Status Summary - Algorithm Visualizer

Sam Goto & Michael Metz

There are 2 parts in this project; Path finding algorithm visualizer and sorting algorithm visualizer.

Sorting algorithm visualizer:

A lot of implementations have been done, including; **input validation, bars rendering, animations options, configuration persistence in local storage, sorting animations, Merge sort, Radix sort, Selection sort,** and some **minor options for rendering**.

Also, many issues have been resolved as well.

The major issues were:

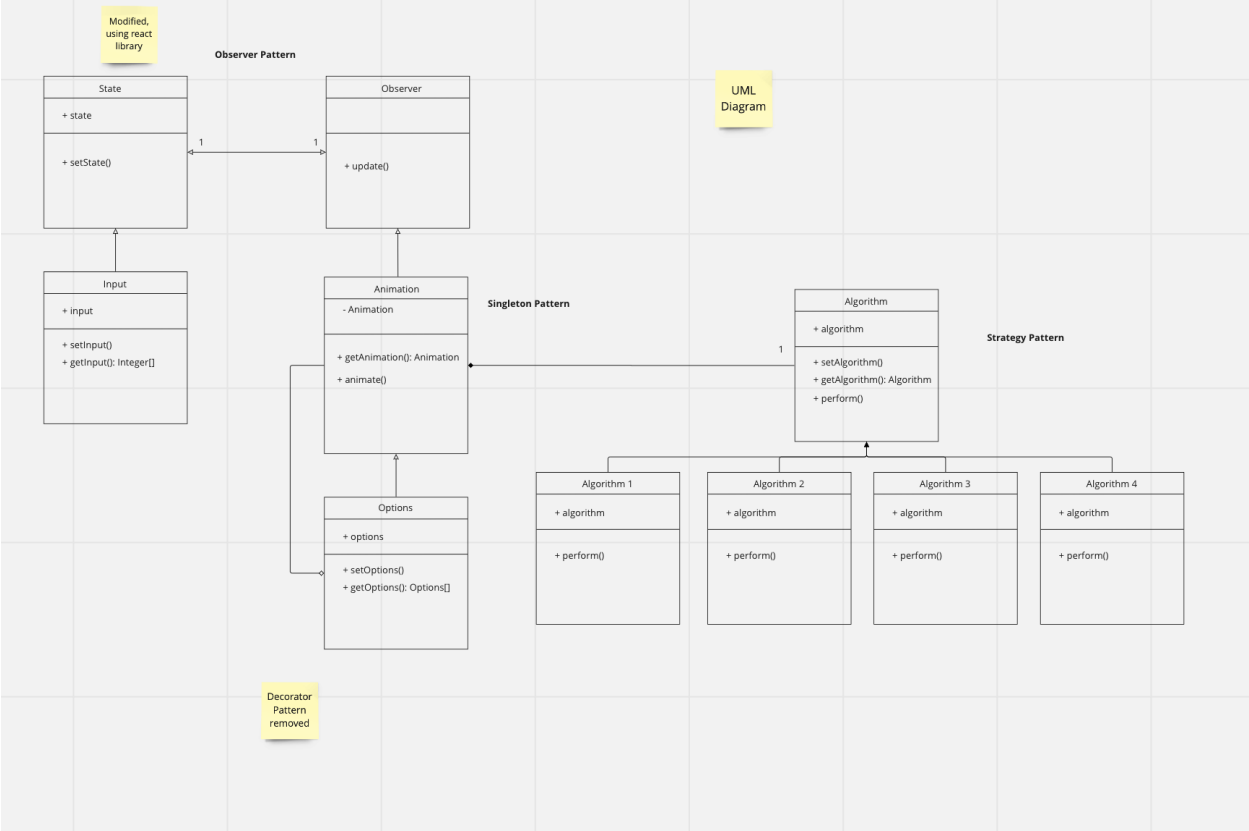
- Calculating the width and height of bars maximizing the width and height
- Rendering issues not returning Promises and not animating correctly,
- Animation issues
- Bars(div elements representing values in array) need to be swapped at the right time with the right bars, and have to be colored with the right color configured by the user. And all animation needs to happen in the delay that the user has configured.
- Sorting algorithm issues
- Some algorithms are harder to animate. For instance, merge sort was a bit tricky to animate, because the initial implementation of the algorithm had no record of which value was swapped with which, and it was entirely a recursive function so it was very hard to animate avoiding infinite loops, wrong swapping, etc.
- Bars value issues
- There is an option in the program to show the value (height) of each bar, but again, the font size needed to be dynamic as the width of the bar could vary, and had to make sure that the bar is high enough to hold the text. Also, because we need to swap the DOM elements during the process of animations, what I ended up doing was to use pseudo elements and inject the height value there conditionally or hide them depending on the condition Currently, it is set to show the values when the bar width > 25px and there is a 8 px padding between the bar and the text.
- Design

- Especially, making everything responsive was a tricky part, as screen size is a key to this project as the size of the bars are all based on the screen size. However, I like designing UI / UX so it's been really fun and I have been learning new things as well.

So far, we have used Singleton pattern to make sure that only one instance of Animation class is created, as the Animation class will have access to the DOM tree to animate things, so it is important that only one instance is created and has access to the DOM tree to avoid messing up the DOM. We also have used the Observer Pattern, to detect various changes in the program, we used React state from the react library to achieve this. Also Strategy pattern has been used to assign different algorithms to animate.

Class Diagram

We have used 3 of the patterns that we had in the initial UML diagram from the project proposal, and removed one of them, which is the Decorator pattern. Initially, we were planning on using Decorator to configure different options for the animation, but it did not fit well in the whole design of the system, so we decided to remove that. Other than that, we have been following the initial plan and the diagrams for the whole system. During the next week, we'll be incorporating one or more design patterns in our system, but we have not decided what pattern so that's something we could work on.



Plan for next Iteration:

We plan to implement a few more algorithms for our sorting visualizer, and resolve minor UI bugs, and animation bugs including not being able to stop the animation once it starts animating (happening in Selection sort).

Aims:

1. Game-Search Algorithms

    A. Search Algorithms:
        i. Linear, breadth first, Greedy, Binary
        ii. Alpha-Pruning: We aim to create this demonstration in the form of a basic board game in which players represented by a type of algorithm compete. Therefore showing how certain, more in depth, algorithms offer a clear advantage in terms of accuracy over the other.

    B. Recursive Algorithms:
        i. Mini-Max: This algorithm we hope to demonstrate through a maze running simulation.

2. UX & Performance :
    - Use faster algorithms to avoid a potential crash when user input is large
    - Provide smooth animations
    - Provide options to change how fast the algorithm should perform.

3. Streamlining the design:

    A. There are a few avenues of approaching this goal to better format the overall design in order to offer a more coherent system of code for any developer that may evaluate or reference the project.

        i. There is a template for site navigation demonstrated on the official React documentation in which the observer pattern is clearly utilized.

        ii. Each algorithm, both search, and sort, can be reduced in accordance with the Strategy pattern of design. Therefore allowing all examples to be displayed on a single page based on the user's selection.