TCP2201 Project

Trimester 2310

by GROUP OMAN

Team Leader: **MUHAMMAD FARID BIN JAYATAN**, 011-20975650,
1211103085@student.mmu.edu.my

Team members:
JACELYN LIM CHIA EE, 016-5991730,
1221303379@student.mmu.edu.my
OO JIA QIAN , 012-7875174,
1221303137@student.mmu.edu.my
YONG RUI CHEN, 019-3531134,
1221303408@student.mmu.edu.my
MUHAMMAD AFIQ BIN FAIDZ, 013-5383499,
1221303455@student.mmu.edu.my

1. Compile and Run Instructions

RUN INSTRUCTIONS

The file that is required to run the program is **Main.java**.

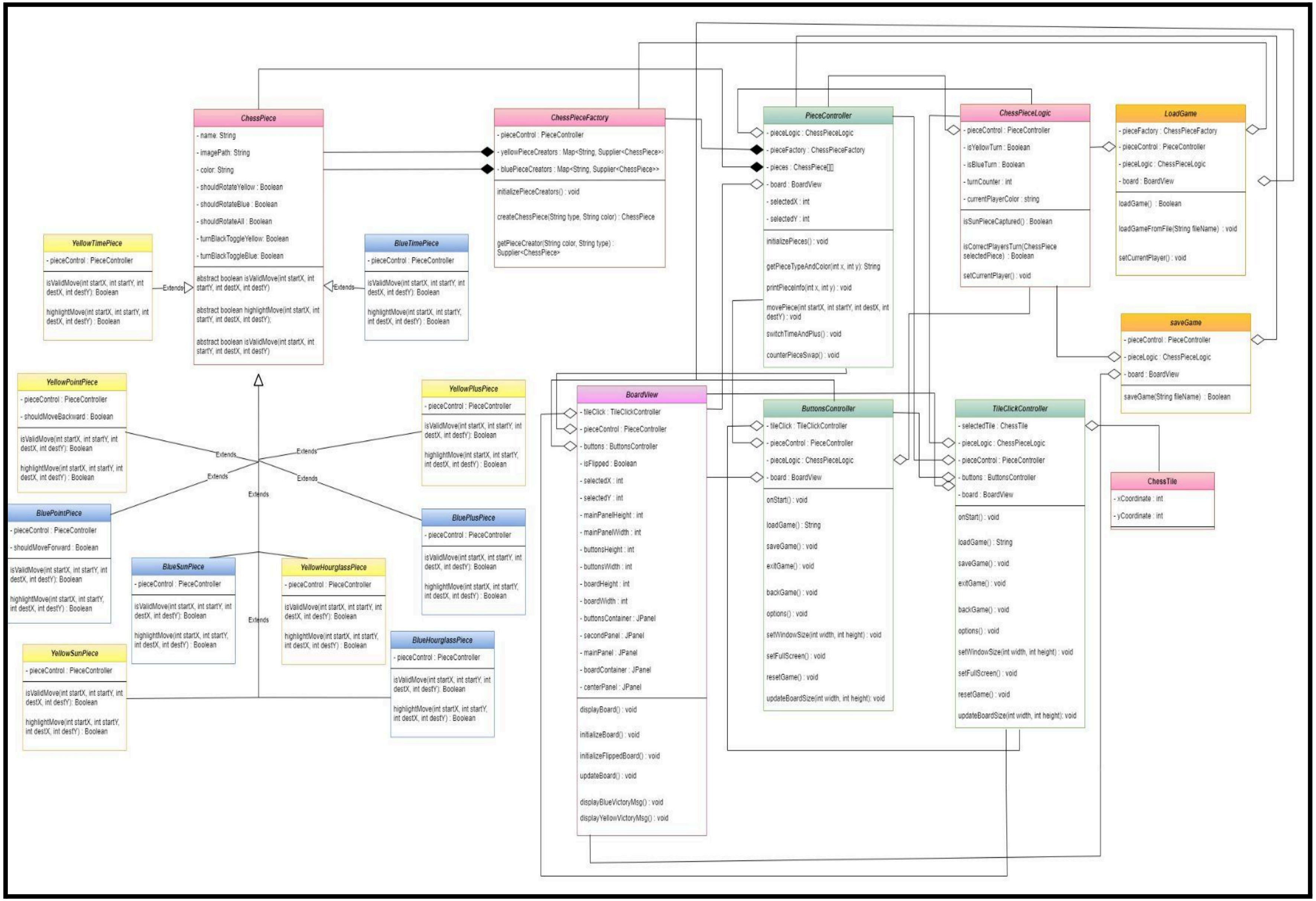
1. Compile the Java source code using javac:

   ```
   javac Main.java
   ```

2. Run the compiled Java program using java:

   ```
   java Main
   ```

2. UML CLASS DIAGRAM


ChessPiece
- name: String
- imagePath: String
- color: String
- shouldRotateYellow : Boolean
- shouldRotateBlue : Boolean
- shouldRotateAll : Boolean
- turnBlackToggleYellow: Boolean
- turnBlackToggleBlue: Boolean
abstract boolean isValidMove(int startX, int startY, int destX, int destY)
abstract boolean highlightMove(int startX, int startY, int destX, int destY);
abstract boolean isValidMove(int startX, int startY, int destX, int destY)
ChessPieceFactory
- pieceControl : PieceController
- yellowPieceCreators : Map<String, Supplier<ChessPiece>>
- bluePieceCreators : Map<String, Supplier<ChessPiece>>
initializePieceCreators() : void
createChessPiece(String type, String color) : ChessPiece
getPieceCreator(String color, String type) : Supplier<ChessPiece>
PieceController
- pieceLogic : ChessPieceLogic
- pieceFactory : ChessPieceFactory
- pieces : ChessPiece[][]
- board : BoardView
- selectedX : int
- selectedY : int
initializePieces() : void
getPieceTypeAndColor(int x, int y): String
printPieceInfo(int x, int y) : void
movePiece(int startX, int startY, int destX, int destY) : void
switchTimeAndPlus() : void
counterPieceSwap() : void
ChessPieceLogic
- pieceControl : PieceController
- isYellowTurn : Boolean
- isBlueTurn : Boolean
- turnCounter : int
- currentPlayerColor : string
isSunPieceCaptured() : Boolean
isCorrectPlayersTurn(ChessPiece selectedPiece) : Boolean
setCurrentPlayer() : void
LoadGame
- pieceFactory : ChessPieceFactory
- pieceControl : PieceController
- pieceLogic : ChessPieceLogic
- board : BoardView
loadGame() : Boolean
loadGameFromFile(String fileName) : void
setCurrentPlayer() : void
saveGame
- pieceControl : PieceController
- pieceLogic : ChessPieceLogic
- board : BoardView
saveGame(String fileName) : Boolean
YellowTimePiece
- pieceControl : PieceController
isValidMove(int startX, int startY, int destX, int destY): Boolean
highlightMove(int startX, int startY, int destX, int destY) : Boolean
Extends
BlueTimePiece
- pieceControl : PieceController
isValidMove(int startX, int startY, int destX, int destY): Boolean
highlightMove(int startX, int startY, int destX, int destY) : Boolean
Extends
YellowPointPiece
- pieceControl : PieceController
- shouldMoveBackward : Boolean
isValidMove(int startX, int startY, int destX, int destY): Boolean
highlightMove(int startX, int startY, int destX, int destY) : Boolean
YellowPlusPiece
- pieceControl : PieceController
isValidMove(int startX, int startY, int destX, int destY): Boolean
highlightMove(int startX, int startY, int destX, int destY) : Boolean
Extends
Extends
Extends
Extends
Extends
BluePointPiece
- pieceControl : PieceController
- shouldMoveForward : Boolean
isValidMove(int startX, int startY, int destX, int destY): Boolean
highlightMove(int startX, int startY, int destX, int destY) : Boolean
BluePlusPiece
- pieceControl : PieceController
isValidMove(int startX, int startY, int destX, int destY): Boolean
highlightMove(int startX, int startY, int destX, int destY) : Boolean
BlueSunPiece
- pieceControl : PieceController
isValidMove(int startX, int startY, int destX, int destY): Boolean
highlightMove(int startX, int startY, int destX, int destY) : Boolean
YellowHourglassPiece
- pieceControl : PieceController
isValidMove(int startX, int startY, int destX, int destY): Boolean
highlightMove(int startX, int startY, int destX, int destY) : Boolean
Extends
YellowSunPiece
- pieceControl : PieceController
isValidMove(int startX, int startY, int destX, int destY): Boolean
highlightMove(int startX, int startY, int destX, int destY) : Boolean
BlueHourglassPiece
- pieceControl : PieceController
isValidMove(int startX, int startY, int destX, int destY): Boolean
highlightMove(int startX, int startY, int destX, int destY) : Boolean
BoardView
- tileClick : TileClickController
- pieceControl : PieceController
- buttons : ButtonsController
- isFlipped : Boolean
- selectedX : int
- selectedY : int
- mainPanelHeight : int
- mainPanelWidth : int
- buttonsHeight : int
- buttonsWidth : int
- boardHeight : int
- boardWidth : int
- buttonsContainer : JPanel
- secondPanel : JPanel
- mainPanel : JPanel
- boardContainer : JPanel
- centerPanel : JPanel
displayBoard() : void
initializeBoard() : void
initializeFlippedBoard() : void
updateBoard() : void
displayBlueVictoryMsg() : void
displayYellowVictoryMsg() : void
ButtonsController
- tileClick : TileClickController
- pieceControl : PieceController
- pieceLogic : ChessPieceLogic
- board : BoardView
onStart() : void
loadGame() : String
saveGame() : void
exitGame() : void
backGame() : void
options() : void
setWindowSize(int width, int height) : void
setFullScreen() : void
resetGame() : void
updateBoardSize(int width, int height): void
TileClickController
- selectedTile : ChessTile
- pieceLogic : ChessPieceLogic
- pieceControl : PieceController
- buttons : ButtonsController
- board : BoardView
onStart() : void
loadGame() : String
saveGame() : void
exitGame() : void
backGame() : void
options() : void
setWindowSize(int width, int height) : void
setFullScreen() : void
resetGame() : void
updateBoardSize(int width, int height): void
ChessTile
- xCoordinate : int
- yCoordinate : int

In the chess application architecture, various chess piece types are represented by distinct classes, each derived from the abstract base class **"ChessPiece"**. These classes include **"YellowTimePiece", "YellowPointPiece", "YellowBluePiece", "YellowSunPiece", "YellowhoursPiece", "BlueHoursPiece", "BlueTimePiece", "BluePointPiece", "BlueBluePiece", and "BlueSunPiece**". This hierarchy is established through abstraction.

The **"ChessPieceFactory"** class plays a central role and is composed of a 1-to-N relationship with instances of **"YellowPieceCreators"** and **"BluePieceCreators"**. It manages the creation of chess pieces through these creator classes.

The **"PieceController"** class, in turn, contains a **"ChessPieceFactory"** object and a **2D array of "ChessPiece"** instances. Additionally, it possesses an object of the **"BoardView"** class, named **"board"**. This relationship demonstrates aggregation.

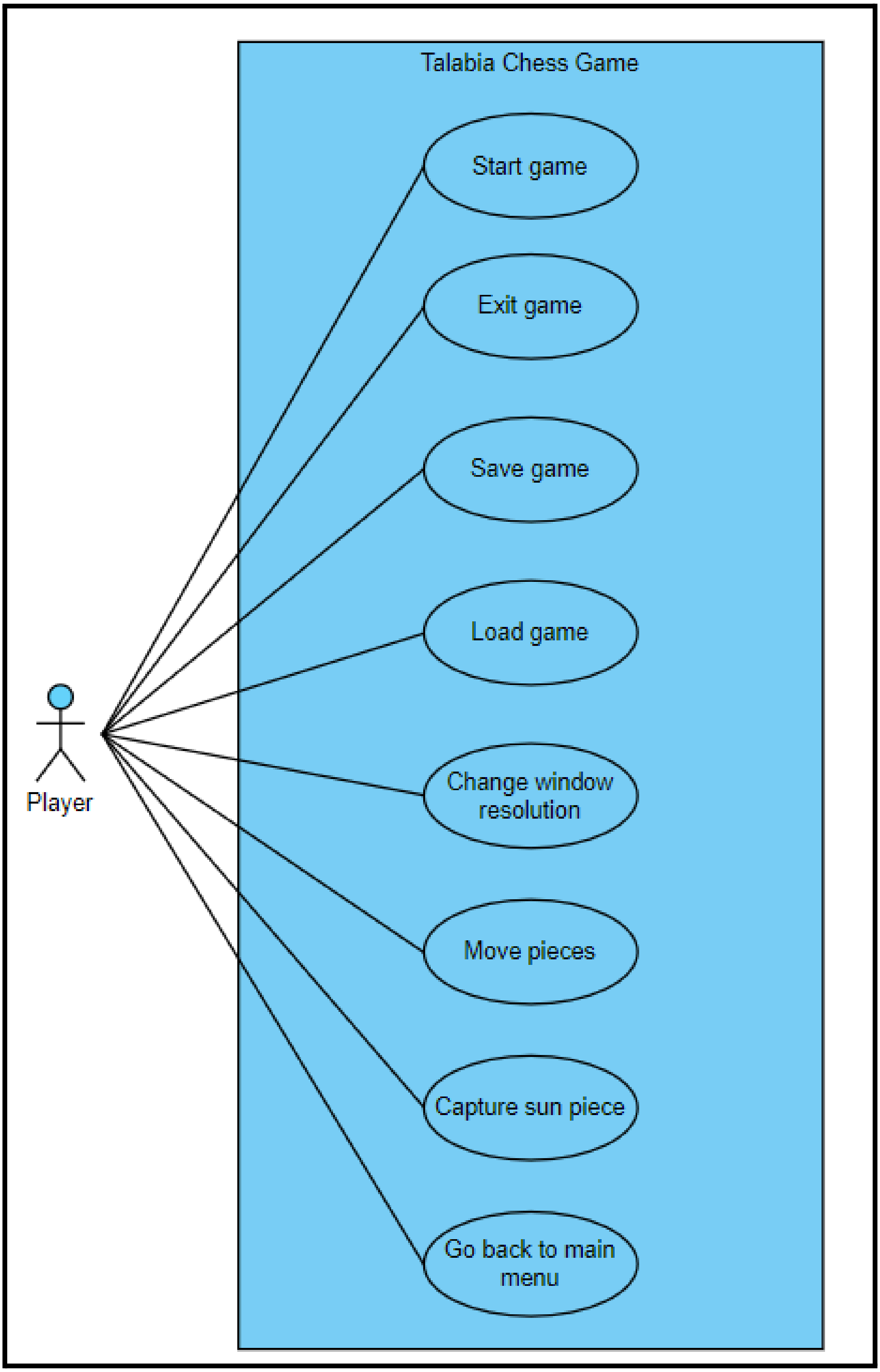
Within the **"BoardView" class**, multiple instances of **"TitleClickController"**, **"PieceController"**, **"ButtonController"**, along with several "int" variables and "JPanel" objects, are encapsulated. The relationships between these components are defined as aggregations.

"TileClickController", "PieceController", "ChessPieceController", and **"BoardView"** share common attributes and exhibit an aggregation relationship with each other. In contrast, **"ButtonController"** holds objects of **"TitleClick", "PieceController",** and **"ChessPieceLogic"** through aggregation and also encompasses instances of these classes.

The **"TitleClickController"** class maintains objects of "**ChessTitle", "ChessPieceLogic", "PieceController", "Button", and "Board"**, establishing aggregation relationships. Furthermore, **"PieceController**" exhibits aggregation relationships with **"ChessPieceFactory", "ChessPieceLogic", "ChessPiece"** (1-to-N relationship), and **"BoardView",** with the class containing objects of these classes.

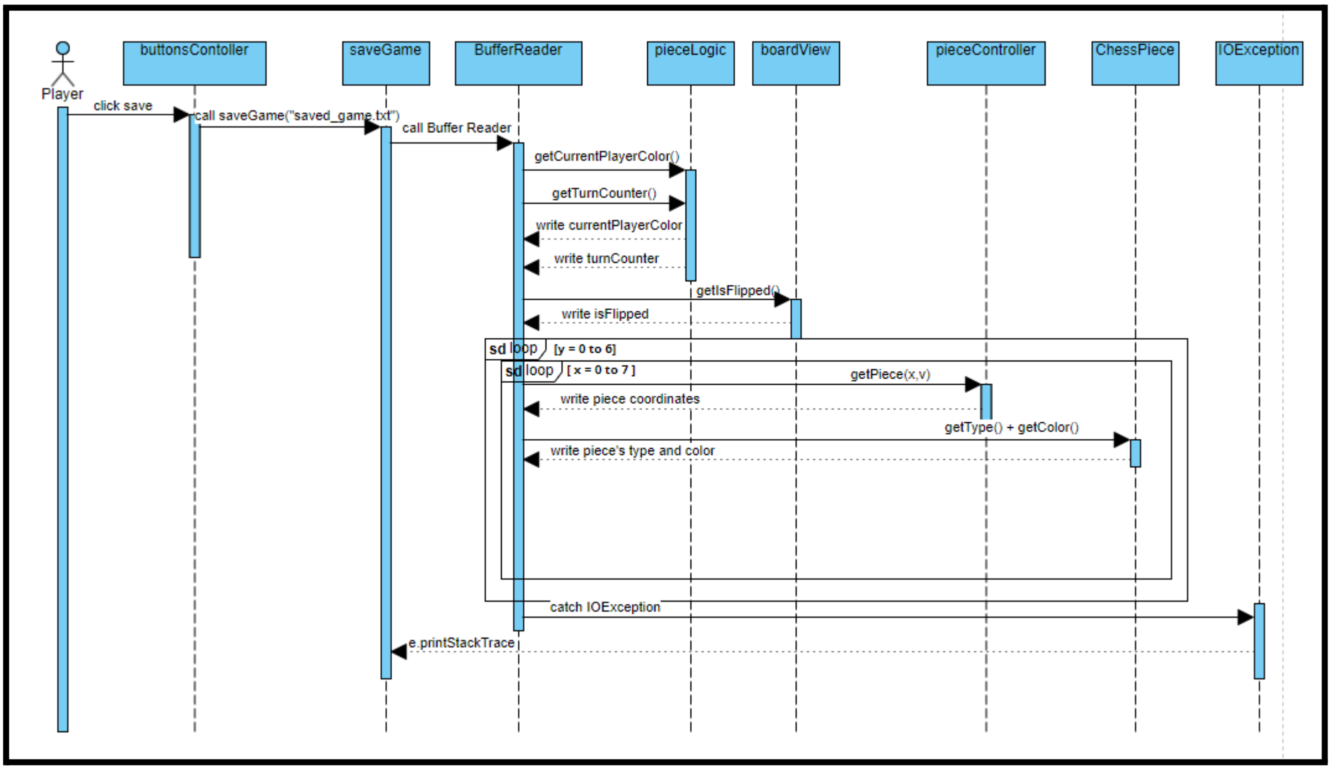
The **"ChessPieceLogic"** class has a one-to-one relationship with **"PieceController"**, signifying a close association between them. Additionally, the **"LoadGame"** class aggregates objects of "**ChessPieceFactory", "PieceController", and "ChessPieceLogic"** through 1-to-1 relationships. Similarly, the **"SaveGame"** class aggregates objects of **"PieceController", "ChessPieceLogic",** and **"BoardView"** through aggregation relationships.

3. USE CASE DIAGRAM

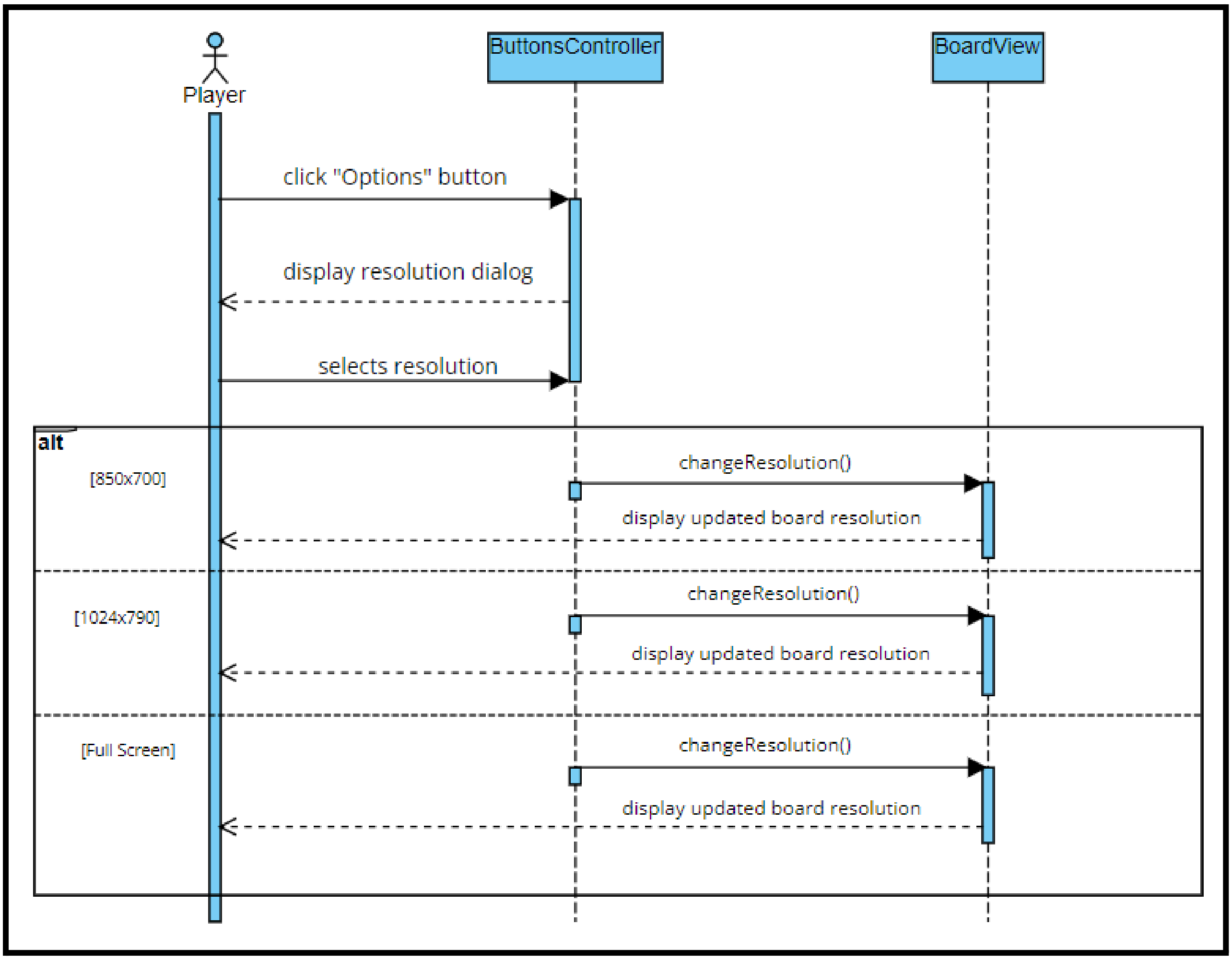


3. SEQUENCE DIAGRAM

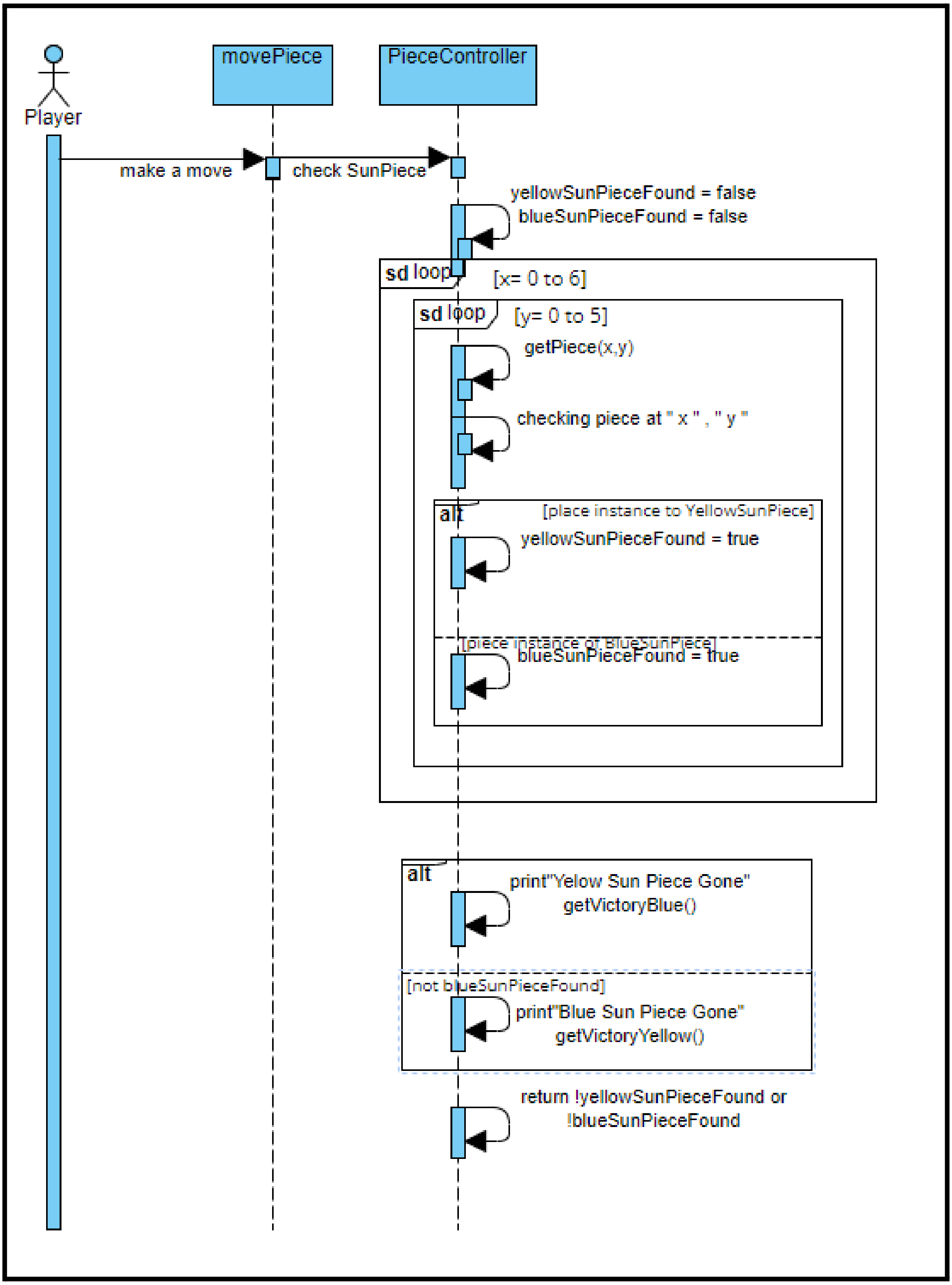
4.1 Save Game



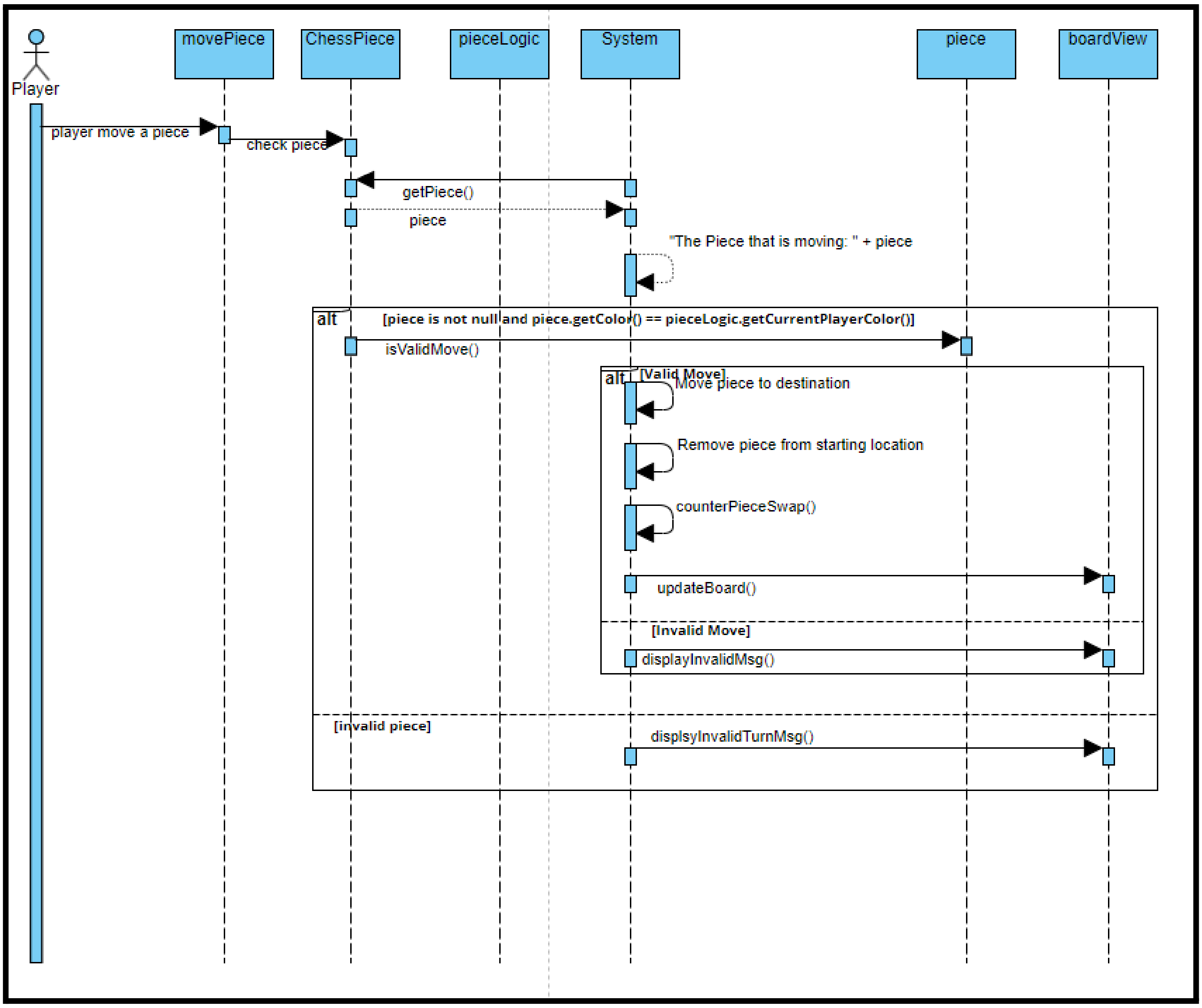
4.2 Change Resolution



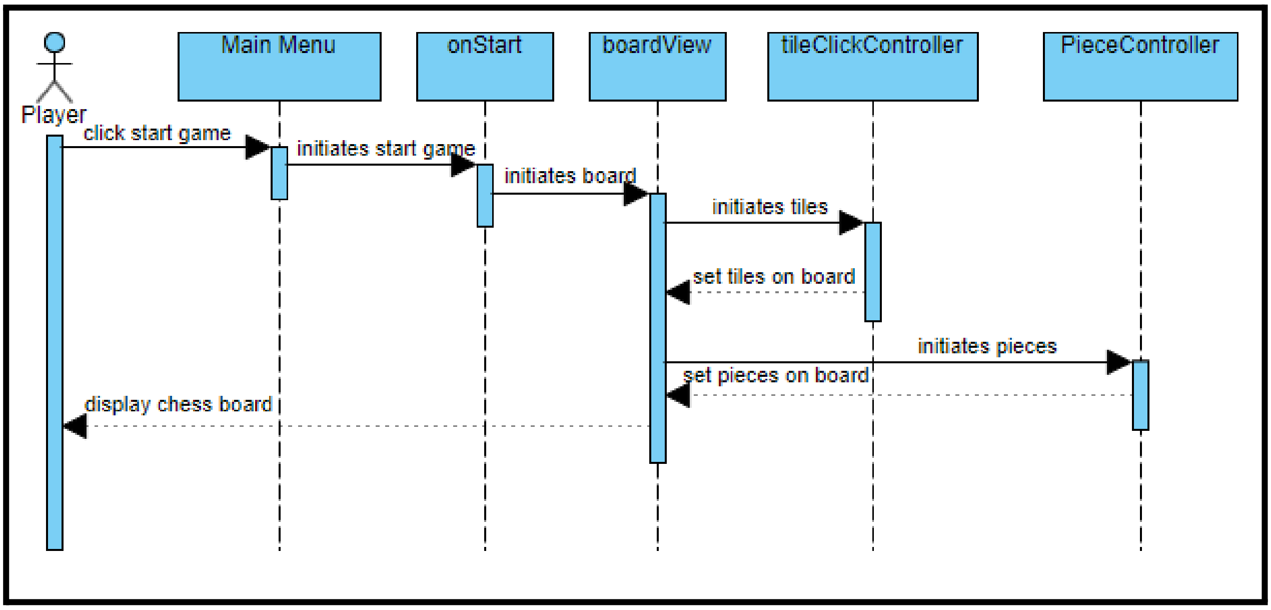
4.3 Capture Sun Piece


Player
movePiece
PieceController
make a move
check SunPiece
yellowSunPieceFound = false
blueSunPieceFound = false
sd loop
[x= 0 to 6]
sd loop
[y= 0 to 5]
getPiece(x,y)
checking piece at " x " , " y "
alt
[place instance to YellowSunPiece]
yellowSunPieceFound = true
[piece instance of BlueSunPiece]
blueSunPieceFound = true
alt
print"Yelow Sun Piece Gone"
getVictoryBlue()
[not blueSunPieceFound]
print"Blue Sun Piece Gone"
getVictoryYellow()
return !yellowSunPieceFound or
!blueSunPieceFound

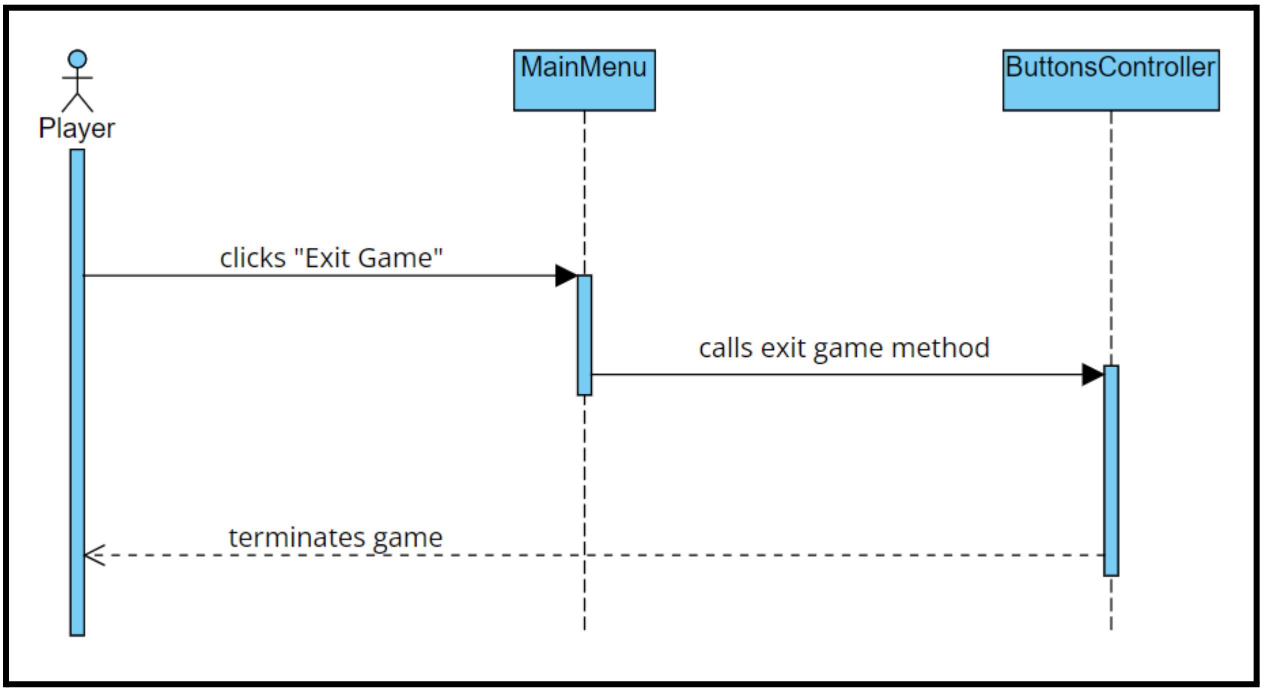
4.4 Move Piece



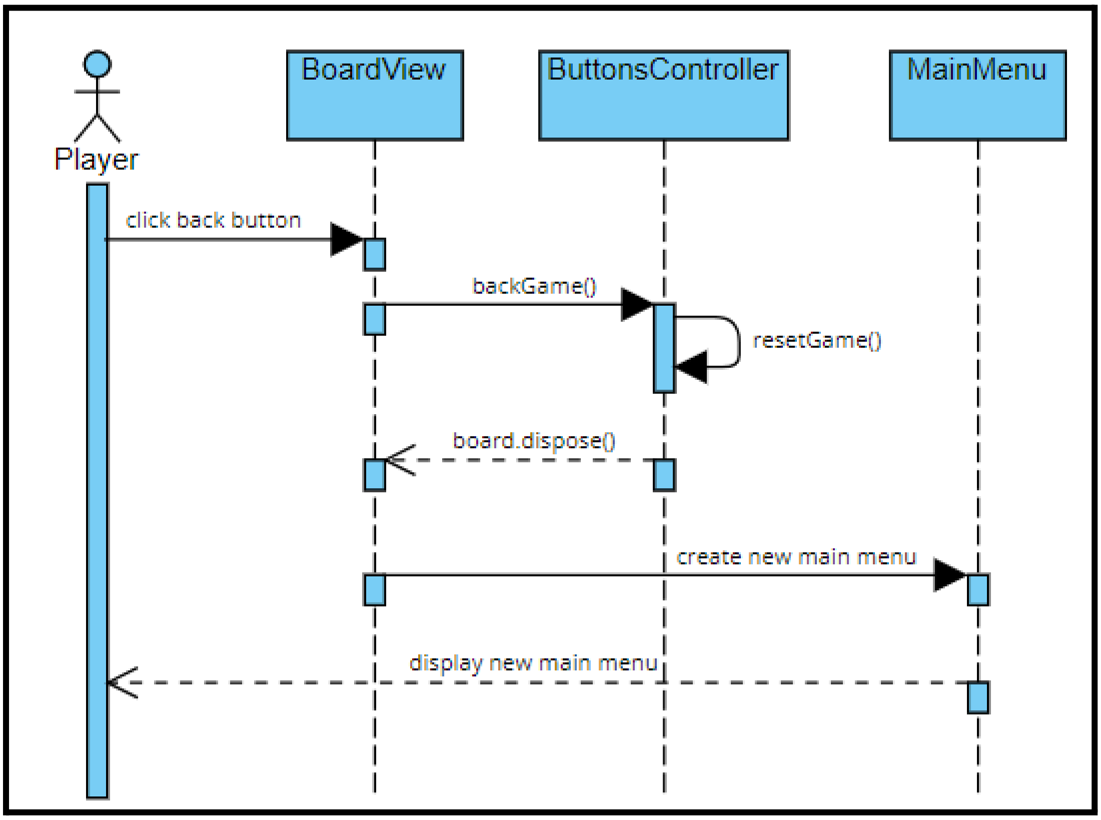
4.5 Start Game



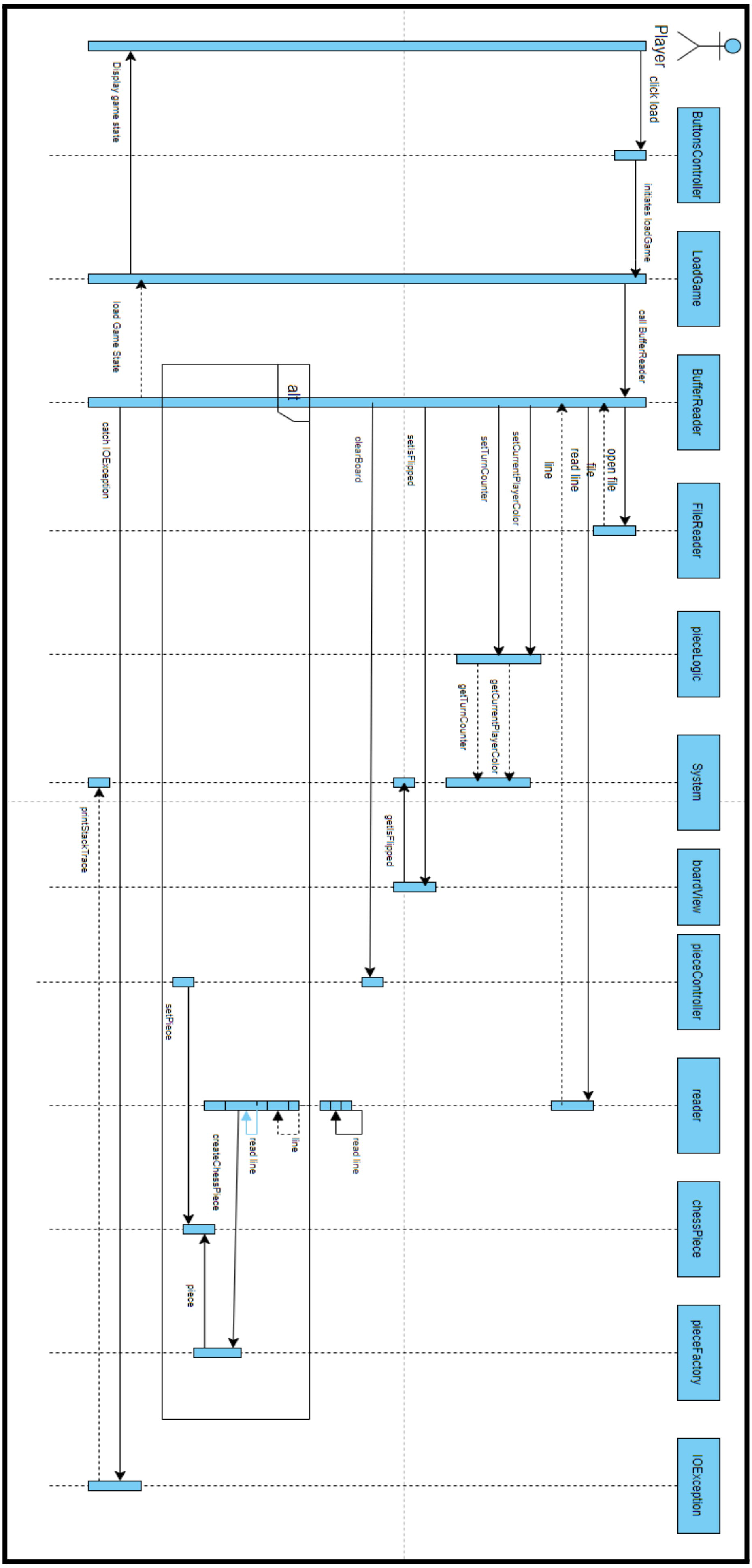
4.6 Exit Game



4.7 Go back to Main Menu



4.8 Load Game


Player
ButtonsController
LoadGame
BufferReader
FileReader
pieceLogic
System
boardView
pieceController
reader
chessPiece
pieceFactory
IOException
click load
initiates loadGame
call BufferReader
open file
file
read line
line
setCurrentPlayerColor
setTurnCounter
getCurrentPlayerColor
getTurnCounter
setIsFlipped
getIsFlipped
clearBoard
alt
read line
line
read line
createChessPiece
piece
setPiece
catch IOException
printStackTrace
load Game State
Display game state

5. User Manual

5.1.1. Introduction

Welcome to Talabia Chess, a unique chess game played on a 7x6 board with distinct pieces and dynamic transformations. This manual will guide you through the installation, gameplay, and advanced features of the game.

5.2. Getting Started

5.2.1 Installation

Install Visual Studio Code and install java in your VS Code.

The Talabia Chess Game file will be given for you to download and run in the VS Code.

5.2.2 Launching the Game

Run the code on VS Code and the Main Menu will pop out before you start the game.

Click “Start Game” button to start playing the Talabia Chess.

5.3. Game Interface

5.3.1 Board Layout

The game board is a 7x6 grid. Each cell represents a unique position where pieces can be placed.

5.3.2 Pieces and Their Abilities

- Point Piece: Moves forward 1 or 2 steps. Turns around at the board's end.
- Hourglass Piece: Moves in a 3x2 L shape, can skip over other pieces.
- Time Piece: Moves diagonally with any distance.
- Plus Piece: Moves horizontally and vertically with any distance.
- Sun Piece: Moves one step in any direction. Game ends when captured.

5.4. Gameplay

5.4.1 Basic Rules

- Follow standard chess rules with the specified movements for each piece.
- Turns alternate between yellow and blue players.
- The game ends when the Sun piece is captured.

5.4.2 Transformations

After 2 turns, all Time pieces transform into Plus pieces, and all Plus pieces transform into Time pieces.

5.4.3 Winning the Game

Capture the opponent's Sun piece to win the game.

5.5. GUI Controls

5.5.1 Menus

The game interface includes menus for exit and saving/loading games. Explore the menus for a user-friendly experience.

5.5.2 Resizable Windows

Adjust the game window size according to your preferences for a comfortable gaming experience.

5.5.3 Flipping the Screen

The screen automatically flips when it's the other player's turn, ensuring a seamless transition between turns.

5.5.4 Save and Load Games

Use the game menu to save your progress. Games are saved in a human-readable text file format. Load saved games from the menu.

5.6. Advanced Features

5.6.1 Design Patterns

The Talabia Chess employs the MVC pattern, emphasizing a clear separation of game logic and GUI code. Object-oriented design concepts, subclassing, delegation, composition, and aggregation are utilized where appropriate.

5.7. Troubleshooting

For any issues or inquiries,please do tell and contact us.

5.8. Credits

Give credit to individuals who contributed to the development of The Talabia Chess.

5.9. Contact Information

Provide contact details for technical support, feedback, and additional information.