COMP SCI 4HC3: Human-Computer Interaction
Assignment 2 – UI Design

Due Date: Thursday, Nov. 10, 2016, at the start of lecture (10:30 am)
Late Submissions: Accepted late until Thursday, Nov. 17, 2016, noon, at a 15% per day penalty.
Grading: This assignment is out of 100 marks, and is worth 10% of your final grade. Note that the grading of this assignment is likely different from what you might be used to – see below for details.
Group Work: This assignment is group work and should be completed in groups of 2 – 4 students. If you wish to use a group different from your project group, this is ok, but you must see me first.

For this assignment, your group will design and implement a UI for one of the possible options listed below. The key idea in all cases is that the UI should be *touchscreen-based*. For our purposes, it is sufficient to develop your application in something like Visual Basic or C#, and imagine that the mouse pointer is the fingertip, i.e., you don't actually have to support a real touchscreen! The actual functionality of each system is very minimal – your primary goal is to design a usable interface, and use your programming framework to simply support navigation between screens and the minimal functionality.

Minimum functionality for each option is outlined below – note that these are intentionally written without regard to the UI, but rather *just* the features the system should support. Implementing this *bare minimum* functionality with an adequate design will yield a grade of approximately a C+. A higher grade will be awarded for better designs and/or additional functionality – but note that and simply adding *additional* features may not yield a higher grade if these features detract from overall usability! The key here is designing the interaction using the concepts discussed in class – not simply adding more and more features.

Recall from class that new technology doesn't automatically improve usability – so your objective here is to think about the design of your system in the context of the usual design principles/guidelines, to develop a usable system.

Option #1: Parking Meter
Mechanical parking meters are extremely usable devices – you put in your coins, it adds time, and you walk away. Modern digital parking meters also support credit cards, but interacting with these systems is complicated and error prone. A quick survey of parking meters around campus reveals little consistency in the design of these, extra instructions taped to the front of the machine, spurious buttons that have no apparent function and no feedback, and various other major usability issues.

For this option, design a better digital parking meter UI that uses a touchscreen. This should be the type of parking meter where you pay a certain amount of cash, and leave a printed ticket on your dashboard indicating how long you can park for. It should support the following features, at minimum:

- Ability to use coins, credit, and debit card (simulated, obviously)
- For credit cards: ability to enter an account number **as a backup to** simulating swiping/inserting a card
- Simulated printing of dash-board ticket (e.g., you can just display a "virtual" ticket in a dialog box – please don't actually print these on paper!)
- Ability to add/remove time prior to printing ticket
- Refund of time for returned tickets; a novel feature – I'm not aware of any parking meters that do this, but the idea is that if you paid for more time than you needed, you can return the ticket for a (partial) refund later

Option #2: Bank Machine
Bank machines have been discussed a few times in this course already. In general, the UIs for these are fairly simple, as users are expected to be able to "walk up and use" with very minimal training. But, there are sometimes problems: consider bank machines that accept the card, process a withdrawal, then release the card. In these cases, it's more likely that a user will forget their card as completing the transaction (getting the cash) likely "completes" the interaction. In contrast, a system that simply *swipes* the card will be less error-prone (users will likely *not* forget their card).

For this option, design a touchscreen-enabled bank machine UI. It should support the following features, at minimum:

- Ability to enter an account number as **an alternative *or* as a backup** to simulating swiping/inserting a card
- Ability to withdraw and deposit cash, updating a bank account balance accordingly
- Ability to simply check/display account balances
- Transfers between accounts

Other Features

As described above, you are free to add additional features beyond the minimum functionality. This is a great opportunity to innovate. However, these features should not feel "tacked on", or added as an afterthought – adding unusable features will not improve your grade (and may reduce it!). As described above, the main objective here is to develop a *usable* system. Hence careful design is needed prior to implementing *any* features in your system mockup. In summary, a very well- designed system that meets the required minimum functionality above will receive a higher grade than one that adds numerous extra features but results in a confusing and difficult-to-use UI. Be sure to document any additional features.

Programming Language / Coding Guidelines

You are free to use any programming language you want for this, with the following caveats:

1) You will not get extra credit for picking a difficult or obscure language (e.g., using C and OpenGL will likely increase difficulty). Similarly, you will not get credit for submitting a partially complete project that doesn't actually meet the goals outlined above because you got bogged down in programming details.
2) Certain languages are very well-suited to this type of task, e.g., Visual Basic and C#.NET provide "drag 'n' drop" interfaces for building UIs with minimal coding required in the background to transition between screens. I strongly recommend considering these options.
3) We don't actually care about your code. It can be as beautiful or ugly as you like but we aren't even going to look at it during grading. We only care about a working system, which will be what is graded. In that context, you should submit in some form that is easy for the TA to *run* your program (better yet, an executable with clear instructions provided). If you don't provide sufficient instructions for the TA to run it, you will be required to demonstrate it to the TA, **and** will lose marks.

Submission Notes

Submit your code, any project files, and some kind of clearly identified executable (preferably Windows-based) or link.
You should include a separate note file which lists instructions for the TA to run your software. Call this "Run Instructions."
Any design documents documenting your decisions should also be submitted. Submit these documents by the date specified above to the link provided in the Avenue to Learn website.
Also submit hard copies of your design documents in lecture. Email submissions to the instructor will not be accepted.
Familiarize yourself with the department's policy on plagiarism and the university regulations on plagiarism and academic misconduct.
Note that this is group work, but sharing between groups is not permitted.