Final Project: Tic-Tac-Toe

04/03/2024

Group 8:

Nape vinthanage Chani Anjalee

An Vu

Parimal Khanpara

Yasasvie Asara

Amber Ward

Link to our Git hub:

https://github.com/vuquocan1987/FinlandEx.git

Here is a brief description of our project:

This project is based on the game of Tic-Tac-Toe. Our team members collaborated and communicated through Zoom, WhatsApp, Trello, and GitHub to make this project into what it is today. This is a two-player game that runs off of user input. For our first step we defined the conditions needed to be able to win the game in a list using brackets. Our second step was to import displays and widgets to give us the interactive squares for the game. After this, we defined the functions of this project and used classes and subclasses to make this game interactive for the user. Then, we used for, if, and elif loops and conditions to continue this game until there was a clear winner. Finally, we called the game in a seperate cell for the user and a friend to be able to play.

In [12]:
```
win_conditions = [
             [0, 1, 2], [3, 4, 5], [6, 7, 8], # rows
             [0, 3, 6], [1, 4, 7], [2, 5, 8], # columns
             [0, 4, 8], [2, 4, 6] # diagonals
```

```
        ]
# In this cell we defined our rows, columns, and the diagonal portions of the winning co
```

In this cell we defined our rows, columns, and the diagonal portions of the winning conditions of the game. We did this form the start so that we could have our objectives easily identified in the system.

In [13]:
```
import ipywidgets as widgets  # Importing our widgets and displays to allow for the squa
from IPython.display import display, clear_output

class TicTacToe:  # In this portion we defined our first set of classes and subclasses f
    def __init__(self):
        self.board = [' ' for _ in range(9)]
        self.current_turn = 'X'   # We also set the parameters for the boxes and defined
        self.game_over = False
        self.buttons = [widgets.Button(description=' ', layout=widgets.Layout(width='60p
        for button in self.buttons:
            button.on_click(self.on_button_click)

        self.output = widgets.Output()   #Finally, we further defined the portions of the
        self.reset_button = widgets.Button(description="Restart")
        self.reset_button.on_click(self.restart_game)
        self.update_ui()

    def update_ui(self):
        row1 = widgets.HBox(self.buttons[0:3]) # We defined the sub class update_ui of t
        row2 = widgets.HBox(self.buttons[3:6]) #further definition of the subclasses.
        row3 = widgets.HBox(self.buttons[6:9])
        display(widgets.VBox([row1, row2, row3, self.reset_button, self.output])) # Afte

    def restart_game(self, b):
        self.__init__()
    # This portion is where the game will need to restart once the game has ended if the
    def on_button_click(self, b): # If the game is not over then we used the if-with con
        index = self.buttons.index(b)
        if self.board[index] == ' ' and not self.game_over:
            self.board[index] = self.current_turn
            b.description = self.current_turn
            if self.check_win():  # This is where we define and list the conditions need
                with self.output:
                    clear_output()
                    print(f"{self.current_turn} Wins!")
                self.game_over = True
                return
            elif ' ' not in self.board: # However, we also took into account the possibi
                with self.output:
                    clear_output()
                    print("It's a draw!")
                self.game_over = True
                return
            self.current_turn = 'O' if self.current_turn == 'X' else 'X' # This was wher
        self.check_game_status()

    def check_game_status(self): # This is further definition of the game and the player
        with self.output:
            clear_output()
            print(f"{self.current_turn}'s turn")

    def check_win(self):

        for condition in win_conditions:
            if self.board[condition[0]] == self.board[condition[1]] == self.board[condit
                return True
        return False
    # Finally, this is where we check to see if all of the winning conditions have been
    # This is where we Instantiate and display the game.

# When you click on a button in the game the, on_button_click will be called which is wh
# When you win the game , click on the restart button and the game will be called allowi
```

First, we imported our widgets and displays to allow for the square interactive feature of our game. In this portion we defined our first set of classes and subclasses for the game. We also set the parameters for the boxes and defined our first player as X and the second player as O. Finally, we further defined the portions of the self class.

In the second section or line 15 of code, we defined the sub class update_ui of the parent self class which is what allows for the interactive blocks. In the classification def restart, we write that this is where the game will need to restart once the game has ended if the players choose to play again. In the def on button click classification we define and list the conditions needed for either player to win the game using an if-with conditional structure dependant on the entries that the users input. For the if self.check win section we check to see if all of the winning conditions have been met and we display the game along with the winning player. However, we also took into account the possibility for a draw so we used an if-elif loop to define and list the conditions that would be necessary for a draw. In the self. current turn section we further defined who our players are and use an if-elif loop to determine who's turn we are on based on the previous conditions. In the def check game section we further defined the game and the players where we tell the game to tell the players who's turn we are on. Finally, when you click on a button in the game the, on_button_click will be called which is what allows the users to interact with the squares of the game and when you win the game , click on the restart button and the game will be called allowing for another round.

In [15]:
```
game=TicTacToe()
```

```
VBox(children=(HBox(children=(Button(description=' ', layout=Layout(height='60px', width
='60px'), style=Button…
VBox(children=(HBox(children=(Button(description=' ', layout=Layout(height='60px', width
='60px'), style=Button…
```

In [4]:
```
!pip install pyppeteer
!pyppeteer-install
```

```
Requirement already satisfied: pyppeteer in ./anaconda3/lib/python3.11/site-packages (2.
0.0)
Requirement already satisfied: appdirs<2.0.0,>=1.4.3 in ./anaconda3/lib/python3.11/site-
packages (from pyppeteer) (1.4.4)
Requirement already satisfied: certifi>=2023 in ./anaconda3/lib/python3.11/site-packages
(from pyppeteer) (2023.11.17)
Requirement already satisfied: importlib-metadata>=1.4 in ./anaconda3/lib/python3.11/sit
e-packages (from pyppeteer) (6.0.0)
Requirement already satisfied: pyee<12.0.0,>=11.0.0 in ./anaconda3/lib/python3.11/site-p
ackages (from pyppeteer) (11.1.0)
Requirement already satisfied: tqdm<5.0.0,>=4.42.1 in ./anaconda3/lib/python3.11/site-pa
ckages (from pyppeteer) (4.65.0)
Requirement already satisfied: urllib3<2.0.0,>=1.25.8 in ./anaconda3/lib/python3.11/site
-packages (from pyppeteer) (1.26.16)
Requirement already satisfied: websockets<11.0,>=10.0 in ./anaconda3/lib/python3.11/site
-packages (from pyppeteer) (10.4)
Requirement already satisfied: zipp>=0.5 in ./anaconda3/lib/python3.11/site-packages (fr
om importlib-metadata>=1.4->pyppeteer) (3.11.0)
Requirement already satisfied: typing-extensions in ./anaconda3/lib/python3.11/site-pack
ages (from pyee<12.0.0,>=11.0.0->pyppeteer) (4.7.1)
chromium is already installed.
```

This was necessary in order to download the jupyter notebook as a PDF.