TextMouse Web Executive: Software Architecture and Tutorial

Elizabeth C. A. Totten and Robert J. Hilderman

Technical Report CS-2001-01
August, 2001

Department of Computer Science
University of Regina
Regina, Saskatchewan, CANADA
S4S 0A2

ISSN 0828-3494
ISBN 0-7731-0425-9

The TextMouse Web Executive: Software Architecture and Tutorial

Elizabeth C. A. Totten and Robert J. Hilderman
Department of Computer Science
University of Regina
Regina, SK, Canada S4S0A2
{totten1e, robert.hilderman}@uregina.ca

Abstract: Computer mouse related repetitive strain injuries are a painful and costly problem in the current age of technology and high mouse usage in the work force. Traditional alternatives to the mouse, including software as well as hardware, still use some sort of computer pointing device. In this paper we introduce a unique solution. We describe the overall design and give a tutorial for TextMouse Web Executive, which is a web browser that uses a keyboard to accept input in lieu of a computer pointing device.

1. Introduction

With the high use of the computer mouse in the workforce [7], there has been an increase in repetitive strain injuries related to the use of the mouse [14]. It is widely recognized that long-term exposure to mouse usage on the computer is responsible for a variety of repetitive strain injuries to the hand, wrist, forearm, shoulder, and neck [7]. While keyboard usage has been widespread for many years, and many repetitive strain injuries can occur using a keyboard, most people do not experience or report significant pain or disability until a significant change in the frequency or duration of mouse usage occurs.

In the past, when an individual suffered repetitive strain injury due to mouse usage, possible solutions to the problem included rest, physical therapy, adjustment of the height and angle of keyboards and mouse surfaces, a different style of mouse, and for persistent cases, a splint [4]. In the worst cases, surgery may be required [4], but this is a highly invasive solution with no guarantee that the problem will be remedied. As a result of these interventions, repetitive strain injuries are not merely an inconvenience, they can have a serious impact on an individual's quality of life. In addition the economic impact on an employee, employer, the health care system, and the worker's compensation system can be significant. With the idea that prevention is the best cure, we have developed a software solution that will minimize use of the computer mouse, which is the cause of so many injuries.

One area where mouse usage is high is with web browsers. This project is focusing on techniques to minimize use of the mouse within the context of web browser software. To this end, we have developed a new web browser graphical interface, called the TextMouse Web Executive, that significantly reduces the need to use the mouse when browsing the web. The TextMouse Web Executive makes use of the mouse optional in most circumstances, and enables web browsing via the keyboard.

The remainder of this paper is organized as follows: Section 2 covers background information. Section 3 describes the approach used in developing TextMouse Web Executive. The software architecture is discussed in Section 4. Section 5 is a tutorial. The paper is concluded with Section 6.

2. Background

Repetitive strain injury (RSI) is defined as "a cumulative trauma disorder that is caused by frequent and regular intervals of repetitive actions" [14]. It includes tendonitis, carpal tunnel syndrome, and other injuries [14]. Symptoms include "headaches, radiating pain, numbness, tingling, and a reduction of hand function" [14]. Treatment involves physical therapy, anti-inflammatory drugs, wrist splints, and, in the most severe cases, surgery [4]. RSI resulting from conditions in the workplace costs an average of $3,000 in benefits and up to $40,000 in medical expenses [2], while awards of $50,000 are average in cases that turn into lawsuits [10]. Not only is RSI costly, it has been suggested that RSI is the fastest increasing disability in today's workplace [14]. "The Occupational and Health Safety Administration reported that 56 percent of all work place injuries reported during 1992 were due to RSI, up from 18 percent in 1981" [14]. RSI has been conclusively linked to use of the computer mouse [7]. Consequently, many software and hardware alternatives to the mouse have been developed.

There are things that people can do to alleviate their pain without any hardware or software. These personal solutions include wearing a wrist splint or doing strengthening exercises. Some say that exercising the opposing muscles to balance the over-used muscles is a good solution [2], while others say that this could, and often does, make the situation worse [4]. Practicing the Chinese martial art of T'ai Chi Ch'uan for about five minutes every hour is another exercise solution [8]. T'ai Chi Ch'uan focuses on the entire body, instead of focusing on the hands and wrists; it promotes an overall well-being [8]. Some agree that one of the main problems is the incorrect placement of the mouse so that the user has to reach across the desk to use it [9] [12]. There are many keyboard shortcuts that are designed so that the user can use the keyboard more and the mouse less. Unfortunately many users do not use these special key combinations because often they are not logically related to the command they are meant to execute and thus they are difficult to remember [5]. Other possible solutions that are built-in include keyboard enhancements such as StickyKeys, MouseKeys, RepeatKeys, SlowKeys, BounceKeys, and ToggleKeys [14]. If these personal solutions do not work, the user may turn to software as a solution.

There is not a lot of variety in software solutions to RSI. There are software systems that remind the user to take frequent breaks from repetitive work such as Screen Play and PowerPause [9]. This helps avoid the cumulative results of working long hours on the computer, however it does not avoid the use of the mouse altogether. Another piece of software, called NoMouse, allows the user to control the cursor with the arrow keys. Alternative hardware is also an option if software does not help.

Hardware solutions to RSI are numerous and diverse. There is a vertical mouse, called Anir™ mouse, which looks like a "joystick on wheels" that appears to help injured workers [11]. A trackball instead of a mouse may also help the user [1]. There are other computer pointing devices based on eye gaze. Manual And Gaze Input Cascaded (MAGIC) pointing [15] is an example of this. MAGIC pointing uses the gaze of the user to start the cursor in the general direction of the desired item, and then the user must use the mouse to do the fine positioning and clicking. One solution that is completely mouse-free is voice recognition, such as DragonDictate [9]. There are also stylus and finger

operated systems that have the computer screen act similar to an automated teller machine [12]. Head and mouth controlled computer input devices exist as well [3]. Other solutions that are worthy of mention include a no hands mouse (uses foot pedals), wrist supports, touch pads, graphics tablets, mouse bridges, and a roller bar (similar to a touch pad that fits just below the space bar on the keyboard) [6]. Fernstrom and Ericson argue that most computer pointing devices probably have advantages as well as drawbacks, depending on which aspects are studied [5].

It is widely agreed upon that what exists for solutions is not yet satisfactory [5][1][7][12]. Some suggest trying to decrease the use of the mouse or make it optional [7][14]. Others suggest that the effect of the device used is personal, and if alternative devices are necessary, the user must watch for signs of worsening symptoms [1]. The reason for this is that most of the current alternatives still involve the use of some handheld input device requiring fine motor skills that, after long-term use, may also produce problems with RSI.

3. Approach

The approach taken with TextMouse Web Executive was that developing software that requires no mouse at all was better than developing a different hardware device. Alternative hardware devices may produce better results immediately, but there is no way to tell what will happen in the long run. These alternative hardware devices may turn out to be great solutions, but they may also cause as many problems as the mouse. It is the repetitive strain that must be changed, not the mouse. Although there are RSI's reported from keyboard use, using the computer mouse puts a person at increased risk of developing RSI [7]. TextMouse Web Executive does not attempt to replace the mouse, as most alternatives do. It provides a different method for the same functionality. The program inserts labels (mostly numeric) beside every place on the screen that the user could click. The user simply uses the keyboard (numeric and alphabetic characters) to indicate the place on the screen he or she would like to "click." There is no pointing device involved in TextMouse Web Executive.

4. Software Architecture

TextMouse Web Executive was written in Microsoft Visual Basic 6.0, and as such, it is an event driven program. The main action of the program occurs in the events "MyBrowser_DocumentComplete," "Form_KeyDown," "KeyLabel_KeyDown," and the functions that they call. The implementation was accomplished by inserting numbers beside the hyperlinks in every HTML document viewed in the browser (in "MyBrowser_DocumentComplete") and by intercepting keystrokes and processing them (in "Form_KeyDown" and "KeyLabel_KeyDown"). In addition to hyperlinks, the menus, address bar, and buttons were also numbered. The two main parts of this program are inserting the numbers and intercepting and processing the keystrokes.

4.1 Inserting the Numbers

The first task in implementing this project was to alter the HTML of the document the web browser displays to show numbers beside the hyperlinks. This was done in the "MyBrowser_DocumentComplete" event. The code in this event uses external functions "updatelinks" and "numberform," which actually do the work of numbering the document. Inserting numbers beside the hyperlinks was accomplished by using the "Document" object in the web browser control. The "Document" object represents the document that is displayed in the web browser. It provides access to the HTML of the document, including access to everything the user sees as well as some things the user does not see. There are several useful properties and methods of this "Document" object, including a collection of "links", a collection of "all" HTML elements in the document, a collection of "frames", and a collection of "forms." The collection of "links" is an array containing all <a> and <area> tags in the document. These are the areas on the web page that the user can click on (i.e. these are the hyperlinks). The "all" collection contains all tags in the entire document, including the <html> tag that goes around everything else. The collection of "frames" is an array of <frame> or <iframe> tags, depending on the structure of the document. Frames will be discussed later. The "forms" collection contains the <form> tags. Forms on a web page contain buttons, drop down menus, radio buttons and checkboxes. All of these collections were useful in the implementation of this project.

As mentioned above, the collection of "links" is a collection of <a> and <area> tags. Because of a problem with image maps (discussed below), only the <a> tags were numbered. The function that inserts the numbers loops through the elements in the links collection and uses a method on the <a> elements called "insertAdjacentHTML" to insert a tag that looks something like this:

<FONT color=[color] size=2>[number]</FONT>,

where color is determined by a string variable (to allow the user to change the color of the numbers) and number is an integer variable that is incremented for each <a> element. This font tag could be inserted before the <a> tag started (Before Begin), just after the <a> tag started (After Begin), just before the <a> tag ended (Before End), or just after the <a> tag ended (After End) (Figure 1).

Before Begin	**<FONT color=[color] size=2>[number]</FONT>**<A href="…">… [hyperlink text] …</A>,
After Begin	<A href="…">**<FONT color=[color] size=2>[number]</FONT>**… [hyperlink text] …</A>,
Before End	<A href="…">…[hyperlink text] …**<FONT color=[color] size=2>[number]</FONT>**</A>,
After End	<A href="…">… [hyperlink text] …</A>**<FONT color=[color] size=2>[number]</FONT>**.

Figure 1: Possible Insertion Points for insertAdjacentHTML

For cosmetic reasons it was inserted before the <a> tag started (Before Begin). The numbers seemed to make more sense if they occurred before the link rather than after the link. Also, if it was inserted after the <a> tag started, the numbers would be underlined if the hyperlinks were underlined. To put it simply, it looked the best when the numbers were inserted before the <a> tag started. Here is what the HTML of a simple web page would look like before the insertion of numbers (see Figure 2 for screen shot):

```
<HTML>
<HEAD>
<TITLE>Display</TITLE>
</HEAD>
<BODY>
<H2>Display</H2>
<P>Click <A href="http://www.cs.uregina.ca/~totten"
        target="_parent">here</A> to start.
</P>
</BODY>
</HTML>
```

Here is what the HTML would look like after the insertion of numbers (see Figure 3 for screen shot):

```
<HTML>
<HEAD>
<TITLE>Display</TITLE>
</HEAD>
<BODY>
<H2>Display</H2>
<P>Click <FONT color=red size=2>17</FONT><A target=_parent
        href="http://www.cs.uregina.ca/~totten">here</A> to start.
</P>
</BODY>
</HTML>
```

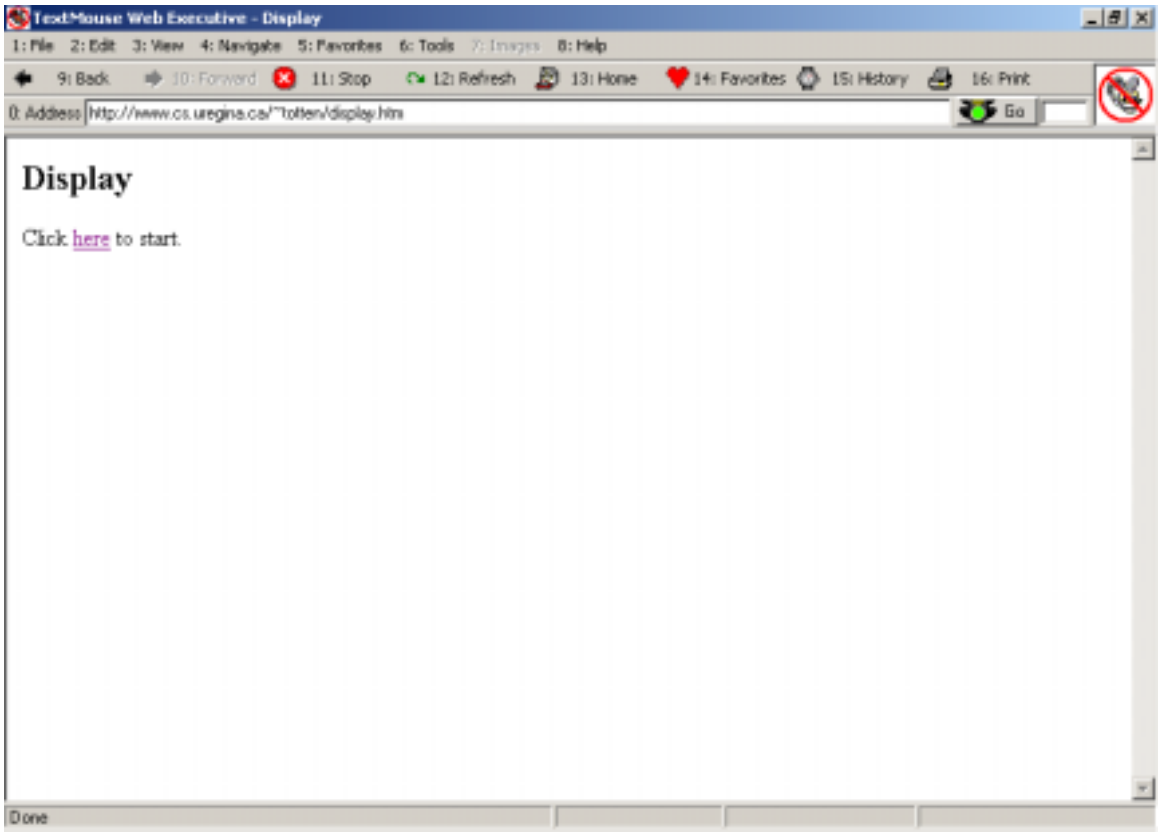

Figure 2: Before Number Insertion

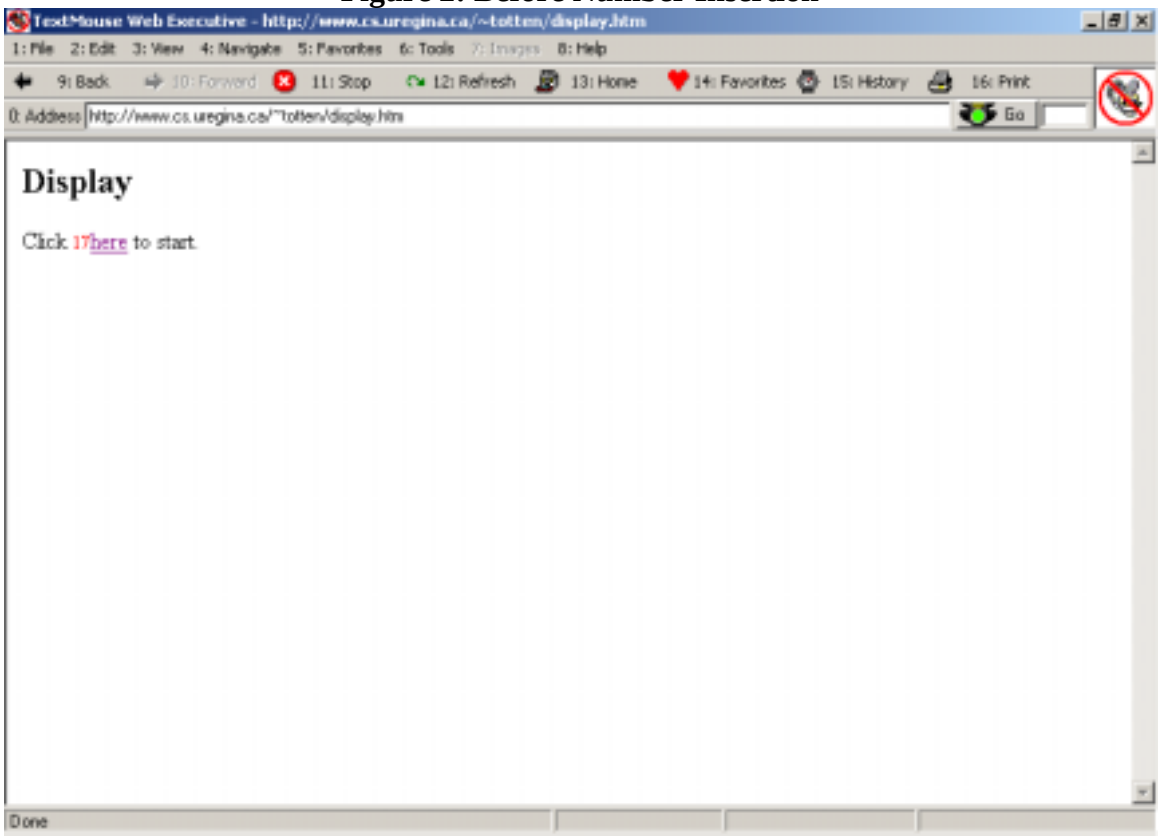

Figure 3: After Number Insertion

The idea of numbering the hyperlinks was fairly simple to begin with. There is a loop to iterate through the "links" collection and insert the appropriate number using the "insertAdjacentHTML" method. However, frames presented a problem because the hyperlinks that appear on the web browser are not found in the document that the web browser is told to display. A web page with frames is a web page that is divided up into two or more separate documents. Each individual document, or "frame," will have its own scroll bar (if necessary), and the content in each frame is specified in a different document. All that is in the document specified by the URL is information about how big each of the frames is and where to find the documents to display in each frame. This is an example of the HTML from a web page with frames (see Figure 5 for screen shot):

```
<html>
<head>
<title>Frame Example 1</title>
</head>
<frameset cols="20%, 80%">//split the web browser into two columns
<frame src="frame1.html">   //indicates what to display in the first frame
<frameset rows="20%, 80%">          //split this frame into two rows
<frame src="frame2.html">   //indicates what to display in the second frame
<frameset cols="50%, 50%">//split this frame into two columns
<frame src="frame3.html">   //indicates what to display in the third frame
<frame src="frame4.html">   //indicates what to display in the fourth frame
</frameset>
</frameset>
</html>
```

Each of the <frame> tags specifies which document to display in that frame. The <frameset> tags specify the number and size of the frames. For example, the line "<frameset cols="20%, 80%">" splits the screen vertically into two frames, the first frame taking up 20% of the screen and the second taking up the remaining 80%. One could split the screen into any number of frames with the <frameset> tag, as long as the percentages add up to one hundred. It is easy to see from this example that the "links" collection for this document will not be of any use in numbering the hyperlinks that the user will see. In the HTML for "http://www.cs.uregina.ca/~totten/frameExOne.html," (shown above) there were no hyperlinks specified, yet there are four hyperlinks on the web page (seen in Figure 5). The text and hyperlinks seen here come from four other documents called "frame1.html," "frame2.html," "frame3.html," and "frame4.html," that are referenced in the <frame> tags in the above HTML. Fortunately there is a collection of "frames" for the "Document" object, which allows access to these documents. The access that you can get to the documents displayed on a frames page through the "frames" collection of the "Document" object is the same access that you can get to a regular web page through the "Document" object. This may seem a little bit confusing. On a regular page, the "Document" object provides access to all the hyperlinks, images, forms, etc. that the user sees. On a frames page, the hyperlinks, images, forms, etc. that the user sees are contained in several separate documents. These documents are accessed

through the “frames” collection of the “Document” object. The “frames” collection actually returns an array of “Document” objects.

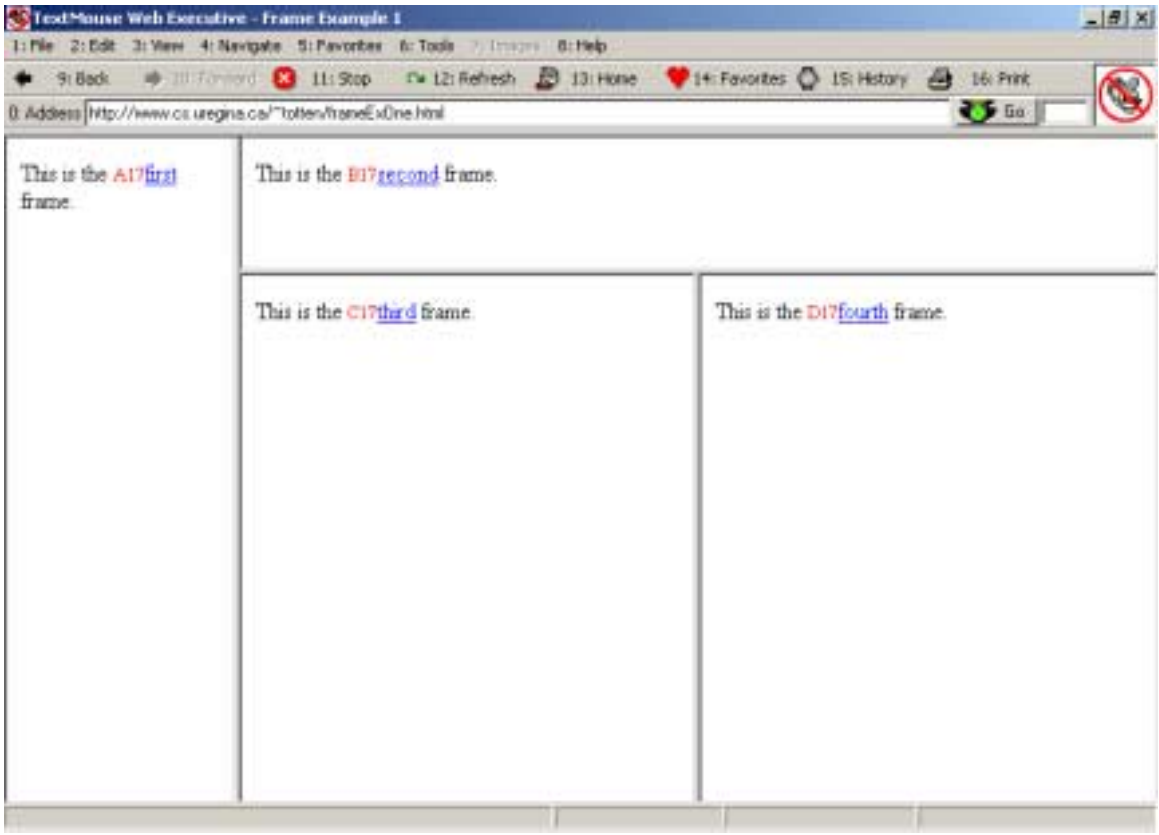

Figure 5: A Simple Web Page with Frames

In the final product, the numbering of links is accomplished in the “updatelinks” function. First this function checks to see whether or not the document has frames in it. If it does, it adds a letter (indicating the frame it is in) to the number that is inserted in the document. That is, all links in the first frame will be numbered “A17, A18, . . .,” all links in the second frame will be numbered “B17, B18, . . .,” and so on. The only additional code necessary to number frames like this is a loop to iterate through the frames and a string variable to keep track of what letter to insert. If there are no frames, it just numbers the document without the letters, as described above. The example frames web page in Figure 6 has no hyperlinks in the first frame (the part that says “www.hawking.org.uk” in large letters and has the three pictures at the top), so the first numbers start with “B.”

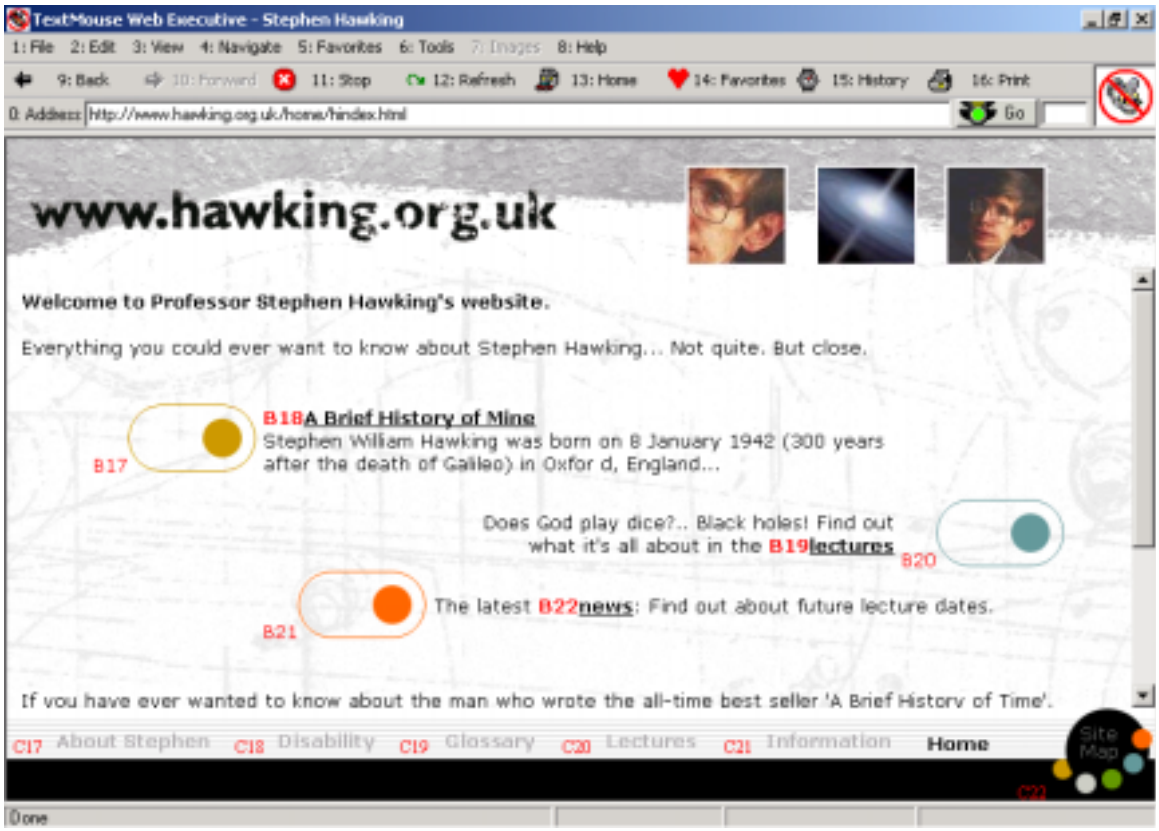

Figure 6: Another Example of Frames

Iframes are different from frames in their looks as well as in their HTML, so they have to be handled a little bit differently. Regular frames split up the entire web page into several individual frames. On a web page with regular frames, the HTML for that web page specifies only where to find the content and how to display the frames. There is no actual content in the HTML for a regular frames page; all the content is stored in separate documents. An <iframe> tag can be used to insert a sort of "floating frame" into a regular web page. This means that while the HTML for that web page holds some of the content, there is also another document that specifies the content that will be displayed in the iframe of the web page.

To handle iframes, "updatelinks" numbers the frames as usual (the "frames" collection sees regular frames and iframes the same), and then it checks to see if there are any hyperlinks in the regular part of the document to number. If there are, then the document contains iframes and not frames. If there are no hyperlinks in the regular part of the document, then it will be treated just like frames, even though the frames may be iframes. Here is the source code and a screen shot (Figure 7) of an example web page with an iframe:

```
<HTML>
<head>
<title>frames</title>
</head>
<body>
<h1>IFrame Example</h1>
```

```
<p>Click <a href="http://www.cs.uregina.ca">here</a> to go to the U of R CS
        Dept website.
<iframe src="display.htm" width="350" height="200" align="center">
</body>
</HTML>
```

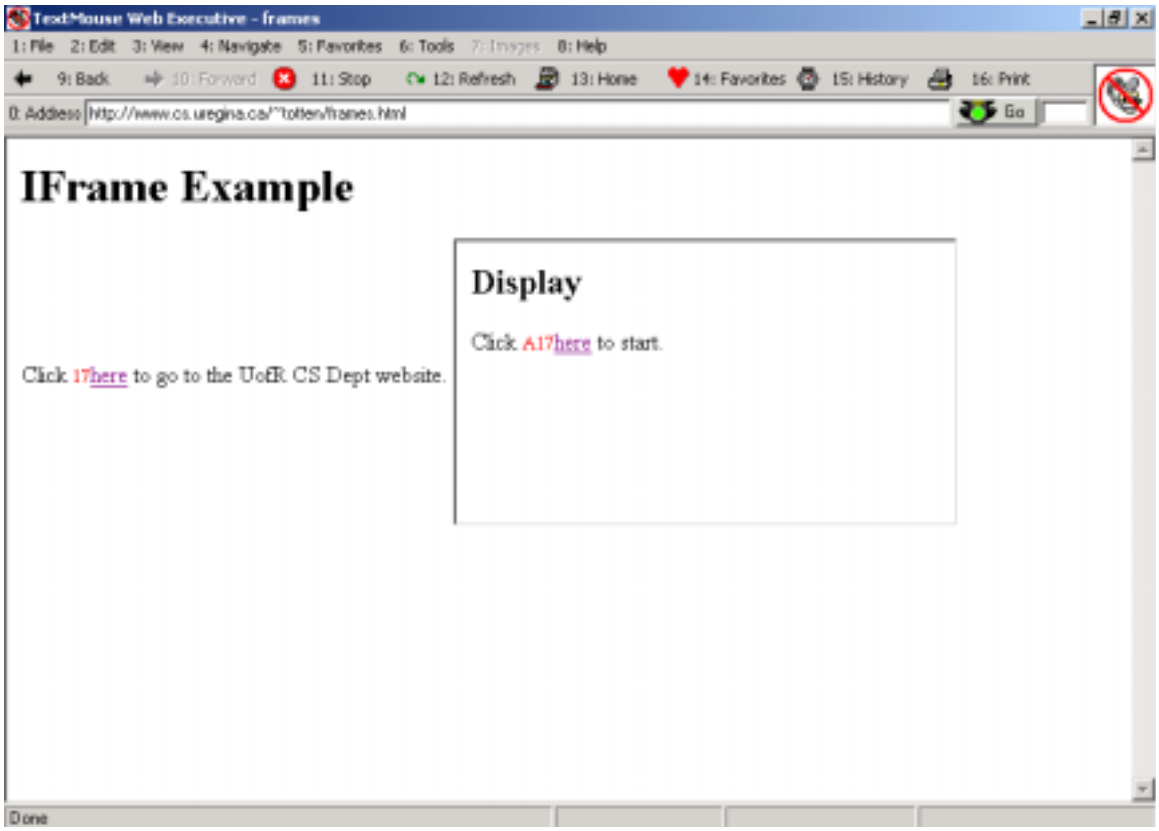

Figure 7: An Iframe Example

Notice that the HTML for the iframe (the part that says “Display” and “Click here to start”) is not contained within the HTML for this web page; rather, the iframe tag contains a reference to the document in which that HTML is stored.

In implementing frames, an intimidating problem was encountered. Regular frames and iframes presented many problems on their own and took a long time to complete. Nested frames appeared even more difficult. Nested frames are frames within frames. For example on a frames page, if one of the documents specifying content for a frame was a frames document itself, then this would be nested frames. Figures 8 and 9 show a comparison between a regular frames web page and a nested frames web page.

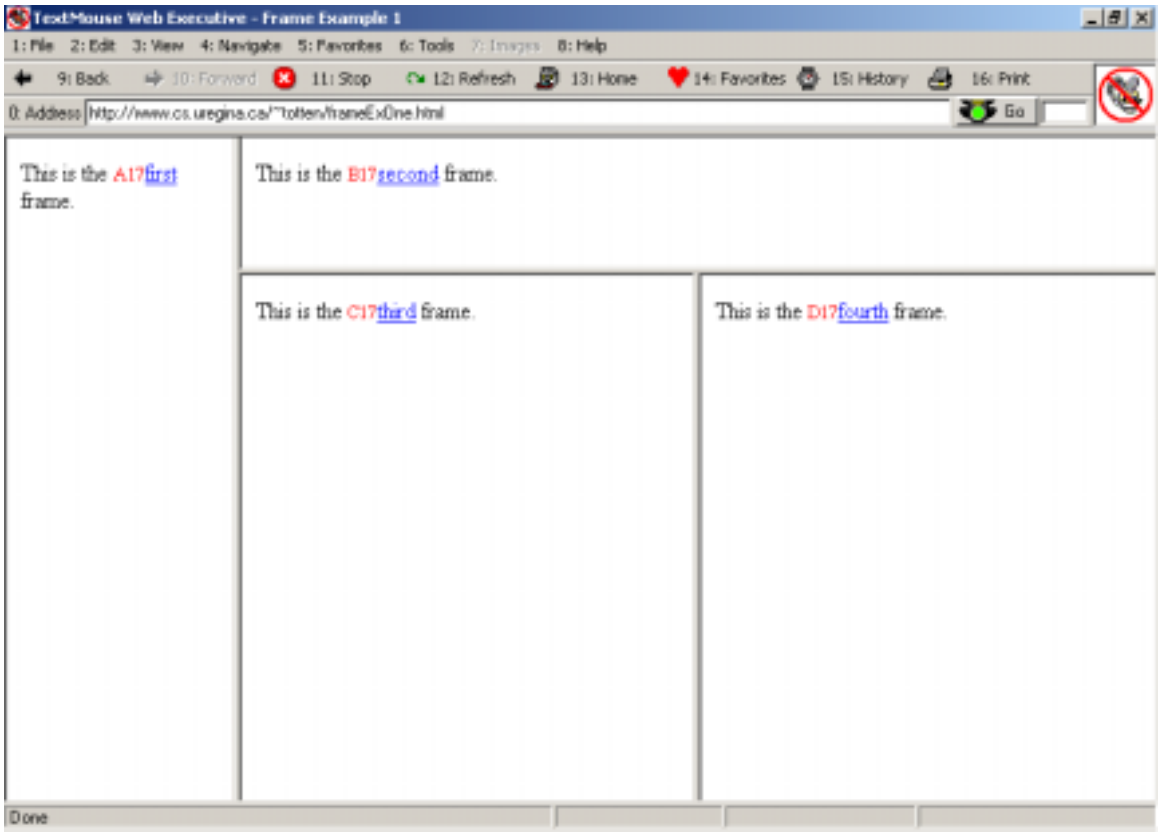

Figure 8: Regular Frames

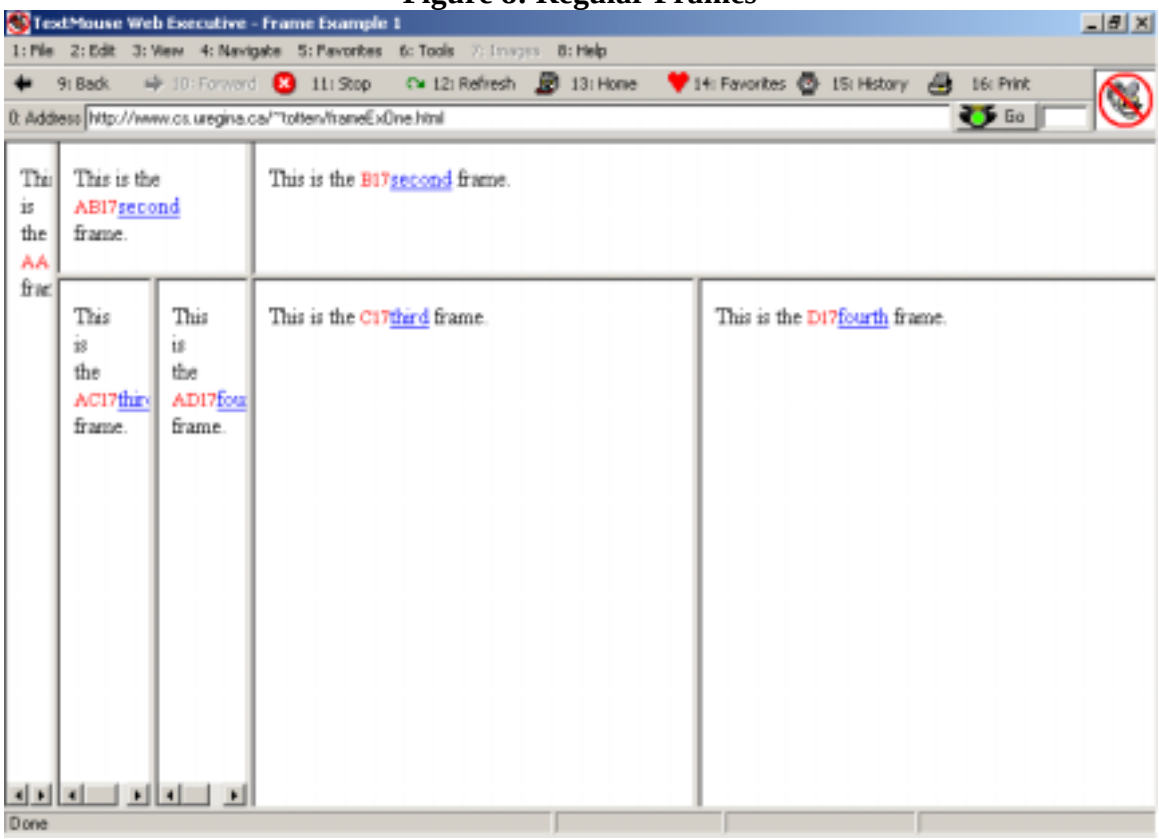

Figure 9: Nested Frames

Where it says, “This is the first frame” in Figure 8, there is a frames page in Figure 9, with four frames, showing exactly what was on the whole browser in Figure 8. There are frames within the first frame on the web page in Figure 9, so this site has nested frames. When it was discovered that nested frames would be a problem, it was decided to not allow nested frames at all. Internet Explorer allows one level of nested frames, but not two levels. It did not seem that it would make that much of a difference if TextMouse Web Executive did not allow any frames. So, to begin with, it did not. If there were frames on a page, it checked for nested frames in the “updatelinks” function. This was done the same way that the check for frames on a regular document was done. On a regular web page, the “length” property of the “frames” collection of the “Document” object is checked. If it is greater than zero, then frames exist on that web page. For frames, each frame’s “Document” object’s “frames” collection’s “length” property was checked through the “frames” collection of the “Document” object for the web page. If any nested frames were found, the browser would just navigate to the site of the frames that were nested.

This decision to disallow nested frames put serious limitations on the TextMouse Web Executive. Some web pages are designed to have nested frames; these web pages were not seen as they were intended to be seen. In fact, sites that are designed with nested frames ended up not making a lot of sense, not looking very nice, and not giving the user the information he or she was looking for. For example, Figures 10 and 11 show a comparison between Internet Explorer (which allows nested frames) and TextMouse Web Executive (which did not at this time allow nested frames), both displaying U2.com. Notice that in Figure 11 (TextMouse Web Executive) the top navigation bar and the bottom timeline are both gone. This is because the middle frame has frames in it (i.e. nested frames), so it navigated to the middle frame. The nested frame is easy to see because some of the numbers in Figure 11 start with “A”; this means that the part with the “A”s is an iframe.

Figure 10: Internet Explorer (allows nested frames)

Figure 11: TextMouse Web Executive (did not allow frames)

Because of the inconvenience of not allowing nested frames at all, it was decided to make the change to allow one level of nested frames. Nested frames were one of the last things to be implemented. Small bits of code were added in where necessary to handle them. To label the hyperlinks on nested frames two letters and a number were used. The first letter indicated the top level of frames and the second letter indicated the second level of frames. The numbers started at seventeen just as for a regular web page. Figure 12 shows what U2.com looks like on TextMouse Web Executive now.

Figure 12: TextMouse Web Executive (allows nested frames)

Notice that the top navigation bar and the bottom timeline are displayed and numbered. Also, the iframe within the middle frame (labeled with “C”s) on the right has the letters “CA” prefixing all the numbers. These letters indicate that it is the first subframe in the third frame in the document. The only difference between numbering regular frames and numbering nested frames is that for nested frames there is an extra “.frames([framenumber]).Document” added onto the end of the statements, as these code fragments show:

```
‘this is the statement that inserts the numbers for regular frames
MyBrowser.Document.frames(x).Document.links(y).insertAdjacentHTML("Befo
reBegin", "<FONT color=" & indexColor & " size=2>" & currletter & prevlinks +
17 & "</FONT>")
```

```
'this is the statement that inserts the numbers for nested frames
MyBrowser.Document.frames(x).Document.frames(y).Document.links(z).insert
AdjacentHTML("BeforeBegin", "<FONT color=" & indexColor & " size=2>" &
currletter & subletter & prevlinks + 17 & "</FONT>")
```

Image maps presented a small problem. An image map is a single image that is split up into several different "clickable" areas. Each different "clickable" area has its own <area> tag that is within one <map> tag for the image. When the image is inserted with an <img> tag, it specifies the label of the map that is to be used. The problem was that the HTML could only be inserted beside the area elements that made up the image map, not actually on top of the image. The numbers usually appeared right before or just after the image. Unfortunately there were no spaces in between the numbers, so it was difficult to tell what the numbers were supposed to be. Also, and more importantly, there was no way to tell which number corresponded to which area on the image map. Here is the HTML representing the image map portion of the Faculty of Science web page (Figure 13) after the numbers have been inserted:

```
<P align=center>
<MAP name=FPMap0>
<FONT color=red size=2>17</FONT><AREA shape=RECT
coords=0,103,148,207 href="biology/index.htm">
<FONT color=red size=2>18</FONT><AREA shape=RECT
coords=0,207,150,325 href="http://www.phys.uregina.ca/">
<FONT color=red size=2>19</FONT><AREA shape=RECT
coords=150,103,275,207 href="http://www.cs.uregina.ca/">
<FONT color=red size=2>20</FONT><AREA shape=RECT
coords=149,206,320,325 href="geology/index.htm">
<FONT color=red size=2>21</FONT><AREA shape=RECT
coords=276,103,476,206 href="http://www.chem.uregina.ca/">
<FONT color=red size=2>22</FONT><AREA shape=RECT
coords=320,203,477,325 href="http://www.math.uregina.ca/">
</MAP>
<IMG height=331 alt="Science Navigation Map" src="images/science3.jpg"
width=483 useMap=#FPMap0 border=0 rectangle="(320,203) (477,325)
http://www.math.uregina.ca/">
</P>
```

Figure 13: Image Maps Problem

The part in between the <map> tags is the part that tells the shape and position of the areas and which URL to navigate to if that area of the web page is clicked. The <img> tag after the <map> tags is the part that actually displays the picture. It is easy to see which number corresponds to which URL from the HTML, but as you can see in Figure 13, the numbers corresponding to the image map (171819202122) on the web browser (to the left at the bottom of the image) are very confusing. Rather than making the web page look strange with the numbers for the image maps sitting below them, apparently one very long number for some hyperlink that you could not see, the area elements were not numbered at all. Instead, a menu item was added called “Images” that, when clicked, generates a list of the area elements on that page (Figure 14). To make it easier to understand (some URL’s do not contain any words that would indicate what the web page is about), the “alt” property was incorporated if it was available. The “alt” property is an optional part of an <img> or <area> tag that is displayed if the image cannot be displayed. Its purpose is to tell the user what the picture is about. The “alt” property can be very useful in identifying where a hyperlink will lead to; however, it cannot always be relied upon, as it is an optional property. From this list of URL’s (and “alt” tags where possible), the user can use the arrow keys (or the mouse) to highlight the item they wish to see and press Enter (or double click) to select it.

Figure 14: Image Maps Display

One last problem encountered relating to inserting numbers was that text boxes, drop down menus, check boxes and buttons were not getting numbered. This is because they are not included in the "links" collection. Another collection of the "Document" object called "forms" was used to solve this problem. In "updatelinks" another function, "numberform" is called. In "numberform" there is a loop to iterate through all the forms on a page that numbers each element in that form (excluding the "hidden" form elements), starting from the last number that was used in the numbering for the hyperlinks on the page.

4.2 Processing the Keystrokes

Once all the items on a page were numbered, something needed to be done with the numbers. Before the keystrokes could be processed, they needed to be intercepted. TextMouse Web Executive went through a series of attempts at intercepting the keystrokes. It started out with a text box in one corner of the form that accepted input. This was great for catching keystrokes, but it was not so great if the user wanted to scroll in the web browser. If the user wanted to scroll in the web browser, he or she would have to use the mouse to click on it. Then when he or she wanted to pick another link, the user would have to click in the text box so that the program could process his or her input. Aside from being inconvenient, this required the use of the mouse. This was obviously not the best way to handle scrolling on the browser. The next attempt consisted of catching the arrow keys from the text box and then sending them to the browser in order

to simulate scrolling. Unfortunately attempting to do this would occasionally cause the keyboard to freeze up.

Much time was spent on this problem, with little success until the "keypreview" property of the form was discovered. When the form loaded, the "keypreview" property of the main web browser form got set to true. This let the "Form_KeyDown," "Form_KeyPress," and "Form_KeyUp" events catch all keystrokes from the user before any control on the form caught them. With this property, focus could be on the web browser to enable scrolling, but the form would also catch the keys that would indicate hyperlinks, menus, and buttons. The only problem with this version was that on some web pages, problems with focus on the web browser caused the "Form_KeyDown" event to not recognize Enter being pressed.

The final version of the browser has a textbox in the upper right corner. When the browser loads, focus is on the browser. The user may use the direction keys to scroll. When the user wants to navigate to a link, he or she may simply input the number (or letter and number combination). Focus is automatically transferred up to the textbox. This allows the user to see what they are typing. When focus is in the text box, if the user presses a direction key, focus is transferred back to the browser. In other words, focus is determined by the keys that the user presses, even though the user doesn't necessarily know what is going on.

Processing the keystrokes was the most difficult part of this project. The simple part is keeping focus where it was relevant, either on the textbox or on the web browser; the tough part is doing something with those keystrokes when the user presses Enter. The keystrokes are caught in the "KeyLabel_KeyDown" and "Form_KeyDown" events and the "gotolink," "calclink," "calcindex," and "ClickLink" functions are all involved in the processing of the keys.

The "Form_KeyPress" event simply watches for letters and numbers being pressed. If it catches a letter or a number, it will transfer focus to the textbox. The other part of the "Form_KeyPress" event has to do with a variable called "textbox," which is discussed later.

The "KeyLabel_KeyDown" event translates all letters into upper case and also watches for the direction keys, which would indicate that a transfer of focus is necessary.

When focus is on the textbox, all editing options are available except the arrow keys. This is because it is assumed that the user will use the arrow keys to scroll on a web page. If the user presses any direction key, focus is automatically transferred to the browser. Also, the input for the textbox is generally very short, and if a number must be changed, the user should use the Backspace key. The user must be careful when using the Backspace key. Some keys have a default action, and there was no way to stop this default action from occurring. Backspace is a key with a default action; it navigates backward in the history list. If focus is on the web browser, it will navigate backward in the history list when Backspace is pressed; if focus is on the textbox, it will remove one character from the end of the text in the textbox.

When the user presses Enter, it is caught in the "KeyLabel_KeyDown" event. Then control is passed to another function called "gotolink." The "gotolink" function first checks to see if the user pressed two letters and no numbers, which would indicate a transfer of focus to a nested frame. The code to handle nested frames was written after this function was already completed, and there was no convenient or logical place to put

this part, so that is why it is at the beginning. Next it checks to see if the keys pressed indicate a menu or button across the top. For these it is just a matter of calling the correct button's "click" event or sending the keys to make the correct menu drop down. Then it checks to see if the key the user pressed was a single letter. If this is the case, it sets focus to the corresponding frame, if it exists. The other cases are split up through "if" statements. They are divided into frames, iframes, and regular web pages.

To handle frames, the characters stored in keys are split up into a letter and a number (the letter being the first character and the number being the rest of the string). First, it checks to see if the second character is also a letter, indicating nested frames. If it is, the index of the nested frame is stored in a different string variable, and it calls "ClickLink" for the nested frames. If it is not, then it calls "ClickLink" for the regular frames. Similar to the "insertAdjacentHTML" calls for frames and nested frames, the only difference between the two "ClickLink" calls is that ".frames([index]).Document" is added on the end for nested frames. The code shows this clearly:

```
'if frames
If frames Then
        'assign temp to the number part of the input
        temp = Mid(KeyLabel.Text, 2)
        'assign framenum to the letter part of the input
        temp_letter = Mid(KeyLabel.Text, 1, 1)
        framenum = Asc(temp_letter) - 65
        'if link is in one of the 26 possible frames
        If temp_letter = "A" Or . . .Or temp_letter = "Z" Then
                subletter = Mid(temp, 1, 1)
            'if the second character in the input is a letter, then nested frames
            If subletter = "A" Or . . .Or subletter = "Z" Then
                'NESTED FRAMES
                    temp = Mid(KeyLabel.Text, 3)
                    thislink = Val(temp) - 17
                    subletter = Asc(subletter) - 65
                'click on the link
                success=ClickLink(MyBrowser.Document.frames(framenu
                    m)Document.frames(subletter).Document,
                    thislink)
                If Not success Then GoTo NoLink
            'else, REGULAR FRAMES
            Else
                thislink = Val(temp) - 17
                'click on the link
                success=ClickLink(MyBrowser.Document.frames(framenu
                        m).Document, thislink)
                If Not success Then GoTo NoLink
            End If
        'else the link is not in one of the 26 possible frames, so it is an error
         Else
```

```
                GoTo NoLink          'NoLink is the error handler for this function
            End If
        End If
```

To handle no frames at all was very easy. It simply calls the ClickLink function with the appropriate parameters.

To handle iframes, it first checks to see if it is a letter number combination (in which case it will execute exactly like frames) or if it is just a number (in which case it will execute exactly like no frames at all).

The "ClickLink" function is a generic function that is called for regular frames, nested frames, iframes, or just a regular web page. It is passed two parameters. The first one is the document in which the link should be found; the second parameter is the index of the desired link. This function does not need to bother with frames and subframes because the document that gets passed in will be the whole document (for a regular page), a single frame (for a frames or iframes page), or a single nested frame (for a nested frames page), depending on where the link occurs. First it uses a function called "calclink" to find the correct index of the link in the "links" collection. The "calclink" function is necessary because of the area elements (which are included in the links collection) that were just ignored while inserting numbers into the HTML. Then "ClickLink" checks to see if the calculated index is within the range of links in the given document. If it is, then its "click" method is used to simulate a mouse click on that link, and the function returns true. If the calculated index is not within the range of links on that frame, then it gets checked to see if it is within the range of form elements in the given document. Another function called "calcindex" is used to calculate the correct index of the element in the "elements" collection of the form because the "hidden" elements were ignored when numbering them. If it is within the range of form elements, then it takes appropriate action. If it is a radio button or a check box, then it gets "clicked". If it is a drop down menu, focus is set on it and pressing "F4" is simulated to force the menu to drop down. If it is anything else (a text box, a button, etc.), focus is set on it. If it is not within the range of form elements, it checks if either the history list or the favorites list is visible. If they are not visible and it has made it this far, then it is an error. If one of them is visible, then it checks if the number is within the range to indicate an item on the history or favorites list (depending on which one is visible). If it is, it sets focus on it, and if it is not then it is an error.

In the "Form_KeyDown" event, there is some code about a variable called "textbox." It gets set at the very beginning of the subroutine, and then as soon as it gets set, it gets checked. It seems rather pointless to set a variable in the lines right before it is checked. Why do this if "textbox" is not used in any other subroutines or functions? The reason why this is necessary is because it has to check for focus on a text box in each different kind of document (frames, nested frames, iframes, and no frames) differently. The code is actually fairly complex for simply setting a variable. The "textbox" variable is only true when focus is on a textbox in the browser. "Textbox" stops the subroutine from checking for letters and numbers and resetting focus on the textbox. For example, if you were searching for some information about databases on the University of Regina web site, you might go to the search page and press 22 to get to the text box to start your search, as shown in Figure 15. Figure 16 shows how it would look if the "textbox"

variable was not used. Notice that without the “textbox” variable (Figure 16), what the user types does not go in the “Search” textbox. Rather, it goes in the textbox in the upper right hand corner of the web browser. If “textbox” is true, the subroutine only checks for the Escape key. The Escape key will get the user out of the text box. All other keys the user presses are ignored. If “textbox” is false, as soon as the user presses a letter or number, focus will be transferred to the textbox in the upper right hand corner of the web browser and anything the user types will appear here instead of in the desired textbox on the web browser.

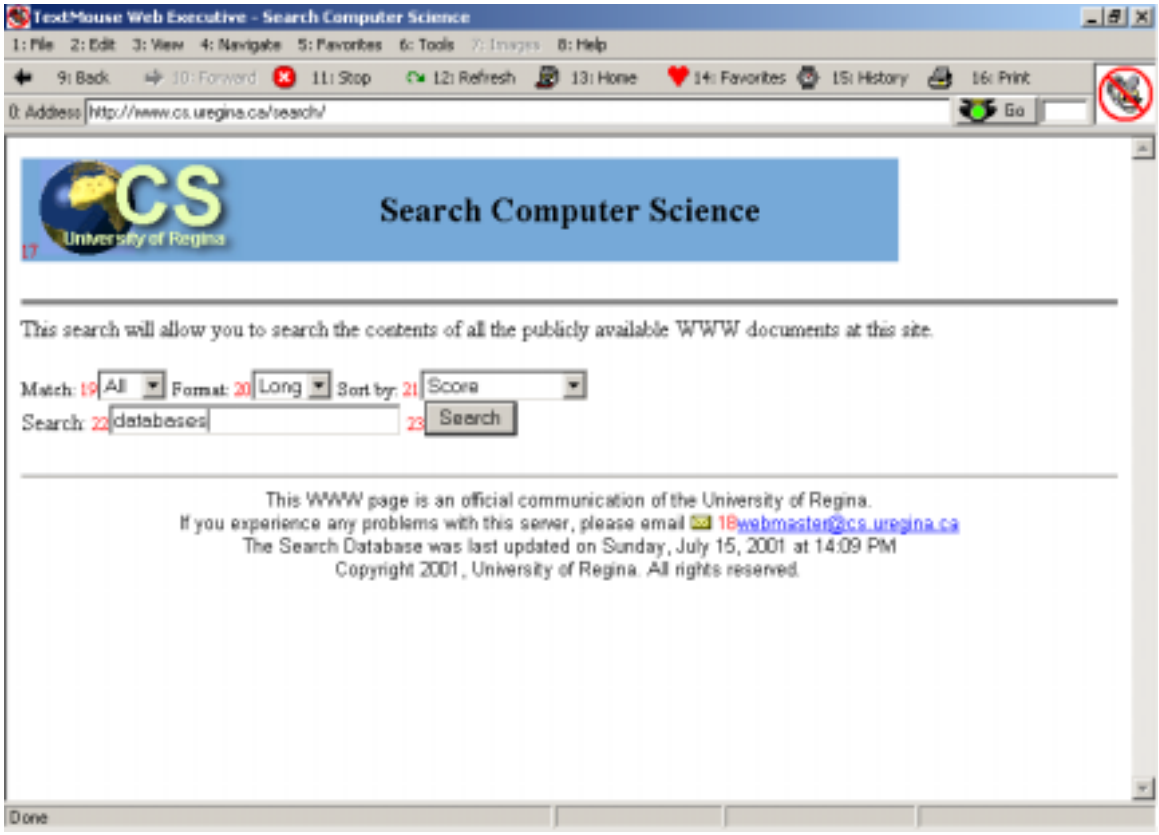

Figure 15: Typing in a Text Box (textbox variable active)

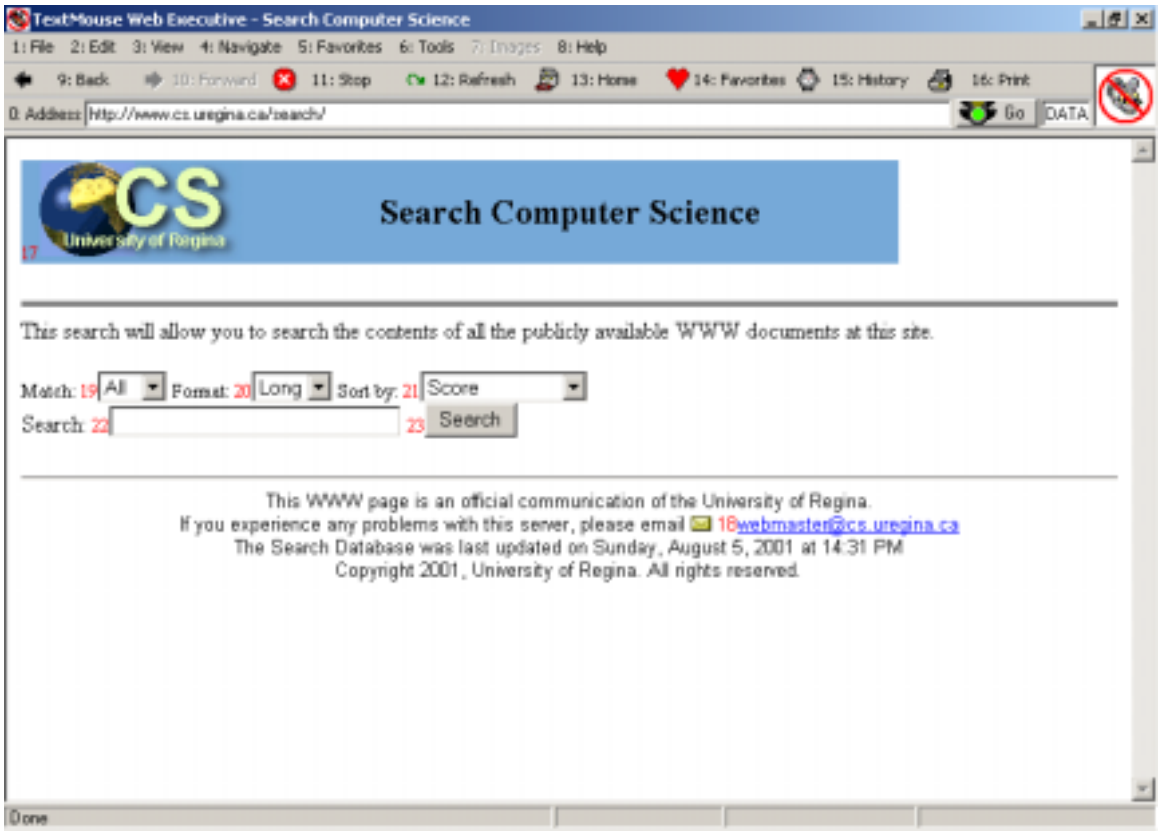

Figure 16: Typing in a Text Box (textbox variable inactive)

4.3 Problems

There were some major problems encountered during the development of TextMouse Web Executive. They are discussed below, along with their solutions and implications for how the project was implemented.

There are several “MyBrowser_DocumentComplete” events fired for a document containing frames, including one for each frame and one for the whole document. This event passes in a parameter that can be checked to be sure when it is the whole page that has finished downloading, not just one frame. Initially, the function checked for the whole page being done before it called “updatelinks” to number the document. Because of some complicated web pages, it now calls “updatelinks” from the “MyBrowser_DocumentComplete” event, whether it is the whole document that is finished or just one frame. Then “updatelinks” calls “checkfornumbers” to make sure nothing gets numbered more than once.

In an early version of TextMouse Web Executive, the “click” method was not used on the <a> elements. Instead, the navigate method of the web browser was used to navigate to the URL specified in the <a> element. This worked fine in most cases. However, there were some links that did not actually do anything unless they were clicked on. This was because they had some code in an “onclick” event and an “href” property that did not go anywhere. Using the “click” method of the links solved this problem; the links actually did what they were supposed to. Unfortunately this success was short lived.

The web browser was working great until an old problem surfaced again. Some links do not actually go to a new page; they just go to a different part on the same page. When the "click" method was used on the links, and a link just went to a different part of the same page, it would double up the numbers on that page. This is because it would fire the events that included the code to call the function to number the links if you "clicked" a link, even if it went to a different part of the same page. Initially this problem was solved by keeping track of the web page that the web browser was on. If a link went to a different part on the same page, the URL would be exactly the same, followed by a "#" and a label. Before it called "updatelinks," it checked to see if the web page that the browser was navigating to was the same with a "#" on the end. This extra checking was eventually abandoned because, as mentioned above, in the final version it renumbered the web page on every "MyBrowser_DocumentComplete" event, always checking through the document for existing numbers before inserting them to make sure that the numbers never get doubled up.

There was also a problem with renumbering when the user hit the refresh button. There was some sort of a timing issue that made it renumber the links first, and refresh the document second (i.e. it would double up the numbers on a page and then it would refresh the page, which would then have no numbers at all). This problem was fixed by introducing a variable called "refresh_bttn" that is set to true if the refresh button is pressed. Then in the "MyBrowser_DownloadComplete" event, if it is a no frames page, it calls "updatelinks." If it is a frames page, it gets renumbered in "MyBrowser_DocumentComplete," where an array called "done" keeps track of which frames have been numbered and which have not. We are still confused as to why it works like this and not any other way. Several different methods were attempted, but having frames handled in the "MyBrowser_DocumentComplete" event and having no frames handled in the "MyBrowser_DownloadComplete" event seemed to be the only way it would work.

After spending the time to incorporate text boxes, buttons, and other form elements into the numbering scheme, they seemed to be causing more problems than they were worth. When control was transferred to one of these form elements, the "Form_KeyDown" event watched for only the Escape key. When that was pressed, focus was set to the browser again to get the user out of the text box or button. Unfortunately this caused the "Form_KeyDown" event to not recognize when Enter was pressed until the user clicked on the browser with the mouse. This was a problem because when it was encountered, all of the key processing was done in the "Form_KeyDown" event because there was no textbox to accept input at this point in the project. This caused problems until the "activeElement" property of the "Document" object was found. This in combination with the "Screen.ActiveControl" property (used to make sure that the web browser is the active control) allowed the program to use a method called "blur" on whatever element had focus. Then when it reset focus to the browser, it could recognize and catch the Enter key with no more problems.

In order to reset the focus after a user has finished typing in a text box (either the address bar or a text box on a web page), there needed to be some signal to the program. The Escape key did not have any other use in this program and seemed logical. When this key is caught in the "Form_KeyDown" event, it takes focus off of the text box (if

applicable) and sets focus on the web browser. The Escape key is used to “escape” from a textbox or a dropdown menu on a web page, from the address bar, and from the menus.

Almost all of the “key catching” that is done in this program is in the “Form_KeyDown” and “KeyLabel_KeyDown” events. There is also a “Form_KeyPress” event, a “Form_KeyUp” event, a “KeyLabel_KeyPress event, and a “KeyLabel_KeyUp” event, which are used to a lesser extent. The “Form_KeyPress” event did not give the information that was needed, so it was not used. The “KeyLabel_KeyPress” event was used for preventing the computer from beeping when certain keys are pressed. The “Form_KeyUp” event handles the case if the user knows some of the Internet Explorer shortcut keys. The only one that could be intercepted was “F6,” which is used to refresh the web page. This keystroke needs to be caught in order to make sure that the web page gets renumbered. If this key is detected, then the subroutine will set the refresh button variable to true so that the web page will get renumbered. The “KeyLabel_KeyUp” event was not used. We are not sure that there is any reason why all of the key handling code could not go in the “Form_KeyUp” and “KeyLabel_KeyUp” events.

There was one error that in many situations that could not be prevented at all. In the “updatelinks” function “Access is Denied” errors kept occurring. The only way to alleviate this problem was to use “On Error Resume Next” within the “updatelinks” function. This works because this error only occurs when checking for something that is not needed for numbering that web page anyway (i.e. it might occur trying to access the “links” collection in an iframe that contains no hyperlinks). For example, on some web pages the following code causes an “Access is Denied” error:

```
For y = 0 To MyBrowser.Document.frames(x).Document.links.length - 1
        If Not (MyBrowser.Document.frames(x).Document.links(y).tagName =
                "AREA") Then
                Call
                        MyBrowser.Document.frames(x).Document.links(y).insert
                        AdjacentHTML ("BeforeBegin", "<FONT color=" &
                        indexColor & " size=2>" & currletter & prevlinks + 17 &
                        "</FONT>")
                prevlinks = prevlinks + 1
        End If
Next
```

For some reason, even though the frame (MyBrowser.Document.frames(x)) exists on the web page, it will not allow us to have access to its “Document” object. We are not sure if there is a better way to handle this problem or not.

As you can see in the screen shot of U2.com (Figure 12), the numbers (and letters) that are inserted into the HTML documents changes the way that they appear. Sometimes graphics get split up and what should look like a picture of something (or a navigation bar, as on U2.com) could end up looking like an incoherent jigsaw puzzle. It is a small price to pay for the functionality that TextMouse Web Executive provides.

5. Tutorial

Figure 17 shows what TextMouse Web Executive looks like. When it starts up, it will show a screen with the name and other information about TextMouse Web Executive, and then it will automatically display the user's homepage.

Across the top there are menus, which all have numbers in front of them. The first menu is the File menu. The first option on the File menu is "New Window." This opens up a new web browser. "Open" brings up a screen that allows the user to browse for a document to open in the web browser. To save a web page, choose the "Save As" option. It will bring up a screen that will allow you to choose the folder to save it in, as well as the extension and filename. "Close" closes the current window, while "Exit" closes all windows if more than one web browser is open. "Page Setup" allows the user to choose how the pages will be set up for printing. To print the web page, select "Print." It will bring up the familiar print screen with several print options. If the user wishes to see information about the document being displayed by the web browser, he or she should select the "Properties" option.

The second menu is the Edit menu. The options "Cut," "Copy," and "Paste" allow the user to cut, copy and paste to and from various locations. These functions use the system clipboard, allowing the user to cut, copy, and paste text to and from other applications as well as within TextMouse Web Executive. The "Select All" option will highlight the entire webpage. If the user wishes to search for some text on the web page, he or she should choose the "Find on Page" option. This brings up a screen that allows the user to type in the search text (Figure 18). Then it will find and highlight each occurrence of the text as the user presses the "Find Next" button or Enter.

Figure 17: TextMouse Web Executive

Figure 18: "Find on Page" from the Edit menu

The View menu has four options. The “Source” option starts an instance of Notepad and allows the user to look at the HTML source code for the web page (Figure 19). The second option, “Minimize,” minimizes the application, similar to pressing the minimize button in the upper right hand corner of the application. At all times only one of the last two options is enabled. If the window is maximized, then the “Restore” option will be available to the user. It will restore the window to its original size. If the web browser window is in its original size, the user may use the “Maximize” option to maximize it.

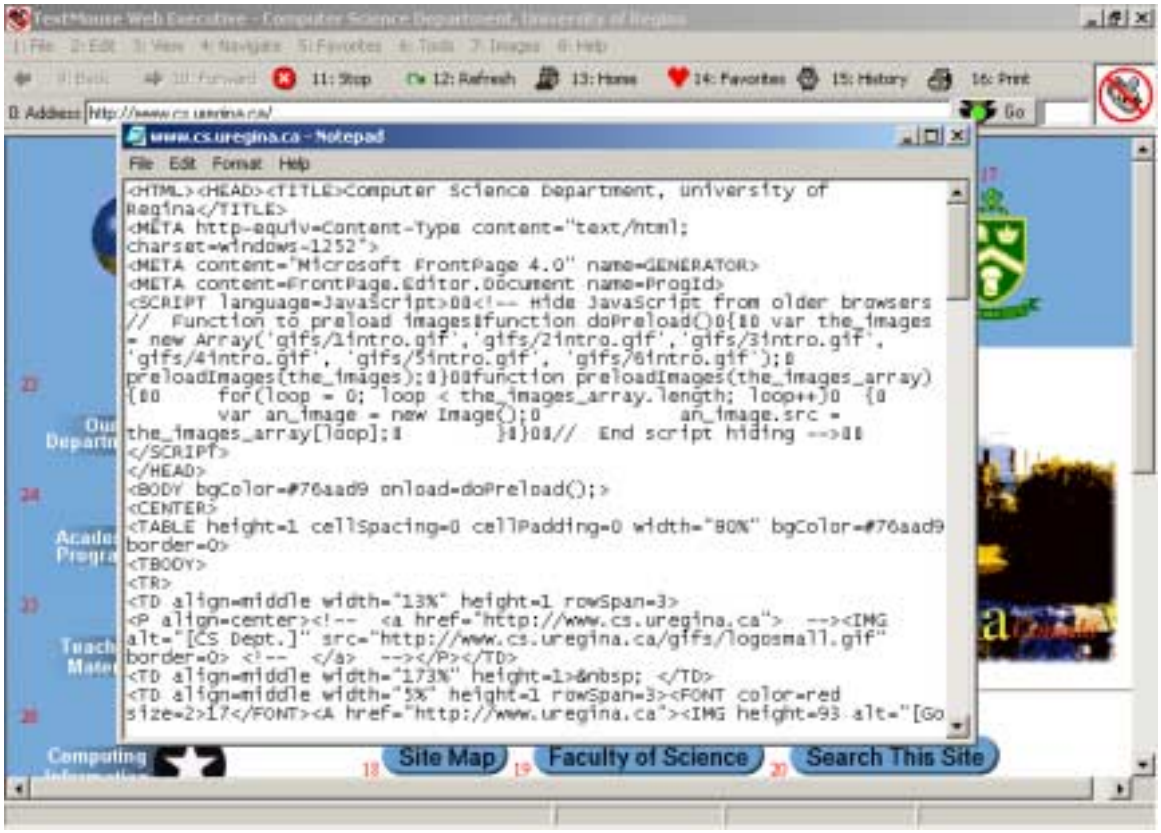

Figure 19: View Source

The Navigate menu is the next menu. “Back” will navigate one web page backward in the history list; “Forward” will navigate one forward in the history list. “Stop” will halt the current navigation. “Refresh” reloads the current web page. “Home” takes the user to his or her home page.

To add the current web page to your Favorites list, select the “Add” option from the Favorites menu. If you want, you may change the title that the web page is stored under. Otherwise, just press Enter when the “Add Favorites” screen pops up. “Organize” will bring up the “Organize Favorites” screen (Figure 20). The user may create folders, rename files and folders, move files and folders, and delete files and folders. Use caution when deleting files and folders. They are permanently deleted; they are not sent to the recycling bin. If the user wants to navigate to a site in his or her Favorites list, the “Go” option should be selected. This will bring up the Favorites list at the left hand side of the web browser, as shown in Figure 21. If you would like to

navigate to a web site that is in your Favorites list, press the number beside the appropriate item, and hit the Enter key. Once the desired item is highlighted, press Enter again. Alternatively, you may also use the mouse and double click on the correct list item. To get rid of the Favorites list at the side of the screen, either click on the "x" in the Favorites list, or press the number beside the "x" and Enter.

Figure 20: Organize Favorites

Figure 21: Favorites List

The "Tools" menu has three menu items. "Options" brings up the Internet Properties screen that allows the user to view and change various internet options, such as which web page is the home page (the web page that is loaded when the web browser starts up). "Change Index Color" brings up a color palette (Figure 22). After a color is chosen and the "OK" button is pressed, the browser will reload the current web page and the numbers (and letters) used to label the hyperlinks will now appear in the chosen color. The default color is red, so if the user wishes to see the numbers on a web page with a red background, he or she would have to change the color of them (Figure 23). If there is a check mark beside the "Clear Display on Lost Focus" option, that means that the textbox that you type in will automatically be cleared if you click on the web page with the mouse or if you begin to scroll by pressing a direction key. Selecting this option will toggle the check mark (this means that if it is not checked, clicking on it will make it checked and if it is checked, clicking on it will make it not checked).

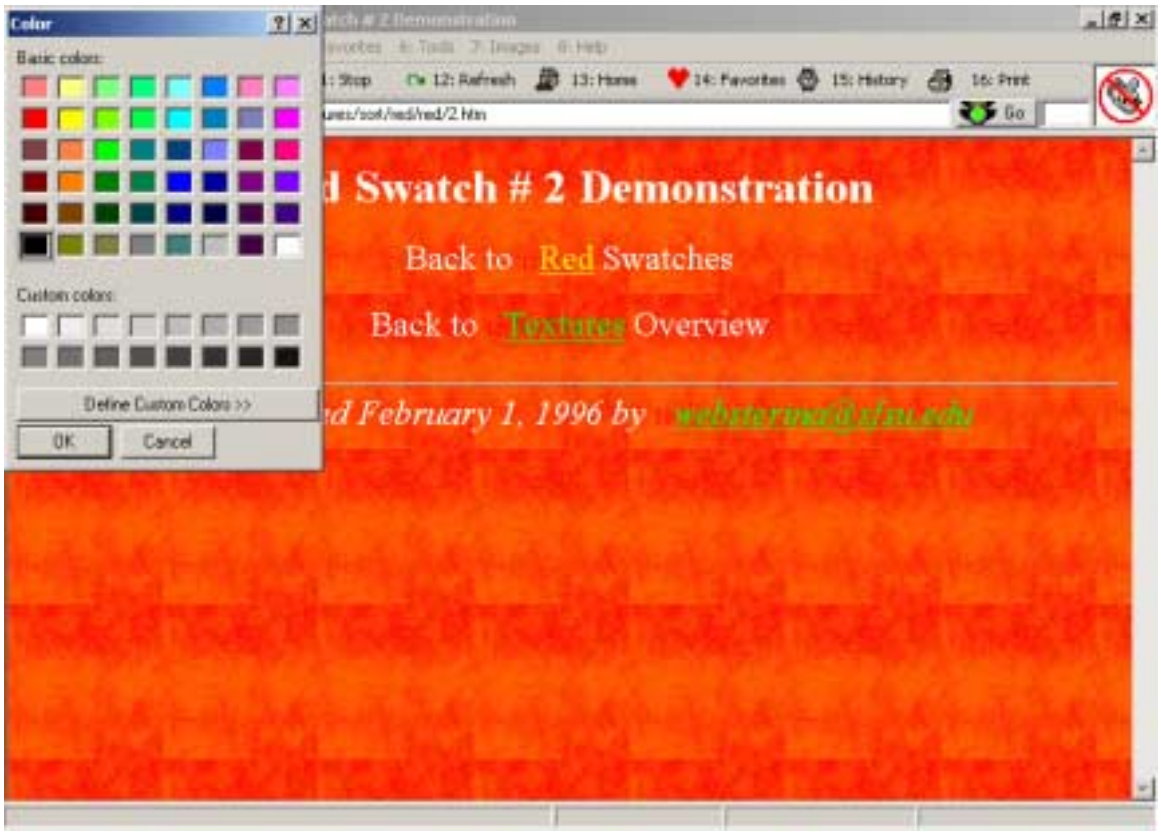

Figure 22: The Color Palatte

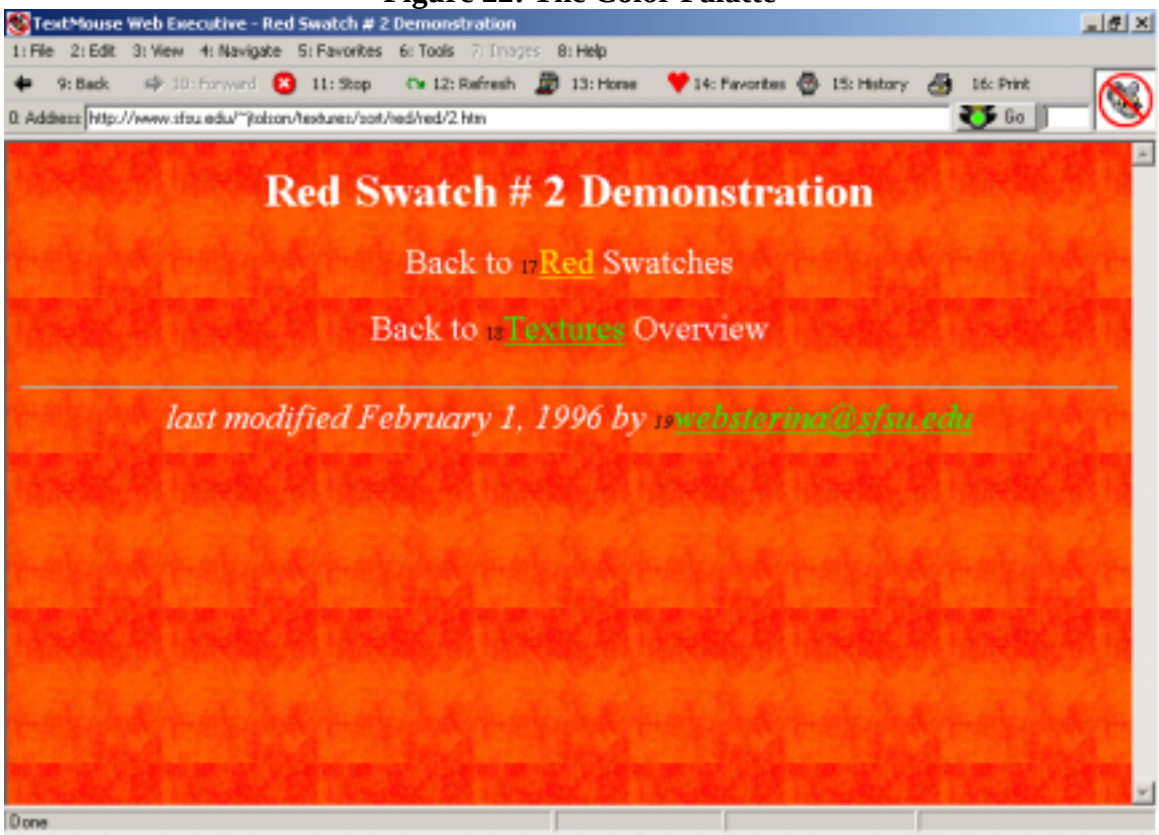

Figure 23: Black numbers on a Red Background

If the “Images” menu is enabled (i.e. the letters are black, not light gray), that means there is an image map on that web page. This means that there is an area on the web page that you can click on, but that could not be numbered. To view the web pages you can access from these areas, press 7 and Enter (Figure 14). To navigate to one of the URL’s listed as an image, the user may use the arrow keys to select the desired item, and then press Enter. Alternatively the user may use the mouse and double click on the desired item.

Under the Help menu, the “Contents and Index” option is still under construction. The “About TextMouse Web Executive” option brings up a screen displaying information about TextMouse Web Executive (Figure 24).

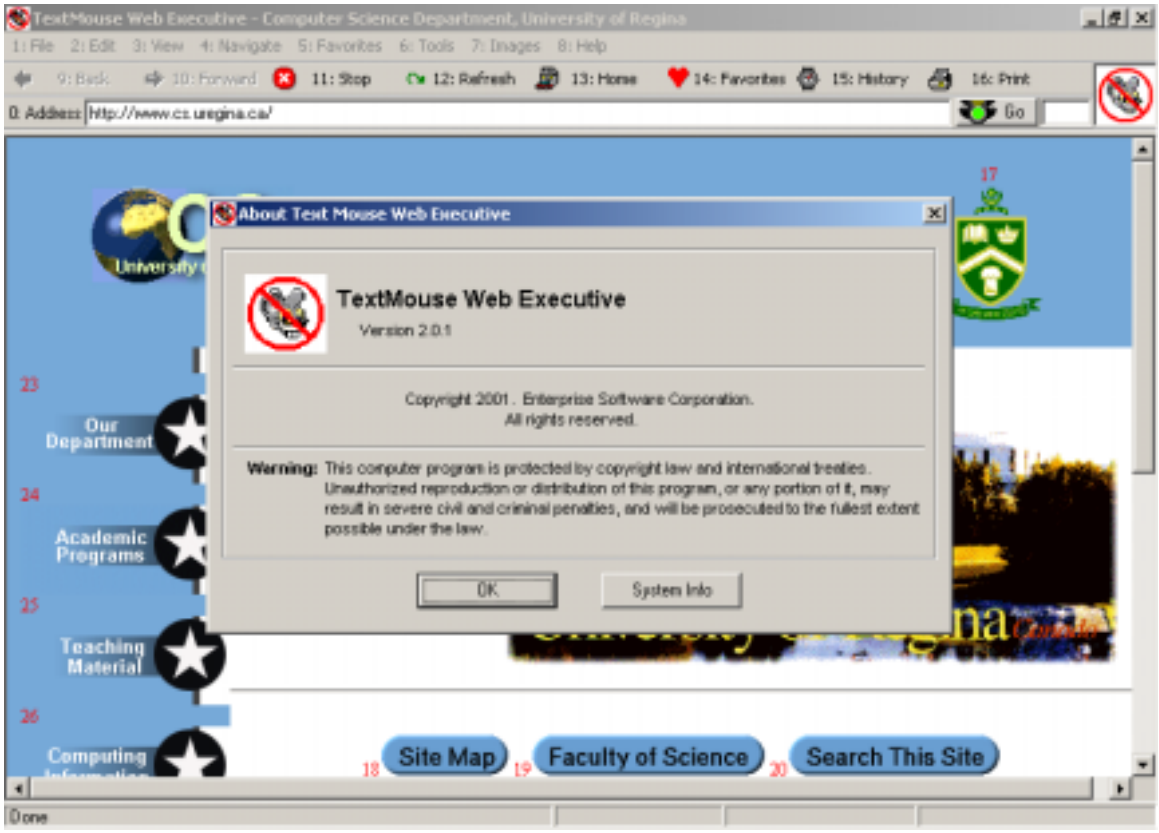

Figure 24: About TextMouse Web Executive

Underneath the menus there are buttons, which are also numbered. The Back button will navigate to the previous web page. The Forward button will navigate to the next web page. If the user wishes to navigate several web pages backwards or several web pages forwards, he or she can enter the number for the command (or press the button) once and then for subsequent navigations the user may just press Enter. The web browser essentially remembers that the last command was navigating forward (or backward) and it will execute that command when you press Enter until you choose to do something different. The next three buttons are Stop, Refresh, and Home, which are all similar to the commands under the Navigate menu. The Favorites button will bring up the list of Favorites on the left hand side, as discussed above (the “Go” option of the

Favorites menu). The History button will bring up the History list on the left hand side, which behaves the same as the Favorites list. The last button on the toolbar is the Print button, which is the same as the "Print" option under the File menu.

Below the buttons is the address bar, with the current URL displayed in it. To the right of the address bar is the "Go" button, which can be clicked to navigate to a new web site. On the other side of the "Go" button is the display box, where any keys that the user presses will be displayed. On the web page itself, there are some numbers in red. These are placed beside the hyperlinks that the user can access by pressing the corresponding numbers. For example to go to the Faculty of Science web site, the user would use the number nineteen.

To "click" on a button, menu, or hyperlink, the user must enter the appropriate number and press Enter. For example by pressing one (displayed in the display box to the right of the "Go" button) (Figure 25) and Enter, the user gains access to the "File" menu, shown in Figure 26.

Figure 25: Press One to Gain Access to the File Menu

Figure 26: The File Menu

Now the user may use the numbers one through eight to execute the menu commands. To get out of the menu, the user must either use a number to execute a menu command or press Escape twice. (Menus are the only place that the user has to press Escape twice. The menus were built in and this was an unavoidable result.)

The user can also type in the address bar by pressing zero and Enter, as seen in Figure 27.

Figure 27: Typing in the Address Bar

Notice that the Go button is not numbered. This is because the user should never have to access it outside of the address bar. When the user has finished typing in the address bar, he or she can press Enter to click this button and navigate to that web site. Alternatively, he or she can press Escape to stop typing in the address bar.

At the bottom of the screen, there is a status bar and progress bar, which show the user how a web site's download is progressing, as seen in Figure 28.

Figure 28: The Status Bar and the Progress Bar

The numbers beside the hyperlinks do not appear until after the web page has fully downloaded.

The History list is cumulative, and as a result, it may become rather large. If you would like to start with a blank History list, simply delete the History folder in the TempBrowser folder, which can be found in the same directory as the application (probably on your C drive with the other programs). The next time you run your program, a new History list will be started.

To move up, down, right, and left on a web page, the user may use the arrow keys or the Home, End, Page Up, and Page Down keys.

If you make a mistake while typing, use the Backspace key to remove one character off the end of the number. Before using the Backspace key, ensure that the cursor is in the textbox. If the cursor is not in the textbox, either click on the textbox with the mouse or press any letter or number. If you do not do this and the cursor is not in the textbox, pressing Backspace will cause the web browser to navigate backwards in the history list, similar to pressing the Back button.

If you make a mistake but do not correct it before you press Enter, the computer will sound a beep to let you know that you have attempted an illegal command. For example if you press 999 and Enter on the Department of Computer Science home page, the computer will sound a beep because there is no link numbered 999 on that web page.

If the web browser does not seem to be getting your input, you may have to click on the textbox with the mouse. Although the browser works well for most web pages, there are some for which it does not work so well. Also, there may be parts of some web

pages that the user may only see by moving the mouse over certain areas of the screen. This cannot be done without use of the mouse.

6. Conclusion

We have presented a unique approach to solving the many costly problems caused by frequent use of the computer mouse. Most common hardware solutions allow the user to use different muscles or more comfortable positioning to operate a computer pointing device. Most common software solutions remind the user to take frequent breaks and/or exercise. There are many software and hardware solutions in existence, but studies show that none of these are satisfactory. TextMouse Web Executive uses a different approach; it uses the keyboard, not as an alternate means of operating the mouse, but as an alternative to the mouse altogether. TextMouse Web Executive does not require the use of any computer pointing device, which is what separates it from numerous other alternatives.

7. References

[1] R. Burgess-Limerick, J. Shemmell, R. Scadden, A. Plooy. *Wrist Posture During Computer Pointing Device Use*. Department of Human Movement Studies, The University of Queensland, 4072 Australia.
[2] "Economical solution for Carpal Tunnel Syndrome," *Information Today*, 10(4): 67, 1993.
[3] Family Village / Computer Hardware & Peripherals
http://www.familyvillage.wisc.edu/at/hardware-peripherals.html
[4] P. M. Fenberg. "Is it carpal tunnel syndrome?" *Occupational Hazards*, 60(6): 34-38, 1998.
[5] E. Fernstrom and M. O. Ericson. "Computer mouse or Trackpoint – effects of muscular load and operator experience," *Applied Ergonomics*, 28(5/6):347-354, 1997.
[6] Index of / point devices http://www.keyalt.com/pointdevices
[7] P. J. Keir, J. M. Bach, and D. Rempel. "Effects of computer mouse design and task on carpal tunnel pressure," *Ergonomics*, 42(10): 1350-1360, 1999.
[8] G. Kohler. "Combating office stress with the martial arts," *Occupational Hazards*, 54(10): 123-126, 1992.
[9] M. Lord. "Is your mouse a trap?" *U.S. News & World Report*, 122(12): 76-78, 1997.
[10] "New laser speeds therapy for repetitive stress injuries," *Health Care Financing Review*, 14(2): 210-211, 1992.
[11] Overview of Proposed Research on Novel Mouse
http://www.curtin.edu.au/curtin/dept/physio/pt/research/iea2000_mouse.html
[12] "Plight of the modern mouse," *Managing Office Technology*, 39(3): 49-50, 1994.
[13] Software from Abacus – NoMouse – Run Windows without a Mouse
http://www.abacuspub.com/catalog/s169.htm
[14] Sun Microsystems Accessibility Program – Accessible Human-Computer Interaction
http://www.sun.com/access/developers/updt.HCI.advance.html
[15] S. Zhai, C. Morimoto, S. Ihde. "Manual And Gaze Input Cascaded (MAGIC) Pointing," CHI 99, pp. 15-20, 1999.