Sizing-up the Competition

Chen-Che (Eric) Ma, Jimmy Bao Nguyen, Kendall Reonal, Rebecca Ta

ZoomBoard

Core idea

ZoomBoard utilizes iterative zooming to accurately select letters on a soft QWERTY keyboard on a screen as small as a US penny. Tapping on the screen will increase the size of the keys around the impact area until they are large enough to be pressed. The level of zooms is customizable, allowing ZoomBoard to be effective in various screen sizes and by various people with different abilities. ZoomBoard also features a keyboard to input symbols that can be accessed with an up-swipe. The zooming feature makes ZoomBoard stand out as an effective small device text entry method. With tiny screen sizes, zooming is helpful in alleviating the fat finger problem and therefore in increasing accuracy.

Critique

Basing the keyboard on the QWERTY layout makes it immediately familiar to many users, leveraging the thousands of hours of practice people have with the standard keyboard layout. The zooming mechanic enlarges otherwise impossible to hit buttons in close proximity on a small screen, solving the fat finger problem. However, every letter input requires at least two taps, making it tedious to input common letters. In other words, the iterative zooming method increases the keystrokes per character (KSPC), which directly increases the time needed to input in a character.

Additionally, if a user accidentally taps on an incorrect area of the board, it is impossible to return to the default state unless they select a letter they did not initially want. This requires an additional backspace input, along with going through the letter input process again, to get the desired character in the right place. The only way to input uppercase letters is to tap and hold, which results in a large amount of time spent when inputting a large amount of uppercase letters. Lastly, there is zero affordance on entering the symbols keyboard with an up-swipe.

Suggestion

To make it less tedious to input in commonly used letters, the buttons for these letters can be made larger than the rest of the letters on the keyboard, making them easier to hit without the use of the zoom feature on certain screen sizes. While this technique would not be effective on ultra-small screens, such as the size of a US penny, no devices on the market currently feature such screen sizes. We can further reduce the time needed to inputting words by implementing

predictive text right above the keyboard. The predictive text feature can also utilize the iterative zooming mechanic, making it possible to input the desired word on a small screen. Again, this feature would not be effective on ultra-small screen sizes because it would be difficult to read the suggested words in the first place.

Allowing users to pan around the keyboard once zoomed in will handle the problem of users accidentally zooming in on an area that does not contain their desired letter. Lastly, there are empty spaces to the left and to the right of the spacebar on the soft keyboard. Implementing a caps-lock button to the left will drastically reduce the time required to input a series of uppercase letters. Having a button on the right of the spacebar to bring up the symbols and numbers keyboard will allow for greater affordance on what ZoomBoard can offer.

Swipeboard

Core idea

SwipeBoard uses a two-swipe action to assist with character selection. It also uses additional gestures for other functions such as switching the keyboard from letters to numbers, and adding spaces and deleting characters. It arranges the letters in the QWERTY layout, but places them into a 3x3 grid. Each section in the grid contains three characters, except for one section that contains four characters. In order to access the desired character, two swipes are required. Each of the grid spaces is associated with a directional swipe gesture that is target agnostic, meaning that it does not matter where the swipe occurs on the touch surface. For example, the letters “QWE,” are accessed when the user performs a swipe to the upper left, since that region is in the upper left of the grid. To access the centermost region, or the letters “FGH,” users simply tap the screen. The first swipe narrows down the selection by grid space, and then the second swipe selects the actual desired character. For example, once the region containing the letters “QWE” has been selected, the screen will display the three letters, with each letter associated with a gesture. Swiping to the left will select the letter on the left, while swiping to the right will select the letter on the right. A tap anywhere on the screen will select the letter in the center. The concept of target agnostic text entry helps SwipeBoard’s case as an effective small device text entry method. The keyboard avoids the “fat finger” problems of precision and occlusion, where those with larger fingers either have difficulty with accuracy or cannot see what they are selecting. By using swipe gestures that do not require the user to actually press a button, the user does not have to worry about pressing the wrong tiny buttons. In addition, SwipeBoard narrows down the regions of the keyboard and increase the target size when a region is selected, making it easier for the user to select the desired character.

Critique

Although it might help users by narrowing down regions of the keyboard to improve accuracy and avoiding the fat finger problem, users will only be viewing a set of three to four letters after the initial swipe. They may occasionally access unwanted letters with no easy way to correct themselves. For example, if a user wants to access the letters “ASD,” they must swipe to the left. If the user accidentally swipes to the upper left, they will access the region containing the letters “QWE,” and won’t have access to “ASD” unless they exit out of the “QWE” board.

Suggestion

When a grid region is selected through one of the assigned swipe gestures, the user should not be locked into only viewing that selected region. Users should not be relied upon to perform the desired swipe gesture with 100% accuracy, so there should be a way for them to easily correct themselves if they make a mistake. If users make an incorrect swipe due to their finger slipping, then the next screen that the user sees should also contain the adjacent grid regions. For example, if a user is trying to access the letters “ASD,” but they accidentally swipe to the upper left instead of directly to the left, they should still be able to access the “ASD” set instead of being locked into the “QWE” set, which an upper left swipe would summon.

A way to accomplish that is to make the second screen another 3x3 grid with the same swipe functions instead of just a three to four character sets. When the user makes a certain swipe gesture, the grid regions with similar gestures should be displayed to mitigate errors. For example, if a user swipes to the left, instead of just displaying “ASD,” the second screen will also display “QWE” and “ZXC” since they are both in the left direction and could be the actual target of the user. For the one region that contains four characters, the second screen could use predictive capabilities to choose which five neighboring characters to place on the new 3x3 grid. Also, since the center region is summoned with a tap and not a directional swipe, it could be left the way it is. Although this suggestion will probably not do anything to increase the speed of text entry using SwitchBoard, it mitigates potential errors by allowing users to easily correct themselves.

DriftBoard

Core idea

The main idea behind Diftboard is that it uses a fixed cursor point and a movable QWERTY keyboard. When a user presses down on the screen, it selects the keyboard. When the user’s finger move, the keyboard will move as well. The user can then move the keyboard to a position where the desired key of the input is under the fixed cursor. Once the desired key is under the

fixed cursor, the desired key will be inputted when the user releases the finger. Note that the movable keyboard used in DriftBoard can extend outside of the interactive screen.The idea behind using a fixed cursor and a movable QWERTY keyboard is that it addresses the fat finger problem; a user with a bigger finge ris able to see which key is being inputted due to the fact that the position of input isn't directly underneath the finger.

Critique

The decision behind using a QWERTY keyboard is helpful in that it is the most common keyboard and that a majority of the user is more familiar with the position of each of the keys, which thus decreases the time it takes to locate the desired key. Finally, because the keyboard of DriftBoard is movable, it can save a lot of space because the screen does not have to show the entire keyboard. Although DriftBoard addresses the fat finger problem, as well as the familiarity of the keyboard allows user to know where the keys are at, there are some issues that arise with this input design. The first is that the size of the keys on the keyboard will require users to be able to accurately move it under the cursor. This means that people who has motor impairments will have a bit of trouble to accurately place the desire key under the fixed cursor. Another issue that arises from this input design is time. Because it takes 3 steps (finger down, move, fingers up) to input 1 character, it will take a bit time to input an entire word.

Suggestion

As discussed in the critique section, the flaws that DriftBoard has is the accuracy of placing the desired key under the cursor due to the size of the key, and time efficiency of input. One way to solve the first problem is making the size of the keyboard bigger. This suggestion is helpful because by increasing the size of the keyboard, users will be able to place the desired key under the fixed with a greater accuracy because there's more surface area on the key. The issue with time efficiency is a bit difficult to fix due to the nature of the way DriftBoard functions. However, one suggestion that can decrease the time it takes to make an input, is to have multiple cursors around the screen. Having multiple cursors round the screen will decrease the distance between the initial point of the key and the targeted destination. By shortening the distance, the time it takes to make an input will also shorten (due to Fitts's Law). The first suggestion of making the size of the keyboard bigger will also shorten the time because it also shortens the distance between the 2 targets.

SplitBoard

Core idea

SplitBoard displays a partial view of a larger QWERTY keyboard. It divides the standard QWERTY into two main sections and includes and extra section for the Caps, numbers,

symbols, and enter keys. This extra section enables the entering of numbers, symbols, and uppercase letters. In the two main sections, there are 3 rows of keys and six keys per row, forming a 3x6 grid. To use SplitBoard, the user flicks horizontally left or right to access the left or right sides of the QWERTY keyboard, then taps on the corresponding key. The space and backspace keys are located at the bottom of the screen, and can be selected by tapping the bezel below the screen. Although the back and space keys are very small, users can tap the bezel below the screen to input those keys, which helps alleviate “fat finger” problems.

Critique

Since the QWERTY is the most dominant keyboard on personal computers and handheld devices, its familiar layout will probably encourage users to use it and it will accommodate a faster transition from novice to expert. Due to its simple layout, SplitBoard has immediate learnability and light processing.

SplitBoard outperformed the entry speed of ZoomBoard, although with a higher error rate. This seems to be an effect of the speed-accuracy tradeoff, where a user performs a task very quickly resulting in a large number of errors.

The letters in SplitBoard are still fairly small; there isn’t a zooming option, which may cause inaccuracies by occlusion. On the other hand, frequent zooming (such as in ZoomBoard) may cause eyestrain.

SplitBoard uses a combination of tapping and swiping to input characters. While tap-slide combinations can save motor effort and are potentially faster for experienced users, planning the tap-slide movements is difficult for new users, which makes these movements time consuming.

The SplitBoard is also inconvenient to use when a sentence requires a lot of switching between keyboard parts. Although SplitBoard has easy learnability with its familiar QWERTY keyboard, text input may be faster if the keyboard is structured around commonly used letters.

Suggestion

Since the tapping and swiping input method may be more difficult for inexperienced users, it may be helpful to combine a tap or key press with other measures, such as tilting the device or applying pressure.

SplitBoard may require a lot of switching between keyboard parts, especially because the section containing the shift and enter keys is at the far right. For example, if a user is typing on the left side of the QWERTY keyboard, they would need to swipe twice to shift, enter, or input numbers or symbols, then swipe twice to get back. It would be more useful to have a shortcut to the extra section on both of the main sections.

As mentioned in the critique, text input may be faster if the keyboard is structured around commonly used letters. By shortening the distance between commonly used letters, the time required to input text could also be shortened because of Fitts' Law.

Lastly, it may be useful to have an auto correcting or predictive text option since there was a high error rate in the study. A predictive text option could further reduce typing times and error rates, due to a lower probability of hitting the wrong key since there isn't a need to type out the full word.

SwipeKey

Core idea

SwipeKey aims to make small device text input easier by reducing the amount of gestures needed to produce output, while also increasing the size of the buttons on the screen. It does this by consolidating multiple letters and characters into each individual button that is displayed to the user. Each button can be swiped over to summon the desired character. At the time of the research paper's publishing, SwipeKey had two different variations: SwipeKey4 and SwipeKey5. Both variations arrange the keyboard alphabetically. SwipeKey4 has eight square buttons; seven of the squares contain four characters and the eighth square is blank. The characters are arranged in a cross-like pattern, so each button contains a character at the top, left, right, and bottom. SwipeKey5, on the other hand, has six square buttons; five of the buttons contain five characters and the sixth button has three characters. The fifth character in each button is placed in the center. Both variations follow one core idea: only one tap or swipe gesture should be needed to select the user's desired character. For SwipeKey4, characters can be selected by the user when they swipe up, down, left, or right over the desired button. SwipeKey5 follows the same pattern, except adds a tap gesture to input the character in the center of each button.

Critique

SwipeKey directly compares itself to SwipeBoard since both small device input technologies utilize swipe gestures. However, SwipeKey tries to distinguish itself by focusing on the fact that it only requires one tap or swipe to input a desired character, while SwipeBoard requires two. In theory, having one less user action will result in increased input speeds as the user grows accustomed to the keyboard. In addition, by containing several characters, each button can be displayed at an increased size, making it a larger target, which can be helpful since SwipeKey utilizes absolute pointing to produce direct input.

One issue regarding SwipeKey is the "fat finger" problem, which is common for input technologies for small devices. Since SwipeKey is dependent on the user's target awareness, a

user with larger fingers may have difficulty knowing exactly where the button that they want to press is located. This is mainly due to occlusion, which is one of the components of the fat finger problem, where the user's view of the screen is obstructed by the size of their finger. The issue of occlusion will negatively affect the user's accuracy when using SwipeKey. Another problem that users may face when using SwipeKey is the arrangement of the keyboard. Currently, the characters are arranged in alphabetical order, which the designers deemed to be easy for users to learn and to have similar performance to traditional QWERTY keyboards. However, the layout is unimaginative and is not leveraging user tendencies. If the layout is arranged in a way that utilizes commonly used letters and letter frequencies, the average words per minute may increase.

Suggestion

SwipeKey can be improved by looking into different arrangements of the characters on the keyboard. Instead of just displaying the letters in alphabetical order, it could be possible to research commonly used letters and letter frequencies in order to create a faster experience for the user. Although the keyboard will seem unfamiliar at first, input speed will only increase over time as the user continues to use SwipeKey. That tradeoff has been apparent with most of the input technologies that have been designed for small devices. If the user is going to need to learn a new input technology, then they should learn one that will give them the potential for an increased input speed. Another suggestion for SwipeKey is to provide a way for users to easily correct themselves or see where they are going to be pressing. This can be accomplished by adding a hover state with feedback on the screen. Users would be able to see where on the screen they are going to touch by highlighting that area on the screen before they place their finger down. Adding a hover state with visual feedback will help increase users' accuracy when they suffer from the fat finger problem, which is important for a technology that requires target awareness.