UI Design Process

When we set out to design Drvr, we wanted to make sure it was a fun and interactive application. Our app had to be not only simple to use, but it should also appeal to a large part of our target market, and the target user should be properly engaged when using our software. One of the reasons for this is that we truly believe that we would be able to make a change in the way people learn about cars, and people will genuinely become well versed about their cars with the use our application.

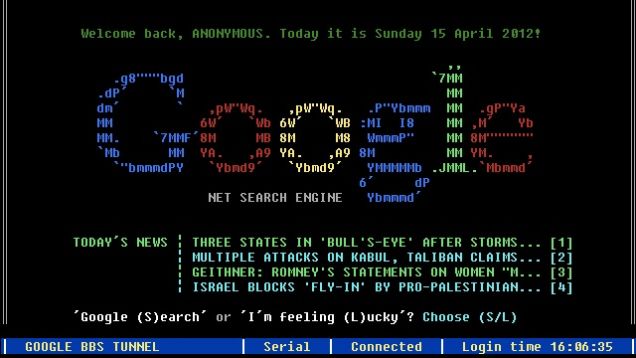
A line that sums up our idea is none other than the design philosophy of a car manufacturer, Volvo. Volvo has a very unique design philosophy, which is “The development of all Volvo products and services must spring from the needs of the people and end with their satisfaction”. In fact, their latest campaign sums up the importance of people centric design, expressed by the following transcript:

“We love technology, only if it makes life less complicated for people. Some say cars are all about going fast. Some say cars are all about looking cool. They aren´t wrong. But our main passion is to help our drivers. To help them take responsibility for the world around them and to help them live life less complicated. Cars are driven by people. And that´s why we design them around you.”

From the statement above, it was clear that we had to ensure the level of usability has to be very high. The thing about design is that if it isn’t done right from the start, once we were to go ahead with the development process it would be very difficult to modify it, as the number of changes that would have to be made would be immense. Another issue is that every time you modify the interface, it is highly likely that you would run into a series of troubles, and this is a fact as per what we have been taught in CSCI318 Software Engineering Practices and Principles, that is currently running in the university.

# Core design: Graphical user interface

It was given from the start that we would have to have our application to ship with a graphical user interface. It is known that professional computer users prefer not to use a GUI due to the command line interface being faster to operate and can run even on low powered machines. However faster is not always better, especially in this situation.

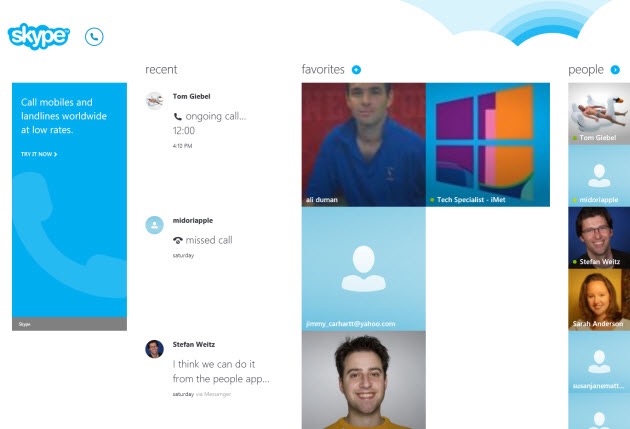


(Image credit: io9, Gawker Media. Sample command line interface with the use of colours and ASCII symbols to make up images)

While it would be somewhat interesting to see how a car would look like in a command line interface, the reality is that a large proportion of users dare not touch the command line tools available on their computers. Also from a usability standpoint, for novice users, it would be better for them to operate our software using a mouse as opposed to a keyboard, as a mouse is easier to handle and the user would not need to remember the instructions he/she needs to type into the system to invoke a function in our program. Having said that, in some cases like setting a new speed of the car, or using the on screen arrow buttons, it would be great to give users the flexibility to use the number buttons on the keyboard or use the arrow buttons on the keyboard for the same functionality, so we would attempt to integrate some control that can also be done by the keyboard.

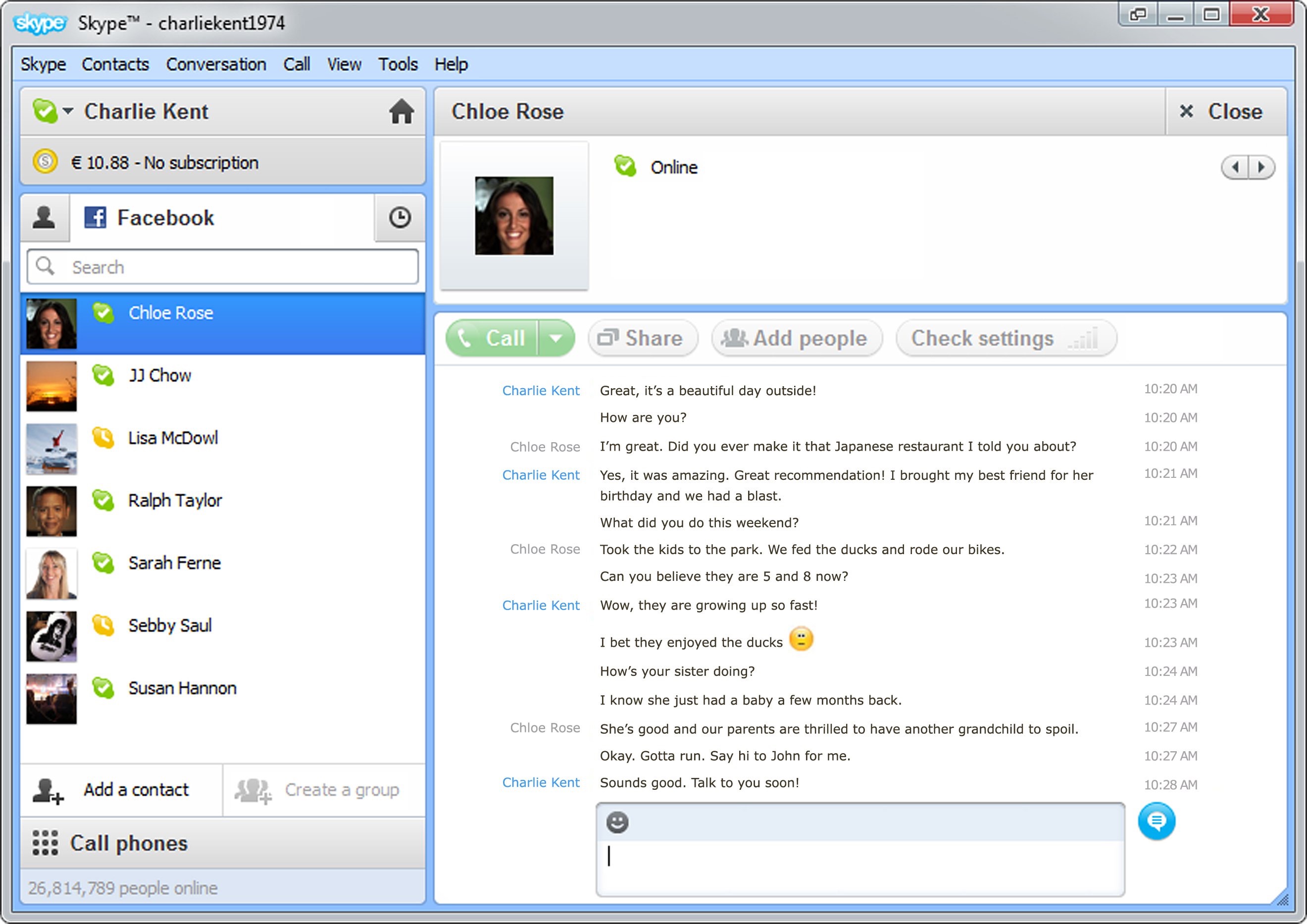
A web-based interface was something we did consider, however in order to realise the multithreaded nature of the simulation part of the system, it would be very difficult. In addition to that, web based interfaces generally are not as responsive as desktop GUIs, due to the lack of many web browsers being able to fully take advantage of the processing power of a computer. While having a web-based interface would increase the appeal, it would potentially reduce the user satisfaction due to poor performance.

As a result of this, we decided that it would be best to work with delivering the app with a GUI. However it isn’t that easy. There are two kinds of GUI systems available for our target OS, Windows 8, namely Modern UI and the Classic UI. The Modern UI is the new interpretation of the GUI on Windows 8 and above machines. It takes advantage of large fonts, crisp and slim text, in addition to an image centric approach. The image below shows Skype, a Modern UI application running on a Windows 8 machine:



(Image credit: PCMag.com)

However one of the main limitations we noted when we tried out some of the Modern UI based apps was how it wasn’t very interactive to use especially when there isn’t a touch screen built into the computer. Because the design depends on gestures for it to work best, it felt very clunky when used with a mouse. Since not all computers with Windows 8 shipped with a touchscreen, in fact many didn’t, it meant we had to look at the Classic UI instead to ensure the experience would not be compromised.



(Skype on the Classic UI for Windows 7. Image Credit: PC Advisor)

One of the great things about the Classic UI was how well it works with a mouse. Also, the added familiarity would mean that users who generally know how to operate a Windows computer with a mouse would be able to operate the application. We noticed how many users would rather use the Classic UI as opposed to the Modern UI, as noted from what we have seen when walking around computer labs and in libraries. Another advantage of building the application with the Classic UI was the vast amount of resources available for us to use, after all this was a UI that has been used publically as of 1995, making it 20 years old.

This Classic UI supports both input from a mouse and a keyboard, but in our case, we would be having our application to be primarily operated by a mouse only, mainly for simplicity, with keyboard support in some situations. We find that it is much easier for a user to operate a program with one kind of input device, as opposed to using multiple input devices at the same time. In addition, it is actually easier from a development point of view, as we are able to control the types of input we are accepting from a user. For example, in the case of adjusting the speed of the car, it would be better for us to have a plus and minus button as opposed to bringing up a dialog box that prompts the user to key in a new vehicle speed. By avoiding the user to have to key in a new value, we can prevent accidental input of an invalid value (like a negative value – which will invoke a warning message that would require the user to check the value entered) or an incorrect value (like keying in 100 as opposed to 10). However, as we understand that some users would prefer to have keyboard input available, we would still try to implement some keyboard support in.

An interesting question we did get during our planning presentation in regards to the UI being a Classic UI but setting the minimum requirements to be Windows 8 as opposed to Windows 7 (which doesn’t have the Modern UI), we decided to use the Classic UI while maintaining a more modern OS mainly because in general we find that Windows 8 is more polished and it is actively being supported by Microsoft, unlike Windows 7 which has ended it’s mainstream support as of January 13, 2015 (Microsoft Support Life Cycle 2015). This means that Windows 7 is only going to be receiving security patches, and no more performance improvements, unlike Windows 8, which will still receive general improvements as part of the mainstream support up to January 9, 2018 (Microsoft Support Life Cycle 2015). In regards to this, it would make sense to work on a platform that has some form of active support, as opposed to a platform that is being deprecated soon.

Constantine and Lockwood’s book on user interface design also gave us some guidelines in the UI process:

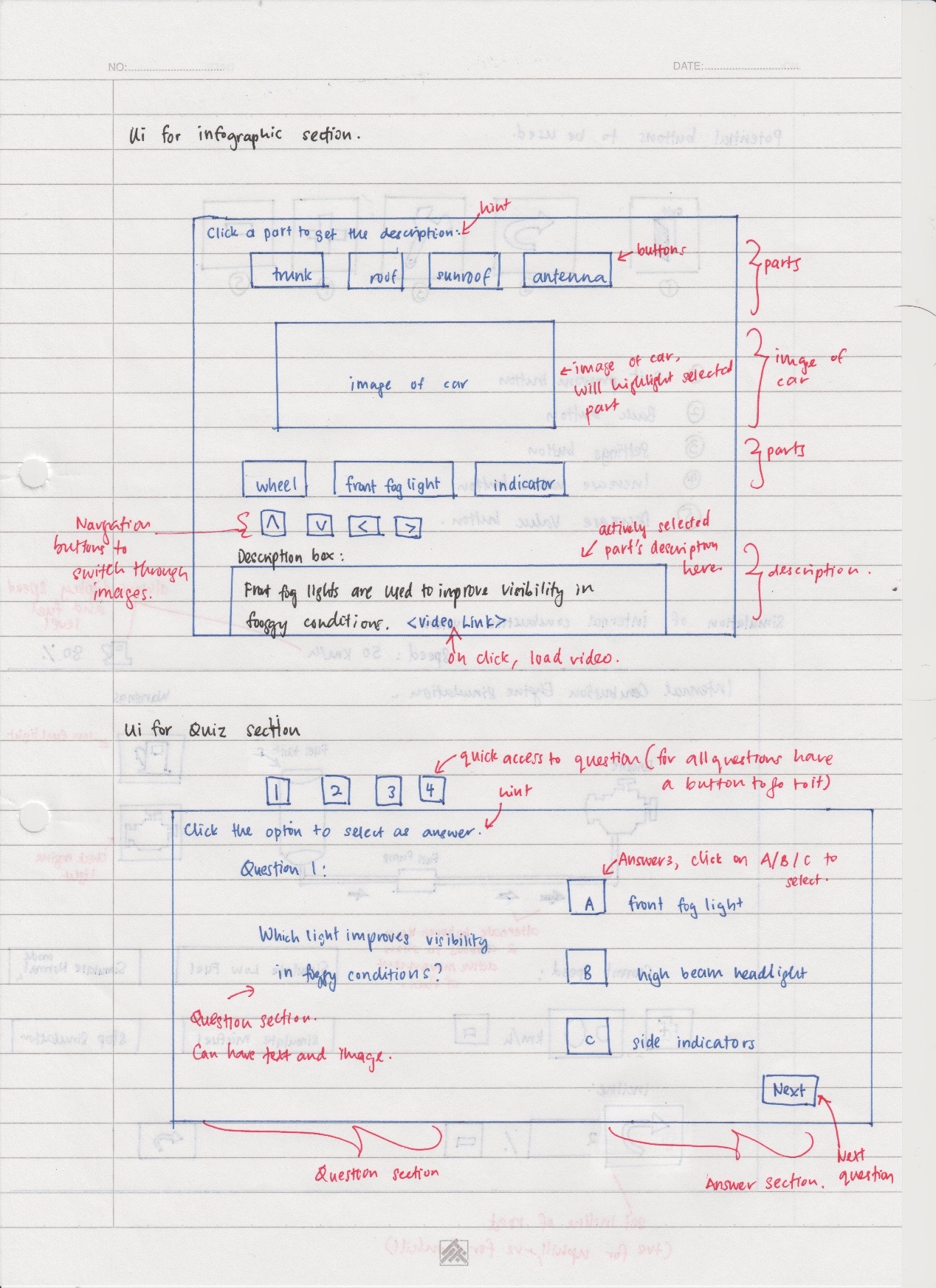
1. The structure principle

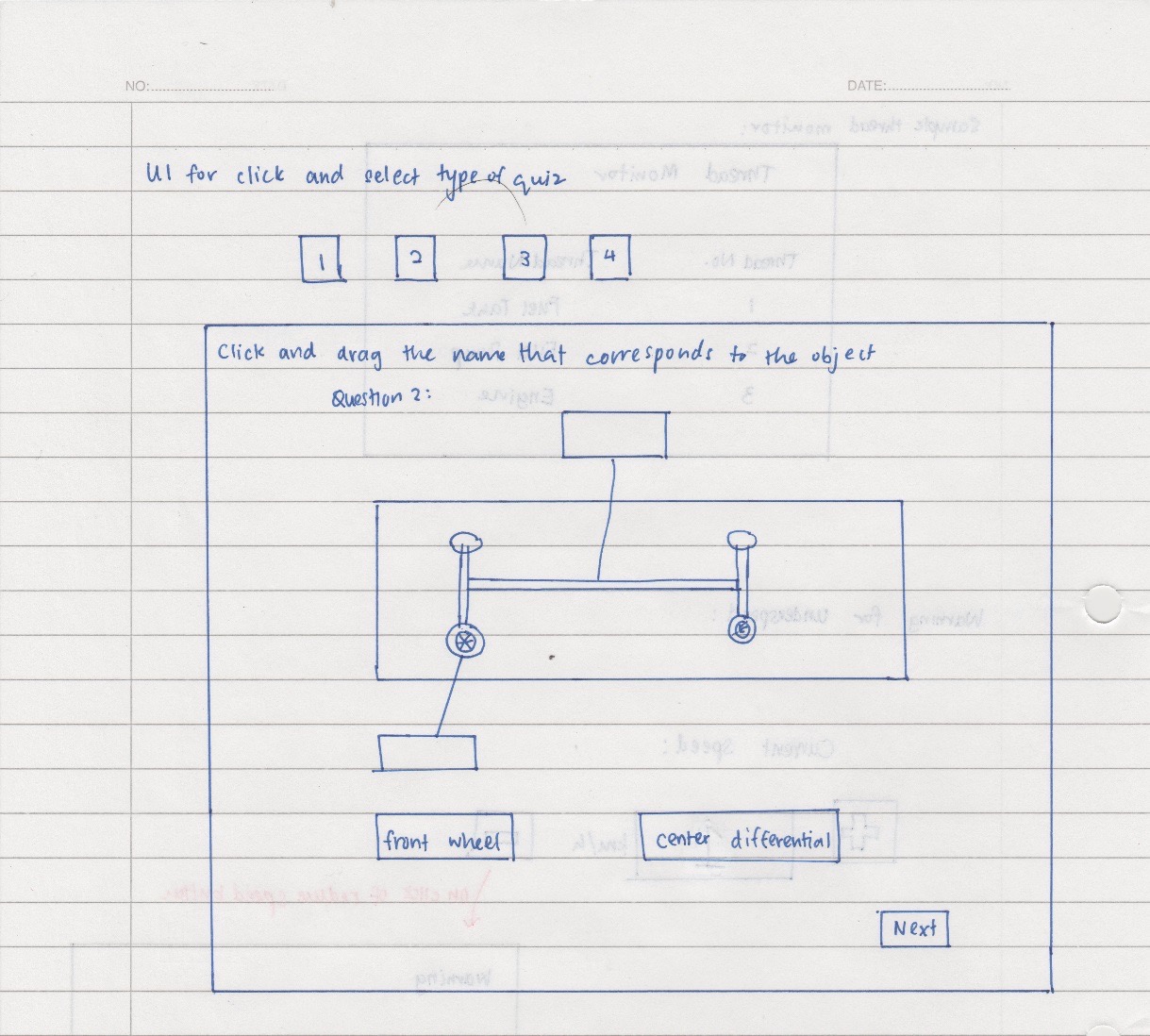
Common things should be grouped together. In our application, we would be grouping the features into 3 distinct parts, namely the infographic section, the simulator and the quiz section.

In the case of the infographic section, there would be a clear distinction between where the names of the parts of the car are listed and the description of the parts of the car.

As for the simulator, common items like the car status would be kept in a specific section of the screen, while the actual simulation itself would be kept in a different part of the screen. The same applies for the quiz section where the question has to be separated clearly from the choice of answers.

The following paper prototype shows how we would be able to achieve this segmentation and separation.





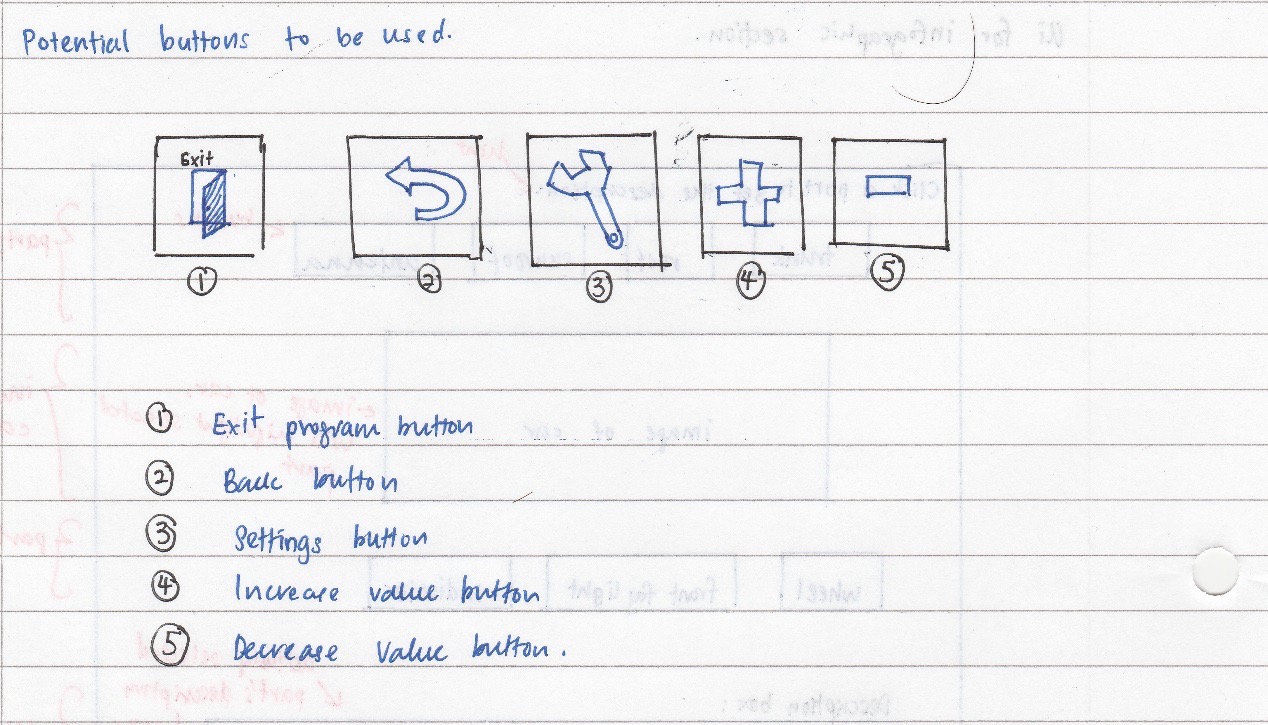
1. Simplicity

The design approach should ensure that simplicity is kept as the main point of our user interface. For a start, buttons should be clearly labelled, and should have meaningful icons (if icons are to be used), for example the exit button should have an icon of an exit door.

There should be adequate guides and hints throughout the application that guides the users to be able to easily navigate the application, in the unlikely event should they get confused. Hints should be clear and be in natural English.

In order to ensure we are on the right track, it would be best if we are able to get some feedback from the target market, like some form of acceptance testing in order to make the system easy to use.

The following shows some prototype buttons that have been designed:

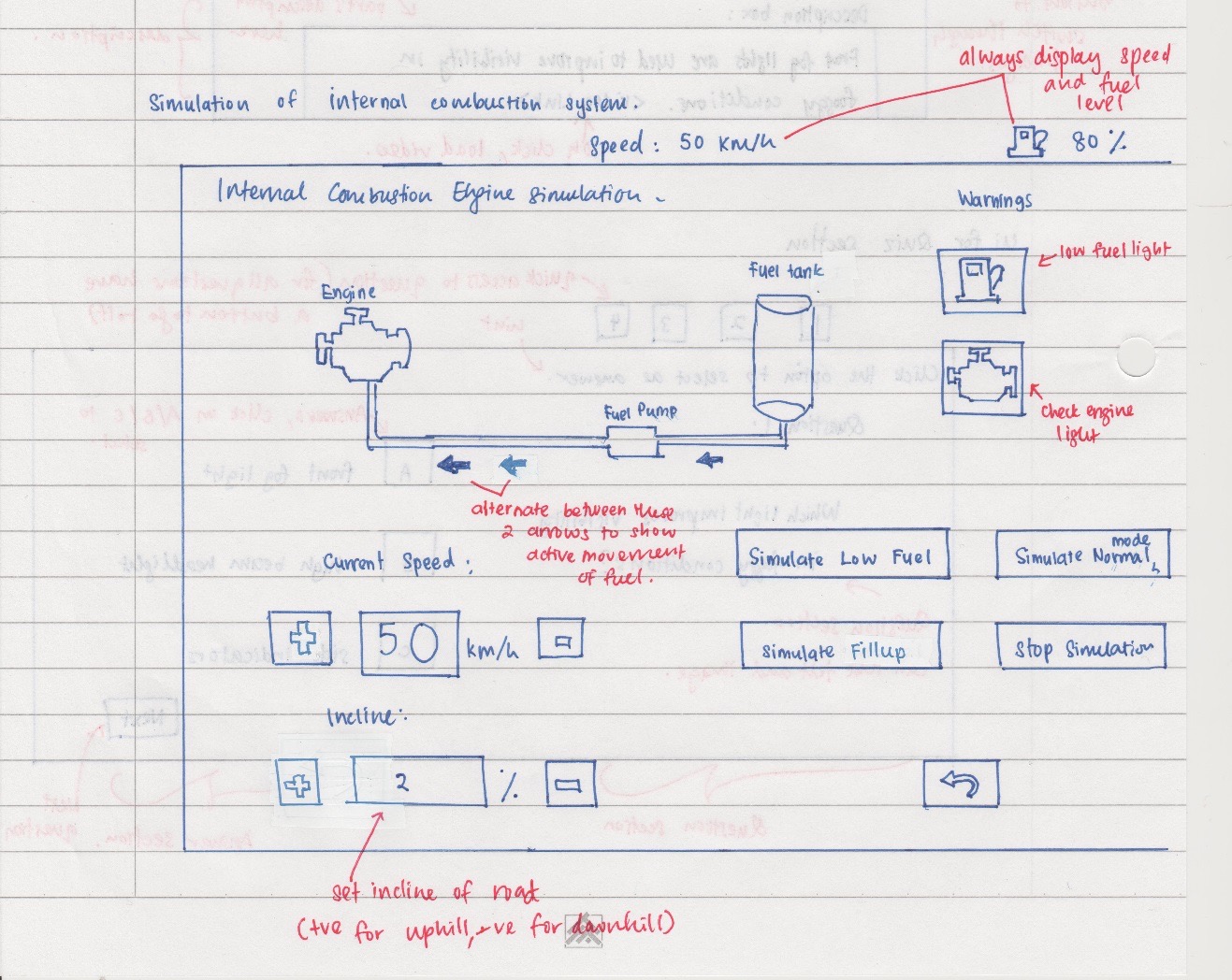


1. Visibility

While a function is being executed, it should be that only buttons and other visual elements that are necessary are displayed. All other visual elements that are not required should be hidden away as this could potentially confuse the user with redundant information.

For example in the case of the simulation of an internal combustion system, it would be redundant for us to have a button that turns on the lights in the car being displayed in the window. There is no relation of the headlights to the internal combustion system. The presence of additional unnecessary visual elements like unused buttons only add to the clutter in the program, affecting the user’s visibility of the actual options they can click on while running a certain function.

The following shows how we only have relevant buttons in the internal combustion engine simulation:



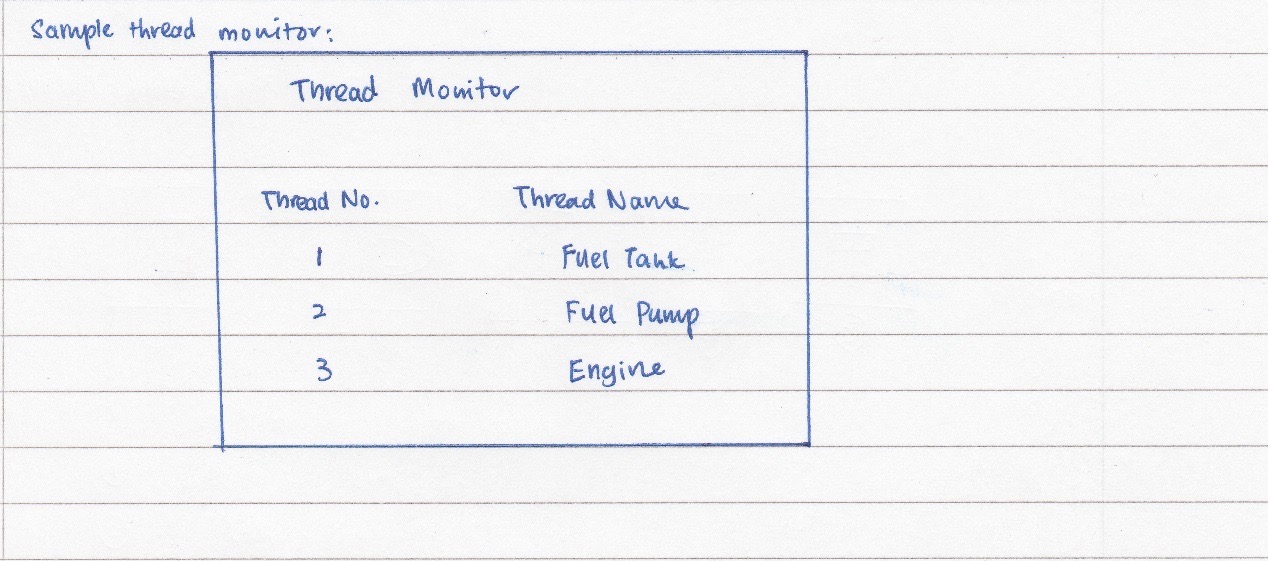
1. Feedback

Throughout the program, there should be some form of feedback, be it visual, acoustic or textual, to keep the user informed of the current operation of the application. Feedback would be important, as it is essential for the user to be able to know what is going on in the system, like in the simulation.

This should be achieved with clear and concise textual feedback, or in the case of some features being turned on; their icons should light up (like in the simulation of the electrical system in the car, should the hazard lights be turned on, the indicators should light up). Or in the case of the simulation of the flow of fuel or the braking power, arrows should be actively displayed showing the direction of flow of fuel or brake fluid in the car, such that the user knows at any one time how are the fluids moving about in this subsystem.

In addition, this principle should also be achieved using a thread monitor that is available throughout the runtime of the program, such that the users can track at any one time what threads are currently being run by the program.

The following shows the potential thread monitor we would have. Refer to the prototype image of the internal combustion engine simulator to see how the movement of fuel can be shown using the arrows.

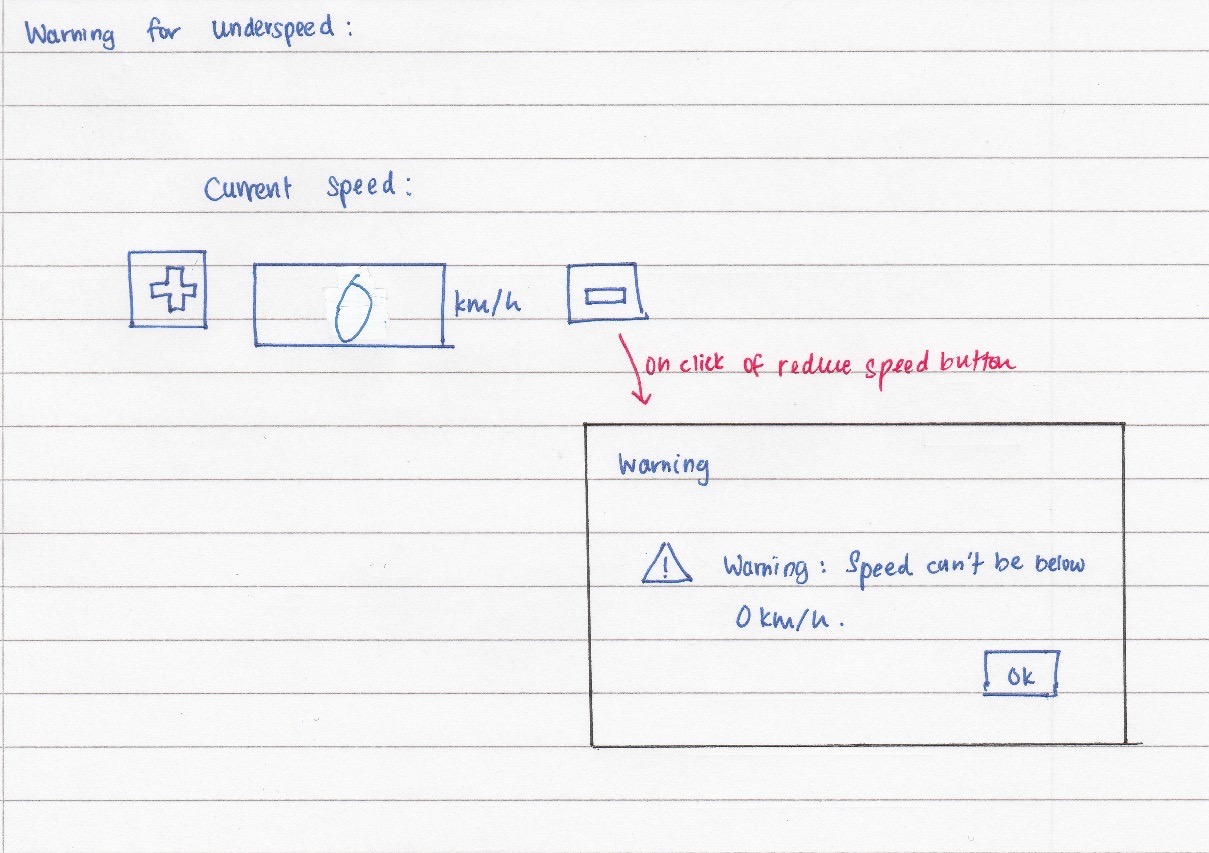


1. Tolerance

Error prevention would be the best approach in our system in order to achieve the tolerance principle. One of the issues for this is that by having a primarily mouse only interface, we actually have a limited set of options the user could possibly take in any situation.

However there has to be some error checking that should be done. For example in the case of simulating the fuel consumption of a car, it should be such that the speed of the vehicle cannot be reduced or increased after a certain speed threshold to prevent invalid or unfeasible values being used in the system. It would be illogical for a car to have a negative speed or a speed of over 200 km/h.

The following shows an underspeed warning, that is caused when attempting to reduce the speed to below 0 km/h.



1. Reuse

As much as possible, reusing design would be able to increase the usability in the system. This is to ensure the user is able to use their familiarity with a section of the to relate to the other parts of the system. However in our system, it would be only best for us to reuse the design of buttons in the system (i.e. ensure consistency in the buttons used, like a button with a picture of an exit door should only act as the exit button throughout all the sections of the program), but not the structure of the UI in each of the sections, as each of the sections require the user to respond differently. For example in the case of the quiz section, it would be messy to have the potential answers to be placed both above and below the question, unlike the infographic section where it makes more sense to have the names of the car parts to be both above and below the image of the car as that helps in differentiating the parts of the car from the top half of the car from the parts from the bottom half of the car, to a certain extent.

[http://profiling.volvogroup.com/en/Volvo/~/media/Files/Volvo/Exterior%20Signs/The%20Volvo%20Design%20Philosophy.ashx](http://profiling.volvogroup.com/en/Volvo/~/media/Files/Volvo/Exterior%2520Signs/The%2520Volvo%2520Design%2520Philosophy.ashx)

<https://www.youtube.com/watch?v=oLZ3np-61Yo>

<http://i.kinja-img.com/gawker-media/image/upload/s--nfSIpIcD--/c_fit,fl_progressive,q_80,w_636/17jo8d3kxw0hjjpg.jpg>

<http://www9.pcmag.com/media/images/363542-skype-for-windows-8.jpg?thumb=y>

<http://www.pcadvisor.co.uk/cmsdata/downloads/23296/largeImg.jpg>

<https://support.microsoft.com/en-us/lifecycle?c2=16796>

<https://support.microsoft.com/en-us/lifecycle?c2=14019>

<http://www.ambysoft.com/essays/userInterfaceDesign.html#Principles>