**Vatsal Kukreti**

**Master Coder submission**

Project: Tic tac toe code

This code will play tic tac toe with you.

**Rules of the game**: The rules of tic-tac-toe are simple: in a grid-like base of nine positions, the first player to get three of their marks in a row (up, down, across, or diagonally) is the winner.

**Concept applied**: The idea is that every time a move is made by the user, the algorithm computes all the movements it might make and all the games it could start, and it gives each move a "score" (higher the score, the lower the chance of losing). Then it moves based on the move with the highest score. The game continues in this cycle till it ends.

Binary search is slightly altered according to the code. It now performs binary search on the second value in a list of tuples.

The code works as follows :

The game is on a while loop. While a definite outcome has not been reached, the game will continue playing

* Every iteration, the board is printed and the user is asked to make a move. Once it is determined that the computer must play a move (and there are more than one possible moves), it enters the score calculation.
* How the calculation works is by creating "for loops" for every possible move. If there are further choices possible, it keeps creating loops until all possible outcomes have been accounted for. If an outcome is found, it adds or subtracts from the score and breaks the loop (there is no point in seeing other possible moves when an immediate move wins or loses the game).
  + *This realisation came late to me, and when it did, it made the code, at least theoretically, unbeatable (you can either lose against it or draw only).*
* A dictionary holds all the scores tied to the iteration number it was on, and here is where binary search is used.
* The scores are tied to the iteration number they were created on, which is irrelevant to their actual positions on the board. So, a list of tuples was created, mapping all iteration values to their position on the board for each time the "score calculator" loop is run.

After finding the max value of score, the iteration number of the best move is found. Using (modified) binary search, the position in list, and hence the board position of the best move is found. This best move is played, and the code once again checks for an output.

Thank you for your time.