Tiny Tastes: Programmer Transfer Documentation

Document Outline

---

1. Overview

Tiny Tastes is an application developed for Apple's iPads using Cocoa Touch iOS SDK 6.0+.

2. Installation

Currently, the application is not uploaded on the Apple app store because the client wishes to continue developing the app and adding features.

The code is available at:
https://github.com/yjm607/CS408--Critter-Friends

Here are some resources from Apple that can help you in uploading the app to the app store:
https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/AppDistributionGuide/Introduction/Introduction.html
https://developer.apple.com/library/ios/documentation/LanguagesUtilities/Conceptual/iTunesConnect_Guide/1_Introduction/Introduction.html

3. Code Structure

3.1 Basic App Design

The application's code was developed using XCode, Apple's proprietary software development tool. iOS apps use Model-View-Controller design, in which the view controller class drives the app's main workflow. Apple provides some introductory readings on the view controller class:
https://developer.apple.com/library/IOS/featuredarticles/ViewControllerPGforiPhoneOS/Introduction/Introduction.html

The app does not require network access, and all data is saved and loaded locally. The default data storage object provided by Apple (NSUserDefaults) is used to store user data.

3.2 View Controllers and Other Important Classes

- HomeAppDelegate
    - Controls the launching and termination of the app. Initializes meal notifications and default custom timers when the app is first launched. Also displays a notification when it is received, either in-app or out-of-app.
- HomeViewController
    - Controls the home screen, which links to 5 different screens (Story Book, Let's Eat, Let's Drink, Tiny Shop, and Settings). Also shows the number of coins a user has earned thus far.
- TinyShopViewController
    - Controls the Tiny Shop screen. This feature is not yet implemented, but it will comprise the portion of the app where a user trades in his/her coins for in-app items, which are yet to be fully determined by the client.
- Settings portion
    - SettingsViewController
        - Controls the main settings screen, which has 4 different options: toggle background sounds on/off, toggle story narration on/off, set custom timers for meals, snacks, and drinks, or set daily meal reminder alerts.
    - CustomTimerViewController
        - Controls the screen that allows a user to set custom timers for meals, snacks, and drinks. Values are defaulted to client-specified times for each category.
    - SetReminderAlertsViewController

        - Controls the screen that displays 5 reminder notifications set at various points throughout the day. Values are defaulted to client-specified times for each category. Allows the user to toggle each notification on/off (but defaulted to on).
    - AddMealNotificationViewController
        - Controls the screen that allows the user to edit an existing meal notification. Once on this screen, the user can either save or cancel his/her changes, which brings the user back to SetReminderAlertsViewController.
- Photo portion
    - PhotoViewController
        - Controls the Photo View screen. The screen supports photo taking and re-taking. Once the photo is taken, it goes through cropping & resizing processes. Certain buttons only appear in specific instances for cleaner view.
    - OverlayView
        - Shows the overlay image when the camera is enabled. Also responds to "touch screen" action (calls PhotoViewController's captureNow method).
- EatViewController
    - Controls the Let's Eat screen. The images of the critter and the food are animated to simulate the eating action. The sound bits are played and stopped depending on the user action.
- ChooseDrinkViewController
    - Controls the Choose A Drink screen. The user is shown 5 different possible drinking devices and is asked to choose one. The user is also given the option to change the drink timer.
- DrinkViewController
    - Controls the Let's Drink screen. The images of the critter holding the drink is animated to simulate the drinking action. The sound bits are played and stopped depending on the user action.
- FeedbackViewController
    - Controls the screen where a user is given feedback depending on how much food/drink he/she consumed in the previous view controller. The user is also awarded a number of coins, the number again depending on how much food/drink he/she consumed in the previous screen. Two view controller link are present in this screen: a link back to the HomeViewController and a link back to the TinyShopViewController.
- Story portion

- StoryDataViewController
  - Responsible for rendering the Scene objects to screen. It takes all of the images, buttons, and whatever else may be in the Scene object in memory, and renders them.
- StoryModelController
  - Responsible for the backbone of the Story flow. It holds the collection of Scene objects and if the user wishes to progress forward, it will call upon the appropriate Scene to use next.
- StoryViewController
  - The overarching view controller that creates the StoryModelController and StoryDataViewController. It is also responsible for generating the "book feel" to the story mode. It also initiates the first scene of the story.
- Scene
  - Data representation of the scenes of the story, stored in story.xml. It holds all of the necessary components of a single scene including images and links, and can be customized to add more features for each scene such as text or music.
- SceneFactory
  - Calls for the creation of the Scene objects out of story.xml (the exact file name can be specified here) at the program initiation by calling on XMLDelegate and then returns the collection of Scenes to the StoryModelController.
- XMLDelegate
  - Responds to the parsing of the story.xml file and creates the Scene objects based on which actions it should perform when the file is parsed.
- story.xml
  - Contains all of the information for all of the scenes in DOM format. Scenes can be modified or created here.

3.3 Notable Program Designs

- Story Mode
  - The Story Mode is first triggered by the creation of the StoryViewController, which then calls the StoryModelController and the StoryDataViewController.
  - The collection of Scenes is stored in an array in StoryModelController.
- Camera API
  - AVFoundation Framework is used to control the camera.

    - The camera view disables screen auto-rotation.
- Sound
    - AVAudioPlayers are used to play client-recorded sound bites throughout the use of the app.
    - Sound is played within the Let's Drink, Let's Eat, and Feedback screens.
- The Use of NSUserDefaults
    - The NSUserDefaults class provides a programmatic interface for interacting with the defaults system. The defaults system allows an application to customize its behavior to match a user's preferences.
    - We've modified NSUserDefaults by storing key/value pairs for app information. For your convenience, here is a list of these keys:
        - Key: "HasLaunchedOnce"; Value: a boolean that represents whether or not the app has been launched before (i.e. if FALSE then this is the first time the app has launched, if TRUE then the app has already been launched)
        - Key: "HasLaunchedDrinkScreenOnce"; Value: a boolean that represents whether or not the Let's Eat screen has been launched before
        - Key: "HasLaunchedEatScreenOnce"; Value: a boolean that represents whether or not the Let's Drink screen has been launched before
        - Key: "mealTimer"; Value: an integer that represents the customized amount of time in minutes for the goal finishing time of meals
        - Key: "snackTimer"; Value: an integer that represents the customized amount of time in minutes for the goal finishing time of snacks
        - Key: "drinkTimer"; Value: an integer that represents the customized amount of time in minutes for the goal finishing time of drinks
        - Key: "backgroundSound"; Value: a boolean that represents the ON or OFF setting for background sounds
        - Key: "storyNarration"; Value: a boolean that represents the ON or OFF setting for story narration sound
        - Key: "coinsKey"; Value: an integer that represents the number of coins a user has earned thus far in the app
    - To look up the value of a key (let's use the key "backgroundSound" for this example), simply write: *[[NSUserDefaults standardUserDefaults] boolForKey:@"backgroundSound"]*. Similarly, to set the value of this key, write: *[[NSUserDefaults standardUserDefaults] setBool:YES forKey:@"backgroundSound"]*.

4. Added Frameworks

AVFoundation: the AVFoundation framework allows sound and video controls. For instance, the sound bits in the Let's Eat screen are played using the framework. Also, the Photo View screen uses the framework to implement custom camera controls .

5. Dependencies

Automatic Reference Counting: Apple introduced the ARC with iOS 5. the ARC is the same as automatic garbage collection. To make the app compatible with lower versions of iOS, all references must be manually released. This requires much re-work on the existing code, and is not recommended unless absolutely necessary.

6. Modifying the App

6.1 Adding Content to the Story

To add scenes to the story, you will need to edit story.xml located in the "Tiny Tastes" folder. The format is relatively straightforward, and looking at the previous examples, you should be able to pick up the basic idea. Each scene is contained in its own tag, beginning with <scene> and ending at </scene>. The scene id is used to identify the scene when used in links, which will be explained shortly. Under each scene tag, there are two tags that can be added -- images and links. Note that you can have multiple of each tag, for example, multiple image tags and multiple link tags.

The image tag has five properties -- path, x, y, w, and h. The path property is used to tell what each image file is named in the computer system, including the file extension. The x tag is used to tell the x-coordinate of the top-left corner and y tag is used to tell the y-coordinate of the top-left corner. The w and h tags tell the width and height, respectively, with which to draw the image.

The link property works very similarly to the image tag. The x and y tags tell the x and y coordinates of the top-left corner of the button, and the w and h tags tell the width and heights of the button as well. The link's id property tells which scene, correlated by the scene's own id property, will be triggered by clicking on the button.

If you wish to change how the properties are read or perhaps even create new properties and new tags, you can modify XMLDelegate.m file to perform certain actions when parsing the XML file.

6.2 Adding and Replacing Images

To add an image, go to File -> Add Files to Tiny Tastes. After including the file in the project, adding the image can be done in either the storyboard or in one of the ViewControllers. When adding images, it is highly recommended that the image size be adjusted before use. Dynamic resizing takes much processing power and memory, which can cause the app to crash. It is also recommended that images are saved as *.jpg files, as it is a more compressed file size format than other image extensions, which enables better app performance.

To replace an image, delete the existing image from the project and add one with the same file name. Remember to adjust the image size appropriately before adding the replacement.

7. Updating the App

Currently, the app is deployed under a profile named "CompSci 408 F13." This profile is managed by Professor Lucic (lucic@cs.duke.edu), and the app can be transferred to another development profile with his consent.

Apple provides a guideline on submitting the app to the app store:
https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/AppDistributionGuide/SubmittingYourApp/SubmittingYourApp.html

For development and testing purposes, the app can be deployed to physical devices without going through the app store. To do so, follow these steps:

- Enroll in the iOS Developer Program (https://developer.apple.com/devcenter/ios/index.action). The approval can take several days.
- Create a Certificate Signing Request
- Create a Development Certificate
- Add the testing device in the Provisioning Portal
- Create an App ID
- Create a Provisioning Profile
- Set code signing identity in Project Navigator to iPad developer
- Select the device in Xcode (instead of the iOS simulator), and build.

A more detailed instruction can be found here:
http://mobile.tutsplus.com/tutorials/iphone/how-to-test-your-apps-on-physical-ios-devices/