CPSC - 3770

Human Computer Interaction

Final Project

“Graphical Heads Up Display”
designed in Qt by the “BDDS” group

Project Report and Tests

Members: Adam Shepley, Chad Klassen, Jason Racine, Jordan Peoples

Contents

Detailed Topic Description:

The idea and topic we chose to pursue was the general idea of unobtrusive helper-displays; specifically, in an overlay format. This is a rather newer kind of idea, but it stems from the basic idea of sights and cross hairs, and to a more abstract sense, the idea of visual feedback *before* action takes place. In our modern version, this set of ideas and ideologies are implemented as "HUDs", or "Heads Up Displays" - overarching displays that provide surveyed information about user surroundings, personal information, and (assumed to be) needed general information to perform a task (which, for an interactive HUD, amounts to tools and tool access).

The ideas and mechanics behind a HUD, specifically the tool access and statistical portions of it, seem to lend themselves more to computer applications than they seem to have previously led on. From here we develop a main goal - to have an overarching group of individual windows (henceforth referred to as widgets) that would function as statistical helper tools and feedback, by presenting us and our users with pertinent information all the time. The ability to toggle the "activeness" of these widgets will be providing the "Heads-up" interactivity portion of it. The toggle functionality would either enable or disable actually selecting or interacting with a widget and/or moving it. When enabled, the user could access and move the widget just like a normal window. When toggle is disabled, the user would not be able to interact with the widgets, but will still be able to see them, and therefore be able to access information contained in them. This is the basis of our project.

Our widgets we have on screen will be built by us and are designed with a use in mind to showcase importance of utilizing HUD software. For example some useful widgets could be a Notepad, Clock, To Do List, Photo Gallery, and the like. With these widgets a user could say, input some information from a website into the notepad, toggle the HUD off, switch over to a word processing program, and start typing an essay. When the user needs their information, they can simply look over and read it off of the notepad, while still typing their essay. This is one example of how we intend to make the HUD an efficient way to perform tasks.

The program contains a system tray icon that enables the user to toggle between active and inactive mode easier, which also adds a more professional feel to the program. If the user ever wants to completely hide the widgets/windows, he/she can simply right click on the icon, and click "Hide". Similarly to re show the widgets/windows, they can click "Show". This will give the program a heart beat since if the user can't see anything, they can still see our system tray icon, and know that the program is still available and running in case they ever need it again.

After considering why a heartbeat is needed users need to verify what state the system is in. Using a visual cue the software changes the color of the icon. Grey is used to indicate inactivity, blue to signify activity. The idea was to play on the current mental model grey showing inactivity, i.e. menu items that are inactive are greyed out. We've realized that further differentiation is needed to signify the differences between inactive and active; color should not solely be relied upon. Designing an icon that shows the difference with a visual cue such as a checkmark or an inactivity icon would help to make the most out of the interaction for more people.

Previous Research in the field:

As previously stated we were inspired by GM's idea to create a display on the windshields of their cars. There idea was to show the user pertinent and vital data about not just what is in front of them, but also what is around the vehicle, and upcoming events such as speed signs or cops with radars. This would assist the diver, giving them information ahead of time on items that otherwise would not be noticeable until later. During a snowstorm or foggy weather, the system could draw lines on the inner windshield to represent the sides of the roads. This would guide the driver safely home.

The window's widgets and the "screenlets daemon" on Linux were another source of inspiration that have been implemented already. They have information on them, that is customizable, and movable to suit the user's preferences, but their only shortfall was you couldn't click through them to activities behind. They have an option to stay on top of all other windows, but even still, they are always active, and can't be toggled inactive. This leads to the user accidentally clicking on them and perhaps changing settings that weren't meant to be changed.

On the other side of things, our application draws its "support role" ideologies from common Operating System applications such as OSX' "dock" and the "supportive extensibility" philosophy espoused by the Windows 7 Superbar.

In terms of information support, we're trying to push into the same boundaries as shown by Rainmeter on Linux and RocketDock on Windows; both products are themselves working case studies of the applicability of our ideas.

Description of implementation:

Initially we were aiming for a multi-platform application, but once we started getting its transparency correctly working, and being able to click through windows, we realized it had to become platform dependent for the time being. So we decided on Linux (Ubuntu) because we had already started developing in this environment. We used QT to do our creating, because, well that is the language of the course, and thought it to be fitting. For actual code development, we used Qt-Creator, Kate, and KDevelop, whichever one the programmer had preference with. We also set up a Google SVN to accurately track and revert code if need be. This saved us many times, as multiple people updating one file got a bit hairy.

One disclaimer/setback we ran into with our widgets is the issue of clicking through them when they are inactive. The drawback appears when you have your mouse over any widget, and you start to move the mouse. You can see the mouse image flickering between 2 images, which are due to the mouse getting focus of the operating system and then focusing on the widget. The system we have in place makes a region around the current mouse position, and that region is where the user can click though and interact with the windows behind the widgets. So on most mouse moments, this square is recalculated and for a split second the user has access to the widgets (if you can beat the system update speed), even though they are inactive. To counteract this we must insist that anyone using our system and wanting to interact with the windows behind the widgets simply stop moving the mouse, and then proceed with said actions. This way there will be no chance of a user accidentally giving focus to the inactive widget.

Description of ideas:

Our basic idea is that general productivity tasks and concepts can be benefited by applying the same augmentations to them as we do to our basic desktop experience already (as seen in the Task Bar and notification systems of modern operating systems), to the entire computing experience.

By this, we mean to say that we're giving the user access to pertinent and frequently used tools and information, rather than having them rely on recall or switching between current tasks.
Ideally, this would reduce the energy and time needed to, and general complexity of, performing certain tasks.

Intersection of Ideas and Implementation

Notification messages

Notification messages will be placed at the edge of the screen in a simple black window and this gives a way to redirect system notifications, errors or incoming communications to a small area. It's easy for the human eye to detect changes in the periphery. Placing notifications around the edge of the screen reduces visual distraction of the user, being careful not to disrupt user concentration on a task. Dealing with how the memory works, the short term memory will only hold 7 +/-2 pieces of information. Therefore a user will want to complete a task before they get distracted, also known as closure. As designers, we must not disrupt this because visual concentration can be disrupted and perhaps annoy the user. Notifications are placed around the edge to notify the user but they may or may not actually look at it.

Eye movement

Transparency in our widgets allows the user to work normally on everyday tasks, while still being able to access other information in our widgets, without interrupting said tasks. The user may move the widgets closer or further away from any point, depending on preference. The information stored in the widgets will reduce user eye movement by showing them the information instead of forcing them to dig through the menus for it. This reduction in movement will reduce eye focus stress and fatigue. The user can also bring all the widgets to a centralized location, which will further reduce the area of focus, which in turn, will reduce the amount of head movement required. Although this isn't much of a concern for users working on small screens, our software's benefits can be observed when the environment changes to larger screens or multiple monitors.

Power law of practice

Prediction based on the power law of practice may be put into place. Since this software is different, it's not common to work with windows that may be shown on top all the time. Therefore users might get the impression that the software is broken or misbehaving. Once the user understands how the software is expected to perform, their performance should begin to improve as they become familiar with the program. The degree of improvement will lessen over time.

Emotion

Software utilizing omni-present, transparent windows with click through abilities, are quite different to what the general population is used to. Care must be taken when developing such software, because the balance between giving the user an exciting advantage, versus degrading the user experience, is a delicate one. Emotion plays a large part in the user experience, and if the software annoys the user, great irritation may ensue, thus making it harder for them to utilize the software properly. The exact opposite may happen if user experience is as seamless as possible, minimizing fluctuations in emotion, and creativity should replace the rage. This creativity and productivity that would follow is what we aimed to create by developing this software. We want the user to feel excited to use our software; both in the sheer wow factor, as well as feeling it could help their productivity.

Pitfalls and Discoveries found before, during, and after the design.

As our project encompassed a wide variety of ideas and possible implementations, it was inevitable that some portions of the design would conflict or be far different than previously imagined.

Throughout the development of the application, we discovered a great wealth of information pertaining to the ideas in our project, in both the realms of possibility and usability, and how these were shaped by our own implementations of them; though there were also interesting results coming from our testing, that will be discussed in greater length in another area.
Granted, the following is generally our *personal* results gathered from design, development, implementation and inner-development based testing.

During our personal/development based iterative testing, we discovered that a *certain level* of distraction actually complimented our application (and its general "non obtrusive assistant" goal) as it seemed to act as, for lack of a better word, 'confirmation' of its own presence, without being overbearing. A fluttering of the cursor due to our invisible window system, served as a subtle hint towards the presence of our HUD. When we maximized our draw rate intending to reduce this flicker, the system would bog down and become sluggish, which was more of a detriment than the flicker itself.

When trying to implement multiple widgets, we found a point where one of our design paradigms had a fundamental split; while the general idea is unobtrusiveness, an important aspect to that is actually the *cohesiveness* of our widget system as a whole. We found that this did not always mean each widget had

to look the same. Some widgets actually fit in better with slightly different aesthetics than the original look.

As we iterated through various designs and tests for potential user control schemes, we discovered that reusing the same on screen real-estate for different states of a program made it feel much more flexible and streamlined than offering multiple controls spread across multiple overlays. For example, by having a *single*-sized area for our inactive and active opacity sliders, that simply toggles between which slider is visible/usable (when switching between modes) our program seems to have gathered a stronger sense of overall completeness – despite the alternative (two ever-present sliders occupying twice as much space) being logically the same, feature and design wise.

One 'major' (philosophical, but not literal) issue that arises with our project is relationships between our widgets; while they all function under the banner of our greater “HUD”, they don't generally have a distinct relationship to one another; For instance, a calculator has no real relationship or connection to a Photo Slideshow widget, and yet both appear on-screen together, and have the same influences and general implied 'importance' as each other.

Oddly enough, the solution to this accidental but inevitable disharmony was in fact an unintended result of correcting a literal idea of widget flexibility. That is to say, the general look and feel of the widgets required the implementation of a central tool bar class and a widget manager class; as an end result, every widget has been granted an unobtrusive side-bar, used for moving the widget, which blends well with their individual design. This similar look and feel creates a symbolic 'link', or family relationship between the various “unrelated” widgets, and should, in the users mind, group the widgets together as one logical working unit.

Test Cases

"The Great Northwestern Text Edit" Test #1

Goal: Take some chunk of information off of a website, paste it into a text editor of some kind, open a text document and copy the information into it. This will simulate a user saving some information from the website into a text document, and then later on, the user wants to access it and put it into an essay or some other document.

Utilizing our software:

1. HUD will be viewable, but inactive.
2. User will be viewing a website with some information.
3. Start timer.
4. User will select and copy this information to the clipboard.
5. User will toggle the HUD active, and paste the information into the notepad widget.
6. User will open an Open Office text document.
7. User will copy the text out of the text widget into the clipboard.
8. User will paste the information from the widget into the open office document.
9. Stop timer.

Without our software:

1. User will be viewing a website with some information.
2. Start timer.
3. User will select and copy some information to the clipboard.
4. User will open an open office document, and copy the information into it.
5. User will open a second open office document.
6. User will copy the text from the 1st document, and paste it into the 2nd one.
7. Stop timer.

"Crunch Those Numbers" Test #2

Goal: While doing a full screen task (like watching a video), the user needs to compute some numbers. So we both switch out of full screen and open the built in calculator, or we utilize the calculator widget that will be present on the screen while the user watched the video.

Utilizing out software:

1. User will have the HUD open and inactive.
2. User will be watching a full screen video.
3. Start timer.
4. User will toggle the HUD to active mode.
5. User will use the on screen calculator widget to compute 55*128.
6. User will toggle HUD to inactive mode.
7. User will then write the answer to the calculation down on a piece of paper beside the computer.
8. Stop timer.

Without our software:

1. User will be watching a full screen video.
2. Start Timer
3. User will toggle out of full screen mode.
4. User will open the built in calculator.
5. User will computer 44*246.
6. User will write the answer to the calculation down on a piece of paper beside the computer.
7. User will toggle video back into full screen mode
8. Stop timer.

"What Time Is It Man?" Test #3

Goal: User will need to access what time it currently is, but they will be in a full screen application. This will simulate when a bunch of friends are around a computer viewing a video, and one of them needs to go soon. So he/she asks the computer owner what time it is, and the owner relays this information to him. Since our widget will always have a time of 0, this test will NOT be performed by the test subjects.

Utilizing our software:

1. User will be viewing a full screen video.
2. HUD will be open and inactive.
3. Start timer.
4. User will look over at on screen clock widget and view the time.
5. Stop timer.

Without our software:

1. User will be viewing a full screen video.
2. Start timer.
3. User will toggle full screen view off on the video.
4. User will check the system tray clock time.
5. User will toggle full screen video back on.
6. Stop timer.

"Massive Calculations" Test #4

Goal: Simulate the user doing some calculations and storing the result, then needing those results for a final calculation. This test will utilize 2 widgets, the calculator and the note pad.

With:

1. User will have the HUD open and inactive.
2. Start timer.
3. Toggle the HUD to active.
4. Calculate 44*6.
5. Store the result in our notepad.
6. Calculate 59*12.

7. Store result in our notepad.
8. Calculate 9*65.
9. Store result in our notepad.
10. Add all three of the results together.
11. Put final result in text box.
12. Stop timer.

Without:

1. Start timer.
2. Open system notepad.
3. Open system calculator.
4. Calculate 44*6.
5. Store in notepad.
6. Calculate 59*12.
7. Store result in our notepad.
8. Calculate 9*65.
9. Store result in notepad.
10. Add all the results together.
11. Store result in notepad.
12. Stop timer.

The tests and times each user took to complete them

	"Northwestern Text Edit"		"Crunch Those Numbers"		"Massive Calculations"	
	With HUD	WithoutHUD	With HUD	WithoutHUD	With HUD	WithoutHUD
User 1	36.86	45.47	19.02	19.67	1.05.74	1.07.03
User 2	25.96	16.05	15.54	15.93	40.1	47.97
User 3	25.0s	18.0s	16s	20s	46s	50s
User 4	39.1s	24.4s	21.9s	18.8s	1:16.5	1:20.5s
User 5	22.3s	25.5s	18.9s	19.2s	52.2s	63.0s

Preference Based Data:

We will also test based on preference. Say for example, we will ask users if they prefer the clock widget, or the task bar built in clock or neither. Another example would be using the built in operating systems notepad, compared to our notepad widget, or neither. This will be determined after the test, we will ask the user did they prefer one to the other, or were they indifferent. This will not only give us physical timed data, but also if they like one over the other in personal preference.

Widget/Software Preferences and Additional Comments

	Banner Comments	Sideshow Comments	Calculator Comments
User 1	Don't see use for it.	If we had 2 screens it would be cool, but other than that, not useful.	built in one (no chaining of commands)
User 2	Good for reminders, not much else. Need to make smaller, or I don't prefer it	Only on big screen or a 2nd screen, prefer to not have it, but cool for a occasional glance at, only if the screen real estate is good enough…	ours is better, but make it smaller
User 3	Could be more useful if it had newsfeeds from RSS working	Not particularly practical in my opinion.	Handy to have available on-screen at any time
User 4	The banner could be used with RSS feeds or showing stock market tickers. However I would not use this widget at all for anything.	I couldn't see slideshows used for anything really important. I use one to slide through a bunch of photos that would run in the background. Every once in a while I will look at the pictures, but doesn't have any significant use.	I could see the convenience of having a calculator in the HUD. I would not utilize the convenience enough to have it on it. So I would never have it.
User 5	Isn't too useful on its own. Seems superfluous.	Seems cool, but not a very productive widget.	Very useful, as it's quick and easy to copy to and from. It's always present.

Widget/Software Preferences and Additional Comments (Continued)

	Clock Comments	Notepad Comments	Overall Software Comments
User 1	Liked ours, because you can change colors.	Liked ours the simple on screen one.	Liked it on the screen, could be very useful for games
User 2	Like ours better, color change is nice.	Make smaller, but very useful for essays or cheat codes etc.	Annoying to have a bunch of windows up, distracting. If window background were 100% transparent it would be much better. I prefer to not have it up.
User 3	Would be useful in games where timing is important. Stopwatch functionality would be nice as well.	Useful to have on screen	Some widgets useful.
User 4	The clock in the taskbar is just as visible in any of my cases. Therefore I would not use the clock.	I could just simply open a notepad document from the start menu. Never have any need to type something into the widget notepad.	The idea for the overall software is solid however there are a few bugs that make it slightly less efficient then intended. Not necessarily a bad thing and would be useful if further development was to continue.
User 5	Very useful, but only more so when doing full screen tasks. It's somewhat more useful than the system clock because of this.	Possibly the most useful widget, having a text editor all the time gives you more flexibility with other applications - it compliments almost everything well.	Overall, I believe that the Clock, Text Editor and Calculator are more useful widgets than they let on. They allow you to keep the context for your current actions and applications while still augmenting the experience.