LOVELY PROFESSIONAL UNIVERSITY

JAVA-PROJECT (CSE-310):

ChessMaster: A Java based Chess Game with GUI

SUBMITTED TO: Dr Ranjith Kumar

SUBMITTED BY: Vansh Pratap Chauhan (12110117)

SECTION: K21ST **ROLL NO:** RK21STA24

❖ Introduction:

The goal of this project was to develop a chess game using Java programming language. The purpose of creating this game was to gain experience in software development using object-oriented programming principles and to learn how to implement game logic.

Chess is a classic board game that has been enjoyed by millions of people for centuries. It is a game that requires strategic thinking and planning ahead, making it an ideal choice for developing programming skills. With this project, we aimed to create a functional and user-friendly chess game that players of all skill levels could enjoy.

Throughout the development of this game, we hoped to learn more about programming concepts such as inheritance, polymorphism, and encapsulation, as well as gain experience in implementing game logic, user interfaces, and testing methodologies. In the following sections, we will provide an overview of the features of our chess game, discuss the implementation details, and provide results of our testing and evaluation. We will also discuss future work and potential improvements for the game.

Project Overview:

Our chess game is a fully functional two-player game that follows the standard rules of chess. The game has a graphical user interface (GUI) that allows players to interact with the game board and pieces. The game features include:

- Chess board: The game board is a standard 8x8 chessboard with black and white squares. We used a two-dimensional array to represent the game board.

- Piece movement: We implemented the movement rules for all six types of chess pieces - pawn, knight, bishop, rook, queen, and king. The pieces move in accordance with the standard rules of chess.

- Capturing: When a player moves a piece to a square occupied by an opponent's piece, that piece is captured and removed from the board.

- Check and checkmate: The game detects when a player's king is under attack and if there is no legal move to get

the king out of check, the game is over and that player is checkmated.

- En passant: We implemented the En passant capture rule for pawns that move two spaces on their first move and land next to an opponent's pawn.

- Game states: The game keeps track of the current state of the game, including whose turn it is and whether the game has ended.

- GUI: We created a user-friendly graphical interface that allows players to click on a piece and then click on the desired square to make a move. The GUI also displays game information, such as whose turn it is and any captured pieces.

Implementation Details:

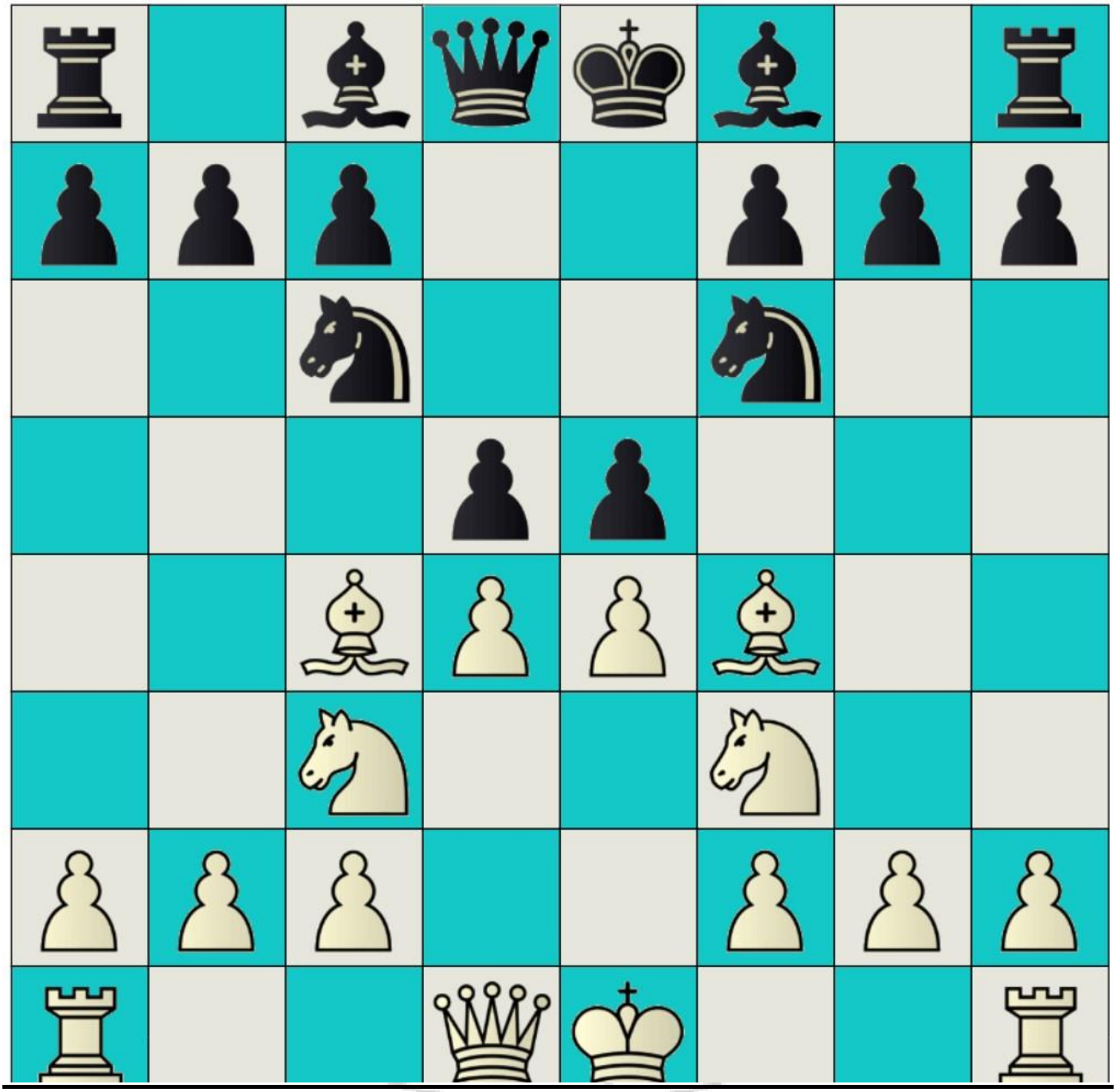
- Our chess game was implemented using Java programming language and followed the object-oriented principles of encapsulation, inheritance, and

polymorphism. The game's implementation was divided into three main components: the model, view, and controller.

- The model component contained the game logic and data structures used to represent the game board, pieces, and their movements. We used a two-dimensional array to represent the game board, and each chess piece was represented as an object of its respective class. We used inheritance to represent the shared properties and behaviours of different pieces, and polymorphism to represent the unique characteristics of each piece type.

- The view component contained the graphical user interface (GUI) used to display the game board and pieces to the player. We used Java Swing to create the GUI, and the view was updated based on the current state of the game in the model component.

This Is How The UI Looks Like:-



Conclusion:

In conclusion, our Java-based chess game successfully implemented the standard rules of chess and provided a user-friendly graphical interface for players to interact with. We followed object-oriented principles of encapsulation, inheritance, and polymorphism to create a well-organized and maintainable codebase. The game's model-viewcontroller architecture allowed for clear separation of concerns and improved maintainability.

Looking forward, there are various possible directions for future improvements to the game. For example, we could expand the AI opponent's capabilities to make it more challenging for players. We could also implement additional game modes or difficulty levels to provide more variety for players. Additionally, we could incorporate a database to allow players to save and load games.

Overall, we are proud of the work we have done on this project and believe that it provides a fun and engaging experience for players. We hope that this project will inspire others to explore the possibilities of Java programming and game development.