HEX

the documentation

by Pattaradanai Lakkananithiphan 6438169421

Requirement:

- Python and Jupyter Notebook
- or Google Colab

Implementation:

- The game board of hex is implemented using a 2D nested list in python
- The marks that represent the players are X and O (“ ” is used to represent empty)
- X will try to win by creating a vertical line through the board
- O will try to win by creating a horizontal line through the board
- BOARD_SIZE is the dimension of the board (its width or length)
- DEPTH is the maximum depth that will be used in the minimax DFS

Functions:

isfull(board):

- check if the 2D list, {board} has any space left
- return True if no space is left otherwise False

path_find(board,player,verbose=False):

- check the {board} and determine whether it contains a valid path of type {player}
- it is a path-finding algorithm based on DFS with a visited-states checking system to prevent looping
- a maximum depth of each piece is kept track of at every iteration for calculating the progress
- the progress is calculated by how far a group of pieces of a player stretches along the direction that is required to win and can be calculated by {max_depth/BOARD_SIZE}
- verbose is used to make the function print each iteration of the fringe
- return 1 if there is a winning path and return the “progress” if there are none

evaluate(board):

- the evaluation function of this game
- evaluate the {board}
- returns 1 or -1 if player X or player O wins respectively
- returns the progress result of X - progress result of O, if nobody wins

minimax(board, depth, alpha, beta, maximizing_player):

- do a DFS minimaxing of the {board} recursively, completing once there is a winner
- the function will recursively call itself until complete or upto {depth}
- for unlimited depth version parse {math.inf} into {depth}

- {alpha} and {beta} are the alpha nad beta parameters used in pruning, and are typically set to -math.inf, and math.inf respectively for the first call of this function
- {maximizing_player} is a boolean. True represents player X the maximizer while False represents player O the minimizer.
- this function returns the minimax result of the board which is a float from -1 to 1

get_best_move(board, player, verbose=False):
- a function that iterates through all valid moves of {board} and evaluates them using the result from the minimax function
- {player} is PLAYER_X or PLAYER_O, which informs the function of which perspective it should be looking at the game with
- {verbose} is used to print the score of each move as the program runs
- This function returns the best move for the player as (i,j) coordinates

get_best_move_iterative(board, player, depth, verbose=False):
- The same as the last function but the minimax depth limitation is parsed in as {depth} instead of the previous's DEPTH
- used to implement IDS minimax

main(verbose=True):
- The main game loop
- Initialize an empty board of size BOARD_SIZE x BOARD_SIZE
- Loop through each round of the game alternating between two players getting their best moves while printing the board at every stage
- {verbose} is used to activate the {verbose} in the get_best_move function
- returns when the game concludes (winner is announced)

main_iterative(max_depth=math.inf, progression=1, verbose=False):
- The same as previous but the minimax is IDS instead of DFS
- The depth is started at 1 and is incremented by {progression} after each round
- {max_depth} controls the maximum depth of the search
- for unlimited depth, parse math.inf into max_depth

To run:

1. Set the BOARD_SIZE
2. Set the DEPTH
3. Select the type of game you want the AI to play
    a. for unlimited DFS minimax: use main() with DEPTH = math.inf
    b. for limited DFS minimax: use main() and DEPTH = the limited depth number
    c. for unlimited IDS minimax: use main_iterative()
    d. for limited IDS minimax: use main_iterative(depth)
4. Run the code

The code can be found in:
<AI/Final.ipynb at main · GemsLoveNLP/AI · GitHub>