DEPARTMENT OF
ELECTRONICS AND COMMUNICATION ENGINEERING
Faculty of Engineering and Technology
SRM Institute of Science and Technology

MINI PROJECT REPORT
ODD Semester, 2020-21

Lab code & subject name : 18ECE201J - Python and Scientific Python

Year & semester **:** 3rd Year and 5th Semester

Project title **:** Tic Tac Game

Lab Supervisor **:** Dr.K.Kalimuthu

Team Members **:**

1. Gagan Deep Rana (RA1811004010325)
2. Pranshu Mishra (RA1811004010327)

AIM: To make a Tic-Tac-Toe game using python

OBJECTIVES:

To make a user interactive program , a Tic-Tac-Toe game.Two players are allowed to play this game i.e. user and computer itself. Rules have been specified in the code .
If either all rows or columns or diagonal row are same player wins.

ABSTRACT:

In order to win the game, a player must place three of their marks in a horizontal, vertical, or diagonal row.
The following example game is won by the first player, X:

Game of Tic-tac-toe, won by X
Players soon discover that the best play from both parties leads to a draw. Hence, tic-tac-toe is most often played by young children, who often have not yet discovered the optimal strategy.

Incidence structure for tic-tac-toe.
Because of the simplicity of tic-tac-toe, it is often used as a pedagogical tool for teaching the concepts of good sportsmanship and the branch of artificial intelligence that deals with the searching of game trees. It is straightforward to write a computer program to play tic-tac-toe perfectly or to enumerate the 765 essentially different positions (the state space complexity) or the 26,830 possible games up to rotations and reflections (the game tree complexity) on this space. If played optimally by both players, the game always ends in a draw, making tic-tac-toe a futile game.

In this second code, we performed the basic operation of the calculator, in this application we took input from the user and performed the given task.

VIDEO LINK:

- https://drive.google.com/file/d/1htqfZwnGgsxqH6IuxFRN-A6y0h2Q6d61/view?usp=sharing

SOFTWARE REQUIRED:

- Jupyter Notebook in Anaconda Navigator.

PROCEDURE FOR TIC TAC GAME:

1) Open the jupyter notebook from the Anaconda Navigator

2) Define a list in order to initialize the board

3) Declare three global variables named game_still_going (in order to Lets us know if the game is over yet), winner (Tells us who is the winner), current_player (Tells us who the current player is)

4) Define a function **"def play_game():"** to play the game of tic tac toe.

5) Show the initial game board and run a while loop to start the game

6) Call in three functions which are defined in the latter half of the code**, handle_turn(current_player) >> Handle a turn , check_if_game_over() >> Check if the game is over, flip_player() >>Flip to the other player**

7) Using *a conditional approach*, print the winner or tie

8) Define a function **def display_board():** , which will display the given board

9) Define another function **'def handle_turn(player):"** to handle the turn to an arbitrary player.

   This function contains position from the player and makes sure that it's a valid input, and the spot is open

10) Define another function "**def check_if_game_over():**" to check if the game is over
   This function is also used to call two more function i.e check_for_winner() check_for_tie()

11) Define a function **" def check_for_winner():"** to see if somebody has won. This function is further used to call three other function
   row_winner = check_rows()
   column_winner = check_columns()
   diagonal_winner = check_diagonals()
   and later *uses if and else approach* to Get the winner

12) A function **"def check_rows():** "is defined to Check the rows for a winner, This function Checks if any of the rows have all the same value (and is not empty). If any row is identical than using the conditional approach it returns a winner

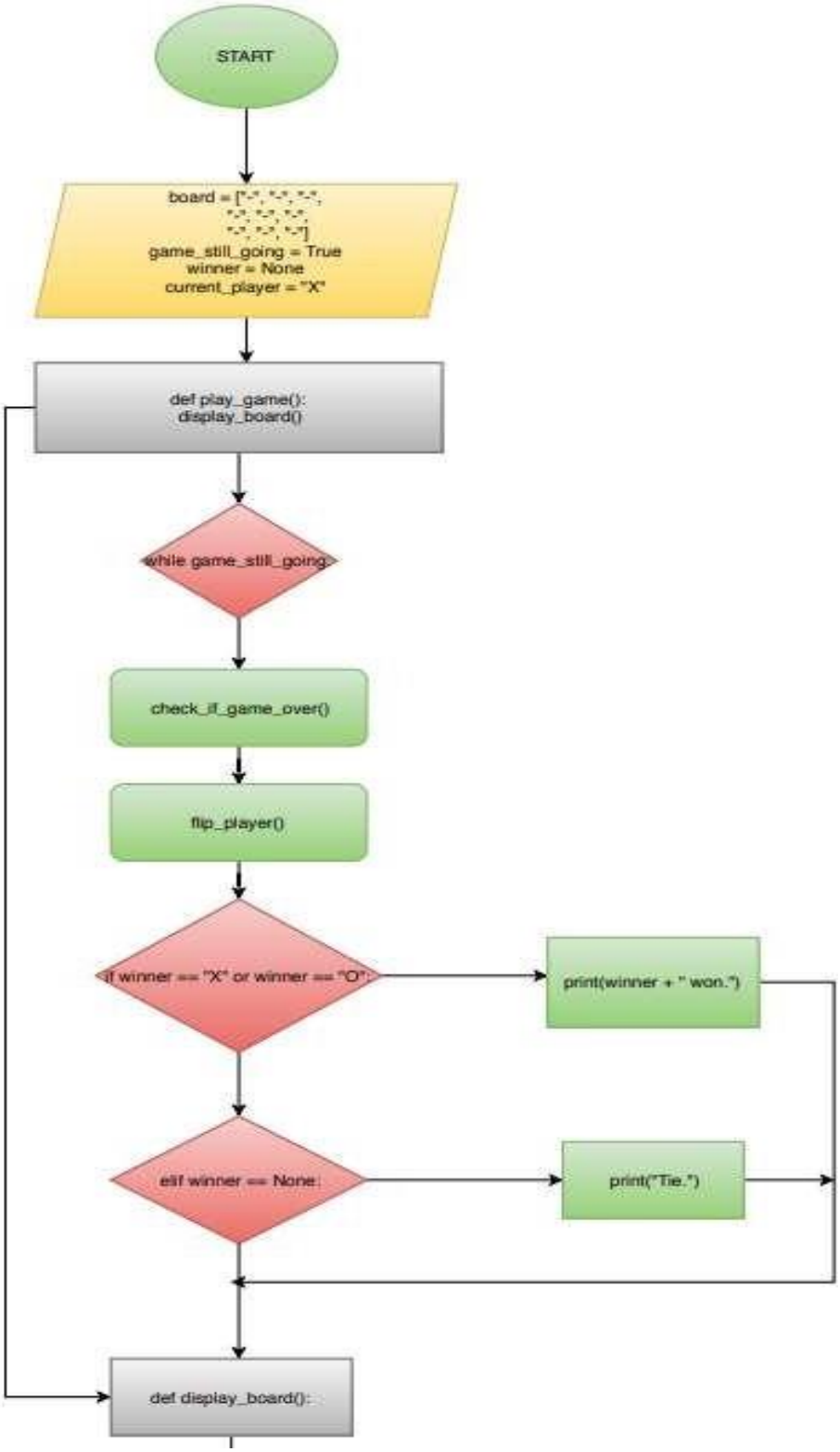
13) A function **"def check_columns():"** is defined to Check the rows for a winner, This function Checks if any of the columns have all the same value (and is not empty). If any columns is identical than using the conditional approach it returns a winner

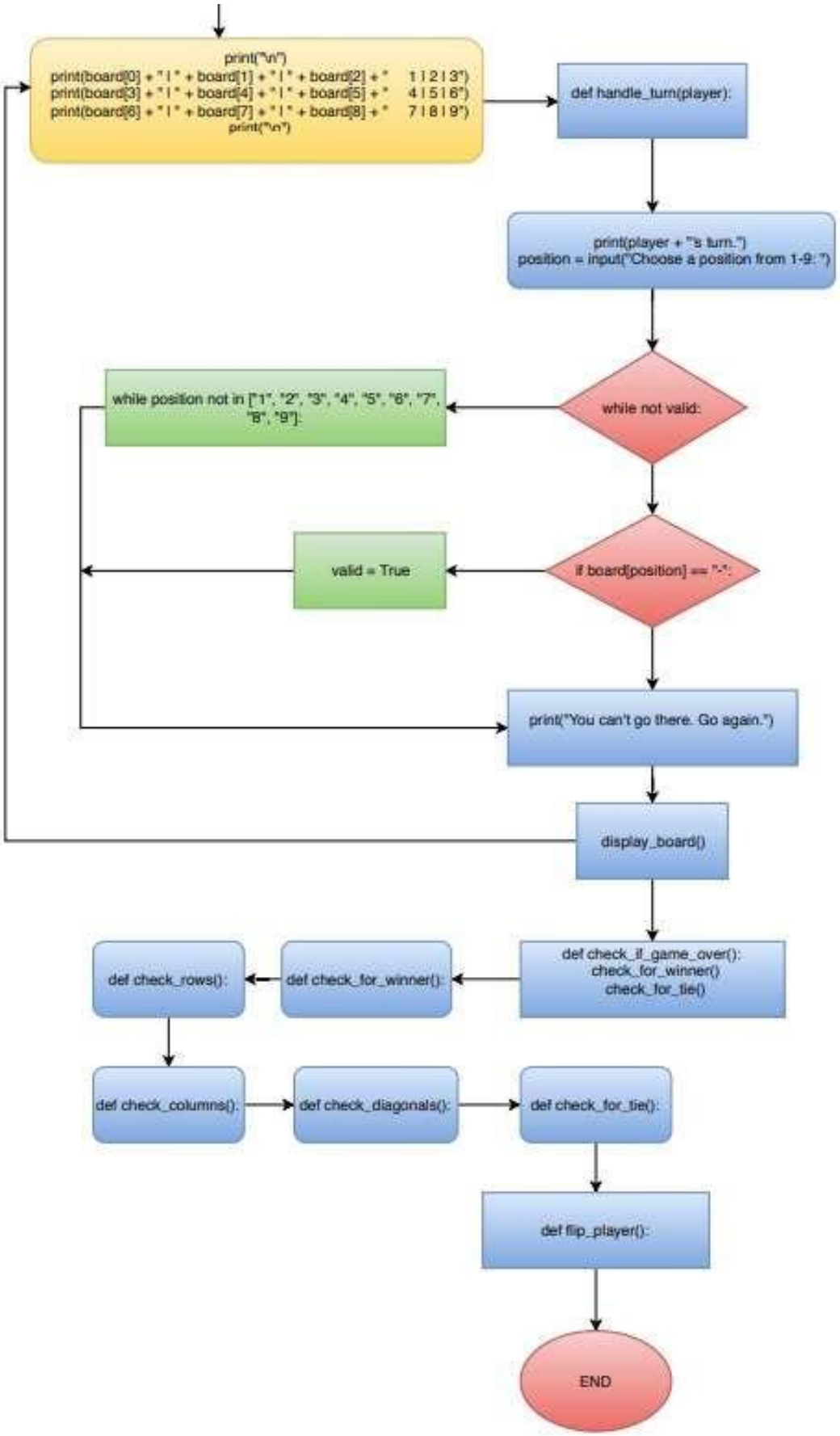
14) A function **"def check_diagonals():"** is defined to check the diagonals for a winner. This function Checks if any of the diagonals have all the same value (and is not empty). If any diagonals is identical than using the conditional approach it returns a winner

15) define a function **“def check_for_tie():”** to Check for a tie , i.e if the If board is full ,than there will be a tie

16) In order to Flip the current player from X to O, or O to X , we define a function **def flip_player():** to do this given operation

FLOWCHART FOR TIC TAC GAME:




print("\n")
print(board[0] + " | " + board[1] + " | " + board[2] + " 1 | 2 | 3")
print(board[3] + " | " + board[4] + " | " + board[5] + " 4 | 5 | 6")
print(board[6] + " | " + board[7] + " | " + board[8] + " 7 | 8 | 9")
print("\n")
def handle_turn(player):
print(player + "'s turn.")
position = input("Choose a position from 1-9: ")
while not valid:
while position not in ["1", "2", "3", "4", "5", "6", "7", "8", "9"]:
if board[position] == "-":
valid = True
print("You can't go there. Go again.")
display_board()
def check_if_game_over():
check_for_winner()
check_for_tie()
def check_for_winner():
def check_rows():
def check_columns():
def check_diagonals():
def check_for_tie():
def flip_player():
END

CODE FOR GAME:

```
# Will hold our game board data
board = ["-", "-", "-",
         "-", "-", "-",
         "-", "-", "-"]

# Lets us know if the game is over yet01
game_still_going = True

# Tells us who the winner is
winner = None

# Tells us who the current player is (X goes first)
current_player = "X"


# ------------- Functions ---------------

# Play a game of tic tac toe
def play_game():

  # Show the initial game board
  display_board()

  # Loop until the game stops (winner or tie)
  while game_still_going:

    # Handle a turn
    handle_turn(current_player)

    # Check if the game is over
    check_if_game_over()

    # Flip to the other player
    flip_player()

  # Since the game is over, print the winner or tie
```

```
  if winner == "X" or winner == "O":
    print(winner + " won.")
  elif winner == None:
    print("Tie.")


# Display the game board to the screen
def display_board():
  print("\n")
  print(board[0] + " | " + board[1] + " | " + board[2] + " 1 | 2 | 3")
  print(board[3] + " | " + board[4] + " | " + board[5] + " 4 | 5 | 6")
  print(board[6] + " | " + board[7] + " | " + board[8] + " 7 | 8 | 9")
  print("\n")


# Handle a turn for an arbitrary player
def handle_turn(player):

  # Get position from player
  print(player + "'s turn.")
  position = input("Choose a position from 1-9: ")

  # Whatever the user inputs, make sure it is a valid input, and the spot is open
  valid = False
  while not valid:

    # Make sure the input is valid
    while position not in ["1", "2", "3", "4", "5", "6", "7", "8", "9"]:
      position = input("Choose a position from 1-9: ")

    # Get correct index in our board list
    position = int(position) - 1

    # Then also make sure the spot is available on the board
    if board[position] == "-":
      valid = True
    else:
```

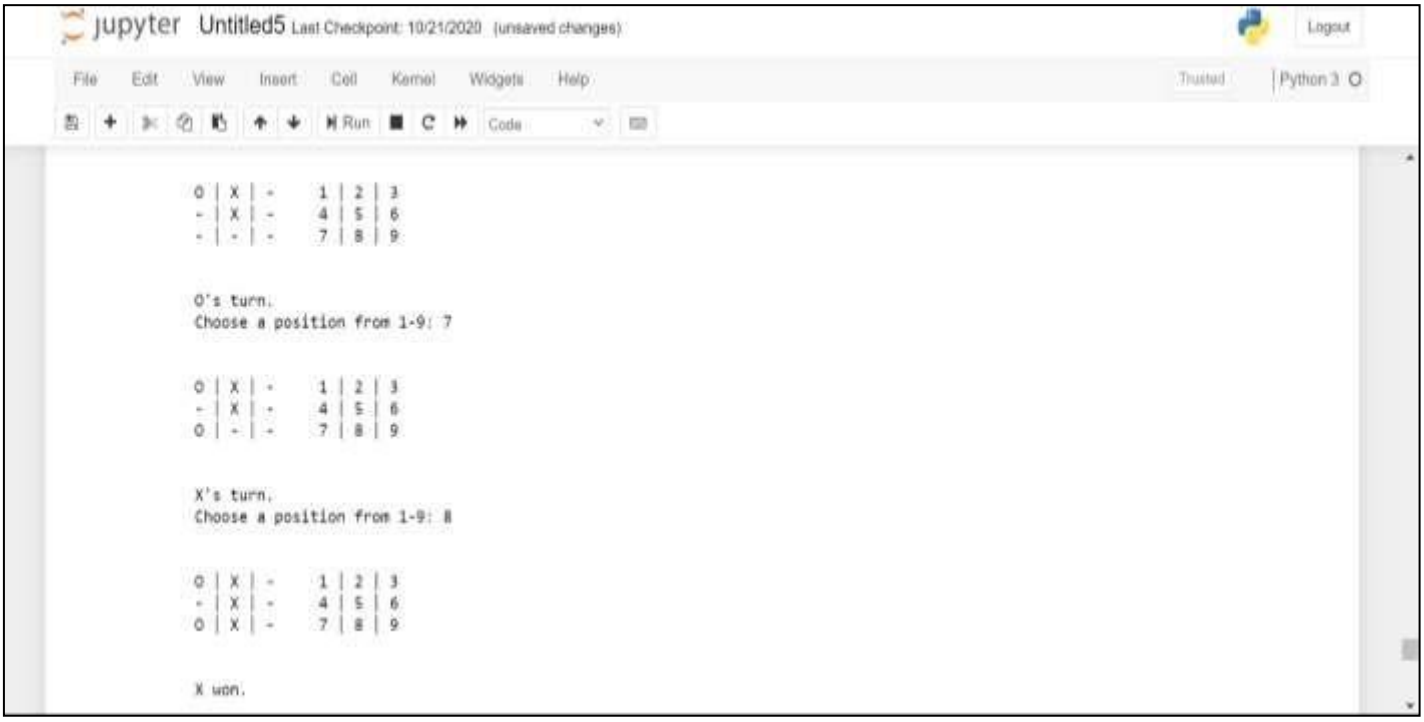
```
    print("You can't go there. Go again.")

  # Put the game piece on the board
  board[position] = player

  # Show the game board
  display_board()


# Check if the game is over
def check_if_game_over():
check_for_winner()
check_for_tie()


# Check to see if somebody has won
def check_for_winner():
  # Set global variables
  global winner
  # Check if there was a winner anywhere
  row_winner = check_rows()
  column_winner = check_columns()
  diagonal_winner = check_diagonals()
  # Get the winner
  if row_winner:
    winner = row_winner
  elif column_winner:
  winner = column_winner
  elif diagonal_winner:
    winner = diagonal_winner
  else:
    winner = None


# Check the rows for a win
def check_rows():
  # Set global variables
```

```
  global game_still_going
  # Check if any of the rows have all the same value (and is not empty)
  row_1 = board[0] == board[1] == board[2] != "-"
  row_2 = board[3] == board[4] == board[5] != "-"
  row_3 = board[6] == board[7] == board[8] != "-"
  # If any row does have a match, flag that there is a win
  if row_1 or row_2 or row_3:
    game_still_going = False
  # Return the winner
  if row_1:
    return board[0]
  elif row_2:
    return board[3]
  elif row_3:
    return board[6]
  # Or return None if there was no winner
  else:
    return None


# Check the columns for a win
def check_columns():
  # Set global variables
  global game_still_going
  # Check if any of the columns have all the same value (and is not empty)
  column_1 = board[0] == board[3] == board[6] != "-"
  column_2 = board[1] == board[4] == board[7] != "-"
  column_3 = board[2] == board[5] == board[8] != "-"
  # If any row does have a match, flag that there is a win
  if column_1 or column_2 or column_3:
  game_still_going = False
  # Return the winner
  if column_1:
    return board[0]
  elif column_2:
  return board[1]
  elif column_3:
```

```
    return board[2]
  # Or return None if there was no winner
  else:
    return None


# Check the diagonals for a win
def check_diagonals():
  # Set global variables
  global game_still_going
  # Check if any of the columns have all the same value (and is not empty)
  diagonal_1 = board[0] == board[4] == board[8] != "-"
  diagonal_2 = board[2] == board[4] == board[6] != "-"
  # If any row does have a match, flag that there is a win
  if diagonal_1 or diagonal_2:
    game_still_going = False
  # Return the winner
  if diagonal_1:
    return board[0]
  elif diagonal_2:
    return board[2]
  # Or return None if there was no winner
  else:
    return None


# Check if there is a tie
def check_for_tie():
  # Set global variables
  global game_still_going
  # If board is full
  if "-" not in board:
    game_still_going = False
    return True
  # Else there is no tie
  else:
    return False
```

```
# Flip the current player from X to O, or O to X
def flip_player():
  # Global variables we need
  global current_player
  # If the current player was X, make it O
  if current_player == "X":
  current_player = "O"
  # Or if the current player was O, make it X
  elif current_player == "O":
    current_player = "X"


# ------------ Start Execution -------------
# Play a game of tic tac toe
play_game()
```

JUPYTER NOTEBOOK OUTPUT 1:



CONCLUSIONS:

Thus, python programming language was used to develop a user interactive, 2-D Tic Tac Game in the Anaconda Environment.

REFERENCE:

https://en.wikipedia.org/wiki/Tic-tac-toe

Index - Jupyter Notebook

notebooks.gesis.org/binder/jupyter/user/ipython-ipython-in-depth-yd7ywipy/notebooks/binder/Index.ipynb

jupyter Index (unsaved changes)

Visit repo | Copy Binder link

File Edit View Insert Cell Kernel Widgets Help | Trusted | Python 3

Run | Markdown | Download | GitHub | Binder | Memory: 131 / 8192 MB

In [*]:
```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 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)

# 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)

# Checks whether 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)

# Checks whether the player has three
# of their marks in a vertical row
```

Index - Jupyter Notebook

notebooks.gesis.org/binder/jupyter/user/ipython-ipython-in-depth-yd7ywipy/notebooks/binder/Index.ipynb

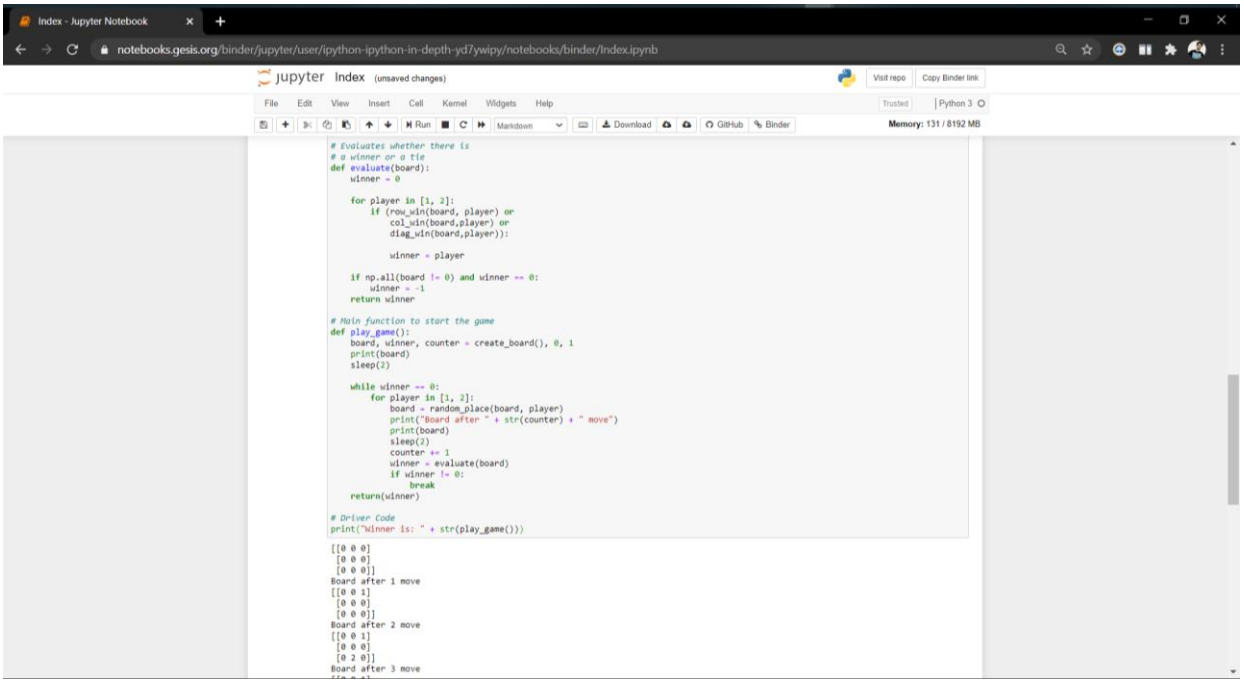
jupyter Index (unsaved changes)

Visit repo | Copy Binder link

File Edit View Insert Cell Kernel Widgets Help | Trusted | Python 3

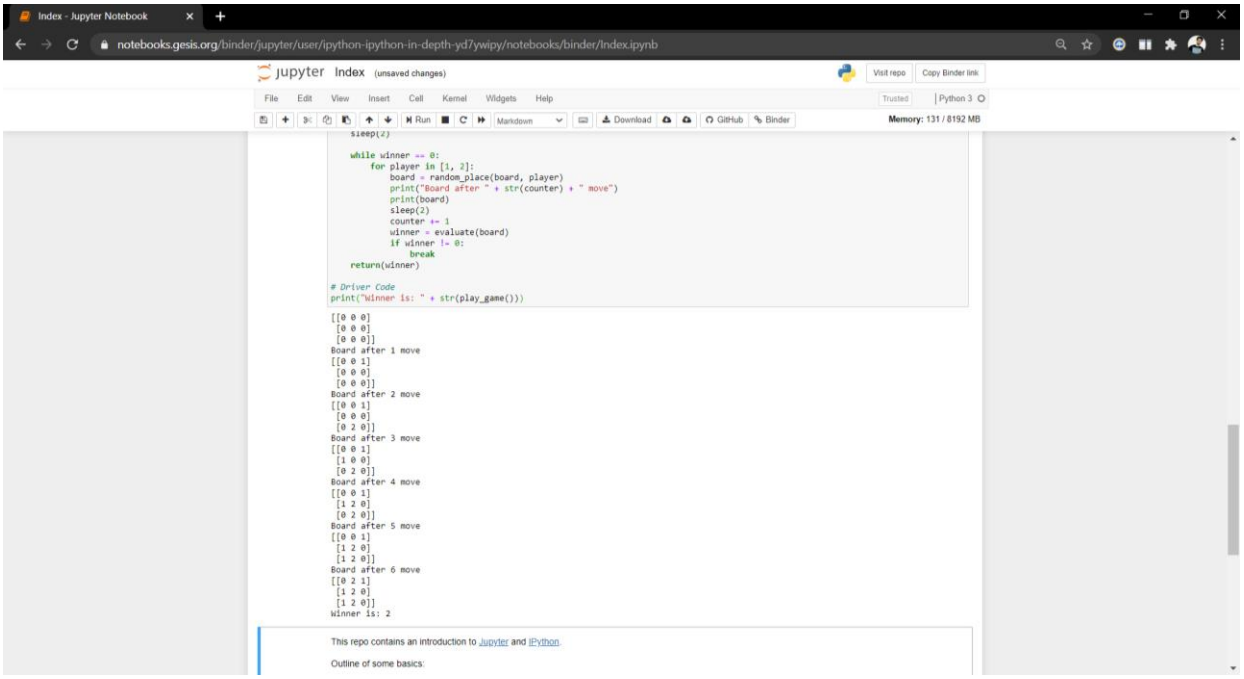
Run | Markdown | Download | GitHub | Binder | Memory: 131 / 8192 MB

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

# Checks whether the player has three
# of their marks in a diagonal row
def diag_win(board, player):
    win = True
    y = 0
    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)):
            y = len(board) - 1 - x
            if board[x, y] != player:
                win = False
    return win

# Evaluates whether 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:
        winner = -1
    return winner

# Main function to start the game
```



```
# Evaluates whether 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:
        winner = -1
    return winner

# 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)
            if winner != 0:
                break
    return(winner)

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

```
[[0 0 0]
 [0 0 0]
 [0 0 0]]
Board after 1 move
[[0 0 1]
 [0 0 0]
 [0 0 0]]
Board after 2 move
[[0 0 1]
 [0 0 0]
 [0 2 0]]
Board after 3 move
```



```
    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)
            if winner != 0:
                break
    return(winner)

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

```
[[0 0 0]
 [0 0 0]
 [0 0 0]]
Board after 1 move
[[0 0 1]
 [0 0 0]
 [0 0 0]]
Board after 2 move
[[0 0 1]
 [0 0 0]
 [0 2 0]]
Board after 3 move
[[0 0 1]
 [1 0 0]
 [0 2 0]]
Board after 4 move
[[0 0 1]
 [1 2 0]
 [0 2 0]]
Board after 5 move
[[0 0 1]
 [1 2 0]
 [1 2 0]]
Board after 6 move
[[0 2 1]
 [1 2 0]
 [1 2 0]]
Winner is: 2
```

This repo contains an introduction to Jupyter and IPython.

Outline of some basics: