attackers capture the piece and all //its following defenders
// calc xray attackersDefenders and xray Defenders
Defenders -> piece.getDefenders();
Attackers -> piece.getAttackers();
exchangeDefenders -> getXrayDefenders(piece)
exchangeAttackers -> getXrayAttackers(piece)
exchangeDefenders.ADD(Defenders);
exchangeAttackers.ADD(Attackers);
IF exchangeAttackers != NULL AND exchangeAttackers.size() > 0 AND exchangeDefenders != NULL
AND exchangeDefenders.size() > 0 AND exchangeAttackers.size() >= exchangeDefenders.size() + 1
THEN
exchangeValue -> piece.getValue();
        FOR x = 0 TO x < exchangeDefenders.size():
                exchangeValue -> exchangeValue - exchangeAttackers.get(x).getValue();
        END FOR
        FOR defender IN exchangeDefenders:
                exchangeValue -> exchangeValue + defender.getValue()
        END FOR
        IF exchangeValue > 0 THEN
                capturablePieces.add(piece)
                RETURN
        END IF
END IF
```

Furthermore, to determine if a piece is capturable and added the array of capturable pieces (`capturablePieces`): if a piece is unprotected and can be captured (if it has more than 1 attacker and is not protected it is capturable -added to the array).

If a piece of lower value can capture a higher value piece that piece is capturable -added to the array of capturable pieces- because the lower value piece, even if the piece is protected, should sacrifice itself to capture this piece and the piece that should capture it is added to the pieces array list `captureWith` (later in the program an arrow is added to this piece to indicate this to the user).

Finally, the algorithm also calculates exchanges to determine if a piece is capturable: depending on the exchange value calculated from the exchange resultant if all the pieces attackers capture the piece along with all its following defenders, if the exchange value is greater than 0 then the player captures higher value pieces from the possible exchange making the piece that may initiate the exchange capturable.

Moreover, to calculate this exchange value the method first must retrieve all the x-ray attackers and x-ray defenders – these pieces attack / defend a piece indirectly, through other pieces. This is done with the methods used to calculate x-ray attacker and x-ray defenders (`getXrayAttackers` and `getXrayDefenders`) these methods iterate through all the pieces attackers / defenders and for each attacker / defender it iterates through each square on the axis from the defender / attacker to the end of the grid in the direction of the piece to the attacker/defender, respectively to find x-ray attackers/defenders. For example, to find x-ray defenders: for each square tending towards the outbounds squares of the board grid in the direction of the piece towards its defender, if the square contains a piece and the piece Boolean negates the statement is of opposing colour and the piece is not in its movement range, end the loop, if not add to array and continue iterating through squares tending towards the squares out of the board grid boundaries.

Once a pieces Xray defenders and Xray attackers have been determined these pieces are added the attackers and defenders array lists respectively (these array lists contain the current pieces attackers and defenders -excluding Xray defenders and Xray attackers (using get method in piece - `getDefenders` and `getAttackers`). If the amount of attackers is the same or equal to the amount of defenders, the initial exchange value (`exchangeValue`) is set to the piece's (the piece the method is checking is capturable) value, the exchange value is then incremented by the value of all the defenders that would be captured in the exchange and decremented by all the value of all the attackers that would be lost in the exchange (size of attackers = (size of attackers – size of defenders).

If the exchange value calculated is greater than 0, the exchange resultant if all the pieces attackers capture the piece along with all its following defenders, would therefore benefit the player; the piece is determined as capturable and is added to the array list of capturable pieces.

X-ray attackers and defenders are not added to the pieces array lists of attackers and defenders previously because they're generally only relevant when calculating these exchanges.

3.4.2.4 Good moves and drops

To calculate good moves and drops the program for every possible legal move / drop (if the piece is captured) a piece can make it checks if the square is safe, to check this it sets the possible of the piece to the location of every possible legal move / drop, at each position it then uses the calculate capturable pieces algorithm to check if this piece becomes capturable at that location, if so that move is not safe and there is not a good move, if it doesn't because capturable it is a good move /

drop. This is a good move because it does not result in the opposing players advantage because they cannot capture a piece.

3.4.2.5 Great moves and drops

To calculate great moves and drops the program for every possible legal move / drop (if the piece is captured) a piece can make, it first checks if the move /drop is safe (using the method mentioned in good moves and drops). It then checks if the move threats an opposing a piece; using a similar method to check if the move is safe but this time instead of checking does the move result in this piece becoming capturable, does it instead result in an opposing piece becoming capturable (using the calculate capturable pieces algorithm mentioned). This is a good move because it doesn't result in the opposing player's advantage because they cannot capture a piece and it threatens an opposing piece meaning the opposing player must respond.

3.4.2.6 Amazing moves and drops

Building on the previously mentioned method to calculate good and great moves / drops, to calculate amazing moves it works very similar to how great moves are calculated however instead of checking if the move is safe and results in one opposing piece becoming capturable it checks if two opposing pieces become capturable. This is a good move because it doesn't result in the opposing players advantage because they cannot capture a piece and it would almost definitely result in a capture on the next move because the opposing player cannot move two pieces at once (the move threats two pieces at once).

3.4.2.7 Get out of Check Hints

When a player is in check the program iterates through all the possible moves out of check (retrieved from the board status variables mentioned previously in *Board Status 3.3.2*) if the king can move out of check this is added to the array of good moves out of check. Otherwise for every other piece if the move is a good move (determined by the method mentioned previously) the move is added to the array (this move also has the potential to be a great or amazing move). If the move get out of check square contains a piece and the piece is capturable this is a great / amazing move because this piece must be checking the king (because it's contained in the moves out of check array), so the player gets out of check and captures an opposing piece that was determined as capturable by the method mentioned previously. Once the moves have been determined / added to the array list the array list of arrow images is updated accordingly.

Here is another example of where the board status variable in this case the array list containing all moves out of check and the methods used for features mentioned previously were very useful.

3.4.2.8 Checkmate hints

To calculate checkmate hints the program for every possible legal move / drop (if the piece is captured) a piece can make, a method in the piece class is used (`isCheckMateMove`) where it temporarily sets the pieces position on the board grid to that move square location and updates the board status, if this results in a player being in checkmate then it return true if not it returns false, once the pieces position has been reverted and the board status is updated again. If it is a checkmate move (check mate method (`isCheckMateMove`) returns true) the move square is added to an array list of check mate moves and an arrow is added to a list of checkmate arrows (that is located on the piece pointing in the direction of the checkmate square (to allow the user to know where to move to block a checkmate or move to checkmate).

I used the method in Piece to check for check mate because this method is also used to make sure pawns cannot be dropped to checkmate the king. The board status feature is very useful here.

3.4.3 Design Inspiration

Chess algorithms I have looked at to calculate the best moves use leaf evaluation to look multiple moves in advance and removing the leaf nodes that represent bad and poor moves (*Background 2.0*).

I did not use leaf evaluation to calculate good moves because the program is made for beginner shogi players, I did not feel it was necessary to look multiple moves in advance to calculate the best move, instead the program just calculates the best moves dependent on the immediate outcome of the move / exchange that would take place, as a felt this was sufficient for beginner shogi players who would most likely not be thinking multiple moves in advance; helping them learn to see immediate good moves to potentially develop them to think moves in advance.

Furthermore, the leaf evaluation using very complex search algorithms that also implements aspects of machine learning, which I did not have time to do and as mentioned did not think it was necessary and too complex for the beginner shogi players trying to learn and understand the game.

The design for the assisted features took inspiration from existing chess programs discovered in my research (*Background 2.3*) for example I used arrows a common feature of other assisted chess programs such as fritz to display move hints, this adds to the minimalistic design. However, because the program does not calculate the best possible moves the range of moves that are recommended with the move's hints are more varied, therefore I used coloured squares to represent this variety of moves: colour coding each possible move, rather than just displaying the best move. I did this because the program is for beginner users, I wanted to display a range of (good, great, amazing) possible moves allowing the user to choose for themselves which move to do, rather than just showing them the best possible moves. I did this because this allows beginner users to learn more about the game.

Another common feature of assisted chess programs includes the ability to request move hints rather therefore I added the option to turn the hints on or off within the program menu.

Furthermore, assisting chess programs such as fritz chess use machine learning algorithms to calculate the best moves that have taken years to create. I simply did not have time to make a similar algorithm, therefore I created algorithms that calculate the best moves determined by the current state of the board: when calculating move hints the program does not look moves ahead (apart from when calculating exchanges). As mentioned, I thought this would be sufficient for the beginner users because the beginner user would not be looking multiple moves in advance therefore by displaying move hints that are calculated by doing this, it could be too advanced for the user, and they may not understand why the move hints are recommended. This would not help them to learn.

3.5 Custom Board

3.5.1 Description

When the user clicks the customize board button in the menu, the customizable board screen is displayed like the game board screen. Text instructions are displayed on the right telling the user how to customize the board layout using this screen. Furthermore, the customizable board functionality is like the game board, the user can select and de-select pieces by clicking on them and clicking on them again to de-select. However, pieces on the right once placed remain on the right of the screen to be placed again

If the user clicks a piece, they can place the piece anywhere on the board; pieces cannot be placed on top of each other, pawns cannot be placed on the same column and pieces cannot be placed if it results in them not being able to move. If a piece that has been placed on the board is selected the delete button can be pressed to remove this piece.

The white turn button when clicked can be inverted to black turns and vis versa allowing the user to choose what players turn it is when the game is played.

If more than 1 of white/black king is placed or less than 1 of white/black king is placed on the board and the play button is pressed the text on the right of the screen instruct the user that they must place one of each king to play. If these conditions are met and the play button is pressed the game board is displayed with the piece layout on the board that the user created; dependent on the turn selected the user can now move that coloured piece and play the game with the custom piece layout they created.

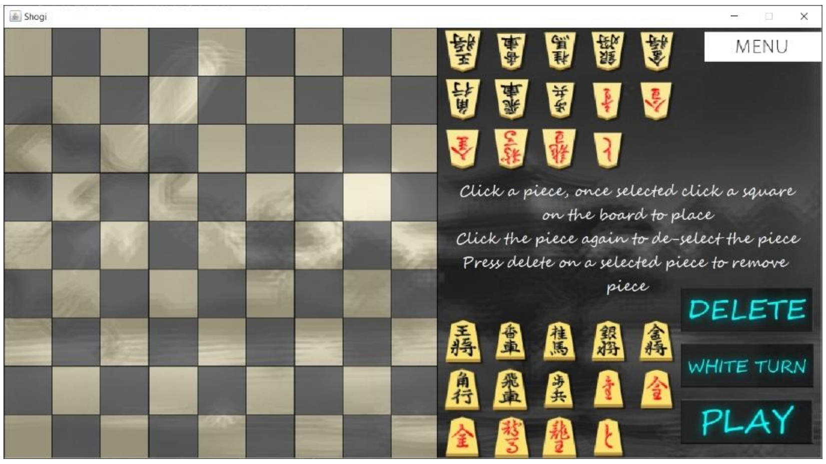

Click a piece, once selected click a square on the board to place
Click the piece again to de-select the piece
Press delete on a selected piece to remove piece

Figure 3.5-1: Screenshot showing the custom board screen

(See *System in Operating 5.4* for screen shots showing the Custom Board features in action)

3.5.2 Code architecture

(Relevant classes: Board, CustomBoard, ImageFactory)

3.5.2.1 Custom Board Class

The `CustomBoard` class is a child class of the Board class therefore inherits most the methods and therefore functionality of the Board class. However, some methods are overridden to change the functionality of this class for example the method that initializes the pieces in the `Board` class is overridden to change the array list of pieces that create the default shogi piece layout within the game board to instead contain pieces in the custom board layout (as shown in Figure 3.5-1).

Rather than creating a base new class the custom board feature, instead by using inheritance and making this a child class of the board parent class. It meant I could utilize polymorphism and only had override methods to create this feature, making this code efficient by making use of these object-orientated features.

3.5.2.1 Button press detection and response

Because the `CustomBoard` class is a child class of the `Board` class the button detection and response is done similarly using java swing. However, the event handling method is overridden to allow for the variation of features with this class.

3.5.2.2 Moving custom pieces
To move custom pieces that the user can place on the board are all set to captured therefore the method that determines legal drops (*Design 3.2.2.6*) is used to determine where the user can place custom pieces on the board. They can drop pieces anywhere on the board however they cannot place pieces on top of each over or out of bounds, pawns cannot be placed in the same column as same colour pawns for example.

3.5.2.3 Creating a board game instance from custom board piece layout and turn selected
An array lists of pieces that have been placed on the custom board is generated and updated every time the user places a piece. The variables that define the custom board include the players turn selected and this array list of pieces, these variables are stored within the menu instance using set methods. Once the user has created the custom board and clicks the play button a `Board` instance is created, and the variables stored in the menu are retrieved and used as parameters for the board instance. Allowing the layout of pieces created by the user (stored in the array list mentioned) to be played within the game board rather than the default shogi board layout (the array list of pieces initialized when the game is played without the user created a custom board).

3.5.3 Design Inspiration
The custom board feature took inspiration from assisted chess program I have researched for example: Lichess has a board editor function that I found to be a useful feature of assisted chess programs. It allows users to play scenarios that they may have previously lost using the assisted features of this program to help them see where they could have done better. I also found this feature was very useful when testing and debugging the program.

3.6 Revert Last Move / Board States

3.6.1 Description
When the user clicks the menu button in a board component, they have the option to revert the last move made pressing the revert last move button; once this button is pressed the menu is closed and the piece layout of the board is reverted to the layout of the board before the last move was made. This button can be clicked multiple times until the starting layout of pieces / start of the game where no moves were made, unless a move was reverted, and another move was made after the revert.

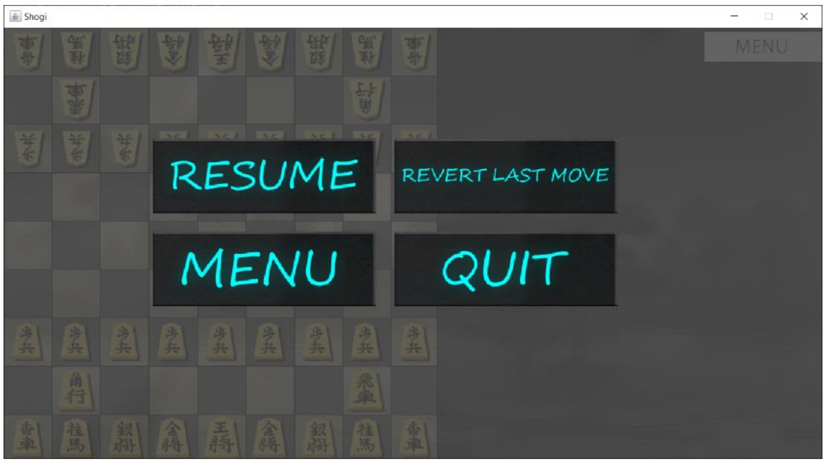


Figure 3.6-1: Screenshot showing in-game menu screen to display the revert last move option

(See *System in Operating 5.5* for screen shots showing the Revert last move features in action)

3.6.2 Code architecture

When the board pieces are initialized and every time the user makes a move a method to clone and store the state of the board is called; this method clones the array list of all pieces on the board (and adds this array to an array containing arrays of pieces (each board state); this array stores the state of the board pieces / layout at every move.

When the revert last move button is clicked a method is called and the last move is reverted this is done by: first the move counter by 1 (move counter is used to determine the players turn) and the last board state (array list of pieces) is removed and the array list of all the board pieces is set the array list of pieces before the last board within the board states array. Once the pieces have been changed to the layout of pieces before the last move was made the board status is updated accordingly.

This feature allows the player to revert multiple moves until the state of the board is back to the beginning unless a move is made directly after a revert. If a move is reverted a Boolean variable is set to true, if another move is made after a revert the array list of board states is reset, only allowing the player to revert a move if another move was not made after the revert, to fix bugs that occurred with this feature.

Furthermore, here is another example of why using OO programming and piece objects contained within array lists is very useful; the state of the board / piece layout for each move can be stored easily.

3.6.3 Design Inspiration

A common feature within assisted chess programs is the ability to revert moves (*Background 2.4*) therefore I found this feature of the game was a fundamental part of my assisted shogi program.

3.7 Menu

3.7.1 Description

When the user runs the program the game menu is displayed, here the program takes user input using this easy-to-use graphical user interface I have created; the menu takes user input by detecting mouse input, if a button on the menu is clicked a corresponding response is generated.

Furthermore, the menu screen allows the user to navigate the program: they can play the assisted-shogi game by clicking the play button, they can quit the program by pressing the quit button or they can create a custom shogi scenario to play instead of playing assisted-shogi from the shogi default initial scenario / board layout (when play is pressed). The user can press the info button to find out more about the program (i.e., controls and rules). The user can also customise settings for their assisted-shogi game when played; they can select on and off features: hints (the assisted feature of the game, giving the play hints where to move) and tutorials (the feature that tells the user how to play, for example where pieces can move within the basic shogi game). Finally, from the menu screen the user can also select what colour pieces they're playing as; dependant on the colour selected / colour pieces the user is playing as, hints will be generated to assist that colour pieces.

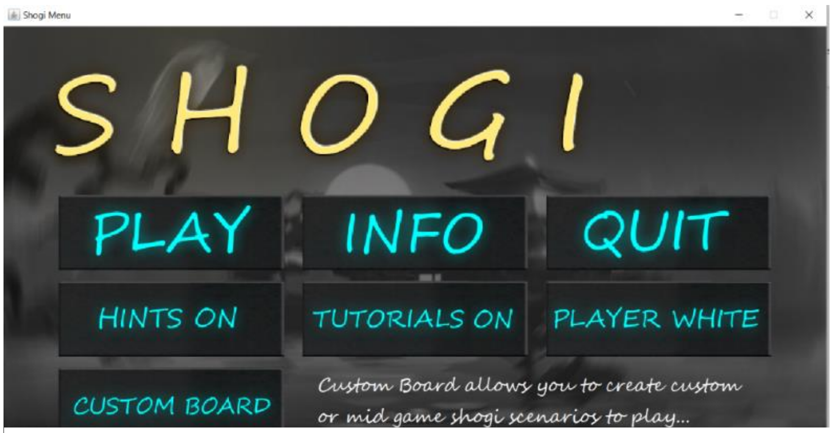

Shogi Menu
SHOGI
PLAY
INFO
QUIT
HINTS ON
TUTORIALS ON
PLAYER WHITE
CUSTOM BOARD
Custom Board allows you to create custom or mid game shogi scenarios to play...

Figure 3.7-1: Screenshot of the program menu screen

(See *System in Operating 5.6* for screen shots showing the Menu features in action)

3.7.2 Code architecture

(Relevant classes: Shogigui, MenuFrame, JFrame, Menu, JComponent, MouseListener2, MouseAdapter, ImageFactory)

When the program is run, the java main method (within the `Shogigui` class) creates a menu window and graphical user interface using java swing. The menu class works similarly to the board class in that the images are created using the image factory class and are stored in array lists of image objects. Once a mouse click is detected within the window the region of the mouse click is used to determine what button was pressed and the corresponding response is generated. (If play or custom board were not pressed) The images within the arrays are displayed using java swing (paintComponent$_1$).

Furthermore, the swing library made button press detection and displaying images very easy with the help of the image factory class I had created (using the factory class design pattern), to generate the response I simply had to change Boolean variables and array lists of images that make up the menu display.

3.7.3 Menu Graphical User Interface Design

The user interface design of the menu uses key, fundamental concepts of design principles and human computer interaction. Examples of design principles used here are minimalism, consistency, and error prevention. Furthermore, the menu interface is made up of a simple background that allows the title and buttons to stand out with the use of depth for the buttons and bold colours. The menu interactive features / buttons are in a neat layout and not scattered with a consistent design (with the same font and button general design) adding to the minimalist and easy to use design. Finally, the interface uses error prevention as it does not allow for incorrect user input that would break the program; the widgets accept only valid input; for example, to select the game setting the user simply presses the buttons to toggle the setting with the buttons display changing accordingly to show this.

3.8 In-Game menu

3.8.1 Description

When the user press's the menu button in the top right within the game window the in-game menu is displayed: here the user can resume the game by clicking the resume button, they can revert the last move made within the game by clicking the revert last move button, they can return to the menu by clicking the menu button or the user can quit the program by clicking the quit button. The design used is similar to the program menu: consistent and minimalist.

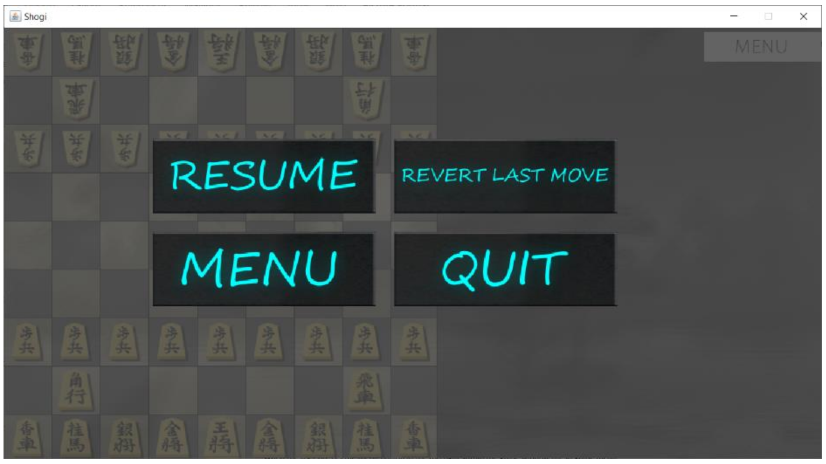


Figure 3.8-1: Screenshot showing the In-Game menu being displayed

(See *System in Operating 5.7* for screen shots showing the In-Game Menu features in action)

3.8.2 Code architecture

When a menu button click is detected within the board instance a Boolean variable is toggled, meaning the mouse click events are now handled within the in-game menu class and the in-game menu images are displayed. Once a button click is detected within the region of a button, the button clicked is determined and the corresponding response is generated. For example, if resume is pressed the Boolean variable is toggled again meaning the in-game menu images are no longer displayed and the mouse click events are handled within the board class.

Furthermore, instead of writing a separate interface for the in-game menu I used the board interface however just change the images within this interface and re-located the event handling method to an event handling method within the in-game menu class, by changing a Boolean variable to true rather than false.

3.9 Game Board, Custom Board, and In-game Menu Graphical User Interface

For the game graphical user interface, I took inspiration form the Nielsen's 10 Design Heuristics where necessary: The interface allows for control and freedom, allowing the user to easily navigate the program using the in-game menu.

The interface is consistent; all the buttons are the same design, and the background has the same design within each window (game and menu), the only button without a consistent design it the menu button within the game, I did this to allow the menu button to stand out from the rest of the in-game features as a feature that does not add to the game but is fundamental.

The interface uses Error Recovery; if the user makes a mistake (i.e., makes the wrong move) the menu allows the user to revert this error with the revert last move feature. If the user selects the wrong piece to move, they can also simply de-select this piece. If the user selects the wrong settings in the menu, they can easily go back to menu using the in-game menu and change the settings. Another example of error recovery is used within the custom board: if the user tries to play a game scenario without one of each king, the game does not allow this breaks the rule of shogi.

The interface uses recognition; the info page tells the user what each move hint colour represents. These colours were chosen as they are mostly universal colours that represent good, bad and great; for each move hint the colour was chosen as its universal representation matches the move hint; for example, if the move is a good move the colour used to represent this move square is green (as green is the universal colour for good). Furthermore, by using colours universal colours accordingly for the move hints and by telling the user what each colour means, they can easily recognise what the good, great, and bad squares to move to / move away from are shown by the move hints are. With the consistency used in the design recognition for the user is also made simpler.

The interface is flexible & efficient; the user can easily navigate the program using the in-game menu and menu screen where navigation and settings are easily selected as mentioned; the user can easily play the game by selecting, de-selecting, dropping, and moving pieces in an efficient way; simply by click on pieces to select and de-select, once selected they can click a square to move. Allowing the user to de-select a piece adds to the flexibility of the interface also.

The interface uses Minimalism; the game consists of the pieces, board, background, and buttons with consistency used without add to the minimalist design of the game. Furthermore, I tried to keep the tutorials and move hints are minimalist are possible, for the hints I used only basic arrows and coloured squares to keep it simple, with neat minimalistic images being displayed as tutorials for select pieces. Furthermore, the interactive features (i.e., pieces and buttons) are kept as minimal as possible while still allowing the interface to have all the fundamental aspects: for example, pieces that are easy to move and a program that is easy to navigate.

The interface uses Error Prevention, the interface was designed to not allow for input that causes error; the users input can only be valid because the game only consists of mouse input that depending on the location in the game generates a response. If the user makes an error, it can be easily recovered as mentioned. An example of error prevention used within the game the user cannot make illegal moves (for example moves that are no in the legal move range of the piece they're trying to move, they cannot make a move that results in them being put into check ext.…) the game does not allow this, making the interface robust also.

The interface provides the user with help where possible, even if the tutorial settings were not selected the user is still given default tutorial text on the right of the board guiding them on how to use the interface; within the menu the user can also press the info button to get more guidance on how to use the interface.

4.0 Implementation

4.1 Game Board

The game board and base shogi game (without an assisted features) was implemented as mentioned (*Design 3.2*) and meets all the requirements (*Design 3.1.1*) within the project proposal with very minor variation (for example capturable pieces are displayed on the right instead of left); the shogi game implemented works as intended with all the fundamental features and functionalities of shogi: the game allows the user and opposing player (using the same GUI to play) to make valid moves, it allows the user to drop captured pieces if the selected square doesn't break the rules, the user is notified if they have been checkmated, allowing them to restart the game, allow the player/players to click a piece and click the square the wish to move it to, show users captured pieces on the left side of the board. It allows the user to click captured pieces and click a valid placement on the board to drop onto the board, the user is notified if they're in check with a pop-up message, if they're check mated an alternative pop-up message appears allowing the user to press buttons to restart the game or quit the program.

Implementing the base shogi game was not too difficult to implement, the main problems and bugs came when implement the check and checkmate features of the game. Implementing the check feature was fairly straight forward however the problems came with deriving valid moves when in check for example I had to implement features / methods that: detect if pieces can block check, detecting if a king moves into and out of check, detecting if pieces moves results in king being put in check, detecting if king is checked by two pieces at the same time (double check) and then detecting how move that stop this, detecting if pieces are blocking check if so don't allow them to move into spaces that doesn't block the check, detecting if a drop doesn't stop check when in check, ext.… Furthermore, the check functionality within the game was more complex than anticipated and was slighter more difficult to implement with lots of small bugs occurring that had to be fixed. The checkmate feature was also slightly challenging to implement however with the methods and variables in place for the check features, it was not too difficult.

4.2 Board Status

The Game Board Status was implemented as mentioned (*Design 3.3*). The methods and variables that define board status meet all the requirements (*Design 3.1.2*) in the project proposal such as the program changes the state of pieces when the user tries to move it to a new square, variables that define the state of the piece include: if the piece is protected or not and if the piece can be promoted or not and an array of all the spaces the opposing players pieces can move to. However, in the implementation I added a lot more variables that define the board status and methods to update these variables such as: `is_blocking_check` (Boolean), `is_blocking_check_from` (Piece), Attackers (array list of Pieces), Defenders (array list of Pieces). Along with the board status variables updated when a player is in check: the `getOutOfcheckMoves`, `getOutOfCheckDrops`, (the array list of pieces) `getOutOfcheckMovePiece` and `getOutOfCheckDropPiece`.

I used the instance variables `is_blocking_check` and `is_blocking_check_from` to determine if a piece can legally move without checking the same colour king, because if I used a different method to do this such as: setting a piece's location to a move, updating the board status and if the same colour king is checked as a result: it is not a legal move. This method did not work because it creates an infinite loop because when the program checks for check the `canMove` method is used and in the `canMove` method this is where the move is checked if it is legal or not / if it checks the same colour king.

Furthermore, a lot more variables were needed to define the status of the board than anticipated to allow for the base game to work as intended and to add the assisted features (variables mainly used for the check functionalities). Implementing these extra variables that make up the board status along with the update method that calculate these variables was challenging and took a long of debugging, however everything works as intended. Moreover, this was difficult because the algorithms that were used to derive and update these variables had to be carefully thought out and had to work perfectly because these variables as mentioned are used a lot in the program for to allow for various functionalities of the program to work, therefore required a lot of testing.

The main problem that occurred with the board status was when refactoring the program to separate functionalities into different classes to make the code more structured and readable, I tried to separate the board status into a separate class; that when initialized would update all the variables that define the board status, these variables could then be retrieved using access method from the newly created board status instance. I was not able to this however because it created far too many bugs, therefore eventually I just left this program functionality un-refactored; to allow the program to continue to work as expected.

4.3 Assisted Features

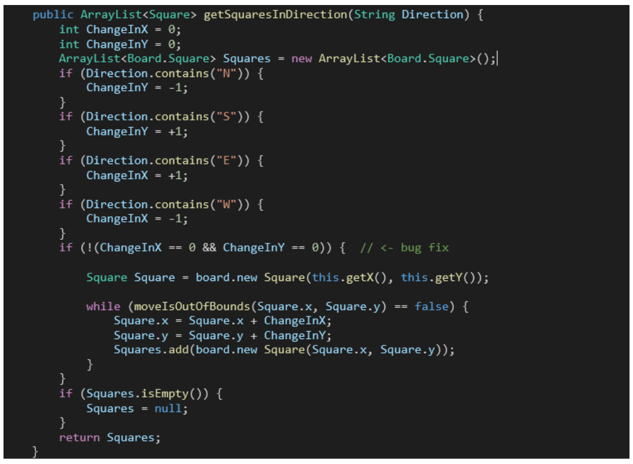
The assisted feature was implemented as mentioned (*Design 3.4*) and differ a fair bit from the requirements (*Design 3.1.3*); instead of detecting bad moves, instead the implementation shows the user all the good, great, and amazing moves instead, furthermore, allowing the user to avoid bad moves and make a lot better moves also.

Capturable pieces are calculated as specified in the requirements however instead of a text display showing the user what pieces are capturable a display on the board is instead used with arrows pointing to highlighted (in green) capturable opposing pieces and the users capturable pieces (dependent on the colour pieces they're playing as are high) are highlighted in red. I did implement the text display to indicate to the user capturable pieces however I found this was a lot more difficult to read and understand than the colourful display I went for to show this in the final implementation. I also made it so that these assisted hints can be turned on and off within the menu, allowing the user to play without these assisted hints or not; I did this because some users may just want to play the base game.

In assisted features I also added tutorials that can be selected on and off in the menu, this was not in the requirements, but I thought because this program is made for beginner users it could be very helpful in allowing them to learn and understand more where shogi pieces can legally move too. Moreover, the tutorial text was first displayed as tutorial images however I felt using text made the program more presentable and was still easy to understand.

The assisted features were quite difficult to implement; for example, the algorithm to determine capturable pieces too a lot of planning and the implementation took a lot of testing. The hardest part of designing this algorithm was calculating piece exchanges (calculating the exchange value calculated from the exchange resultant if all the pieces attackers capture the piece and all its following defenders) to determine if a piece if capturable, determining x-ray defenders and x-ray attackers / pieces behind pieces that contribute to the exchange was also quite difficult. The value of each piece I determined using the Tanigawa (NHK 2006) piece values. Once I had the capturable pieces method working calculating good, great, and amazing moves was not too difficult as the capturable pieces method was used to derive most of these moves.

A big problem during implementation came when implementing the calculate capturable pieces method; when calculating piece exchange, and when determining the x-ray attackers and x-ray defenders this created a memory leak bug that kept crashing the program. First, I was not sure where this bug was created in the program, but I narrowed it down to this method and found that the bug was created in the `getSquaresInDirection` method in the Piece class. Furthermore, this memory bug occurred because if the direction specified in this method did not contain any of the strings "N", "S","E","W" the change in y and change in x would remain at 0 therefore the same square was constantly added to the return array and the loop never ended; crashing the program with a memory error - bug fix is shown below.



```java
public ArrayList<Square> getSquaresInDirection(String Direction) {
    int ChangeInX = 0;
    int ChangeInY = 0;
    ArrayList<Board.Square> Squares = new ArrayList<Board.Square>();
    if (Direction.contains("N")) {
        ChangeInY = -1;
    }
    if (Direction.contains("S")) {
        ChangeInY = +1;
    }
    if (Direction.contains("E")) {
        ChangeInX = +1;
    }
    if (Direction.contains("W")) {
        ChangeInX = -1;
    }
    if (!(ChangeInX == 0 && ChangeInY == 0)) {  // <- bug fix

        Square Square = board.new Square(this.getX(), this.getY());

        while (moveIsOutOfBounds(Square.x, Square.y) == false) {
            Square.x = Square.x + ChangeInX;
            Square.y = Square.y + ChangeInY;
            Squares.add(board.new Square(Square.x, Square.y));
        }
    }
    if (Squares.isEmpty()) {
        Squares = null;
    }
    return Squares;
}
```

Figure 4.3-1: Screenshot showing code where memory leak bug occurred, showing the bug fix as a comment in the code.

This bug took a while to fix, but when it was fixed and the algorithm was complete, implemented and tested for the calculate capturable pieces method the rest of the assisted features functionality was not too difficult to implement.

Finally, I did not plan on going as in depth as I did with the assisted feature hints, but I am very happy with the assisted features part of the program, and it works as intended. Furthermore, for the calculate capturable pieces method I created a much more advanced algorithm than intended; this method is used to derive good, great, and amazing moves, therefore this algorithm had to be made as precise as possible. Furthermore, I went more in depth with this feature then intended because (1) I found I had given myself a lot of time to complete this section within the plan and (2) I found even for beginner users the original idea for just good and bad move hints was too simple and found that by providing a range of different move rather than just good moves (good, great, amazing moves) would allow for greater understand of the game and game play.

In addition, in the future I could use the advanced calculate capturable pieces method I have created to potentially calculate good, great, and amazing moves in advance. In the future with more time I would also implement a game and moves made analysis feature at the end of each game similarly to other assisted chess programs I have looked at.

4.4 Custom Board

The custom board feature was implemented as mentioned (*Design 3.5*); it was not in the requirements (*Design 3.1.4*) in the project proposal, however I thought this would be a good feature to add, because not only did it help me with testing and debugging the program but I felt it was also a very useful feature for users because it allows them to create and play custom scenarios, allowing them to not only use this program for full shogi games but they can also use it for different shogi game scenarios or use it in a mid-shogi game also.

This feature was partially straight forward to implement because I used inheritance from the main shogi board class already created therefore, I only had to create / override a couple of methods.

4.5 Revert Last Move / Board States

The revert last moves / board states feature was implemented as mentioned (*Design 3.6*) and runs as expected it meets the requirements (*Design 3.1.5*) in the project proposal this feature allow to not only take back bad moves but all moves, they have made, to allow them to recover from mistakes.

This feature was partially straightforward to implemented because the program was made using object-oriented programming; I simply made a copy of the array list of pieces on the board at the beginning of the game and at every move, allowing moves to be reverted by simply retrieving the array of pieces before the last move.

A problem occurred with this feature however when the user reverted a move and then moved again and reverted that move, the program did not function as expected and a bug occurred. Un-sure on why this bug occurred to prevent it I detect this test case and reset the board states if this happens; allowing the user to revert moves but once they move again after reverting a move, they cannot revert that move.

4.6 Menu

The menu was implemented as mentioned (*Design 3.7*); it was not in the project proposal requirements (*Design 3.1.6*); the menu was implemented after the game board was created when implementing the game board, I felt the menu was a necessary and useful feature, as it allows the user to customize their game scenario. This is important because the shogi game has a lot of extra features such as tutorials that a no beginner user may not want to use. The menu is also important because by allowing the user to select what colour pieces they're playing as and generate hints corresponding to their colour meant that the program doesn't have to generate the hints for both colour pieces, adding to optimization. Furthermore, another useful feature of the menu that I felt was very necessary for the game is the information page; as mentioned the game has a lot of features and extra features: explaining these features and how the game works / how to play I felt was very important for the useability of the program and the GUI design Heuristics concepts implemented as mentioned previously.

The menu was not partially difficult to implement because having implemented a lot of the game board already I was very familiar with creating a GUI in java with the swing library; a lot of the API was very similar to API implemented in the game board. Furthermore, no significant problems were encountered.

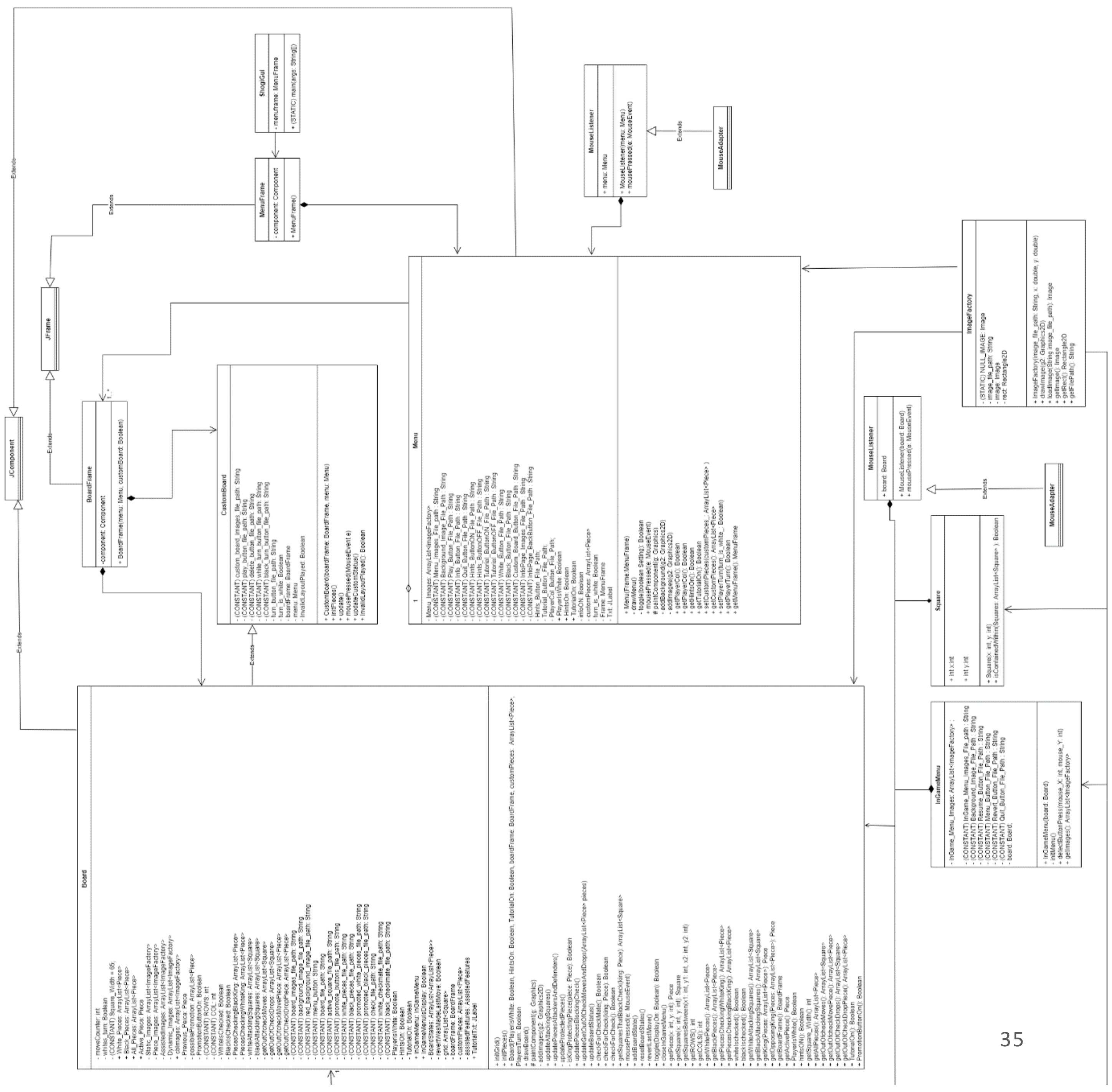
4.7 In-Game menu

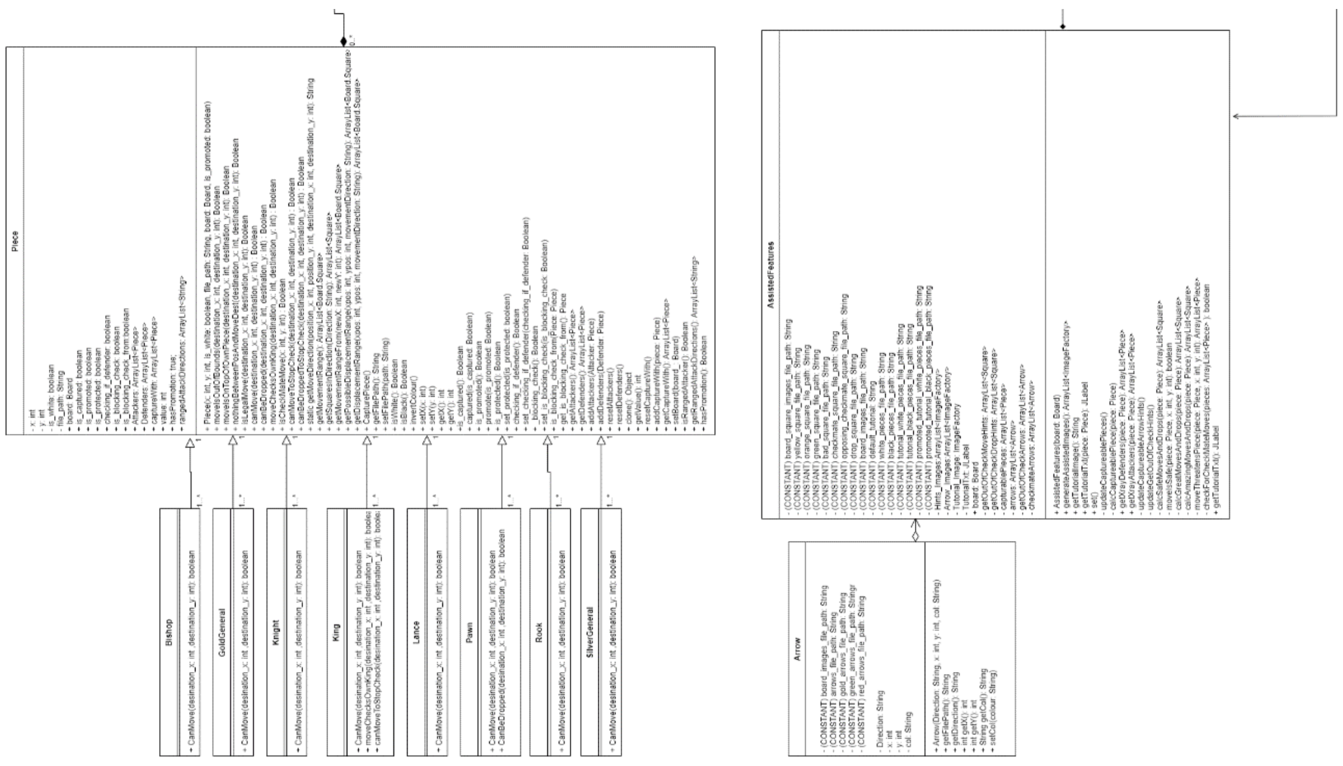
The In-Game menu was implemented as mentioned (*Design 3.8*); it was not in the requirements (*Design 3.1.7*) in the project proposal however because the program was larger than expected I thought it was a fundamental part of the program for useability; the user can easily navigate all the features of the program. This was relatively easy to implement, once designed the implementation was similar to the rest of the implementation for the user interface in the rest of the program (using java swing).

4.8 Generating Images

Images are generated as mentioned in the code architecture; the requirements in the project proposal did not specify how I would generate image and the graphical user interface, however in the implementation I used java swing library to do this because it is a commonly used and well-known library. Furthermore, to generate images I created my own factory class using the factory class design pattern, as mentioned in the code architecture. I encountered few problems with implementing the graphic user interface and images because I have used the Swing library and the factory design pattern before. The program I have designed using swing and my own designed images looks and runs as expected.

4.9 Program UML Diagram



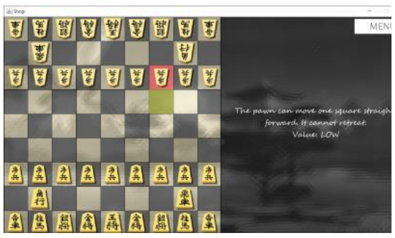

Piece
Bishop
GoldGeneral
Knight
King
Lance
Pawn
Rook
SilverGeneral
AssistedFeatures
Arrow

5.0 System in Operation

5.1 Game Board

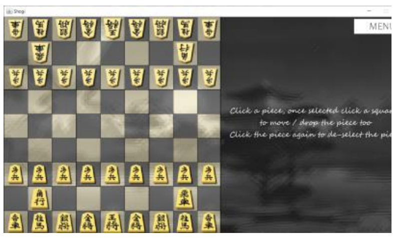
The screen shots below show a piece (white pawn) being selected, un-selected, and moved:

1.



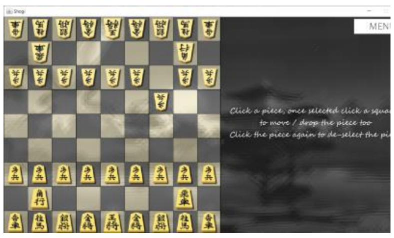
The pawn can move one square straigh
forward. It cannot retreat.
Value: LOW

2.



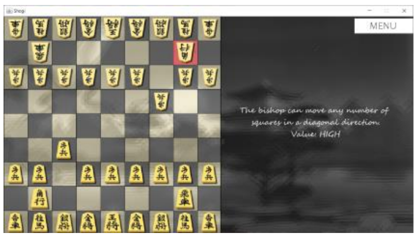
Click a piece, once selected click a squa
to move / drop the piece too
Click the piece again to de-select the pi

3.



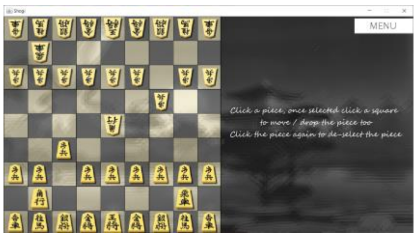
Click a piece, once selected click a squa
to move / drop the piece too
Click the piece again to de-select the pi

4.



The bishop can move any number of
squares in a diagonal direction.
Value: HIGH

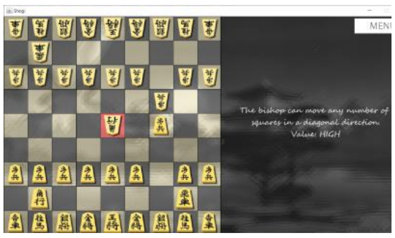
5.



Click a piece, once selected click a square
to move / drop the piece too
Click the piece again to de-select the piece

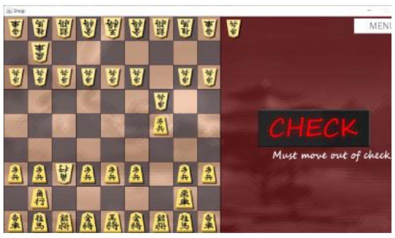
The screen shots below show a capture and check – white bishop moves to capture black pawn and check black king; the black king then moves out of check (in this scenario the player chooses not to promote the bishop):

1.



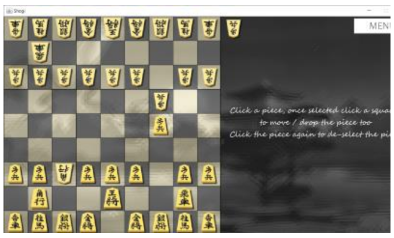
The bishop can move any number of
squares in a diagonal direction.
Value: HIGH

2.



CHECK
Must move out of check

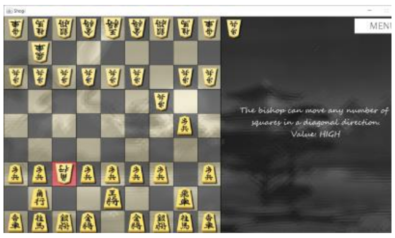
3.



Click a piece, once selected click a squa
to move / drop the piece too
Click the piece again to de-select the pi

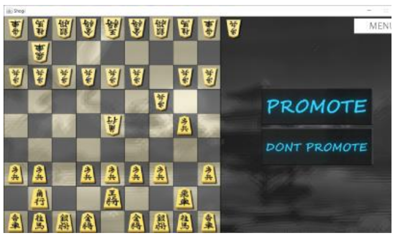
The screen shots below show a promotion - a white bishop moves out of the promotion zone and gets promoted:

1.

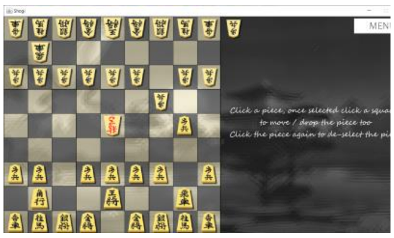


The bishop can move any number of
squares in a diagonal direction.
Value: HIGH

2.



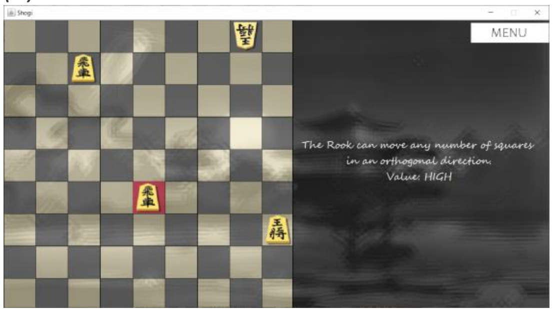
3.



Click a piece, once selected click a squa
to move / drop the piece too
Click the piece again to de-select the pi

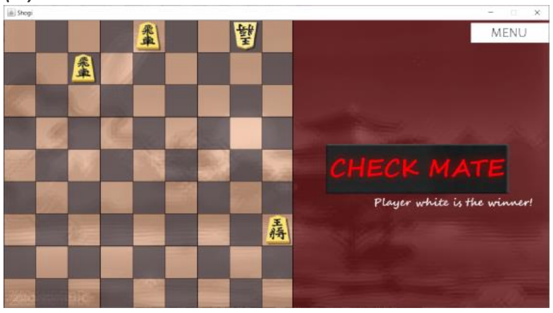
The screen shots below show various check mates – (1) Two black Rooks checkmate white king, (2) black Rook and black promoted pawn checkmate white king, (3) White Promoted pawn moves to checkmate black king (protected by white bishop):

(1)



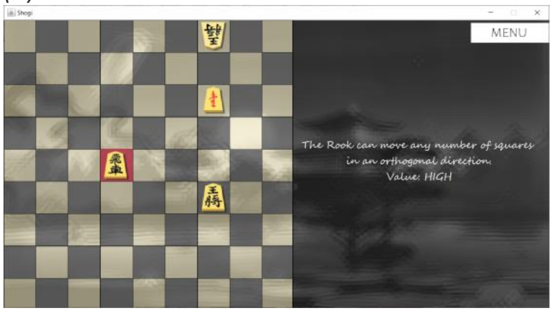
The Rook can move any number of squares in an orthogonal direction.
Value: HIGH

(2)



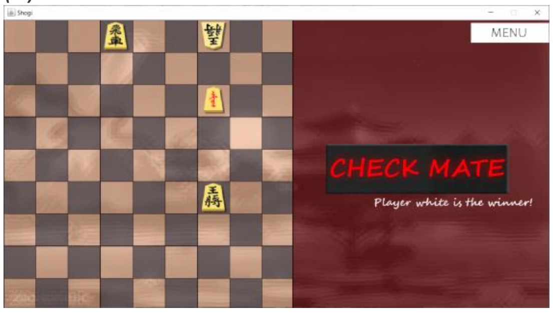
CHECK MATE
Player white is the winner!

(1)



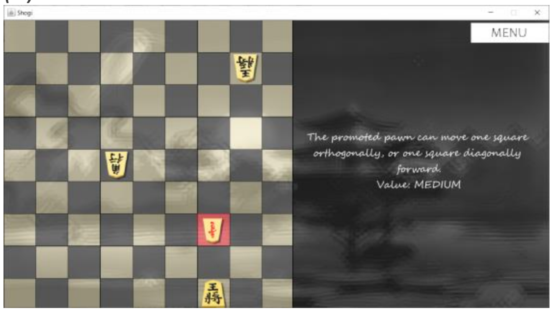
The Rook can move any number of squares in an orthogonal direction.
Value: HIGH

(2)



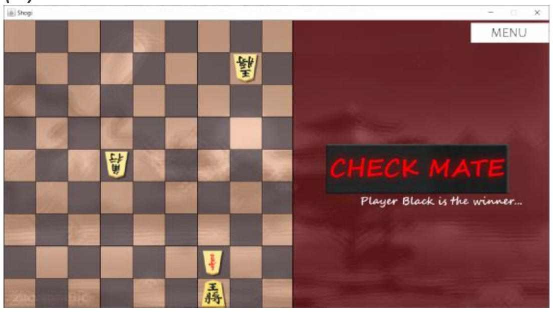
CHECK MATE
Player white is the winner!

(1)



The promoted pawn can move one square orthogonally, or one square diagonally forward.
Value: MEDIUM

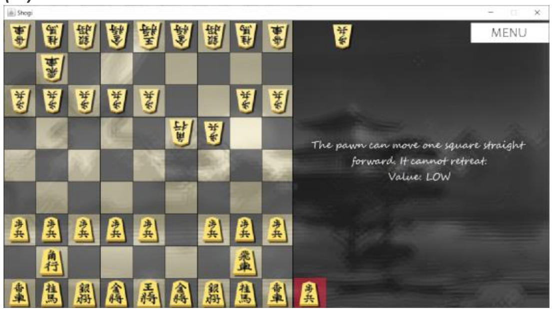
(2)



CHECK MATE
Player Black is the winner...

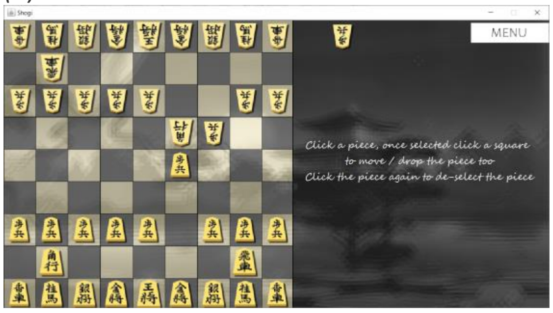
The screen shot below shows a drop – black pawn is dropped:

(1)



The pawn can move one square straight forward. It cannot retreat.
Value: LOW

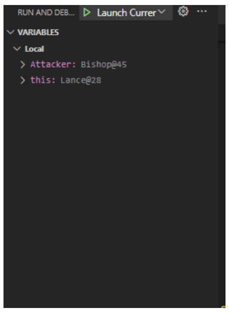
(2)



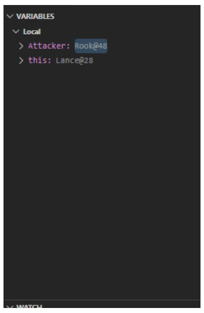
Click a piece, once selected click a square to move / drop the piece too
Click the piece again to de-select the piece

5.2 Board Status

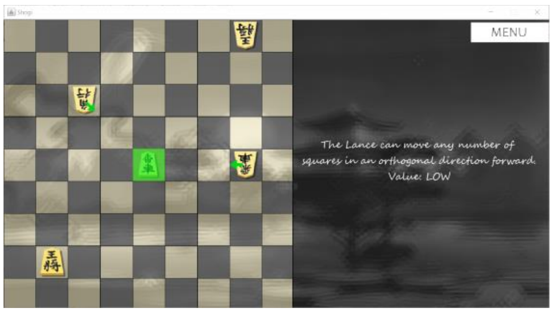
The screen shots below show a brief example of the board status functionally; two attackers are added to a piece's attackers array list – the white lance has two attackers: black bishop and black rook add to its attacker's array list:



```
RUN AND DEB...  ▷ Launch Currer
VARIABLES
  Local
    > Attacker: Bishop@45
    > this: Lance@28
```



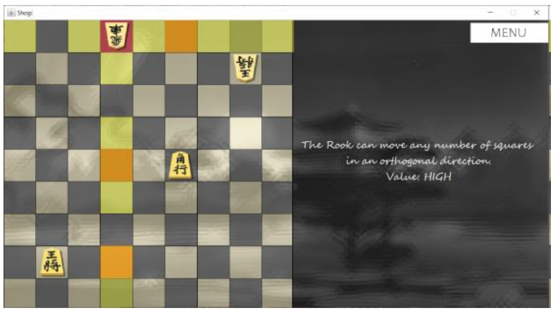
```
VARIABLES
  Local
    > Attacker: Rook@48
    > this: Lance@28
```



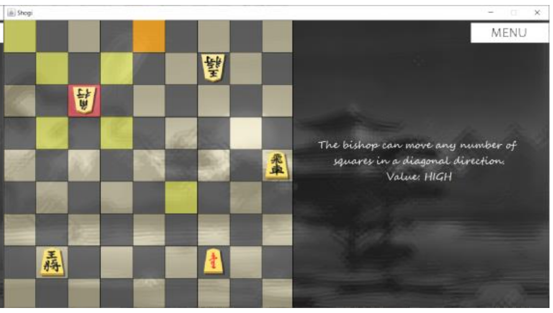
The Lance can move any number of squares in an orthogonal direction forward.
Value: LOW

5.3 Assisted Features

The screenshots below examples of good and great move hints for a white rook and white bishop – good moves that are safe (yellow squares) and great moves: moves that threaten pieces (orange squares).



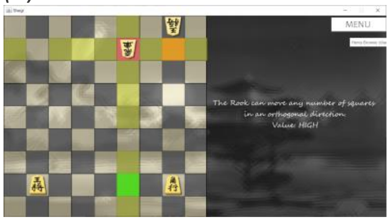
The Rook can move any number of squares in an orthogonal direction.
Value: HIGH



The bishop can move any number of squares in a diagonal direction.
Value: HIGH

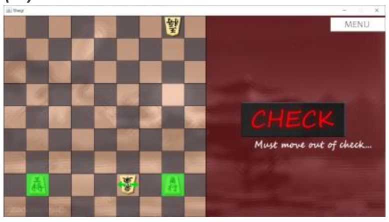
The screenshots below show an example of an amazing move hint (represented by a green square)- the move below is an amazing move because it results in a capture: this is because the result of the move threatens the king and the bishop at once; meaning the opposing player must move the king resulting the bishop capture as shown below:

(1)



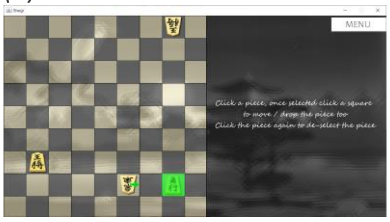
The Rook can move any number of squares in an orthogonal direction.
Value: HIGH

(2)



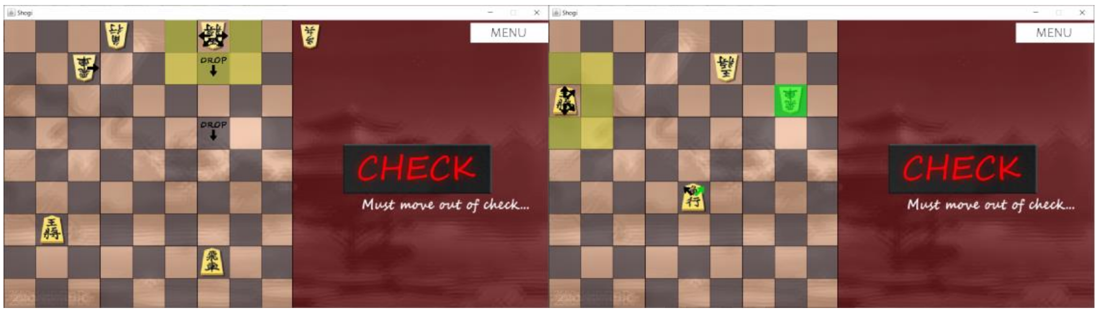
CHECK
Must move out of check...

(3)



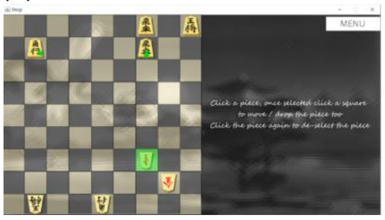
Click a piece, once selected click a square to move / drop the piece too
Click the piece again to de-select the piece

The screenshots below examples of moves and drops out of check hints for a white and a black player (the hint below show moves and drops that are safe but result in the player no longer being in check):

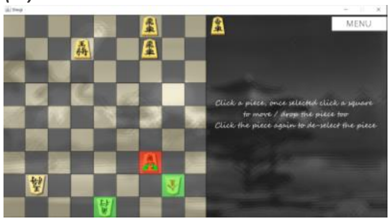


The screen shots below show a piece (white promoted pawn) as capturable because if the resultant possible exchange of pieces that follows -shown in the screen shots- results in the player's advantage: the player captures a promoted bishop (value 7), a promoted silver general (value 6), and a bishop (value 8) and loses a rook (value 10) and a bishop (value 8); meaning the player gets the higher value pieces from the exchange -exchange value= +3.

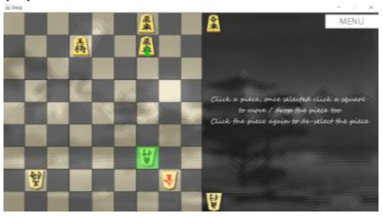
(1)



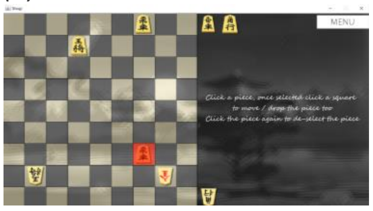
(2)



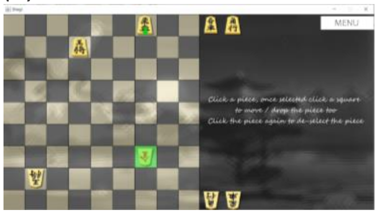
(3)



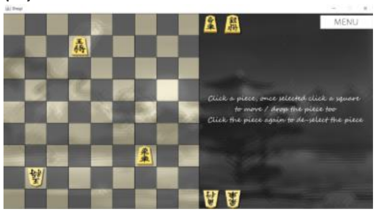
(4)



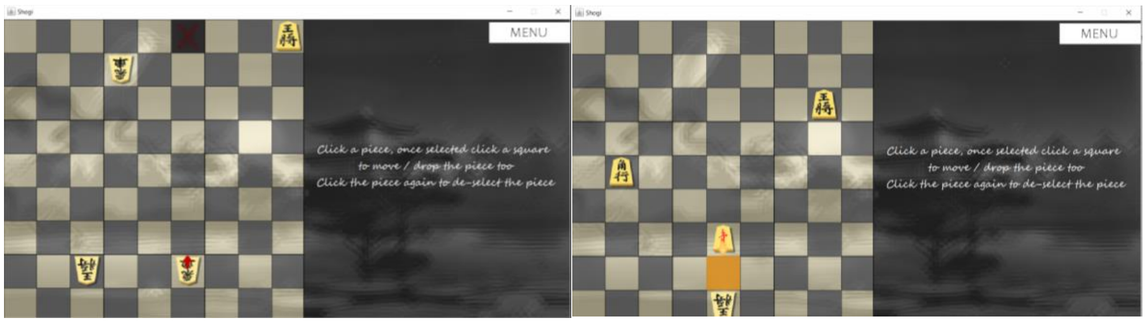
(5)



(6)



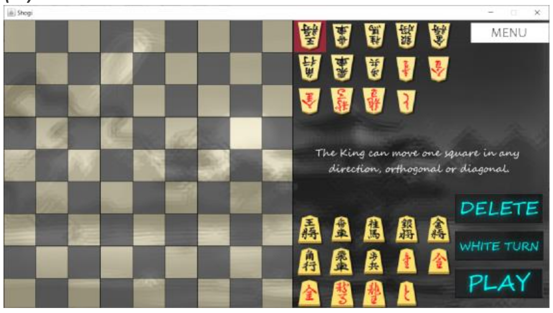
The screenshots below show examples of possible checkmate and opposing checkmate hints -for the opposing checkmate hint the white rook can move to checkmate the black king (player is white), for the possible checkmate hint the black promoted pawn can move to checkmate the white king - protected by the black bishop (player is black):


MENU
Click a piece, once selected click a square
to move / drop the piece too
Click the piece again to de-select the piece
MENU
Click a piece, once selected click a square
to move / drop the piece too
Click the piece again to de-select the piece

5.4 Custom Board

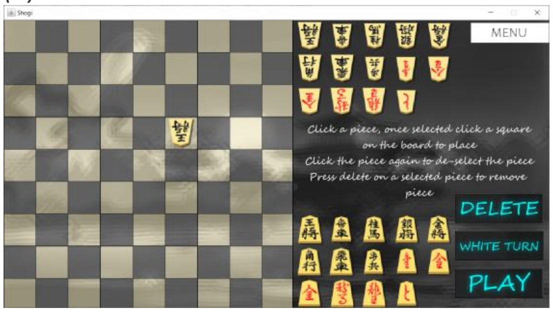
The screenshots below show an example of the custom board functionalities: the user selects and places a white king; the user then selects the same king and clicks the delete button to remove the piece.

(1)



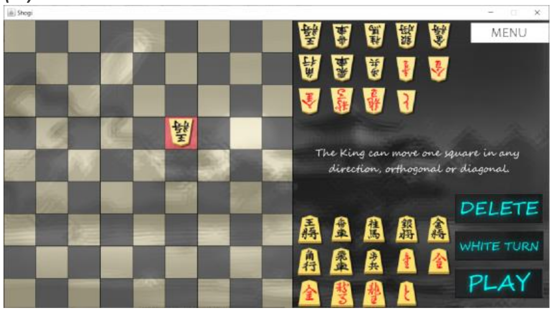
The King can move one square in any direction, orthogonal or diagonal.

(2)



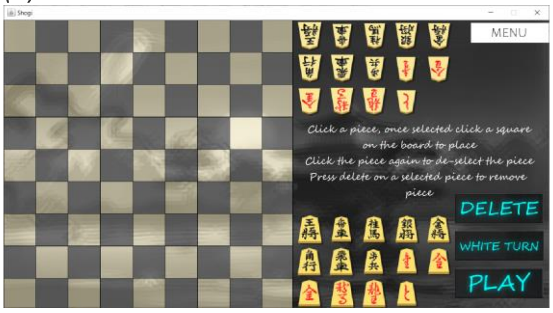
Click a piece, once selected click a square on the board to place
Click the piece again to de-select the piece
Press delete on a selected piece to remove piece

(3)



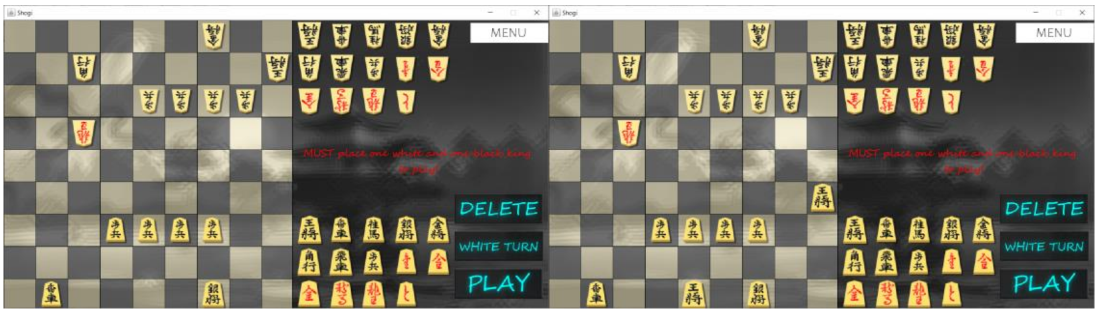
The King can move one square in any direction, orthogonal or diagonal.

(4)



Click a piece, once selected click a square on the board to place
Click the piece again to de-select the piece
Press delete on a selected piece to remove piece

The screen shot below shows a user trying to play the custom board with not the right number of kings placed:

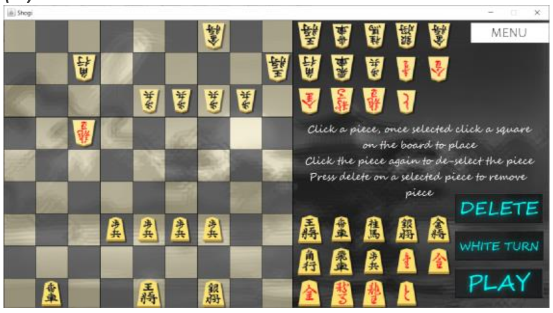


MUST place one white and one black king to play!

MUST place one white and one black king to play!

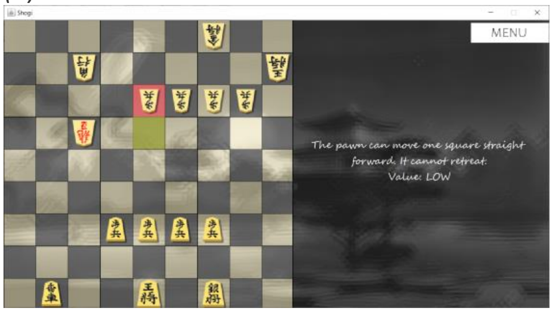
The screenshots below show the user having created a custom board (with two kings), clicking the play button to play the scenario, and selecting a piece to move (white pawn).

(1)



Click a piece, once selected click a square on the board to place
Click the piece again to de-select the piece
Press delete on a selected piece to remove piece

(2)

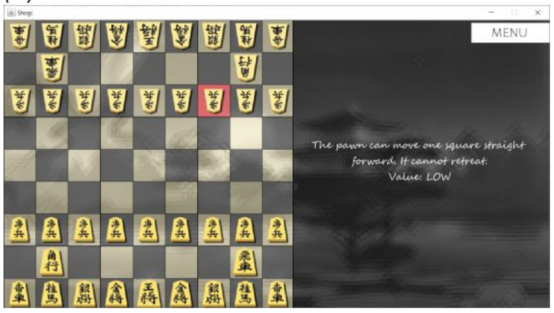


The pawn can move one square straight forward. It cannot retreat.
Value: LOW

5.5 Revert Last Move / Board States

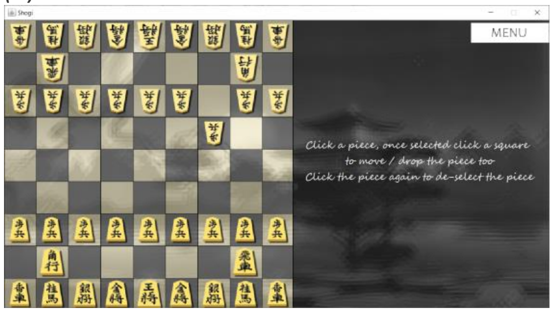
The screenshot below shows an example of a move being reverted – a white pawn is moved, the menu button is then clicked, and the in-game menu is opened, the revert last move button is pressed, and the move is reverted:

(1)



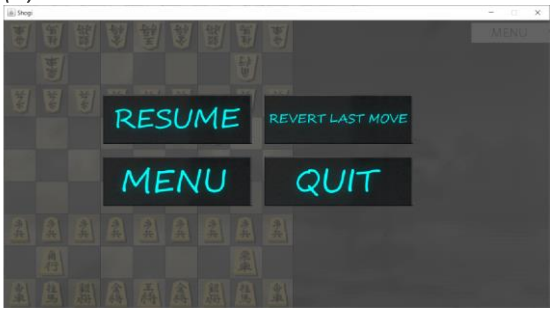
The pawn can move one square straight forward. It cannot retreat. Value: LOW

(2)

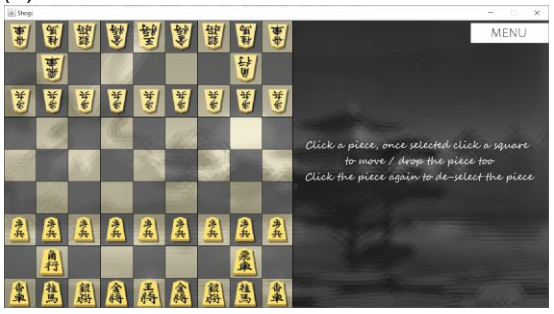


Click a piece, once selected click a square to move / drop the piece too Click the piece again to de-select the piece

(3)



(4)

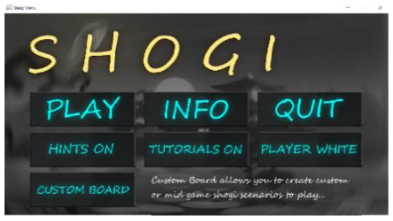


Click a piece, once selected click a square to move / drop the piece too Click the piece again to de-select the piece

5.6 Menu

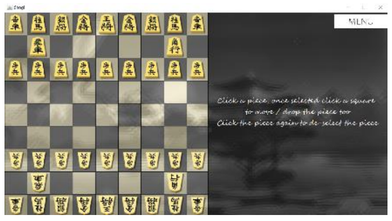
The screen shots bellow shows examples of the menu functions: (1) the play button is clicked, (2) the hints button is clicked, (3) the hints button is pressed again (toggling hints on again) and tutorials button is clicked, (4) the tutorial button is clicked (toggling tutorials on again) and the player colour button is clicked, (5) the custom board button is clicked.

Menu:



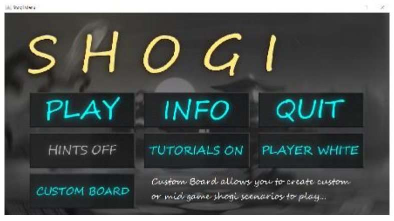
Custom Board allows you to create custom or mid game shogi scenarios to play...

(1)



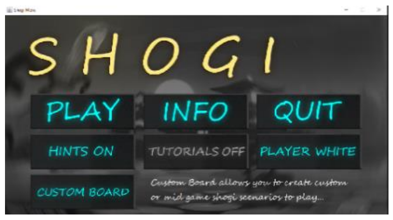
Click a piece, once selected click a square to move / drop the piece too Click the piece again to de-select the piece

(2)



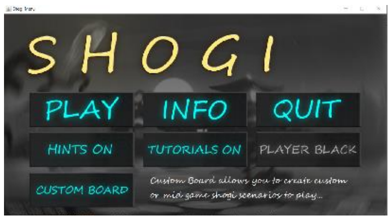
Custom Board allows you to create custom or mid game shogi scenarios to play...

(3)



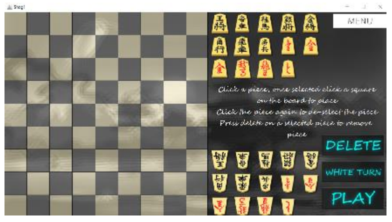
Custom Board allows you to create custom or mid game shogi scenarios to play...

(4)



Custom Board allows you to create custom or mid game shogi scenarios to play...

(5)

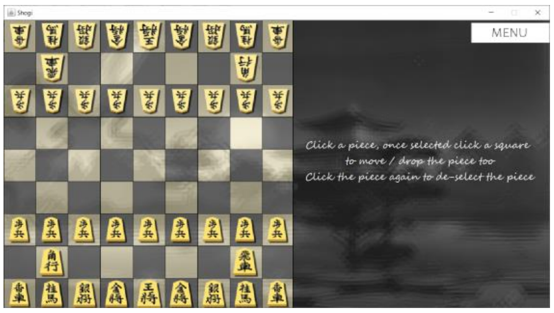


Click a piece, once selected click a square on the board to place Click the piece again to de-select the piece Press delete on a selected piece to remove piece

5.7 In-Game Menu

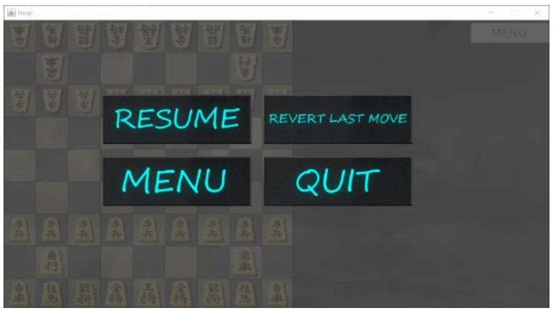
The screenshot below shows the menu button being clicked within the game window resulting in the in-game window to show:

1.



Click a piece, once selected click a square to move / drop the piece too
Click the piece again to de-select the piece

2.



6.0 Testing and Evaluation

6.1 Introduction

Before conducting user testing, I followed all the procedures and generated the appropriate ethics forms for users. To conduct user testing I first created an anonymous survey using Lancaster Qualtrics. Next, I gathered beginner shogi players as participants that (1) were willing to test the program and (2) had basic knowledge of the game and some experience prior. Having arranged times to meet in the Lancaster library, I got participants to use a shogi application using my device to play against online users of the same level (keeping the participant anonymous) whilst using the shogi application I have created to see how helpful it was for the beginner shogi player participants.

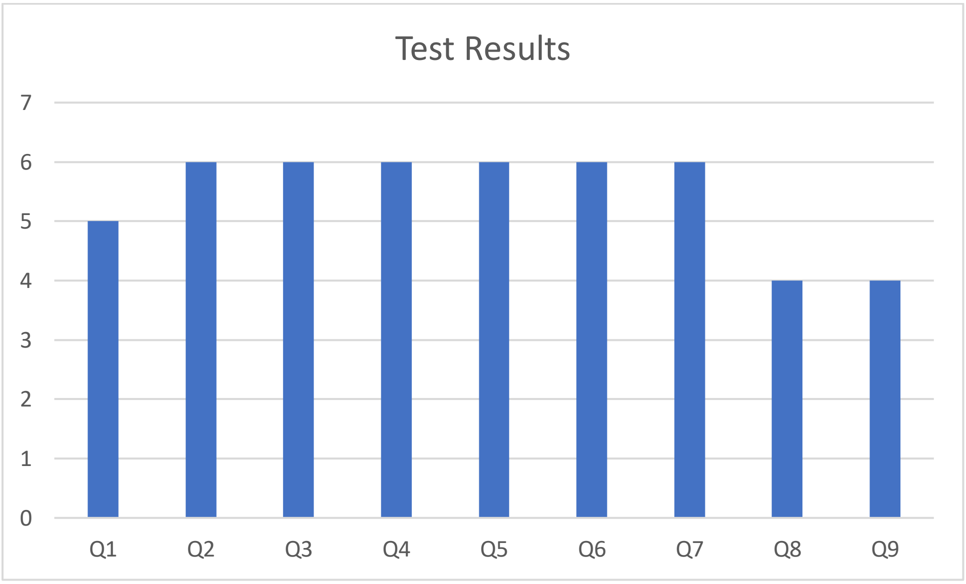
I used this methodology to conduct the testing because by keeping the participants anonymous the results are potentially more honest it also tackles many other issues that may occur because of the experiment. Furthermore, I got the participants to test the program again an application of my choosing to keep the participants anonymous while playing online, it also allowed me to make sure participants were playing against beginners also. Finally, I used a survey because it allowed me collect lots of data in short period; using Qualtrics for the survey it made it easy to analyse results.

6.2 Testing Phase

For the experiment I managed to gather 6 participants. These participants were university students and had the background in shogi I was looking for -as mentioned. Having arranged times to meet in the Lancaster library the experiment was conducted with each participant separately using the convention mentioned in the introduction.

Because participants had to meet the criteria: beginner shogi players with some experience, it was hard to find participants therefore I could only find 6 participants. However, I felt this was sufficient to gather the data needed, because the range for anomalous results in this experiment was not very large because the experiment was conducted in a very controlled environment – mentioned in the previous section.

6.3 Test Results



Q1: Did the program help you beat your opponent?

Q2: Were the move hints helpful?

Q3: Were the tutorials helpful?

Q4: Were the hints and tutorials easy to understand?

Q5: Was the custom board helpful and easy to understand?

Q6: Did the program help you understand the game better?

Q7: Was the graphical user interface easy to use and understand?

Q8: Were there any issues with the program?

Q9: Would you use the program again?

In summary the user experience feedback and examining the results attained from this experiment the consensus was mostly positive in that the program did help most users beat their opponents with good feedback for the hints option with all participants considering this tool as useful and easy to use. Similarly, most participants found the tutorials to be very useful and easy to understand. Furthermore, the response to the application was very positive in that ≈80% of participants found the application did help them beat their opponent and ≈70% of participants would use the application again. However, a frequent issue from the program that has arisen from conducting this experiment that is the game needs slightly more optimization because response time was slow for some users also the custom board needs a feature to allow users to add captured pieces to the custom scenario.

6.4 Evaluation

In summary I am happy with the results attained from the experiment; I feel the program did what it was supposed to with a user-friendly interface. Furthermore, the aims of the project (*Introduction 1.1*) tested here have mostly been accomplished. The main take away from this experiment however is that the program needs a lot more optimization and a slight alteration need to be made to the custom board feature, although almost every other aspect of the program was a success: meeting the project aims.

7.0 Conclusion

7.1 Review of Project Aims

7.1.1 User Requirements

The project user requirements (*Design 3.1*) were outlined within the project proposal; the program was designed using these requirements in mind to allow the program to meet the project aims.

The project's user requirements have been met with slight alterations from the original project plan, allowing to better meet the aims of the project. Furthermore, the project differs slightly from mostly non-functional requirements -as mentioned in *4.0 Implementation* - this was done because when developing the program, I found the aim for a user-friendly interface could have been met more sufficiently by implementing some non-functional requirements slightly differently: for example, the main difference being rather than textual displays using visual representations (for move hints for example) I found to be more user-friendly.

Furthermore, some features of the program I have implemented were not originally in the user-requirements such as the menu (*Implementation 4.6*), and (*Implementation 4.7*) in-game menu however during implementation I found these features were fundamental for meeting the user-friendly design and easy-to-use graphical users' interface -as mentioned in *Design 3.9*. More features implemented that were not originally intended include the custom board (*Implementation 4.4*) and tutorial feature within the assisted features (*Implementation 4.3*). Although these features were not in the requirements, they really added to meeting the aims for example: helping beginner users understand and get better at the game.

7.1.2 Project Aims

Having met the majority of the user-requirements with slight alterations -as mentioned- to allow for the aims to be met more sufficiently. Because I have met the requirements with additions of extra features to aid in meeting the aims. I feel the program meets the aims of the project.

> (1) *Create a user-friendly and easy-to-use shogi interface that allows the user to play against a user on the same machine allowing them to play the shogi board game with all the rules and fundamentals of the game.*

The graphical user interface was designed attentively with key design inspirations being considered (see *3.9 Design **for examples***), with the addition of the testing feedback I can conclude that the aim for a user-friendly and easy-to-use GUI was sufficiently met not only for the shogi game interface but for the whole program.

Furthermore, I feel I have created a user-friendly and easy-to-use shogi interface that allows the user to play against another user on the same machine allowing them to play the shogi board game with all the rules and fundamentals of the game: see *Design 3.1.1.*

> (2) *The program aims to implement assisted features to the base game that also assists a user on potential good moves and advising them about potential bad moves. Users should be able to play against opponents -in an online game for example- while copying the moves of the game into the application, using the application assisted features to beat users of the same level: beginner users.*

I feel the program has met all the aims regarding the assisted features and more: as mentioned in *Design 5.3*, the assisted features provide even more features than originally set out to create -see *Implementation 4.3* For example, not only are good move hints displayed for users (when the hints are turned on) but a variation of good moves are hints can be displayed (good, great, amazing) as well as checkmate hints move hints. If the user does not use the move hints to make good moves as mentioned and a bad move this move is shown as a bad move and the user can revert the move.

Furthermore, from testing the program myself (as a beginner shogi player) and from test results (*6.0 Testing and evaluation*) gathered these move hints I have found to be sufficient in helping beginner users beat beginner opponents.

(3) The programs design aims to be for beginner users to help them get better and understand the game more.

Finally, from my own testing (as a beginner shogi player) I can conclude that the program does help beginner users get better and understand the game more. Having used the program, myself I have found that because of the assisted features: move hints and tutorials, it really helped to understand the game more allowing me to be a better player and the test results also suggest this (*6.0 Testing and evaluation*).

7.2 Personal Reflections

The general approach to the project I took, and the plan set in place within the project proposal for creating this project I found to work well for the overall development. Generally, I stuck to the timeline that was created during planning the project: the Gantt chart was useful in setting targets for sections of the project to complete within certain timeframes. With the setbacks encountered as mentioned in *4.0 Implementation* generally being accounted for within the Gantt chart because I gave myself plenty of time for each section. With efficient use of my time allowing me to mostly stay on schedule. The main issue I found with the timeline created in the Gantt chart was that most the time was set aside for the creation of the program, leaving less time for the testing and report, although because I remained mostly on schedule this was not too much of an issue.

The knowledge in programming attained from modules within my degree thus far have provided a strong basis for the work undertaken with lots of programming knowledge more specifically in java being extremely useful for this project. Compared to the commencement of the project, I now feel a lot more confident in programming and look forward to applying the lessons taught to future projects.

As a whole I am happy with how the project turned out: with the addition of more features than intended and outlined within the project proposal I feel I have sufficiently met the aims for the project. I am happy with the base game I have created within the program and feel it accurately represents the board game shogi and all its aspects of the game. Moreover, I feel with inspiration from other assisted chess programs I feel the level of assisted features I have created will provide help to many beginner users that use the program.

However, if I had more time for the project, I would make alterations to the program to fix issues that arose during testing (*Testing and Evaluation 6.3**)*** such as adding slight optimizations, although as mentioned because of the size of the project created I could not leave myself must time for the testing section of the project. If I were to do the project again, I would potentially use bit boards (*Background 2.4*) rather than array lists for the program because this would improve the program optimization, although this would make the program more complex to understand I feel I have learned enough from this project to make it work.

7.3 Future Work

Working on this project I feel has improved my programming skills a fair amount therefore with less of a time limitation I feel I could develop the program even further. For example, I could implement a feature that allows the user to select more advanced assisted features and move hints, allowing more advanced players to use this program to beat opponents. As mentioned, (*Implementation 4.3*) I could calculate hints for the player that uses an algorithm to look moves ahead to allow for move hints that would be more sufficient for advanced players; I could use leaf evaluation to do. I could also potentially (*Background 2.4*) combining this concept with the fairly concise algorithm I have designed to detect capturable pieces on each leaf (*Design 3.4.2.3*); by doing this it would allow the program to look moves a head using the leaf evaluation concept and then use the capturable pieces algorithm to determine if the sequence of moves -determined by the leaf evaluation- result in the user being able to capture an opposing piece: if so keep this tree node and generate an appropriate move hint – this hint would be sufficient for more advanced players.

7.4 Closing Remarks

This project was very challenging to complete, with no prior experience of creating a project of this scale on my own. I had to be very efficient with my time however as mentioned am happy with the result and feel I have created unique program that is a useful tool for beginner shogi players. I consider this program unique because as mentioned not many assisted shogi program exist (*Background 2.2*) let alone assisted shogi programs made for beginner players.

References

[1]

https://en.wikipedia.org/wiki/Shogi_strategy#:~:text=YSS%207.0%20%20%20%20Piece%20%20,silver%20in%20hand%20%203%20more%20rows%20

[2]
https://en.wikipedia.org/wiki/Computer_chess#:~:text=%EE%80%80Computer%20chess%EE%80%81%20%EE%80%80programs%EE%80%81%20consider%20%EE%80%80chess%EE%80%81%20moves%20as%20a,final%20%22leaf%22%20position%20has%20been%20reached%20%28e.g.%20checkmate%29.

[3]

https://lichess.org

[4]

https://fritz.chessbase.com/en/Fritz

[5] (Piece images used within the program)

https://github.com/Ka-hu/shogi-pieces

Appendix A. Project Proposal

3rd Year Project: Programming Assisted Shogi

Task

The goal for my project is to create a shogi interface that allows the user to play against a user on the same machine, which also assists the user on potential good moves and advising them about potential bad moves.

Game Context

Shogi is the Japanese version of chess. Like chess if your king can be captured in your opponent's next turn your king is in check and must move into safely in the next turn, however if it cannot move into safety the king is in checkmate and the opponent wins this game (this is the aim of the game). Furthermore, like chess pieces include:

The bishop: this piece can move any number of spaces diagonally only

The rook: this piece can move any number of spaces vertically and horizontally

The knight: this can move 2 spaces vertically then one space horizontally. It can also jump over pieces

The king: this can move only one space in any direction (vertically, horizontally, or vertically)

The pawn: this piece can move one space forward.

Shogi has however the addition of pieces not also used in chess, as shown below:

The lance: The lance can move any number of spaces horizontally forward

The silver general: The silver general can move one space diagonally in any direction or one space forward.

The gold general: The gold general can move one space in any direction except diagonally backwards.

Shogi also has the addition of promoted pieces unlike chess, a piece can be promoted if it makes it to the furthest 1/3 of the board, the player can promote the piece at the end of their turn or choose not to, however if the piece makes it to the end of the opposing side of the board it must be promoted. Each promoted piece moves the same with an addition of addition moves it can now make.

The promoted bishop can now additionally move 1 space in any direction.

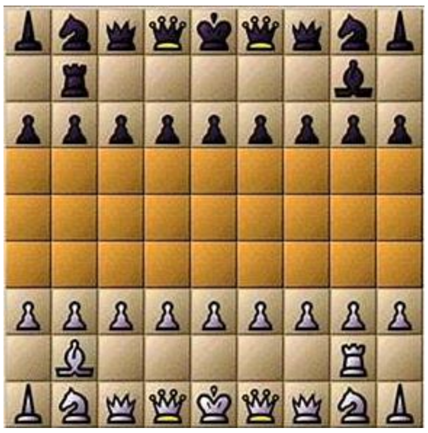
The promoted rook can now additionally move 1 space in a diagonal direction.

All other promoted pieces (silver general, knight, lance, pawn) can now additionally move the same as the gold general (one spaces in any direction except diagonally backwards).

Furthermore, in shogi pieces you have captured can be dropped anywhere on the board instead of moving a piece; the piece will be un-promoted.

However, you cannot drop a pawn in the same row as another one of your un promoted pawns. A pawn cannot be dropped to give immediate checkmate however other pieces can be dropped to achieve this. You cannot drop a piece into a space where it cannot move. If you drop a piece on a promoted zone, you cannot promote the piece until it moves on a future turn.

The shogi board is laid out as shown below:



Related work and motivation

Currently I cannot find any examples of an assisted shogi program that works and is available, therefore giving me motivation to create this program. I would like to create a good working primitive version of this program to be developed even further in the future. Furthermore, assisted programs are available for chess but not shogi: for example, of an assisted chess program I have found is the fritz chess program online. This program gives the user hints of the best next move; this program has given me ideas for my program, as I will implement a similar feature that hints the user if an opponent’s piece is not protected and can be captured.

Task Approach

Functional Requirements

(Rules and fundamental game)

It shall allow the user and opposing player (using the same GUI to play) to make valid moves.

It shall allow the user to drop captured pieces if the selected square doesn't break the rules.

It shall notify the user if they have been checkmated, allowing them to restart the game.

It shall notify the user of good and bad moves/ drops, allowing them to take back bad moves.

(Higher level fundamentals)

It shall change the state of pieces when the user tries to move it to a new square, variables that define the state of the piece include: if the piece is protected or not and if the piece can be promoted or not.

To detect if it is a bad move/ drop or not the program shall save the state of the board and contain an array of all the spaces the opposing players pieces can move to, if the square the user tries to move to is contained in the array and the piece is unprotected it will be detected as a bad move/drop.

If the user is notified of a bad move/ drop and uses the GUI to cancel the move, the program shall return the piece to the original state.

To notify the user of a good move the program shall notify the user if there is an opponent's piece that is unprotected and can be captured; the array of all the spaces the opposing players pieces can move to could be used to detect if the players pieces are un-protected, if the variables that define the state of the piece suggest it is un-protected and it is contained in an array of all the possible squares the user can move to, it will be detected as a capturable piece.

Non-function requirements

The program shall use a similar looking GUI as the one above, promoted pieces will be red to indicated they're promoted.

It shall allow the player/players to click a piece and click the square the wish to move it to.

It shall assist the user by allowing them to make a move and if it is a bad move: the piece is unprotected and can be taken the user will get a prompt telling them it is a bad move and why, it will allow them to continue to make the move if they choose or they can cancel the move.

It shall assist the user by showing textual messages on the right-hand side of the board, telling the user if and what piece is capturable.

It shall show users captured pieces on the left side of the board.

It shall allow the user to click captured pieces and click a valid placement on the board to drop onto the board.

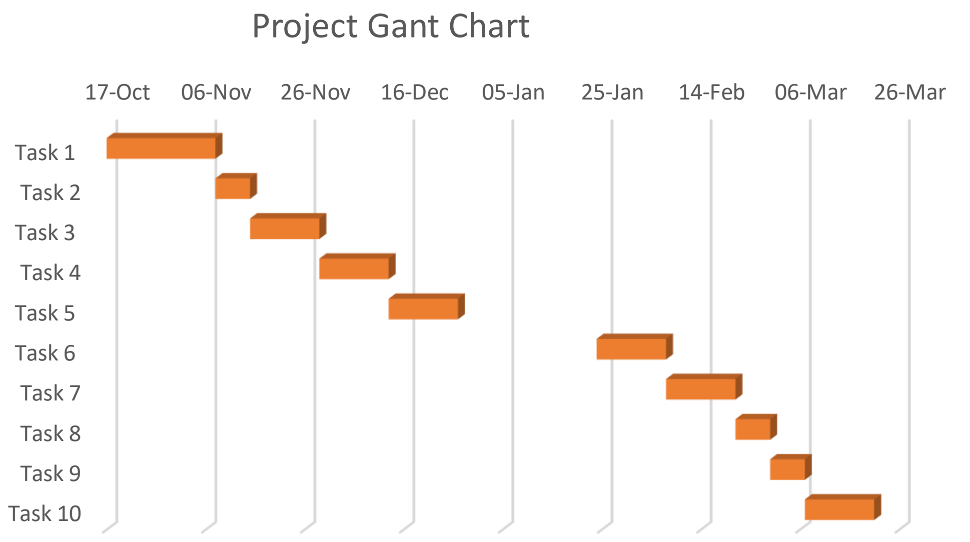
It shall also notify the user if it is a bad place to drop and allow them to cancel the drop, with a pop-up window on the right of the screen.

It shall notify the user if they're in check with a pop-up message, if they're check mated an alternative pop-up message will appear allowing the user to press buttons to restart the game or quit the program.

Task Difficulties

The difficulties of the task include creating an algorithm to compare complex arrays of all the possible spaces the user and opponent can move to and using this data to define the state of each piece on the board (protected/ not protected) and furthermore to determine if a move is a good or bad move and to determine if an opponent's piece is capturable.

Plan



	Task 10	Task 9	Task 8	Task 7	Task 6	Task 5	Task 4	Task 3	Task 2	Task 1
Start date	07-Mar	28-Feb	21-Feb	07-Feb	24-Jan	13-Dec	29-Nov	15-Nov	08-Nov	11-Oct
🟧 Time to complete (days)	14	7	7	14	14	14	14	14	7	28

Task 1: Complete plan
Task 2: Create GUI and classes for program
Task 3: Allow the user and opponent make valid moves
Task 4: Allow the user and opponent to capture pieces and drop pieces and allow the user and opponent to promote pieces.
Task 5: Create array of all the users and opponent's user possible moves and use this to define the state of pieces (protected/ capturable), use the data created previously to notify the user of bad move and allow them to take it back
Task 6: Notify the user of check or check mate and display appropriate message, begin final write up.
Task 7: Create array of all the users and opponent's user possible moves and use this to define the state of pieces (protected/ capturable), use the data created previously to notify the user of bad move and allow them to take it back
Task 8: Use the data to notify the user if they're able to capture an un-protected piece
Task 9: Test, debug and finalize project
Task 10: complete the final write-up

Evaluation

To decide whether I have met the objectives of the game each fundamental and non-fundamental requirement must be met to achieve the working game. Furthermore, once the game requirements have been met the game will be tested thoroughly by myself, I could also get example users to test the game to ensure the fundamental game with some assisted features (as previously listed) work; allowing new shogi players to play the game and with assistance allowing them to improve. The example users will also then comment on how successfully the game and features works and furthermore how well the assisted helped them to understand the game and moves better.

References

https://www.chess.com/forum/view/suggestions/shogi-japanese-chess

https://fritz.chessbase.com/en/Fritz

Appendix B. Testing Documents

LANCASTER UNIVERSITY DEPARTMENT OF COMPUTING

RESEARCH INFORMATION SHEET

TITLE OF RESEARCH:

Assisted Shogi Program Research

PRINCIPAL INVESTIGATOR:

Henry Browne

Address: Computing Department, Infolab21, Lancaster University, Lancaster LA1 4WA

INTRODUCTION

You will be taking part in a research study concerned with determining the usefulness and usability of an assisted shogi program.

We invite you to participate in a study that will involve the collection of questionnaire data. As participants in this research, we invite you to test the program and provide anonymous feedback via questionnaire

It is important that you read and understand several principles that apply to all who take part in our studies:

a) taking part in the study is entirely voluntary

b) personal benefit may not result from taking part in the study, but knowledge may be gained that will benefit others

c) any significant findings will be discussed with you if you desire

d) you may withdraw from the study at any time.

The nature of the study, the risks, inconveniences, discomforts, and other pertinent information about the study are discussed below. You are urged to discuss any questions you have about this study with the investigator before you sign this consent.

In accord with all of our research protocols, privacy will be fully protected and confidentiality maintained at all times.

BACKGROUND & PURPOSE:

This research study is concerned with determining the useability and usefulness of this assisted shogi program, that has the potential to help beginner shogi players learn and understand the game a lot better. Participants will need at least basic knowledge and understanding of the game, they will need a least some previous experience of playing the game before participating.

STUDY PROCEDURE:

You are being asked to participate in a study that will require your cooperation in one or more of the following:

….

This investigation will take participants between 20 – 40 minutes to complete. Participants will need to attend a small one to one meeting on the university campus in the library with myself at a set time specified beforehand having been contacted by myself. In this meeting the participant will be asked to test the program in against an online player of the same ability, using a phone application set up by myself using my device.

When writing the results from our questionnaire into a project report or any other form of documentation, steps are taken to ensure anonymity for all those involved in the study. No personal details will be recorded. Confidentiality will be always maintained. Any recordings that are made are the property of the researcher and will be kept in a secure environment and destroyed at the conclusion of the research.

RISKS OF PARTICIPATION IN THE STUDY:

The risks of participating in this study are minimal.

It is the investigators' intention that your identity in these studies will remain confidential.

Your particular contribution to the study – what you disclose during … – will be anonymized.

BENEFITS:

There may be no personal benefit to you from participating in this project. However, some personal benefits of this research may include learning more about shogi.

We believe this work can make an important contribution to potentially helping people discover and learn / discover more about the shogi game.

COSTS AND COMPENSATION:

You will not be paid for participating in this study.

There is unlikely to be any cost - financial or other - to you for participation in the study.

CONFIDENTIALITY:

All information collected in this study belongs to the fieldworker and will be maintained in a confidential manner at Lancaster University. Nobody, other than the fieldwork researcher, will have access to the data. Any identifiable data (including recordings of participants' voices) on portable devices (eg audio recorders, etc) will be erased from it as quickly as possible and in the meantime the device will be stored securelyAny recordings will be destroyed at the end of the project. Although rare, it is possible that disclosure may be required by law. Otherwise, the information will not be disclosed to third parties without your permission. If the study is published, your name will be kept confidential.

Furthermore, the participant will be playing against an online player within a mobile application, the online application will have anonymous name, so the participant is anonymous is kept as confidential as possible.

PEOPLE TO CONTACT:

If you have further questions related to this research study, you may email the Supervisor Paul Dempster, Dept of Computing and Communications, Lancaster University

Email: p.dempster@lancaster.ac.uk
Address: Computing Department, Infolab21, Lancaster University, Lancaster LA1 4WA

You may also if you wish contact an independent person about this research – specifically, Nigel Davies, Head of School.

School of Computing and Communications

(http://www.scc.lancs.ac.uk/)

InfoLab 21 Building, South Drive,

Lancaster University, Lancaster LA1 4WA, England

email: n.a.davies@lancaster.ac.uk

SCC Undergraduate Ethics Form

1. Basic information

Name of Student: Henry Browne **Student ID:** 33733273

Course: Computer Science

Name of Supervisor: Paul Dempster

Project Title: Assisted Shogi

Aim(s) of the research project. (3-4 sentences)

The aim of the research project is to determine how useful the program is for beginner shogi players; does it give the player an advantage against other shogi players of the same level, does it help them understand the how to play the game more. Determine the usability of the program.

In line with GDPR the lawful purpose of this research is a for scientific research in accordance with safeguards. As stated: https://www.lancaster.ac.uk/research/participate-in-research/data-protection-for-research-participants/

2. Proposed research methods and analysis

Provide details about:

The research method I will see is a Questionnaire; The participants will be asked to play a shogi game on a device of their choosing, they will use the program in addition to playing the game; the move they make in the game, they will also make within the program. The program will generate assisted hints and tutorials for each move, the participant can then use these hints and tutorials while playing against the opponent. Once the game is complete, they will fill in the questionnaire to see how useful the program was when playing against their opponent.

3. Information about Human Participants

If applicable, provide details about:

What type of participants will be used in the study?

Beginner shogi players (participants must have basic knowledge and understanding of the game with some experience)

What age range is to be used?

Age 18 and above

What characteristics (if any) are to be used in selecting participants?

The player must have very basic knowledge and understanding of how to play shogi with some experience

How many participants will be involved?

6

How will participants be recruited?

I will recruit participants from within my social network

Does the research involve deception, trickery or other procedures that may contravene participants' informed consent, without timely and appropriate debriefing, or activities that cause stress, anxiety or involve physical contact?

NO

Access to records of personal or other confidential information, including genetic or other biological information, concerning identifiable individuals, without their knowledge or consent?

NO

Does the research project & associated experiments potentially risk the physical safety of yourself or the participants?

NO

Does the research involve travel to areas where you might be at risk?

NO

4. Information about non-human participants such as animals

If applicable, provide details about:

Does the research involve animals?

NO

5. Data handling

Provide details about:

What type of data will be collected?

Questionnaire data from user feedback

How will this be stored?

Data will be stored as a Qualtrics questionnaire using: https://lancasteruni.eu.qualtrics.com/jfe/form/SV_9zy3hHWgWkkMlcG

What steps will be taken to ensure the anonymity of the data collected?

The questionnaire will be carefully created / worded to allow for the anonymity of each participant. Furthermore, the questionnaire used will not ask for any personal information (name, email, ext..). Responses within the questionnaire will also be regulated by myself to not allow participants to reveal any details that will reveal their identity.

What steps will be taken to ensure the confidentiality of the data collected? State how individual identifying information will be removed, where the data will be stored and who will have access to the data.

The data collected will be stored in securely as mentioned; only myself, my supervisor and the project module markers will have access to this data.

6. Please complete all sections by ringing the appropriate answer.

1. RISKS

Do any aspects of the study pose a possible risk to participants' physical well-being (e.g. use of substances such as alcohol or extreme situations such as sleep deprivation)?	Y	[✓ N]
Are there any aspects of the study that participants might find embarrassing or be emotionally upsetting?	Y	[✓ N]
Are there likely to be culturally sensitive issues (e.g. age, gender, ethnicity etc)?	Y	[✓ N]
Does the study require access to confidential sources of information (e.g. medical, criminal, educational records etc.)?	Y	[✓ N]
Might conducting the study expose the researcher to any risks (e.g. collecting data in potentially dangerous environments)?	Y	[✓ N]
Does the intended research involve vulnerable groups (e.g. prisoners, children, older or disabled people, victims of crime etc.)	Y	[✓ N]

2. DISCLOSURE

Does the study involve covert methods?	Y	[✓ N]
Does the study involve the use of deception, either in the form of withholding essential information about the study or intentionally misinforming participants about aspects of the study.	Y	[✓ N]

3. DEBRIEFING

Do the planned procedures include an opportunity for participants to ask questions and/or obtain general feedback about the study after they have concluded their part in it?	NA	[✓ Y]	N

4. INFORMED PARTICIPATION/CONSENT

Will participants in the study be given accessible information outlining: a) the general purpose of the study, b) what participants will be expected to do c) individuals' right to refuse or withdraw at any time?		[✓ Y]	N
Will participants have an opportunity to ask questions prior to agreeing to participate?		[✓ Y]	N
Have appropriate authorities given their permission for participants to be recruited from or data collected on their premises (e.g. shop managers, head teachers, classroom lecturers)?		[✓ Y]	N

5. ANONYMITY AND CONFIDENTIALITY

Is participation in the study anonymous?		[✓ Y]	N
If anonymity has been promised, do the general procedures ensure that individuals cannot be identified indirectly (e.g. via other information that is taken)?		[✓ Y]	N
Have participants been promised confidentiality?		[✓ Y]	N
If confidentiality has been promised, do the procedures ensure that the information collected is truly confidential (e.g. that it will not be quoted verbatim)?		[✓ Y]	N
Will data be stored in a secure place which is inaccessible to people other than the researcher?		[✓ Y]	N
If participants' identities are being recorded, will the data be coded (to disguise identity) before computer data entry?	N/A	Y	N

7. SUMMARY OF ETHICAL CONCERNS

If any of the boxes below require ticks, more detail may be required to get ethical approval. If none of the boxes require ticks, then it is reasonable to expect approval.

If you have answered 'YES' to any of the questions in Section 1 (risks), please tick the box	
If you have answered 'YES' to any of the questions in Section 2 (Disclosure/covert methods), please tick the box	
If you have answered 'NO' to any of the questions in Section 3 (debriefing), please tick the box	
If you have answered 'NO' to any of the questions in Section 4 (consent), please tick the box	
If you have answered 'NO' to any of the questions in Section 5 (confidentiality), please tick the box	

8. Declaration

I confirm that this is an accurate record of the project to be undertaken.

Student signature Date

Henry Browne 06 / 03 / 2022

________________________________ __________________

I confirm that I have read this proposal and agree that it is a clear and accurate assessment of the project to be undertaken. I have emailed a copy of this ethics form to the teaching office.

Project supervisor Date

___Paul Dempster________________ _____2022/03/18________

Research Participant Consent Form

PROJECT TITLES: Assisted Shogi

INVESTIGATORS: Henry Browne

PARTICIPANT NAME: ________________________________

TITLE: ________________________________

I agree to participate in the project named above, the particulars of which have been explained to me. I have read the Research Project Description a written copy of which has been given to me to keep.

I understand that any information I provide is confidential, and that, subject to the limitations of the law, no information that could lead to the identification of any individual will be directly disclosed in any reports on the project, or to any other party.

I agree to being: (tick as appropriate):

- ☐ observed while using relevant applications
- ☐ used in a questionnaire

I agree to the following data being collected:

- ☐ field notes
- ☐ questionnaire answers

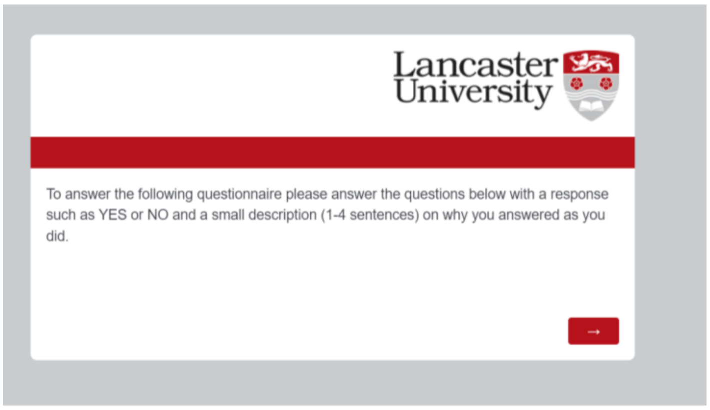
I also agree to the data above being used for later analysis by the researchers above only. To preserve anonymity, I understand that all written work referring to this data will use pseudonyms for me unless written permission is later obtained. I also understand that direct access to the identity of participants is restricted to named researchers above only.

I acknowledge that:

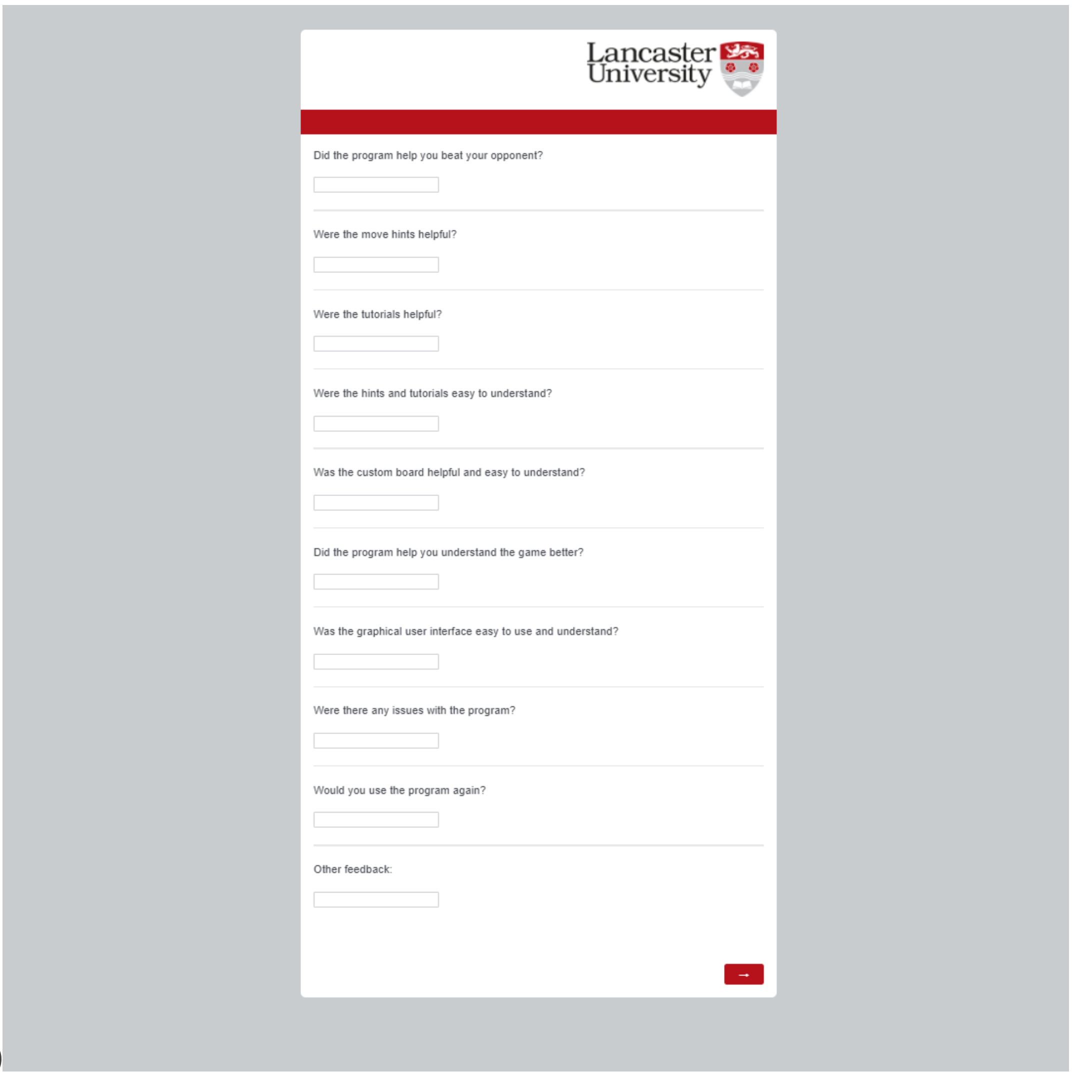
a) I have been informed that I am free to withdraw from the project at any time without explanation or prejudice and to withdraw data previously supplied.
b) Participation in this project is voluntary.

Signature: ____________________________ Date: _______________
(Participant)

Finally, the screenshots below show the survey that was used when conducting user testing followed by a link to the survey.


To answer the following questionnaire please answer the questions below with a response such as YES or NO and a small description (1-4 sentences) on why you answered as you did.

(1)


Did the program help you beat your opponent?

Were the move hints helpful?

Were the tutorials helpful?

Were the hints and tutorials easy to understand?

Was the custom board helpful and easy to understand?

Did the program help you understand the game better?

Was the graphical user interface easy to use and understand?

Were there any issues with the program?

Would you use the program again?

Other feedback:

(2)

https://lancasteruni.eu.qualtrics.com/jfe/form/SV_9zy3hHWgWkkMlcG

Appendix C. GitHub Repository

https://github.com/henryepic121/Shogi.git