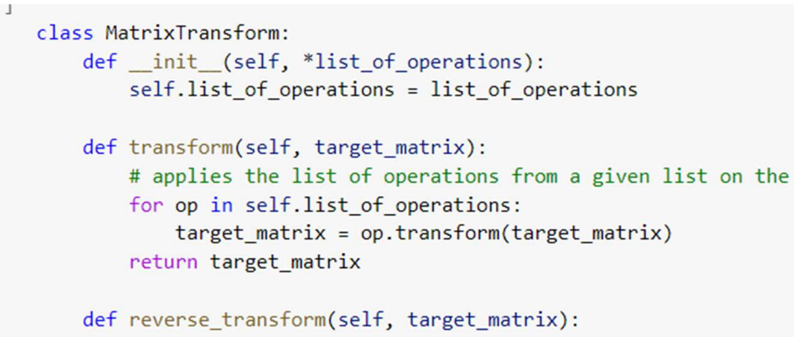
Q3 Report-> Monte Carlo Tree Search Agent

NOTEBOOK NAME - **Q3_RL_A5_CS21MDS14025_MCTS.ipynb**

Q3(a) Develop an MCTS agent with No. of Simulations being configurable.

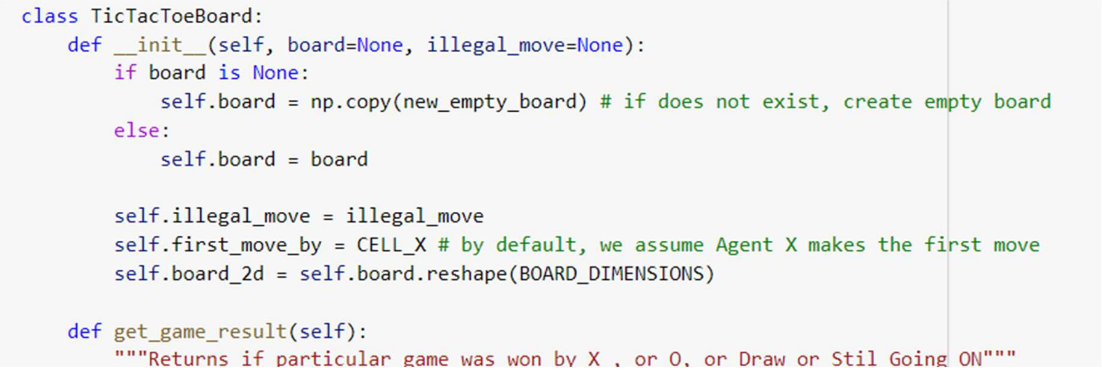
The code has these important blocks/segments in it :

1. A matrix transforms class & associated classes that do rotations & flips –helpful in finding equivalent board configurations



```
class MatrixTransform:
    def __init__(self, *list_of_operations):
        self.list_of_operations = list_of_operations

    def transform(self, target_matrix):
        # applies the list of operations from a given list on the
        for op in self.list_of_operations:
            target_matrix = op.transform(target_matrix)
        return target_matrix

    def reverse_transform(self, target_matrix):
```

2. A class for Tic Tac Toe environment that decides illegal moves, if a player won(*get_game_result()*), prints current board configuration, decides it is the turn for which player (*give_turns_to_alternate_players()*), etc

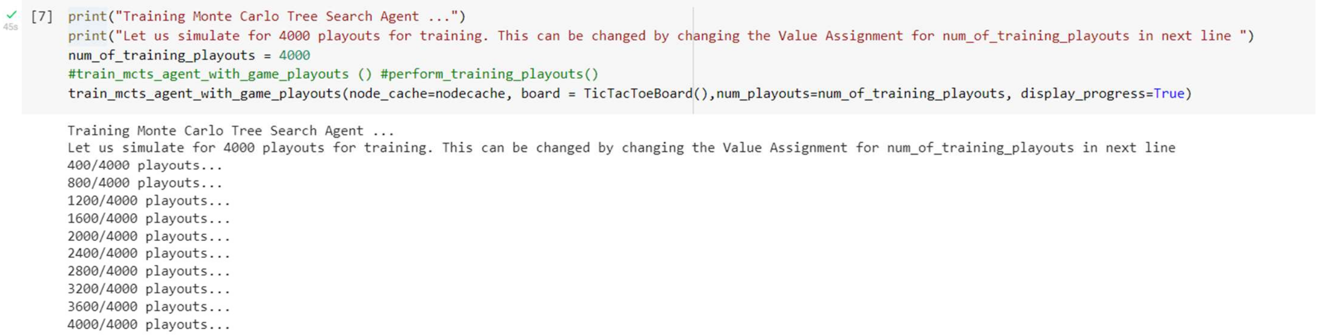


```
class TicTacToeBoard:
    def __init__(self, board=None, illegal_move=None):
        if board is None:
            self.board = np.copy(new_empty_board) # if does not exist, create empty board
        else:
            self.board = board

        self.illegal_move = illegal_move
        self.first_move_by = CELL_X # by default, we assume Agent X makes the first move
        self.board_2d = self.board.reshape(BOARD_DIMENSIONS)

    def get_game_result(self):
        """Returns if particular game was won by X , or O, or Draw or Stil Going ON"""
```

3. Then we have **helper functions** like *play_game_till_end*()→ plays till the end of the episode & play_games_n_return_stats ()→ plays game for n episodes /simulations & returns the win /loss percentage; there are other important functions like play_random_move(); if_given_player_won()
4. *Class BoardConfigCache* stores the results for each past unique configurations (that is those who are not equivalent upon rotating or flipping)
5. *Class TreeNode* that implements MCTS

We can set the no. of Simulations run for training here –(i.e. it is not hard coded & can be input by user)

By changing the parameter *num_of_training_playouts* – we can train the model a different number of simulation episodes.

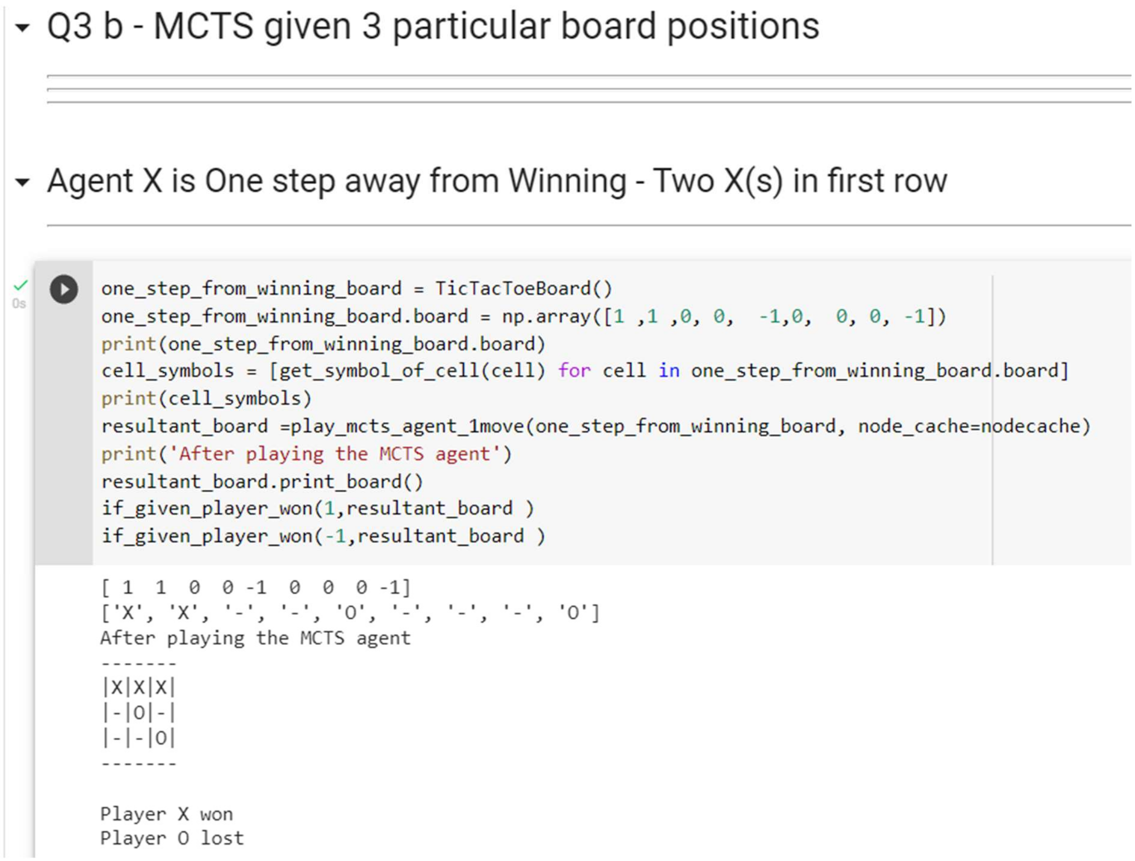


```
[7] print("Training Monte Carlo Tree Search Agent ...")
    print("Let us simulate for 4000 playouts for training. This can be changed by changing the Value Assignment for num_of_training_playouts in next line ")
    num_of_training_playouts = 4000
    #train_mcts_agent_with_game_playouts () #perform_training_playouts()
    train_mcts_agent_with_game_playouts(node_cache=nodecache, board = TicTacToeBoard(),num_playouts=num_of_training_playouts, display_progress=True)
```

```
Training Monte Carlo Tree Search Agent ...
Let us simulate for 4000 playouts for training. This can be changed by changing the Value Assignment for num_of_training_playouts in next line
400/4000 playouts...
800/4000 playouts...
1200/4000 playouts...
1600/4000 playouts...
2000/4000 playouts...
2400/4000 playouts...
2800/4000 playouts...
3200/4000 playouts...
3600/4000 playouts...
4000/4000 playouts...
```

Q3 b) Test the MCTS Moves in three particular Board configurations:

CASE 1 of 3(b): MCTS agent (Agent X) is one move away from win (BELOW)



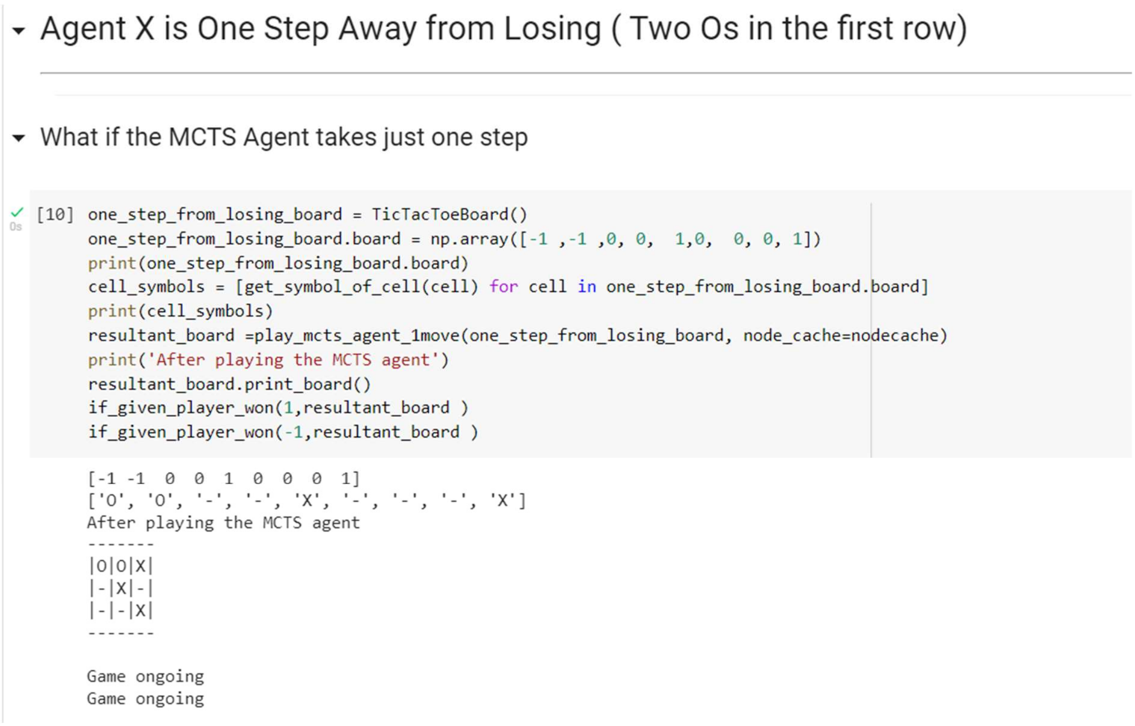
Q3 b - MCTS given 3 particular board positions

Agent X is One step away from Winning - Two X(s) in first row

```
one_step_from_winning_board = TicTacToeBoard()
one_step_from_winning_board.board = np.array([1 ,1 ,0, 0,  -1,0,  0, 0, -1])
print(one_step_from_winning_board.board)
cell_symbols = [get_symbol_of_cell(cell) for cell in one_step_from_winning_board.board]
print(cell_symbols)
resultant_board =play_mcts_agent_1move(one_step_from_winning_board, node_cache=nodecache)
print('After playing the MCTS agent')
resultant_board.print_board()
if_given_player_won(1,resultant_board )
if_given_player_won(-1,resultant_board )
```

```
[ 1  1  0  0 -1  0  0  0 -1]
['X', 'X', '-', '-', 'O', '-', '-', '-', 'O']
After playing the MCTS agent
-------
|X|X|X|
|-|O|-|
|-|-|O|
-------

Player X won
Player O lost
```

CASE 2 of 3b- MCTS agent (Agent X) is one move away from loss (BELOW)



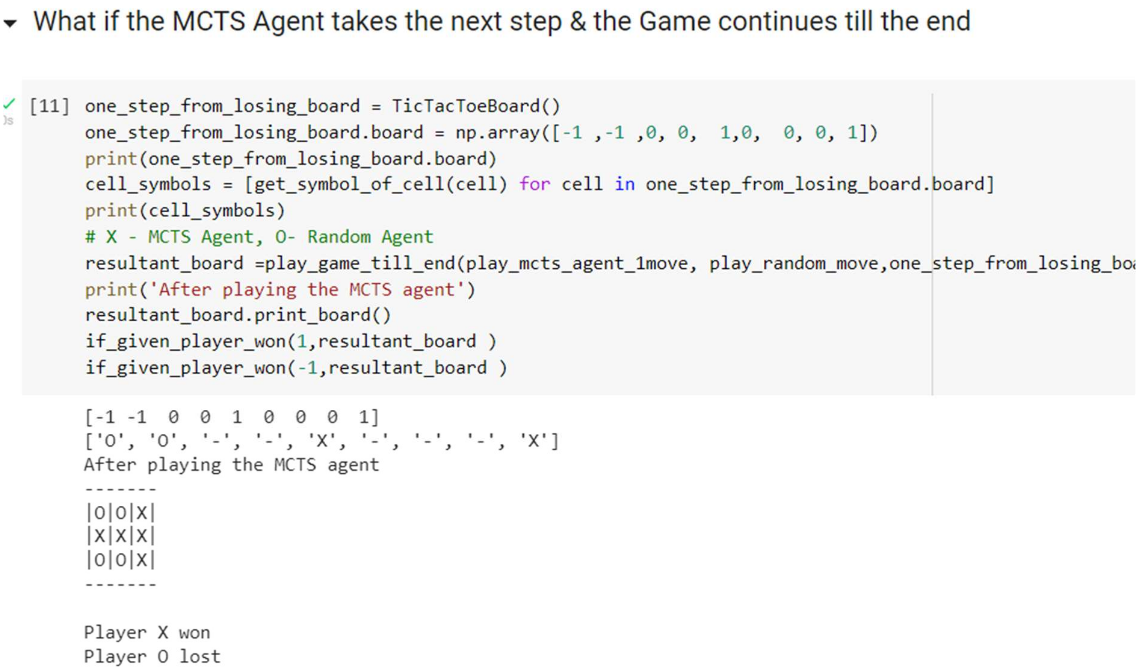
Agent X is One Step Away from Losing (Two Os in the first row)

What if the MCTS Agent takes just one step

```
[10] one_step_from_losing_board = TicTacToeBoard()
     one_step_from_losing_board.board = np.array([-1 ,-1 ,0, 0,  1,0,  0, 0, 1])
     print(one_step_from_losing_board.board)
     cell_symbols = [get_symbol_of_cell(cell) for cell in one_step_from_losing_board.board]
     print(cell_symbols)
     resultant_board =play_mcts_agent_1move(one_step_from_losing_board, node_cache=nodecache)
     print('After playing the MCTS agent')
     resultant_board.print_board()
     if_given_player_won(1,resultant_board )
     if_given_player_won(-1,resultant_board )
```

```
[-1 -1  0  0  1  0  0  0  1]
['O', 'O', '-', '-', 'X', '-', '-', '-', 'X']
After playing the MCTS agent
-------
|O|O|X|
|-|X|-|
|-|-|X|
-------

Game ongoing
Game ongoing
```

The above configuration if the MCTS Agent takes just one step. If we go ahead till the end of the episode (game over) this is what happens:



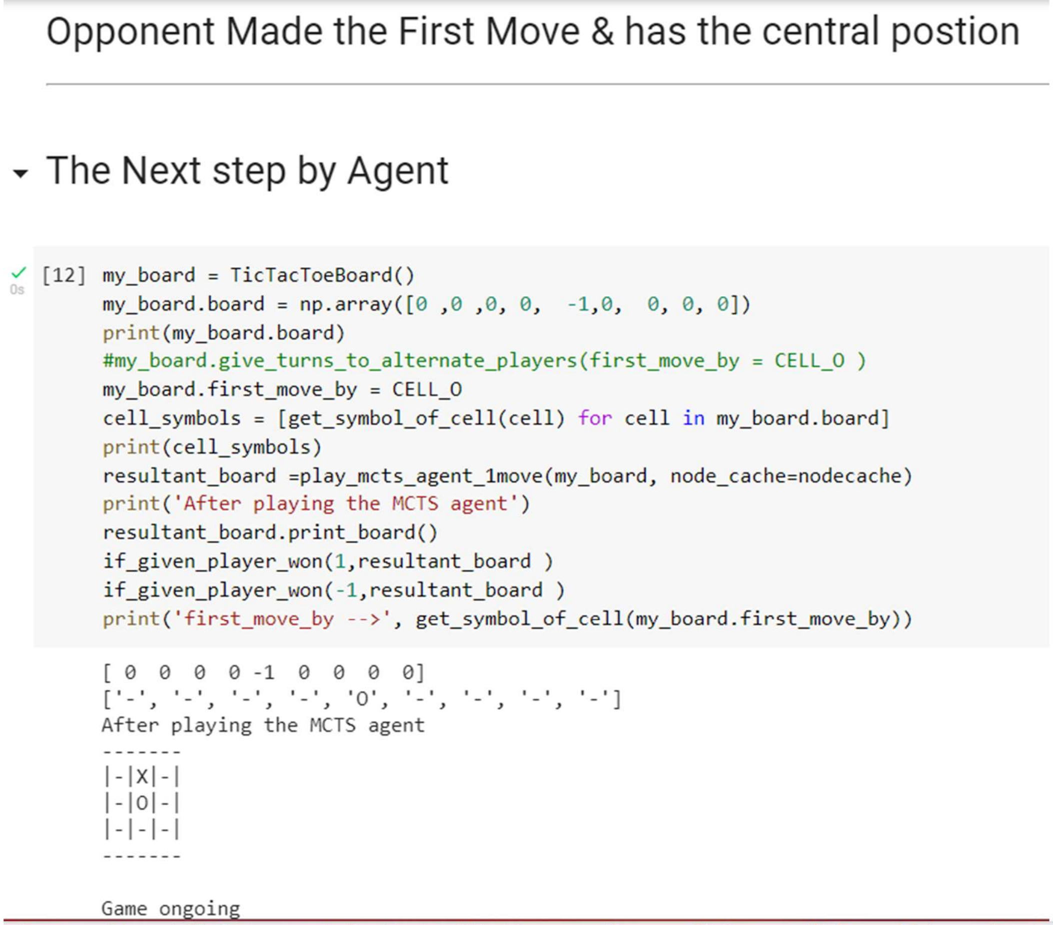
What if the MCTS Agent takes the next step & the Game continues till the end

```
[11] one_step_from_losing_board = TicTacToeBoard()
     one_step_from_losing_board.board = np.array([-1 ,-1 ,0, 0,  1,0,  0, 0, 1])
     print(one_step_from_losing_board.board)
     cell_symbols = [get_symbol_of_cell(cell) for cell in one_step_from_losing_board.board]
     print(cell_symbols)
     # X - MCTS Agent, O- Random Agent
     resultant_board =play_game_till_end(play_mcts_agent_1move, play_random_move,one_step_from_losing_boa
     print('After playing the MCTS agent')
     resultant_board.print_board()
     if_given_player_won(1,resultant_board )
     if_given_player_won(-1,resultant_board )
```

```
[-1 -1  0  0  1  0  0  0  1]
['O', 'O', '-', '-', 'X', '-', '-', '-', 'X']
After playing the MCTS agent
-------
|O|O|X|
|X|X|X|
|O|O|X|
-------

Player X won
Player O lost
```

We see the MCTS Agent Wins even though in the Starting Position it was one step away from Losing.

CASE 3 of 3(b): The board configuration where the opponent (Agent O) has made the first move and has occupied the centre square

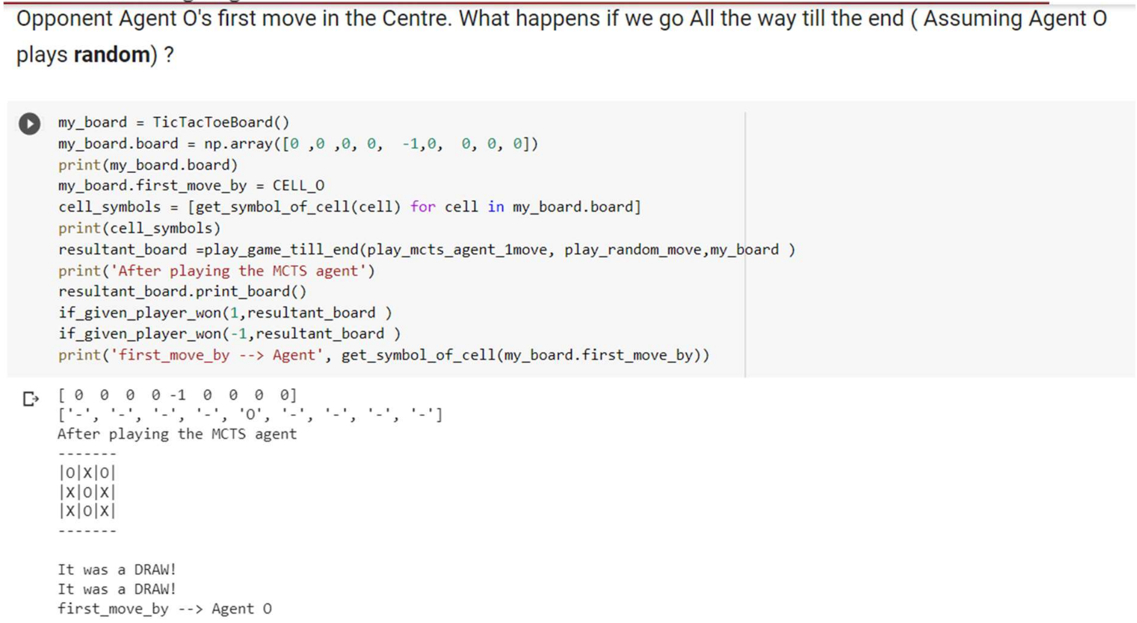


Opponent Made the First Move & has the central postion

The Next step by Agent

```
my_board = TicTacToeBoard()
my_board.board = np.array([0 ,0 ,0, 0,  -1,0,  0, 0, 0])
print(my_board.board)
#my_board.give_turns_to_alternate_players(first_move_by = CELL_O )
my_board.first_move_by = CELL_O
cell_symbols = [get_symbol_of_cell(cell) for cell in my_board.board]
print(cell_symbols)
resultant_board =play_mcts_agent_1move(my_board, node_cache=nodecache)
print('After playing the MCTS agent')
resultant_board.print_board()
if_given_player_won(1,resultant_board )
if_given_player_won(-1,resultant_board )
print('first_move_by -->', get_symbol_of_cell(my_board.first_move_by))
```

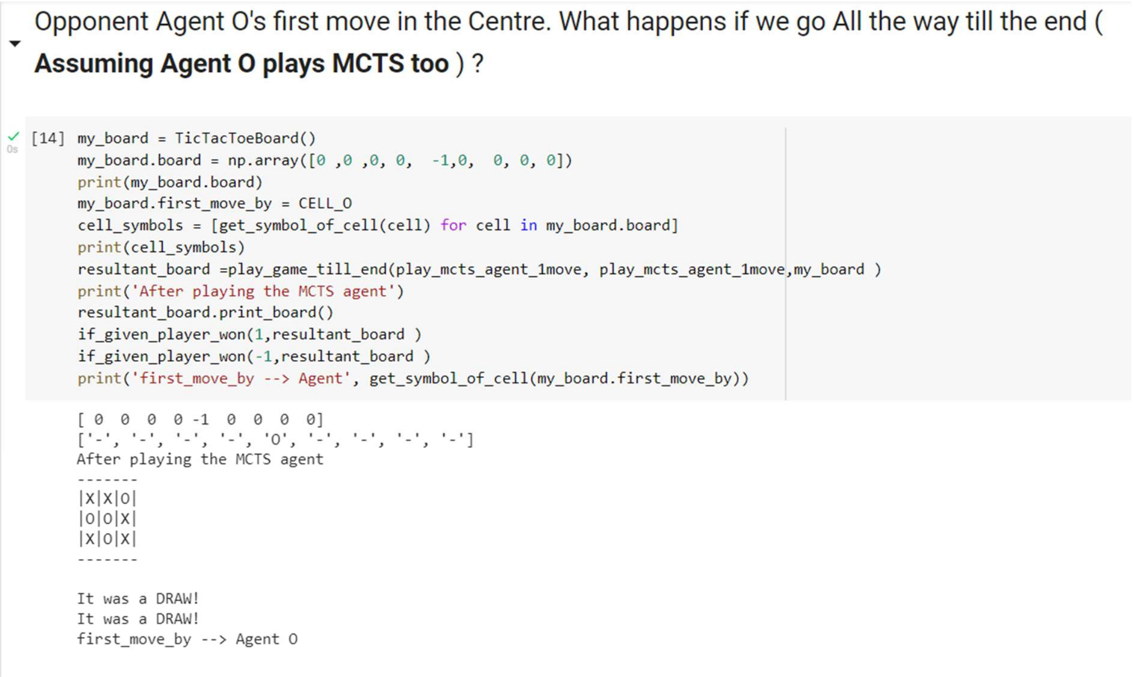
```
[ 0  0  0  0 -1  0  0  0  0]
['-', '-', '-', '-', 'O', '-', '-', '-', '-']
After playing the MCTS agent
-------
|-|X|-|
|-|O|-|
|-|-|-|
-------

Game ongoing
```



Opponent Agent O's first move in the Centre. What happens if we go All the way till the end (Assuming Agent O plays **random**) ?

```
my_board = TicTacToeBoard()
my_board.board = np.array([0 ,0 ,0, 0,  -1,0,  0, 0, 0])
print(my_board.board)
my_board.first_move_by = CELL_O
cell_symbols = [get_symbol_of_cell(cell) for cell in my_board.board]
print(cell_symbols)
resultant_board =play_game_till_end(play_mcts_agent_1move, play_random_move,my_board )
print('After playing the MCTS agent')
resultant_board.print_board()
if_given_player_won(1,resultant_board )
if_given_player_won(-1,resultant_board )
print('first_move_by --> Agent', get_symbol_of_cell(my_board.first_move_by))
```

```
[ 0  0  0  0 -1  0  0  0  0]
['-', '-', '-', '-', 'O', '-', '-', '-', '-']
After playing the MCTS agent
-------
|O|X|O|
|X|O|X|
|X|O|X|
-------

It was a DRAW!
It was a DRAW!
first_move_by --> Agent O
```

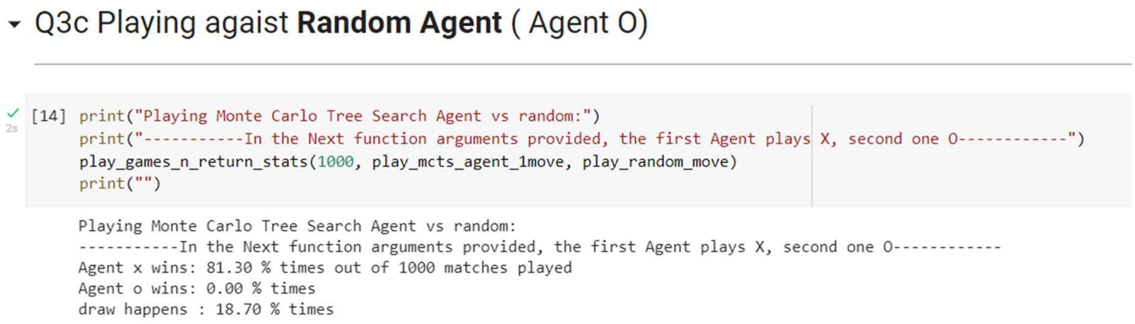


Opponent Agent O's first move in the Centre. What happens if we go All the way till the end (**Assuming Agent O plays MCTS too**) ?

```
my_board = TicTacToeBoard()
my_board.board = np.array([0 ,0 ,0, 0,  -1,0,  0, 0, 0])
print(my_board.board)
my_board.first_move_by = CELL_O
cell_symbols = [get_symbol_of_cell(cell) for cell in my_board.board]
print(cell_symbols)
resultant_board =play_game_till_end(play_mcts_agent_1move, play_mcts_agent_1move,my_board )
print('After playing the MCTS agent')
resultant_board.print_board()
if_given_player_won(1,resultant_board )
if_given_player_won(-1,resultant_board )
print('first_move_by --> Agent', get_symbol_of_cell(my_board.first_move_by))
```

```
[ 0  0  0  0 -1  0  0  0  0]
['-', '-', '-', '-', 'O', '-', '-', '-', '-']
After playing the MCTS agent
-------
|X|X|O|
|O|O|X|
|X|O|X|
-------

It was a DRAW!
It was a DRAW!
first_move_by --> Agent O
```

Q3 C) Record the number of wins, loss and draws by letting the MCTS agent play 1000 games against random and safe agents of Assignment 2

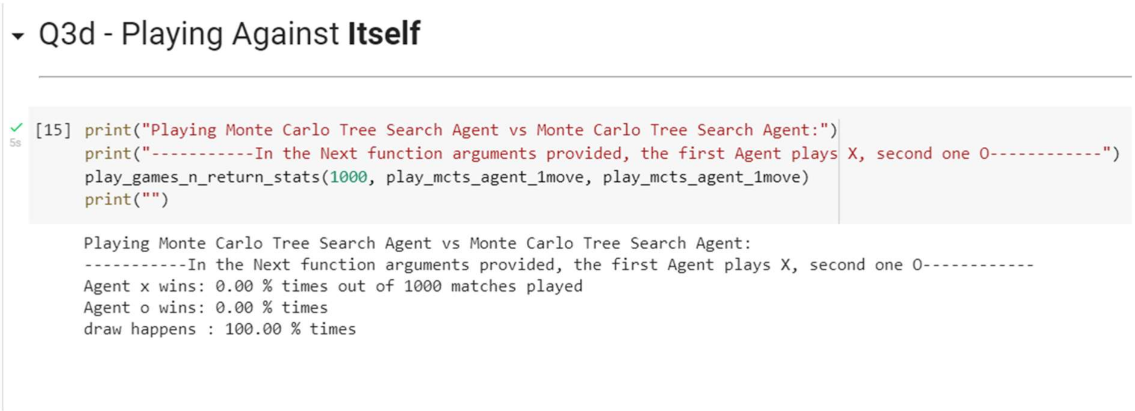


Q3c Playing agaist **Random Agent** (Agent O)

```
print("Playing Monte Carlo Tree Search Agent vs random:")
print("-----------In the Next function arguments provided, the first Agent plays X, second one O------------")
play_games_n_return_stats(1000, play_mcts_agent_1move, play_random_move)
print("")
```

```
Playing Monte Carlo Tree Search Agent vs random:
-----------In the Next function arguments provided, the first Agent plays X, second one O------------
Agent x wins: 81.30 % times out of 1000 matches played
Agent o wins: 0.00 % times
draw happens : 18.70 % times
```

Q3 D) Record of the number of wins, loss and draws by letting the MCTS agent play 1000 games against itself



Q3d - Playing Against **Itself**

```
print("Playing Monte Carlo Tree Search Agent vs Monte Carlo Tree Search Agent:")
print("-----------In the Next function arguments provided, the first Agent plays X, second one O------------")
play_games_n_return_stats(1000, play_mcts_agent_1move, play_mcts_agent_1move)
print("")
```

```
Playing Monte Carlo Tree Search Agent vs Monte Carlo Tree Search Agent:
-----------In the Next function arguments provided, the first Agent plays X, second one O------------
Agent x wins: 0.00 % times out of 1000 matches played
Agent o wins: 0.00 % times
draw happens : 100.00 % times
```