Project 1

Cap 4630

Dr. Marques

Andrew Donate

Tic-Tac-Toe: Baseline Solution (Minmax)

Prologue

The following document describes of Project 1 for the Intro to Artificial Intelligence course (CAP 4630). This project was done entirely by Andrew Donate as the “architect”, “developer”, and “reporter”. This project does take inspiration from another project but is sourced down below in the references section.

Introduction

The purpose of this python program was to produce a functional AI that can compete against other AI of its complexity, random movement AI, and human players. The game allows a human to play against the AI on a 3 x 3 grid and place either X’s or O’s to compete in Tic-Tac-Toe. The program follows the rules of Tic-Tac-Toe by tracking each players moves and determines when the game has a winner or ends in a draw.

Functions

Following along with the python code top to bottom it consists of the following functions and their uses:

1. printGrid: Prints the current board with the filled in slots of player moves and if given the input of true it will print out the selection board which consists of a grid 0 to 8 to show the human player where to put their marker (X or O.)
2. playerCurrentMove: Checks during the current players move if the selected slot they chose is empty or filled, based on that it will either set a marker or have the user redo

selection. During the marker setting it also checks if during the current players turn if they have won the game.

3. foundWinner: during the playerCurrentMove winning check this function is called to return a true or false depending on the Tic-Tac-Toe game rules. The winning conditions are if all markers are lined up horizontally, vertically, or diagonally. The latter is checked with the method diagonalWinCheck.
4. diagonalWinCheck: called in foundWinner and is used to check from top left of the grid to the bottom right and top right to the bottom left and sees if all the slots diagonally are the same marker.
5. doesGridHaveEmptySlots: checks if the game grid has any empty slots
6. numberOfEmptySlots: returns the number of empty slots on the grid
7. availibleSlots: returns the number of slots that are not taken on the grid
8. announceMoveAndSwitchPlayers: after the end of all players turn it announces who move to what slot on the grid, announces the winner of the game (if applicable) and switches whose turn it is once the previous players turn ends.
9. playGame: starts the game of Tic Tac Toe, executes printGrid, assigns what player goes first, and loops through the game checking if either the grid no longer has empty slots or if there is a winner announced. If the former, it then announces the game ends in a tie.
10. getMove: depending on which class is calling the function it will either place a marker where the human player chooses, randomly places a marker in available slots, or using the minMax function it simulates future moves and chooses the best one.
11. validMove: checks if the human player picks a valid slot or if the input is even in the range of 0 to 8.
12. minMax: Based on Kylie Ying's Tic Tac Toe AI, it assigns the AI player as the current maxPlayer, checks if the game has ended first as the function can be called recursively and assigns a score of positive or negative based on the winner, if the grid is filled by this point assigns a score of 0. After which if the current maxPlayer is the AI it assigns a score of negative infinity or positive infinity if the AI is not the maxPlayer.

13. simulateMoves: for moves left in the grid the function will check if the slot if taken and if not, it will simulate future moves and compare each with the minMax method by comparing scores. Once all moves are simulated the best one is picked and chosen for the AI current move.

Initial Implementation Issues

In the original code located at the end of the document issues emerged that caused an entire rewrite of the project code.

1. Complexity of the grid and slot selection: The initial implementation was complicated compared to the current one. The player markers and the grid used for the board were hard coded and the grid was created as a two-dimensional array which at the time was difficult to use to check whether a player won the game or not.
2. The Human input value checker was very bulky compared to the newer system as it involved a lot of if and else statements which could have led to more issues or cases that were not thought of.

Due to these issues, the initial implementation had to be redone or the project would not be able to progress much farther into the main gameplay of the program such as the smart and random AI.

Revised Implementation

1. Improved grid backend: although the grid is once again two-dimensional it was improved upon to which it would no longer be hard-coded and was able to shrink the two functions in the initial implementation selectionBoard and printBoard into printGrid simplifying the code.
2. Updated human move check: human input was improved upon by using try statements to prevent the program from exiting abruptly and to ask the user to retry their inputs. This led to more cases being covered which would lead to a more secure program.
3. Fixed winning check: the winning check was finally implemented fully in which the program was now able to check horizontally, vertically, and diagonally compared to the

former code where it was not possible due to the grid backend being more complex to use.

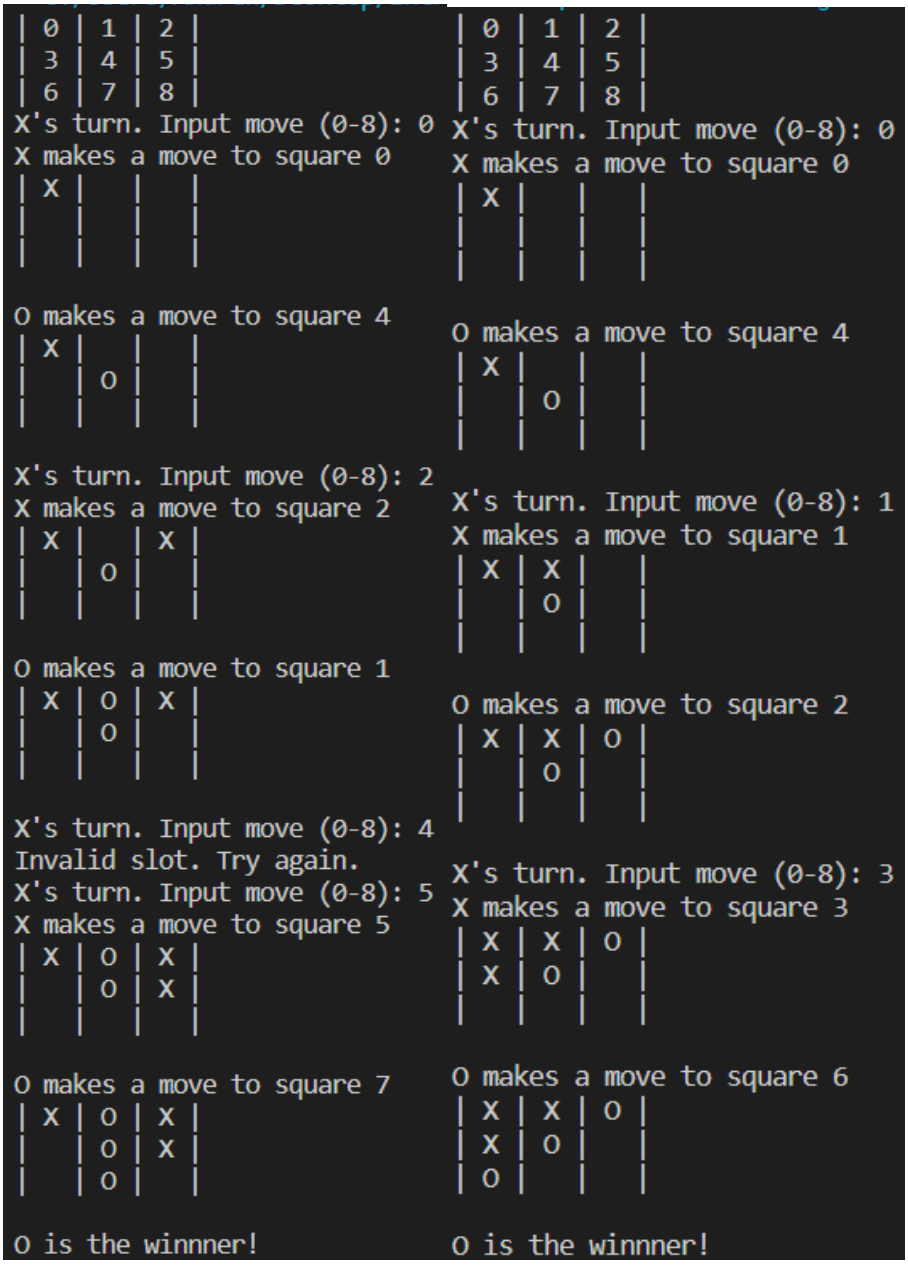
4. Inspiration by CodeAesthetic the project's code was sectioned off into more functions than the original as to make the code more readable and easier to follow along with.

Limitations and Future Improvements

Although the revised implementation does fix some of the issues in the initial, there are still limitations and improvements that could be done in the future.

1. Lack of a user interface pre-game: before the game of Tic Tac Toe even starts the human player is assigned as X and the selected O player is assigned with the AI player. This does not give the human a choice on whether they wanted to be the X or O player as well as not being able to choose against another human, a random selection AI, or the "intelligent" AI.
2. In the future if the project were to be expanded on it would be best to implement this program into a web-based solution as it would make the game more accessible to those who can not run the program natively. Another improvement would be to implement the previously mentioned pre-game user interface as for a program it is crucial to allow the user to have options in what they would like to do.

Demo


0	1	2
3	4	5
6	7	8
X's turn. Input move (0-8): 0
X makes a move to square 0
X		
O makes a move to square 4
X		
O		
X's turn. Input move (0-8): 2
X makes a move to square 2
X		X
O		
O makes a move to square 1
X	O	X
O		
X's turn. Input move (0-8): 4
Invalid slot. Try again.
X's turn. Input move (0-8): 5
X makes a move to square 5
X	O	X
O	X	
O makes a move to square 7
X	O	X
O	X	
O		
O is the winnner!
0	1	2
3	4	5
6	7	8
X's turn. Input move (0-8): 0
X makes a move to square 0
X		
O makes a move to square 4
X		
O		
X's turn. Input move (0-8): 1
X makes a move to square 1
X	X	
O		
O makes a move to square 2
X	X	O
O		
X's turn. Input move (0-8): 3
X makes a move to square 3
X	X	O
X	O	
O makes a move to square 6
X	X	O
X	O	
O		
O is the winnner!

Previous Code

```
import os

##Global variables
#If move is
#0: not used
#1: X
#2: O
usedMoves = [
    [0, 0, 0],
    [0, 0, 0],
    [0, 0, 0]
]
symbols = {
    0: ' ',
    1: 'X',
    2: 'O'
}

##Functions
#Print selection board
def selectionBoard():
    print("| 0 | 1 | 2 |")
    print("| 3 | 4 | 5 |")
    print("| 6 | 7 | 8 |")

#Print current board
def printBoard():
    for row in usedMoves:
        for value in row:
            if value in symbols:
                symbol = symbols[value]
            print("| " + symbol + "", end=' ')
        print("|")
```

```
    print()

#User selection
def userBoardSelection():
    while (True):
        try:
            userMove = int(input("X's turn. Input move (0-8): "))
            row = userMove // len(usedMoves[0])
            col = userMove % len(usedMoves[0])

            if 0 <= row < len(usedMoves) and 0 <= col < len(usedMoves[0]):
                if usedMoves[row][col] == 0:
                    print("X makes a move to square " + str(userMove))
                    usedMoves[row][col] = 1
                    printBoard()
                    break
                else:
                    print("Already taken.")
            else:
                print("Out of range!")

        except ValueError:
            print("Not a number!")

#Bot selection
def botBoardSelection():
    while (True):
        try:
            userMove = int(input("X's turn. Input move (0-8): "))
            row = userMove // len(usedMoves[0])
            col = userMove % len(usedMoves[0])

            if 0 <= row < len(usedMoves) and 0 <= col < len(usedMoves[0]):
                if usedMoves[row][col] == 0:
                    print("O makes a move to square " + str(userMove))
```

```
                    usedMoves[row][col] = 2
                    printBoard()
                    break
                else:
                    print("Already taken.")
            else:
                print("Out of range!")

        except ValueError:
            print("Not a number!")


#Main Game
def game():
    #TODO fix end game when board is filled
    while not all(value == 1 or value == 2 for row in usedMoves for value in
row):
        userBoardSelection()
        botBoardSelection()

    print("Game ended in a draw!")

#Row Check
def rowCheck():
    for row in usedMoves:
        if not all(value == 1 or value == 2 for value in row):
            return False
    return True
#Col Check
def colCheck():
    for col in range(len(usedMoves[0])):
        values = [row[col] for row in usedMoves]
        if all(value == 1 or value == 2 for value in values):
            continue
        else:
```

```
            return False
    return True

#Diag Check

#TODO fix winnerCheck and its sub functions
#Winning check
def winnerCheck():
    if rowCheck():
        print("Winner with rows!")
    if colCheck():
        print("Winner with cols!")



##Start of game

#Clear current screen (Just for testing)
os.system('cls')

#Short 3 line description of program
print("Welcome to Tic-Tac-Toe!\nThis Program was made for the class CAP4630 Intro to AI\nThe purpose of this program is to demonstrate a simple AI that can play the game Tic-Tac-Toe.\n")
selectionBoard()
game()
```

References

"Why You Shouldn't Nest Your Code." *YouTube*, YouTube, 6 Dec. 2022, https://www.youtube.com/watch?v=CFRhGnuXG-4. Accessed 3 June 2023.

Ying, Kylie. "Tic-Tac-Toe." *Github*, 27 Jan. 2022, https://github.com/kying18/tic-tac-toe. Accessed 3 June 2023.

Hurbans, R. "Grokking artificial intelligence algorithms" (pp. 74–83). Manning Publications. Accessed 3 June 2023.