FIT9136 Algorithms and Programming Foundations in Python

2023 Semester 2

Assignment 1

Student name: Ashwin Gururaj
Student ID: 33921199
Creation date: 09/08/2023
Last modified date: 25/08/2023

In [ ]:

```
# Libraries to import (if any)
import random
```

3.1 Game menu function

In [ ]:

```
# Implement code for 3.1 here
def game_menu():
    """
    Description:
            Display the game options available for the user.

    Parameter:
            No parameter/argument

    Return:
            No return type
    """
    print("*** Welcome to Gomoku ***")
    print("Main Menu")
    print("1. Start a new game")
    print("2. Print the Board")
    print("3. Place a Stone")
    print("4. Reset the Game")
    print("5. Exit the game")
```

In [ ]:

```
# Test code for 3.1 here [The code in this cell should be commented]
# game_menu()
# This is the expected output

#     print("*** Welcome to Gomoku ***")
#     print("Main Menu")
#     print("1. Start a new game")
#     print("2. Print the Board")
#     print("3. Place a Stone")
#     print("4. Reset the Game")
#     print("5. Exit the game")
```

3.2 Creating the Board

In [ ]:

```
# Implement code for 3.2 here
def create_board(size):
    """
    Description:
            Create a game board with unoccupied intersections.
            Using multi-dimensional array to create a unoccupied empty game board
            Considering initially the board has unocupied intersections.

    Parameter:
            Int
            Size of the board (For both rows and columns)

    Return: List
            Game Board created.
    """
    game_board = []
    for i in range(size):
        row = []
        for j in range(size):
            row.append(' ')
        game_board.append(row)
    return game_board
```

In [ ]:

```
# Test code for 3.2 here [The code in this cell should be commented]
#create_board(9)

#This is the expected output
# [[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#  [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#  [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#  [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#  [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#  [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#  [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#  [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#  [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']]
```

3.3 Is the target position occupied?

In [ ]:

```
# Implement code for 3.3 here
def is_occupied(board, x, y):
    """
    Description:
            To check whether a specific position on the board is occupied by a stone

    Parameters:
            board: The current state of the board.
            x: The row index.
            y: The column index.

    Return:
            Boolean
            True if the position is occupied, else False.
    """

    if board[x][y] == " ":
        return False
    else:
        return True
```

In [ ]:

```
# Test code for 3.3 here [The code in this cell should be commented]
# board = [[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#   [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#   [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#   [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#   [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#   [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#   [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#   [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#   [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']]
# is_occupied(board, 3, 4)

#This is the expected output
#False - since the place is not occupied
#True - since the place is occupied
```

3.4 Placing a Stone at a Specific Intersection

In [ ]:

```
# Implement code for 3.4 here
def place_on_board(board, stone, position):
    """
    Description:
            To place the stone on the board on a specific position

    Parameters:
            Board: The current state of the board.
            Stone: The value of stone, either "●" or "o".
            Position: The position of stone. It would be a tuple of strings
                      with first index being row value and second index being third

    Return:
            Boolean
            True, if the stone is placed is successfully
            False, if the stone is impossible to place
    """
    row_index = int(position[0])
    col_index = ord(position[1].upper()) - ord('A')
    size = len(board)

    if 0 <= row_index < size and 0 <= col_index < size:
        if is_occupied(board, row_index, col_index):
            print("Position is occupied.")
            return False
        else:
            board[row_index][col_index] = stone
            print("Position is not occupied.")
            return True
```

In [ ]:

```
# Test code for 3.4 here [The code in this cell should be commented]
# board = [[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#   [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#   [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#   [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#   [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#   [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#   [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#   [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#   [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']]
# is_occupied(board, 3, 4)
# place_on_board(board,"●",("2","A"))

# This is the expected output
# True
```

3.5 Printing the Board

In [ ]:

```
 # Implement code for 3.5 here
def print_board(board):
    """
    Description:
            To visualize the board in human readable format.

    Parameter:
            Board: The current state of the board.

    Return:
            None
    """
    size = len(board)
    col_headers = "  ".join([chr(65 + i) for i in range(size)])  # A=65, B=66, ...
    print(f"{col_headers}")
    for i in range(size):
        row_string = ""
        for j in range(size):
            if board[i][j] == ' ' and j == size - 1:
                row_string += " "
            elif board[i][j] == ' ':
                row_string += " --"
            elif j == size - 1:
                row_string += board[i][j]
            else:
                row_string += board[i][j] + "--"
        print(row_string + " " + str(i))
        if i < size - 1:
            print("|  " * size)
        else:
            break
```

In [ ]:

```
# Test code for 3.5 here [The code in this cell should be commented]
# board = [[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#  [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#  [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#  [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#  [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#  [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#  [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#  [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#  [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']]
# print_board(board)

# This is the expected output
# A  B  C  D  E  F  G  H  I
#  -- -- -- -- -- -- -- --  0
# |  |  |  |  |  |  |  |  |
#  -- -- -- -- -- -- -- --  1
# |  |  |  |  |  |  |  |  |
#  -- -- -- -- -- -- -- --  2
# |  |  |  |  |  |  |  |  |
#  -- -- -- -- -- -- -- --  3
# |  |  |  |  |  |  |  |  |
#  -- -- -- -- -- -- -- --  4
# |  |  |  |  |  |  |  |  |
#  -- -- -- -- -- -- -- --  5
# |  |  |  |  |  |  |  |  |
#  -- -- -- -- -- -- -- --  6
# |  |  |  |  |  |  |  |  |
#  -- -- -- -- -- -- -- --  7
# |  |  |  |  |  |  |  |  |
#  -- -- -- -- -- -- -- --  8
```

3.6 Check Available Moves

In [ ]:

```
 # Implement code for 3.6 here
def check_available_moves(board):
    """
    Description:
            To let the player know about the available moves in the game board.

    Parameter:
            Board: The current state of the board.

    Return:
            List of Tuples
            All the available moves in form tuples inside a list

    """
    moves_available = []
    size = len(board)
    for i in range(size):
        for j in range(size):
            if board[i][j] == ' ':
                moves_available.append((str(i), chr(65 + j)))
    print("Total number of moves available are: ",len(moves_available))
    return moves_available
```

In [ ]:

```
# Test code for 3.6 here [The code in this cell should be commented]
# board = [[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#  [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#  [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#  [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#  [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#  [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#  [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#  [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#  [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']]
# check_available_moves(board)

# This is an example of expected output
# Total number of moves available are:  81
# [('0', 'A'),
#  ('0', 'B'),
#  ('0', 'C'),
#  ('0', 'D'),
#  ('0', 'E'),
#  ('0', 'F'),
#  ('0', 'G'),
#  ('0', 'H'),
#  ('0', 'I'),
#  ('1', 'A'),
#  ('1', 'B'),
#  ('1', 'C'),
#  ('1', 'D'),
```

3.7 Check for the Winner

In [ ]:

```
 # Implement code for 3.7 here
def check_for_winner(board):
    """
    Description:
            Function is to check the winner of the game. Winner is decided on the co
            when a player forms a continuous line of five stones in their colour, ei
            ,vertically or diagonally
    Parameters:
            board: Current state of the game board
    Return:
            1. Return the respective stone when a continuous line of five stones is
            2. Return "Draw" when the board is full but none of the players achieve
            3. Return None when no one wins the game and moves are still available i
    """
    size = len(board)
    is_drawn = True
    for i in range(size):
        for j in range(size):
            stone = board[i][j]
            if stone == " ":
                is_drawn = False
                continue
            if j <= size - 5 and all(board[i][j+k] == stone for k in range(5)):
                return stone
            if i <= size - 5 and all(board[i+k][j] == stone for k in range(5)):
                return stone
            if i <= size - 5 and j <= size - 5 and all(board[i+k][j+k] == stone for
                return stone
            if i <= size - 5 and j >= 4 and all(board[i+k][j-k] == stone for k in ra
                return stone
    if is_drawn:
        return "Draw"
    return None
```

In [ ]:

```
# Test code for 3.7 here [The code in this cell should be commented]
# board = [[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#  ['o', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#  ['o', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#  ['o', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#  ['o', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#  ['o', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#  ['o', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#  [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#  [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']]
# print(check_for_winner(board))

# This is the expected output for horizontal, vertical and diagonal winners
# o
# This is the expected output when a match is a tie breaker
# Draw
# This is the expected output when a no player is a winner and there are moves left
# None
```

3.8 Random Computer Player

In [ ]:

```
 # Implement code for 3.8 here
def random_computer_player(board, player_move):
    """
    Description:
            To allow the computer opponent to counter the player by randomly selecti
            the position in the board based on the player's previous move and
            available positions.

    Paramaters:
            Board: Current state of the game board
            player_move:

    Return:
            Tuple
            Move played by the computer based on player's previous move and availabl
    """
    size = len(board)
    row, col = int(player_move[0]), ord(player_move[1]) - ord('A')
    possible_moves = []
    for i in range(row - 1, row + 2):
        for j in range(col - 1, col + 2):
            if 0 <= i < size and 0 <= j < size and is_occupied(board,i,j)==False:
                possible_moves.append((str(i), chr(j + ord('A'))))
                if (str(i), chr(j + ord('A'))) == player_move:
                    possible_moves.remove((str(i), chr(j + ord('A'))))
    if possible_moves:
        return random.choice(possible_moves)
    else:
        return random.choice(check_available_moves(board))
```

In [ ]:

```
# Test code for 3.8 here [The code in this cell should be commented]
# board = [[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#  [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#  [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#  [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#  [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#  [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#  [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#  [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
#  [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']]
# random_computer_player(board,("2","B"))

# This is the expected output
# ('1', 'B')
```

3.9 Play Game

In [ ]:

```python
# Implement code for 3.9 here
def play_game():
    """
    Description:
        A function to initialize and play the Gomoku game.

    Parameter:
        No parameters.

    Return:
        None
    """
    game_status = ""
    while True:
        valid_modes = ["PvP","PvC","pvp","pvc"]
        game_menu()
        choice = input("Enter your choice: ")
        if choice == "1":
            board_size = int(input("Enter the size of the board: "))
            mode = input("Select mode (PvP or PvC): ")
            while mode not in valid_modes:
                mode = input("Invalid mode. Please enter PvP, PvC, pvp, pvc: ")
            game_board = create_board(board_size)
            print("Hello! Your game being set up. Have fun!")
            while game_status == True:
                user_wish = input("If you wish to restart the game or continue the g
                if user_wish.lower() == "y":
                    new_board = create_board(board_size)
                    print_board(new_board)
        elif choice == "2":
            print_board(game_board)
        elif choice == "3":
            print("Dropping a stone at the point you want.")
            if mode.lower() == "pvp":
                while True:
                    print("")
                    list_of_available_moves = len(check_available_moves(game_board))
                    if list_of_available_moves % 2 != 0:
                        stone = '●'
                        print("It is now Player 1's turn to place a stone.")
                        position = input("Enter the position to place the stone: ")
                        place_on_board(game_board, stone, position)
                        print_board(game_board)
                    else:
                        stone = 'o'
                        print("It is now Player 2's turn to place a stone.")
                        position = input("Enter the position to place the stone: ")
                        place_on_board(game_board, stone, position)
                        print_board(game_board)
                    print(check_for_winner(game_board))
                    winner = print(check_for_winner(game_board))
                    if winner == "●" or winner == "o" or winner == "Draw":
                        game_menu()
                        create_board(game_board)
                    else:
                        continue
            else:
                while True:
                    list_of_available_moves = len(check_available_moves(game_board))
                    if list_of_available_moves % 2 != 0:
                        stone = '●'
                        print("Player 1 has to place the stone.")
                        position = input("Enter the position to place the stone: ")
                        place_on_board(game_board, stone, position)
                        print_board(game_board)
                    else:
                        stone = 'o'
                        print("Computer has to place the stone: ")
                        player_move = position
                        random_move = random_computer_player(game_board, player_move
                        row_idx = random_move[0]
                        col_idx = random_move[1]
                        position_of_stone = row_idx+col_idx
                        place_on_board(game_board,stone,position_of_stone)
                        print_board(game_board)
                    winner = print(check_for_winner(game_board))
                    if winner == "●" or winner == "o" or winner == "Draw":
                        create_board(game_board)
                        game_menu()
                    else:
                        continue

        elif choice == "4":
            print("Resetting the current game.")
            create_board(board_size)
            print("New board has been created. Enjoy your game!")
        elif choice == "5":
            print("Exiting the game. Hope to see you soon. Have a good day!")
            return False
        else:
            print("Invalid option. Please select a valid choice between (1-5).")
```

In [ ]:

```
# Test code for 3.9 here [The code in this cell should be commented]

#play_game()
```

In [ ]:

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