Canonical Solution: Tic Tac Toe & Minimax

Eckel, TJHSST AI 1, Fall 2019

```python
import sys

win_sets = [[0, 1, 2], [3, 4, 5], [6, 7, 8], [0, 3, 6], [1, 4, 7], [2, 5, 8], [0, 4, 8], [2, 4, 6]]


def print_board(board):
    print("\nCurrent board: \n" + board[0:3] + "     012\n" + board[3:6] + "     345\n" + board[6:9] + "     678\n")


def end_test(state):  # Returns 1 for X victory, 0 for tie, -1 for O victory, -2 for unfinished.  How does this work?
    for i1, i2, i3 in win_sets:
        if "X" == state[i1] == state[i2] == state[i3]:
            return 1
        if "O" == state[i1] == state[i2] == state[i3]:
            return -1
    if state.count(".") == 0:
        return 0
    return -2


def get_moves(state):
    return [index for index in range(len(state)) if state[index] == "."]


def max_move(state):  # Implements the max part of maximin.  Read it carefully.  How does this work?
    test = end_test(state)
    return test if test >= -1 else max([min_move(state[:move] + "X" + state[move + 1:]) for move in get_moves(state)])


def min_move(state):  # Implements the min part of maximin.
    test = end_test(state)
    return test if test >= -1 else min([max_move(state[:move] + "O" + state[move + 1:]) for move in get_moves(state)])
```

```
def play_game(state, is_human_player):  # Recursive method that plays each round of the game.
    print_board(state)
    test = end_test(state)
    if test > -2:
        return test
    team = state.count(".") % 2  # Numerical representation of current piece; X is 1 and O is 0 (zero)
    piece = "OX"[team]           # Character representation of current piece
    moves = get_moves(state)
    if is_human_player:
        print("You can move to any of these spaces: " + ', '.join([str(x) for x in moves]) + ".")  # How does this work?
        location = int(input("Your choice? "))
        return play_game(state[:location] + piece + state[location + 1:], False)
    else:
        choice, current_best = moves[0], -1
        for move in moves:
            result = [max_move, min_move][team](state[:move] + piece + state[move + 1:])  # How does this work?  :-)
            if team == 0:
                result = -1 * result  # So the next three lines are easier, make it so 1 always means victory
            print("Moving at %s results in a %s." % (move, ["loss", "tie", "win"][result + 1]))
            if result > current_best:
                choice, current_best = move, result
        print("\nI choose space %s." % choice)
        return play_game(state[:choice] + piece + state[choice + 1:], True)


def run_game(board):
    player_first = False  # Computer moves first by default
    if board == ".........":
        if input("Should I be X or O? ") == "O":  # Player only goes first if board is blank and O is specified
            player_first = True
    victor = play_game(board, player_first)  # Plays the game, getting value for victor based on end_test
    if victor == (-1 if player_first else 1):  # How does this line work?
        print("I win!")
    elif victor == 0:
        print("We tied!")
    else:
        print("You win!")


run_game(sys.argv[1])
```