MONASH University

MONASH INFORMATION TECHNOLOGY

FIT9136 Algorithms and Programming Foundations in Python

2023 Semester 2

Assignment 1

Student name:*Sachin Shivaramaiah*

Student ID:*34194037*

Creation date:*08/08/2023*

Last modified date:*27/08/2023*

In [ ]:
```
# Libraries to import (if any)
import random
```

3.1 Game menu function

In [1]:
```
 # Implement code for 3.1 here
def game_menu():
    print("Game Menu:")
    print("1. Start a Game")
    print("2. Print the Board")
    print("3. Place a Stone")
    print("4. Reset the Game")
    print("5. Exit")
```

In [6]:
```
# Test code for 3.1 here [The code in this cell should be commented]
""" The Function Game Menu is a set of print statement to guide the user,
it doesn't accept any parameters and returns no value. Below is the test code for the

def game_menu():
    print("Game Menu:")
    print("1. Start a Game")
    print("2. Print the Board")
    print("3. Place a Stone")
    print("4. Reset the Game")
    print("5. Exit")
game_menu()

#This will print the ouput as shown below
#Game Menu:
#1. Start a Game
#2. Print the Board
```

```
#3. Place a Stone
#4. Reset the Game
#5. Exit
```

3.2 Creating the Board

In [27]:
```
# Implement code for 3.2 here
def create_board(size):
    return[[' ' for vertical in range (size) ] for horizontal in range (size)]
```

In [9]:
```
# Test code for 3.2 here [The code in this cell should be commented]
""" Considering Multidimensional List for 2-D operation
The Fuction 'create_board()' takes a single input int parameter named 'size'
This Data Structure List[[]]will store the current status of the board"""

def create_board(size):
    #Nested List for 2-D, the inner loop of the list creats emtpy space with respect t
    return[[' ' for vertical in range (size) ] for horizontal in range (size)]

size=int(input("Enter the board size:"))
print("2-D List",create_board(size))

#This will take size as input and prints in 2-D List format
#Enter the board size:2
#2-D List [[' ', ' '], [' ', ' ']]
```

3.3 Is the target position occupied?

In [ ]:
```
 # Implement code for 3.3 here
def is_occupied(board,x,y):
    return board[x][y] ==' '
```

In [ ]:
```
# Test code for 3.3 here [The code in this cell should be commented]
''''With 2_D list ready to store the Data, A function 'is_occupied' is intitiated with
data, x for row index & y for column index, this test function will check if the 2-D l
a boolean value'''''

def create_board(size): #Implementation of board function
    return[[' ' for vertical in range (size) ] for horizontal in range (size)]

def is_occupied(board,x,y):
    return board[x][y] ==' ' #'board' will contain 2-D List x and y will represent use
size=int(input("Enter the board size:"))
board=create_board(size)     #Assining creat_board(size) function to board
x=int(input("Enter row index: ")) #user values for rows
y=input("Enter column index: ") #user values for columns
y=ord(y) - ord('A')
print(is_occupied(board,x,y))   #calling and printing is_occupued(board,x,y), which wi

#Functions checks if the entered row & cloumns are emtpy
#Enter the board size:6
#Enter row index: 1
#Enter column index: A
#True
```

3.4 Placing a Stone at a Specific Intersection

In [ ]:
```
 # Implement code for 3.4 here
def create_board(size):
    return[[' ' for vertical in range (size) ] for horizontal in range (size)]

def is_occupied(board,x,y):
        if(board[x][y] !=' '):
            print ("This place is occupied please enter other index")
            return False
        else:
            print("This place is not occupied")
            return True

def place_on_board(board,stone,position):
    position=(x,y)
    print(position)
    if is_occupied(board,x,y):
        board[x][y]=stone
        position=(x,y)
        return True
    return False
def print_board(board):
    size=len(board)


    z=0
    for j in range(((size-1) * 3) + 1):
        if(j % 3 == 0):
            print(chr(65 + z), end="")
            z += 1;
        else:
            print(" ", end="")

    print()

    for i in range(((size-1) * 2) + 1):
        for j in range(((size-1)* 3) + 1):
            if(i % 2 == 0):
                if(j == (size-1)*3):
                    print(board[i//2][j//3], i//2, end="")
                elif(j % 3 == 0):
                    print(board[i//2][j//3], end="")
                else:
                    print("-", end="")
            else:
                if(j % 3 == 0):
                    print("|", end="")
                else:
                    print(" ", end="")
        print()

size=int(input("Enter the board size:"))
board=create_board(size)
v =True
while v :
    x=int(input("Enter row index: "))
    y=input("Enter column index: ")
```

```
    y=ord(y) - ord('A')
    position=0
    stone=str(input("Enter the stone to be placed: "))
    print(place_on_board(board,stone,position))
```

In [ ]:

```
# Test code for 3.4 here [The code in this cell should be commented]

def create_board(size): #Implementation of board function
    return[[' ' for vertical in range (size) ] for horizontal in range (size)]

def is_occupied(board,x,y):
        if(board[x][y] !=' '):
            print ("This place is occupied please enter other index")
            return False #should exit the condition /fucntion if element is not preser
        else:
            print("This place is not occupied")
            return True #element is present

def place_on_board(board,stone,position):
    position=(x,y)
    print(position)
    if is_occupied(board,x,y):
        board[x][y]=stone
        position=(x,y)
        #print("Position of the Stone: ", position)
        return True
    return False
def print_board(board):
    size=len(board)
    #print("The Updated Board : ", board)
    #print("The Size of the Board :", size)
    z=0
    for j in range(((size-1) * 3) + 1): #Iterating through Columns
        if(j % 3 == 0):              #Selecting index which are modulus of 3,6,9,12
            print(chr(65 + z), end="")  #Printing ASCII Characters
            z += 1;                     #incrementing ASCII Character values till n
        else:
            print(" ", end="")          #print empty space if % of coulmn index is !=

    print()                            #print index

    for i in range(((size-1) * 2) + 1):    #Interating through rows for the size
        for j in range(((size-1)* 3) + 1): #Interating through coulmn for the size
            if(i % 2 == 0):             #selecting row index which are only 2,4,6,8
                if(j == (size-1)*3):       #to print row index *3 because rows end at
                    print(board[i//2][j//3], i//2, end="")
                elif(j % 3 == 0):       #blank space for pawn movement
                    print(board[i//2][j//3], end="")  #blank space for pawn movement
                else:
                    print("-", end="")  #print -- since it is % of 2
            else:
                if(j % 3 == 0):         #prinitig column index
                    print("|", end="")  #print cloumn index at multiples of 3
                else:
                    print(" ", end="")
        print()

size=int(input("Enter the board size:"))
board=create_board(size)
```

```
v =True
while v :
    x=int(input("Enter row index: "))
    y=input("Enter column index: ")
    y=ord(y) - ord('A') #Converting Ascii values to int w.r.t row index
    position=0
    stone=str(input("Enter the stone to be placed: "))
    print(place_on_board(board,stone,position))

#board=creat_board_size(9)
#x=1
#y=A
#Stone=●
#place_on_board(board,stone,position)
#Returns True
```

3.5 Printing the Board

In [ ]:
```
 # Implement code for 3.5 here
def print_board(board):
    size=len(board)
    x=0
    for j in range(((size-1) * 3) + 1):
        if(j % 3 == 0):
            print(chr(65 + x % size), end="")
            x += 1;
        else:
            print(" ", end="")

    print()
    for i in range(((size-1) * 2) + 1):
        for j in range(((size-1)* 3) + 1):
            if(i % 2 == 0):
                if(j == (size-1)*3):
                    print(" ", i//2, end="")
                elif(j % 3 == 0):
                    print(" ", end="")
                else:
                    print("-", end="")
            else:
                if(j % 3 == 0):
                    print("|", end="")
                else:
                    print(" ", end="")

        print()
size=int(input("Enter Board Size"))
LIS=[[' ' for vertical in range (size) ] for horizontal in range (size)]
board=LIS
print_board(board)
```

In [ ]:
```
# Test code for 3.5 here [The code in this cell should be commented]
def create_board(size): #Implementation of board function
    return[[' ' for vertical in range (size) ] for horizontal in range (size)]


def is_occupied(board,x,y):
        if(board[x][y] !=' '):
```

```
            print ("This place is occupied please enter other index")
            return False #should exit the condition /fucntion if element is not preser
        else:
            print("This place is not occupied")
            return True #element is present

def place_on_board(board,stone,position):
    position=(x,y)
    print(position)
    if is_occupied(board,x,y):
        board[x][y]=stone
        position=(x,y)
        print("Position of the Stone: ", position)
        return True
    return False
def print_board(board):
    size=len(board)
    #print("The Updated Board : ", board)
    print("The Size of the Board :", size)
    z=0
    for j in range(((size-1) * 3) + 1): #Iterating through Columns
        if(j % 3 == 0):              #Selecting index which are modulus of 3,6,9,12
            print(chr(65 + z), end="")  #Printing ASCII Characters
            z += 1;                     #incrementing ASCII Character values till n
        else:
            print(" ", end="")          #print empty space if % of coulmn index is !=

    print()                            #print index

    for i in range(((size-1) * 2) + 1):    #Interating through rows for the size
        for j in range(((size-1)* 3) + 1): #Interating through coulmn for the size
            if(i % 2 == 0):             #selecting row index which are only 2,4,6,8
                if(j == (size-1)*3):       #to print row index *3 because rows end at
                    print(board[i//2][j//3], i//2, end="")
                elif(j % 3 == 0):       #blank space for pawn movement
                    print(board[i//2][j//3], end="")  #blank space for pawn movement
                else:
                    print("-", end="")  #print -- since it is % of 2
            else:
                if(j % 3 == 0):         #prinitig column index
                    print("|", end="")  #print cloumn index at multiples of 3
                else:
                    print(" ", end="")
        print()

size=int(input("Enter the board size:"))
board=create_board(size)
v =True
while v :
    v=print_board(board)
    print(v)
#size=9
#board=create_board(size)
#print_board(board)
```

3.6 Check Available Moves

In [ ]:
```
 # Implement code for 3.6 here [The code in this cell should be commented]
def create_board(size): #Implementation of board function
    return[[' ' for vertical in range (size) ] for horizontal in range (size)]

def is_occupied(board,x,y):
    if(board[x][y] ==' '):
        y=chr(65+y)
        return x,y

def check_available_moves(board):
    available_moves = []
    size = len(board)
    for x in range(size):
        for y in range(size):
            available_moves.append(is_occupied(board, x, y))
    print("Number of Available moves",len(available_moves))
    return available_moves

size=int(input("Enter the board size:"))
board=create_board(size)
print(check_available_moves(board))
```

In [3]:
```
# Test code for 3.6 here [The code in this cell should be commented]

def create_board(size): #Implementation of board function
    return[[' ' for vertical in range (size) ] for horizontal in range (size)]

def is_occupied(board,x,y):
    if(board[x][y] ==' '):
        y=chr(65+y)
        return x,y

def check_available_moves(board):
    available_moves = []
    size = len(board)
    for x in range(size):
        for y in range(size):
            available_moves.append(is_occupied(board, x, y))
    print("Number of Available moves",len(available_moves))
    return available_moves

size=int(input("Enter the board size:"))
board=create_board(size)
print(check_available_moves(board))

#b=create_board(8)
#len(Check_Available_moves(b))
#output(64)
```

3.7 Check for the Winner

In [1]:
```
 # Implement code for 3.7 here [The code in this cell should be commented]
def create_board(size):
    return [[' ' for vertical in range(size)] for horizontal in range(size)]


def is_occupied(board, x, y):
```

```
    if board[x][y] != ' ':
        print("This place is occupied please enter other index")
        return False  # should exit the condition /fucntion if element is not present
    else:
        print("This place is not occupied")
        return True  # element is present


def place_on_board(board, stone, position):
    x, y = position
    print(position)
    if is_occupied(board, x, y):
        board[x][y] = stone
        position = (x, y)
        print("Position of the Stone: ", position)
        return True
    return False


def print_board(board):
    size = len(board)
    print("The Updated Board:")
    z = 0
    for j in range(((size - 1) * 3) + 1):
        if j % 3 == 0:
            print(chr(65 + z), end="")
            z += 1
        else:
            print(" ", end="")
    print()

    for i in range(((size - 1) * 2) + 1):
        for j in range(((size - 1) * 3) + 1):
            if i % 2 == 0:
                if j == (size - 1) * 3:
                    print(board[i // 2][j // 3], i // 2, end="")
                elif j % 3 == 0:
                    print(board[i // 2][j // 3], end="")
                else:
                    print("-", end="")
            else:
                if j % 3 == 0:
                    print("|", end="")
                else:
                    print(" ", end="")
        print()


def check_for_winner(board, player):
    size = len(board)

    # Check rows
    for row in board:
        if "".join(row).count(player * 5) >= 1:
            return True

    # Check columns
    for col in range(size):
        column_values = "".join([board[row][col] for row in range(size)])
        if column_values.count(player * 5) >= 1:
```

```
            return True

    # Check diagonals
    for row in range(size - 4):
        for col in range(size - 4):
            diagonal = "".join([board[row + i][col + i] for i in range(5)])
            anti_diagonal = "".join([board[row + i][col + 4 - i] for i in range(5)])
            if diagonal.count(player * 5) >= 1 or anti_diagonal.count(player * 5) >= 1
                return True

    return False

def test():

    size = int(input("Enter the board size: "))
    board = create_board(size)
    current_player = "●"

    while True:
        print("Current Player:", current_player)
        print_board(board)

        x = int(input("Enter row index: "))
        y = input("Enter column index: ")
        y = ord(y.upper()) - ord('A')
        position = (x, y)
        stone = current_player

        if place_on_board(board, stone, position):
            if check_for_winner(board, current_player):
                print_board(board)
                print(f"Player {current_player} wins!")
                break

            current_player = "○" if current_player == "●" else "●"

test()
```

In [ ]:

```
# Test code for 3.7 here [The code in this cell should be commented]
def create_board(size):
    return [[' ' for vertical in range(size)] for horizontal in range(size)]


def is_occupied(board, x, y):
    if board[x][y] != ' ':
        print("This place is occupied please enter other index")
        return False  # should exit the condition /fucntion if element is not present
    else:
        print("This place is not occupied")
        return True  # element is present


def place_on_board(board, stone, position):
    x, y = position
    print(position)
    if is_occupied(board, x, y):
        board[x][y] = stone
        position = (x, y)
        print("Position of the Stone: ", position)
```

```
        return True
    return False


def print_board(board):
    size = len(board)
    print("The Updated Board:")
    z = 0
    for j in range(((size - 1) * 3) + 1):
        if j % 3 == 0:
            print(chr(65 + z), end="")
            z += 1
        else:
            print(" ", end="")
    print()

    for i in range(((size - 1) * 2) + 1):
        for j in range(((size - 1) * 3) + 1):
            if i % 2 == 0:
                if j == (size - 1) * 3:
                    print(board[i // 2][j // 3], i // 2, end="")
                elif j % 3 == 0:
                    print(board[i // 2][j // 3], end="")
                else:
                    print("-", end="")
            else:
                if j % 3 == 0:
                    print("|", end="")
                else:
                    print(" ", end="")
        print()


def check_for_winner(board, player):
    size = len(board)

    # Check rows
    for row in board:
        if "".join(row).count(player * 5) >= 1:
            return True

    # Check columns
    for col in range(size):
        column_values = "".join([board[row][col] for row in range(size)])
        if column_values.count(player * 5) >= 1:
            return True

    # Check diagonals
    for row in range(size - 4):
        for col in range(size - 4):
            diagonal = "".join([board[row + i][col + i] for i in range(5)])
            anti_diagonal = "".join([board[row + i][col + 4 - i] for i in range(5)])
            if diagonal.count(player * 5) >= 1 or anti_diagonal.count(player * 5) >= 1
                return True

    return False

def test():

    size = int(input("Enter the board size: "))
```

```
    board = create_board(size)
    current_player = "●"

    while True:
        print("Current Player:", current_player)
        print_board(board)

        x = int(input("Enter row index: "))
        y = input("Enter column index: ")
        y = ord(y.upper()) - ord('A')
        position = (x, y)
        stone = current_player

        if place_on_board(board, stone, position):
            if check_for_winner(board, current_player):
                print_board(board)
                print(f"Player {current_player} wins!")
                break

            current_player = "○" if current_player == "●" else "●"

test()

#size=5
#row_index=1
#Cloumn_indes='A'
#Gives Final wining player
```

3.8 Random Computer Player

In [ ]:

```
 # Implement code for 3.8 here
import random

def create_board(size):
    return [[' ' for vertical in range(size)] for horizontal in range(size)]


def is_occupied(board, x, y):
    if board[x][y] != ' ':
        print("This place is occupied please enter other index")
        return False  # should exit the condition /fucntion if element is not present
    else:
        print("This place is not occupied")
        return True  # element is present


def place_on_board(board, stone, position):
    x, y = position
    print(position)
    if is_occupied(board, x, y):
        board[x][y] = stone
        position = (x, y)
        print("Position of the Stone: ", position)
        return True
    return False


def print_board(board):
```

```
    size = len(board)
    print("The Updated Board:")
    z = 0
    for j in range(((size - 1) * 3) + 1):
        if j % 3 == 0:
            print(chr(65 + z), end="")
            z += 1
        else:
            print(" ", end="")
    print()

    for i in range(((size - 1) * 2) + 1):
        for j in range(((size - 1) * 3) + 1):
            if i % 2 == 0:
                if j == (size - 1) * 3:
                    print(board[i // 2][j // 3], i // 2, end="")
                elif j % 3 == 0:
                    print(board[i // 2][j // 3], end="")
                else:
                    print("-", end="")
            else:
                if j % 3 == 0:
                    print("|", end="")
                else:
                    print(" ", end="")
        print()

def check_for_winner(board, player):
    size = len(board)

    # Check rows
    for row in board:
        if "".join(row).count(player * 5) >= 1:
            return True

    # Check columns
    for col in range(size):
        column_values = "".join([board[row][col] for row in range(size)])
        if column_values.count(player * 5) >= 1:
            return True

    # Check diagonals
    for row in range(size - 4):
        for col in range(size - 4):
            diagonal = "".join([board[row + i][col + i] for i in range(5)])
            anti_diagonal = "".join([board[row + i][col + 4 - i] for i in range(5)])
            if diagonal.count(player * 5) >= 1 or anti_diagonal.count(player * 5) >= 1
                return True

    return False

def random_computer_player(board, player):
    available_moves = []
    size = len(board)

    for x in range(size):
        for y in range(size):
            if board[x][y] == ' ':
                available_moves.append((x, y))
```

```
    if available_moves:
        return random.choice(available_moves)
    else:
        return None

def test():
    size = int(input("Enter the board size: "))
    board = create_board(size)
    current_player = "●"

    while True:
        print("Current Player:", current_player)
        print_board(board)

        if current_player == "●":
            x = int(input("Enter row index: "))
            y = input("Enter column index: ")
            y = ord(y.upper()) - ord('A')
            position = (x, y)
            stone = current_player

            if place_on_board(board, stone, position):
                if check_for_winner(board, current_player):
                    print_board(board)
                    print(f"Player {current_player} wins!")
                    break

                current_player = "○"
        else:
            computer_move = random_computer_player(board, current_player)
            if computer_move:
                x, y = computer_move
                print(f"Computer placing stone at row {x} and column {chr(y + ord('A')
                position = (x, y)
                stone = current_player

                if place_on_board(board, stone, position):
                    if check_for_winner(board, current_player):
                        print_board(board)
                        print(f"Player {current_player} wins!")
                        break

                    current_player = "●"

test()
```

In [2]:

```
# Test code for 3.8 here [The code in this cell should be commented]
import random

def create_board(size):
    return [[' ' for vertical in range(size)] for horizontal in range(size)]


def is_occupied(board, x, y):
    if board[x][y] != ' ':
        print("This place is occupied please enter other index")
        return False  # should exit the condition /fucntion if element is not present
    else:
        print("This place is not occupied")
```

```
        return True  # element is present


def place_on_board(board, stone, position):
    x, y = position
    print(position)
    if is_occupied(board, x, y):
        board[x][y] = stone
        position = (x, y)
        print("Position of the Stone: ", position)
        return True
    return False


def print_board(board):
    size = len(board)
    print("The Updated Board:")
    z = 0
    for j in range(((size - 1) * 3) + 1):
        if j % 3 == 0:
            print(chr(65 + z), end="")
            z += 1
        else:
            print(" ", end="")
    print()

    for i in range(((size - 1) * 2) + 1):
        for j in range(((size - 1) * 3) + 1):
            if i % 2 == 0:
                if j == (size - 1) * 3:
                    print(board[i // 2][j // 3], i // 2, end="")
                elif j % 3 == 0:
                    print(board[i // 2][j // 3], end="")
                else:
                    print("-", end="")
            else:
                if j % 3 == 0:
                    print("|", end="")
                else:
                    print(" ", end="")
        print()

def check_for_winner(board, player):
    size = len(board)

    # Check rows
    for row in board:
        if "".join(row).count(player * 5) >= 1:
            return True

    # Check columns
    for col in range(size):
        column_values = "".join([board[row][col] for row in range(size)])
        if column_values.count(player * 5) >= 1:
            return True

    # Check diagonals
    for row in range(size - 4):
        for col in range(size - 4):
            diagonal = "".join([board[row + i][col + i] for i in range(5)])
```

```
            anti_diagonal = "".join([board[row + i][col + 4 - i] for i in range(5)])
            if diagonal.count(player * 5) >= 1 or anti_diagonal.count(player * 5) >= 1
                return True

    return False

def random_computer_player(board, player):
    available_moves = []
    size = len(board)

    for x in range(size):
        for y in range(size):
            if board[x][y] == ' ':
                available_moves.append((x, y))

    if available_moves:
        return random.choice(available_moves)
    else:
        return None

def test():
    size = int(input("Enter the board size: "))
    board = create_board(size)
    current_player = "●"

    while True:
        print("Current Player:", current_player)
        print_board(board)

        if current_player == "●":
            x = int(input("Enter row index: "))
            y = input("Enter column index: ")
            y = ord(y.upper()) - ord('A')
            position = (x, y)
            stone = current_player

            if place_on_board(board, stone, position):
                if check_for_winner(board, current_player):
                    print_board(board)
                    print(f"Player {current_player} wins!")
                    break

                current_player = "○"
        else:
            computer_move = random_computer_player(board, current_player)
            if computer_move:
                x, y = computer_move
                print(f"Computer placing stone at row {x} and column {chr(y + ord('A')
                position = (x, y)
                stone = current_player

                if place_on_board(board, stone, position):
                    if check_for_winner(board, current_player):
                        print_board(board)
                        print(f"Player {current_player} wins!")
                        break

                    current_player = "●"

test()
```

```
#takes all the required input
#Comensces game between Player and Computer with random Moves
```

3.9 Play Game

In [1]:
```
 # Implement code for 3.9 here
def play_game():
    print("Hello There, Welcome to the Game of Gomuku")
    board_size = 0  # Initialize board size to 0
    board = []  # Initialize an empty board
    current_player = "●"  # Start with the black stone
    game_mode = ""
    game_status = False
    move_status = []

    while True:
        game_menu()
        choice = input("Enter your choice: ")

        if choice == "1":
            if game_status:
                reset_choice = input("A game is already in progress. \n Enter 1 to res
                if reset_choice == "1":
                    play_game()
                    #board = create_board(board_size)
                    #current_player = "●"  # Reset to black stone
                    #move_history = []  # Reset move history
                else:
                    print("Continuing with the current game.")
            else:
                    if board_size > 0:
                        continue_choice = input("A game is already in progress. \n Ent
                                                "Enter 2 to continue the game: ")
                        if continue_choice== "1":
                            board_size = 0
                            board = []
                            current_player = "●"
                            game_status = False
                            print("Starting a new game.")
                        else:
                            print("Continuing with the current game.")
                    else:
                        while True:
                            board_size = int(input("Enter the board size (minimum 5):
                            if board_size > 4:
                                break
                            else:
                                print("Invalid input. Please enter a board size greate

                        board = create_board(board_size)
                        current_player = "●"  # Reset to black stone
                        while True:
                            game_mode = input("Enter 1 or 2 to select game mode:\n"
                            "1. Player v/s Player \n2. Player v/s Computer:")
                            if game_mode == "1":
                                print("Now you have selected:", game_mode, "\n Player
                                      "\n Proceed by entering 2 to view your game boar
```

```
                            break
                        elif game_mode == "2":
                            print("Now you have selected:", game_mode, "\nPlayer v
                                  "\n Proceed by entering 2 to view your game boar
                            break
                        #elif game_mode == "6":
                            #print("Going to Main Menu.")
                            #break

                        else:
                            print("Invalid choice. Please enter 1 or 2.")
                    game_status = True

            elif choice == "2":
                if not game_status:
                    print("Game not yet started. Please select option 1 to start the game.
                else:
                    print_board(board)
                    print("Enter 3 to place a stone")

            elif choice == "3":
                if not game_status:
                    print("Game not yet started. Please select option 1 to start the game.
                else:
                    while True:
                        print(f"Current player: {current_player}")
                        move_input = input("Enter row and column indices (e.g. 2 A): ").sp

                        if len(move_input) != 2:
                            print("Invalid input. Please enter valid row and column indice
                            continue

                        x, y = move_input
                        if not x.isdigit() or not y.isalpha():
                            print("Invalid input. Please enter valid row and column indice
                            continue

                        x = int(x)
                        y = ord(y.upper()) - ord('A')  # Convert column to index
                        position = (x, y)

                        if place_on_board(board, current_player, position):
                            move_status.append((current_player, position))  # Save the mov
                            winner = check_for_winner(board, current_player)
                            if winner:
                                print_board(board)
                                print(f"Player {current_player} wins!")
                                board = create_board(board_size)  # Reset the board
                                move_status = []  # Reset move history
                            else:
                                current_player = "o" if current_player == "●" else "●"  #

                            if game_mode == "2" and not winner:
                                computer_move = random_computer_player(board, current_play
                                place_on_board(board, current_player, computer_move)
                                move_status.append((current_player, computer_move))  # Sav
                                winner = check_for_winner(board, current_player)
                                if winner:
                                    print_board(board)
                                    print(f"Player {current_player} wins!")
```

```
                                board = create_board(board_size)  # Reset the board
                                move_status = []  # Reset move history
                            current_player = "o" if current_player == "●" else "●"  #

                        print_board(board)
                        print("Enter 3 to Continuing Playing")
                        break  # Exit the loop if everything is successful


        elif choice == "4":
            if not game_status:
                print("Game not yet started. Please select option 1 to start the game.
            else:
                available_moves = check_available_moves(board)
                print("Available Moves:", available_moves)

        elif choice == "5":
            if not game_status:
                print("Game not yet started. Please select option 1 to start the game.
            else:
                board = create_board(board_size)
                current_player = "●"  # Reset to black stone
                move_status = []  # Reset move history
                print("Game Reset Complete")

        elif choice == "6":
            print("Exiting the game, Good Bye.")
            break

    else:
        print("Invalid choice. Please select a valid options Available below.")

play_game()
```

In [ ]:

```
# Test code for 3.9 here [The code in this cell should be commented]
# Implement the game mode based on user's choice

def game_menu():
    print("Game Menu:")
    print("1. Start a Game")
    print("2. Print the Board")
    print("3. Place a Stone")
    print("4. Check for available moves")
    print("5. Reset the Game")
    print("6. Exit")

def create_board(size):
    return[[' ' for vertical in range (size) ] for horizontal in range (size)]

def is_occupied(board,x,y):
        if(board[x][y] !=' '):
            print ("This place is occupied please enter other index")
            return False    #should exit the condition /fucntion if element is not pre
        else:
            y=chr(65+y)
            return x,y
        #for l in board[]
def place_on_board(board,stone,position):
    x,y=position
```

```
    #position=(x,y)
    #print(position)
    if is_occupied(board,x,y):
        board[x][y]=stone
        position=(x,y)
        print("Position of the Stone: ", position)
        return True
    return False

def print_board(board):
    #board=create_board(size)
    size=len(board)
    #print("The Updated Board : ", board)
    #print("The Size of the Board :", size)
    z=0
    for j in range(((size-1) * 3) + 1):
        if(j % 3 == 0):
            print(chr(65 + z), end="")
            z += 1;
        else:
            print(" ", end="")

    print()

    for i in range(((size-1) * 2) + 1):
        for j in range(((size-1)* 3) + 1):
            if(i % 2 == 0):
                if(j == (size-1)*3):
                    print(board[i//2][j//3], i//2, end="")
                elif(j % 3 == 0):
                    print(board[i//2][j//3], end="")
                else:
                    print("-", end="")
            else:
                if(j % 3 == 0):
                    print("|", end="")
                else:
                    print(" ", end="")
        print()

def check_available_moves(board):
    available_moves = []
    size = len(board)
    for x in range(size):
        for y in range(size):
            available_moves.append(is_occupied(board, x, y))
            #print(len(available_moves))
    return available_moves , len(available_moves)

def check_for_winner(board, player):
    size = len(board)

    # Check rows
    for row in board:
        if "".join(row).count(player * 5) >= 1:
            return True

    # Check columns
    for col in range(size):
        column_values = "".join([board[row][col] for row in range(size)])
```

```
        if column_values.count(player * 5) >= 1:
            return True

    # Check diagonals
    for row in range(size - 4):
        for col in range(size - 4):
            diagonal = "".join([board[row + i][col + i] for i in range(5)])
            anti_diagonal = "".join([board[row + i][col + 4 - i] for i in range(5)])
            if diagonal.count(player * 5) >= 1 or anti_diagonal.count(player * 5) >= 1
                return True

    return False
import random

def random_computer_player(board, player):
    available_moves = []
    size = len(board)

    for x in range(size):
        for y in range(size):
            if is_occupied(board, x, y):
                available_moves.append((x, y))

    if available_moves:
        return random.choice(available_moves)
    else:
        return None


def play_game():
    board_size = 0  # Initialize board size to 0
    board = []  # Initialize an empty board
    current_player = "●"  # Start with the black stone
    game_mode = ""
    game_status = False
    move_status = []

    while True:
        game_menu()
        choice = input("Enter your choice: ")

        if choice == "1":
            if game_status:
                reset_choice = input("A game is already in progress. \n Enter 1 to res
                if reset_choice == "1":
                    play_game()
                    #board = create_board(board_size)
                    #current_player = "●"  # Reset to black stone
                    #move_history = []  # Reset move history
                else:
                    print("Continuing with the current game.")
            else:
                    if board_size > 0:
                        continue_choice = input("A game is already in progress. \n Ent
                                                "Enter 2 to continue the game: ")
                        if continue_choice== "1":
                            board_size = 0
                            board = []
                            current_player = "●"
                            game_status = False
```

```
                print("Starting a new game.")
            else:
                print("Continuing with the current game.")
        else:
            while True:
                board_size = int(input("Enter the board size (minimum 5):
                if board_size > 4:
                    break
                else:
                    print("Invalid input. Please enter a board size greate

            board = create_board(board_size)
            current_player = "●"  # Reset to black stone
            while True:
                game_mode = input("Enter 1 or 2 to select game mode:\n"
                "1. Player v/s Player \n2. Player v/s Computer:")
                if game_mode == "1":
                    print("Now you have selected:", game_mode, "\n Player
                          "\n Proceed by entering 2 to view your game boar
                    break
                elif game_mode == "2":
                    print("Now you have selected:", game_mode, "\nPlayer v
                          "\n Proceed by entering 2 to view your game boar
                    break
                #elif game_mode == "6":
                    #print("Going to Main Menu.")
                    #break

                else:
                    print("Invalid choice. Please enter 1 or 2.")
            game_status = True

    elif choice == "2":
        if not game_status:
            print("Game not yet started. Please select option 1 to start the game.
        else:
            print_board(board)
            print("Enter 3 to place a stone")

    elif choice == "3":
        if not game_status:
            print("Game not yet started. Please select option 1 to start the game.
        else:
            while True:
                print(f"Current player: {current_player}")
                move_input = input("Enter row and column indices (e.g. 2 A): ").sp

                if len(move_input) != 2:
                    print("Invalid input. Please enter valid row and column indice
                    continue

                x, y = move_input
                if not x.isdigit() or not y.isalpha():
                    print("Invalid input. Please enter valid row and column indice
                    continue

                x = int(x)
                y = ord(y.upper()) - ord('A')  # Convert column to index
                position = (x, y)
```

```
                    if place_on_board(board, current_player, position):
                        move_status.append((current_player, position))  # Save the mov
                        winner = check_for_winner(board, current_player)
                        if winner:
                            print_board(board)
                            print(f"Player {current_player} wins!")
                            board = create_board(board_size)  # Reset the board
                            move_history = []  # Reset move history
                        else:
                            current_player = "o" if current_player == "●" else "●"  #

                        if game_mode == "2" and not winner:
                            computer_move = random_computer_player(board, current_play
                            place_on_board(board, current_player, computer_move)
                            move_status.append((current_player, computer_move))  # Sav
                            winner = check_for_winner(board, current_player)
                            if winner:
                                print_board(board)
                                print(f"Player {current_player} wins!")
                                board = create_board(board_size)  # Reset the board
                                move_status = []  # Reset move history
                            current_player = "o" if current_player == "●" else "●"  #

                        print_board(board)
                        print("Enter 3 to Continuing Playing")
                        break  # Exit the loop if everything is successful


        elif choice == "4":
            if not game_status:
                print("Game not yet started. Please select option 1 to start the game.
            else:
                available_moves = check_available_moves(board)
                print("Available Moves:", available_moves)

        elif choice == "5":
            if not game_status:
                print("Game not yet started. Please select option 1 to start the game.
            else:
                board = create_board(board_size)
                current_player = "●"  # Reset to black stone
                move_status = []  # Reset move history
                print("Game Reset Complete")

        elif choice == "6":
            print("Exiting the game, Good Bye.")
            break

    else:
        print("Invalid choice. Please select a valid options Available below.")

play_game()
```

In [2]:

```
#Run the game (Your tutor will run this cell to start playing the game)
def game_menu():
    print("Game Menu:")
    print("1. Start a Game")
    print("2. Print the Board")
    print("3. Place a Stone")
```

```
    print("4. Check for available moves")
    print("5. Reset the Game")
    print("6. Exit")

def create_board(size):
    return[[' ' for vertical in range (size) ] for horizontal in range (size)]

def is_occupied(board,x,y):
        if(board[x][y] !=' '):
            print ("This place is occupied please enter other index")
            return False    #should exit the condition /fucntion if element is not pre
        else:
            y=chr(65+y)
            return x,y
        #for l in board[]
def place_on_board(board,stone,position):
    x,y=position
    #position=(x,y)
    #print(position)
    if is_occupied(board,x,y):
        board[x][y]=stone
        position=(x,y)
        print("Position of the Stone: ", position)
        return True
    return False

def print_board(board):
    #board=create_board(size)
    size=len(board)
    #print("The Updated Board : ", board)
    #print("The Size of the Board :", size)
    z=0
    for j in range(((size-1) * 3) + 1):
        if(j % 3 == 0):
            print(chr(65 + z), end="")
            z += 1;
        else:
            print(" ", end="")

    print()

    for i in range(((size-1) * 2) + 1):
        for j in range(((size-1)* 3) + 1):
            if(i % 2 == 0):
                if(j == (size-1)*3):
                    print(board[i//2][j//3], i//2, end="")
                elif(j % 3 == 0):
                    print(board[i//2][j//3], end="")
                else:
                    print("-", end="")
            else:
                if(j % 3 == 0):
                    print("|", end="")
                else:
                    print(" ", end="")
        print()

def check_available_moves(board):
    available_moves = []
    size = len(board)
```

```
    for x in range(size):
        for y in range(size):
            available_moves.append(is_occupied(board, x, y))
            #print(len(available_moves))
    return available_moves , len(available_moves)

def check_for_winner(board, player):
    size = len(board)

    # Check rows
    for row in board:
        if "".join(row).count(player * 5) >= 1:
            return True

    # Check columns
    for col in range(size):
        column_values = "".join([board[row][col] for row in range(size)])
        if column_values.count(player * 5) >= 1:
            return True

    # Check diagonals
    for row in range(size - 4):
        for col in range(size - 4):
            diagonal = "".join([board[row + i][col + i] for i in range(5)])
            anti_diagonal = "".join([board[row + i][col + 4 - i] for i in range(5)])
            if diagonal.count(player * 5) >= 1 or anti_diagonal.count(player * 5) >= 1
                return True

    return False
import random

def random_computer_player(board, player):
    available_moves = []
    size = len(board)

    for x in range(size):
        for y in range(size):
            if is_occupied(board, x, y):
                available_moves.append((x, y))

    if available_moves:
        return random.choice(available_moves)
    else:
        return None


def play_game():
    print("Hello There! Welcome to the Game of Gomuku, Please select your options belo
    board_size = 0  # Initialize board size to 0
    board = []  # Initialize an empty board
    current_player = "●"  # Start with the black stone
    game_mode = ""
    game_status = False
    move_status = []

    while True:
        game_menu()
        choice = input("Enter your choice: ")

        if choice == "1":
```

```
            if game_status:
                reset_choice = input("A game is already in progress. \n Enter 1 to res
                if reset_choice == "1":
                    play_game()
                    #board = create_board(board_size)
                    #current_player = "●"  # Reset to black stone
                    #move_history = []  # Reset move history
                else:
                    print("Continuing with the current game.")
            else:
                    if board_size > 0:
                        continue_choice = input("A game is already in progress. \n Ent
                                                "Enter 2 to continue the game: ")
                        if continue_choice== "1":
                            board_size = 0
                            board = []
                            current_player = "●"
                            game_status = False
                            print("Starting a new game.")
                        else:
                            print("Continuing with the current game.")
                    else:
                        while True:
                            board_size = int(input("Enter the board size (minimum 5):
                            if board_size > 4:
                                break
                            else:
                                print("Invalid input. Please enter a board size greate

                        board = create_board(board_size)
                        current_player = "●"  # Reset to black stone
                        while True:
                            game_mode = input("Enter 1 or 2 to select game mode:\n"
                            "1. Player v/s Player \n2. Player v/s Computer:")
                            if game_mode == "1":
                                print("Now you have selected:", game_mode, "\n Player
                                      "\n Proceed by entering 2 to view your game boar
                                break
                            elif game_mode == "2":
                                print("Now you have selected:", game_mode, "\nPlayer v
                                      "\n Proceed by entering 2 to view your game boar
                                break
                            #elif game_mode == "6":
                                #print("Going to Main Menu.")
                                #break

                            else:
                                print("Invalid choice. Please enter 1 or 2.")
                        game_status = True

        elif choice == "2":
            if not game_status:
                print("Game not yet started. Please select option 1 to start the game.
            else:
                print_board(board)
                print("Enter 3 to place a stone")

        elif choice == "3":
            if not game_status:
                print("Game not yet started. Please select option 1 to start the game.
```

```
            else:
                while True:
                    print(f"Current player: {current_player}")
                    move_input = input("Enter row and column indices (e.g. 2 A): ").sp

                    if len(move_input) != 2:
                        print("Invalid input. Please enter valid row and column indice
                        continue

                    x, y = move_input
                    if not x.isdigit() or not y.isalpha():
                        print("Invalid input. Please enter valid row and column indice
                        continue

                    x = int(x)
                    y = ord(y.upper()) - ord('A')  # Convert column to index
                    position = (x, y)

                    if place_on_board(board, current_player, position):
                        move_status.append((current_player, position))  # Save the mov
                        winner = check_for_winner(board, current_player)
                        if winner:
                            print_board(board)
                            print(f"Player {current_player} wins!")
                            board = create_board(board_size)  # Reset the board
                            move_history = []  # Reset move history
                        else:
                            current_player = "o" if current_player == "●" else "●"  #

                        if game_mode == "2" and not winner:
                            computer_move = random_computer_player(board, current_play
                            place_on_board(board, current_player, computer_move)
                            move_status.append((current_player, computer_move))  # Sav
                            winner = check_for_winner(board, current_player)
                            if winner:
                                print_board(board)
                                print(f"Player {current_player} wins!")
                                board = create_board(board_size)  # Reset the board
                                move_status = []  # Reset move history
                            current_player = "o" if current_player == "●" else "●"  #

                        print_board(board)
                        print("Enter 3 to Continuing Playing")
                        break  # Exit the loop if everything is successful


        elif choice == "4":
            if not game_status:
                print("Game not yet started. Please select option 1 to start the game.
            else:
                available_moves = check_available_moves(board)
                print("Available Moves:", available_moves)

        elif choice == "5":
            if not game_status:
                print("Game not yet started. Please select option 1 to start the game.
            else:
                board = create_board(board_size)
                current_player = "●"  # Reset to black stone
                move_status = []  # Reset move history
```

```
                print("Game Reset Complete")

        elif choice == "6":
            print("Exiting the game, Good Bye.")
            break

    else:
        print("Invalid choice. Please select a valid options Available below.")

play_game()
```

Documentation of Optimizations

If you have implemented any optimizations in the above program, please include a list of these optimizations along with a brief explanation for each in this section.

--- End of Assignment 1 ---

In [ ]: