FIT9136 Algorithms and Programming Foundations in Python

2023 Semester 2

Assignment 1

Student name: Bhavna Balakrishnan
Student ID: 33954437
Creation date: 18/08/2023
Last modified date: 25/08/23

In [1]:
```
# Libraries to import (if any)
#importing random library.
import random
```

3.1 Game menu function

In [2]:
```
 # Implement code for 3.1 here

    #Define a variable "options" so that players can input when they enter the game by entering a menu cho
    #Here per instructions the game menu is:

#1. Start a Game
#2. Print the Board
#3. Place a Stone
#4. Reset the Game
#5. Exit

#Start showing the Game Menu for user to make a choice:

print("Welcome to Gomoku Game.Please see Game Menu Options:")
print("1. Start Game")
print("2. Print the Board")
print("3. Place a Stone")
print("4. Reset the Game")
print("5. Exit")

options = input("enter your menu choice to continue in game")

#Now so that the the player can make a choice from the options we can use def function to print the player
#We can use def function and combine a loop for creating a menu using if and elif and else functions to pr

def game_menu():
    if options == "1":
        print("Start a Game")
    elif options == "2":
        print("Print the Board")
    elif options == "3":
        print("Place a stone")
    elif options == "4":
        print("Reset the Game")
    elif options == "5":
        print("Exit")
    else:
        print("Invalid option.Kindly select numbers from 1 to 5.")

#Print the game_menu() as seen in code
game_menu()
```

```
Welcome to Gomoku Game.Please see Game Menu Options:
1. Start Game
2. Print the Board
3. Place a Stone
4. Reset the Game
5. Exit
enter your menu choice to continue in game1
Start a Game
```

In [3]:
```
# Test code for 3.1 here [The code in this cell should be commented]


    #Define a variable "options" so that players can input when they enter the game by entering a menu cho
    #Here per instructions the game menu is:

#1. Start a Game
#2. Print the Board
#3. Place a Stone
#4. Reset the Game
#5. Exit

#Start showing the Game Menu for user to make a choice:

#print("Welcome to Gomoku Game.Please see Game Menu Options:")
#print("1. Start Game")
#print("2. Print the Board")
#print("3. Place a Stone")
#print("4. Reset the Game")
#print("5. Exit")

#options = input("enter your menu choice to continue in game")

#We can use def function and combine a loop for creating a menu using if and elif and else functions per i

#def game_menu():
   # if options == "1":
       # print("Start a Game")
   # elif options == "2":
       # print("Print the Board")
   # elif options == "3":
       # print("Place a stone")
   # elif options == "4":
       # print("Reset the Game")
   # elif options == "5":
       # print("Exit")
   # else:
       # print("Invalid option.Kindly select numbers from 1 to 5.")

#Print the game_menu() as seen in code
#game_menu()
```

3.2 Creating the Board

In [3]:
```
 # Implement code for 3.2 here

 #invoke a def function 'create_board' with the argument size where we will plug in 9 for a 9x9 board.

def create_board(size):

#initialize arrays to define the row and column sizes
#Two lists created as follows-
# 'r' will be the row indices from 0 to size-1
# 'c' will be the column with the labels A till size-1 using ASCII variables.
#ord function returns UNICODE symbols

    r = list(range(size))
    c = [chr(ord('A') + i) for i in range(size)]

#Make a dictionary board to identify the board positions like (row,column) as tuples because tuples are im

    board = {}
#Iterate through rows 'r' and columns 'j'
    for i in r:
        for j in c:
            board[(i, j)] = 0

#return finished board after populating once the loops have run

    return board

board_size = 9  # For a 9 by 9 board
board = create_board(board_size) #now call the def function with size as 9
print(board)
```

```
{(0, 'A'): 0, (0, 'B'): 0, (0, 'C'): 0, (0, 'D'): 0, (0, 'E'): 0, (0, 'F'): 0, (0, 'G'): 0, (0, 'H'): 0,
(0, 'I'): 0, (1, 'A'): 0, (1, 'B'): 0, (1, 'C'): 0, (1, 'D'): 0, (1, 'E'): 0, (1, 'F'): 0, (1, 'G'): 0, (
1, 'H'): 0, (1, 'I'): 0, (2, 'A'): 0, (2, 'B'): 0, (2, 'C'): 0, (2, 'D'): 0, (2, 'E'): 0, (2, 'F'): 0, (2
, 'G'): 0, (2, 'H'): 0, (2, 'I'): 0, (3, 'A'): 0, (3, 'B'): 0, (3, 'C'): 0, (3, 'D'): 0, (3, 'E'): 0, (3,
'F'): 0, (3, 'G'): 0, (3, 'H'): 0, (3, 'I'): 0, (4, 'A'): 0, (4, 'B'): 0, (4, 'C'): 0, (4, 'D'): 0, (4, '
E'): 0, (4, 'F'): 0, (4, 'G'): 0, (4, 'H'): 0, (4, 'I'): 0, (5, 'A'): 0, (5, 'B'): 0, (5, 'C'): 0, (5, 'D
'): 0, (5, 'E'): 0, (5, 'F'): 0, (5, 'G'): 0, (5, 'H'): 0, (5, 'I'): 0, (6, 'A'): 0, (6, 'B'): 0, (6, 'C'
): 0, (6, 'D'): 0, (6, 'E'): 0, (6, 'F'): 0, (6, 'G'): 0, (6, 'H'): 0, (6, 'I'): 0, (7, 'A'): 0, (7, 'B')
: 0, (7, 'C'): 0, (7, 'D'): 0, (7, 'E'): 0, (7, 'F'): 0, (7, 'G'): 0, (7, 'H'): 0, (7, 'I'): 0, (8, 'A'):
0, (8, 'B'): 0, (8, 'C'): 0, (8, 'D'): 0, (8, 'E'): 0, (8, 'F'): 0, (8, 'G'): 0, (8, 'H'): 0, (8, 'I'): 0
}
```

In [5]:
```
# Test code for 3.2 here [The code in this cell should be commented]

#defines a function called create_board that takes an argument, size, which represents the dimensions of t
#def create_board(size):

#initialize arrays to define the row and column sizes

   # r = list(range(size))
   # c = [chr(ord('A') + i) for i in range(size)]

#Intialize an empty dictionary to stor player moves and identity here it 0 means empty,1 is black and 2 is

   # board = {}

   #for i in r:
        #for j in c:
            #board[(i, j)] = 0

    #return board

#board_size = 9  # For a 9 by 9 board
#board = create_board(board_size)
#print(board)
```

3.3 Is the target position occupied?

In [4]:
```
 # Implement code for 3.3 here

#Use def function to identify if there are occupied positions on the board:
#Here we use three arguments where board will represent the dictionary we initialized in 3.2.
#'x' is row position and 'y' is column position.

def is_occupied(board, x, y):

#check if x,y are present in the board dictionary's key if position occupied:

    if (x, y) in board:

#Check if unoccupied and return false
        if board[(x, y)] == 0:
            return False
#Check if occupied and return true
        else:
            return True
#If invalid inputs return false as they do not exist on board
    else:
        return False

# Plug in board size value as 9
size = 9
board = create_board(size)
board[(0, 'A')] = 1  # Indicates that there is player Occupying the cell at (0, 'A')

#Taking an example to see if the code runs

print(is_occupied(board, 0, 'A'))  # Should print True
print(is_occupied(board, 3, 'A'))  # Should print False
```

```
True
False
```

In [ ]:
```
# Test code for 3.3 here [The code in this cell should be commented]

#Use def function to identify if there are occupied positions on the board:
#Here we use three arguments where board will represent the dictionary we initialized in 3.2.
#'x' is row position and 'y' is column position.

#def is_occupied(board, x, y):
    #if (x, y) in board:
        #if board[(x, y)] == 0:
            #return False
        #else:
            #return True
    #else:
        #return False

# Plug in board size value as 9
#size = 9
#board = create_board(size)
#board[(0, 'A')] = 1  # Occupying the cell at (0, 'A')

#Taking an example to see if the code runs

#print(is_occupied(board, 0, 'A'))  # Should print True
#print(is_occupied(board, 3, 'A'))  # Should print False
```

3.4 Placing a Stone at a Specific Intersection

In [5]:
```
 # Implement code for 3.4 here

#now we will go back to the previous def functions 'create_board' and 'is_occupied'


def create_board(size):

#this will generate the dictionary with rows and colums as numbers and alphabets as labels.
    return {(i, chr(ord('A') + j)): 0 for i in range(size) for j in range(size)}

#Check occupancy of position on the board:
def is_occupied(board, x, y):
    return (x, y) in board and board[(x, y)] != 0

#Def function to place stones on board using arguements of board,stone value and position
def place_on_board(board, stone, position):

# take x,y as position for each tuple in dictionary
    x, y = position
    if (x, y) in board and not is_occupied(board, x, y):

#Assign values in dictionary as 1 for black stone and 2 for white.

        board[(x, y)] = 1 if stone == "●" else 2

#return true if the placement is successful and false if already occupied or position is invalid.
        return True
    return False

board_size = 9
board = create_board(board_size)

#Examples To depict placing of stone on the board-

result1 = place_on_board(board, "●", (0, 'A'))
result2 = place_on_board(board, "○", (1, 'B'))

#To show how the stone can not be kept in an already occupied spot-

result3 = place_on_board(board, "○", (0, 'A'))



print(result1) #Should be True
print(result2) #Should be True
print(result3) #Should be False
```

```
True
True
False
```

In [7]:
```
# Test code for 3.4 here [The code in this cell should be commented]

#  #now we will go back to the previous def functions 'create_board' and 'is_occupied'


# def create_board(size):

# #this will generate the dictionary with rows and colums as numbers and alphabets as labels.
#     return {(i, chr(ord('A') + j)): 0 for i in range(size) for j in range(size)}

# #Check occupancy of position on the board:
# def is_occupied(board, x, y):
#     return (x, y) in board and board[(x, y)] != 0

# #Def function to place stones on board using arguements of board,stone value and position
# def place_on_board(board, stone, position):

# # take x,y as position for each tuple in dictionary
#     x, y = position
#     if (x, y) in board and not is_occupied(board, x, y):

# #Assign values in dictionary as 1 for black stone and 2 for white.

#         board[(x, y)] = 1 if stone == "●" else 2

# #return true if the placement is successful and false if already occupied or position is invalid.
#         return True
#     return False

# board_size = 9
# board = create_board(board_size)

# #Examples To depict placing of stone on the board-

# result1 = place_on_board(board, "●", (0, 'A'))
# result2 = place_on_board(board, "○", (1, 'B'))

# #To show how the stone can not be kept in an already occupied spot-

# result3 = place_on_board(board, "○", (0, 'A'))



# print(result1) #Should be True
# print(result2) #Should be True
# print(result3) #Should be False
```

3.5 Printing the Board

In [6]:
```
 # Implement code for 3.5 here

# taking def function calling the dictionary board to identify positions and placements

def print_board(board):
    value_index_col = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
#to have list with column labels from A to I for a 9 by 9 board.
    size = 9
#For each value of i print label values in uppercase
    for i in value_index_col:
#end column labels with four spaces and start to next line
        print(i.upper(), end='    ')
    print('\n')
# Run loops to decide iterations over which the characters should print given stone positions and use char
    for row in range(size):
        for col in value_index_col:
            cell_value = board[(row, col)]

#Determine if it is not column:
            if col != value_index_col[-1]:

#What to print with given the cell value:

                if cell_value == 0:
                    print("0 -- ", end='')
                elif cell_value == 1:
                    print("● -- ", end='')
                elif cell_value == 2:
                    print("O -- ", end='')

#Determine last column:

            if col == value_index_col[-1]:
                if cell_value == 0:
                    print("0", end='')
                elif cell_value == 1:
                    print("●", end='')
                elif cell_value == 2:
                    print("O", end='')

#Print Row numbers:

        print("  " + str(row), end="")

#Print new line character to move to next row:

        print("")
        if row != size - 1:
            print("|    " * size)

# Example usage:
board_size = 9
board = create_board(board_size)

# Placing stones on the board
place_on_board(board, "●", (0, 'B'))
place_on_board(board, "O", (1, 'A'))

# Printing the board
print_board(board)
```

```
A    B    C    D    E    F    G    H    I

0 -- ● -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0  0
|    |    |    |    |    |    |    |    |
O -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0  1
|    |    |    |    |    |    |    |    |
0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0  2
|    |    |    |    |    |    |    |    |
0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0  3
|    |    |    |    |    |    |    |    |
0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0  4
|    |    |    |    |    |    |    |    |
0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0  5
|    |    |    |    |    |    |    |    |
0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0  6
|    |    |    |    |    |    |    |    |
0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0  7
|    |    |    |    |    |    |    |    |
0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0  8
```

In [ ]:
```
# Test code for 3.5 here [The code in this cell should be commented]

# # taking def function calling the dictionary board to identify positions and placements

# def print_board(board):
#     value_index_col = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
# #to have list with column labels from A to I for a 9 by 9 board.
#     size = 9
# #For each value of i print label values in uppercase
#     for i in value_index_col:
# #end column labels with four spaces and start to next line
#         print(i.upper(), end='    ')
#     print('\n')
# # Run loops to decide iterations over which the characters should print given stone positions and use ch
#     for row in range(size):
#         for col in value_index_col:
#             cell_value = board[(row, col)]

# #Determine if it is not column:
#             if col != value_index_col[-1]:

# #What to print with given the cell value:

#                 if cell_value == 0:
#                     print("0 -- ", end='')
#                 elif cell_value == 1:
#                     print("● -- ", end='')
#                 elif cell_value == 2:
#                     print("○ -- ", end='')

# #Determine last column:

#             if col == value_index_col[-1]:
#                 if cell_value == 0:
#                     print("0", end='')
#                 elif cell_value == 1:
#                     print("●", end='')
#                 elif cell_value == 2:
#                     print("○", end='')

# #Print Row numbers:

#         print("  " + str(row), end="")

# #Print new line character to move to next row:

#         print("")
#         if row != size - 1:
#             print("|    " * size)

# # Example usage:
# board_size = 9
# board = create_board(board_size)

# # Placing stones on the board
# place_on_board(board, "●", (0, 'B'))
# place_on_board(board, "○", (1, 'A'))

# # Printing the board
# print_board(board)
```

3.6 Check Available Moves

In [7]:
```
 # Implement code for 3.6 here

#Create def function with the board dictionary as the arguement

def check_available_moves(board):

#Initialize an empty list available_moves to store the moves to make
    available_moves = []
    size = int(len(board) ** 0.5)  # Take square root as the board is a square dictionary

#create for loops to iterate over rows and columns

    for row in range(size):
        for col in range(size):

#initialize variable position to depict tuple of current stone position:

            position = (row, chr(ord('A') + col))
#so now is_occupied recognizes whether position is occupied or not. if not it will append to list availabl

            if is_occupied(board, row, chr(ord('A') + col)):
               continue
            available_moves.append(position)

    return available_moves

# Example usage for a 9 by 9 board:
board_size = 9
board = create_board(board_size)

# Placing stones on the board for testing
place_on_board(board, "●", (0, 'A'))
place_on_board(board, "○", (1, 'B'))

# Checking available moves
moves = check_available_moves(board)
print(moves)
```

```
[(0, 'B'), (0, 'C'), (0, 'D'), (0, 'E'), (0, 'F'), (0, 'G'), (0, 'H'), (0, 'I'), (1, 'A'), (1, 'C'), (1,
'D'), (1, 'E'), (1, 'F'), (1, 'G'), (1, 'H'), (1, 'I'), (2, 'A'), (2, 'B'), (2, 'C'), (2, 'D'), (2, 'E'),
(2, 'F'), (2, 'G'), (2, 'H'), (2, 'I'), (3, 'A'), (3, 'B'), (3, 'C'), (3, 'D'), (3, 'E'), (3, 'F'), (3, '
G'), (3, 'H'), (3, 'I'), (4, 'A'), (4, 'B'), (4, 'C'), (4, 'D'), (4, 'E'), (4, 'F'), (4, 'G'), (4, 'H'),
(4, 'I'), (5, 'A'), (5, 'B'), (5, 'C'), (5, 'D'), (5, 'E'), (5, 'F'), (5, 'G'), (5, 'H'), (5, 'I'), (6, '
A'), (6, 'B'), (6, 'C'), (6, 'D'), (6, 'E'), (6, 'F'), (6, 'G'), (6, 'H'), (6, 'I'), (7, 'A'), (7, 'B'),
(7, 'C'), (7, 'D'), (7, 'E'), (7, 'F'), (7, 'G'), (7, 'H'), (7, 'I'), (8, 'A'), (8, 'B'), (8, 'C'), (8, '
D'), (8, 'E'), (8, 'F'), (8, 'G'), (8, 'H'), (8, 'I')]
```

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

# #Create def function with the board dictionary as the arguement

# def check_available_moves(board):

# #Initialize an empty list available_moves to store the moves to make
#     available_moves = []
#     size = int(len(board) ** 0.5)  # Take square root as the board is a square dictionary

# #create for loops to iterate over rows and columns

#     for row in range(size):
#         for col in range(size):

# #initialize variable position to depict tuple of current stone position:

#             position = (row, chr(ord('A') + col))
# #so now is_occupied recognizes whether position is occupied or not. if not it will append to list availa

#             if is_occupied(board, row, chr(ord('A') + col)):
#                continue
#             available_moves.append(position)

#     return available_moves

# # Example usage for a 9 by 9 board:
# board_size = 9
# board = create_board(board_size)

# # Placing stones on the board for testing
# place_on_board(board, "●", (0, 'A'))
# place_on_board(board, "○", (1, 'B'))

# # Checking available moves
# moves = check_available_moves(board)
# print(moves)
```

3.7 Check for the Winner

In [8]:
```
 # Implement code for 3.7 here

# create a def function check_winner with the arguement using the dictionary 'board'

def check_for_winner(board):
    size = int(len(board) ** 0.5)  # As the board is a square dictionary take square root as in a dictiona
#create for loops to iterate through rows and colums to see if position occupied and decide the colour.

    for row in range(size):
        for col in range(size):
            if is_occupied(board, row, col):
                stone_color = board[(row, col)]

#To search for 4 consecutive stone positions

                directions = [(1, 0), (0, 1), (1, 1), (1, -1)]

#Look for consecutive 5 stones of same colour.

                for a, b in directions:
                    is_winning = True

#Look for 5 consecutive stones:

                    for i in range(5):
                        x, y = row + i * a, chr(ord('A') + col + i * b)
                        if not (0 <= x < size and 0 <= ord(y) - ord('A') < size) or board[(x, y)] != stone
                            is_winning = False
                            break
                    if is_winning:
                        return stone_color

#If there is no winner then check for more available moves and  draw if none.

    available_moves = check_available_moves(board)
    return "Draw" if len(available_moves) == 0 else None

# Example usage for a 9 by 9 board:
board_size = 9
board = create_board(board_size)

# Placing stones on the board for testing
place_on_board(board, "●", (0, 'A'))
place_on_board(board, "○", (1, 'B'))
place_on_board(board, "●", (1, 'A'))
place_on_board(board, "○", (2, 'B'))
place_on_board(board, "●", (2, 'A'))
place_on_board(board, "○", (3, 'B'))
place_on_board(board, "●", (3, 'A'))
place_on_board(board, "○", (4, 'B'))

# Checking the winner of the game
winner = check_for_winner(board)
if winner:
    print(f"Winner: {winner}")
else:
    print("No winner yet.")
```

```
No winner yet.
```

In [ ]:
```
# Test code for 3.7 here [The code in this cell should be commented]

# # create a def function check_winner with the arguement using the dictionary 'board'

# def check_winner(board):
#     size = int(len(board) ** 0.5)  # As the board is a square dictionary take square root as in a dictic
# #create for loops to iterate through rows and colums to see if position occupied and decide the colour.

#     for row in range(size):
#         for col in range(size):
#             if is_occupied(board, row, col):
#                 stone_color = board[(row, col)]

# #To search for 4 consecutive stone positions

#                 directions = [(1, 0), (0, 1), (1, 1), (1, -1)]

# #Look for consecutive 5 stones of same colour.

#                 for a, b in directions:
#                     is_winning = True

# #Look for 5 consecutive stones:

#                     for i in range(5):
#                         x, y = row + i * a, chr(ord('A') + col + i * b)
#                         if not (0 <= x < size and 0 <= ord(y) - ord('A') < size) or board[(x, y)] != sto
#                             is_winning = False
#                             break
#                     if is_winning:
#                         return stone_color

# #If there is no winner then check for more available moves and  draw if none.

#     available_moves = check_available_moves(board)
#     return "Draw" if len(available_moves) == 0 else None

# # Example usage for a 9 by 9 board:
# board_size = 9
# board = create_board(board_size)

# # Placing stones on the board for testing
# place_on_board(board, "●", (0, 'A'))
# place_on_board(board, "○", (1, 'B'))
# place_on_board(board, "●", (1, 'A'))
# place_on_board(board, "○", (2, 'B'))
# place_on_board(board, "●", (2, 'A'))
# place_on_board(board, "○", (3, 'B'))
# place_on_board(board, "●", (3, 'A'))
# place_on_board(board, "○", (4, 'B'))

# # Checking the winner of the game
# winner = check_winner(board)
# if winner:
#     print(f"Winner: {winner}")
# else:
#     print("No winner yet.")
```

3.8 Random Computer Player

In [9]:
```
 # Implement code for 3.8 here

#import random to generate random choices from module.

import random

#def function with arguements of the board dictionary and the last move by human player as 'player_move'

def random_computer_player(board, player_move):
    size = int(len(board) ** 0.5)  # As the board is a square dictionary

#now to compute valid moves after the player makes a move-

    x, y = player_move
    valid_moves = []

#cover all adjacent positions in p and q

    for p in range(-1, 2):
        for q in range(-1, 2):

#create moves to challenge the player positions as new_x and new_y

            new_x, new_y = x + p, chr(ord(y) + q)
            if 0 <= new_x < size and 'A' <= new_y <= chr(ord('A') + size - 1) and not is_occupied(board, n
                valid_moves.append((new_x, new_y))

#computer checks for more valid moves or else goes with one of the available moves

    if len(valid_moves) == 0:
        available_moves = check_available_moves(board)
        return random.choice(available_moves)
    else:
        return random.choice(valid_moves)

# Example usage for a 9 by 9 board:
board_size = 9
board = create_board(board_size)

# Placing stones on the board for testing
place_on_board(board, "●", (4, 'E'))
player_move = (4, 'E')

# Getting the computer's next move
computer_next_move = random_computer_player(board, player_move)
print(f"Computer's next move: {computer_next_move}")
```

```
Computer's next move: (5, 'F')
```

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

# #import random to generate random choices from module.

# import random

# #def function with arguements of the board dictionary and the last move by human player as 'player_move'

# def random_computer_player(board, player_move):
#     size = int(len(board) ** 0.5)  # As the board is a square dictionary

# #now to compute valid moves after the player makes a move-

#     x, y = player_move
#     valid_moves = []

# #cover all adjacent positions in p and q

#     for p in range(-1, 2):
#         for q in range(-1, 2):

# #create moves to challenge the player positions as new_x and new_y

#             new_x, new_y = x + p, chr(ord(y) + q)
#             if 0 <= new_x < size and 'A' <= new_y <= chr(ord('A') + size - 1) and not is_occupied(board,
#                 valid_moves.append((new_x, new_y))

# #computer checks for more valid moves or else goes with one of the available moves

#     if len(valid_moves) == 0:
#         available_moves = check_available_moves(board)
#         return random.choice(available_moves)
#     else:
#         return random.choice(valid_moves)

# # Example usage for a 9 by 9 board:
# board_size = 9
# board = create_board(board_size)

# # Placing stones on the board for testing
# place_on_board(board, "●", (4, 'E'))
# player_move = (4, 'E')

# # Getting the computer's next move
# computer_next_move = random_computer_player(board, player_move)
# print(f"Computer's next move: {computer_next_move}")
```

3.9 Play Game

In [11]:
```
  # Defining a function called create_board (assuming it's defined elsewhere)
def create_board(size):
    # Implementation of create_board function
    board = {}  # Replace this with your actual board creation logic
    return board

# Defining a function called is_occupied (assuming it's defined elsewhere)
def is_occupied(board, row, column):
    # Implementation of is_occupied function
    return board.get((row, column)) is not None

# Defining a function called place_on_board (assuming it's defined elsewhere)
def place_on_board(board, stone, position):
    # Implementation of place_on_board function
    board[position] = stone

# Defining a function called computer_move (assuming it's defined elsewhere)
def computer_move(board, position):
    # Implementation of computer_move function
    # Replace this with your actual computer move logic
    return (position[0], chr(ord(position[1]) + 1))

# Defining a function called check_winner (assuming it's defined elsewhere)
def check_winner(board):
    # Implementation of check_winner function
    # Replace this with your actual winner checking logic
    return None  # Return the winner's stone ("●" or "○") or "Draw"

# Defining a function called print_board
def print_board(board, size):
    # Implementation of print_board function
```

```
    value_index_col = [chr(ord('A') + i) for i in range(size)]

    for i in value_index_col:
        print(i.upper(), end='    ')
    print('\n')

    for row in range(size):
        for col in value_index_col:
            cell_value = board.get((row, col), 0)

            if col != value_index_col[-1]:
                if cell_value == 0:
                    print("0 -- ", end='')
                elif cell_value == 1:
                    print("● -- ", end='')
                elif cell_value == 2:
                    print("○ -- ", end='')
            if col == value_index_col[-1]:
                if cell_value == 0:
                    print("0", end='')
                elif cell_value == 1:
                    print("●", end='')
                elif cell_value == 2:
                    print("○", end='')

        print("  " + str(row), end="")
        print("")
        if row != size - 1:
            print("|    " * size)

# Defining a function called play_game
def play_game():
    # Initializing variables 'board' and 'mode' as none as neither of them have been initiated
    board = None
    mode = None

    # Create an infinite loop to process player inputs
    while True:
        # Displaying game menu options
        print("Game Menu Options:")
        print("1. Start Game")
        print("2. Print the Board")
        print("3. Place a Stone")
        print("4. Reset the Game")
        print("5. Exit")

        # Ask the user for their choice
        choice = input("Enter your menu choice: ")

        # Menu choices
        if choice == "1":
            # Checking if a game is already in progress
            if board is not None:
                print("A game is already in progress.")
                reset_choice = input("Do you want to reset and restart the game? (y/n): ")
                if reset_choice.lower() == "y":
                    board = None
                else:
                    continue

            # Board size and validation of it
            size = int(input("Enter the board size (9, 13, or 15): "))
            if size not in [9, 13, 15]:
                print("Invalid board size.")
                continue

            # Choose game mode
            mode_choice = input("Enter the mode (PvP or PvC): ")
            if mode_choice.lower() not in ["pvp", "pvc"]:
                print("Invalid mode choice.")
                continue
            mode = mode_choice.lower()

            # Creating the game board and notifying the user
            board = create_board(size)
            print("Game started!")

        elif choice == "2":
            # Printing the board if a game is in progress
            if board is None:
                print("No game in progress.")
            else:
                print_board(board, size)
```

```
        elif choice == "3":
            # Placing stones on the board
            if board is None:
                print("No game in progress.")
                continue

            # Handling player vs. player mode
            if mode == "pvp":
                player_stone = "●"
                player_move = input("Enter your move (row column): ").split()
                row = int(player_move[0])
                column = player_move[1].upper()
                position = (row, column)

                if is_occupied(board, row, column):
                    print("Invalid move. Position already occupied.")
                else:
                    place_on_board(board, player_stone, position)
                    print_board(board, size)

            # Handling player vs. computer mode
            elif mode == "pvc":
                player_stone = "●"
                player_move = input("Enter your move (row column): ").split()
                row = int(player_move[0])
                column = player_move[1].upper()
                position = (row, column)

                if is_occupied(board, row, column):
                    print("Invalid move. Position already occupied.")
                else:
                    place_on_board(board, player_stone, position)
                    computer_move_position = computer_move(board, position)
                    computer_stone = "○"
                    place_on_board(board, computer_stone, computer_move_position)
                    print_board(board, size)

            # Checking for a winner and handling game outcome
            winner = check_winner(board)
            if winner:
                if winner == "Draw":
                    print("It's a draw!")
                else:
                    print(f"Winner: {winner}")
                board = None

        elif choice == "4":
            # Resetting the game
            board = None
            mode = None
            print("Game reset.")

        elif choice == "5":
            # Exiting the game
            print("Exiting the game.")
            break

        else:
            print("Invalid choice. Please select a valid option.")

# Starting the game loop by calling the play_game function
play_game()
```

```
Game Menu Options:
1. Start Game
2. Print the Board
3. Place a Stone
4. Reset the Game
5. Exit
Enter your menu choice: 1
Enter the board size (9, 13, or 15): 13
Enter the mode (PvP or PvC): pvp
Game started!
Game Menu Options:
1. Start Game
2. Print the Board
3. Place a Stone
4. Reset the Game
5. Exit
Enter your menu choice: 2
A    B    C    D    E    F    G    H    I    J    K    L    M
```

```
0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0  0
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |
0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0  1
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |
0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0  2
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |
0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0  3
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |
0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0  4
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |
0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0  5
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |
0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0  6
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |
0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0  7
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |
0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0  8
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |
0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0  9
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |
0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0  10
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |
0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0  11
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |
0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0  12
Game Menu Options:
1. Start Game
2. Print the Board
3. Place a Stone
4. Reset the Game
5. Exit
Enter your menu choice: 3
Enter your move (row column): 1 a
A    B    C    D    E    F    G    H    I    J    K    L    M

0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0  0
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |
0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0  1
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |
0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0  2
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |
0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0  3
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |
0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0  4
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |
0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0  5
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |
0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0  6
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |
0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0  7
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |
0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0  8
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |
0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0  9
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |
0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0  10
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |
0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0  11
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |
0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0 -- 0  12
Game Menu Options:
1. Start Game
2. Print the Board
3. Place a Stone
4. Reset the Game
5. Exit
Enter your menu choice: 4
Game reset.
Game Menu Options:
1. Start Game
2. Print the Board
3. Place a Stone
4. Reset the Game
5. Exit
Enter your menu choice: 5
Exiting the game.
```

In [ ]:
```
# #  # Implement code for 3.9 here

# # Defining a function called create_board (assuming it's defined elsewhere)
# def create_board(size):
#     board = {}  # Initialize an empty dictionary to represent the game board
#     return board
```

```
# # Defining a function called is_occupied (assuming it's defined elsewhere)
# def is_occupied(board, row, column):
#     return board.get((row, column)) is not None
#     # Returns True if the cell at (row, column) on the board is occupied, else False

# # Defining a function called place_on_board (assuming it's defined elsewhere)
# def place_on_board(board, stone, position):
#     board[position] = stone
#     # Places the given 'stone' (player's symbol) at the specified 'position' on the board

# # Defining a function called computer_move (assuming it's defined elsewhere)
# def computer_move(board, position):
#     return (position[0], chr(ord(position[1]) + 1))
#     # Generates a computer move based on the player's move by shifting the column position

# # Defining a function called check_winner (assuming it's defined elsewhere)
# def check_winner(board):
#     return None  # Placeholder for winner checking logic, returns the winner's stone ("●" or "○") or "Dr

# # Defining a function called print_board
# def print_board(board, size):
#     value_index_col = [chr(ord('A') + i) for i in range(size)]
#     # Creates column labels like 'A', 'B', 'C', ... based on the board's size

#     for i in value_index_col:
#         print(i.upper(), end='    ')  # Print column labels in uppercase with spacing
#     print('\n')

#     for row in range(size):
#         for col in value_index_col:
#             cell_value = board.get((row, col), 0)
#             # Get the value of the cell at (row, col) or return 0 if it's unoccupied

#             if col != value_index_col[-1]:
#                 if cell_value == 0:
#                     print("0 -- ", end='')
#                 elif cell_value == 1:
#                     print("● -- ", end='')
#                 elif cell_value == 2:
#                     print("○ -- ", end='')
#             if col == value_index_col[-1]:
#                 if cell_value == 0:
#                     print("0", end='')
#                 elif cell_value == 1:
#                     print("●", end='')
#                 elif cell_value == 2:
#                     print("○", end='')

#         print("  " + str(row), end="")
#         print("")  # Print the row index and move to the next line
#         if row != size - 1:
#             print("|    " * size)  # Print vertical separators for rows

# # Defining a function called play_game
# def play_game():
#     board = None  # Initialize variables for the game board and mode
#     mode = None

#     while True:  # Create an infinite loop to process player inputs
#         # Displaying game menu options
#         print("Game Menu Options:")
#         # Menu options are displayed for the player to choose from
#         print("1. Start Game")
#         print("2. Print the Board")
#         print("3. Place a Stone")
#         print("4. Reset the Game")
#         print("5. Exit")

#         # Ask the user for their choice
#         choice = input("Enter your menu choice: ")

#         # Handling the different menu choices
#         if choice == "1":
#             # Code for starting a new game
#             # ...
#         elif choice == "2":
#             # Code for printing the board
#             # ...
#         elif choice == "3":
#             # Code for placing stones on the board
#             # ...
#         elif choice == "4":
```

```
#             # Code for resetting the game
#             # ...
#         elif choice == "5":
#             # Exiting the game
#             print("Exiting the game.")
#             break
#         else:
#             print("Invalid choice. Please select a valid option.")

# # Starting the game loop by calling the play_game function
# play_game()
```

In [ ]:
```
#Run the game (Your tutor will run this cell to start playing the game)
```

REFERENCES

Python Software Foundation. (n.d.). Python Tutorial. Python.org. https://docs.python.org/3/tutorial/index.html Pygame Community. (n.d.). Pygame Tutorials. Pygame. https://www.pygame.org/wiki/tutorials Sweigart, A. (2012). Invent with Python: Making Games with Python & Pygame. https://inventwithpython.com/pygame/ Elgabrys, O. (n.d.). Creating a Gomoku Game in Python (Part 1-9). Medium. https://medium.com/omarelgabrys-blog/creating-a-gomoku-game-in-python-part-1-9-815ea7d6981e Real Python. (n.d.). Introduction to Python Control Flow. https://realpython.com/courses/introduction-python-control-flow/ Real Python. (n.d.). Introduction to Functions in Python. https://realpython.com/courses/introduction-functions-python/ Real Python. (n.d.). Python Lists and Tuples. https://realpython.com/courses/python-lists-tuples/ Real Python. (n.d.). Python Input and Output. https://realpython.com/courses/python-input-output/ Real Python. (n.d.). Python random Module. https://realpython.com/courses/python-random/

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 ---