```
"""
Monte Carlo Tic-Tac-Toe Player"""

import random
import poc_ttt_gui
import poc_ttt_provided as provided

# Constants for Monte Carlo simulator
# You may change the values of these constants as desired, but
#  do not change their names.
NTRIALS = 100       # Number of trials to run
SCORE_CURRENT = 1.0 # Score for squares played by the current player
SCORE_OTHER = 1.0   # Score for squares played by the other player

# Add your functions here.
def mc_trial(board, player):
    """This function palys the board randomly placing X and O"""
    avail_squares = board.get_empty_squares()

    for dummy in range(len(avail_squares)):
        i_square = random.randrange(len(avail_squares))
        pos_square = avail_squares[i_square]
        board.move(pos_square[0], pos_square[1], player)

        if(board.check_win() != None):
            break

        avail_squares.pop(i_square)
        player = provided.switch_player(player)


def mc_update_scores(scores, board, player):
    """This function updates the score of the trail board updating +1 for machine player and -1 for other
player """


    dim = board.get_dim()
    scores_trial =[[0 for dummy_i in range(dim)] for dummy_j in range(dim)]
    other_player = provided.switch_player(player)


    if(board.check_win() == player):
        for row in range(dim):
            for col in range(dim):
                if(board.square(row,col) == player):
                    scores_trial[row][col] += SCORE_CURRENT
                if(board.square(row,col) == other_player):
                    scores_trial[row][col] -= SCORE_OTHER


    if(board.check_win() == other_player):
         for row in range(dim):
            for col in range(dim):
                if(board.square(row,col) == player):
                    scores_trial[row][col] -= SCORE_CURRENT
                if(board.square(row,col) == other_player):
                    scores_trial[row][col] += SCORE_OTHER


    for row in range(dim):
        for col in range(dim):
            scores[row][col] += scores_trial[row][col]
```

```
def get_best_move(board, scores):
    """ returns the next best square to move"""

    empty_squares = board.get_empty_squares()
    if (len(empty_squares) != 0):
        square_list = []
        max_score = float("-inf")
        for dummy in range(len(empty_squares)):
            temp = empty_squares[dummy]
            row = temp[0]
            col = temp[1]
            if(scores[row][col] > max_score):
                max_score = scores[row][col]

        for dummy in range(len(empty_squares)):
            temp = empty_squares[dummy]
            row = temp[0]
            col = temp[1]
            if(scores[row][col] == max_score):
                square_list.append((row,col))

        dummy_i = random.randrange(len(square_list))
        return square_list[dummy_i]



def mc_move(board, player, trials):
    """ moves the machine player to the next best predicted square"""

    score_b = [[0 for dummy_i in range(board.get_dim())]for dummy_j in range(board.get_dim())]
    print score_b

    for dummy in range(trials):
        board_trial = board.clone()
        mc_trial(board_trial, player)
        mc_update_scores(score_b, board_trial, player)

    return get_best_move(board, score_b)




# Test game with the console or the GUI.  Uncomment whichever
# you prefer.  Both should be commented out when you submit
# for testing to save time.

#provided.play_game(mc_move, NTRIALS, False)
poc_ttt_gui.run_gui(3, provided.PLAYERX, mc_move, NTRIALS, False)
```