Project Report: Chess Player Game

Introduction

The goal of this project was to create a basic chess game in Python using the Pygame library. The objective was to create a playable game using the MIN-MAX algorithm.

Project Overview

The project is a command-line based chess game that allows a player to play against an AI. The game includes the following features:

- Player movement and capture
- Check and checkmate detection
- Draw by insufficient material
- All chess rules except En passant, castling.
- Game reset and exit
- User defined depth in MIN-MAX
- Alpha Beta Pruning

Design and Implementation

Evaluation Function

There were numerous evaluation functions to choose from, with all having their own strengths and weaknesses.

Some of the evaluation functions which we considered adding were:

- **Material Evaluation**: This is the simplest form of evaluation, where each piece is assigned a static value.
- **Mobility**: Evaluates the number of legal moves available to each side. More mobility indicates a stronger position.

- **King Safety**: Evaluates how vulnerable each king is. This includes factors like pawn shelter, open files towards the king, and the presence of attacking pieces nearby.
- **Development**: Considers how well each side has developed their pieces. Rapid development can lead to a stronger position.

Considering the computational power and performance of the AI we decided to implement a slightly smarter version of Material Evaluation. The evaluation function which we implemented was Piece Activity.

Piece Activity: Evaluate the activity of each piece by considering their centralization and control of key squares. This is implemented by assigning higher values to pieces centralized on the board.

```
def evaluate(self):
    white_points = 0
    black_points = 0
    for i in range(8):
        for j in range(8):
            if isinstance(self[i][j], ChessPiece):
                piece = self[i][j]
                if piece.color == 'white':
                    white_points += piece.get_score() + self.calculate_piece_activity(i, j, 'white')
                else:
                    black_points += piece.get_score() + self.calculate_piece_activity(i, j, 'black')
    if self.game_mode == 0:
        return black_points - white_points
    return white_points - black_points

def calculate_piece_activity(self, row, col, color): #if the pieces are in central position give them more poiints
    activity_points = 0

    if (row, col) in [(3, 3), (3, 4), (4, 3), (4, 4)]:
        activity_points += 1  # Adjust as needed
    return activity_points
```

- Each piece was given a standard static value (pawn = 1, knight = 3, bishop = 3, rook = 5, queen = 9)
- The pieces occupying the central 4 squares were given an extra 1 point. Hence motivating central board control.

Representation Of The Game

The game state is represented by a 2D list. This ensured easy implementation of graphics+AI and optimal use of space.

```
def initialize_board(self):
    for i in range(8):
        self.board.append(['empty-block' for _ in range(8)])
```

User Defined Depth

The game implements a starting screen which requires the user to enter the depth used for MIN-MAX. The depth is then used in computation.

```
depth=get_depth()
draw_background(board) # game starts

if board.game_mode == 1 and board.ai:
    #print(11111)
    get_ai_move(board,depth)
    draw_background(board)
```

Artificial Intelligence

Min-Max with alpha beta pruning was implemented to reduce the computational power and space. Although pruning was used it is still computationally heavy.

These are the time taken for AI to response for every depth (using a 4.2 GHZ Processor)

DEPTH	NUMBER OF MOVES	TIME
1	20	Instant
2	400	Instant
3	8000	1-2 second
4	160000	3-5 seconds
5	3200000	1 minute++

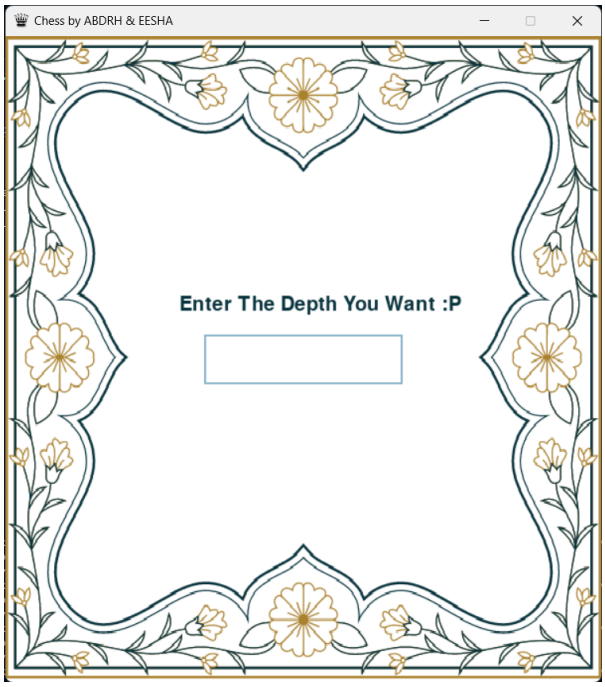
```python
def minimax(board, depth, alpha, beta, max_player, save_move, data): #min max with alpha beta

    if depth == 0 or board.is_terminal():
        data[1] = board.evaluate()
        return data

    if max_player:
        max_eval = -math.inf
        for i in range(8):
            for j in range(8):
                if isinstance(board[i][j], ChessPiece) and board[i][j].color != board.get_player_color():
                    piece = board[i][j]
                    moves = piece.filter_moves(piece.get_moves(board), board)
                    for move in moves:
                        board.make_move(piece, move[0], move[1], keep_history=True)
                        evaluation = minimax(board, depth - 1, alpha, beta, False, False, data)[1]
                        if save_move:
                            if evaluation >= max_eval:
                                if evaluation > data[1]:
                                    data.clear()
                                    data[1] = evaluation
                                    data[0] = [piece, move, evaluation]
                                elif evaluation == data[1]:
                                    data[0].append([piece, move, evaluation])
                        board.unmake_move(piece)
                        max_eval = max(max_eval, evaluation)
                        alpha = max(alpha, evaluation)
                        if beta <= alpha:
                            break
        return data
```

```python
    else:
        min_eval = math.inf
        for i in range(8):
            for j in range(8):
                if isinstance(board[i][j], ChessPiece) and board[i][j].color == board.get_player_color():
                    piece = board[i][j]
                    moves = piece.get_moves(board)
                    for move in moves:
                        board.make_move(piece, move[0], move[1], keep_history=True)
                        evaluation = minimax(board, depth - 1, alpha, beta, True, False, data)[1]
                        board.unmake_move(piece)
                        min_eval = min(min_eval, evaluation)
                        beta = min(beta, evaluation)
                        if beta <= alpha:
                            break
        return data
```

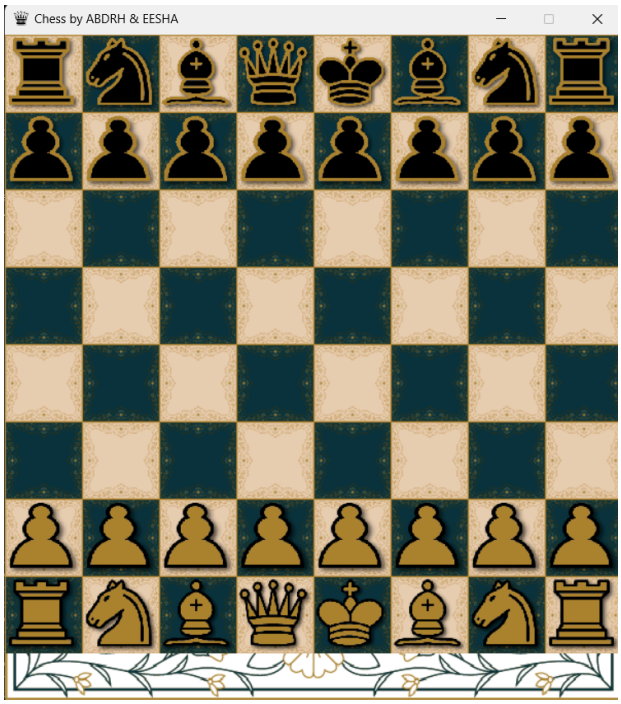
Graphical User Interface

The assets are statically loaded and the library used is pygames.

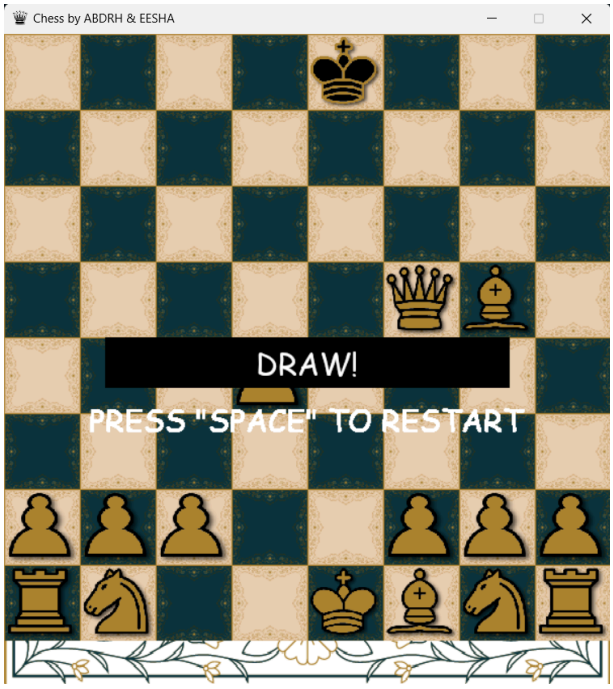
Starting Screen



Board



Draw/Win



Conclusion

In conclusion, the project was a success and achieved its objectives. Future developments could include adding more features, such as

- Enpassant and castling
- A better more robust AI
- Multiplayer feature
- AI ratings and Player Ratings

References

Pygame. (n.d.). Pygame: Python Game Development.

Skeleton Chess Code