CHESS AI

PROJECT REPORT

PROJECT X MENTORSHIP PROGRAM

AT

COMMUNITY OF CODERS (COC), VEERMATA JIJABAI

TECHNOLOGICAL INSTITUTE, MUMBAI.

AUGUST 2023

ACKNOWLEDGEMENT

We are extremely grateful to our mentor Siddheshsingh Tanwar for his support and guidance throughout the duration of the project.

We would also like to thank all the members of COC VJTI for their timely support as well as for organizing Project X and giving us a chance to work on this project.

Rohan Parab

rohaneducation2004@gmail.com

Aditya Yedurkar

aditya.yedurkar@gmail.com

OVERVIEW

The project deals with the development of a chess game that involves design of interactive chess GUI for gameplay along with implementation of various rules & standards taken into consideration while playing it.

The fundamental concept used in this gameplay enables the player to play against a highly intelligent AI opponent, by incorporating various algorithms and parameters that simplify the evaluation of current gamestate.

1. PYGAME

Pygame is a popular Python library used for creating 2D games and multimedia applications. It provides a range of features and functions for working with graphics, sound, input devices, and more. It uses Simple Direct Media Layer(SDL) library which allows real- time computer game development using high level programming language Python (as well as C, Assembly, Cython). SDL includes vector math, collision detection, 2D sprite scene graph management, MIDI support, camera, pixel array manipulation, transformation, filtering, drawing. Provides a low level access to audio, keyboard, mouse, joystick, graphics hardware via OpenGL and Direct3D (programming interface for 2D and 3D vector graphics). Pygame is designed to work on multiple platforms, including Windows, macOS, and Linux.

Overview of key components of Pygame library :

1) Surface : In Pygame, everything is drawn on a Surface. A Surface appears to be a window, an image, or any other drawable area. We can create and manipulate these surfaces to display graphics on it.
2) Drawing : Pygame provides functions for rendering basic shapes (rectangles, circles, lines), text, and images on a Surface. We can also load and display images, making it suitable for sprite-based games.
3) Colors : We can specify colors using RGB values. Pygame provides functions to work with colors and manage the color of various elements on the screen.
4) Events : Pygame handles events like keyboard and mouse input. We can check for events and respond to user's actions in the game or application.
5) Sprites : Pygame has a Sprite class that simplifies working with 2D game objects. Sprites can be organized into groups for efficient updates and rendering.
6) Sound & Music : We can load and play sound effects and music in our games using Pygame. It supports various audio formats.

7) Rectangles : Pygame provides the Rect class for managing rectangular areas. Its commonly used for collision detection and managing object positions.
8) Clock : We can control the frame rate of your game by using the Clock class. This helps in maintaining a consistent game speed across different platforms.
9) Font : Pygame allows us to render text on the screen using different fonts. Thereby, we can load fonts, set the size, and display text with ease.
10) Input Handling : Pygame enables keyboard and mouse input handling, making it suitable to create interactive applications and games.
11) Collision Detection : Pygame provides tools for collision detection between objects, which is essential in most games.

To get started with Pygame, we need to install the Pygame library, create a Pygame window, and then start building our game or application by handling input, drawing graphics, and managing game logic.

2. GAME LOGIC

2.1 BOARD REPRESENTATION :

The chess board of alternate black and white squares is designed by traversing through nested for loop such that if sum of row & column for square is divisible by 2 its a white square else a black one. This is implemented using function drawboard. Here row & column number ranges from 0-7. So for e4 square its row 4 and column 4.

Now, moving at piece representation which is implemented with help of board nested list which stores in rows and the elements encountered at a column in that row. The convention used for pieces is as follows :
p -> pawn , B -> bishop , N -> knight , R -> rook , Q -> queen , K -> king ,
w -> white and b -> black , -- -> no piece
So, if we want to represent black bishop at d6, we express it as :
self.board[2][3] = 'bB' where 2 & 3 are row and column numbers respectively.

For drawing the pieces on board we are blitting the images of respective pieces on that particular square by accessing its row and column, for that we have used drawpieces.

2.2 MOVELOG PANEL :

There is a move panel attached adjacent to the board which records the move made by both white and black. It also incorporates special case move notations like castling, captures, etc with the help of drawMoveLog and getChessNotation.

2.3 MOVE GENERATION :

In this gameplay, introducing validity to moves is a crucial task. For that we first create a list of moves in getAllPossibleMoves and by traversing through the board for individual pieces we fetch their possible moves using functions like ; getPawnMoves, getKnightMoves, getBishopMoves, getRookMoves, getQueenMoves, getKingMoves. Append the moves returned by these functions to the list of moves. These moves are then filtered out based on their validity through the getvalidmoves which practices the Naive algorithm.

Naive algorithm implementation :

1) Firstly generating all possible moves.

2) For each move in the list of moves, make that move by makeMove method and the move gets appended to the list of valid moves.

3) Then generate all the possible opponent's moves for that particular move.

4) If any of that opponents move attack our king directly ie tries to capture it which is impractical in chess, we will remove our move from list of valid moves. This can be verified using inCheck and squareUnderAttack methods.

5) Undo that move which was made previously and reset to the original gamestate.

Finally when the user performs any move, the program searches that move into the list of valid moves and checks for the validity of that move before performing it onto the board.

2.4 UNDO MOVE :

This feature is engaged within game through undoMove method wherein the moves made on the board are recorded into the list called moveLog. So if we want to undo the preceding moves made we can remove them from moveLog list and the other parameters are reclaimed accordingly.

2.5 SPECIAL MOVES :

2.5.1 CASTLING :

The Castling feature can be implemented with the help of class CastleRights which includes flags such as wks,bks,wqs,bqs for whitekingside, blackkingside, whitequeenside, blackqueenside castling moves. The currentCastlingRights at the start of the game holds the values for all the above mentioned flags as True. And as some castling move is figured out through getCastleMoves, getKingsideCastleMoves, getQueensideCastleMoves corresponding flags are switched to value False through updateCastleRights method and currentCastlingRights are modified. Also if a castled move is to be undone, then to retain the castling rights present at that move up to which it is undone we maintain a record of these rights via castleRightsLog.

2.5.2 PAWN PROMOTION :

The makePawnPromotion is used for this purpose wherein it generates a menu as a new game window to the user so as to choose the piece to which the pawn should get promoted. So, here the game handles the user's input using the KEYDOWN event, pressing an appropriate key for that particular piece.

2.5.3 EN PASSANT CAPTURE :

This feature is incorporated using enpassantPossible tuple which stores in the coordinates of the squares where an en passant capture is possible. Further we keep a track of these moves within enpassantPossibleLog list, so that while undoing en passant move we can check it with a flag isEnpassantMove.

2.6 HIGHLIGHTING SQUARES :

Using the HighlightSquares method the previously move made gets highlighted, not only this but the square where piece selected by the user of its own exists also gets marked. Along with that all the valid moves related to that piece gets marked down by the above mentioned method.

2.7 ANIMATING MOVES :

So, when a move is made on the board, with the help of animateMove function move gets some animation(motion) so that it appears to the user that the piece is travelling from its starting square to the ending square.

3. CHESS AI :

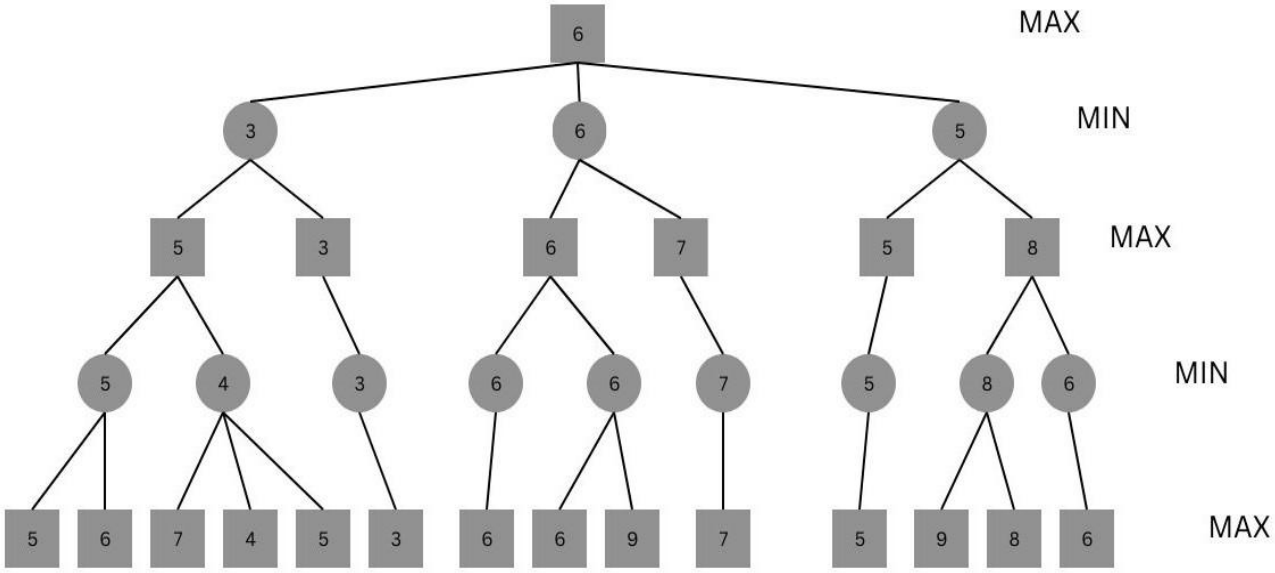
3.1 MINIMAX ALGORITHM :

The Minimax algorithm is a commonly used strategy for decision-making in two-player games like chess. It is a recursive algorithm that helps a player determine the best move to make by considering all possible future moves and their outcomes. In chess, the two players are typically referred to as "Max" (the player trying to maximize their score) and "Min" (the opponent, who is trying to minimize Max's score).

Here there exists an evaluation function that assigns a numerical value to the current game state. So the goal of the player is to maximize their evaluation function but in this case the opponent also tries to maximize their evaluation function. Certainly this results the opponent into minimizing player’s evaluation function.

The Minimax algorithm constructs a game tree that represents all possible moves and their outcomes. It begins by considering the current gamestate and generates all legal moves for the player whose turn it is (Max). For each of these moves, it simulates the opponent's responses (Min) by generating all possible replies. This process continues until a specified depth limit or until a terminal game state is reached (checkmate, stalemate, or a draw). Therefore, for each node in the tree representing Max's turn, it selects the move that maximizes the evaluation function, and for each node representing Min's turn, it selects the move that minimizes the evaluation function.

Lets understand through a diagrammatic representation. Here squares represent Max nodes whereas circles represent Min nodes.



This algorithm is implemented in the game using findMoveMinMax.

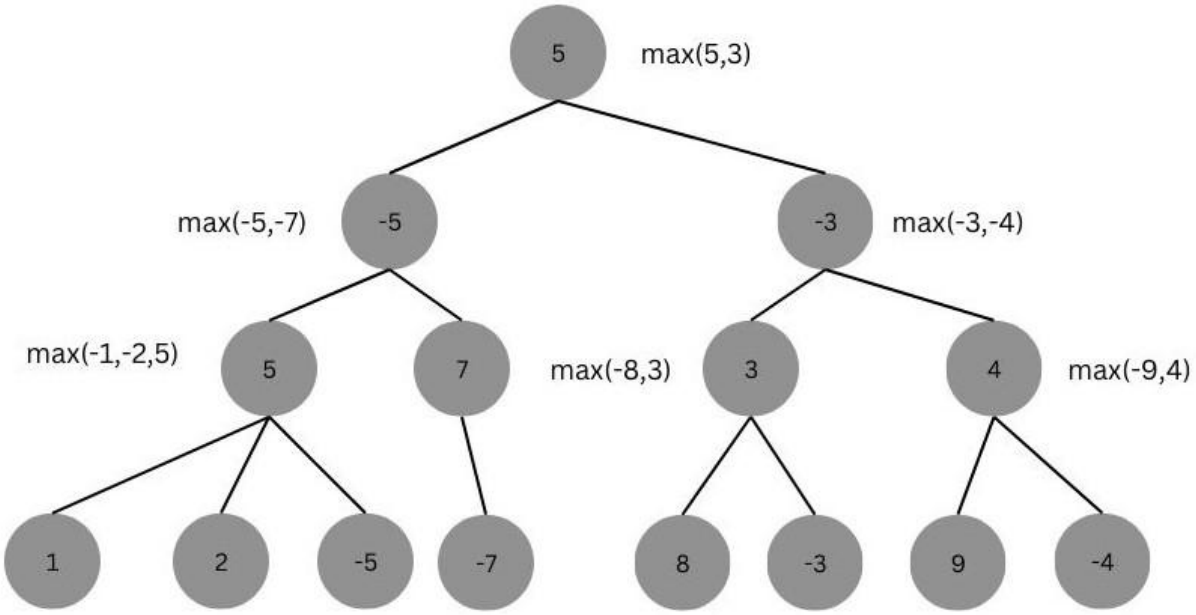
3.2 NEGAMAX ALGORITHM :

The Negamax algorithm is a variation of the Minimax algorithm that simplifies the implementation and reduces code complexity.This is a form of MinMax algorithm which depends on the zero sum property of 2 player games, ie, min(a,b) = - max(-b,-a). The key idea behind Negamax is that it combines the roles of the maximizing player (Max) and the minimizing player (Min) into a single player, resulting in more concise and efficient code.

Just as in Minimax, Negamax builds a game tree representing all possible moves and their outcomes and evaluates each with an evaluation function. It starts with the current position and generates all legal moves for the player whose turn it is (Max/Min). For each move, it recursively explores the tree to evaluate the positions at different depths, either until a specified depth limit is reached or until a terminal game state (checkmate, stalemate, or a draw) is encountered.

In Negamax, the key idea is to maximize the negative value of the opponent's best move. Instead of explicitly considering the opponent's move as minimizing, it takes the negation of their best move's evaluation.When a terminal game state is reached, the evaluation function is used to determine the score for that position. A high positive score indicates a good position for the current player, and a high negative score indicates a good position for the opponent. The key point is that the algorithm reverses the sign of the evaluation for the opponent, so it effectively handles both maximizing and minimizing in a unified manner.

This algorithm is implemented in the gameplay using findMoveNegaMax. Lets understand through a diagrammatic representation.



3.3 ALPHA-BETA PRUNING :

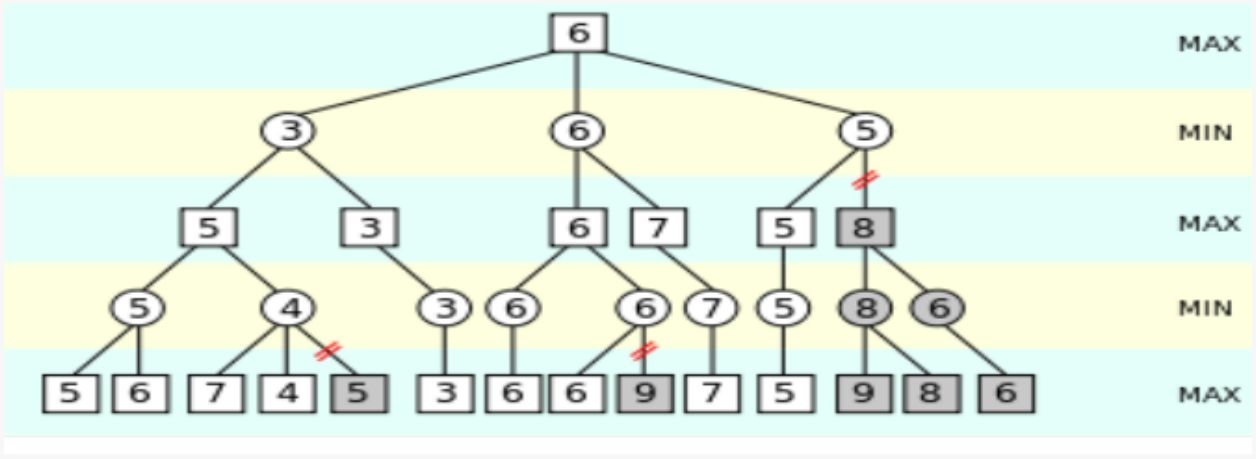
Alpha-beta pruning is a search algorithm used in chess games, to improve the efficiency of evaluating potential moves while searching for the best move. It is a more efficient version of the Minimax/Negamax algorithm and is designed to reduce the number of positions that need to be evaluated during a game tree search.

In the alpha-beta pruning algorithm, two values, alpha and beta, are used to keep track of the best values found so far for the maximizing and minimizing players, respectively. Alpha (α) represents the best value found for the maximizing player (initially set to negative infinity). Beta (β) represents the best value found for the minimizing player (initially set to positive infinity) that is the convention.

The algorithm recursively explores the game tree, starting from the current position and considering various moves. During the search, when evaluating a move, the algorithm compares the current evaluation of the move to the alpha and beta values. If the current move is better for the maximizing player (greater than alpha), the alpha value

is updated. If it's better for the minimizing player (less than beta), the beta value is updated. If at any point the beta value becomes less than or equal to the alpha value, it means that the opponent has a better move elsewhere in the tree, and thereby a cutoff condition is achieved. In that case, there's no need to explore the rest of the moves from the current position, and the search is pruned, saving computational effort.

This algorithm is incorporated within game using findMoveNegaMaxAlphaBeta. Let's understand this through an example. In the below example, the gray blocks of the search tree are not traversed as its pruned, resulting in an efficient code.



3.4 EVALUATION FUNCTION & PARAMETERS :

The evaluation function here is computed by taking into consideration the material value existing on the board, the positional weights of various pieces on the board, and also some additional parameters are also accepted by referring to a paper mentioned in the references section. This is implemented in the gameplay using the method ScoreBoard.

Almost out of the 33 parameters referred in the paper nearly 20 parameters are incorporated through the matrices of positional scores for pieces at all specific positions onto the board. For that we have referred an article on

PeSTO’s (Piece - Squares Table Only) evaluation function by Ronald Friederich as tried in his chess engine RofChade. It performs a tapered evaluation to interpolate by current game stage between piece-square tables values for opening and endgame, optimized by Texel's tuning method. The remaining parameters are introduced using their respective methods for calculation of score returned by the final evaluation function. Those are King castled, King Pawn shield, Knight Support, Double Pawns, Queen Mobility, King Mobility, Freedom of Moves, Pieces surrounding neighbouring Squares of King.

So, the evaluation function computes the score in the following manner : so it's the summation of pieceScore, piece_position_score, value of all other parameters. The value is computed using the formula : Σ Weight[n] $*$ F[n] = Val. Here the weights to these parameters are decided manually by optimizing to the best values that can be achieved each time by playing the game. Also the material values are decided as per the PeSTO’s evaluation function ie as follows : King -> 0, Queen -> 1200, Rook -> 500, Bishop -> 400, Knight -> 500, Pawn -> 82. These weights are a bit calibrated by us as well. It computes the final score for white by adding it with previous scores and for black by subtracting from previous scores that is convention followed.

FUTURE PROSPECTS :

1) We have decided to work on the Face Recognition feature so that the feature of authentication is getting implemented with the help of a database. Currently, we have introduced this feature but it only works for both of us.
2) We have planned to incorporate a Genetic algorithm (as we have already worked on a few of its mini projects) into it so as to get the AI's moves more optimized and computation becomes faster at higher depths.
3) Also with the help of a database we are going to store the results of the previous games played by that user.

REFERENCES :

1) Pygame Documentation :
   https://www.pygame.org
2) Pygame Playlist & Miniprojects :
   https://www.youtube.com/playlist?list=PLu0W_9lII9ailUQcxEPZrWgDoL36BtPYb
3) Chess Engine Playlist :
   https://www.youtube.com/playlist?list=PLBwF487qi8MGU81nDGaeNE1EnNEPYWKY_
4) Artificial Intelligence Playlist :
   https://www.youtube.com/playlist?list=PLp6ek2hDcoNB_YJCruBFjhF79f5ZHyBuz
5) Evaluation Function Parameters Paper :
   https://www.cmpe.boun.edu.tr/content/tuning-chess-evaluation-function-using-genetic-algorithm

   https://www.cmpe.boun.edu.tr/~gungort/undergraduateprojects/Tuning%20of%20Chess%20Evaluation%20Function%20by%20Using%20Genetic%20Algorithms.pdf

6) Article on PeSTO's evaluation function :

   https://www.chessprogramming.org/PeSTO%27s_Evaluation_Function

https://ccrl.chessdom.com/ccrl/4040/

7) Genetic Algorithm Playlist & Miniprojects :

https://www.youtube.com/playlist?list=PLRqwX-V7Uu6bJM3VgzjNV5YxVxUwzALHV