Programming Skills (Design Patterns and C#)
CSCI-541 / CSCI-641

02/20/2015

Lab 2: WPF (T9 Messager)

Due: Monday, March 16, 2015 at 11:59pm

1 Goals

Familiarize yourself with the WPF framework, using the MVC design pattern, event driven programming, and built in data structures for .NET

2 Overview

Often times when calling an office, you are given the option to do an automated lookup of the employee phone directory by using your keypad on your phone. For example, to get an employee with the last name of *Brown*, users would type in 27696. In another example if one types 2-3-3-7, there are many legitimate words that could be spelled, since the "2" key represents "a", "b" and "c" the "3" key represents "d", "e", and "f", and the "7" key represents "p", "q", "r", and "s". Legitimate words might include "adds", "beer", "beds", "bees" and "beep", depending on the dictionary from which you're working.

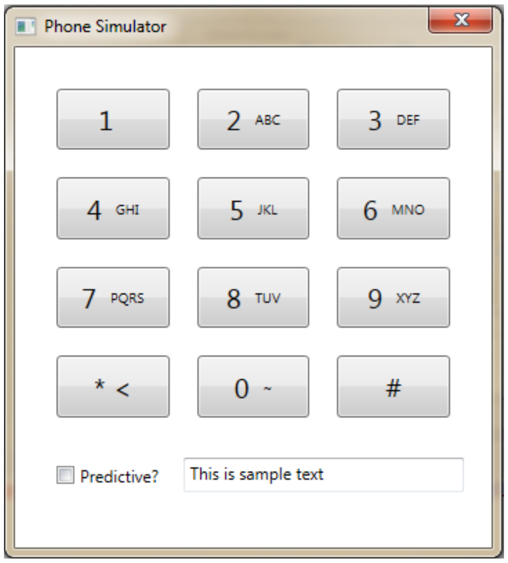
In some older systems, users would be required to type 22 77 666 9 66 in order to get the name 'Brown'. In these systems, "2" represents "A", "22" represents "B", and "222" represents "C". Clearly the former system is more complicated as it requires a dictionary to do matching. Your project will support both methods for input depending on user preference.

3 Requirements

Using WPF you must build a graphical representation of these parts of a touch tone phone.

- a 4-row-by-3-column 12-key key pad. The symbols assigned to the keys, in row-major order, are:
  - 1
  - 2,a,b,c
  - 3,d,e,f
  - 4,g,h,i
  - 5,j,k,l
  - 6,m,n,o
  - 7,p,q,r,s
  - 8,t,u,v
  - 9,w.x,y,z
  - *,<
  - 0,˜
  - # (space)
- a one-line alphanumeric display

A display example is shown below, though your interface need **NOT** look like the one below.



The 4x3 array of buttons are to be clicked by the mouse.

3.1 Non-Predictive Mode

If you are **NOT** in predictive mode, the application behavior is simple. Multiple *clicks* of a button will result in a change of the letter. For example, clicking "2" and waiting will cause the letter "A" to appear. Clicking "2" twice rapidly will cause a "B". The trick here will be making your program correctly deal with this behavior, as there are different events generated for *MouseDoubleClick* versus a regular *click*, though this is only one way to solve this issue (there are several other preferred methods. Part of your assignment is dealing with these issues. While there are multiple different ways to handle these clicks, but your program must be responsive and functional.

3.2 Predictive Mode

Your program will always interpret button 2-9 presses as letters. The challenge is to decide which letter was intended. The output text is at the bottom is the display. It gets updated as the user presses the buttons. The display must hold at least 20 characters and need not be more than one line. At any point in time the display shows all previously typed words and the program's first choice for the word currently being typed. This could be a complete word or the prefix of an incomplete word. If no legal word is possible using the prefix typed, the display will show hyphens ("-") whose count is equal to the number of keys pressed so far. The button marked "<" is the backspace. It deletes the most recent button press from the letter rows. Your application behaves as if the remaining letters were pressed for the first time.

The button marked 0 / ˜(tilde) is the next function. It circulates to the next word choice that can be assembled from the sequence of button presses so far. Its behavior is only well-defined if a complete word is showing. If only a prefix of a valid word is showing, the behavior is undefined except that it must not impede the user from continuing to use the application normally. If the "invalid prefix" hyphen string is showing, the next button has no effect.

There is no symbol under "#". It is the space button. Once the user enters a space, she or he may press a backspace to delete the space, and in addition to the space, the previous word is deleted.

The set of legal words comes from a file english-words.txt. Download it. It is a text file containing one word per line. It contains about 211,000 words, all lower text. You may ignore case. Assume all letters entered are lower case, i.e., there is no way to "shift". Of course, your program should work with any word file.

The basic implementation should work as follows:

1. If there are valid words of the given length, display them
2. If no words of the given length are available, show words with that prefix
3. If no words with a valid prefix are found, show the - for each letter already entered

4 Design

For the model part of the software, one design approach may immediately come to mind.

In computational algorithm design, on-line means doing things at the last minute, or "in real time". A sequence of buttons pressed so far (since the last space) is kept, perhaps in a list. Each time the user presses another button, the sequence is augmented and a list of possible word choices is generated from that sequence. This is a somewhat tricky counting problem. Think of the algorithm you'd have to write to generate a sequence of integers in order when the integers are represented by sequences of digits – but then skip some that are considered invalid.

Here is a recursive counting algorithm. To "count a sequence of buttons" is to return all possible character strings that can be generated from it. To count an N-character sequence of buttons:

```
If N is 1
  Create a set of singleton strings using the
    possible characters from that button.
else
  for each possible character c from the first button of the sequence
  {
    Call the set you get from counting the sequence made from the
      2nd through the Nth buttons s
    for each string t in s
    {
      Place c at the beginning of t, and place that string in the
        set to be returned.
    }
  }
```

Notice that this generates all possible strings, be they valid words or not. You must then go through all the words in the set and eliminate the invalid ones. You are then left with a small set of valid words through which the user can rotate when he or she hits the next button.

Note that you cannot eliminate invalid words as you go, because prefixes of valid words may not be valid. In addition, every time the user hits an additional button, the entire process must be repeated, because the set of valid words is now entirely different.

Seem complicated? It is. There are several more interesting, and efficient, ways to solve the problem. Feel free to discuss this in the course discussion topic for this homework on myCourses.

Some possibilities include a linked hash table, linear regular expression matching, use of the LINQ functions in .NET (To be covered in lecture later), and a variation on the trie associative data structure. Some of these choices favor ease of development, others performance.

4.1 Constraints

For this project you must create a WPF project and use XAML to specify the user interface. (The XAML code does not have to be handwritten.)

You must make a reasonable attempt to set the user interface from the model. Full MVC-style is not required; though an attempt must be made to adhere to the principals of the pattern.

5 Submission

This project is to be submitted by zipping (not RAR, not 7zip, or anything other new fangled format, just plain zip) your entire VisualStudio solution directory into a file named lab2.zip and putting it into the appropriate drop box in myCourses.

6 Grading

There is a rubric available for this assignment, which you should review.