CS6613
Joby Joy
jj2196@nyu.edu

Mini- Checkers

Mini-Checkers is developed as a game that is to be played by a human against a computer. The game is played on a game board size of 6 x 6 squares made up of alternating light and dark squares.

Description:

- Each player starts out with six playing pieces, with the white pieces for the human and the black pieces for the computer. Note that all the pieces are placed on dark squares.
- At the start of the game, the human player can choose to move first or second.
- Each player takes turn to make a move. There are two types of moves: regular moves and capture moves.
- In a regular move, a piece can move forward diagonally to an adjacent square that is empty.
- In a capture move, a piece can jump over and capture an opponent's piece and land on an empty square (landing on a square that is not empty is not allowed.) The jump must be in the forward diagonal direction and no consecutive jumps are allowed. In addition, every opportunity to jump must be taken. In the case where there are two or more possible jump moves, the player can choose which one to take.
- No vertical, horizontal or backward moves are allowed for both regular and capture moves.
- If a player has no legal move to take, his/her turn will be forfeited, and the other player will make the next move.
- A player wins when he/she captures all of the other player's pieces. If both players do not have any legal move to take, the game will end and the player with the most number of pieces left wins; if the two players have the same number of pieces left, the game is a draw.

Implementation:

Design:

- Mini checkers is developed using python.
- Mini checkers defaults to the black piece being played by the Computer (ai) and the white piece played by human.
- The Player are initialized using the Player class and the Piece class instantiates the piece on the board.
- The board is created as a 6x6 matrix with -1 indicating human, 1 for ai and 0 for no piece.
- Pygame is the library used to build the GUI.
- Loggers are used to print debug and info level logs in the game mode.
- The game listens to keyboard interrupts for the user. It allows the user to play first or second.

- Once the game starts, board is initialized, and Pieces are created.
- When user selects the piece, mouse position at which the selection is recorded and the mouse_click function saves the position of the first select.
- For the position where the piece has to be placed, possible_moves function calculates all the available moves and capture moves using the is_regular_move and is_capture_move function and allows the user to make the move.
- Once the user moves, the turn is switched to ai and ai_plays() method is called.
- ai_plays() calls minimax() which is built using the alpha beta pruning algorithm with iterative deepening DFS. The ply_depth for iterative deepening is initialized at the start of the game.
- Every time the depth reaches the max permissible depth, the eval_heuristic() is called and the score or the terminal state value is calculated following which backtracking happens.
- The best move is selected based on the heuristic score at the ply_depth at 0.
- The statistics of the game i.e the node values are printed to the logfile and turn is switched to be played by the human.
- The process repeats continuously until the final game condition win or draw state is reached.

Game Compilation and Execution:

- To run Mini_ Checkers, run the command in the terminal
  Python "**foldername**"/6_board_mini_checkers.py
  Please install pygame before executing the game.

Evaluation Function:

We use two evaluation functions in the alpha beta algorithm as part of the game design.

- eval_heauristic() → assigns a score of 100 points to each piece that exists in the board and if the turn is white deducts the total scores from all the white pieces from black pieces and vice versa for black's turn.
- Eval_heuristic_pos(): assigns a score of 200 to all the pieces that are on the opposite side of the board i.e. if the white piece is more closer towards the opponents side, the likelihood of it being eliminated is less and hence a higher weight is assigned to the corresponding piece.
- The score from both the evaluation functions are added and pruning conditions are assessed accordingly.
- Utility value for the terminal states of win, lose and draw are calculated by the total remaining pieces on the board for each white and black piece's, the higher the number of which decides the terminal state of the game.
  - *If >(white,black) human wins, ai loses.*
  - *If <(white,black) human loses, ai wins.*
  - *If =(white,black) human draws, ai draws.*

CS6613
Joby Joy
jj2196@nyu.edu

Game Mode:

Mini checkers enables the user to play in three game modes configured as buttons with different colors on the GUI.

- Easy(Yellow) mode: (default)
  In the easy mode, the ply depth as part of the iterative deepening strategy is set to a random value from 1 to 4.

- Medium (Green) mode:
  In the easy mode, the ply depth as part of the iterative deepening strategy is set to a random value from 5 to 8.

- Hard (Red) mode:
  In the easy mode, the ply depth as part of the iterative deepening strategy is set to a random value from 9 to 15.

Steps to Play:

- Start the game.
- Select the game mode in which you would like to play (By default the human plays in easy mode)
- Press C on the keyboard if you would like the computer to play first.
- Press the button at any stage of the game resets the game to corresponding game mode.
- Start playing. Have fun….

References:

- S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 3rd edition, 2010.
- https://pythonprogramming.net/pygame-python-3-part-1-intro/
- https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/

Credits:

- Professor Edward Wong, ewong@nyu.edu, Associate Professor, Computer Science and Engineering, NYU School of Engineering.