Tic Tac Toe

November,2020

DEsign tic tac toe game where the first player will have the winning strategy irrespective of the opponents move.

Team Members:
Anshul Sharma - 18ucs061
Mohammad Rahim Khan - 18ucs092
Ankush Agrawal - 18dcs011
Naman Indranil - 18ucs213

Introduction

Tic-tac-toe (a.k.a zeroes and crosses or Os-Xs) was originally a pen and paper game played in pairs of two by aligning either three crosses or three zero in a straight line that is either in a row or a column or even a diagonal. Jumping into the problem statement first let us know about some important details related to the game of noughts and crosses.

Tic Tac Toe comes off as an ideal case of perfect information game and zero sum. In other words it means that each player's profit is equivalent to the other person's loss i.e. net result of each move will be zero as a whole (+1 for one = -1 for other) and information about game state and possible moves are fully known.

We will be implementing this using the MINIMAX algorithm with alpha beta pruning which is best suited for such scenarios It is based on the idea of choosing the best possible move at a given state in the game assuming that the other player is playing optimally.

Also for coding the implementation based on MINIMAX algorithm we will be using Python which is a high-level,general-purpose programming language that emphasises on code simplicity readability. Benefits of using python includes ease of programming by simplifying the code with lesser steps in comparison to Java or C++. The Python is widely used in bigger organizations because of its multiple programming paradigms. They usually involve imperative and object-oriented functional programming. It has a comprehensive and large standard library that has automatic memory management and dynamic features.

Algorithm applied

Minimax algorithm with Alpha Beta Pruning

Minimax is a recursive algorithm which is based on the idea of choosing the best possible move at a given state in the game assuming that the other player is playing optimally. As the name of the algorithm suggests, the goal is to “minimize the maximum loss” (minimize the worst case scenario). Alpha-Beta pruning is not actually a new algorithm, rather an optimization technique for minimax algorithm. It reduces the computation time by a huge factor. This allows us to search much faster and even go into deeper levels in the game tree. It cuts off branches in the game tree which need not be searched because there already exists a better move available.

Alpha is the best value that the **maximizer** currently can guarantee at that level or above.

Beta is the best value that the **minimizer** currently can guarantee at that level or above.

METHODS & ALGORITHMS

1. **def printBoard(board)** : This method is called to print the board after each turn showing the places which are free to make a move on and the current state of board based on ‘X’ & ‘O’ positions.
2. **def insertLetter()** : This method takes input the player’s chosen character and returns a set containing Player-AI character.
3. **def whoGoesFirst()** : This method asks Player who plays the first move.
4. **def makeMove(board, character, pos)** : This method is called each time a move is played either by AI or by human player. It assigns the current play character to the chosen position on board.
5. **def playAgain()** : This method returns if the game is to be played again or not.
6. **def isWinner(board,character)** : This function returns “true” if player wins else it returns “false”. All possible winning combinations are checked in this function are:

(board[1]==character and board[2]==character and board[3]==character)

(board[4]==character and board[5]==character and board[6]==character)

(board[7]==character and board[8]==character and board[9]==character)

(board[1]==character and board[4]==character and board[7]==character)

(board[2]==character and board[5]==character and board[8]==character)

(board[3]==character and board[6]==character and board[9]==character)

(board[1]==character and board[5]==character and board[9]==character)

(board[3]==character and board[5]==character and board[7]==character)

7. **def isSpaceFree(board, pos)** : This function checks if a position on board is free or not.
8. **def getPlayerMove(board)** : This method checks whether the player has made a valid move or not. It also calls isSpaceFree() to check if the position played by Player was free or not.
9. **def minimax(board, depth, isMax, alpha, beta, computerLetter)** : This function uses the minimax algorithm along with alpha beta pruning to calculate the best score for the current move.
10. **def findBestMove(board, computerLetter)** : This function takes input the current state of board and the computer character and uses the minimax method to compute the best move with winning strategy.
11. **def isBoardFull(board)** : This method returns “true” if all positions on board are taken otherwise it returns “false”.
12. **The main body of the code** :

```
while playing:

        if turn == 'player':

            printBoard(theBoard)

            move = getPlayerMove(theBoard)

            makeMove(theBoard, playerLetter, move)
            if isWinner(theBoard, playerLetter):
                printBoard(theBoard)
                print('You won the game, Good Job!')
                playing = False
            else:
                if isBoardFull(theBoard):
                    printBoard(theBoard)
                    print('The game is a tie')
                    break
                else:
                    turn = 'computer'
        else:
            move = findBestMove(theBoard, computerLetter)
            makeMove(theBoard, computerLetter, move)

            if isWinner(theBoard, computerLetter):
                printBoard(theBoard)
                print('Sorry!, You lose the game')
                playing = False
            else:
                if isBoardFull(theBoard):
```

```
        printBoard(theBoard)

        print('The game is a tie')

        break

    else:

        turn = 'player'
```

Here the while loop runs till the game is over due to no available positions on board or one of the player or computer wins or the game ends in a draw or tie. The different methods which were explained above have been called inside the while for game play.

Experiments(Results)

To test the working of the project we subjected the gameplay according to the suggested moves for best possible result . Attached below are the examples of worst test case scenarios

(a)Starting from corner

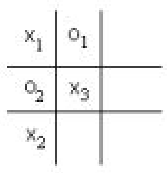

figure 1.1

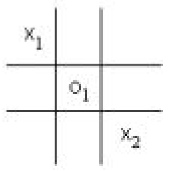

figure 1.2

Now our only hope is that the opponent will continue the symmetry as shown in figure 1.3.

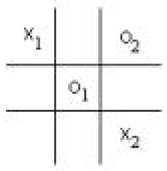

figure 1.3

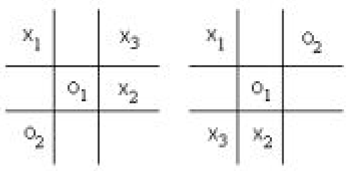

figure 1.4

As suggested in the above case,starting from the corner opens many possibilities of winning but the AI makes its moves in such a fashion so as to nullify our winning options.

(b) Starting from center

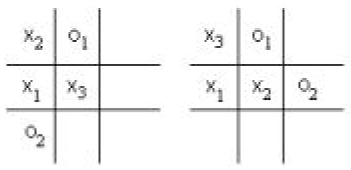

figure 1.5

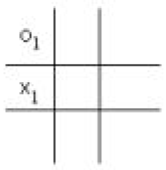

figure 1.6

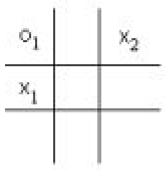

figure 1.7

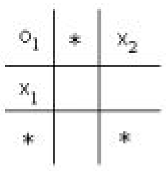

figure 1.8

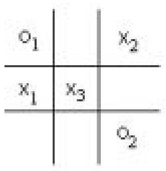

figure 1.9

Even starting from sides AI blocks our winning chances so the best result playing against this AI is to get a draw.

(c)Starting from center

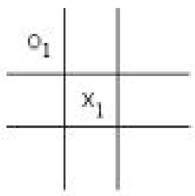

figure 1.10

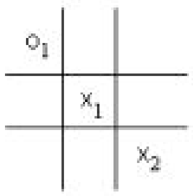

figure 1.11

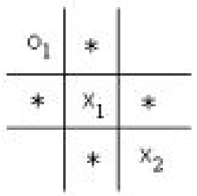

figure 1.12

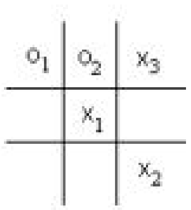

figure 1.13

Another possible scenario is to start from the center and try to build on it to have a winning chance but even in this scenario we are not able to win against AI's gameplay.

DISCUSSIONS, INSIGHTS & OBSERVATIONS:

Q1. Which algorithm would be the best suited to be used in making unbeatable Tic-Tac-Toe?

Ans. An algorithm can be thought of as a representation of the human thought process which thinks, “If I make this move, then my opponent can only make such moves, and each of those would let me win. So this is the right move to make.” And by referring to various articles we came to the conclusion of using the MINIMAX algorithm with Alpha-Beta pruning for this project.

Q2. Why is the Minimax algorithm appropriate to use?

Ans. Minimax is a recursive algorithm which is used to choose an optimal move for a player assuming that the opponent is also playing optimally. Its goal is to minimize the maximum loss which in other words means minimize the chance of the worst scenario.

Q3. What makes the algorithm unbeatable?

Ans. Due to the relatively small state space, which is approx. 3^9 = 196839 possible board combinations, it can easily search the whole game tree for an optimal solution, treating the game as a fully deterministic environment. Also, using Alpha-Beta pruning further optimizes the minimax algorithm.

Q4. What would be several other relevant functions.

Ans. Choosing which player makes the first move, choosing X or O ,printing the board after each move, etc are some of the relevant auxiliary functions which will be required for this project.

Conclusion:

During the research & development of this project we got to learn about working with the Minimax algorithm and getting hands-on experience in coding a program to carry out its functionality. We learnt how to make a code which thinks almost like a human when making a decision inorder to achieve best results.

While working on this project we have come to know about the search space related limitation that we can still use the Minimax algorithm for larger search spaces , but we have to limit the depth of our search, otherwise, we can end up computing results for a very long time or even forever. A similar case where we could have used MINIMAX algorithm but due to large search space size it is not recommended.

Long story short, the smaller the state space, the better results we can achieve with the Minimax algorithm and use of tic tac toe is a perfect example of how MINIMAX algorithm can be utilised with Alpha-Beta pruning in similar scenarios with appropriate search space size.

References & Bibliography:

Sr. No.	Link/Reference	Comment
1.	https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-3-tic-tac-toe-ai-finding-optimal-move/?ref=rp	The article was referenced for the theory and basic idea for implementation
2.	https://towardsdatascience.com/tic-tac-toe-creating-unbeatable-ai-with-minimax-algorithm-8af9e52c1e7d#:~:text=By%20now%20you%20should%20be,a%20wide%20variety%20of%20applications	The article was referenced for finding solutions to issues which came up during discussion of the project.
3.	https://www.cs.jhu.edu/~jorgev/cs106/ttt.pdf	The paper shows the best moves one can make to win a game of tic-tac-toe. These were tested against our project and the game ended in a DRAW or the human player losing the game
4.	Stuart Russell, Peter Norvig, *Artificial Intelligence – A Modern Approach*, 3rd Edition, Pearson Education, 2009	The book was referenced for getting acquainted with some concepts.
5.	https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/	The article was referenced to understand the implementation and working of alpha-beta pruning

Contribution of Team Members :

The following project being a group activity involved cohesive and cooperated contribution by all the team members. To begin with, we all collaborated to discuss relevant algorithms to be used for an Unbeatable tic tac toe AI Player.

We discussed the important functions that are to be used which included the ones mentioned in the methods used. Individual functions were assigned to all the members -

NAMAN

(1) Function for making the move for AI's turn.
(2) Function to check if Board Full or not
(3) Function for printing the board after each move

ANSHUL

(1) Function for finding the best move for AI's turn
(2) Function to get Human Player's move.
(3) Function to ask which player goes first.

RAHIM

(1) MINIMAX algorithm function.
(2) Is Winner function to determine the winner of the game
(3) Play again function

ANKUSH

(1) Main function body
(2) Function for determining if a space is free
(3) Function for inserting the character on each move.

In addition to above duties each member was assigned to test the individual functions as well as the end project result when it competed against varying difficulties and varying strategies available in Tic-Tac-Toe.

The Links For This Project Repository:

1. GitHub : https://github.com/namanindranil2299/tictactoe_ai
2. Drive : https://drive.google.com/drive/folders/1xi0XxEPE2Zc8biNkn2ZHdFBge9h-ep0g?usp=sharing

   Note : To view in Google Drive use the LNMIIT provided google account