1 Introduction

In this Project, you’ll be implementing two programs. Although they look very different, the first is designed to help you implement the second.

The first program is named LightGrid. In this program, you will have a 4x4 array of double, which you will draw as a grid with StdDraw. The program will continually watch for the location of the mouse; where the mouse is, it will set that element of the array to 255. All of the rest of the elements of the array will exponentially decline - at a rate of 1% every 10 milliseconds.

The picture below shows LightGrid as it runs. The 255 shows the current location of the mouse (sorry, my software didn’t include the mouse in the picture). The large values represent places where the mouse has been recently; smaller values represent places that have been declining for longer.

The second program is named TilePuzzle. In this program, you’ll implement a sliding puzzle game (https://en.wikipedia.org/wiki/15_puzzle). In this game, you have 15 tiles (in grid of sixteen), with one hole. Each time
that the player clicks on a tile, your program will try to move it into the hole. The player tries to re-arrange the tiles of the puzzle so that the numbers are in order.

This program will look roughly like this:

```
+-----+-----+-----+-----+
| 1   | 7   | 4   | 13  |
+-----+-----+-----+-----+
| 15  | 6   | 14  |     |
+-----+-----+-----+-----+
| 2   | 9   | 5   | 3   |
+-----+-----+-----+-----+
| 12  | 11  | 10  | 8   |
+-----+-----+-----+-----+
```

2 LightGrid Requirements

The point of the LightGrid program is to give you a little more experience with watching the mouse. In this program, we’ll have large tiles (instead of small squares), and it will be important

Expect that your first version of this program will have some problems with getting the picture to line up with the mouse. Common problems will include having the lower row of tiles (and the left column) be halfway off of the screen; having tiles overlap each other; and ArrayIndexOutOfBoundsException exceptions when you try to use the input from the mouse. That’s the whole point of this program: to get some experience with fixing those bugs, before you write the (somewhat) more complex tile puzzle.

The LightGrid program will be worth 40% of your total grade for this project.

2.1 Pseudocode

Pseudocode is not required for the LightGrid program.

2.2 The Program Itself

Your program must:
• Allocate a 2D array of double to store the values.

• Every 10 milliseconds, update the grid and redraw it:
  – If the mouse is over any square on the grid, set that square to 255.
  – Decrease the value of all other grid squares by 1%.
  – Redraw the grid from scratch.

• Each time that you draw the grid, use StdDraw.setPenColor(int,int,int) to set the color for each square.
  I recommend 0 for red and green, with blue set to the current value of that square - but if you come up with another system, that’s OK. However, it must change gradually from one color (when the value is zero) to the other extreme (when the value is 255).

• Each time that you draw the grid, also draw the current value of the grid square (rounded down to an integer) in the middle of the square on screen. Use the method StdDraw.text(double,double,String) to do this.
  (Make sure that you draw the text using a color which will show up well - no matter what color the square is.)

  Of course, make sure that your program never crashes with any exceptions; it should run forever, no matter what you do with the mouse.

2.3 Drawing Strategy

Programs that have to draw the screen over and over again are common. Almost always, they use a “game loop,” which basically performs the following operations (in this order):

• Check the user interface (in this case, the mouse position)

• Update the state of the game

• Clear the screen

• Redraw the screen

• Call StdDraw.show(int) to update the picture on the screen.
  (If you’re not using StdDraw, then the way that you push the picture to the screen maybe different - but the concept will be the same!)

• Wait for a few milliseconds (in our programs, StdDraw.show(int) does this for us).

  I strongly encourage you to use this style in both programs in this Project (and in all your future graphics-related projects).
3 TilePuzzle Requirements

TilePuzzle is the main program of this Project. Things you’ve learned in LightGrid will make it easier to keep the mouse and the tiles lined up in this program.

The heart of this program is an infinite loop, which checks for mouse clicks, and changes the board if necessary. The exact logic for how you respond to a mouse click will take some thought: you need to see if the mouse is being clicked, and then determine if it is being clicked somewhere on one of the 16 tiles. But even then - if you find that it has been clicked - you will have to implement logic to figure out how to slide the tiles around.

The TilePuzzle program (including its pseudocode) will be worth 60% of your total grade for this project.

NOTE: You do not have to check to see if the user has “won.” Let the user play forever - as long as they want.

3.1 How the Tiles Slide

Consider the following picture of the tile puzzle:

There are several places that you might click the puzzle. If you click on tiles 13, 14, or 3, the tile should slide to fill the hole. This is the simple case - and if you implement only this, you will be most of the points for this assignment.

However, if you click further away from the hole, you could slide multiple tiles. For instance, if you click on 15, then the tiles 14, 6, and 15 should all move. (You can skip this feature and still get most of the points for the assignment.)

Of course, there are lots of places where you might click, and nothing happens. If you click on the hole, nothing will move. Likewise, if you click on the
tiles 5, 11, or 1 (and many others), then nothing will move - because they are not in a straight line away from the hole.

3.2 Pseudocode

As with Project 7, you only have to write pseudocode for the most interesting parts of your program. If you want, these may be separate methods - or they can just be parts of a larger method (such as main()).

Write pseudocode for the following portions of your program:

- The code to fill the puzzle with the 15 tiles, in random positions
- The code to determine whether or not the user has clicked on a tile - and then to update the 2D array if it has.

Spend some time on your pseudocode. Remember, the point of pseudocode is not to come up with a perfect algorithm - so it's OK if it doesn't match your Java code. What matters is that you demonstrate that you took some time to think it all through. Are you considering the inputs and the outputs? Do you have a rough idea of the loops that will be required? Etc.

You will be graded on the quality of your pseudocode. Again, we're not looking for perfect algorithms - but evidence of serious thought about them.

3.3 The Program

To receive most (but not all) of the points for this assignment, your program must:

- Allocate a 2D array to represent the state of the puzzle. Designate some particular value, which you store into the array, to represent the hole.

- Fill in the 2D array with 15 tiles, numbered 1 though 15. Arrange them randomly. Make sure that no number shows up twice; also make sure that no number is missing.
  
  **Hint:** Try randomly placing the number 1. Then try randomly placing the number 2 - how would you know if it overlapped the tile already in place? How would you fix this problem? Then repeat this process, for all of the rest of the numbers.

- Draw the board, over and over (for partial credit, it is OK if you redraw the board even if nothing changes). Make sure that each tile has its number written on it. You are encouraged to make it look like my example - but if you want to experiment with ways to make it better, that's OK.

- When the user clicks on one of the tiles, check to see if the tile is next to the hole. If it is, then slide the tile over, into the hole. Note that you must implement sliding in all 4 directions - but be careful to avoid ArrayIndexOutOfBoundsException.
If no move is possible (including the case where the user clicks on the hole itself), then simply ignore the mouse click and move on; you don’t have to give any message or notification.

For partial credit, it is OK to only slide single tiles - you never need to shift an entire line, all at once.

To receive full credit, your program must additionally do the following:

- Only redraw the screen when the board actually changes. (Until that happens, keep checking StdDraw.mouseClicked(), but don’t redraw the screen.)

- When the user clicks on a tile, slide multiple tiles (all in a row or column) if possible. For instance, in the example picture above, if the user clicked on 15, then the tiles 14, 6, and 15 should all move. However, if the user clicks on 5, then nothing should happen.

4 Implementation

You may, if you prefer, implement each of these programs in the main() method of their respective classes.

However, I think that it will be easier to write (and test, and debug, and read) if you use methods. For instance, my solution to each program includes a method which draws the screen - it takes the current 2D array as a parameter. You might also want to write a method which adds one tile to the TilePuzzle - since that logic is a little bit tricky. Likewise, the logic to implement sliding tiles around probably would work best in its own method.

It’s up to you whether or not you make use of methods in your program.

5 Pseudocode

Download the latest pseudocode requirements from:
http://www.cs.arizona.edu/classes/cs127a/spring16/projects/
Project 8 will be graded based on v1.3 of the requirements.

6 Java code

After you complete the pseudocode (and turn it in!), start working on translating it to Java. (For a list of steps for converting the pseudocode to Java, see the Project 1 spec.)

Your classes must have exactly the names given above: LightGrid and TilePuzzle. Make sure that your Java files are named to match.
6.1 Header Comment

Every Java file that you submit must have a Header Comment - which is a simple comment, at the top of your file, which gives basic information about it. A header comment must include:

- The name of the Java class
- The name of this class ("CSc 127A Spring 16")
- The assignment name
- Your name
- Your Section Leader’s name (or letter)
- A quick description of the Java class.

Here’s my suggested header comment for Program 6:

```java
/* class ScanString
 *
 * CSc 127A Spring 16, Project 08
 *
 * Author: TODO: your name here
 * SL Name: TODO: the name of your SL here
 *
 * ---
 *
 * TODO: put a quick (2-3 sentence) description of the class here
 */
```

Late Day Clarifications

Pseudocode

The syllabus says that you can use Late Days for late projects - however, the syllabus does not explicitly say how this works with pseudocode. The SLs and I talked it over, and we decided on the following policy:

- Late days cannot be spent on pseudocode. Turn it in on time, or you’ll just miss that part of the project grade.
- You must turn in your pseudocode using `turnin`; email will no longer be accepted.

(We were flexible about both of these points in the first two projects - but starting with Project 3, we’re going to enforce this.)
Java code

Starting with Project 3, if you email us your code (instead of using turnin),
it will count as a Late Day, even if you do it before the deadline. Please use
turnin. And remember: if you are having trouble connecting from home, it’s
always possible to come to the 228 lab, and upload your file using a thumbdrive.

7 Turning in Your Programs

See the class website for information about how to upload your files to lectura,
and how to use the ‘turnin’ utility.

7.1 Pseudocode

Your pseudocode file must be either TXT or PDF files (one per program), but
you may name them whatever you want. You must turn them in using the
assignment name

    cs127a_s16_sX_proj08_pseudocode

(replace the X with your section letter). REMEMBER: The pseudocode is
due two days before the Java code!

7.2 Java Code

You must turn in file(s) named exactly this:

    LightGrid.java
    TilePuzzle.java

You must turn in the Java file(s) using the assignment name

    cs127a_s16_sX_proj08

(replace the X with your section letter).