**CCP6224 Project**

**Trimester 2310**

**by Abrar TT8L C**

Team Leader: Eng Wei Jiun, 60127817355, [eng.wei.jiun@student.mmu.edu.my](mailto:eng.wei.jiun@student.mmu.edu.my)

Team members:

Lai Joon Li, 01126314373, [lai.joon.li@student.mmu.edu.my](mailto:lai.joon.li@student.mmu.edu.my)

Soukmead Ong Yu Kang, 01116849939, [soukmaed.ong.yu@student.mmu.edu.my](mailto:SOUKMAED.ONG.YU@STUDENT.MMU.EDU.MY)

**Table Content**

[**Compile and Run Instructions 1**](#_l1r22ozi4qs9)

[**UML Class Diagram 2**](#_wd1h4nb0ncj9)

[**Model-View-Controller (MVC) Pattern 3**](#_92e4jtkl78v9)

[**Strategy Pattern 5**](#_4u2c25hh95ri)

[**Use Case Diagram 6**](#_ntxohg5kshmu)

[**Sequence Diagrams 7**](#_u5rz4rjan5qn)

[**User Documentation 10**](#_71lg2a1utnj0)

## **Compile and Run Instructions**

1. **Visual Studio Code**

* Open the project folder in VSCode.
* Make sure you have the Java Extension Pack installed.
* Locate the main class (Main.java) and right-click on it.
* Choose “Run Java” to compile and run the program.

1. **BlueJ**

* Open BlueJ and open the project.
* Right-click on the Main.java and select “Compile”.
* Right-click on the Main.java and select “void main(String[] args)" to run the

program.

1. **Eclipse**

* Open Eclipse and import the project.
* Right-click on the Main class and select “Run As” > Java Application.
* You can run the program by clicking the “Run” button or by right-clicking on the Main.java and selecting “Run As” > "Java Application.

1. **IntelliJ IDEA**

* Open IntelliJ and open the project.
* Locate the Main.java and right-click on it.
* Choose Run Main.java to compile and run the program.
* You can run the program directly from IntelliJ by clicking the green “Run” arrow next to the main method.

1. **NetBeans**

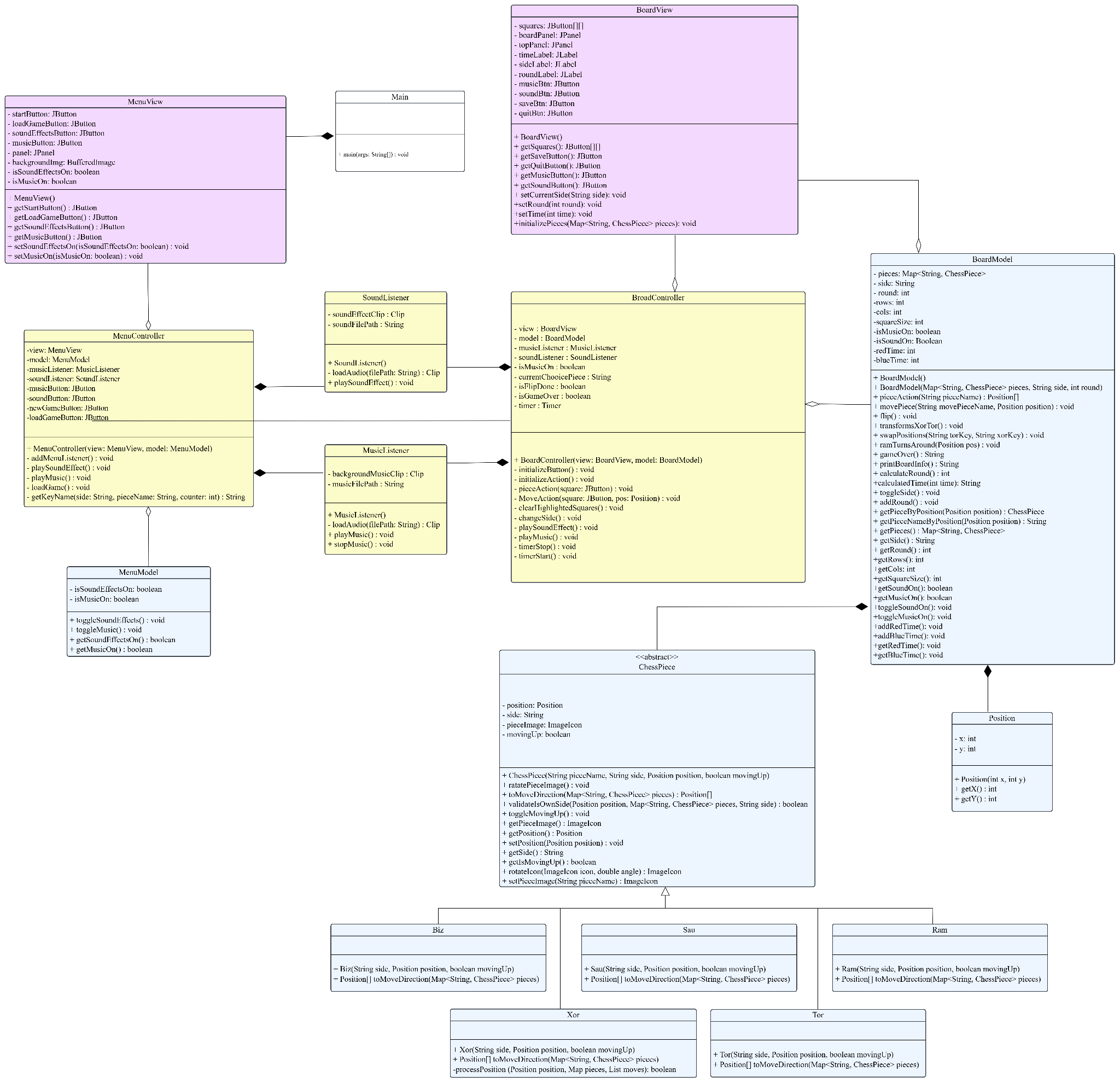
* Open NetBeans and open the project.
* Right-click on the Main.java and select “Run File”.
* You can run the program by clicking the “Run” button or by right-clicking on the

Main.java and selecting "Run File”.

1. **Command Line (MacOS/Linux)**

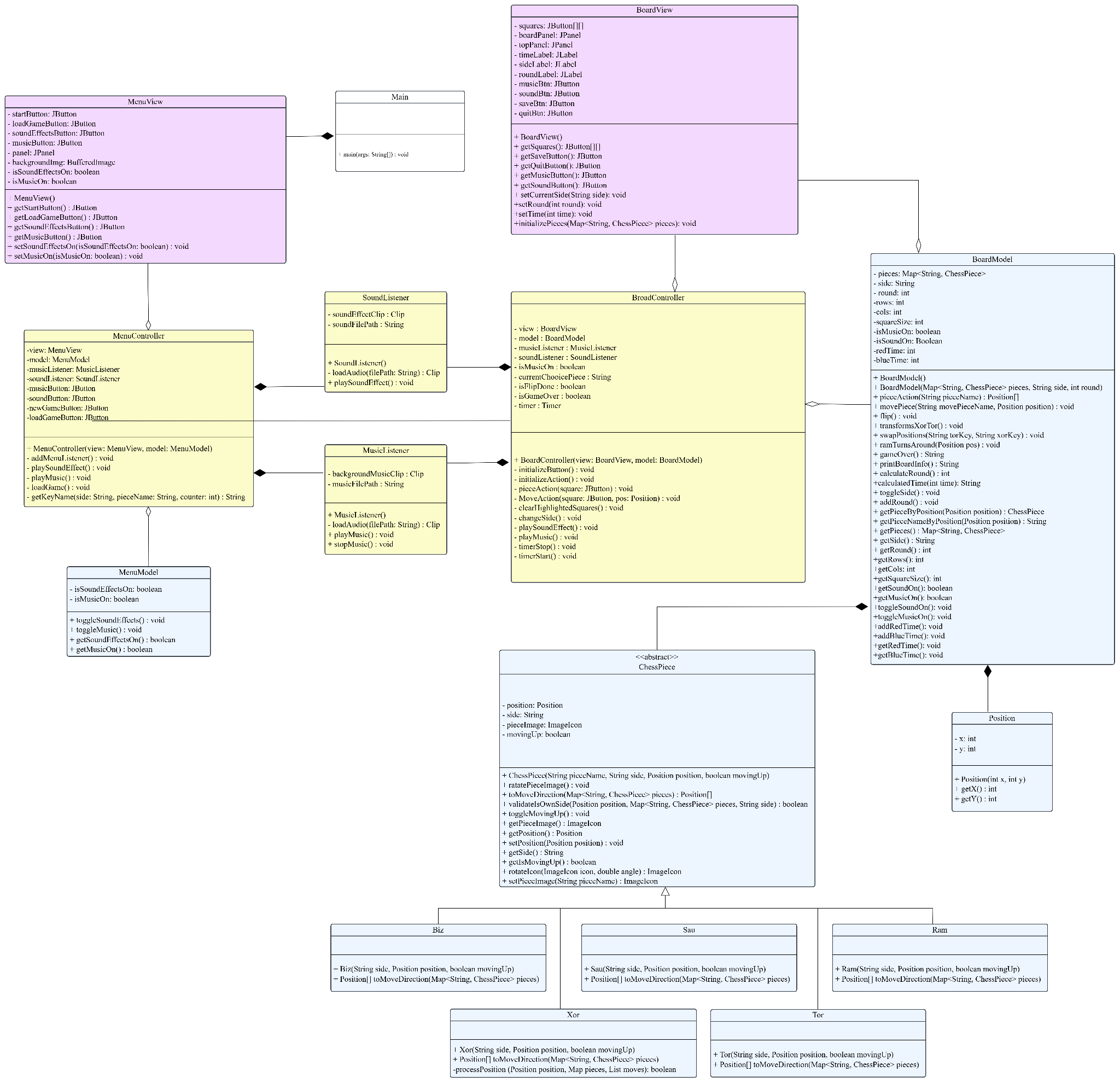
* Open Terminal.
* Navigate to the project directory.
* Run the following command:
* **javac** Game.**java**
* **java** Game

## **UML Class Diagram**

****

This is the UML Class Diagram for our Kwazam Chess Game involving 15 classes. We have implemented Model-View-Controller (MVC) and Strategy Pattern as our Design Pattern.

## **Model-View-Controller (MVC) Pattern**

****

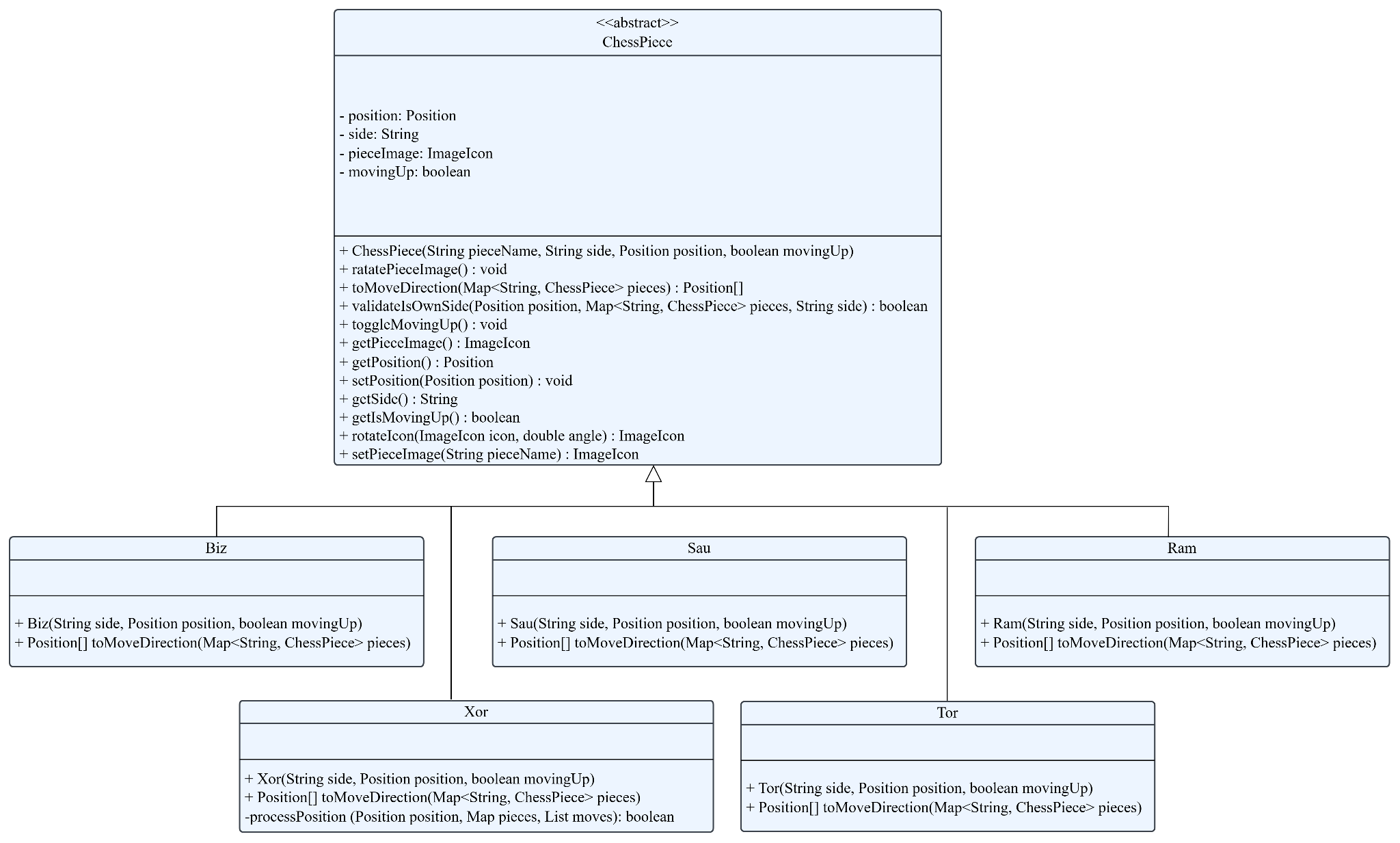
For the implementation of the Kwazam Chess game, the Model-View-Controller (MVC) pattern was chosen. This makes the program easier to manage and reuse in the future. It contains three components, namely, the model, the view, and the controller.

The model is the core functional part of the program. In Kwazam Chess, classes such as "BoardModel", "ChessPiece" and its subclasses ("Biz", "Tor", "Xor", "Ram", "Sau") represent the model. They encapsulate the game's data and the logic of how the game is played. The ChessPiece class and its subclasses are responsible for defining the properties of each piece and the allowed moves. The Board Model class holds the state of the game, including the positions of all pieces, the number of moves, and whether the game is won or not.

The view is what the user sees and interacts with. It displays the game state and responds to user input. In the Kwazam Chess game, "MenuView" and "BoardView" are used as views. "MenuView" acts as the starting point of the game and displays the main menu, while "BoardView" renders the board and pieces on the screen.

These classes update the visual presentation of the game based on changes in the model. Controllers process input and update the model. Controller classes in Kwazam Chess include "MusicListener" and "SoundListener". These controllers interpret user input (such as mouse clicks and menu selections) and convert them into actions on the model.

## **Strategy Pattern**

****

In this Kwazam Chess game, we use the Strategy pattern to handle the movement behavior of each chess piece. Each piece, such as the Ram, Tor, Xor, Biz, and Sau, has unique movement rules. By using the Strategy pattern, we can separate the movement logic of each piece into its own class. This makes the code easier to understand and maintain, as each piece's behavior is clearly defined and can be modified without affecting other parts of the program.

We chose the Strategy pattern because it allows us to define a common interface for movement behaviors. Each chess piece implements this interface, providing its specific movement logic. For example, the Tor piece can override the toMoveDirection() method to define its movement rules, while the Xor piece does the same for its unique behavior. This design makes it simple to add new pieces or change existing movement rules without needing to modify the base ChessPiece class or the rest of the program.

Overall, the Strategy pattern helps to keep the code modular and flexible. It supports the "open/closed principle," meaning the code is open for extension but closed for modification. By isolating the movement logic, we reduce the risk of bugs and make the chess game easier to expand in the future. This approach ensures that each chess piece is responsible for its own behavior while allowing the game to operate smoothly.

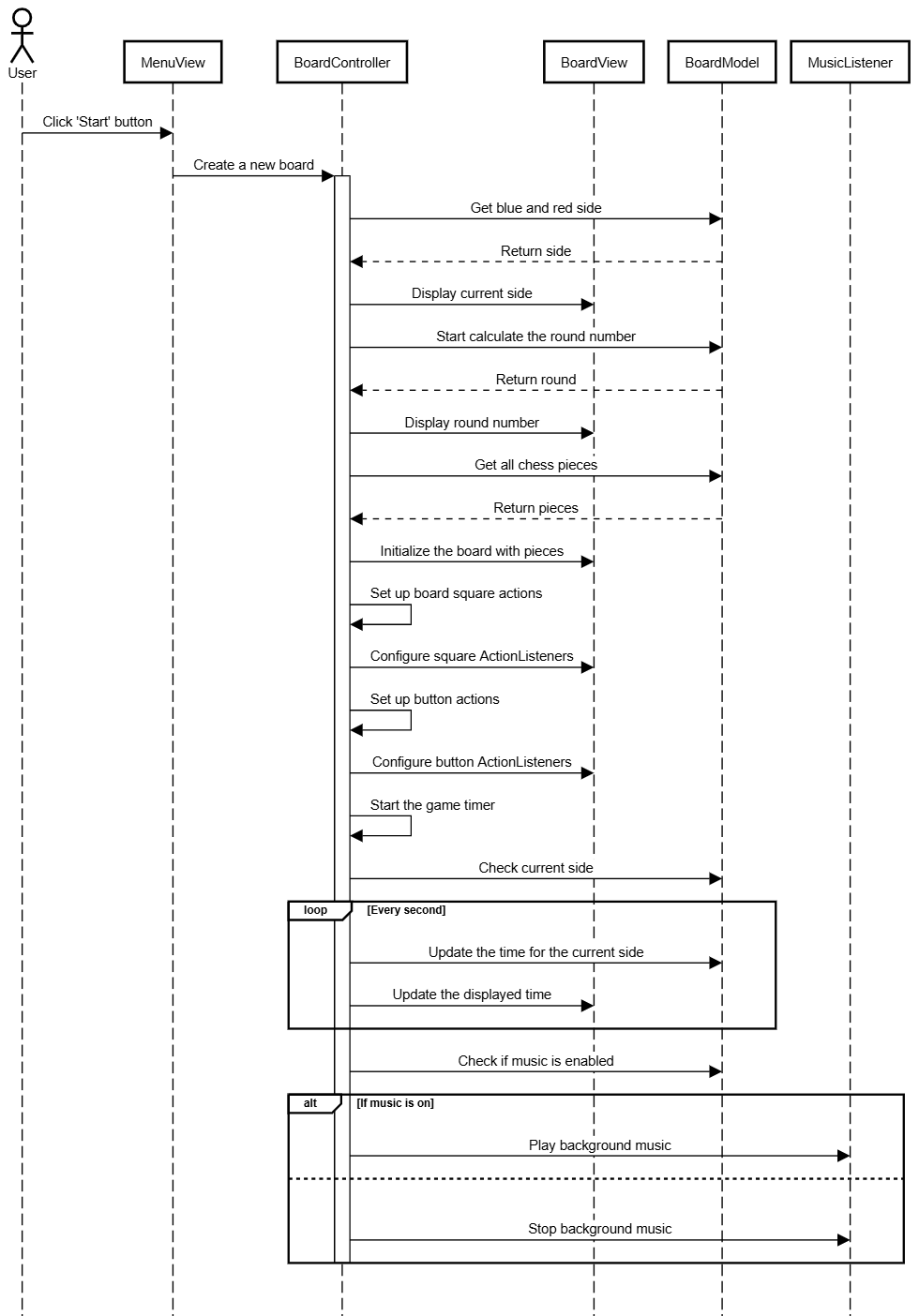
## **Use Case Diagram**

****

The Use Case Diagram shows the interactions between the user and the system. It represents the different actions or tasks the user can perform. The user can perform several actions, including starting or quitting the game, moving chess pieces across the board, and capturing the opponent's pieces based on the game rules. Players also have the ability to transform specific pieces, reverse their direction, and flip the board for better visibility during their turn. Additionally, the diagram highlights options for saving the game progress, loading a previously saved game, and toggling features like music or sound effects. These use cases ensure that players have a smooth and engaging experience while playing the game, with full control over the system's functionalities.

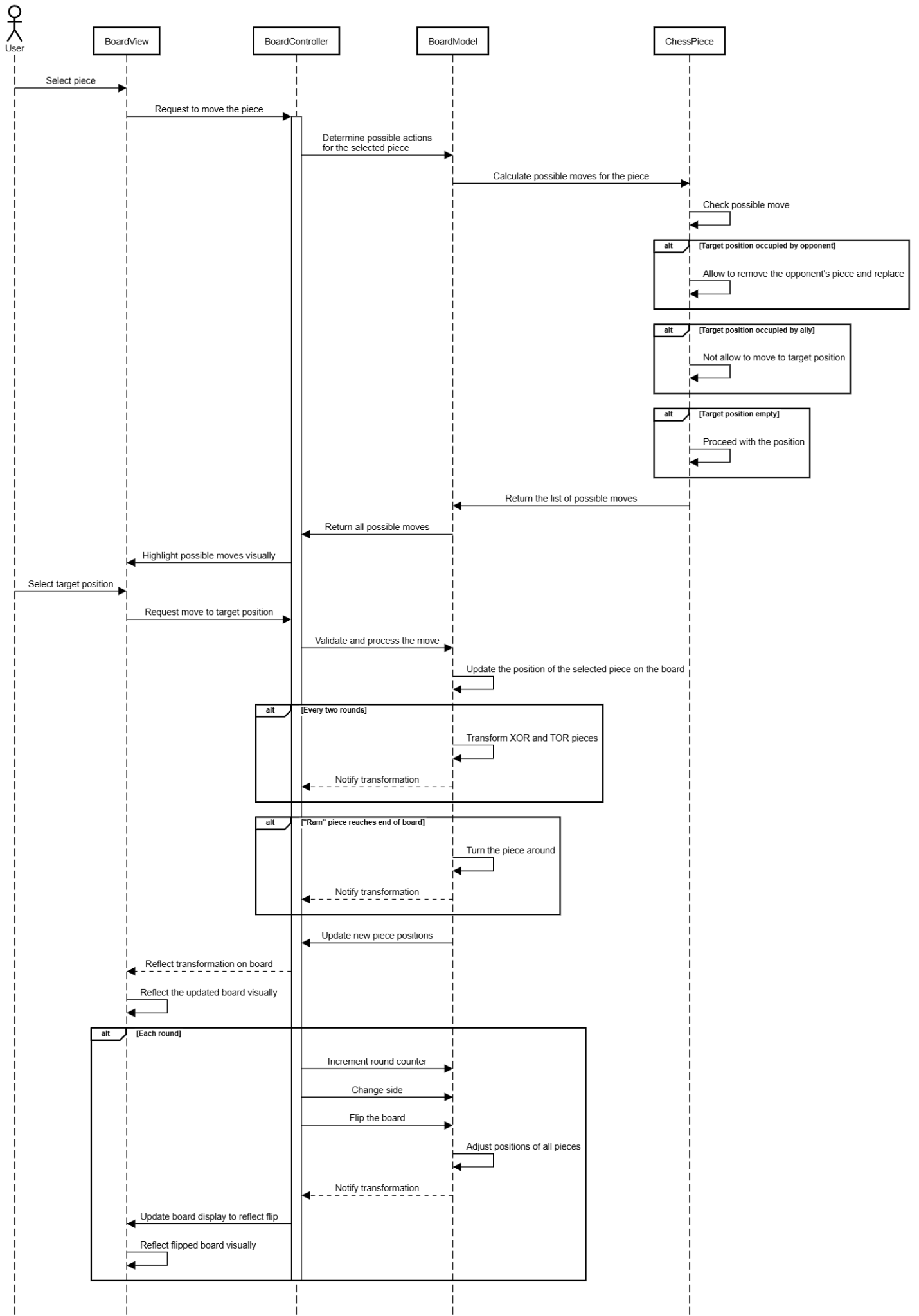
## **Sequence Diagrams**

**Start Game**

****

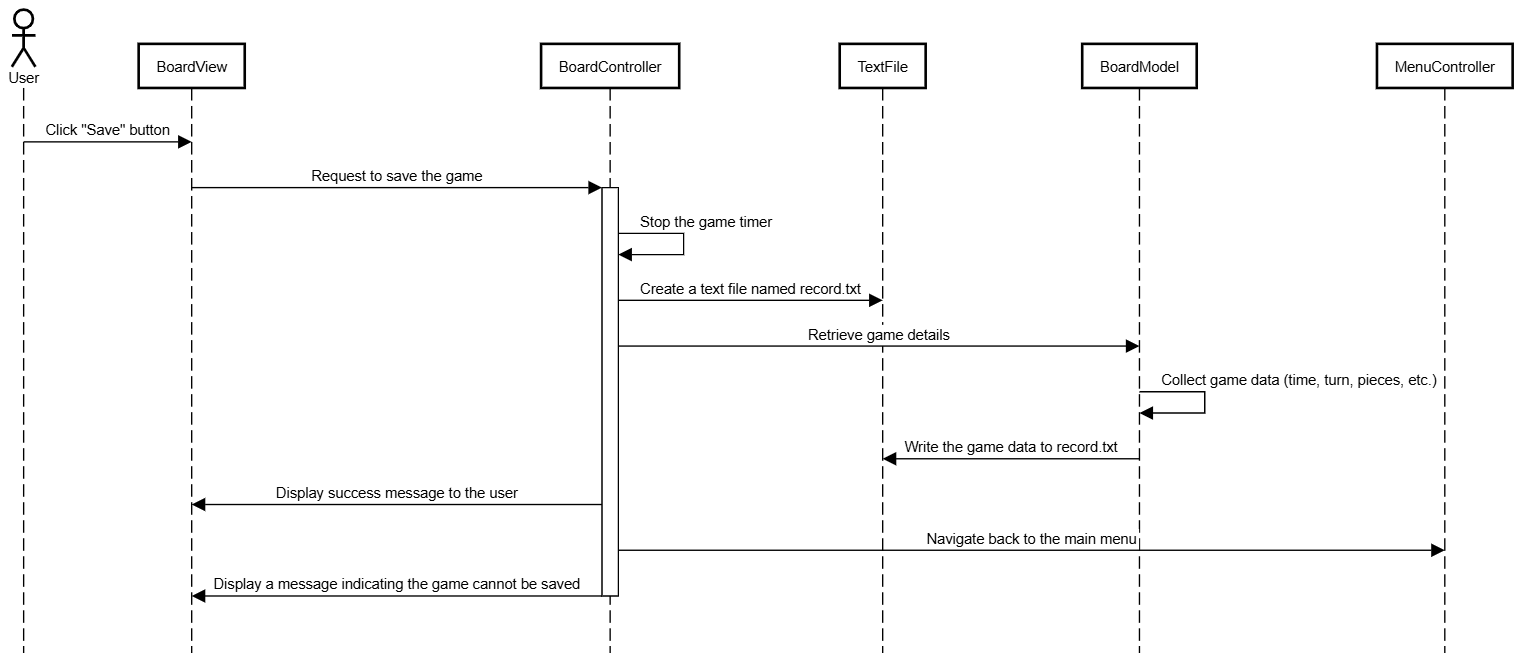
This sequence diagram outlines the process that occurs when a user clicks the 'Start' button in the chess game. It shows how the MenuView triggers the creation of the BoardController, which then initializes the board, sets the game state (e.g., side and round), and starts the game timer. Additionally, it handles music playback based on the user's settings.

**Chess Pieces**

****

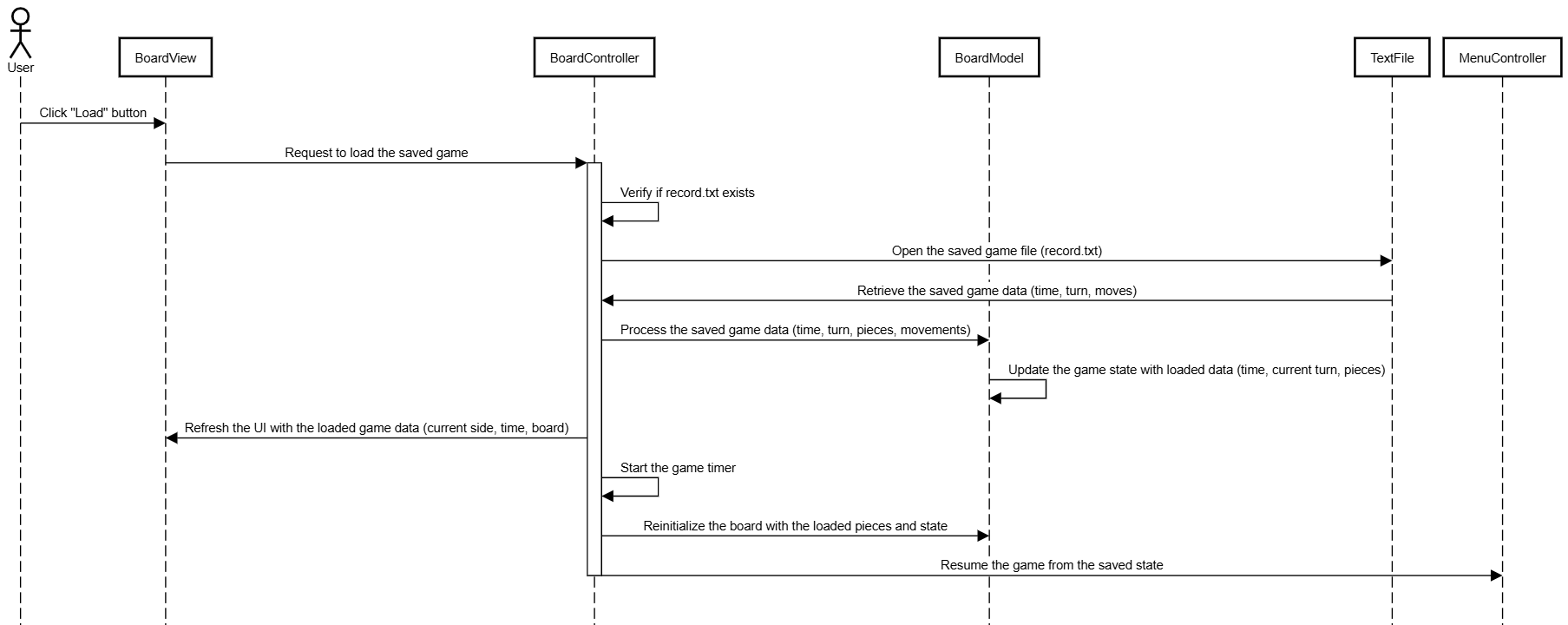
This sequence diagram outlines the process of moving a chess piece during the game. It shows how the user's interaction triggers actions across various components to validate and execute the move, update the board state, and visually reflect the changes.

**Save Game**

****

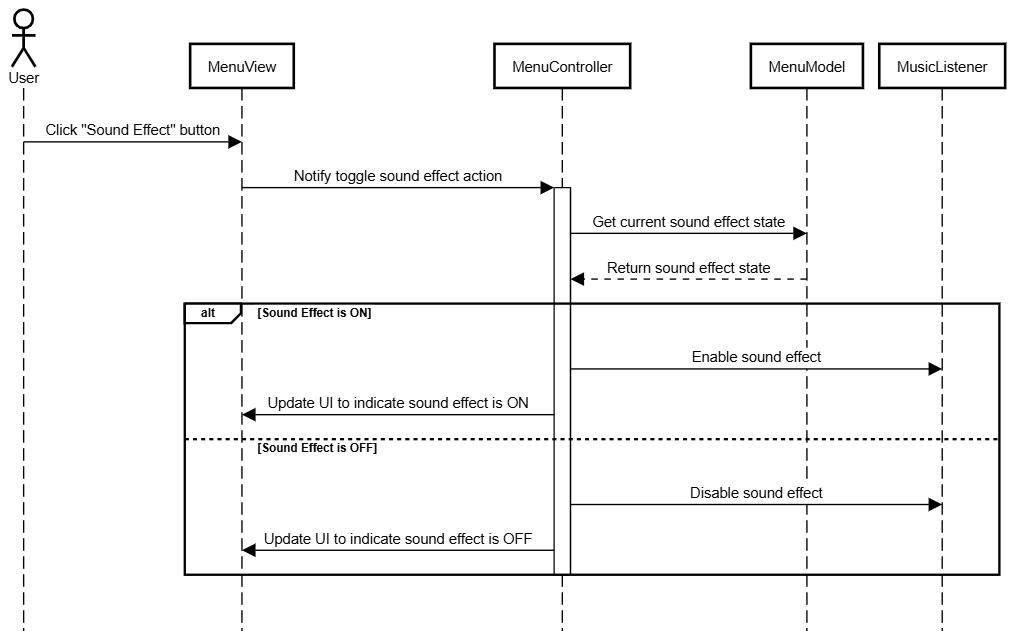
This sequence diagram outlines the process of saving the current game state in the chess application. It shows how the interactions between the user and the various components involved in verifying and handling the save operation.

**Load Game**

****

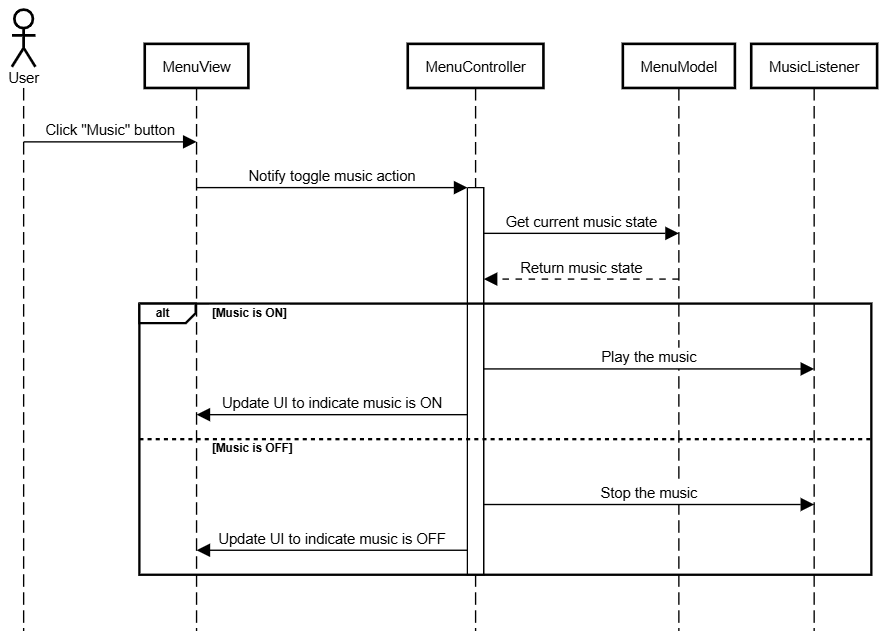
This sequence diagram describes the process of loading a previously saved game in the chess application. It highlights the interactions between the user and various components responsible for retrieving and restoring the game state.

**Menu Toggle Sound Effect**



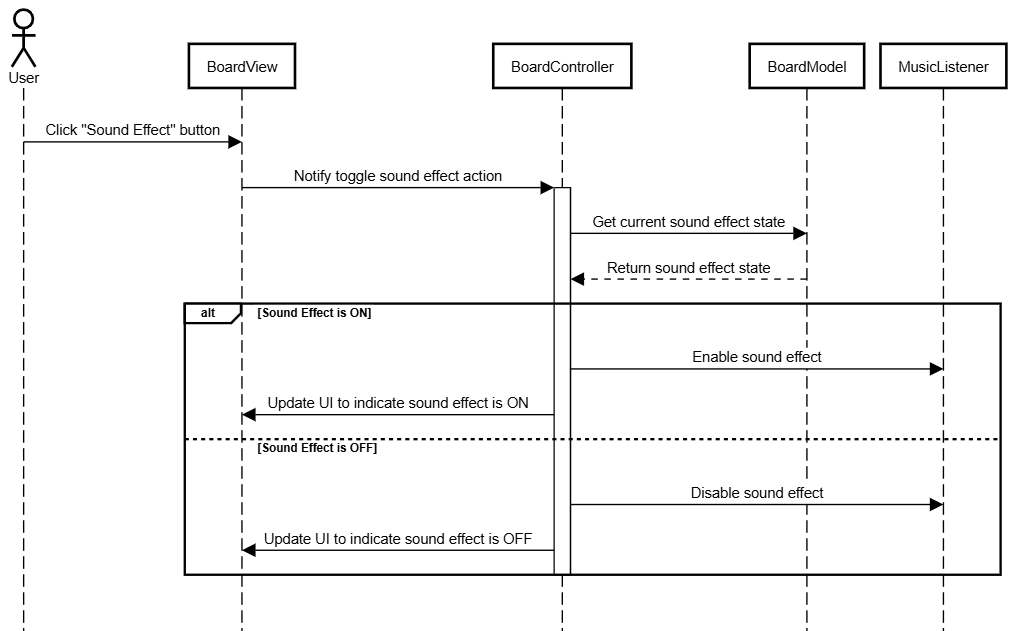
This sequence diagram describes the process of toggling the sound effect state in the menu. It shows how the user interacts with the system to switch the sound effect on or off, highlighting the communication between MenuView, MenuController, MenuModel and MusicListener.

**Menu Toggle Music**

****

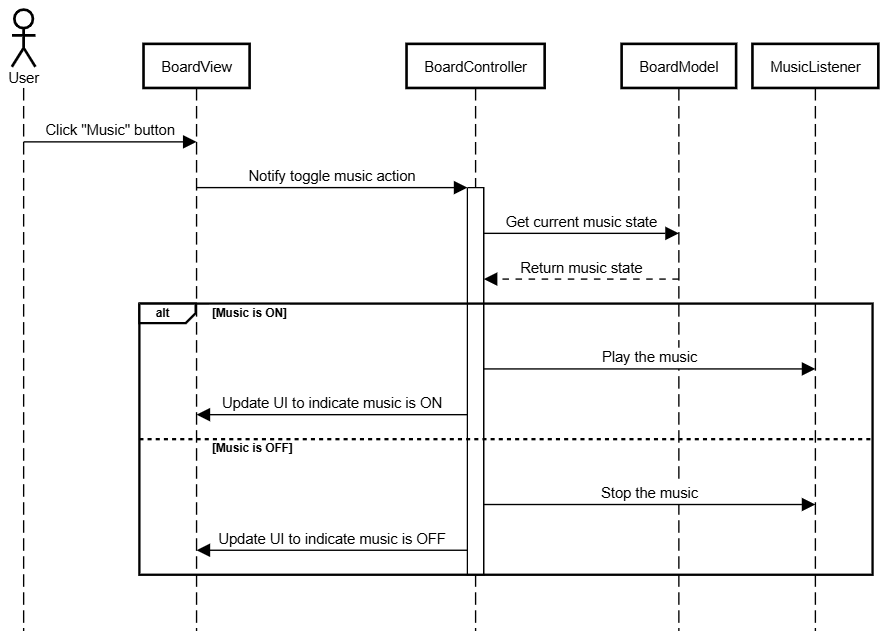
This sequence diagram describes the process of toggling the music state in the menu. It shows how the user interacts with the system to switch the music on or off, highlighting the communication between MenuView, MenuController, MenuModel and MusicListener.

**Board Toggle Sound Effect**

****

This sequence diagram describes the process of toggling the music state in the chess application. It shows how the user interacts with the system to switch the music on or off, highlighting the communication between BoardView, BoardController, BoardModel and BoardListener.

**Board Toggle Music**

****

This sequence diagram describes the process of toggling the music state in the app application. It shows how the user interacts with the system to switch the music on or off, highlighting the communication between BoardView, BoardController, BoardModel and BoardListener.

## **User Documentation**

****

After the software has completed properly, the Kwazam Chess main screen will appear. The primary interface features four buttons: “Start”, “Load Gamet”, “Sound Effect” and “Music”. If the user presses the "Start" button, the Main Chess Board will display. By hitting the "Load game" button, the software will load the previous saved game including the time and who’s turn. User clicks on the “Sound Effect” and “Music” will turn off the sound effect of the moving chess sound effect and turn off the music of this software.



If the user clicks on the “Sound Effect” and “Music” again will turn on the sound effect of the moving chess sound effect and turn on the music of this software to improve game experience.

****

The Kwazam Chess Board is a 5x8 board and it shows 20 pieces of chess piece. Each side has 10 pieces. From the bottom left, the pieces are Xor, Biz, Sau, Biz, Tor. The arrows are called the Ram piece. The Ram piece can only move forward 1 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 Xor piece can move diagonally only but can go any distance. It cannot skip over other pieces. However, after 2 turns, it transforms into the Tor piece. It cannot skip over other pieces. The Sau piece can move only one step in any direction. The game ends when the Sau is captured by the other side. The game ends when the Sun is captured by the other side. The game is played by 2 players, one from the blue side and one from the red side. There also display a control panel on the top of the board which is “Music” and “Sound Effect” button, “Turn” sign, “Round” and “Time” display, “Save” and ”Quit” button.



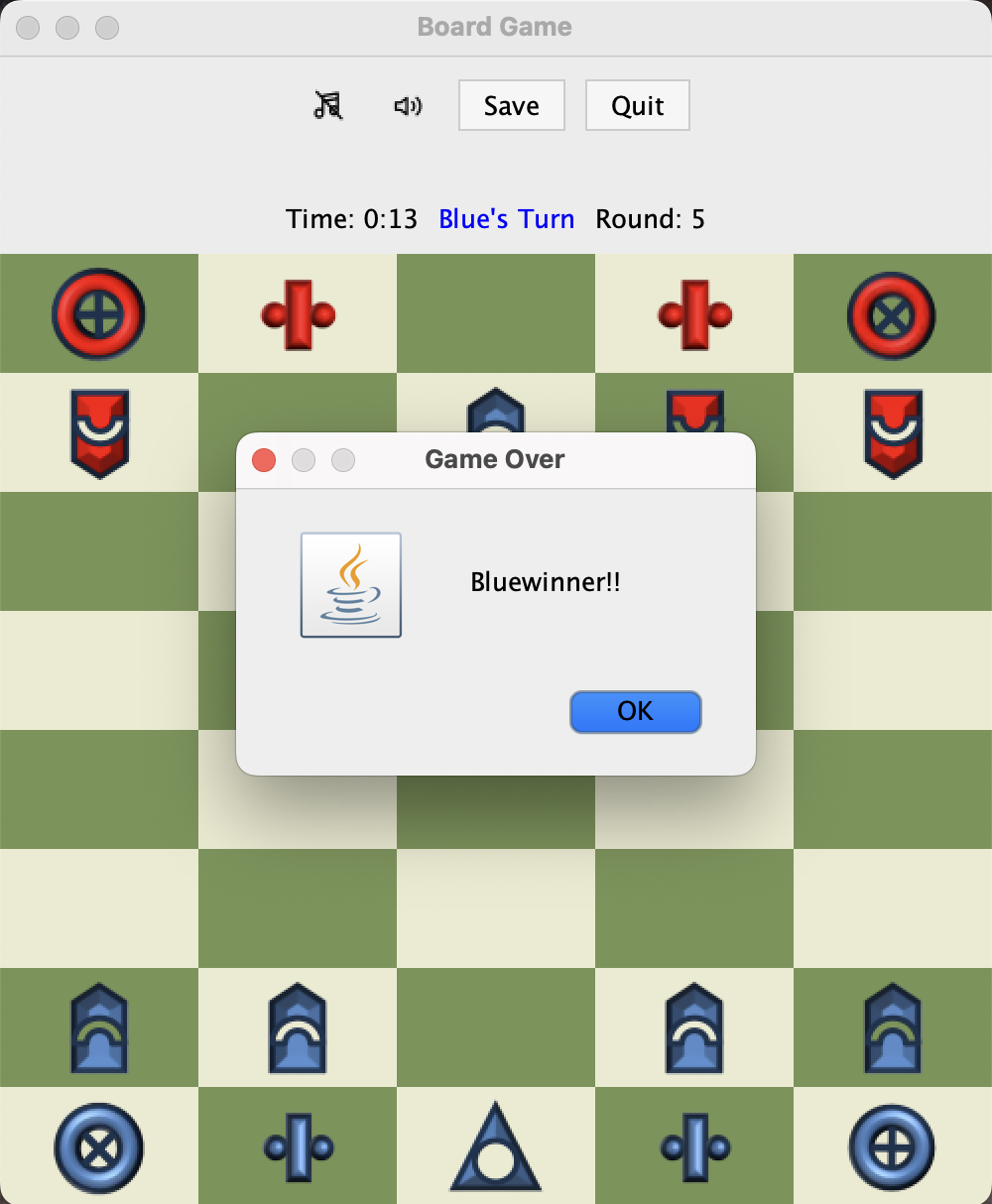
The user just has to click on the chess piece that they want to move. When the user clicks on a chess piece, the board will highlight the selected chess piece in blue frame and the boxes that it can move to will be highlighted in red.



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



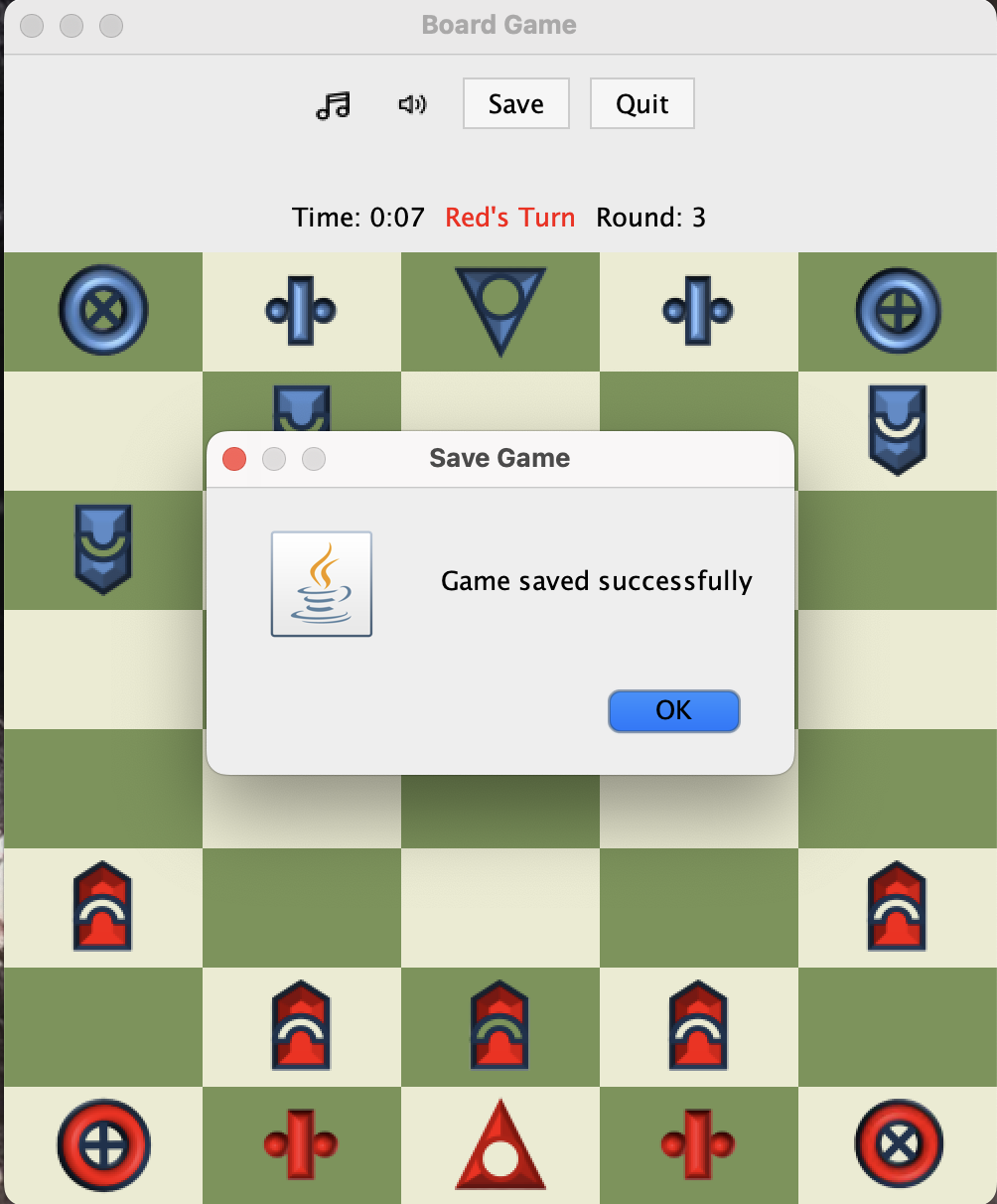
After 2 turns, counting one red move and one blue move as one turn, all Tor pieces will turn into Xor pieces, and all Xor pieces will turn into Tor pieces.



If the user successfully captures the Sau piece. A message “Blue 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”.



The user can quit the game in the middle of the game by clicking on “Quit”. After the program will navigate the user go back to menu interface.



The user can save the current state of the board by clicking on “Save Game”. After the user can also click on “Load Game’ to load the last game.