KeyMenu: A Keyboard Based Hierarchical Menu

Kent Lyons, Nirmal J. Patel and Thad Starner
College of Computing, GVU Center
Georgia Institute of Technology
Atlanta, GA 30332-0280 USA
{kent,merik,thad}@cc.gatech.edu

ABSTRACT

KeyMenu is a keyboard based hierarchical menu system originally designed for use on a wearable computer. The menu uses a one to one mapping between buttons on the keyboard and menu items. The KeyMenu leverages off of the advantages of other pointer based menus such as Marking Menus. It provides support for both novice and expert user interaction through the use of a delay in popping up the menu. Finally, by using a consistent physical action we support the transition from novice to expert.

KEYWORDS: Marking Menus, one-handed keypad interaction, wearable computers

INTRODUCTION

The KeyMenu is a new keyboard based hierarchical menu which utilizes the same input method to navigate and activate menu items for both the novice and expert. KeyMenu is a pop-up menu using a grid of 3 columns and 4 rows of menu items. Each item on the menu corresponds to a key on the keyboard in the same relative position (Figure 1). The user presses a special menu key to activate the menu and then proceeds through the menu hierarchy by pressing a sequence of keys.

In this paper we present the motivation for creating a keyboard based menu system and highlight some of the problems of keyboard access to traditional menus. We then examine alternative menus and present the design of KeyMenu along with its advantages. Finally we discuss areas of possible future work.

MOTIVATION

The KeyMenu is an alternative to traditional menus which often require the use of a mouse and was originally inspired for use on a wearable computer [5]. On a wearable, a pointing device is often not appropriate. The wearable user is regularly interacting with the real world, and the user's primary focus of attention is not on the screen shown in her head mounted display. This style of use breaks the tight feedback loop required for the hand-eye coordination needed to manipulate a pointer on the display. The problem is compounded by the small screen size both in terms of pixels and field of view making the visual acquisition of the cursor on the display difficult.

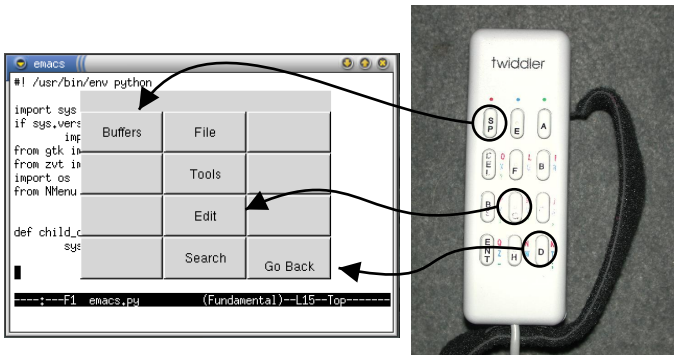

Figure 1: Each of the twelve keys on the front of the Twiddler (right) correspondence directly to a menu item on the KeyMenu (left) in the same relative position.

The physical control of the pointer's input device is another problem on a wearable computer. Although there are a variety of devices that can be used to control a pointer, many become difficult to use when the user is not stationary. The mechanical noise created by the user moving through her environment often affects the pointing device creating erratic movement in the cursor. This effect was prominent in data we collected while building a system to capture a wearable user's interaction in a realistic setting [4].

While using a pointing device is problematic while mobile, touch typing using a one handed keyboard such as the Twiddler is still possible while running down a flight of stairs. The Twiddler, shown on the right of Figure 1 and in use in Figure 2, is a chording keyboard with twelve buttons on the front operated by the user's four fingers and four to six modifier buttons on the back operated by the thumb. The KeyMenu is designed to leverage this physical layout in navigating menus.

TRADITIONAL MENUS

Traditional menus can be navigated with a variety of techniques such as using a mouse to select a top level menu item

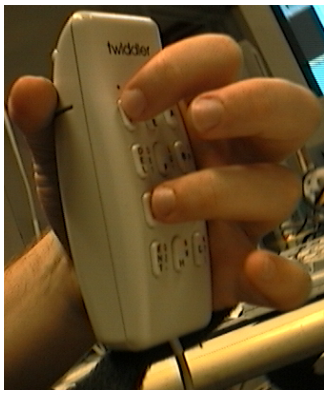

Figure 2: The Twiddler chording keyboard has three columns and four rows of buttons on the front operated by the user's four fingers. Four to six modifier keys (such as "Alt") are on the back and operated with the thumb.

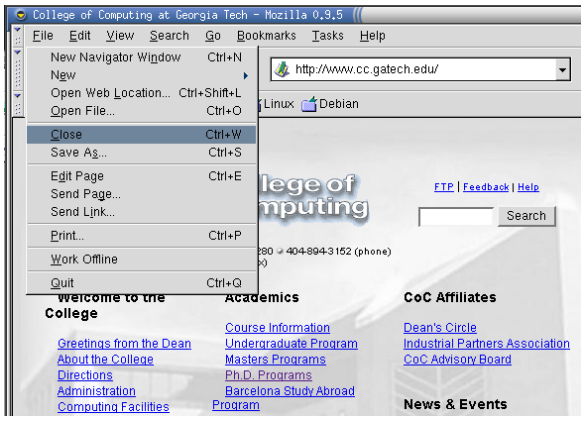

Figure 3: In Mozilla, there are several ways to activate "Close" using either the keyboard or mouse.

followed by either sub-menus or commands. Many menus can also be navigated with the keyboard, and some commands can be executed directly with a shortcut key sequence.

For example, the web browser Mozilla (Figure 3) supports numerous ways to close the current window. With the mouse, the user can select *File* and then *Close*. The next option is to press "Alt+F" to open the *File* menu. The user then has the choice of navigating the sub-menu using the arrow keys (pressing down 4 times then "Enter") or to activate *Close* directly by pressing "C". Finally, an expert user can bypass the menu entirely and use the shortcut key, pressing "Ctrl+W" from the main window.

These methods offer varying levels of support for novice and expert users; however, there is a disconnect between the different ways of executing a command. If the user is to change between methods, she would need to make a mode switch in how she controls the system. Changing from the mouse to keyboard requires a very different interaction, and even switching from navigating the menus with a keyboard to using shortcuts requires the use of different key presses. The only support for making a transition between using the techniques are the cues placed on the menu. In the above example the "F" in *File* is underlined indicating it can be activated with "Alt+F". In the menu, the "C" is underlined indicating that it can be activated by pressing that letter. The last cue is the shortcut key sequence on the right side of the menu item.

Unfortunately, even the keyboard interaction is inconsistent. The user presses "Alt+F" to open the *File* menu, but once the menu is open "Alt+C" does not activate *Close* even though both are highlighted with an underlined letter. Likewise, if the *File* menu is open, pressing the shortcut key only dismisses the menu and does not execute the command. This is not an isolated occurrence; a quick survey of several different web browsers on different platforms (Mozilla and Netscape Navigator on Linux and Navigator and Internet Explorer on Microsoft Windows) shows similar problems.

RELATED WORK

Pie Menus [1] and Marking Menus [3] are an alternative to traditional menus which differ in layout and mouse utilization. The menus are organized in a circle and the user selects a menu item by moving the mouse pointer at the correct angle into a wedge of that circle. Hierarchical Marking Menus [2] allow multiple levels of menus. The user draws a stroke into a wedge of the circle which can then pop-up a sub-menu. The user selects the sub-menu by continuing the gesture in the correct direction to select the desired element.

These radial menus support novice users by showing the menu after a short delay. The display provides a prompt for the user of the available menu options when needed. An expert uses the same physical motion as the novice but does not pause during the stroke. When executed without delay the menu is not shown, and the command is activated directly. Thus, a novice can progress to an expert skill level with the interface without needing to learn new actions. KeyMenu transfers these benefits into a keyboard only interaction.

KeyMenu DESIGN

The KeyMenu is designed around a keyboard only input interaction that provides menu prompts to a novice user and allows an expert user to execute the command directly without distractions. The design eliminates the mismatch in the interaction methods of traditional menus highlighted in the Web browser example. As in Marking Menus, we aid the transition from novice to expert use by utilizing a consistent input technique.

The KeyMenu uses a grid of three columns and four rows that corresponds directly to the physical buttons on the Twiddler keyboard or the keypad of a mobile phone (Figure 1). The user activates the KeyMenu by pressing "Alt" which serves as the menu key. This button is activated by the thumb on the Twiddler and is not part of the twelve buttons used to navigate the menu items. Once the menu is activated, the display

shows up to twelve menu items. The user navigates the menu hierarchy by pressing buttons in the corresponding positions on the keyboard. For example, to activate the menu item in the top left corner of the menu, the user presses the top left key on the keyboard. The current position in the menu hierarchy is displayed at the top of the KeyMenu providing feedback to the user.

Items on the KeyMenu are structured using a combination of frequency of use and and ease of pressing the associated keyboard button. This layout is used as an alternative to the convention where the first letter of the menu item instantiates the command. As with traditional menus, there is often more than one item that could use the same letter. For perspective, a default install of Emacs has over 200 commands bound to keys and there are over 2000 runnable commands in total. Another difficulty with using the first letter of the menu item is that very commonly accessed items might be in very inconvenient locations. KeyMenu accounts for such difficulties. For example, on the Twiddler the index finger is used to press buttons on the top row. The "home row" where the user's fingers rest is the middle column of buttons. Consequently, the optimal placement for the most commonly accessed menu items is a "T" across the top and down the middle of the KeyMenu.

KeyMenu INTERACTION

We used KeyMenu to provide a menu for Emacs. Figure 4 illustrates a novice user's interaction with the system using a Twiddler style keyboard. To pop-up the menu the user presses the menu key using her thumb. The user then waits for a short period (approximately 0.5s) and the system displays the main menu showing the available options. Next, the user presses the top middle key activating the *File* submenu. Finally, the user presses the top right key activating the *Close* command.

KeyMenu utilizes a delay in displaying the menu similar to the Marking Menu [3]. After activating the KeyMenu, whenever there is a pause greater than 0.5s the menu is shown to the user. This enables us to support both novice and expert users with the same technique. In the above example the user waits after pressing the menu key to see the available options. As the user traverses the menu hierarchy by pressing keys, the menu updates showing the corresponding submenus. The display is dismissed when the user activates a command or cancels the menu by pressing the menu key.

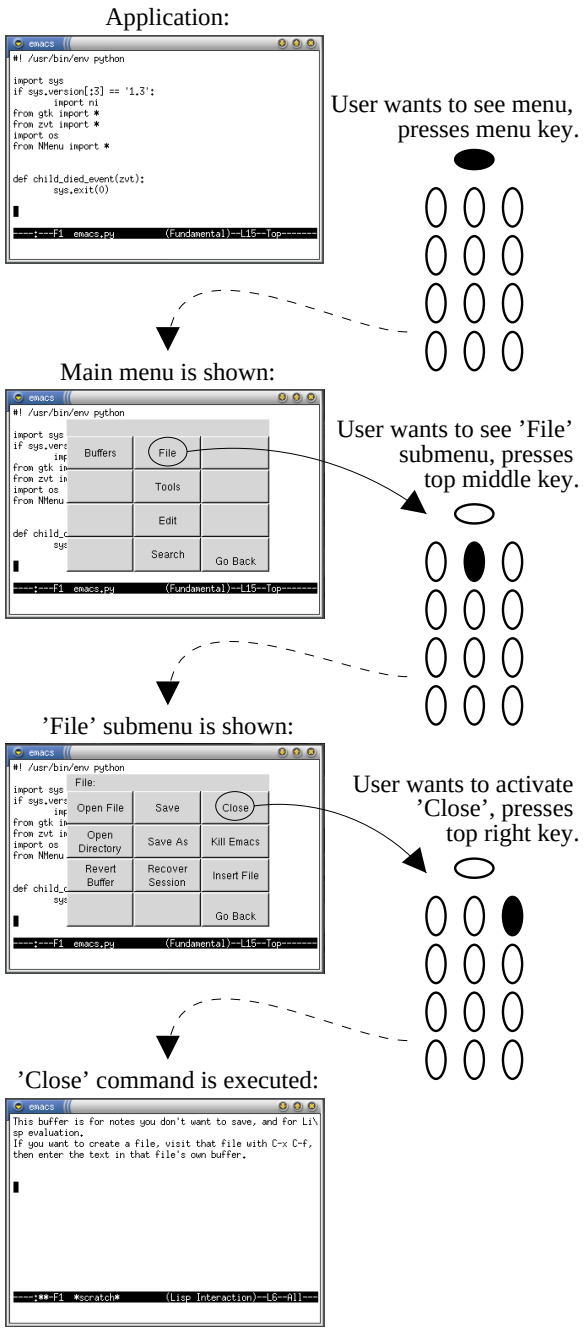

Figure 4: The KeyMenu is used in Emacs with a Twiddler style keyboard. The left column shows the application state, and the right column represents the buttons pressed on the keyboard.

The delay in showing the menu also affects more advanced users. An intermediate user might know that *Close* is under *File* but is not sure of exactly which key corresponds to *Close*. The user can press the menu key and then the key for *File* in quick succession. After a small pause the *File* submenu is shown. The user can verify that she is in the correct sub-menu be reading the top line of the menu which now shows "File:". The user can now see what key to press and activates *Close*. An expert user can enter the entire key sequence without pause and the command is executed directly, never displaying the menu. Here the key sequence acts like a shortcut key.

We simplify user interaction by always using the same key sequence to activate a command. KeyMenu gives prompts to the user when needed by incorporating a delay which activates the menu's display showing the user the available menu options. Through repetition a novice user learns the key se-

quences used to select menu items. The KeyMenu responds to the speed of the user's interaction and changes feedback accordingly. When the user knows the sequence of keys to press, the menu stays hidden allowing the user to execute the command directly without distraction.

While leveraging off the advantages of Pie Menus and Marking Menus, the KeyMenu does have some additional benefits. The detection system is much simpler for the KeyMenu. Instead of interpreting a potentially ambiguous stroke the system only needs to register a sequence of key presses. The radial menus are also limited by the accuracy with which a user can manipulate the pointer to select the region, which in turn limits the number of menu items. With KeyMenu, the user has the advantage of tactile feedback from the physical buttons on a keyboard. The limit for the physical articulation of KeyMenu becomes equivalent to what the user can touch-type easily.

FUTURE WORK

There are several areas of future work. The first is in the area of evaluation. Does transition from novice to expert happen in the same way as in Marking Menus? Is the physical articulation easier with the KeyMenu and is there a similar corresponding decrease in error rate, reaction time or cognitive load?

Another question concerns the number of items to use on the menu. A grid of three columns and four rows in very natural with the Twiddler. However a 3x3 grid might be easier to conceptualize and may provide benefits to novice users.

As with other menus there are layout issues that could be examined. It would be interesting to explore techniques for menu item placement that exploit higher level semantic information of an item (similar to the first letter convention of traditional menus) in addition to utilizing the "T" of locations that offer the easiest use on the KeyMenu. Also, many commands in applications such as Emacs have duals (search forwards or backwards) or operate on different targets such as checking the spelling of a single word, a selected region or a whole file. There is a tradeoff between making the menu broader to incorporate each different permutation explicitly and creating a deeper menu hierarchy such that the different features can be composed using sub-menus. Kurtenbach explored some of these issues for Marking Menus [3]; it would be interesting to determine if there are similar effects in KeyMenus.

Mobile phones are increasingly becoming an area of user interface research. They share some similar problems with wearable computers with limited screen real estate and use in a variety of environments and situations. Given the very different interaction, a traditional desktop style pointer is not appropriate and the interaction using various types of buttons becomes critical. On these phones, menus are common allowing access to the various features of the phone and allowing configuration. The KeyMenu could leverage the phone's display and keypad (also a three column by four row layout) to allow navigation of the menus.

Another area to investigate with the KeyMenu is with using traditional desktop keyboards. Our prototype supports these keyboards by rotating the menu 90 degrees into a 4x3 grid. The home row of the right hand is used for the center row on the menu. The question to explore is if the mapping is still meaningful to the user. On the Twiddler or mobile phone the mapping of keys to menu items is one to one. Which set of twelve keys to use on the normal keyboard is less obvious to the user and could potentially cause problems.

CONCLUSION

We have presented the KeyMenu, a keyboard based hierarchical menu system that mirrors the menu layout to the physical layout of the keyboard. Each key on the keyboard maps directly to a menu item in the same relative layout.

We support novice use by using a delay similar to Marking Menus. When the user pauses in entering a key sequence, the menu is shown providing a prompt. As a user learns the key sequences, she needs less support from the system and can enter them swiftly. An expert enters the same sequence without pause which bypasses the menu display entirely, activating the command directly.

ACKNOWLEDGMENTS

This work is funded in part by NSF Career Grant #0093291.

REFERENCES

1. J. Callahan, D. Hopkins, M. Weiser, and B. Shneiderman. An empirical comparison of pie vs. linear menus. In *Proceedings of CHI'88*, pages 95–100, 1988.

2. G. Kurtenbach and W. Buxton. The limits of expert performance using hierarchic marking menus. In *Proceedings of CHI'93*, pages 482–487, 1993.

3. G. Kurtenbach and W. Buxton. User learning and performance with marking menus. In *Proceedings of CHI'94*, pages 258–264, 1994.

4. K. Lyons and T. Starner. Mobile capture for wearable computer usability testing. In *Proceedings of IEEE International Symposium on Wearable Computing (ISWC 2001)*, Zurich, Switerland, 2001.

5. T. Starner. *Wearable Computing and Context Awareness*. PhD thesis, MIT Media Laboratory, Cambridge, MA, May 1999.