ARTIFICIAL INTELLIGENCE

FINAL PROJECT

The Game Of Domineering

Prof. Soma Dhavala

By:

Ananthram R U
PES120700088
Section - H

Akanksha
PES1201701799
Section - H

Mithun H M
PES1201700197
Section - E

ABSTRACT:

Domineering is a mathematical game played between two players who alternate by placing a domino of size 2x1 on a square board. The first player places the dominoes vertically and the second player places the dominoes horizontally on the board. The player unable to place his domino on the board loses.

Here in our game the user plays against the AI and thus the training of AI becomes crucial , thus we have used Min - Max algorithm along with Alpha-Beta pruning to help the AI figure out the moves .

The user is allowed to pick the size of domino board and is allowed to choose the type of domino (horizontal/vertical) he/she prefers to use and if he/she prefers to play first .

After this initial step the game with AI continues until one of them wins . Thus the Decision made by AI is based on the implementation of Min-Max algorithm . And all of this could be visualised on terminal by the gui( ) function.

INTRODUCTION:

The game of Domineering is is one of the simple yet intelligent games . Two players have a collection of dominoes which they place on the grid in turn, covering up squares. One player, Left, plays tiles vertically, while the other, Right, plays horizontally. As in most games in combinatorial game theory , the first player who cannot move loses. It is a two person , zero sum , complete information known , alternate moves , deterministic game.

APPROACH:

The approach we used to solve this problem is the same as we would to do to solve any two player board game , as the two players (user and AI) will be having alternate moves , the AI will be having complete information on the moves of the game and is supposed to figure out the moves based on the known info and alternating moves strategy . Unlike a puzzle game which is set to find the final state and is not influenced by any of the other moves , here the state of game changes after every move and the next best state must be evaluated in this zero -sum game.

Thus the proper usage of Min-Max algorithm along with Alpha -Beta pruning , which improves the estimation capacity of the game dramatically with larger board sizes , as more moves are left to be depicted , the above mentioned approach provides one of the solid solutions to this game. But unlike many other zero sum games, this game never ends in a draw and hence the game can only be evaluated to 1 and -1 where in the former, the AI wins(you lose), and in the latter , you

win. The idea behind Alpha-Beta is to reduce the complexity of the Min-Max algorithm by ignoring moves that would result in a negative outcome, basically it reduces the number of nodes that are evaluated by the minimax algorithm in its search tree.

```
def ai_turn(board,alpha,beta,move,first_player,second_player):

    if(not gamenotfinish(board,move)):
        score = win(move,first_player,second_player)
        return [-1,-1,score],board

    if(move == second_player):
        value = [-1,-1,-infinity]
        for row in range(len(board)):
            for column in range(len(board[row])):
                if(makemove(row,column,len(board),len(board[row]),board,move)):
                    state = board
                    score,state = ai_turn(state,alpha,beta,first_player,first_player,second_player)
                    state = undomove(row,column,state,move)
                    score[0], score[1] = row,column
                    if(value[2] < score[2]):
                        value = score
                    alpha = max(alpha,value[2])
                    if(alpha >= beta):
                        break
        return value,state

    else:
        value = [-1,-1,infinity]
        for row in range(len(board)):
            for column in range(len(boa
                if(makemove(row,column,len(board),len(board[row]),board,move)):
                    state = board
                    score,state = ai_turn(state,alpha,beta,second_player,first_player,second_player)
                    state = undomove(row,column,state,move)
                    score[0], score[1] = row,column
                    if(value[2] > score[2]):
                        value = score
                    beta = min(beta,value[2])
                    if(alpha >= beta):
                        break
        return value,state
```

```
def len(o: Sized)
Return the number of items in a container.
```

As you can see above ,**the idea or strategy** the algorithm uses is that it maintains **two values, alpha and beta**, which represent the **minimum score** that the maximizing player is assured of and the **maximum score** that the minimizing player is assured of respectively. Initially, alpha is negative infinity and beta is positive infinity, i.e. both players start with their worst possible score. Whenever the maximum score that the minimizing player (i.e. the "beta" player) is assured of becomes less than the minimum score that the maximizing player (i.e., the "alpha" player) is assured of (i.e. **beta < alpha**), the maximizing player need not consider further descendants of this node, as they will never be reached in the actual play.

The coding language used was python , the reason being its simplicity , and the following functions are used in the coding part of the solution :

```
19 > def choose_domino(first_player,second_player):···
32 > def gui(moves):···
39
40 > def makemove(x,y,rows,columns,moves,move):···
63
64 > def undomove(x,y,state,move):···
74
75 > def win(player,first_player,second_player):···
80
81 > def gamenotfinish(moves,move):···
92
93 > def startgame(first_player,second_player,rows,columns,moves):···
116
117 > def startgame_ai(first_player,second_player,rows,columns,state):···
178
179 > def ai_turn(board,alpha,beta,move,first_player,second_player):···
216
217 > def main():···
```

choose_domino(*first_player*,*second_player*):

This allows the players to choose among the vertical or horizontal domino he/she wishes , the AI is assigned the other domino.

gui(*moves*):

This function displays the user interface for the moves made by both the user and the AI.

makemove(*x*,*y*,*rows*,*columns*,*moves*,*move*):

This function takes the rows and columns as arguments and after checking if the move is valid or invalid , the move is made. This is used as a utility function in one for the other main functions as startgame() and startgame_ai().

def undomove(*x*,*y*,*state*,*move*):

As the name says it undoes a particular move , this is used as a utility function in one for the other main functions as startgame() and startgame_ai().

def win(*player*,*first_player*,*second_player*):

This function determines the winner of the game.

```
gamenotfinish(moves,move):
```

This is used as a utility function in one of the other functions.

```
startgame(first_player,second_player,rows,columns,moves):
```

This being one of the major functions in our code is used for two players to play against each other.

```
startgame_ai(first_player,second_player,rows,columns,state):
```

This is similar to the above function but is for a player to play against the AI.

```
ai_turn(board,alpha,beta,move,first_player,second_player):
```

This is the most important function of this code , which uses all the aspects of the artificial intelligence mechanism used in solution , this in turn is used in the above startgame_ai function and offers the best move for the AI.

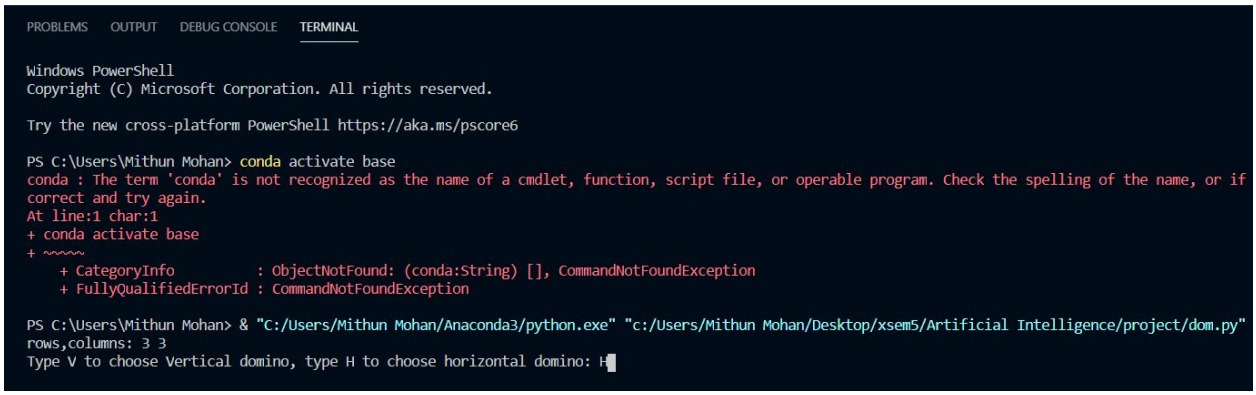
ADDITIONAL:

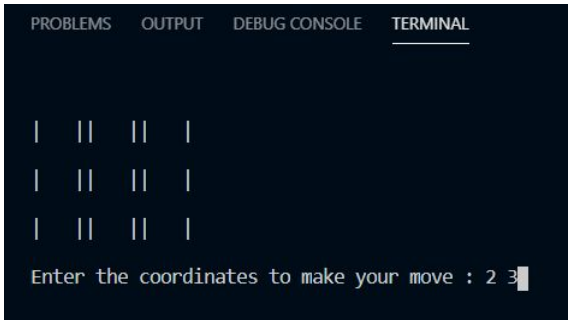
```
clean():
```

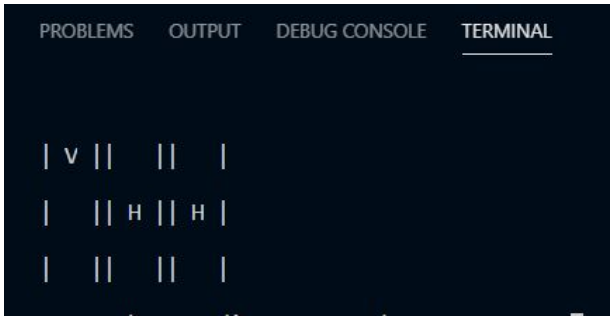
This function is used to clear the console , is used in startgame() and startgame_ai() functions.

RESULTS:

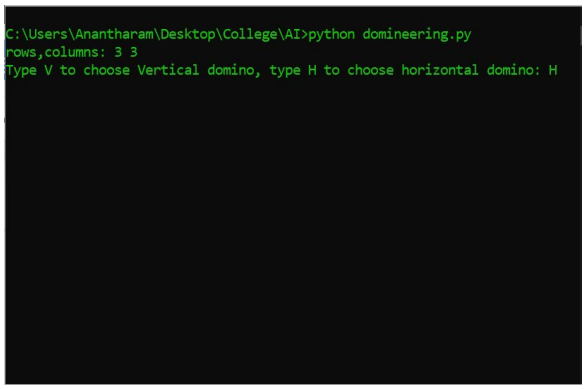
Thus by using this above mentioned approach this algorithm successfully ends up playing the game against human user . This could be visualized below:

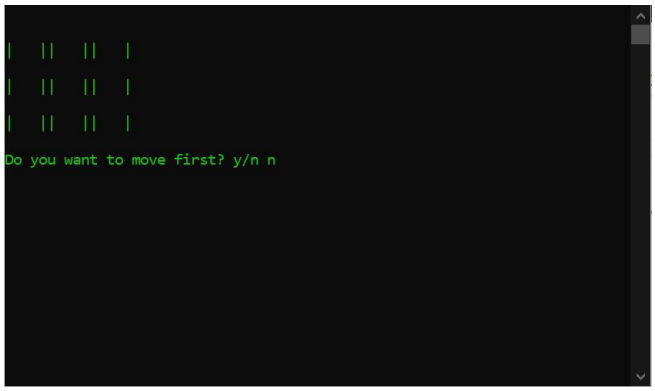




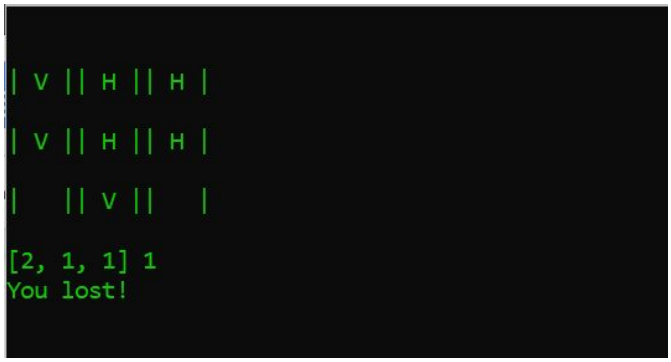


The **below** snapshots will give more insight on the **correctness** of the algorithm,





After a couple of moves ...



As you can see that the AI places the vertical domino in the correct square represented by the coordinates (2,1) rather than the other squares which would otherwise lead to a loss for the AI.

Thus we can see the player choosing 3 x 3 domino and his moves along with him losing , this was among some of the test cases used .

SUMMARY:

Thus a simple yet smooth utilisation of one of the most popular AI algorithms , Min - Max along with Alpha -Beta pruning can be seen , the experience of working on this solution is memorable and helpful. The AI algorithms thus could be improved further to produce even complicated results and solve tougher games in the future.