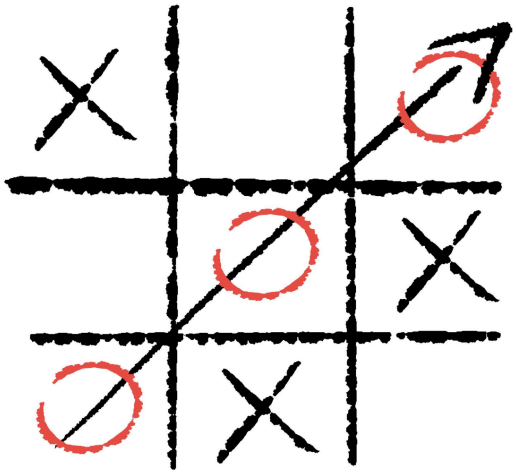
Making a Tic Tac Toe game by using Python

2024.01.20

I used to love playing games and the reason why I started coding was because I wanted to make my games. To make a variety of games, I'm going to start by making simple games, and then developing them into more complicated ones. So, the first game I decided to program was Tic Tac Toe which is a two-player game where one player plays X and the other plays O and takes turns marking on a grid of three-by-three cells. If one of the players gets three marks in a row in any way, the player wins the game (Ramos).



(Screen shot of Tic Tac Toe photo link: https://t0.gstatic.com/licensed-image?q=tbn:ANd9GcQsJEpbrBLWAyjRR82eWUxKb4WpXjIOVB-g-KFOkyaP1zSPympYcoYd61hef9xEnRzT)

The image above is an example display of how I want to create my game.

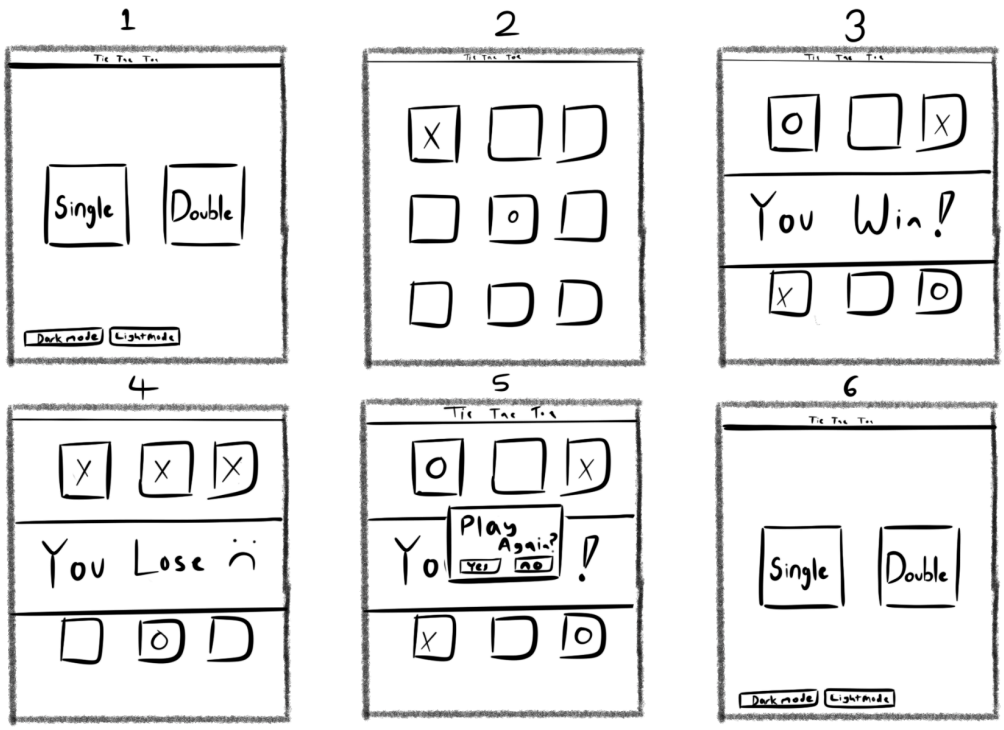
1. Planning

I first thought of what the requirements were to create a Tic Tac Toe game.

1. It should be played with a computer
2. It should be programmed with Python
3. There should be no age limit
4. There should be options to play singles or doubles
5. There should be graphic elements such as buttons and pictures to play the game.

Then with these requirements, I went into more detail about what was going on. The first thing I thought of was the visual of my game since it is the easiest to do and then continued with making the functionalities.

The first thing I did was make basic sketches of how it would look like:



(A sketch made by me of how I want the Tic Tac Toe game would look like)

For the visuals, I was thinking that I would make the screen size 500 by 500 (in pixels) with the first page (the rectangle frame that is described as 1 in the picture above) being the selection of either playing a singles game against a robot or a doubles game against a friend. Also, in the bottom right corner, I wanted to add the option for the user to choose either dark mode or light mode so that aesthetically wise people will be more comfortable with it. I decided to not add other colors and only those options because on most occasions, it's either dark when your eyes hurt too much or light when you can't see anything.

For the second page that the user will go to, which is the actual game of Tic Tac Toe (the rectangle frame that is described as 2 in the picture above), I arranged 3 buttons in 3 rows which in total is 9 buttons. The user and another user will have back-and-forth turns until all the buttons are clicked or if one of the users connects 3 of the same symbol ('O' or an 'X'). From the user's point of view when they're playing against a robot (single player), if the player or the robot wins or ties, there will be a label appearing at the center of the screen saying that one of them has either won or tied (the rectangle frame

that is described as 3 and 4 in the picture above). After it displays who had won or tied, another small screen will pop up at the center of the screen asking the user if they want to play again or not (shown in the image above with the small individual frames that are labeled as 5). If the user has clicked “yes”, it will go back to the first page however if the player clicks “no” the game will entirely shut down and stop functioning.

The first step I did was creating the GUI by adding buttons and labels without the functionality part at the beginning. I started coding the visuals of the 3 main pages: the start page, the doubles page, and the singles page.

I created a GUI window that is 500 by 500 pixels with the title “Tic Tac Toe” on the top.

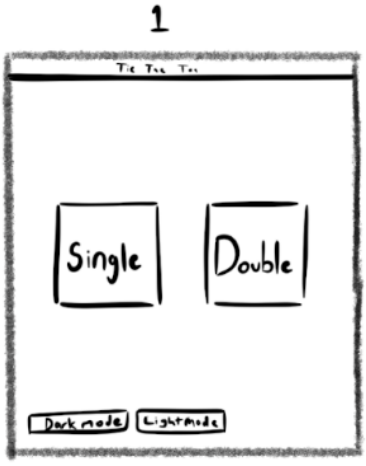
```
from tkinter import* #Importing everything from the tkinter module

window = Tk() #Creating a GUI window
window.title('Tic Tac Toe')#Setting the title of the window
window.geometry('500x500') #Setting the size of the window by 500 x 500 in
pixels
window.mainloop() #Running the main event loop to display the window and handle
events
```

With this code, the display will look like:

(Screenshot of the display of GUI with the use of the code above)

This is the main base of my Tic Tac Toe game where I will start adding visual elements like buttons and labels.

Following my action plan, I started with the start page first. Which looks like the image in Figure 2. There is the option to either play single-player or double-player. Also, there is the option to change the mood of the GUI on the bottom left corner of the start page GUI.



(Screenshot of the sketch of how the starting page will look like)

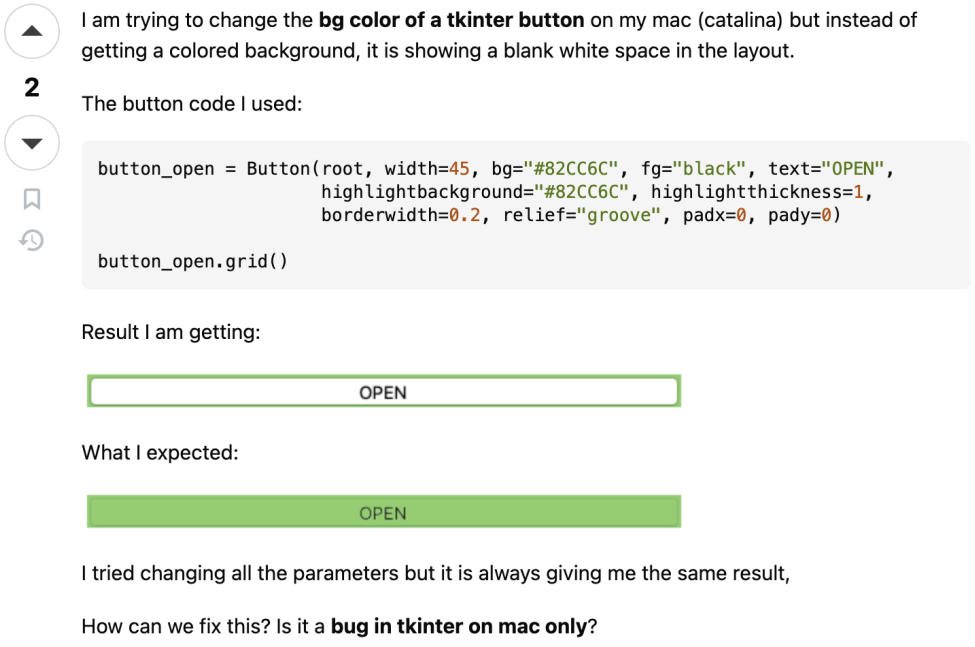
After programming the start page without any functionality, the code looked like this:

```
from tkinter import* #Importing everything from the tkinter module
from tkmacosx import Button #Importing a button widget specifically designed
for macOS

def Startpage():#Defining a function for the start page

   #Creating a frame as the background of the window
   Background = Frame(window, width=500, height=500, bg='white')
   #Area of placement of the Background which is (0,0)
   Background.place(x=0, y=0)

   #Creating a button for single player mode
   single = Button(window, bg='grey', text="Single\nPlayer", font=("Ariel",
50))
   #Area of placement of the button which is (30,250) and also adjusting the
height
   single.place(x=30, y=250, width=200, height=150)

   #Creating a button for double player mode
   doubles = Button(window, bg='grey', text='Double\nPlayer', font=("Ariel",
50))
   #Area of placement of the button which is (270,250) and also adjusting the
height
   doubles.place(x=270, y=250, width=200, height=150)
```

```
    #Creating a button for activating dark mode
    dark = Button(window, text="Dark mode", bg='Grey')
    #Area of placement of the button which is (10, 460)
    dark.place(x=10, y=460)

    #Creating a button for activating light mode
    light = Button(window, text="Light mode")
    #Area of placement of the button which is (120, 460)
    light.place(x=120, y=460)

#GUI
window = Tk() #Creating a GUI window
window.title('Tic Tac Toe')#Setting the title of the window
window.geometry('500x500') #Setting the size of the window by 500 x 500 in
pixels

Startpage() #Calling the Startpage function to display the start page
window.mainloop() #Running the main event loop to display the window and handle
events
```

While creating the code, I found out halfway through that the background color and the font color of the button wouldn't change if I just added any name of the color. I've tried it on a Windows computer and it was able to function normally however when I coded it on a Mac computer, it didn't work. Since I want it to be able to work just as well on a Windows computer on a Mac computer, I decided to go on a search about the problem because other people might have faced the same situation as me.

This is what I have found online:



2

I am trying to change the **bg color of a tkinter button** on my mac (catalina) but instead of getting a colored background, it is showing a blank white space in the layout.

The button code I used:

```
button_open = Button(root, width=45, bg="#82CC6C", fg="black", text="OPEN",
                     highlightbackground="#82CC6C", highlightthickness=1,
                     borderwidth=0.2, relief="groove", padx=0, pady=0)

button_open.grid()
```

Result I am getting:

OPEN

What I expected:

OPEN

I tried changing all the parameters but it is always giving me the same result,

How can we fix this? Is it a **bug in tkinter on mac only**?

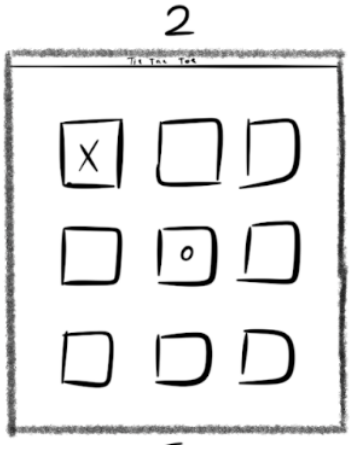
(Screenshot of stack overflow which is a website where people share their problems to solve it as a community: https://stackoverflow.com/questions/67388140/button-width-and-height-not-working-in-tkinter-tkmacosx)

I realized that it was just a problem on Mac in general and I've read the replies below where someone had made a tkmacosx module where it can be imported in codes so that the color of the button could change in Mac computers. So it was able to help me with coloring buttons.

After finishing the start page layout, the code looked like this:

(Screenshot of the display of GUI with the use of the code above)

The next page that I had to create buttons and labels for was the page where the game happens.



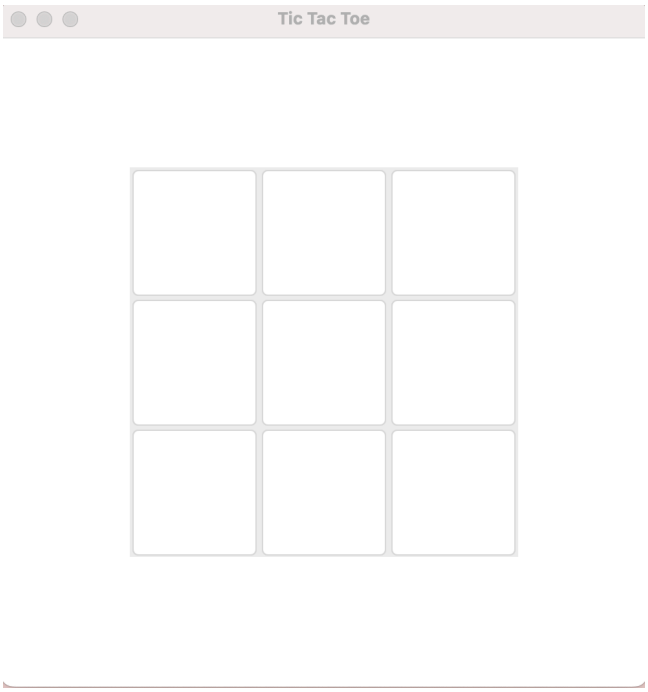
(Screenshot of the sketch of how the gameplay page will look like)

After programming the start page without any functionality, the code looked like this:

```
def gameStart(): #Defining a function for the gameplay page

   #Creating a frame as the background of the window
   Background1 = Frame(window, width=500, height=500, bg='white')
   #Area of placement of the Background which is (0,0)
   Background1.place(x=0, y=0)

   #Creating a button for the first cell of the game grid
   btn1 = Button(window, bg='white', font=("Ariel", 40))
   #Placing the button at specified coordinates and dimensions
   btn1.place(x=100, y=100, width=100, height=100)
```

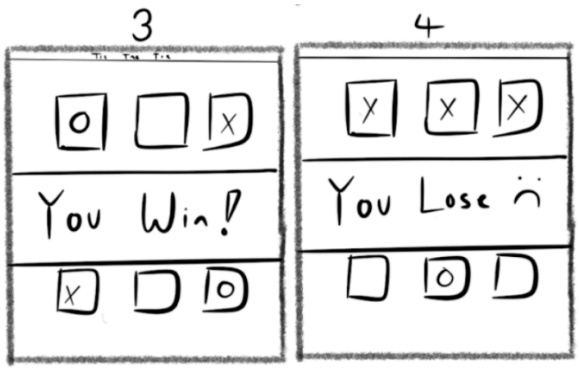
(The code is added below the Startpage function and is part of a code and not the full code)

I created 9 buttons which where called btn1, btn2, btn3, btn4, btn5, btn6, btn7, btn8, and btn9. Were coded just like the creation of btn1 on the code above but with the placement different. So then, the GUI looked like the image below.



(Screenshot of the display of GUI with the use of the code above)

Then I created a frame that displayed if ‘O’ won or ‘X’ won or if they both tied like the images below.



(Screenshot of the sketches of how the display would look like)

The code for adding the frame and the label of who won:

```
def gameFinish():#Defining a function when the game ends
   #Game result (who won)

   #Creating a frame for the display bar at the center
   Victory = Frame(window, width=400, height=150, bg='light blue')
   #Area of placement of the Victory frame which is at (50, 170)
   Victory.place(x=50, y=170)

   #Label display of the winner
   VicLabel = Label(window, text='X wins!', font=("Ariel", 50), bg='light
blue')
   #Area of placement of the Victory label which is at (160, 170)
   VicLabel.place(x=160, y=170)
```

(The code is added below the game start function and is part of a code and not the full completed code as there is no functionality added)

All the visuals were now coded so then I continued to the functionalities of the code. I first divided what I had to do into two sections: single-player mode and double-player mode. The reason I did this was because the single-player mode would have to include an AI bot that would be able to play against the person and I’ve never tried creating one before so I put that priority at last and just focused on creating the double-player mode. Then, I added the functionality that when the user clicked the double player mode button, it would send them to the game page to play Tic Tac Toe.

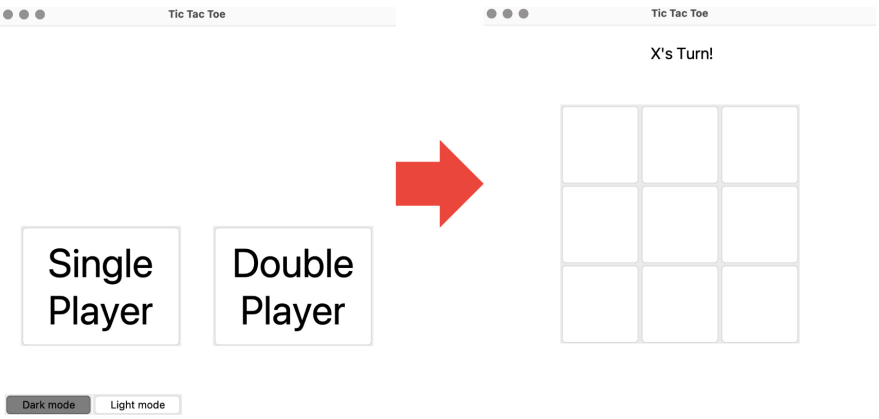
```
doubles = Button(window, bg=”Grey”, text='Double\nPlayer', font=("Ariel", 50))
doubles.place(x=270, y=250, width=200, height=150)
doubles['command'] = gameStart #ADDED CODE
```

(The code part of the Startgame function and is not the full complete code)

The double player button had the command “gameStart” which would delete all the buttons, labels, and frames of the beginning of the page and then display the nine buttons to play the game. In the game start function, I had to include:

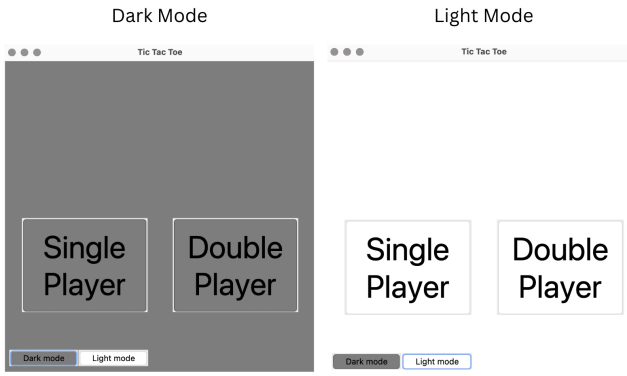
```
def gameStart():
  #By adding global to these variables, it allows them to be modified within
the function's scope while referencing the global variable outside the
function.
  global Background, single, doubles, dark, light

  Background.destroy()
  single.destroy()
  doubles.destroy()
  dark.destroy()
  light.destroy()
```

(The code part of the game start function is not the full complete code)



(Screenshot of how the functionality would work if the user had clicked the Double Player button)

The next thing that I did was make the “Dark mode” and “Light mode” buttons work. What these buttons would do is change the mood of the GUI.



(Screenshot of how the color of the GUI looks like if the user had clicked on the “Dark mode” button and the “Light mode” button)

To make this happen, I created another function called “themeChange” which will function when either the “dark mode” or the “light mode” is pressed like the code below.

```
def Startpage():
    ...
    dark = Button(window, text="Dark mode", bg='Grey')
    dark.place(x=10, y=460)
    #Assigning a command to the dark mode button to change the theme to 'dark'
when clicked
    dark['command'] = lambda: themeChange('dark')#ADDED CODE

    light = Button(window, text="Light mode")
    light.place(x=120, y=460)
    #Assigning a command to the light mode button to change the theme to 'light'
when clicked
    light['command'] = lambda: themeChange('light')#ADDED CODE
    ...

def themeChange(select):
    #Global variables that will be modified within the function
    global Background, single, doubles, dark, light
    global theme, bgcolor, fgcolor

    theme = select #Set the theme based on the provided parameter

    #Adjust background and foreground colors based on the selected theme
    if theme == 'light':
        bgcolor = 'white'
        fgcolor = 'black'
    else:
        bgcolor = 'grey'
        fgcolor = 'white'

    #Update background colors of widgets
    Background['bg'] = bgcolor
    single['bg'] = bgcolor
    doubles['bg'] = bgcolor


#Initialize theme-related variables
theme = 'light'  #Default theme
bgcolor = 'white'  #Default background color
fgcolor = 'black'  #Default foreground color
```

I realized that if I had to name any background color of a button, label, or frame would have to be “bgcolor” so that it can change color when either the “dark mode” or the “light mode” had been pressed. So then I went through the code that I already had and replaced the color of a background that was “white” at first with “bgcolor.”

I then went on to the functions that the actual game of Tic Tac Toe had needed. For a double-player mode, the functions I needed to add were:

1. Mark the button when one of the players clicks a button out of the 9 buttons to either an “O” or an “X”
    - Make sure that the button that has been marked cannot be clicked again.
2. Add a checker if there are any winners yet that have connected three “O”s in a row or three “X”s in a row.
    - If either “O” or “X” connects three in a row, display who had won
    - If “O” and “X” ties, display who had tied
        - After five seconds, show a dialogue that would ask the user if they would want to play the game again.

This is the code for the marking function:

```
def gameStart():
   global btn1, btn2, btn3, btn4, btn5, btn6, btn7, btn8, btn9, Turn, turnlb
   global Background, single, doubles, dark, light

   ...
   btn1 = Button(window, bg=bgcolor, font=("Ariel", 40))
   btn1.place(x=100, y=100, width=100, height=100)
   #when a btn1 is clicked, it will start the marking function
   btn1['command'] = lambda: marking(btn1)
   ...

def marking(btn):

   global turn, gameover
   #When the game is over and someone accidentally clicks a button, don’t mark
the button.
   if gameover:
       return
   #if a button that has already been marks has been clicked, don’t mark it
   if btn['text'] != '':
       return

   if turn == 'X': #if it is X's turn
       btn['text'] = 'X' #mark the button clicked with "X"
       turn = 'O' #change the turn to “O”

   else: #if it is O's turn
       btn['text'] = 'O' #mark the button clicked with "O"
       turn = 'X' #change the turn to “O”

   #change the label above to either "O" or "X" depending who's turn it is
```

```
    turnlb['text'] = turn +"'s turn!"
```

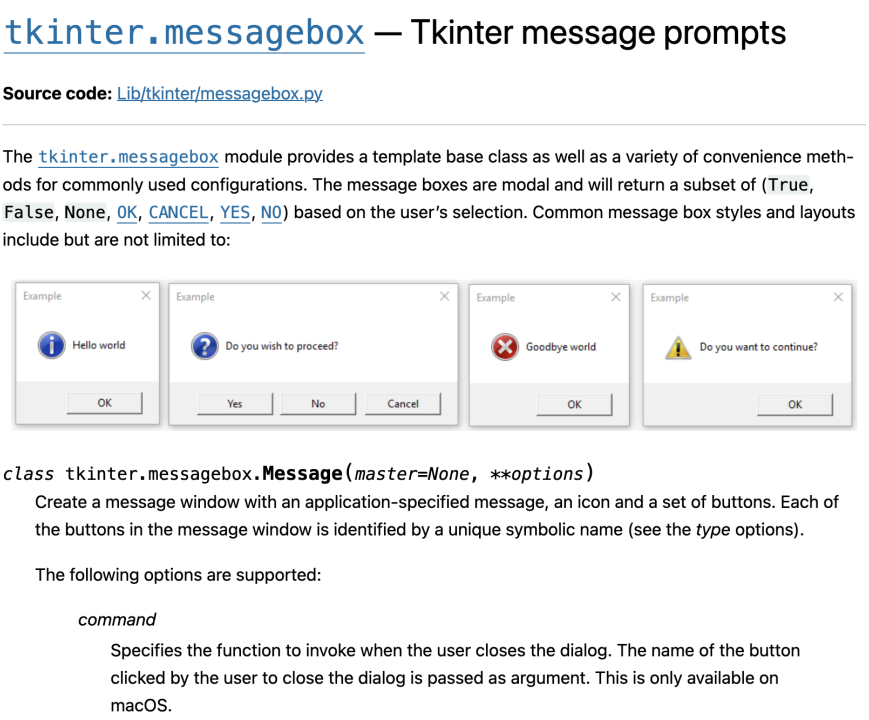
When one of the 9 buttons is clicked, it will go to the marking function which will then mark the button that was clicked as either “O” or “X” depending on whose turn it is. Before the marking function determines whether to mark it “O” or “X,” the function first determines the situation of when the game is over to not mark any buttons, and when a button is already marked, it doesn’t change the mark again. After the marking function marks the button, it then checks whether “O” or “X” has won or tied.

```
result = check()  #After either "O" or "X" marks a button, check if either of
them had connected three in a row

# if x wins, 'X', if o wins 'O', if it's tied '=', if the game isn't finished
'x'

if result == 'X': #When X has connected 3 "X"s
    finish = gameFinish() #Display the label and frame that X has won
    VicLabel['text'] = "X has won\nthe game!" #Change the label that X has won
    Victory.place(x=50, y=170) #Area of placement of the frame
    VicLabel.place(x=100, y=170) #Area of placement of the label
    gameover = True #It is notified that the game is over
    window.after(3000, notify) #Show the dialogue if the user wants to play the
game again

elif result == 'O': #When O has connected 3 "O"s
    finish = gameFinish() #Display the label and frame that O has won
    VicLabel['text'] = "O has won\nthe game!" #Change the label that O has won
    Victory.place(x=50, y=170) #Area of placement of the frame
    VicLabel.place(x=100, y=170) #Area of placement of the label
    gameover = True #It is notified that the game is over
    window.after(3000, notify) #Show the dialogue if the user wants to play the
game again

elif result == '=': #When nor X or O has connected 3 "O"s or "X"s
    finish = gameFinish() #Display the label and frame that they have tied
    VicLabel['text'] = "The game\nis tied!" #change the label that they tied
    Victory.place(x=50, y=170) #area of placement of the frame
    VicLabel.place(x=100, y=170) #area of placement of the label
    gameover = True #it is notified that the game is over
    window.after(3000, notify) #show the dialogue if the user wants to play the
game again


def check():  #who won?
    if btn1["text"] == btn2["text"] == btn3["text"] == 'X': #if the top row has
three consecutive "X"s
        return "X" #return "X" to the result
    elif btn1["text"] == btn2["text"] == btn3["text"] == 'O': #if the top row
has three consecutive "O"s
        return "O" #return "O" to the result
```

```
    ... #there are many different arguments of who won (just not shown)
```

The marking function after marking connects with the check function which has a list of arguments if “O” won, “X” won, or if they tied. The check function would return an “X”, “O”, or a “=” to the result which then determines who has won. For example, if “O” has connected three in a row, the result variable will become “O” which will go to the game finish function that displays a frame and a label that “O” had won the game.

When the game is completely over a victory or a tie, after three seconds I created a dialogue that would appear asking the user if they had wanted to play the game again. If they have selected “Yes” the GUI will go back to the beginning of the page but if they have selected “No” the game will entirely stop and delete the GUI. I didn’t know at the beginning how to do a dialogue so I had to do some research.



tkinter.messagebox — Tkinter message prompts

Source code: Lib/tkinter/messagebox.py

The tkinter.messagebox module provides a template base class as well as a variety of convenience methods for commonly used configurations. The message boxes are modal and will return a subset of (True, False, None, OK, CANCEL, YES, NO) based on the user's selection. Common message box styles and layouts include but are not limited to:

class tkinter.messagebox.**Message**(*master=None*, ***options*)

Create a message window with an application-specified message, an icon and a set of buttons. Each of the buttons in the message window is identified by a unique symbolic name (see the *type* options).

The following options are supported:

command

Specifies the function to invoke when the user closes the dialog. The name of the button clicked by the user to close the dialog is passed as argument. This is only available on macOS.

(Screenshot of a website that demonstrated the use of message boxes in Tkinter. Link: https://docs.python.org/3/library/tkinter.messagebox.html)

The image above shows the source that I looked at to see how I understood message boxes and learned how to apply them to my code.

Then this is how the code looked after applying my knowledge:

```
#Allows me to add a messagebox that appears at the end of the game that asks
the user if they want to play the game again
from tkinter.simpledialog import messagebox as sd

def marking(btn):
```

```
    ...
    if result == 'X':
        print("X wins!")
        finish = gameFinish()
        VicLabel['text'] = "X has won\nthe game!"
        Victory.place(x=50, y=170)
        VicLabel.place(x=100, y=170)
        gameover = True
        window.after(3000,notify) #notify function appears
    ...

def notify():
    global btn1, btn2, btn3, btn4, btn5, btn6, btn7, btn8, btn9, turnlb,
VicLabel, Victory, gameover, turn
    but = [btn1, btn2, btn3, btn4, btn5, btn6, btn7, btn8, btn9]

    select = sd.askyesno(("Game Over!"), "Do you want to play more?")
    if select == False: #if the user selected no
        window.destroy() #delete the GUI

    else: #if the user selected yes
        for i in but: #delete all the 9 buttons
            i.destroy()
        turnlb.destroy() #delete the turnlb
        VicLabel.destroy() #delete the VicLabel
        Victory.destroy() #delete the Victory
        turn = "X" #reset the turn to "X"s turn
        gameover = False #reset the gameover variable
        Startpage() #go back to the beginning of the game which is the start
page
```

The double-player mode was now finished and completed and I just had to program the single-player mode area which was something difficult for me since I had to create an AI bot that would play against the person who was playing by themselves.

```
...
def StartpageSinlge():
    global Startpage, playsingle
    playsingle = True #single player mode is turned on
    gameStart() #goes to the page where the game is played
    aiTurn() #the AI bot that is used to play against the single player is
summoned

def Startpage():
    ...
    single = Button(window, bg=bgcolor, text="Single\nPlayer", font=("Ariel",
50))
    single.place(x=30, y=250, width=200, height=150)
```

```
   single['command'] = StartpageSinlge #when the Single Player button is
clicked the StartpageSingle function is turned on
   …
```

Because I had the visuals all done and some of the double-player functions were the same as the single-player function, it was faster to set up the game. By looking at the code above, you can see that when the single button is pressed, it will start the StartpageSingle function which will begin the gameStart function and summon an AI bot. The gameStart function was already finished I just input it but then I had to create an AI bot.

For the AI bot, I first began thinking of the situations while playing the game of Tic Tac Toe. The first scenario I thought of was when there are two consecutive “X”s (since the user will always be “X” and the AI bot will be “O”) the bot would place the “O” that would block the user from winning the game.

Here are the scenarios that I added to make the AI bot to do.:

- When the user’s turn is done and the center is not marked, mark the center
- Horizontally, vertically, and diagonally if there are two “O”s in a row, mark the last position with “O” to win the game
- Horizontally, vertically, and diagonally if there are two “X”s in a row, mark the last position with “O” to prevent “X” from winning.

```
def aiTurn():
   global
playsingle,btn1,btn2,btn3,btn4,btn5,btn6,btn7,btn8,btn9
   but = [btn1,btn2,btn3,btn4,btn5,btn6,btn7,btn8,btn9]
   if not playsingle:
       return
   if gameover:
       playsingle = False
       return
   if turn == 'O': # when it is "O"s turn (the AI bot's turn)
       if but[4]['text'] == '': #and there is no markings done
in the center (which is button four)
           marking(but[4]) #mark button 4 with "O"
       if sorted([btn1["text"], btn2["text"], btn3["text"]]) ==
['','O','O']: #when the first row on the top has two Os
           for i in range(0,3): #mark the final button with "O"
               if but[i]['text'] == '':
                   marking(but[i])
                   break
…
```

After this code was finished, I connected this function to the single-player mode button and finished making the game.