Computational Problem Solving CSCI-603
Lasers! Lab 5

6/2/2023

1 Requirements

You will implement a program, `lasers.py`, that can solve any *rectangular (N rows by M columns)* laser puzzle.

1.1 Command Line Arguments

The program should execute on the command line as:

```
$ python3 lasers.py filename
```

Where `filename` is the name of the text file containing the grid as one row per line, with single integers `0-9`, separated by spaces.

If the filename is not provided, or there are too many arguments, the program should display the following message to standard output and exit:

```
Usage: python3 lasers.py filename
```

If the filename is provided, it is guaranteed to be correctly formatted.

1.2 Program Execution

These are the steps the program should follow. Refer to the output section for the format of the messages.

1. Check the command line arguments and display/exit if the number of arguments is incorrect.
2. Read the file, save it to a collection, and display the puzzle.
3. Prompt the user for the number of optimal lasers to find. If the user asks for more lasers than can validly be placed, display an error message and exit.
4. Determine the optimal placement/s for the specified number of lasers and display them in descending order by largest sum.
5. Display the total sum of the optimal laser placement/s.

1.3 Output Specifications

All output should happen to *standard output*.

1. If the number of command line arguments is incorrect, display the following message and exit:

   ```
   Usage: python3 lasers.py filename
   ```

2. Otherwise, read the file and display the puzzle (one row per line, with one space between each digit).
3. Next, prompt the user for the number of lasers to place and read the value from standard input (denoted with curly braces below):

   ```
   Enter number of lasers: {value}
   ```

   Here, `{value}` is guaranteed to be a valid positive integer.
4. If the requested number of lasers is too many, display the following message and exit:

   ```
   Too many lasers to place!
   ```

5. Otherwise, display a header message for the optimal laser placements:

   ```
   Optimal placement:
   ```

6. Next, display the requested number of lasers, starting from the highest overall sum and descreasing to the next highest until the number of lasers is satisfied. *In the case of a tie, you can choose any laser with the same overall sum.* The lasers should be displayed one per line as:

   ```
   loc: (r,c), facing: F, sum: #
   ```

   (a) For the laser location, `r` is the row number and `c` is the column number. The top-leftmost cell in the grid is at coordinate `(0,0)`.
   (b) For the laser facing value, `F`, it should be a single character corresponding to the orientation:

Orientation	**Value**
North	N
East	E
South	S
West	W

   (c) The laser sum, `#`, should be a positive integer indicating the sum of the three cells the laser hits.
7. Finally, display the sum of all the lasers, where `#`, is a positive integer:

   ```
   Total: #
   ```

Refer to the sample runs on the next page for examples of all the output messages.

1.4 Sample Runs

There are six sample puzzle files in this data.zip file. Here are some sample runs using the puzzle file `puzzle3.txt`:

```
$ python3 lasers.py
Usage: python3 lasers filename

$ python3 lasers.py puzzle3.txt
Loaded: puzzle3.txt
0 3 7 9
2 5 1 4
3 3 2 1
4 6 8 4
Enter number of lasers: 4
Optimal placement:
loc: (1,2), facing: N, sum: 16
loc: (3,1), facing: N, sum: 15
loc: (2,1), facing: W, sum: 14
loc: (0,2), facing: S, sum: 13
Total Sum: 58

$ python3 lasers.py puzzle3.txt
Loaded: puzzle3.txt
0 3 7 9
2 5 1 4
3 3 2 1
4 6 8 4
Enter number of lasers: 13
Too many lasers to place!

$ python3 lasers.py puzzle3.txt
Loaded: puzzle3.txt
0 3 7 9
2 5 1 4
3 3 2 1
4 6 8 4
Enter number of lasers: 0
Total: 0
```

1.5 Implementation Details

You may find it useful to sort some items in your solution. To do this, please use and modify one of the sorts presented in lecture this week. Leave this in a separate file, making sure to include the proper authorship tags (but if you modify it, you should add yourselves as authors). You can then import this into the file with your main code. **The usage of the built-in Python sort is not allowed** but you can use it to verify your results when testing.

2 Testing

Be sure to test your program for the following things:

1. An incorrect number of command line arguments should result in an error message.
2. The `8 x 8` puzzle from problem solving should work for any valid number of lasers.
3. A rectangular puzzle (non-square) should work fine.
4. A puzzle that is too small to place any lasers, e.g. `2 x 2`, should not allow any to be placed, but should only error if more than 0 lasers are requested.
5. A request to place more lasers than the puzzle can hold should result in an error message.

3 Grading

Your grade will be determined as follows:

- 20%: results of problem solving
- 5%: Design
- 65%: Functionality
  - 5%: Proper selection and use of data structures
  - 10%: Error handling
  - 25%: Proper generation of all potential laser placements
  - 25%: Proper selection of optimal laser placements, including avoiding placements at identical locations
- 10%: Code Style and Documentation

4 Submission

Create a ZIP file named **lab5.zip** that contains all your source code. Submit the ZIP file to the MyCourses assignment before the due date (if you submit another format, such as 7-Zip, WinRAR, or Tar, you will not receive credit for this lab).