# PROJECT REPORT

CSM216 (PYTHON PROJECT)

(CHESS GAME)

COMPUTER SCIENCE AND ENGINEERING

#### Submitted by:

T Satyanarayana Reddy

12303112

K23CH – G2 ROLL NUMBER: 48

#### Submitted to:

Mr. AMAN KUMAR

LOVELY PROFESSIONAL UNIVERSITY



# ACKNOWLEDGEMENT

I, T Satyanarayana Reddy, a student of Bachelor of Technology under Computer Science and Engineering discipline with Data Science and Machine

Learning specialization at Lovely Professional

University, Punjab, hereby declare that all the information furnished in this project report is based on my own work and is genuine.

T Satyanarayana Reddy 12303112

**TABLE OF CONTENTS**

|  |  |
| --- | --- |
| **1.Introduction** | 4 |
| 2.Objectives and scope of the project | 4 |
| 3.Application tools | 5 |
| 4.Project Design | 6 |
| 5.Flow Chart | 8 |
| 6.Project Implementation | 9 |
| 7.Testing and Validation | 12 |
| 8.Conclusion | 13 |
| 9.References | 14 |

### INTRODUCTION

The Chess Game using Tkinter is a Python project developed to simulate the traditional board game of chess in a digital format. The project combines the timeless appeal of chess with the versatility of Python's Tkinter library, creating an engaging, interactive, and visually appealing platform for players to enjoy. The primary aim of this project is to provide a user-friendly environment where players can experience the strategic depth of chess while exploring the possibilities of programming and GUI development.

Chess, being one of the oldest and most intellectually stimulating board games, requires a strong grasp of strategy, foresight, and decision-making. By developing this game digitally, the project bridges traditional gaming with modern technology, making chess more accessible and interactive. This implementation adheres strictly to the official rules of chess, ensuring a realistic experience for players.

The chess game offers the following benefits:

* Educational Value: It serves as an excellent learning tool for understanding chess strategies and Python programming concepts.
* Interactive Design: Provides an intuitive interface, enhancing the user experience.
* Logical Problem-Solving: Encourages strategic thinking by adhering to the standard rules of chess.

The game is designed for two players who can alternately make moves, with features like piece validation, movement constraints, and capture mechanics being accurately implemented. Tkinter's capabilities are utilized to create a graphical interface that features an 8x8 chessboard and distinguishable chess pieces, enhancing the visual and interactive appeal of the game.

The Chess Game project also serves as a platform to demonstrate key programming concepts, including object-oriented programming, event-driven programming, and algorithmic logic. It highlights the integration of programming with design to achieve a cohesive and functional product.

This report provides a comprehensive overview of the Chess Game project, detailing its objectives, design methodology, implementation process, challenges faced, and future scope. Through this project, developers and users can appreciate both the complexities of chess as a game and the nuances of developing a software application.

### OBJECTIVES AND SCOPE OF THE PROJECT

The Chess Game using Tkinter aims to bring the traditional game of chess to a digital platform, combining the intellectual challenge of chess with the flexibility and simplicity of Python programming. The project focuses on delivering a functional, visually appealing, and user-friendly chess application. Below, we detail the objectives and scope of this project.

## Objectives

The primary objectives of the Chess Game project are as follows:

* Digital Chess Platform: Develop a digital version of chess that adheres to official rules and provides an engaging gameplay experience.
* User-Friendly Interface: Design a simple yet visually appealing GUI using Python’s Tkinter library to ensure accessibility for players of all ages and skill levels.
* Chess Rule Enforcement: Implement the full set of chess rules, including piece movements, check, checkmate, and special moves such as castling, en passant, and pawn promotion.
* Interactive Gameplay: Provide interactive features like move validation, highlighting of legal moves, and real-time updates to the game state.
* Educational Value: Offer a platform for players to learn and practice chess while also serving as a learning project for programming and software development.
* Scalability: Lay the foundation for future enhancements, such as AI opponents, online multiplayer support, or advanced game analysis.

## Scope of the Project

The Chess Game project has a well-defined scope that emphasizes its usability, technical complexity, and future potential.

##### Functional Scope:

* + Development of a fully functional two-player chess game.
  + Implementation of all standard chess rules and mechanics.
  + Ability to detect and display check and checkmate conditions.
  + Real-time updates to the game board with piece movements and captures.

##### Technical Scope:

* + Utilization of Python for backend logic and Tkinter for the GUI.
  + Application of object-oriented programming (OOP) to structure game elements like pieces, the board, and game logic.
  + Event-driven programming to handle user interactions such as piece selection and moves.

##### Design Scope:

* + Creation of a visually appealing chessboard layout with clearly distinguishable pieces.
  + Highlighting features for valid moves and selected pieces to enhance user experience.
  + Support for error handling, such as invalid moves or rule violations.

##### Educational and Developmental Scope:

* + Provides hands-on experience with Python programming, GUI design, and game development.
  + Encourages logical thinking and problem-solving through the implementation of complex game mechanics.

##### Future Expansion:

* + Possibility to integrate AI-based opponents using algorithms like Minimax.
  + Adding online multiplayer capabilities for remote gameplay.
  + Inclusion of a replay or game analysis feature to study completed games.

### APPLICATION TOOLS

The development of the Chess Game application utilizes a combination of tools and technologies to ensure efficient implementation and optimal gameplay experience. These tools include:

##### Programming Language:

* + Python**:** The core programming language used for its simplicity, versatility, and strong community support, making it ideal for game development.

##### Game Development Library:

* + Pygame**:** A powerful library that provides functionalities for game development, including graphics rendering, sound management, and event handling.

##### Development Environment:

* + Integrated Development Environment (IDE**):** Tools like PyCharm, VS Code, or IDLE can be used to write, debug, and test the Python code effectively.

##### Graphic Assets:

* + Image Files:Custom graphics for the snake, food items, and background can be created or sourced from online repositories to enhance the visual appeal of the game.

##### Input Handling:

* + Keyboard Input**:** Pygame's event handling capabilities are utilized to capture and respond to player inputs for controlling the snake's movement.

##### Error Handling and Debugging Tools:

* + Python Debugger (pdb**):** Used to troubleshoot and debug the application during development, ensuring smooth gameplay.
  + Linting Tools**:** Tools like Pylint or Flake8 help maintain clean and readable code, making it easier to identify potential issues.

##### Testing and Deployment:

* + Manual Testing**:** Conducted to ensure that the game functions correctly, is enjoyable, and meets the intended gameplay experience.
  + PyInstaller**:** Can be used to package the game as an executable file for easy distribution and installation on various platforms.

### PROJECT DESIGN

The design of the Chess Game using Tkinter involves multiple layers that collectively ensure functionality, usability, and adherence to chess rules. Below is an overview of the project's key design aspects.

##### Architectural Overview

The architecture follows a modular approach, separating the logic, user interface, and event handling.

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

* + - Model: Handles the internal logic of the game, including board setup, piece movements, and rule enforcement.
    - View: The graphical representation of the chessboard and pieces using Tkinter widgets.
    - Controller: Manages interactions between the user and the model, handling events like piece selection and move execution.
  + Modular Design: Each component, such as the board, pieces, and rules, is encapsulated in separate classes or functions to ensure reusability and maintainability.

##### User Interface (UI) Design

The user interface is designed with simplicity and intuitiveness in mind, ensuring a smooth user experience.

##### Chessboard Layout:

* + - An 8x8 grid displayed using Tkinter's Canvas widget.
    - Alternating light and dark squares for clarity.

##### Piece Representation:

* + - Chess pieces (king, queen, rook, bishop, knight, pawn) are represented with images or Unicode symbols.

##### Interactive Elements:

* + - Highlighting the selected piece and its valid moves.
    - Feedback for invalid moves or illegal actions.

##### Menu Options:

* + - Start a new game.
    - Quit the application.
    - (Optional) Save and load game states.

##### Functional Design

The functional design ensures the game adheres to standard chess rules while maintaining flexibility for future enhancements.

##### Game Initialization:

* + - Pieces are placed in their standard starting positions.
    - The game begins with White’s turn.

##### Move Validation:

* + - Each piece’s movement is restricted according to chess rules.
    - Special moves like castling, en passant, and pawn promotion are implemented.

##### Game State Management:

* + - The game checks for conditions like check, checkmate, and stalemate after each move.

##### Player Interaction:

* + - Players click to select and move pieces.
    - Invalid moves trigger error messages or visual feedback.

##### Logical Design

* + Class Structure:
    - Piece: Base class for all chess pieces, with attributes like position and color.
    - Board: Handles the state of the chessboard and pieces.
    - Game: Manages game flow, including turn tracking and rule enforcement.

##### Algorithms:

* + - Pathfinding for piece movement (e.g., rook, bishop, queen).
    - King safety checks for determining check and checkmate.

##### Error Handling Design

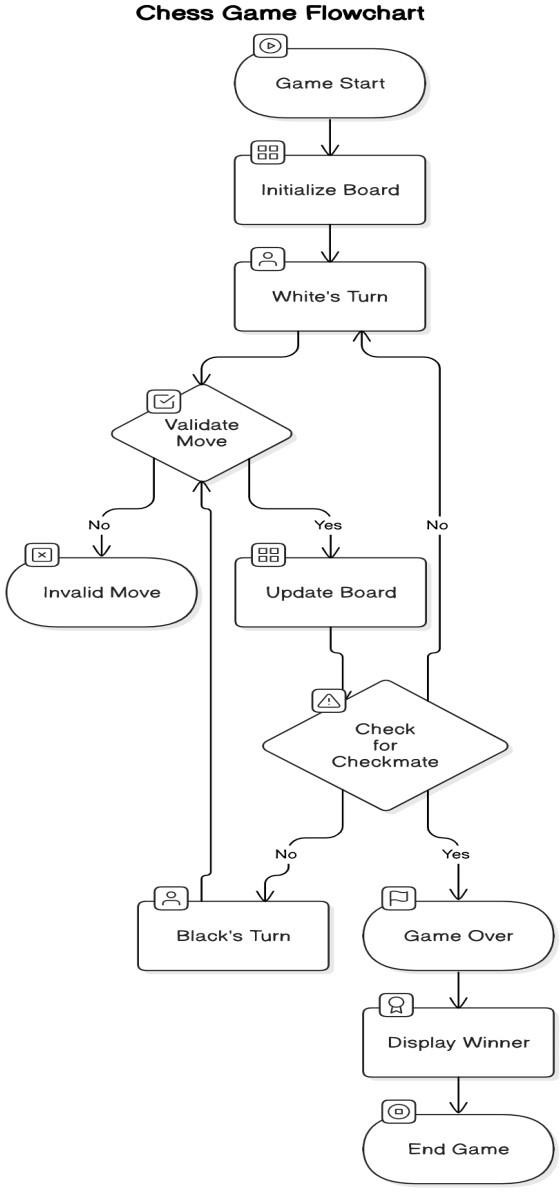
* + Prevents illegal moves and enforces chess rules through validation mechanisms.
  + Provides user feedback for invalid actions via error messages or visual cues.

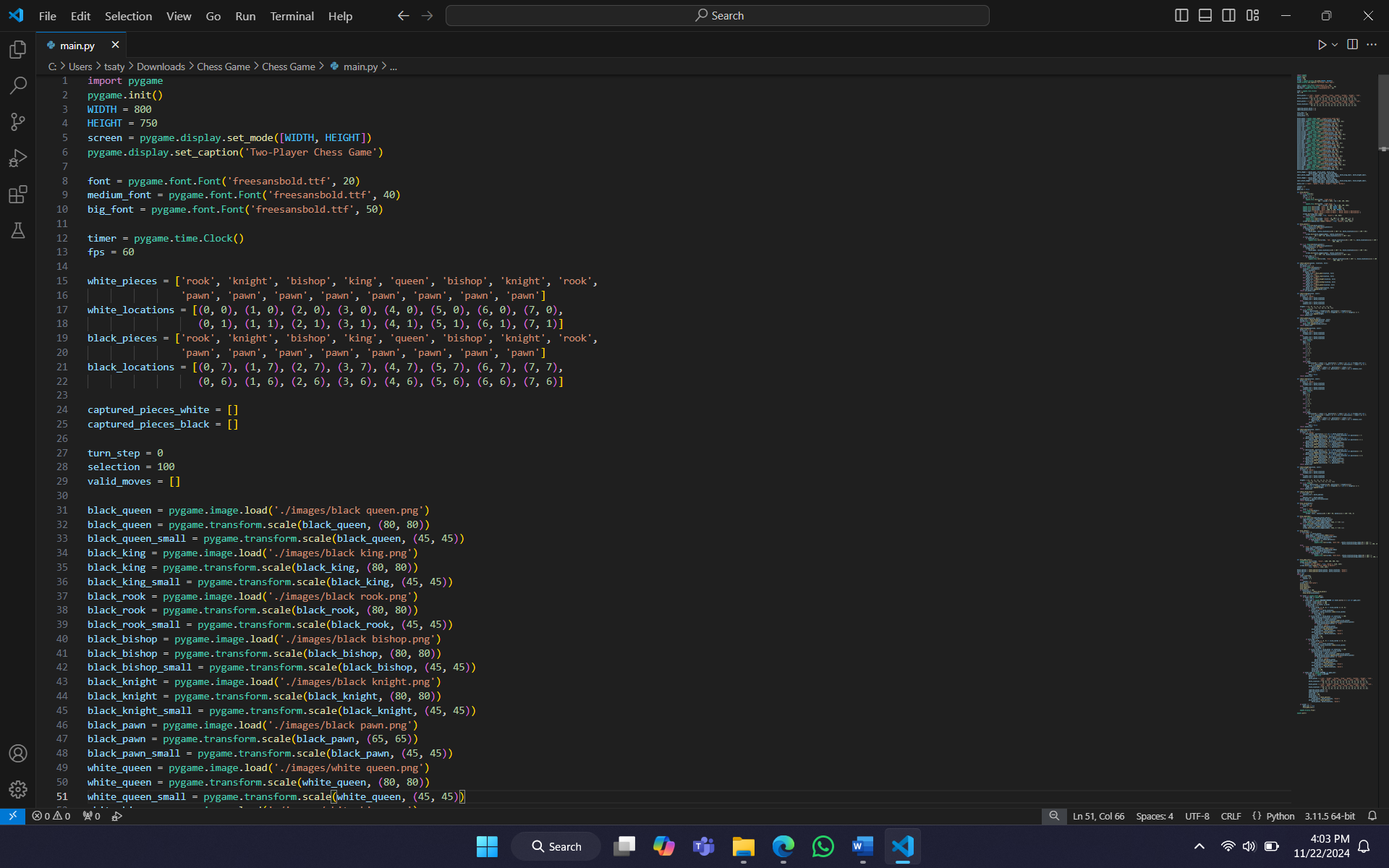
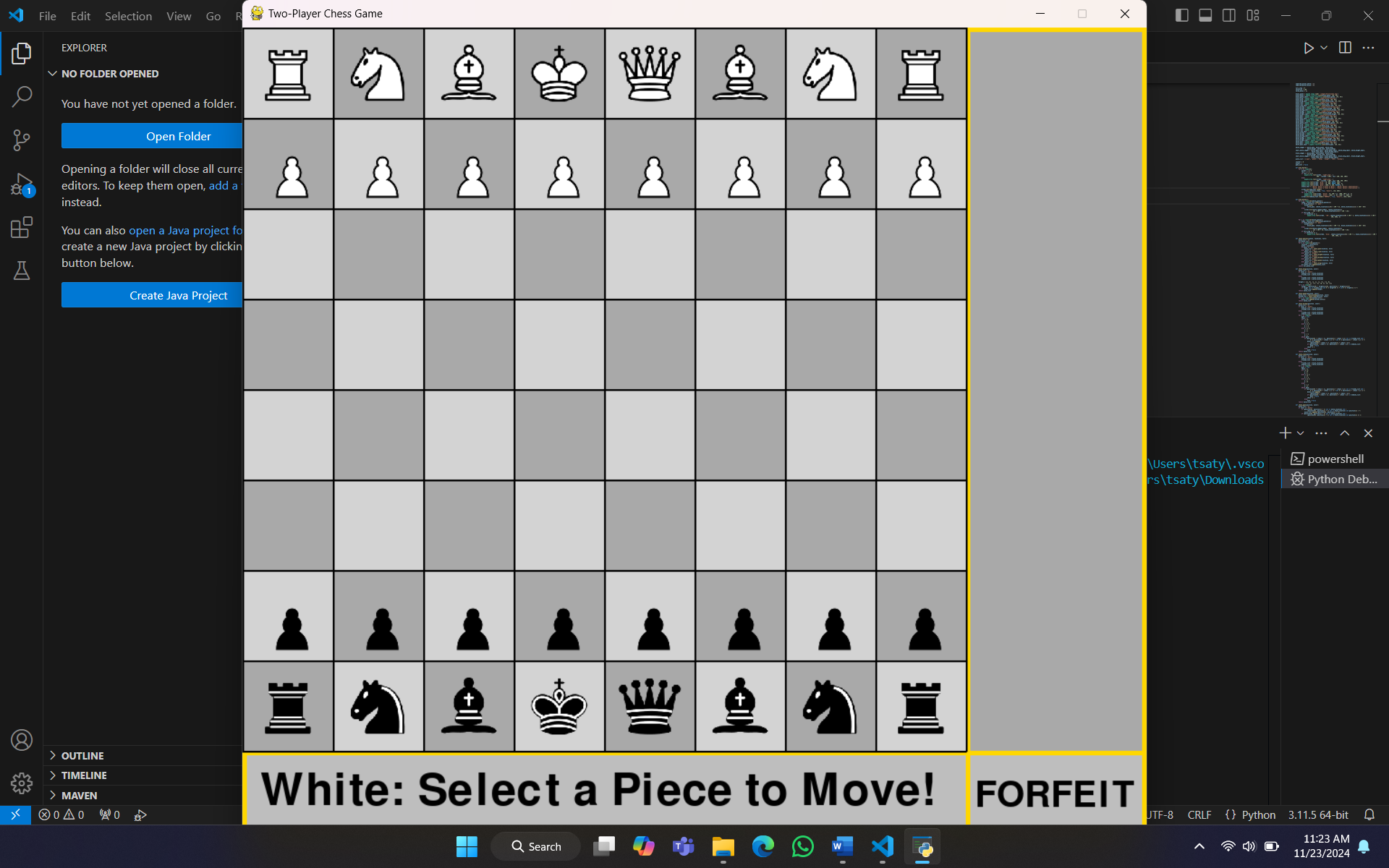
##### Scalability and Extensibility

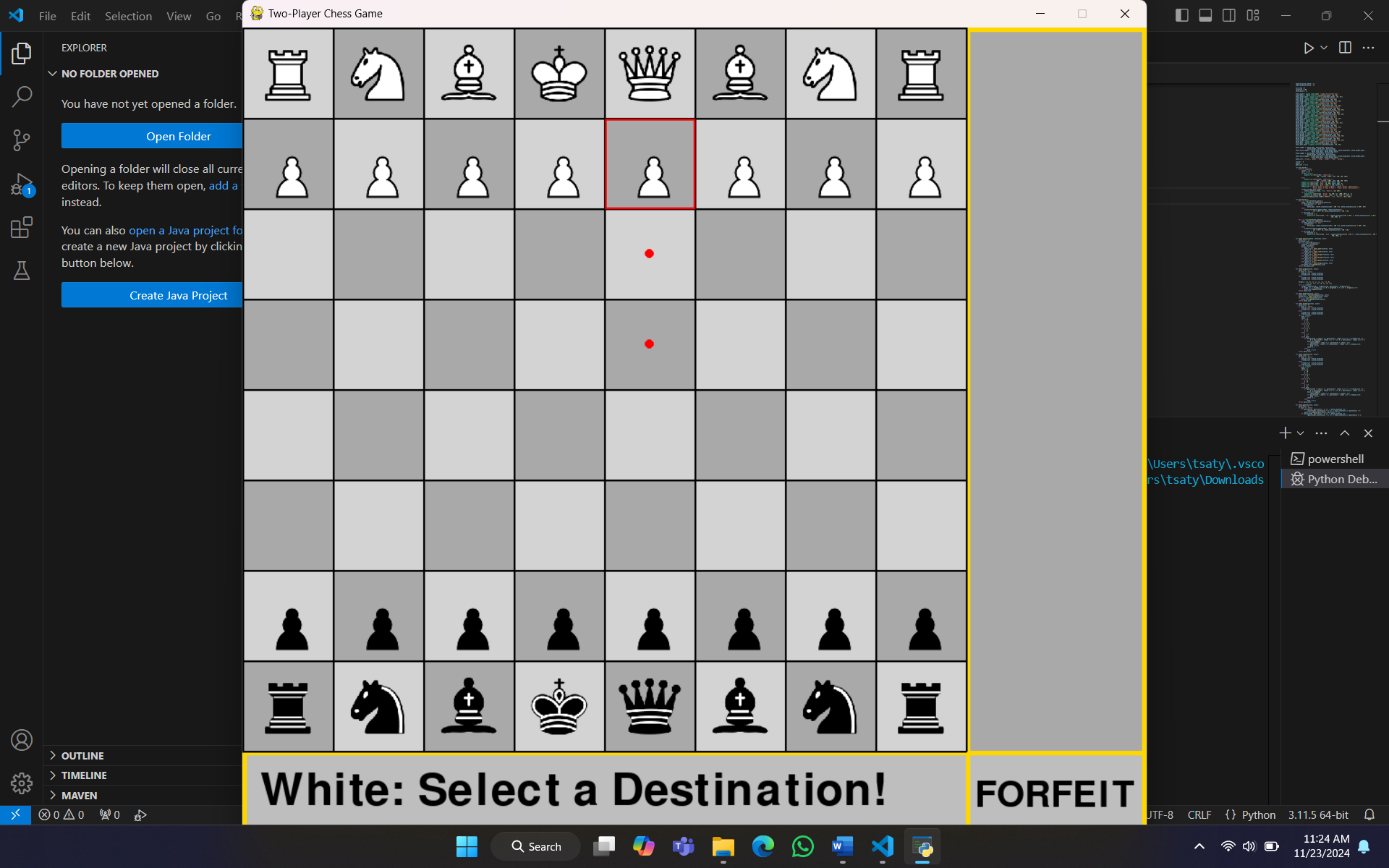
The project is designed to accommodate future features, such as:

* + AI Opponent: Using Minimax or other algorithms.
  + Online Multiplayer: Integrating network communication modules.
  + Game Analysis: Adding move history or replay functionality.

#### Flow Chart



1. **Project Implementation**



A screenshot of a computer

Description automatically generated

A screenshot of a computer game

Description automatically generated

A screenshot of a computer game

Description automatically generated

##### Testing and Implementation

##### Unit Testing

The goal of unit testing is to independently confirm that each component—functions and modules— is functioning correctly.

##### Items to be Tested

* 1. Piece Movement Validation

Verify that each piece follows its movement rules. For example, pawns move forward, rooks move in straight lines, bishops move diagonally, etc.

##### Check Detection

Confirm that the game correctly detects when a king is in check.

##### Turn Alternation

Ensure that the game alternates turns between the two players (White and Black).

##### Boundary Conditions

Verify that pieces cannot move outside the bounds of the board (e.g., a8, h1).

Piece Movement Test

def test\_piece\_movement():

board = ChessBoard() # Assuming ChessBoard is the main class pawn = board.get\_piece("e2")

assert pawn.is\_valid\_move("e4"), "Pawn did not move correctly" assert not pawn.is\_valid\_move("e5"), "Pawn moved too far"

bishop = board.get\_piece("c1")

assert bishop.is\_valid\_move("f4"), "Bishop did not move diagonally" assert not bishop.is\_valid\_move("c3"), "Bishop moved incorrectly"

Check Detection Test

def test\_check\_detection(): board = ChessBoard()

board.place\_piece("e1", "White King") board.place\_piece("e8", "Black Rook")

assert board.is\_check("White"), "Check was not detected" board.move\_piece("e8", "e7")

assert not board.is\_check("White"), "Check was incorrectly detected"

##### Integration Testing

The goal of integration testing is to confirm that various components function together as a unit.

##### Test Scenarios

1. Piece Movement and Check Detection

Verify that moving a piece places the opposing king in check when appropriate.

##### Checkmate Scenario

Confirm that the game ends when a player’s king is checkmated.

##### Game Reset

Ensure that resetting the game clears the board and resets scores and turn tracking.

##### Capture and Board Update

Verify that capturing a piece removes it from the board and updates the game state

Game Reset Test

def test\_game\_reset(): game = ChessGame()

game.move\_piece("e2", "e4") # Make a move game.reset\_game()

assert game.current\_turn == "White", "Turn did not reset to White" assert len(game.board.get\_pieces()) == 32, "Board did not reset"

#### Conclusion

The project aimed at developing a simple yet functional chess game using Python has proven to be a valuable exercise in applying key programming principles, including object-oriented

design, game logic implementation, and software testing. The game provides a solid foundation for simulating a chessboard, with the ability to manage different pieces and their movements according to the official rules of chess. The project incorporates essential gameplay elements such as alternating turns, valid move validation, and capturing pieces,

while offering a user-friendly interface for interactive gameplay.

Throughout the development process, various challenges were encountered and successfully

addressed, including managing the state of the game, implementing move logic for each piece, and handling edge cases such as check and checkmate scenarios. The use of classes and methods allowed for a modular approach to the design, making the code more maintainable and extensible. This modularity also facilitated the inclusion of additional features, such as a move history tracker, game reset functionality, and the integration of a scoring system, which can be expanded further in the future.One of the key aspects of the project was the implementation of unit and integration testing. Unit tests were designed to ensure that individual components, such as piece movement and check detection, functioned correctly. The integration tests confirmed that the components worked together as expected, ensuring smooth game operation. The testing phase highlighted the importance of debugging and refining the code, leading to a more stable and reliable chess game. Furthermore, the tests provided valuable insights into areas where improvements could be made, such as adding features for advanced chess rules and improving the AI for single-player mode. The project also provided a deeper understanding of the complexities involved in game development, particularly in simulating real-world rules and ensuring that all possible user actions are accounted for. It emphasized the importance of considering various edge cases, including boundary conditions, game resets, and the detection of special scenarios like checkmate and stalemate. In conclusion, this project successfully demonstrates the application of programming concepts in creating an interactive chess game. While the current version offers a basic playable experience, it serves as a strong base for further enhancements, such as adding an AI opponent, improving the graphical user interface, and implementing advanced chess strategies.The project highlights the potential for growth in both the codebase and the features, offering exciting opportunities for further development and refinement in future versions.

#### References

CodePal

offers a detailed tutorial on building a chess game using Python and Tkinter. It provides explanations for designing the chessboard, handling player interactions, and managing the graphical interface

GitHub Repository**:**

Chess-with-Python contains a fully commented chess game implementation with a graphical user interface. It demonstrates practical use of Tkinter and Python for building a complete chess game. You can access the code and explanations

FreeCodeCamp's Chess Game Tutorial

provides step-by-step guidance on creating a chess game using Python. It is beginner-friendly and includes code snippets and a focus on implementing core game logic and the user interface​

THANK YOU