Xueyang (Sean) Wang
XW1154
12/14/2017

CS4613 AI Project: Mini Camelot

The project is coded in Python with an imported library TIME to ensure time restriction and prevent the search from going too deep. Python has often been used for data entry and organization. By using Python, it provides quick data manipulation and fast calculation for location and coordinates with its build-in structure. C++ was taken into consideration at the beginning; however, it was taken out of the formula in the end, because it does not have much libraries that support user input. Although user interface was not implemented, the programming structure was aimed to use HTML as its interface and user input from mouse in the future

Instruction to Compile the Program (update user interface)

1. Open AI_Game.py using console or be run on Python version 3.xx
2. Player will be prompted to play the game using keyboard
3. Player will decide to make move first or second, level of difficulty
4. To move a piece:
    a. Coordinate are in the format of Capital Letter + Number, for example, “A1”
    b. Cantering Move/Capturing Move/Plain Move
    c. Three lines input:
        i. coordinate of the piece to move (Double Left)
        ii. coordinate to move the piece to (Double Right)
        iii. coordinate of the piece jumped over (Double Middle)
    d. Plain Move has a plain first box “A1”
5. Successive (*Extra Credit, Part A*)
    a. If a cantering or a capturing move is made, player will be able to make successive moves
    b. Player can choose to continue making successive moves or not even when there are available moves that they can make
6. AI information printed after making a move based on the given guideline to track progress
7. Winning Condition checked

Description of Design and Program

Class:

Board

- The initialization included four arrays to construct the basic gameboard o Empty space array to ensure empty spaces around the boarder
    - Dashing space array to ensure empty spaces to indicate available space
    - White pieces array to locate the initial location
    - Black pieces array to locate the initial coordinate
- Printing gameboard by running for-loops on the initialized four arrays and the board coordinates on the rows and columns with set structures by using '\t'

Method:

move (board, player = None)

- This function is intended to calculate possible moves that both players and AI can make based on current board.
- By passing in board information and user information, the function calculates the possible moves by decoding coordinate into an integer. Adding or subtracting movements from the integer will determine the validity if it is within range of the gameboard. If a capturing move is available, the possible move is limited to capturing only based on given prompt

player_move (board)

- This function prompts player input to get what piece player wants to move and where player wants to move the piece to.
- By getting possible moves from move (board, player = None), it compares player input with possible moves and executes if valid.
- If a valid input is given from player and a capturing move is available, the move must be the capturing move, otherwise invalidate and asked again.
- If player has given a capturing or a cantering move, player will be asked to continue or not for a successive move
- If an invalid input is given from player, player will be asked to input a valid one

ai_move (board, given_move)

- By getting possible moves from move (board, player = None), it compares the move calculated from ab_search () with possible moves and executes if valid.

ab-search (board, depth)

- Using global variable to calculate max Depth, Nodes generated, max Nodes pruned, min nodes pruned, time, and utility value
- By using a for-loop, a new board is constructed based on possible moves.
- Each board is then evaluated with pruning to find utility values (alpha and beta = maxV () and minV ())
- Moves with high utility values are added to a dictionary {move: value}
- Making sure that the search is terminated within 10 seconds and the best move is returned to the ai_move (board, given_move) and executed

maxV (board, alpha, beta, depth)

- Using global variables to update values: alpha, beta, max Depth, Nodes generated, max Nodes pruned, min Nodes pruned, time, and utility value
- A recursion with increased depth on the new_board minV (new_board, alpha, beta, depth+1)
  o Search tree and pruning
- If the new value is bigger than beta, than the node is pruned, and the value is returned
- Alpha value is updated by comparing current with previous

minV (board, alpha, beta, depth)

- Using global variables to update values: alpha, beta, max Depth, Nodes generated, max Nodes pruned, min Nodes pruned, time, and utility value
- A recursion with increased depth on the new_board maxV (new_board, alpha, beta, depth+1)
  o Search tree and pruning
- If the new value is less than alpha, than the node is pruned, and the value is returned
- Beta value is updated by comparing current with previous

coord_decode (coord)

- This function decodes coordinate (string) on the board into an integer; for example, 'A1' = 0

coord_encode (index)

- This function encodes an integer into a coordinate on the board; for example, 0 = 'AI'

terminate (board, depth)

- This function checks for conditions that will terminate the search
- If time exceeds 10 seconds, terminate
- If depth exceeds depth limit, terminate
- If a winning condition is present, terminate

win (board)

- This function checks if a winning condition based on the current board
- If there one piece left for white and black draw
- If there is no piece left for white and more than two pieces left for black black wins
- If there is no piece left for black and more than two pieces left for white white wins
- If both coordinates of white castle is occupied by black pieces black wins
- If both coordinates of black castle is occupied by white pieces white wins

result ()

- This function prints max Depth, Nodes generated, max Nodes pruned, and min Node pruned

u_eval (board) (*heuristics update, Extra Credit, Part B-3*)

- The evaluation function calculates whether white or black is more likely to win based on the minimal steps required to win and the distance to the castle
- By running the coordinate of the white pieces, it finds the smaller distance among black pieces and the black castle

- By running the coordinate of the black pieces, it finds the smaller distance among white pieces and the white castle

This evaluation function provides a base line for the game to recognize its current state. The return value will be within a range of [-value, value]. Like alpha and beta, it showcases how likely one is going to win over the other one. 0 represents that there is an equal chance of winning for both players.

Player Interface (*Extra Credit, Part B-1)*

- HumanTurn1/HumanTurn2/HumanTurn3 Are used to prompt user input
- suc_move(board) determines if player is making a successive move or not
- getOrder() get if player wants to go first or second
- getLevel() get at what level of difficulty player wants to play at.

Setting up the scene

- Initialization of global variable
- Prompt user input for: o Move first or second
    - Level of difficult (*Extra Credit, Part B-2*)
    - Default selection is invalid input is entered
    - While loop for game to run and refresh with new information
    - Ending game when a winning condition is present
- To have correct level of difficult two concepts are taken into considerations
    - Maximum and minimum utility value Evaluation: this restricts how many nodes are pruned and limits the process of searching since values are restricted. For implementation, lower level will have a small utility value range whereas higher level will have a high utility value range so that more information/steps will be taken into consideration
    - DSL for abusers
        - Evaluation: this restricts how deep the Alpha Beta Search algorithm will go into. By limiting the depth of the tree
            - It returns the favorable value of the current limited level instead of traversing through the entire tree
            - It saves time
            - The deeper it goes, the better move it will return; Therefore, the less value it foresees, the easer the AI will be