Heuristic Evaluation for Keyboard

Edgar M. Salazar

esalazar@mit.edu

6.813 HW2

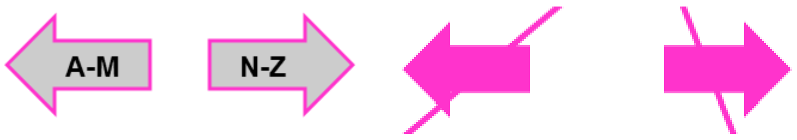
1 Introduction

This report intends to provide a heuristic evaluation of the web application *Keyboard*. The evaluation follows the Nielsen Heuristics and other principles learned in class dividing into four sections: learnability, safety, efficiency, and design.

2 Learnability

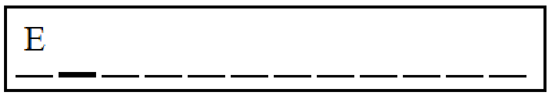
2.1 Good: Arrows as Affordances

There is a good sense of how to manipulate the state of your keyboard application because of the arrows pointing left and right.



2.2 Cosmetic: Emphasis of underscore of selected letter in text

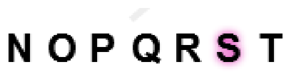
It was difficult to initially understand which position I am currently changing.



It might be better to make a stronger emphasis on this position in order to better suit people who are using TVs or a screen that is not that close to them.

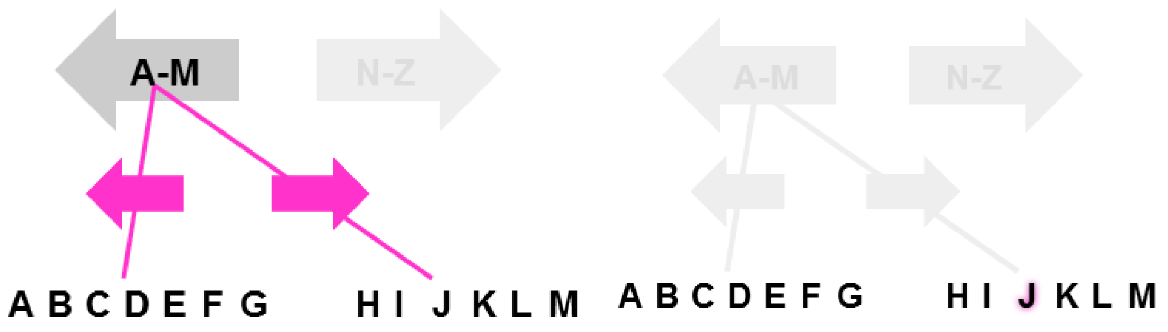
2.3 Cosmetic: Emphasis of selected letter in tree

Similar to my previous comment, it is difficult to assess which letter I am about to select.



2.4 Minor: Animate selected path

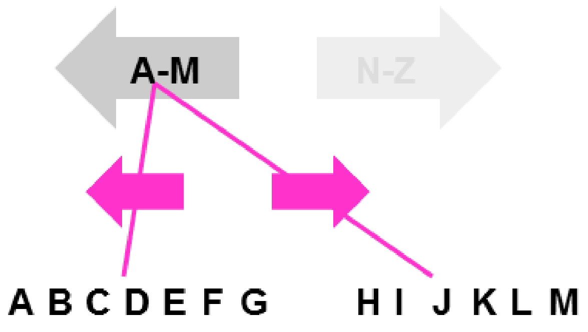
It is difficult to see transitions and feedback over what was selected and might leave users confused over what they selected and how the flow of the application works. If I select A-M, it would be nice as a first time user to see slowly what my next state is, because it is a lot of information to consider if you just blast the next step at me.



These two steps have no intermediate feedback that allows the user understand what is happening.

2.5 Major: Lines can get confused as arrows

Because you are using arrows as affordances, the lines in between {A-M} and {ABCDEFG} or {HIJKLM} shown below can be distracting and might confuse a user. It gets confusing if I should be going down or up, which is incorrect.



It might be better to redo how you portray this stage, mentioned below. I would recommend removing the lines.

2.6 Catastrophic: Confusion of Up/Down functionality

It is clear what left and right does, but up and down can be difficult to understand. There are several learnability and safety issues (mentioned in the next section) involving up and down. It is awkward to have up for going up and tree and down NOT to go down the tree. I think this brings in a few inconsistencies that might be difficult to assess. My initial thinking of how the interface worked was by using my instinct, pressing the down button to go “down the tree”. I think you need to restructure how the layout looks. Suppose you rid the lines (not arrows) and attempt to encapsulate the {ABCDEFG} inside its own left arrow. This is just a suggestion; the rest of my comments might include this same issue.

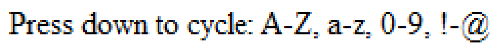
3 Safety

3.1 Good: Select other characters to replace them

It's good to see that you can traverse through the main text to replace characters.

3.2 Cosmetic: Simplification of help text

The text help can probably be better simplified. The current text is more of a long sentence. I am sure there is a way to condense the text in order to reduce the overhead of reading it or reduce the chances of people skip it.


Press down to cycle: A-Z, a-z, 0-9, !-@

3.3 Major: Deletion of characters

I am not sure, but it didn't seem like you can erase a character. Suppose you want to type "The Walking Dead" and end up typing "The WalkingxDead" I couldn't find a way to insert an empty character or remove the "x".

3.4 Major: Attempt to go back to the previous option

This is related to the previous issues of up/down confusion. Suppose that I accidently press the down button because my Xbox controller is a little messed up. This would switch the state to numbers or capital letters. My first reaction is to undo this by pressing the up button, which will just leave me to go up the tree. I will have to go back down the tree and press down until I regain my correct letter state.

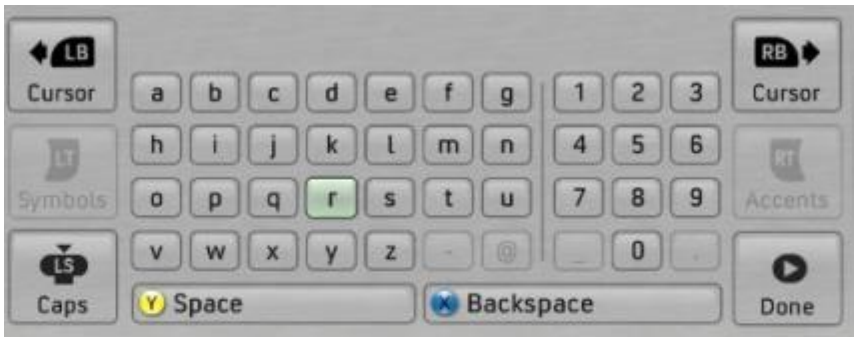
4 Efficiency

4.1 Good: Binary Tree

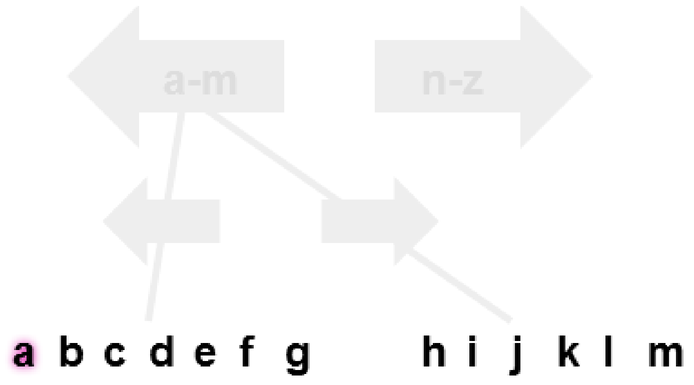
This is a good approach than many other linear solutions that I have seen.

4.2 Catastrophic: Number of traversals (button hits)

In other systems, like the Xbox, the farthest you will ever have to traverse the keyboard is a maximum of 5 keys (if I am not mistaken). The farthest would be getting from r to g. We assume that we can wrap around, meaning that the best way to get from h to n is to simply go left and it will wrap to the other side of the keyboard falling to n.



Now your solution would have us going to a maximum of 6 characters. Which is the case to go to find a (left, left, then go to a by traversing d c and b). Because you restart the entire keyboard then we can't make comparisons from previous keys.



Is this seen as more efficient than other approaches? Perhaps not in the maximum cases, but either way can there be a better way of reducing that maximum traversal? What if after you select a letter, you don't restart the system and stay at the same state of selecting between a and m? The maximum case increases, but will it be better for the user to get to nearby keys on average? These are all just thoughts that I hope you consider.

5 Design

5.1 Good: Unique approach

I really like this project because it is very unique and might be in fact very useful.

5.2 Good: Attempt to use on multiple devices

This project does attempt to use a very minimal number of commands that are universally found on all remotes/devices. This is a great scalable project.

5.3 Minor: Arbitrary selection of breadth/depth of tree

How did you select the breadth/depth of the tree? Would it decrease the maximum number of traversals by making a tree with 3 children?

5.4 Minor: Use of other buttons besides navigation keys

Because you only say you want to use up/down/left/right on the keyboard, have you considered using the enter key more often? For example you only use it to select the letter, but what if you also use it on the initial step to allow users to go up and down to select between [a-z], [A-Z], etc. This would no longer confuse users on using up/down in that state. It is also possible to use this enter as enter/exit of changing that state.

5.5 Minor: Autocomplete

Have you considered how autocomplete would work on these systems using your keyboard? It doesn’t seem like something that was considered and perhaps should be since most applications do include autocomplete, like a movie finder.

6 Conclusion

Overall the project is very interesting and does have a few kinks to work out. I think that most of these issues can be worked out by doing intensive user testing. This would help you find the optimal keyboard design.