code

```
import math


class TicTacToe:
    def __init__(self):
        self.board = [' ' for _ in range(9)] # Representing the 3x3 board as a list
        self.current_winner = None

    def print_board(self):
        for row in [self.board[i*3:(i+1)*3] for i in range(3)]:
            print('| ' + ' | '.join(row) + ' |')

    @staticmethod
    def print_board_nums():
        number_board = [[str(i) for i in range(j*3, (j+1)*3)] for j in range(3)]
        for row in number_board:
            print('| ' + ' | '.join(row) + ' |')

    def available_moves(self):
        return [i for i, spot in enumerate(self.board) if spot == ' ']

    def empty_squares(self):
        return ' ' in self.board

    def num_empty_squares(self):
        return self.board.count(' ')

    def make_move(self, square, letter):
        if self.board[square] == ' ':
            self.board[square] = letter
            if self.winner(square, letter):
                self.current_winner = letter
            return True
        return False

    def winner(self, square, letter):
        row_index = square // 3
        row = self.board[row_index*3:(row_index+1)*3]
        if all([spot == letter for spot in row]):
            return True

        col_index = square % 3
        column = [self.board[col_index+i*3] for i in range(3)]
        if all([spot == letter for spot in column]):
            return True

        if square % 2 == 0:
            diagonal1 = [self.board[i] for i in [0, 4, 8]] # Top-left to bottom-right diagonal
            if all([spot == letter for spot in diagonal1]):
                return True
```

```
            diagonal2 = [self.board[i] for i in [2, 4, 6]] # Top-right to bottom-left diagonal
            if all([spot == letter for spot in diagonal2]):
                return True

        return False

    def minimax(board, maximizer, alpha, beta):
        if board.current_winner:
            if board.current_winner == 'O':
                return -1
            else:
                return 1
        elif not board.empty_squares():
            return 0

        if maximizer:
            max_eval = -math.inf
            for move in board.available_moves():
                board.make_move(move, 'X')
                eval = minimax(board, False, alpha, beta)
                board.board[move] = ' '
                max_eval = max(max_eval, eval)
                alpha = max(alpha, eval)
                if beta <= alpha:
                    break
            return max_eval
        else:
            min_eval = math.inf
            for move in board.available_moves():
                board.make_move(move, 'O')
                eval = minimax(board, True, alpha, beta)
                board.board[move] = ' '
                min_eval = min(min_eval, eval)
                beta = min(beta, eval)
                if beta <= alpha:
                    break
            return min_eval

    def get_best_move(board):
        best_move = None
        best_eval = -math.inf
        alpha = -math.inf
        beta = math.inf
        for move in board.available_moves():
            board.make_move(move, 'X')
            eval = minimax(board, False, alpha, beta)
            board.board[move] = ' '
            if eval > best_eval:
                best_eval = eval
                best_move = move
                alpha = max(alpha, eval)
        return best_move

    def play():
        game = TicTacToe()
        print("Welcome to Tic-Tac-Toe!")
        print("Here's the current board:")
        game.print_board_nums()
        print("To make a move, enter a number from 0-8 indicating the position on the board.")
```

```
    print("You are 'O' and the AI is 'X'. Let's start!")
    while game.empty_squares():
        human_move = None
        while human_move not in game.available_moves():
            try:
                human_move = int(input("Enter your move (0-8): "))
            except ValueError:
                print("Please enter a number from 0-8.")
        game.make_move(human_move, 'O')
        if game.current_winner:
            break
        ai_move = get_best_move(game)
        game.make_move(ai_move, 'X')
        print("\nAI has made its move:")
        game.print_board()
        if game.current_winner:
            break

    if game.current_winner:
        print(f"{game.current_winner} wins!")
    else:
        print("It's a tie!")

play()
```

© 2018-2020 dndsofthub All Rights Reserved