University of Colorado, Boulder

Electrical & Computer Engineering 2703

Discrete Mathematics

Python AI and Tic Tac Toe

April 29, 2019

by Connor Humiston

Introduction

Whether drawn with chalk on the playground cement, with crayons on the kid's menu before dinner, or waiting just about anywhere with a piece of scratch paper, many of us might remember playing the childhood game of Tic Tac Toe. To recap the basic rules, Tic Tac Toe is a strategy game for two players, X and O, who take turns marking the spaces in a 3×3 grid. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal line wins the game. If neither player has a line of letters when the grid is filled, the game is a draw. The purpose of this project is to utilize Python to design an interactive game of Tic Tac Toe where a user plays against a simple Artificial Intelligence (AI) that can skillfully respond to the other player's moves.

A structural outline for how the game works is displayed in Figure 1 below. Inputs to the virtual Tic Tac Toe board follow in Figure 2.

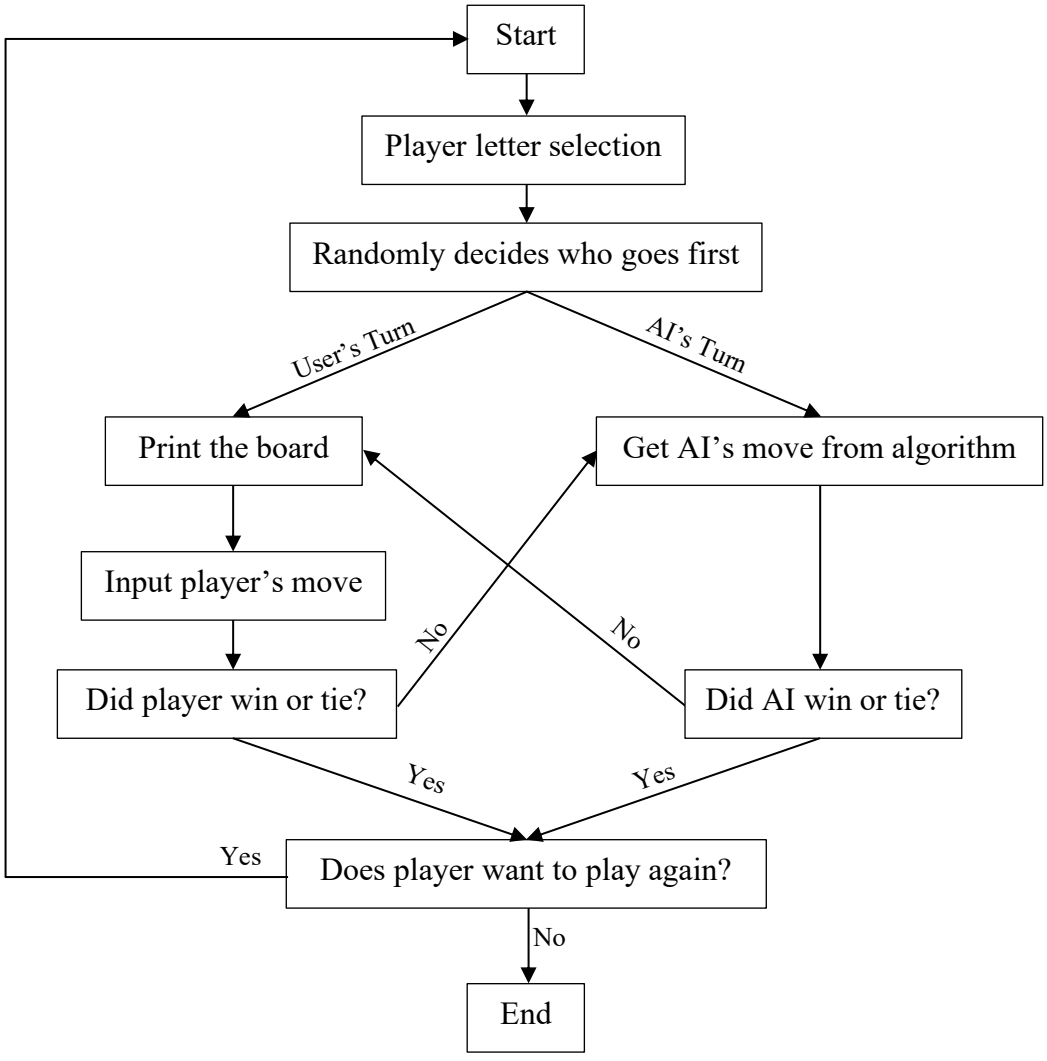

Figure 1: Flow chart for Tic Tac Toe program

7	8	9
4	5	6
1	2	3

Figure 2: Numbered board spaces corresponding to player inputs

Code Analysis

The Python code in its entirety can be found in Appendix A.

How the Game Works

The interactive Tic Tac Toe game starts by printing a welcome message to the user and asks if they would like to be X or O. The user can input any character, but the `playerLetterInput()` function will continue asking for an X or an O until a lower or uppercase X or O has been entered. Once the user has chosen a letter, `firstPlay()` utilizes the random module to randomly decide which player goes first. This decision is displayed and the board is printed from the `drawBoard()` function using a series of strings and the list of board positions that it is passed. If the player goes first, the grid is printed and the player is asked which space they would like to mark. The program first checks to see if that move has won the game for the user by passing the grid with updated marks and the player letter to the `isWinner()` function that checks all eight possibilities for winning on a 3×3 grid (down the middle, across the middle, and along the diagonals, bottom, top, left, and right). If the player has won, that is printed and the `playAgain()` function checks to see if the player would like another match. If the player has not won and the board is deemed full by `isBoardFull()` that loops through all nine spaces to check if a space is empty, then the game must be a tie. The tie is then printed for the user who is asked for another match in the following line. Otherwise, if the game is neither a win nor a draw on that play, then the turn is passed to the AI that follows the same steps as the player after choosing its move. The AI's algorithm for choosing the best move is described in the following section.

The AI Algorithm

The AI's move is determined in the `getAIMove()` function that is passed the current board and whichever letter the AI has been assigned. The algorithm begins by checking if it can win in the next move. A copy of the current grid is constructed in a for loop that loops through all the free spaces of the copy. Looping through all the empty spaces one by one, the AI exploits the `makeMove()` function to place its letter in the corresponding spot on the grid copy. The

isWinner() function is passed the current space index and tests if the AI would win if it were to make that move. If that play is deemed a winning move, then the index of that spot is returned and becomes the AI's play.

If the AI cannot win in the next move, then it checks to see if the user can be blocked. Again looping through all the possible empty spaces to block the other player in a copy of the grid, the isWinner() function is used, but passed the other player's letter to simulate a user win. If the user could win in that spot the next round, then that space becomes the AI's next move. Hence, the only way to beat the AI is if there is more than one way for the user to get three marks in a row since the AI can only play one block at a time.

Finally, if the AI can neither win nor block the other player, its move becomes one of three options: a corner, the center, or a side. First, a random corner is chosen by the chooseRandomMove()function that creates a copy of the current board and appends all the empty spaces into a list called possibleMoves. It then returns a random choice from the list if the list is not empty and None if it is empty. If no corners are free, then the center (square 5) is returned for the AI's next move. Lastly, if no other spots are open, then one of the sides is randomly chosen to return for the AI's next move. The order of these selections was chosen with best strategy in mind.

Conclusion

The purpose of this project was to utilize Python to design an interactive game of Tic Tac Toe where a user plays against a simple AI that can logically respond to the other player's moves. This goal was successfully accomplished as described above with a gameplay program that takes in user inputs, prints a grid with correct marks on it, exploits an AI algorithm, and determines the game's outcome. The AI algorithm was relatively simple to conceive for a 3×3 board due to the fact that there were only nine spaces on the board and eight lines for which either player could win along. Avid players of Tic Tac Toe, however, have long discovered that the best plays from both parties ultimately leads to a draw; one reason the game is most often played by youth. With this in mind, this project could be expanded in a future project from a 3×3 grid to an $n \times n$ grid, or even a 3D model, to make the game more difficult for both parties. Such an expansion would require a significantly more complex AI, but presents an exciting possibility, nonetheless.

Appendix A

The Tic Tac Toe Python Code in its entirety

```
# Connor Humiston
# Discrete Math
# AI and Tic Tac Toe

import random

def drawBoard(board):
    # This function prints the board
    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 playerLetterInput():
    # this function allows the player to type which letter they want
    letter = ''
    while not (letter == 'X' or letter == 'O'):
        print('Would you like to be X or O?')
        letter = input().upper()

    # the first element in the tuple is the player's letter, and the second is the AI's
    if letter == 'X':
        return ['X', 'O']
    else:
        return ['O', 'X']
    # returns a list with the player's letter first and the AI's letter second

def firstPlay():
    # Randomly chooses which player goes first
    if random.randint(0, 1) == 0:
        return 'AI'
    else:
        return 'player'

def playAgain():
    print('Would you like to play again? (yes or no)')
```

```
    return input().lower().startswith('y')
    # This function returns True if the player input starts with y (aka yes) and False otherwise

def makeMove(board, letter, move):
    # places letter on the board
    board[move] = letter

def isWinner(board, letter):
    # passed the current board and a player's letter, this function returns True if that player has
won
    return ((board[7] == letter and board[8] == letter and board[9] == letter) or # a win across the
top (all top filled with letter)
    (board[4] == letter and board[5] == letter and board[6] == letter) or # a win across the middle
    (board[1] == letter and board[2] == letter and board[3] == letter) or # a win across the bottom
    (board[7] == letter and board[4] == letter and board[1] == letter) or # a win down the left side
    (board[8] == letter and board[5] == letter and board[2] == letter) or # a win down the middle
    (board[9] == letter and board[6] == letter and board[3] == letter) or # a win down the right
side
    (board[7] == letter and board[5] == letter and board[3] == letter) or # a win on the first
diagonal
    (board[9] == letter and board[5] == letter and board[1] == letter)) # a win on the second
diagonal

def getBoardCopy(board):
    # creates a duplicate of the board list and returns it
    dupeBoard = []
    for i in board:
        dupeBoard.append(i)
    return dupeBoard

def isSpaceFree(board, move):
    return board[move] == ' '
    # returns true if the passed move is free, aka an empty space, on the board that is passed

def getPlayerMove(board):
    # allows the player to type in their move
    move = ' '
    while move not in '1 2 3 4 5 6 7 8 9'.split() or not isSpaceFree(board, int(move)):
        # if the move is not in the given spaces or not free the selection line is reprinted and any
proper input is returned
        print('What is your next move? (1-9)')
        move = input()
    return int(move)

def chooseRandomMove(board, movesList):
    # returns a valid move from the passed move list on the board or none if no valid move
```

```
    possibleMoves = []
    for i in movesList:
        if isSpaceFree(board, i):
            possibleMoves.append(i) # appends all the free spaces
    if len(possibleMoves) != 0: # if the number of possible moves is not zero
        return random.choice(possibleMoves) #randomly chooses one
    else:
        return None

def getAIMove(board, AILetter):
    # with a board and the AI's letter, determines best move and returns that move
    if AILetter == 'X':
        playerLetter = 'O'
    else:
        playerLetter = 'X'

    # The AI algorithm:
    # first, checks if AI can win in the next move
    for i in range(1, 10):
        copy = getBoardCopy(board)
        if isSpaceFree(copy, i): # if a space is free on the board
            makeMove(copy, AILetter, i) # letter placed in that spot
            if isWinner(copy, AILetter): # if the AI is theoretically the winner on board copy,
                return i                # then it returns that spot to play there

    # checks if the player could win on his next move to block them
    for i in range(1, 10): # looping through the possible spots
        copy = getBoardCopy(board)
        if isSpaceFree(copy, i): # if the space is free
            makeMove(copy, playerLetter, i) # and letter in that spot
            if isWinner(copy, playerLetter): # if the player could win there,
                return i                    # then it returns that spot to block the opponent

    # otherwise the AI tries to take one of the corners first if they are free
    move = chooseRandomMove(board, [1, 3, 7, 9])
    if move != None:
        return move

    # then tries to take the center if it's free
    if isSpaceFree(board, 5):
        return 5

    # lastly, if no other spots open, makes move on one of the sides
    return chooseRandomMove(board, [2, 4, 6, 8])

def isBoardFull(board):
```

```
    # returns True if every space on the board has been taken and False otherwise
    for i in range(1, 10):
        if isSpaceFree(board, i):
            return False
    return True


print('Welcome to Tic Tac Toe!')

while True:
    # reseting the board
    theBoard = [' '] * 10
    playerLetter, AILetter = playerLetterInput() # player letter assigned to input
    turn = firstPlay() # the first turn is assigned to random variable
    print('The ' + turn + ' will go first.')
    gameIsPlaying = True # let the games begin

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

            if isWinner(theBoard, playerLetter): # first if the player is a winner after move made,
prints and ends game
                drawBoard(theBoard)
                print('You win!')
                gameIsPlaying = False
            else:
                if isBoardFull(theBoard): # if the board is full and not already a winner then it's a tie
                    drawBoard(theBoard)
                    print('The game is a tie.')
                    break
                else:
                    turn = 'AI' # otherwise it's the AI's turn

        else:
            # For AI's turn
            move = getAIMove(theBoard, AILetter)
            makeMove(theBoard, AILetter, move)

            if isWinner(theBoard, AILetter): # if the AI is the winner, prints and ends game
                drawBoard(theBoard)
                print('The AI has beaten you. You lose.')
                gameIsPlaying = False
```

```
        else:
            if isBoardFull(theBoard): # if not a winner and board full, then a tie
                drawBoard(theBoard)
                print('The game is a tie.')
                break
            else:
                turn = 'player' # otherwise back to the player

    if not playAgain(): # if play again is false then breaks
        break
```