

# **OBJECT ORIENTED ANALYSIS AND DESIGN TCP2201 PROJECT**

**Trimester 2310**

*by <<Only Friends>>*

**(TOPIC : TALABIA CHESS)**



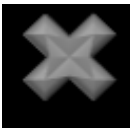

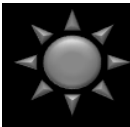
**FACULTY OF COMPUTING AND INFORMATICS (FCI)**

Team Leader			
Name	Student ID	Phone Number	Email
Ku Jing Hao	1201203310	017-4126838	1201203310@student.mmu.edu.my

Team Members			
Name	Student ID	Phone Number	Email
Ng Min Hoong	1201203289	017-6146828	1201203289@student.mmu.edu.my
Wan Aqel Hakimi Bin Mohd Zamri	1211305491	013-2695486	1211305491@student.mmu.edu.my
Tam Yu Heng	1221304730	012-7667076	1221304730@student.mmu.edu.my
See Jian Man	1201203321	011-61328186	1201203321@student.mmu.edu.my

# 1 Talabia Chess Rules

This project is a Webale Chess game which is a GUI-based Java Application chess game with a dimension of 7 x 6.

	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 (kind of like the knight in standard chess.) 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.

\*\*None of the pieces are allowed to skip over other pieces.

## Special Features

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. (This makes Talabia chess different from other chess games because the pieces will transform like that.)

## 2 Compile and Run Instructions

### Requirements

1. Java version 1.8.0\_261 or above
2. Javac 1.8.0\_261 or above
3. Ensure compatibility version between the Java Runtime Environment (JRE) that corresponds to “java” and Java Development Kit (JDK) that corresponds to “javac”

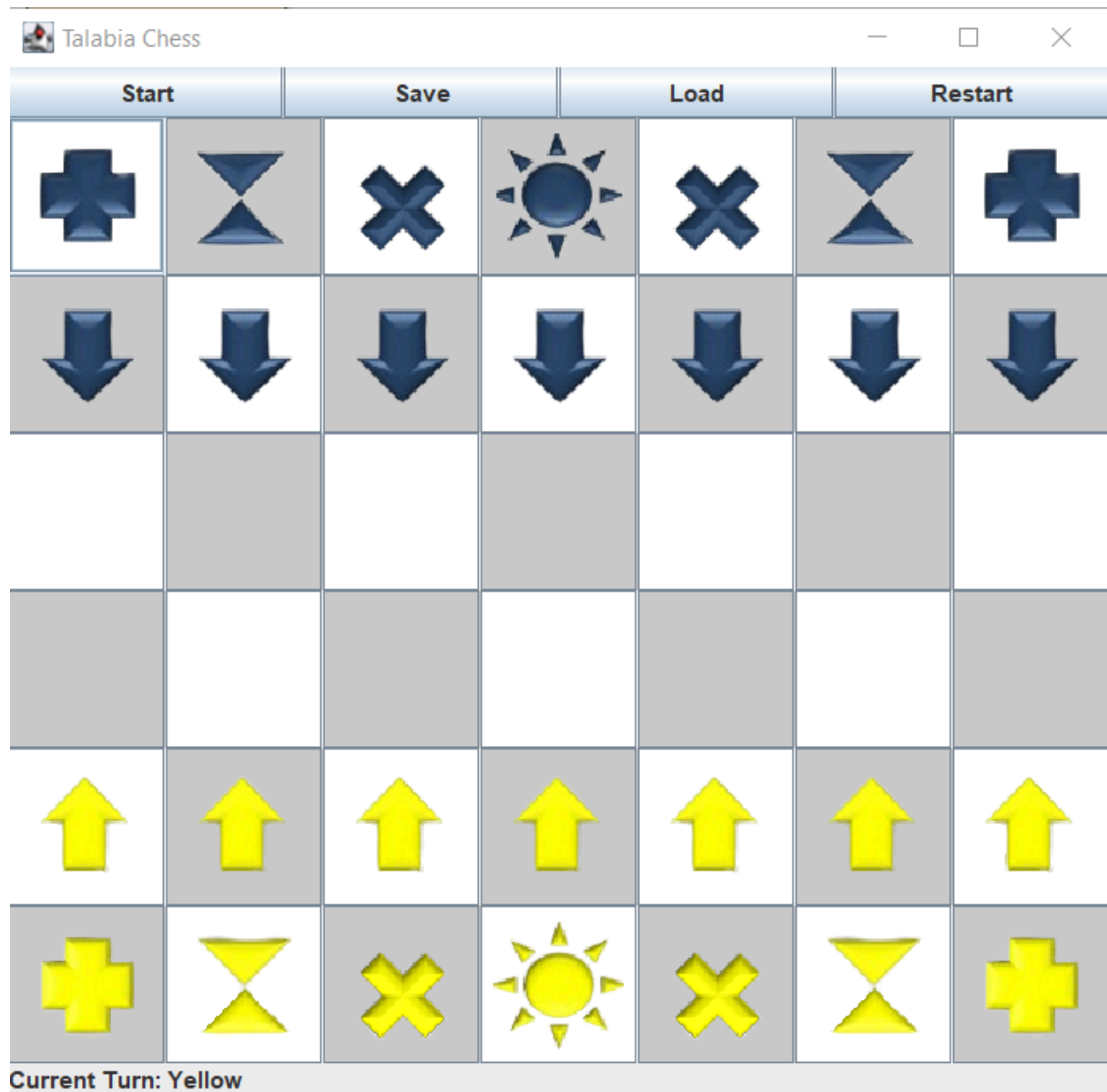
### To be Noticed

- ★ To verify the versions of the Java Development Kit (JDK) and Java Runtime Environment (JRE) installed on your system.
  - javac -version
  - java -version
- ★ If you haven't installed JDK and JRE
  1. Download and install the Java Development Kit (JDK) and Java Runtime Environment (JRE) from the official Oracle website. During installation, ensure the development tools, including the Java compiler (javac), are selected.
  2. Add the bin directory of the JDK to the system's PATH variable. Find the path (e.g., C:\Program Files\Java\jdk1.8.0\_391\bin) and edit the system environment variables accordingly.
  3. Restart the Command Prompt or your terminal for changes to take effect.

### Compilation and Running the game

1. Download files
2. Open Command Prompt (You can use either a or b)
  - a. Open the Start menu and type terminal/command prompt
  - b. Windows key + R. Type cmd or cmd.exe in the Run command box
3. Set path to the file (your file name)
  - a. cd YOURFILEPATH
    - i. If you mistakenly enter the wrong file, use "cd.." to navigate back.
    - ii. In case you forget the filename, type "dir" to list all files and folders in the current directory for reference.
4. Compile and run TalabiaChessMain.java with the following commands
  - a. javac TalabiaChessMain.java
  - b. java TalabiaChessMain

### 3 Talabia Chess Interface Design



## 4 UML Class Diagram

### Observer Pattern:

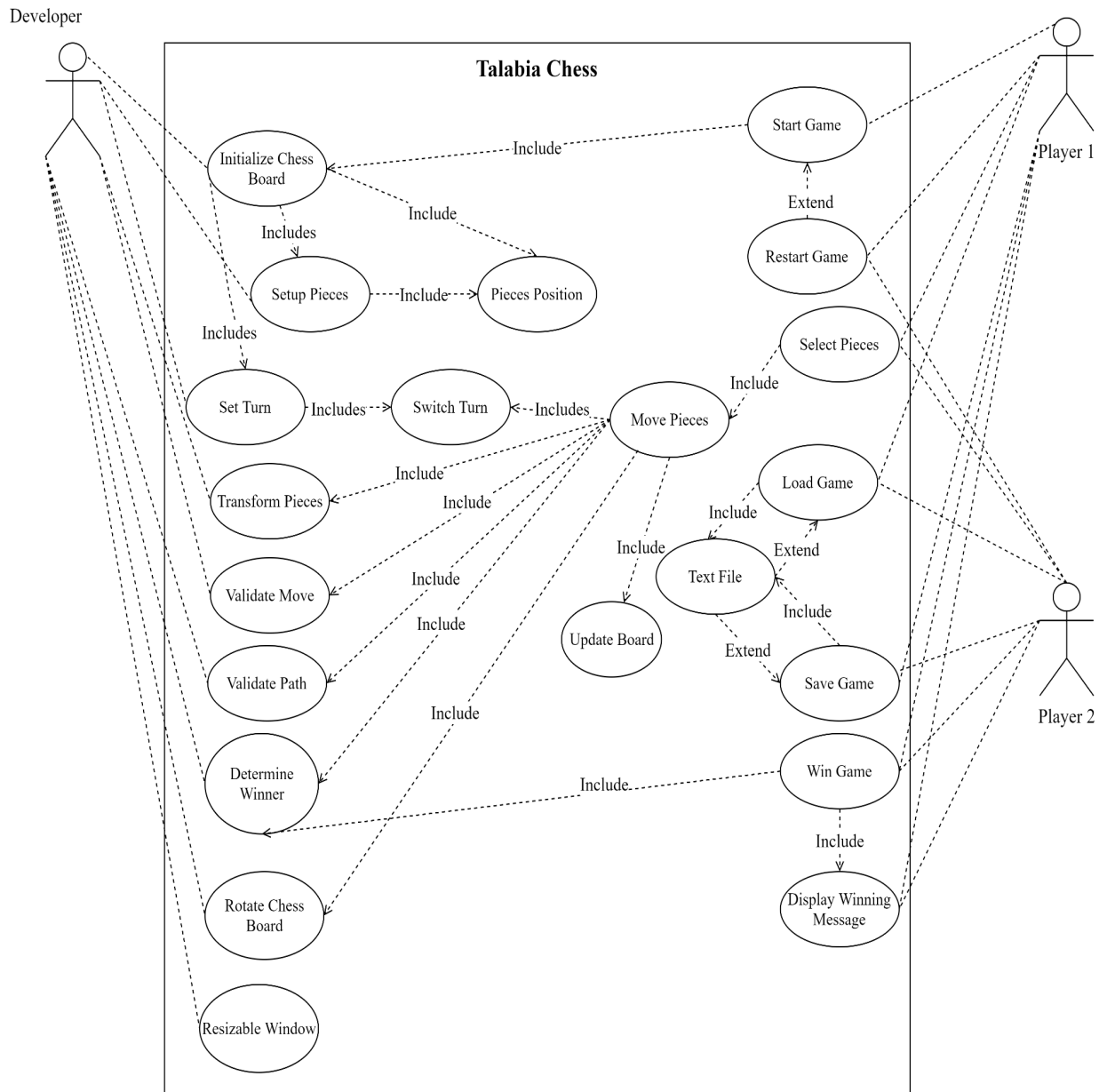
- Classes: 'TalabiaChessModel', 'TalabiaChessView'
- Roles:
  - 'TalabiaChessModel' acts as the subject that maintains a list of observers (observers).
  - 'TalabiaChessView' implements the 'Observer' interface and is registered as an observer to 'TalabiaChessModel'. It updates its state when notified.

### Singleton Pattern:

- Classes: GameFileManager
- Roles:
  - The Singleton pattern is applied to the 'GameFileManager' class to ensure that it has a single instance responsible for managing game files. The private constructor in 'GameFileManager' prevents direct instantiation from outside the class, and the 'getInstance()' method provides a globally accessible point to obtain the single instance. This design guarantees that there is only one instance of 'GameFileManager' throughout the application, making it responsible for centralized game file management.



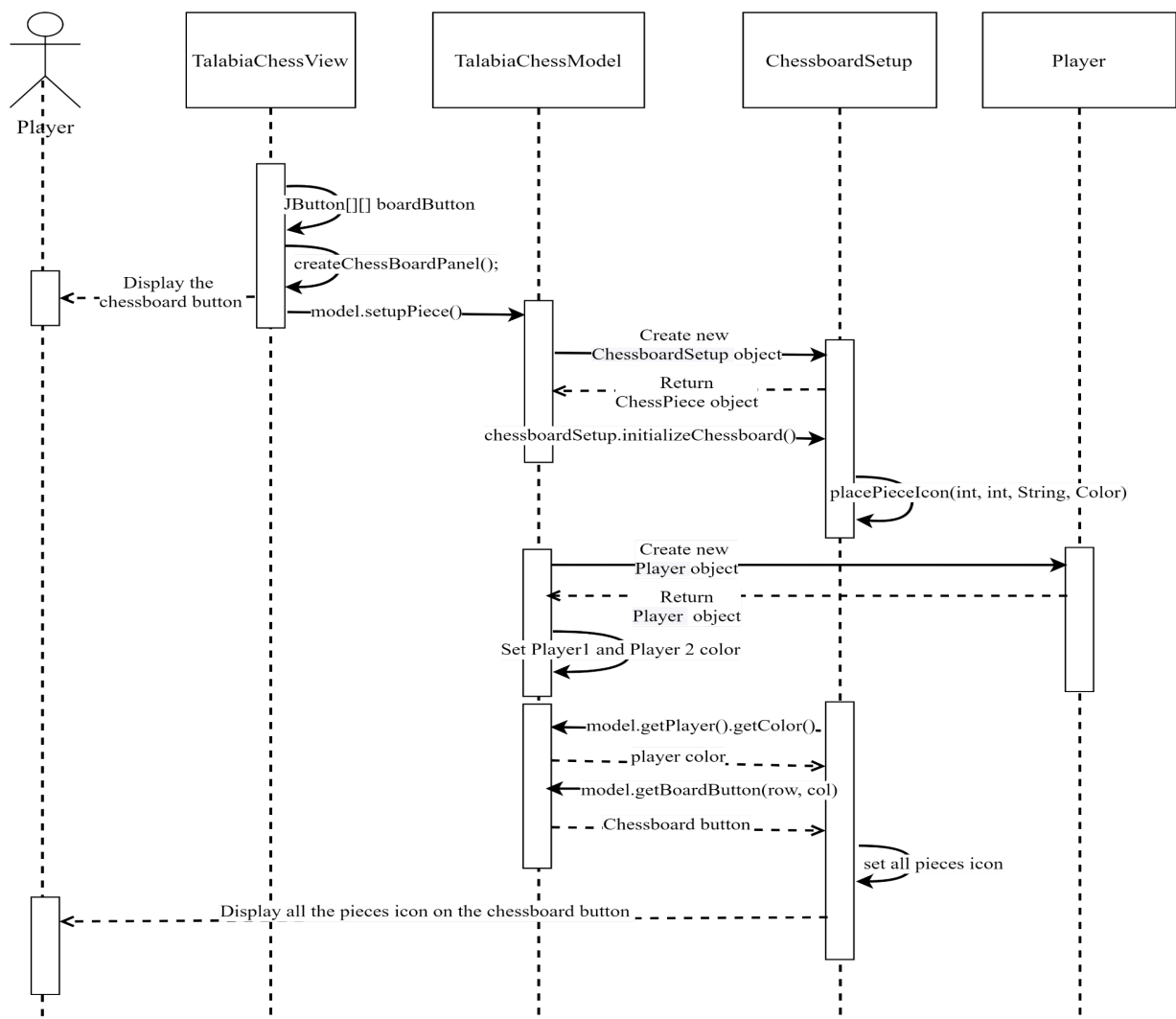
## 5 Use Case Diagram



## 6 Sequence Diagrams

### Sequence Diagram : Initialize Chess Board, Setup Pieces, Pieces Position

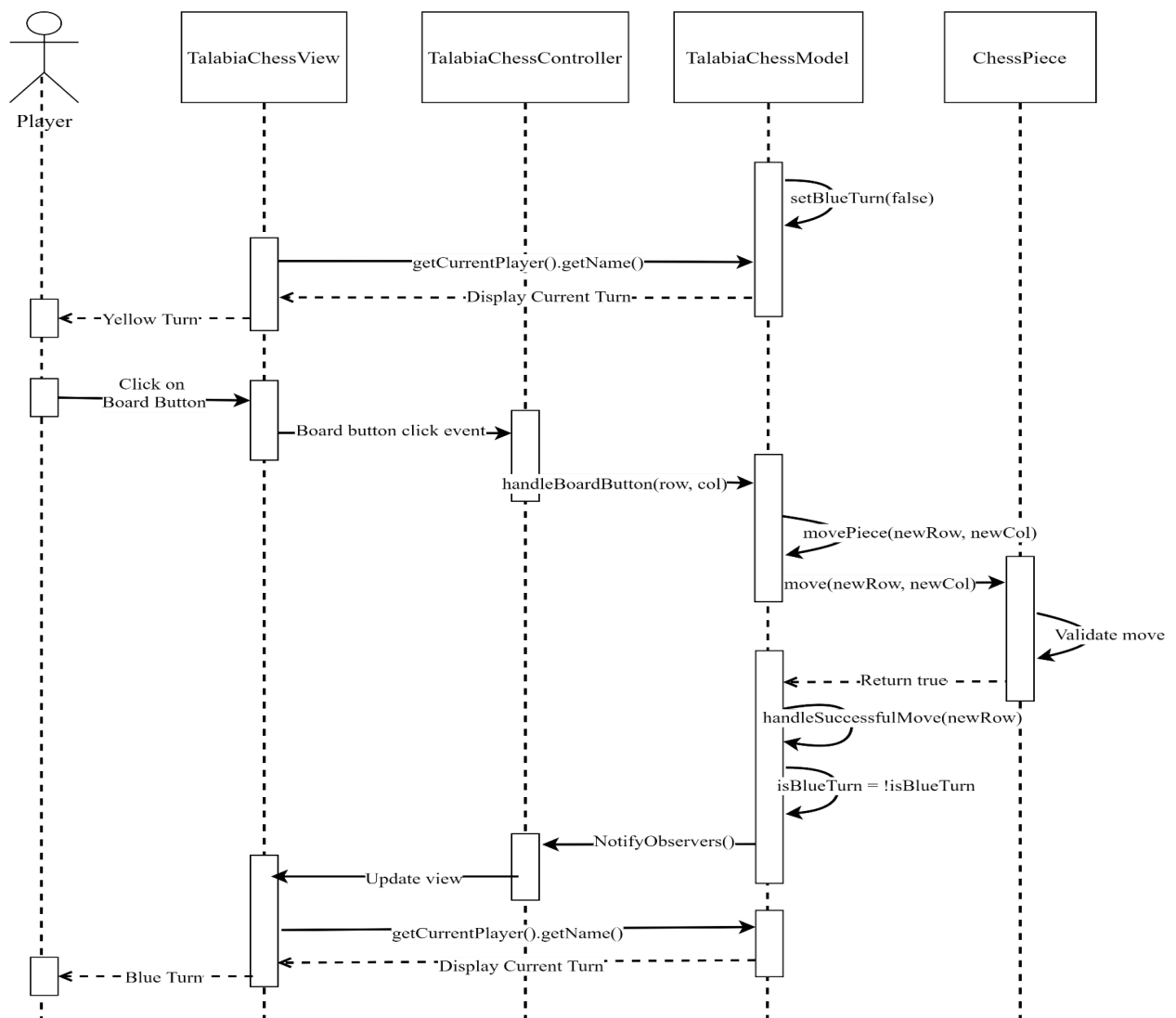
The sequence unfolds with the user interacting with the graphical user interface represented by the TalabiaChessView. Upon clicking on the chessboard buttons, the view triggers the initialization process. The TalabiaChessView initialises the chessboard buttons using the CreateChessBoardPanel() method. This step involves creating the array of interactive buttons (JButton[][] boardButton) that forms the visual representation of the chessboard. Once the initialization is complete, the user is presented with a visually rendered chessboard on the interface, allowing for intuitive gameplay. Moving to the TalabiaChessModel, the setup of chess pieces begins. The model creates an instance of ChessboardSetup to handle the placement and arrangement of chess pieces on the initialised chessboard. This setup process ensures that the board is populated with the appropriate chess pieces, establishing the initial game state.





## Sequence Diagram : Setup Turn, Switch Turn

The sequence begins with the initialization of the boolean variable "isBlueTurn" set to "false". TalabiaChessView utilizes the method "getCurrentPlayer().getName()" to visually present the current player's turn. When a player interacts with the board, clicking on the board button to move a piece, the isBlueTurn variable toggles its state to switch turns to the Yellow player. Following this, the method "notifyObservers()" is invoked, signalling TalabiaChessController to register the view as an observer to the model. Subsequently, TalabiaChessView updates its display by calling "getCurrentPlayer().getName()" again, ensuring that the current turn is accurately reflected for the player. The sequence continues with players taking turns, selecting pieces, and making moves until the game concludes.

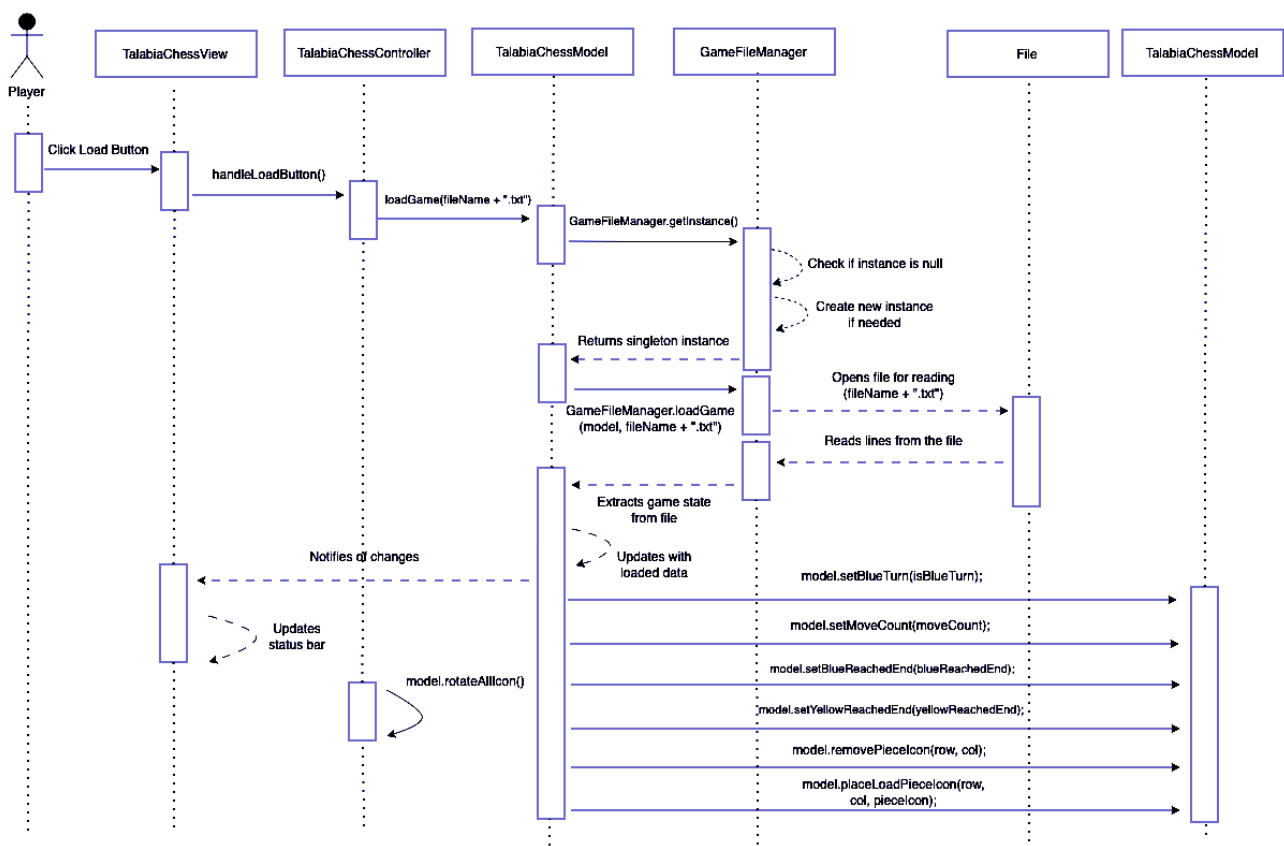


In TalabiaChessView, a board button click triggers communication to TalabiaChessController, which then informs TalabiaChessModel to execute the selectPiece method for piece selection. TalabiaChessModel creates a ChessPiece object and notifies TalabiaChessController, prompting an update in TalabiaChessView to display the selected piece. For user-initiated moves, a second button click in TalabiaChessView is communicated to TalabiaChessController. The controller directs TalabiaChessModel to execute the movePiece method. Within this method, ChessPiece validates the move and updates the position of the moved piece by invoking a method in ChessboardSetup. ChessboardSetup, in turn, employs various checks, such as isPathClear and isDestinationValid, to ensure the validity of the move. If the move is valid, TalabiaChessModel handles the successful move and provides user feedback. Validation failures result in appropriate error messages.



## Sequence Diagram : Load Game

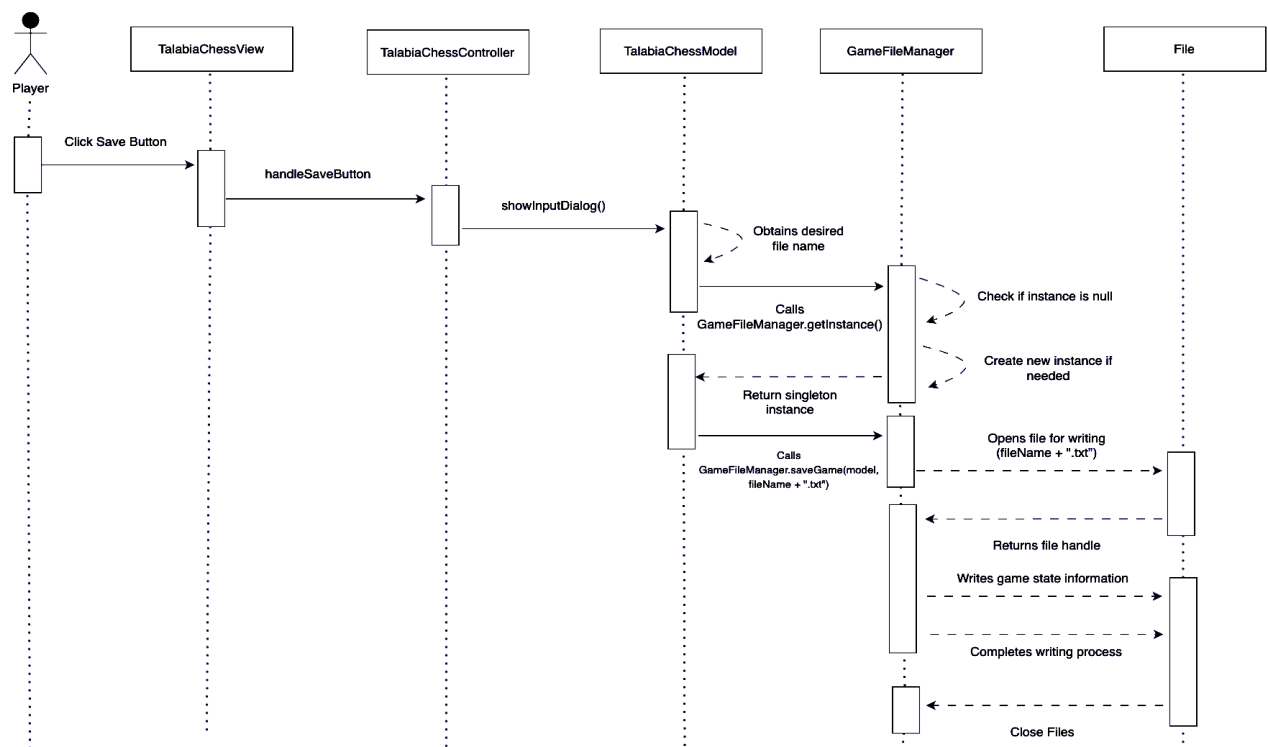
According to the sequence diagram, the TalabiaChessController calls the `handleLoadButton()` function in response to the user clicking the 'Load' button on the TalabiaChessView. The TalabiaChessModel and the controller communicate inside of `handleLoadButton()`. In order to guarantee that there is only one instance for controlling game file operations, the TalabiaChessModel subsequently initialises the GameFileManager singleton using `GameFileManager.getInstance()`. The GameFileManager `loadGame()` function is then called via the `model.loadGame(fileName + ".txt")` call. By reading the given file, this technique extracts pertinent game state data, such as the player's turn, move count, achieved end lists, and piece placements. This loaded data is then used to update the TalabiaChessModel, which in turn updates the TalabiaChessView, which refreshes the status bar and other UI elements. The model also uses `model.rotateAllIcon()` to rotate every icon on the chessboard if necessary. This intricate series of events demonstrates the cooperation between several components in loading a game state, from user interaction to internal model operations and file management.



## Sequence Diagram : Save Game

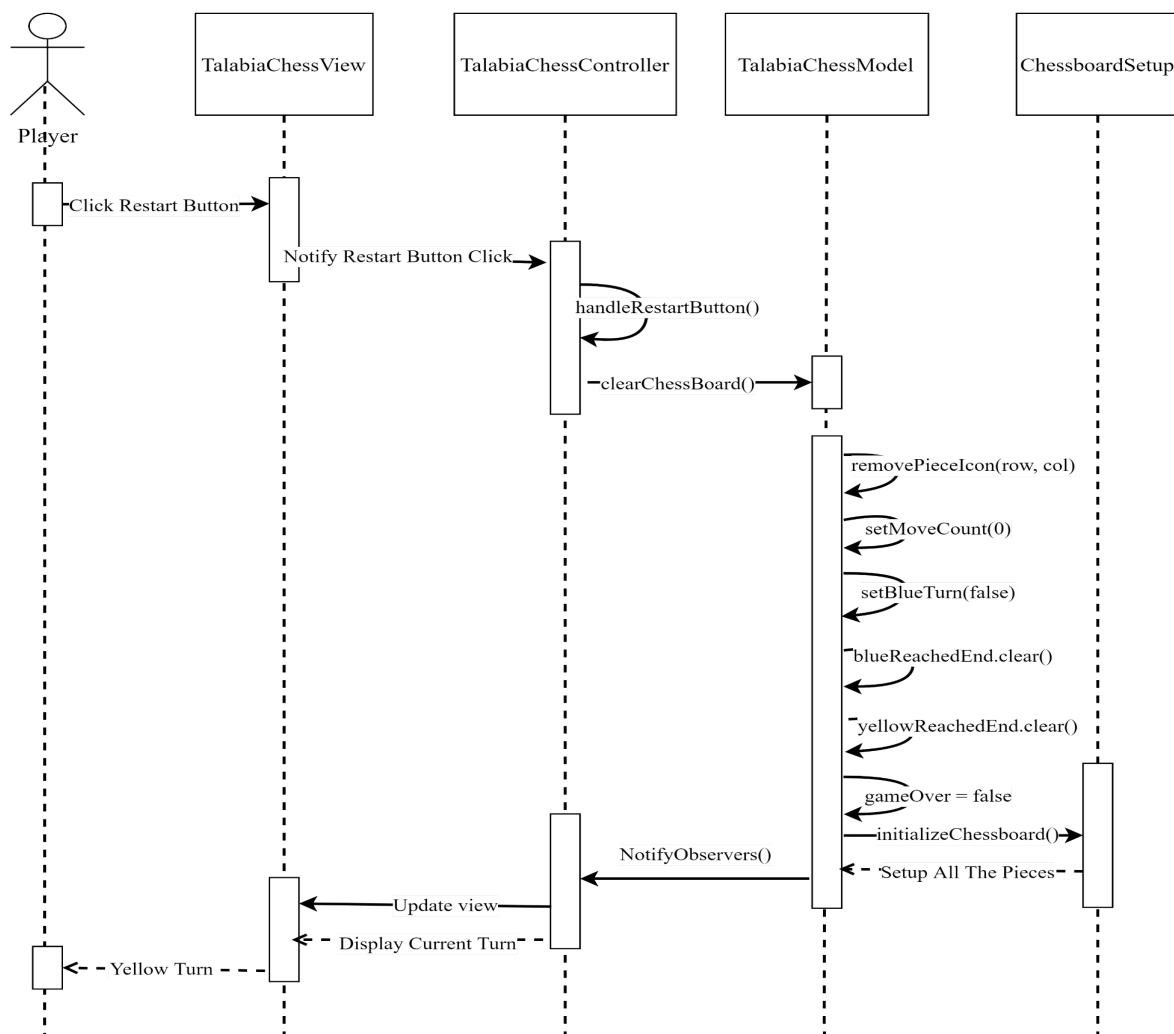
This sequence depicts how to save a game in the TalabiaChess programme. The TalabiaChessView's 'Save' button is clicked by the user to start it. In response, the TalabiaChessController calls `handleSaveButton()`. This causes the TalabiaChessModel to call `showInputDialog()`, which asks the user to enter the name of the desired file. After that, the TalabiaChessModel talks to the GameFileManager singleton to confirm its presence and get the instance. The GameFileManager, which is in charge of handling file activities, verifies and, if required, produces a new instance before giving it back to the TalabiaChessModel. The GameFileManager is called by the TalabiaChessModel launching the saving process with `saveGame(model, fileName + ".txt")`.

In order to communicate with the file system, the GameFileManager opens the given file for writing (`fileName + ".txt"`). It writes the game state data, completes the writing operation, and then shuts the file after getting the file handle. If more modifications are required, the TalabiaChessModel makes sure the game state is changed accordingly. At last, the TalabiaChessController notifies the TalabiaChessView that button click handling is finished. This flow covers all of the activities that take place when a game is saved, including file management, controller actions, model modifications, and user input.



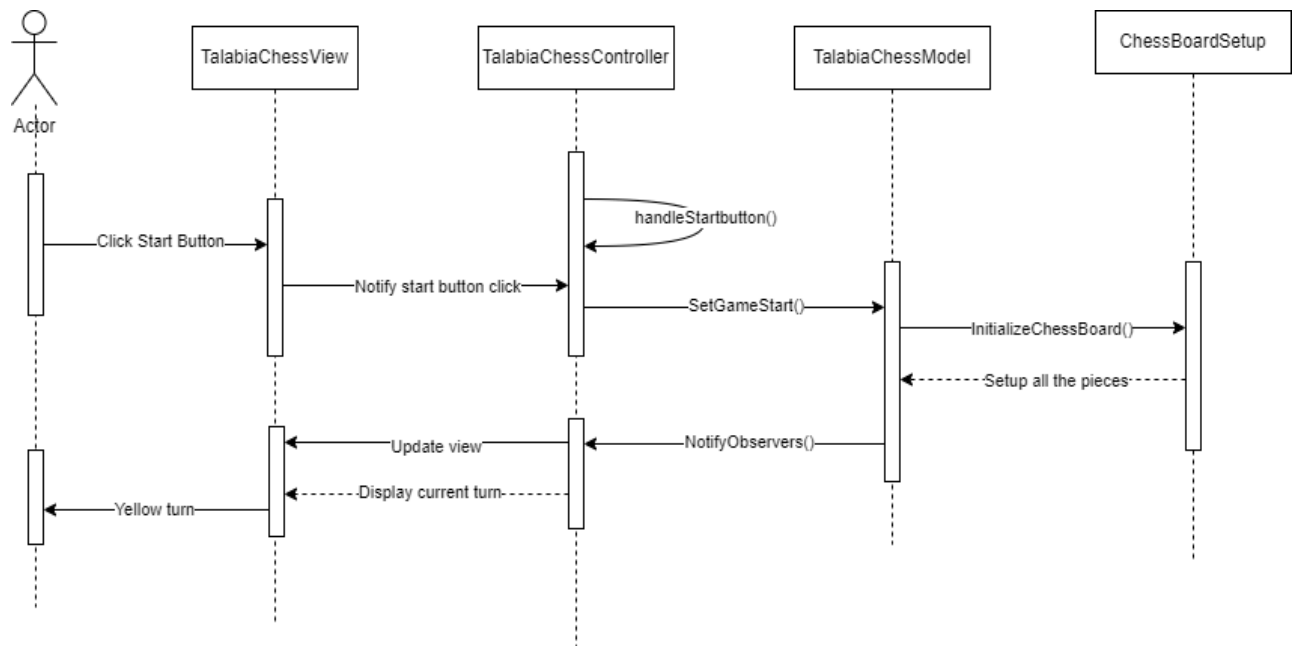
## Sequence Diagram : Restart Game

The user clicks the "Restart" button in the TalabiaChessView interface to begin the process of restarting the game in the Talabia Chess application. An application communication flow is started by this user action. When a button is clicked, the TalabiaChessController, which manages user input, detects it and calls the `handleRestartButton` function. The controller then sends a message to the TalabiaChessModel telling it to start the `clearChessboard` function. In response, the TalabiaChessModel moves all of the pieces off the chessboard and resets important game parameters like turn and move count. The model alerts all of its observers including the TalabiaChessController after this operation. The controller updates the TalabiaChessView as an observer to show the cleared chessboard graphically. The TalabiaChessView, in turn, provides user feedback, signalling the successful restart through status messages or visual indicators. This coordinated sequence ensures a seamless restart of the Talabia Chess game, involving interactions between the user interface, controller, and model components.



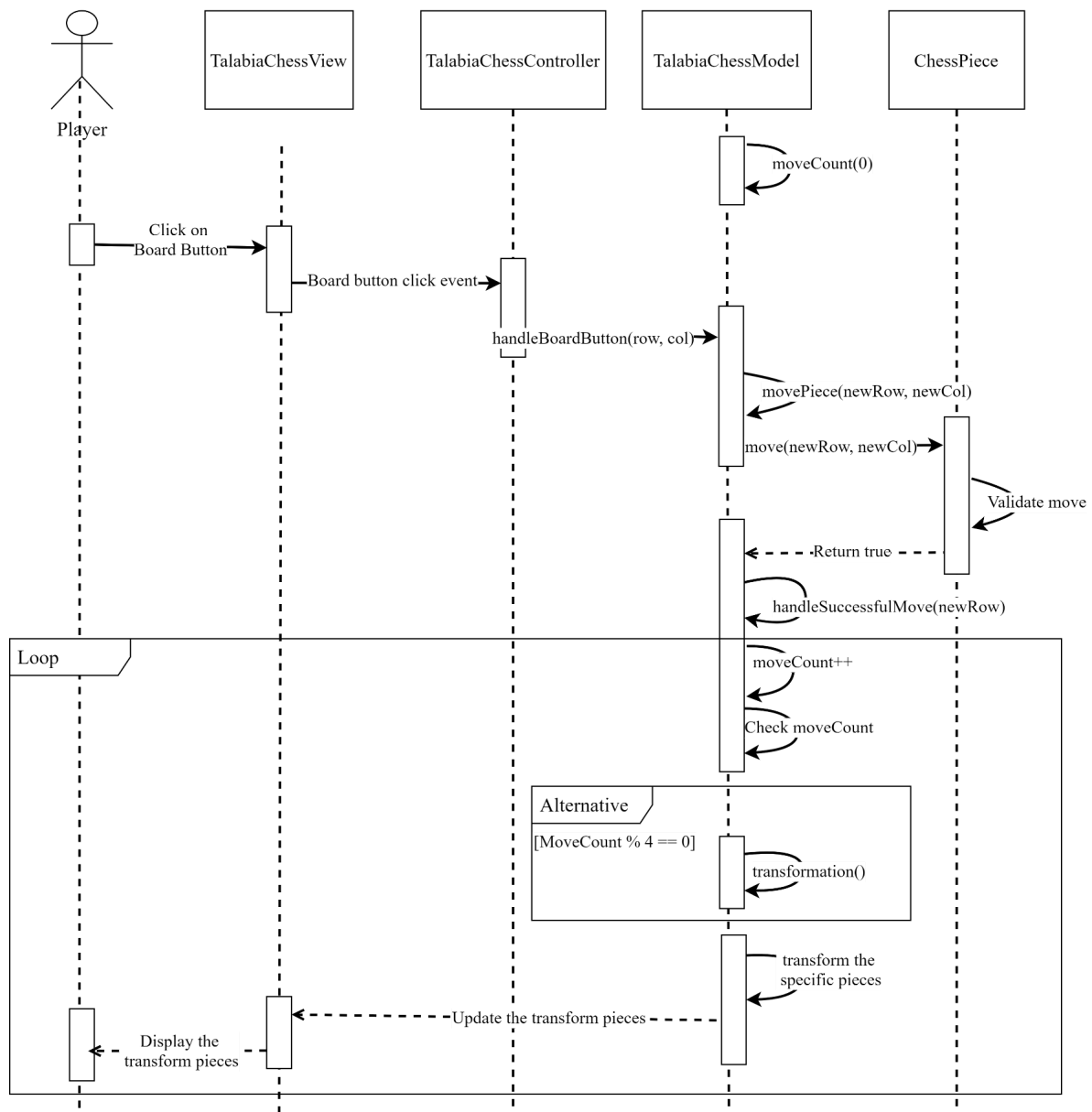
## Sequence Diagram : Start Game

The sequence diagram displays the flow of actions initiated by a user within a chess game application. When the user clicks the "Start Button," the TalabiaChessView recognizes this action and informs the TalabiaChessController. The controller then handles this event by invoking the `handleStartButton()` method, which in turn signals the TalabiaChessModel to set the game as started using the `setGameStart()` method. Once the game state is set, the TalabiaChessModel calls upon the ChessBoardSetup to initialize the chessboard through the `initializeChessboard()` method. The pieces are then set up on the chessboard. After the initial setup is complete, the TalabiaChessModel updates all observers of the change in game state, which triggers the TalabiaChessView to update the display and show the current turn, indicating the game has started and it is now the Yellow player's turn.

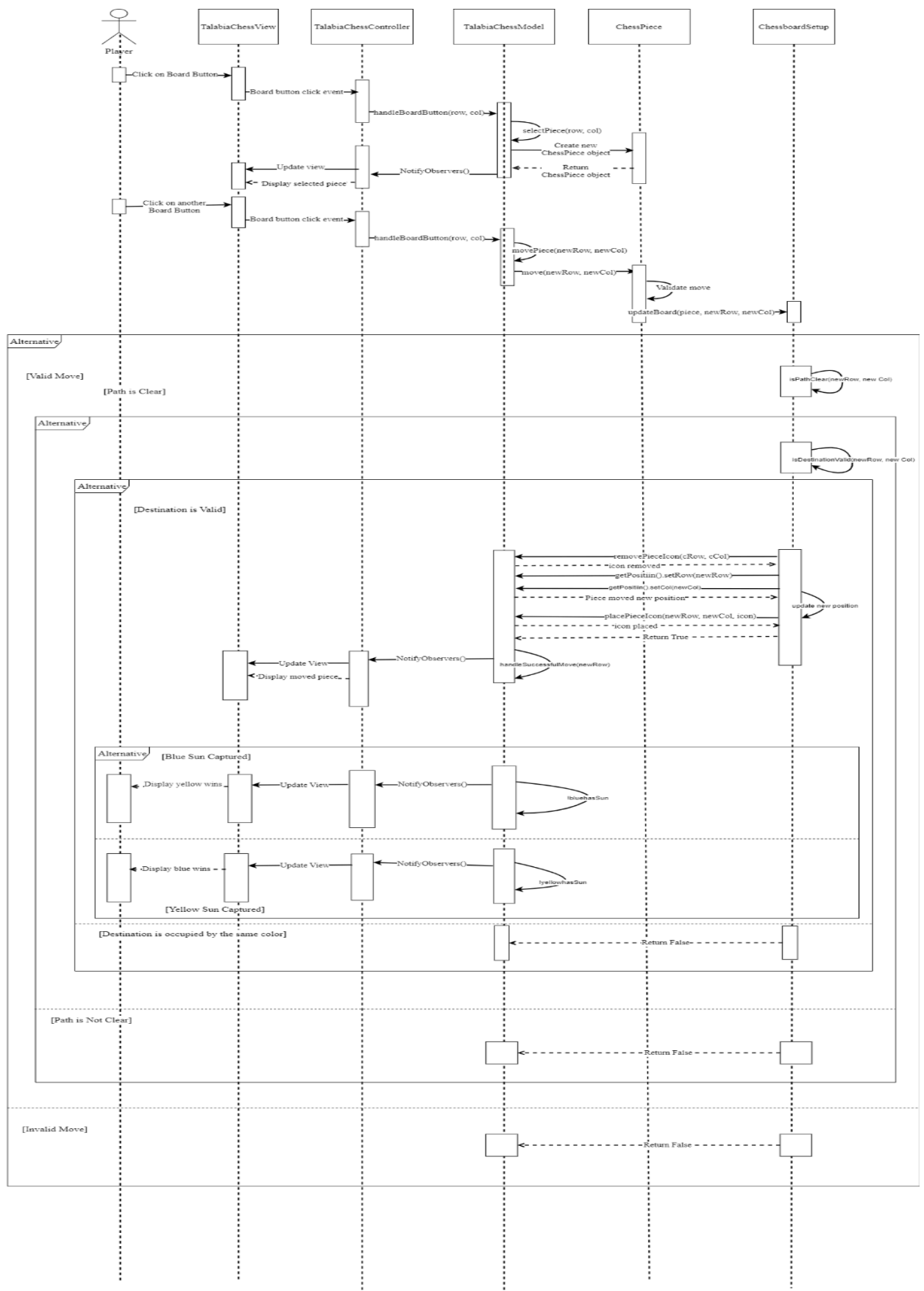


## Sequence Diagram : Transform Pieces

This sequence diagram depicts the transformation of chess pieces in Talabia Chess. Initially, the move count is set to 0. When a user clicks a board button, it triggers the handling of the event through the Talabia Chess Controller and Model. After a successful move, the move count is incremented, and if the move count is divisible by 4, indicating every fourth move, a transformation process occurs. The specific pieces undergo transformation, and the updated pieces are displayed to the user.



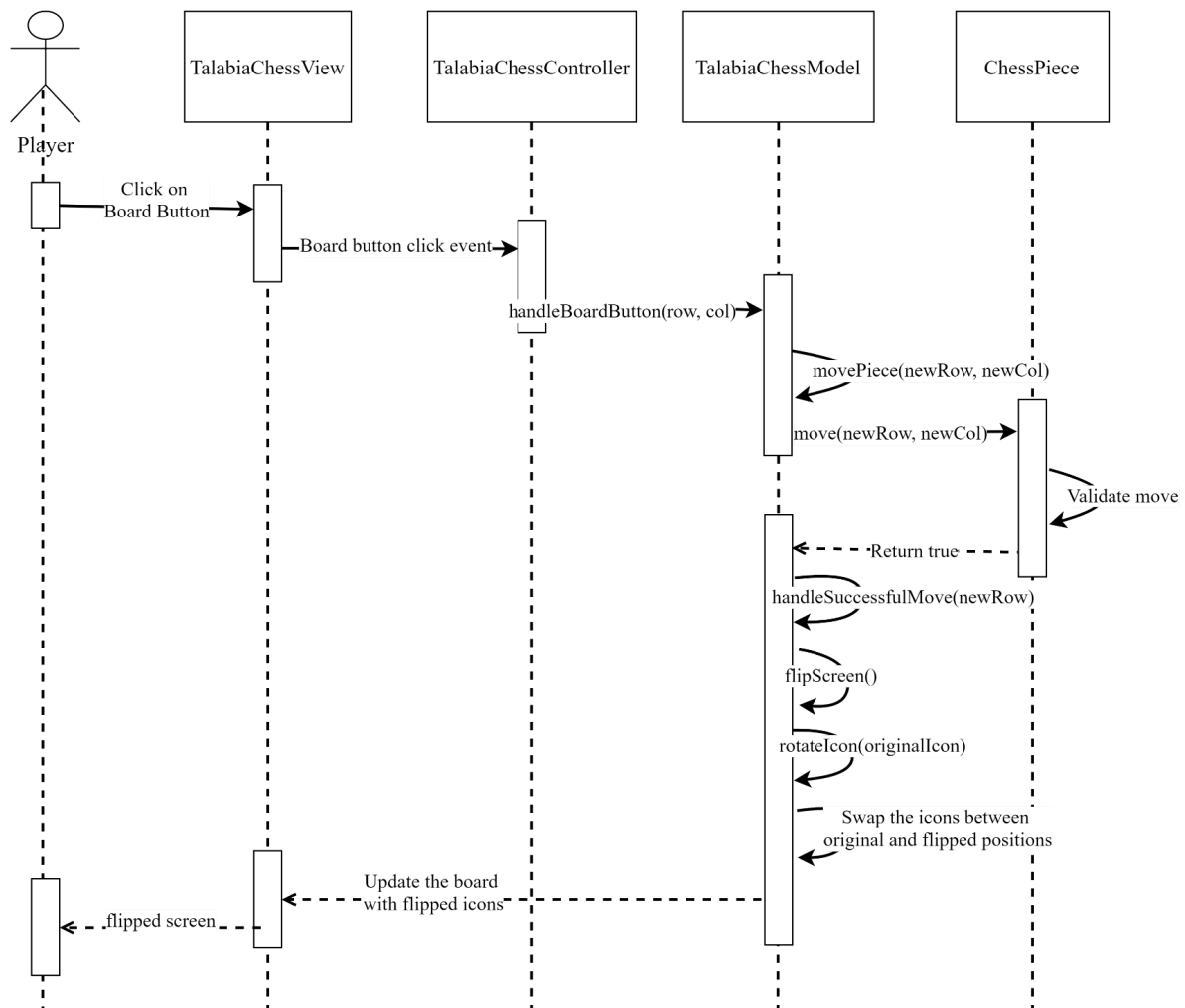
### Sequence Diagram : Determine Winner





## Sequence Diagram : Rotate Chess Board

The sequence diagram illustrates the rotation of the chessboard in response to a user clicking on a board button. The process involves the user's action triggering events in TalabiaChessController and TalabiaChessModel. After a successful move validation, TalabiaChessModel handles the move by flipping the screen and rotating the icons. The updated board with flipped icons is then displayed to the user in TalabiaChessView.



## 7 User Documentation

### Functional requirements

- Start - Click the start button to start the game by selecting pieces to move
- Save - Save the current game state in a .txt file.
- Restart - Restart the game to its original state.
- Load - Load a previously saved game from a .txt file onto the board.
- Select and Move Pieces - To move a piece, first, click the button corresponding to the piece you wish to move, then click the destination button for the new position. If you change your selection, double-click the button for the new piece you want to move. For instance, if you initially select a point piece but decide to move an hourglass piece instead, double-click the hourglass piece's position button to confirm your new selection.
- How to Win The Game - The player who captures their opponent's sun piece wins the game.

## 8 Team Member Contribution

### Coding Part

Ku Jing Hao	Design the entire codebase with Model, View, and Controller, incorporating two design patterns: the Observer pattern and the Singleton pattern.  Point Piece Movement, including reach the end position and turn-around Rotate The ChessBoard (flipping the screen when it is the other player's turn) Initialize Chess Board, Setup Pieces Resizable Windows Save Function Load Function Start Function Transform Pieces Set the Winning Method
Ng Min Hoong	Hourglass Piece Movement Restart Function
See Jian Man	Plus Piece Movement
Wan Aqel Hakimi Bin Mohd Zamri	Time Piece Movement
Tam Yu Heng	Sun Piece Movement Set the Winning Method

### Report Part

Ku Jing Hao	Wrote the compile and run instructions, Talabia Chess Rule, User Documentation  Use Case Diagram, Class Diagram, Sequence Diagram (Select Pieces, Move Pieces, Validate Move, Validate Path, Update board), Sequence Diagram (Setup Turn), Design the Sequence Diagram ( Initialize Chess Board, Setup Pieces, Pieces Position)
Ng Min Hoong	Sequence Diagram (Restart)
See Jian Man	Sequence Diagram (Transform Pieces) , Sequence Diagram (Rotate Chess Board)
Wan Aqel Hakimi Bin Mohd Zamri	Sequence Diagram (Determine Winner), Sequence Diagram (Start),
Tam Yu Heng	Sequence Diagram (Save), Sequence Diagram (Load)

## Coding Method

Ku Jing Hao	<p><b>TalabiaChessModel Class</b></p> <pre>public void setupPiece() public void saveGame(String fileName) public void loadGame(String fileName) public void selectPiece(int row, int col) public void movePiece(int newRow, int newCol) private void handleSuccessfulMove(int newRow) public void rotatePointIcon() public void rotateAllIcon() public void flipScreen() private ImageIcon rotateIcon(ImageIcon originalIcon) private void transformation() private void determineWinner() public List&lt;Integer&gt; getBlueReachedEnd() public List&lt;Integer&gt; getYellowReachedEnd() public void setBlueReachedEnd(List&lt;Integer&gt; blueReachedEnd) public void setYellowReachedEnd(List&lt;Integer&gt; yellowReachedEnd) public boolean getIsBlueTurn() public Player getCurrentPlayer() public Player getPlayer1() public Player getPlayer2() public void placePieceIcon(int row, int col, String imageName) public void placeLoadPieceIcon(int row, int col, String imageName) public void removePieceIcon(int row, int col) public boolean isPieceIconPresent(int row, int col) public String getPieceIcon(int row, int col) public boolean getGameStart() public void setGameStart(boolean gameStart) public void setBlueTurn(boolean isBlueTurn) public int getMoveCount() public void setMoveCount(int moveCount) public ChessPiece getCurrentPiece() public boolean getGameOver()</pre> <p><b>Abstract ChessPiece Class</b></p> <pre>public ChessPiece(int row, int col) public Position getPosition() protected boolean updateMove(int newRow, int newCol) public abstract boolean move(int newRow, int newCol) public abstract String pieceIcon()</pre> <p><b>TalabiaChessController Class</b></p> <pre>public TalabiaChessController(TalabiaChessModel model, TalabiaChessView view) private void initializeButtonsListeners() private void handleStartButton() private void handleSaveButton() private void handleLoadButton() private void handleBoardButton(int row, int col)</pre>
-------------	--

	<p><b>TalabiaChessView Class</b>  public TalabiaChessView()  private void createButtonPanel()  private void createChessBoardPanel()  @Override public void update()  public static JButton getBoardButton(int row, int col)  public JButton getStartButton()  public JButton getSaveButton()  public JButton getLoadButton()  public JButton getRestartButton()</p> <p><b>GameFileManager Class</b>  private GameFileManager()  public static void saveGame(TalabiaChessModel model, String fileName)  public static void loadGame(TalabiaChessModel model, String fileName)</p> <p><b>Player Class</b>  public Player(String name, Color color)  public String getName()  public Color getColor()</p> <p><b>ChessboardSetup Class</b>  public void initializeChessboard()  private void placePieceIcon(int row, int col, String pieceType, Color color)  public boolean updateBoard(ChessPiece piece, int newRow, int newCol)  public boolean isDestinationValid(int newRow, int newCol)  public boolean isPathClear(ChessPiece piece, int newRow, int newCol)</p> <p><b>Position Class</b>  public Position(int row, int col)  public int getRow()  public void setRow(int row)  public int getCol()  public void setCol(int col)</p> <p><b>Point Class</b>  Every method in point class</p>
Ng Min Hoong	<p><b>TalabiaChessModel Class</b>  public void clearChessboard()</p> <p><b>TalabiaChessController Class</b>  private void handleRestartButton()</p> <p><b>Hourglass Class</b>  Every method in hourglass class</p> <p><b>ChessboardSetup Class</b>  public boolean isDestinationValid(int newRow, int newCol)</p>

See Jian Man	<b>Plus Class</b> Every method in plus class
Wan Aqel Hakimi Bin Mohd Zamri	<b>Time Class</b> Every method in time class
Tam Yu Heng	<b>TalabiaChessModel Class</b> private void determineWinner()  <b>Sun Class</b> Every method in sun class