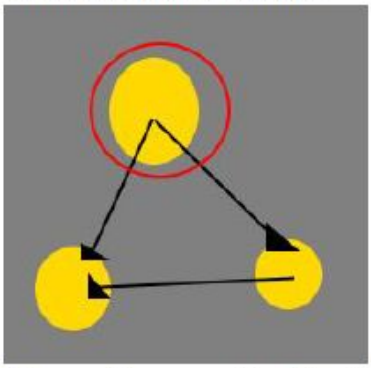
Etude 9: Lights With GUI

Introduction:

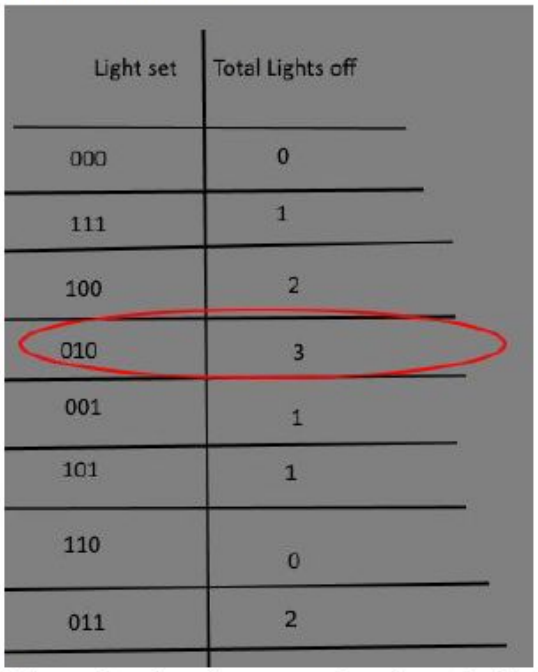
The Lights with GUI project required a program that could take 2 line of parameters, a set of letters and a set of connections. Each letter represented a light, and each connection represented an 'edge' between the two lights given. If the first light wa pressed, the second light would also switch its on state. This had to be represented by an interactive visualization of the lights and their relevant connections.

Analytics of the problem:

An important aspect of the problem was the 'state' of the lights. This was quickly established to be best represented by an array of booleans (True-on, false-off).
Originally , on paper we decided the best solution would be one that iterate through each light, seeing how many lights would turn off if it was switched. The program would iterate through, pressing these, skipping over a light if switching it resulted in the same light state array as 2 'switches' ago, preventing loops.



However, this appeared to not be able to deal with solving a test halfway through. If a light with lots of connections was off, the program would press it on to turn the other lights off. The program would be unable to find a way to switch a third 'on' light to toggle the first after this however, as sometimes the best solution did not involve flicking on the maximum connection light first.
After establishing that each light could only have 2 things occur in between an unsolved and solved state- that being ;switched' or 'not switched' , and order did not matter, we realized we could bruteforce the problem by attempting every combination of 'lights' and testing which had the largest ''off lights' count, then iterate through that list of switches.



Light set	Total Lights off
000	0
111	1
100	2
010	3
001	1
101	1
110	0
011	2

This was done by calling a recursive solve method that took a local boolean array- switches, that for each loop would have the value at index 'depth' false if the light in the light array at 'depth' was looped through without switching or 'true' if it was looped through as with that switch pressed. Once the break case of the final loop was reached, the count of all lights off would be compared against a global 'bestcount'. If best count was less than this new value, the final switch array would replace a global switch array.
Once all iterations of the solve array had run, the global switch array could be iterated through, switching the lights in the light array at any index in the best switch array that was 'true'.

GUI Implementation:

The GUI uses 2 primary sections in a 'JFrame' component. The first, on the left hand side, is a selection of Components such as JButtons and JTextAreas. Text sections were made scrollable where possible to prevent oversaturation of the text areas. The second section is a DrawPanel- an extension of the JPanel class. This handles the drawing of the light array, by taking values from the left hand side that are assigned the the DrawPanel object when the 'fill' or 'solve; buttons are pressed

External Resources Used

https://www.quora.com/How-do-you-add-a-MenuBar-in-a-JPanel