NEW HORIZON
COLLEGE OF ENGINEERING
Autonomous College Permanently Affiliated to VTU, Approved by AICTE & UGC
Accredited by NAAC with 'A' Grade, Accredited by NBA

A

MINI PROJECT REPORT

ON

"BOARD GAME: OTHELLO"

Submitted in the partial fulfilment of the requirements in the 3rd semester of

BACHELOR OF ENGINEERING

IN

INFORMATION SCIENCE AND ENGINEERING

BY

ANARGHYA RAO – 1NH19IS012

FOR

COURSE NAME: MINI PROJECT

COURSE CODE: 19ISE391

Under the guidance of,

Prof. B. Mounica
Professor,
Dept. of ISE, NHCE

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

NEW HORIZON COLLEGE OF ENGINEERING

(Autonomous College Permanently Affiliated to VTU, Approved by AICTE, Accredited by NAAC with 'A' Grade & NBA)

Ring Road, Bellandur Post, Near Marathahalli,

Bengaluru-560103, INDIA

www.newhorizonindia.edu

NEW HORIZON
COLLEGE OF ENGINEERING
Autonomous College Permanently Affiliated to VTU, Approved by AICTE & UGC
Accredited by NAAC with 'A' Grade, Accredited by NBA

CERTIFICATE

Certified that the project work entitled "BOARD GAME: OTHELLO" carried out by Mr./Ms. ANARGHYA RAO, bearing USN, 1NH19IS012 a bonafide student of 3th semester in partial fulfilment for the award of Bachelor of Engineering in Information Science & Engineering of the Visveswaraiah Technological University, Belagavi during the year 2020-21. It is certified that all corrections / suggestions indicated for Internal Assessment have been incorporated. The project report has been approved as it satisfies the academic requirements in respect of Mini Project work prescribed for the said Degree.

Name & Signature of Guide	Name & Signature of HOD	Name & Signature of Principal
Mrs. B Mounica	Dr.Anandhii R.J.	Dr.Manjunatha

Examiners:

	Name	**Signature**
1.	……	……
2.	……	……

NEW HORIZON COLLEGE OF ENGINEERING

Autonomous College, Affiliated to VTU | Approved by AICTE New Delhi & UGC
Accredited by NAAC with 'A' Grade & Accredited by NBA

50 Golden Jubilee YEARS OF EXCELLENCE IN EDUCATION

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

CERTIFICATE ON PLAGIARISM CHECK

1	Name of the Student	ANARGHYA RAO
2	USN	1NH19IS012
3	Course	UG
4	Department	ISE
5	Mini Project/Project Report / Internship Report/Seminar Report/ Paper Publication/Ph.D Thesis	Mini Project
6	Title of the Document	BOARD GAME OTHELLO
7	Similar Content(%)identified	3%
8	Acceptable maximum limit (%) of similarity	30%
9	Date of Verification	13.03.2021
10	Checked by (Name with signature)	Dr.Mangai Velu
11	Specific remarks, if any :	1st Attempt

We have verified the contents as summarized above and certified that the statement made above are true to the best of our knowledge and belief.

Research coordinator

Abstract

Salient features of the project are: -

1) The code is written using the popular programming language, python.
2) The game is played on python shell.
3) Code of program is written such that all the rules of the game are followed properly.
4) The gameplay is user-friendly, hassle-free and easy-to-follow.
5) Besides the option to play against another player, the game also provides the option to play against a competent computer opponent.
6) Instructions on how to play the game are also provided, thus allowing new players to easily grasp the game.
7) The game helps build up a person's strategic thinking skills.

Acknowledgement

Any project is a task of great enormity and it cannot be accomplished by an individual without support and guidance. I am grateful to a number of individuals whose professional guidance and encouragement has made this project completion a reality.

I have a great pleasure in expressing my deep sense of gratitude to the beloved Chairman Dr. Mohan Manghnani for having provided me with a great infrastructure and well-furnished labs.

I take this opportunity to express my profound gratitude to the Principal Dr. Manjunatha for his constant support and management.

I am grateful to Dr. R J Anandhi, Professor and Head of Department of ISE, New Horizon College of Engineering, Bengaluru for her strong enforcement on perfection and quality during the course of my project work.

I would like to express my thanks to the guide Mrs. B. Mounica,, Professor, Department of ISE, New Horizon College of Engineering, Bengaluru who has always guided me in detailed technical aspects throughout my project.

I would like to mention special thanks to all the Teaching and Non-Teaching staff members of Information Science and Engineering Department, New Horizon College of Engineering, Bengaluru for their invaluable support and guidance.

ANARGHYA RAO

1NH19IS012

TABLE OF CONTENTS

List of tables and figures

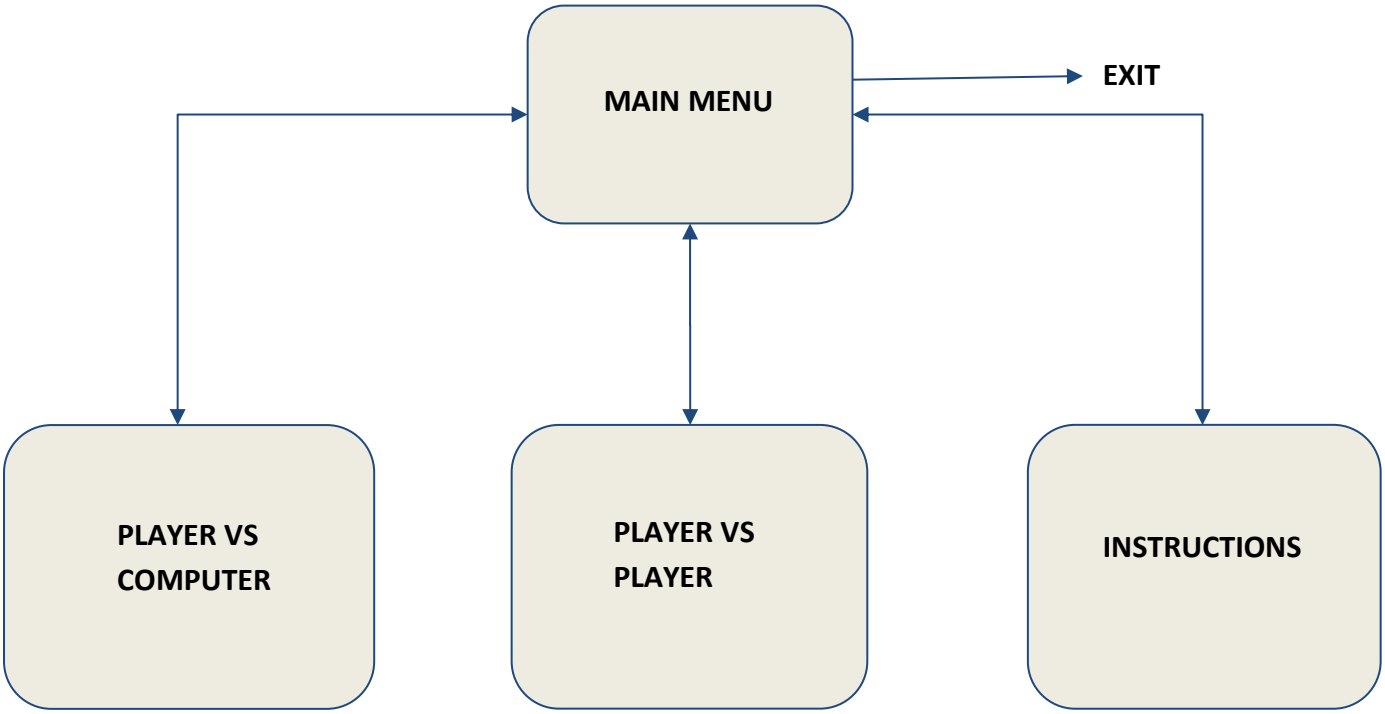

Fig3.1: Block diagram

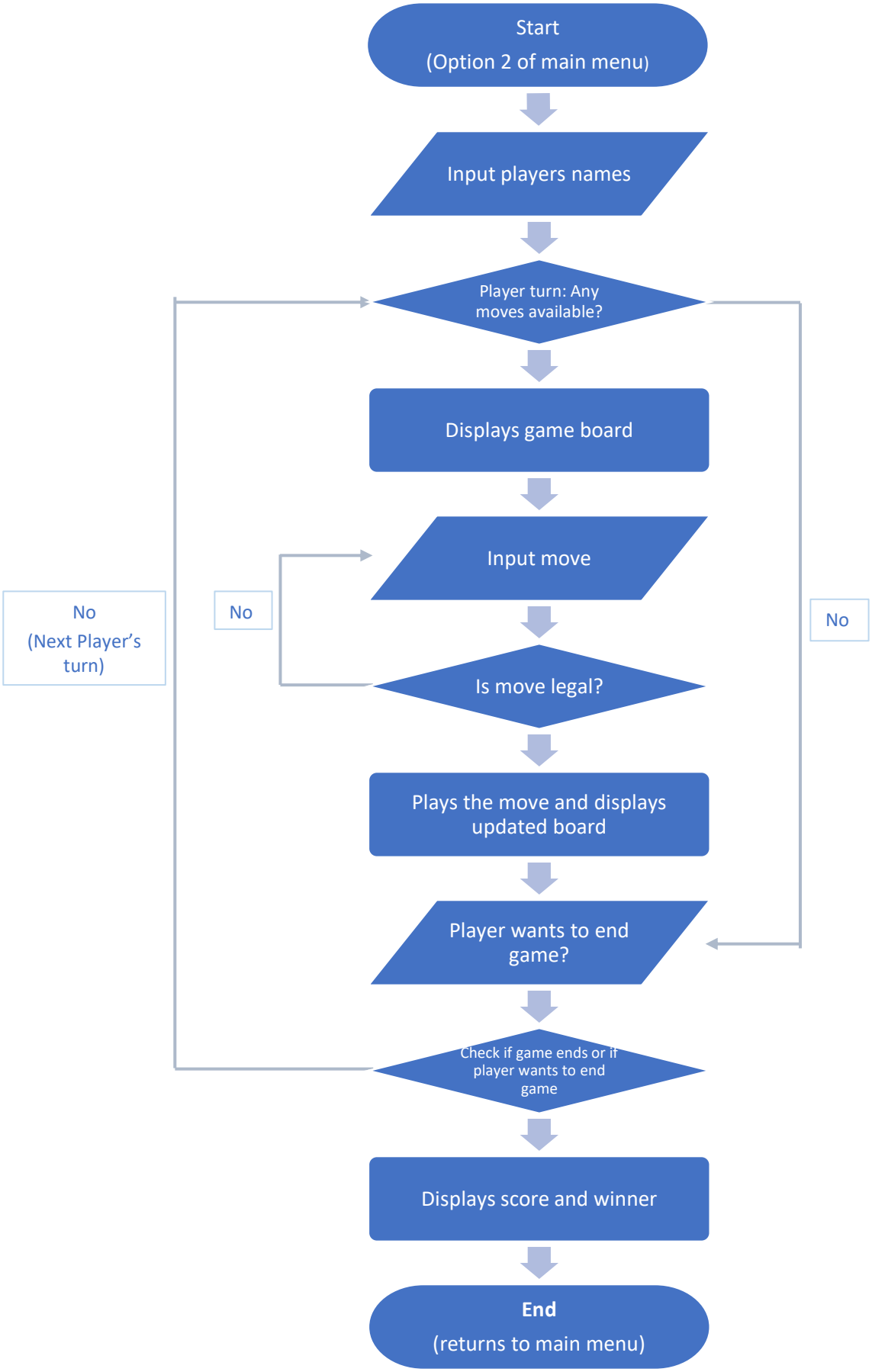

Fig3.2: Player VS Player module Flowchart

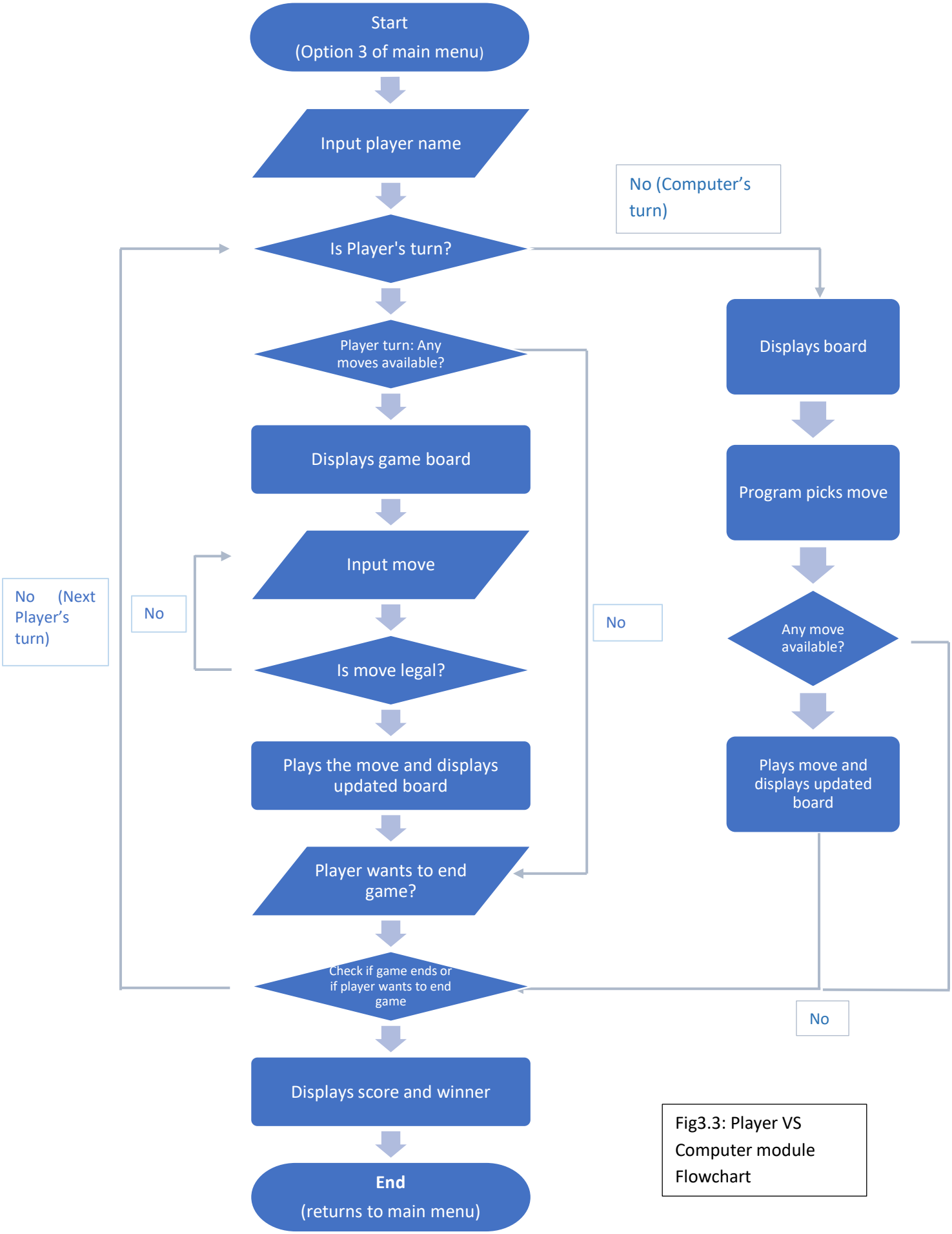

Fig3.3: Player VS Computer module Flowchart

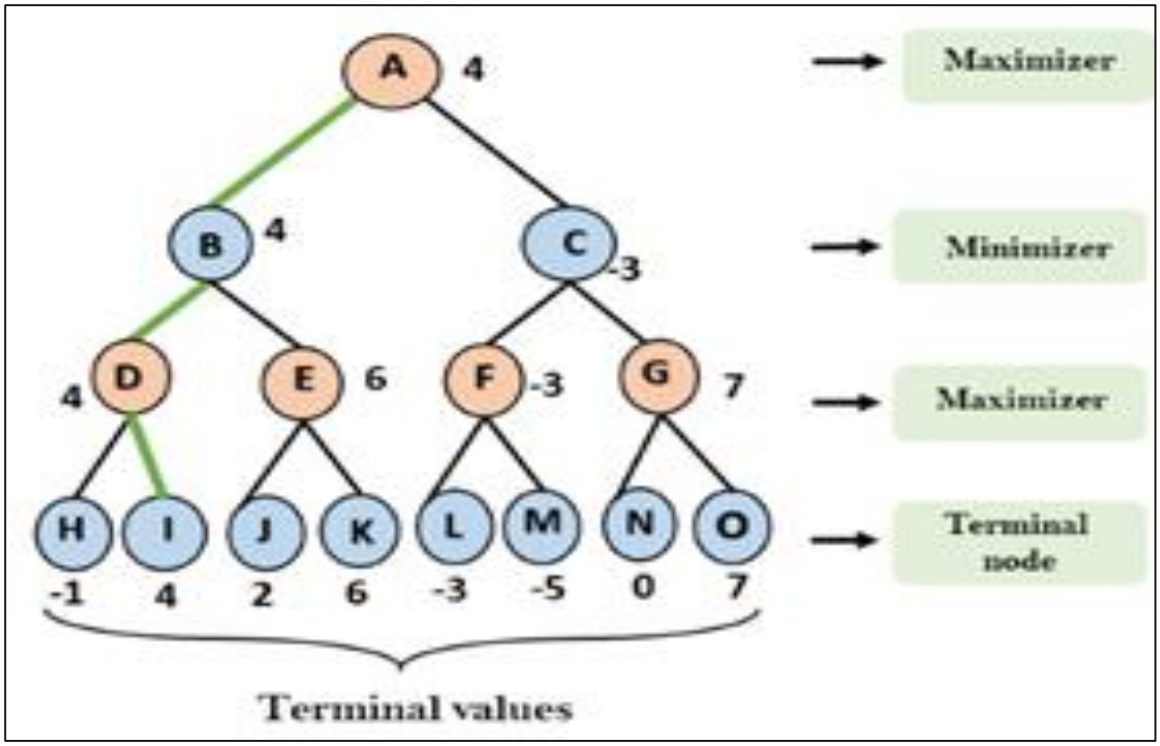

Fig3.4: Minimax algorithm in tree format

Game number	Won / Lost	Computer's score	Player's score
Game 1	Lost	41	23
Game 2	Won	8	56
Game 3	Won	30	34
Game 4	Lost	49	15
Game 5	Won	28	36

Fig4.1: Computer VS Player

CHAPTER 1: INTRODUCTION

1.1 Motivation of Project

Learning and practicing programming through coding for games is a wonderful way to improve one's coding skills, since it provides a unique opportunity to learn while also having fun. Coding for a game is the perfect challenge for a beginner, such as me, to test out their skills without the challenge being too advanced. Thus, the decision to choose this topic.

Motivation for the project idea "Board game: Othello" primarily came from taking the NPTEL course *"The Joy of Computing using Python"*, where a majority of the coding was taught by means of coding for games. The inspiration for the "Player VS Computer" module of my project, where the program selects the best possible move on its own and thus plays against the human player, was from the short Coursera course "*Machine Learning for All"*, where we were introduced to AI through the Chess-playing Computer "Deep Blue" and the Go-playing program "AlphaGo". Besides this, owning and having enjoyed playing the board game Othello multiple times myself, I was naturally inspired to try my hand at coding for it.

1.2 Problem Statement

1.2.1 Othello: The Basics

Othello is a 2-player strategy board game. The board is an 8x8 board, i.e., has 8 rows and 8 columns and so consists of 64 squares. The game also comes with 64 two-colour sided (black and white) discs (or "coins"), 32 discs for each player. Each player picks one of the colours and uses that for the remainder of the game.

Fig1.1: The Board, discs and set-up

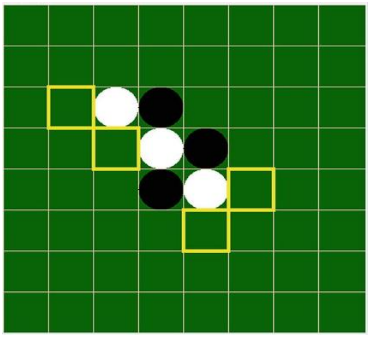

Fig1.2: E.g., all possible moves for black

The game is played turn-wise and every move is an attempt to outflank the opponent's discs and thus flip them to face our colour. "Outflanking" means to place a disc such that the player's discs are on both ends of a row, column or diagonal of the opponent's discs. These can then be flipped to face our colour.

The aim of the game is to have higher number of discs of your colour on the board than your opponent at the end of the game.

1.2.2 Problem definition

The project is an effort to translate this physical two-player board game "Othello" into a python program. The program must be written such that all the rules and particulars of the game are followed and accounted for.

CHAPTER 2: SYSTEM REQUIREMENT

2.1 Software/Hardware used

2.1.1 Software:

The program was built using the following software: -

1) Python 3.8.5 and Python Shell
2) Operating System (OS): Microsoft Windows 10

2.1.2 Hardware:

The program was built using the following hardware: -

1) Processor: Intel CORE i5 8th gen
2) RAM: 8.00 GB
3) Laptop used: DELL

CHAPTER 3: SYSTEM DESIGN

3.1 Modules

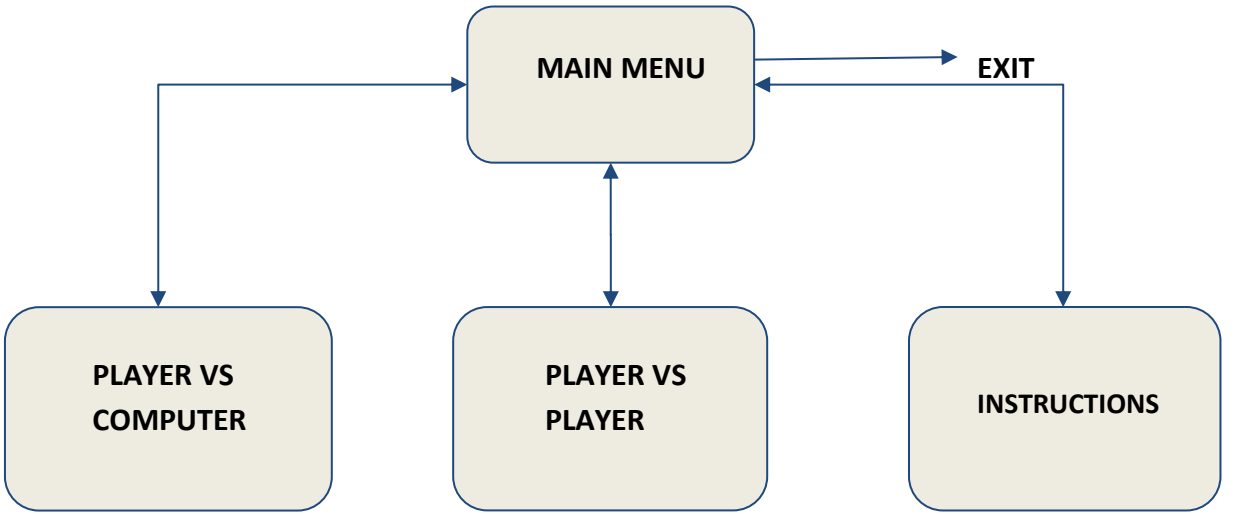

Fig3.1: Block diagram

The program is divided into four modules based on functionality: -

1) Main menu module:

   This is the first module the user sees when the program is executed. It provides the user with options for various actions. Every other module of the program is accessed through this module. After the end of execution of the other module, the control is returned to the main menu module.

2) Instructions module:

   This module is composed of a text file containing the instructions on how to play the game, the rules, etc..

3) Player VS Player module:

   This module contains the code that implements the "Player VS Player" version of the gameplay. In this gameplay, two Players play against each other. All the rules and particulars of the game are followed in the gameplay. The game board is generated and moves from the players are accepted to play the game.

4) Player VS Computer module:

   This module contains the code that implements the "Player VS Computer" version of the gameplay. In this gameplay, one Player plays against the Computer, i.e., program-generated moves. All rules and particulars are followed. Game board is generated and moves of Player are accepted to play the game

3.2 Architecture [Flowchart]

Player VS Player module:

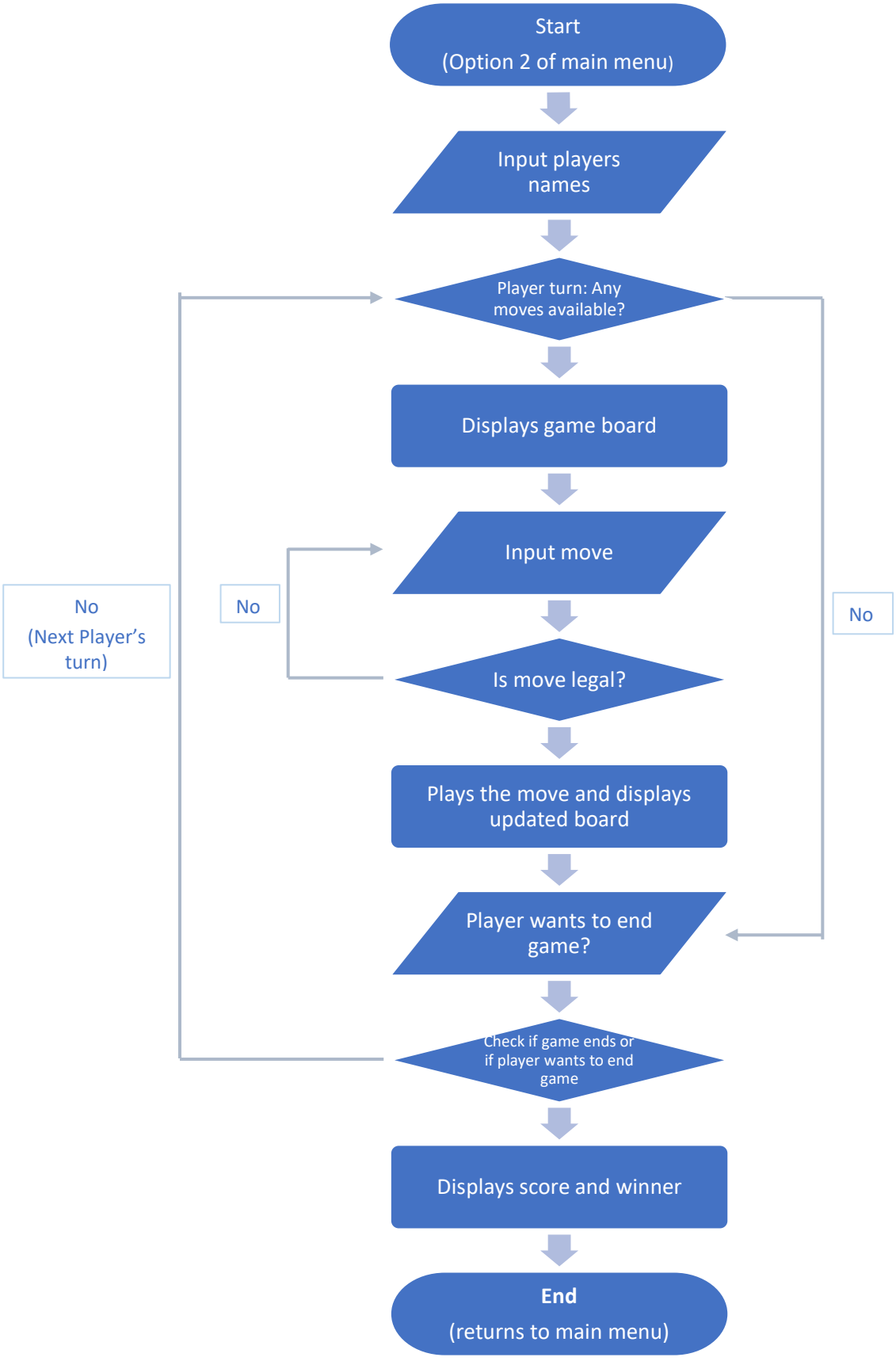

Fig3.2: Player VS Player

Player VS Computer module:

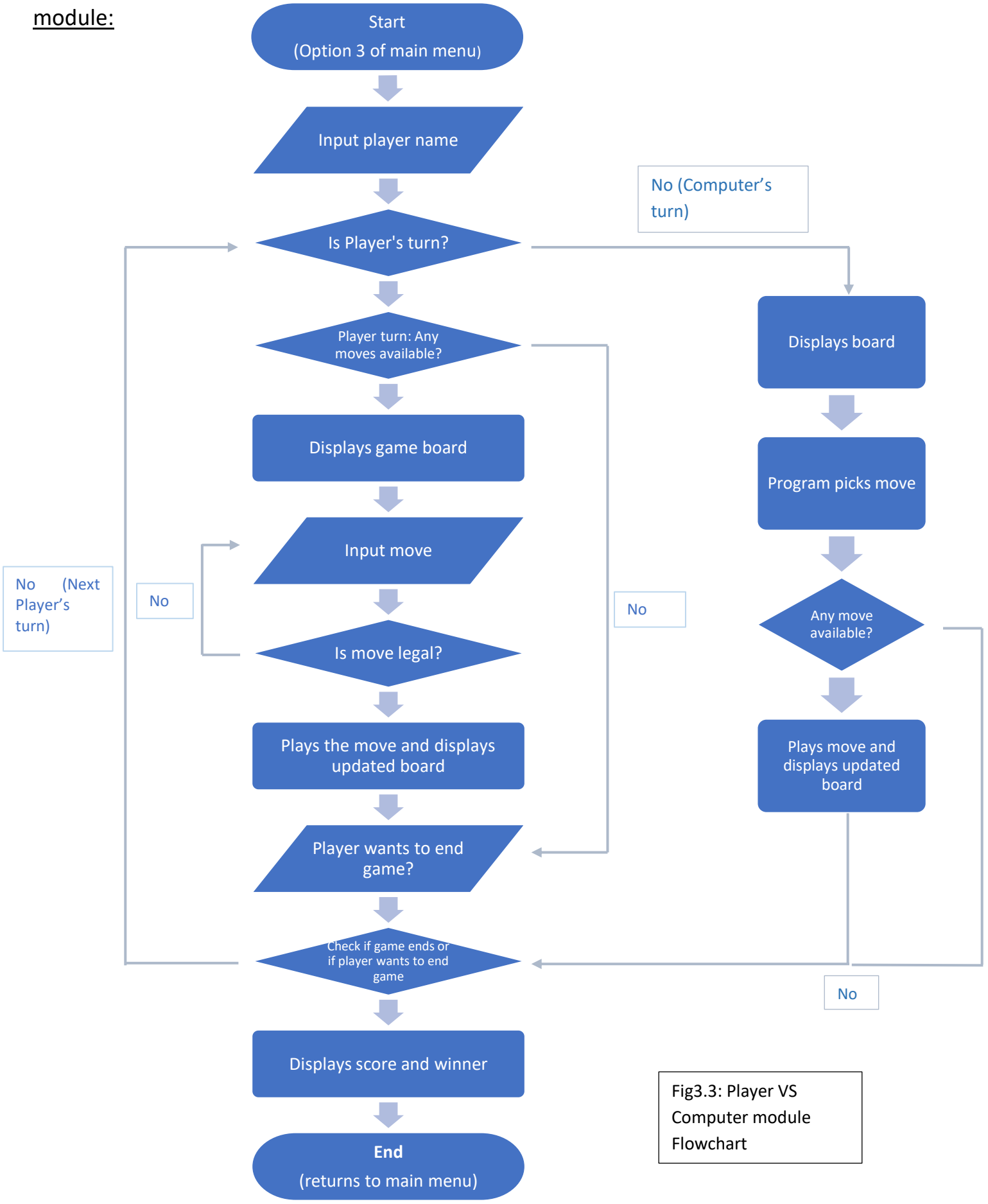

Fig3.3: Player VS Computer module Flowchart

3.3 Algorithm used

Step 1: Start

Step 2: Display main menu with four options.

If option 1, jump to Step 3.

Else if option 2, jump to Step 5.

Else if option 3, jump to Step 12.

else jump to Step 24.

Step 3: Open Instructions text file and print instructions.

Step 4: Return to Step 2.

Step 5: Input both players' names.

Step 6: Create and display new board.

Step 7: While players do not want to quit game or until game does not end, repeat steps 8 to 10.

Step 8: If player has possible moves,board is displayed and player inputs their move. Else jump to Step 10.

Step 9: Move is played, board updated and displayed.

Step 10: If player wants to quit game or game ends, jump to Step 11. Else play passed to the other player.

Step 11: Score is displayed and winner is declared. Return to Step 2.

Step 12: Input Player's name.

Step 13: Create and display new board.

Step 14: While Player does not want to quit game or until game does not end, repeat steps 15 to 22.

Step 15: Player's turn.

Step 16: If player has possible moves, board is displayed and player inputs the move.Else jump to Step 18.

Step 17: Move is played, board updated and displayed.

Step 18: If player wants to quit game or game ends, jump to Step 23.

Step 19: Play passes to Computer.

Step 20: Board is displayed.

Step 21: Program picks best possible move.

Step 22: If no moves available, if game ends, go to Step 23. Else move is played, updated board is displayed.

Step 23: The scores are displayed, and winner is declared. Return to Step 2.

Step 24: Stop.

3.3.1 Minimax algorithm

The Minimax algorithm is a decision-making algorithm commonly used in turn-based strategy games for AI opponents. It is also called the backtracking algorithm. This algorithm has been used to implement the Player VS Computer play in the program.

The algorithm is a recursive algorithm and makes use of trees. In this algorithm, all possible moves upto endgame or a specified depth are drawn as a tree and each node is given a board state value through recursion. In the game, one player is called the maximizing player and their opponent is called the minimizing player. The maximizing player, in their turn, always tries to maximize the board state while the minimizing player always tries to minimize the board state. Thus, the algorithm assumes that both players are playing optimally and based on this, picks the best possible move out of all possible moves.

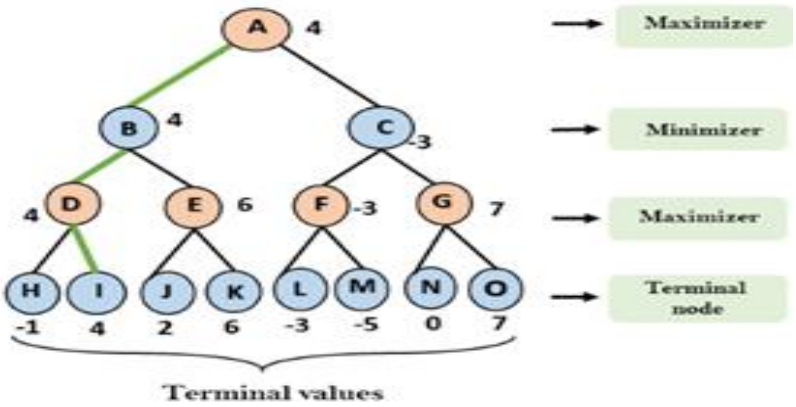

Fig3.4: Minimax algorithm in tree format

3.4 Code and implementation

The program is written using Python programming language. It is written in IDLE and run (and played) on Python shell.

3.4.1 Main menu module:

```
while(1):
   print()
   print()
   print("              WELCOME TO THE GAME OTHELLO! ")
   print()
   print("                  Select an option")
   print()
   print("             1.      How to play")
   print("             2. Let's Play : Player VS Player")
   print("             3. Let's Play : Player VS Computer")
   print("             4.       Exit")
   print()
   choice=int(input("Enter your choice: "))
   if choice==1:
      print()
      with open("Othello_instructions.txt") as fp:
         lines=fp.readlines()
         for i in range(16):
            print(lines[i])
         print()
         a=int(input("Press 1 to read more or 0 to go back to main menu: "))
```

```
        if a==1:
            for i in range(16,46):
                print(lines[i])
            print()
            a=int(input("Press 1 to read more or 0 to go back to main menu: "))
            if a==1:
                for i in range(46,99):
                    print(lines[i])
                print()
                a=input("Enter any key to return to main menu")
    elif choice==2:
        play()
    elif choice==3:
        print()
        SEM3_Othello_AI.play()
    else:
        break   #Exits game
```

Fig3.5: Main menu module code

```
                WELCOME TO THE GAME OTHELLO!

                     Select an option

                1.      How to play
                2.Let's Play : Player VS Player
                3.Let's Play : Player VS Computer
                4.        Exit

Enter your choice: 2
Let's Play!
Enter Player 1 name: Black (X): AAA
Enter Player 2 name: White (O): BBB
```

Fig3.6: Main menu module outcome

The main menu module is primarily implemented using while loop and elif-ladder.

3.4.2 Instructions module

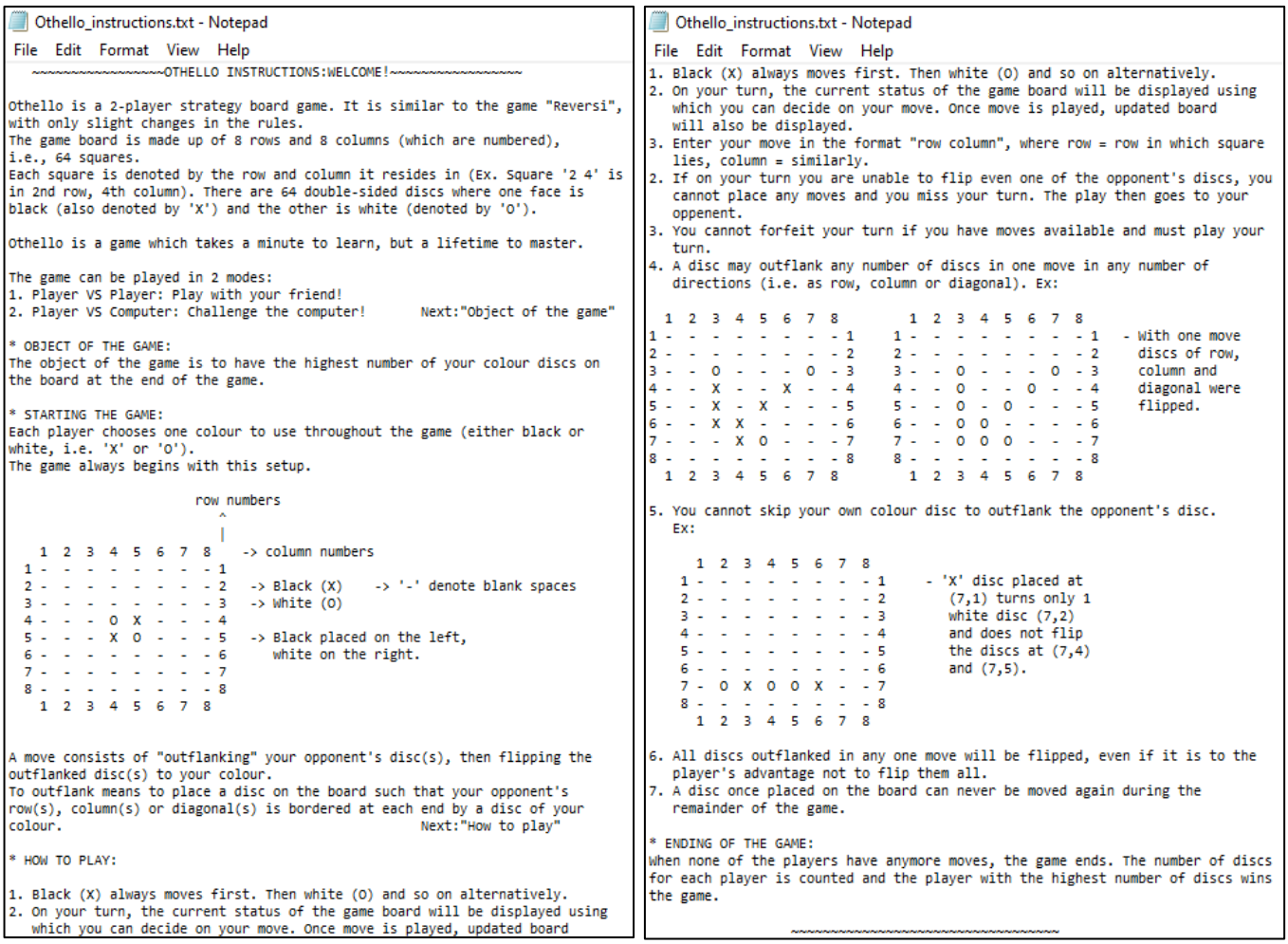

Fig3.7: Instructions text file

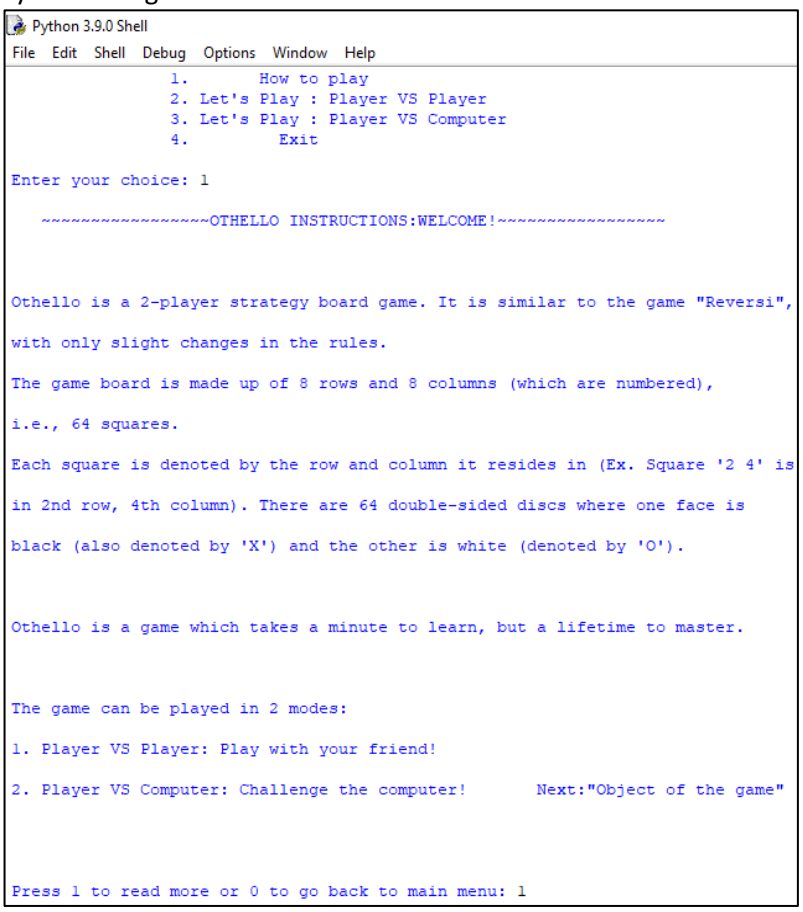

Fig3.8: Instructions outcome

The Instructions module is written in notepad. When the user selects option 1 in the main menu, the file is opened and read and printed (part-by-part). It is opened using context manager “with” and uses for loops and “readlines” function to read.

3.4.3 Player VS Player module

```
import SEM3_Othello_AI
#Global variables:
board=[]
pos_list=[]
#Create board (brand-new board) with initial set-up
def create_board():
    global board
    board=[[' ','1','2','3','4','5','6','7','8',' '], [1,'-','-','-','-','-','-','-','-',1], [2,'-','-','-','-','-','-','-','-',2], [3,'-','-','-','-','-','-','-','-',3], [4,'-','-','-','O','X','-','-','-',4], [5,'-','-','-','X','O','-','-','-',5], [6,'-','-','-','-','-','-','-','-',6], [7,'-','-','-','-','-','-','-','-',7], [8,'-','-','-','-','-','-','-','-',8], [' ','1','2','3','4','5','6','7','8',' ']]
```

```
    print()
    print('Game Board')
    print()
    for row in board:
        for col in row:
            print(col,end=' ')
        print()
    print()
    print("Note:","\'-\' denote empty spaces",
          "\'X\' denotes black coins (Player 1)",
          "\'O\' denotes white coins (Player 2)",
          "\'1,2,3...\' denote row/coloumn numbers",sep='\n')
    print()
    print()


# Display_board function
def display_board():
    for row in board:
        for col in row:
            print(col,end=' ')
        print()


def create_pos_list(sym):
    global pos_list
    pos_list.clear()       #Clears previous entries of this list so it can make new list for new pos
    temp=[]
    if sym=='X':
        oppnt='O'
    else:
        oppnt='X'
```

```
    #Range such that only board is considered, outer position numbers are not.

    for r in range(1,9):

        for c in range(1,9):

            if (board[r][c]=='-'):      #Thus, doesn't take positions that are occupied

                any_of_eight=check_eight(r, c, oppnt)   #function returns T/F # if even one cell(of eight) is
#oppnt(opponent), returns true, else false.

                if any_of_eight:

                    temp.append((r,c))

    for loc in temp:

        if check_if_turn(loc[0],loc[1],oppnt,sym):  #function returns T/F # if even one direction (after

    #checking if immediate is oppnt) will lead to turning, returns true, else false.

            pos_list.append(loc)    #Will have the absolute final list of all the possible (all conditions checked)
#pos for that symbol.


# Function: check_eight.

# check the eight locations with extra condition to prune out and get a smaller list of possibilities. Then
#every pos checked for turning using check_if_turn function.

def check_eight(r,c,oppnt):

    if (board[r-1][c]==oppnt) or (board[r-1][c+1]==oppnt) or (board[r][c+1]==oppnt) or
(board[r+1][c+1]==oppnt) or (board[r+1][c]==oppnt) or (board[r+1][c-1]==oppnt) or (board[r][c-
1]==oppnt) or (board[r-1][c-1]==oppnt):

        return True

    else:

        return False


# Function: check_if_turn.

# If even one direction of the eight directions returns true, then valid position. Else returns false.

def check_if_turn(r,c,oppnt,sym):

    if board[r-1][c]==oppnt:

        i=r-2

        while(board[i][c]==oppnt):

            i=i-1
```

```
        if board[i][c]==sym:
            return True
    if board[r-1][c+1]==oppnt:
        i,j=r-2,c+2
        while(board[i][j]==oppnt):
            i,j=i-1,j+1
        if board[i][j]==sym:
            return True
    if board[r][c+1]==oppnt:
        j=c+2
        while(board[r][j]==oppnt):
            j=j+1
        if board[r][j]==sym:
            return True
    if board[r+1][c+1]==oppnt:
        i,j=r+2,c+2
        while(board[i][j]==oppnt):
            i,j=i+1,j+1
        if board[i][j]==sym:
            return True
    if board[r+1][c]==oppnt:
        i=r+2
        while(board[i][c]==oppnt):
            i=i+1
        if board[i][c]==sym:
            return True
    if board[r+1][c-1]==oppnt:
        i,j=r+2,c-2
        while(board[i][j]==oppnt):
            i,j=i+1,j-1
```

```
        if board[i][j]==sym:
            return True
    if board[r][c-1]==oppnt:
        j=c-2
        while(board[r][j]==oppnt):
            j=j-1
        if board[r][j]==sym:
            return True
    if board[r-1][c-1]==oppnt:
        i,j=r-2,c-2
        while(board[i][j]==oppnt):
            i,j=i-1,j-1
        if board[i][j]==sym:
            return True
    else:
        return False


# Function: check_for_moves
# Sees if there are any moves at all for player, if not, must transfer turn to other player, else must allow
#the player to continue his turn.
def check_for_moves():
    if len(pos_list)==0:
        print("Player has no possible moves! Passing the play to other player.")
        print()
        return False
    else:
        return True
# Function: check_with_pos_list
# Checks if the move is in pos_list, only then it will be a valid move. Else must loop until valid input is
#given.
```

```
def check_with_pos_list(r,c):
    move=(r,c)
    if move in pos_list:
        return True
    else:
        print("Invalid move, please try again.")
        return False


# Function: turn_coins
# Turns the coins for that player's move
def turn_coins(r,c,sym,oppnt):
    global board
    if board[r-1][c]==oppnt:
        i=r-2
        while(board[i][c]==oppnt):
            i=i-1
        if board[i][c]==sym:
            board[r][c]=sym
            # Must turn the coins then
            a=r-1
            while(board[a][c]==oppnt):
                board[a][c]=sym
                a=a-1
    if board[r-1][c+1]==oppnt:
        i,j=r-2,c+2
        while(board[i][j]==oppnt):
            i,j=i-1,j+1
        if board[i][j]==sym:
            board[r][c]=sym
            # Must turn the coins then
```

```
        a,b=r-1,c+1
        while(board[a][b]==oppnt):
            board[a][b]=sym
            a,b=a-1,b+1
    if board[r][c+1]==oppnt:
        j=c+2
        while(board[r][j]==oppnt):
            j=j+1
        if board[r][j]==sym:
            board[r][c]=sym
            # Must turn the coins then
            b=c+1
            while(board[r][b]==oppnt):
                board[r][b]=sym
                b=b+1
    if board[r+1][c+1]==oppnt:
        i,j=r+2,c+2
        while(board[i][j]==oppnt):
            i,j=i+1,j+1
        if board[i][j]==sym:
            board[r][c]=sym
            # Must turn the coins then
            a,b=r+1,c+1
            while(board[a][b]==oppnt):
                board[a][b]=sym
                a,b=a+1,b+1
    if board[r+1][c]==oppnt:
        i=r+2
        while(board[i][c]==oppnt):
            i=i+1
```

```
        if board[i][c]==sym:
            board[r][c]=sym
            # Must turn the coins then
            a=r+1
            while(board[a][c]==oppnt):
                board[a][c]=sym
                a=a+1
    if board[r+1][c-1]==oppnt:
        i,j=r+2,c-2
        while(board[i][j]==oppnt):
            i,j=i+1,j-1
        if board[i][j]==sym:
            board[r][c]=sym
            # Must turn the coins then
            a,b=r+1,c-1
            while(board[a][b]==oppnt):
                board[a][b]=sym
                a,b=a+1,b-1
    if board[r][c-1]==oppnt:
        j=c-2
        while(board[r][j]==oppnt):
            j=j-1
        if board[r][j]==sym:
            board[r][c]=sym
            # Must turn the coins then
            b=c-1
            while(board[r][b]==oppnt):
                board[r][b]=sym
                b=b-1
    if board[r-1][c-1]==oppnt:
```

```
        i,j=r-2,c-2
        while(board[i][j]==oppnt):
            i,j=i-1,j-1
        if board[i][j]==sym:
            board[r][c]=sym
            # Must turn the coins then
            a,b=r-1,c-1
            while(board[a][b]==oppnt):
                board[a][b]=sym
                a,b=a-1,b-1
    print("Turned! Updated board:")


# Function: check_endgame
# Game ends if:
    # neither players have moves
        # ie create_pos_list for both (use 'and') symbols give empty lists
# Then must check who won (by counting no. of cells for each player and then declare winner with score).
def check_endgame(sym1,sym2):
    create_pos_list(sym1)
    l1=pos_list[:]    #Copying sym1's pos_list to 'l1'
    create_pos_list(sym2)
    l2=pos_list[:]    #Copying sym2's pos_list to 'l2'
    if(len(l1)==0 and len(l2)==0):   #Only if both pos_lists are empty, game over
        print("Players have no more moves! Game over.")
        return 1
    else:
        return 0


# Function: check_winner
# Checks who won by counting no. of coins of each player on board and then declares winner (whoever
```

```
#with higher coin count) with their coin counts.
def check_winner(sym1,sym2,P1,P2):
    sym1_count=0
    sym2_count=0
    for r in range(1,9):
        for c in range(1,9):
            if board[r][c]==sym1:
                sym1_count+=1
            elif board[r][c]==sym2:
                sym2_count+=1
    if sym1_count>sym2_count:
        print("Scores:")
        print("Number of Player 1 coins: ",sym1_count)
        print("Number of Player 2 coins: ",sym2_count)
        print("Player 1,",P1,"wins!!")
    elif sym2_count>sym1_count:
        print("Scores:")
        print("Number of Player 1 coins: ",sym1_count)
        print("Number of Player 2 coins: ",sym2_count)
        print("Player 2,",P2,"wins!!")
    else:
        print("Scores:")
        print("Number of Player 1 coins: ",sym1_count)
        print("Number of Player 2 coins: ",sym2_count)
        print("It's a tie!!")

# Play game
def play():
    print("Let's Play!")
    P1=input("Enter Player 1 name: Black (X): ")
```

```
sym1='X'
P2=input("Enter Player 2 name: White (O): ")
sym2='O'
create_board()
# Shows * * * * as if loading
print("Setting up game")
for times in range(4):
    for delay in range(10000000):
        if delay==5000000:
            print('*',end='  ')
#End loading sequence
print()
print()
turn=0
while(1):
    #Player1
    if turn%2==0:
        create_pos_list(sym1)
        print("Player 1:",P1,"'s (X) turn:")
        checking=check_for_moves()
        if checking==True:
            print()
            display_board()
            print()
            while(1):
                row,column=input("Enter move:(format:row column) ").split()
                row,column=int(row),int(column)
                correct=check_with_pos_list(row,column)
                if correct:
                    break
```

```
            turn_coins(row,column,sym1,sym2)
            display_board()   #Shows players updated board
            print()
            #Board displayed
            #Move checked and turned
            #Board updated and displayed
        #Player2
        else:
            create_pos_list(sym2)
            print("Player 2:",P2,"'s (O) turn:")
            checking=check_for_moves()
            if checking==True:
                print()
                display_board()
                print()
                while(1):
                    row,column=input("Enter move:(format:row column) ").split()
                    row,column=int(row),int(column)
                    correct=check_with_pos_list(row,column)
                    if correct:
                        break
                turn_coins(row,column,sym2,sym1)
                display_board()   #Shows players updated board
                print()
            #Board displayed
            #Move checked and turned
            #Board updated and displayed
        option=int(input("(Press 1 to continue the game and 0 to quit) "))
        print()
        end=check_endgame(sym1,sym2)
```

```
    if (end==1 or option==0):
        check_winner(sym1,sym2,P1,P2)
        print()
        print("Thank you for playing!")
        break
    turn+=1
```

Fig3.9: Player VS Player code

```
                WELCOME TO THE GAME OTHELLO!

                     Select an option

               1.       How to play
               2. Let's Play : Player VS Player
               3. Let's Play : Player VS Computer
               4.         Exit

Enter your choice: 2
Let's Play!
Enter Player 1 name: Black (X): AAA
Enter Player 2 name: White (O): BBB

Game Board

  1 2 3 4 5 6 7 8
1 - - - - - - - - 1
2 - - - - - - - - 2
3 - - - - - - - - 3
4 - - - O X - - - 4
5 - - - X O - - - 5
6 - - - - - - - - 6
7 - - - - - - - - 7
8 - - - - - - - - 8
  1 2 3 4 5 6 7 8

Note:
'-' denote empty spaces
'X' denotes black coins (Player 1)
'O' denotes white coins (Player 2)
'1,2,3...' denote row/coloumn numbers


Setting up game
*  *  *  *

Player 1: AAA 's (X) turn:

  1 2 3 4 5 6 7 8
1 - - - - - - - - 1
```

```
8 - - - - X - - - 8
  1 2 3 4 5 6 7 8

Enter move:(format:row column) 6 1
Turned! Updated board:
  1 2 3 4 5 6 7 8
1 - - - - O - - - 1
2 - - - - O - - - 2
3 - O - - O X - - 3
4 X O O O X X - - 4
5 X O O X O - - - 5
6 X X X - O - - - 6
7 - - - - X O - - 7
8 - - - - X - - - 8
  1 2 3 4 5 6 7 8

(Press 1 to continue the game and 0 to quit) 1

Player 2: BBB 's (O) turn:

  1 2 3 4 5 6 7 8
1 - - - - O - - - 1
2 - - - - O - - - 2
3 - O - - O X - - 3
4 X O O O X X - - 4
5 X O O X O - - - 5
6 X X X - O - - - 6
7 - - - - X O - - 7
8 - - - - X - - - 8
  1 2 3 4 5 6 7 8

Enter move:(format:row column) 1 1
Invalid move, please try again.
Enter move:(format:row column) 6 4
Turned! Updated board:
  1 2 3 4 5 6 7 8
1 - - - - O - - - 1
2 - - - - O - - - 2
3 - O - - O X - - 3
4 X O O O X X - - 4
5 X O O O O - - - 5
6 X X X O O - - - 6
7 - - - - X O - - 7
8 - - - - X - - - 8
  1 2 3 4 5 6 7 8
```

```
Enter move:(format:row column) 1 1
Invalid move, please try again.
Enter move:(format:row column) 6 4
Turned! Updated board:
  1 2 3 4 5 6 7 8
1 - - - - O - - - 1
2 - - - - O - - - 2
3 - O - - O X - - 3
4 X O O O X X - - 4
5 X O O O O - - - 5
6 X X X O O - - - 6
7 - - - - X O - - 7
8 - - - - X - - - 8
  1 2 3 4 5 6 7 8

(Press 1 to continue the game and 0 to quit) 0

Scores:
Number of Player 1 coins:  10
Number of Player 2 coins:  14
Player 2, BBB wins!!

Thank you for playing!
```

Fig3.10: Player VS Player implementation

3.4.4 Player VS Computer module

```
import math    # For defining infinity

from copy import deepcopy

#Create board (brand-new board) with initial set-up

def create_board():

   board=[[' ','1','2','3','4','5','6','7','8',' '], [1,'-','-','-','-','-','-','-','-','-',1], [2,'-','-','-','-','-','-','-','-','-',2], [3,'-','-','-','-','-','-','-','-','-',3], [4,'-','-','-','O','X','-','-','-','-',4], [5,'-','-','-','X','O','-','-','-','-',5], [6,'-','-','-','-','-','-','-','-','-',6],[7,'-','-','-','-','-','-','-','-','-',7],[8,'-','-','-','-','-','-','-','-','-',8], [' ','1','2','3','4','5','6','7','8',' ']]

   print()

   print('Game Board')

   print()

   for row in board:

      for col in row:

         print(col,end=' ')

      print()

   print()

   print("Note:","\'-\' denote empty spaces",

      "\'X\' denotes black coins (Player 1)",
```

```
        "\'O\' denotes white coins (Player 2)",
        "\'1,2,3...\' denote row/coloumn numbers",sep='\n')
    print()
    print()
    return board


# Display_board function
# Just displays the current board situation
# So that player can see the board and make their required move.
def display_board(board):
    for row in board:
        for col in row:
            print(col,end=' ')
        print()


def create_pos_list(board,sym):
    #First, generate list of moves for that symbol
    pos_list=[]
    temp=[]
    if sym=='X':
        oppnt='O'
    else:
        oppnt='X'
    #for r in range(1,9):
    #for c in range(1,9):    #Range such that only board is considered, outer position numbers are not.
    for r in range(1,9):
        for c in range(1,9):
            if (board[r][c]=='-'):     #Thus, doesn't take positions that are occupied
                any_of_eight=check_eight(board, r, c, oppnt) #function returns T/F # if even one cell(of eight) is
#oppnt(opponent),returns true,else false.
                if any_of_eight:
```

```
                temp.append((r, c))
    for loc in temp:
        if check_if_turn(board, loc[0], loc[1], oppnt, sym):  #function returns T/F
            pos_list.append(loc)    #Will have the absolute final list of all the possible (all conditions checked)
#pos for that symbol.
    return pos_list


# Function: check_eight.
# check the eight locations with extra condition to prune out and get a smaller list of possibilities. then
#every pos checked for turning using another function.
# Extra condition: all eight empty or with player's symbol, then skip.
def check_eight(board,r,c,oppnt):
    if (board[r-1][c]==oppnt) or (board[r-1][c+1]==oppnt) or (board[r][c+1]==oppnt) or
(board[r+1][c+1]==oppnt) or (board[r+1][c]==oppnt) or (board[r+1][c-1]==oppnt) or (board[r][c-
1]==oppnt) or (board[r-1][c-1]==oppnt):
        return True
    else:
        return False
    #return T/F
# Function: check_if_turn.
# If even one direction (after checking if immediate is oppnt) will lead to turning, returns #true, else false.
def check_if_turn(board,r,c,oppnt,sym):
    if board[r-1][c]==oppnt:
        i=r-2
        while(board[i][c]==oppnt):
            i=i-1
        if board[i][c]==sym:
            return True
    if board[r-1][c+1]==oppnt:
        i,j=r-2,c+2
        while(board[i][j]==oppnt):
            i,j=i-1,j+1
```

```
        if board[i][j]==sym:
            return True
    if board[r][c+1]==oppnt:
        j=c+2
        while(board[r][j]==oppnt):
            j=j+1
        if board[r][j]==sym:
            return True
    if board[r+1][c+1]==oppnt:
        i,j=r+2,c+2
        while(board[i][j]==oppnt):
            i,j=i+1,j+1
        if board[i][j]==sym:
            return True
    if board[r+1][c]==oppnt:
        i=r+2
        while(board[i][c]==oppnt):
            i=i+1
        if board[i][c]==sym:
            return True
    if board[r+1][c-1]==oppnt:
        i,j=r+2,c-2
        while(board[i][j]==oppnt):
            i,j=i+1,j-1
        if board[i][j]==sym:
            return True
    if board[r][c-1]==oppnt:
        j=c-2
        while(board[r][j]==oppnt):
            j=j-1
        if board[r][j]==sym:
```

```
            return True
    if board[r-1][c-1]==oppnt:
        i,j=r-2,c-2
        while(board[i][j]==oppnt):
            i,j=i-1,j-1
        if board[i][j]==sym:
            return True
    else:
        return False


# Function: check_for_moves
# Sees if there are any moves at all for player, if not, must transfer turn to other player, else must allow
#the player to continue his turn.
def check_for_moves(pos_list):
    if len(pos_list)==0:
        print("Player has no possible moves! Passing the play to other player.")
        print()
        return False
    else:
        return True


# Function: check_with_pos_list
# Checks if the move is in pos_list, only then it will be a valid move. Else must loop until valid input is
#given.
def check_with_pos_list(pos_list,r,c):
    move=(r,c)
    if move in pos_list:
        return True
    else:
        print("Invalid move, please try again.")
        return False
```

```
# Function: turn_coins
# Turns the coins for that player's move
def turn_coins(board,r,c,sym,oppnt,sim=False):    #sim = simulation
    if sim==True:
        board=deepcopy(board)
    if board[r-1][c]==oppnt:
        i=r-2
        while(board[i][c]==oppnt):
            i=i-1
        if board[i][c]==sym:
            board[r][c]=sym
            # Must turn the coins then
            a=r-1
            while(board[a][c]==oppnt):
                board[a][c]=sym
                a=a-1
    if board[r-1][c+1]==oppnt:
        i,j=r-2,c+2
        while(board[i][j]==oppnt):
            i,j=i-1,j+1
        if board[i][j]==sym:
            board[r][c]=sym
            # Must turn the coins then
            a,b=r-1,c+1
            while(board[a][b]==oppnt):
                board[a][b]=sym
                a,b=a-1,b+1
    if board[r][c+1]==oppnt:
        j=c+2
        while(board[r][j]==oppnt):
            j=j+1
```

```
    if board[r][j]==sym:
        board[r][c]=sym
        # Must turn the coins then
        b=c+1
        while(board[r][b]==oppnt):
            board[r][b]=sym
            b=b+1
  if board[r+1][c+1]==oppnt:
    i,j=r+2,c+2
    while(board[i][j]==oppnt):
        i,j=i+1,j+1
    if board[i][j]==sym:
        board[r][c]=sym
        # Must turn the coins then
        a,b=r+1,c+1
        while(board[a][b]==oppnt):
            board[a][b]=sym
            a,b=a+1,b+1
  if board[r+1][c]==oppnt:
    i=r+2
    while(board[i][c]==oppnt):
        i=i+1
    if board[i][c]==sym:
        board[r][c]=sym
        # Must turn the coins then
        a=r+1
        while(board[a][c]==oppnt):
            board[a][c]=sym
            a=a+1
  if board[r+1][c-1]==oppnt:
    i,j=r+2,c-2
```

```
    while(board[i][j]==oppnt):
        i,j=i+1,j-1
    if board[i][j]==sym:
        board[r][c]=sym
        # Must turn the coins then
        a,b=r+1,c-1
        while(board[a][b]==oppnt):
            board[a][b]=sym
            a,b=a+1,b-1
if board[r][c-1]==oppnt:
    j=c-2
    while(board[r][j]==oppnt):
        j=j-1
    if board[r][j]==sym:
        board[r][c]=sym
        # Must turn the coins then
        b=c-1
        while(board[r][b]==oppnt):
            board[r][b]=sym
            b=b-1
if board[r-1][c-1]==oppnt:
    i,j=r-2,c-2
    while(board[i][j]==oppnt):
        i,j=i-1,j-1
    if board[i][j]==sym:
        board[r][c]=sym
        # Must turn the coins then
        a,b=r-1,c-1
        while(board[a][b]==oppnt):
            board[a][b]=sym
            a,b=a-1,b-1
```

```
    if sim==False:
        print("Turned! Updated board:")
    return board


# Function: check_endgame
# Game ends if:
    # neither players have moves
        # ie create_pos_list for both (use 'and') symbols give empty lists
# Then must check who won (by counting no. of cells for each player and then declare winner
# with score).
def check_endgame(board,sym1,sym2):
    l1=create_pos_list(board,sym1)
    l2=create_pos_list(board,sym2)
    if(len(l1)==0 and len(l2)==0):   #Only if both pos_lists are empty, game over
        print("Players have no more moves! Game over.")
        return 1
    else:
        return 0


# Function: check_winner
# Checks who won by counting no. of coins of each player on board and then declares winner (whoever
#with higher coin count) with their coin counts.
def check_winner(board,sym1,sym2,P1,P2="Computer"):
    sym1_count=0
    sym2_count=0
    for r in range(1,9):
        for c in range(1,9):
            if board[r][c]==sym1:
                sym1_count+=1
            elif board[r][c]==sym2:
                sym2_count+=1
```

```
    if sym1_count>sym2_count:
        print("Scores:")
        print("Number of Player 1 coins: ",sym1_count)
        print("Number of Player 2 coins: ",sym2_count)
        print("Player 1,",P1,"wins!!")
    elif sym2_count>sym1_count:
        print("Scores:")
        print("Number of Player 1 coins: ",sym1_count)
        print("Number of Player 2 coins: ",sym2_count)
        print("Player 2, Computer wins!!")
    else:
        print("Scores:")
        print("Number of Player 1 coins: ",sym1_count)
        print("Number of Player 2 coins: ",sym2_count)
        print("It's a tie!!")


def get_score(board):
    sym1_score=0
    sym2_score=0
    for r in range(1,9):
        for c in range(1,9):
            if board[r][c]=='X':
                if (r,c) in [(1,1),(1,8),(8,8),(8,1)]:  #Bonus points for corners
                    sym1_score+=1000
                elif (r,c) in [(1,4),(1,5),(4,1),(5,1),(8,4),(8,5),(4,8),(5,8)]:    #Bonus points for edges
                    sym1_score+=10
                elif     (r,c) in [(2,1),(2,2),(1,2),(1,7),(2,7),(2,8),(7,1),(7,2),(8,2),(7,7),(7,8),(8,7)]:  #Subtracting
#points for X,C positions on board
                    sym1_score-=20
                elif (r,c) in
[(3,1),(3,2),(3,3),(2,3),(1,3),(1,6),(2,6),(3,6),(3,7),(3,8),(6,1),(6,2),(6,3),(7,3),(8,3),(6,6),(6,7),(6,8),(7,6),(8,6)]:
#Bonus points
```

```
                sym1_score+=15                  #for positions adjacent to the X,C positions.
            else:      #Every other position
                sym1_score+=1
        elif board[r][c]=='O':
            if (r,c) in [(1,1),(1,8),(8,8),(8,1)]:  #Bonus points for corners
                sym2_score+=1000
            elif (r,c) in [(1,4),(1,5),(4,1),(5,1),(8,4),(8,5),(4,8),(5,8)]:    #Bonus points for edges
                sym2_score+=10
            elif (r,c) in [(2,1),(2,2),(1,2),(1,7),(2,7),(2,8),(7,1),(7,2),(8,2),(7,7),(7,8),(8,7)]:  #Subtracting points
#for X,C positions on board
                sym2_score-=20
            elif (r,c) in
[(3,1),(3,2),(3,3),(2,3),(1,3),(1,6),(2,6),(3,6),(3,7),(3,8),(6,1),(6,2),(6,3),(7,3),(8,3),(6,6),(6,7),(6,8),(7,6),(8,6)]:
#Bonus points
                sym2_score+=15                  #for positions adjacent to the X,C positions.
            else:      #Every other position
                sym2_score+=1
    return sym2_score-sym1_score       # More +ve value indicates computer winning and more -ve value
#indicates player winning


def minimax_max_node(board, depth, alpha, beta):
    cur_max = -math.inf      #-infinity
    copy_board = deepcopy(board)
    moves = create_pos_list(board,'O')
    if len(moves)==0 or depth==0:
        return (-1,-1),get_score(board)
    else:
        for i in moves:
            copy_board = turn_coins(board,i[0],i[1],'O','X',True)
            new_move, new_score = minimax_min_node(copy_board, depth - 1, alpha, beta)
            if new_score > cur_max:
                cur_max = new_score
```

```
            best_move = i
        alpha = max(new_score, alpha)
        if beta <= alpha:
            break
    return best_move, cur_max


def minimax_min_node(board, depth, alpha, beta):
    cur_min = math.inf
    copy_board = deepcopy(board)
    moves = create_pos_list(board, 'X')
    if len(moves)==0 or depth==0:
        return (-1,-1), get_score(board)
    else:
        for i in moves:
            copy_board = turn_coins(board,i[0],i[1],'X','O',True)
            new_move, new_score = minimax_max_node(copy_board, depth - 1, alpha, beta)
            if new_score < cur_min:
                cur_min = new_score
                best_move = i
            beta = min(new_score, beta)
            if beta <= alpha:
                break
        return best_move, cur_min


# Given a board and a player color, decide on a move.
# The return value is a tuple of integers (i,j), where
# i is the row and j is the column on the board.
def auto_move_minimax(board):
    best_move, score = minimax_max_node(board, 4, -math.inf, math.inf)
    return best_move
```

```
# Play game
def play():
   print("Let's Play!")
   P1=input("Enter Player 1 name: Black (X): ")
   sym1='X'
   print("Player 2: White (O): Computer")
   sym2='O'
   board=create_board()
   # Shows * * * * as if loading
   print("Setting up game")
   for times in range(4):
      for delay in range(10000000):
         if delay==5000000:
            #print(' '*1,'*',end='')
            print('*',end='  ')
   #End loading sequence
   print()
   print()
   turn=0
   while(1):
      #Player1
      if turn%2==0:
         pos_list=create_pos_list(board, sym1)
         print("Player 1:",P1,"'s (X) turn:")
         checking=check_for_moves(pos_list)
         if checking==True:
            print()
            display_board(board)
            print()
            while(1):
               row,column=input("Enter move:(format:row column) ").split()
```

```
            row,column=int(row),int(column)
            correct=check_with_pos_list(pos_list,row,column)
            if correct:
                break
        board=turn_coins(board,row,column,sym1,sym2)
        display_board(board)   #Shows players updated board
        print()
        #Board displayed
        #Move checked and turned
        #Board updated and displayed
    option=int(input("(Press 1 to continue the game and 0 to quit) "))
    print()
    end=check_endgame(board,sym1,sym2)
    if (end==1 or option==0):
        check_winner(board,sym1,sym2,P1)
        print()
        print("Thank you for playing!")
        break
#Player2
else:
    print("Player 2: Computer's (O) turn:")
    print()
    display_board(board)
    print()
    print("Computing move")
    move=auto_move_minimax(board)
    row,column=move[0],move[1]
    if row==-1 and column==-1:
        print("Computer has no possible moves! Passing the play to player.")
        print()
    else:
```

```
            board=turn_coins(board, row,column,sym2,sym1)  #Turns the coins based on #selected move.
            print("Move played:", row, column)
            print()
            display_board(board)   #Shows the updated board (i.e. with computer's move)
            print()
            #Board displayed
            #Move checked and turned
            #Board updated and displayed
        option=int(input("(Press 1 to continue the game and 0 to quit) "))
        print()
        end=check_endgame(board, sym1,sym2)
        if (end==1 or option==0):
            check_winner(board, sym1,sym2,P1)
            print()
            print("Thank you for playing!")
            break
    turn+=1
```

Fig3.8(?): Player VS Computer code

```
Let's Play!
Enter Player 1 name: Black (X): AAA
Player 2: White (O): Computer

Game Board

  1 2 3 4 5 6 7 8
1 - - - - - - - - 1
2 - - - - - - - - 2
3 - - - - - - - - 3
4 - - - O X - - - 4
5 - - - X O - - - 5
6 - - - - - - - - 6
7 - - - - - - - - 7
8 - - - - - - - - 8
  1 2 3 4 5 6 7 8

Note:
'-' denote empty spaces
'X' denotes black coins (Player 1)
'O' denotes white coins (Player 2)
'1,2,3...' denote row/coloumn numbers


Setting up game
*  *  *  *
```

```
Player 2: Computer's (O) turn:

  1 2 3 4 5 6 7 8
1 - - - - - - - - 1
2 - - - - X - - - 2
3 - O - - X X - - 3
4 - - O O X X - - 4
5 - - X X X X - - 5
6 - - O - - - - - 6
7 - - - - - - - - 7
8 - - - - - - - - 8
  1 2 3 4 5 6 7 8

Computing move
Turned! Updated board:
Move played: 4 7

  1 2 3 4 5 6 7 8
1 - - - - - - - - 1
2 - - - - X - - - 2
3 - O - - X X - - 3
4 - - O O O O O - 4
5 - - X X X X - - 5
6 - - O - - - - - 6
7 - - - - - - - - 7
8 - - - - - - - - 8
  1 2 3 4 5 6 7 8

(Press 1 to continue the game and 0 to quit) 1

Player 1: AAA 's (X) turn:

  1 2 3 4 5 6 7 8
1 - - - - - - - - 1
2 - - - - X - - - 2
3 - O - - X X - - 3
4 - - O O O O O - 4
5 - - X X X X - - 5
6 - - O - - - - - 6
7 - - - - - - - - 7
8 - - - - - - - - 8
```

```
Player 1: AAA 's (X) turn:

  1 2 3 4 5 6 7 8
1 - - - - - - - - 1
2 - - - - X - - - 2
3 - O - - X X - - 3
4 - - O O O O O - 4
5 - - X X X X - - 5
6 - - O - - - - - 6
7 - - - - - - - - 7
8 - - - - - - - - 8
  1 2 3 4 5 6 7 8

Enter move:(format:row column) 3 4
Turned! Updated board:
  1 2 3 4 5 6 7 8
1 - - - - - - - - 1
2 - - - - X - - - 2
3 - O - X X X - - 3
4 - - O X X O O - 4
5 - - X X X X - - 5
6 - - O - - - - - 6
7 - - - - - - - - 7
8 - - - - - - - - 8
  1 2 3 4 5 6 7 8

(Press 1 to continue the game and 0 to quit) 1

Player 2: Computer's (O) turn:

  1 2 3 4 5 6 7 8
1 - - - - - - - - 1
2 - - - - X - - - 2
3 - O - X X X - - 3
4 - - O X X O O - 4
5 - - X X X X - - 5
6 - - O - - - - - 6
7 - - - - - - - - 7
8 - - - - - - - - 8
  1 2 3 4 5 6 7 8
```

Fig3.9(?): Player VS Computer implementation

CHAPTER 4: RESULTS AND DISCUSSION

4.1 Results

- On playing against the AI opponent (Computer) of the “Player VS Computer” play, the following results were obtained: -

Game number	Won / Lost	Computer’s score	Player’s score
Game 1	Lost	41	23
Game 2	Won	8	56
Game 3	Won	30	34
Game 4	Lost	49	15
Game 5	Won	28	36

Fig4.1: Computer VS Player

- The board game Othello is successfully translated into a python program where all the rules and other nuances of the game are followed.
- User-interface is user-friendly and easy-to-follow.
- Instructions are well-written and understandable.

4.2 Conclusion

- The physical board game “Othello” can be successfully translated into a python program.
- It requires the use of if-else statements, loops, user-defined functions and data structures like lists, tuples and dictionaries. For the “Player VS Computer” play, it also requires the use of the minimax algorithm.
- From the table (Fig4.1), we conclude that the AI opponent in the Player VS Computer play is a competent opponent and provides for a stimulating gameplay.
- By studying advanced playing strategies of Othello, a better AI opponent can be created.
- Playing Othello helps build up one’s strategic thinking skills in a fun and interesting way.

References

- NPTEL course – “The Joy of Computing using python”
- www.radagast.com
- www.instructables.com/member/The Puma
- Artificial Intelligence at Play — Connect Four (Minimax algorithm explained) | by Jonathan C.T. Kuo | Analytics Vidhya | Medium
- www.GeeksforGeeks.com
- **www.youtube.com/watch?v=l-hh51ncgDI**