CSE 331: Software Design & Engineering Fall 2024

Homework 2

Due: Saturday, October 12th, 11pm

As in Problem Set 1, a key part of this assignment is practicing debugging, this time in a more complex application. Task 4 asks you to submit a log describing all the time spent debugging and the nature and causes of the bugs. While you submit your log in Task 4, the debugging itself will take place while working on Tasks 1–3.

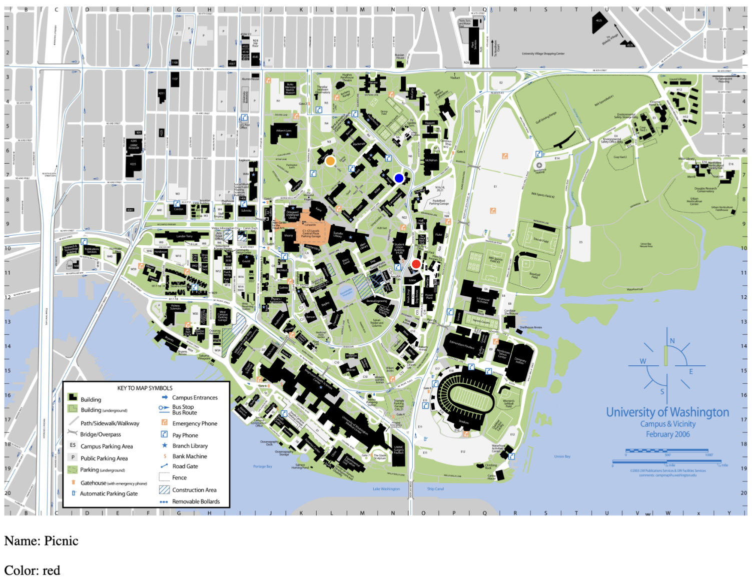
Before you begin Task 1, be sure to read the instructions of Task 4 to learn what information you need to keep track of while debugging. Then, as you work on each coding task, whenever you see a bug, open up your debugging log and record that information so that you can submit it in Task 4.

If you get stuck on a task during the debugging process, move on to the next one and come back later. Having an initial implementation of all the tasks before turning in the assignment is more important than debugging one task completely.

Check out the starter code for this assignment:

```
git clone https://gitlab.cs.washington.edu/cse331-24au-materials/hw2-locations.git
```

Navigate to the `hw2-locations` directory and run `npm install --no-audit`. Then you can run the application with `npm run start` and open it at `http://localhost:8080`.



Name: Picnic

Color: red

The application displays a campus map along with a number of "markers" indicating particular things happening on campus. The `App` displays the current set of markers on top of the map. It also maintains an optional "selected" marker. If a marker is selected, the `Editor` component is displayed below the map, allowing the user to edit the details of a particular component. Initially, this component simply displays the name of the selected item. In Tasks 1–3, you will improve the functionality of the `Editor` to let users edit the name, color, and location of markers in various ways.

In order to figure out what marker was clicked on (if any), the `App` component maintains the markers in a tree (as well as in a simple array). This is similar to a binary search tree except that, because the lookup keys are (x, y) locations, nodes of the tree split into NW, NE, SE, SW quadrants instead of left and right. The details of the data structure are not important to the assignment (you won't use it in the code you write), but if you are curious, you can look in `marker_tree.ts` to see the details.

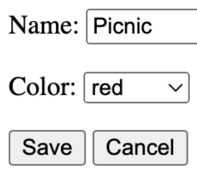
For your work, however, it is important to understand that the keys in this tree are (x, y) pairs, which have type `Location` (see `marker.ts`). As in a binary search tree, the keys are arranged carefully in the tree based on their relationship to each other. If a key is mutated, the tree may not work properly! (The same is true of `Map`s in Java.) In principle, `Location`s that are not being used as keys might be okay to mutate, but we strongly recommend that you take the safest approach and **avoid mutating** any `Location` objects in your code.

You'll notice that the `Editor` component includes the function `componentDidUpdate`. This is a built-in function of React that automatically runs (i.e. we don't need to manually call it) when the props or state of the component are changed. It takes the values of the props and state prior to the update which allows for comparing to find the exact change. In this case, we pass in props identifying the currently selected marker and initialize states of the `Editor` with those values, so when the props are updated for a new marker, we want to update the `Editor` states also.

For this assignment, we have **no restrictions on the functions you use** (as long as they do not require downloading any new packages). However, we recommend, for the sake of your debugging experience, that you stick to the functions we've seen in lecture.

Task 1 – It's All Fun and Names [25 pts]

Update the `Editor` component to display the name of the marker in a `<input type="text">` element and the color in a `<select>` element. The list of available colors is provided in the `COLORS` array in `marker.ts`. Your UI should look something like this[1]:



When the user clicks the "Save" button, you should invoke the `onSaveClick` callback passed in props. This will cause the App to re-render, showing the updated color on the map. When the user clicks the "Cancel" button, you should invoke the `onCancelClick` callback passed in props. This will remove the editor from the screen.

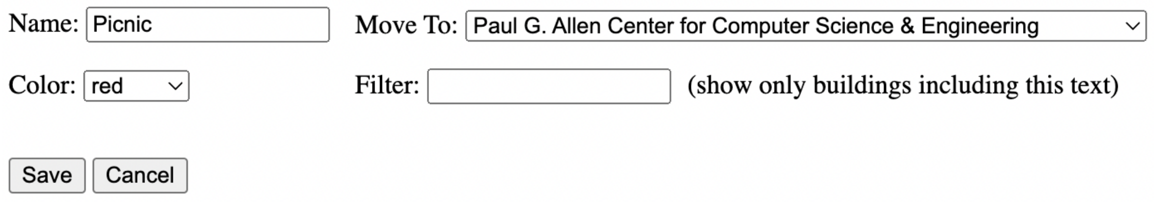
Task 2 – Snap, Crackle, Drop [25 pts]

Update the `Editor` component so that the user can choose a building from a drop-down (an HTML `select` tag) and then click "Save" to move the marker to the location of that building.

To make this easier, add an `input` text box that allows the user to filter the drop-down to only show buildings whose (long) names include the given text. Additionally, these filtered buildings should be ordered in the drop-down lexicographically (alphabetically). For example, typing "computer" into the box should reduce the dropdown to just the Gates and Allen buildings, with "Bill & Melinda Gates..." first and "Paul G. Allen..." second. When no there is no text entered to filter on (in the initial state or if the user types a filter and then deletes it), the buildings in the drop-down should maintain their original ordering (as given in the `BUILDINGS` array).

Your dropdown should start with an option that, when selected, leaves the marker in the same location. (Without this, it would not be possible to change name or color without also moving the marker!)

Your UI should look something like this[2]:


Name: Picnic
Move To: Paul G. Allen Center for Computer Science & Engineering
Color: red
Filter:
(show only buildings including this text)
Save
Cancel

As before, the item should not move until the user clicks "Save". "Cancel" should leave it unchanged.

Make sure that, if you move a marker away from its original location, you can click on a different marker, click back on the first marker, and then move it a second time.

[1]The precise details of the layout and styling are not important. This is not a UI design class.

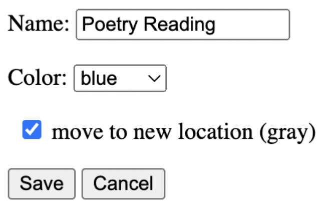
[2]Again, the precise details of the layout and styling are not important. This is not a UI design class.

Task 3 – Click 'Em When They're Down [25 pts]

Update the `Editor` component so that the user can move to a new location by clicking on the map instead of selecting the name of a building.

The `App` component already detects the case where the user clicks on a marker and then clicks elsewhere on the app. In that case, it will pass in the new location in the `moveTo` property of the `Editor` (in all other cases, its value is `undefined`). When that location is provided, you should show UI that allows the user to move the marker to that location or, by checking a box, not move at all.

Your UI should look something like this[3]:



Note that, in this case, the user should **not** see the UI created in Task 2 that lets them move to a building with a dropdown. They instead should see just this simple UI confirming that they want to move to this new location.

Again, the item should not move until the user clicks "Save". "Cancel" should leave it unchanged.

Task 4 – Does This Log Your Memory? [25 pts]

Submit your log of all time spent debugging, along with an explanation of the cause of the bug. Specifically, for each bug, provide the following information:

- What (incorrect) behavior you saw that told you there was a bug.
- How many **minutes** it took you to find the bug.
- What kind of experiments you performed to try to locate the bug.
- Where an error first occurs in the console that leads to the incorrect behavior.
  Note that when we ask you for the line where the error occurs we want the **actual code not just the line number**!
- What the defect was that caused the bug (if you ever found it).
- Was the bug the result of *mutating* a field or array that was not supposed to be mutated?

Again, we have provided a debugging log website for you to record your debugging. Don't forget to save your log!

Since debugging is the most important part of this assignment, you can still get full credit for submitting an incorrect solution provided that you fully document at least **6 hours** (360 minutes) of debugging. You are forbidden from spending significantly more than 6 hours debugging.

If you think your solution is correct, and you have not yet reached 6 hours of debugging, be careful! Have someone else try your app to see if they catch any bugs.

[3]Don't make me say it again.

Submission

After completing Task 4, download your logging as a PDF following the instructions from HW1. Again, make sure the downloaded file is called "DebuggingLog.pdf". Submit the following files to the "HW 2" assignment on Gradescope:

DebuggingLog.pdf `Editor.tsx`

After you submit your work, an autograder will run to verify you have submitted the correct files; wait for it to finish and check that the submission looks correct. Don't forget to check that the submitted files are up-to-date with all implementation and debugging you completed!