FIT9136 Algorithms and Programming Foundations in Python

2023 Semester 2

Assignment 1

Student name: Bao Ngoc Truong
Student ID: 34301054
Creation date: 07/08/2023
Last modified date: 24/08/2023

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

3.1 Game menu function

In [2]:
```
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 [3]:
```
# Test code for 3.1 here [The code in this cell should be commented]

# game_menu()
# the game menu should be printed like this:
# ===== 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 [4]:
```
# Define function create_board
def create_board(size):
```

```
    board = [[' ' for indx in range(size)] for indx in range(size)]
    return board
```

In [5]:
```
# Test code for 3.2 here [The code in this cell should be commented]
# b = create_board(9)
# print(len(b))
# len(b[0])
# # this should give output as 9 for both
```

3.3 Is the target position occupied?

In [6]:
```
 # Implement code for 3.3 here
def is_occupied(board, x, y):
    return board[x][y] != ' '
```

In [7]:
```
# Test code for 3.3 here [The code in this cell should be commented]
# game_board = [
#     ['X', 'O', 'X', 'O', 'O'],
#     ['O', ' ', ' ', ' ', 'X'],
#     ['O', ' ', ' ', ' ', ' '],
#     ['O', ' ', ' ', ' ', 'X'],
#     ['X', 'O', 'X', 'O', 'X']
# ]
# x = 1
# y = 3
# is_occupied(game_board, x, y)
# # Output should be: False
```

3.4 Placing a Stone at a Specific Intersection

In [8]:
```
# Implement code for 3.4 here
def is_valid_position(board, x, y):
    return 0 <= x < len(board) and 0 <= y < len(board[0])

def place_on_board(board, stone, position):
    x, y = position
    x = int(x)

    # Convert the column index to a numeric value
    y = ord(y.upper()) - ord('A')

    # Check if the position is valid and unoccupied then place the stone on the board
    if is_valid_position(board, x, y) and not 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]
# Enter input for rows and column index to identify the location
# game_board = [
#     ['X', 'O', 'X', 'O', 'O'],
#     ['O', ' ', ' ', ' ', 'X'],
#     ['O', ' ', ' ', ' ', ' '],
#     ['O', ' ', ' ', ' ', 'X'],
#     ['X', 'O', 'X', 'O', 'X']
```

```
# ]
# a = 1
# b = 'B'
# position_to_place = a, b
# stone_to_place = "●"

# if place_on_board(game_board, stone_to_place, position_to_place):
#     print("Stone placed successfully!")
# else:
#     print("Invalid or occupied position. Cannot place the stone.")
# Output should be: Stone placed successfully!
```

3.5 Printing the Board

In [10]:
```
# Assign create board function to board
def print_board(board):
    size = len(board)
    col_indices = [chr(ord('A') + i) for i in range(size)]

    print('    ' + '    '.join(col_indices))
    for i, row in enumerate(board):
        formatted_row = [stone if stone != ' ' else ' ' for stone in row]
        print('   '+ ' |   '* (len(row)))
        print('    ' + ' -- ' .join(formatted_row) + f'{i:2}')
```

In [11]:
```
# Test code for 3.5 here [The code in this cell should be commented]
# board = create_board(9)
# print_board(board)
# board should have column A to I and row 0 to 8
```

3.6 Check Available Moves

In [12]:
```
 # Implement code for 3.6 here
def check_available_moves(board):
    available_moves = []
    for x in range(len(board)):
        for y in range(len(board[0])):
            if not is_occupied(board, x, y):
                available_moves.append((str(x), chr(y + ord('A'))))
    return available_moves
```

In [13]:
```
# Test code for 3.6 here [The code in this cell should be commented]
#check_available_moves(board)
# b = create_board(9)
# len(check_available_moves(b))
# # this should output 81
```

3.7 Check for the Winner

In [14]:
```
 # Implement code for 3.7 here
def check_for_winner(board):
    directions = [(1, 0), (0, 1), (1, 1), (1, -1)]  # Horizontal, Vertical, Diagonal1,

    for x in range(len(board)):
        for y in range(len(board[0])):
```

```
            if board[x][y] != ' ':
                for dx, dy in directions:
                    if is_valid_position(board, x + 4 * dx, y + 4 * dy):
                        stones = [board[x + i * dx][y + i * dy] for i in range(5)]
                        if all(stone == stones[0] for stone in stones):
                            return stones[0]

    available_moves = check_available_moves(board)
    if not available_moves:
        return "Draw"

    return None
```

In [15]:
```
# Test code for 3.7 here [The code in this cell should be commented]
# game_board = [
#     ['X', 'O', 'X', 'O', 'O'],
#     ['O', 'X', 'O', 'X', 'X'],
#     ['O', 'X', 'X', 'X', ' '],
#     ['O', 'X', 'O', 'X', 'X'],
#     ['X', 'O', 'X', 'O', 'X']
# ]

# winner = check_for_winner(game_board)
# if winner:
#     print("Winner:", winner)
# else:
#     print("No winner yet.")
# Output should be: Winner: X
```

3.8 Random Computer Player

In [16]:
```
 # Implement code for 3.8 here
def random_computer_player(board, player_move):
    directions = [(-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 1), (1, -1), (1, 0), (1, 1)
    available_moves = []

    for dx, dy in directions:
        x = int(player_move[0]) + dx
        y = ord(player_move[1].upper()) + dy - ord('A')
        if is_valid_position(board, x, y) and not is_occupied(board, x, y):
            available_moves.append((str(x), chr(y + ord('A'))))

    if available_moves:
        return random.choice(available_moves)

    all_available_moves = check_available_moves(board)
    return random.choice(all_available_moves)
```

In [17]:
```
# # Test code for 3.8 here [The code in this cell should be commented]
# game_board = [
#     [' ', 'X', 'O', ' ', ' '],
#     ['X', 'O', ' ', ' ', ' '],
#     ['O', 'X', ' ', ' ', ' '],
#     [' ', ' ', ' ', ' ', ' '],
#     [' ', ' ', ' ', ' ', ' ']
# ]

# player_move = ("1", "C")
```

```
# computer_move = random_computer_player(game_board, player_move)
# print("Computer's move:", computer_move)
# # Output should be Computer's move ranging from ('2', 'C'), ('2', 'D'),('1', 'D'),(
```

3.9 Play Game

In [18]:

```
 # Implement code for 3.9 here
def play_game():
    game_board = None
    game_mode = None
    current_turn = "●"
    game_in_progress = False

    while True:
        game_menu()
        choice = input("Enter your choice (1-5): ")
        if choice == "1":
            #player choose to restart or continue the current game
            if game_in_progress:
                pint("A game is already in progress. Do you want to restart or complet
                restart_choice = input("Enter 'restart' or 'complete': ")
                if restart_choice == "restart":
                    game_board = None
                else:
                    print("Complete the current game.")
                    continue
            #player choose size of the board
            size = int(input("Enter the board size (9/13/15): "))
            if size not in [9, 13, 15]:
                print("Invalid board size. Please choose from 9, 13, or 15.")
                continue
            # Player choose mode to play versus person or computer
            mode = input("Enter the mode (PvP or PvC): ")
            if mode.lower() not in ["pvp", "pvc"]:
                print("Invalid mode. Please choose either PvP or PvC.")
                continue

            game_board = create_board(size)
            game_mode = mode
            current_turn = "●"
            game_in_progress = True
            print("New game started!")

        elif choice == "2":
            if game_board is not None:
                print_board(game_board)
            else:
                print("No game has been started yet.")
        # choose location to place the stone
        elif choice == "3":
            if game_board is None:
                print("No game in progress.")
                continue
            # 2 players mode
            if mode.lower() == 'pvp':
                player_move = input(f"Player {current_turn} - Enter move (e.g., '2 C')
                row, column = player_move.split()
                player_move = (row, column)
```

```
            if place_on_board(game_board, current_turn, player_move):
                print_board(game_board)
                winner = check_for_winner(game_board)
                if winner:
                    print("Player", winner, "wins!")
                    game_board = None
                else:
                    current_turn = '●' if current_turn == 'o' else 'o'
            else:
                print("Invalid move. Try again.")
        # player vs computer mode
        elif mode.lower() == 'pvc':
            player_move = input(f"Player {current_turn} - Enter move (e.g., '2 C')
            row, column = player_move.split()
            player_move = (row, column)

            if place_on_board(game_board, current_turn, player_move):
                print_board(game_board)
                winner = check_for_winner(game_board)
                if winner:
                    print("Player", winner, "wins!")
                    game_board = None
                else:
                    current_turn = '●' if current_turn == 'o' else 'o'

                if current_turn == 'o':
                    computer_move = random_computer_player(game_board, player_move
                    if computer_move:
                        print(f"Computer moves: {computer_move[0]} {computer_move[
                        place_on_board(game_board, current_turn, computer_move)
                        print_board(game_board)
                        winner = check_for_winner(game_board)
                        if winner:
                            print("Player", winner, "wins!")
                            game_board = None
                        else:
                            current_turn = '●'

        else:
            print("Invalid game mode.")
            continue
    # Reset the game
    elif choice == "4":
        game_board = None
        game_mode = None
        player_turn = "●"
        game_in_progress = False
        print("Game reset.")
    # Exit the game
    elif choice == "5":
        print("Exiting the game.")
        break

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

In [19]:
```
# Test code for 3.9 here [The code in this cell should be commented]
# play_game()
```

```
# output should be:
#     1. game menu and input your choice
#     2. option 1: start a new game, Ask for a board size from 9, 13 or 15
#     3. ask for a mode from the user, either Player vs. Player (PvP) or Player vs. Co
#     4. if choose 1 and a game is in progress, ask if you want to continue the currer
#     5. if choose Option 2, visualise the current state of the board
#     6. option "3", Ask the user to place a stone at a position with input format "[r
#     7. For PvP mode, only one stone from the corresponding player will be placed on
#     8. For PvC mode, after checking the player's move, the computer player should pl
#     9. If one of the players win, or no more move is available, the game needs to au
#     10. if choose "4", the function should reset the game
#     11. option "5", the function should exit the program.
```

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

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

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 ---