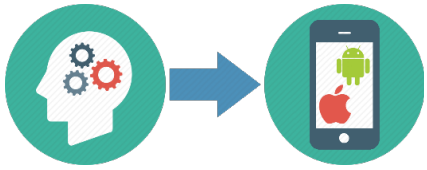
Mobile App Development

CS 4153



Weekly Assignment 02

Last updated: Aug 28, 2018 1:30 PM Due: Sep 6 11:59PM

Using text fields, buttons, outlets, and actions, create an iOS 12 app that counts points for a game of dominoes.

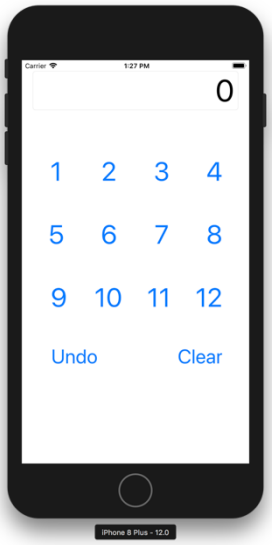
Dominoes (or **dominos**) is a game played with rectangular "domino" tiles. The domino gaming pieces make up a domino set, sometimes called a deck or pack. Each domino is a rectangular tile with a line dividing its face into two square ends. Each end is marked with a number of spots (also called pips, nips, or dobs) or is blank. The backs of the dominoes in a set are indistinguishable, either blank or having some common design. A domino set is a generic gaming device, similar to playing cards or dice, in that a variety of games can be played with a set.

– http://wikipedia.org/wiki/Dominoes

In most games of dominoes, the winner is the player (or team) with the fewest collective spots on the dominoes still in their hand(s). So, if one ends up with a lot of dominoes in hand, then counting the spots can be tedious and error prone.

Domino sets usually are *double-six* (meaning they contain double-blank through double-six dominoes and all combinations between), *double-nine*, or *double-twelve*. Your iOS app is to help count spots (points) for any set up to double-twelve.

When your app is started, it should look something like the image to the right.
(I have increased the font size on the UI components beyond the default sizes.)



1. Create a new, Swift 4.2 project in Xcode 10-beta; its UI should be formatted for the iPhone 8 Plus. Name the project
   W02_lastName_firstName

2. The text field contains the current spot count. When the app begins, this count is “0” (zero). The value is right-justified, as it is in many calculators.

3. There are 12 numeric buttons, representing the non-blank number of spots on one end of a domino. When the user taps a numeric button, the number on the button is added to the total in the text field, and the new total is displayed.

4. The user taps the *undo* button to undo the effect of the previously-tapped numeric button. When the user taps *undo* twice in succession, the effects of the previous two numeric button taps are undone. And so forth for three and more successive taps of the *undo* button.
    - Hint: For *undo* to work properly, you will keep a list (perhaps an array) of previously-tapped numeric buttons, adding to and deleting from the list as numeric and undo buttons are tapped.

5. The *clear* button resets the app back to its initial state.
    - This implies that the list of previously-tapped numeric buttons will be reset, as well.

General notes

- You can initialize the text field by entering a value in the *text* box of its attributes inspector (in the utilities area). The *alignment* menu – also in the attributes inspector – permits you to left-, right-, or center-align the text in the field.
- Any variables you need to share among the class methods (such as those for keeping track of a running total or an array of previously-tapped values) should be declared as class *properties* (i.e., member variables of the class). As with most other OOP languages, these are declared inside the class, but outside the methods. Do not declare them as global variables.
- If multiple buttons perform the same function, you can add an action to the first button, and then duplicate that button so that all of the other buttons use the same action method. For example, in this project, the method associated with a number button could retrieve its button text (e.g. "12"), convert the text to an integer, add the integer to the total, and display the updated total, and other actions, as needed; that way, buttons 1 through 12 all could use the same action method rather than having 12 identical action methods, one per button.
- Feel free to modify font family, font size (as I did in the image, above), font color (be tasteful), or the background color of any UI components.
- http://stackoverflow.com is a good source for answers to programming bugs that you may encounter.

Submitting your solution

- To compress a file or folder on macOS, right-click on the file or folder name, and then select *Compress …* from the context menu that appears. A *zip* file with the same prefix as the file or folder will appear.
- After closing your project in Xcode, compress (*zip*) the project folder and submit it to the appropriate dropbox on the course BrightSpace page.