Tic Tac Toe Journal

November 23 2023

Planning:

Timeline:

[12:15pm]

| **Date** | **Goal** |
| --- | --- |
| November 23 2023  November 24 2023 | Review code and do planning  Requirements: flowcharts, Timeline, pseudocode |
| November 25 2023 | Program Journal  Implement code  (bug test) - if time |
| November 26 2023 | Complete if not finished  Crash test |

Program Requirements:

* Preserve as much as the code as I can
* Understand the given program
* Plan changes to existing code – set goals
* Plan with gantt chart deadlines
* Create flowchart
* Create pseudocodes
* Implement functions
  + validMove() - checks if spot is available on board
  + win(player) - checks for winner
  + playAgain() - asks user if they would like to play again
* Computer AI features
  + makeMove() - have all AI moves within
  + blockMove() - blocks if player is about to win
  + winningMove() - chooses winning spot
  + trickMove() - attempt to choose location where possible 2 winning spots
* Test code frequently

Pseudocode Functions:

[1:45pm]

**validMove()**

function validMove(move):

# convert move

r = (move - 1) // 3

c = (move - 1) % 3

IF 1 <= move <= 9 and board[r][c] NOT IN (' x ', ' o '):

RETURN True

else:

RETURN False

[2:05pm]

**win(player)**

function win(player):

FOR EACH row from 0 to 2:

IF board[r][0] == board[r][1] == board[r][2] == player:

RETURN True

FOR EACH column c from 0 to 2):

IF board[0][c] == board[1][c] == board[2][c] == player:

RETURN True

IF board[0][0] == board[1][1] == board[2][2] == player:

RETURN True

else if board[2][0] == board[1][1] == board[0][2] == player:

RETURN True

[2:19pm]

**makeMove()**

function makeMove():

win\_move = winningMove() // Check if there's a winning move available

IF win\_move != -1:

RETURN win\_move // Return the winning move if available

// If no winning move, check for a block move

block\_move = blockMove()

IF block\_move != -1:

RETURN block\_move // Return the blocking move if available

// If neither winning nor blocking moves available, pick a random slot

available\_moves = []

FOR each slot in board:

IF slot is empty:

add slot to available\_moves list

RETURN random.choice(available\_moves) // Return a random available move

[2:45pm]

**blockMove()**

function winningMove():

FOR each i in range(3):

IF board[i][0] == board[i][1] == ' x ' and board[i][2] NOT IN ('x', 'o'):

RETURN i \* 3 + 3 // Third slot in row

ELIF board[i][0] == board[i][2] == ' x ' and board[i][1] NOT IN ('x', 'o'):

RETURN i \* 3 + 2 // Second slot in row

ELIF board[i][1] == board[i][2] == ' x ' and board[i][0] NOT IN ('x', 'o'):

RETURN i \* 3 + 1 // First slot in row

ELIF board[0][i] == board[1][i] == ' x ' and board[2][i] NOT IN ('x', 'o'):

RETURN 7 + i // Last slot in column

ELIF board[0][i] == board[2][i] == ' x ' and board[1][i] NOT IN ('x', 'o'):

RETURN 4 + i // Second slot in column

ELIF board[1][i] == board[2][i] == ' x ' and board[0][i] NOT IN ('x', 'o'):

RETURN 1 + i // First slot in column

// Check for wins in diagonals

IF board[0][0] == board[1][1] == ' x ' and board[2][2] NOT IN ('x', 'o'):

RETURN 9 // Diagonal top-left to bottom-right

ELIF board[0][0] == board[2][2] == ' x ' and board[1][1] NOT IN ('x', 'o'):

RETURN 5 // Diagonal top-right to bottom-left

ELIF board[1][1] == board[2][2] == ' x ' and board[0][0] NOT IN ('x', 'o'):

RETURN 1 // Diagonal bottom-right to top-left

ELIF board[2][0] == board[1][1] == ' x ' and board[0][2] NOT IN ('x', 'o'):

RETURN 3 // Diagonal bottom-left to top-right

ELIF board[2][0] == board[0][2] == ' x ' and board[1][1] NOT IN ('x', 'o'):

RETURN 5 // Middle slot in diagonal

ELIF board[1][1] == board[0][2] == ' x ' and board[2][0] NOT IN ('x', 'o'):

RETURN 7 // Middle slot in diagonal

RETURN -1 // Return -1 if no winning move found

[3:30pm]

**winningMove()**

function winningMove():

FOR each i from 0 to 2:

IF board[i][0] == board[i][1] == 'o' and board[i][2] NOT IN ('x', 'o'):

RETURN i \* 3 + 3 // Third slot in row

ELIF board[i][0] == board[i][2] == 'o' and board[i][1] NOT IN ('x', 'o'):

RETURN i \* 3 + 2 // Second slot in row

ELIF board[i][1] == board[i][2] == 'o' and board[i][0] NOT IN ('x', 'o'):

RETURN i \* 3 + 1 // First slot in row

ELIF board[0][i] == board[1][i] == 'o' and board[2][i] NOT IN ('x', 'o'):

RETURN 7 + i // Last slot in column

ELIF board[0][i] == board[2][i] == 'o' and board[1][i] NOT IN ('x', 'o'):

RETURN 4 + i // Second slot in column

ELIF board[1][i] == board[2][i] == 'o' and board[0][i] NOT IN ('x', 'o'):

RETURN 1 + i // First slot in column

// Check for wins in diagonals

IF board[0][0] == board[1][1] == 'o' and board[2][2] NOT IN ('x', 'o'):

RETURN 9 // Diagonal top-left to bottom-right

ELIF board[0][0] == board[2][2] == 'o' and board[1][1] NOT IN ('x', 'o'):

RETURN 5 // Diagonal top-right to bottom-left

ELIF board[1][1] == board[2][2] == 'o' and board[0][0] NOT IN ('x', 'o'):

RETURN 1 // Diagonal bottom-right to top-left

ELIF board[2][0] == board[1][1] == 'o' and board[0][2] NOT IN ('x', 'o'):

RETURN 3 // Diagonal bottom-left to top-right

ELIF board[2][0] == board[0][2] == 'o' and board[1][1] NOT IN ('x', 'o'):

RETURN 5 // Middle slot in diagonal

ELIF board[1][1] == board[0][2] == 'o' and board[2][0] NOT IN ('x', 'o'):

RETURN 7 // Middle slot in diagonal

RETURN -1 // Return -1 if no winning move found

[3:45pm]

**trickMove()**

FUNCTION trickMove():

FOR i FROM 1 TO 9:

IF validMove(i):

temp\_board = COPY\_BOARD(board)

trap = 0

temp\_board[(i - 1) // 3][(i - 1) % 3] = ' o ' // Simulate the move

FOR j FROM 0 TO 2:

// Logic to check rows and columns for potential forks

IF temp\_board[j][0] == temp\_board[j][1] == ' o ' AND temp\_board[j][2] NOT IN (' x ', ' o ') THEN

trap = trap + 1

ELSE IF temp\_board[j][0] == temp\_board[j][2] == ' o ' AND temp\_board[j][1] NOT IN (' x ', ' o ') THEN

trap = trap + 1

ELSE IF temp\_board[j][1] == temp\_board[j][2] == ' o ' AND temp\_board[j][0] NOT IN (' x ', ' o ') THEN

trap = trap + 1

ELSE IF temp\_board[0][j] == temp\_board[1][j] == ' o ' AND temp\_board[2][j] NOT IN (' x ', ' o ') THEN

trap = trap + 1

ELSE IF temp\_board[0][j] == temp\_board[2][j] == ' o ' AND temp\_board[1][j] NOT IN (' x ', ' o ') THEN

trap = trap + 1

ELSE IF temp\_board[1][j] == temp\_board[2][j] == ' o ' AND temp\_board[0][j] NOT IN (' x ', ' o ') THEN

trap = trap + 1

// Check diagonals for potential forks

IF temp\_board[0][0] == temp\_board[1][1] == ' o ' AND temp\_board[2][2] NOT IN (' x ', ' o ') THEN

trap = trap + 1

ELSE IF temp\_board[0][0] == temp\_board[2][2] == ' o ' AND temp\_board[1][1] NOT IN (' x ', ' o ') THEN

trap = trap + 1

ELSE IF temp\_board[1][1] == temp\_board[2][2] == ' o ' AND temp\_board[0][0] NOT IN (' x ', ' o ') THEN

trap = trap + 1

ELSE IF temp\_board[2][0] == temp\_board[1][1] == ' o ' AND temp\_board[0][2] NOT IN (' x ', ' o ') THEN

trap = trap + 1

ELSE IF temp\_board[2][0] == temp\_board[0][2] == ' o ' AND temp\_board[1][1] NOT IN (' x ', ' o ') THEN

trap = trap + 1

ELSE IF temp\_board[1][1] == temp\_board[0][2] == ' o ' AND temp\_board[2][0] NOT IN (' x ', ' o ') THEN

trap = trap + 1

IF trap >= 2 THEN

RETURN i // Return the position that creates a two-win trap

RETURN -1 // Return -1 if no fork found

[5:30pm]

**playAgain()**

Function playAgain()

Data = INPUT Would you like to play again? y/n:.lower()

IF Data == ‘y’:

RETURN True

IF Data == ‘n’:

RETURN FALSE

ELSE:

PRINT “ERROR”

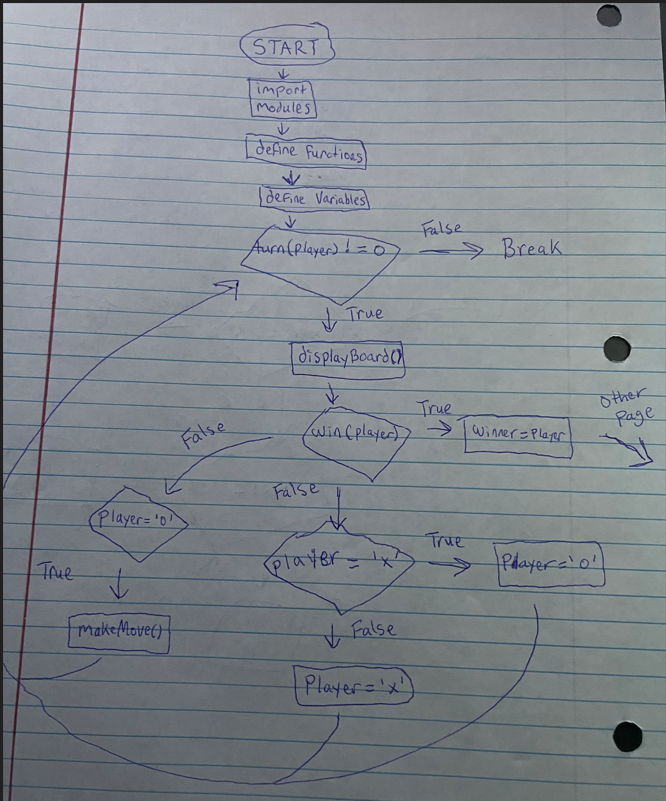
[6:00pm] I will now end today as I finished all the planning, requirements and pseudocodes for the functions that I will use in my game and I feel as though I made great progress today towards it. Tomorrow I will aim to create the game loop flowchart that will be used for my game.

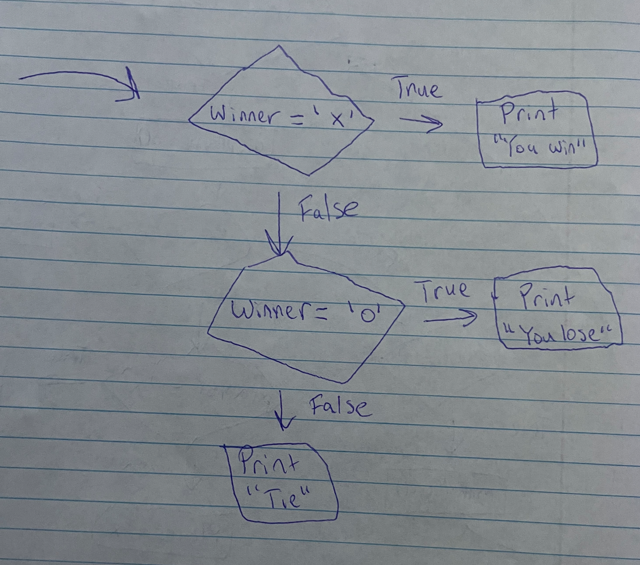
November 24 2023

Flowcharts:

[2:00pm]

Game Loop:





[3:30pm] Today, I just worked on the game loop and how it will overall work and tomorrow, I will start to implement all my planning of pseudocodes and flowcharts and see how my game turns out.

November 25 2023

Implementing code:

[1:05pm] Firstly, I changed the instructions on the start screen as it states it's a game meant for two players and not against an AI. I then added my validMove() function within the turn function to further improve it so that it can detect if a spot is already taken and or if it is within the valid range of 1-9 to account for user input error. When I tested out the new function everything was working fine however, when I attempted to type 0 to quit the program it wasn’t quitting anymore. To fix this issue, I had to change the if statement in my validMove() function so that it was equal to or greater than 0 instead of 1. The enter 0 to quit is working as intended again.

# checks to see if input is not taken already and within range

def validMove(move):

# convert move to board form

r = (move - 1) // 3

c = (move - 1) % 3

# Check if the move is within range and the position is empty

if 0 <= move <= 9 and board[r][c] not in (' x ', ' o '):

return True

else:

return False

[1:15pm] Next, I will add a global winner variable and the function to check to see if there is a winner at any given point during the game of tic tac toe. It uses range functions to check for wins within the rows and columns and if statements to check if there is a winner for diagonals. I then added the function to the game loop so it may break out of the game if someone wins and it is working as intended.

while turn(player) != 0:

displayBoard()

# stop game and print winner if there is one

if win(player):

winner = player

if winner == ' x ':

print(Fore.GREEN + "You won!" + Fore.RESET)

break

elif winner ==' o ':

print(Fore.RED + "You lose!" + Fore.RESET)

break

if player == ' x ': # switch player between moves

player = ' o '

else:

player = ' x '

I forgot to include a way for it to detect if the board is full (tie) so as of this moment it gets stuck in a constant loop and doesn’t break out. So, I created a function to return true if all the cells are taken up by x’s and o’s and I defined it as isFull() and added it after it checks for the winner but then I was having an issue where it would never get activated and still got stuck. After debugging the code, I realized it would never get activated because it was under the iswin() condition so it wouldn’t activate. I fixed the indentation around the function, however, I noticed it still wasn’t working, there was a problem that is within my isfull() function. I noticed I didn’t add a space within the if cell != ‘ x ‘ and cell != ‘ o ‘ and after using the correct spacing it was working as intended, I also don’t like how the game loop looks so I will change it so that it prints the winner after everything else.

**isFull(board)**

Function isFull(board):

For each row in board:

For each cell in row:

If cell != ‘x’ and cell != ‘o’

Return False

Return True

def isFull(board):

for row in board:

for cell in row:

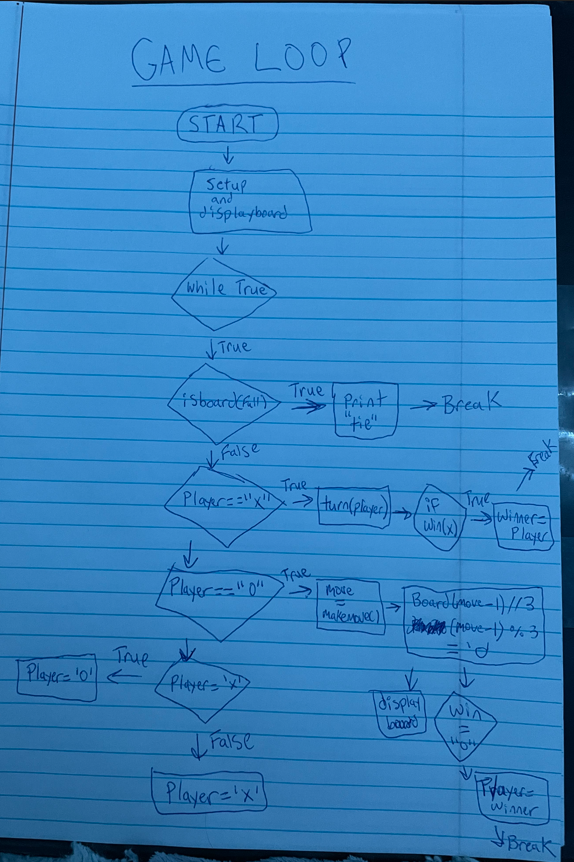
if cell != 'x' and cell != 'o':

return False

return True

[4:00pm] Now that I have a fully functional two-player game of tic tac toe, it is now time to add the AI for the second player instead of another user input. When I brainstorm about how I will add in the AI, I come into an issue where it will never get off the user input (turn) and never allow the AI to play due to the while loop, so I will have to make some changes and improvements to my flowchart.

**Updated flowchart:**



[5:30pm] After putting in my new game loop, and after I fixed an indentation error I had on the trickmove() function, I was really surprised at how well my game was working out. Everything was working good, the computer AI was finding a way to go for winning position, block position, and trap the player very well. However, one thing that broke again was the fact that I couldn’t enter 0 to quit the program all together. It was an easy fix by changing the exit to the exit() function and putting a print before it uses the exit function and it is working perfectly once again.

loc = int(data) # the location of the move

if loc == 0:

print("Thank you for playing!")

exit()

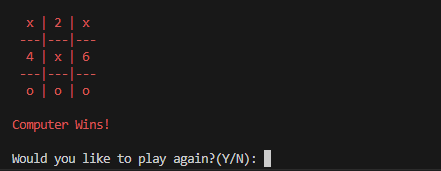
[5:45pm] Lastly, I must add in my playAgain() function and ensure it is resetting all the global variables and the board so that a new game can be played at the users will. To do this, I will add an outer loop to my game loop so it can easily loop back to the start if the user chooses so in the data input. I had an issue where I could enter something that wasn’t yes or no and it would still break out of the loop so I had to create a loop in my playagain() function so that it could get out unless the user chooses yes or no. After that whenever I would playagain() the computer would end up taking the first turn and that’s not how I want it to be so to fix this I added a player = x to reset the global variable back to the player first.

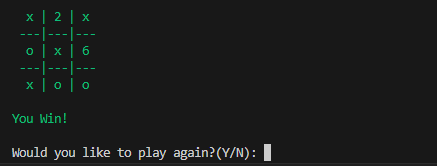
[6:30pm] After that, everything seems to be implemented into my code and I will be stopping for today so that I can do some crash tests and change the looks of my game to make it look more appealing on the eyes tomorrow.

November 26 2023

Crash Test & Color/Looks:

[2:00pm] While testing through my game, I had an issue where after I won / lost after using the playagain() function, the next game if I were to tie versus the AI it would send both messages of tied and you won. I had to reset the global winner variable at the start of the loop to fix this. Everything seems to be stable and running how it should and I can’t crash the program. I will now add color and spacing to my liking to make the game look nicer with color and so it’s not as compact.





Conclusion

[3:00pm] I am very glad I got to do this project as I felt like it always had me thinking about what I could do better and thinking about the function code I would have to make for the game, especially the trickMove() function. I feel as though it helped me work on my AI skills and coding huge chunks of if statements of how the AI should react and it helped me with my complex functions such as converting the tic tac toe board. Overall, I think this was a great project and it was a lot of fun to bring my thoughts to life and think about how the AI should react to certain moves.