FIT9136 Algorithms and Programming Foundations in Python

2023 Semester 2

Assignment 1

Student name:Smriti Singh
Student ID:34274227
Creation date:17/08/2023
Last modified date:17/08/2023

In [1]:
```
# Libraries to import (if any)
import random  #imported random library as per specification in assignment
```

3.1 Game menu function

In [2]:
```
# Test code for 3.1 here [The code in this cell should be commented]
def game_menu():
    print("Welcome to Gomoku!")
    print("\nGame 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 [3]:
```
# # Test code for 3.1 here [The code in this cell should be commented]
#game_menu()
```

3.2 Creating the Board

In [4]:
```
 # Implement code for 3.2 here
def create_board(size):
    game_board ={} # empty dictionary where key are position and value is state (empty/filled)
    for row in range(size):
        for col in range(size):
            place = (row,chr(col+65)) #key for dictionary game_board
            game_board[place] = ' '   #for initial stage value is same for all the keys as all place are unoccupied.
    return game_board
```

In [5]:
```
# # Test code for 3.2 here [The code in this cell should be commented]
# create_board(9).keys()
```

3.3 Is the target position occupied?

In [6]:
```
 # Implement code for 3.3 here
def is_occupied(board, x, y):
    position = (x,y) # x and y are position entered by user
    if board[(x, y)]!= ' ': # board check whether value at x,y is  not equal to empty space
        return False    #then it will return false means position is already occcupied
    else:
        return True  #otherwise it will return true means position is unoccupied
```

In [7]:
```
# Test code for 3.3 here [The code in this cell should be commented]
#board = {(0, 0): 'X', (0, 1): 'O', (1, 0): ' ', (1, 1): ' '}
#print(is_occupied(board, 0, 0))  # if it return False (occupied by 'X')
#print(is_occupied(board, 1, 0))  # if it return True (unoccupied)
```

3.4 Placing a Stone at a Specific Intersection

In [8]:
```
 # Implement code for 3.4 here
def place_on_board(board, stone, position):
    x = int(position[0])
    y = position[1]
```

```
    if is_occupied(board, x, y):
        board[(x, y)] = stone
        return True
    else:
        return False
```

In [9]:
```
# Test code for 3.4 here [The code in this cell should be commented]

#board = create_board(5)
#board[(0,'A')] = '●'
#place_on_board(board,'●',(0,'A'))
```

3.5 Printing the Board

In [10]:
```
#  # Implement code for 3.5 here
def print_board(board):
    size=int(len(board)**0.5)
    alpha_string =''
    for i in range(size):
        alpha_string += chr(65+i)+'  '
    print(alpha_string)
    for row in range(size):
        string =''
        for col in range(size):
            if col<size-1:
                string= string+ board[(row,chr(col+65))]+'--'
            else:
                string= string+ board[(row,chr(col+65))]
        print(string+'  '+str(row))
        if row < size-1:
            print('|  '*size)
```

In [11]:
```
# Test code for 3.5 here [The code in this cell should be commented]
# Example board setup
# b = create_board(9)
# b[(3,'C')] = '●'
# print_board(b)
```

3.6 Check Available Moves

In [12]:
```
 # Implement code for 3.6 here
def check_available_moves(board):
    keys = board.keys()
    available_moves =[]
    for key in keys:
        if board[key]==' ':
            available_moves.append(key)
    return available_moves
```

In [13]:
```
# Test code for 3.6 here [The code in this cell should be commented]
# b = create_board(3)
# b[(2,'C')] = '●'
# check_available_moves(b)
```

3.7 Check for the Winner

In [14]:
```
 # Implement code for 3.7 here

def check_for_winner(board):
    if winner_check(board,'●'):
        return "Player1"
    elif winner_check(board,'○'):
        return "Player2"
    elif len(check_available_moves(board))==0:
        return "Draw"
    else:
        return "None"

def winner_check(board,stone):
    size=int(len(board)**0.5)
    # Check for a horizontal win.
    for row in range(size):
        if all(board[(row, chr(i+65))] == stone for i in range(1, size)):
            return True

    # Check for a vertical win.
    for col in range(size):
        if all(board[(i, chr(col+65))] == stone for i in range(1, size)):
            return True

    # Check for a diagonal win.
```

```
    for i in range(size - 4):
        for j in range(size-4):
            if all(board[(i + k, chr(j+65))] == stone for k in range(5)):
                return True
            if all(board[(i + k, chr(size - 1 - j+65))] == stone for k in range(5)):
                return True
    return False
```

In [15]:
```
# Test code for 3.7 here [The code in this cell should be commented]
# x= create_board(5)
# check_for_winner(x)
```

3.8 Random Computer Player

In [16]:
```
 # Implement code for 3.8 here
def random_computer_player(board, player_move):
    valid_positions = []
    size=int(len(board)**0.5)
    moves=check_available_moves(board)
    for i in range(player_move[0] - 1, player_move[0] + 2):
        for j in range(ord(player_move[1])-65- 1, ord(player_move[1])-65+ 2):
            move=(i, chr(j+65))
            if move in moves:
                valid_positions.append(move)

    # If all positions within the 3 * 3 square are invalid, randomly select from one of the available positions on the board.
    if not valid_positions:
        valid_positions = movess

    # Randomly select one of the available positions.
    return random.choice(valid_positions)
```

In [17]:
```
# Test code for 3.8 here [The code in this cell should be commented]
# Example usage
#board = create_board(9)  # Simulating an empty board
#player_move = (4, 'B')  # Example player's move
#board[player_move]='●'
#computer_move = random_computer_player(board, player_move)
#print("Computer's move:", computer_move)
```

3.9 Play Game

In [ ]:

```
# Test code for 3.9 here [The code in this cell should be commented]
# Run the game
def play_game():
    board = None
    mode = None
    turn=1

    while True:
        game_menu()
        choice = input("Select an option: ")

        if choice == "1":
            if board:
                print("Game in progress. Do you want to restart or continue the current game?")
                print("1. Restart")
                print("2. Continue")
                restart_choice = input("Select an option: ")
                if restart_choice == "1":
                    board = create_board(8)
                    print("Game restarted.")
                elif restart_choice == "2":
                    continue
                else:
                    print("Invalid choice. Please select a valid option.")
                    continue

            size = int(input("Enter board size (greater than 5) in digits: "))
            if size not in range(5,25):
                print("Invalid board size. Please choose a valid size.")
                continue

            mode = input("Enter mode (PvP or PvC): ").lower()
            if mode not in ["pvp", "pvc"]:
                print("Invalid mode. Please choose a valid mode.")
                continue

            board = create_board(size)
            print(f"Game started with board size {size} and mode {mode.upper()}.")

        elif choice == "2":
```

```
            print_board(board)

        elif choice == "3":
            size=int(len(board)**0.5)
            available_moves=check_available_moves(board)

            if mode is None:
                print("Please start a game first.")
                continue

            if mode == "pvp":
                if turn%2!=0:
                    stone="●"
                else:
                    stone="○"
                print_board(board)
                position = input(f"Player {stone}: Enter position (e.g. 2 F): ").split()
                row_index = position[0]
                column_index = position[1].upper()
                if len(position)==2 and row_index.isnumeric() and column_index.isalpha():
                    if ((int(row_index),column_index) not in available_moves):
                        print("Position already occupied or invalid position. choose a different one.")
                        continue
                    place_on_board(board, stone, (int(row_index), column_index))
                    turn+=1
                    winner = check_for_winner(board)
                else:
                    print("Position entered is wrong,please enter again.")

            elif mode == "pvc":
                stone="●"
                computer_stone="○"
                player_position = input(f"Player {stone}: Enter position (e.g. 2 F): ").split()
                player_row_index = player_position[0]
                player_column_index = player_position[1].upper()
                if len(player_position)==2 and player_row_index.isnumeric() and player_column_index.isalpha():
                    if ((int(player_row_index),player_column_index) not in available_moves):
                        print("Position already occupied or invalid position. choose a different one.")
                        continue

                    place_on_board(board,stone , (int(player_row_index), player_column_index))
                    computer_move=random_computer_player(board, (int(player_row_index), player_column_index))
                    print("Computer position of stone",computer_move)
                    place_on_board(board,computer_stone,computer_move)
```

```
                    turn=1
                    winner = check_for_winner(board)
                else:
                    print("Position entered is wrong,please enter again.")

            if winner=="Player1" or winner=="Player2":
                print_board(board)
                print(f"Player {winner} wins!")
                board = create_board(len(board))
            elif winner=="Draw":
                print_board(board)

        elif choice == "4":
            board = create_board(len(board))
            mode = None
            print("Game reset.")

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

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

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