Lights Out

1 Overview

Lights out is a simple button based game in which lights are toggled on and off by pressing on them. Pressing a light toggles it and the four lights around it. The object of the game is to take a scrambled board back to being all off. In this lab you will be writing a model for the lights out in which you will use the better-to-ask-forgiveness-than-permission to avoid boundary checks. In addition you will be building a loader that is capable of parsing a game state from a file, and throw exceptions when the the task is not possible.

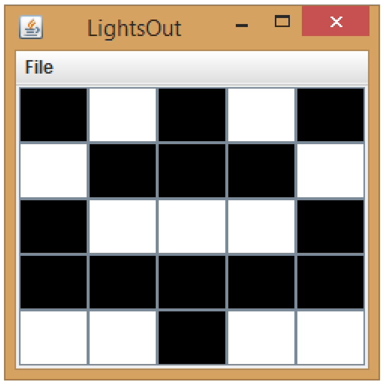

Fig. 1: Lights Out being played with a Swing application.

2 Learning Outcomes

By the end of this project students should be able to:

- write programs using try-catch statements;
- write custom exception classes;
- throw exceptions;
- work effectively with a partner using pair-programming;
- write an effective report that describes the students' problem solving process.

3 Lab Instructions

Do this part in lab:

LightsOut
LightsOut(size : int) getSize( ) : int isLit(row : int, col : int) : boolean forceLit(row : int, col : int, value: boolean) : void press(row : int, col : int) : void

LightsOutFileLoader
load(game : LightsOut, inputfile : File) : void

UnsupportedLightsOutFileException

Tab. 1: UML Class Diagram

Step 1

In this lab we will create classes following the following class specifications:

Start by implementing class with stubs for each of the methods, either doing nothing or returning a placeholder value. Make sure your placeholder value for getSize() is greather than zero! Once you have stubs for all of the methods, copy the LightsOutPlayer.class file and the two supporting class files provided into the same folder and then running the LightsOutPlayer class. If your LightsOut object worked it should bring up a window. If it crashes, you are likely not correctly implementing the contract. Ensure you can open the player before moving on to step 2.

Step 2

Now that we know we have the proper public contract, we will provide the class's implementation. First decide what internal state the LightsOut class will require and record your thought process for the planning stage of your report. Now, implement each method.

LightsOut (size : int)

For our basic constructor initialize your state to represent an empty lights out puzzle. The board should be square with the size representing the number of lights along each axis. Use the menu bar to make sure different sizes work for your game.

getSize ( ) : int

Returns the size of the board as initialized in the constructor.

isLit (row : int, col : int) : boolean

Should take a row and column number and return true if that spot is currently lit.

forceLit (row : int, col : int, value : boolean) : void

Takes a row, column and boolean value, and sets the puzzle to store value at the given location.

press (row : int, col : int) : void

Simulates a press of a light in the game. This toggles the location specified as well as the four surrounding lights. **REQUIREMENT: Do not check boundaries before setting lights! Instead practice easier-to-ask-forgiveness-than-permission using exception handling!**

load (game : LightsOut, inputfile : File) : void

This function accepts an existing lights out game and forces its lights to match the configuration of a specific file. In our file format an X will represent an off light, and an underscore will represent an on light. An example file is provided that shows lights on and off in a checkerboard. **Should you encounter an invalid character in the file (something other than an underscore, X or character return) you should raise an exception using our custom UnsupportedLightsOutFileException**. Use the menu bar to make sure you can load a configuration file.

```
X_X_X
_X_X_
X_X_X
_X_X_
X_X_X
```

Once you have completed these tasks, you should be able to run your code with the LightsOutPlayer and play a complete game of light outs, as well as load an existing lights out configuration from a file.

4 Lab Report

Each pair of students will write a single lab report together and each student will turn in that same lab report on BBLearn. Submissions from each student on a pair should be identical. In order to recieve points, you MUST make a BBLearn submission.

Your lab report should begin with a preamble that contains:

- The lab assignment number and all partner names
- Your name(s)

- The date
- The lab section

It should then be followed by four numbered sections:

1. Problem Statement

In this section you should describe the problem in **your** own words. The problem statement should answer questions like:

- What are the important features of the problem?
- What are the problem requirements?

This section should also include a reasonably complete list of requirements in the assignment. Following your description of the problem, include a bulleted list of specific features to implement. If there are any specific funtions, classes or numeric requirements given to you, they should be represented in this bulleted list. It is recomended that you complete this section before you begin coding the problem, as gathering these requirements may help you organize your thoughts before you begin your soulation.

2. Planning

In the second section you should describe what planning you did in order to solve the problem. You should include planning artifacts like sketches, diagrams, or pseudocode you may have used. You should also describe your planning process. List the specific data structures or techniques you plan on using, and why. How do you plan on breaking up the problems into functions, classes and methods?

3. Implementation and Testing

In the third section you should demonstrate your implementation. As directed by the lab instructor you should (as appropriate) include:

- a copy of your source code (Submitted in BBLearn as .java files, should not be in your report document)
- a screen shot of your running application / solution. This should include a screenshot of the output, be it a terminal window or graphical interface
- results from testing. This is specific to the lab, and not every lab will include this element.

4. Reflection

In the last section you should reflect on the project. What part of the projec was the most difficult? Where there other solutions to this problem that you considered? Where there any new solutions you can think of now that you couldn't then? Do you think the way you chose was the best option, or would you go about the problem differently if you had to start over? How might the problem had been broken up into easier problems? If you had one more day to work this problem, what improvements might you make? Every problem has alternative solutions and solution has tradeoffs or improvements, you are required to identify some of these elements.

5. Partner Rating

Every assignment you are required to rate your partner with a score -1, 0 or +1. This should be submitted in the comment section of the BBLearn submission, and not in the report document. If you don't want to give your partner a negative rating making sure not to use a dash before listing the number! You do not have to tell your partner the rating you assign them. A rating of 1 indicates that your partner was particularly helpful or contributed exceptional effort. A rating of 0 indcates that your partner met the class expectations of them. Rating your partner at -1 means that they refused contribute to the project, failed to put in a resonable effort or actively blocked you from participating. If a student recieves three ratings of -1 they must attend a mandatory meeting with the instructor to dicuss the situation, and recieving additional -1 ratings beyond that, the student risks losing a letter grade, or even failing the course.

Colophon

This project was developed by Richard Lester and Dr. James Dean Palmer of Northern Arizona University. Except as otherwise noted, the content of this document is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License.