TCP2201 Project

Trimester 2310

by Ipoh

Team Leader: Emily Phang Ru Ying, 0122270304, 1211102687@student.mmu.edu.my

Team members:

Teo Yu Jie, 0122615818, 1211102751@student.mmu.edu.my

Lim Cai Qing, 0143001623, 1211102753@student.mmu.edu.my

Toh Ee Lin, 01115455622, 1221303203@student.mmu.edu.my

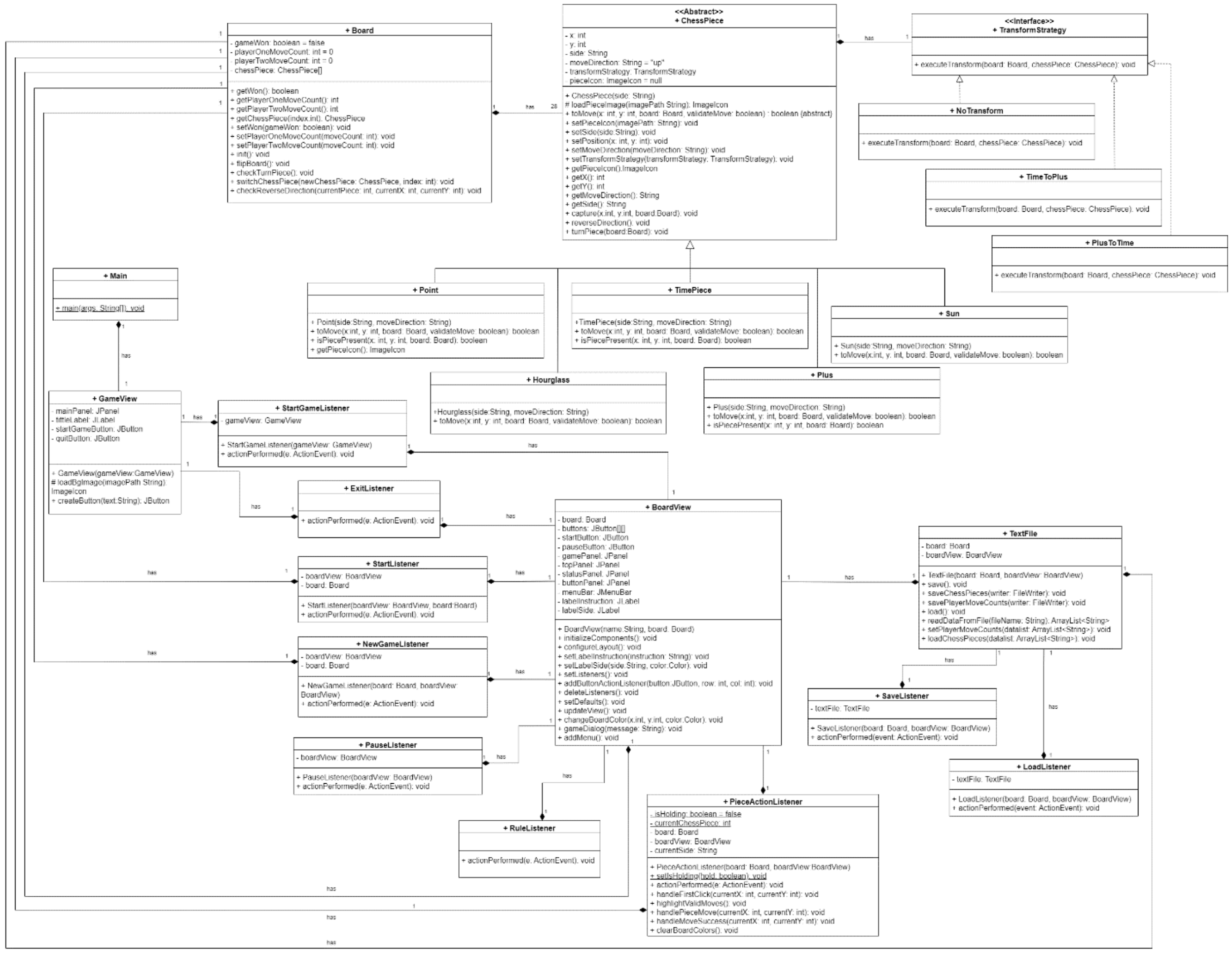
Compile and Run Instructions

If one were to run the program using the command prompt, the user has to compile the file on the command prompt by typing the following command line: "javac Main.java" as "Main.java" is the class that contains the main function to run the program. The following command line will also generate a file named "Main.class" as it is compiled into bytecode using the Java compiler. Therefore, the user just has to type "java Main" as the command line into the command prompt to execute the program.

Different IDEs use different ways to execute the program. Our team used Visual Studio Code as our IDE to write and execute the program. In Visual Studio Code, all the related files must include the package name ("package app"), which is the name of the folder containing them, on the first line of each Java file. One just has to unzip the source code file named "OOAD Talabia Chess" and open the source code with Visual Studio Code. Then, just press "Run" or "F5" on the Main java file to run the program.

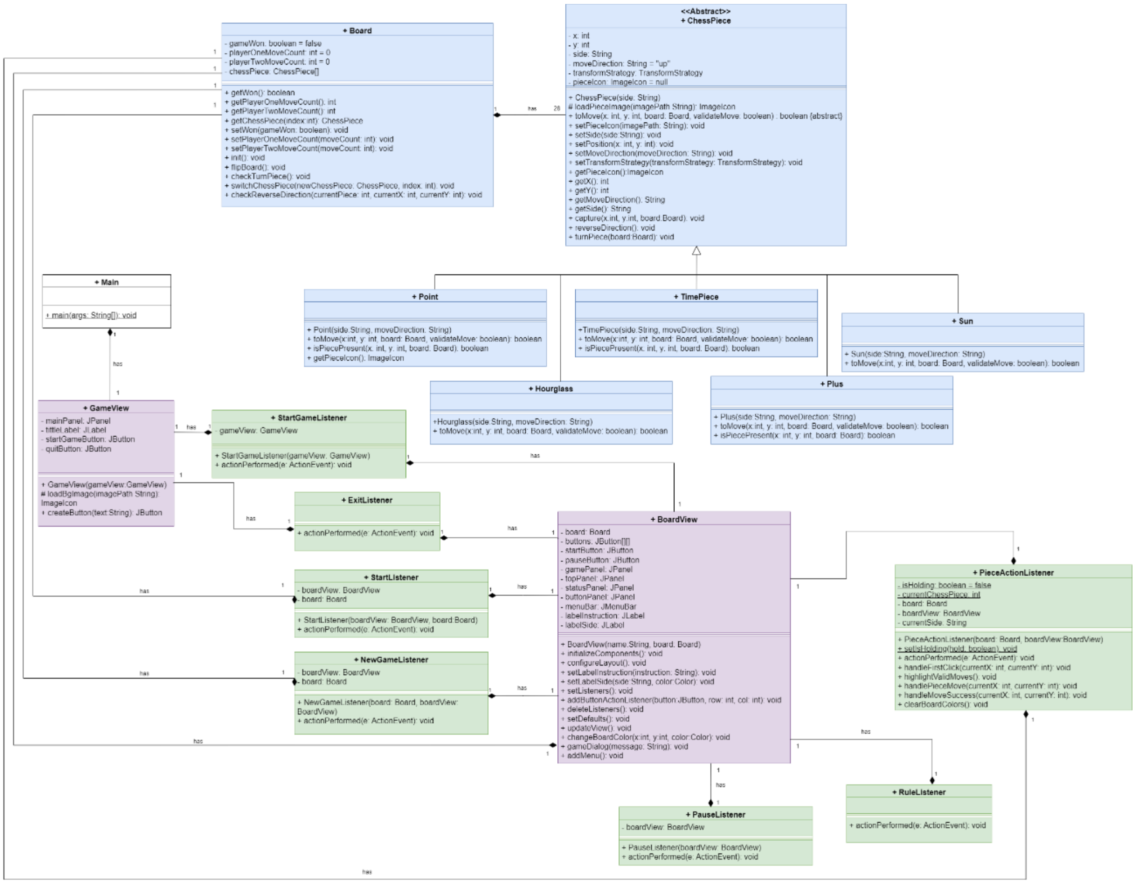
If the user does not have the same IDE as ours or uses a different version, the user just needs to create a jar file with Main.java as the main class or copy the Jar file named "OOAD Talabia Chess.jar" provided in the zip file that is located together with the source code into the user's computer that has the same or newer version of Java installed. Then, the user just has to double-click on the Jar file on the new computer. The program will execute successfully even if the other computer is using a different operating system as long as Java is installed and is Java version 17 or newer.

UML Class Diagram


+ Board
- gameWon: boolean = false
- playerOneMoveCount: int = 0
- playerTwoMoveCount: int = 0
- chessPiece: ChessPiece[]
+ getWon(): boolean
+ getPlayerOneMoveCount(): int
+ getPlayerTwoMoveCount(): int
+ getChessPiece(index.int). ChessPiece
+ setWon(gameWon: boolean): void
+ setPlayerOneMoveCount(moveCount: int): void
+ setPlayerTwoMoveCount(moveCount: int): void
+ init(): void
+ flipBoard(): void
+ checkTurnPiece(): void
+ switchChessPiece(newChessPiece: ChessPiece, index: int): void
+ checkReverseDirection(currentPiece: int, currentX: int, currentY: int): void
1
has
28
<<Abstract>>
+ ChessPiece
- x: int
- y: int
- side: String
- moveDirection: String = "up"
- transformStrategy: TransformStrategy
- pieceIcon: ImageIcon = null
+ ChessPiece(side: String)
# loadPieceImage(imagePath String): ImageIcon
+ toMove(x: int, y: int, board: Board, validateMove: boolean) : boolean (abstract)
+ setPieceIcon(imagePath: String): void
+ setSide(side:String): void
+ setPosition(x: int, y: int): void
+ setMoveDirection(moveDirection: String): void
+ setTransformStrategy(transformStrategy: TransformStrategy): void
+ getPieceIcon().ImageIcon
+ getX(): int
+ getY(): int
+ getMoveDirection(): String
+ getSide(): String
+ capture(x.int, y.int, board.Board). void
+ reverseDirection(): void
+ turnPiece(board:Board): void
<<Interface>>
+ TransformStrategy
+ executeTransform(board: Board, chessPiece: ChessPiece): void
+ NoTransform
+ executeTransform(board: Board, chessPiece: ChessPiece): void
+ TimeToPlus
+ executeTransform(board: Board, chessPiece: ChessPiece): void
+ PlusToTime
+ executeTransform(board: Board, chessPiece: ChessPiece): void
+ Main
+ main(args: String[]): void
+ Point
+ Point(side:String, moveDirection: String)
+ toMove(x:int, y: int, board: Board, validateMove: boolean): boolean
+ isPiecePresent(x: int, y: int, board: Board): boolean
+ getPieceIcon(): ImageIcon
+ TimePiece
+TimePiece(side:String, moveDirection: String)
+ toMove(x:int, y: int, board: Board, validateMove: boolean): boolean
+ isPiecePresent(x: int, y: int, board: Board): boolean
+ Sun
+ Sun(side:String, moveDirection: String)
+ toMove(x:int, y: int, board: Board, validateMove: boolean): boolean
+ Hourglass
+Hourglass(side:String, moveDirection: String)
+ toMove(x:int, y: int, board: Board, validateMove: boolean): boolean
+ Plus
+ Plus(side:String, moveDirection: String)
+ toMove(x:int, y: int, board: Board, validateMove: boolean): boolean
+ isPiecePresent(x: int, y: int, board: Board): boolean
+ GameView
- mainPanel: JPanel
- titleLabel: JLabel
- startGameButton: JButton
- quitButton: JButton
+ GameView(gameView:GameView)
# loadBgImage(imagePath String): ImageIcon
+ createButton(text:String): JButton
+ StartGameListener
- gameView: GameView
+ StartGameListener(gameView: GameView)
+ actionPerformed(e. ActionEvent). void
+ ExitListener
+ actionPerformed(e: ActionEvent): void
+ BoardView
- board: Board
- buttons: JButton[][]
- startButton: JButton
- pauseButton: JButton
- gamePanel: JPanel
- topPanel: JPanel
- statusPanel: JPanel
- buttonPanel: JPanel
- menuBar: JMenuBar
- labelInstruction: JLabel
- labelSide: JLabel
+ BoardView(name.String, board. Board)
+ initializeComponents(): void
+ configureLayout(): void
+ setLabelInstruction(instruction: String): void
+ setLabelSide(side.String, color.Color). void
+ setListeners(): void
+ addButtonActionListener(button:JButton, row: int, col: int): void
+ deleteListeners(): void
+ setDefaults(): void
+ updateView(): void
+ changeBoardColor(x.int, y.int, color.Color). void
+ gameDialog(message: String): void
+ addMenu(): void
+ TextFile
- board: Board
- boardView: BoardView
+ TextFile(board: Board, boardView: BoardView)
+ save(): void
+ saveChessPieces(writer: FileWriter): void
+ savePlayerMoveCounts(writer: FileWriter): void
+ load(): void
+ readDataFromFile(fileName: String): ArrayList<String>
+ setPlayerMoveCounts(datalist: ArrayList<String>): void
+ loadChessPieces(datalist: ArrayList<String>): void
+ StartListener
- boardView: BoardView
- board: Board
+ StartListener(boardView: BoardView, board:Board)
+ actionPerformed(e. ActionEvent). void
+ NewGameListener
- boardView: BoardView
- board. Board
+ NewGameListener(board: Board, boardView: BoardView)
+ actionPerformed(e: ActionEvent): void
+ SaveListener
- textFile: TextFile
+ SaveListener(board: Board, boardView: BoardView)
+ actionPerformed(event: ActionEvent): void
+ PauseListener
- boardView: BoardView
+ PauseListener(boardView: BoardView)
+ actionPerformed(e: ActionEvent): void
+ LoadListener
- textFile: TextFile
+ LoadListener(board: Board, boardView: BoardView)
+ actionPerformed(event: ActionEvent): void
+ RuleListener
+ actionPerformed(e: ActionEvent). void
+ PieceActionListener
- isHolding: boolean = false
- currentChessPiece: int
- board: Board
- boardView: BoardView
- currentSide: String
+ PieceActionListener(board: Board, boardView:BoardView)
+ setIsHolding(hold: boolean): void
+ actionPerformed(e: ActionEvent): void
+ handleFirstClick(currentX: int, currentY: int): void
+ highlightValidMoves(): void
+ handlePieceMove(currentX: int, currentY: int): void
+ handleMoveSuccess(currentX: int, currentY: int): void
+ clearBoardColors(): void

This is the UML Class Diagram for our Talabia Chess Game involving 24 classes. We have implemented Model-View-Controller (MVC) and Strategy Pattern as our Design Pattern.

Model-View-Controller (MVC) Pattern


+ Board
- gameWon: boolean = false
- playerOneMoveCount: int = 0
- playerTwoMoveCount: int = 0
- chessPiece: ChessPiece[]
+ getWon(): boolean
+ getPlayerOneMoveCount(): int
+ getPlayerTwoMoveCount(): int
+ getChessPiece(index:int): ChessPiece
+ setWon(gameWon: boolean): void
+ setPlayerOneMoveCount(moveCount: int): void
+ setPlayerTwoMoveCount(moveCount: int): void
+ init(): void
+ flipBoard(): void
+ checkTurnPiece(): void
+ switchChessPiece(newChessPiece: ChessPiece, index: int): void
+ checkReverseDirection(currentPiece: int, currentX: int, currentY: int): void
1
has
28
<<Abstract>>
+ ChessPiece
- x: int
- y: int
- side: String
- moveDirection: String = "up"
- transformStrategy: TransformStrategy
- pieceIcon: ImageIcon = null
+ ChessPiece(side: String)
# loadPieceImage(imagePath String): ImageIcon
+ toMove(x: int, y: int, board: Board, validateMove: boolean) : boolean {abstract}
+ setPieceIcon(imagePath: String): void
+ setSide(side:String): void
+ setPosition(x: int, y: int): void
+ setMoveDirection(moveDirection: String): void
+ setTransformStrategy(transformStrategy: TransformStrategy): void
+ getPieceIcon():ImageIcon
+ getX(): int
+ getY(): int
+ getMoveDirection(): String
+ getSide(): String
+ capture(x.int, y.int, board.Board): void
+ reverseDirection(): void
+ turnPiece(board:Board): void
+ Main
+ main(args: String[]): void
+ Point
+ Point(side.String, moveDirection: String)
+ toMove(x:int, y: int, board: Board, validateMove: boolean): boolean
+ isPiecePresent(x. int, y. int, board. Board). boolean
+ getPieceIcon(): ImageIcon
+ TimePiece
+TimePiece(side.String, moveDirection: String)
+ toMove(x:int, y: int, board: Board, validateMove: boolean): boolean
+ isPiecePresent(x. int, y. int, board. Board). boolean
+ Sun
+ Sun(side:String, moveDirection: String)
+ toMove(x:int, y: int, board: Board, validateMove: boolean): boolean
+ Hourglass
+Hourglass(side:String, moveDirection: String)
+ toMove(x:int, y: int, board: Board, validateMove: boolean): boolean
+ Plus
+ Plus(side:String, moveDirection: String)
+ toMove(x:int, y: int, board: Board, validateMove. boolean): boolean
+ isPiecePresent(x: int, y: int, board: Board): boolean
+ GameView
- mainPanel: JPanel
- titleLabel: JLabel
- startGameButton: JButton
- quitButton: JButton
+ GameView(gameView:GameView)
# loadBgImage(imagePath String): ImageIcon
+ createButton(text:String): JButton
+ StartGameListener
- gameView: GameView
+ StartGameListener(gameView: GameView)
+ actionPerformed(e: ActionEvent): void
+ ExitListener
+ actionPerformed(e: ActionEvent): void
+ StartListener
- boardView: BoardView
- board: Board
+ StartListener(boardView: BoardView, board:Board)
+ actionPerformed(e: ActionEvent): void
+ NewGameListener
- boardView: BoardView
- board: Board
+ NewGameListener(board: Board, boardView: BoardView)
+ actionPerformed(e: ActionEvent): void
+ BoardView
- board: Board
- buttons: JButton[][]
- startButton: JButton
- pauseButton: JButton
- gamePanel: JPanel
- topPanel: JPanel
- statusPanel: JPanel
- buttonPanel: JPanel
- menuBar: JMenuBar
- labelInstruction: JLabel
- labelSide: JLabel
+ BoardView(name:String, board: Board)
+ initializeComponents(): void
+ configureLayout(): void
+ setLabelInstruction(instruction: String): void
+ setLabelSide(side:String, color:Color): void
+ setListeners(): void
+ addButtonActionListener(button:JButton, row: int, col: int): void
+ deleteListeners(): void
+ setDefaults(): void
+ updateView(): void
+ changeBoardColor(x:int, y:int, color:Color): void
+ gameDialog(message: String): void
+ addMenu(): void
+ PieceActionListener
- isHolding: boolean = false
- currentChessPiece: int
- board: Board
- boardView: BoardView
- currentSide: String
+ PieceActionListener(board: Board, boardView:BoardView)
+ setIsHolding(hold: boolean): void
+ actionPerformed(e: ActionEvent): void
+ handleFirstClick(currentX: int, currentY: int): void
+ highlightValidMoves(): void
+ handlePieceMove(currentX: int, currentY: int): void
+ handleMoveSuccess(currentX: int, currentY: int): void
+ clearBoardColors(): void
+ PauseListener
- boardView: BoardView
+ PauseListener(boardView: BoardView)
+ actionPerformed(e: ActionEvent): void
+ RuleListener
+ actionPerformed(e: ActionEvent): void
has

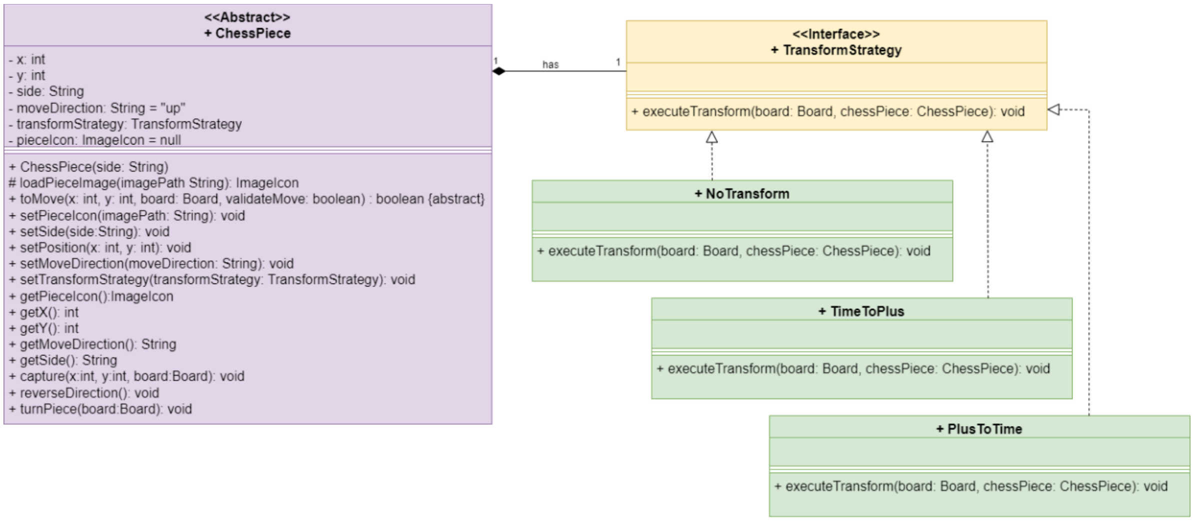
For the implementation of the Talabia Chess game, the Model-View-Controller (MVC) pattern was chosen. This makes the program easier to manage and reuse in the future. It contains three components which are Model, View and Controller.

The Model is the core functional part of the program. In Talabia Chess, classes such as ‘Board’, ‘ChessPiece’, and its subclasses (‘Point’, ‘Hourglass’, ‘Plus’, ‘TimePiece’, ‘Sun’) represent the Model. They encapsulate the game's data and the logic of how a chess game is played. The ChessPiece class and its subclasses are responsible for defining the properties and the allowable moves for each piece. The Board class holds the state of the chess game, including the positions of all pieces, the count of moves, and whether the game has been won.

The View is what the user sees and interacts with. It displays the game state and responds to user input. In the Talabia Chess Game, ‘GameView’ and ‘BoardView’ serve as the View. ‘GameView’ acts as the starting point of the game, presenting the main menu, while ‘BoardView’ renders the chessboard and pieces on the screen. These classes update the visual presentation of the game based on changes to the Model.

The Controller handles input and updates the Model. Controller classes in Talabia Chess include ‘ExitListener’, ‘NewGameListener’, ‘PauseListener’, ‘PieceActionListener’, ‘RuleListener’, ‘StartGameListener’, and ‘StartListener’. These controllers interpret user input such as mouse clicks and menu selections and translate them into actions on the Model.

Strategy Pattern

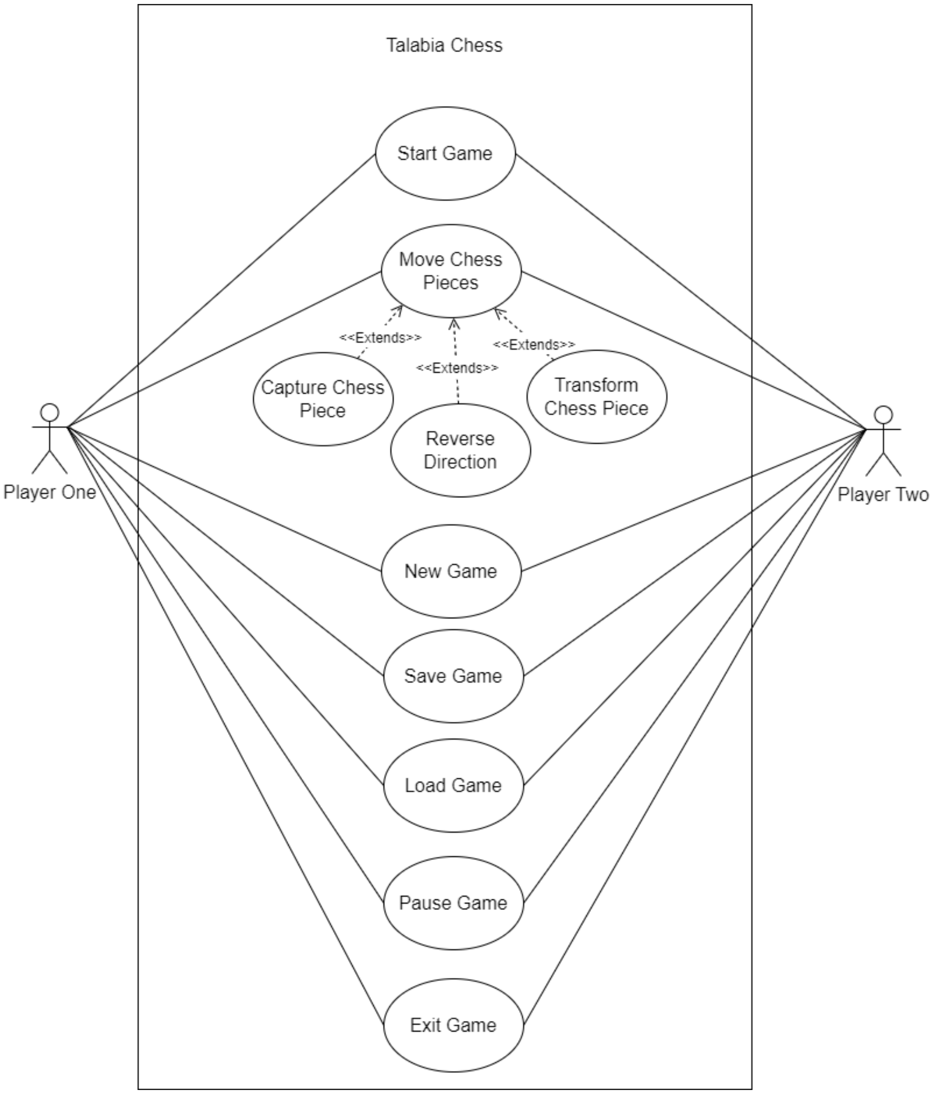

<<Abstract>>
+ ChessPiece
- x: int
- y: int
- side: String
- moveDirection: String = "up"
- transformStrategy: TransformStrategy
- pieceIcon: ImageIcon = null
+ ChessPiece(side: String)
# loadPieceImage(imagePath String): ImageIcon
+ toMove(x: int, y: int, board: Board, validateMove: boolean) : boolean {abstract}
+ setPieceIcon(imagePath: String): void
+ setSide(side:String): void
+ setPosition(x: int, y: int): void
+ setMoveDirection(moveDirection: String): void
+ setTransformStrategy(transformStrategy: TransformStrategy): void
+ getPieceIcon():ImageIcon
+ getX(): int
+ getY(): int
+ getMoveDirection(): String
+ getSide(): String
+ capture(x:int, y:int, board:Board): void
+ reverseDirection(): void
+ turnPiece(board:Board): void
1
has
1
<<Interface>>
+ TransformStrategy
+ executeTransform(board: Board, chessPiece: ChessPiece): void
+ NoTransform
+ executeTransform(board: Board, chessPiece: ChessPiece): void
+ TimeToPlus
+ executeTransform(board: Board, chessPiece: ChessPiece): void
+ PlusToTime
+ executeTransform(board: Board, chessPiece: ChessPiece): void

In the Talabia Chess Game, there is a requirement where after 2 turns (counting one yellow move and one blue move as one turn), all Time pieces will turn into Plus pieces, and all Plus pieces will turn into Time pieces. We choose to implement this feature using the Strategy Pattern. The transformation of chess pieces after a certain number of turns is a perfect scenario for this pattern as new transformation behaviors can be easily added without changing the existing code structure. Besides, the pattern allows for a dynamic change of the transformation behavior at runtime. The classes that are involved in the Strategy Pattern are ‘ChessPiece’, ‘TransformStrategy’ interface, ‘NoTransform’, ‘TimeToPlus’ and ‘PlusToTime’.

The ’ChessPiece’ acts as the context in the pattern. It maintains a reference to one of the concrete strategies and communicates with this strategy only via the strategy interface. It delegates the work of executing transformation behavior to the current ‘TransformStrategy’. The ‘TransformStrategy’ is the interface for all supported algorithms. In this case, it declares the executeTransform method that must be implemented by all concrete strategies. ‘NoTransform’, ‘TimeToPlus’ and ‘PlusToTime’ are concrete strategies, they implements the ‘TransformStrategy’ interface in different ways to provide multiple algorithms for transformation. ‘NoTransform’ is a do-nothing strategy, useful for pieces that don't transform.‘TimeToPlus’ and ‘PlusToTime’ encapsulate the specific algorithms for transforming Time pieces to Plus pieces and vice versa.

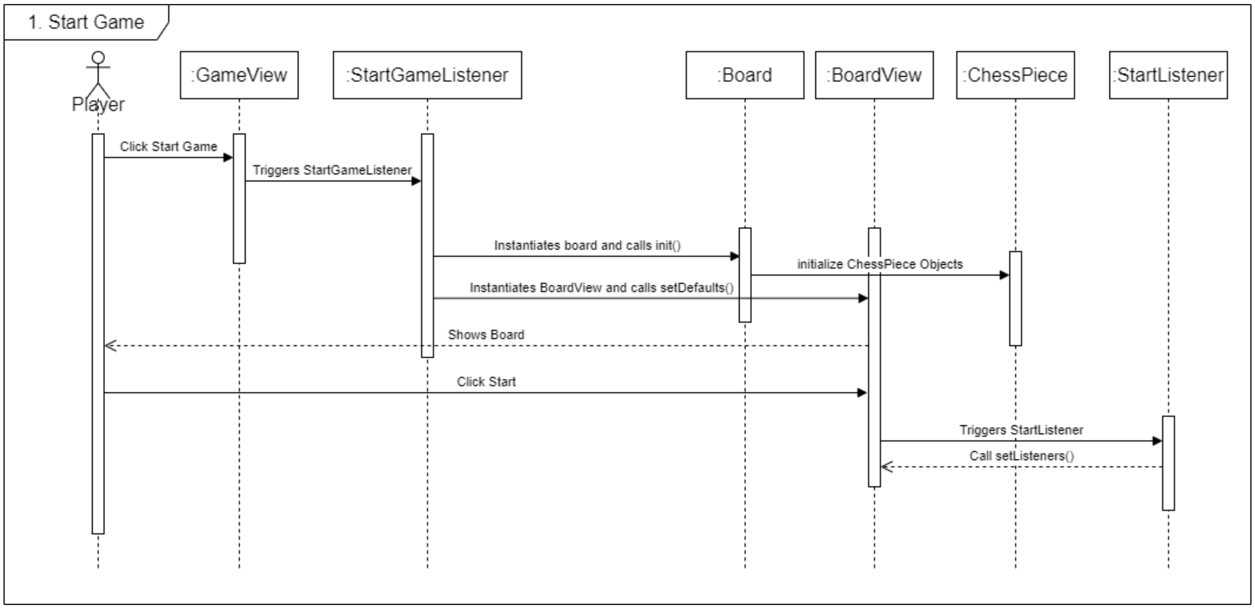
Each ChessPiece starts with a default NoTransform strategy, indicating no transformation. After two turns, the game's logic changes the strategies of the relevant chess pieces to either TimeToPlus or PlusToTime. The ChessPiece then invokes the turnPiece method, which in turn calls executeTransform on the current strategy object to perform the actual transformation.

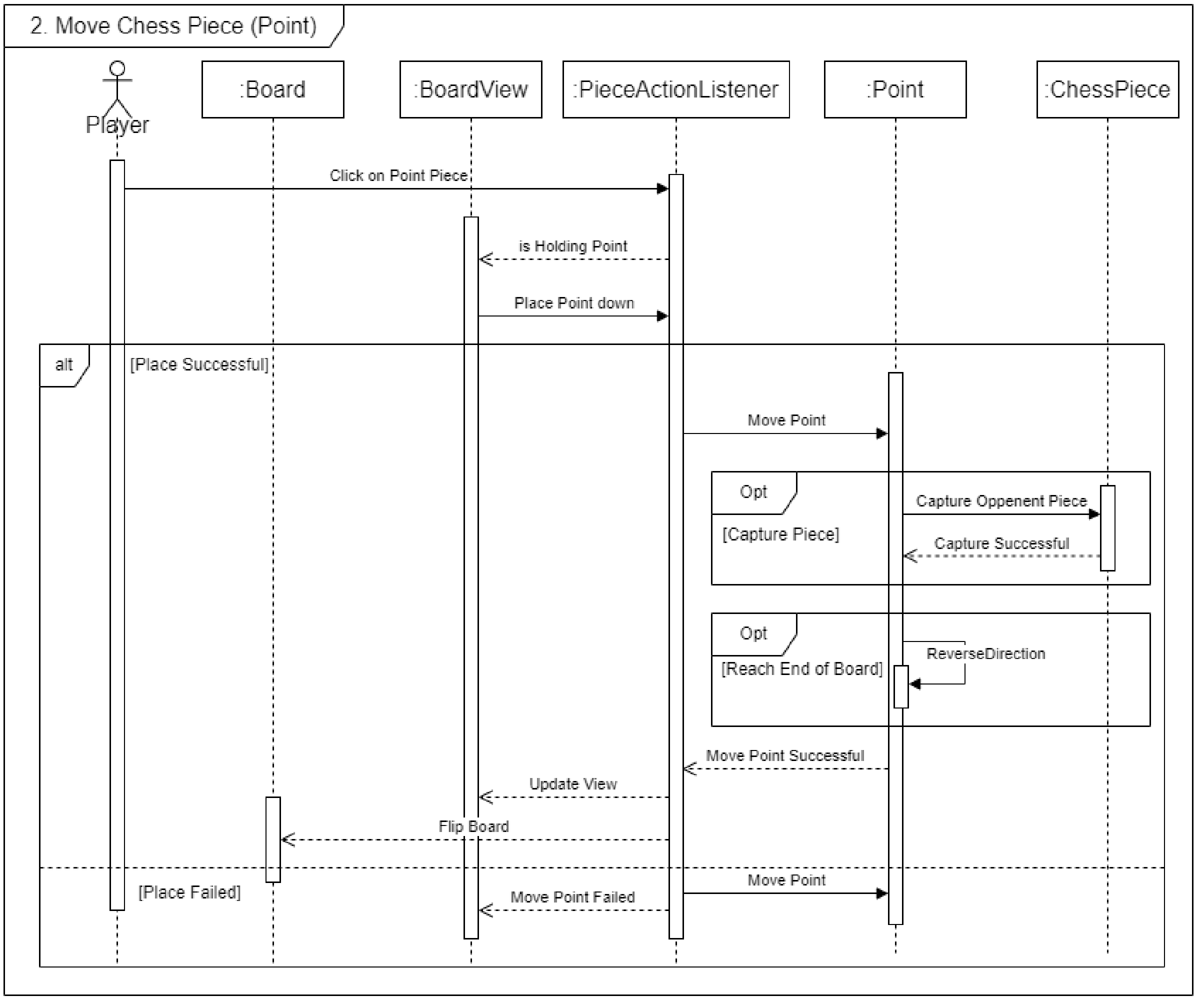
Use Case Diagram

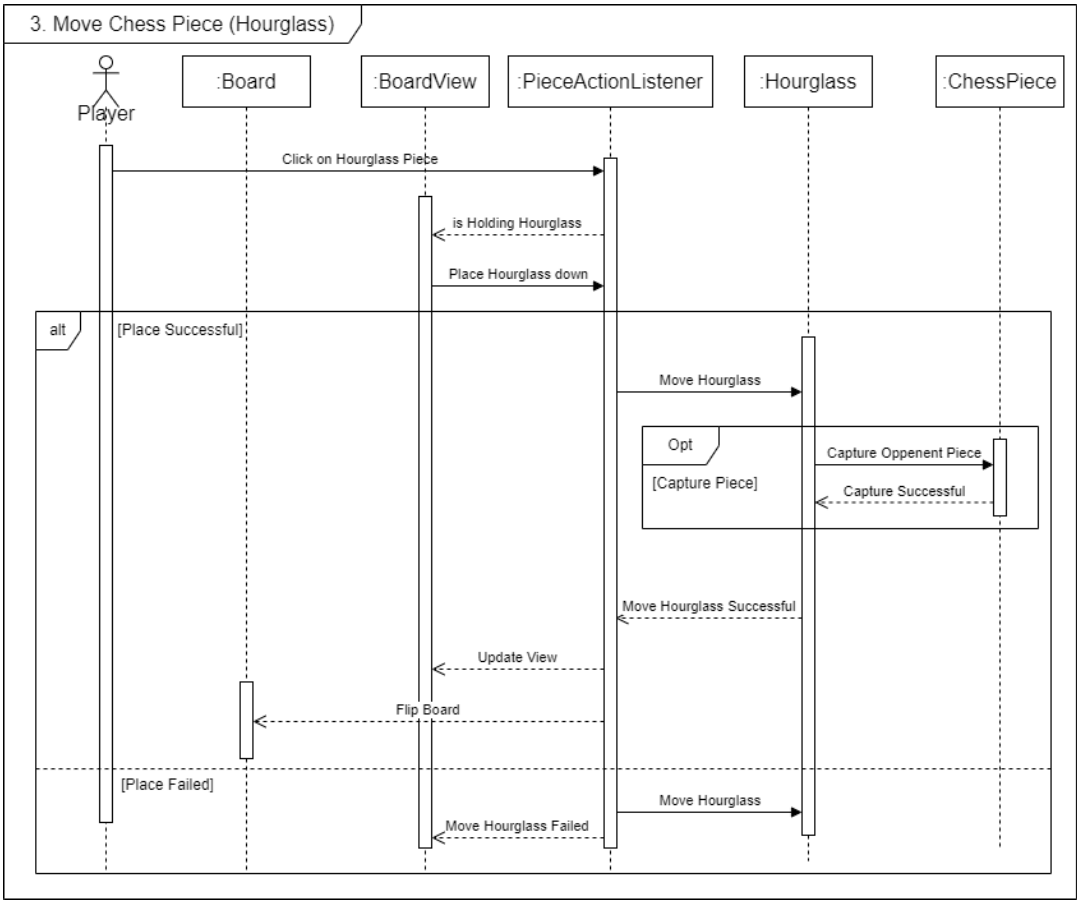


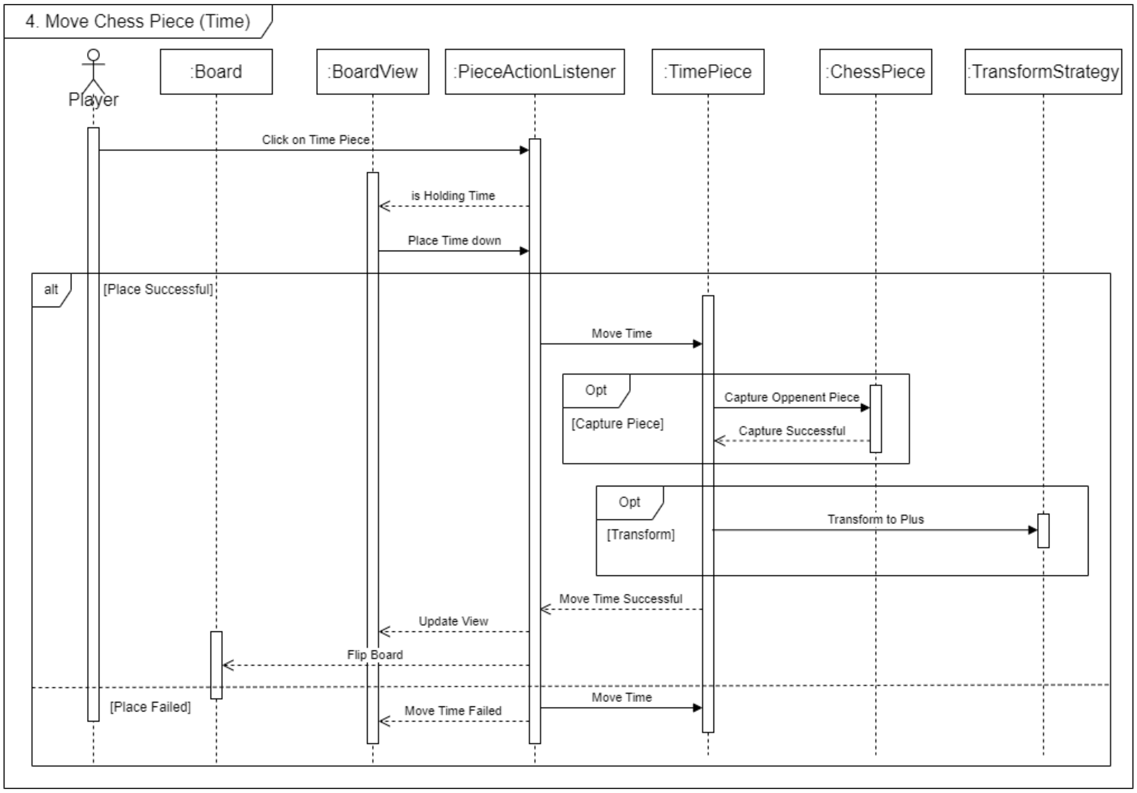
The Talabia Chess game use case diagram above shows the use cases for two players—Player One and Player Two. Both players can start or exit a game, move chess pieces around the board, capture the opponent's pieces, and transform or reverse the direction of their own pieces under certain game rules. Additionally, players have the option to create a new game, pause the game, save their progress to resume later, or load a previously saved game to continue from where they left off.

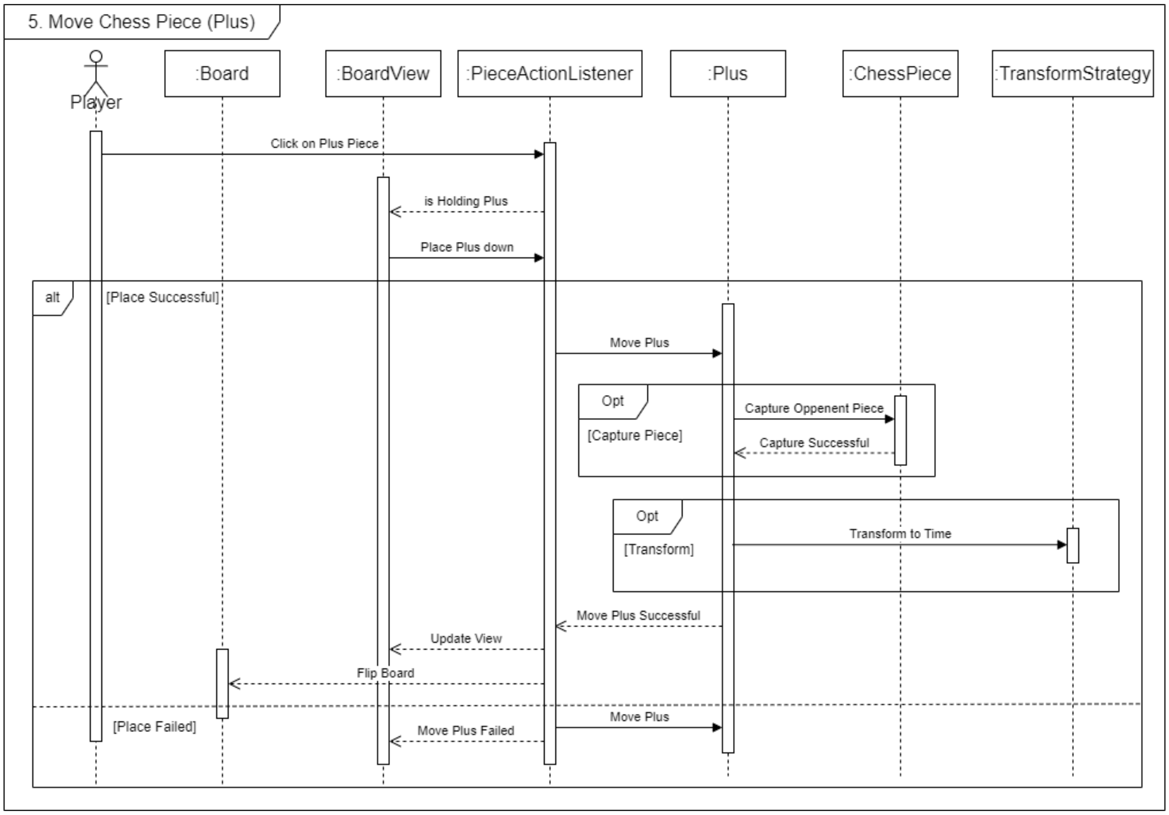
Sequence Diagrams

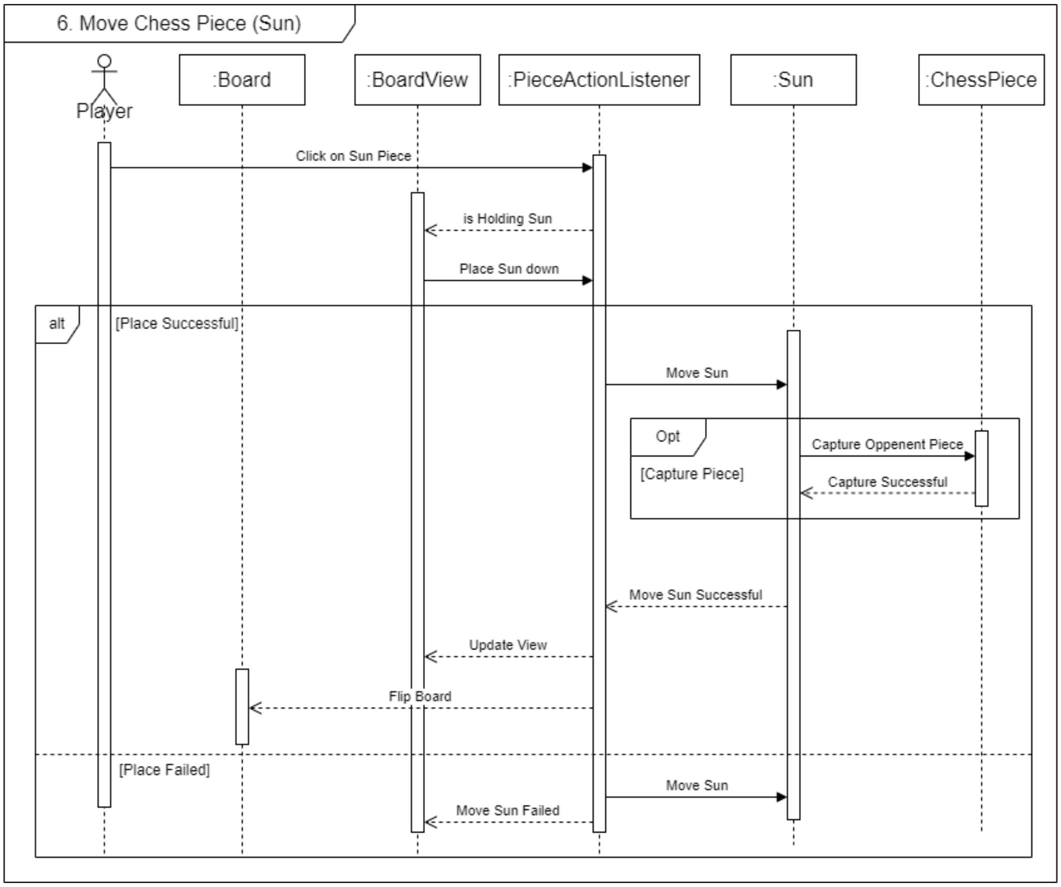


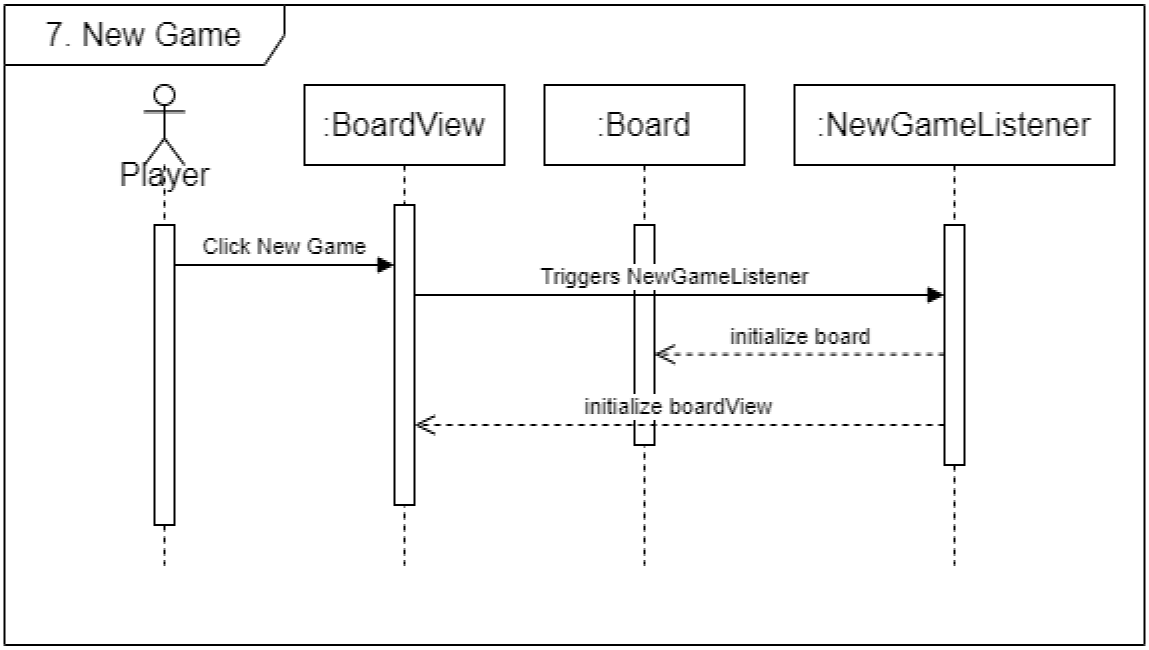


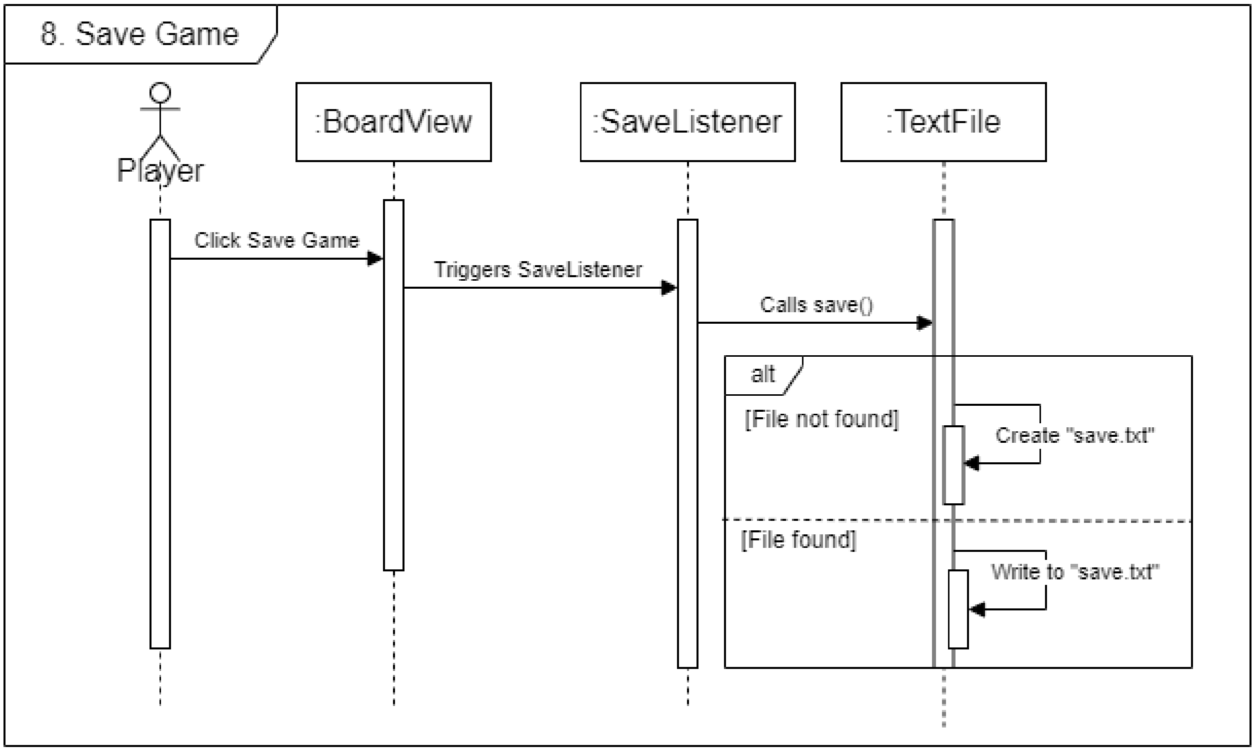

3. Move Chess Piece (Hourglass)
Player
:Board
:BoardView
:PieceActionListener
:Hourglass
:ChessPiece
Click on Hourglass Piece
is Holding Hourglass
Place Hourglass down
alt
[Place Successful]
Move Hourglass
Opt
[Capture Piece]
Capture Oppenent Piece
Capture Successful
Move Hourglass Successful
Update View
Flip Board
[Place Failed]
Move Hourglass
Move Hourglass Failed

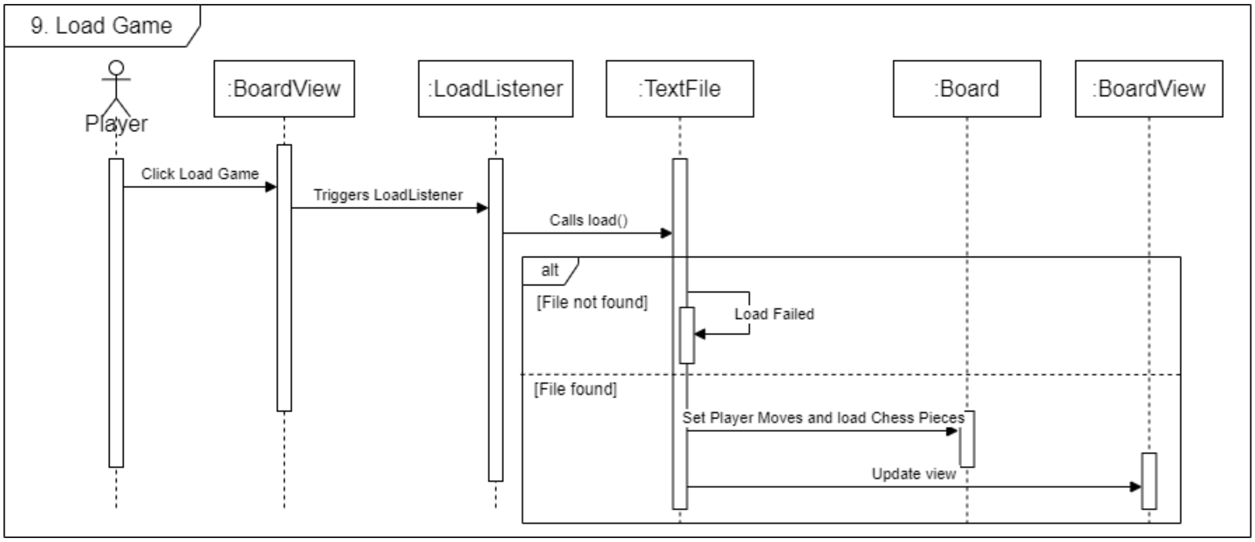

4. Move Chess Piece (Time)
Player
:Board
:BoardView
:PieceActionListener
:TimePiece
:ChessPiece
:TransformStrategy
Click on Time Piece
is Holding Time
Place Time down
alt
[Place Successful]
Move Time
Opt
[Capture Piece]
Capture Oppenent Piece
Capture Successful
Opt
[Transform]
Transform to Plus
Move Time Successful
Update View
Flip Board
[Place Failed]
Move Time
Move Time Failed

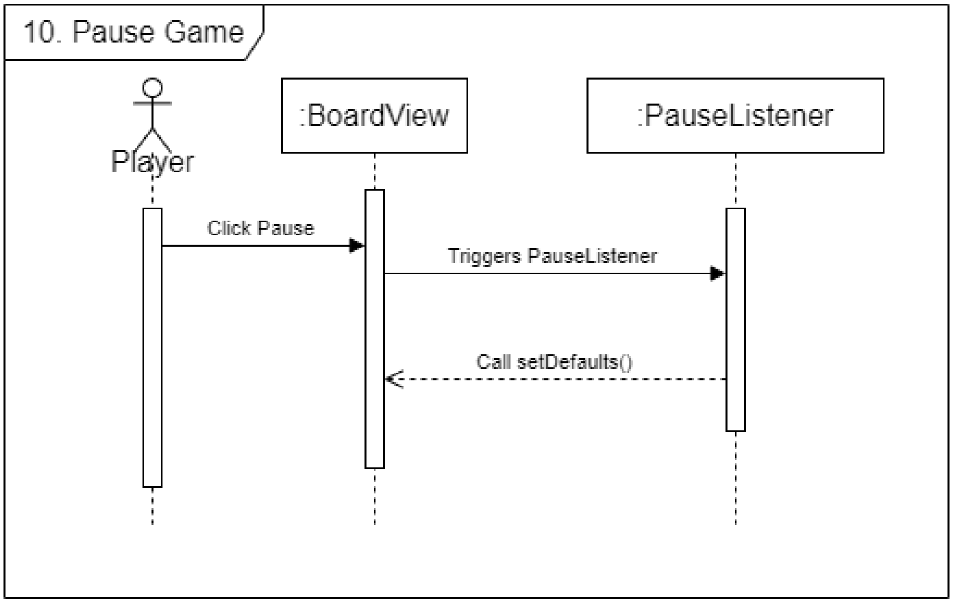

5. Move Chess Piece (Plus)
Player
:Board
:BoardView
:PieceActionListener
:Plus
:ChessPiece
:TransformStrategy
Click on Plus Piece
is Holding Plus
Place Plus down
alt
[Place Successful]
Move Plus
Opt
[Capture Piece]
Capture Oppenent Piece
Capture Successful
Opt
[Transform]
Transform to Time
Move Plus Successful
Update View
Flip Board
[Place Failed]
Move Plus
Move Plus Failed

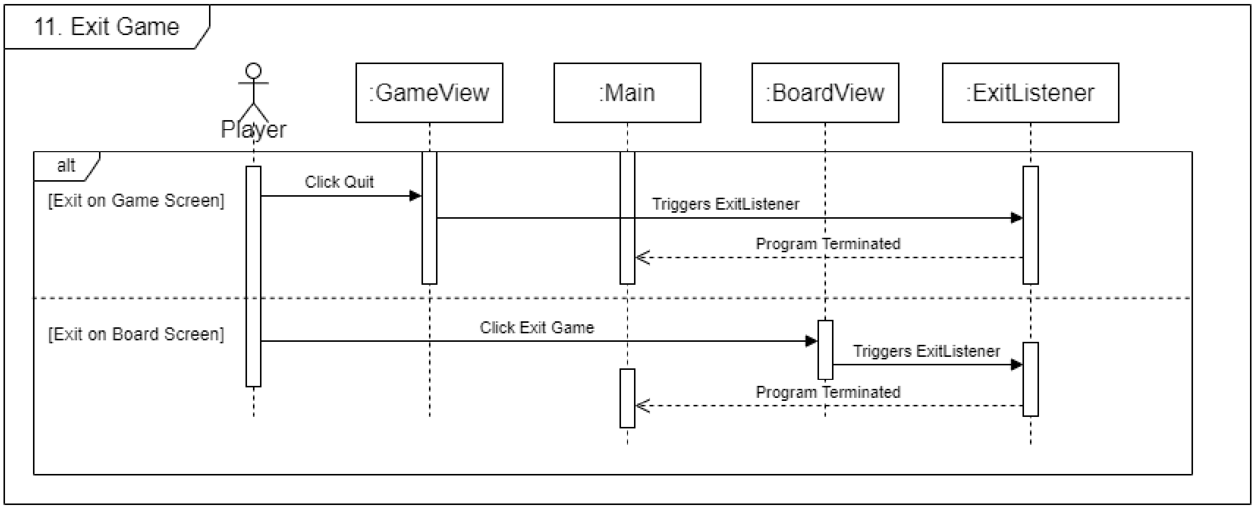

6. Move Chess Piece (Sun)
Player
:Board
:BoardView
:PieceActionListener
:Sun
:ChessPiece
Click on Sun Piece
is Holding Sun
Place Sun down
alt
[Place Successful]
Move Sun
Opt
[Capture Piece]
Capture Oppenent Piece
Capture Successful
Move Sun Successful
Update View
Flip Board
[Place Failed]
Move Sun
Move Sun Failed


7. New Game
Player
:BoardView
:Board
:NewGameListener
Click New Game
Triggers NewGameListener
initialize board
initialize boardView

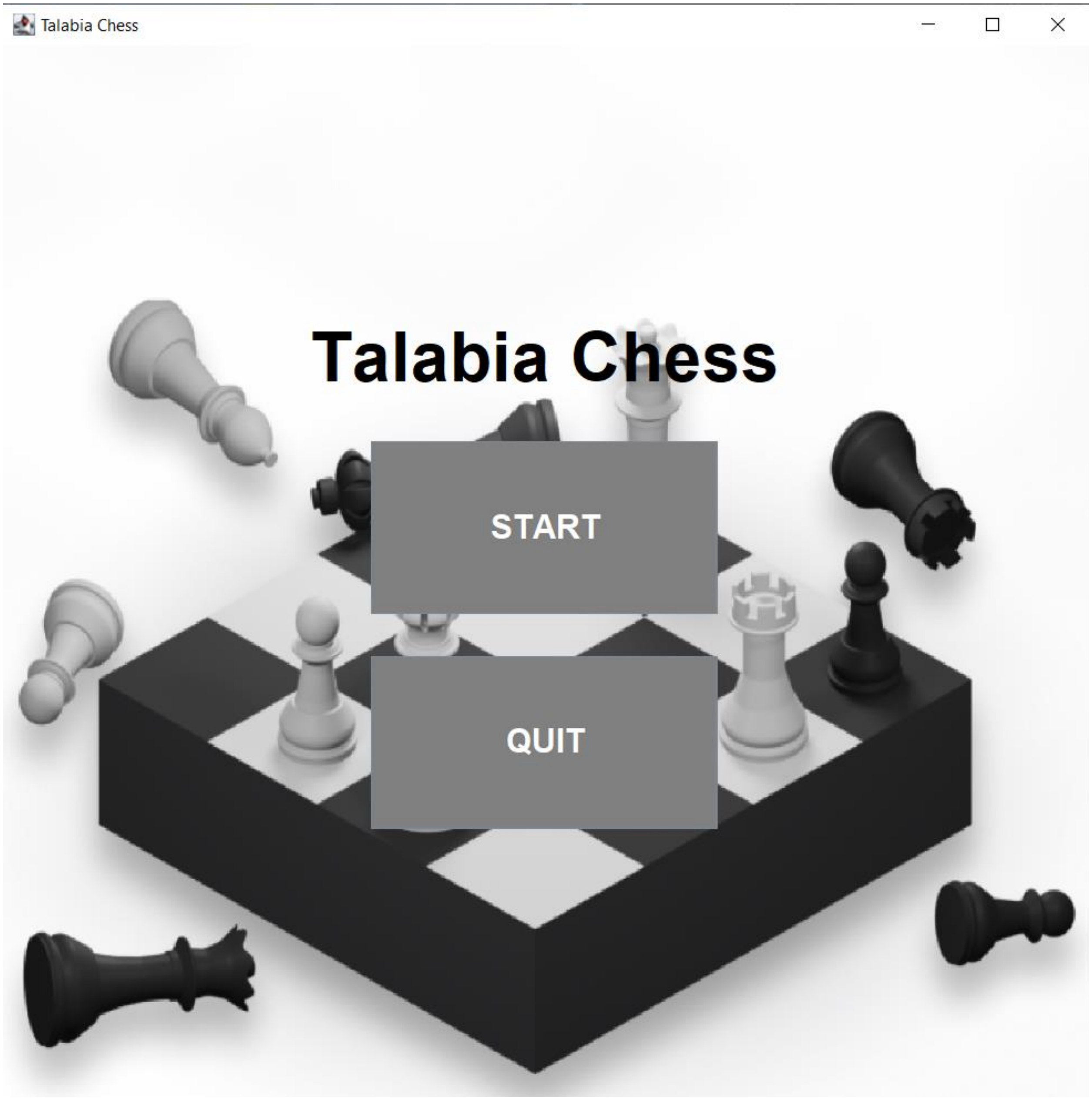

8. Save Game
Player
:BoardView
:SaveListener
:TextFile
Click Save Game
Triggers SaveListener
Calls save()
alt
[File not found]
Create "save.txt"
[File found]
Write to "save.txt"


9. Load Game
Player
:BoardView
:LoadListener
:TextFile
:Board
:BoardView
Click Load Game
Triggers LoadListener
Calls load()
alt
[File not found]
Load Failed
[File found]
Set Player Moves and load Chess Pieces
Update view

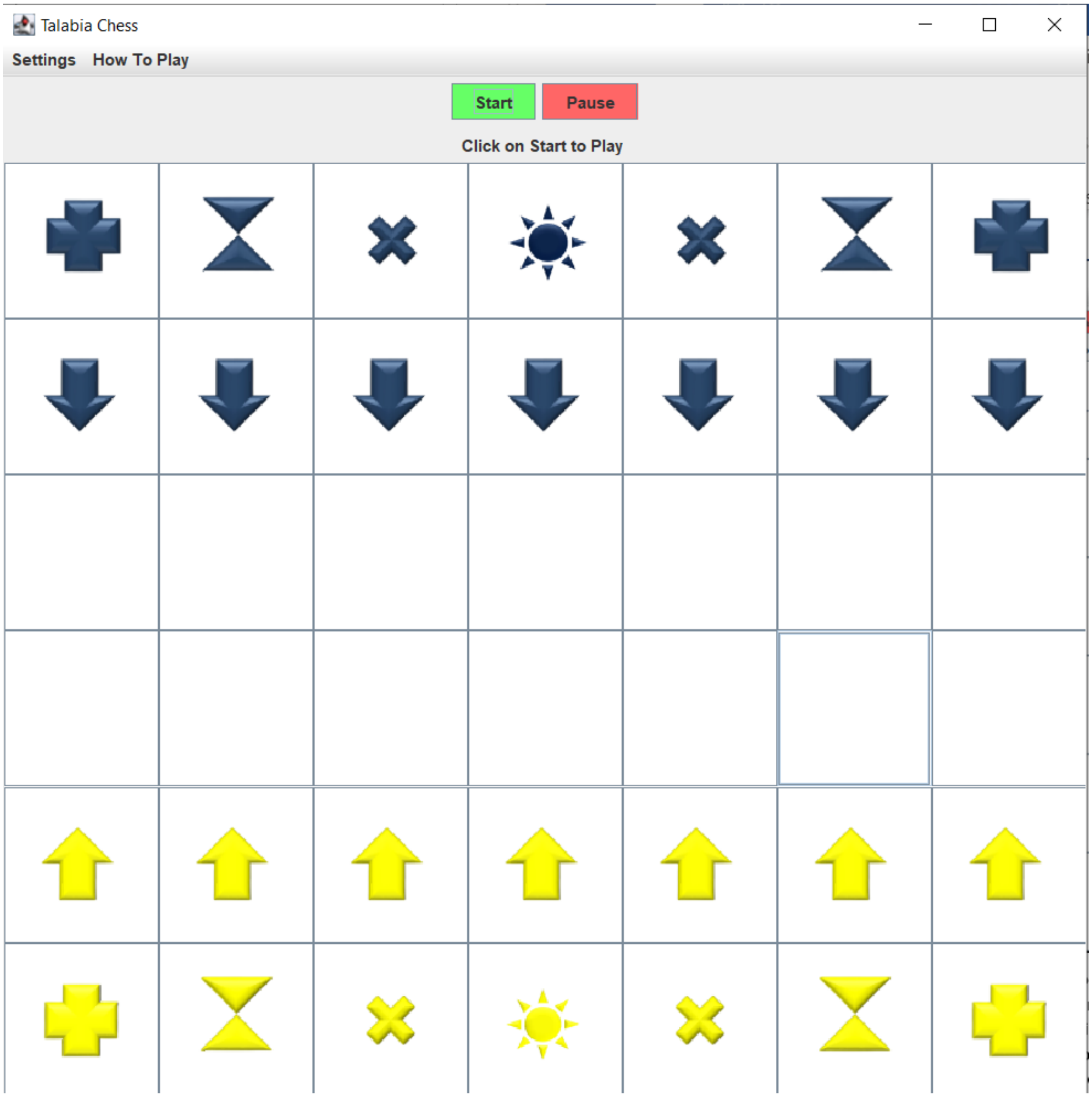

10. Pause Game
Player
:BoardView
:PauseListener
Click Pause
Triggers PauseListener
Call setDefaults()


11. Exit Game
Player
:GameView
:Main
:BoardView
:ExitListener
alt
[Exit on Game Screen]
Click Quit
Triggers ExitListener
Program Terminated
[Exit on Board Screen]
Click Exit Game
Triggers ExitListener
Program Terminated

User Documentation

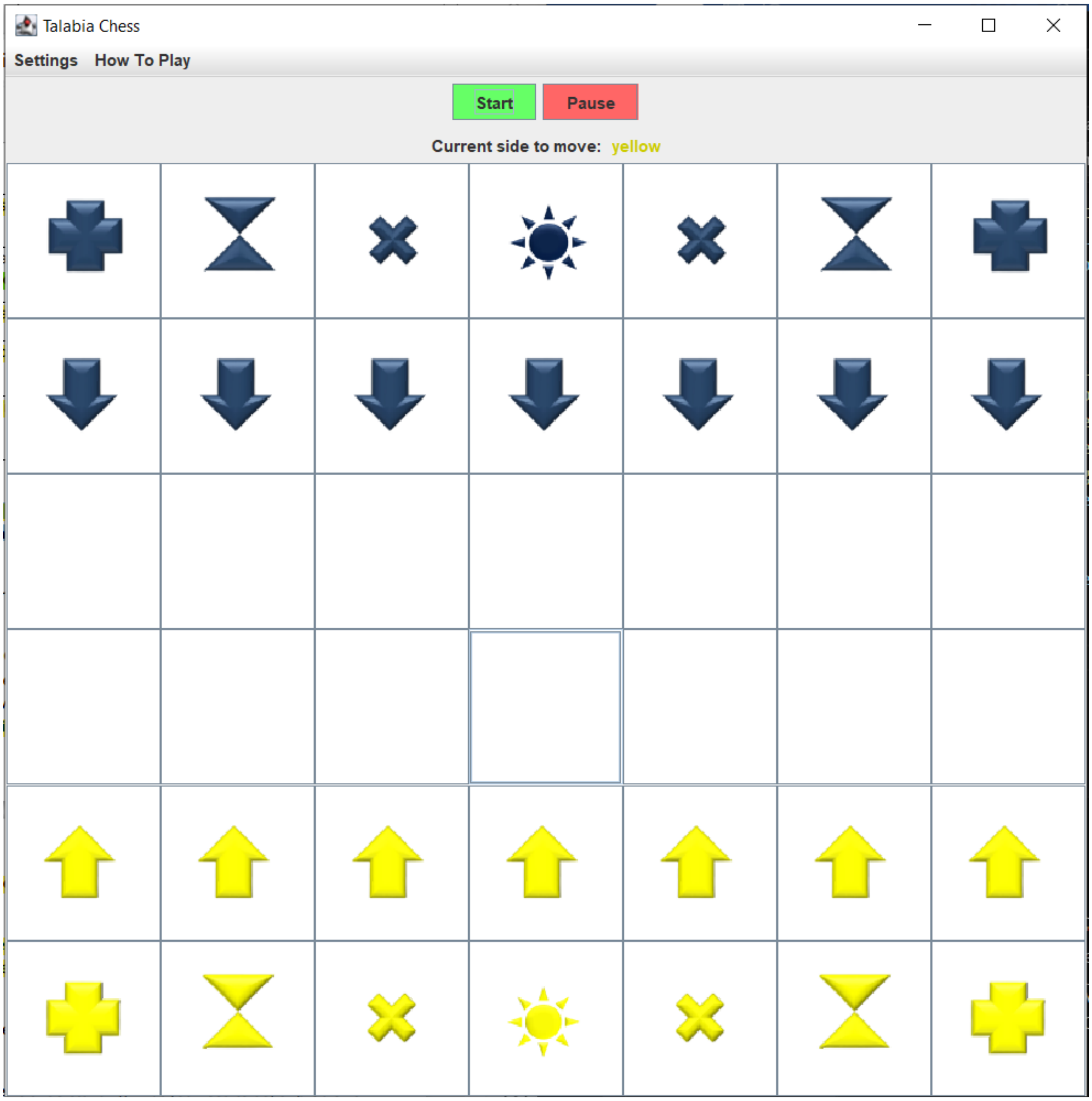


After the program executes successfully, a Talabia Chess main screen will appear. The main screen shows two buttons called “Start” and “Quit”. By pressing “Quit” the program will terminate itself. If the user clicks on “Start” the Talabia Chess Board will appear.

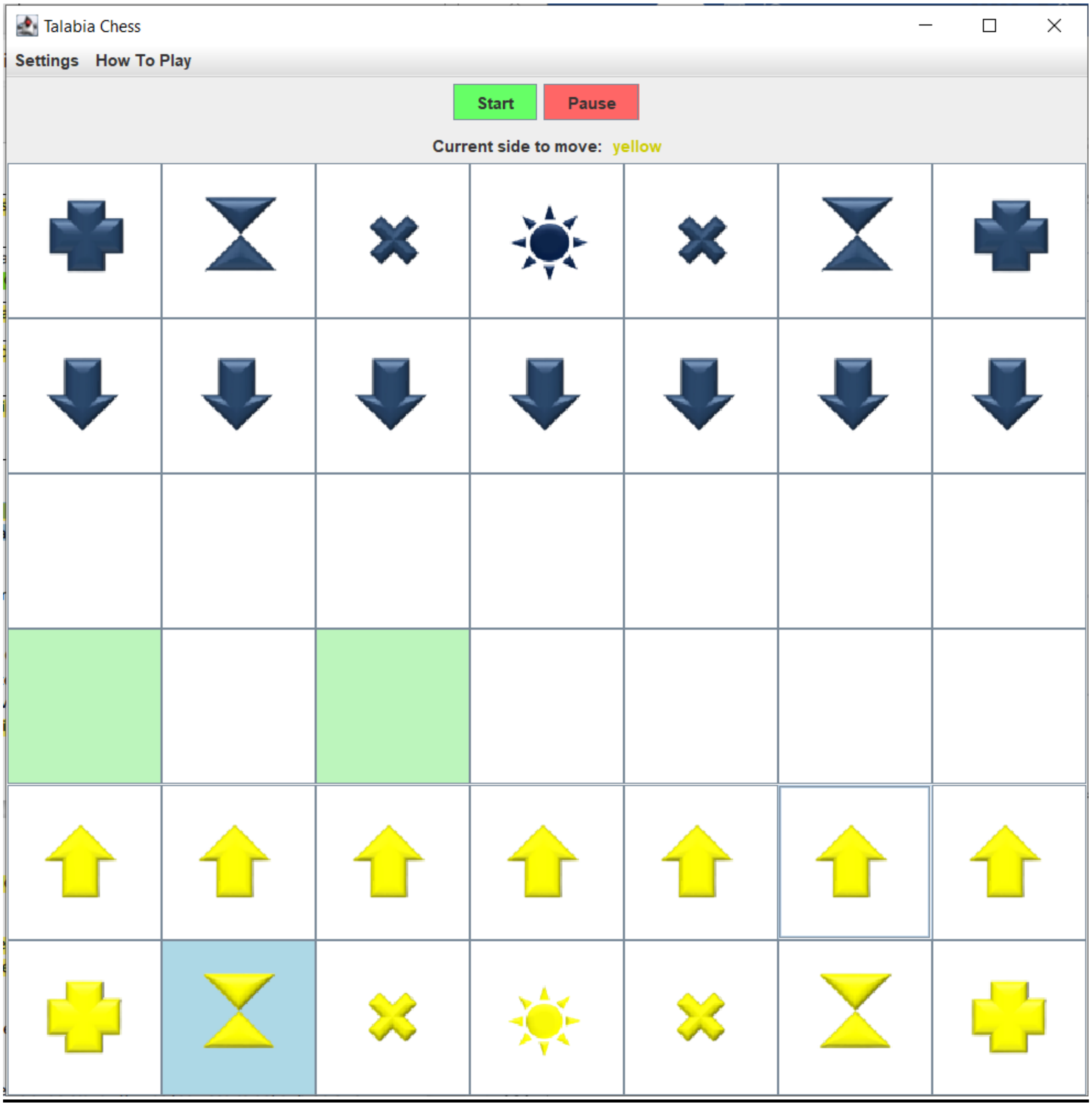


Click on Start to Play

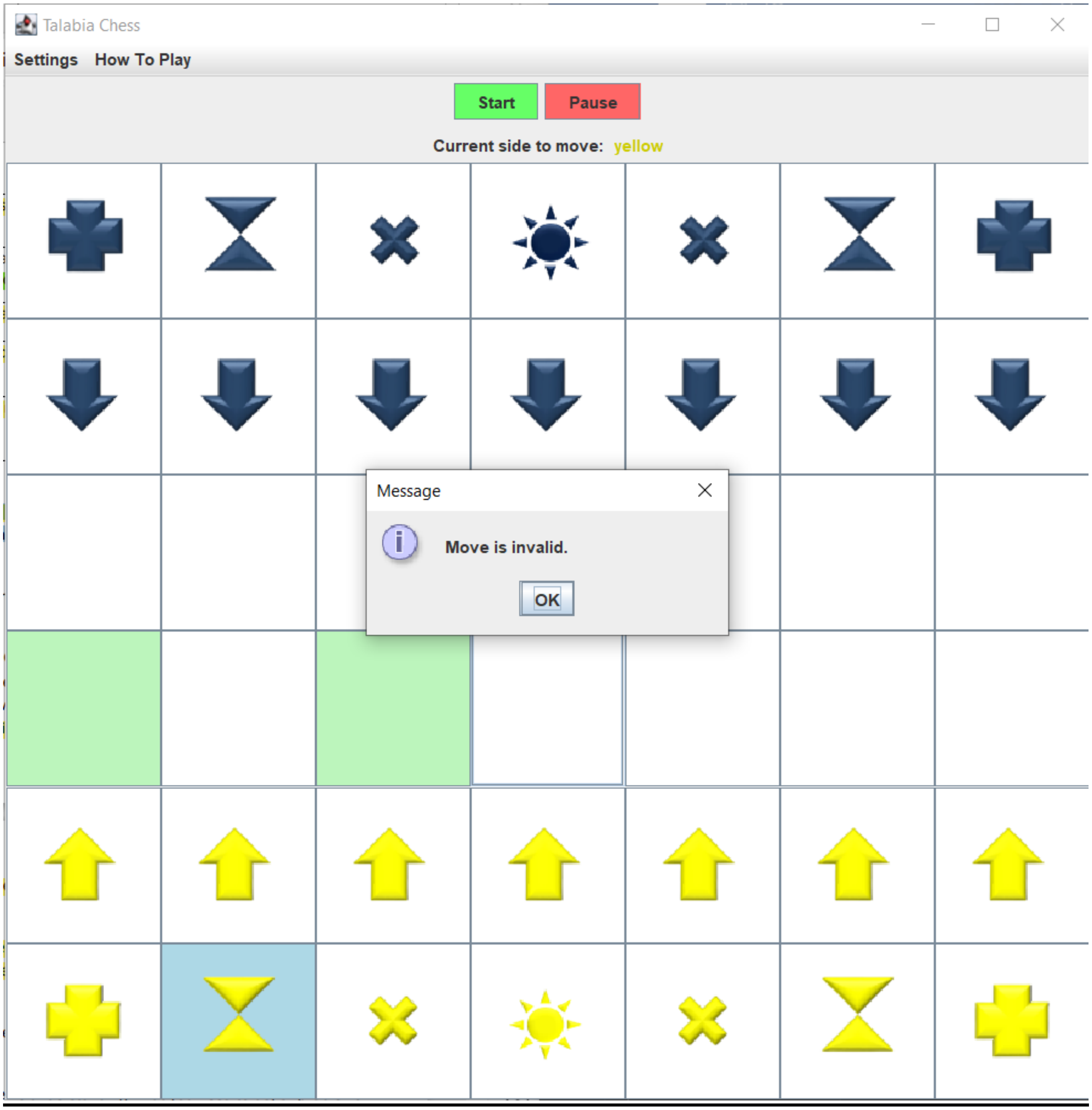
The Talabia Chess Board is a 7x6 board and it shows 28 pieces of chess piece. Each side has 14 pieces. From the bottom left, the pieces are Plus, Hourglass, Time, Sun, Time, Hourglass and Plus. The arrows are called the Point piece. The Point piece can only move forward, 1 or 2 steps. If it reaches the end of the board, it turns around and starts heading back the other way. It cannot skip over other pieces. The Hourglass piece moves in a 3x2 L shape in any orientation. This is the only piece that can skip over other pieces. The Time piece can only move diagonally but can go any distance. It cannot skip over other pieces. The Plus piece can move horizontally and vertically only but can go any distance. It cannot skip over other pieces. The Sun piece can move only one step in any direction. The game ends when the Sun is captured by the other side. The game is played by 2 players, one from the yellow side and one from the blue side.



The user has to click on “Start” to play the game. The board will state the current side to move at the moment. If the user does not click on “Start” the board will stay in it’s original state and the user will not be able to move any pieces. When the game starts, the user can also choose to pause the game by clicking “Pause”, this is to prevent the user from misclicking when they want to do something.



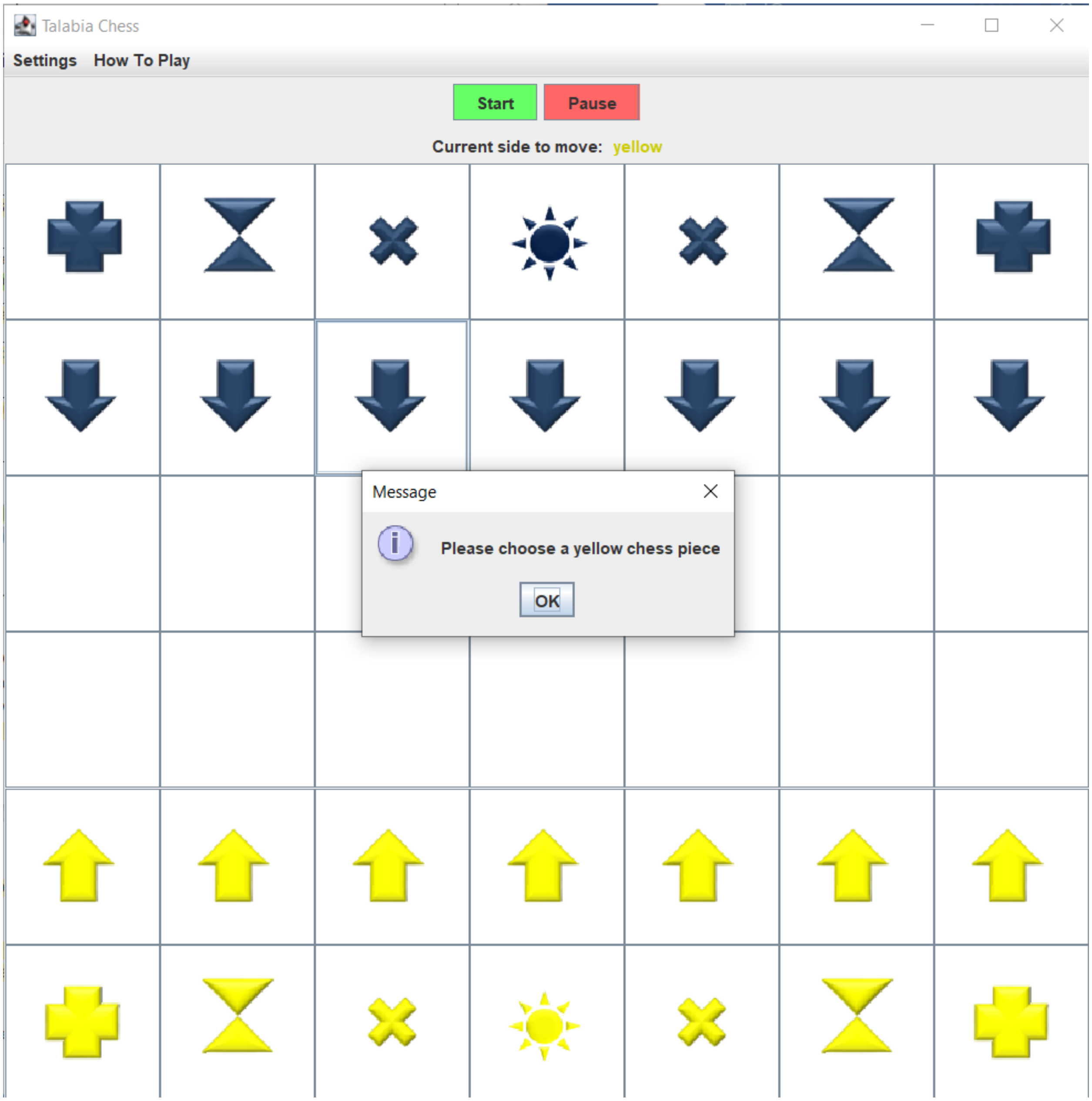
The user just has to click on the chess piece that they want to move. When the user clicks on a chess piece, the board will highlight the selected chess piece in blue and the boxes that it can move to will be highlighted in green.



Current side to move: yellow

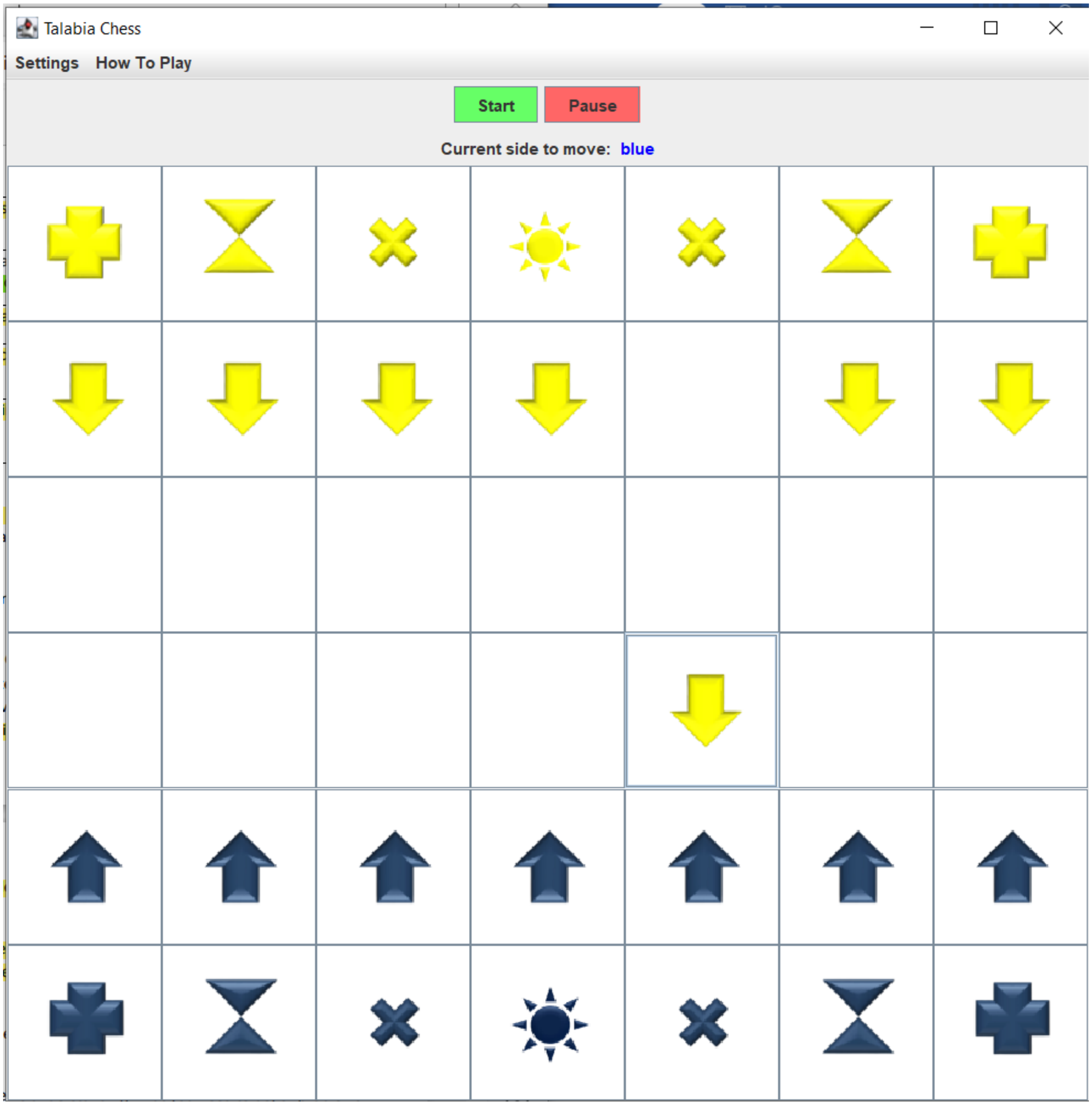
Move is invalid.

All chess pieces can only move in the direction that it is allowed to. Different chess pieces have different movements. If the user moves the chess piece to a box that it is not allowed to, a message " Move is invalid." will pop up. Besides, if there is a chess piece blocking the chess piece from moving to its desired box, the chess piece will be unable to move. The chess pieces are not allowed to capture the pieces of the same side but are allowed to capture the chess piece from the other side.



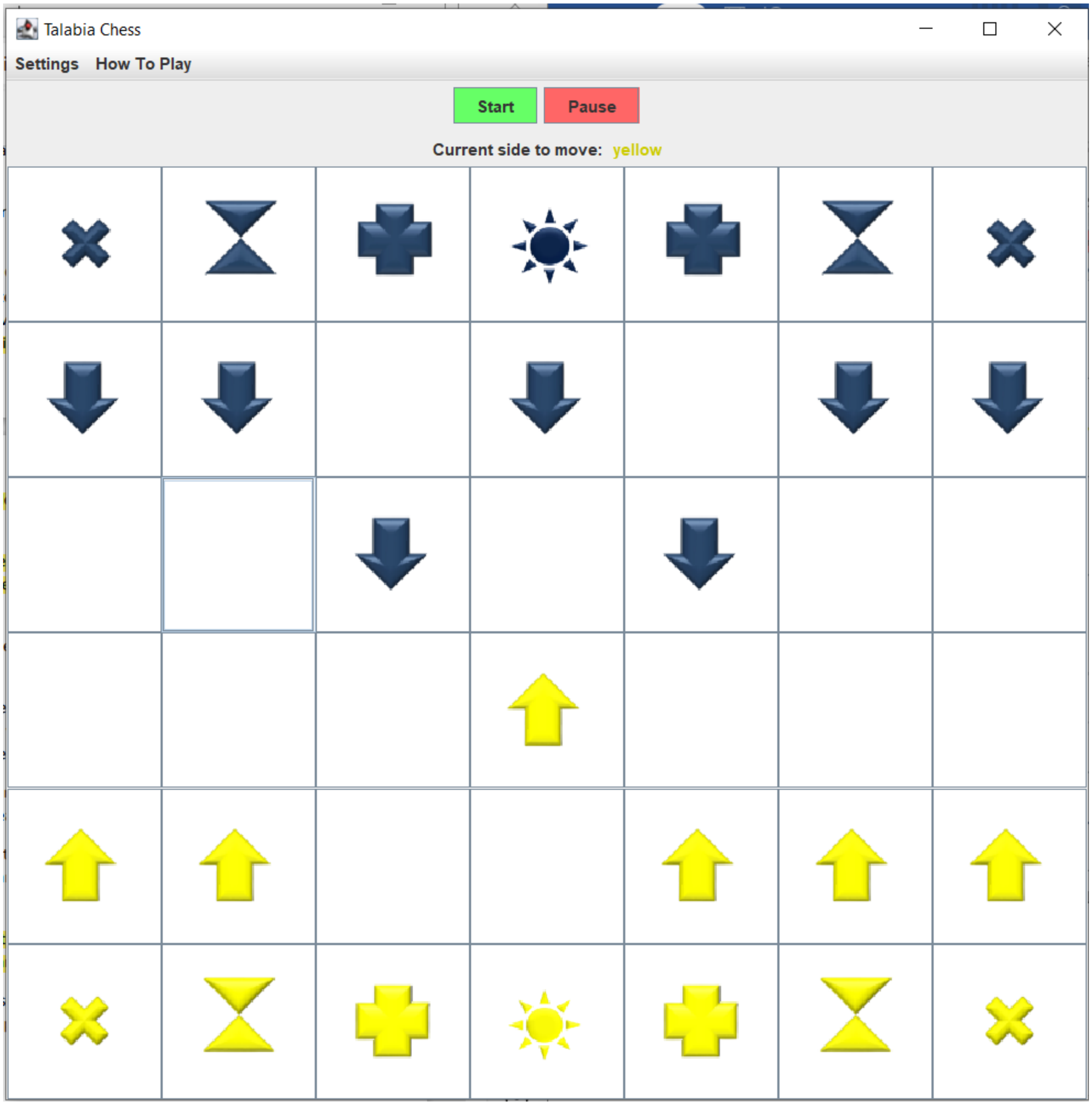
Please choose a yellow chess piece

If it is the yellow side's turn but the user wants to move the blue chess piece, a message "Please choose a yellow chess piece" will pop up. If it's the blue side's turn and the user tries to move the yellow piece, a message "Please choose a blue chess piece" will pop up.

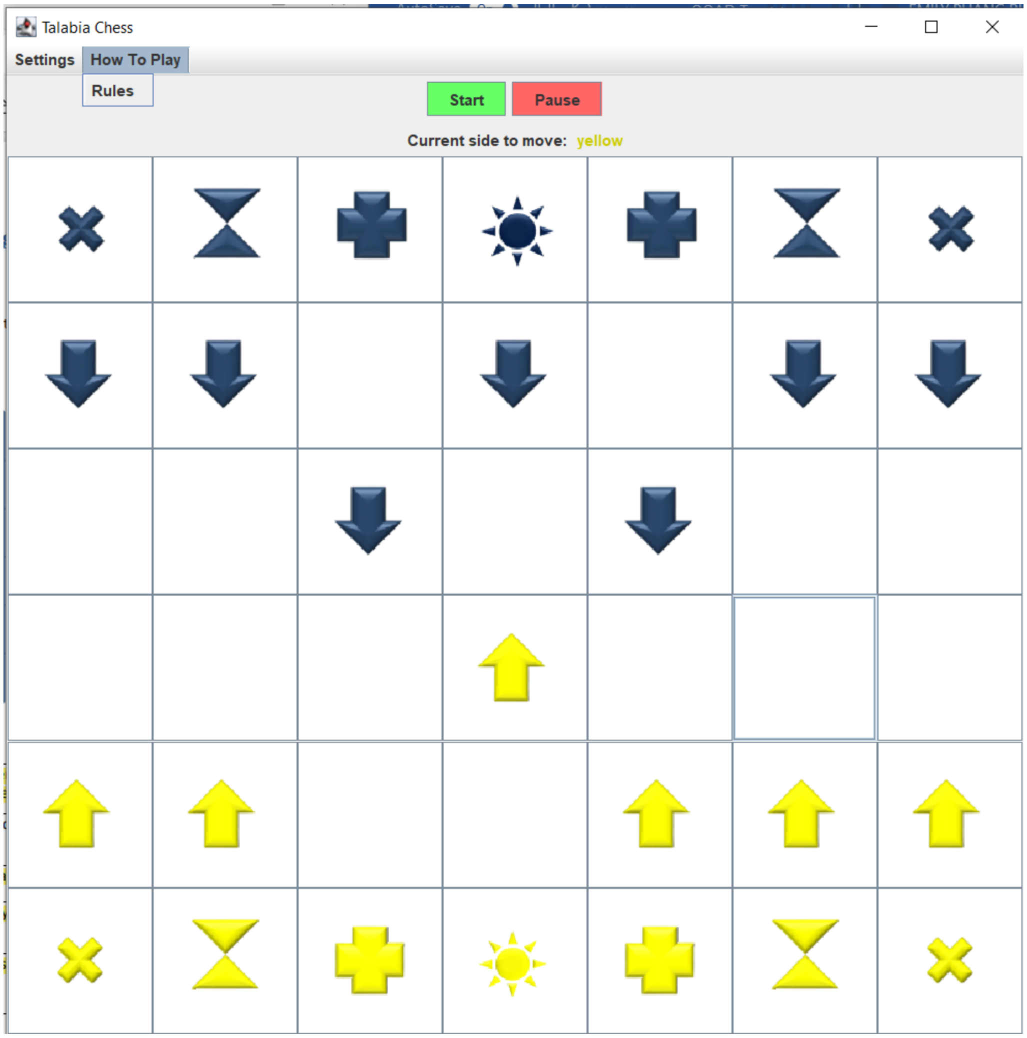


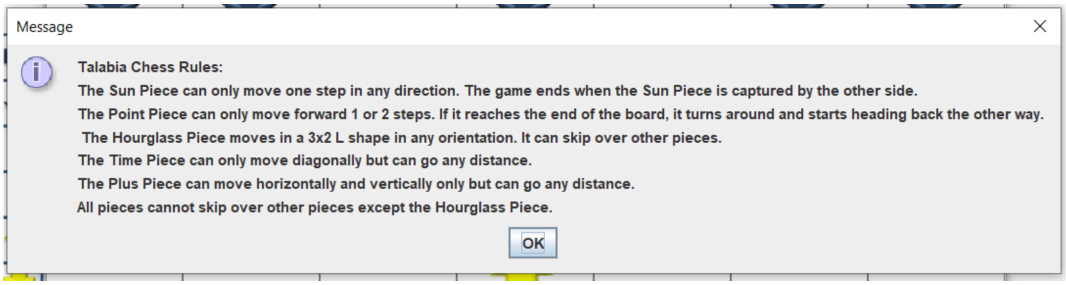
Current side to move: blue

After the yellow side makes a move, the board will flip and it is now the blue side’s turn to play.



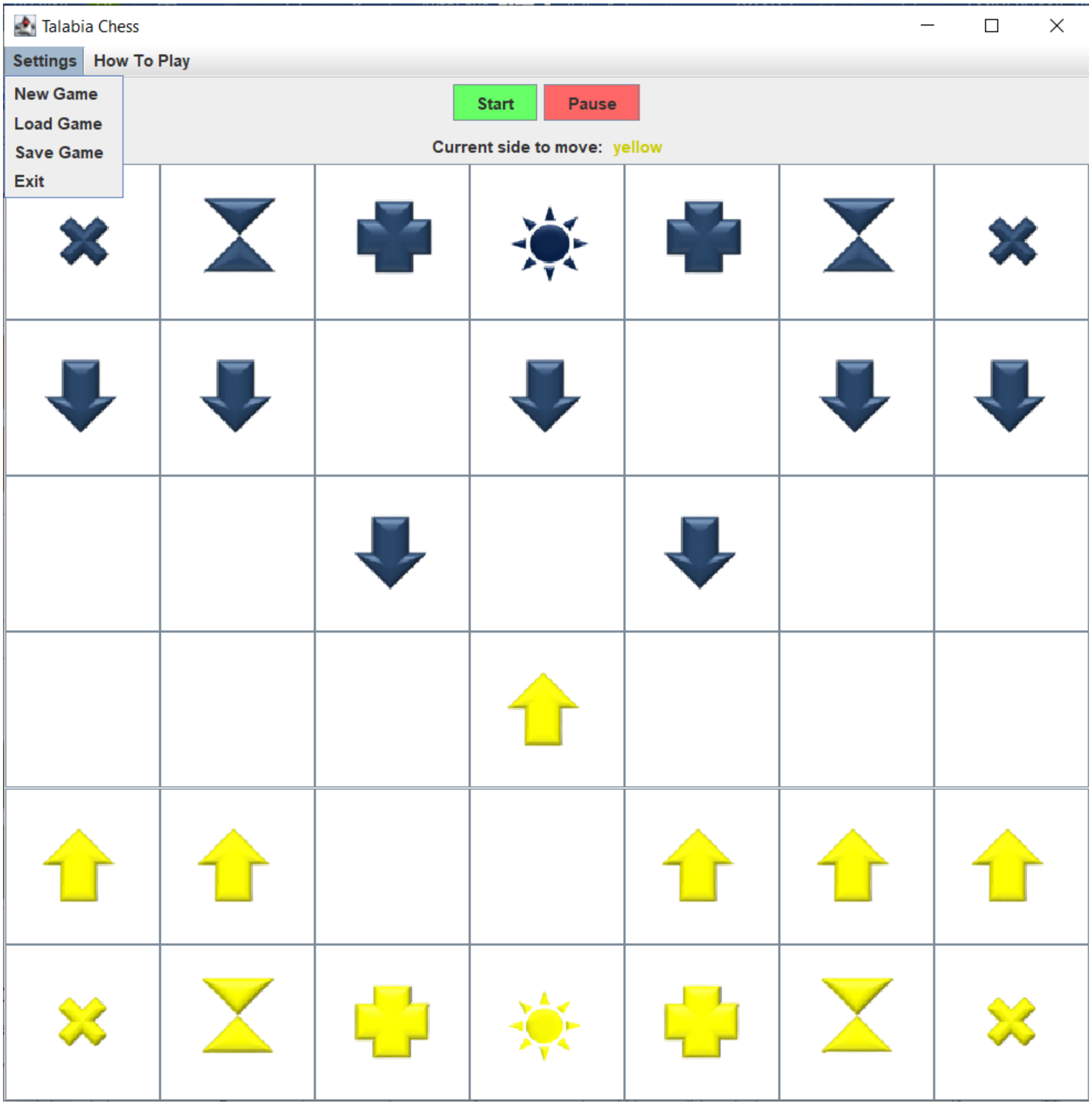
After 2 turns, counting one yellow move and one blue move as one turn, all Time pieces will turn into Plus pieces, and all Plus pieces will turn into Time pieces.





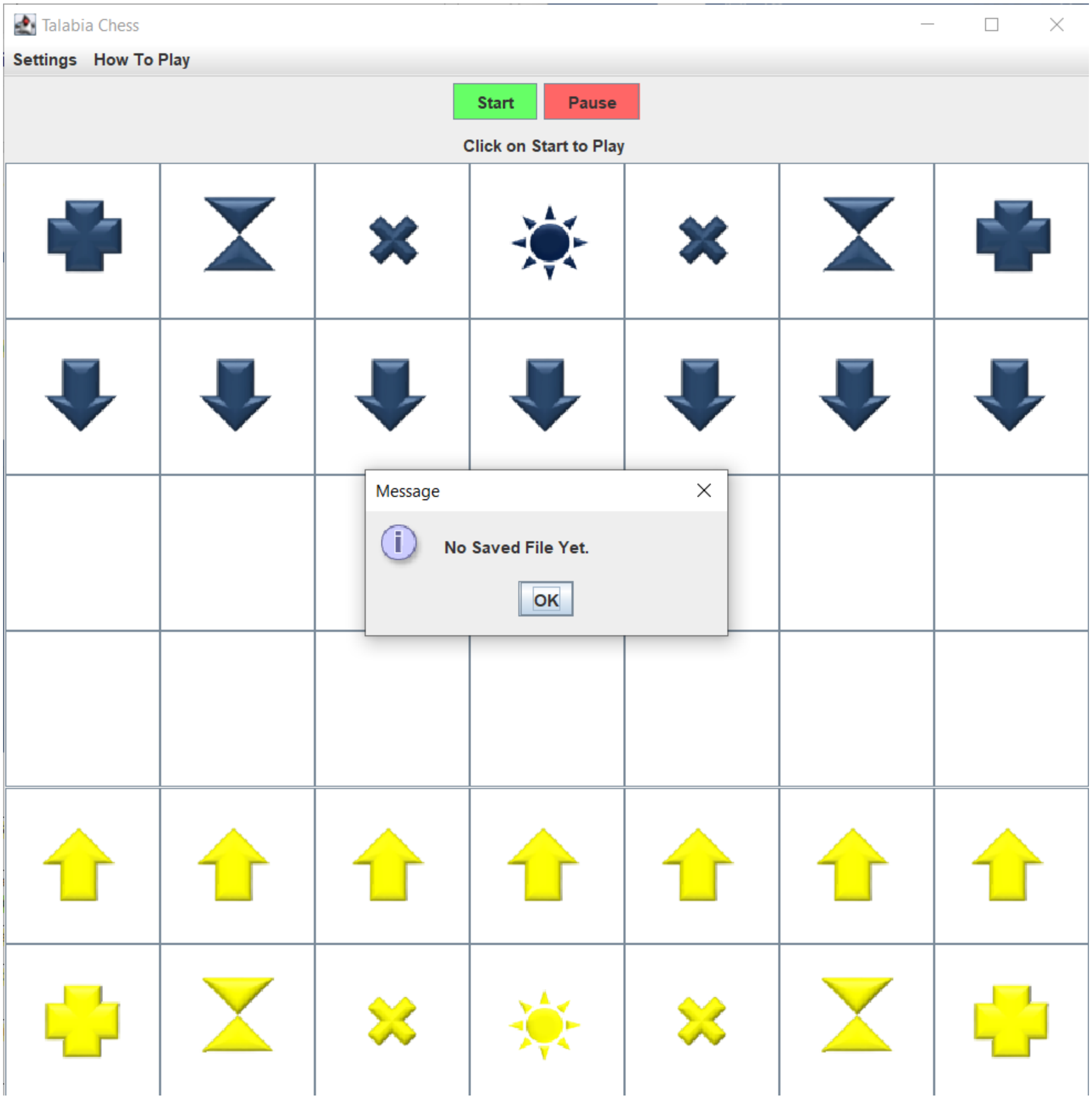
Talabia Chess Rules:
The Sun Piece can only move one step in any direction. The game ends when the Sun Piece is captured by the other side.
The Point Piece can only move forward 1 or 2 steps. If it reaches the end of the board, it turns around and starts heading back the other way.
The Hourglass Piece moves in a 3x2 L shape in any orientation. It can skip over other pieces.
The Time Piece can only move diagonally but can go any distance.
The Plus Piece can move horizontally and vertically only but can go any distance.
All pieces cannot skip over other pieces except the Hourglass Piece.

If the user wishes to know more about how to play the game, the user can click on the "How To Play" section on the menu bar and click on "Rules". A message with Talabia Chess Rules will pop up.



Current side to move: yellow

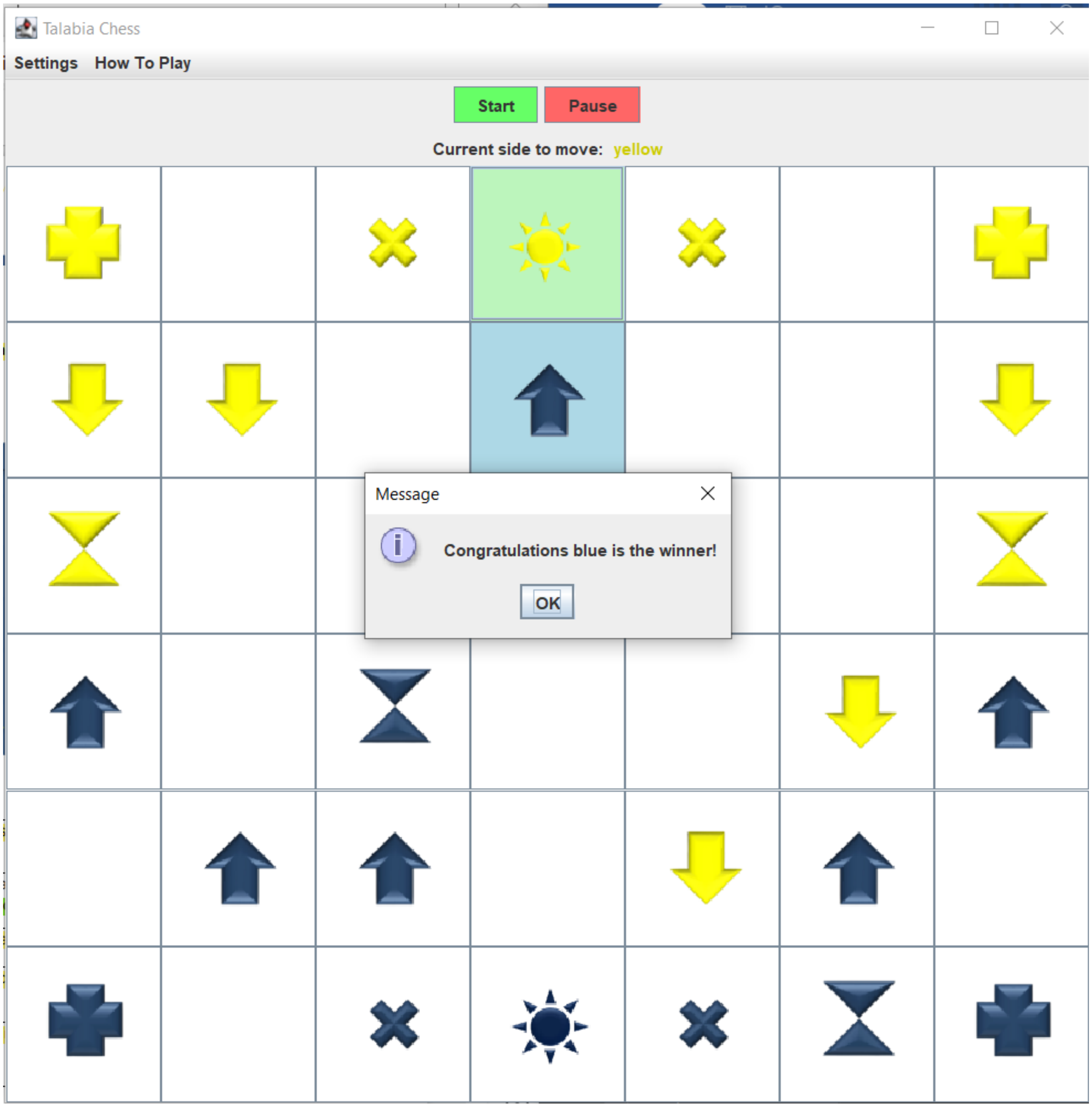
The user can click on “Settings” on the menu bar to see more options to do with the board. The user can click on “New Game” to generate a new board and start from the beginning. The user can also save the current state of the board by clicking on “Save Game”. The user can also click on “Load Game’ to load the game from a saved file. Lastly, the user can click on “Exit” to exit the game. The program will then terminate itself.



Click on Start to Play

No Saved File Yet.

If the user has not saved a file before but chooses to load the game, a message “No Saved File Yet” will pop up.



Current side to move: yellow

Message

Congratulations blue is the winner!

If the user successfully captures the Sun piece. A message “Congratulations (winner’s side) is the winner!” will pop up to let the user know which side has won. The board will then reset and start a new game from the beginning after the user clicks on “OK”.