TCP2201 Project

Trimester 1, 2020/2021

By <<Poland>>

Name	Student ID	Phone Number	Email
Mang Yu Jie (Team Leader)	1181101662	01110780001	1181101662@student.mmu.edu.my
Lee Guang Shen	1171103543	0135881838	1171103543@student.mmu.edu.my
Leim Jing Han	1171103793	0186617713	1171103793@student.mmu.edu.my
Low Min Xuan	1181100653	01126351496	1181100653@student.mmu.edu.my
Oi Zhen Fan	1181300513	0193521017	1181300513@student.mmu.edu.my

Methods to Compile the Program (JDK 8 and above needed)

1. Open Webale_Chess and open src
2. In the directory of src file, type "cmd"

```
C:\Windows\System32\cmd.exe

Microsoft Windows [Version 10.0.19041.508]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\gslee\Desktop\Webale_Chess\src>_
```

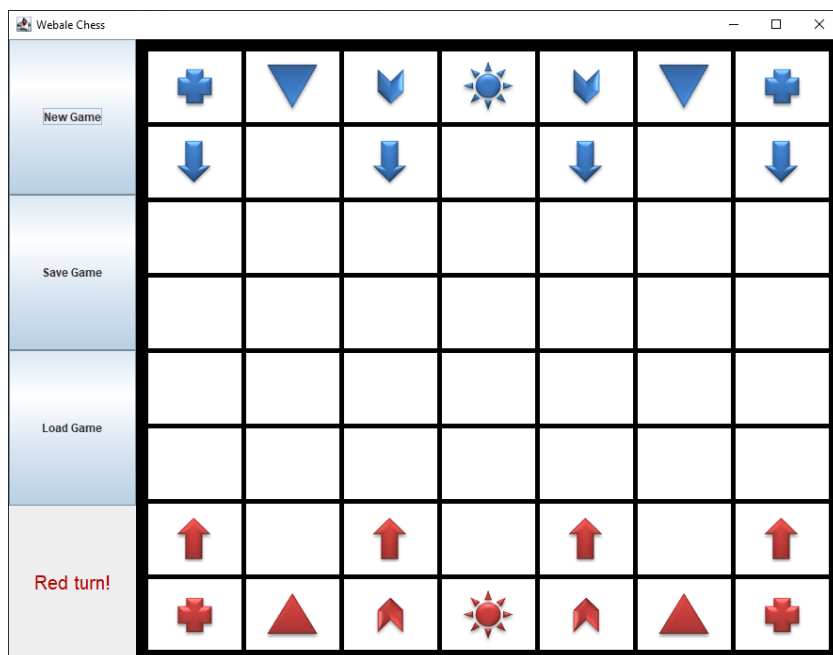3. Type "javac *.java" to compile the java program

```
C:\Windows\System32\cmd.exe

Microsoft Windows [Version 10.0.19041.508]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\gslee\Desktop\Webale_Chess\src>javac *.java_
```
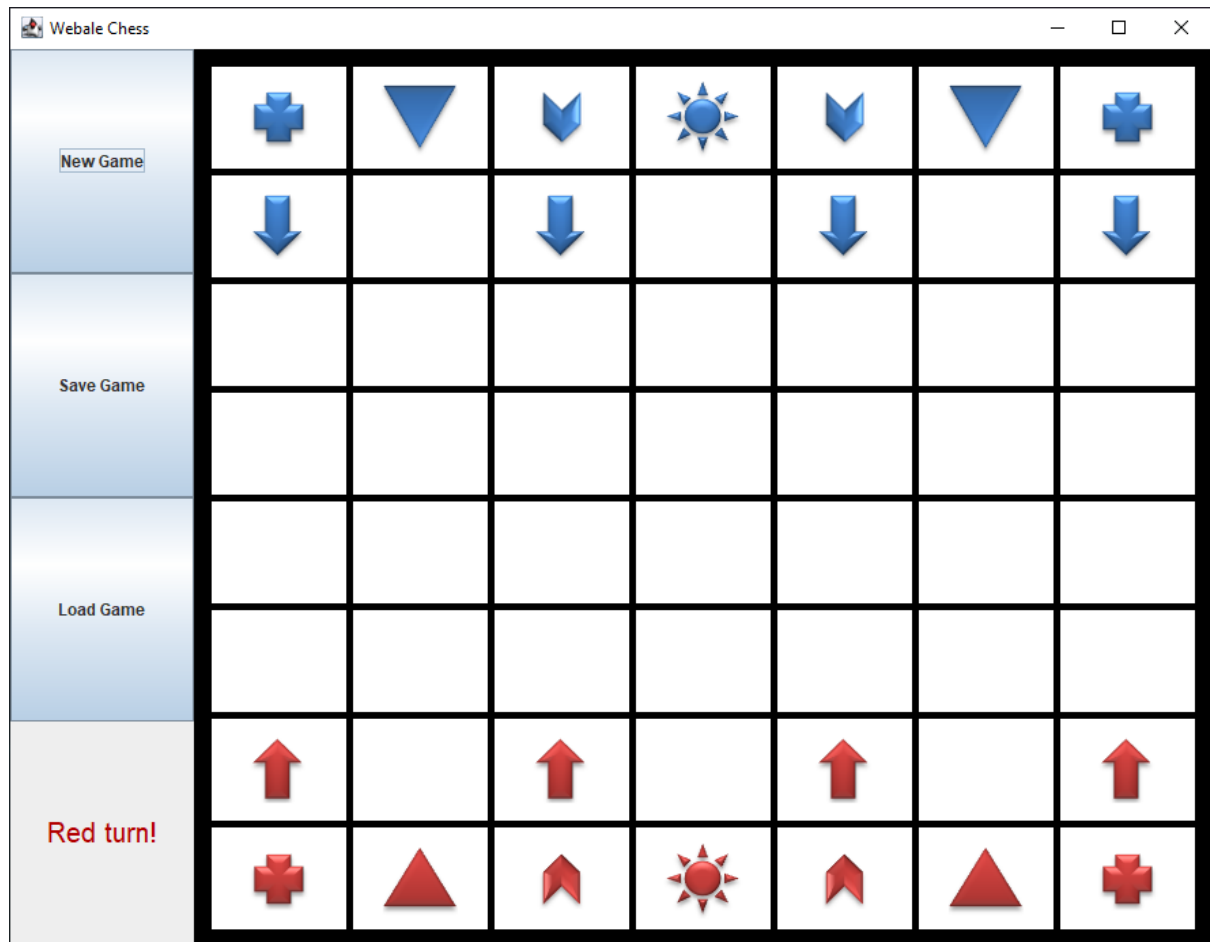
4. Type "java JChess" to run the program.

5. Program will pop out.

Command: [Directory where you have saved the file]\Webale_Chess\src>javac *.java

[Directory where you have saved the file]\Webale_Chess\src>java JChess

Documentation of the Program

Figure 1

Figure 1 shows the Webale Chess game interface when the program runs . The players are presented with the chessboard, with Red Pieces and Blue Pieces, both representing each player1 and player2. On the left, there are three buttons for initializing a new game, saving a game and loading a previous game from a txt file. There is also a text display on the bottom left corner that indicates the current player's turn.
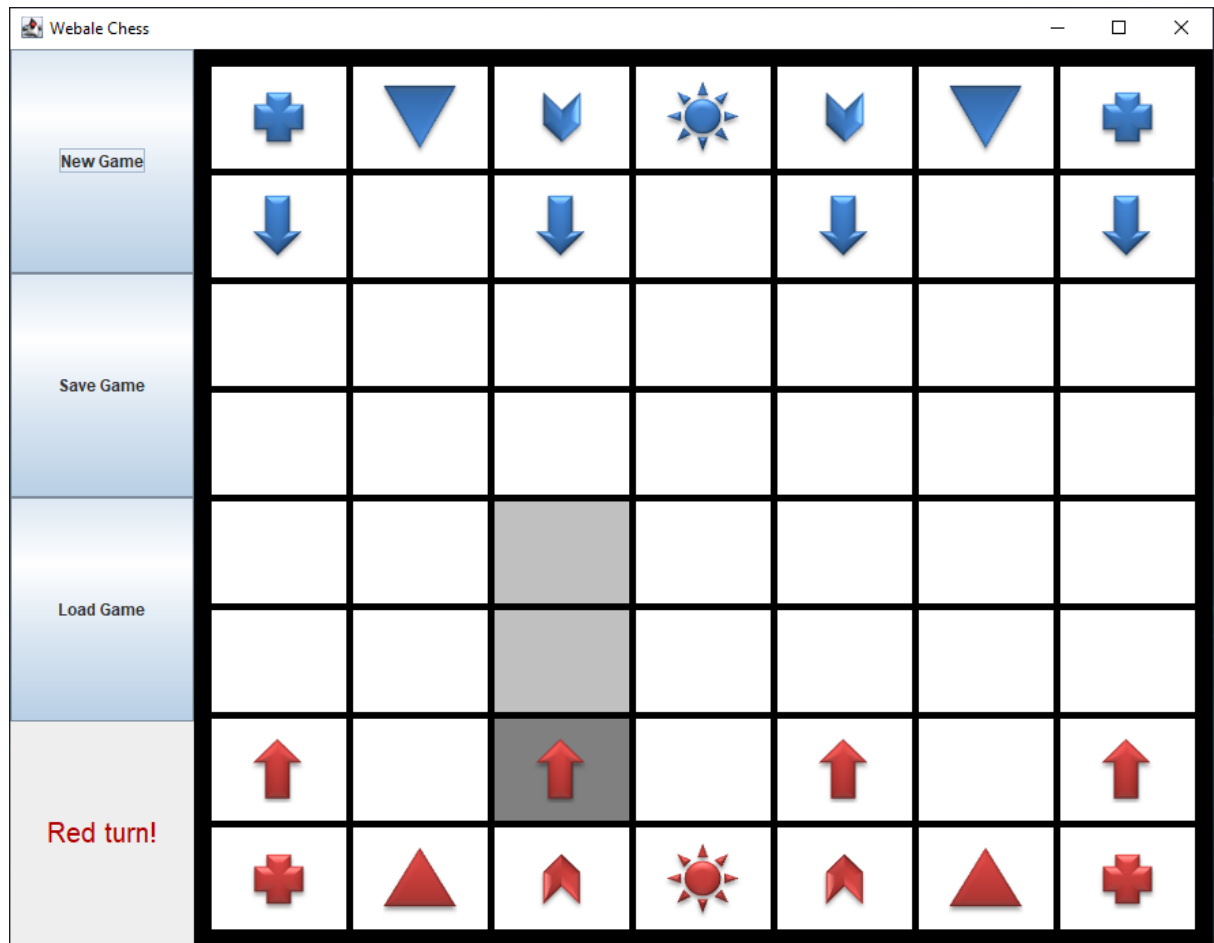
Figure 2

As shown in Figure 2, when the player clicks on a chess piece, the selected piece will have a darker highlight and the available moves will be highlighted in a lighter shade. In this case, Player 1 which is controlling the Red chess piece, the selected chess piece is an Arrow.
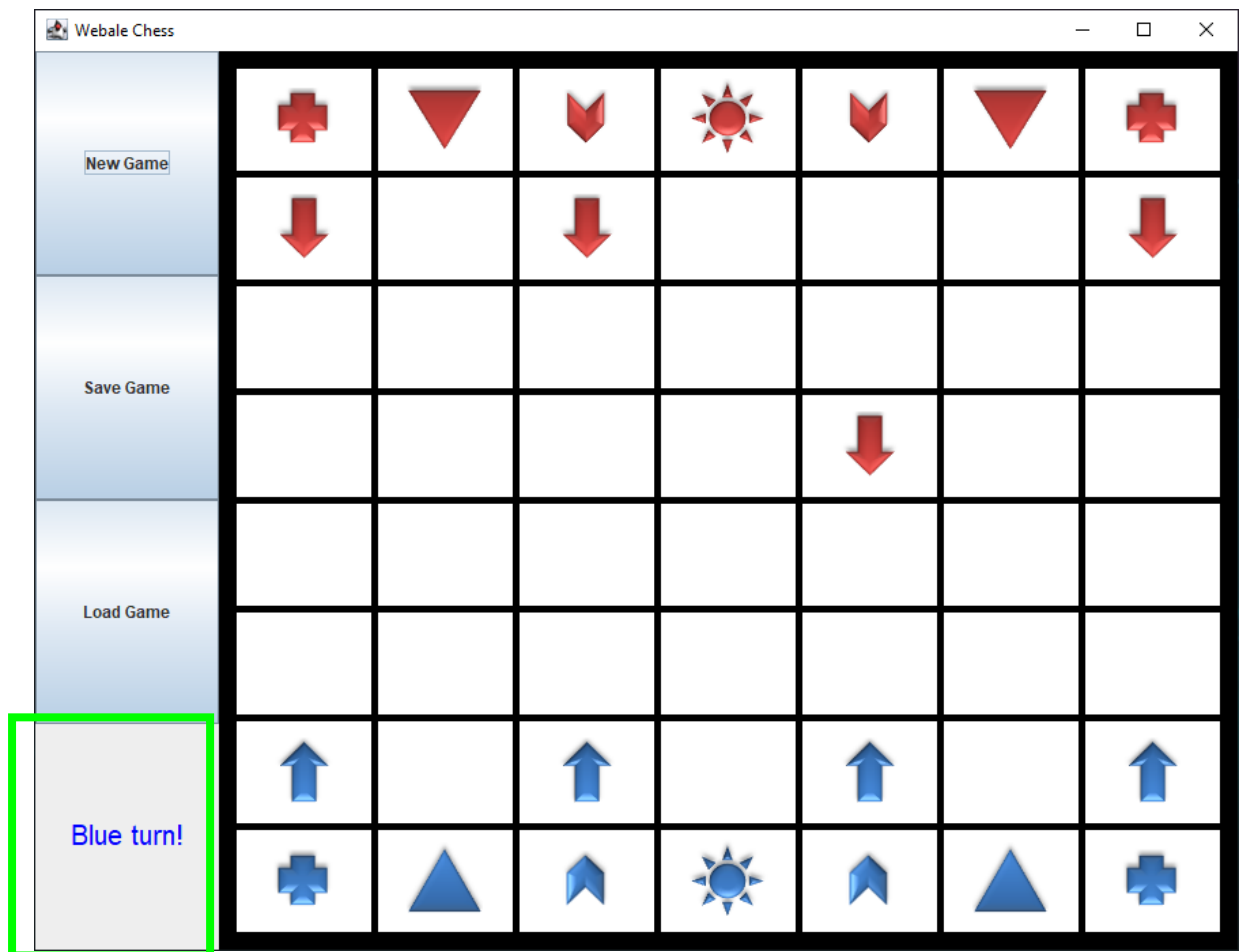
Figure 3

In Figure 3, when Player1 has moved the Red Arrow Chess Piece , the green-highlighted area displays the current player's turn. In this case, it's Player2's turn. The board will then rotate to have the Blue Chess Pieces to the bottom for Player2 to ease the player's view.
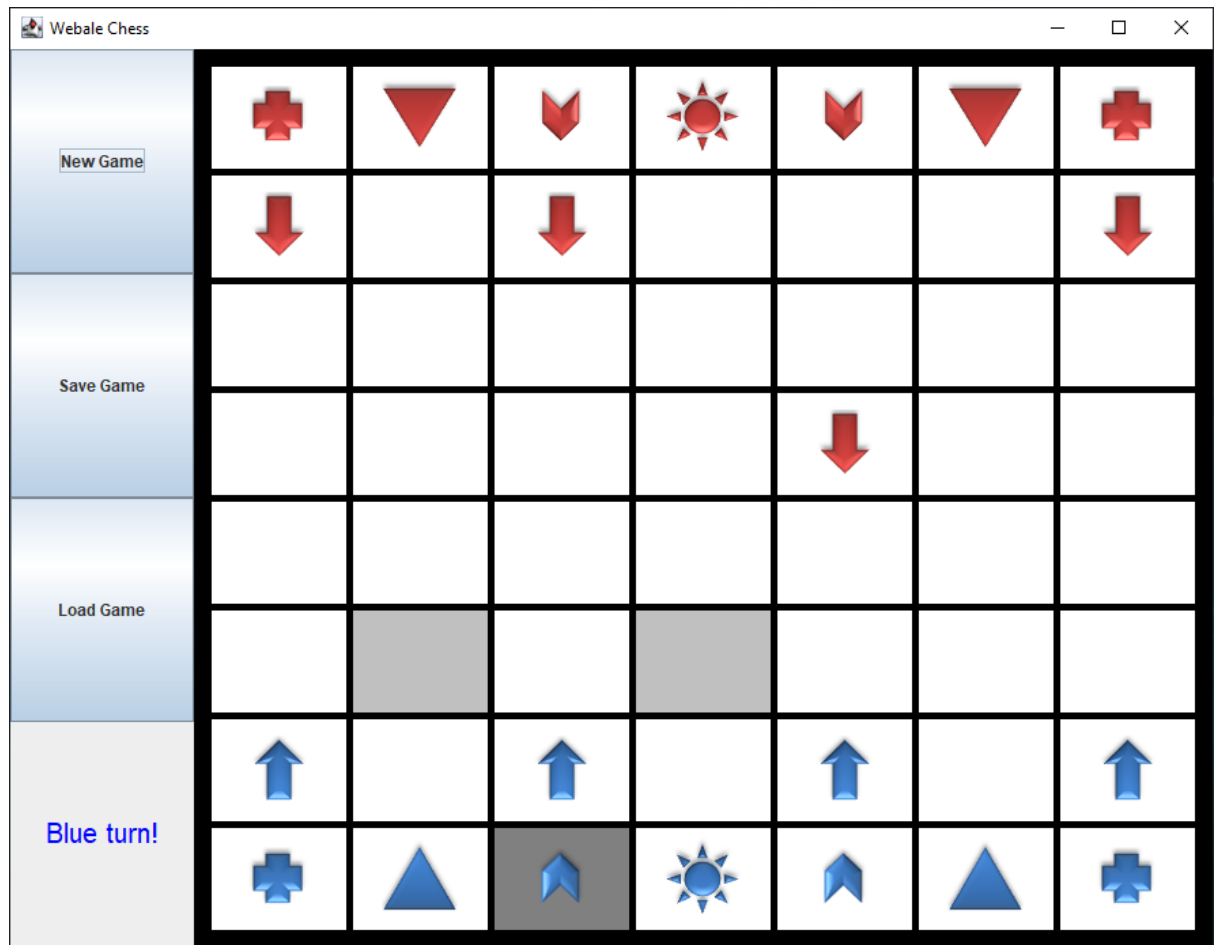
Figure 4

When Player2 has selected the piece that he/she wanted to move, the piece will then have the highlights showing the chess piece's available moves. As shown in Figure 4 above, the Chevron Chess Piece movement is moving 2 steps in one direction then 1 step perpendicular to it, it can skip over other chess pieces.
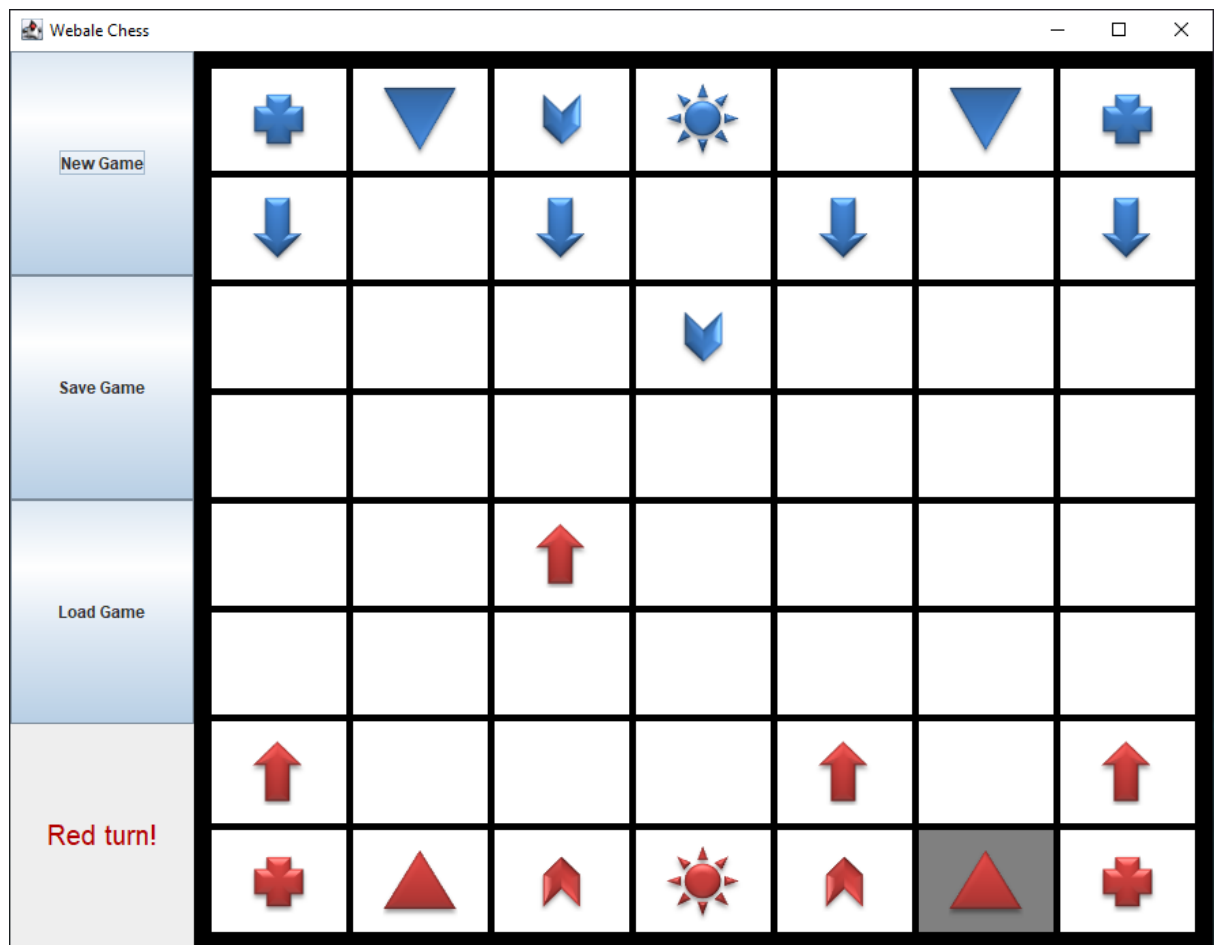
Figure 5

If the chess piece selected is being blocked by other chess pieces of the same player, it will not show any highlights as it has no available moves.
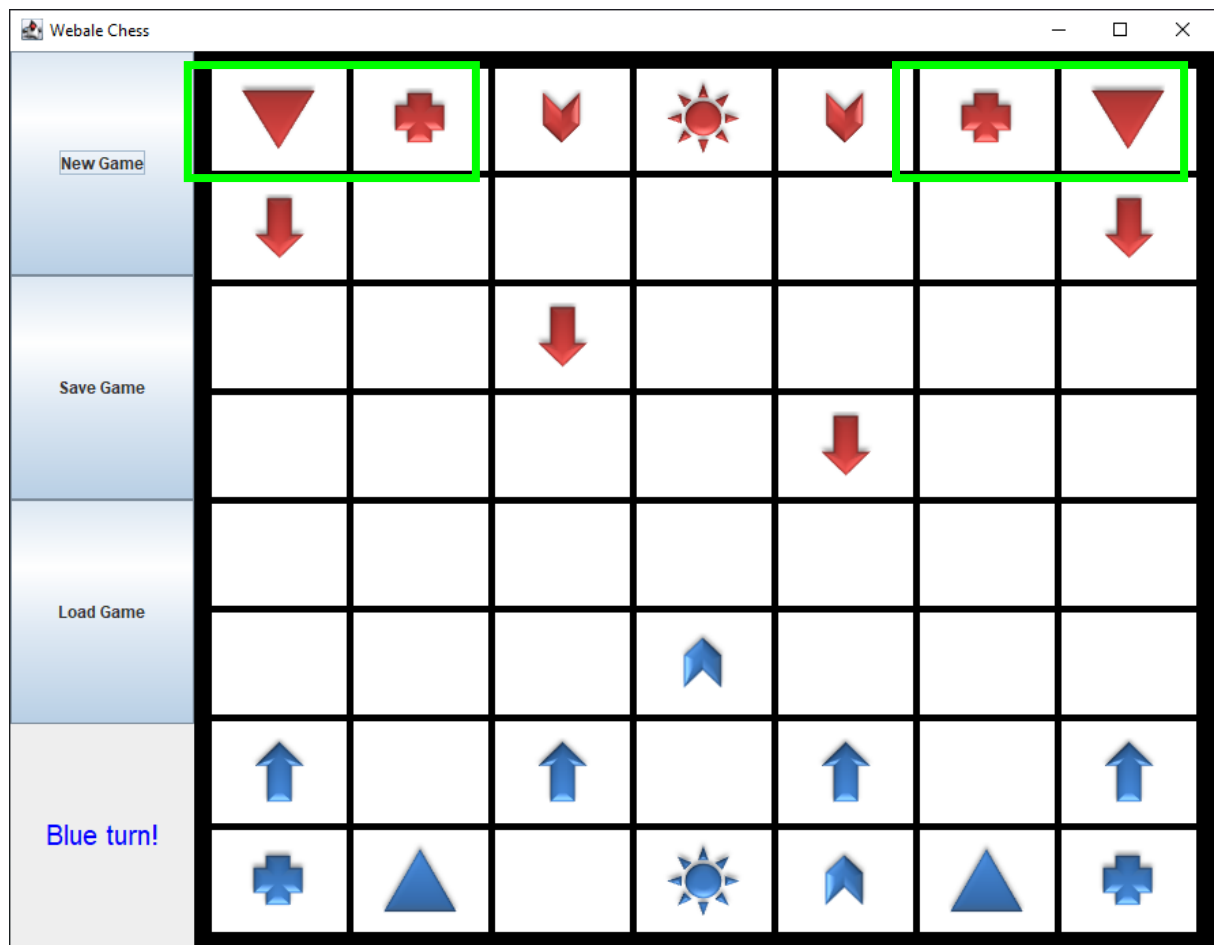
Figure 6

After every 2 turns for each player, all the Triangles will turn into Pluses and vice versa. For example, as shown in Figure 6, all the red Triangles will turn into Plusses and vice versa.
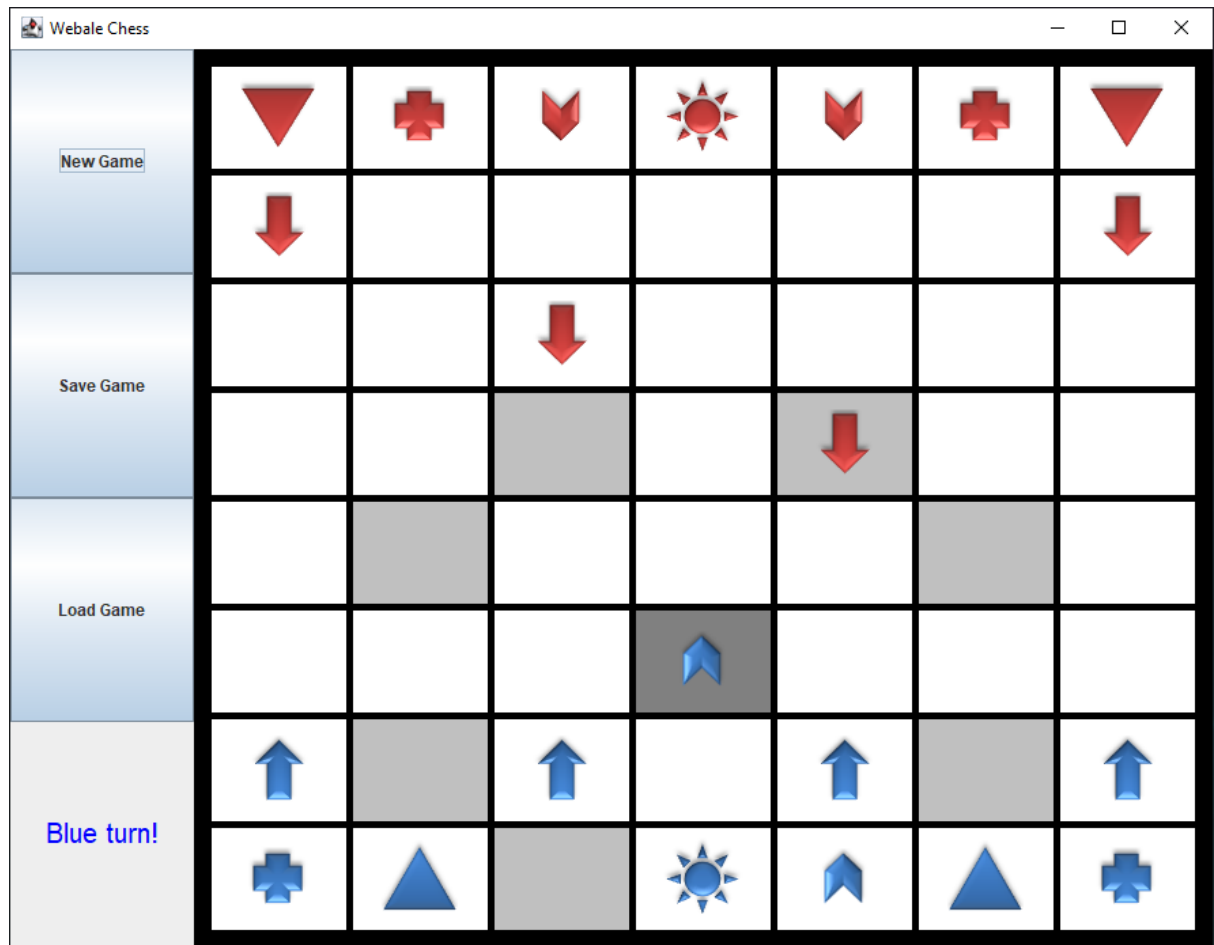
Figure 7

When Player2 has selected the piece that he/she wanted to move, the piece will then have the highlights showing the chess piece's available moves, including if the chess piece of Player1 can be captured. As shown in Figure 7, the Blue Chevron can capture the Red Arrow, as the Red Arrow is in the highlighted area of the Chevron available moves.
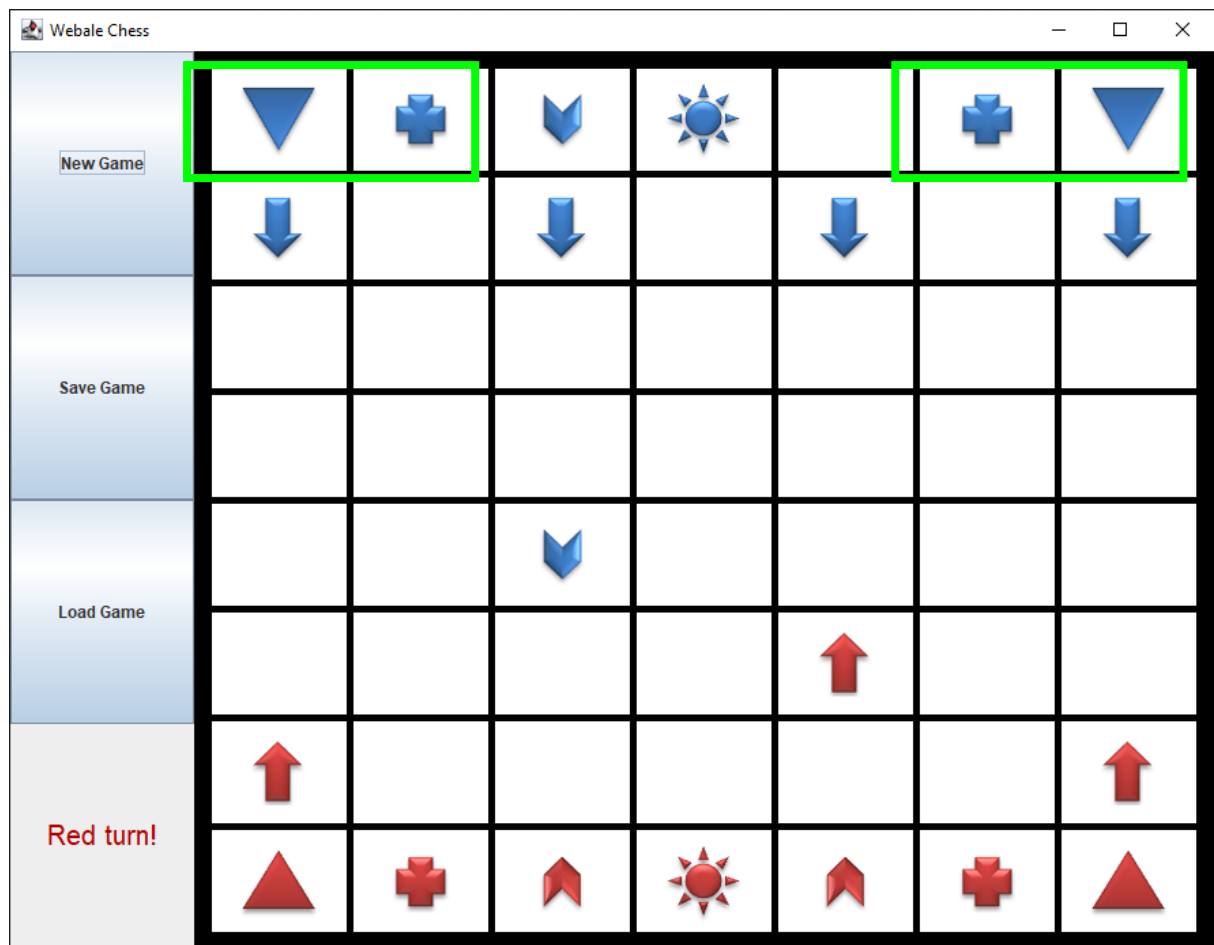
Figure 8

After every 2 turns for each player, all the Triangles will turn into Pluses and vice versa. For example, as shown in Figure 8, all the blue Triangles will turn into Pluses and vice versa.
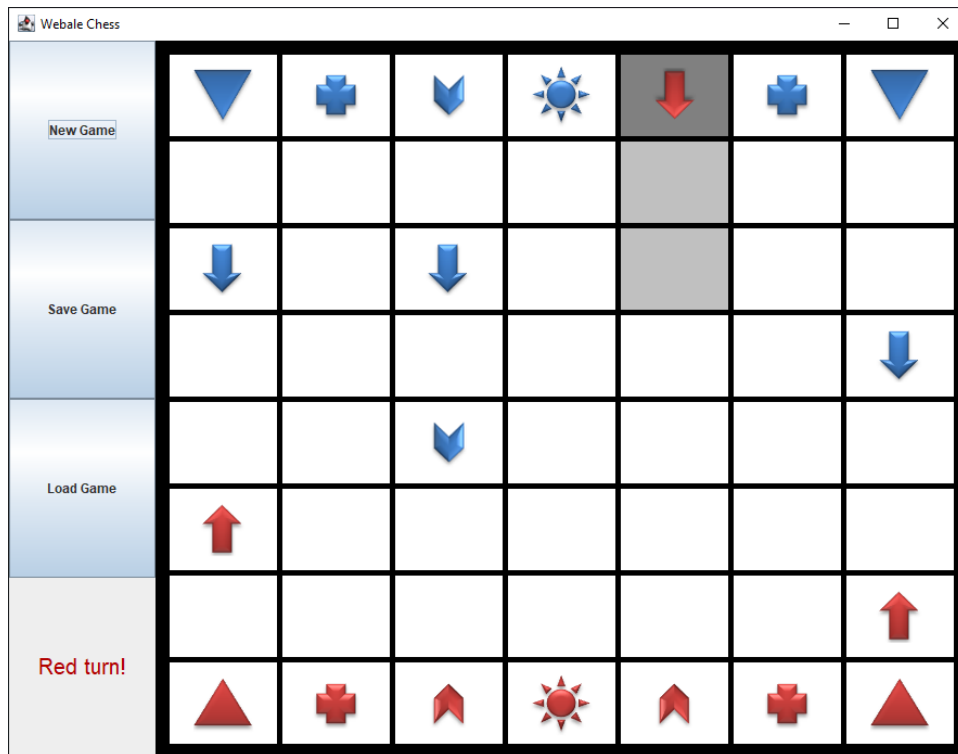
Figure 9
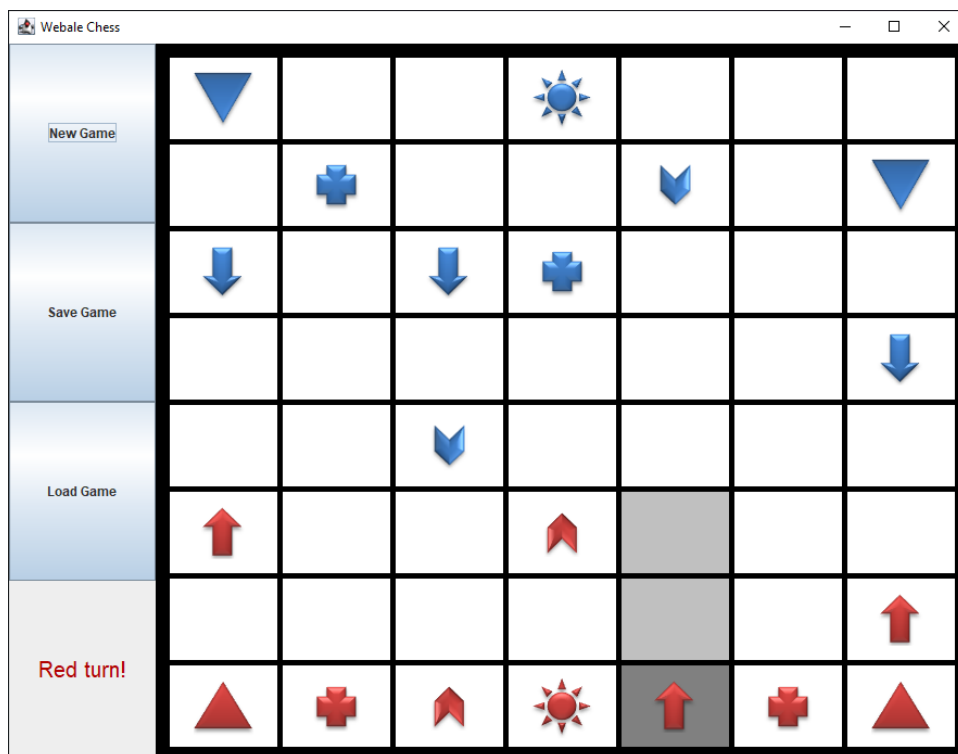
Figure 10

If Arrow reaches the other edges of the board, it turns around and heads back in the opposite direction. Examples are shown in Figure 9 and 10 above.
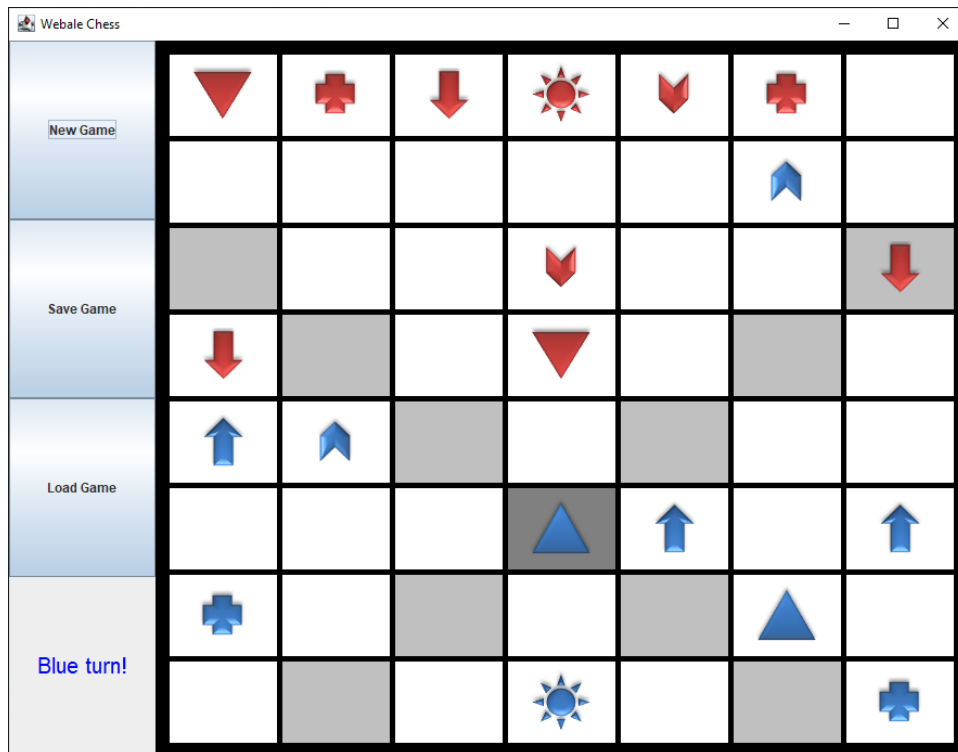
Figure 11

As shown in Figure 11, the Triangle can move any number of steps diagonally.
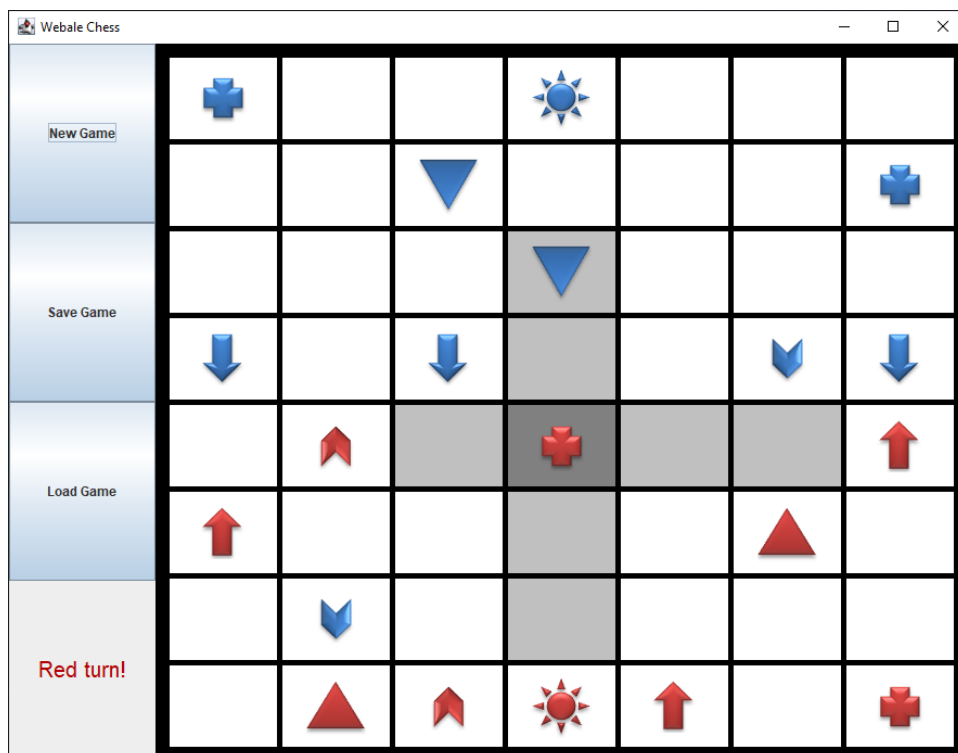
Figure 12

As shown in Figure 12, the Plus can move any number of steps up and down, or left and right.
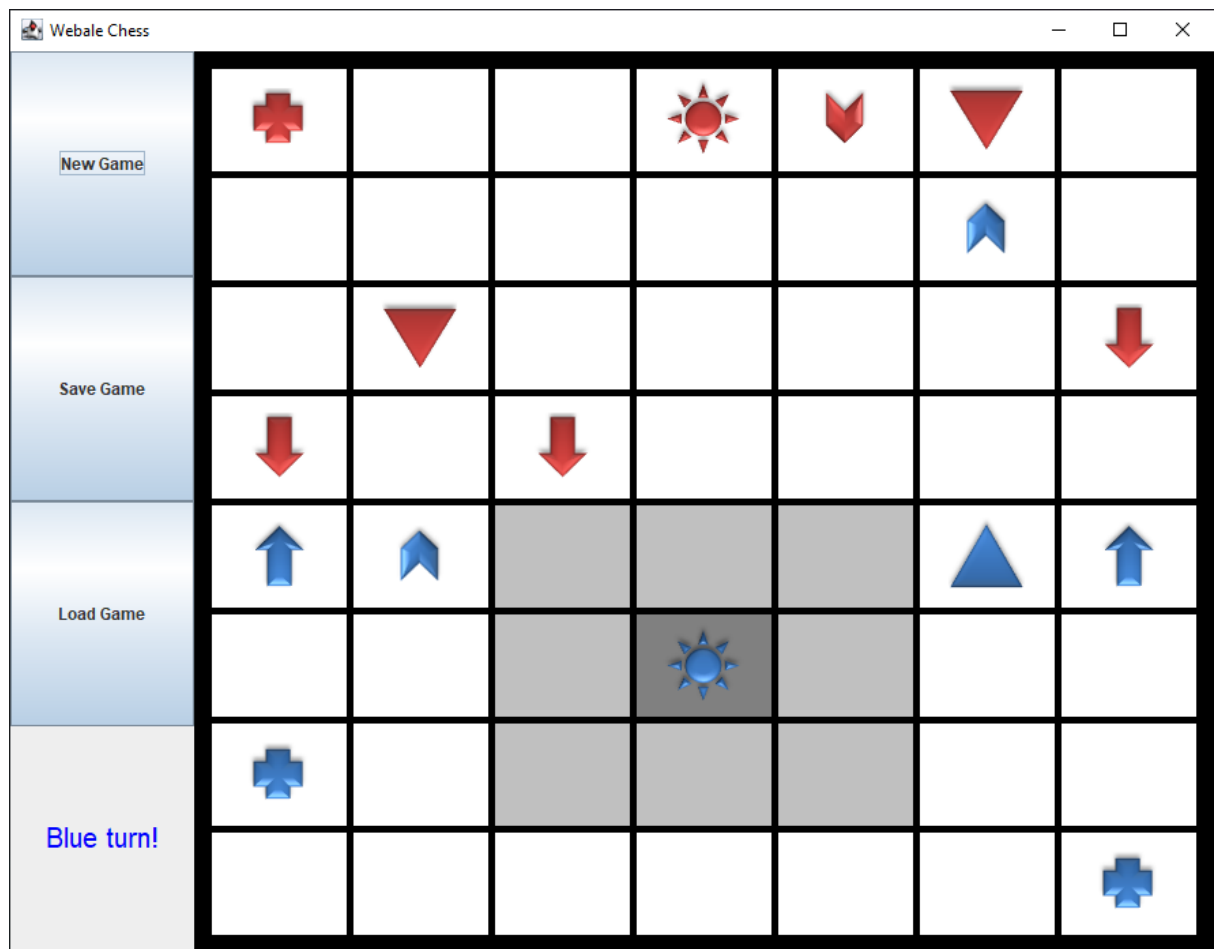
Figure 13

The Sun can only move one step in any direction.
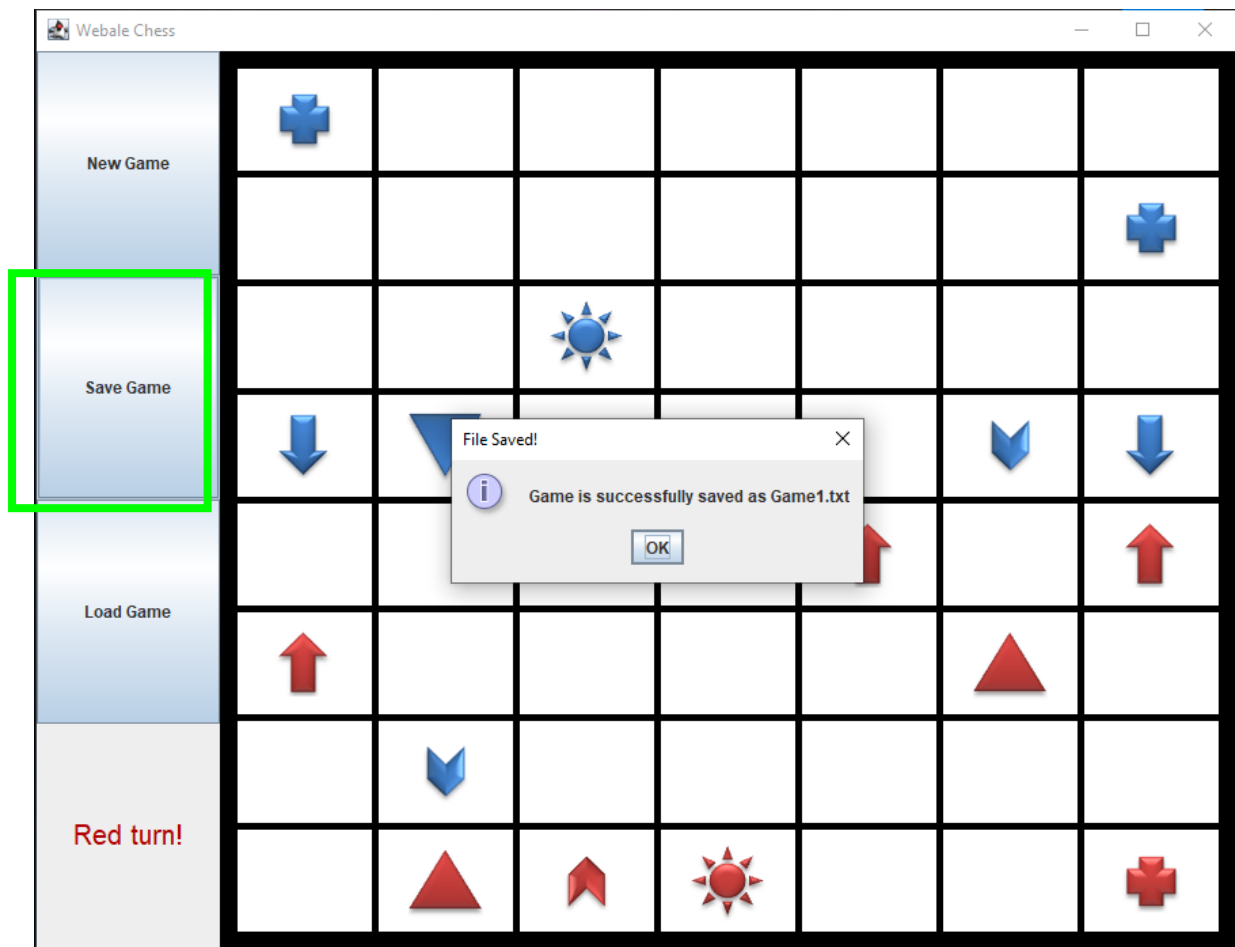
Figure 14

When <Save Game> is clicked, the current state of the chess game will be saved into a text file and stored in a file named "GameFile". After the game state is saved successfully, a pop up will be prompted.
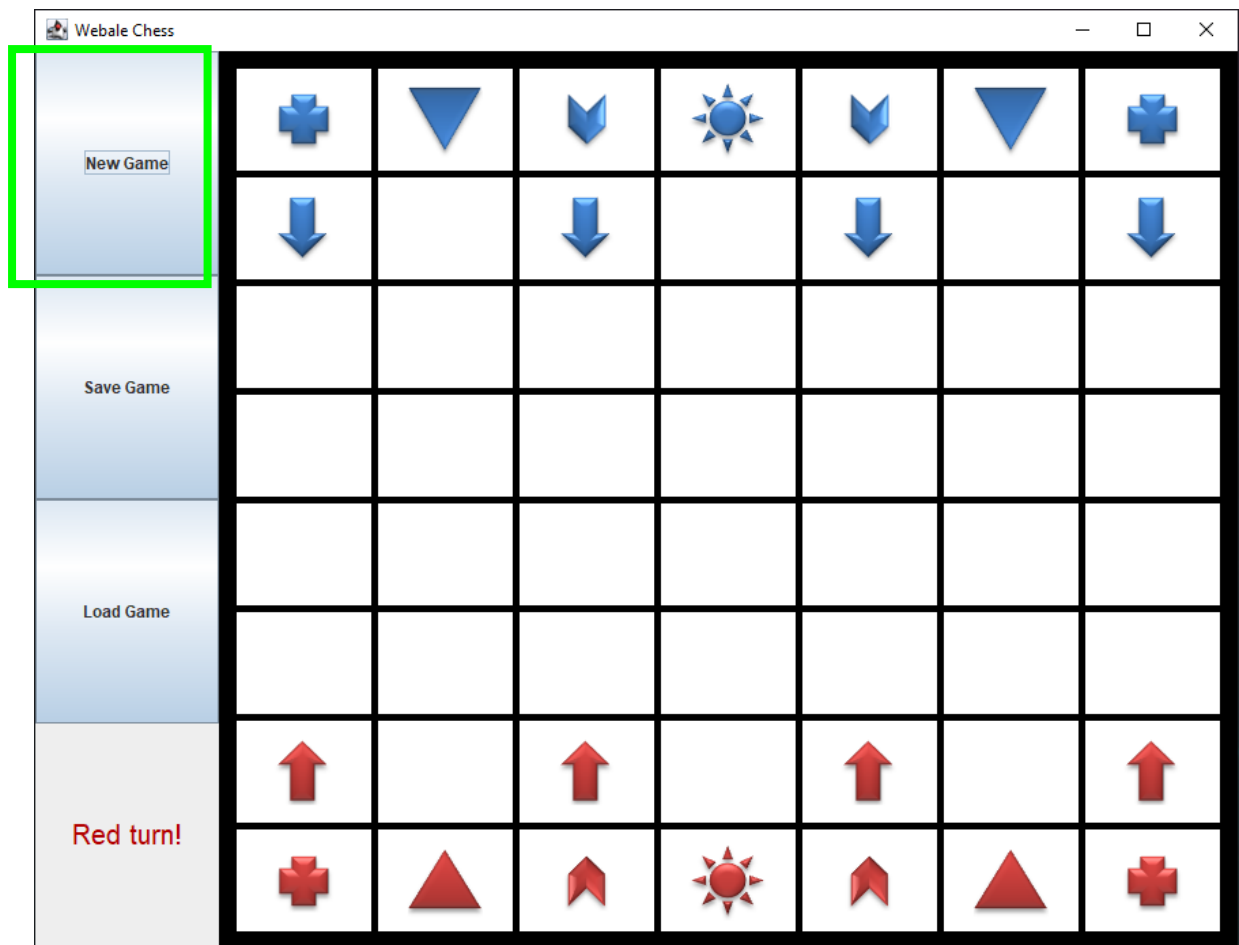
Figure 15

If the <New Game> button is clicked, the board will be initialized and the game will be started over.
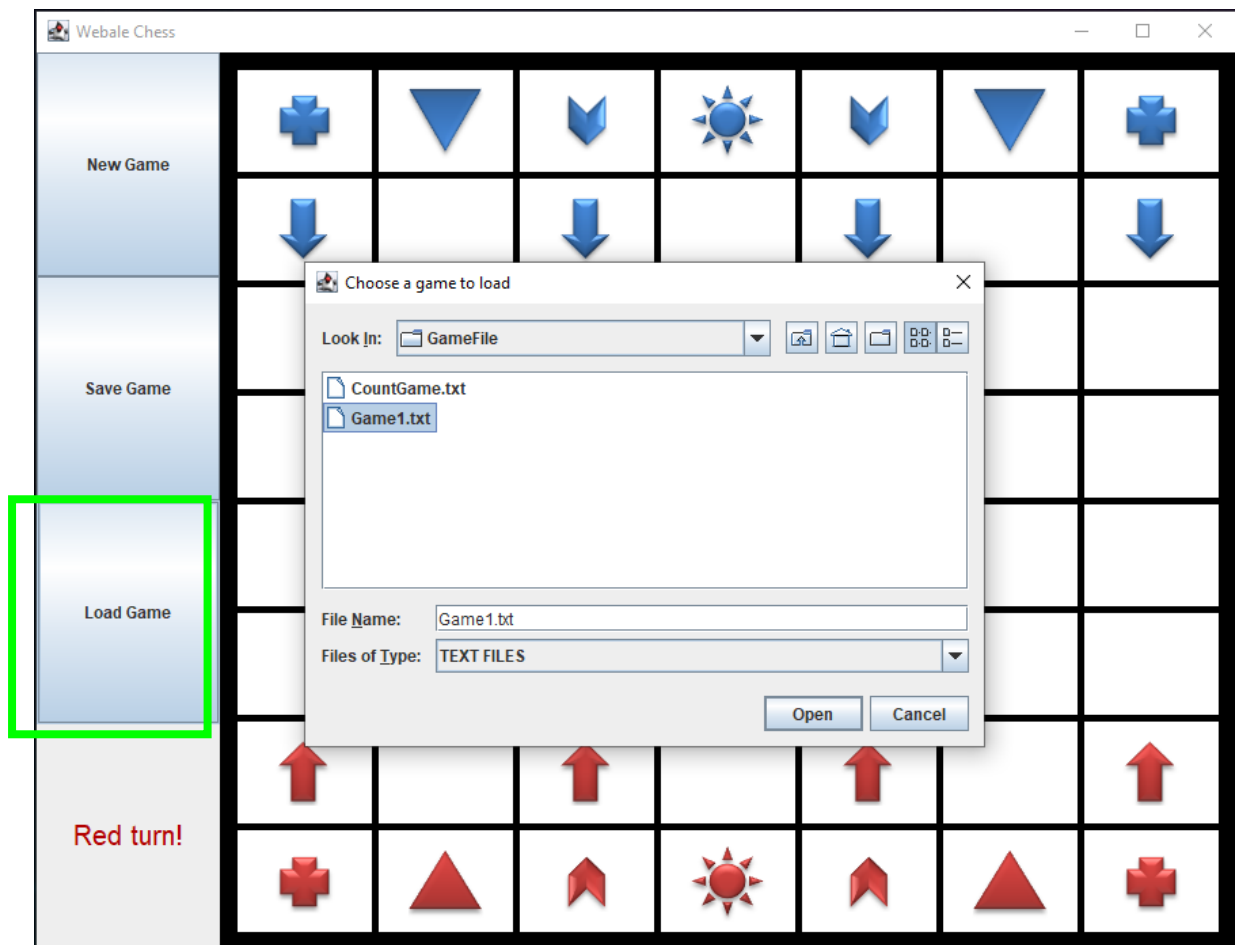
Figure 16

When the <Load Game> button is clicked, the player can choose a game to load from the file "GameFile".
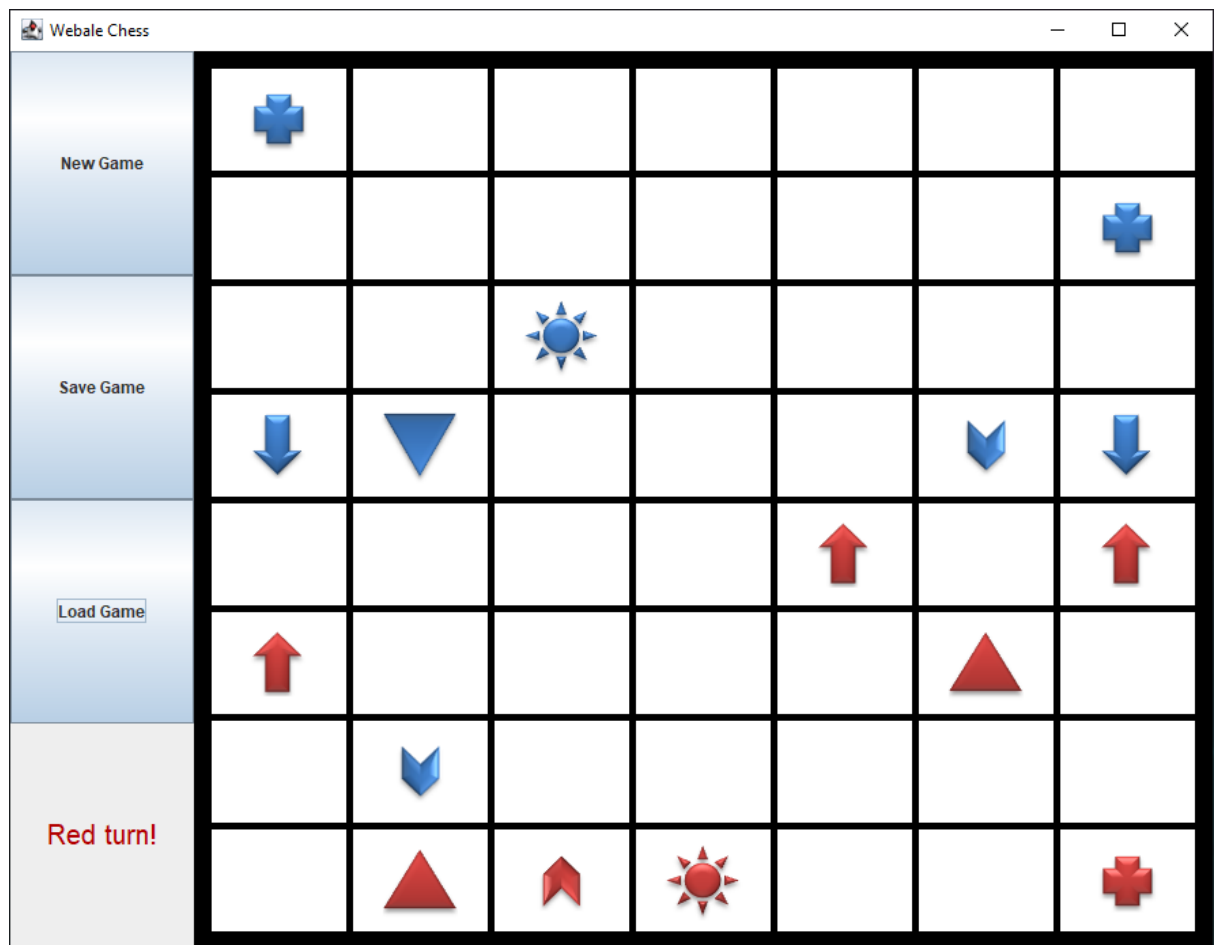
Figure 17

The game will be set to the same as how it was saved previously.
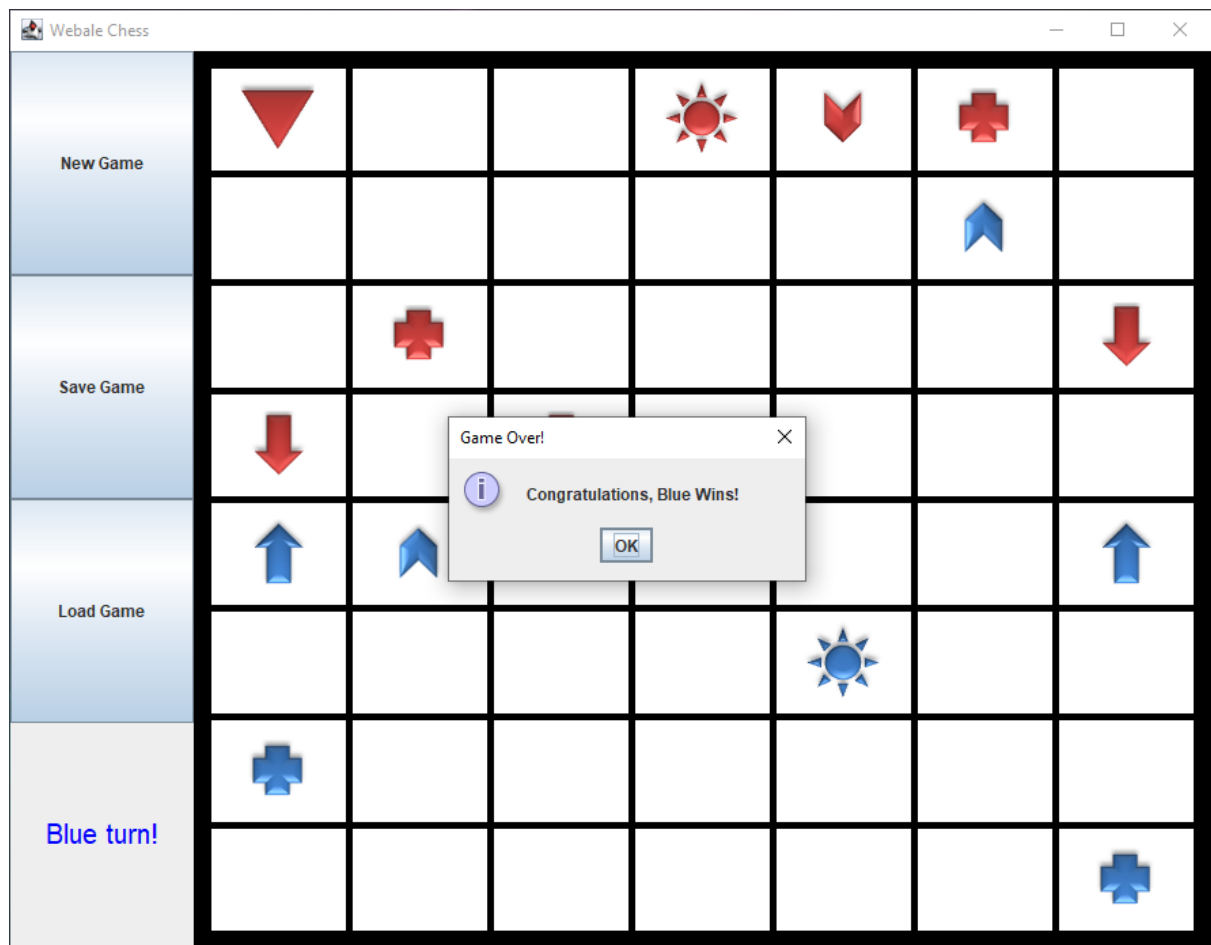
Figure 18

When a Sun is being captured, the opposite player will be declared as a winner through a pop up.
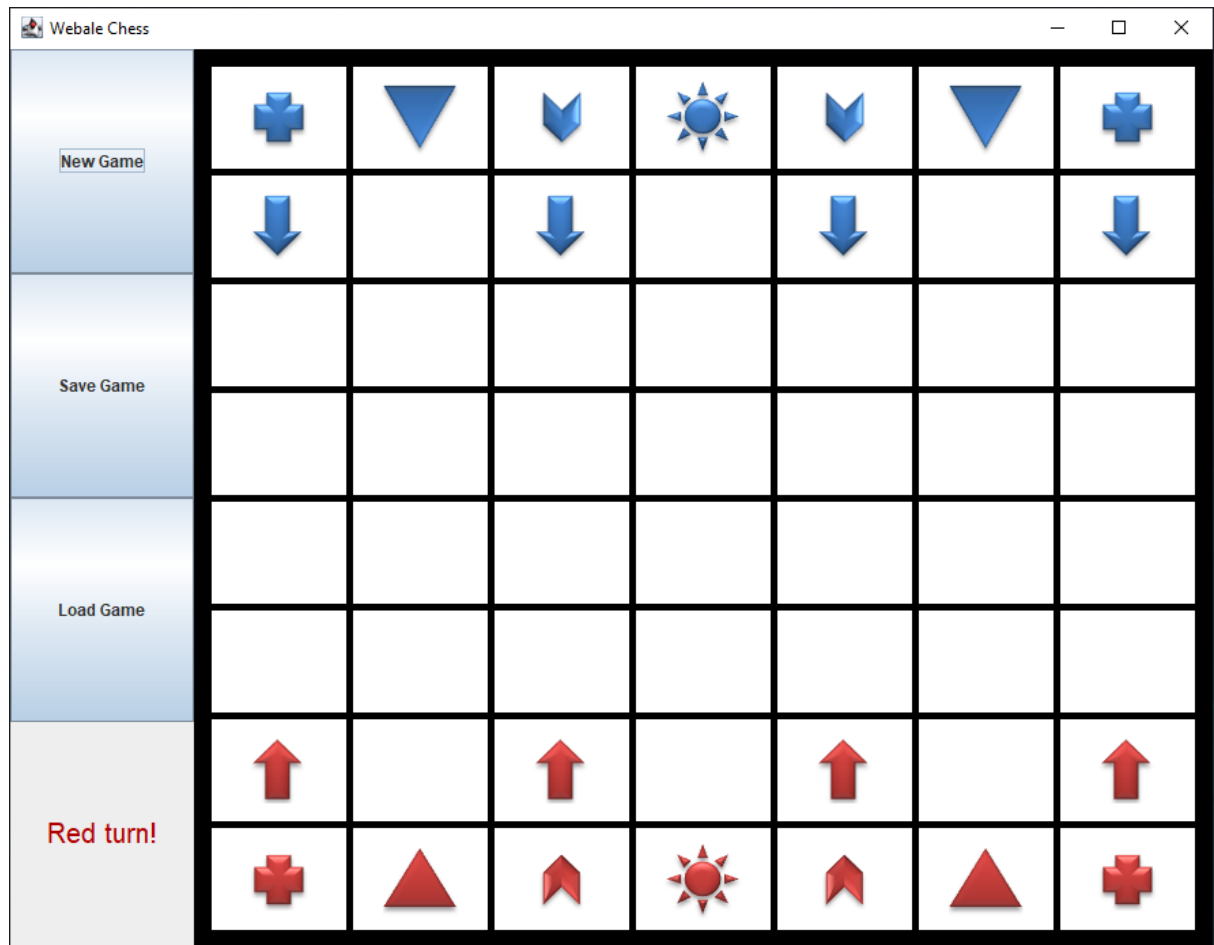
Figure 19

After the winner has been declared, the game will once again be initialized and ready for a new game.

UML Class Diagram

<<Abstract Class>> Piece — Model

- # index: int
- # chessPieceColor: String

- + Piece (index: int, chessPieceColor: String)
- + abstract getChessPieceType(): String
- + abstract getIndex(): int
- + abstract getChessPieceColor(): String
- + abstract setMoveDirection(direction: String): void
- + abstract getMoveDirection(): String

0..22

<<Java Class>> Triangle — Model.Chess

- - chessPieceType: String
- - moveDirection: String

- + Triangle(index: int, chessPieceColor: String)
- + configure(index: int, chessPieceColor: String, chessPieceType: String): void
- + getChessPieceType(): String
- + getIndex(): int
- + getChessPieceColor(): String
- + getMoveDirection(): String
- + setMoveDirection(moveDirection: String):void

<<Java Class>> Plus — Model.Chess

- - chessPieceType: String
- - moveDirection: String

- + Plus(index: int, chessPieceColor: String)
- + configure(index: int, chessPieceColor: String, chessPieceType: String): void
- + getChessPieceType(): String
- + getIndex(): int
- + getChessPieceColor(): String
- + getMoveDirection(): String
- + setMoveDirection(moveDirection: String):void

<<Java Class>> Arrow — Model.Chess

- - chessPieceType: String
- - moveDirection: String

- + Arrow(index: int, chessPieceColor: String)
- + configure(index: int, chessPieceColor: String, chessPieceType: String): void
- + getChessPieceType(): String
- + getIndex(): int
- + getChessPieceColor(): String
- + getMoveDirection(): String
- + setMoveDirection(moveDirection: String):void

<<Java Class>> Chevron — Model.Chess

- - chessPieceType: String
- - moveDirection: String

- + Chevron(index: int, chessPieceColor: String)
- + configure(index: int, chessPieceColor: String, chessPieceType: String): void
- + getChessPieceType(): String
- + getIndex(): int
- + getChessPieceColor(): String
- + getMoveDirection(): String
- + setMoveDirection(moveDirection: String):void

<<Java Class>> Sun — Model.Chess

- - chessPieceType: String
- - moveDirection: String

- + Sun(index: int, chessPieceColor: String)
- + configure(index: int, chessPieceColor: String, chessPieceType: String): void
- + getChessPieceType(): String
- + getIndex(): int
- + getChessPieceColor(): String
- + getMoveDirection(): String
- + setMoveDirection(moveDirection: String):void

<<Java Class>> JChess

- + main(String[] args): void

<<Java Class>> BoardPanel — View

- + BoardPanel()

<<Java Class>> StatePanel — View

- - newGameBtn: JButton
- - loadGameBtn: JButton
- - saveGameBtn: JButton
- - playerTurn: JLabel

- + StatePanel(controls: ChessController)
- + setChessController(controls: ChessController): void
- + initializeButton(sc: ChessController): void
- + createLabel(): void

<<Java Class>> Panelling — View

- + labels: ArrayList<JLabel>
- + cc: ChessController
- + playerTurn: JLabel

- + Panelling(cc: ChessController)
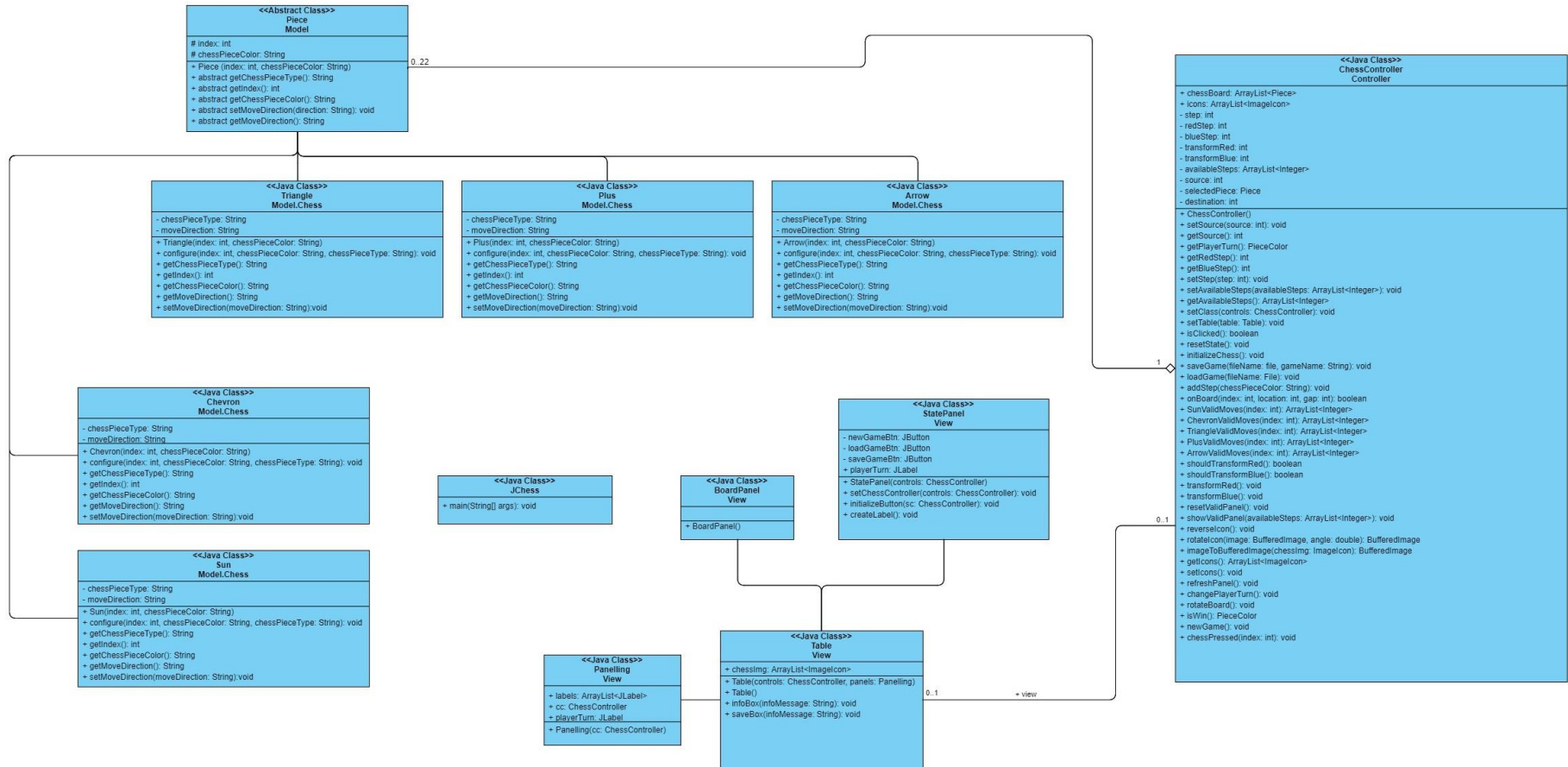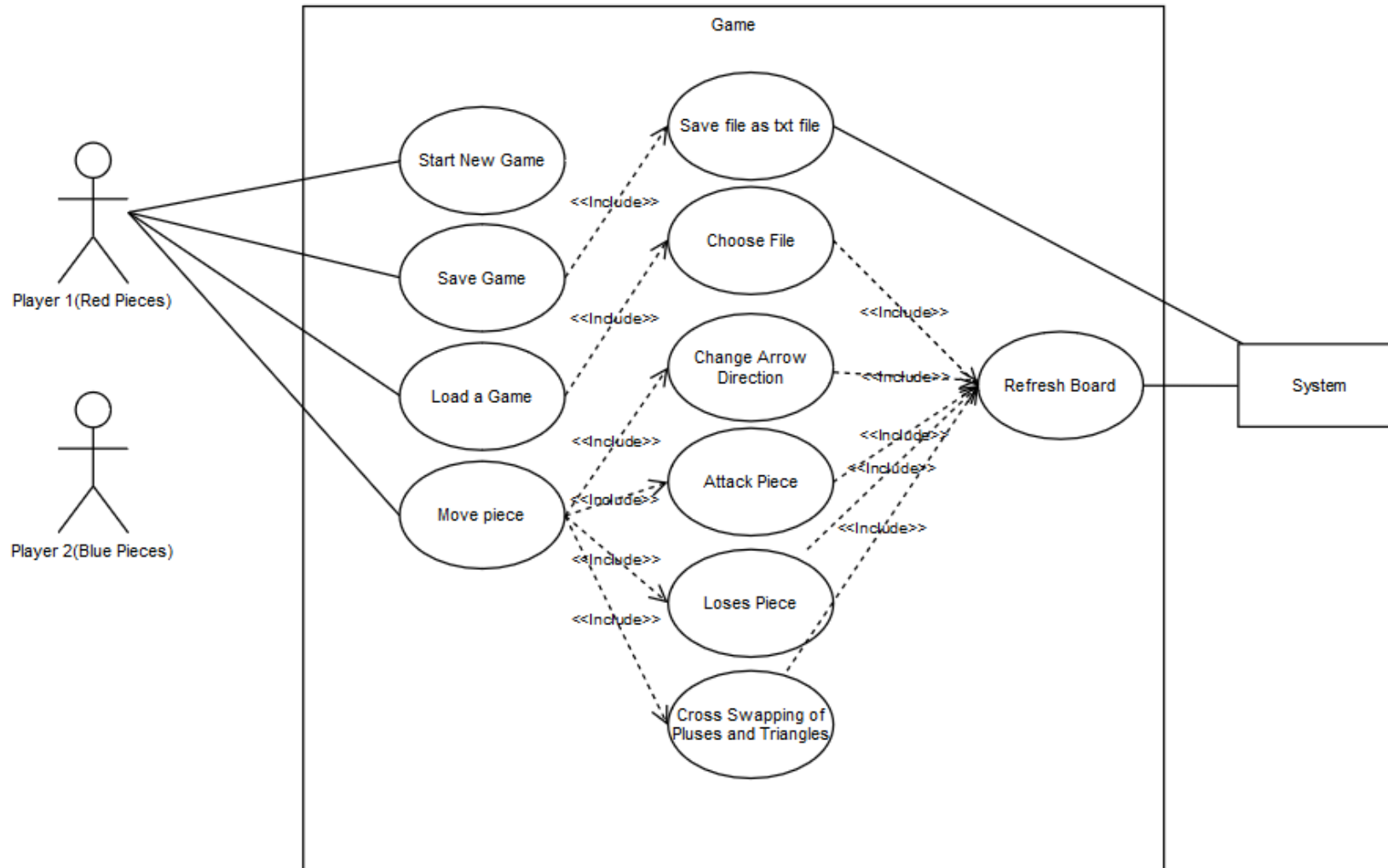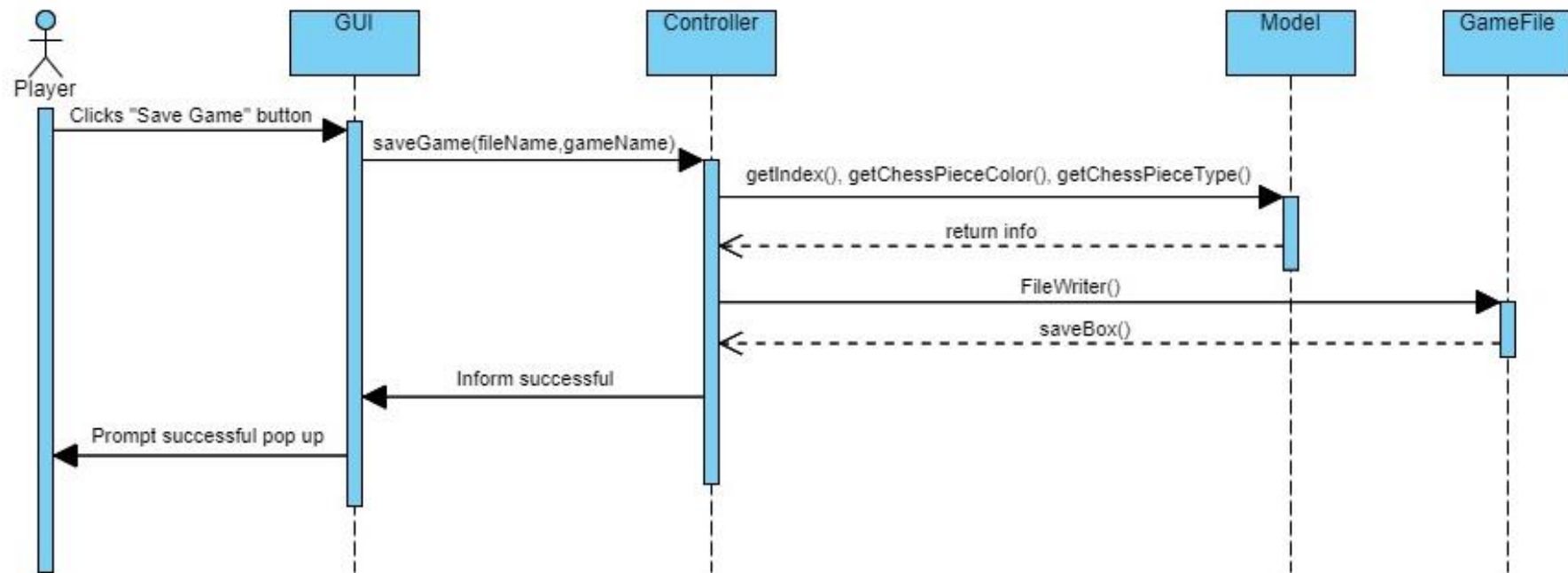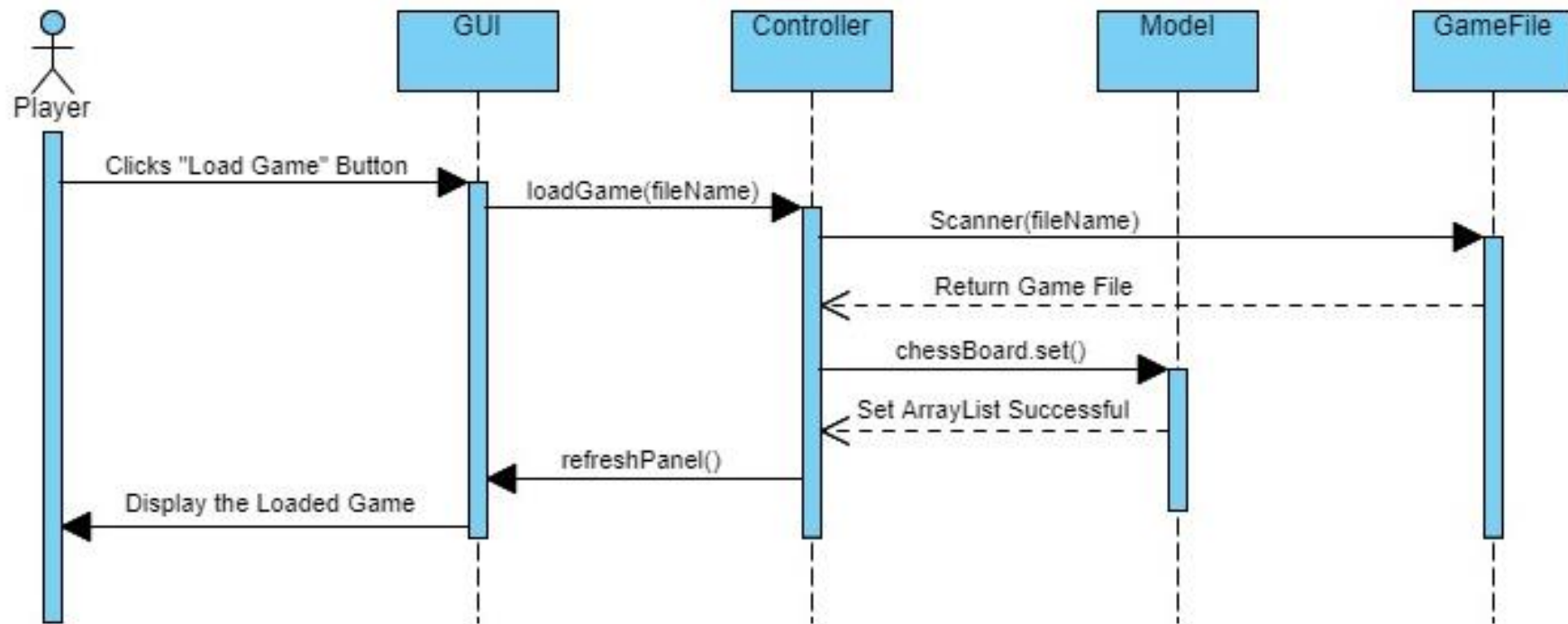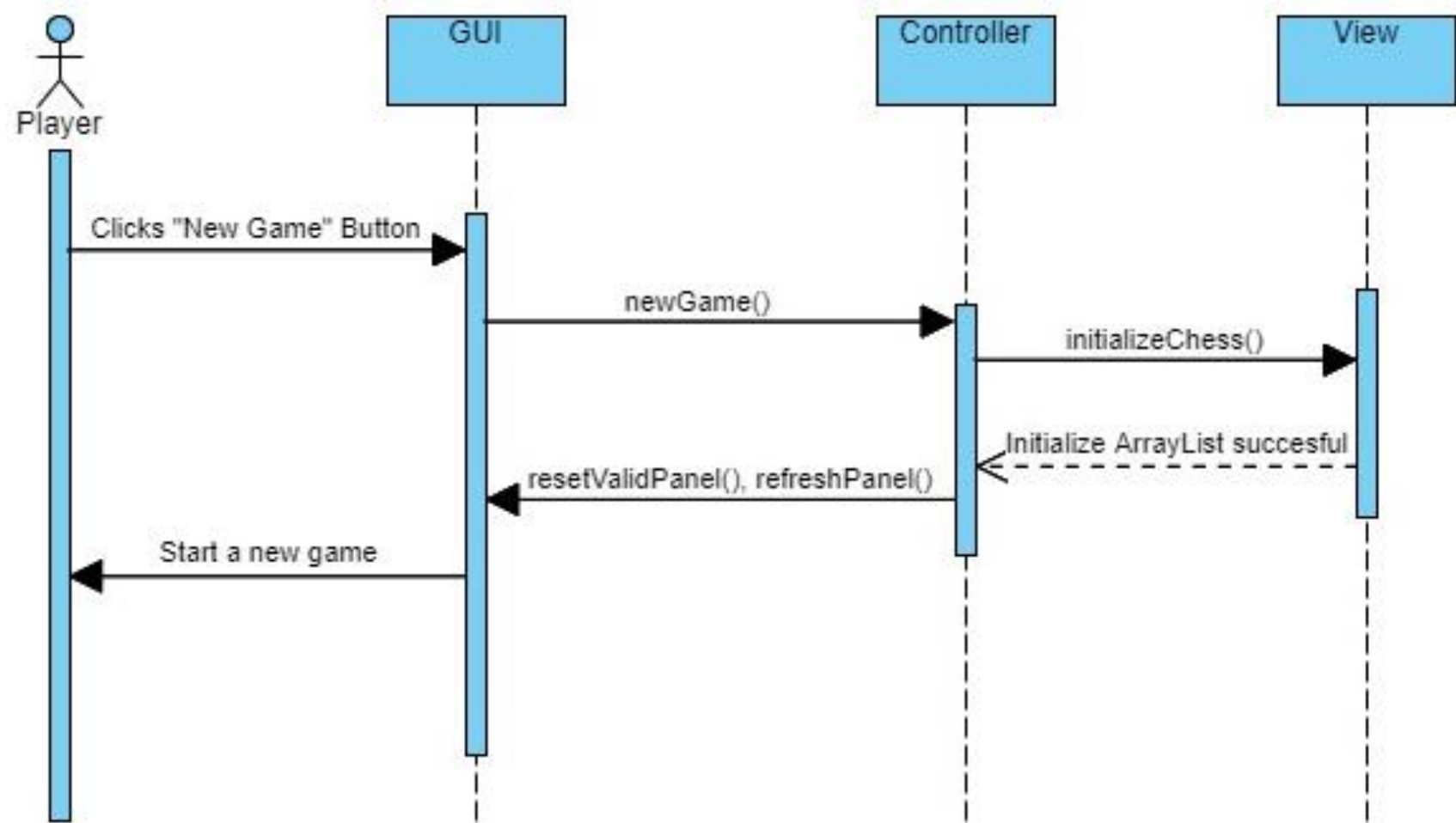
<<Java Class>> Table — View

- + chessImg: ArrayList<ImageIcon>

- + Table(controls: ChessController, panels: Panelling)
- + Table()
- + infoBox(infoMessage: String): void
- + saveBox(infoMessage: String): void

0..1   + view

<<Java Class>> ChessController — Controller

- + chessBoard: ArrayList<Piece>
- + icons: ArrayList<ImageIcon>
- - step: int
- - redStep: int
- - blueStep: int
- - transformRed: int
- - transformBlue: int
- - availableSteps: ArrayList<Integer>
- - source: int
- - selectedPiece: Piece
- - destination: int

- + ChessController()
- + setSource(source: int): void
- + getSource(): int
- + getPlayerTurn(): PieceColor
- + getRedStep(): int
- + getBlueStep(): int
- + setStep(step: int): void
- + setAvailableSteps(availableSteps: ArrayList<Integer>): void
- + getAvailableSteps(): ArrayList<Integer>
- + setCC(controls: ChessController): void
- + setTable(table: Table): void
- + isClicked(): boolean
- + resetState(): void
- + initializeChess(): void
- + saveGame(fileName: file, gameName: String): void
- + loadGame(fileName: File): void
- + addStep(chessPieceColor: String): void
- + onBoard(index: int, location: int, gap: int): boolean
- + SunValidMoves(index: int): ArrayList<Integer>
- + ChevronValidMoves(index: int): ArrayList<Integer>
- + TriangleValidMoves(index: int): ArrayList<Integer>
- + PlusValidMoves(index: int): ArrayList<Integer>
- + ArrowValidMoves(index: int): ArrayList<Integer>
- + shouldTransformRed(): boolean
- + shouldTransformBlue(): boolean
- + transformRed(): void
- + transformBlue(): void
- + resetValidPanel(): void
- + showValidPanel(availableSteps: ArrayList<Integer>): void
- + reverseIcon(): void
- + rotateIcon(image: BufferedImage, angle: double): BufferedImage
- + imageToBufferedImage(chessImg: ImageIcon): BufferedImage
- + getIcons(): ArrayList<ImageIcon>
- + setIcons(): void
- + refreshPanel(): void
- + changePlayerTurn(): void
- + rotateBoard(): void
- + isWin(): PieceColor
- + newGame(): void
- + chessPressed(index: int): void

1   0..1

Use Case Diagram

Sequence Diagram

Save Game

Load Game

New Game

Move Piece