



CSC1024 PROGRAMMING PRINCIPLES

# **A Computer Versus Human Tic-Tac-Toe Computer Game**

LIM XIWEI

21045596

2 December 2022.

Presentation video web link:

<https://www.youtube.com/watch?v=6drj89VOP2I>

```

round_count = 1
again = True
f = open("logfile_21045596.txt", "w")

while again:
    f = open("logfile_21045596.txt", "a")
    f.write("\nRound " + str(round_count) + "\n")

    board = create_board()
    show_board()

    player = choose_first_player()
    c_mark, h_mark = assign_mark()
    #c-mark = computer's piece, h_mark, user's piece

    print("Tic-Tac-Toe Game Start")
    start(board, player, c_mark, h_mark)

    again = play_again()
    round_count += 1

f.close()

```

- Variable round\_count is set to 1. This variable determines the current round number.
- Open the file "logfile\_21045596.txt" and overwrite the content.
- Open the file again in append mode.
- Write the current round count in the text file; if the user plays again, it is the second round.
- Creates and prints out the board.
- Chooses the first player.
- Choosing the piece for the computer and user.
- Calls the start(board, player, c\_mark, h\_mark) function.
- After the start(board, player, c mark, h mark) function has been completed, the play\_again() function is called.
- Closes the text file.

```
import random
```

```
def create_board():  
    board = []  
    for i in range(3):  
        row = []  
        for j in range(3):  
            row.append('-')  
        board.append(row)  
    return board
```

```
def show_board():  
    for row in board:  
        for item in row:  
            print(item, end = " ")  
        print()
```

## What create\_board() does:

- For each number in range 3, a sublist called row is created and appends the character hyphen three times before the board list appends the row list.

```
row_list = ['-']  
row_list = ['- ', '-']  
row_list = ['- ', '- ', '-']  
board_list = [['-', '-', '-']]  
row_list = ['-']  
row_list = ['- ', '-']  
row_list = ['- ', '- ', '-']  
board_list = [['-', '-', '-'], ['- ', '- ', '-']]  
row_list = ['-']  
row_list = ['- ', '-']  
row_list = ['- ', '- ', '-']  
board_list = [['-', '-', '-'], ['- ', '- ', '-'], ['- ', '- ', '-']]
```

## What show\_board() does:

- Each item in the sublist is printed on a single line, followed by the items in the next sublist on a new line.

```
  _ _ _  
  
  _ _ _  
  
  _ _ _
```

```
def choose_first_player():
    player_list = ['Computer', 'Player']
    i = random.randint(0,1)
    player = player_list[i]
    return player

def assign_mark():
    mark_list = ['X', 'O']
    i = random.randint(0,1)
    c_mark = mark_list[i]

    #check if computer's and player's piece is same
    h_mark = 'X'
    if h_mark == c_mark:
        h_mark = 'O'

    return c_mark, h_mark
```

## What choose\_first\_player() does:

- player\_list is created and stores the items: 'Computer' and 'Player'.
- The variable i will hold a value of either 1 or 0.
- The value from player\_list with index i is assigned to the variable player.
- The variable player is returned.

## What assign\_mark() does:

- mark\_list is created and stores the items: 'X' and 'O'.
- The variable i will hold a value of either 1 or 0.
- The value from mark\_list with index i is assigned to the variable c\_mark.
- The variable h\_mark is assigned the value 'X'. If h\_mark equals c\_mark, the h\_mark value is changed to 'O'.
- The values of c\_mark and h\_mark are returned.

```

def start(board, player, c_mark, h_mark):
    turn = 1
    ongoing = True

    while ongoing:
        if player == 'Computer':
            computer_move('Computer', 1, c_mark, turn)
        else:
            human_move('Player', h_mark, turn)

        show_board()

        #checking if win or tie condition is met
        if is_player_win(player):
            print(player, "win!")
            ongoing = False
        elif is_board_filled():
            print("Game Over. It's a tie.")
            ongoing = False

        turn += 1
        player = swap_turn(player)

```

## What start(board, player, c\_mark, h\_mark) does:

- Variable turn, which is used to keep a record of the move count, is initialized 1.
- Identifies the current player and invokes the appropriate function
- Prints out the updated board
- Checks to see if any player has won or if the board is filled. If any of them is True, the while loop will be terminated.
- Variable turn increment by 1.
- The current player is switched.

```
def computer_move(player, text, mark, turn):  
    row = random.randint(0,2)  
    col = random.randint(0,2)  
  
    #prevent repetitive text when computer fix the piece position again  
    if text == 1:  
        print("Waiting for computer...")  
  
    mark_position(row, col, mark, player, turn)
```

## What computer\_move(player, text, mark, turn) does:

- row and col will each hold an integer value between 0 and 2.
- If the parameter called text equals 1, the sentence will be printed. (To avoid print repetitively)
- The function mark\_position(row, col, mark, player, turn) is called. The values stored in row, col, mark, player and turn are passed to the function as parameters.

```
def human_move(player, mark, turn):  
    print("Player's turn, you are \"", mark, "\"")  
    error = True  
  
    #handle input exceed row/col available error  
    while error:  
        row = int(input("Enter row number to place your piece: "))  
        col = int(input("Enter col number to place your piece: "))  
        if row > 3 or col > 3:  
            print(" «Exceed available row or column number.» ")  
        else:  
            error = False  
  
    mark_position(row-1, col-1, mark, player, turn)
```

## What human\_move(player, mark, turn) does:

- Prints a statement informing the user that it is their turn and their piece.
- While the variable error is true, prompt the user to provide row and column numbers. If any of the row and column numbers entered by the user exceeds the existing number of rows and columns, print the error message; otherwise, the loop condition is set to False.
- The function mark\_position(row, col, mark, player, turn) is called.



```
def mark_position(row, col, mark, player, turn):
    #check if position is filled, if yes prompt current player to fix a position again
    if board[row][col] == '-':
        log_file(player, row, col, mark, turn)
        board[row][col] = mark
    elif player == 'Player':
        print(" «Error, position filled.» ")
        human_move('Player', mark, turn)
    else:
        computer_move('Computer', 0, mark, turn)
```

## What mark\_position(row, col, mark, player, turn) does:

- If the position specified by the user/computer is already marked, the current player is identified, and the function for human move/computer move depending on the current player is called, prompting the player to refix a position. If the position is not marked, the current player's piece replaces the item in the list called board specified by the indexes row and col. The function log\_file(player, row, col, mark, turn) is also called.



```
def log_file(player, row, col, mark, turn):  
    if player == "Computer":  
        player = 'C'  
    else:  
        player = 'H'  
    f.write(str(turn) + ", " + str(player) + ", " + str(row+1) + ", " + str(col+1) + ", " + str(mark) + "\n")
```

## What log\_file(player, row, col, mark, turn) does:

- If the current player is 'Computer,' the variable player holds the value 'C', otherwise, it holds the value 'H'.
- Writes multiple strings to the log file.

```

def is_player_win(player):
    win = None
    n = len(board)

    if player == 'Computer':
        mark = c_mark
    else:
        mark = h_mark

    #checking rows
    for i in range(n):
        win = True
        for j in range(n):
            if board[i][j] != mark:
                win = False
                break
        if win:
            return win

    #checking columns
    for i in range(n):
        win = True
        for j in range(n):
            if board[j][i] != mark:
                win = False
                break
        if win:
            return win

```

## What is\_player\_win(player) does:

- The variable win has a null value.
- The variable n is given the value of the number of items in the list called board.
- If the current player is 'Computer,' the variable mark holds the value of the variable c\_mark: otherwise, it holds the value of the variable h\_mark.
- Before checking the coordinates, win is always set to True. After guaranteeing that the win condition cannot be met, it is set to False.
- Check if each item in each row on the printed board is filled with the mark. If win is True after the for loop, the return statement quits the current function and returns the value of the variable win; otherwise, the rest of the function code continues to execute.
- Check if each item in each column on the printed board is filled with the mark.

```

#checking diagonals
#Leading diagonal
win = True
for i in range(n):
    if board[i][i] != mark:
        win = False
        break
if win:
    return win

#Anti-diagonal
win = True
for i in range(n):
    if board[i][n - 1 - i] != mark:
        win = False
        break
if win:
    return win
return False

```

- Check if each item in the leading diagonal on the printed board is filled with the mark.
- Check if each item in the anti-diagonal on the printed board is filled with the mark.

Value [i]	Value [n-1-i]	Coordinates
0	2	[0][2]
1	1	[1][1]
2	0	[2][0]

```
def is_board_filled():  
    for row in board:  
        for item in row:  
            if item == '-':  
                return False  
    return True
```

```
def swap_turn(player):  
    if player == 'Computer':  
        player = 'Player'  
    else:  
        player = 'Computer'  
    return player
```

## What is\_board\_filled() does:

- Checks whether at least one of the items in all the sublists in the list called board equals '-'. If yes, it returns False; otherwise, it returns True.

## What swap\_turn(player) does:

- If the value of the parameter called player equals 'Computer', the value of the variable player is changed to 'Player'. If it is not, the value of the variable player is changed to 'Computer'.
- The variable player is returned.

```
def play_again():  
    invalid = True  
    while invalid:  
        retry = input("Do you wish to play again? [Y/N]:")  
  
        if retry.upper() == 'Y':  
            return True  
        elif retry.upper() == 'N':  
            print("Game end.")  
            return False  
        else:  
            print("《Incorrect input, please try again.》 ")
```

## What play\_again() does:

- Ask the user if they want to play again. If yes, the function returns True; otherwise, it returns False.
- If the user enters an incorrect option, the while loop condition remains True, prompting the user to enter the correct option again.

# Thank You!

By Lim Xiwei

```
- - -  
- - -  
- - -
```

```
Tic-Tac-Toe Game Start  
Waiting for computer...
```

```
- - -  
- - 0  
- - -
```

```
Player's turn, you are " X "  
Enter row number to place your piece: 1  
Enter col number to place your piece: 1
```

```
X - -  
- - 0  
- - -
```

```
Waiting for computer...  
X - 0
```

```
- - 0  
- - -
```

```
Player's turn, you are " X "  
Enter row number to place your piece: 2  
Enter col number to place your piece: 2
```

```
X - 0  
- X 0  
- - -
```

```
Waiting for computer...  
X - 0
```

```
- X 0  
- - 0
```

```
Computer win!  
Do you wish to play again? [Y/N]:_
```