3. WRITTEN RESPONSES

3 a.

3.a.i.

The **purpose** of this Python program was to create a **fun tic tac toe game** that **users could play**.

3.a.ii.

The **functionality** of this program is that it **allows the user to play tic tac toe against a bot or other players** by clicking buttons on the screen.

3.a.iii.

All of the **input in this program works by clicking on buttons on the application window**. The **output of the program is the board shown on screen and various messages** in text. In the video, the user starts by clicking the 0 players option. This causes 2 bots to play a game and the result (bot 2's win) to be displayed. They then press a square to reset the board and click again to start a new round where bot 2 won again (shown by the board and win message). After clicking reset, they selected the 1 player going first option. The user takes their turn by clicking on squares where their symbol will then be placed. That being said, when they choose a taken square, an error message will appear. After beating the bot, the player clicked reset and started up a 1 player going second game. The user clicked until the bot won, then pressed reset. Afterwards, they selected two players, clicked some squares (some of which were invalid), and kept going until player 1 won (output by the board and message) before pressing the reset button and ending the video.

3 b.

3.b.i.

```
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
  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=buttonText, grid=[x, y], width=3)

    currBoard[pos] = button
```

3.b.ii.

```
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)
```

3.b.iii.

This program uses the list **named "currBoard"**.

3.b.iv.

The list “currBoard” stores the **9 buttons that make up the** tic tac toe **game board**. These **buttons also have their own** info, the most relevant of which is their **text**.

3.b.v.

Writing this program without the use of this “currBoard” list would be possible, but it would be much more difficult. This program works by creating 9 almost identical buttons (besides location) which are stored in the list. Since they couldn’t be stored in a list in this hypothetical, they’d need to all be unique. This means that I **couldn't use a loop to create the buttons** (since they’d all need different names), I **couldn't loop through each button** in the list easily (so I’d have to have 9 different statements go through each button), and I’d need **8 or 9 if statements if I wanted to only read from or edit one specific button**. This would make the program code longer, more difficult to write, and harder to scale.

3 c.

3.c.i.

```
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 little for the \nbot's game to finish"
```

3.c.ii.

```
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
```

3.c.iii.

The procedure in the above images, “two_bots_button”, takes a position on the board and the current board and **has two bots go through a whole game of tic tac toe** which is then displayed. This **contributes to the functionality of the program by allowing the buttons to operate (have the bots "play" or reset the board) in the 0 players (2 bots) game mode**. For clarity, the procedure has the argument “pos” which is unused. This is due to trying to keep the calls of the function more consistent with calls to the other button functions (which can be seen in the second code segment). The other parameter, “currBoard”, is, however, actually used.

3.c.iv.

This algorithm starts when one of the buttons on the board is pressed during the two bots game mode. It then uses functions as abstractions to check if someone has won (someone has three in a row) or there’s a tie (the board is full). From there, the function will repeatedly check which bot’s turn it is and have them go through their turn with a 0.5 second delay between turns. This loop will end if a bot wins the game or there’s a tie. If the game is over when the button is pressed, then the board will be cleared, the reset button and win message will become invisible, and the user prompt will change.

3 d.

3.d.i.

First call:

The first call for the procedure is **0 for “pos”** and a **list of 9 buttons all with text values of spaces (“ “) for “currBoard”**.

Second call:

The second call for the procedure is **8 for “pos”** and a **list of 9 buttons having the ordered text values of “X”, “X”, “O”, “O”, “O”, “ “, “O”, “X”, and “X” for “currBoard”**.

3 d.ii.

Condition(s) tested by first call:

When the procedure is first called with 0 and a list of buttons with spaces (“ “), the program will **check that there isn’t a tie and that no one has won**. Since the board is empty and no one has won, the loop inside will run.

Condition(s) tested by second call:

The second call will also **check if there’s a winner or a tie and find that someone (the “O” bot) has won** the game. Because of this, the aforementioned loop will not run and instead the board will be reset.

3.d.iii.

Results of the first call:

The result of the first call described above would be the **bots “playing” against each other until one wins or there’s a tie**.

Results of the second call:

The second call would cause the program to **fill “currBoard” with spaces (“ “) which would be shown by the board on the screen, make the reset button and win message invisible, and change the user prompt text.**