EN.540.635 "Software Carpentry"

Lazor Group Project

Due on November 4 at 4:30 PM - To be Submitted through Github (no changes past deadline!)

For this assignment, you will write a program that will automatically find solutions to the "Lazors" game on iPhone and Steam. The game is no longer listed on Android, unfortunately, but you can also find play-throughs on YouTube and such. Work will be done in groups (max of 3 people per group), and as such the code should be collaborated on via GitHub. **The final project submission on the shared Form should be a link to the Github repo with the full name of everyone in your group. Please also submit the Github URL through Canvas. Please tag the final version of your code and create a release that we will download and use for grading. By Nov 4, please submit a brief peer review for each member in your group. Instructions for the peer review are on Canvas.**

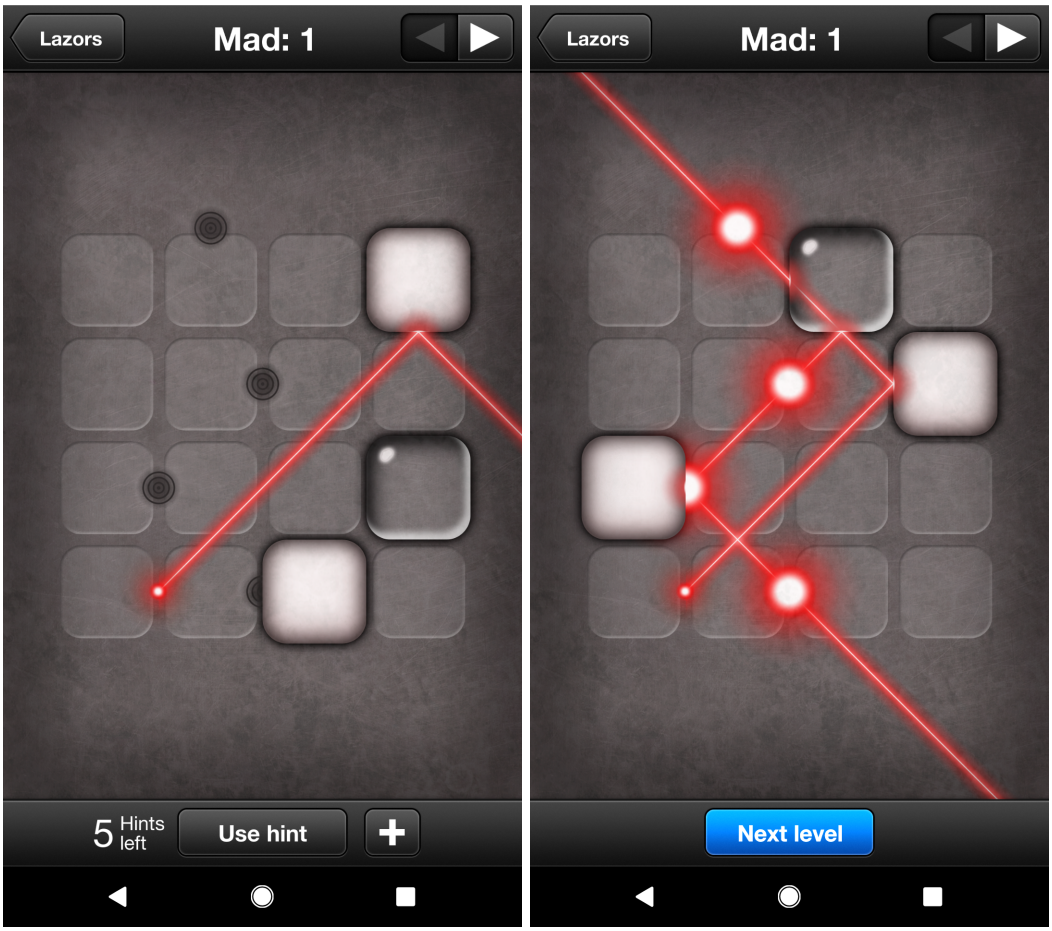

Figure 1: An example of a level in Lazors (Mad 1).

You will be given free reign in how to tackle this problem; however, some criteria must be met:

1. You must read in the Lazor board via a text file with a specific format (.bff). The specified file format will be further elaborated below.
2. You must use a class object to describe the blocks within the game.
3. You must have an output file that shows the valid solution (text file, png image, etc). Output file format is up to you; however, it must be easy to understand.

4. The code should NOT be slow. We have included various boards to use for benchmarking. We will be using these (and possibly others) to test out your code. Try to make it such that all the test boards are solved at a reasonable pace. None of them should take more than 1 minute (some should be under 5 seconds); however, your code has 2 minutes to solve each individual board. If your solution of any single board takes more than 2 minutes, points will be deducted. If your solution is absurdly fast for all boards, we may give bonus points!
5. Your boards should allow for three types of blocks: (a) reflect blocks, (b) opaque blocks, and (c) refract blocks. Further, you should also account for boards in which a block is fixed in a starting position.

Board File Format (.bff)

```
# This is a comment
# This example is for mad 1 in Lazor
#   x = no block allowed
#   o = blocks allowed
#   A = fixed reflect block
#   B = fixed opaque block
#   C = fixed refract block

# Grid will start at top left being 0, 0
# Step size is by half blocks
# Thus, this leads to even numbers indicating
# the rows/columns between blocks, and odd numbers
# intersecting blocks.

GRID START
o  o  o  o
o  o  o  o
o  o  o  o
o  o  o  o
GRID STOP

# Here we specify that we have 2 reflect blocks and 1 refract block
A 2
C 1

# Now we specify that we have two lasers
#    x, y, vx, vy
# NOTE! because 0, 0 is the top left, our axis
# are as follows:
#
#       ----------\ +x
#       |          /
#       |
#       |
#       |
#      \|/ +y
#
L 2 7 1 -1

# Here we have the points that we need the lazers to intersect
P 3 0
P 4 3
P 2 5
```

P 4 7

Points Considered During Grading

GitHub

- Was it used appropriately?
- Does the code exist online for us to access?
- Is there a README explaining how to use the program?

Reading in BFF Files

- Are the .bff files read in correctly?
- Is it robust?
- Does it work without any errors or requirements?

Use of Class Objects

- Were classes used appropriately or were they simply jammed into the code?

Blocks Accounted For

- Were all blocks (reflect, opaque, and refract) accounted for?
- Were fixed blocks of the above types accounted for?

Solution

- Is the solution correct?
- Did it take too long to solve the boards?
- Is the solution output in a file?
- Is the solution easy to parse back to a board (is it easy to understand)?

Misc

- Are docstrings used?
- Is PEP8 styling used?
- Is the code commented appropriately?
- Were unit tests written?
- Are errors handled appropriately?

Optional Challenge, Possibility of Bonus Points

If you've played around with this solver for a little bit, you'll quickly come to the realization that the larger the boardgame, the longer the solution takes. This can be narrowed down to two culprits:

1. The generation of potential boards
2. The play through of each board

There are many ways of speeding up both of these sections. This completely optional challenge is to figure out how to go about speeding up this code. NOTE! If you do find some way of speeding this up, and are able to adequately explain how it works (don't just copy things from online...), then you will also get bonus points on this assignment. **Keep in mind though, if your base code is slow due to the algorithm/approach you took, and you speed it up using some method like parallelization, there may still be point reductions (and no bonus points given). So make sure you focus on the base assignment first before taking on this optional challenge!**

Group Contribution

We expect that you properly use Github to work on this project in collaboration with your group members - the work should be divided as evenly as possible between group members, and every group member should be making commits to the remote repository where all your code is located on Github. **If there are any issues regarding your team members, or should you fall ill during this time, please contact us as soon as possible so we can properly address the issue.**