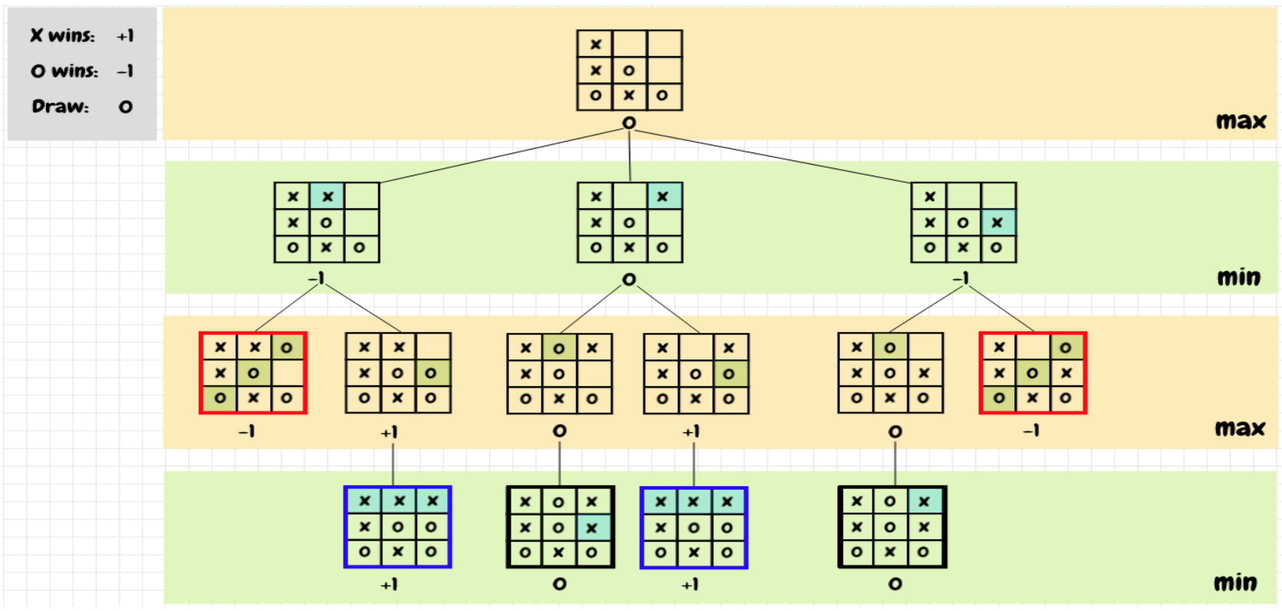
The Unbeatable Tic Tac Toe AI

By: Daniel Dockins

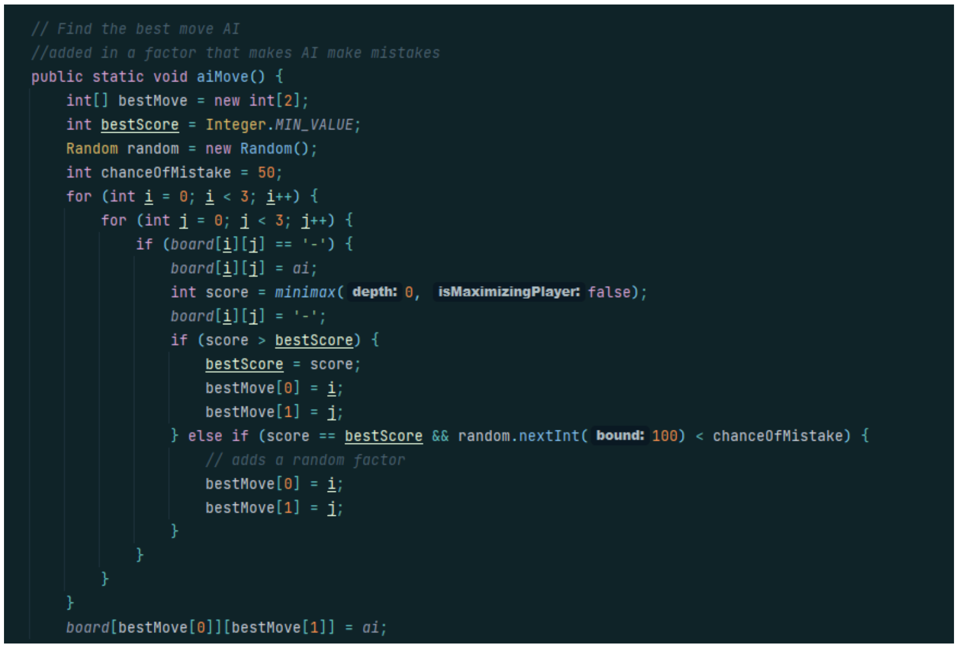
For this extra credit assignment I wanted to do something that I found interesting. Recently I have been watching a lot of videos on AI (Artificial Intelligence) playing video games or Chat GPT making things such as websites and streaming personalities and it made me wonder if I could make my own AI. In this research I started by thinking about what game I would play against an AI with. My first thought was Pong but I'm still a novice when it comes to programming so I thought about previous assignments and how they could help me when deciding a game. I finally came up with making an AI to play against me in Tic-Tac-Toe. Tic-Tac-Toe is a fairly easy game to make in Java but I wasn't sure on how to add AI into it. I watched videos and asked teachers and finally found what I needed by adding a MinMax algorithm.

The MinMax algorithm is a decision making algorithm that finds the best move in order to beat the opponent. In this algorithm one player is called the Maximizer and the other is called the Minimizer. The Maximizer works to get the highest score while the minimizer tries to get the lower score by using counter moves.

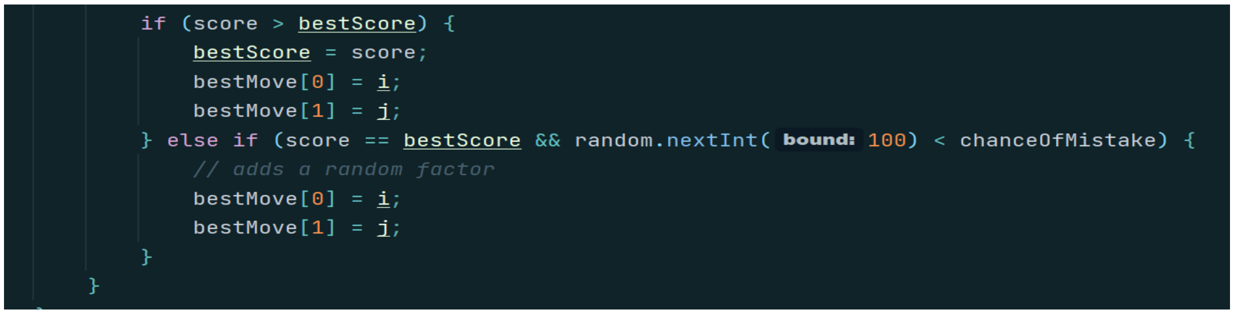


An interesting fact I discovered while reading about this algorithm is that they used the MinMax algorithm in the DeepBlue project (A.I chess game) to defeat Garry Kasparov, a GM chess master at that time. After reading this I didnt feel so bad after losing to this AI in Tic-Tac-Toe.

My code is an easy Tic-Tac-Toe game with an AI opponent using the minimax algorithm. The game is played on a 3x3 Tic-Tac-Toe board, and the player who I made (X) competes against the AI who I made (O). The game starts by generating the game board with empty cells using ('-') and displaying it. The game then enters a loop that continues until the game is over meaning you lose or you tie with the AI. In each loop the player makes a move by entering the row and column of their desired move using the numbers 1, 2 and 3 and the game makes sure the move is valid so it ensures that entering a row that doesn't exist does not break the game . After the player makes thier move the game board is printed again showing what the AI chose. If the game is not over after the player's move the AI will make a move using the minimax algorithm. The minimax algorithm evaluates all possible moves from the current state of the game and assigns a score to each move based on whether the move leads to a loss, win or a draw. The AI will choose the move with the highest score since it is the Maximizer and based off of that will make the best move. A random factor is also introduced to make the AI occasionally make mistakes making it seem like you are playing a real person but it seems like that never happens considering I have lost to it several times. The game is considered over when either the game board is full meaning there are no more cells or the AI has won. The winner is determined by the basic rules of Tic-Tac-Toe meaning you have to create a pair of 3 X's or O's be it linear, horizontal or diagonal. If a winner is found, the game ends, and the winner is printed. Otherwise, the game is a draw and it will print a '-'.



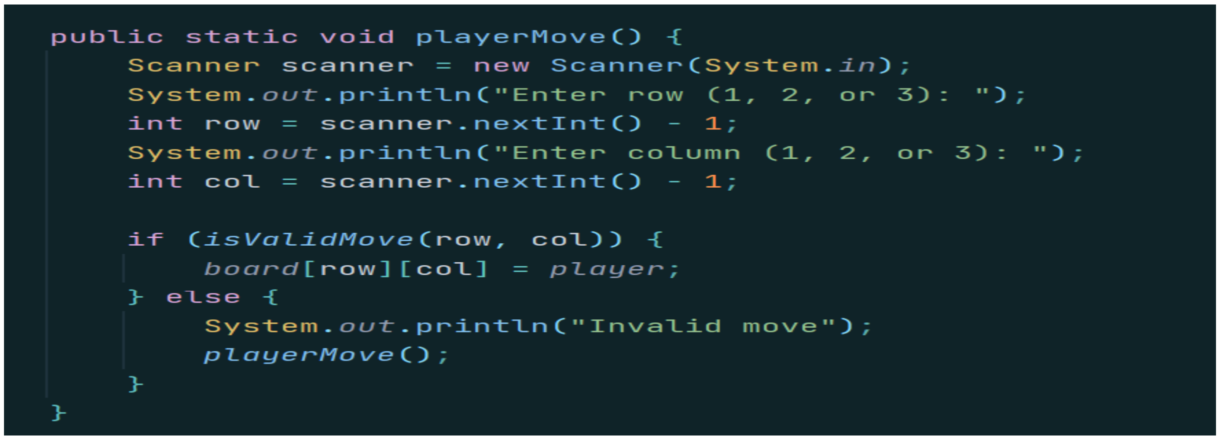
```java
// Find the best move AI
//added in a factor that makes AI make mistakes
public static void aiMove() {
    int[] bestMove = new int[2];
    int bestScore = Integer.MIN_VALUE;
    Random random = new Random();
    int chanceOfMistake = 50;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (board[i][j] == '-') {
                board[i][j] = ai;
                int score = minimax( depth: 0,  isMaximizingPlayer: false);
                board[i][j] = '-';
                if (score > bestScore) {
                    bestScore = score;
                    bestMove[0] = i;
                    bestMove[1] = j;
                } else if (score == bestScore && random.nextInt( bound: 100) < chanceOfMistake) {
                    // adds a random factor
                    bestMove[0] = i;
                    bestMove[1] = j;
                }
            }
        }
    }
    board[bestMove[0]][bestMove[1]] = ai;
```



```java
                if (score > bestScore) {
                    bestScore = score;
                    bestMove[0] = i;
                    bestMove[1] = j;
                } else if (score == bestScore && random.nextInt( bound: 100) < chanceOfMistake) {
                    // adds a random factor
                    bestMove[0] = i;
                    bestMove[1] = j;
                }
            }
```

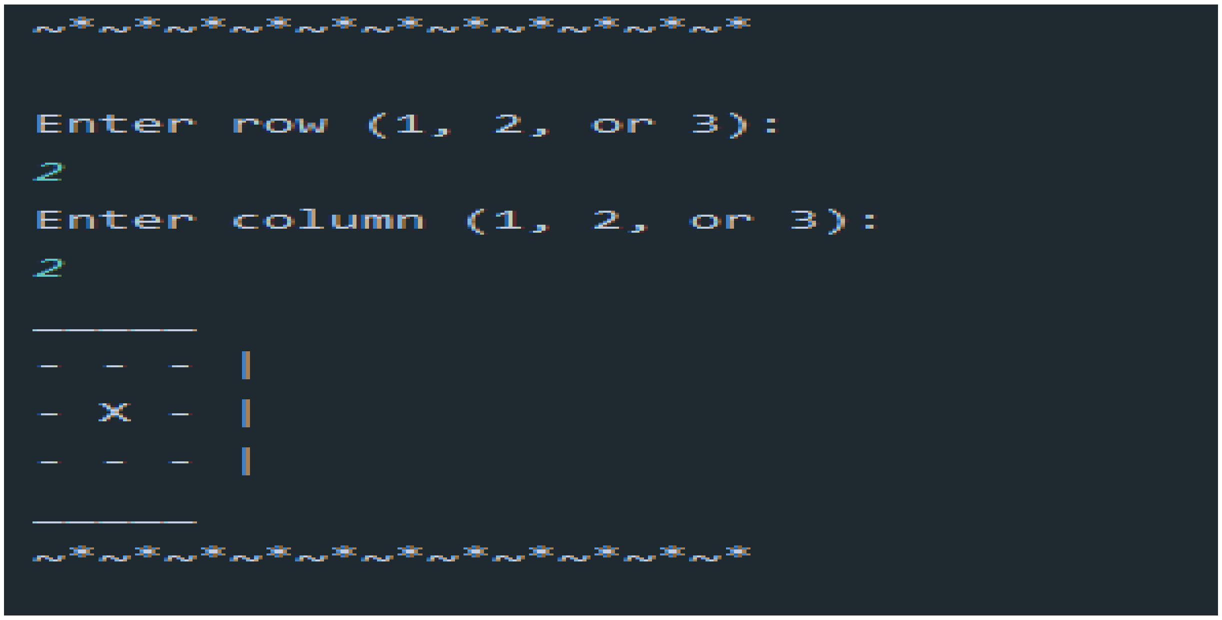
The chanceOfMistake variable was supposed to represent the probability of the AI making a random mistake and it is set to 50% shown in the code above. This introduces randomness in the AI's decision making process making it seem like you are playing a human. The randomness seems to not work though seeing as it always wins. I guess the AI takeover is starting with the Tic-Tac-Toe rebelion.

The player's moves are made based on the user's input in the playerMove() method in the code shown below. This method takes the row and column of the game board as inputs, representing the position where the player wants to mark their (X). The playerMove() method first checks if the specified position is a valid move or if the position is empty on the game board. If the position is a valid move it will update the game board with the player's move by marking the corresponding position to (X) as shown in the second picture below.



```java
public static void playerMove() {
    Scanner scanner = new Scanner(System.in);
    System.out.println("Enter row (1, 2, or 3): ");
    int row = scanner.nextInt() - 1;
    System.out.println("Enter column (1, 2, or 3): ");
    int col = scanner.nextInt() - 1;

    if (isValidMove(row, col)) {
        board[row][col] = player;
    } else {
        System.out.println("Invalid move");
        playerMove();
    }
}
```



```
~*~*~*~*~*~*~*~*~*~*~*

Enter row (1, 2, or 3):
2
Enter column (1, 2, or 3):
2
________
-  -  -  |
-  X  -  |
-  -  -  |
________
~*~*~*~*~*~*~*~*~*~*~*
```

Implementing a game like Pong or Chess with an AI using the Minimax algorithm would require some additional knowledge compared to Tic-Tac-Toe. These games have more complex rules and gameplay mechanics that would be hard for me to program in Java. For Pong the Minimax algorithm could be used to determine the optimal move for the AI player by considering the current state of the game, meaning the position of the ball and the position of the human player's paddle. For Chess you would need to have a good understanding of the rules. You would need to input the chessboard, pieces, and legal moves. The AI player would need to evaluate the board state like in the Tic-Tac-Toe game and generate possible moves using the Minimax algorithm to search through the game and determine the best move. Both of these implementations would need a good understanding of the game mechanics and programming skills that I am still learning to achieve. During the summer in my free time I hope to gain said skills and try out these games to better understand AI in programming. During making the Tic-Tac-Toe AI I had no understanding of how to code an AI but after I found an interesting field in computer science that I hope to pursue more in the future.

<u>Work Cited</u>

1.
baeldung. “Introduction to Minimax Algorithm with a Java Implementation.” *Baeldung*, 26 May 2022,
https://www.baeldung.com/java-minimax-algorithm#:~:text=Minimax%20is%20a%20decision%2Dmaking,other%20player%20is%20a%20minimizer.

2.Forbes, Elliot. “The Min-Max Algorithm in Java.” *TutorialEdge*,
https://tutorialedge.net/artificial-intelligence/min-max-algorithm-in-java/.

3.Lague, Sabastian. “Algorithms Explained – Minimax and Alpha-Beta Pruning.” *YouTube*, YouTube, 20 Apr. 2018, https://www.youtube.com/watch?v=l-hh51ncgDI.