Faculty of Computing and Informatics (FCI)

Multimedia University, Cyberjaya

CCP 6224 – OOAD

Trimester 2430

Assignment: Kwazam Chess Report

Section: TC3L

TT9L

Group G

Group Leader:

Mei Yong Peng	011-3150 9211	1211109159

Group Members:

1. Chan Ka Ken	012-273 3660	1211109440
2. Fong Kai Chun	011-5686 8061	1211108430
3. Lo Shia Yang	016-809 8079	1211108901

Table of Contents

Chapter 1: Instructions

1.1 How to Compile Program 3

1.1.1 Command Prompt 3

1.1.2 Visual Studio Code 4

Chapter 2: Requirements

2.1 Requirements 5

2.2 Non-Requirements 6

Chapter 3: UML Class Diagram

3.1 Class Diagram 7

Chapter 4: Use Case Diagram

4.1 Use Case Diagram 10

Chapter 5: Sequence Diagram

5.1 Sequence Diagram 11

Chapter 6: User Manual

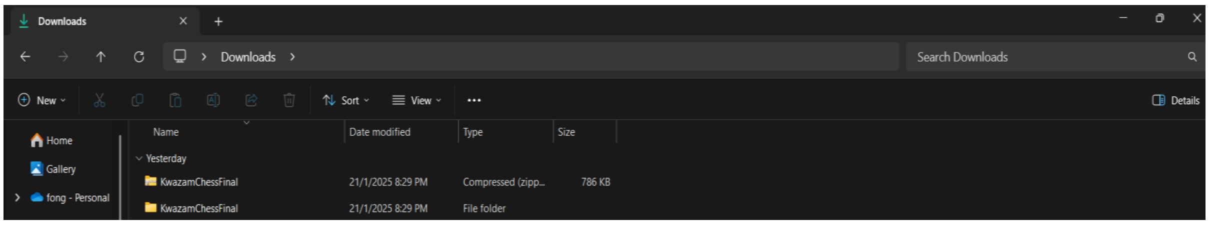
6.1 User Manual Table 13

Chapter 1: Instructions

1.1 How to Compile Program

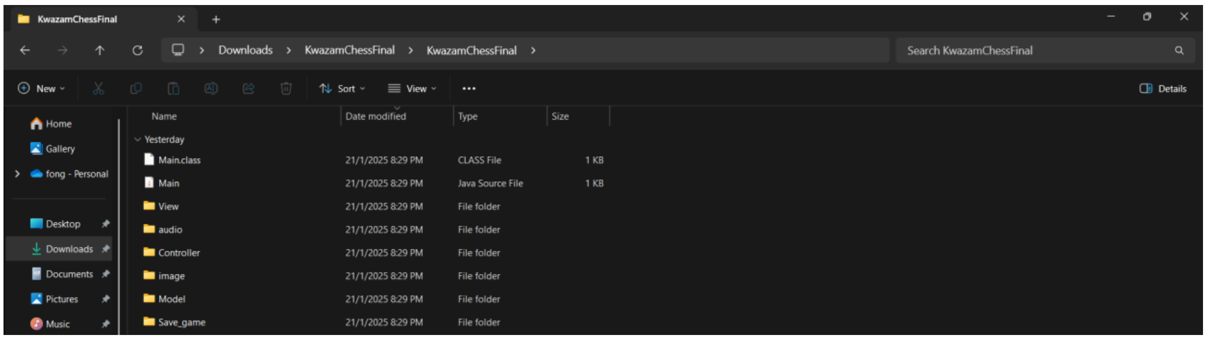
1.1.1 Command Prompt

1. First, download “KwazamChessFinal” zip file. Locate the file and extract it. After extracted our file, you will see a folder named “KwazamChessFinal” inside your selected directory as the image shown below.



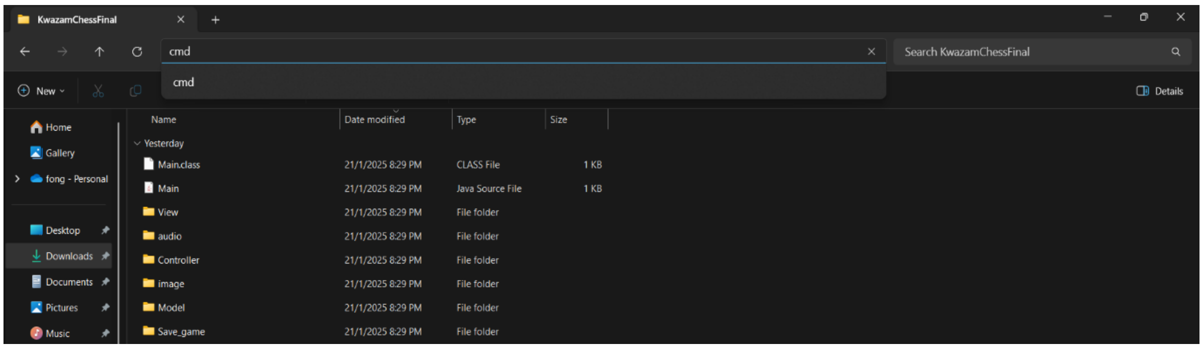
Name	Date modified	Type	Size
KwazamChessFinal	21/1/2025 8:29 PM	Compressed (zipp...	786 KB
KwazamChessFinal	21/1/2025 8:29 PM	File folder	

2. Navigate to the directory as shown below inside the extracted “KwazamChessFinal”.



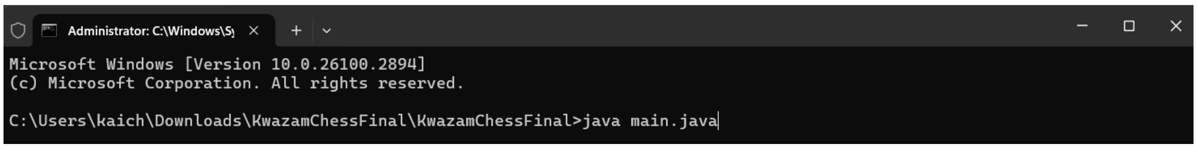
Name	Date modified	Type	Size
Main.class	21/1/2025 8:29 PM	CLASS File	1 KB
Main	21/1/2025 8:29 PM	Java Source File	1 KB
View	21/1/2025 8:29 PM	File folder	
audio	21/1/2025 8:29 PM	File folder	
Controller	21/1/2025 8:29 PM	File folder	
image	21/1/2025 8:29 PM	File folder	
Model	21/1/2025 8:29 PM	File folder	
Save_game	21/1/2025 8:29 PM	File folder	

3. Type “cmd” on the address bar, then press enter.



Name	Date modified	Type	Size
Main.class	21/1/2025 8:29 PM	CLASS File	1 KB
Main	21/1/2025 8:29 PM	Java Source File	1 KB
View	21/1/2025 8:29 PM	File folder	
audio	21/1/2025 8:29 PM	File folder	
Controller	21/1/2025 8:29 PM	File folder	
image	21/1/2025 8:29 PM	File folder	
Model	21/1/2025 8:29 PM	File folder	
Save_game	21/1/2025 8:29 PM	File folder	

4. Type “java main.java” command and press enter to run the code.

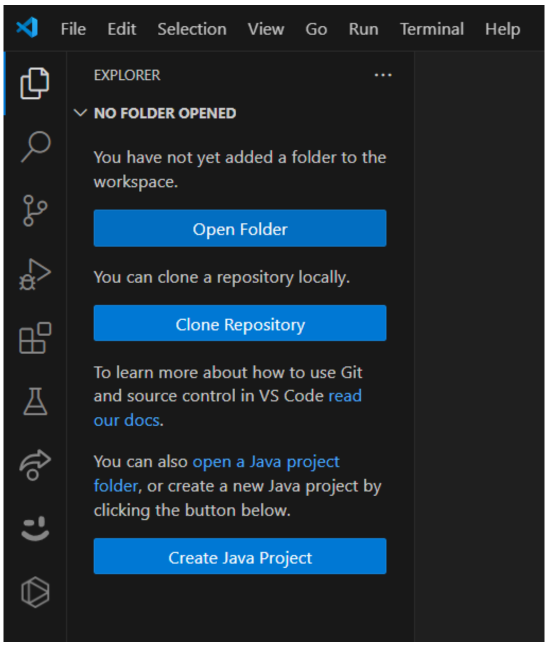


```
Microsoft Windows [Version 10.0.26100.2894]
(c) Microsoft Corporation. All rights reserved.

C:\Users\kaich\Downloads\KwazamChessFinal\KwazamChessFinal>java main.java
```

1.1.2 Visual Studio Code

1. After downloaded the zip file and extracted it, open Visual Studio Code and navigate to the explorer. Press the “Open Folder” button to find the “KwazamChessFinal” and open it.



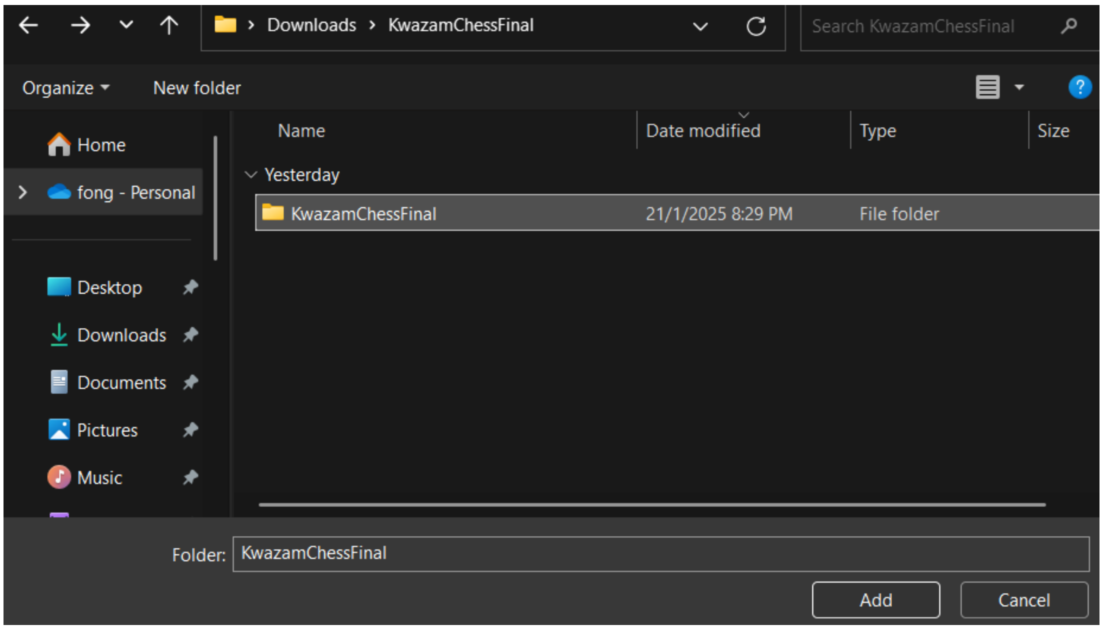
You have not yet added a folder to the workspace.

You can clone a repository locally.

To learn more about how to use Git and source control in VS Code read our docs.

You can also open a Java project folder, or create a new Java project by clicking the button below.

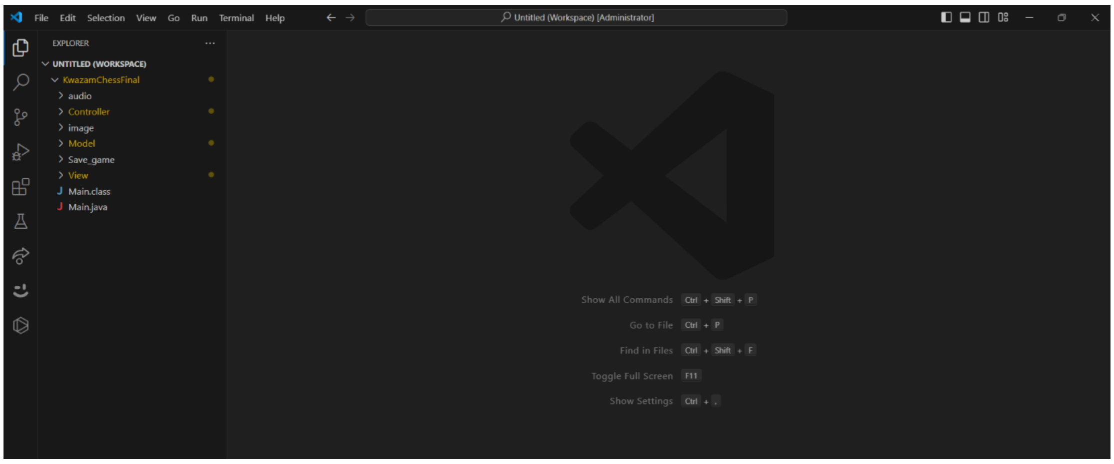
2. Select the folder named “KwazamChessFinal” and press the “Add” button to add the folder into your workspace as the image shown below.



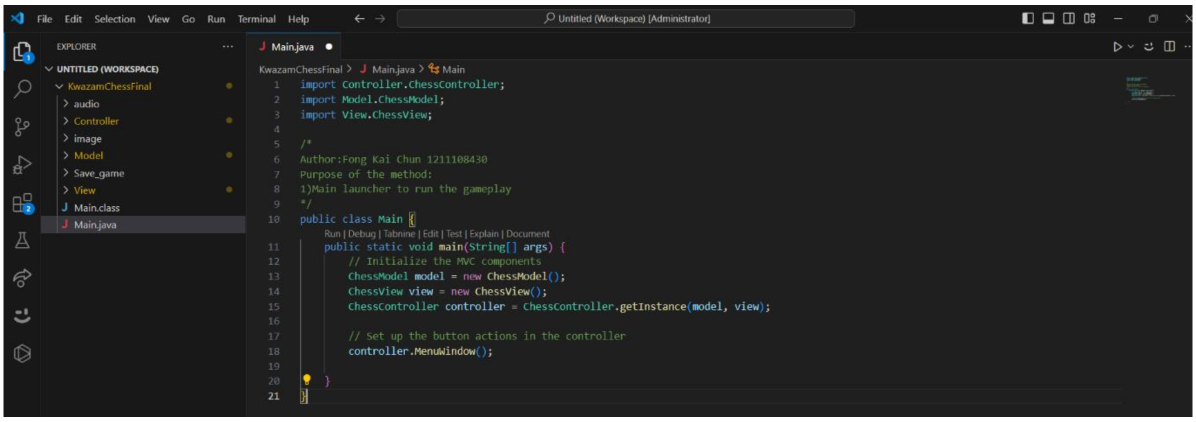
Name	Date modified	Type	Size
Yesterday			
KwazamChessFinal	21/1/2025 8:29 PM	File folder	

Folder: KwazamChessFinal

3. After adding the folder into workspace, workspace should look like the image below:



4. Open the main.java by double clicking it.



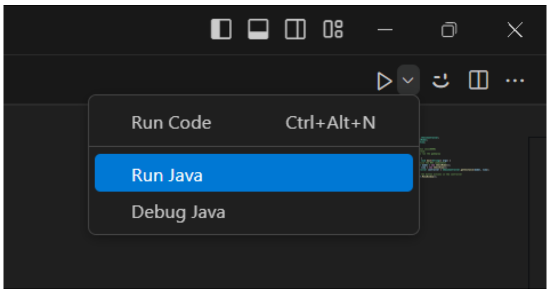
```java
import Controller.ChessController;
import Model.ChessModel;
import View.ChessView;

/*
Author:Fong Kai Chun 1211108430
Purpose of the method:
1)Main launcher to run the gameplay
*/
public class Main {
    Run | Debug | Tabnine | Edit | Test | Explain | Document
    public static void main(String[] args) {
        // Initialize the MVC components
        ChessModel model = new ChessModel();
        ChessView view = new ChessView();
        ChessController controller = ChessController.getInstance(model, view);

        // Set up the button actions in the controller
        controller.MenuWindow();

    }
}
```

5. To run the code, choose the "Run Java" on the top right corner as shown below:



Chapter 2: Requirements

2.1 Requirements

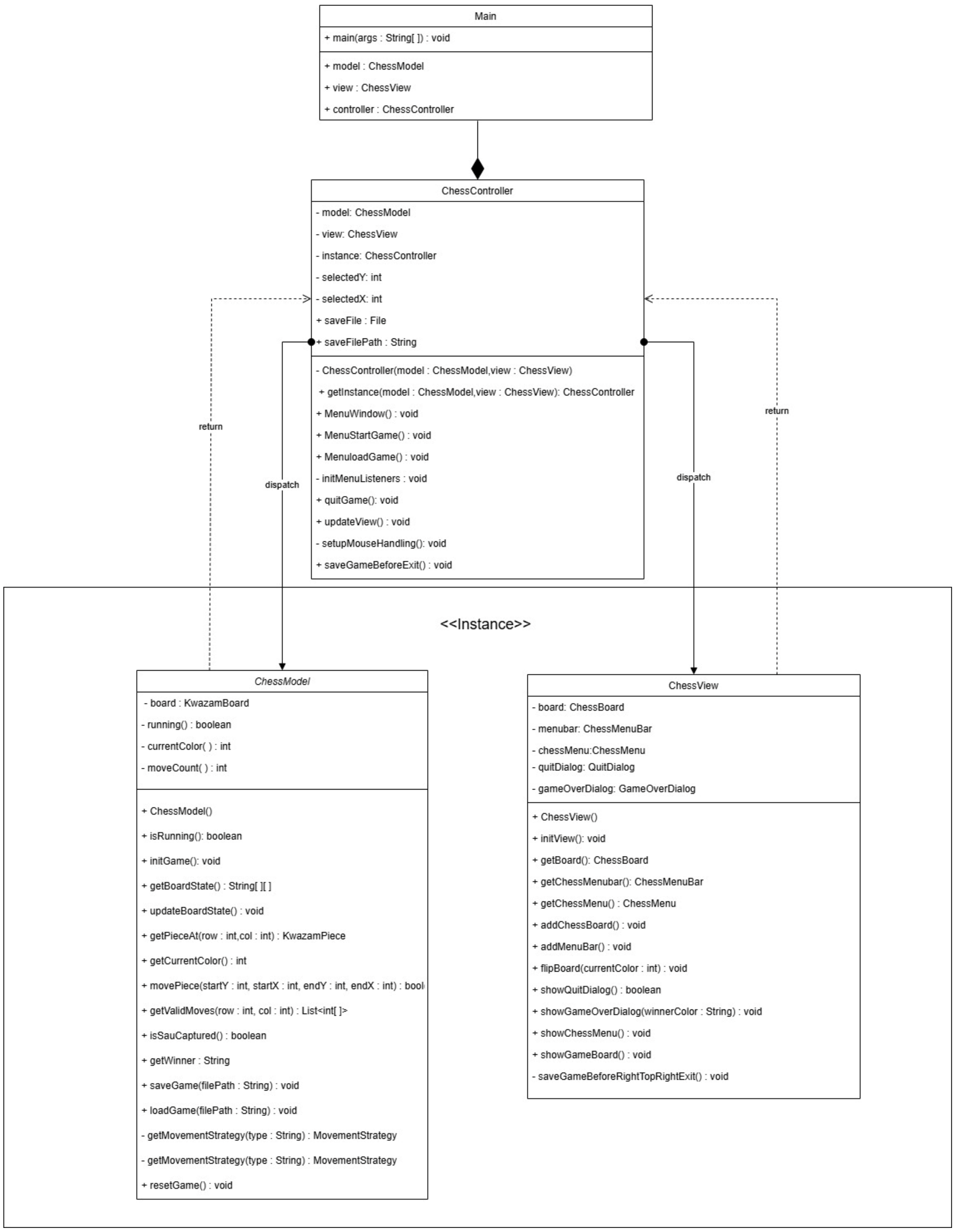
No	Requirements	Remarks
1	Proper indentation	✓
2	Every class must be commented	✓
3	Must use MVC	✓
4	At least one design pattern	✓
5	Ensure proper use of OO concepts	✓
User-Friendly Features		
•	Resizable windows	✓
•	Menus are working during gameplay	✓
•	The board will flip for each player's turn	✓
•	Save and load game	✓
•	The saved game file is a human-readable text file	✓

2.2 Non-Requirements

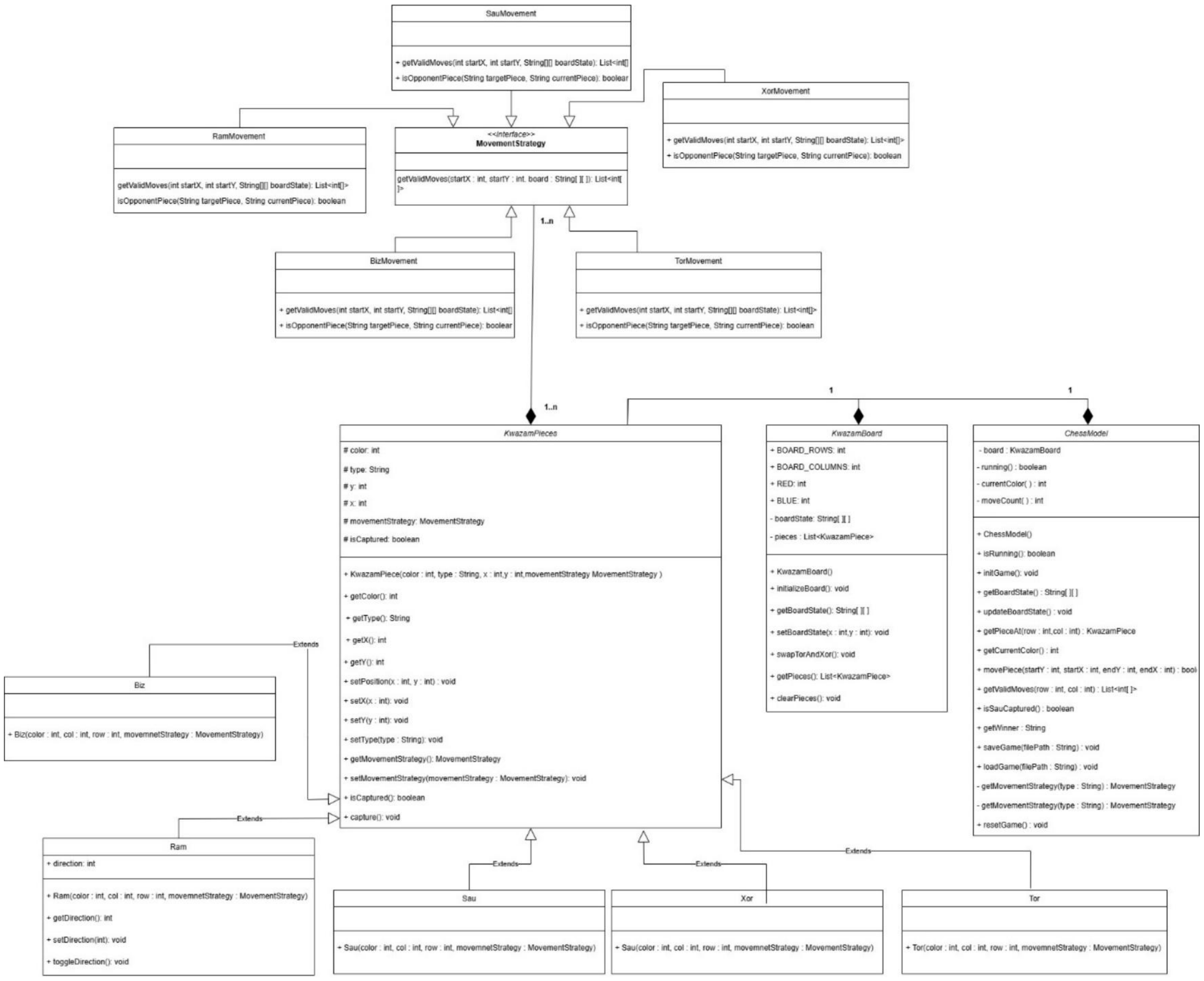
No	Non-Requirements	Remarks
1	Add audio	✓
2	Custom chessboard color	✓
3	Add logo	✓
4	Add highlight for every pieces move	✓

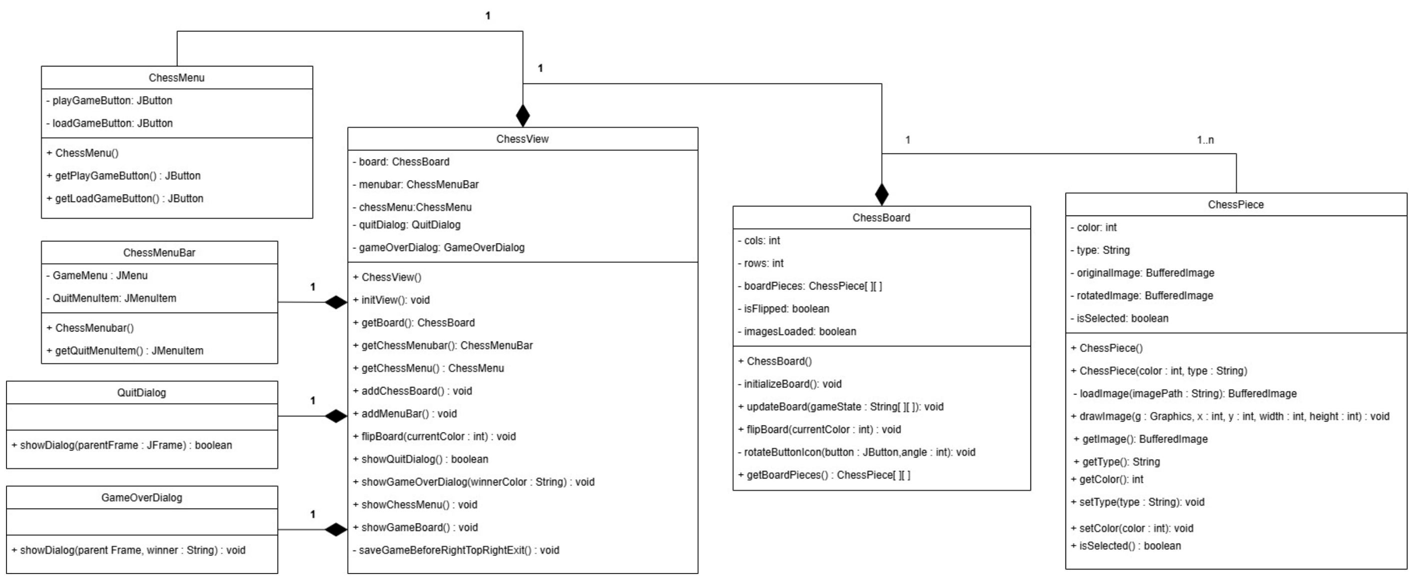
Chapter 3: UML Class Diagran

3.1 Class Diagram



Main
+ main(args : String[ ]) : void
+ model : ChessModel
+ view : ChessView
+ controller : ChessController
ChessController
- model: ChessModel
- view: ChessView
- instance: ChessController
- selectedY: int
- selectedX: int
+ saveFile : File
+ saveFilePath : String
- ChessController(model : ChessModel,view : ChessView)
+ getInstance(model : ChessModel,view : ChessView): ChessController
+ MenuWindow() : void
+ MenuStartGame() : void
+ MenuloadGame() : void
- initMenuListeners : void
+ quitGame(): void
+ updateView() : void
- setupMouseHandling(): void
+ saveGameBeforeExit() : void
return
dispatch
dispatch
return
<<Instance>>
ChessModel
- board : KwazamBoard
- running() : boolean
- currentColor( ) : int
- moveCount( ) : int
+ ChessModel()
+ isRunning(): boolean
+ initGame(): void
+ getBoardState() : String[ ][ ]
+ updateBoardState() : void
+ getPieceAt(row : int,col : int) : KwazamPiece
+ getCurrentColor() : int
+ movePiece(startY : int, startX : int, endY : int, endX : int) : bool
+ getValidMoves(row : int, col : int) : List<int[ ]>
+ isSauCaptured() : boolean
+ getWinner : String
+ saveGame(filePath : String) : void
+ loadGame(filePath : String) : void
- getMovementStrategy(type : String) : MovementStrategy
- getMovementStrategy(type : String) : MovementStrategy
+ resetGame() : void
ChessView
- board: ChessBoard
- menubar: ChessMenuBar
- chessMenu:ChessMenu
- quitDialog: QuitDialog
- gameOverDialog: GameOverDialog
+ ChessView()
+ initView(): void
+ getBoard(): ChessBoard
+ getChessMenubar(): ChessMenuBar
+ getChessMenu() : ChessMenu
+ addChessBoard() : void
+ addMenuBar() : void
+ flipBoard(currentColor : int) : void
+ showQuitDialog() : boolean
+ showGameOverDialog(winnerColor : String) : void
+ showChessMenu() : void
+ showGameBoard() : void
- saveGameBeforeRightTopRightExit() : void


SauMovement
+ getValidMoves(int startX, int startY, String[][] boardState): List<int[]
+ isOpponentPiece(String targetPiece, String currentPiece): boolear
XorMovement
+ getValidMoves(int startX, int startY, String[][] boardState): List<int[]>
+ isOpponentPiece(String targetPiece, String currentPiece): boolean
RamMovement
getValidMoves(int startX, int startY, String[][] boardState): List<int[]>
isOpponentPiece(String targetPiece, String currentPiece): boolean
<<interface>>
MovementStrategy
getValidMoves(startX : int, startY : int. board : String[ ][ ]): List<int[ ]>
1..n
BizMovement
+ getValidMoves(int startX, int startY, String[][] boardState): List<int[]
+ isOpponentPiece(String targetPiece, String currentPiece): boolear
TorMovement
+ getValidMoves(int startX, int startY, String[][] boardState): List<int[]>
+ isOpponentPiece(String targetPiece, String currentPiece): boolean
1..n
1
1
KwazamPieces
# color: int
# type: String
# y: int
# x: int
# movementStrategy: MovementStrategy
# isCaptured: boolean
+ KwazamPiece(color : int, type : String, x : int,y : int,movementStrategy MovementStrategy)
+ getColor(): int
+ getType(): String
+ getX(): int
+ getY(): int
+ setPosition(x : int, y : int) : void
+ setX(x : int): void
+ setY(y : int): void
+ setType(type : String): void
+ getMovementStrategy(): MovementStrategy
+ setMovementStrategy(movementStrategy : MovementStrategy): void
+ isCaptured(): boolean
+ capture(): void
KwazamBoard
+ BOARD_ROWS: int
+ BOARD_COLUMNS: int
+ RED: int
+ BLUE: int
- boardState: String[ ][ ]
- pieces : List<KwazamPiece>
+ KwazamBoard()
+ initializeBoard(): void
+ getBoardState(): String[ ][ ]
+ setBoardState(x : int,y : int): void
+ swapTorAndXor(): void
+ getPieces(): List<KwazamPiece>
+ clearPieces(): void
ChessModel
- board : KwazamBoard
- running() : boolean
- currentColor( ) : int
- moveCount( ) : int
+ ChessModel()
+ isRunning(): boolean
+ initGame(): void
+ getBoardState() : String[ ][ ]
+ updateBoardState() : void
+ getPieceAt(row : int,col : int) : KwazamPiece
+ getCurrentColor() : int
+ movePiece(startY : int, startX : int, endY : int, endX : int) : bool
+ getValidMoves(row : int, col : int) : List<int[ ]>
+ isSauCaptured() : boolean
+ getWinner : String
+ saveGame(filePath : String) : void
+ loadGame(filePath : String) : void
- getMovementStrategy(type : String) : MovementStrategy
- getMovementStrategy(type : String) : MovementStrategy
+ resetGame() : void
Extends
Biz
+ Biz(color : int, col : int, row : int, movemnetStrategy : MovementStrategy)
Extends
Ram
+ direction: int
+ Ram(color : int, col : int, row : int, movemnetStrategy : MovementStrategy)
+ getDirection(): int
+ setDirection(int): void
+ toggleDirection(): void
Extends
Extends
Extends
Sau
+ Sau(color : int, col : int, row : int, movemnetStrategy : MovementStrategy)
Xor
+ Sau(color : int, col : int, row : int, movemnetStrategy : MovementStrategy)
Tor
+ Tor(color : int, col : int, row : int, movemnetStrategy : MovementStrategy)


1
1
ChessMenu
- playGameButton: JButton
- loadGameButton: JButton
+ ChessMenu()
+ getPlayGameButton() : JButton
+ getLoadGameButton() : JButton
ChessView
- board: ChessBoard
- menubar: ChessMenuBar
- chessMenu:ChessMenu
- quitDialog: QuitDialog
- gameOverDialog: GameOverDialog
+ ChessView()
+ initView(): void
+ getBoard(): ChessBoard
+ getChessMenubar(): ChessMenuBar
+ getChessMenu() : ChessMenu
+ addChessBoard() : void
+ addMenuBar() : void
+ flipBoard(currentColor : int) : void
+ showQuitDialog() : boolean
+ showGameOverDialog(winnerColor : String) : void
+ showChessMenu() : void
+ showGameBoard() : void
- saveGameBeforeRightTopRightExit() : void
ChessMenuBar
- GameMenu : JMenu
- QuitMenuItem: JMenuItem
+ ChessMenubar()
+ getQuitMenuItem() : JMenuItem
1
QuitDialog
+ showDialog(parentFrame : JFrame) : boolean
1
GameOverDialog
+ showDialog(parent Frame, winner : String) : void
1
1
1..n
ChessBoard
- cols: int
- rows: int
- boardPieces: ChessPiece[ ][ ]
- isFlipped: boolean
- imagesLoaded: boolean
+ ChessBoard()
- initializeBoard(): void
+ updateBoard(gameState : String[ ][ ]): void
+ flipBoard(currentColor : int) : void
- rotateButtonIcon(button : JButton,angle : int): void
+ getBoardPieces() : ChessPiece[ ][ ]
ChessPiece
- color: int
- type: String
- originalImage: BufferedImage
- rotatedImage: BufferedImage
- isSelected: boolean
+ ChessPiece()
+ ChessPiece(color : int, type : String)
- loadImage(imagePath : String): BufferedImage
+ drawImage(g : Graphics, x : int, y : int, width : int, height : int) : void
+ getImage(): BufferedImage
+ getType(): String
+ getColor(): int
+ setType(type : String): void
+ setColor(color : int): void
+ isSelected() : boolean

The class diagram above shows the architecture of the KwazamChess game application, in which employing a Model-View-Controller (MVC) pattern. This application is composed of a Main class, which is the entry point, and three major components: ChessModel (the model), ChessView (the view), and ChessController (the controller). The ChessController acts as the intermediary, managing the interaction between the ChessModel and ChessView. Dashed lines indicate that the Main class creates a ChessController, which then instantiates ChessModel and ChessView.

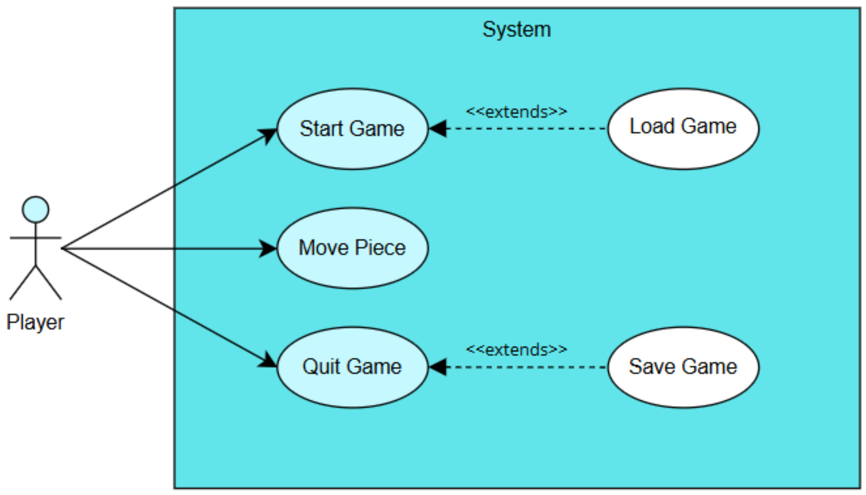
- **Model (ChessModel):**
  The ChessModel class contains the game logic and state. It holds information about the current board using a KwazambBoard object, and manages the current player's color, move count, and whether the game is running. It handles piece movement through an association with movement strategy classes like SauMovement, RunMovement, XorMovement, and TorMovement, all inheriting from an abstract MovementStrategy class. It can initialize a game, move pieces, validate moves and check for captures and wins. It loads and saves game states, too. KwazambPiece class represents a single piece. Board extends the KwazambBoard to help keep track of pieces as well.

- **View (ChessView):**
  The ChessView is responsible for displaying the game's visual interface. It contains a ChessBoard object that holds ChessPiece objects. It also manages the menu system, including a ChessMenuBar, ChessMenu, QuitDialog, and GameOverDialog to provide UI functionality. It handles displaying the board, adding components, and showing game over and quit dialogs. It also interacts with the board to flip the board visually.

- **Controller (ChessController):**
  The ChessController acts as a bridge between the ChessModel and ChessView. It receives user input from the ChessView and updates the ChessModel accordingly. Then, based on the changes in ChessModel, it informs the ChessView to update. The ChessController also handles menu events, saving and loading games, and quitting the application.

Chapter 4: Use Case Diagram

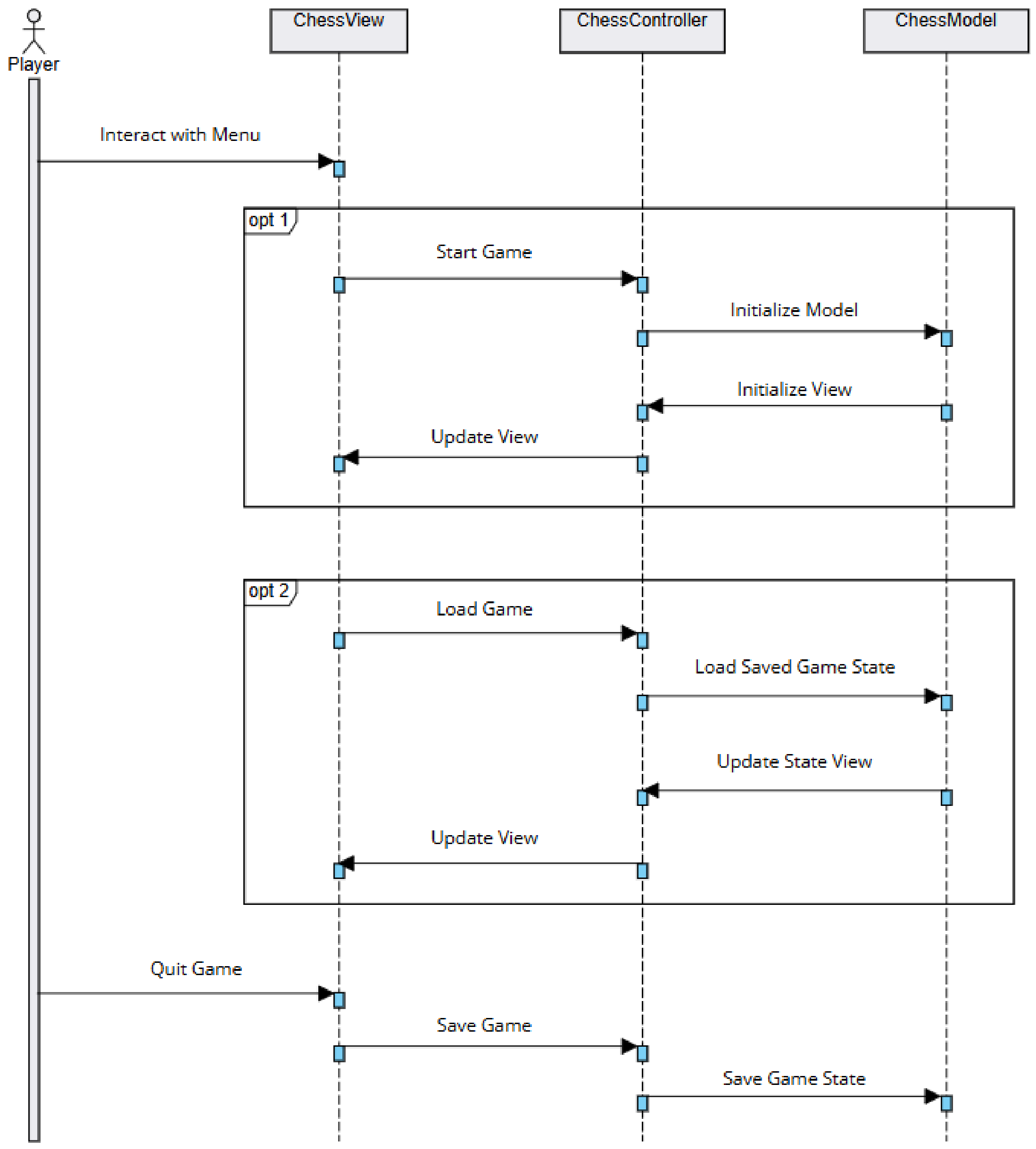
4.1 Use Case Diagram



The Use Case Diagram above shows how a User interact with Subject Registration System. The user can initiate actions like "Add Subject," "Register Subject," and "View Registrations." The "Add Subject" and "View Registrations" use cases inherently include "Check Subject Capacity" and "Display Registered Students" respectively, as integral parts of their processes. Furthermore, the "Register Subject" use case can optionally be extended by "Notify Results," indicating that result notifications are conditional upon the registration process. In conclusion, this diagram shows the main features of the system that can be displayed to user.

Chapter 5: Sequence Diagram

5.1 Sequence Diagram



The sequence diagram above shows the interactions within the Kwazam Chess game application by following a Model-View-Controller (MVC) architectural pattern. It shows the series of steps that a player initiates and that the ChessView, ChessController, and ChessModel components process.

This diagram shows three main scenarios:

- **Option 1: Starting a New Game**

  The player interacts with the menu in the ChessView and selects "Start Game". This triggers the ChessView to call the ChessController's "Start Game" function. The ChessController then initializes the ChessModel, and subsequently instructs the ChessView to initialize and update itself based on the new game model.

- **Option 2: Loading a Saved Game**

  Similar to starting a game, the player interacts with the menu in the ChessView and selects "Load Game". The ChessView calls the ChessController's "Load Game" function. The ChessController loads the saved game state into the ChessModel, updates the ChessView's state based on the loaded model, and finally updates the view to display the loaded game.

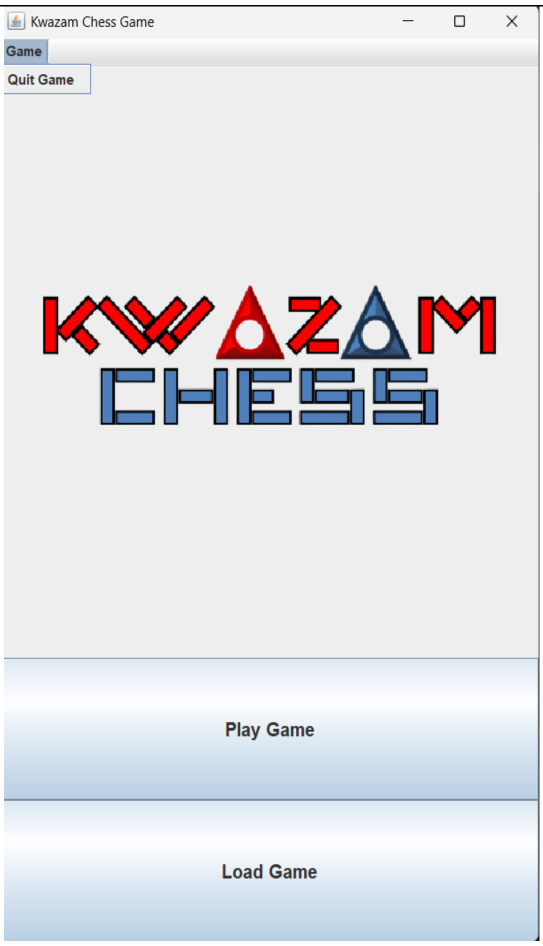
- **Quit the Game**

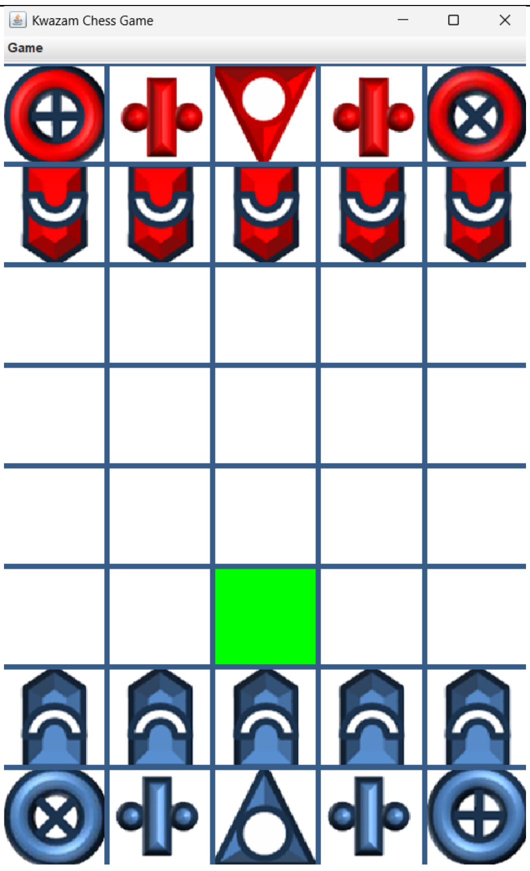
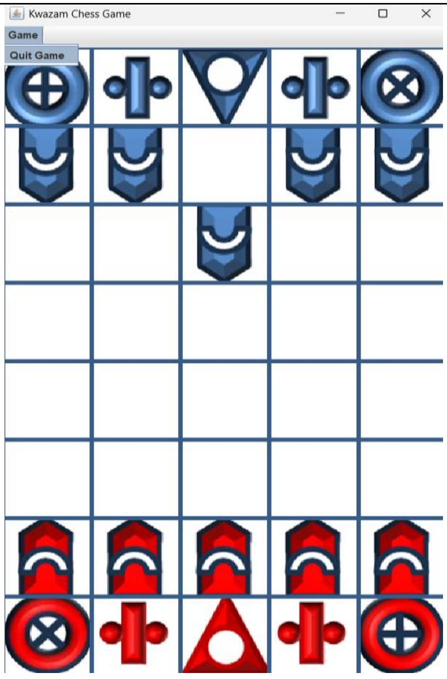
  When the Player chooses to "Quit Game", the ChessView calls the ChessController's "Save Game" function. The ChessController then instructs the ChessModel to save the current game state. This action prepares the application for termination, potentially saving the game progress before exiting.

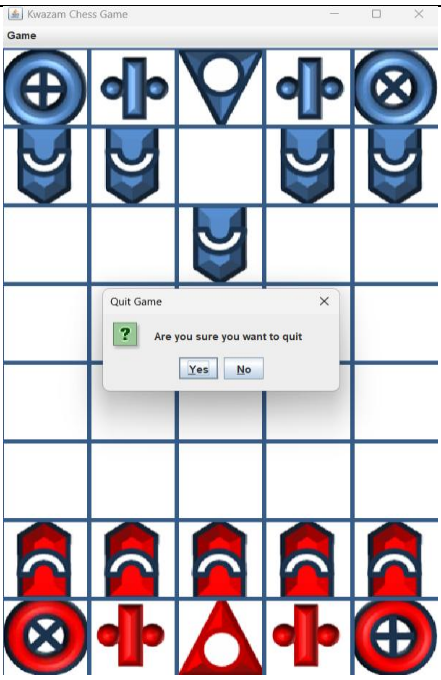
In short, the diagram shows how the player interacts with the View, which then manage the Model and Controller to handle game logic, data management, and view updates, showcasing the MVC pattern in action for a Kwazam Chess game.

Chapter 6: User Manual

6.1 User Manual Table

 Kwazam Chess Game Game Quit Game KWAZAM CHESS Play Game Load Game	The image placed beside is the main menu. In this main menu, there are 2 buttons which are "Play Game" and "Load Game" button. Press "Play Game" button to start a whole new game. Whereas press the "Load Game" button to load the previously saved game. On the top-left corner, there is a "Quit Game" option which located within a dropdown menu under the "Game" menu item. Click on it to quit the game.

	Click on the chess piece to select. Then, click on the highlighted button (green), which is the valid moves to move the piece.
	On the top left-corner, there is a "Quit Game" option which located within a dropdown menu under the "Game" menu item.

 Kwazam Chess Game Game Quit Game Are you sure you want to quit Yes No	After clicked on the “Quit Game” option, a quit dialog with “Yes” and “No” option will appear. Click on the “Yes” option to quit the game. At the same time, the current chess game progress will be saved. On the other hand, click on the “No” option to return back into the game.

[END OF REPORT]