Maharaja Agrasen Business School

PYTHON MINI PROJECT GAME

"CHESS"

Submitted by:

Sakshi Saraiya- 24011

Khushi Goyal- 24034

Krish Garg- 24042

Ankit Mittal- 24055

Tanisha Singhania- 24057

Introduction

The Chess Game project is a full implementation of the iconic strategy board game, Chess, to be played in Python. The project shows real-world uses of Object-Oriented Programming (OOP) concepts, including class hierarchies, inheritance, and polymorphism. It also underscores fundamental programming ideas like input checking, real-time interaction, and game logic. The aim of this report is to record the project design, development, and implementation along with emphasizing the problems encountered, technical solutions adopted, and future improvements.

Problem Statement

The game of chess is a board game for two players, on an 8x8 grid, where both players have 16 pieces: pawns, rooks, knights, bishops, a queen, and a king. The objective is to checkmate the other player's king, i.e., to place the king in a way that it cannot move away from capture. In this project, the task is to design a complete, functional, and interactive game of chess that:

• Guarantees and enforces the movement rules for every chess piece.

• Manages turns between players, toggling between the white and black sides.

• Identifies check and checkmate situations.

• Offers a user interface where users can engage with the game via basic text-based commands.

In the process, the test is to adequately portray the complicated rules of chess, such as how the pieces move, how they affect each other, and how to conduct a valid state of the game during the gameplay.

Objective

The main goal of this project is to create a text-based chess game in Python through which two players can play the game between them. The game must follow the basic rules of chess and implement functionalities like:

- **Legal Move Validation**: Ensuring that every piece moves as per its respective rules.
- **Turn-Based Gameplay**: Switching turns between the two players.
- **Check and Checkmate Detection**: Detection of when a player's king is in danger (check) and when it is checkmated.

- **Interactive User Interface**: A minimal text-based UI for entering moves and giving feedback.

The project is also intended to show the application of OOP concepts in implementing a real-world application that manages a complex game state and applies rules.

Reason for making chess

The reason for selecting Chess for this project is based on several factors:

- **Logical Complexity**: Chess requires both straightforward rules and intricate strategic moves. Symbolic representation of this logic makes a perfect context for showing and honing programming and problem-solving capabilities.
- **Use of OOP**: Chess has an inherent inclination toward Object-Oriented Programming concepts. Each chess piece on the board can be viewed as an object with different behaviours, which can be programmed using classes and inheritance.
- **Interactive Gameplay**: A text-based interface offers a simple yet efficient way to mimic real-time interaction within the game, making it simple to test and demonstrate.
- **Real-World Use Case**: Creating a chess game provides good experience in handling real-world game logic, managing user input, and maintaining consistency in game rules.
- **Foundation for Future Development**: This project is a step towards developing more advanced chess applications, for example, AI players or Graphical User Interfaces (GUIs).

Tools and Technologies

In developing this project, the following tools and technologies have been utilized:

• **Python 3**: The game is developed completely in Python, a general-purpose language that is famous for its simplicity and readability. Python also has good support for OOP, which makes it perfect for this project.

• **Jupyter Notebook**: The development was done in a Jupyter Notebook to give an interactive coding environment that enables testing each function and seeing the results immediately.

• **Pygame** : Even though the present implementation is text-based, Pygame can be utilized in the future to create a graphical user interface (GUI) for the game.

• **Libraries**: Python's list, dictionary, and exception handling modules were employed to handle the board, pieces, and game logic.

Code Architecture

The structure of the Chess Game is made to adhere to a modular and object-oriented style:

- **Piece Class**: This is the base class that all chess pieces inherit from. It includes general methods for setting piece position, validating moves, and tracking piece state.
    - valid_move() is an abstract method that will be overridden by subclasses for specific piece movements**.**
- **King, Queen, Bishop, Knight, Rook, Pawn Classes**: These are subclasses of the Piece class. Each class defines the movement logic for that piece. For example:
    - The Knight class overrides valid_move() to verify an "L-shaped" move.
    - The Queen class incorporates the reasoning for both Rook and Bishop movement.
- **Board Class**: This class creates an 8x8 grid (list of lists in Python) that simulates the chessboard. It records the position of pieces and refires the board after a move. It has methods to:
        - Initialize pieces to their initial positions.
        - Display the board to the user.
        - Update the board after a piece has moved.
- **Game Logic**: The game logic controls player turns, checks for valid moves, and checks for special situations such as check and checkmate. It also monitors the game state and deals with invalid input.

Key Functionalities

• **Board Initialization**: The game begins with the board showing the normal initial setup. Every piece is placed in its proper position.

• **Move Validation**: For every move, the system validates if the move is valid for the chosen piece. It verifies that the move does not break the piece's movement rule (e.g., bishops can move diagonally, pawns can move forward, knights move in an L-shape).

• **Turn Handling**: The game rotates between the two players. White always initiates, and the turn switches to the opponent after every valid move.

• **Check Detection**: When a player's move puts the opponent's king in check, a message is shown, warning the player.

• **Game Continuity**: The game goes on until a checkmate or stalemate situation arises, or the player chooses to quit.

Sample Output

Here's an example of what some game outputs would be:

- **Initial Board State**: A depiction of the chessboard with all pieces in their initial positions.
- **Following a few moves**: The board is refreshed following a player's move. E.g., pushing a pawn ahead.
- **Invalid Move**: An announcement that a move is illegal, e.g., a bishop attempting to move as a rook.

Challenges Faced

• **Complicated Movement Logic**: Making sure that each piece was moved per the chess rules was one of the primary challenges. For example, knights have unique movement rules which involved additional logic for verification.

• **Turn Management**: Making sure players switched turns and could not make two moves in succession was difficult, particularly with simultaneous input.

• **Game State Verification**: After every move, the game needed to be verified for such conditions as check, checkmate, or stalemate.

• **Error Handling**: The system had to process invalid inputs (e.g., a player trying to enter an out-of-range square) gracefully.

Learnings

This project gave me several useful lessons:

• **Object-Oriented Programming**: How to use OOP principles in real life, including inheritance, encapsulation, and method overriding.

• **Game Logic**: Creating and implementing rules for a game as intricate as chess showed me how to divide big problems into smaller, manageable pieces.

• **Debugging**: Debugging was extremely important, as minor errors in logic or handling of input had the potential to easily crash the game.

• **User Experience**: Making sure the game was fun and easy to play and error messages were concise and helpful was a key consideration.

Future Scope

The future for this project would include several improvements:

• **Graphical User Interface (GUI)**: Pygame or Tkinter can be employed to draw the chessboard and pieces graphically, with interactive capabilities using a mouse instead of text inputs.

• **AI Opponent**: The program can be modified to enable one player to compete against the computer, employing Minimax algorithms or others to achieve simulated intelligent playing.

• **Multiplayer Mode**: Incorporating network feature to enable two players to engage remotely via the internet.

Conclusion

The Chess Game project was able to implement a two-player chess game using Python. It gave hands-on practice with OOP concepts, game creation, and handling of input. Now that the game is fully operational, the next thing is to enhance the user interface and introduce more advanced functionalities such as AI and multiplayer.