```python
# Tic Tac Toe with GUI
# 5/10/21

# The way that this works is that you need to set the button's
# text to 'X' or 'O' (buttonName.text = "X"). Since these buttons
# all have the same name though and are stored in the list, we use
# currBoard[pos].text = "X" instead.

import guizero as gui
from math import floor
from random import randint
from time import sleep

def num_to_grid(gridNum):
  """
  Takes a grid number (1-9) and returns the number as
  a grid array (x, y)
  """

  x = gridNum % 3
  y = floor(gridNum / 3)
  return (x, y)

def valid_location(currBoard, location):
    """
    Takes in the current board and a potential location and checks if placing
    something in that square is a valid move or not. Ends by returning true
    if the move's valid and false otherwise.
    """

    if (currBoard[location].text != " "): # Checks if the location is taken
        return False
    else:
        return True

def win_checker(currBoard):
    """
    Takes in the current board, finds whether or not someone can win
    and who wins, and returns a tuple with these two parts.
    """

    # This var is all of the locations that need to be checked to look over the 8 possible ways to win
    # 8 checks - 3 vertical, 3 horizontal, 2 crosses
    locationsToCheck = [[0, 3, 6], [1, 4, 7], [2, 5, 8], [0, 1, 2], [3, 4, 5], [6, 7, 8], [0, 4, 8], [2, 4, 6]]

    # Checks if any win condition is true
    for i in range(8):
        pos1, pos2, pos3 = locationsToCheck[i]
        # If someone has 3 in a row, then True is returned
        if ((currBoard[pos1].text == currBoard[pos2].text) and (currBoard[pos2].text == currBoard[pos3].text) and (currBoard[pos1].text != " ")):
            return (True, currBoard[pos1].text)

    # Final return statement which says that no one has won
```

```python
        return (False, "none")

    def tie_checker(currBoard):
        """
        Takes in the current board and checks if there's a tie. To check for a tie,
        the script searches for whether or not the board is filled. Returns true
        if the board is filled and false if not.
        """

        # Finds how many squares are filled in
        filledInCount = 0
        for pos in currBoard:
            if (pos.text != " "):
                filledInCount += 1

        # Checks if all the squares are filled (if there's a tie)
        if (filledInCount == 9):
            return True
        else:
            return False

    def flip_symbol(symbol):
        """
        Takes in the symbol of the current player and flips it
        """

        global currSymbol
        if (symbol == "X"):
            currSymbol = "O"
        else:
            currSymbol = "X"

    def is_game_over(numOfPlayers):
        """
        Checks if the game is over and calls the final print function
        if it is.
        """

        # Checks if someone has won or there is a tie
        if ((win_checker(currBoard))[0] or tie_checker(currBoard)):
            return (final_printer(currBoard, numOfPlayers))
        else:
            return 0

    def places_to_win(currBoard, checkingFor):
        """
        Takes in the current board and the enemy's symbol and checks if
        the enemy can win next turn (has 2 in a row with an empty space
        after). All of the positions where the enemy could win next turn
        are then returned.
        """

        def win_pos_finder(posChecking):
            """
```

Takes in a list of 3 positions that would allow someone to win
and checks if the enemy has 2/3 of them. If this is the case, the
potential win position is returned. Otherwise, 10 is returned as
a placeholder.
"""

```python
    if (currBoard[posChecking[0]].text == currBoard[posChecking[1]].text and currBoard[posChecking[0]].text == checkingFor and currBoard[posChecking[2]].text == " "):
        return posChecking[2] # Returns the 3rd element if that position is where the player could win
    elif (currBoard[posChecking[1]].text == currBoard[posChecking[2]].text and currBoard[posChecking[1]].text == checkingFor and currBoard[posChecking[0]].text == " "):
        return posChecking[0] # Returns the 1st element if that position is where the player could win
    elif (currBoard[posChecking[0]].text == currBoard[posChecking[2]].text and currBoard[posChecking[0]].text == checkingFor and currBoard[posChecking[1]].text == " "):
        return posChecking[1] # Returns the 2nd element if that position is where the player could win
    else:
        return 10 # Returns 10 if there is no position where the player could win (for this specific way to win)

    locationsToCheck = [[0, 3, 6], [1, 4, 7], [2, 5, 8], [0, 1, 2], [3, 4, 5], [6, 7, 8], [0, 4, 8], [2, 4, 6]] # List of all of the ways to win
    posToBlockWin = []

    # Removes 10's (placeholders) from the list of positions to block
    for i in range(8):
        potentialLocation = win_pos_finder(locationsToCheck[i])
        if (potentialLocation != 10):
            posToBlockWin.append(potentialLocation)

    return posToBlockWin # A list of all of the places where the enemy could win next turn

def random_location(currBoard):
    """
    Chooses a random valid location for the bot to place its 'X' or 'O'
    this turn. This location is then returned.
    """

    location = randint(0, 8) # Picks a random move location
    while not(valid_location(currBoard, location)): # Checks if the move is valid
        location = randint(0, 8) # Picks a new random move location

    return location # Returns a random move location

def game_board_colorer(pos):
    """
    Takes in an integer (0-8) and changes the text color for the
    button in that location on the board.
    """

    if (currSymbol == "X"): # Colors X square's text green
        currBoard[pos].text_color = "#3ffc0a"
    elif (currSymbol == "O"): # Colors O square's text magenta
        currBoard[pos].text_color = "#f505e5"

def bot_turn(opponentSymbol, friendlySymbol):
```

```python
    """
    Takes the current board, the enemy's symbol (like 'X'), and the
    friendly symbol (like 'O'). It then has the bot 'choose' a
    position and place its symbol there before printing the new
    board. It ends by returning the current board.
    """

    if (win_checker(currBoard)[0] or tie_checker(currBoard)):
        return # Doesn't let the bot take a turn if the game is over

    # Finds a location where the player needs to win next turn
    squaresToProtect = places_to_win(currBoard, opponentSymbol)

    # Finds where to place the bot's 'O' for this turn
    if (len(squaresToProtect) >= 1): # Checks if there's a way the enemy can win
        if (randint(0, 2) == 0): # Has a 2/3 chance to protect where the enemy can win
            toPlace = random_location(currBoard) # Picks a random location to place the 'O'
        else: # Chooses the protecting option
            toPlace = squaresToProtect[0]
    else: # Since there's nowhere to proteect, the bot chooses a random spot
        toPlace = random_location(currBoard)

    # Adds the move to the board and prints the new board
    currBoard[toPlace].text = friendlySymbol

    # Gets the board ready for the next turn
    game_board_colorer(toPlace)
    flip_symbol(currSymbol)

def reset_symbol():
    """
    This functions resets the global variable currSymbol to 'X'.
    """

    global currSymbol
    currSymbol = "X" # Sets currSymbol to 'X'

def who_is_first(whoIsFirst):
    """
    After the player turn number selector is pressed, this function
    will create the new board or button functions for that situation.
    """

    orderSelector.visible = False # Makes the order selector invisble

    global botNeedsTurn

    if (whoIsFirst == 1):
        # Tells the game that the player goes first
        creatorInfo = ("X", "player_first")
        botNeedsTurn = False
    else:
        # Tells the game that the bot goes first
        creatorInfo = ("O", "bot_first")
```

```python
        botNeedsTurn = True

        # Makes the instructional message from before invisible
        newInstructionalMessage.visible = False
        game_creator(creatorInfo[0], creatorInfo[1]) # Creates the game

def num_players_selector(x):
    """
    Takes in the number of players and helps sets up the
    next phase. If there's one player, the player will be
    able to select whether they go first or not. Otherwise,
    it sets up the game board.
    """

    # Makes the selector for number of players and instructional message invisible
    numOfPlayersBoard.visible = False
    instructionMessage.visible = False

    if (x == 1): # 1 player
        # Instructional text
        global newInstructionalMessage
        newInstructionalMessage = gui.Text(app, color = "white", text= "\nClick '1' to go first or '2' to go second")

        # Board for the player to pick if they go first or not
        global orderSelector
        orderSelector = gui.Box(app, layout="grid")

        for x in range(1, 3):
            button = gui.PushButton(orderSelector, command=who_is_first, args=[x], text=str(x), grid=[x, 0], width=5, height=3)
            button.text_color = "green"

    elif (x == 2): # 2 players board creator
        game_creator("X", "two_players")
    else: # 0 players (2 bots) board creator
        game_creator("X", "two_bots")

def main(createApp):
    """
    The starting and primary function of the program.
    It sets up the app, allows the user to enter the
    number of players, and displays the app.
    """

    if (createApp): # Checks that the app needs to be created
        # App creation
        global app
        app = gui.App(title="Tic Tace Toe")
        app.bg = "black"

    # Instructional text
    global instructionMessage
    instructionMessage = gui.Text(app, color = "white", text="\nSelect the number of players")
```

```python
  # Board Creation
  global numOfPlayersBoard
  numOfPlayersBoard = gui.Box(app, layout="grid")

  # Button creation
  for x in range(3):
    button = gui.PushButton(numOfPlayersBoard, command=num_players_selector, args=[x], text=str(x), grid=[x, 0]
, width=5, height=3)
    button.text_color = "green"

  # Displaying the app
  app.display()

def board_resetter():
  """
  Resets the board by clearing the buttons, resetting
  the symbol, and making the reset button and win
  message invisible.
  """

  for i in range(9): # Clears the buttons' text
    currBoard[i].text = " "
  reset_symbol() # Resets the symbol
  appResetter.visible = False # Makes the reset button invisible
  winMessage.visible = False # Makes the win message invisible

def two_bots_button(pos, currBoard):
  """
  This function is activated when a button in a
  two bot game is pressed. If the game isn't over,
  the bots will go through their turns until the
  game is over. Otherwise, the game will be reset.
  """

  # Checks that the game isn't over
  if (not(win_checker(currBoard)[0]) and not(tie_checker(currBoard))):
    while True: # Has a bot go through a turn
      if (currSymbol == "X"): # Bot 1's turn
        bot_turn("X", "O")
      else: # Bot 2's turn
        bot_turn("O", "X")

      # Stops the game if someone has won or there's a tie
      isGameOver = is_game_over(0)
      if (isGameOver != 0):
        break

      sleep(0.5)

  else: # If the game is over
    # Resets the game
    board_resetter()
    userPrompt.value = "Click any of the squares to have\nthe bots play against each other. \nOnce clicked, wait a littl
e for the \nbot's game to finish"
```

```python
def one_player_button(pos, currBoard, botIsFirst):
    """
    This function is activated when a button in a
    one player game is pressed. If the game isn't
    over (and the move is valid), their symbol will
    be placed. Otherwise, the game will be reset.
    """

    # Checks that the game isn't over
    if (not(win_checker(currBoard)[0]) and not(tie_checker(currBoard))):
        if (valid_location(currBoard, pos)): #Checks if move is valid
            currBoard[pos].text = currSymbol
            errorMessage.value = ""
        else: # Prompts user to pick a valid square
            errorMessage.value = "Error, please click on a square that isn't taken already"
            return

        # Setting text color
        game_board_colorer(pos)

        # Stops the game if someone has won or there's a tie
        isGameOver = is_game_over(1)
        if (isGameOver != 0):
            return

        # Lets the bot go through a turn
        flip_symbol(currSymbol)
        if (currSymbol == "O"):
            bot_turn("X", "O")
            isGameOver = is_game_over(1) # Stops game if it's over
            if (isGameOver != 0):
                return

    else: # If the game is over
        # Resets the game
        board_resetter()

        # Give the bot a starting turn if they're first
        if (botIsFirst):
            sleep(0.5)
            bot_turn("X", "O")

            # Colors in the symbol for the bot's turn
            for i in range(9):
                game_board_colorer(i)

            flip_symbol(currSymbol)

        userPrompt.value = "Click on an open square to place an 'X' there"

def two_players_button(pos, currBoard):
    """
    This function is activated when a button in a
```

```python
    two player game is pressed. If the game isn't
    over (and the move is valid), their symbol will
    be placed. Otherwise, the game will be reset.
    """

    # Checks that the game isn't over
    if (not(win_checker(currBoard)[0]) and not(tie_checker(currBoard))):
        if (valid_location(currBoard, pos)): #Checks if move is valid
            currBoard[pos].text = currSymbol
            errorMessage.value = ""
        else: # Prompts user to pick a valid square
            errorMessage.value = "Error, please click on a square that isn't taken already"
            return

        # Setting text color
        game_board_colorer(pos)

        # Stops the game if someone has won or there's a tie
        isGameOver = is_game_over(2)
        if (isGameOver != 0):
            return

        # Sets up for the next turn
        flip_symbol(currSymbol)
        userPrompt.value = "Click on an open square to place an '" + currSymbol + "' there"

    else: # If the game is over
        # Resets the gameBoard
        board_resetter()
        userPrompt.value = "Click on an open square to place an 'X' there"

def button_press(pos, currBoard, buttonPressFunc):
    """
    When one of the main board buttons is pressed,
    this function will 'send' the press over to the
    current function based on the game mode.
    """

    if (buttonPressFunc == "two_bots"): # 0 players (2 bots)
        two_bots_button(pos, currBoard) # Calls the 2 bot button
    elif (buttonPressFunc == "player_first"): # 1 player (player first)
        one_player_button(pos, currBoard, False) # Calls the one player button
    elif (buttonPressFunc == "bot_first"): # 1 player (bot first)
        one_player_button(pos, currBoard, True) # Calls the one player button
    else: # 2 players
        two_players_button(pos, currBoard) # Calls the two player button

def game_creator(startSymbol, buttonPressFunc):
    """
    This function will create the board and set up the screen
    for the actual game.
    """

    # Creates the button storage and current symbol
```

```python
  global currBoard, currSymbol
  currBoard = [" ", " ", " ", " ", " ", " ", " ", " ", " "]
  currSymbol = startSymbol

  # Board Creation
  global gameBoard
  gameBoard = gui.Box(app, layout="grid")

  # Button creation
  for pos in range(9):
    buttonText = currBoard[pos]
    x, y = num_to_grid(pos)
    button = gui.PushButton(gameBoard, command=button_press, args=[pos, currBoard, buttonPressFunc], text=butt
onText, grid=[x, y], width=3)

    currBoard[pos] = button

  # Instructional text
  global userPrompt, errorMessage
  userPrompt = gui.Text(app, color="white", text="Click on an open square to place your symbol there")
  errorMessage = gui.Text(app, color = "red", text="")

  if (buttonPressFunc == "bot_first"): # 1 player (bot first)
    bot_turn("X", "O") # Lets the bot have the first turn
    # Prompts the user to click a square
    userPrompt.value = "Click on an open square to place an 'X' there"
  elif (buttonPressFunc == "two_bots"): # 0 players (two bots)
    # Prompts the user to click a square to start the bot game
    userPrompt.value = "Click any of the squares to have\nthe bots play against each other. \nOnce clicked, wait a littl
e for the \nbot's game to finish"
  elif (buttonPressFunc == "player_first"): # 1 player (player first)
    # Prompts the suer to click a square
    userPrompt.value = "Click on an open square to place an 'X' there"

def reset_app():
  """
  This function will reset the app by making
  the board, user prompt, error message, reset
  button, and win message from the previous game
  after the reset button is pressed. It will then
  call main to let the user pick a new game mode.
  """

  # Makes various elements invisble
  gameBoard.visible = False
  userPrompt.visible = False
  errorMessage.visible = False
  appResetter.visible = False
  winMessage.visible = False
  main(False) # Lets the user pick a new game mode

def final_printer(currBoard, numOfPlayers):
  """
  Adds a message to the app based on if someone
```

```
  won, the winner, and the number of players.
  """

  global winMessage
  winMessage = gui.Text(app, text="", color="yellow")
  userPrompt.value = "Click any of the squares to restart\nthe game with the current settings"
  global appResetter
  appResetter = gui.PushButton(app, command=reset_app, text="Reset", width=5)
  appResetter.text_color = "red"

  # Checks if someone has won and if so who won
  anyoneWon, winner = win_checker(currBoard)
  if (anyoneWon):
    # Returns the win message
    if (winner == "X"): # Player / Player 1 / Bot 1 win return
      if (numOfPlayers == 1): # Player win vs Bot
        winMessage.value = ("You have triumphed over the bot!\nCongratulations on the victory!\n")
      elif (numOfPlayers == 2): # Player 1 win vs Player 2
        winMessage.value = ("Congratulations player 1 on your\nvictory against your opponent!\n")
      else: # Bot 1 win
        winMessage.value = ("Bot number 1 has won the epic\nbattle against the other bot!\n")
    elif (winner == "O"): # Bot / Player 2 / Bot 2 win return
      if (numOfPlayers == 1): # Bot win vs Player
        winMessage.value = ("The bot has bested you! Better\nluck next time.\n")
      elif (numOfPlayers == 2): # Player 2 win vs Player 1
        winMessage.value = ("Congratulations player 2 on your\nvictory against your opponent!\n")
      else: # Bot 2 win
        winMessage.value = ("Bot number 2 rules the day after\ndefeating bot 1!\n")
  else: # Retuns a draw message
    if (numOfPlayers == 1): # Draw in Player vs Bot
      winMessage.value = ("You have tied with the bot. Kind\nof anticlimactic. Hopefully you'll\nwin next time.\n")
    elif (numOfPlayers == 2): # Draw in Player vs Player
      winMessage.value = ("You have tied with each other. Kind\nof anticlimactic. Hopefully someone\nwill win next time.\n")
    else: # Draw in Bot vs Bot
      winMessage.value = ("The bots have tied with each other.\nKind of anticlimactic. Hopefully one will\nwin next time.\n")

main(True)
```