COSC 1336 Lab: 10

Relevant reading: Chapter 8, esp. 8.8

Due: Nov. 29, 2:30 pm

(Late date: Dec. 6, 2:30 pm)

50 Points

Problem 1. [**10 points**] For this problem, you will use several of the functions from the last problem on the last lab, in conjunction with one more new function to create a bar graph of the world series data. Download the files `draw_functions2.py`, `WorldSeriesWinners.txt`, and `world_series_graphics.py`, and do the following:

a. Copy and paste the following three functions from your solution to problem 2 on the last lab to the indicated spot in `world_series_graphics.py`:

- `sorted_copy_no_duplicates`
- `read_winning_teams`
- `get_wins`

If you didn't get these functions working on the previous lab, please get help during lab to get them working so you can continue.

b. Write a function called `get_data` which receives the list of winning teams for each year, along with the list of teams with duplicates removed. The function should return a list of tuples, where each tuple has a team name in the first position, and the number of wins the team has in the second position. That is, the function should return a list the beginning of which looks like this:

```
[('Anaheim Angels', 1), ('Arizona Diamonbacks', 1), ('Atlanta Braves', 1),...
```

c. Once you've completely debugged your `get_data` function, uncomment the three lines in the file that actually create the bar graph. If you've written the `get_data` function correctly, the bar graph should appear. You may find it interesting to look and and/or play around with the `draw_bar_graph` function. But once you get a bar graph displayed, you're done with the problem.

Problem 2. [**25 points**] Download the file `data.txt` and create a new Python file called `twod_list_count.py`. Then write a program that does the following. For this problem all the code may go in the `main` function unless you feel particularly inspired to make functions.

a. [**10 points**] The `data.txt` file contains 50 real-valued numbers. Read these values into a 10-row, 5-column two-dimensional list called `data`. That is, the first five values in the file should be the first row, values 6-10 should be the second row, etc.

b. [**15 points**] Read in a minimum and maximum value from the user. For each column of the two-dimensional list, count the number of elements in the column that are within the range given by the user, and display the count.

The following transcript shows how the program should run, and what the results should be for one test case:

```
Enter the minimum: 35
Enter the maximum: 65
Data in range for each column: 3 1 3 1 4
```

Problem 3. [**15 points**] For this problem, you will modify the Tic-Tac-Toe game that we wrote in lecture to allow the computer player to be a smarter opponent. Download the `tic_tac_toe.py` file for a version of the game that is structured basically the same as the one we wrote in class (it has all the same functions, though the *exact* implementation may vary because it was written before we wrote ours in class).

Before you try to make any modifications to the code, you should read through the program so far to be sure you understand everything, and ask questions if you find something that is confusing. In particular, you want to have a firm grasp of what the `calc_game_over` function does; not necessarily *how* it does it, but how you call it and what it does for you when you call it. The function you will be modifying is the `choose_computer_move` function, so you will want to have a solid understanding of it as well.

Right now the computer just chooses moves randomly until it finds one that is not already taken. Change the function so that the computer's strategy for choosing moves has the following properties.

- [**6 points**] If the computer can win, it always will.
- [**6 points**] If the computer can block the user from winning on the next move, it always will.
- [**3 points**] If the center is free, the computer will always go there.

On the next page is some high-level pseudocode for a smarter algorithm (I put it on the next page so you won't accidentally look at it if you want to work it out for yourself). By "high-level", I mean that each line of pseudocode may translate to more than one line of Python code, so you have some work to do to implement the algorithm.

For this lab, you should submit the following files:

- `world_series_graphics.py`
- `twod_list_count.py`
- `tic_tac_toe.py`

Here's some pseudocode to help you with the last problem.

```
choose_computer_move(board):
   chose_move = false

   # first, win if possible
   for each empty location in the board
      put an 'o' in the location
      if the move causes the computer to win
         stop searching (we have a winning move!)
      else
         remove the 'o' from the location
      end if
   end for

   # if we can't win, block our opponent from winning on the next move
   for each empty location in the board
      put an 'x' in the location
      if the move causes the computer to lose
         put an 'o' in the location, blocking that move
         stop searching (we have a blocking move!)
      else
         remove the 'x' from the location
      endif
   end for

   # if no blocking is necessary
   if chose_move is false and the center is free
      put an 'o' in the center
   endif

   if chose_move is false
      choose randomly (using original algorithm)
   endif
```