Object-Oriented Analysis and Design

TCP2201 Project

Trimester 2310 by Zainudin T17L Group E

Name	Email	Phone Number
TL: Lai Cal Wyn	1231303570@student.mmu.edu.my	011-10831797
Mishal Mann Nair	1221305145@student.mmu.edu.my	019-443 2285
Chan Ga Wai	1221305898@student.mmu.edu.my	011-6412 9980
Harrish Panicker	1221308940@student.mmu.edu.my	018-919 1495

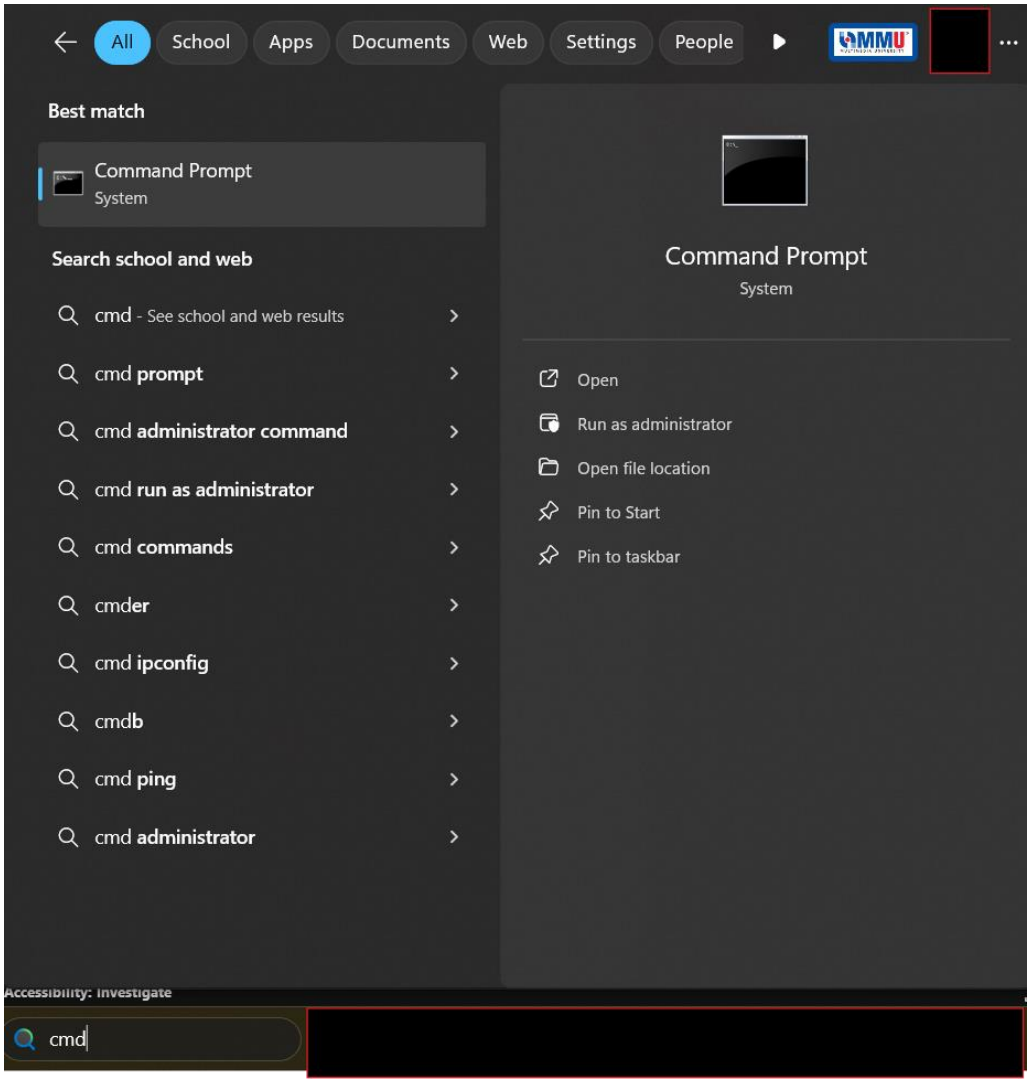
Table of Content

Compile and Run Instructions

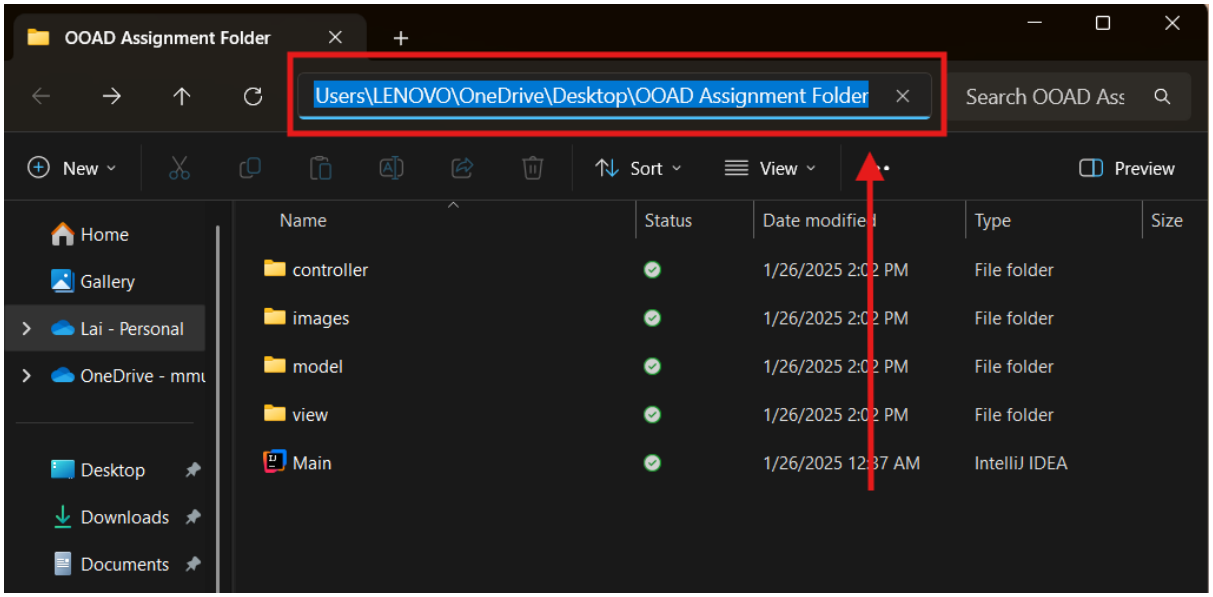
How to use from Command Line

Assumptions: You have the latest java or openJdk installed in your device.

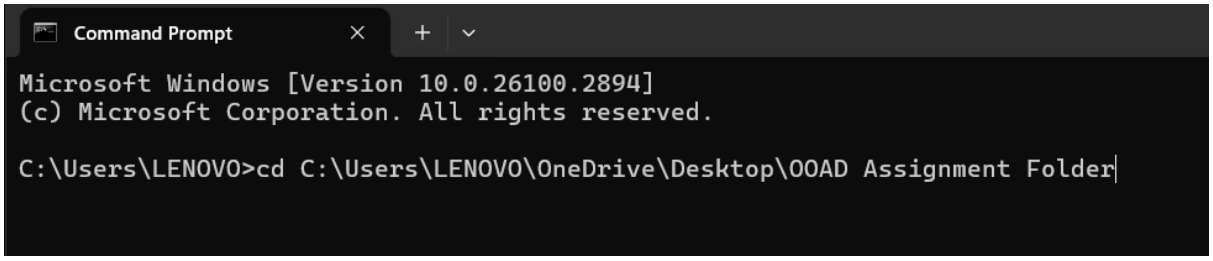
Step 1: Open your command line by pressing the (Windows + R) button, type “cmd” and hit Enter.



Step 2: Navigate to the directory where the assignment file is. You can do this by checking on the top of the bar and copy that.

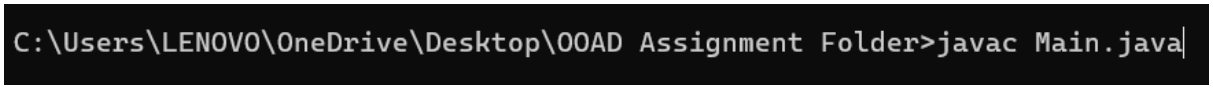


Step 3: Go back to the command line and type “cd” space follow by what you just copied.

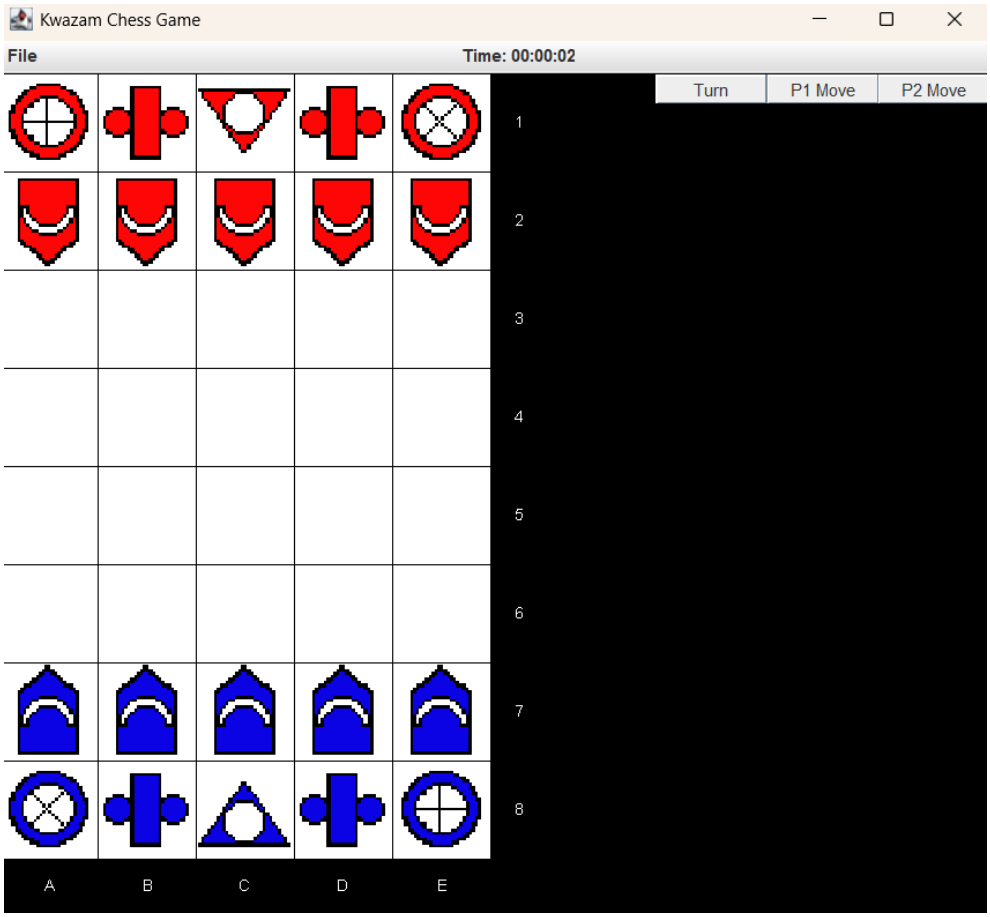


Step 4: Hit Enter and your command line should display the path following the assignment file.

Step 5: Type in the “javac Main.java” and press Enter.

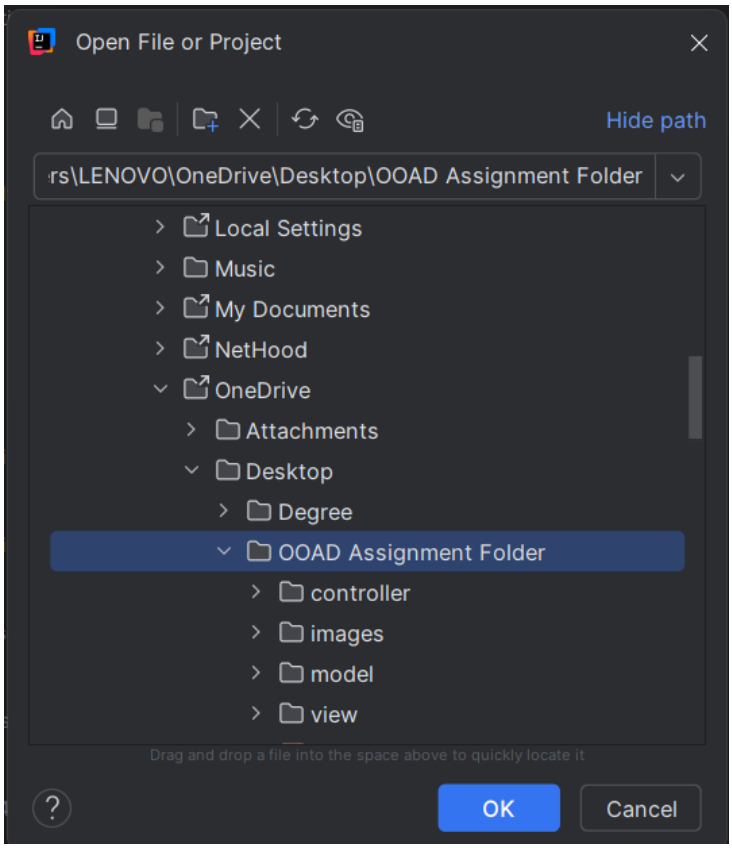


This will compile the Main.java file and show the Kwazam Chess application. As shown in the example below:

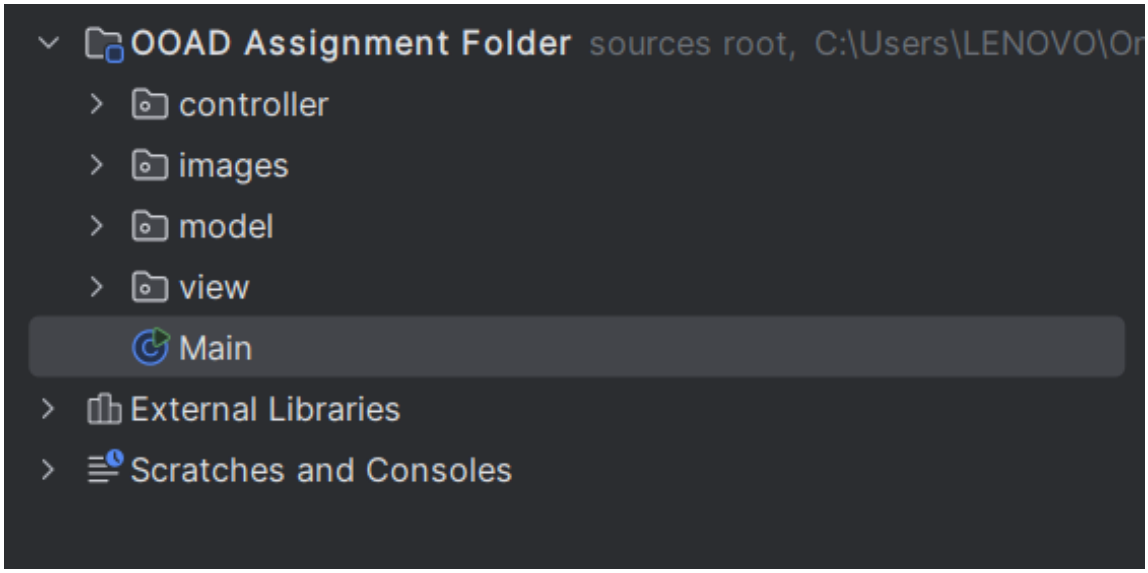

Kwazam Chess Game
File
Time: 00:00:02
Turn
P1 Move
P2 Move
1
2
3
4
5
6
7
8
A
B
C
D
E

How to use from the terminal in Intellij IDEA

Step 1: Open the Assignment folder.



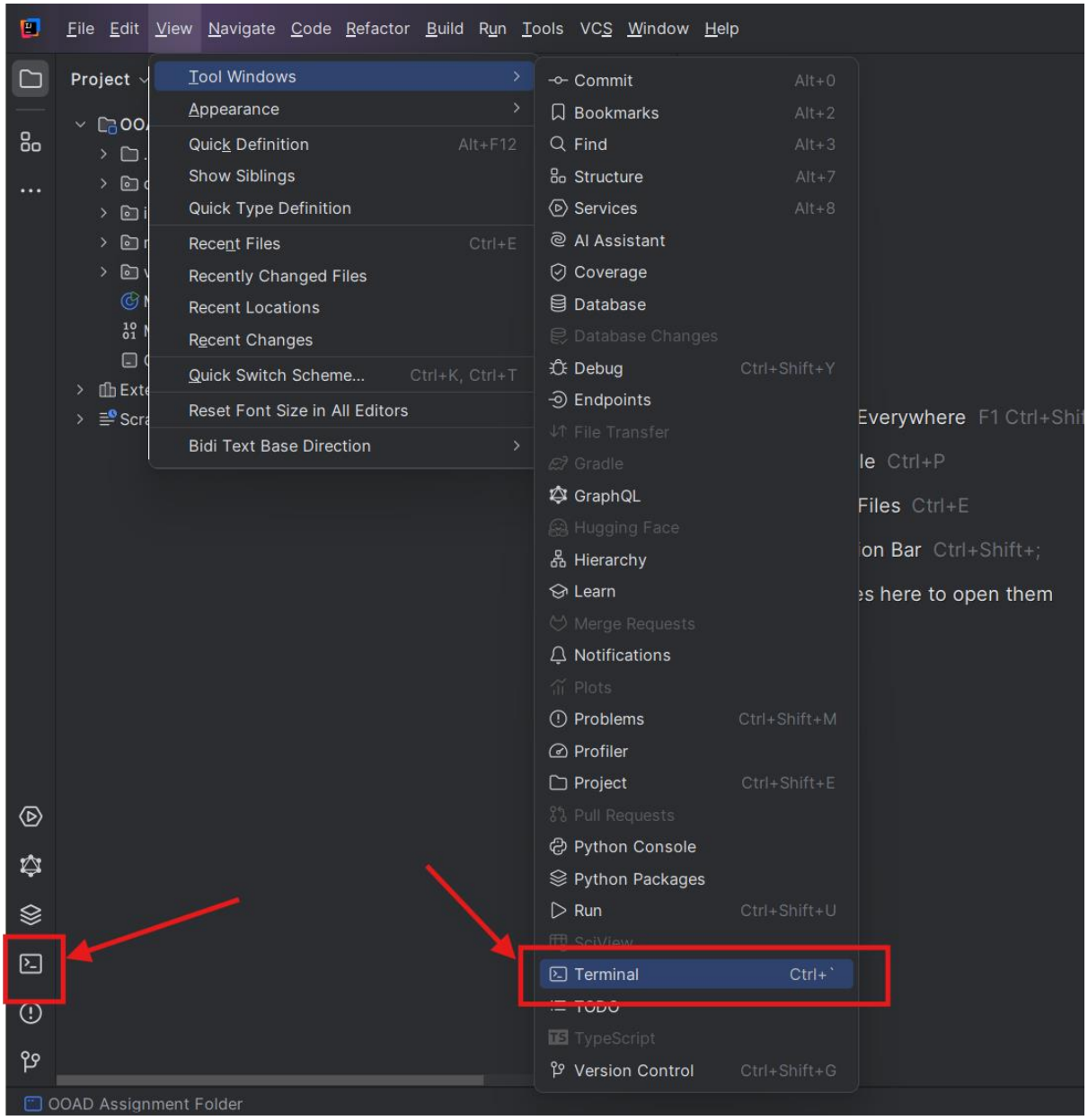
Step 2: Select the Main.java file.



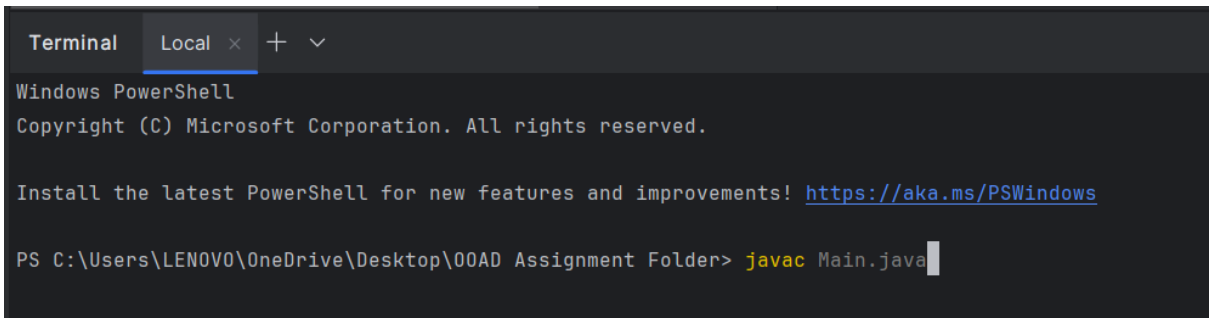
Step 3: Run the code

Or

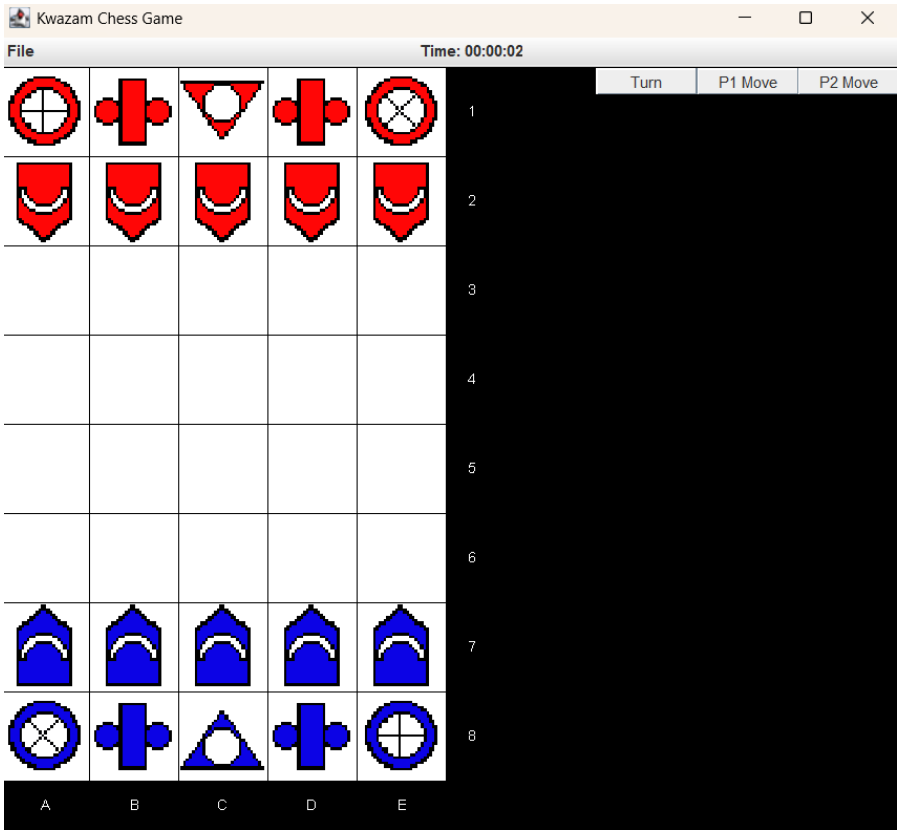
Step 2: Open the terminal using “ Ctrl + ‘ ” or select the highlighted areas.



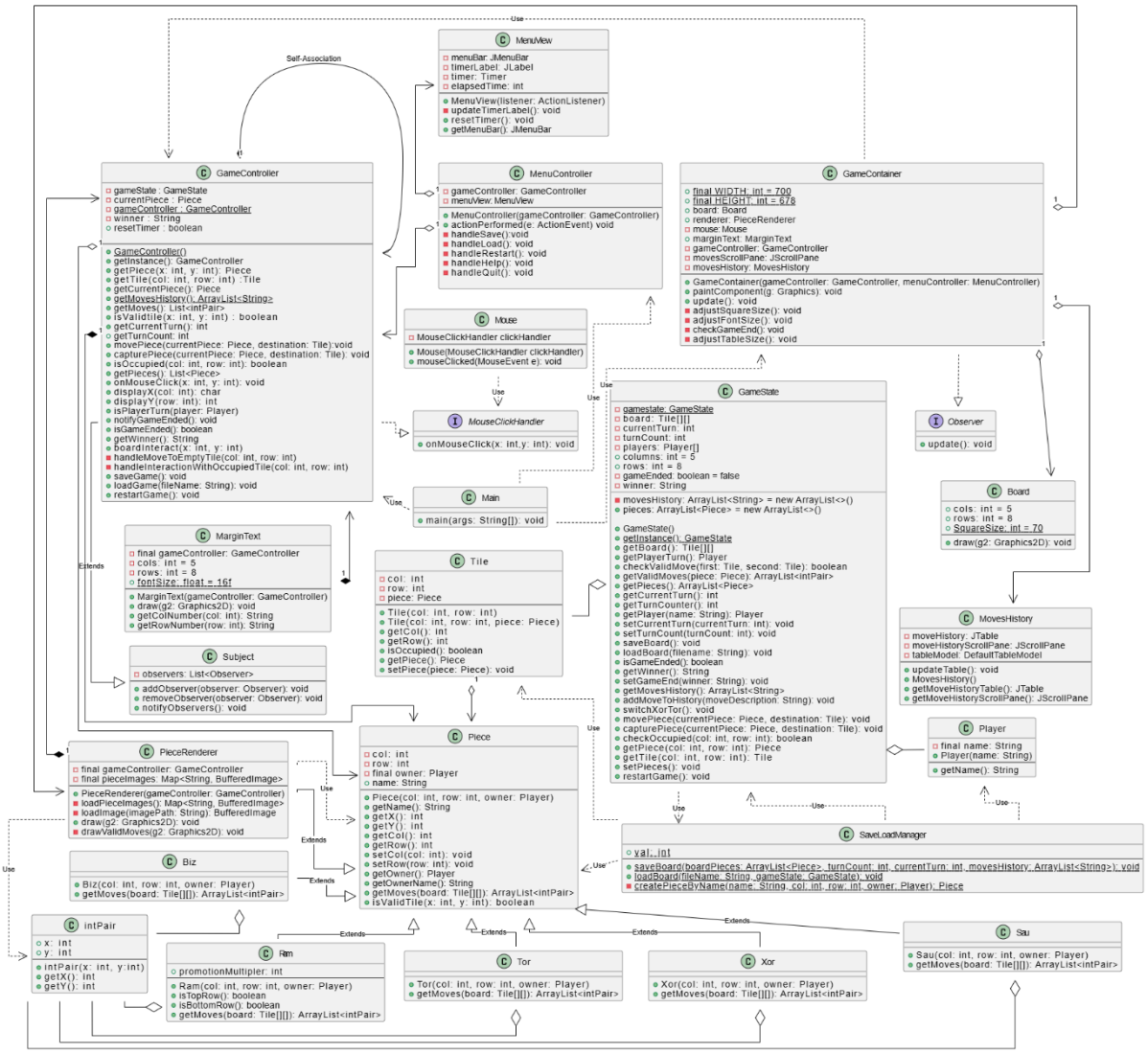
Step 3: Type in “javac Main.java” and press Enter.



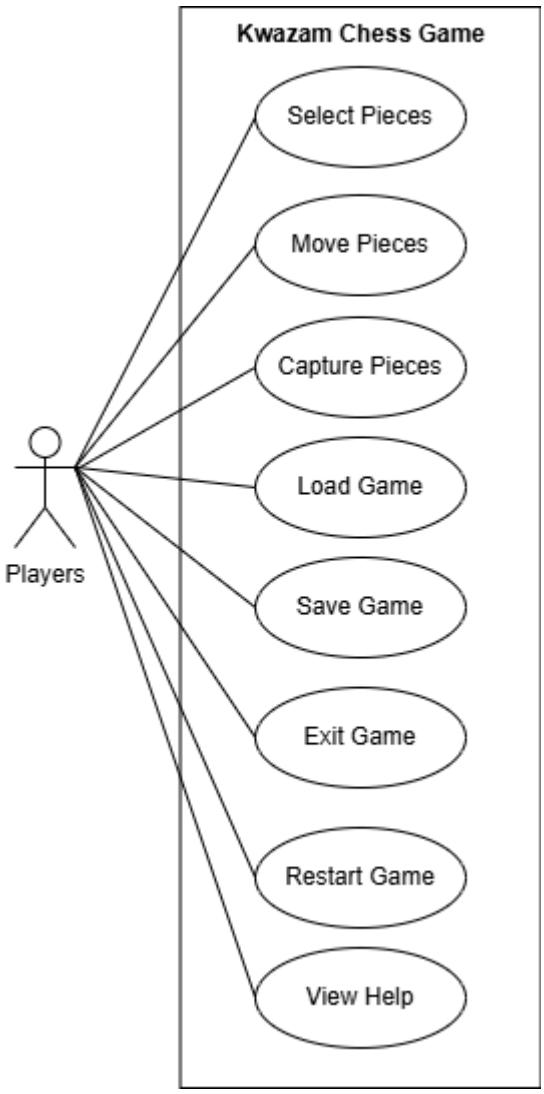
This will compile the Main.java file and show the Kwazam Chess application. As shown in the example below:


Kwazam Chess Game
File
Time: 00:00:02
Turn
P1 Move
P2 Move
1
2
3
4
5
6
7
8
A
B
C
D
E

UML Class Diagram

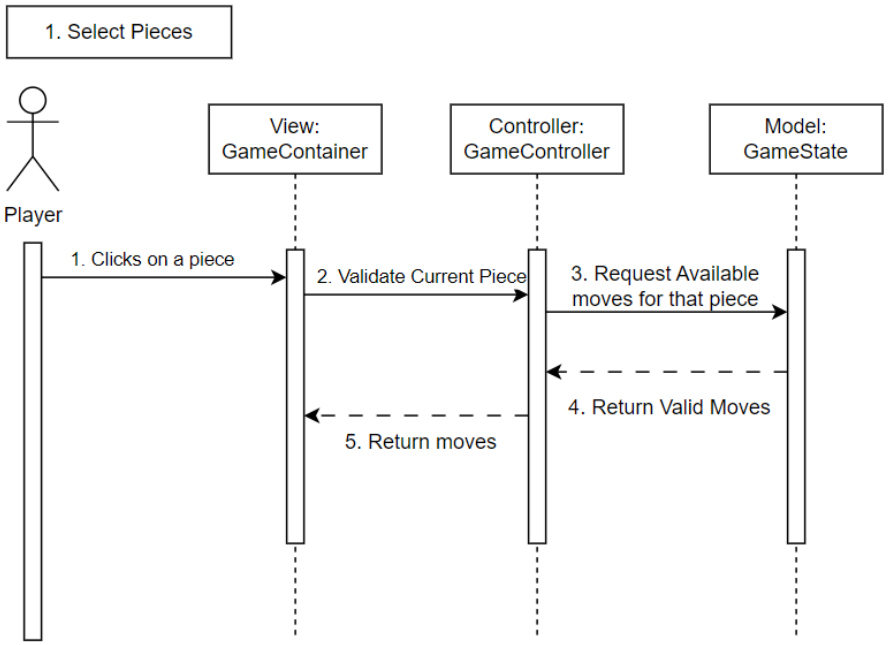


Use Case Diagram

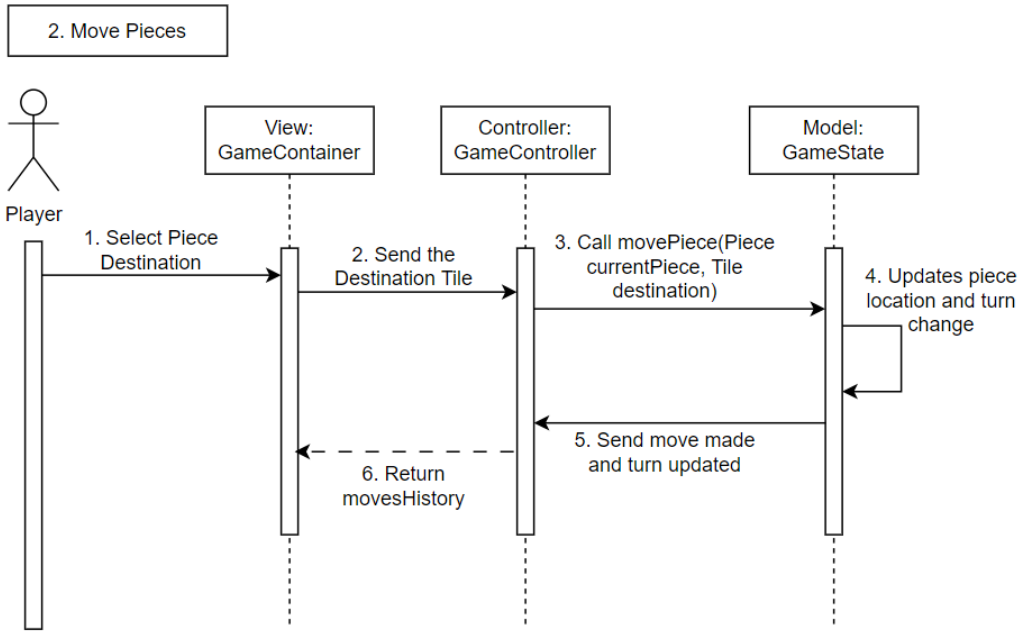


Sequence Diagrams

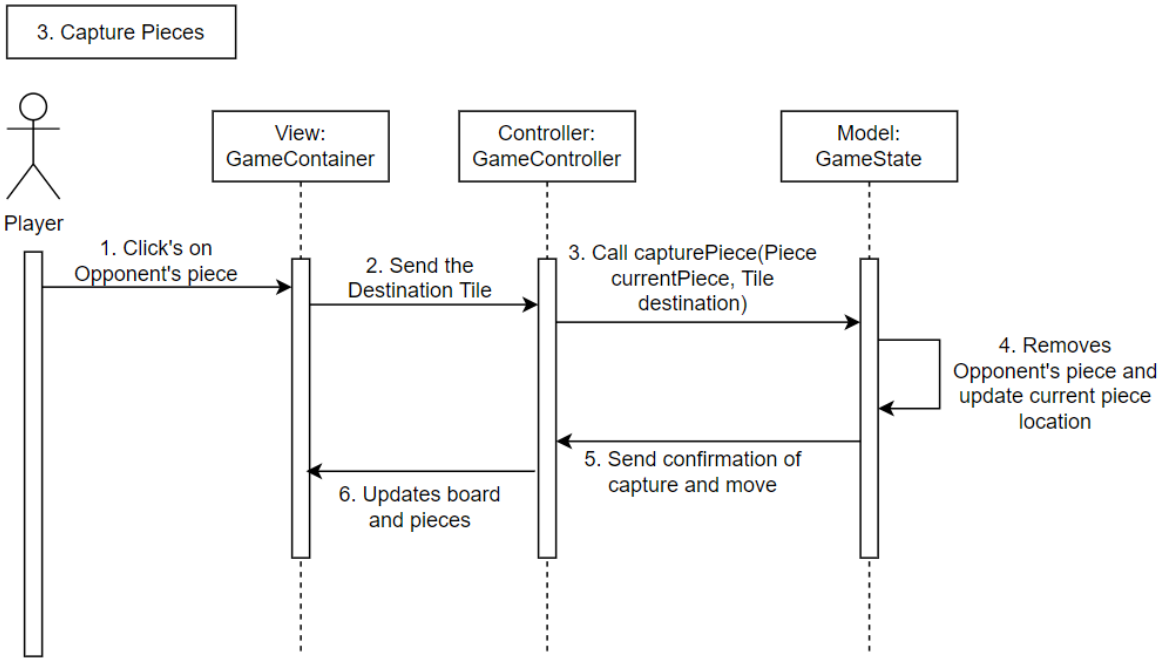
1. Select Pieces



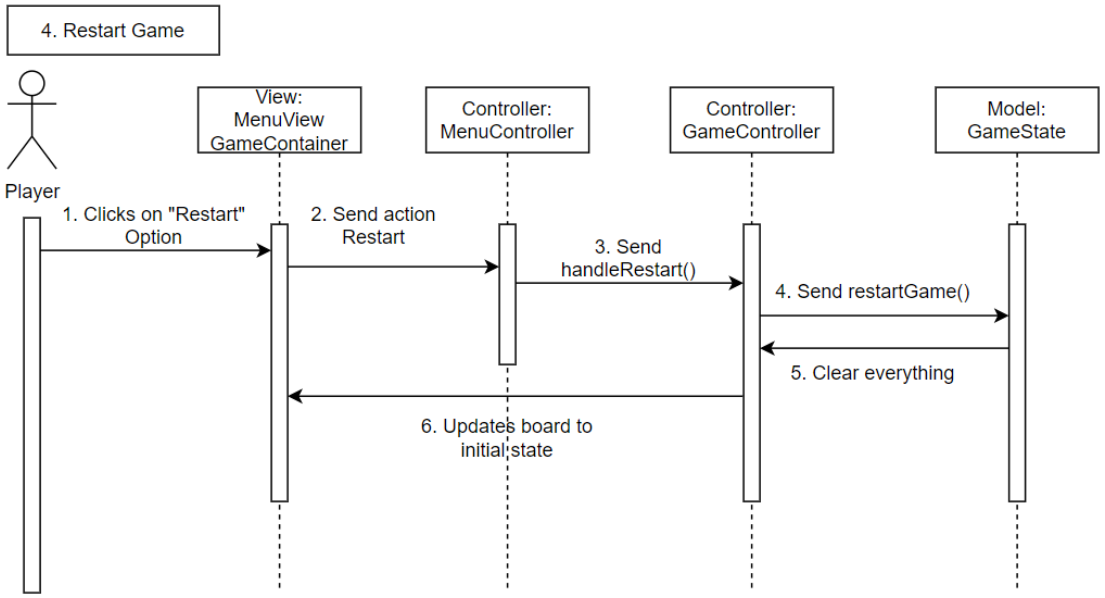
2. Move Pieces



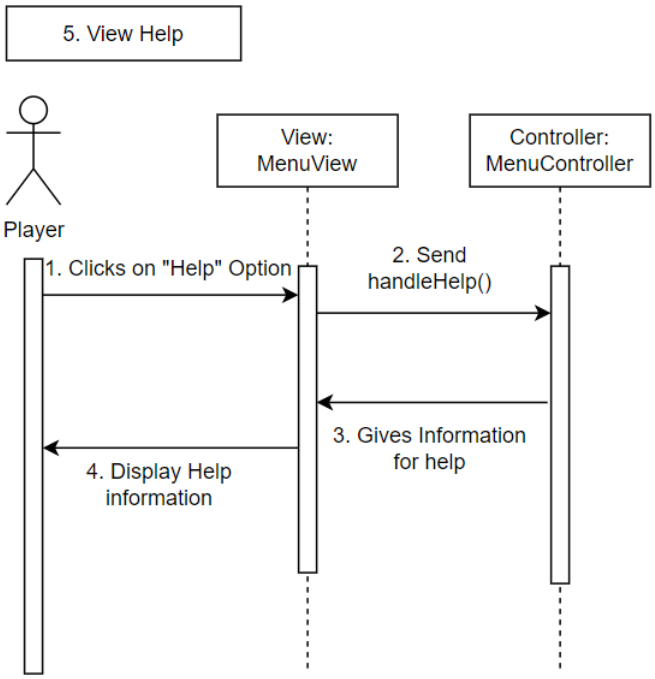
3. Capture Pieces



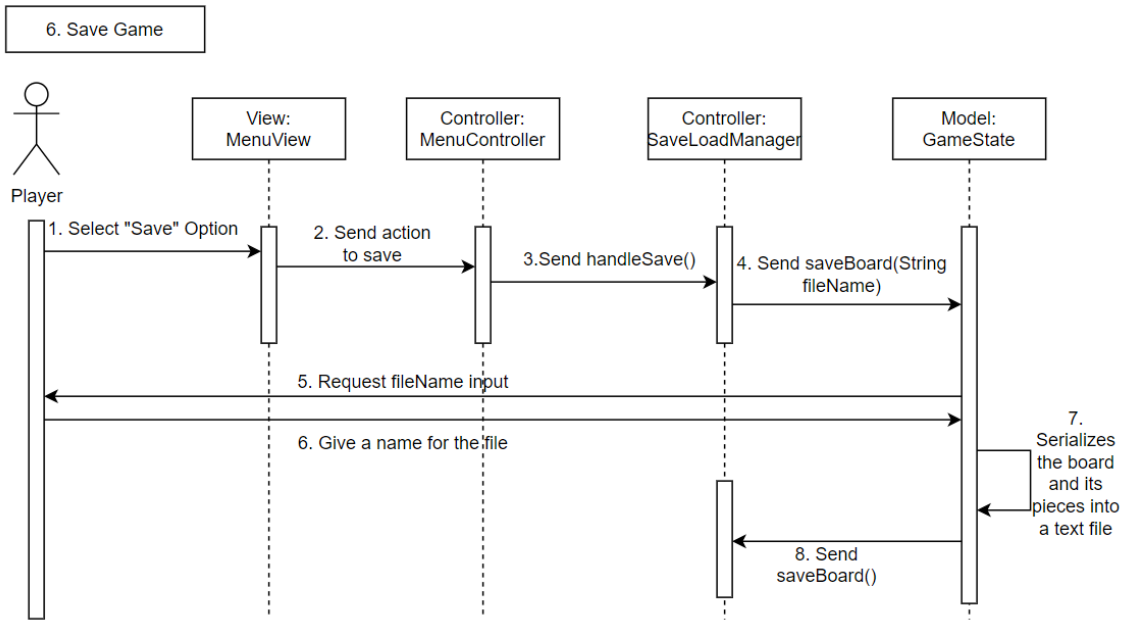
4. Restart Game



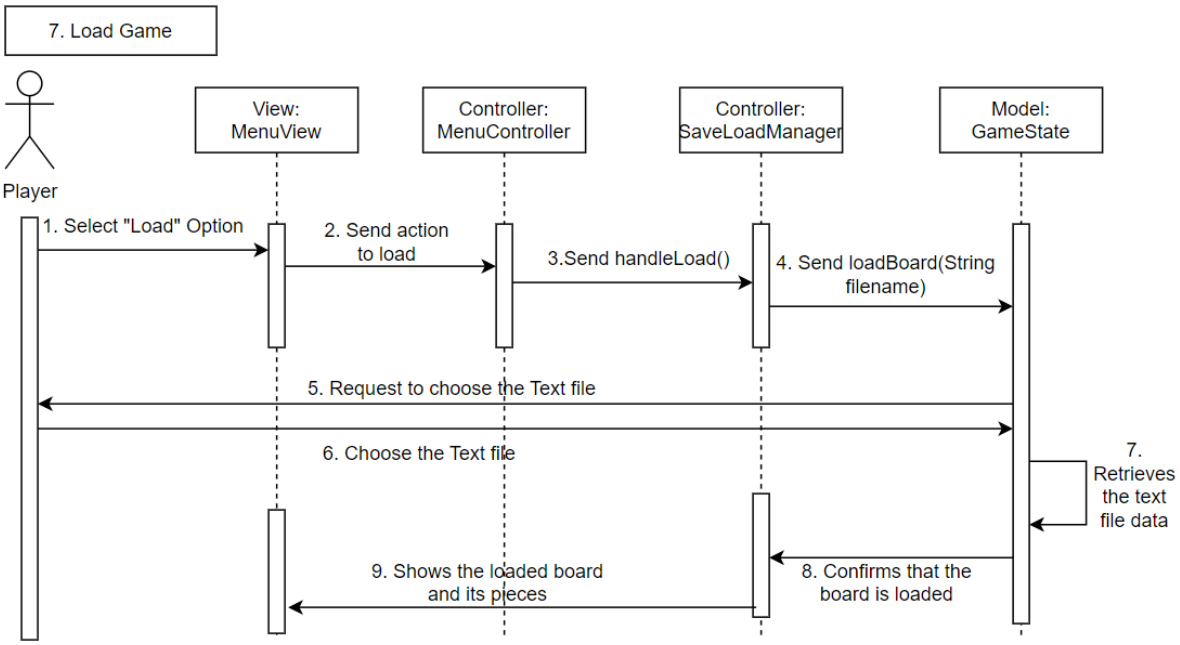
5. View Help



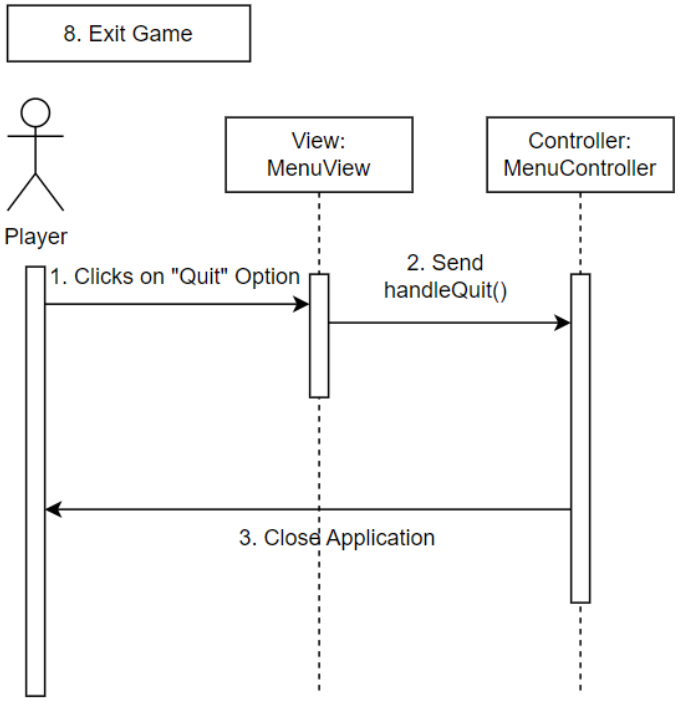
6. Save Game



7. Load Game


7. Load Game
Player
View:
MenuView
Controller:
MenuController
Controller:
SaveLoadManager
Model:
GameState
1. Select "Load" Option
2. Send action
to load
3.Send handleLoad()
4. Send loadBoard(String
filename)
5. Request to choose the Text file
6. Choose the Text file
7.
Retrieves
the text
file data
8. Confirms that the
board is loaded
9. Shows the loaded board
and its pieces

8. Quit Game


8. Exit Game
Player
View:
MenuView
Controller:
MenuController
1. Clicks on "Quit" Option
2. Send
handleQuit()
3. Close Application

Design Patterns

1. **Singleton Pattern:**

Implemented in the GameController class (not shown here but inferred by GameController.getInstance()). This ensures that only one instance of the GameController exists, providing a centralized point of control for game state management.

2. **Observer Pattern:**

Implemented in the relationship between GameContainer (as an observer) and GameController (as the subject). GameContainer listens for changes in GameController and updates the view accordingly using the update() method.

3. **MVC Pattern:**

The program follows the Model-View-Controller (MVC) architecture:

Model: Game state and pieces (e.g., Piece class, moves history in GameController).

View: Components like GameContainer, Board, MarginText, MenuView, and PieceRenderer, which handle visual representation.

Controller: GameController and MenuController, which manage user inputs and game logic.

4. **Adapter Pattern**:

The PieceRenderer acts as an adapter between the GameController's data (piece locations and types) and the graphical representation of pieces.

Subclassing, Delegation, Composition, and Aggregation

1. Subclassing:

GameContainer extends JPanel, inheriting behavior and properties while providing custom functionality like board rendering and resizing logic.

2. Delegation:

PieceRenderer and MarginText delegate rendering tasks to their respective components, avoiding a monolithic GameContainer class. The MenuView delegates save and load tasks to the MenuController via listeners.

3. Composition:

GameContainer composes multiple components (Board, PieceRenderer, MarginText, etc.), each responsible for a specific part of the view.

4. Aggregation:

MovesHistory aggregates the moves from GameController using a shared reference to display them in a JTable. Aggregation allows for loose coupling between components.

Polymorphism

Polymorphism is evident in methods like getColNumber and getRowNumber in MarginText, which provide different behaviors based on the player's turn. Additionally, the draw() methods in Board and PieceRenderer allow the Graphics2D object to be passed without knowing the exact drawing logic in advance.

Encapsulation

The GameController class encapsulates the game's state, exposing only necessary methods to manipulate or access data. Similarly, Board.SquareSize is encapsulated and adjusted dynamically based on the window size.

Abstraction

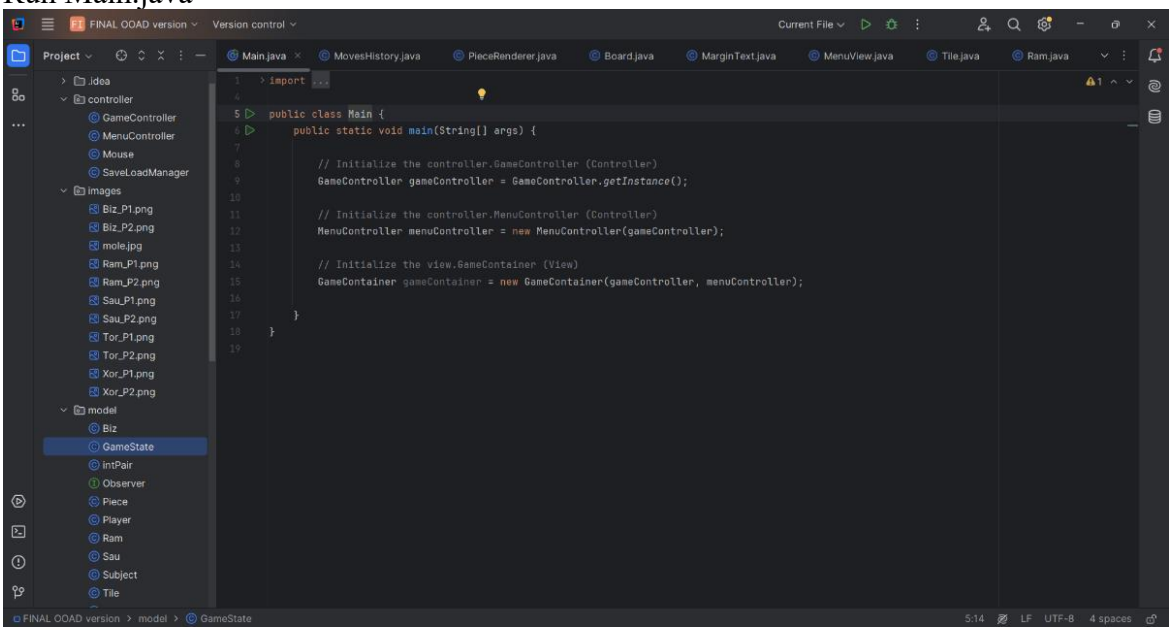
Abstracting game mechanics into GameController and visual details into components like Board and PieceRenderer separates concerns, making the program easier to maintain and extend.

Why?

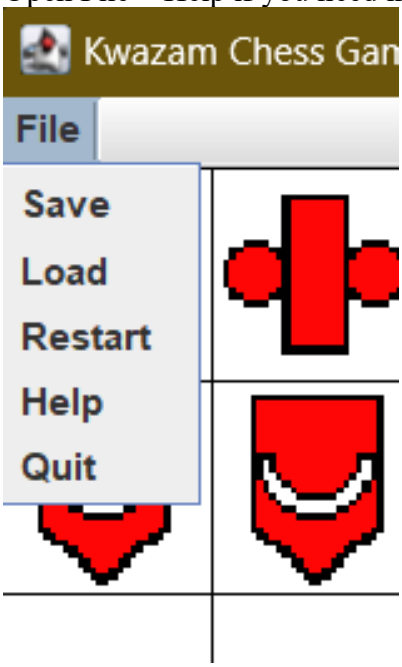
These design principles and patterns work together to create a robust, scalable, and maintainable application. By adhering to object-oriented principles like encapsulation and abstraction, the code achieves clarity and modularity. Patterns like MVC and Observer ensure clear separation of concerns, while mechanisms like delegation and composition reduce redundancy and enhance reusability. This approach ultimately enables flexibility for future features, like introducing new piece types or altering game logic.

User Manual

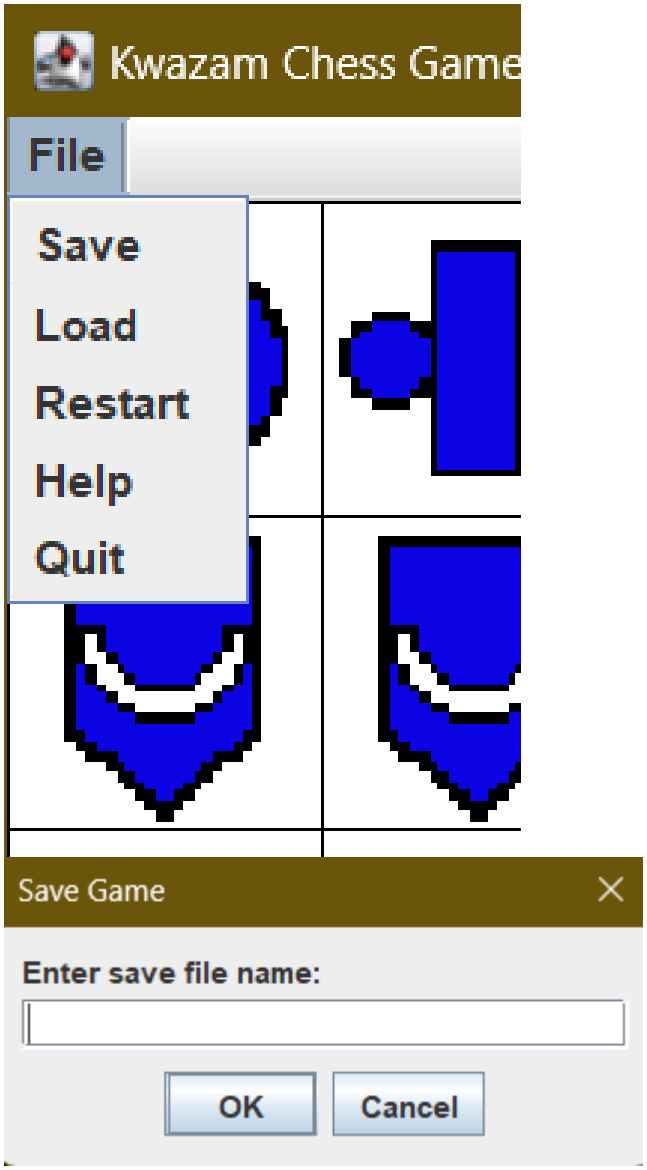
1. Open all the .java files in a compiler such as BlueJ or IntelliJ
2. Run Main.java



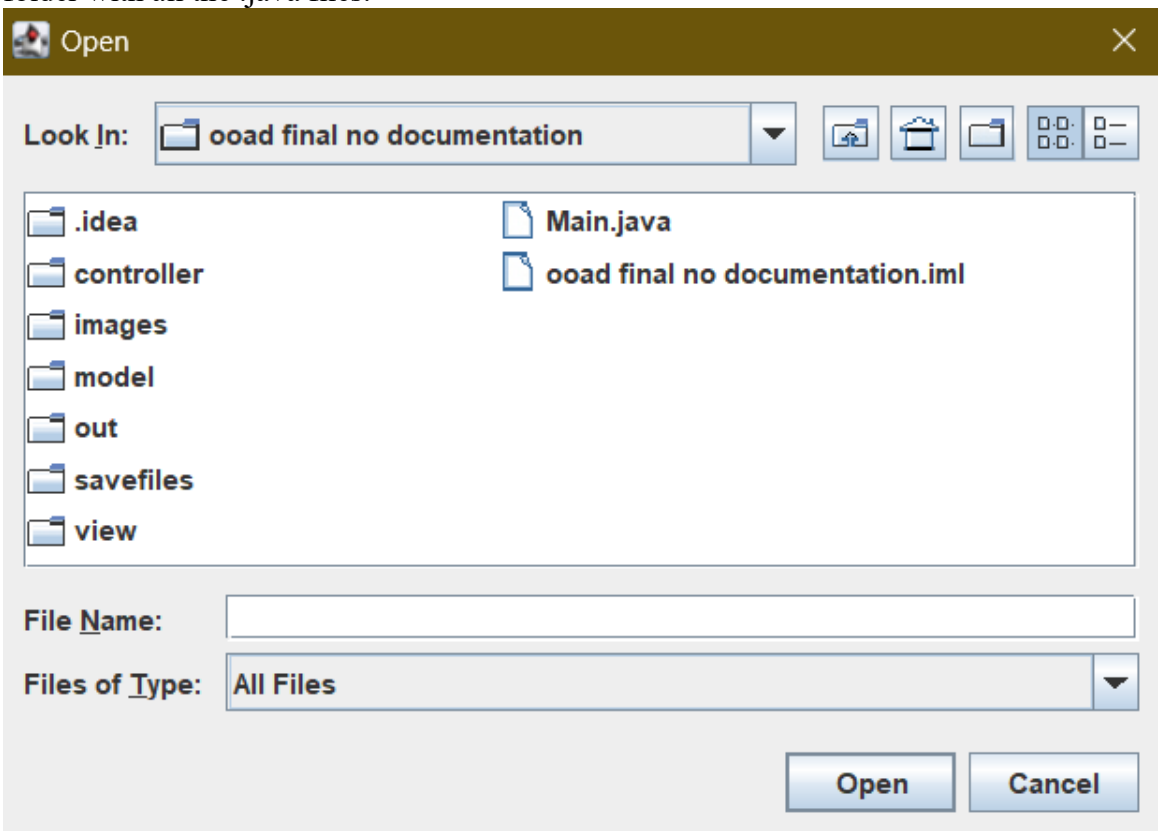
3. Open File > Help if you need instruction on how the game works



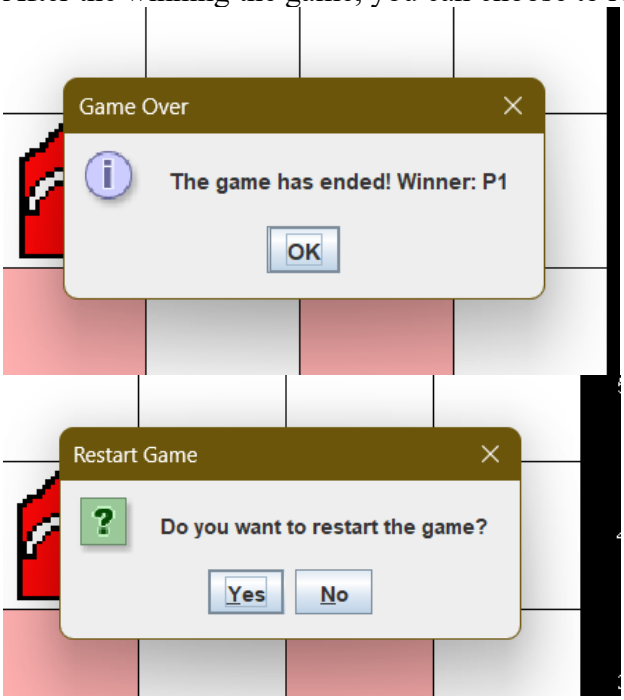
4. Open File > Save and then type in the file name you want to save it as when the prompts come out



5. If you want to load in the save file, open File > Load and find the file you named in step 4. The file should be in the format of “savefilename”.txt. The default save file is in the folder with all the .java files.



6. After the winning the game, you can choose to restart the game.



7. You can also choose to restart the game by opening File>Restart.
8. To exit the game, you can either press the “X” on the top-right corner or you can open File>Quit