USMAN INSTITUTE OF TECHNOLOGY

Department of Computer Science
CS321 Artificial Intelligence

Lab# 03
Intelligent Agent in Python

Objective:

This experiment demonstrates implementation of an intelligent agent using Python. It guides students through the working of a two player tic tac toe agent. Such techniques can be used to implement similar agent programs.

Name of Student: _____

Roll No:_____ **Sec.**_____

Date of Experiment: _____

Marks Obtained/Remarks: _____

Signature: _____

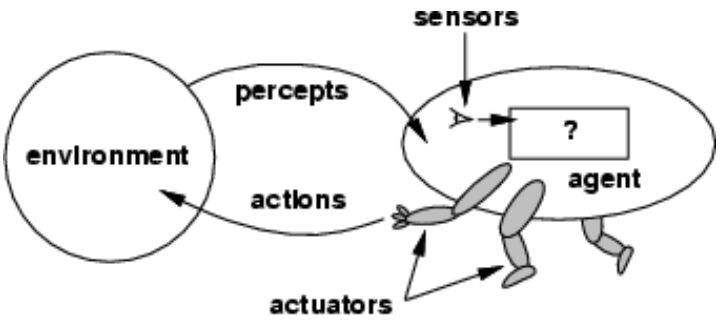
Lab 03: Intelligent Agent in Python

Agents

An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators

Human agent: eyes, ears, and other organs for sensors; hands, legs, mouth, and other body parts for actuators

Robotic agent: cameras and infrared range finders for sensors; various motors for actuators



Agent Types

Four basic types in order of increasing generality:

- Simple reflex agents
- Model-based reflex agents
- Goal-based agents
- Utility-based agents

Vacuum-Cleaner Agent

Percepts: location and contents, e.g., [A, Dirty]

Actions: Left, Right, Suck, No Op

function Reflex-Vacuum-Agent([location,status]) returns an action

if status = Dirty then return Clean

else if location = A then return Right

else if location = B then return Left

Multi player Tic Tac Toe Agent

Two human players, play Tic Tac Toe with paper and pencil. One player is „X" and the other player is „O". Players take turns placing their „X" or „O". If a player gets three of their marks on the board in a row, column or one of the two diagonals, they win. When the board fills up with neither player winning, the game ends in a draw.

Let"s get started by looking at a sample run of the program. The game is played between a human player and computer agent. The player makes their move by entering the number representing the desired location. Mapping of the location is shown in figure 02.

Sample Run of Tic Tac Toe is shown in figure 01.

```
Welcome to a game of Tic Tac Toe!
The computer will go first.
   |   |
 O |   |
   |   |
-----------
   |   |
   |   |
   |   |
-----------
   |   |
   |   |
   |   |
What is your next move? (1-9)
3
   |   |
 O |   | O
   |   |
-----------
   |   |
   |   |
   |   |
-----------
   |   |
   |   | X
   |   |
```

Figure 1 –a Sample Run

```
What is your next move? (1-9)
8
   |   |
 O | X | O
   |   |
-----------
   |   |
   |   |
   |   |
-----------
   |   |
 O |   | X
   |   |
What is your next move? (1-9)
5
   |   |
 O | X | O
   |   |
-----------
   |   |
 O | X |
   |   |
-----------
   |   |
 O |   | X
   |   |
The computer has beaten you! You lose.
```

Figure 1–b Sample Run

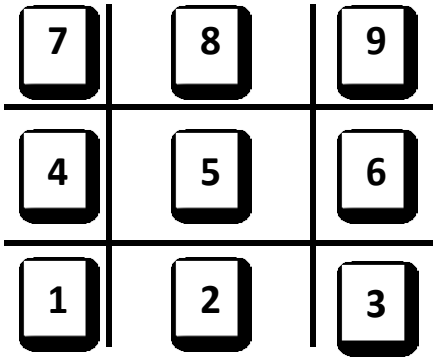


Figure 2 Location Mapping

Flow of The Agent Program

Figure 3 displays the flow chart of Tic Tac Toe program. The computer agent goes first and uses the symbol „O".

The boxes on the left side of the flow chart are what happens during the player"s turn. The right side shows what happens on the computer's turn. After the player or computer makes a move, the program checks if they won or caused a tie, and then the game switches turns. After the game is over, the program asks the player if they want to play again.

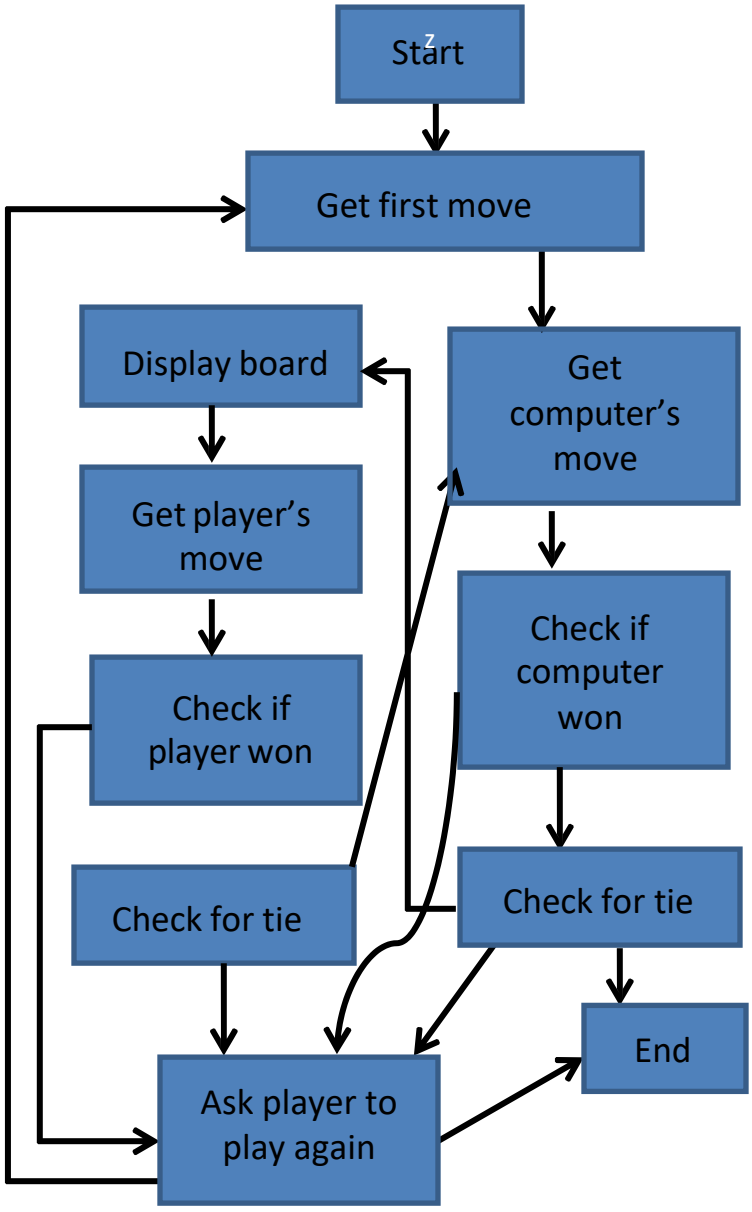


Figure 3 Flow Chart of Tic Tac Toe

The program can be further modified to a less rigid approach to let the human player choose if they want to be „X" or „O". Who takes the first turn is randomly chosen. Then the player and computer take turns making moves.

Representing the Board

First, you must figure out how to represent the board as a variable. On paper, the Tic Tac Toe board is drawn as a pair of horizontal lines and a pair of vertical lines, with either an X, O, or empty space in each of the nine spaces.

In the program, the Tic Tac Toe board is represented as a list of strings. Each string will represent one of the nine spaces on the board. The strings will either be 'X' for the X player, 'O' for the O player, or a single space ' ' for a blank space.

So if a list with ten strings was stored in a variable named board, then board[7] would be the top-left space on the board. board[5] would be the center. board[4] would be the left side space, and so on. The program will ignore the string at index 0 in the list. The player will enter a number from 1 to 9 to tell the game which space they want to move on.

Game AI

The AI needs to be able to look at a board and decide which types of spaces it will move on. To be clear, we will label three types of spaces on the Tic Tac Toe board: corners, sides, and the center. Figure 4 is a chart of what each space is.

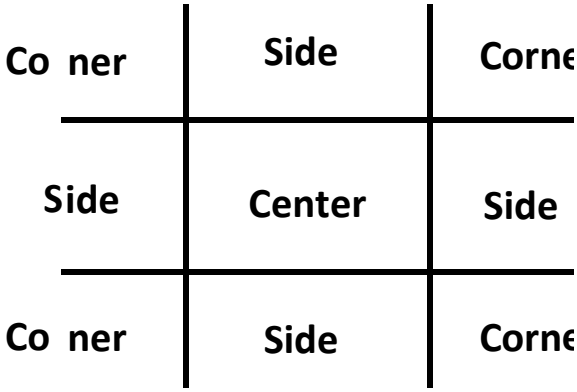


Figure 4 Space Types

The AI‟s algorithm will have the following steps:

1. First, see if there‟s a move the computer can make that will win the game. If there is, take that move. Otherwise, go to step 2.
2. See if there‟s a move the player can make that will cause the computer to lose the game. If there is, move there to block the player. Otherwise, go to step 3.
3. Check if any of the corner spaces (spaces 1, 3, 7, or 9) are free. If so, move there. If no corner piece is free, then go to step 4.
4. Check if the center is free. If so, move there. If it isn‟t, then go to step 5.
5. Move on any of the side pieces (spaces 2, 4, 6, or 8). There are no more steps, because if the execution reaches step 5 the side spaces are the only spaces left.

This all takes place in the “Get computer's move” function shown in the following code:

In [1]:
```
import random
def drawBoard(board):
    # This function prints out the board that it was passed.

    # "board" is a list of 10 strings representing the board (ignore index 0)
    print('   |   |')
    print(' ' + board[7] + ' | ' + board[8] + ' | ' + board[9])
    print('   |   |')
    print('-----------')
    print('   |   |')
    print(' ' + board[4] + ' | ' + board[5] + ' | ' + board[6])
    print('   |   |')
    print('-----------')
    print('   |   |')
    print(' ' + board[1] + ' | ' + board[2] + ' | ' + board[3])
    print('   |   |')

def inputPlayerLetter():
    # Lets the player type which letter they want to be.
    # Returns a list with the player's letter as the first item, and the computer's letter as the second.
    # For simplification, keeping X as the player's letter and 0 as the computer's letter
    return ['X', 'O']

def whoGoesFirst():
    # for simplification letteing the computer go first
    return 'computer'

def playAgain():
    # This function returns True if the player wants to play again, otherwise it returns False.
    print('Do you want to play again? (yes or no)')
    return input().lower().startswith('y')

def makeMove(board, letter, move):
    # This function simply marks the planned move (location of the borad) with the player's letter.
    board[move] = letter
```

```
def isWinner(bo, le):
    # Given a board and a player's letter, this function returns True if that player has won.
    # We use bo instead of board and le instead of letter so we don't have to type as much.
    return ((bo[7] == le and bo[8] == le and bo[9] == le) or # across the top
    (bo[4] == le and bo[5] == le and bo[6] == le) or # across the middle
    (bo[1] == le and bo[2] == le and bo[3] == le) or # across the bottom
    (bo[7] == le and bo[4] == le and bo[1] == le) or # down the left side
    (bo[8] == le and bo[5] == le and bo[2] == le) or # down the middle
    (bo[9] == le and bo[6] == le and bo[3] == le) or # down the right side
    (bo[7] == le and bo[5] == le and bo[3] == le) or # diagonal
    (bo[9] == le and bo[5] == le and bo[1] == le)) # diagonal
```

```
def getBoardCopy(board):
    # Make a duplicate of the board list and return it the duplicate.
    dupeBoard = []

    for i in board:
        dupeBoard.append(i)

    return dupeBoard

def isSpaceFree(board, move):
    # Return true if the passed move is free on the passed board.
    return board[move] == ' '

def getPlayerMove(board):
    # Let the player type in his move.
    move = ' '
    while move not in '1 2 3 4 5 6 7 8 9'.split() or not isSpaceFree(board, int(move)):
        print('What is your next move? (1-9)')
        move = input()
    return int(move)
```

```
def chooseRandomMoveFromList(board, movesList):
    # Returns a valid move from the passed list on the passed board.
    # Returns None if there is no valid move.
    possibleMoves = []
    for i in movesList:
        if isSpaceFree(board, i):
            possibleMoves.append(i)

    if len(possibleMoves) != 0:
        return random.choice(possibleMoves)
    else:
        return None

def getComputerMove(board, computerLetter):
    # Given a board and the computer's letter, determine where to move and return that move.
    if computerLetter == 'X':
        playerLetter = 'O'
    else:
        playerLetter = 'X'

    # Here is our algorithm for our Tic Tac Toe AI:
    # First, check if we can win in the next move
    for i in range(1, 10):
        copy = getBoardCopy(board)
        if isSpaceFree(copy, i):
            makeMove(copy, computerLetter, i)
            if isWinner(copy, computerLetter):
                return i
```

```
    # Check if the player could win on his next move, and block them.
    for i in range(1, 10):
        copy = getBoardCopy(board)
        if isSpaceFree(copy, i):
            makeMove(copy, playerLetter, i)
            if isWinner(copy, playerLetter):
                return i

    # Try to take one of the corners, if they are free.
    move = chooseRandomMoveFromList(board, [1, 3, 7, 9])
    if move != None:
        return move

    # Try to take the center, if it is free.
    if isSpaceFree(board, 5):
        return 5

    # Move on one of the sides.
    return chooseRandomMoveFromList(board, [2, 4, 6, 8])

def isBoardFull(board):
    # Return True if every space on the board has been taken. Otherwise return False.
    for i in range(1, 10):
        if isSpaceFree(board, i):
            return False
    return True
```

In [4]:
```
print('Welcome to a game of Tic Tac Toe!')

while True:
    # Reset the board
    theBoard = [' '] * 10
    playerLetter, computerLetter = inputPlayerLetter()
    turn = whoGoesFirst()
    print('The ' + turn + ' will go first.')
    gameIsPlaying = True
```

```
    while gameIsPlaying:
        if turn == 'player':
            # Player's turn.
            drawBoard(theBoard)
            move = getPlayerMove(theBoard)
            makeMove(theBoard, playerLetter, move)

            if isWinner(theBoard, playerLetter):
                drawBoard(theBoard)
                print('Hooray! You have won the game!')
                gameIsPlaying = False
            else:
                if isBoardFull(theBoard):
                    drawBoard(theBoard)
                    print('The game is a tie!')
                    break
                else:
                    turn = 'computer'

        else:
            # Computer's turn.
            move = getComputerMove(theBoard, computerLetter)
            makeMove(theBoard, computerLetter, move)

            if isWinner(theBoard, computerLetter):
                drawBoard(theBoard)
                print('The computer has beaten you! You lose.')
                gameIsPlaying = False
            else:
                if isBoardFull(theBoard):
                    drawBoard(theBoard)
                    print('The game is a tie!')
                    break
                else:
                    turn = 'player'

    if not playAgain():
        break
```

Student Exercise

Task 1

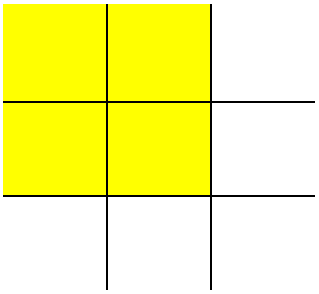
Consider the given Tic Tac Toe program designed for a match between Human and Agent. Convert it in to a program that demonstrates a play between Agent vs Agent, using two approaches:

i. Simple Reflex Agent:
   a. Am I winning the game?
   b. Am I losing the game?
   c. Go for a random move.

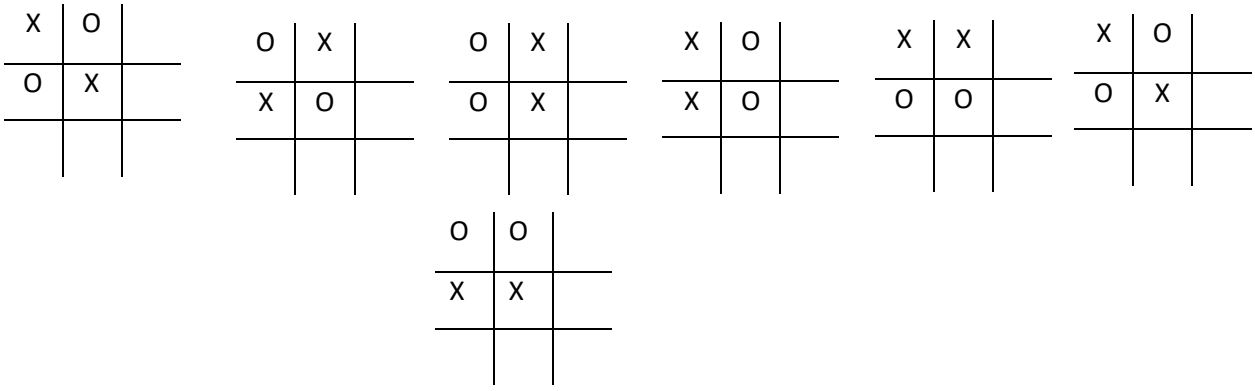
ii. Lookup Table:

To reduce the number of possibilities your lookup table should hold, start by identifying 4 boxes that will always be filled at the start of you game. Then plan for the remaining game accordingly. For example:

Your game always starts with any four boxes (of your choice) already filled in any specific order, again of your choice.



Now develop the reflex and lookup agents that can take the game forward from all possible configurations of these 4 boxes. In the sequence you are always playing 'X' and its always your turn next.



Task 2
Alter the agents that you have written so that they can handle the scenario when the computer goes first or the player/agent goes first.

Task 3
Alter the agents you have written so that they can handle all the combinations that may exist for the 4 cells you have selected.