Lab – Bubble Game

OVERVIEW

Date Due: Thursday, Feb. 25th by 11:59 p.m. EST.   ✧   **Value:** 40 points

Objective: Build a fully functional game!

This week you will learn to work with views, controls and event basics. Additionally, you'll get practice using Interface Builder and creating an application using Xcode. You are going to be provided with some starter code, and expected to implement the remainder of the application logic. The game, is called "BubbleGame".

The application has the following main features:

  - Taps in empty screen areas will create a new bubble with random graphics and movement
  - Taps on a bubble will "pop" the bubble
  - A settings screen allows users to control the initial velocity of new bubbles

Begin by downloading http://www.cs.umd.edu/class/spring2010/cmsc498i/files/labs/lab3_resources.zip. The resources zip contains source and other things to get you started.

  - BubbleInfo.[hm] represent information about a single bubble, and can be imported unmodified
  - BubbleCollection.[hm] represents a collection of BubbleInfo objects, and can be imported unmodified. Besides storing BubbleInfo object, it also manages the game simulation by updating the position of your bubbles over time.
  - Bubble-xxx.png graphics are going to be the images we use for our bubbles
  - TabBarIcon-xxx.png graphics are going to be used as the icons in our tab bar

The zip also contains contains some starter code that you will need to modify. That is, some functionality is already implemented, but you need to add your own code to complete the project. Besides adding your own code and methods, there are some stubbed out methods you should implement. These can be found by searching for "IMPLEMENT-ME".

  - BubbleGameView.[hm] is a subclass of UIView where all the bubbles will go
  - BubbleGameViewController.[hm] will manages the tabbar, game view, and settings

To complete this assignment, you will do the following: set up an xcode project, import the pre-written code and supplied resources, set up UI and connection in IB, write some code for BubbleGameViewController and BubbleGameView, and finally have some fun playing a game! At the beginning of this lab, you will see a demo of the finished product. For future reference, screenshots have been included on the next page.

Note on grading: First and foremost, you need to write a working and bug free application. Grading will also take into acount good design and code readability. Finally, going the extra mile and making sure your app is polished and covers all the edge cases you can think of will also be considered.

SCREENSHOTS

From Left To Right

- Main Screen before any bubbles exist on screen: Note the tab bar, and tap to play label.
- Bubbles!: Randomly chosen items, moving in random directions.
- Settings View: Sliders that control an additional "user multiplier" when picking random velocities.

INSTRUCTIONS

1. Download today's starter files.

2. Before writing any code

    2.1. Read the instructions all the way to the end

    2.2. Understand the what the starter code is providing to you by reading all of the ".h" files

    2.3. Scan the starter code ".m" files and find all the "IMPLEMENT-ME" comments

3. Create a new "View–based Application" in Xcode, and call it "BubbleGame"

4. Incorporate the sources you downloaded in step 1

    4.1. Replace Xcodes boilerplate BubbleGameViewController.[hm].

    4.2. Import BubbleGameView.[hm], BubbleInfo.[hm], BubbleCollection.[hm], and UITabBar_Additions.[hm]

    4.3. Import the graphics into the projects Resources (Bubble-xxx.png, and TabBarIcon-xxx.png)

    4.4. Import BubbleFactorySettings.plist into the projects Resources

5. Set up your user interface. Refer to the screenshots included in the handout

    5.1. Open BubbleGameViewController.xib. Note that the document window opens and comes with one empty view. This is where we are going to put everything – our tab bar, settings view and game view are going be children of this view. The game view and settings view will be swapped in and out in response to tab bar selections

    5.2. Tab Bar: Add a Tab Bar and set it's delegate to be the BubbleGameViewController (which is the "File's Owner"). You should also connect the BubbleGameViewController's tabBar_ outlet now.

    5.3. Settings View: Drag out a new custom view into your BubbleGameViewController.xib's document window. Configure it to look like the settings view in the screenshots. Make sure the height the view takes is 411 pixels since we don't want it to overlap with the tab bar.

    5.4. Game view: You can either drag out a new custom view in IB and set its class to be BubbleGameView, or create a BubbleGameView in code. If you choose to create the view in IB, drag a UIView from the Library Objects palette into your BubbleGameViewController document. Then all you need to do is bring up the Identify Inspector (Tools->Identity Inspector), and change the Class Identify from UIView to BubbleGameView.

    5.5. Tap to Play: BubbleGameView.h defines a "IBOutlet UIView *tapToPlayView_", and has code to show and hide the view for you. Create a UILabel, configure it to look like you want, and connect it to this outlet.

6. Set up additional outllets (IBOutlet) and actions (IBAction) in your BubbleGameViewController.[hm], and BubbleGameView.[hm] that you need. For example, you probably want to have an IBAction that the sliders can message when they are manipulated. Also, you'll want IBOutlets for your settings, and game views (unless you plan to create your BubbleGameView in code instead of IB)

7. Write code

    7.1. View switching. When the user changes the tab selection, you need to swap the game and settings view. In the future, we'll learn about controller that does this for us. For now though, it is instructive to see how this is done yourself.

    7.2. User scale factor. The intent of the sliders is to allow users to modify the velocities used when creating a new bubble. That is, velocity will be set to (random velocity value * user scale factor).

    7.3. Tap handling in BubbleGameView – adding bubbles. When the user clicks in an empty area, you will create a new BubbleInfo and add it to your BubbleCollection. Each BubbleInfo should end up with a corresponding view that will draw the bubble. It is recommended that you use UIImageView to "draw" the bubble.

    7.4. Simulation. BubbleCollection will update the position of its BubbleInfo objects if you tell it to start simulating time. To turn on the simulation, use the simulationActive property. You will notice that that BubbleCollection uses this property and the presence of any bubbles to turn on a periodic timer (no need to run the timer when there are no bubbles!)

    7.5. In response to BubbleCollection layout change notifications, you should update bubble positions. Make sure to read the "Useful Notes" section for some details about "normalized" coordinates and velocity. Setting the bubble view's coordinates will cause UIKit to automatically redraw the view at the new position. But first, what are notifications you ask? Find out about notifications in the "Useful Notes" section.

    7.6. Tap handling in BubbleGameView – popping a bubble. When the user clicks on a bubble, remove it and use the provided -animateBubblePop: to get the "pop" animation.

8. Get everything compile and build and have fun!

USEFUL NOTES

- **BundleFactorySettings.plist** – To load the "factory settings" plist, use `[[NSBundle mainBundle]` `pathForResource:@"BubbleFactorySettings" ofType:@"plist"]` to get the path of the plist.

- **Graphics** – To load a UIImage for any of the Bubbles, use [UIImage imageNamed:@"Bubble-xxx.png"]. The tab bar images can be used directly in IB's inspectors.

- **Normalized Values** – Notice that BubbleInfo.h defines its center coordinate and velocity as "normalized" values. Normalized coordinates are values which have been mapped, or scaled, from the view bounds to the range [0, 1]. That is, a center coordinate of (0.5, 0.5) represents a bubble in the center of the screen. When drawing bubbles on screen, you will need to map from normalized coordinates back to view coordinates. This may seem like a hastle, but it is good design to seperate the models and the views. Using normalized values, our models can operate without knowing anything about view coordinates, screen size, etc. For example, BubbleCollection will automatically remove a BubbleInfo from the collection whenever is no longer visible (represented by a value outside of range [0, 1]).

- **Randomizing**

  - **Bubbles** – To pick a random set of values for your bubbles, randoming pick an integer in the range [0, 2], and use that to dig default properties out of BubbleFactorySettings.plist.

  - **Velocity** – Once you've picked a bubble, randomize the default properties. For example, the user can declare the maximum velocity in the settings UI. To pick a random velocity based on this, you would let: `velocity = 'max velocity' * randomizer`, where randomizer is in the range [-1, 1]. To get a random value in the range [-1, 1], use: `randomizer = (rand() % 2 ? 1.0f : -1.0f) * (rand() / (1.0f + RAND_MAX));`

- **Notifications**

  - The BubbleCollection object manages a set of BubbleInfo objects and among other things runs a simulation that will update the state (position) of each BubbleInfo as the simulation runs. Whenever a bubble or set of bubbles moves, it will post a custom event in the form of an NSNotification which you can hear about.

  - Cocoa provides a mechanism whereby named events can be broadcast, and interested parties can watch for those events. Events are broadcast, or posted, as NSNotification's and are identified by an NSString name. Events may be observed by any object by telling the NSNotificationCenter the notification name, and the object you are interested in observing.

  - To observe a notification, you tell the NSNotificationCenter the notification name and object you are interested in observing (use *nil* for the object to mean any object). Additionally, you provide a method to be called when the notification is posted. To observe a bubble collection and have layoutInfoDidChange: called whenever BubbleCollectionItemLayoutInfoDidChange is posted, you would do the following:

    ```
    // Observe the BubbleCollectionItemLayoutInfoDidChange notification
    NSNotificationCenter *nc = [NSNotificationCenter defaultCenter];
    [nc addObserver:self selector:@selector(layoutInfoDidChange:)
             name:BubbleCollectionItemLayoutInfoDidChange object:[self bubbleCollection]];
    ```

  - You should stop observing notifications in your -dealloc and any other point that makes sense. To stop observing a notification, you would do the following:

    ```
    // Stop observing BubbleCollectionItemLayoutInfoDidChange for all objects
    NSNotificationCenter *nc = [NSNotificationCenter defaultCenter];
    [nc removeObserver:self name:BubbleCollectionItemLayoutInfoDidChange object:nil];
    ```