

The workings behind AI in games

For Extended Project Qualification



March 4, 2021

Emmanuel Adio

Contents

[Workings behind AI in games 2](#_Toc65790766)

[Abstract 2](#_Toc65790767)

[Introduction 2](#_Toc65790768)

[Tic-Tac-Toe 3](#_Toc65790769)

[Introduction 3](#_Toc65790770)

[What is tic-tac-toe? 3](#_Toc65790771)

[Why did I pick tic-tac-toe? 3](#_Toc65790772)

[Tic-tac-toe: Player v player (Level 0) 3](#_Toc65790773)

[Plan 3](#_Toc65790774)

[Program analysis 4](#_Toc65790775)

[Play through. 8](#_Toc65790776)

[Tic-tac-toe Player vs Random AI (level1) 12](#_Toc65790777)

[Plan 12](#_Toc65790778)

[Program analysis 13](#_Toc65790779)

[Play throughs 19](#_Toc65790780)

[Tic-tac-toe: Player v Smart AI (level2) 22](#_Toc65790781)

[Plan 22](#_Toc65790782)

[Program analysis 24](#_Toc65790783)

[Play throughs of level 2/2.5 31](#_Toc65790784)

[Tic-tac-toe: Player v Minimax AI (level 4)) impossible (Theory) 36](#_Toc65790785)

[What is the minimax algorithm? 36](#_Toc65790786)

[Plan 36](#_Toc65790787)

[Program analysis 37](#_Toc65790788)

[Play throughs. 45](#_Toc65790789)

[Evaluation 48](#_Toc65790790)

[Testing 48](#_Toc65790791)

[What went well? 48](#_Toc65790792)

[References 49](#_Toc65790793)

# Workings behind AI in games

## Abstract

Artificial intelligence (AI), the study and development of machine learning, has been skyrocketing lately, in fact in the last decade AI has made significant progress and has implemented itself into every corners of today’s society and our day-to-day life, e.g., self-driving and parking vehicles, digital assistants etc. It has frequently been referred to as the future of technology as its potential is seen to be unlimited current AIs have already shown great understanding in Literature, music and maths, in some cases better than humans do. This idea of machine being completely self-sufficient and being able to take in the data that they receive from the outside world and then understand this information would have been unheard of even back in the day. The further development of AI will even help humans to understand the world around us better and even the universe.my aim with my project is to make a game using python to unravel the capabilities of AI in game development and how it can increase the enjoyment/ difficulty of games for the user. In addition, link this with a suitable levelling system that separates the complexity or the AI the player will be up against (higher level, more complex AI).

## Introduction

AI even play a significant role in video games having roles usually playing against or with the player in a roles known as NPC (non-player character), a character that interacts with the player as if they were a human themselves e.g. as villagers in a village assigning quests and rewards to the player to move the storyline along for the player, or as mobs, also non-player character that play as the opponent for the player usually following a set movement pattern, they even allow for the difficulty level of games to be controlled allowing for players with low skill cap to still be able to play the game and allows for them to increase their confidence as they slowly develop their skills as they increase the game difficulty and this all generally leads to the player enjoying the game more.

AI have even begun to make their own games a study performed by Matthew Gurdial and his team at the University of Alberta used AI help recreate the exact game they wanted to play (Balaganur, 2020); they did this by having the AI watch game footage of five levels of games like Super Mario, Kirby’s Adventure and Mega man. As it watched the gameplay, it made guesses of the rules of the game and started putting it all together. The result was high standard of there was some faults but overall, it showed the very high capability AI had in game development.

The aim with my project is to be able to display the programming behind AIs in computer video games. I want to experience what game developers have to go through when creating opponents and implementing the AI into the characters the player interacted with while playing the game. I would like to explore how the actions of the player can affect the behaviour of the AI and this is programmed into the game.

I am interested in exploring the link between the complexity of the game and the complexity of the AI and how that will link with the enjoyment of the game for the user, I want to answer the question behind how a levelling system for my game can be created by increasing or decreasing the intricacy of my AI.

# Tic-Tac-Toe

## Introduction

For my EPQ I have decide to make a game of tic-tac-toe to display the range that AI have in games.

### What is tic-tac-toe?

Tic-tac-toe is a pen and pencil game played between two players, X and O, which take turns to make marks on a 3X3 board. Rules of the game are that each player must pick either to be an X or an O and only mark the bored using that letter, they are to take turns marking the board once inside the squares, with the letter they have chosen, the player who chose X goes first. The winner of the game is decided by the first player to be able to get a row diagonally, horizontally, or vertically of the same letter on the board.

### Why did I pick tic-tac-toe?

There are many reasons why I picked tic-tac-toe to be the main feature in my project such as it is:

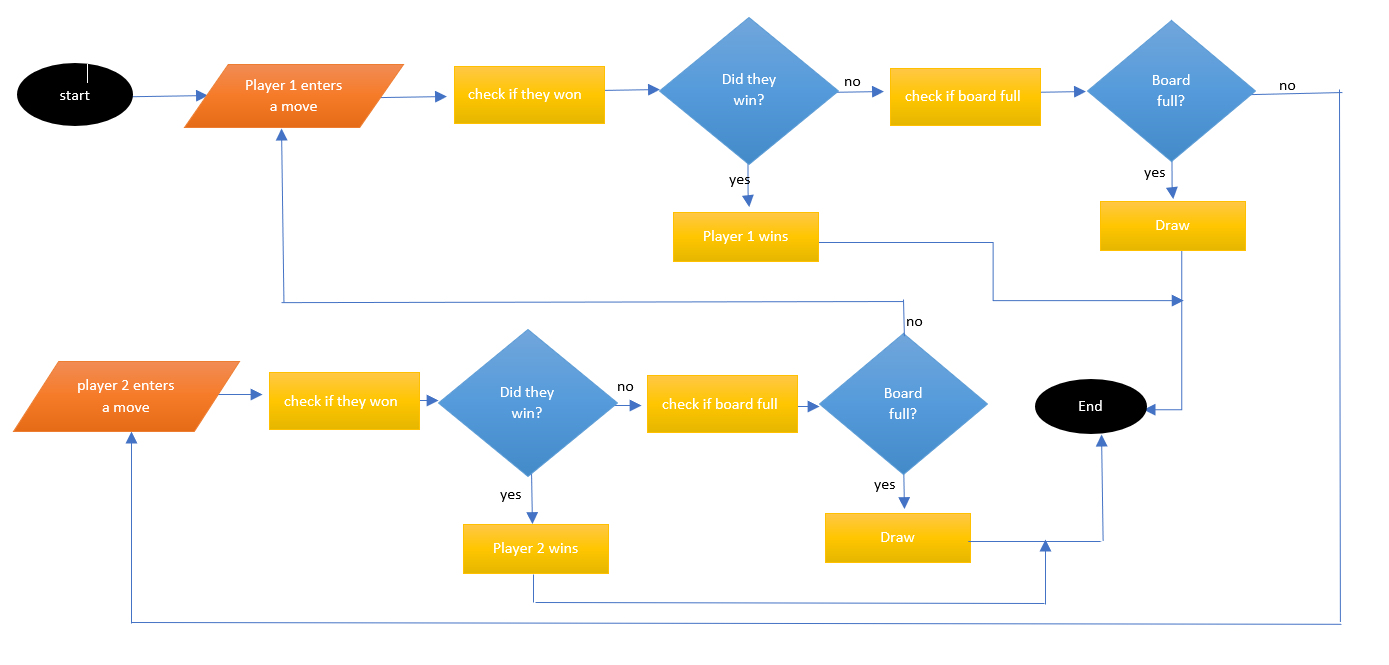
* A simple game, it is very easy for someone who has never played it before to now be able to pick it up and enjoy the game.
* Has a wide range of possible moves and endings this allows for me to be able to show the full capabilities of my AI.
* Quite challenging to program as there is surprisingly more functions needed than originally expected and I thought this would allow me to better develop my skills as a programmer.
* A fun game to play, I find playing tic-tac-toe to be quite enjoyable traditionally and I thought it would be nice to also play in my computer as well.

## Tic-tac-toe: Player v player (Level 0)

### Plan

#### Flowchart

*diagram made by me\**



#### Worded plan

1. Ask player 1 for the move the point on the board they would like to mark (between 1 and 9)
2. Check if player 1 has won the game.
3. If player won Has won
   1. Display victory message.
   2. Display final board and exit game.
4. If board is full
   1. Display that they drew.
   2. Display final board and exit.
5. Ask player 2 for the move the point on the board they would like to mark (between 1 and 9)
6. Check if player 2 has won the game.
7. If player one has won
   1. Display victory message.
   2. Display final board and exit game.
8. If board is full
   1. Display that they drew.
   2. Display final board and exit.
9. Repeat steps 1 to 8 until final board has been displayed or someone wins.

### Program analysis

*Program information in red.*

#### Code:

# simple tic Tac Toe game in python.

# make the board.

board = ["Ignore me!", "1", "2", "3", "4", "5", "6", "7", "8", "9"]

round = 0

#boolean variable to continue playing

playing = True

#player letter variable

player = ""

#board creation

def DisplayBoard(board):

print("+---" + "+--" + "-+" + "---+\n" +

"| " + board[7] + " | " + board[8] + " | " + board[9] + " |\n" +

"+---" + "+--" + "-+" + "---+\n" +

"| " + board[4] + " | " + board[5] + " | " + board[6] + " |\n" +

"+---" + "+--" + "-+" + "---+\n" +

"| " + board[1] + " | " + board[2] + " | " + board[3] + " |\n" +

"+---" + "+--" + "-+" + "---+\n")

#welcome message funtion that leads to the player movement funtion.

def welcome():

print("Welcome to TicTacToe!")

print("+---" + "+--" + "-+" + "---+\n" +

"| " + "7" + " | " + "8" + " | " + "9" + " |\n" +

"+---" + "+--" + "-+" + "---+\n" +

"| " + "4" + " | " + "5" + " | " + "6" + " |\n" +

"+---" + "+--" + "-+" + "---+\n" +

"| " + "1" + " | " + "2" + " | " + "3" + " |\n" +

"+---" + "+--" + "-+" + "---+\n")

print("player one will start as x")

global rounds

rounds = round + 1

player = "X"

PlayerMove()

#function to detecting if space on board it empty or taken

def Empty(move, player):

if board[move] == "X" or board[move] == "O":

return False

elif board[move] != "X" or board[move] != "O":

return True

#function to make player's move

def PlayerMove():

#loading the needed global variables needed in function:

global rounds

global board

global player

XorO = rounds % 2

#this prints out what round the game is currently on odd rounds would be X's turn whilst even rounds are O's turn

print("round", rounds)

#the selective statement for deciding which players move it is.

if XorO == 1:

#code behind X player's move:

player = "X"

move = int(input("X's move: enter a value between 1 and 9 for a point on the board:\n"))

#loop to keep player entering moves until valid move between 1 and 9 is made.

while (move < 1 or move > 9):

print("invalid move enter a value between 1 and 9")

move = int(input())

#funtion to check if players moved into a taken spot

Empty(move, player)

while Empty(move, player) == False:

move = int(input("SpaceTaken! reenter move:\n"))

Empty(move,player)

#placeing players move onto the board:

board[move] = player

rounds = rounds + 1

#funtion to check players win

check()

DisplayBoard(board)

else:

#code behind O players move:

#similar to X player's move function.

player = "O"

move = int(input("O's move: enter a value between 1 and 9 for a point on the board:\n"))

while (move < 1 or move > 9):

print("invalid move enter a value between 1 and 9")

move = int(input())

Empty(move, player)

while Empty(move, player) == False:

move = int(input("SpaceTaken! reenter move:\n"))

Empty(move,player)

board[move] = player

rounds = rounds + 1

check()

DisplayBoard(board)

#check if board is full function

def Full():

spaces = False#sets free paces to false

for i in range(1,9):

if board[i] != "X" and board[i] != "O":#used to go through the whole board

spaces = True #changes to true if there is space only if ther is a space

if spaces == False:

return True

else:

return False

#end of game check function

def check():

global playing

#checking if the borard is full

if Full():

print("it is a draw End of Game!! thanks for playing")

playing = False

# straight middle row

if board[4] == board[5] and board[5] == board[6]:

print(player, "wins")

playing = False

# straight top row

elif board[7] == board[8] and board[8] == board[9]:

print(player, "wins")

playing = False

# straight bottom row

elif board[1] == board[2] and board[2] == board[3]:

print(player, "wins")

playing = False

# diagonal 1

elif board[7] == board[5] and board[5] == board[3]:

print(player, "wins")

playing = False

# diagonal 2

elif board[1] == board[5] and board[5] == board[9]:

print(player, "wins")

playing = False

# straight column middle

elif board[8] == board[5] and board[5] == board[2]:

print(player, "wins")

playing = False

# straight column right

elif board[7] == board[4] and board[4] == board[1]:

print(player, "wins")

playing = False

# straight column left

elif board[9] == board[6] and board[6] == board[3]:

print(player, "wins")

playing = False

#calling the main functions of the code

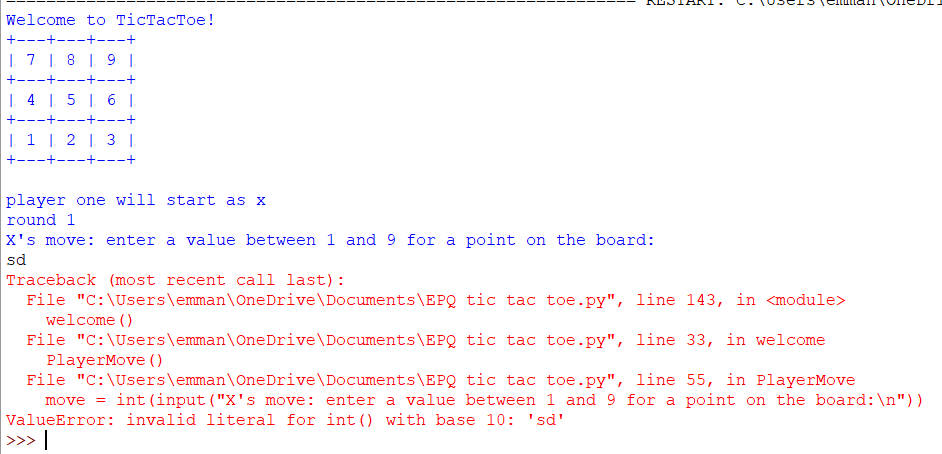
welcome()

#while loop to keep the game going

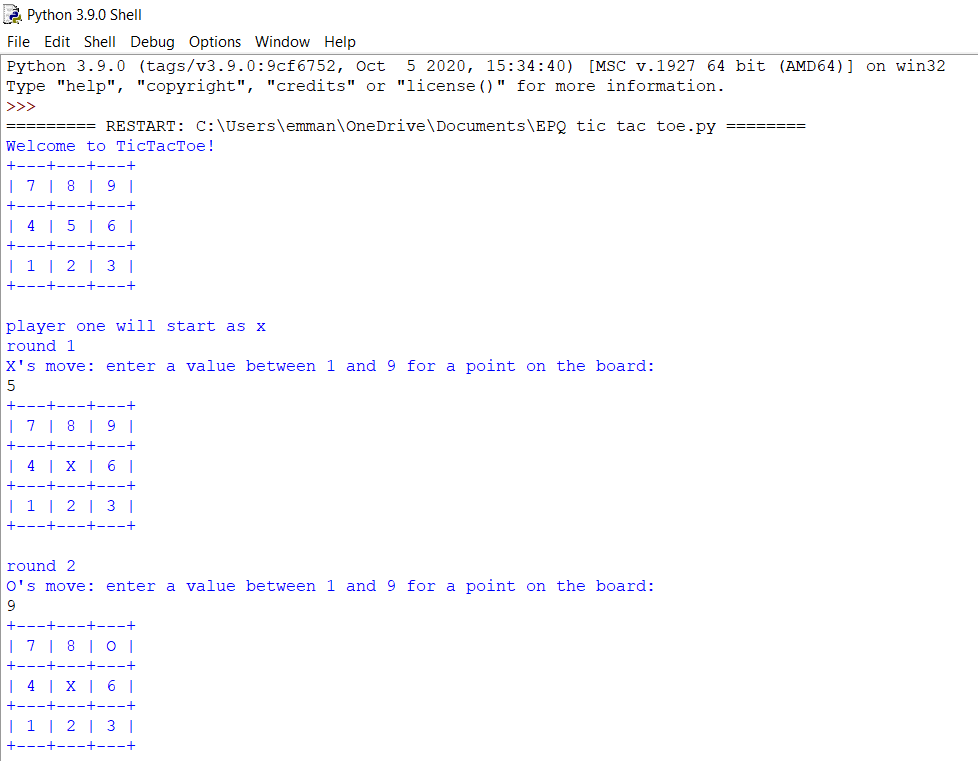
while playing:

PlayerMove()

#### Problems:

* An integer value, or whole number, is needed to be entered for each move so program will crash if non integer value between 1 and 9 is entered as a move.
* The Program needs to be repeated until the user decides that the do not want to play anymore.

### Play through.

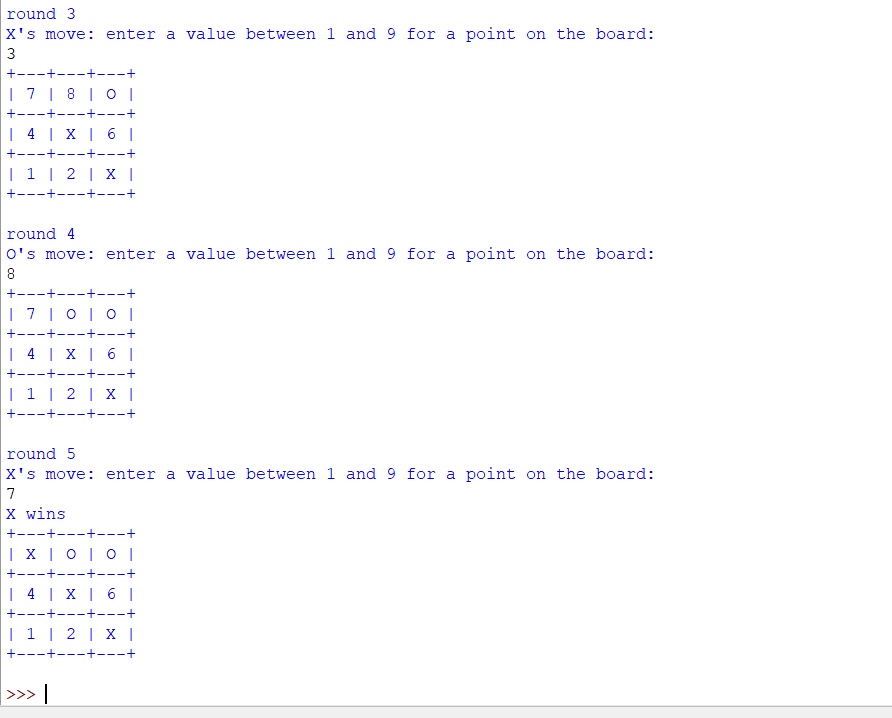


Normal run of the game

Welcome message is displayed. With the numbers that represent the points on the board displayed

Round 1: player X marks position 5 on the board.

Round 2: player O marks position 9 on the board.

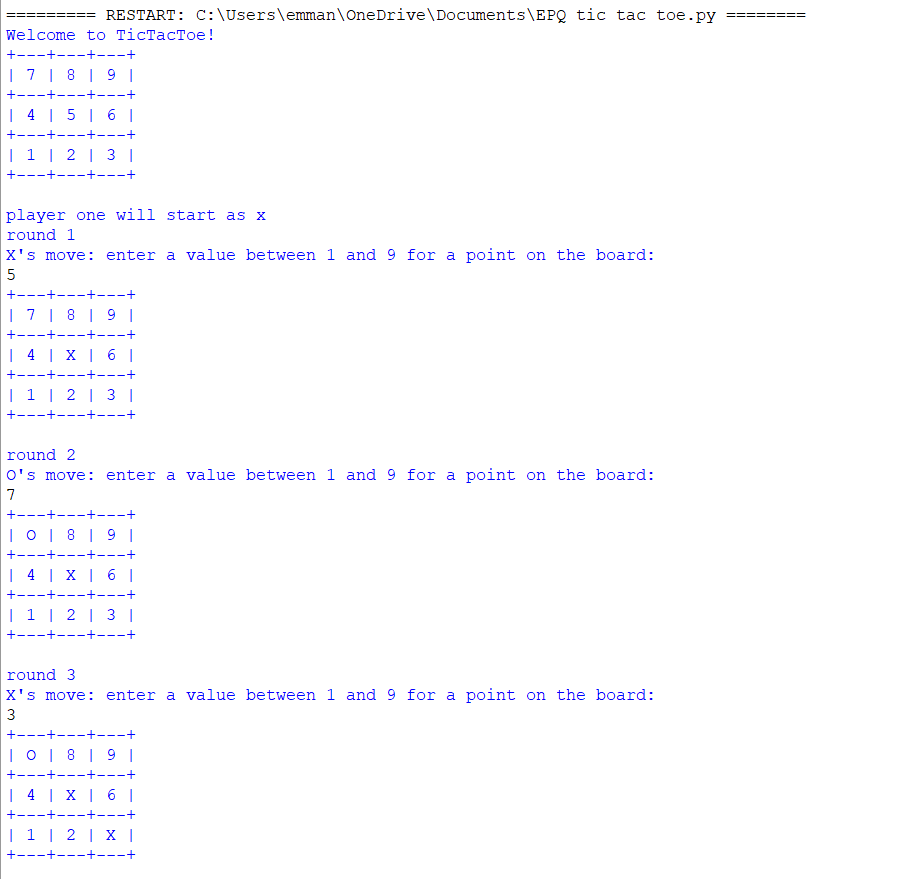


Round 3: player X marks position 3 on the board

Round 4: player O marks position 8 on the board.

Round 5: player X marks position 7 on the board to win the game!

Displays player wins and program ends.



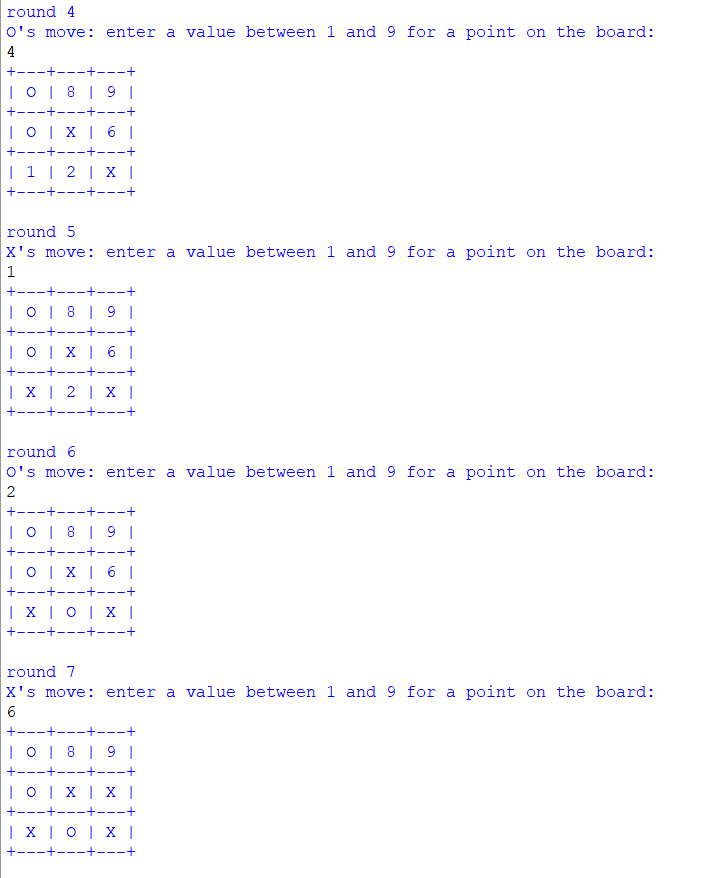
Make the game draw

Displays welcoming

Round 1: player X marks position 5 on the board.

Round 2: player O marks position 7 on the board.

Round 3: player X marks position 3 on the board.

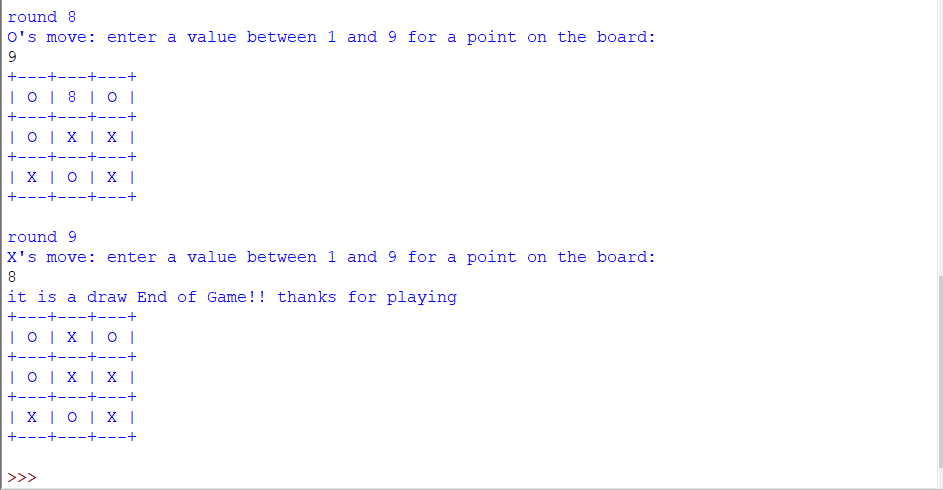


Round 4 : player O marks position 4 on the board.

Round 5: player X marks position 1 on the board.

Round 6: player O marks position 2 on the board.

Round 7: player X marks position 6 on the board.



Round 8 player O marks point 9 on the board to block player X

Round 9 player X marks point 8 on the board.

Game board full program

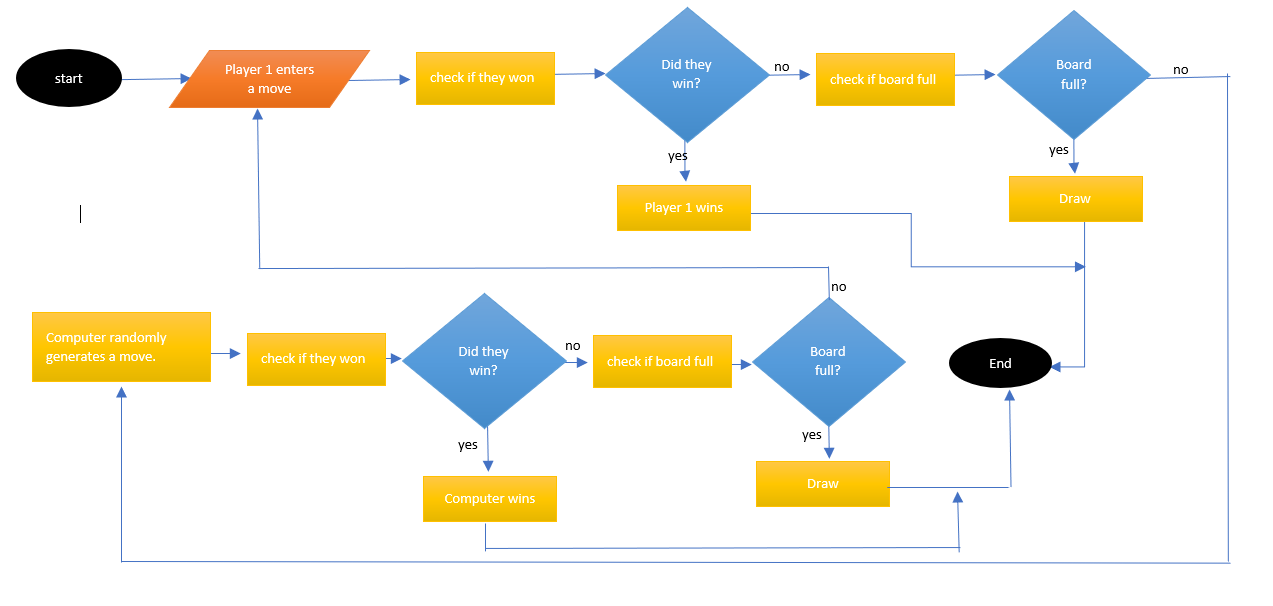
displays final bord and exits the program.

## Tic-tac-toe Player vs Random AI (level1)

### Plan

#### Flowchart

*diagram made by me\**



#### Worded plan

This one works the same as the player v player version but instead of player two inputting a move every other round the computer will select a random number between 1 and 9 (the range of the positions on the board) and make a move.

Add an option to allow the user to pick with letter they would like to play (X or O)

Also does not have to worry about having an error message from a mark being made on an already taken spot or errors form miss clicked values entered as a move that fall out the range of the board or are not integers.

This format is completely random meaning all the AI’s possible win will be by complete chance and it’s very easy to win.

### Program analysis

*Program information in red.*

#### Full Code

# Random AI tic Tac Toe game in python.

# import random

import random

# make the board.

board = ["Ignore me!", "1", "2", "3", "4", "5", "6", "7", "8", "9"]

rounds = 0

playing = True

choice = ""

def DisplayBoard(board):

print("+---" + "+--" + "-+" + "---+\n" +

"| " + board[7] + " | " + board[8] + " | " + board[9] + " |\n" +

"+---" + "+--" + "-+" + "---+\n" +

"| " + board[4] + " | " + board[5] + " | " + board[6] + " |\n" +

"+---" + "+--" + "-+" + "---+\n" +

"| " + board[1] + " | " + board[2] + " | " + board[3] + " |\n" +

"+---" + "+--" + "-+" + "---+\n")

def welcome():

global rounds

global choice

print("Welcome to TicTacToe!")

print("+---" + "+--" + "-+" + "---+\n" +

"| " + "7" + " | " + "8" + " | " + "9" + " |\n" +

"+---" + "+--" + "-+" + "---+\n" +

"| " + "4" + " | " + "5" + " | " + "6" + " |\n" +

"+---" + "+--" + "-+" + "---+\n" +

"| " + "1" + " | " + "2" + " | " + "3" + " |\n" +

"+---" + "+--" + "-+" + "---+\n")

#ask the user which letter the user what character they want to play

choice = input("enter which player you will be: (O or X):\n").upper()

while choice != "X" and choice != "O":

choice = input("invalid character enter either O or X\n").upper()

rounds = rounds + 1

FirstMove(choice)

#funtion to check if position on board is empty

def Empty(move, player):

if board[move] == "X" or board[move] == "O":

return False

elif board[move] != "X" or board[move] != "O":

return True

#function to decide if computer will be making a move or if player will be making a move

def FirstMove(choice):

global rounds

XorO = rounds % 2

if XorO == 1:

if choice == "X":

player = choice

PlayerMove()

elif choice == "O":

player = choice

ComputerMove()

elif XorO == 0:

if choice == "X":

player = choice

ComputerMove()

elif choice == "O":

player = choice

PlayerMove()

#code behind Player's movement

#comments are just for the changes that i mad to simple tic tac to game

def PlayerMove():

global rounds

global board

global player

print("round: ",rounds)

print("player move!:")

XorO = rounds % 2

if XorO == 1:

player = "X"

#implementing Try and accept function to code to make it more robust

try:

move = int(input("X's move: enter a value between 1 and 9 for a point on the board:\n"))

while (move < 1 or move > 9) :

print("invalid move enter a value between 1 and 9")

move = int(input())

Empty(move, player)

while Empty(move, player) == False:

move = int(input("SpaceTaken! reenter move:\n"))

Empty(move,player)

board[move] = player

rounds = rounds + 1

check()

DisplayBoard(board)

except:

print("not a legal move try again")

else:

player = "O"

try:

move = int(input("O's move: enter a value between 1 and 9 for a point on the board:\n"))

while (move < 1 or move > 9) :

print("invalid move enter a value between 1 and 9")

move = int(input())

Empty(move, player)

while Empty(move, player) == False:

move = int(input("SpaceTaken! reenter move:\n"))

Empty(move,player)

board[move] = player

rounds = rounds + 1

check()

DisplayBoard(board)

except:

print("not a legal move try again the value you entered must be a number")

#funtion to make the computer's move

def ComputerMove():

global rounds

global board

global player

#diaplay the round that the game is on

print("round: ",rounds)

print("computer move!:")

XorO = rounds % 2

if XorO == 1:

#how the computer's X player is coded

player = "X"

#a random numerical value is calculated between 1 and 9 to represent the posible possitions on the board

move = random.randint(1,9)

#checks if move picked is available on the board

Empty(move, player)

#if the move is not available continue to pick a random number untill available move is picked

while Empty(move, player) == False:

move = random.randint(1,9)

Empty(move,player)

#mark the board with the computer's letter

board[move] = player

rounds = rounds + 1

check()

DisplayBoard(board)

else:

#repeat X for O

player = "O"

move = random.randint(1,9)

Empty(move, player)

while Empty(move, player) == False:

move = random.randint(1,9)

Empty(move,player)

board[move] = player

rounds = rounds + 1

check()

DisplayBoard(board)

#funtion to check if board if full

def Full():

spaces = False

#checks each position on the board and sees if it occupided with a player's letter

for i in range(1,9):

if board[i] != "X" and board[i] != "O":

spaces = True

if spaces == False:

return True

else:

return False

#functio to check if its the end of the game

def check():

global playing

#checking if the borard is full

if Full():

print("it is a draw End of Game!! thanks for playing")

playing = False

# straight middle row

if board[4] == board[5] and board[5] == board[6]:

print(player, "wins")

playing = False

# straight top row

elif board[7] == board[8] and board[8] == board[9]:

print(player, "wins")

playing = False

# straight bottom row

elif board[1] == board[2] and board[2] == board[3]:

print(player, "wins")

playing = False

# diagonal 1

elif board[7] == board[5] and board[5] == board[3]:

print(player, "wins")

playing = False

# diagonal 2

elif board[1] == board[5] and board[5] == board[9]:

print(player, "wins")

playing = False

# straight column middle

elif board[8] == board[5] and board[5] == board[2]:

print(player, "wins")

playing = False

# straight column right

elif board[7] == board[4] and board[4] == board[1]:

print(player, "wins")

playing = False

# straight column left

elif board[9] == board[6] and board[6] == board[3]:

print(player, "wins")

playing = False

#calling the main funtions of the game to start it.

def MainGame():

welcome()

while playing:

FirstMove(choice)

check()

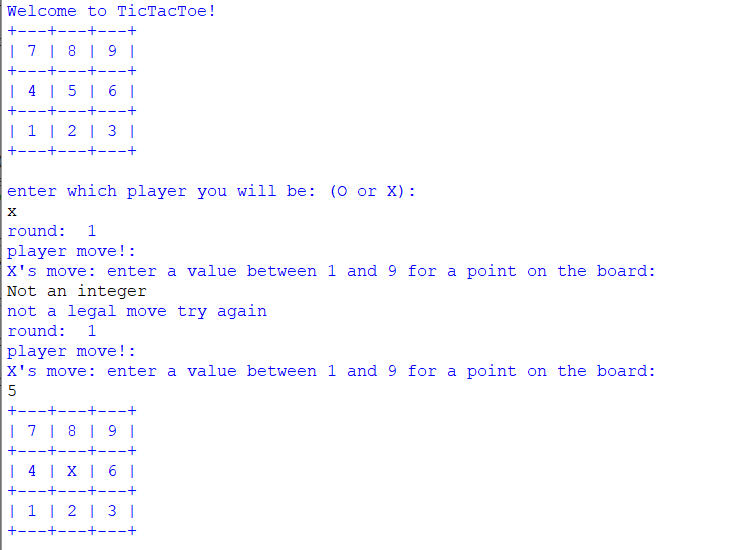
MainGame()

#### Problems

* The Program needs to be repeated until the user decides that the do not want to play anymore.

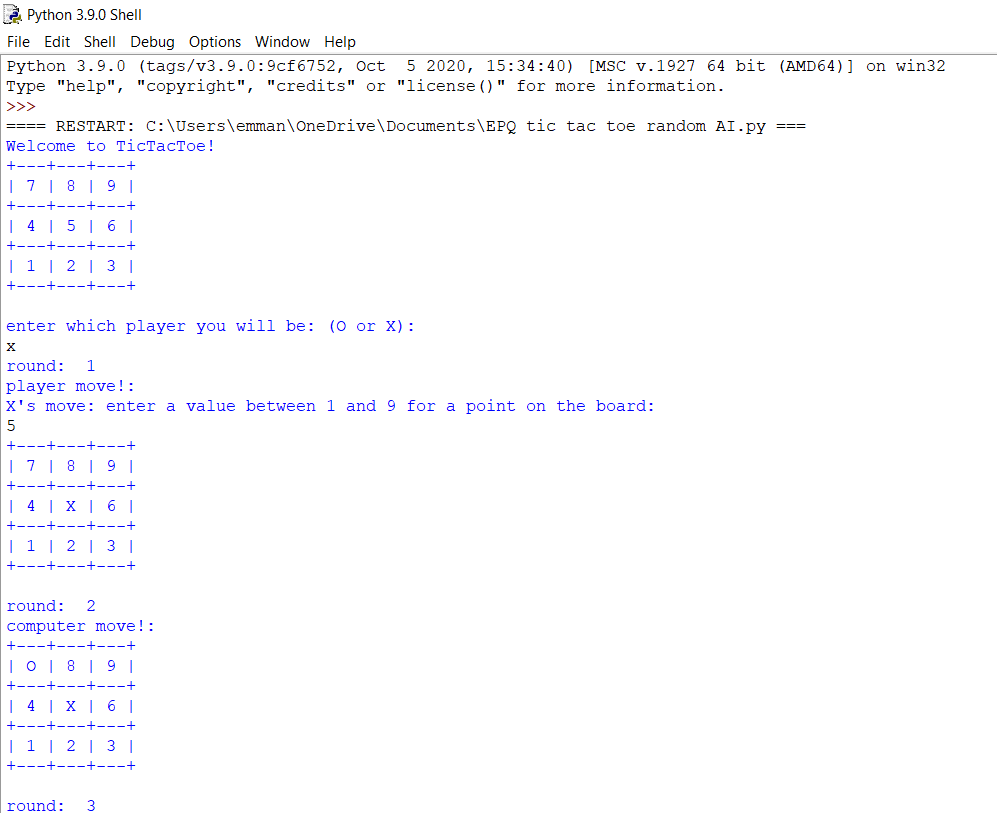
#### Fixes to previous Level

* An integer value, or whole number, is needed to be entered for each move so program will crash if non integer value between 1 and 9 is entered as a move.



* + added the try and except function to stop the error message from coming up when a non integer value is entered

### Play throughs



Playing as X

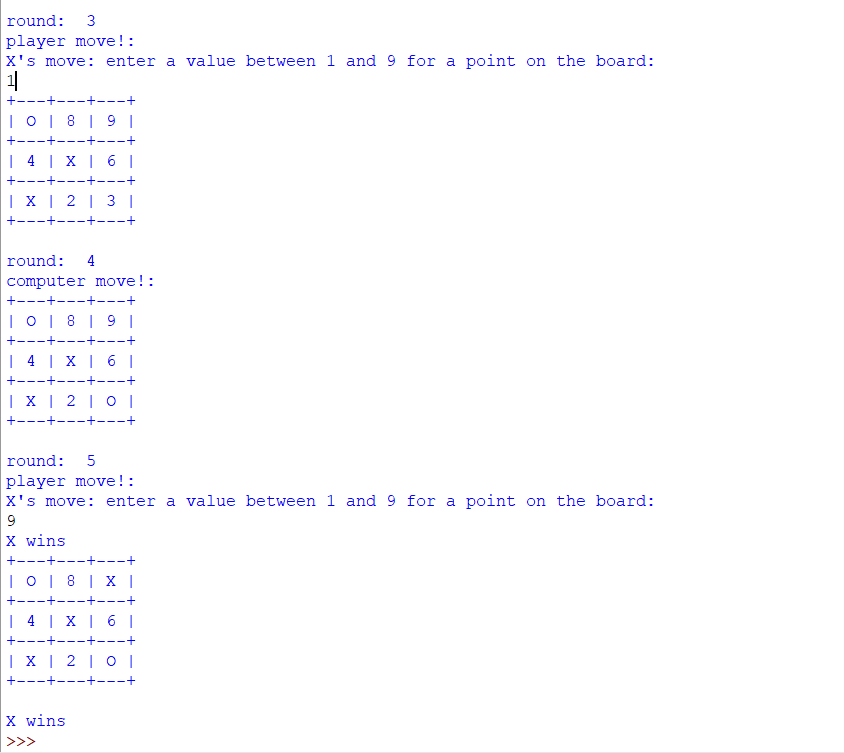
Displays welcome message

User picks character that they want to play which is X in this case

Game begins!

Round 1 user marks position 5 on the board

Round 2 computer which plays O marks position 7 bon the board

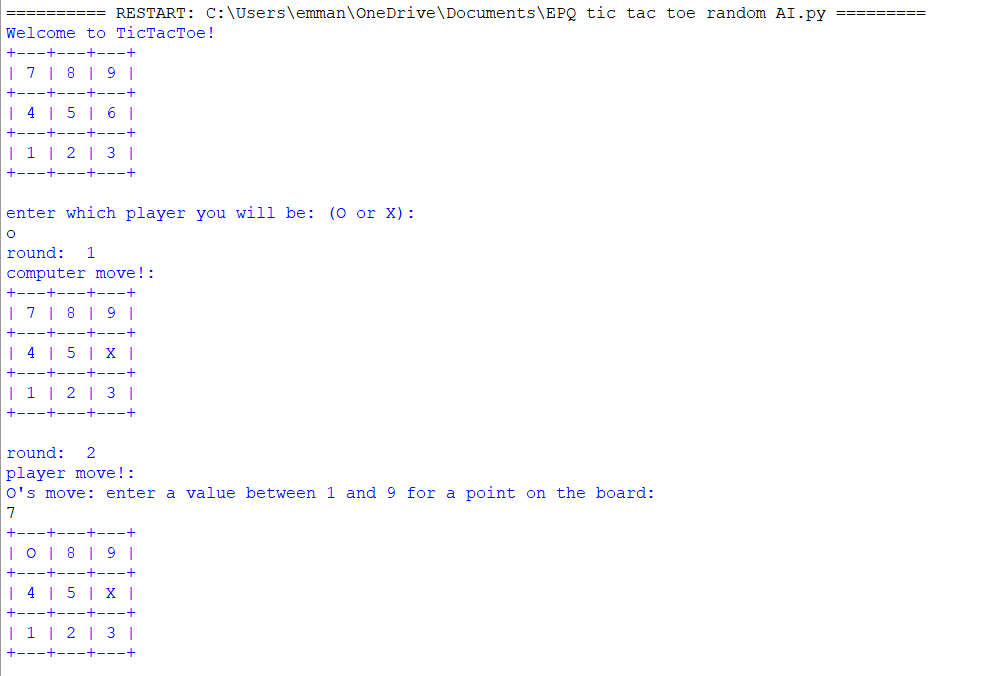


Round 3: user marks position 1 on the board

Round 4: computer marks position 3 on the board

Round 5: user marks position 9 on the board to win the game.

Displays that the player X wins (which is the user and ends the program)



User playing as O

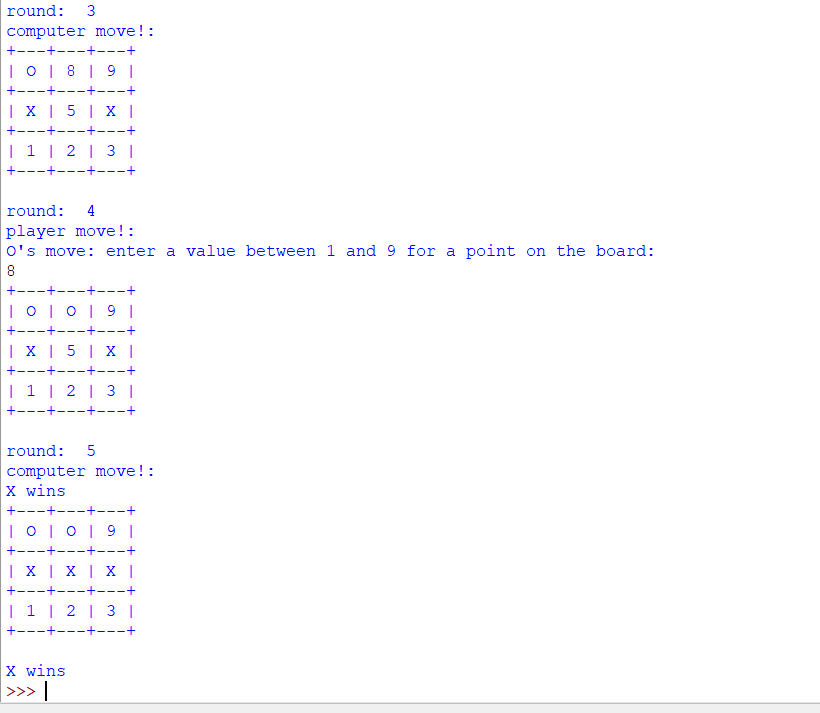
Displays welcome message

User then asked to pick which character it wants to play , users chooses to play O

Game begins

Round 1 computer starts as it is X marks position 6 on the board

Round 2 user marks position 7 on the board.



Round 3 computer marks position 4 on the board

Round 4 user marks position 8 on the board

Round 5 computer marks position 5 and wins the game.

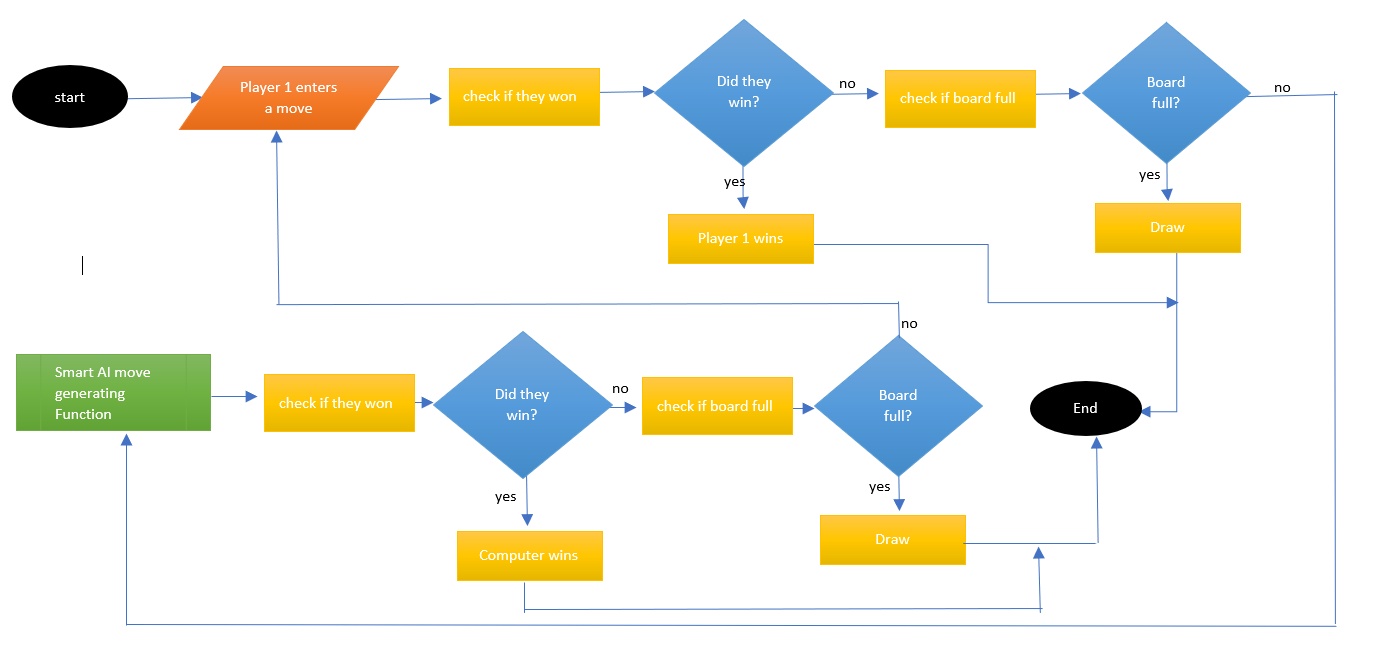
Displays final board and ends the program.

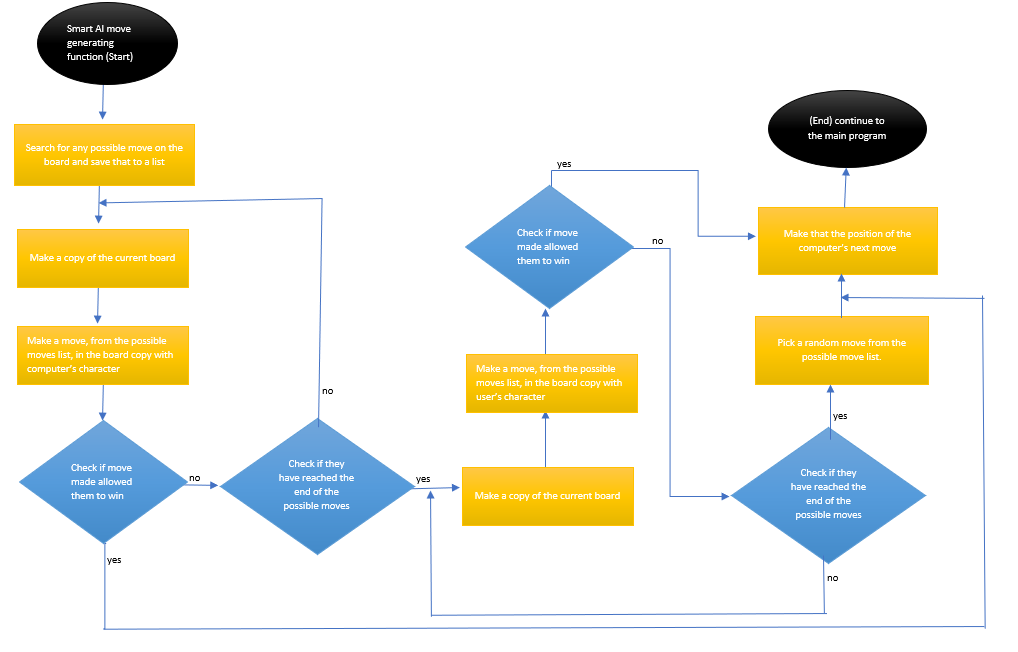
## Tic-tac-toe: Player v Smart AI (level2)

### Plan

#### Flowcharts

*diagram made by me\**





#### Worded plan

Structure of this comes from:

* Used to help make the Simple AI used in the this level. (Sweigart, n.d.)

This level has the same main run as the player vs random AI code but instead of the computer picking a position on the board at random, it picks a more competent way to place marks on the board.

The programs AI move function is the place in the program that I defined the behaviour of the AI when it is making a move on the board.

How the AI’s function works:

1. makes a list of the possible moves the computer could make
   1. going through that list of possible moves create a copy of the board and make a move from the list into the copied.
   2. if this move means that the computer wins
      1. make that the computer’s move on the real board
      2. then leave function
2. repeat step 1 until every value in the possible moves list has been tested or until one gives the computer a win.
3. If no win is available for the computer repeat steps 1 and 2 for the user’s character instead
4. If a win is available for the user in the possible moves list
   1. Make that the computer’s move on the real board
   2. Then leave the function
5. If no possible wins for user is found either
   1. *just pick a random position from the possible moves*

##### Improvements for step 5

1. If no possible wins for user is found either
   1. *(Instead of just picking a random position from the possible moves) try and pick a random corner then exit function. Positions [1,3,7,9]*
   2. *If no corner available, try and pick the centre then exit function. Position [5]*
   3. *If the centre and no corners are available, try and pick a side then exit function. positions [2,6,4,8]*

Level 2.5 concept from the improvement

Corners have more value in Tic-Tac-Toe than sides and centre, just adds alittle bit more complexity to the AI and makes it harder to Beat

### Program analysis

*Program information in red.*

#### Code

# simple tic Tac Toe game in python.

# import random

import random

# make the board.

board = ["Ignore me!", "1", "2", "3", "4", "5", "6", "7", "8", "9"]

rounds = 0

playing = True

choice = ""

ComputerLet = ""

def DisplayBoard(board):

print("+---" + "+--" + "-+" + "---+\n" +

"| " + board[7] + " | " + board[8] + " | " + board[9] + " |\n" +

"+---" + "+--" + "-+" + "---+\n" +

"| " + board[4] + " | " + board[5] + " | " + board[6] + " |\n" +

"+---" + "+--" + "-+" + "---+\n" +

"| " + board[1] + " | " + board[2] + " | " + board[3] + " |\n" +

"+---" + "+--" + "-+" + "---+\n")

#welcoming function

def welcome():

#makeing variables global so they can be used in function

global rounds

global choice

#displays welcome message

print("Welcome to TicTacToe!")

print("+---" + "+--" + "-+" + "---+\n" +

"| " + "7" + " | " + "8" + " | " + "9" + " |\n" +

"+---" + "+--" + "-+" + "---+\n" +

"| " + "4" + " | " + "5" + " | " + "6" + " |\n" +

"+---" + "+--" + "-+" + "---+\n" +

"| " + "1" + " | " + "2" + " | " + "3" + " |\n" +

"+---" + "+--" + "-+" + "---+\n")

choice = input("enter which player you will be: (O or X):\n").upper()

while choice != "X" and choice != "O":

choice = input("invalid character enter either O or X\n").upper()

rounds = rounds + 1

FirstMove(choice)

#function that checks if the place on board is empty

def Empty(move, player):

if board[move] == "X" or board[move] == "O":

return False # gives out false meaning the place has been taken move not able to be done

elif board[move] != "X" or board[move] != "O":

return True# gives out true as a legal move can be made here

#function ot coordinate who is making the moves

def FirstMove(choice):

global rounds

global ComputerLet

XorO = rounds % 2

if XorO == 1:

if choice == "X":

Player = choice

PlayerMove()

elif choice == "O":

ComputerMove()

check(board,ComputerLet)

DisplayBoard(board)

elif XorO == 0:

if choice == "X":

ComputerMove()

check(board,ComputerLet)

DisplayBoard(board)

elif choice == "O":

player = choice

PlayerMove()

#function to pick a random place in the possible moves list

def SelectRandom(spaces):

length = len(spaces)

index = random.randint(0,(length-1))

picked = spaces[index]

return picked

#function that allows for the player movement

def PlayerMove():

#makeing varibles global for later use in function

global rounds

global board

global player

print("Round:",rounds)

print("player's move!:")

XorO = rounds % 2

if XorO == 1:

player = "X"

try:

move = int(input("X's move: enter a value between 1 and 9 for a point on the board:\n"))

while (move < 1 or move > 9):

print("invalid move enter a value between 1 and 9")

move = int(input())

Empty(move, player)

while Empty(move, player) == False:

move = int(input("SpaceTaken! reenter move:\n"))

Empty(move,player)

board[move] = player

rounds = rounds + 1

check(board, player)

DisplayBoard(board)

except:

print("not a legal move try again")

else:

player = "O"

try:

move = int(input("O's move: enter a value between 1 and 9 for a point on the board:\n"))

while (move < 1 or move > 9):

print("invalid move enter a value between 1 and 9")

move = int(input())

Empty(move, player)

while Empty(move, player) == False:

move = int(input("SpaceTaken! reenter move:\n"))

Empty(move,player)

board[move] = player

rounds = rounds + 1

check(board,player)

DisplayBoard(board)

except:

print("not a legal move try again the value u entered must be a number")

def ComputerMove():

global rounds

global board

global player

global ComputerLet

print("Round:",rounds)

print("Computer's move!:")

XorO = rounds % 2

if XorO == 1:

rounds = rounds + 1

ComputerLet = "X" #computer letter

PlayerLet = "O" #player letter

#make a list of possible moves

possiblemoves = []

for place, possible in enumerate(board):#enumerate gives me the value that is a possition of th elist

if (possible != "O" and possible != "X") and (place != 0): #value in list position is saved to possible place in list is saved to place

possiblemoves.append(place)

#checking if the player can win or computer can win

for player in [ComputerLet,PlayerLet]:

for place in possiblemoves: # goes through all possible moves on the board

boardcopy = board[:] #make a copy of board

boardcopy[place] = player

if check2(boardcopy,player):

board[place] = ComputerLet #this still works even if both the

return

move = SelectRandom(possiblemoves)

#mark the board with the computer's letter

board[move] = ComputerLet

return

else:

rounds = rounds + 1

ComputerLet = "O"#computer letter

PlayerLet = "X"#player letter

#make a list of possible moves

possiblemoves = []

for place, possible in enumerate(board):#enumerate gives me the value that is a possition of th elist

if (possible != "O" and possible != "X") and (place != 0):#value in list position is saved to possible place in list is saved to place

possiblemoves.append(place)

#checking if the player can win or computer can win

for player in [ComputerLet,PlayerLet]:

for place in possiblemoves:

boardcopy = board[:]

boardcopy[place] = player

if check2(boardcopy,player):

board[place] = ComputerLet #this still works even if both the

return

move = SelectRandom(possiblemoves)

#mark the board with the computer's letter

board[move] = ComputerLet

return

#function to check if the board is full

def Full():

spaces = False#sets free paces to false

for i in range(1,9):

if board[i] != "X" and board[i] != "O":#used to go through the whole board

spaces = True #changes to true if there is space only if there is a space

if spaces == False:

return True

else:

return False

#this check if there is winner on board

def check2(board,player):

# straight middle row

if ((board[4] == board[5] and board[5] == board[6])

# straight top row

or (board[7] == board[8] and board[8] == board[9])

# straight bottom row

or (board[1] == board[2] and board[2] == board[3])

# diagonal 1

or (board[7] == board[5] and board[5] == board[3])

# diagonal 2

or (board[1] == board[5] and board[5] == board[9])

# straight column middle

or( board[8] == board[5] and board[5] == board[2])

# straight column right

or (board[7] == board[4] and board[4] == board[1])

# straight column left

or (board[9] == board[6] and board[6] == board[3])):

return True

else:

return False

#function to check if the board has come to and ending state

def check(board, player):

global playing

#checking if the board is full

if Full():

print("it is a draw End of Game!! thanks for playing")

playing = False

if check2(board,player):

print(player, "wins")

playing = False

#main part of the game calls all the functions

def MainGame():

welcome() #displayes welcome message

while playing:

FirstMove(choice)

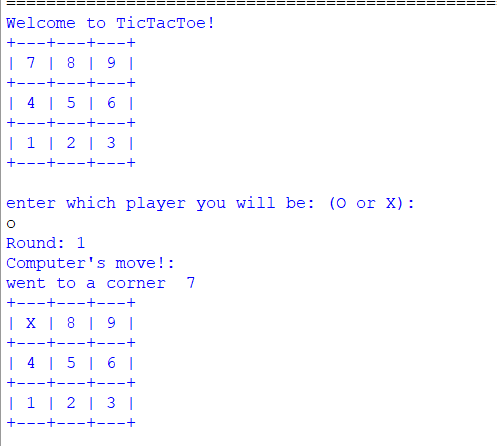
MainGame() #calls the main game function to start the game

#### Problems

* The Program needs to be repeated until the user decides that they do not want to play anymore.
* Should display the move the compute made all the time.

#### Improvements:

* For level 2.5 computer’s picked position is displayed



* Improved level 2 with the concept idea of level 2.5 (Will be known as level 3 in the final complete programmed project).
* The differences between 2.5 and 2 are:
  + Level 2.5 is uses a more complex AI algorithm that makes it harder than level

Different piece of code for level 2.5:

#function for the computer to make a decisive random move

def GoodRandomMove(possiblemoves):

global board

#check if corner open

opencorners = []

#make the list of open corners

for place in possiblemoves:

if place in [7,9,1,3]:

opencorners.append(place)

#pick a random corner from the list

if len(opencorners) > 0:

move = SelectRandom(opencorners)

board[move] = ComputerLet

print("went to a corner ",move)

return

#check if center open

elif 5 in possiblemoves:

move = 5

board[move] = ComputerLet

print("went to a center ",move)

return

#check if side open

opensides = []

#make the list of open side

for place in possiblemoves:

if place in [8,4,6,2]:

opensides.append(place)

#pick a random side

if len(opensides) > 0:

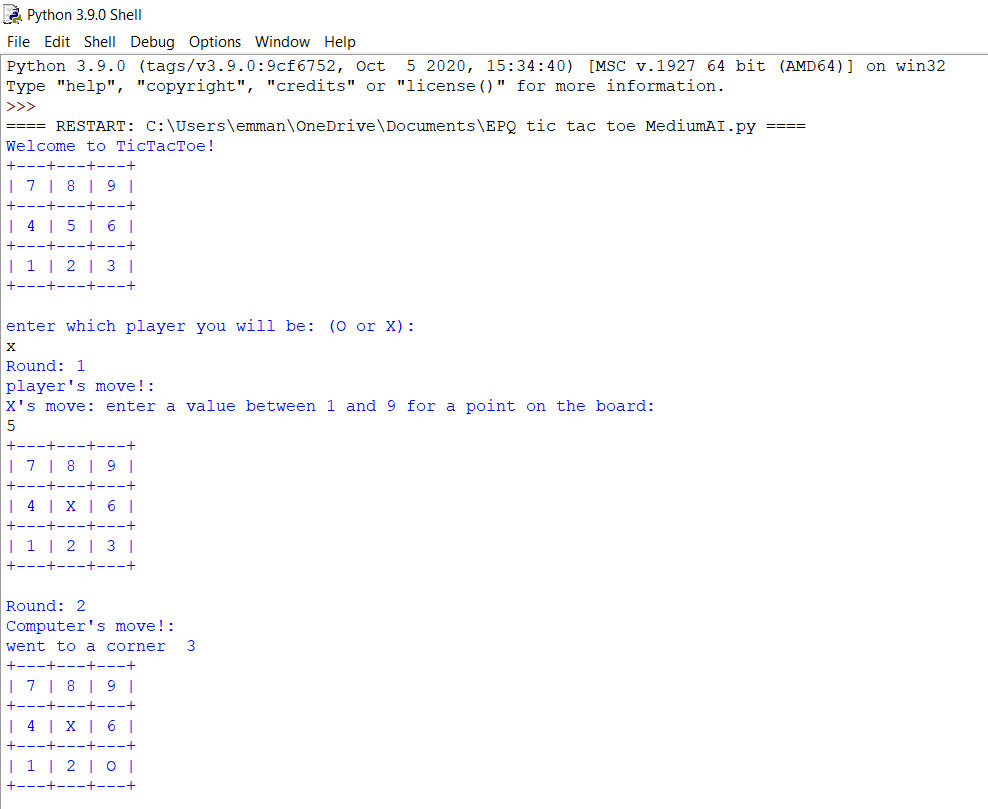
move = SelectRandom(opensides)

board[move] = ComputerLet

print("went to a side ",move)

return

### Play throughs of level 2/2.5



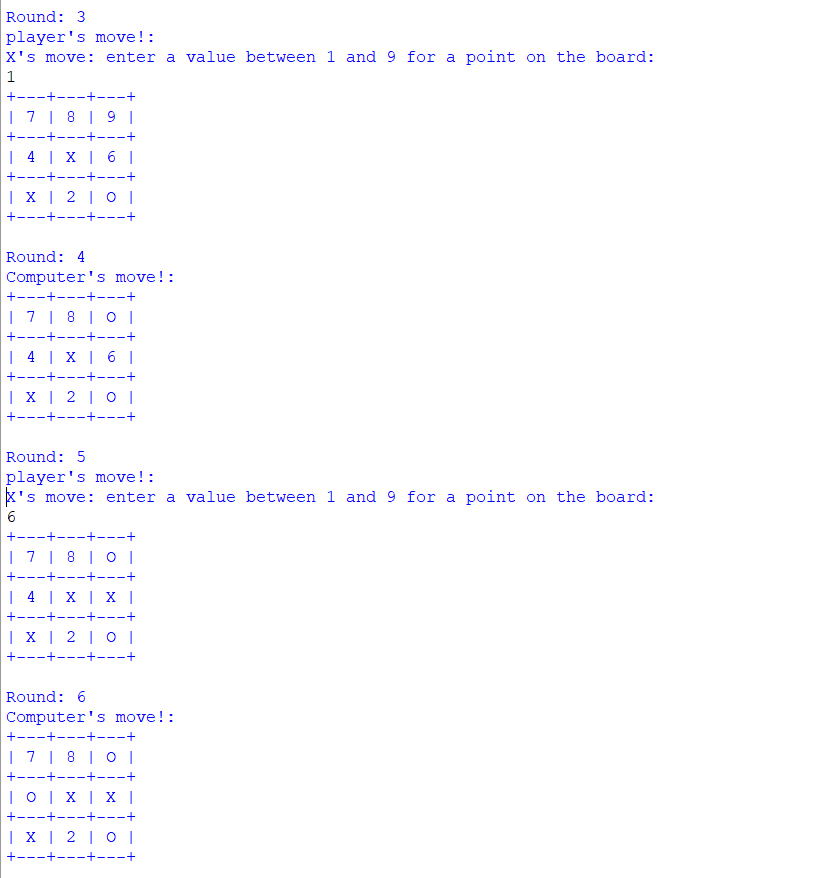
User plays as X

Welcome message displayed.

User pick to play as character x.

Round 1 user make a mark on position 5 on the board.

Round 2 computer(AI) makes a mark on position 3 on the board.

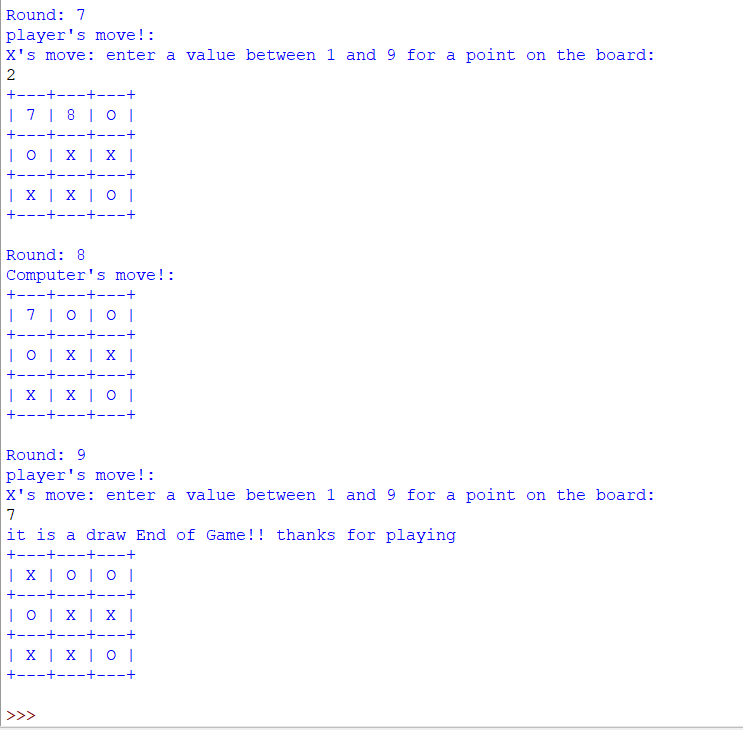


Round 3 user marks position 1 on the board

Round 4 computer (AI) marks position 9 on the board blocking the user’s diagonal win

Round 5 user marks position 6 on the board

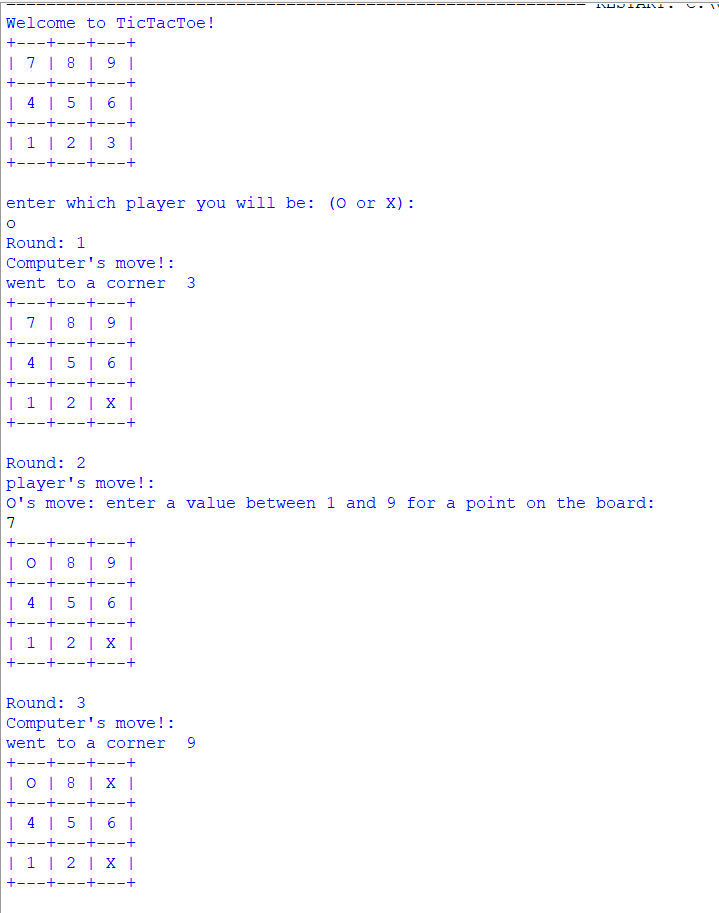
Round 6 Computer (AI) marks position 4 on the board to block user’s horizontal win



Round 7 user marks position 2 on the board

Round 8 computer marks position 8 on the board blocking user’s vertical win

Round 9 user marks position 7 and leave the game at a draw.



User playing as O

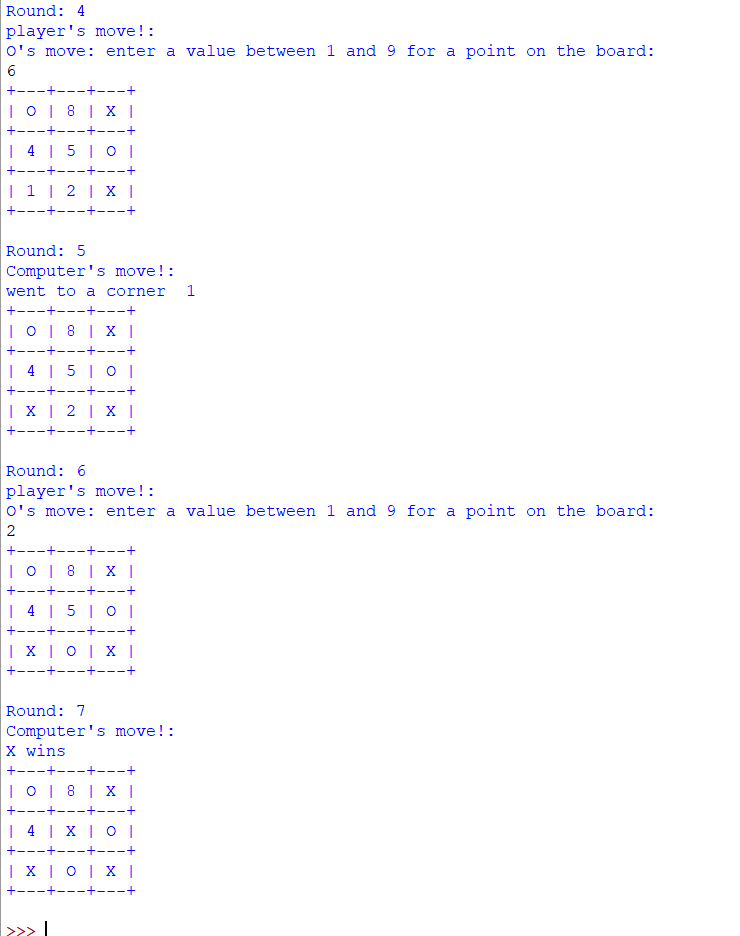
Welcome message displayed

User is asked to pick with character they want to play they pick O

Round 1 the computer marks position 3 on the board4

Round 2 the user marks position 7 on the board

Round 3 the computer marks position 9 on the board



Round 4 user marks position 6 on the board to block computer vertical win!

Round 5 computer marks position 1 on the board

Round 6 user marks position 2 on the board to block computers horizontal win

Round 7 computer marks position 5 to win the game

Display win and end program

## Tic-tac-toe: Player v Minimax AI (level 4)) impossible (Theory)

### What is the minimax algorithm?

And algorithm is a sequence of steps used to solve a problem. The minimax It is a decision-making algorithm in which the computer tries to get the maximum results while going against something trying to get the minimum result or vis versa. (Cruz, n.d.)

The way my tic-tac-toe minimax AI will go about making the decision, of which move the computer should make for the best possible result, by simulating the further rounds of the game with the possible moves the compute or the player could make until a Game Over state is reached and then evaluating the state by giving it a score according to the results.

Due to the large amount to simulating the AI will have to do at the beginning of the game , as tic tac toe has a very large number of possible outcomes the game could have that the computer has to go through, I will also be using opening book (Krivokuća, n.d.) ,which is the act of inputting starting moves for the computer that are known to increase the chance of victory, to have good randomly generated moves prepared for the computer to make instead of it having to decide them itself

### Plan

Influences on my minimax program:

* (nextProgram, n.d.) and (Krivokuća, n.d.) - Understanding of the minimax algorithm and alpha-beta pruning.
* (Ying, n.d.) - Idea for diagram and scoring system used in program

How I plan the AI should work when picking it move:

At the start of the game in the first few round the computer should pick a move to play form the opening books either according to what the player picked as their first move (if player is playing as X) or pick a random good starting move if the computer plays as X

Opening books starts including:

* Playing corners as the provide a good advantage when played.
* Playing centre to counter player going to a corner as starting move followed by a play to the side to counter the player going to another corner and two-way tricking the computer.
* Etc…

Then for the following rounds:

1. Collect all possible move, that can be made on the board, into a list.
2. AI should them simulate one of the moves being made on the board.
3. Then collect a list of the possible moves left again.
4. Repeat step 2 to 3 until end point of the game is simulated ( so on the board copy the game has come to an end as someone won or there was a draw.)
   1. Give a value to this end game board state.
   2. If this value is better that the best score possible collect the move made and set that move to be the computer’s next move
5. Then repeat steps 2 to 4 until ever possible move has been made and all end game state values have been calculated.

#### Visual representation of how the minimax algorithm show work when picking the best possible move.

*\*diagram made by me\**

Maximum’s move

Minimums’s move

Maximum’s move

*Same scoring system as* (Ying, n.d.)

Value of end state is calculated using the formula:

Value = winner \* (number of possible moves left + 1)

If winner is the max player(X,) winner = 1 as to mean the maximum score possible

If winner is the min player(O), winner = -1 as to mean the minimum score possible

If no one wins, winner = 0 as to mean the state has no value cause no one won

Form the graph the highest possible move which leads to the best end state for the AI is X going to the middle square on the bottom row (position 2)

### Program analysis

*Program information in red.*

#### Code

# MiniMax tic Tac Toe game in python.

# import random

import random

# import system to change recursion limit

import sys

x = 1000000000

sys.setrecursionlimit(x)

# make the board.

board = ["Ignore me!", "1", "2", "3", "4", "5", "6", "7", "8", "9"]

rounds = 0

playing = True

choice = ""

ComputerLet = ""

depth = 0

global count

count = 0

def DisplayBoard(board):

print("+---" + "+--" + "-+" + "---+\n" +

"| " + board[7] + " | " + board[8] + " | " + board[9] + " |\n" +

"+---" + "+--" + "-+" + "---+\n" +

"| " + board[4] + " | " + board[5] + " | " + board[6] + " |\n" +

"+---" + "+--" + "-+" + "---+\n" +

"| " + board[1] + " | " + board[2] + " | " + board[3] + " |\n" +

"+---" + "+--" + "-+" + "---+\n")

#welcoming function

def welcome():

#makeing variables global so they can be used in function

global rounds

global choice

#displays welcome message

print("Welcome to TicTacToe!")

print("+---" + "+--" + "-+" + "---+\n" +

"| " + "7" + " | " + "8" + " | " + "9" + " |\n" +

"+---" + "+--" + "-+" + "---+\n" +

"| " + "4" + " | " + "5" + " | " + "6" + " |\n" +

"+---" + "+--" + "-+" + "---+\n" +

"| " + "1" + " | " + "2" + " | " + "3" + " |\n" +

"+---" + "+--" + "-+" + "---+\n")

choice = input("enter which player you will be: (O or X):\n").upper()

while choice != "X" and choice != "O":

choice = input("invalid character enter either O or X\n").upper()

rounds = rounds + 1

FirstMove(choice)

#function that checks if the place on board is empty

def Empty(move, player):

if board[move] == "X" or board[move] == "O":

return False # gives out false meaning the place has been taken move not able to be done

elif board[move] != "X" or board[move] != "O":

return True# gives out true as a legal move can be made here

#function to coordinate who is making the moves

def FirstMove(choice):

global rounds

global ComputerLet

XorO = rounds % 2

if XorO == 1:

if choice == "X":

PlayerMove()

elif choice == "O":

ComputerMove()

check(board,ComputerLet)

DisplayBoard(board)

elif XorO == 0:

if choice == "X":

ComputerMove()

check(board,ComputerLet)

DisplayBoard(board)

elif choice == "O":

player = choice

PlayerMove()

def SelectRandom(spaces):

length = len(spaces)

index = random.randint(0,(length-1))

picked = spaces[index]

return picked

#function that allows for the player movement

def PlayerMove():

#makeing varibles global for later use in function

global rounds

global board

global player

print("Round:",rounds)

print("player's move!:")

XorO = rounds % 2

if XorO == 1:

player = "X"

try:

move = int(input("X's move: enter a value between 1 and 9 for a point on the board:\n"))

while (move < 1 or move > 9):

print("invalid move enter a value between 1 and 9")

move = int(input())

Empty(move, player)

while Empty(move, player) == False:

move = int(input("SpaceTaken! reenter move:\n"))

Empty(move,player)

board[move] = player

rounds = rounds + 1

check(board, player)

DisplayBoard(board)

except:

print("not a legal move try again")

else:

player = "O"

try:

move = int(input("O's move: enter a value between 1 and 9 for a point on the board:\n"))

while (move < 1 or move > 9):

print("invalid move enter a value between 1 and 9")

move = int(input())

Empty(move, player)

while Empty(move, player) == False:

move = int(input("SpaceTaken! reenter move:\n"))

Empty(move,player)

board[move] = player

rounds = rounds + 1

check(board,player)

DisplayBoard(board)

except:

print("not a legal move try again the value u entered must be a number")

#function to generate a good random move

def GoodRandomMove(possiblemoves):

#check if corner open

opencorners = []

#make the list of open corners

for place in possiblemoves:

if place in [7,9,1,3]:

opencorners.append(place)

#pick a random corner from the list

if len(opencorners) > 0:

move = SelectRandom(opencorners)

board[move] = ComputerLet

print("went to a corner ",move)

return

#check if center open

elif 5 in possiblemoves:

move = 5

board[move] = ComputerLet

print("went to a center ",move)

return

#check if side open

opensides = []

#make the list of open side

for place in possiblemoves:

if place in [8,4,6,2]:

opensides.append(place)

#pick a random side

if len(opensides) > 0:

move = SelectRandom(opensides)

board[move] = ComputerLet

print("went to a side ",move)

return

def possiblemoves(state):

places = [place for place, possible in enumerate(state) if (possible != "O" and possible != "X") and (place != 0)]

return places

#generate score of game state.

def Evaluate(state,depth,Player):

board = state[:]

if GameOver(board,Player)!= "Draw":

if Player == "X":

score = 1\*(depth)

elif Player == "O":

score = -1\*(depth)

return score

else:

score = 0

return score

def SimulateGoodMoves(state,best,ComputerLet):

global count

# count added to limit recursion and not crash code

count = count + 1

boardcopy = state[:]

if ComputerLet == "X":

PlayerLet = "O"

else:

PlayerLet = "X"

PossiblePlaces = possiblemoves(boardcopy)

#checking if the player can win or computer can win

for player in [ComputerLet,PlayerLet]:

for place in PossiblePlaces: # goes thriugh all possible moves on the board

boardcopy = state[:] #make a copy of board

boardcopy[place] = player

if GameOver(boardcopy,player) == "Player":

move = place #this still works even if both the

return move

depth = len(boardcopy)

for player in ["X","O"]:

for places in PossiblePlaces:

boardcopy[places] = places

boardcopyCon = boardcopy[:]

boardcopyCon.remove(places)

depth = len(boardcopy)

if count < 1000:

MiniMax(boardcopyCon,depth,player,best)

if (depth == 0) or (GameOver(boardcopy,player) != "Not Ended") or count == 100:

bestscore = Evaluate(boardcopy,depth,player)

print(bestscore)

if ComputerLet == "X":

if bestscore > best:

move = places

best = bestscore

return move, best

else :

if bestscore < best:

move = places

best = bestscore

return move, best

else:

move = places

return move

#the minimax algorithm

def MiniMax(state,depth,ComputerLet,best):

if count == 0:

if ComputerLet == "X":

best = 1-1000000000001

else:

best = 1000000000000

Bbest = best

move = SimulateGoodMoves(board,Bbest,ComputerLet)

return move

#computers move generator

def ComputerMove9):

# call all the variables needed

global rounds

global board

global player

global ComputerLet

print("Round:",rounds)

print("Computer's move!:")

count = 0

XorO = rounds % 2

if XorO == 1:

rounds = rounds + 1

nextmoves = possiblemoves(board)

depth = len(nextmoves)

ComputerLet = "X"#computer letter -maxplayer

PlayerLet = "O"#player letter - min player

if rounds < 3:

#good move start of game

GoodRandomMove(nextmoves)

else:

state = board

#worst possible value for the max player

best = 1-1000000000001

#run the minimax function

move = MiniMax(state,depth,ComputerLet,best)

print("computer moved to = ",move)

board[move] = ComputerLet

else:

rounds = rounds + 1

nextmoves = possiblemoves(board)

depth = len(nextmoves)

ComputerLet = "O” #computer letter

PlayerLet = "X" #player letter

print("actual round =", rounds)

#opening books - thing to do to get victory

if rounds < 4 :

if rounds == 3:

stopit = False

corners = [7,9,1,3]

for places in corners:

if board[places] == PlayerLet:

stopit = True

if stopit == True:

board[5] = ComputerLet

else:

#good move to be picked to start of game

GoodRandomMove(nextmove)

else:

GoodRandomMove(nextmoves)

else:

state = board

#worst score for teh computer as a min player

best = 100000000000

#run the minimax function to get next move

move = MiniMax(state,depth,ComputerLet,best)

print("computer moved to = ",move)

board[move] = ComputerLet

#funtion to check if the board is full

def Full():

spaces = False#sets free paces to false

for i in range(1,9):

if board[i] != "X" and board[i] != "O":#used to go through the whole board

spaces = True #changes to true if there is space only if ther is a space

if spaces == False:

return True

else:

return False

#this check if there is winner on board

def ThreeInRow(board):

for i in ["O","X"]:

# straight middle row

if ((board[4] == board[5] and board[5] == board[6] and board[4] == i)

# straight top row

or (board[7] == board[8] and board[8] == board[9] and board[7] == i)

# straight bottom row

or (board[1] == board[2] and board[2] == board[3] and board[1] == i)

# diagonal 1

or (board[7] == board[5] and board[5] == board[3] and board[7] == i)

# diagonal 2

or (board[1] == board[5] and board[5] == board[9] and board[1] == i)

# straight column middle

or( board[8] == board[5] and board[5] == board[2] and board[8] == i)

# straight column right

or (board[7] == board[4] and board[4] == board[1] and board[7] == i)

# straight column left

or (board[9] == board[6] and board[6] == board[3]) and board[9] == i):

winner = i

return winner

return False

#function to check if the board has come to and ending state

def check(board, winner):

global playing

#checking if the board is full

if Full():

print("it is a draw End of Game!! thanks for playing")

playing = False

#check if there is a winner to the game and display that winner

if ThreeInRow(board):

print(winner, "wins")

playing = False

# checking for an end-game state for the simulating

def GameOver(board, winner):

global playing

#checking if the borard is full

if Full():

return "Draw"

if ThreeInRow(board) == winner:#check if there is a winer on the board and that winner is the perosn who make the last move

return "Player"

elif (ThreeInRow(board) != winner) and (ThreeInRow(board) != False):

return "Other Player"

else:

return "Not Ended"

#main part of the game calls all the functions

def MainGame():

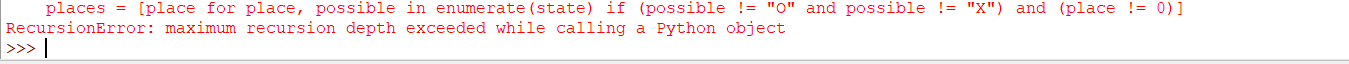
welcome() #displayes welcome message

while playing:

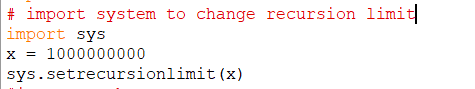
FirstMove(choice)

MainGame()#calls the main game funtion to start the game

#### Problems I came across.

* Recursion limit was always exceeded when running the minimax algorithm causing the program to constantly crash.
* Had trouble Implementing Alpha-Beta pruning (Krivokuća, n.d.) to shorten how many moves the minimax AI had to simulate, but that kept on causing logic issues when playing the game.
* Tried fixing recursion issue. By adding a count to how many times the minimax function could call itself but that caused the issue of stopping the minimax function before finding the best move to make and that made the computer make bad moves and always loose the game.

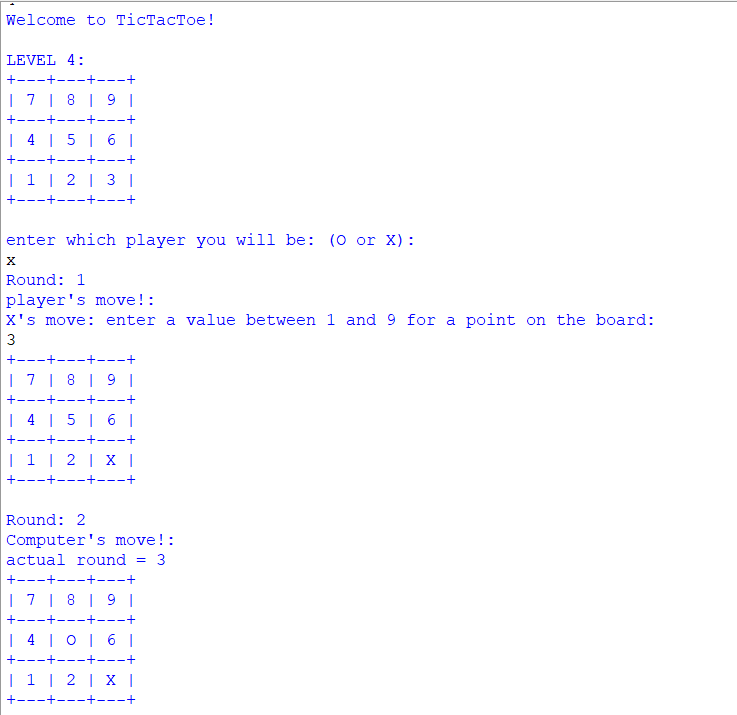
#### Fixes/improvements made:

* Recursion issue was resolved when I increased it from its original limit.(though due to some other reason I can’t figure out after a 1194 recursions the pram gives no error message and just crashes completely – most likely equipment issues but I am not sure)

### Play throughs.

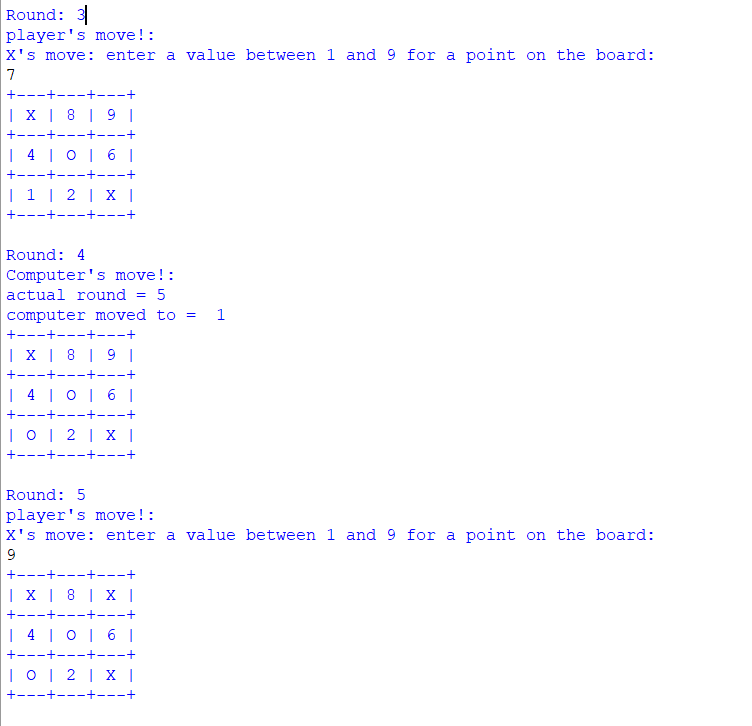
Bug was detected. There was a logical error with my program, so it works but it is beatable which is not the goal. Reasons for failure in making impossible level are:

* By using opening books at the beginning, to reduce the amount of simulating done by the minimax function, experienced human players can take advantage of this and beat the level very easily. E.g.



Round 1: Player is “X” and makes a move to one of the corners on the board, position 3.

Round 2: As per opening books the program will make a move to the centre of the board to counter the players cornering position.



Round 3: Player makes a move to the opposite corner

Round 4: Computer plays corner 1, as opening books says that corners have more value than sides.

Round 5: Player make a move to corner 9 on the board essentially winning the game as no matter what move is played by the computer the player can still win.

***Properly programmed minimax would have countered this by placing next move at a side position instead.***

* Due to inability to use alpha-beta pruning to reduce the simulating needed to be done to get the best possible moves, program cannot get the best possible move with its limited number of simulations it can do.

## Evaluation

### Testing

#### What the test table should show:

I believe the test table should show that there is a correlation between the AI’s complexity and its difficulty when I count the number of times the player won against the different AI I should see that against the more complex Ais, the higher levels, the player’s wins should be lower.

More complex program means it requires more lines of code as it goes through more procedures to when getting to its final output.

#### Test table for the number of win when played on each level.

This were collected by playing 10, 5 as “X” and 5 as “O”, games on each level with another computer program (level 2’s AI as that is around the same skill level as an average player) give a fair result,

Simple computer vs Levels results.

Playing as X or O:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Levels | X | | | | O | | | |
| Wins | draws | losses | Win Rate (%) | Wins | draws | loses | Win Rate (%) |
| Level 1 | 5 | 0 | 0 | 100 | 3 | 2 | 0 | 80 |
| Level 2 | 2 | 3 | 0 | 70 | 2 | 3 | 0 | 70 |
| Level 3 | 2 | 1 | 2 | 50 | 0 | 0 | 5 | 0 |
| Level 4 | 2 | 3 | 0 | 70 | 0 | 3 | 2 | 30 |

Overall score:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Levels | wins | draws | losses | Win Rate (%) |
| Level 1 | 8 | 2 | 0 | 90 |
| Level 2 | 4 | 6 | 0 | 70 |
| Level 3 | 2 | 1 | 7 | 25 |
| Level 4 | 2 | 6 | 2 | 50 |

* Win Rate = ((wins+(0.5\*draws))/total games played)\*100

#### What does my tests show?

* Ignoring level 4’s results as the complexity, between each level increases the difficulty as is shown by the decreasing win/loss ratio as the levels increase.
* Playing as x causes more wins for the player. - so, the X player in tic tac toe has an advantage

### What went well?

I was able to show that there was an increase in difficulty when playing again the more complex level AIs (from level 1 to level 3).

#### What could I have done better!?

Successfully applying Alpha-Beta Pruning to my minimax program to make it work better and to also prove my point further on more complex AI’s leads to more difficulty when playing against them.

# References

Balaganur, S. (2020, 2 12). Retrieved from Analytics India Magazine: https://analyticsindiamag.com/can-ai-create-video-games/

Cruz, C. (n.d.). *Cledersonbc/tic-tac-toe-minimax*. Retrieved from GitHub: https://github.com/Cledersonbc/tic-tac-toe-minimax

Krivokuća, M. (n.d.). *Minimax with Alpha-Beta Pruning in Python*. Retrieved from Stack Abuse: https://stackabuse.com/minimax-and-alpha-beta-pruning-in-python/

nextProgram. (n.d.). *I made an unbeatable Tic Tac Toe AI (Minimax algorithm)*. Retrieved from YouTube: https://www.youtube.com/watch?v=3nzupVMpZeA&t=363s

Sweigart, A. (n.d.). Retrieved from Inventwithpython.com: http://inventwithpython.com/chapter10.html

Ying, K. (n.d.). *Coding an UNBEATABLE Tic Tac Toe AI (Game Theory Minimax Algorithm EXPLAINED)*. Retrieved from YouTube: https://www.youtube.com/watch?v=fT3YWCKvuQE&t=762s