MULTIMEDIA UNIVERSITY

CCP6224-OOAD Assignment
Trimester 2430
By Group D

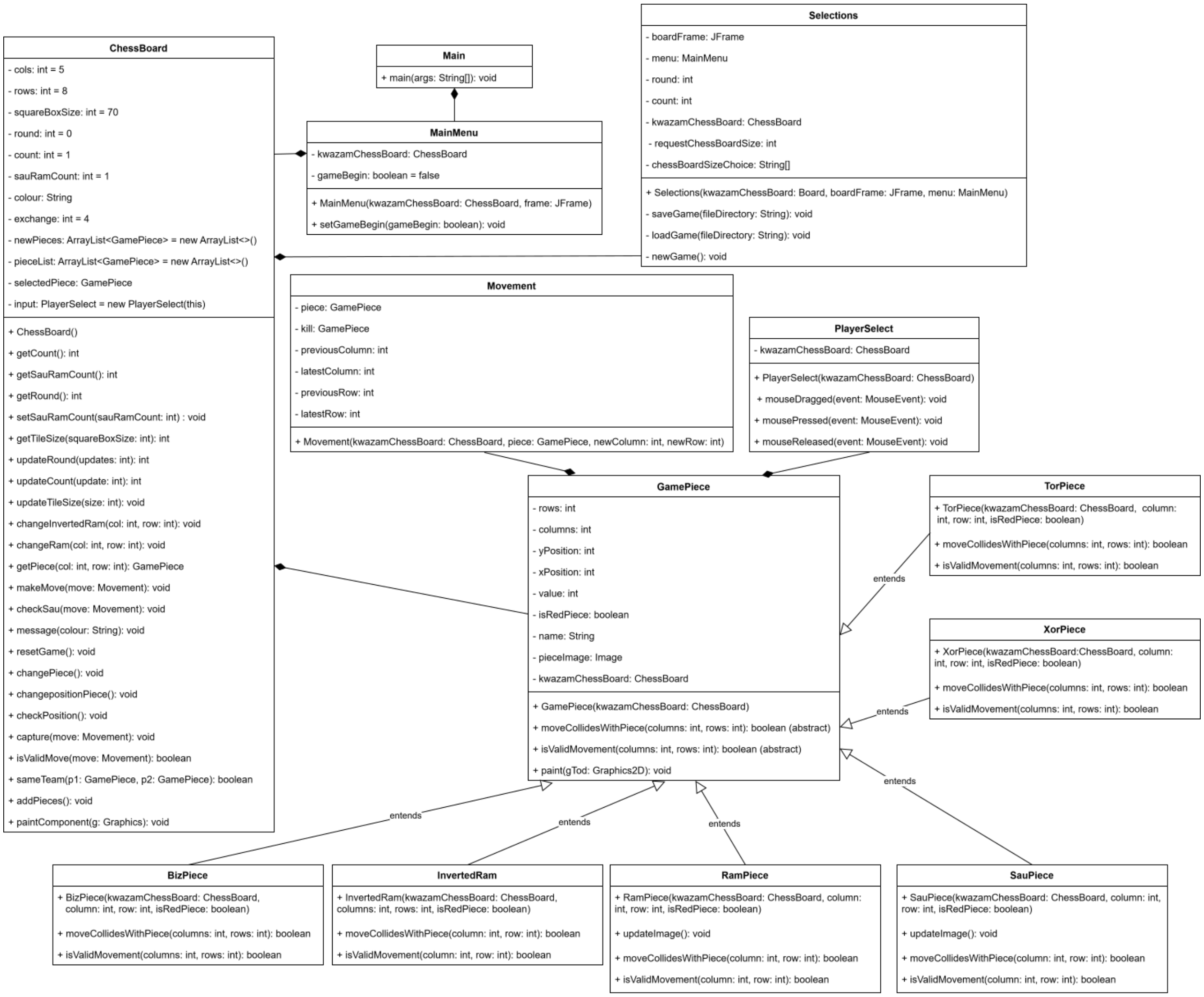
Student Name	Student ID	Email	Phone Number
CHIN ZHEN HO	1221102540	1221102540@student.mmu.edu.my	018-206 7278
ERIC TEOH WEI XIANG	1221102007	1221102007@student.mmu.edu.my	017-406 3708
BERNARD RYAN SIM KANG XUAN	1221101777	1221101777@student.mmu.edu.my	016-716 8548
GAN SHAO YANG	1221103201	1221103201@student.mmu.edu.my	01154218068

CONTENTS

Table of contents:

1.0 UML Class Diagram

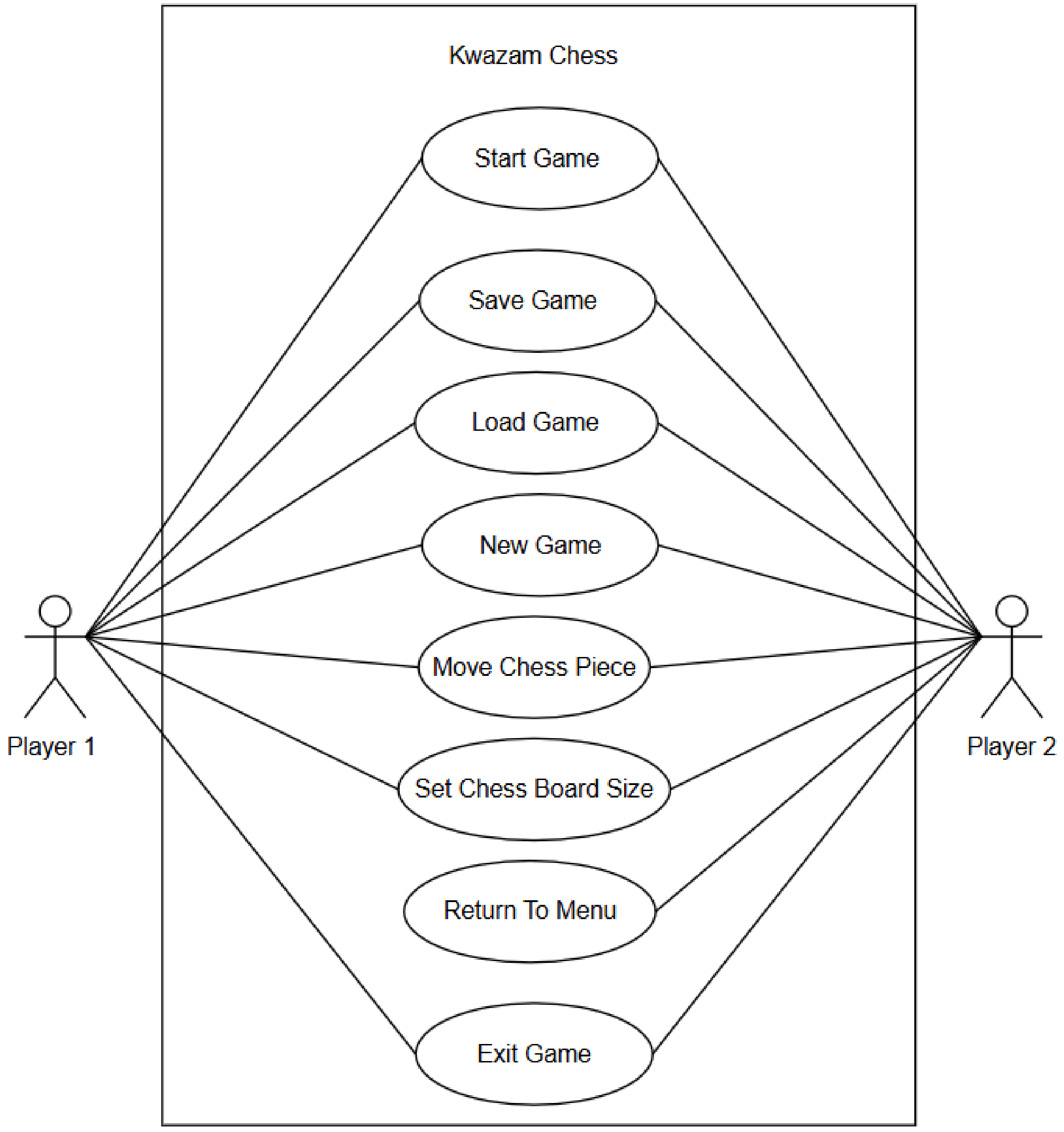
1.0 Uml Class Diagram based on our code:


ChessBoard
- cols: int = 5
- rows: int = 8
- squareBoxSize: int = 70
- round: int = 0
- count: int = 1
- sauRamCount: int = 1
- colour: String
- exchange: int = 4
- newPieces: ArrayList<GamePiece> = new ArrayList<>()
- pieceList: ArrayList<GamePiece> = new ArrayList<>()
- selectedPiece: GamePiece
- input: PlayerSelect = new PlayerSelect(this)
+ ChessBoard()
+ getCount(): int
+ getSauRamCount(): int
+ getRound(): int
+ setSauRamCount(sauRamCount: int) : void
+ getTileSize(squareBoxSize: int): int
+ updateRound(updates: int): int
+ updateCount(update: int): int
+ updateTileSize(size: int): void
+ changeInvertedRam(col: int, row: int): void
+ changeRam(col: int, row: int): void
+ getPiece(col: int, row: int): GamePiece
+ makeMove(move: Movement): void
+ checkSau(move: Movement): void
+ message(colour: String): void
+ resetGame(): void
+ changePiece(): void
+ changepositionPiece(): void
+ checkPosition(): void
+ capture(move: Movement): void
+ isValidMove(move: Movement): boolean
+ sameTeam(p1: GamePiece, p2: GamePiece): boolean
+ addPieces(): void
+ paintComponent(g: Graphics): void
Main
+ main(args: String[]): void
MainMenu
- kwazamChessBoard: ChessBoard
- gameBegin: boolean = false
+ MainMenu(kwazamChessBoard: ChessBoard, frame: JFrame)
+ setGameBegin(gameBegin: boolean): void
Selections
- boardFrame: JFrame
- menu: MainMenu
- round: int
- count: int
- kwazamChessBoard: ChessBoard
- requestChessBoardSize: int
- chessBoardSizeChoice: String[]
+ Selections(kwazamChessBoard: Board, boardFrame: JFrame, menu: MainMenu)
- saveGame(fileDirectory: String): void
- loadGame(fileDirectory: String): void
- newGame(): void
Movement
- piece: GamePiece
- kill: GamePiece
- previousColumn: int
- latestColumn: int
- previousRow: int
- latestRow: int
+ Movement(kwazamChessBoard: ChessBoard, piece: GamePiece, newColumn: int, newRow: int)
PlayerSelect
- kwazamChessBoard: ChessBoard
+ PlayerSelect(kwazamChessBoard: ChessBoard)
+ mouseDragged(event: MouseEvent): void
+ mousePressed(event: MouseEvent): void
+ mouseReleased(event: MouseEvent): void
GamePiece
- rows: int
- columns: int
- yPosition: int
- xPosition: int
- value: int
- isRedPiece: boolean
- name: String
- pieceImage: Image
- kwazamChessBoard: ChessBoard
+ GamePiece(kwazamChessBoard: ChessBoard)
+ moveCollidesWithPiece(columns: int, rows: int): boolean (abstract)
+ isValidMovement(columns: int, rows: int): boolean (abstract)
+ paint(gTod: Graphics2D): void
TorPiece
+ TorPiece(kwazamChessBoard: ChessBoard, column: int, row: int, isRedPiece: boolean)
+ moveCollidesWithPiece(columns: int, rows: int): boolean
+ isValidMovement(columns: int, rows: int): boolean
entends
XorPiece
+ XorPiece(kwazamChessBoard:ChessBoard, column: int, row: int, isRedPiece: boolean)
+ moveCollidesWithPiece(columns: int, rows: int): boolean
+ isValidMovement(columns: int, rows: int): boolean
entends
entends
entends
entends
entends
entends
BizPiece
+ BizPiece(kwazamChessBoard: ChessBoard, column: int, row: int, isRedPiece: boolean)
+ moveCollidesWithPiece(columns: int, rows: int): boolean
+ isValidMovement(columns: int, rows: int): boolean
InvertedRam
+ InvertedRam(kwazamChessBoard: ChessBoard, columns: int, rows: int, isRedPiece: boolean)
+ moveCollidesWithPiece(column: int, row: int): boolean
+ isValidMovement(column: int, row: int): boolean
RamPiece
+ RamPiece(kwazamChessBoard: ChessBoard, column: int, row: int, isRedPiece: boolean)
+ updateImage(): void
+ moveCollidesWithPiece(column: int, row: int): boolean
+ isValidMovement(column: int, row: int): boolean
SauPiece
+ SauPiece(kwazamChessBoard: ChessBoard, column: int, row: int, isRedPiece: boolean)
+ updateImage(): void
+ moveCollidesWithPiece(column: int, row: int): boolean
+ isValidMovement(column: int, row: int): boolean

2.0 Use Case Diagram

2.0 USE CASE

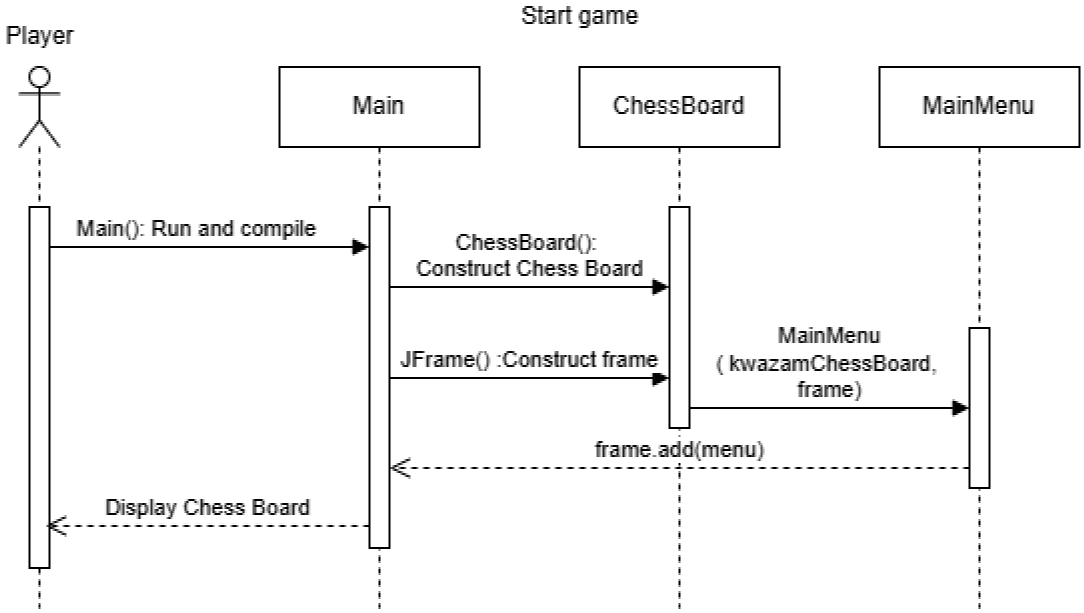
These are the main functions performed by the players



3.0 Sequence Diagram

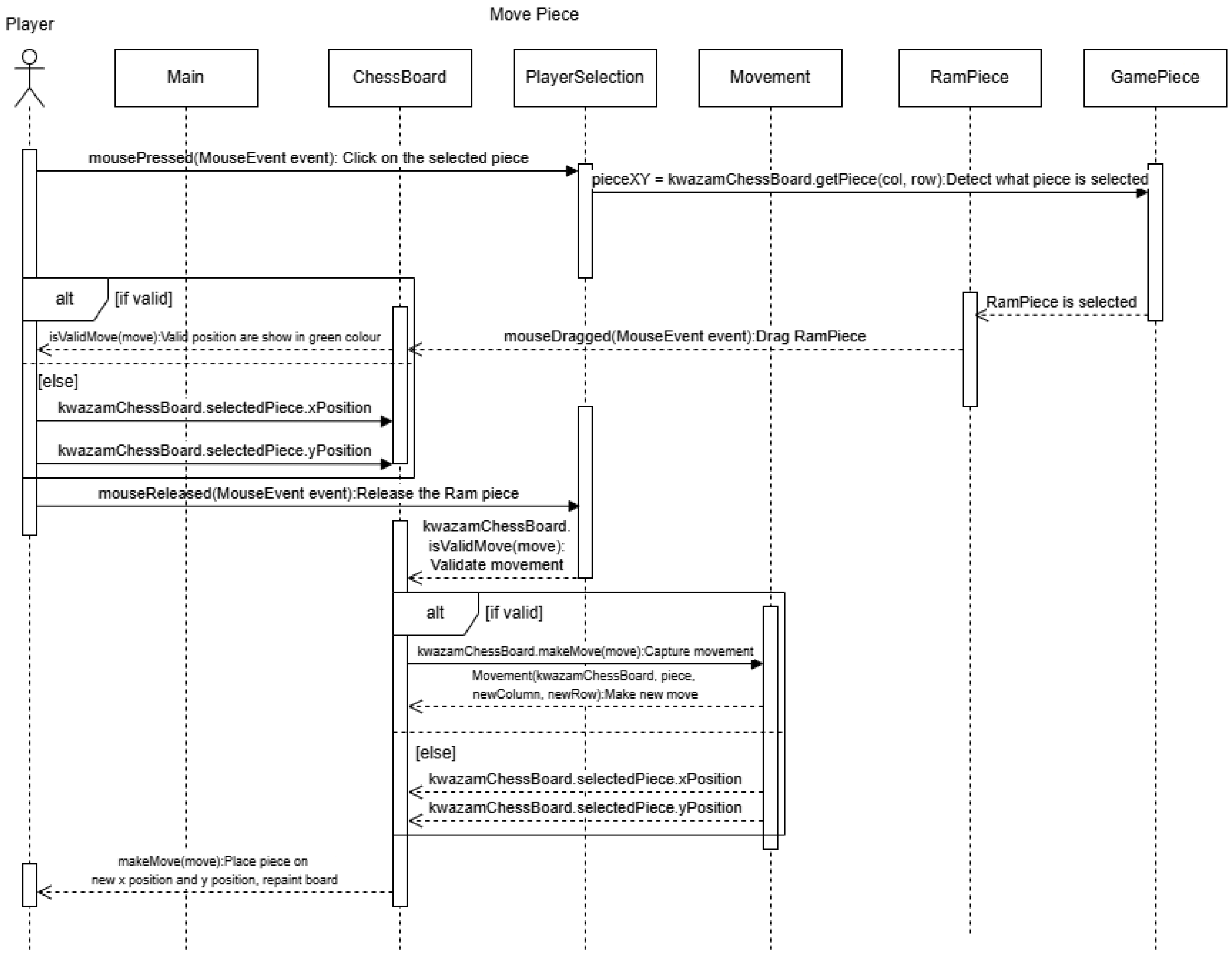
3.0 Sequence diagram based on our use case:

3.1 Start Game:



Player compiles and runs **Main** class (**Main.java**), the Main class then constructs the **ChessBoard()** object which initializes the game's visual board and creates a **JFrame**, the **MainMenu** object is then created and added to **frame**, setting up the user interface for starting or interacting with the game. Finally, the **chessboard** is displayed to the player.

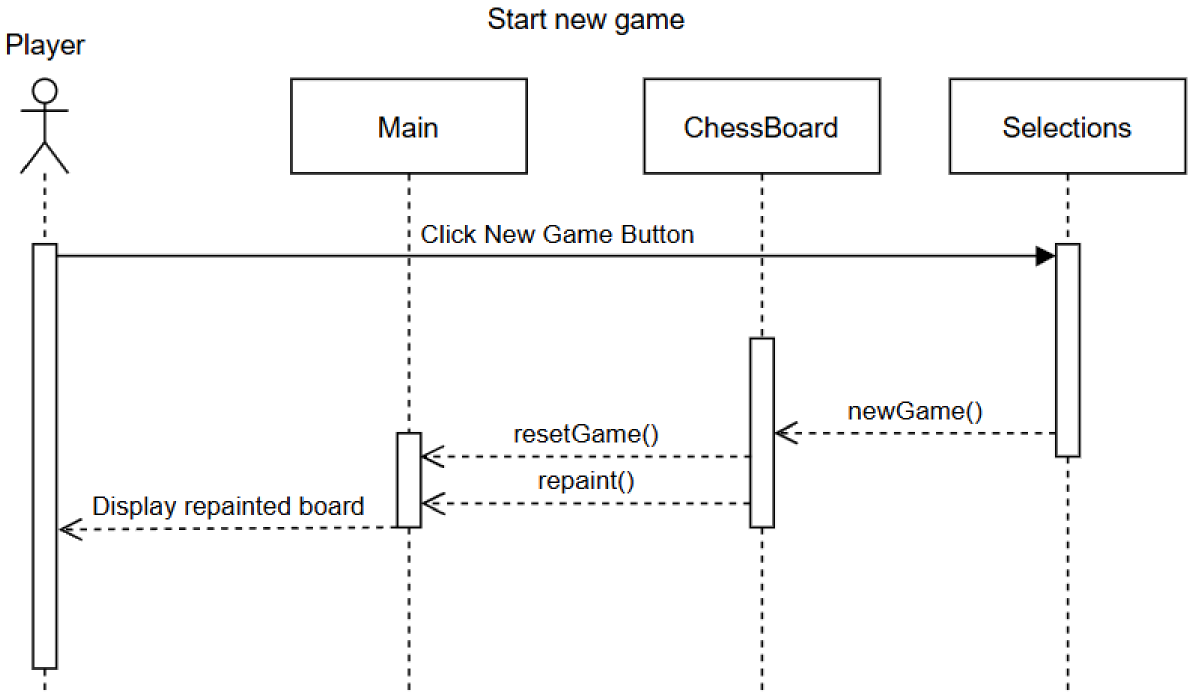
3.2 Moving the chess piece:


Move Piece
Player
Main
ChessBoard
PlayerSelection
Movement
RamPiece
GamePiece
mousePressed(MouseEvent event): Click on the selected piece
pieceXY = kwazamChessBoard.getPiece(col, row):Detect what piece is selected
alt
[if valid]
RamPiece is selected
isValidMove(move):Valid position are show in green colour
mouseDragged(MouseEvent event):Drag RamPiece
[else]
kwazamChessBoard.selectedPiece.xPosition
kwazamChessBoard.selectedPiece.yPosition
mouseReleased(MouseEvent event):Release the Ram piece
kwazamChessBoard.
isValidMove(move):
Validate movement
alt
[if valid]
kwazamChessBoard.makeMove(move):Capture movement
Movement(kwazamChessBoard, piece,
newColumn, newRow):Make new move
[else]
kwazamChessBoard.selectedPiece.xPosition
kwazamChessBoard.selectedPiece.yPosition
makeMove(move):Place piece on
new x position and y position, repaint board

The player clicks on a piece (**mousePressed**), the ChessBoard identifies it using **getPiece(col, row)**, if the move is valid (**isValidMove**), then possible moves are highlighted for that piece, when players drag the piece (**mouseDragged**), the movement is tracked.

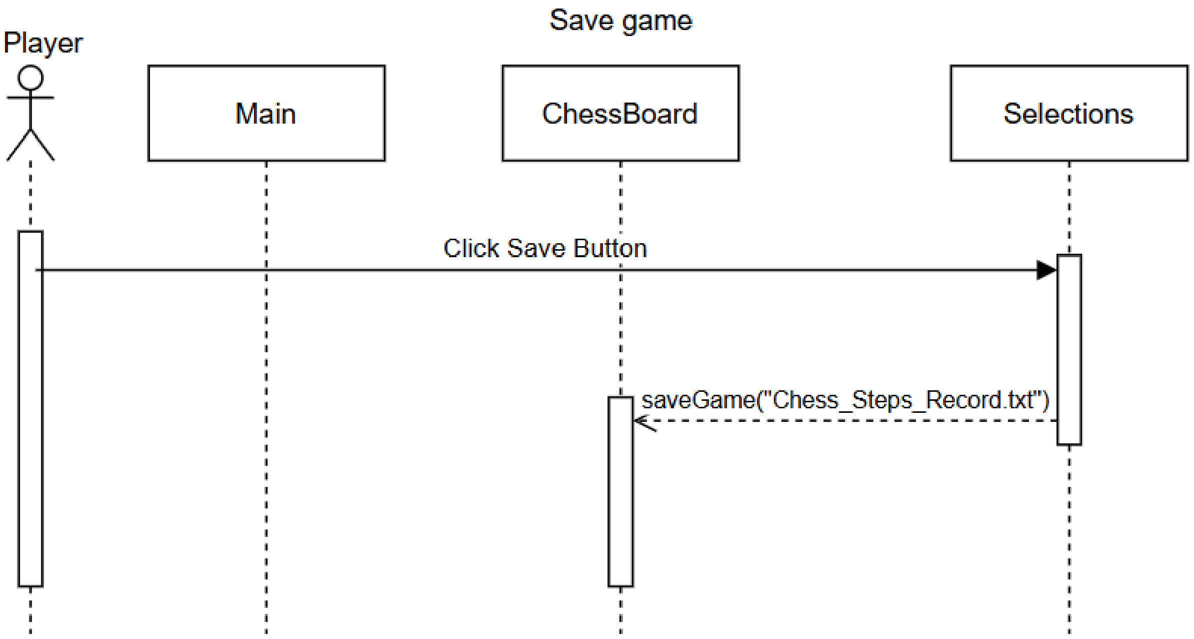
When the **Player** releases the piece (**mouseReleased**), the move is validated again. If valid, **makeMove** updates the board and the piece's position; otherwise, the piece returns to its original position. This ensures proper validation and updates for every move.

3.3 Start a New Game:



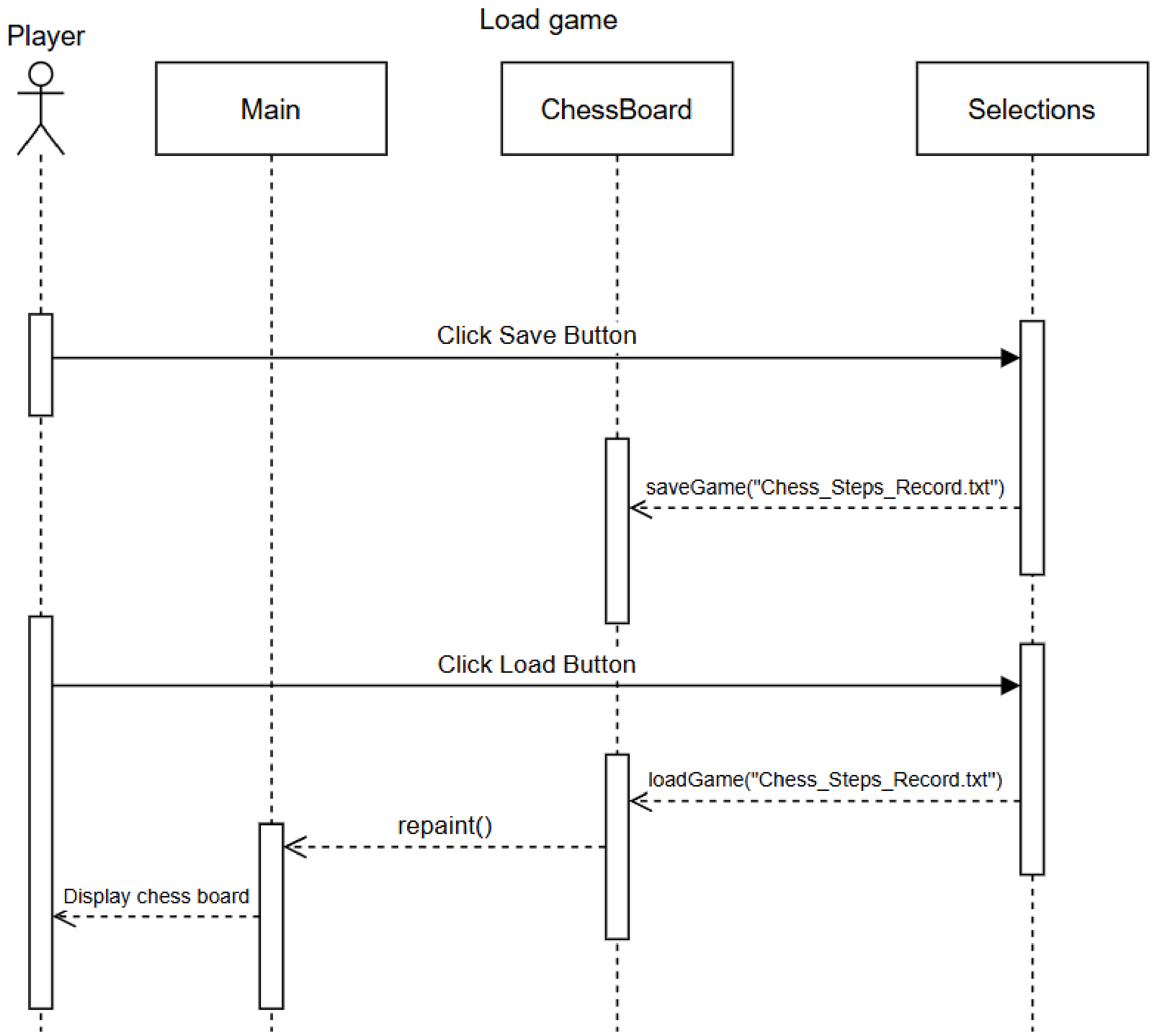
To start a **new game**, the player has to click a new game button which has an action listener located in the **Selections** , **the newGame()** would be invoked causing the count and round of the game to reset. This triggers **resetGame()** in **ChessBoard**, followed by **repaint()** to update the board visually. The updated chessboard is then displayed to the player.

3.4 Save Game:



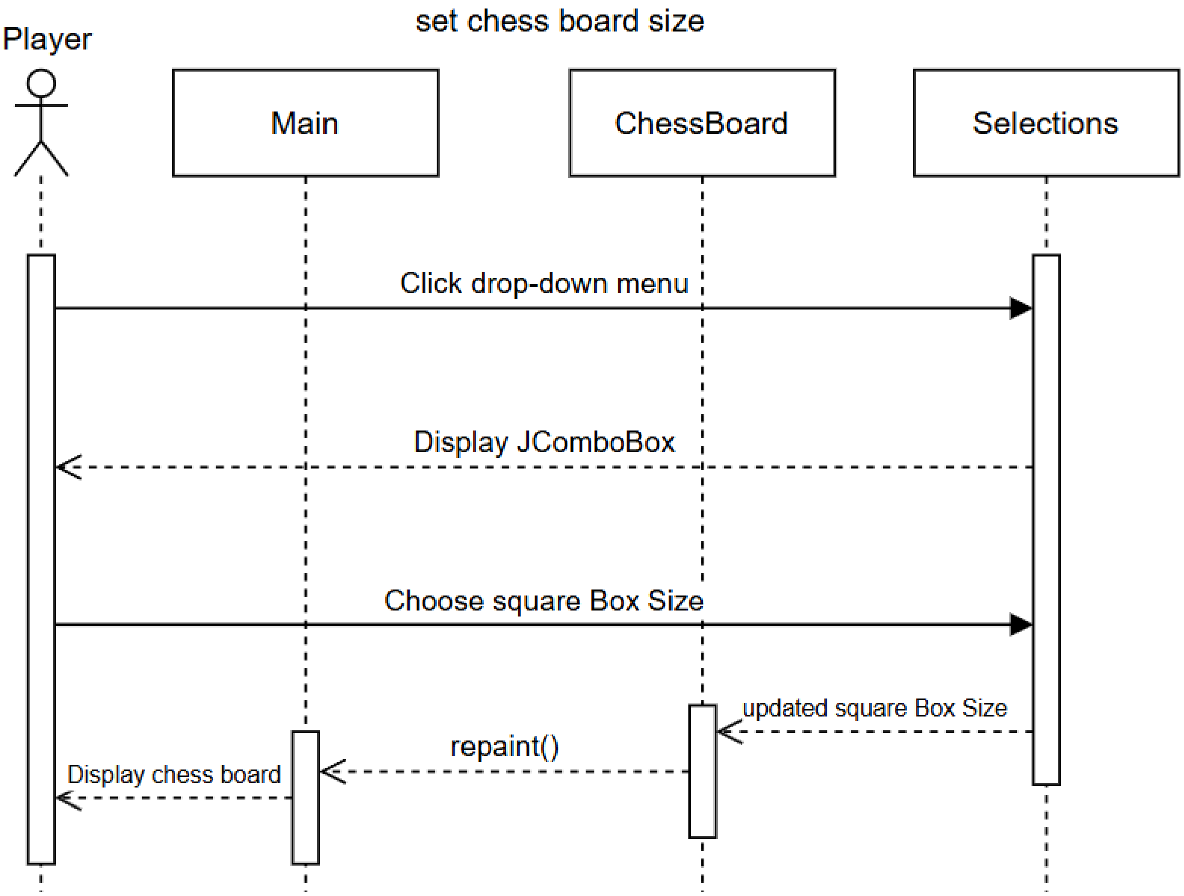
To save a game, the player has to click the **save button** which has an action listener located in the **Selections** , the details of the current game would be saved into the **Chess_Steps_Record.txt.**

3.5 Load Game:



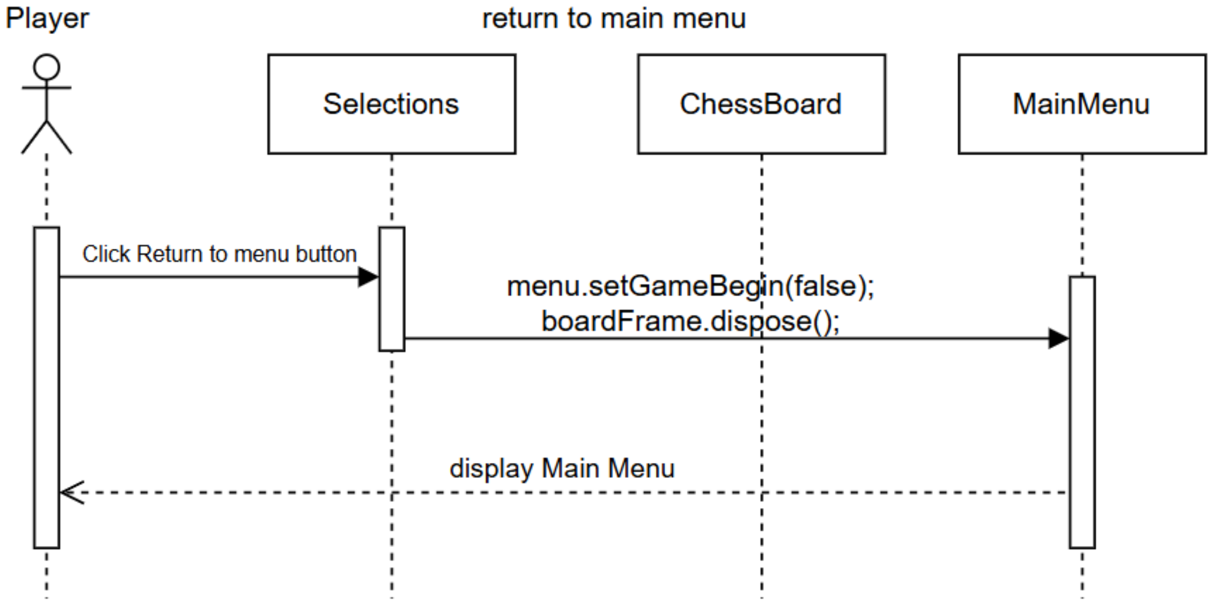
To load a game, the player has to click the **load button** where it has an action listener located in the **Selections** which would allow the **repaint()** to repaint the chess board based on the last saved details of the game. And display the chess board.

3.6 Set Chess Board Size:



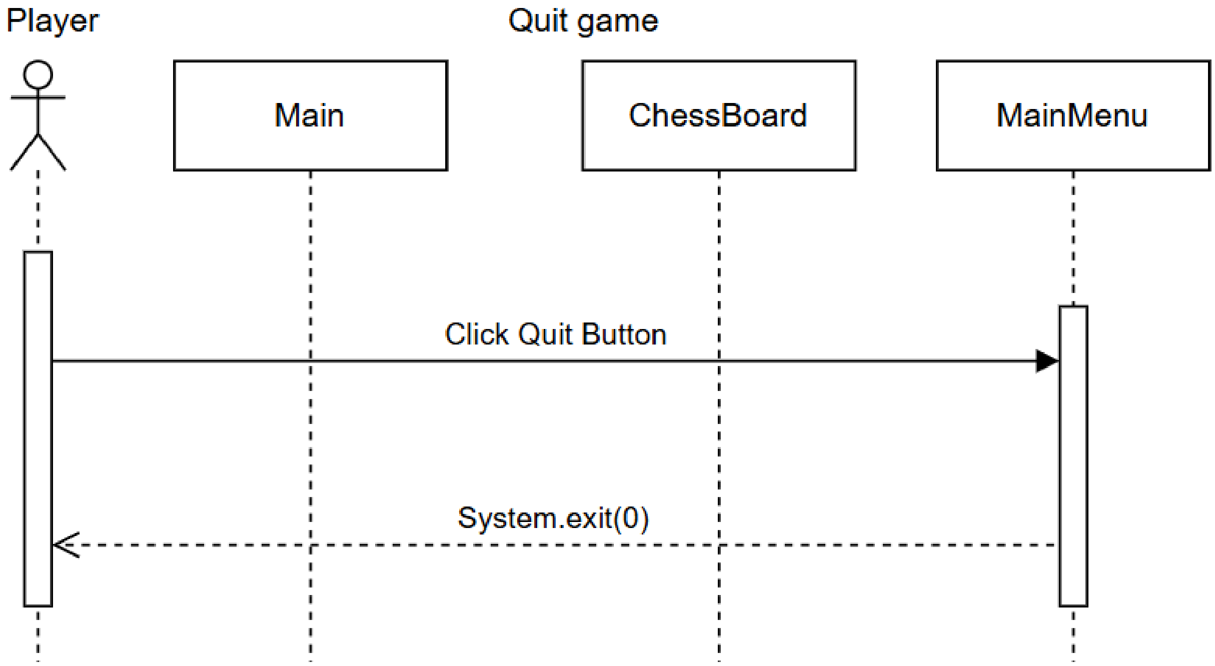
The player has to use the **Drop-down menu** which has an action listener in the **Selections** , The **JCombobox** will have three options for chess board size which are "**Standard**", "**Mid-Sized**", and "**Extra-Large**". The options square box size for **Standard = 70**, **Mid-Sized = 80**, and **Extra-Large = 90**. After choosing the square box size, it will update the square box size to the chess board. The **Selections** which would allow the **repaint()** to repaint new square box size for the chess board. And display the chess board.

3.7 Return to main menu:



The player has to click **return to menu button** which it has an action listener in the **MainMenu**, the **menu.setGameBegin()** will set the boolean to false so that the board would be dispose and the frame of the board would be closed and the frame of **Main Menu** will be displayed to the player.

3.8 Quit Game:



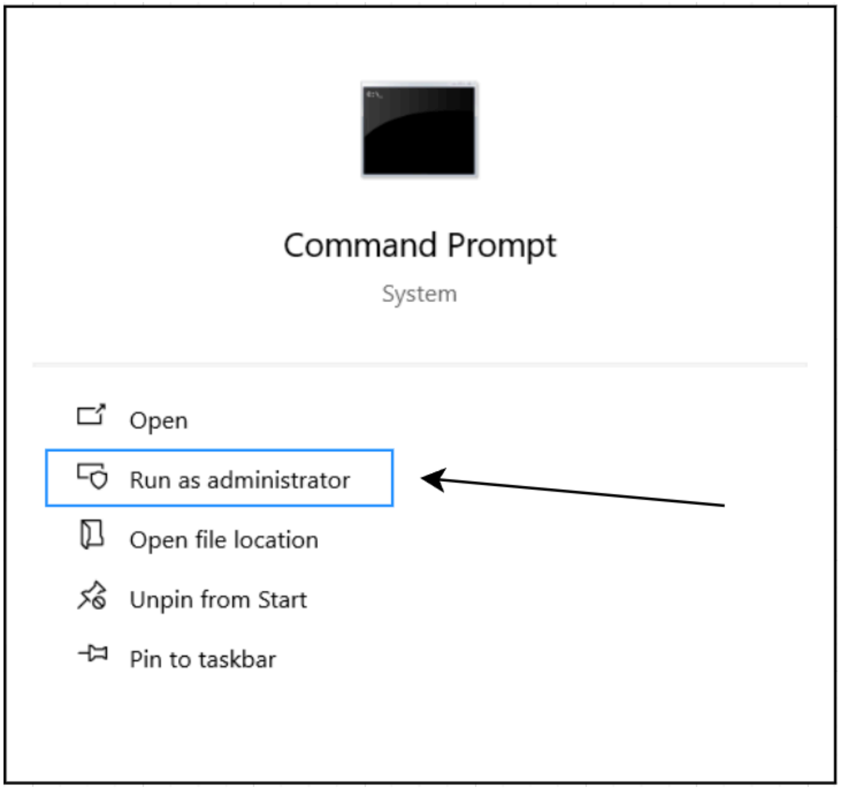
The player has clicked the **quit game button** and the system status will be set to 0.

4.0 Documentation

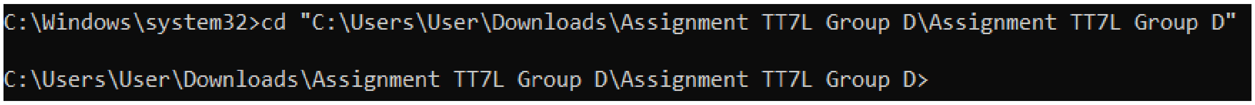
4.0 User Manual

4.1 Compile and run instructions in the command line guide

1. Open a terminal or command prompt as administrator.



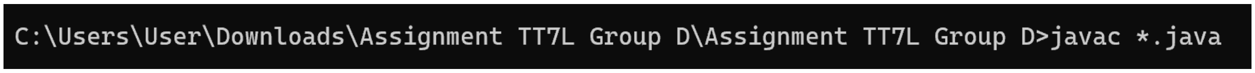
2. Navigate to the directory where your Java files are located using the ‘cd’ command like such in the example below. Remember to replace it with the actual path to your Java source files.



```
C:\Windows\system32>cd "C:\Users\User\Downloads\Assignment TT7L Group D\Assignment TT7L Group D"

C:\Users\User\Downloads\Assignment TT7L Group D\Assignment TT7L Group D>
```

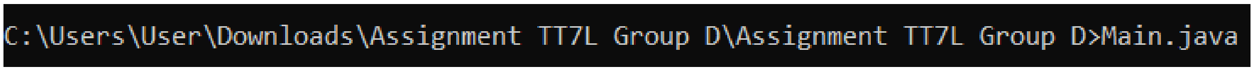
3. Compile the Java source files using the ‘javac’ command. The javac command is the Java Compiler, and using *.java as an argument means it will compile all .java files in the current directory.



```
C:\Users\User\Downloads\Assignment TT7L Group D\Assignment TT7L Group D>javac *.java
```

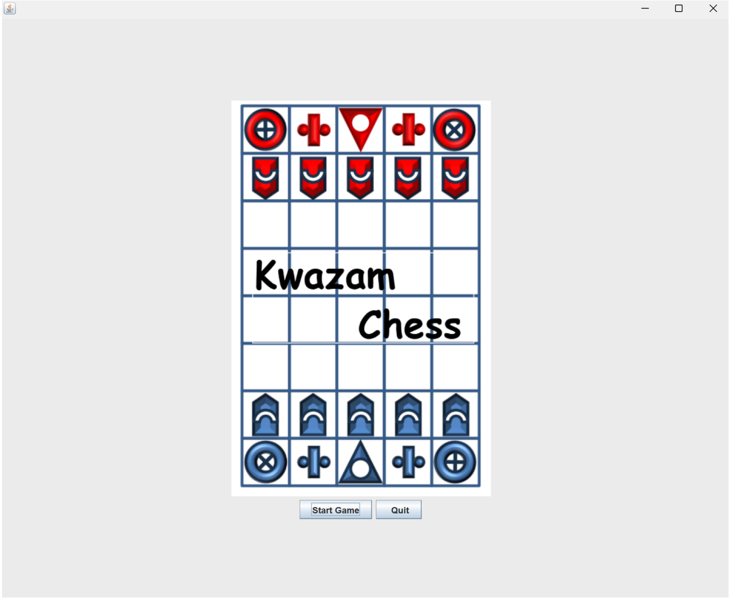
4. Check the terminal for any error messages. If there are no errors, the compilation process is successful.

5. After successful compilation, run the Java program using the java command. Specify the name of the class containing the main method. In this case, the main class is called “Main”.



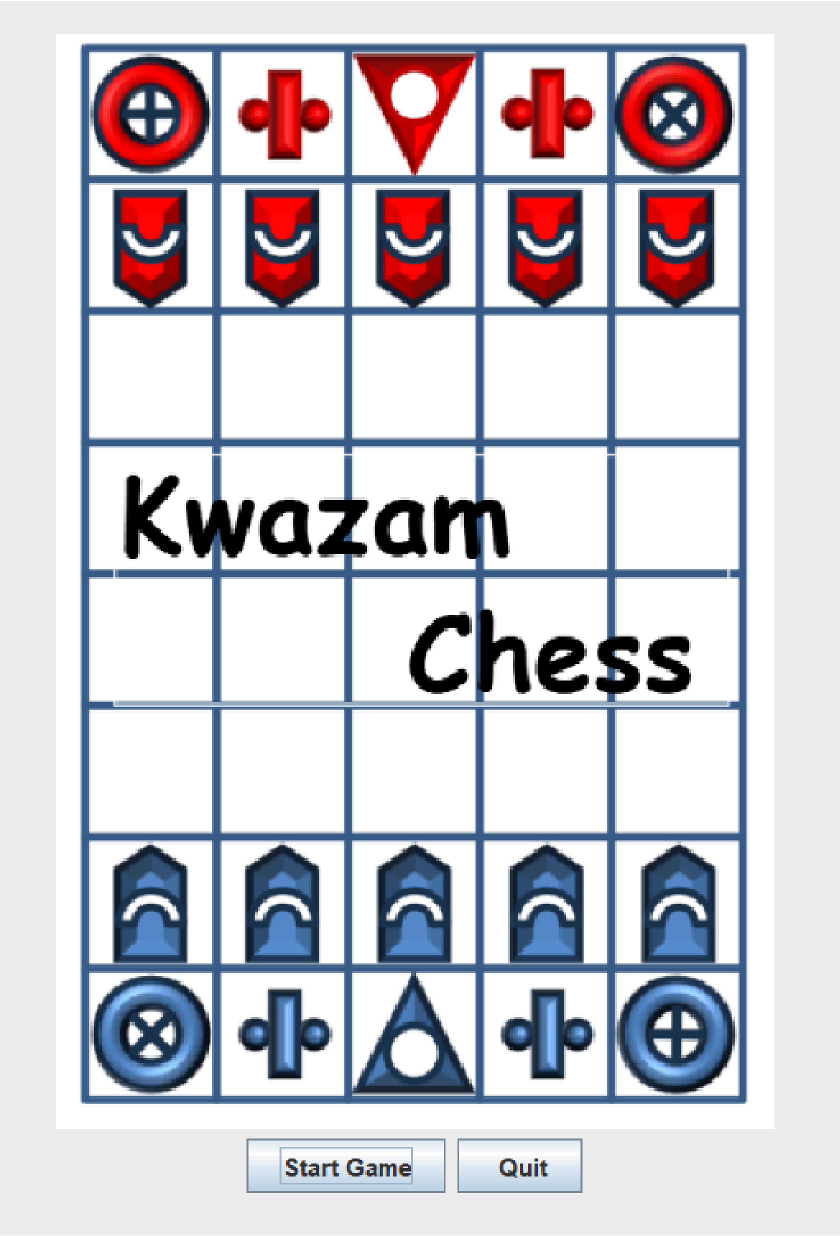
```
C:\Users\User\Downloads\Assignment TT7L Group D\Assignment TT7L Group D>Main.java
```

6. The program should now execute, you should see the output in the terminal. Enjoy playing Kwazam Chess.

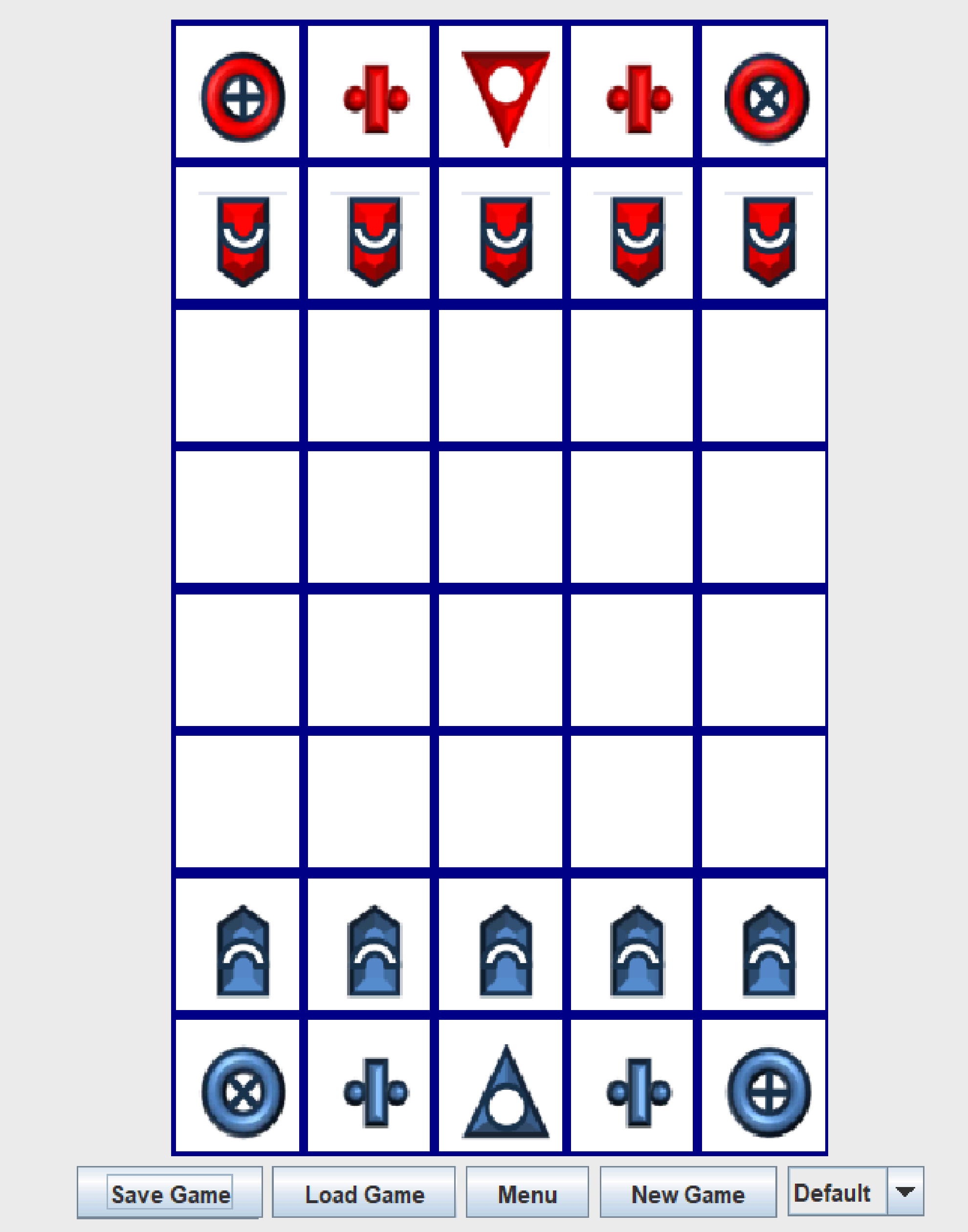


4.2 Kwazam Chess RULES and Gameplay

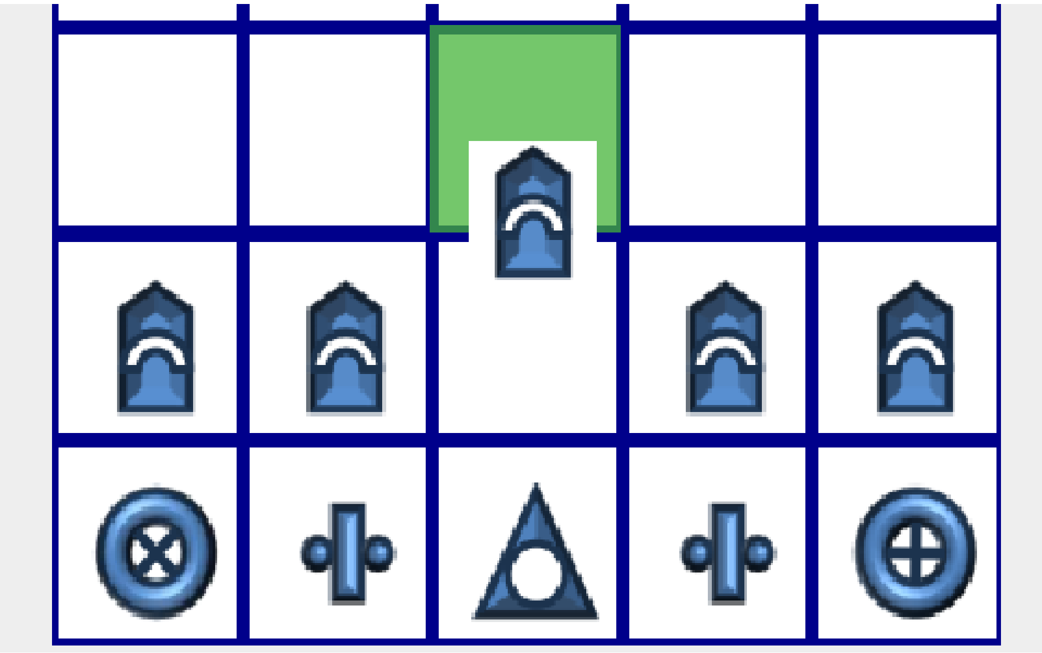
1. The **Main Menu** of Kwazam Chess in which the player is allowed to select "**New Game**" to start a brand new save file and "**Quit**" to exit the program.



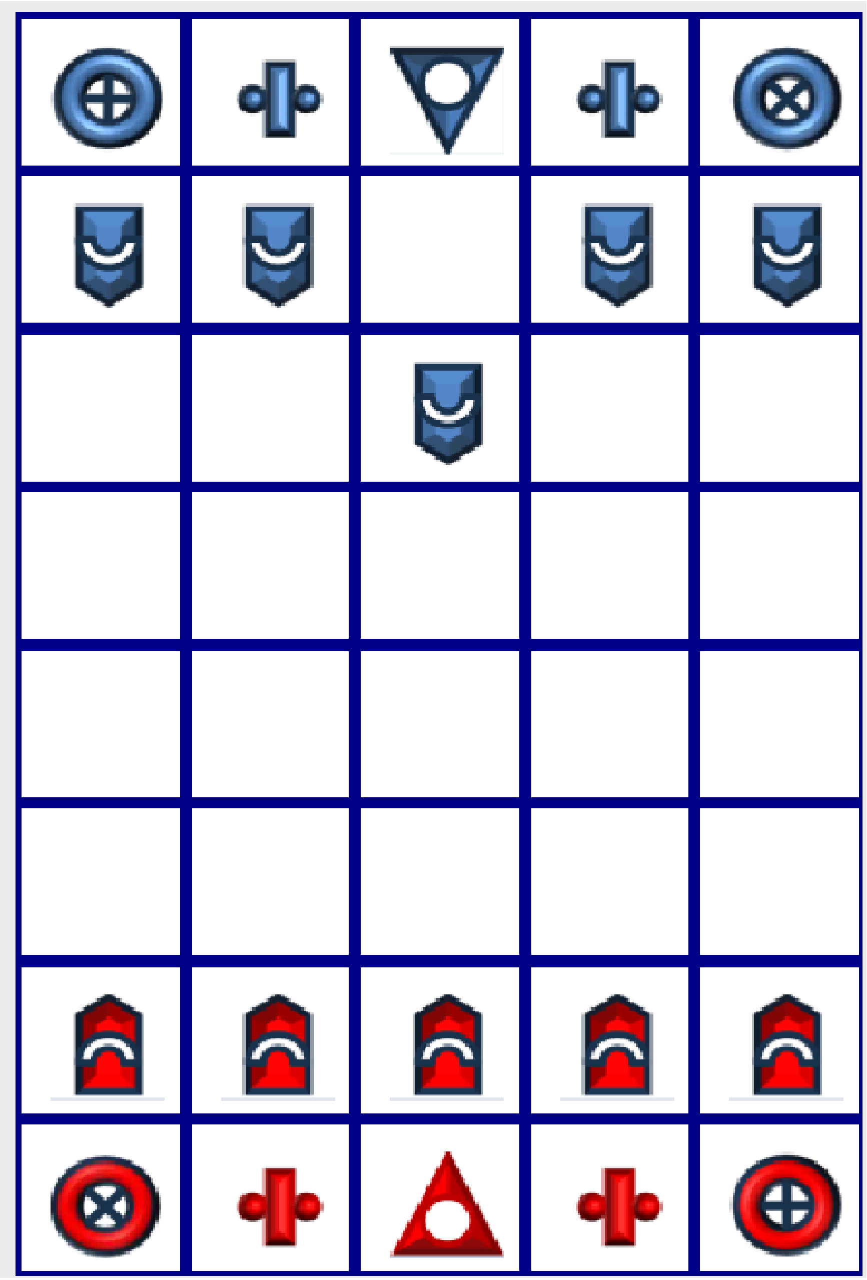
2. The chess board, where the **main interface** for players to play the game at.



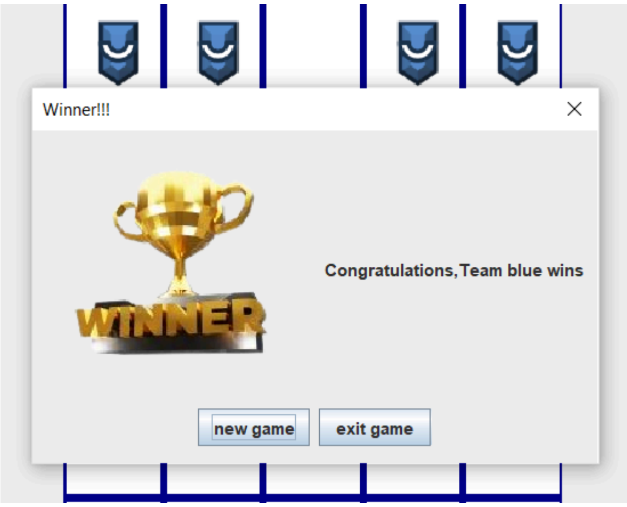
3. In the following picture, this shows that whenever the player wants to move a piece from the board, it will illustrate, in green, the move that the player can attempt in case the player doesn't know the moveset.



4. Once player 1 does their move, the board will switch sides and player 2 will get to have a turn.

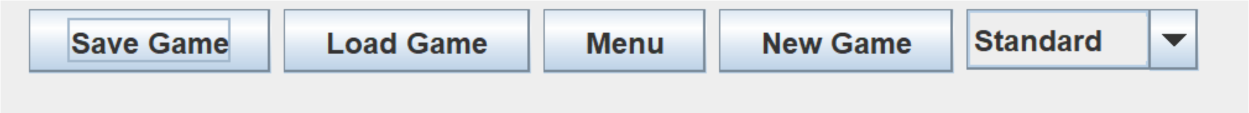


5. The game ends whenever either side of the player can capture the other side’s **Sau piece** and a victory screen will appear if this happens. Players can once again press “new game” to continue or “exit game” to quit.

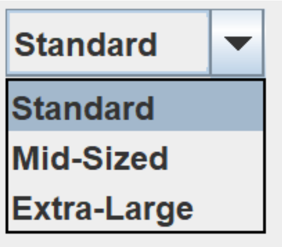


Congratulations,Team blue wins

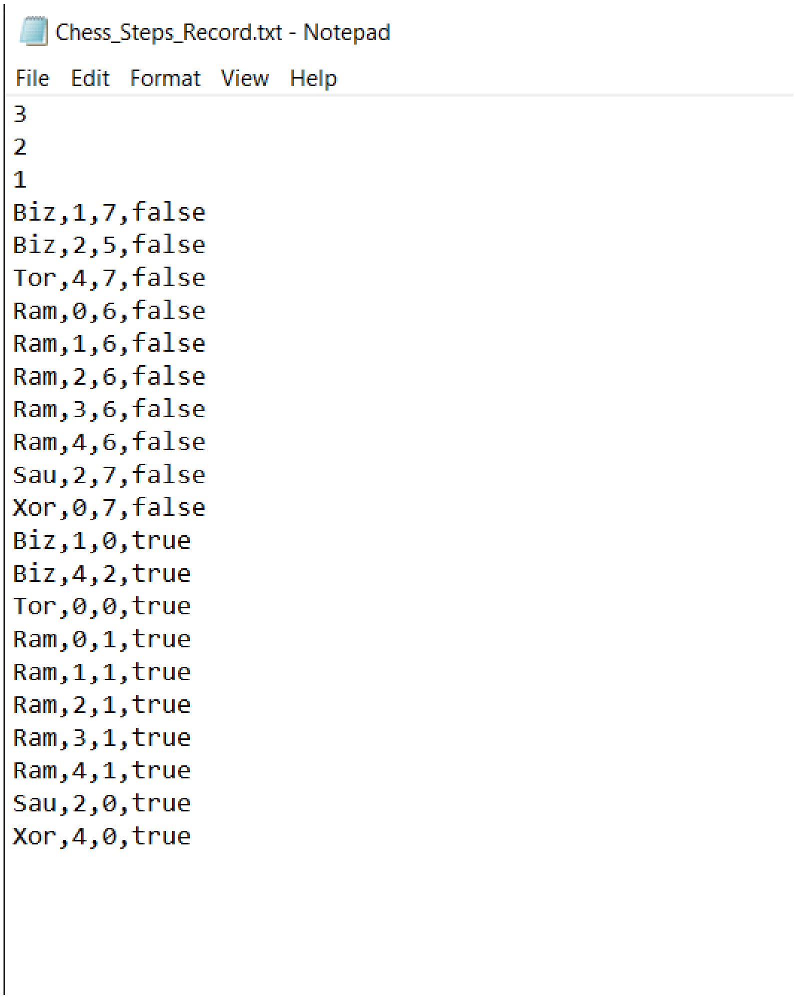
6. During the game, players have the option to either “**save game**” which saves the current state as a text file, “**load game**” to continue a game saved, “**menu**” to go back to the main menu and lastly, “**new game**” to start a new state.



7. And besides the options there is a dropdown menu which allows players to choose the size of the board.

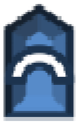
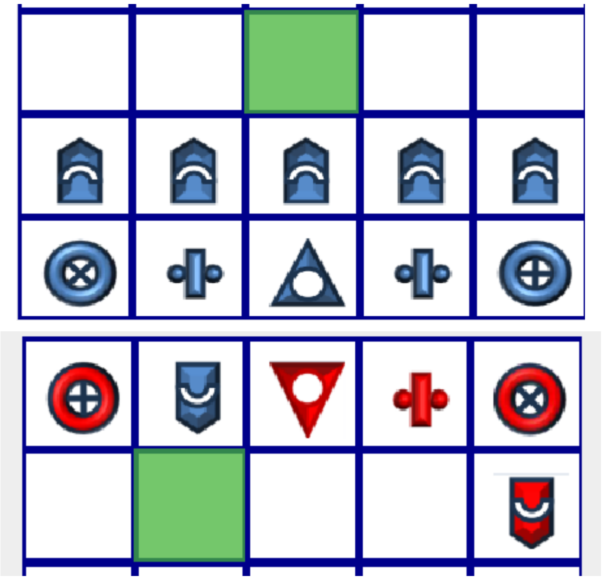
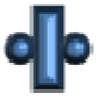
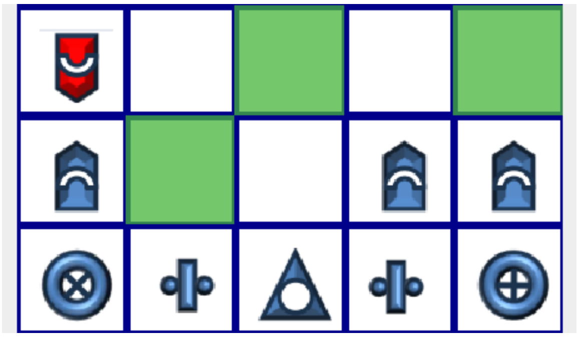
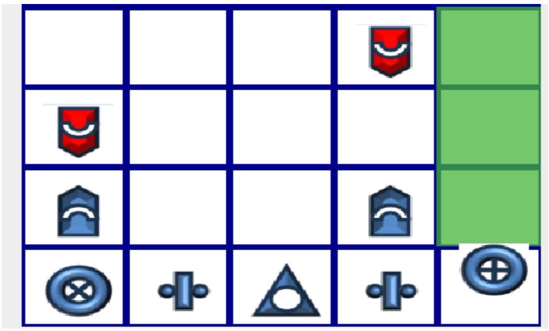


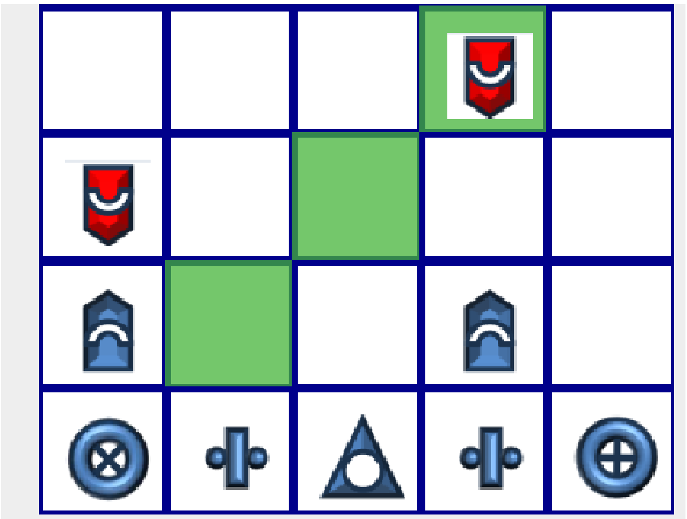
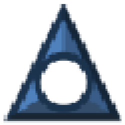
8. The text file for the saved state.
    - The first row (3) is mean Save whose turn it is (Count to check Red or Blue turn)
    - The second row (2) is mean Save the current round (Round to change the position of Tor and Xor)
    - The third row (1) is mean Save the SauRam count (Count to change the Image Sau and Ram)
    - Start from the Fourth row it will save the Chess Piece that move (Piece type, Piece Column, Piece Row, Piece Color(Red Piece or Blue Piece) true is red and false is blue)



```
3
2
1
Biz,1,7,false
Biz,2,5,false
Tor,4,7,false
Ram,0,6,false
Ram,1,6,false
Ram,2,6,false
Ram,3,6,false
Ram,4,6,false
Sau,2,7,false
Xor,0,7,false
Biz,1,0,true
Biz,4,2,true
Tor,0,0,true
Ram,0,1,true
Ram,1,1,true
Ram,2,1,true
Ram,3,1,true
Ram,4,1,true
Sau,2,0,true
Xor,4,0,true
```

Kwazam Chess Moves

The Ram Piece 	This piece can only move forward 1 step once per turn and at the end of the board, it will turn around and head the other way. 
The Biz Piece 	This piece moves in a 3x2 L shape in any orientation and can skip over other pieces. 
The Tor piece 	This piece can move orthogonally only but can go any distance. It cannot skip over other pieces. However, after 2 turns, it transforms into the Xor piece. 

The Xor piece 	This can move diagonally only but can go any distance. It cannot skip over other pieces. However, after 2 turns, it transforms into the Tor piece. 
The Sau piece 	Basically the equivalent of the Queen from chess, can only move 1 step in any direction but if it gets captured, you lose.