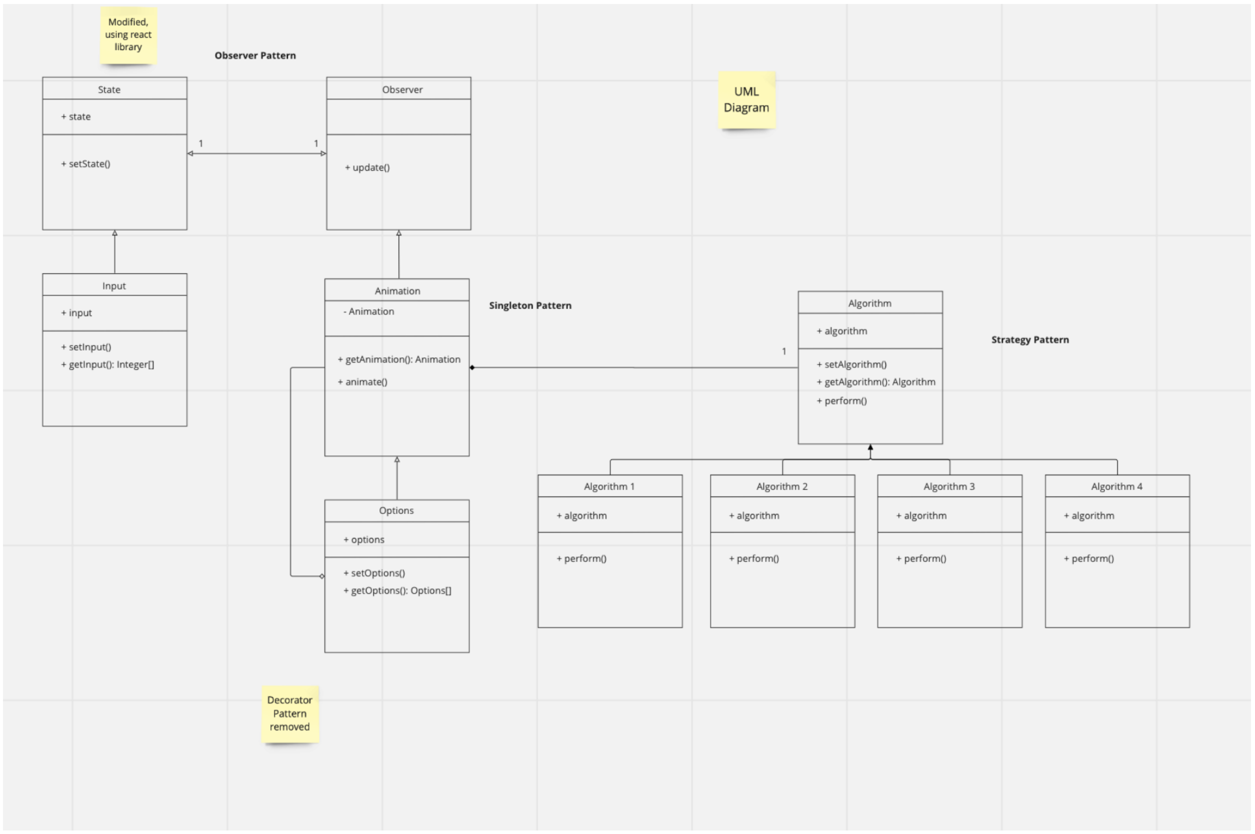
Algorithm Visualizer

Sam Goto & Michael Metz

We have been working on the project following the initial plans / ideas in project 5 - 6 so there have not been any major changes.
The sorting algorithm visualizer has features like: input validation, random array generation, bars rendering, customizable options, option persistence in local storage, animation rendering, and different sorting algorithms, have been implemented. The only thing that has changed is that we decided not to use the decorator pattern to apply different options. Instead, we used the web browser's local storage to apply different options for animations and to make it persistent.

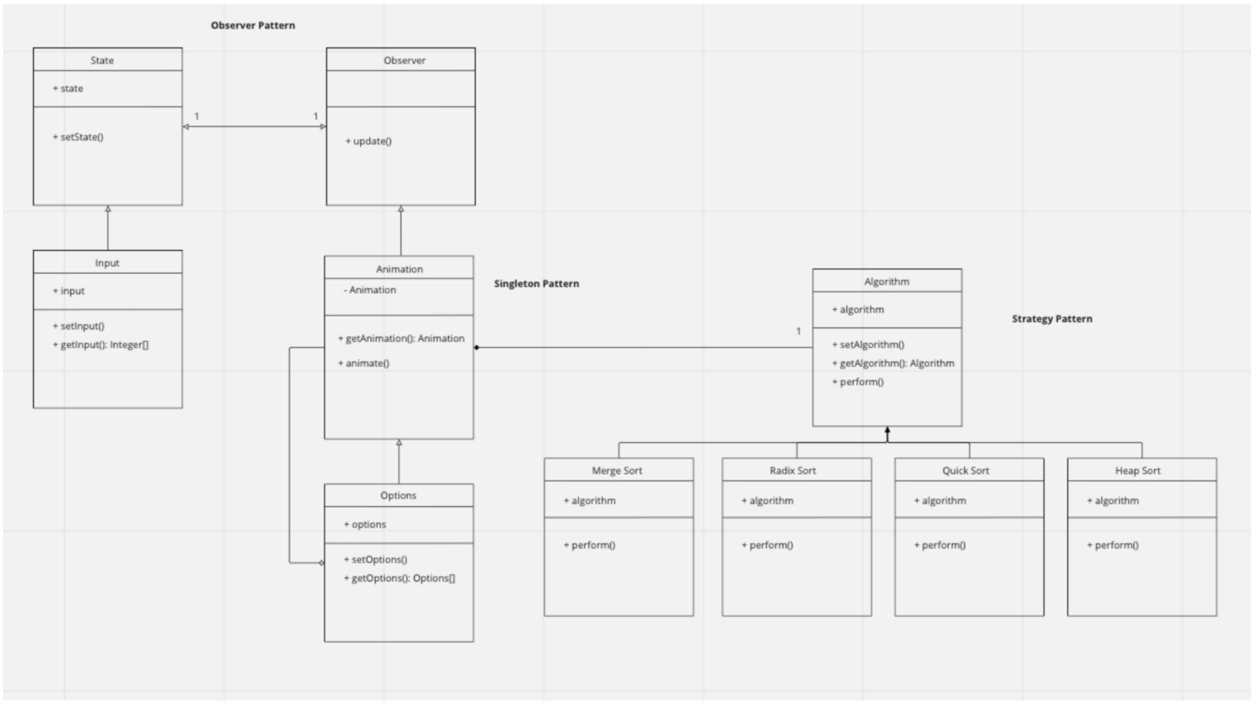
UML Diagrams

A diagram from project 6, where we had removed the decorator pattern and we just had arbitrary names for the algorithms.



Updated UML Diagram

As mentioned earlier, we have not had any major changes since the previous project assignments, so there are no major changes in UML diagram either, but this time we have implemented actual algorithms so we have added concrete names and still have the decorator pattern removed (details above).



Third party Code vs Original Code

*Since most of our code is original, we'll only specify which framework / code is **NOT** original.

Sorting algorithm visualizer:

Third Party Code / Frameworks:

- React template from npm
- Tailwind css template from npm
- Random array generation code from stackoverflow (which has been removed in code and it's been commented so it's not present in production)

Statement on the OOAD process for your overall Semester Project

- Strategy Pattern worked out really well for our project. We used it to assign different algorithms to the animation phase.
- Use of singleton pattern. We initially did not use it and we had different instances of Animation class, which has access to the DOM tree and is responsible for modifying the tree elements, but the issue we faced was that multiple instances of the Animation class were basically performing the same operations to the DOM tree and causing a lot of issues in animations. So we implemented the singleton pattern in Animation class, which resolved the issue.
- One issue with the use of the Observer pattern was when rendering updated data that the observer has received. I came to the conclusion that it also has something to do with the react framework and the way its rendering process works, but we had an issue where it goes into an infinite rendering loop and the web browser crashes especially during the process of the animations. So what we ended up doing was instead of rendering a new page every time there is an update, we decided to render the page conditionally when there are important changes in the UI components, which resolved the issue.