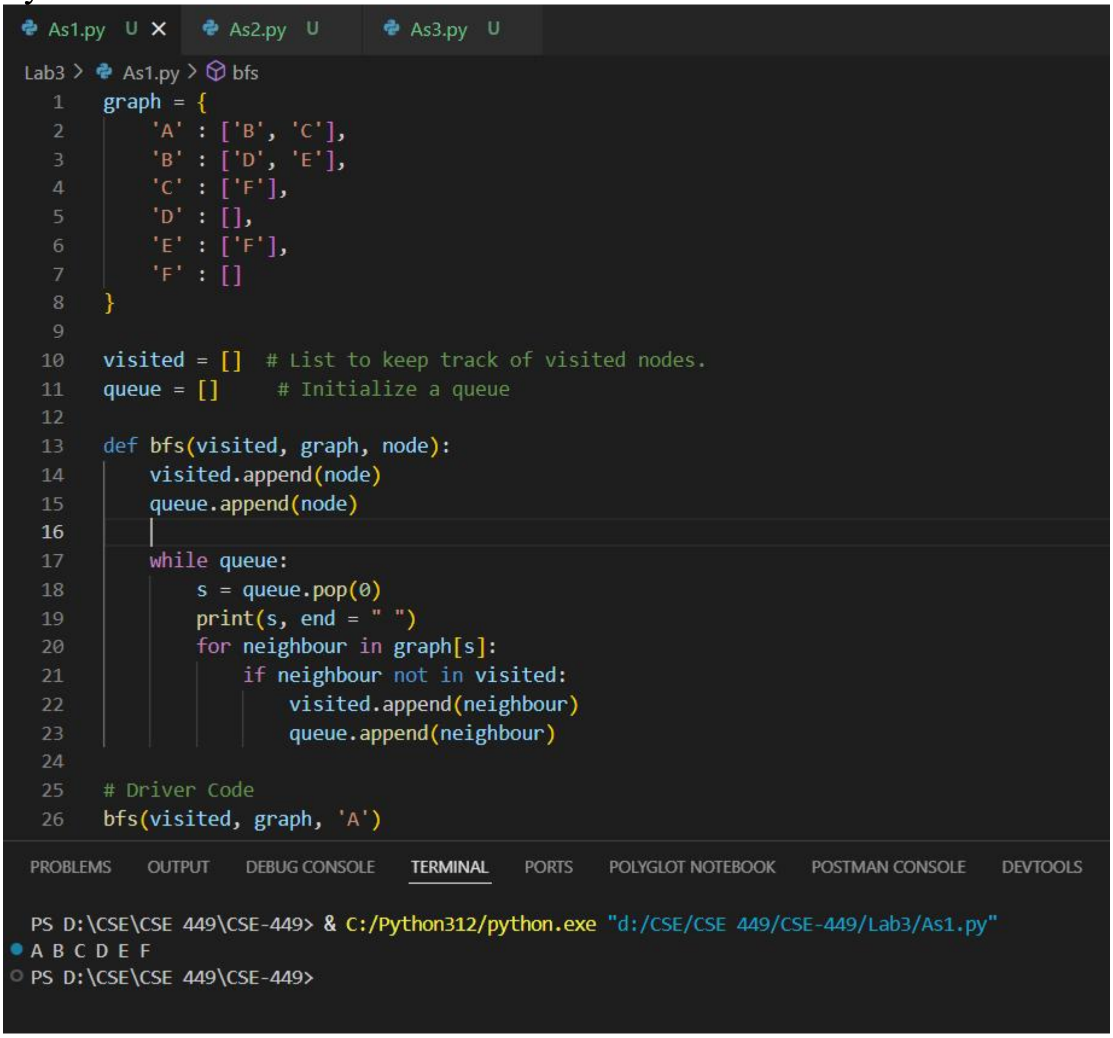
CSE 449 Lab 3

Name: Nguyễn Hữu Hào
Student Id: 2031200078

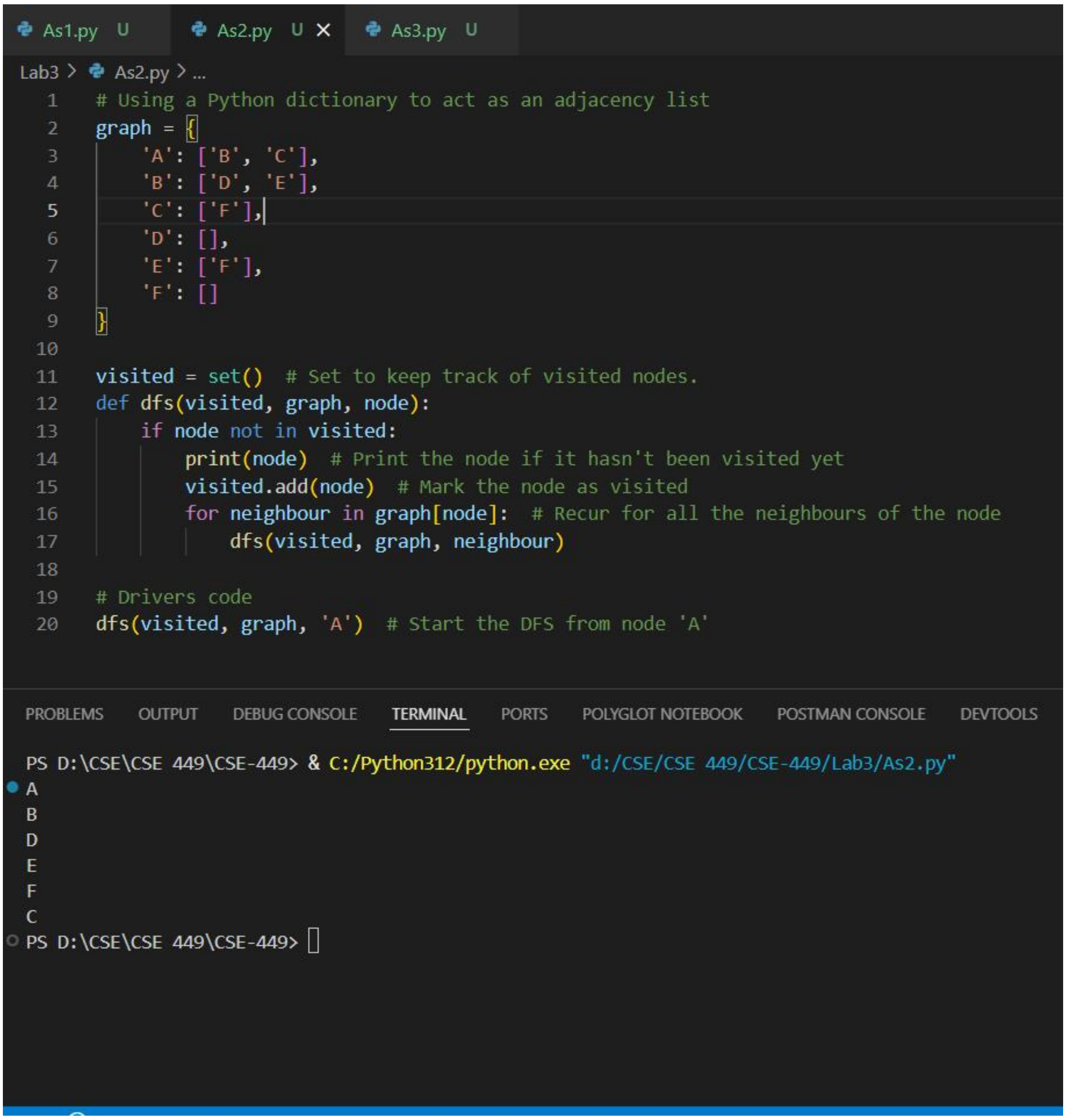
Assignment 1: Write a Program to Implement Breath First Search using Python



```python
graph = {
    'A' : ['B', 'C'],
    'B' : ['D', 'E'],
    'C' : ['F'],
    'D' : [],
    'E' : ['F'],
    'F' : []
}

visited = []  # List to keep track of visited nodes.
queue = []     # Initialize a queue

def bfs(visited, graph, node):
    visited.append(node)
    queue.append(node)

    while queue:
        s = queue.pop(0)
        print(s, end = " ")
        for neighbour in graph[s]:
            if neighbour not in visited:
                visited.append(neighbour)
                queue.append(neighbour)

# Driver Code
bfs(visited, graph, 'A')
```

```
PS D:\CSE\CSE 449\CSE-449> & C:/Python312/python.exe "d:/CSE/CSE 449/CSE-449/Lab3/As1.py"
A B C D E F
PS D:\CSE\CSE 449\CSE-449>
```

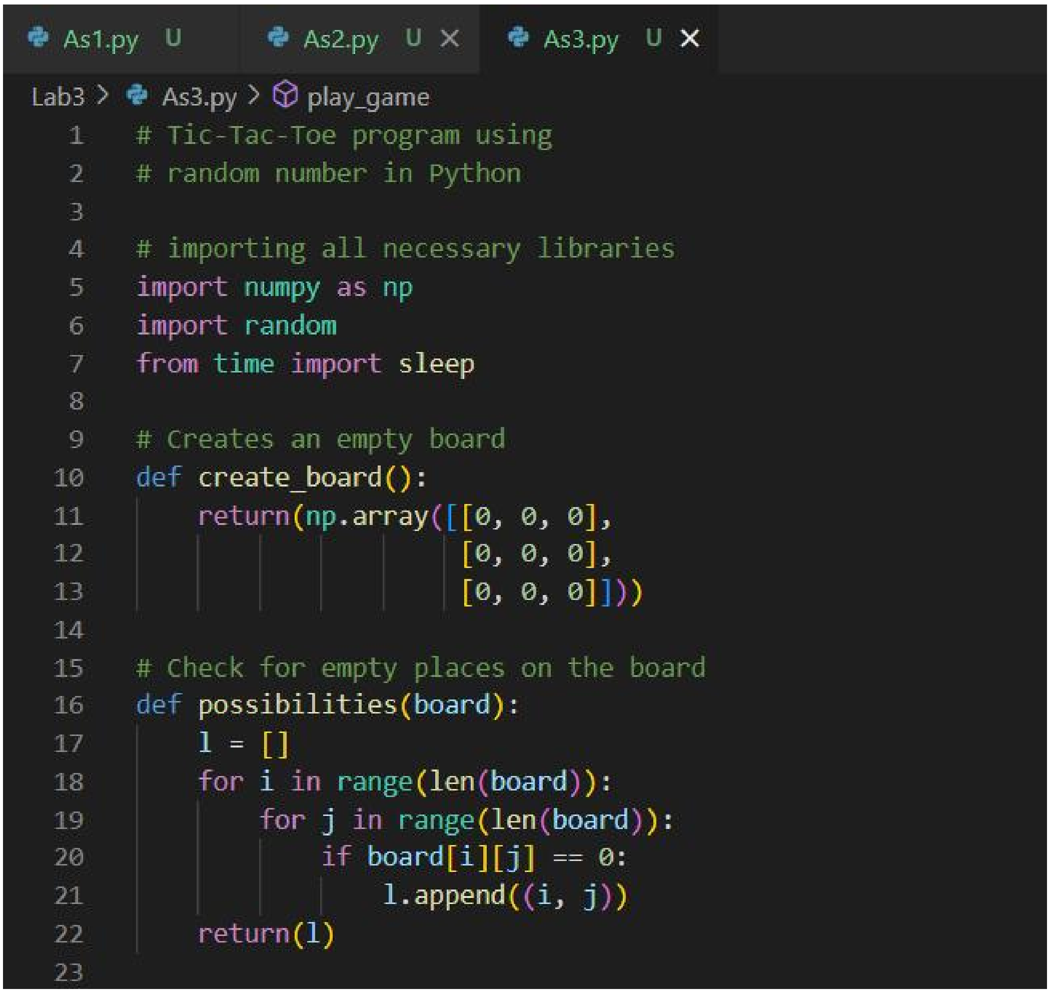
Assignment 2: Write a Program to Implement Depth First Search using Python



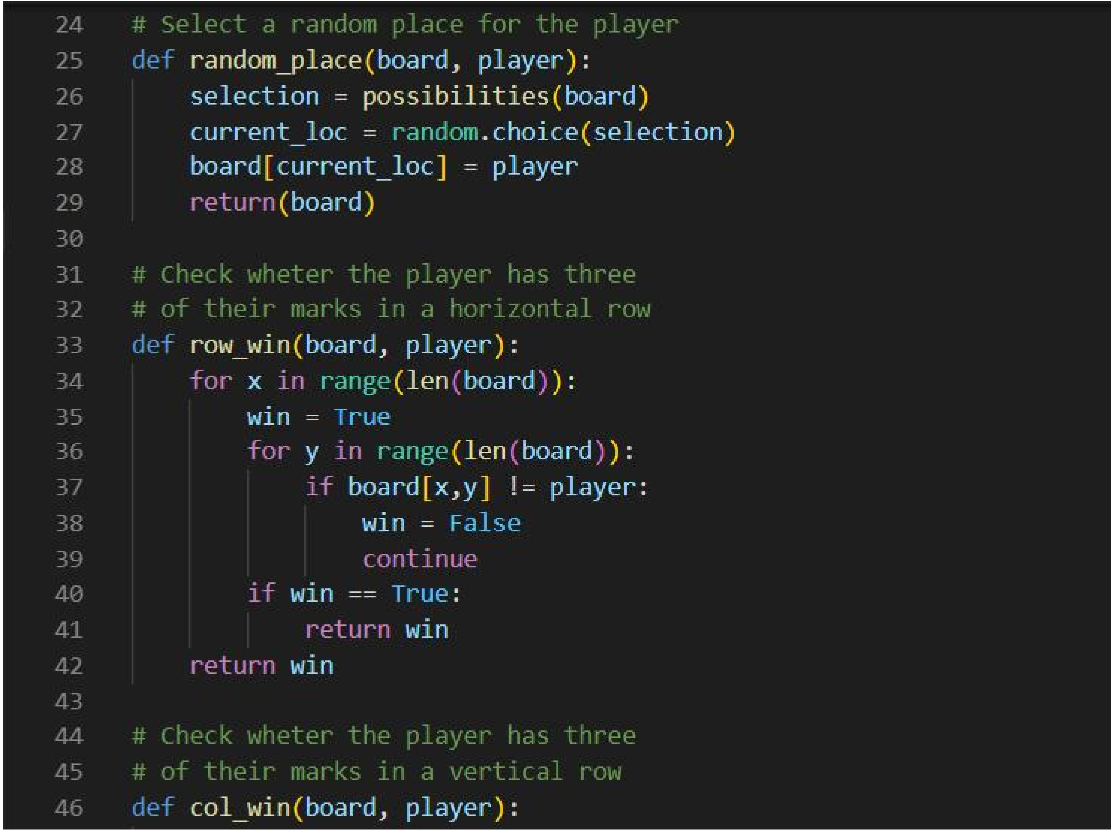
```python
# Using a Python dictionary to act as an adjacency list
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': []
}

visited = set()  # Set to keep track of visited nodes.
def dfs(visited, graph, node):
    if node not in visited:
        print(node)  # Print the node if it hasn't been visited yet
        visited.add(node)  # Mark the node as visited
        for neighbour in graph[node]:  # Recur for all the neighbours of the node
            dfs(visited, graph, neighbour)

# Drivers code
dfs(visited, graph, 'A')  # Start the DFS from node 'A'
```

```
PS D:\CSE\CSE 449\CSE-449> & C:/Python312/python.exe "d:/CSE/CSE 449/CSE-449/Lab3/As2.py"
A
B
D
E
F
C
PS D:\CSE\CSE 449\CSE-449>
```

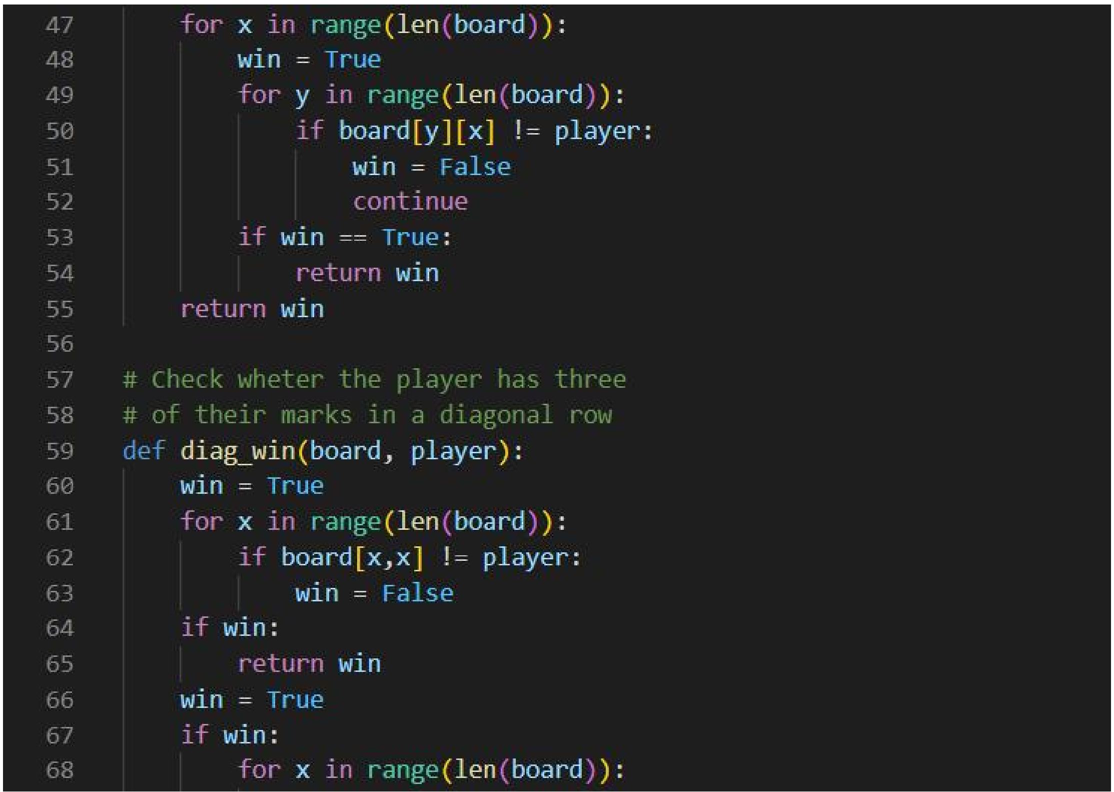
Assignment 3: Write a Program to Implement Tic-Tac-Toe game using Python



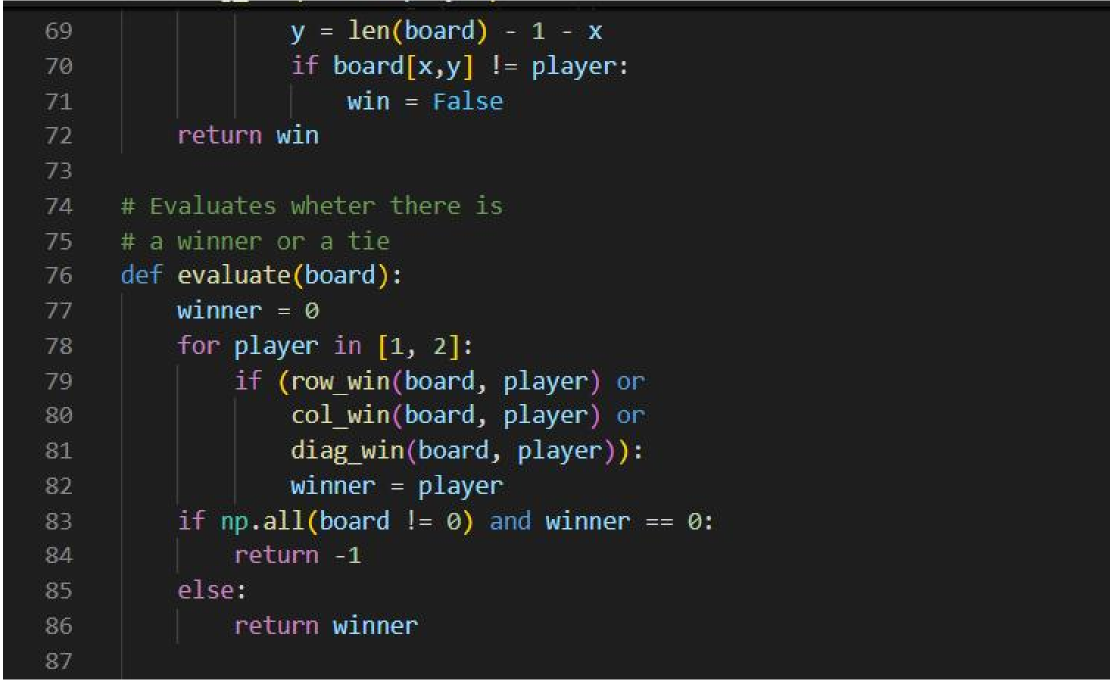
```python
# Tic-Tac-Toe program using
# random number in Python

# importing all necessary libraries
import numpy as np
import random
from time import sleep

# Creates an empty board
def create_board():
    return(np.array([[0, 0, 0],
                     [0, 0, 0],
                     [0, 0, 0]]))

# Check for empty places on the board
def possibilities(board):
    l = []
    for i in range(len(board)):
        for j in range(len(board)):
            if board[i][j] == 0:
                l.append((i, j))
    return(l)

```



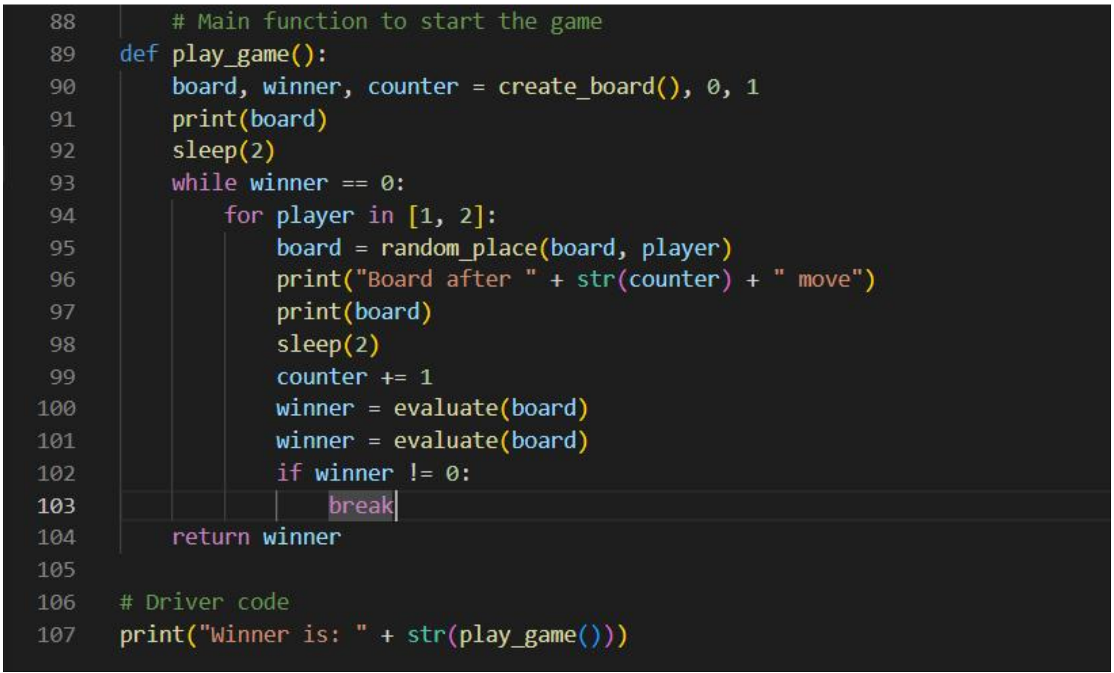
```python
# Select a random place for the player
def random_place(board, player):
    selection = possibilities(board)
    current_loc = random.choice(selection)
    board[current_loc] = player
    return(board)

# Check wheter the player has three
# of their marks in a horizontal row
def row_win(board, player):
    for x in range(len(board)):
        win = True
        for y in range(len(board)):
            if board[x,y] != player:
                win = False
                continue
        if win == True:
            return win
    return win

# Check wheter the player has three
# of their marks in a vertical row
def col_win(board, player):
```



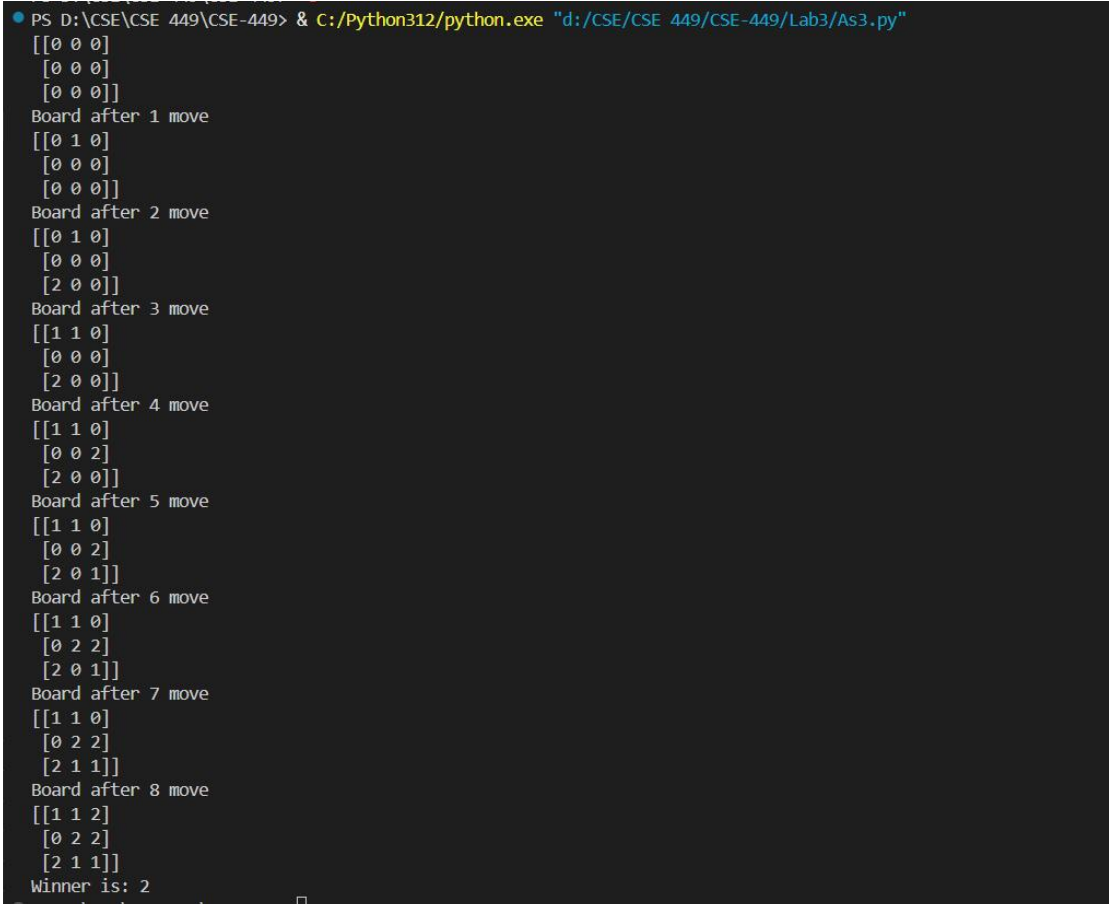
```python
    for x in range(len(board)):
        win = True
        for y in range(len(board)):
            if board[y][x] != player:
                win = False
                continue
        if win == True:
            return win
    return win

# Check wheter the player has three
# of their marks in a diagonal row
def diag_win(board, player):
    win = True
    for x in range(len(board)):
        if board[x,x] != player:
            win = False
    if win:
        return win
    win = True
    if win:
        for x in range(len(board)):
```



```python
            y = len(board) - 1 - x
            if board[x,y] != player:
                win = False
    return win

# Evaluates wheter there is
# a winner or a tie
def evaluate(board):
    winner = 0
    for player in [1, 2]:
        if (row_win(board, player) or
            col_win(board, player) or
            diag_win(board, player)):
            winner = player
    if np.all(board != 0) and winner == 0:
        return -1
    else:
        return winner

```



```python
    # Main function to start the game
def play_game():
    board, winner, counter = create_board(), 0, 1
    print(board)
    sleep(2)
    while winner == 0:
        for player in [1, 2]:
            board = random_place(board, player)
            print("Board after " + str(counter) + " move")
            print(board)
            sleep(2)
            counter += 1
            winner = evaluate(board)
            winner = evaluate(board)
            if winner != 0:
                break
    return winner

# Driver code
print("Winner is: " + str(play_game()))
```



```
PS D:\CSE\CSE 449\CSE-449> & C:/Python312/python.exe "d:/CSE/CSE 449/CSE-449/Lab3/As3.py"
[[0 0 0]
 [0 0 0]
 [0 0 0]]
Board after 1 move
[[0 1 0]
 [0 0 0]
 [0 0 0]]
Board after 2 move
[[0 1 0]
 [0 0 0]
 [2 0 0]]
Board after 3 move
[[1 1 0]
 [0 0 0]
 [2 0 0]]
Board after 4 move
[[1 1 0]
 [0 0 2]
 [2 0 0]]
Board after 5 move
[[1 1 0]
 [0 0 2]
 [2 0 1]]
Board after 6 move
[[1 1 0]
 [0 2 2]
 [2 0 1]]
Board after 7 move
[[1 1 0]
 [0 2 2]
 [2 1 1]]
Board after 8 move
[[1 1 2]
 [0 2 2]
 [2 1 1]]
Winner is: 2
```