The University of Nottingham

UNITED KINGDOM · CHINA · MALAYSIA

In-Vehicle Music Player Interface Design and Implementation

Zongqi Wang

ID: 4219232

School of Computer Science

University of Nottingham

I hereby declare that this dissertation is all my own work, except as indicated in the text:

Signature ______________________

Date _____/_____/_____

I hereby declare that I have all necessary rights and consents to publicly distribute this dissertation via the University of Nottingham's e-dissertation archive.

Table of Contents

1 Abstract

This project contains the design of new interface for in-vehicle information system (IVIS). The music player interface was fully developed as an application for iPad. Experiments were conducted to evaluate the interface. The results showed that the interface was effective and required less visual demand to perform operations, such as change volume and change music, than traditional interface.

2 Introduction

2.1 Motivation and purpose

In the first meeting, my supervisor expressed that he wanted a real application that could run on a capacitive touchscreen device for research purpose. The application would be used in a driving simulator to create some secondary tasks for the driver to complete. The application need to play background noise and to provide a music player interface where driver could change the volume and music. My interest in this project is to explore the possibility of designing a user interface that requires less visual demand than traditional interface. In order to achieve this, gestures were considered to be the promising solution. A study conducted in the University of Nottingham revealed the possibility of applying swipe gestures on in-vehicle touch screens (Burnett, Crundall, Large, Lawson & Skrypchuk, 2015). Therefore, I want to implement a music player interface on a touch enabled device and then conduct experiments to study the performance of the swipe gestures in real application.

2.2 Background

With the development of touch screen devices, it is common to have in-vehicle information system (IVIS) that takes advantages of this technique. Traditional interfaces heavily relied on the physical buttons. This physical constraints limited the extendibility. The applications installed in the touch screen device are not limited by the unchangeable constraints. A touchscreen is a combination of view and control. With new interface designs, the interactions between the user and the device would also be modified. However, the touchscreen also has disadvantages: it requires more visual demands than the traditional physical interface. Enormous amount of researches were conducted in this area, mainly focused on the usability and safety issues. In 2014, Kim and Song conducted experiments to evaluate various common used gestures in IVIS. According to the result, the pan gesture (a swipe gesture is a sub-category of pan gesture), which required least visual demand, was the most suitable gesture in the IVIS during driving. A research has been conducted by Kim, Kwon, Heo, Lee and Chung in 2014 to investigate the relationship between the size of button on touchscreen and safety and usability. In the result, they claimed that the usability and safety would be optimal at button size 17.5mm in their study.

3 User Story

The user story was represented in the format "As a <stakeholder>, I want <outcome>, so that <value>.". Discussions and interviews were made to make the user story complete. As this project was initially proposed by the researcher, therefore the product of this project would be more research orientated.

1. As a researcher, I want the application to run on a touch enabled device so that I could place the device in the driving simulator for test.
2. As a researcher, I want the application to play fan noise in the background so that the driver in the test would be more likely to feel that he/she was in reality.
3. As a researcher, I want the application to be able to play music so that in the driving simulator the driver could casually listen to the music.
4. As a researcher, I want the application to cause as little visual distraction as possible so that the driver would be more likely to focus on safe driving in the simulator.
5. As a researcher, I want the music player interface to provide operations including: increase volume, decrease volume, play next song, play previous song, play / pause the current song. These operations would act as additional tasks in the driving simulator in experiments.

6. As a user, I want the music player provide a clean interface where I can easily control volume and switch songs.
7. As a user, I want the music to contain a collection of music, such as classical music, rock music and etc.

4 User Requirement

UR1: The product of the project shall be able to run on touch enabled devices. E.g. Android tablet, iPad and touch enabled windows notebook.

UR2: The product shall present a set of functions for users to navigate and choose within 10 seconds.

UR3: The product shall have the function to play fan noise in background.

UR4: The product shall have the function to stop fan noise that is playing in the background.

UR5: The product shall have the function to play and pause music.

UR6: Users could change the volume of the music player within 10 seconds.

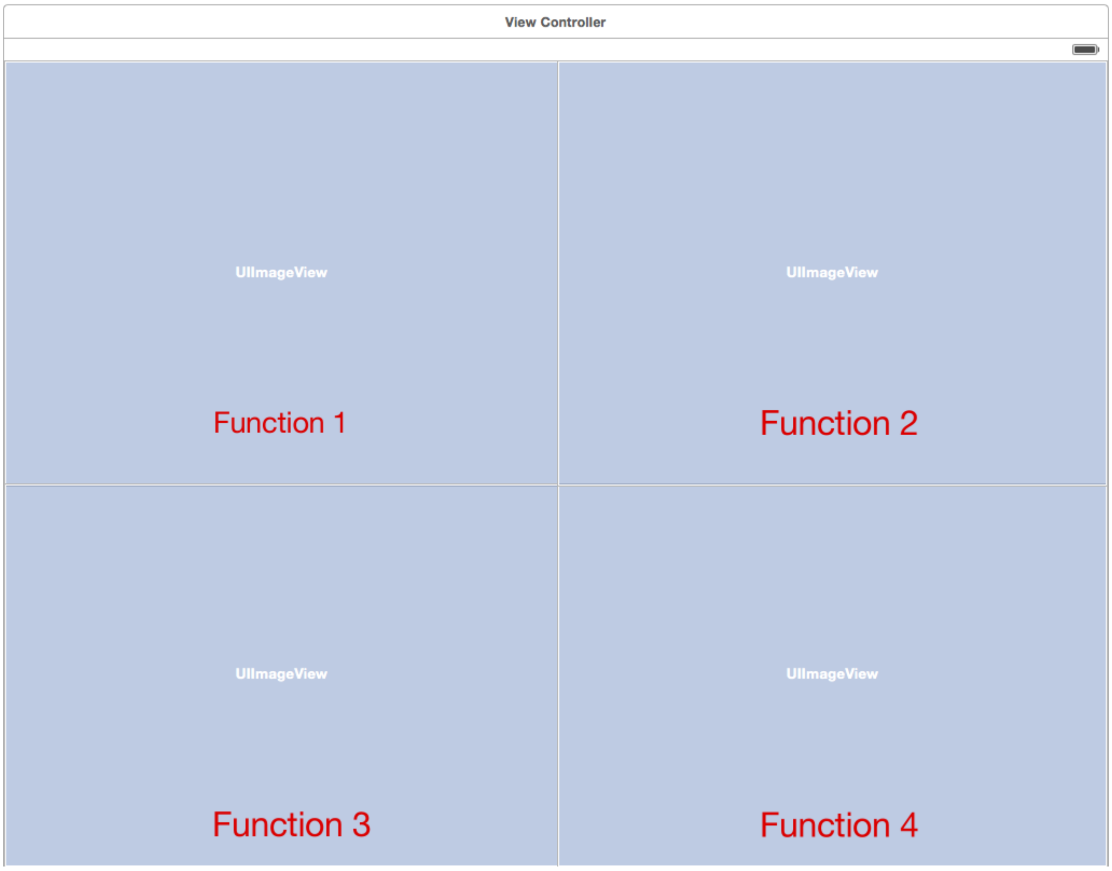
UR7: Users could choose to play next song or play previous song within 10 seconds.

UR8: The complete product shall have at least 40 songs with different music style, such as light music, pop music and etc., in the application.

5 Prototype

The prototype was created by software XCode in the “storyboard”. The prototype would mainly focus on the interface design and user interfaction. The functionalities would be implemented in the implementation phase.

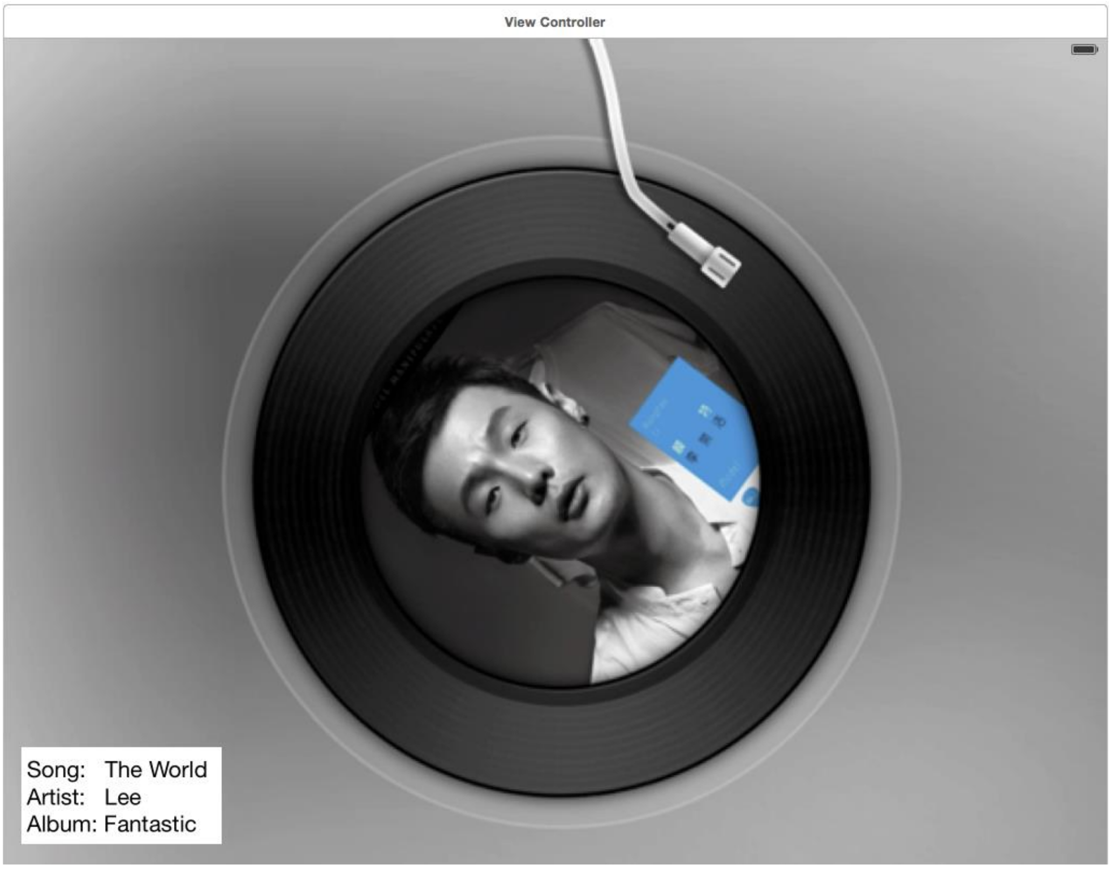
5.1 Centre interface



Graph5-1: Centre Prototype

As stated in the user requirements, the application needed to present an interface to hold several functions. Traditionally, the interface could be look like the "Launch" interface in Mac, where 5 x 7 tiles would be placed in the screen for user to choose. However, this application was designed to run on an iPad, the screen size was comparably smaller. In order to present the tiles more clearly, the size of the tiles was designed to be nearly ¼ size of the screen. In this way, users would only see 4 functions in this interface. Therefore, it was necessary that the interface shall provide a "scroll to navigate" function so that the user could scroll in direction left or right to view more functions. To facilitate the navigation, functions were grouped as pages, and each scroll would present the previous or next group of functions as a new page.

5.2 Music Player Interface



Song: The World
Artist: Lee
Album: Fantastic

Graph5-2: Music Player Prototype

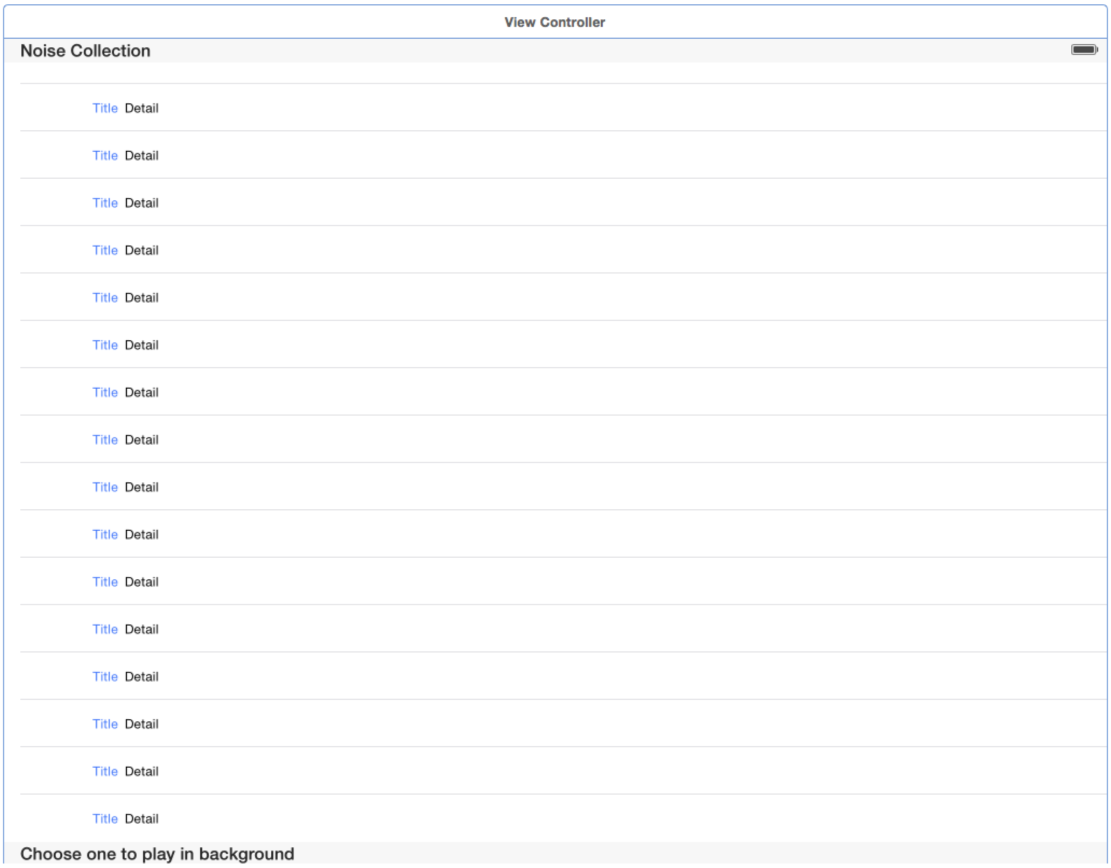
The graph5-2 was the initial design for the music player interface. At the centre of the screen is a black disk. At the center of the black disk is the CD image of the song. On the bottom left was the related information of the music, including the music name, artist and album. The background was designed to be stretched CD image with blur effects applied.

To play the music, user needs to tap the CD at the center of the screen. When the music is playing, the CD would rotate in clockwise direction. The rotation animation was designed to give feedback to the user that the music is playing. In this way, user would feel more confident with the “play” operation. To pause the music, user needs to tap the CD while the music is playing. Once the music is paused, the rotation animation would stop immediately. With this type of instant feedback from the application, the users would be confident with the play/pause operation, and result in good user experience.

To play next or previous song, user needs to perform the swipe gesture in direction left or right accordingly. In traditional interfaces, two buttons with arrow directions would be required to be associated with corresponding functions: the user needs to press the button with left arrow to play previous song, or press the button with right arrow to play next song. In the prototype, in order to decrease the visual distraction, swipe gestures were considered to be appropriate to replace the buttons. With the gestures, user could simply touch anywhere inside the screen and perform a swipe with direction left or right to change the sound track to the next or previous one.

To change the volume of the music, user needs to perform the swipe gesture in direction up or down accordingly. Two designs were frequently applied in the traditional interface: 1, two buttons were required. One button for decreasing the volume and the other for increasing the volume; 2, a slider was required. A circle shaped button was placed on a horizontal line, indicating the current volume. User could change the volume by pressing on the button and then dragging it to left to decrease the volume or to right to increase the volume. In the prototype, the solution became quite simple. User only needs to perform a swipe gesture from anywhere inside the screen. The distance, direction and speed of the gesture together determine the amount of change.

5.3 Background Noise Interface



Graph5-3: Fan Noise Prototype

The graph5-3 showed the initial design of the fan noise interface. The purpose of the interface was for the researcher to create background noise in driving simulator. Therefore, it was clear that the participants in the simulator do not need to interact with this interface directly. The interface was presented as a table, with each row representing a sample noise. The researcher could scroll vertically to navigate the whole collection of the samples. By tapping one of the rows in the table, a sample noise would be played in the background. To stop the sample noise from playing, the researcher simply need to tap other sample noise.

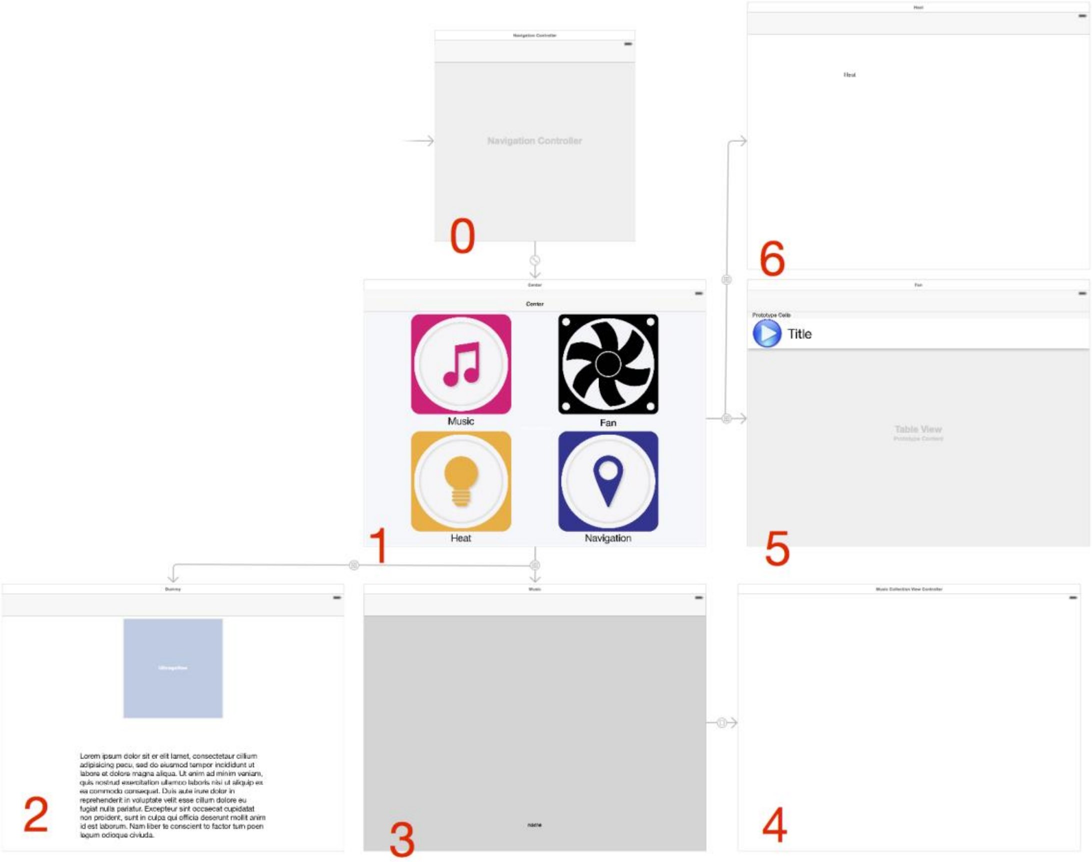
6 Implementation

This chapter would firstly introduce the related interfaces briefly. Then the related code would be presented as pictures to show how the key functionalities were supported.

6.1 Deployment environment

The product of this project is a complete IOS application that is designed to run on all iPads with version 8.0 or later in landscape right orientation. XCode with version 6.4 (6E35b) was used in this project. The development language is the new language Apple promoted in the past few years: Swift with version 2.0. The APIs used in the project includes Foundation, UIKit and AVFoundation.

6.2 Interface Overview



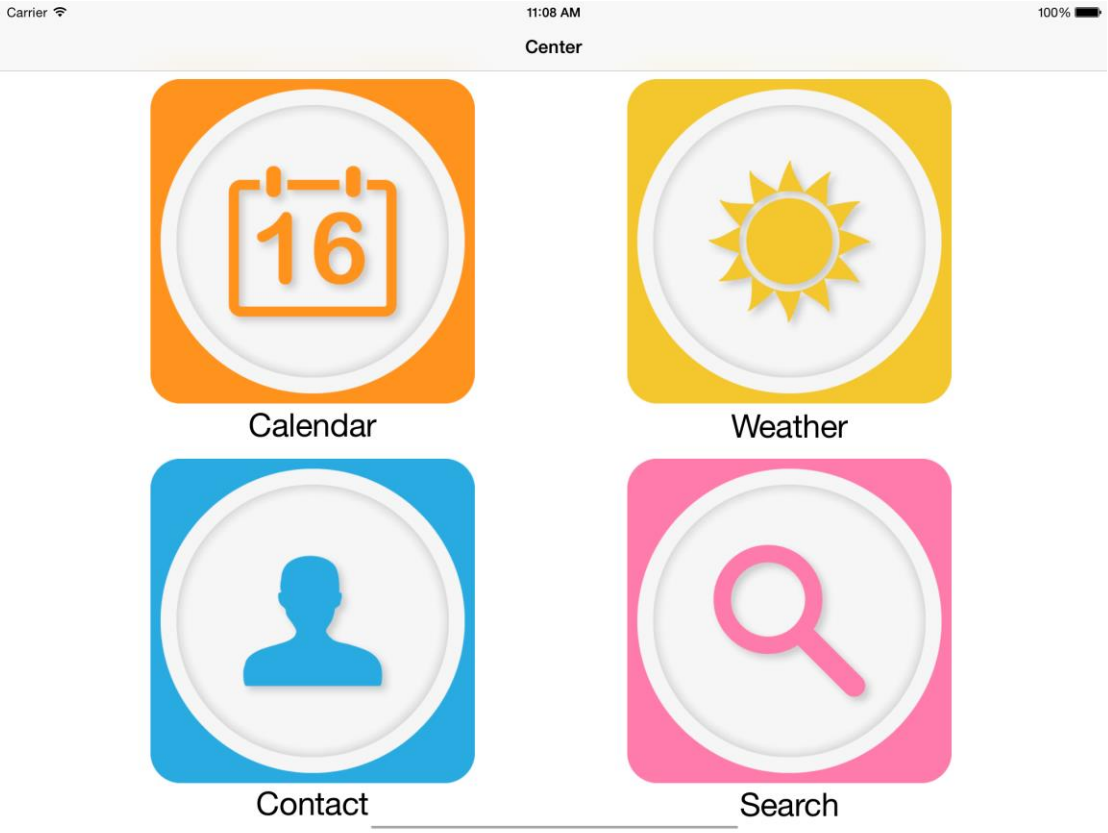
Lorem ipsum dolor sit er elit lamet, consectetaur cillium adipisicing pecu, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Nam liber te conscient to factor tum poen legum odioque civiuda.

Graph6.2-1: Interface Navigation Overview

As presented in the Graph6.2-1, this application contains 6 scenes and 1 navigation controller. All the 6 scenes were embedded in the navigation controller so that the user could easily navigate through the all the scenes by default.

6.2.1 Scene 1: Centre Interface.

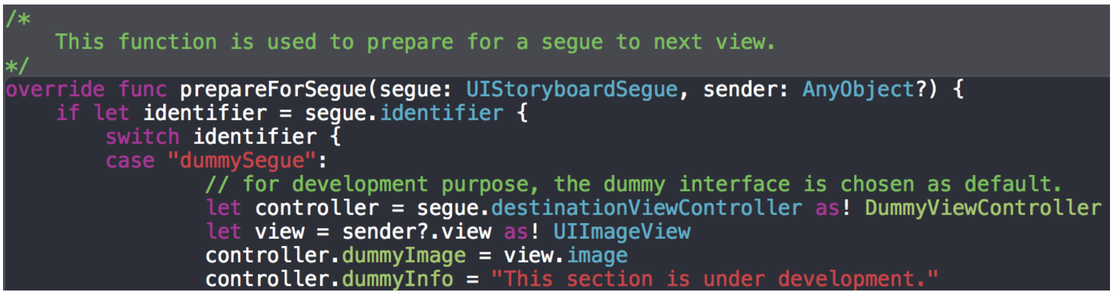
This interface was designed to hold multiple icons, for each icon represents a function. Users could scroll to navigate all the functions in this interface. In the implementation phase, in order to facilitate the navigation, the scroll function was configured to scroll as pages so that for each swipe, users could navigate to the next or previous 4 functions. As the graph6.2.1-1 showed. Users has just arrived the second group of 4 functions, the scroll bar stopped automatically to present the 4 functions as a page. The related code would be explained in the data flow with other classes.



Graph6.2.1-1: Centre Interface Overview

6.2.2 Scene 2: Dummy Interface.

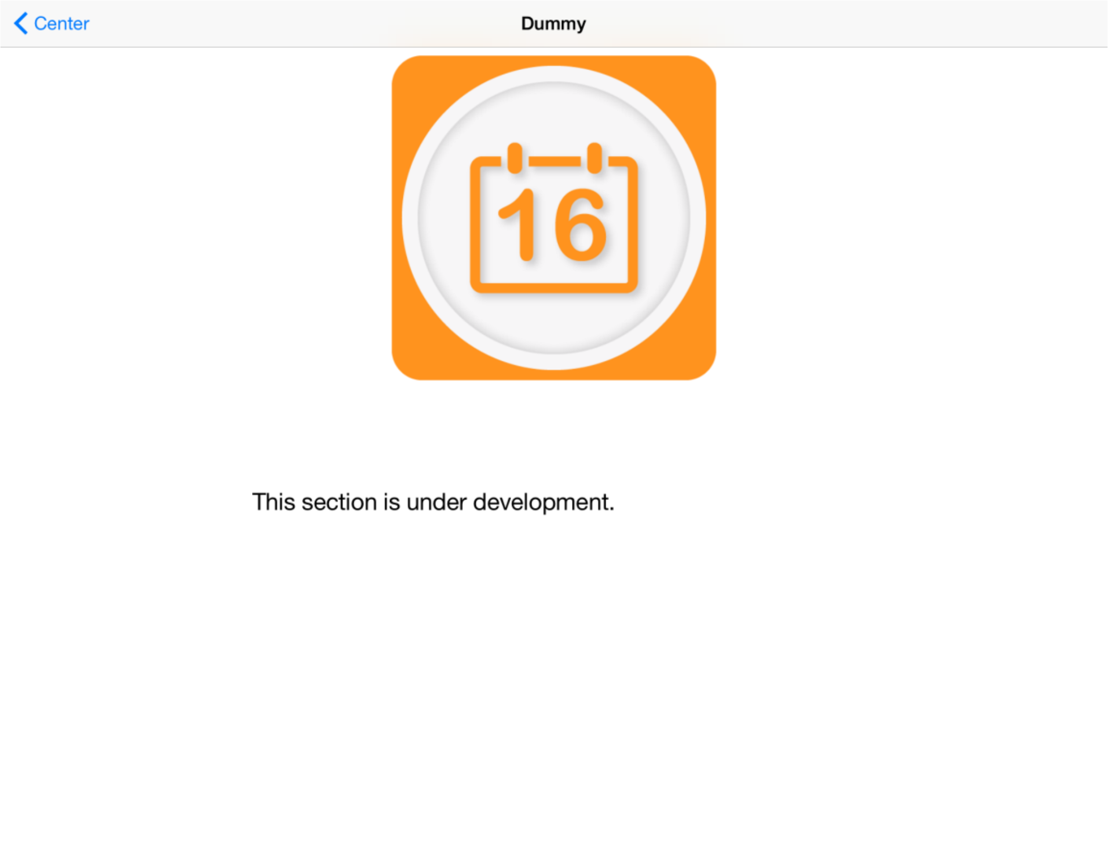
The dummy interface was designed to represent the functions that were not implemented in this project. The content of the interface was actually nothing, but it was important in the development process.



```
/*
    This function is used to prepare for a segue to next view.
*/
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if let identifier = segue.identifier {
        switch identifier {
        case "dummySegue":
                // for development purpose, the dummy interface is chosen as default.
                let controller = segue.destinationViewController as! DummyViewController
                let view = sender?.view as! UIImageView
                controller.dummyImage = view.image
                controller.dummyInfo = "This section is under development."
```

Graph6.2.2-1: Dummy Interface Segue Sample Code

A segue is a link in the navigation control system that links all interfaces together. For the functions that were not developed in this project, dummy interface would be applied temporarily.

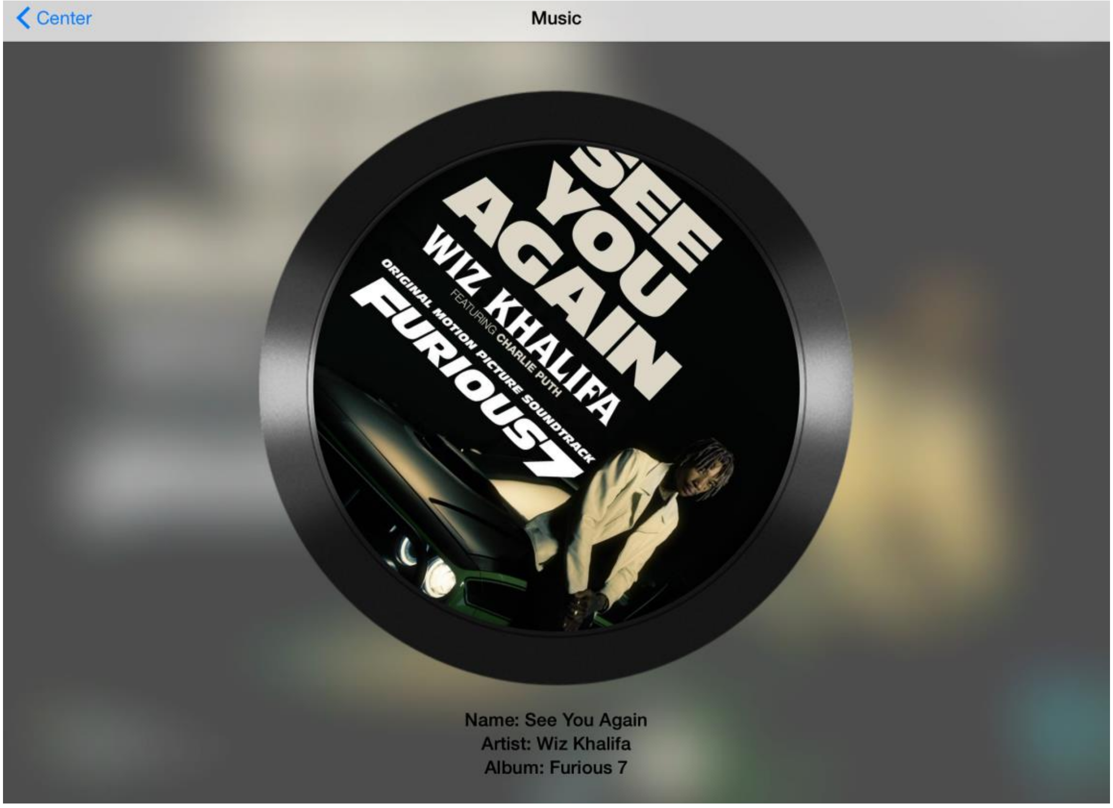


This section is under development.

Graph6.2.2-2: Dummy Interface Overview

6.2.3 Scene 3: Music Interface.

The music interface is the core of this project. Graph6.2.3-1 is the screenshot. The interface contains a background image with blur effects applied. On the central part of the screen is the CD image of the song that has been modified so that the image is in circle shape. When the music is playing, the CD image would rotate slowly in clockwise to imitate the reality that a CD player is working. Below the CD image is the texts showing the related information of the music, including the name, artist, and the album of the music. Multiple gestures were supported in this interface to allow users to control the music player. Drag left to play next song; drag right to play previous song; scroll up to increase volume; scroll down to decrease volume; tap the CD to play or pause the song; swipe from the right edge to navigate to music collection interface; swipe left to navigate back to the centre interface.

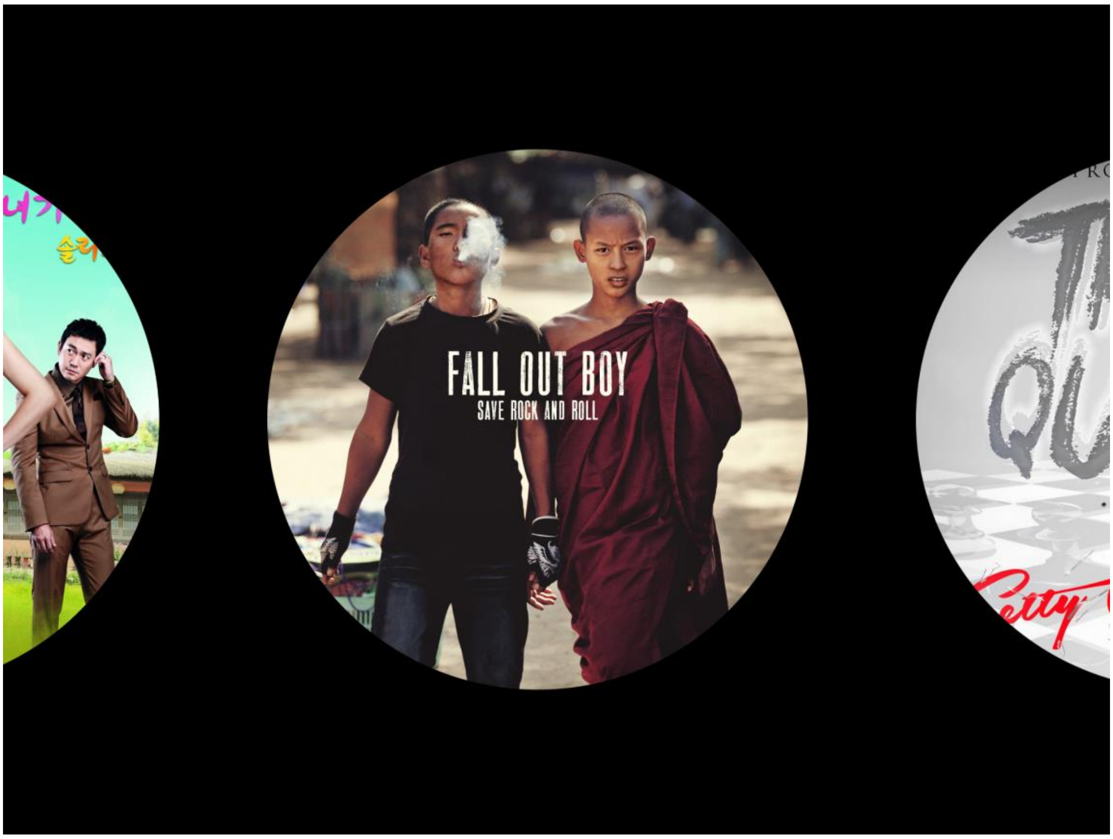


Name: See You Again
Artist: Wiz Khalifa
Album: Furious 7

Graph6.2.3-1: Music Interface Overview

6.2.4 Scene 4: Music Collection Interface.

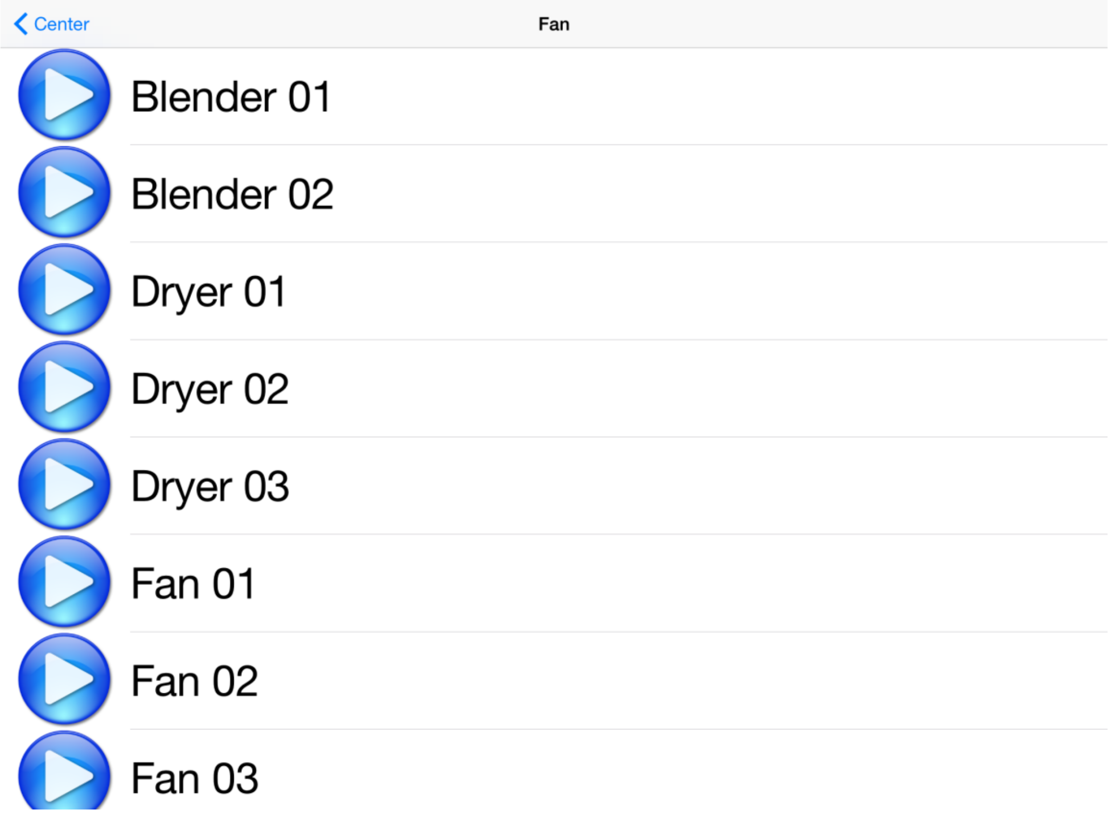
Graph6.2.4-1 is the screenshot of this interface. This interface provided a collection of CDs where users could scroll to view the CDs. By tapping one of the CDs, the application would transfer the user interface to the Music Interface and play the selected music. Due to the fact that is function is not required in the user requirement, as an additional function it is sufficient. Further improvements would include: 1, add labels for each CD; 2, add sorting and filter functionalities to facilitate music selection.



Graph6.2.4-1: Music Collection Interface Overview

6.2.5 Scene 5: Background Noise Interface.

Graph6.2.4-1 is the screenshot of the interface. This interface basically provided a collection of noise sample in table format. Users could scroll to view all the noise samples, and choose one of the samples to play in background, as required in the user requirements.



< Center

Fan

Blender 01

Blender 02

Dryer 01

Dryer 02

Dryer 03

Fan 01

Fan 02

Fan 03

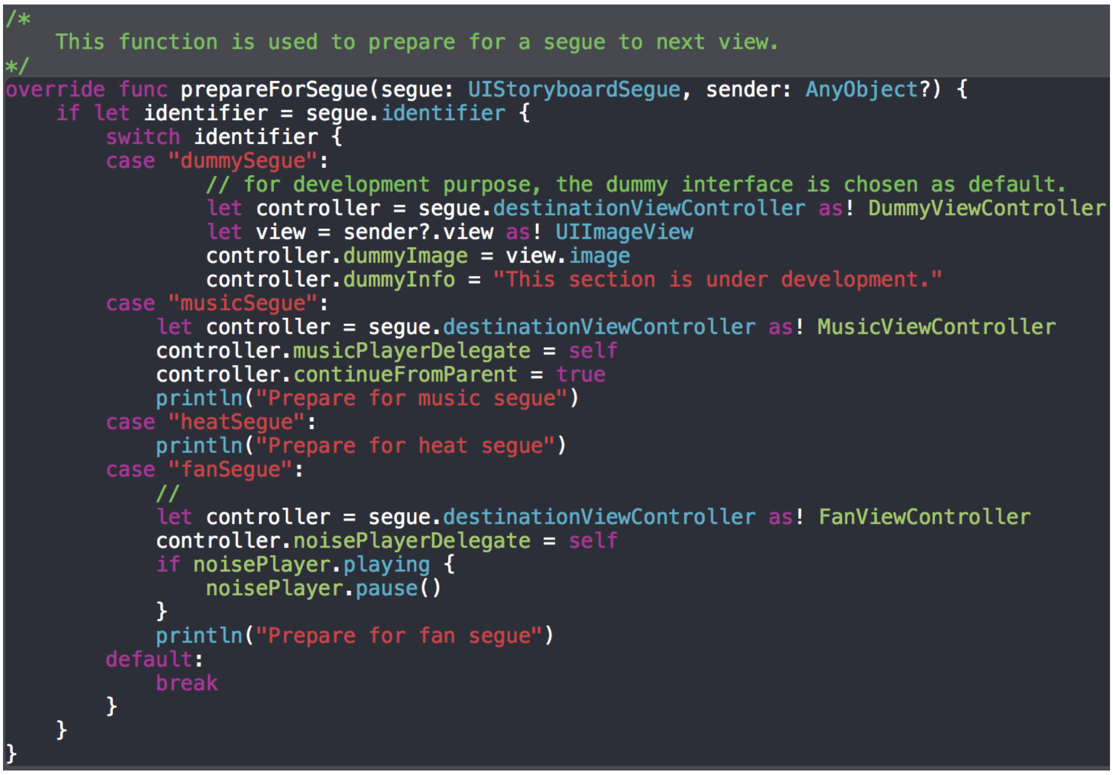
Graph6.2.4-1: Background Noise Interface Overview

6.3 Function implementation

This section would mainly focus on the functionalities that related to the music player interface.

6.3.1 Segue and unwind

Segue is the key to link all interfaces together in the navigation controller. When an interface is about to change to an interface that linked by a segue, the function “prepareForSegue” would be called. Graph6.3.1-1 presented sample code in screenshot.

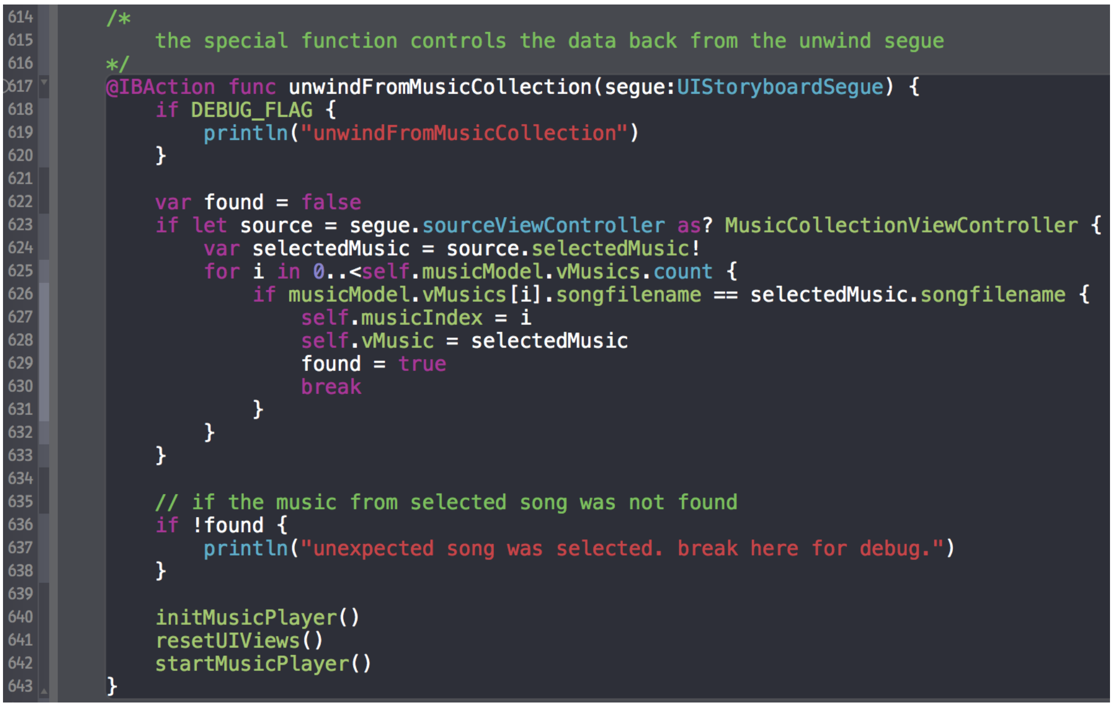


```
/*
    This function is used to prepare for a segue to next view.
*/
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if let identifier = segue.identifier {
        switch identifier {
        case "dummySegue":
                // for development purpose, the dummy interface is chosen as default.
                let controller = segue.destinationViewController as! DummyViewController
                let view = sender?.view as! UIImageView
                controller.dummyImage = view.image
                controller.dummyInfo = "This section is under development."
        case "musicSegue":
            let controller = segue.destinationViewController as! MusicViewController
            controller.musicPlayerDelegate = self
            controller.continueFromParent = true
            println("Prepare for music segue")
        case "heatSegue":
            println("Prepare for heat segue")
        case "fanSegue":
            //
            let controller = segue.destinationViewController as! FanViewController
            controller.noisePlayerDelegate = self
            if noisePlayer.playing {
                noisePlayer.pause()
            }
            println("Prepare for fan segue")
        default:
            break
        }
    }
}
```

Graph6.3.1-1: Segue Sample Code

The sample code is a section from file “ViewController.swift”. In the sample code, the identifier of the segue was examined. E.g. if the identifier is “musicSegue”, then set the source class as “musicPlayerDelegate”. This enables the destination view controller to access the music player.

When the user selected a music in the music collection interface, the music collection interface would disappear and instead the music player interface would appear. This is the unwind segue: go back to the interface that created this interface. Graph6.3.1-2 is the screenshot of unwind sample code.



```
/*
    the special function controls the data back from the unwind segue
*/
@IBAction func unwindFromMusicCollection(segue:UIStoryboardSegue) {
    if DEBUG_FLAG {
        println("unwindFromMusicCollection")
    }
    
    var found = false
    if let source = segue.sourceViewController as? MusicCollectionViewController {
        var selectedMusic = source.selectedMusic!
        for i in 0..<self.musicModel.vMusics.count {
            if musicModel.vMusics[i].songfilename == selectedMusic.songfilename {
                self.musicIndex = i
                self.vMusic = selectedMusic
                found = true
                break
            }
        }
    }
    
    // if the music from selected song was not found
    if !found {
        println("unexpected song was selected. break here for debug.")
    }
    
    initMusicPlayer()
    resetUIViews()
    startMusicPlayer()
}
```

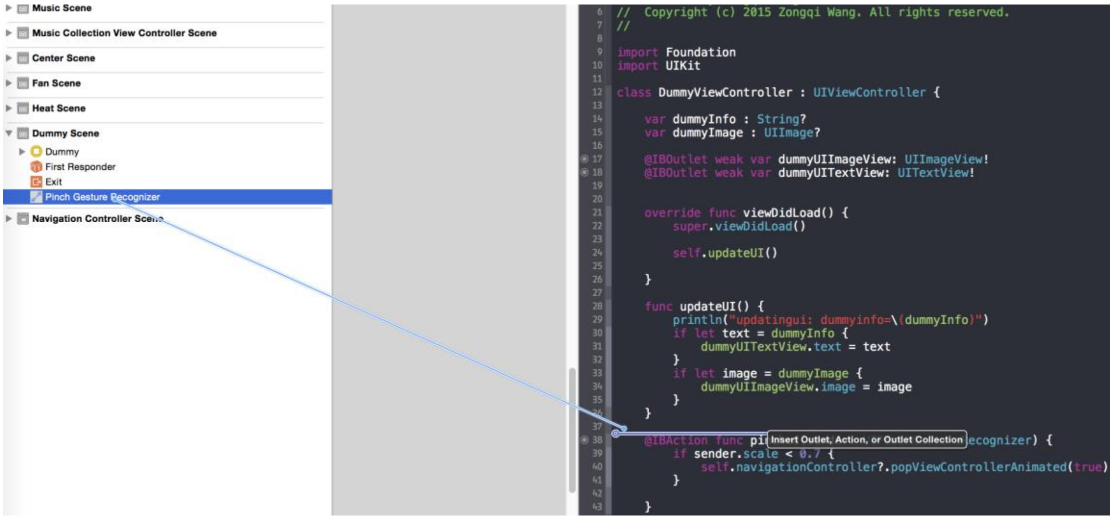
Graph6.3.1-2: Unwind Sample Code

The sample code demonstrated the unwind function that was used when the user selected a music in the music collection interface. The “selectedMusic” was saved in the source segue and was then used in a loop to match the index in the destination segue class. When the music was found, the loop broke and start to initialize the music player, update the corresponding UI and start playing the music.

6.3.2 Gesture Recognizer

There are plenty of gestures used in the project. The following sub-sections would go through each one of them with an example from this application.

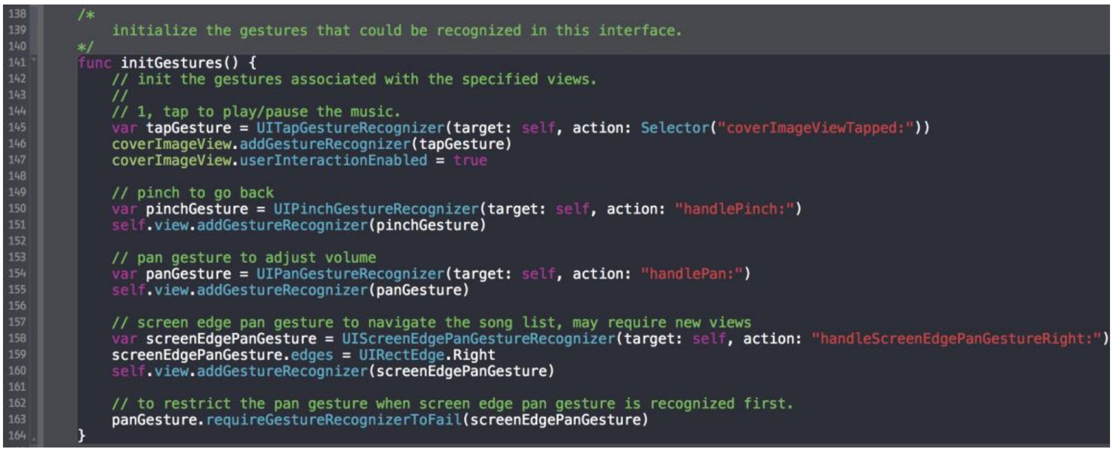
6.3.2.1 Setting up gesture recognizers


Music Scene
Music Collection View Controller Scene
Center Scene
Fan Scene
Heat Scene
Dummy Scene
Dummy
First Responder
Exit
Pinch Gesture Recognizer
Navigation Controller Scene
Insert Outlet, Action, or Outlet Collection

```
//  Copyright (c) 2015 Zongqi Wang. All rights reserved.
//

import Foundation
import UIKit

class DummyViewController : UIViewController {

    var dummyInfo : String?
    var dummyImage : UIImage?

    @IBOutlet weak var dummyUIImageView: UIImageView!
    @IBOutlet weak var dummyUITextView: UITextView!


    override func viewDidLoad() {
        super.viewDidLoad()

        self.updateUI()

    }

    func updateUI() {
        println("updatingui: dummyinfo=\(dummyInfo)")
        if let text = dummyInfo {
            dummyUITextView.text = text
        }
        if let image = dummyImage {
            dummyUIImageView.image = image
        }
    }

    @IBAction func pi ecognizer) {
        if sender.scale < 0.7 {
            self.navigationController?.popViewControllerAnimated(true)
        }

    }
```

Graph6.3.2.1-1: Set up gesture recognizer using storyboard

This sample code presented in Graph6.3.2.1-1 was a section from "DummyViewController.swift". This function was set to be invoked when the user performed the pinch gesture with two fingers. When the scale of the pinch decreased to 0.7 than its initial value, the "Dummy Interface" would automatically disappear, and the "Centre Interface" would pop up with animation instead.



```
138     /*
139         initialize the gestures that could be recognized in this interface.
140     */
141     func initGestures() {
142         // init the gestures associated with the specified views.
143         //
144         // 1, tap to play/pause the music.
145         var tapGesture = UITapGestureRecognizer(target: self, action: Selector("coverImageViewTapped:"))
146         coverImageView.addGestureRecognizer(tapGesture)
147         coverImageView.userInteractionEnabled = true
148 
149         // pinch to go back
150         var pinchGesture = UIPinchGestureRecognizer(target: self, action: "handlePinch:")
151         self.view.addGestureRecognizer(pinchGesture)
152 
153         // pan gesture to adjust volume
154         var panGesture = UIPanGestureRecognizer(target: self, action: "handlePan:")
155         self.view.addGestureRecognizer(panGesture)
156 
157         // screen edge pan gesture to navigate the song list, may require new views
158         var screenEdgePanGesture = UIScreenEdgePanGestureRecognizer(target: self, action: "handleScreenEdgePanGestureRight:")
159         screenEdgePanGesture.edges = UIRectEdge.Right
160         self.view.addGestureRecognizer(screenEdgePanGesture)
161 
162         // to restrict the pan gesture when screen edge pan gesture is recognized first.
163         panGesture.requireGestureRecognizerToFail(screenEdgePanGesture)
164     }
```

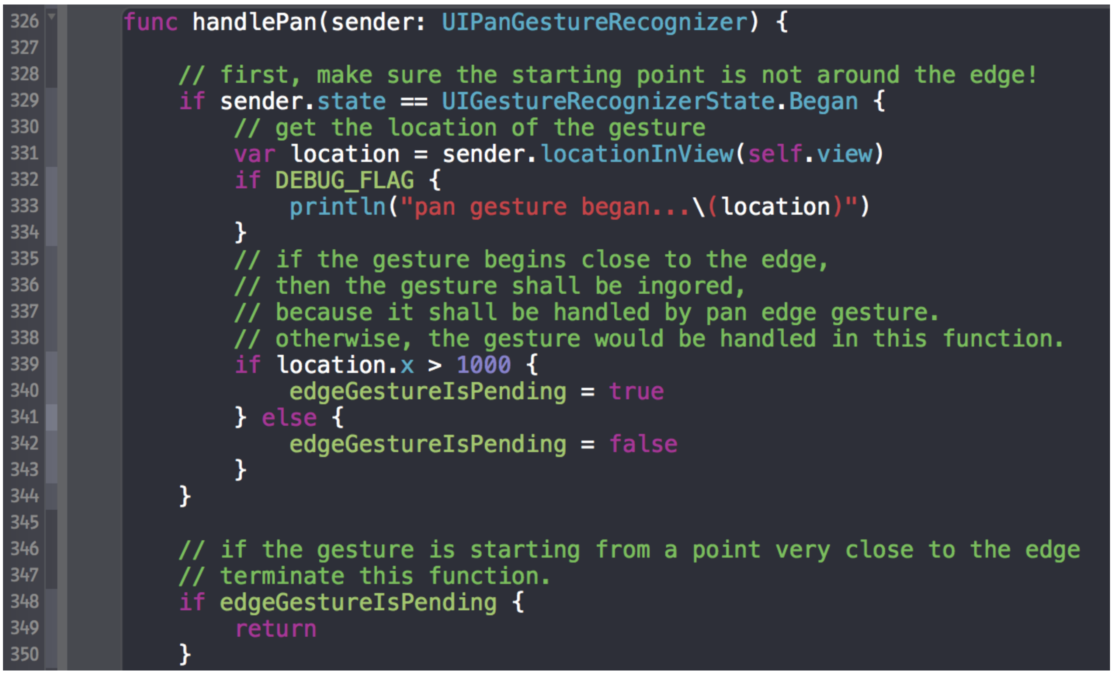
Graph6.3.2.1-2: set up gesture recognizers programmatically

The code presented in the graph6.3.2.1-2 came from "MusicViewController.swift". The sample code showed the way to programmatically create and link the gestures to specific functions to handle the invocation of the corresponding gestures. On the line

163, the code explicitly specified that for the “panGesture” to work, the “screenEdgePanGesture” has to fail, so that pan gestures would not conflict.

6.3.2.2 Music Player Control

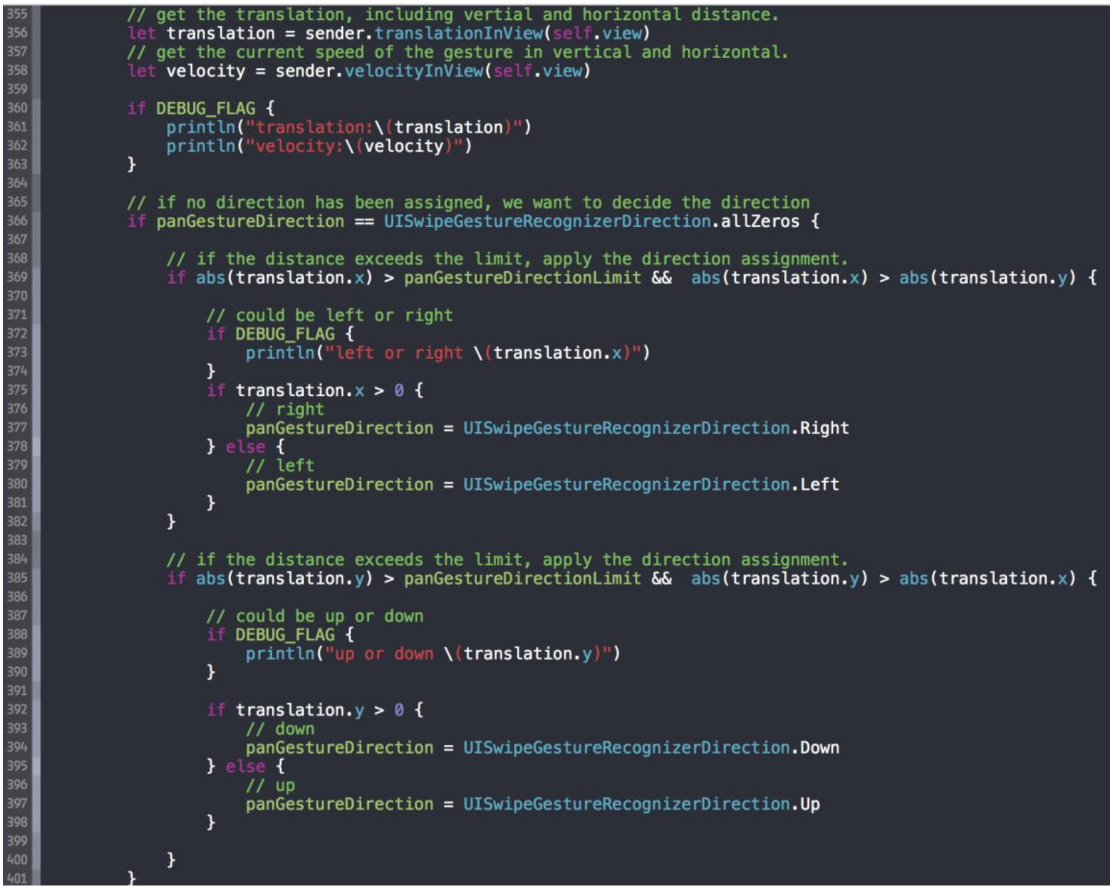
In this project, the music player control included 4 possible gestures: swipe left, swipe right, swipe up, and swipe down. The code sample would be presented in graphs and explained in detail.



```
func handlePan(sender: UIPanGestureRecognizer) {

    // first, make sure the starting point is not around the edge!
    if sender.state == UIGestureRecognizerState.Began {
        // get the location of the gesture
        var location = sender.locationInView(self.view)
        if DEBUG_FLAG {
            println("pan gesture began...\(location)")
        }
        // if the gesture begins close to the edge,
        // then the gesture shall be ingored,
        // because it shall be handled by pan edge gesture.
        // otherwise, the gesture would be handled in this function.
        if location.x > 1000 {
            edgeGestureIsPending = true
        } else {
            edgeGestureIsPending = false
        }
    }

    // if the gesture is starting from a point very close to the edge
    // terminate this function.
    if edgeGestureIsPending {
        return
    }
```

Graph6.3.2.2-1: pan gesture recognizer – part 1

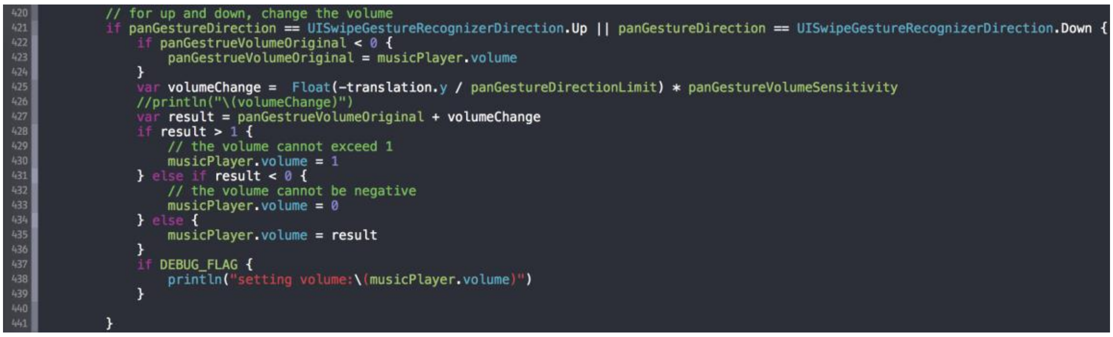
The code sample presented in graph6.3.2.2-1 is the first part of the function. The purpose of the sample code is to determine if the function “handlePan” is conflicting with the “UIScreenEdgePanGestureRecognizer”. After several tests, it was clear that the beginning position was the key to avoid this confliction. Therefore, in the code, the starting position was tested if it was close to the edge. If the position was close to the right edge, then the flag “edgeGestureIsPending” was set to be true and this function would terminate immediately.



```
        // get the translation, including vertial and horizontal distance.
        let translation = sender.translationInView(self.view)
        // get the current speed of the gesture in vertical and horizontal.
        let velocity = sender.velocityInView(self.view)

        if DEBUG_FLAG {
            println("translation:\(translation)")
            println("velocity:\(velocity)")
        }

        // if no direction has been assigned, we want to decide the direction
        if panGestureDirection == UISwipeGestureRecognizerDirection.allZeros {

            // if the distance exceeds the limit, apply the direction assignment.
            if abs(translation.x) > panGestureDirectionLimit &&  abs(translation.x) > abs(translation.y) {

                // could be left or right
                if DEBUG_FLAG {
                    println("left or right \(translation.x)")
                }
                if translation.x > 0 {
                    // right
                    panGestureDirection = UISwipeGestureRecognizerDirection.Right
                } else {
                    // left
                    panGestureDirection = UISwipeGestureRecognizerDirection.Left
                }
            }

            // if the distance exceeds the limit, apply the direction assignment.
            if abs(translation.y) > panGestureDirectionLimit &&  abs(translation.y) > abs(translation.x) {

                // could be up or down
                if DEBUG_FLAG {
                    println("up or down \(translation.y)")
                }

                if translation.y > 0 {
                    // down
                    panGestureDirection = UISwipeGestureRecognizerDirection.Down
                } else {
                    // up
                    panGestureDirection = UISwipeGestureRecognizerDirection.Up
                }

            }
        }
```

Graph6.3.2.2-2: pan gesture recognizer – part 2

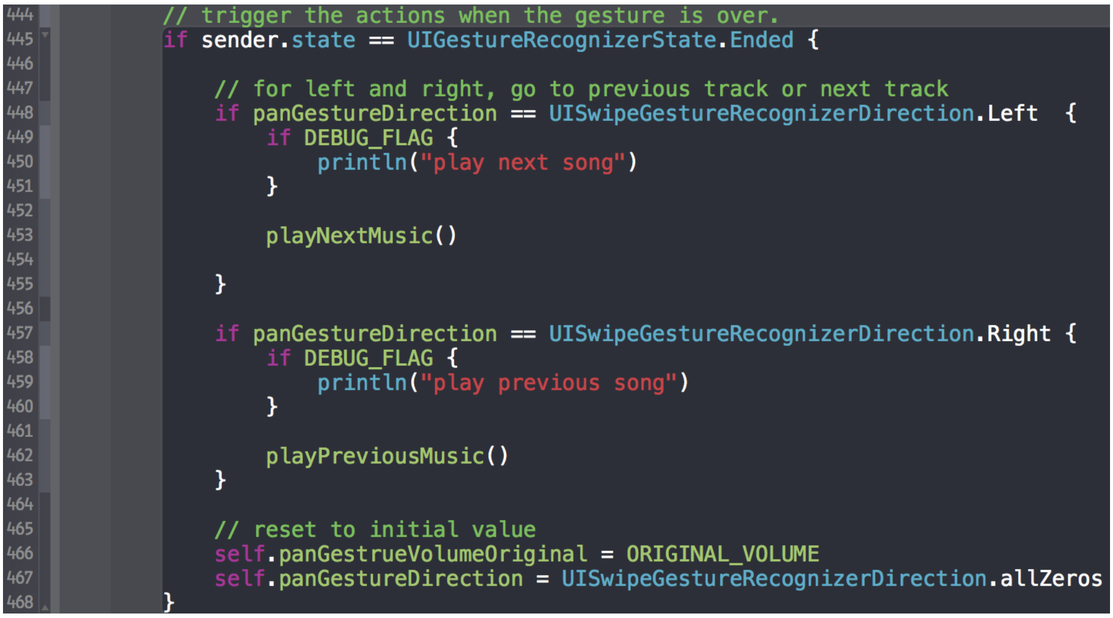
The second part of the sample code mainly focused on the direction of the gesture. The initial state of the gesture shall be “allZeros”, meaning no direction was assigned before this. If the current gesture exceeded the “panGestureDirectionLimit”, the direction assignment would be invoked. Depending on the “translation”, the direction would be assigned accordingly.



```
        // for up and down, change the volume
        if panGestureDirection == UISwipeGestureRecognizerDirection.Up || panGestureDirection == UISwipeGestureRecognizerDirection.Down {
            if panGestrueVolumeOriginal < 0 {
                panGestrueVolumeOriginal = musicPlayer.volume
            }
            var volumeChange =  Float(-translation.y / panGestureDirectionLimit) * panGestureVolumeSensitivity
            //println("\(volumeChange)")
            var result = panGestrueVolumeOriginal + volumeChange
            if result > 1 {
                // the volume cannot exceed 1
                musicPlayer.volume = 1
            } else if result < 0 {
                // the volume cannot be negative
                musicPlayer.volume = 0
            } else {
                musicPlayer.volume = result
            }
            if DEBUG_FLAG {
                println("setting volume:\(musicPlayer.volume)")
            }

        }
```

Graph6.3.2.2-3: pan gesture recognizer – part 3

The third part of the sample code focused on the volume control. If the direction of the gesture has been assigned to be up or down, the volume would be changed accordingly.



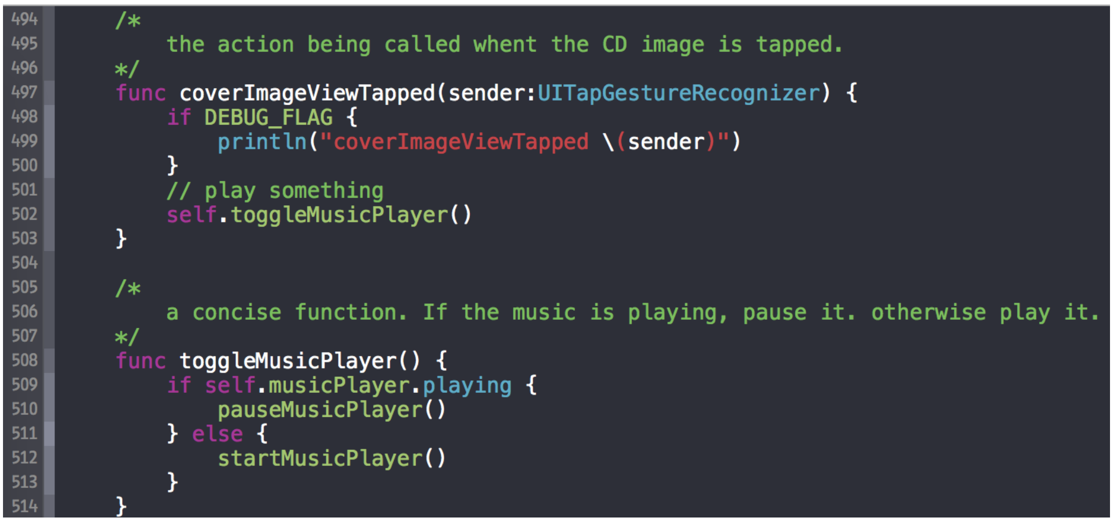
```
// trigger the actions when the gesture is over.
if sender.state == UIGestureRecognizerState.Ended {

    // for left and right, go to previous track or next track
    if panGestureDirection == UISwipeGestureRecognizerDirection.Left  {
        if DEBUG_FLAG {
            println("play next song")
        }

        playNextMusic()

    }

    if panGestureDirection == UISwipeGestureRecognizerDirection.Right {
        if DEBUG_FLAG {
            println("play previous song")
        }

        playPreviousMusic()
    }

    // reset to initial value
    self.panGestrueVolumeOriginal = ORIGINAL_VOLUME
    self.panGestureDirection = UISwipeGestureRecognizerDirection.allZeros
}
```

Graph6.3.2.2-4: pan gesture recognizer – part 4

Sample code presented in graph6.3.2.2-4 mainly focused on the command on playing next or previous song. The command was triggered when the gesture was ended. Depending on the direction of the gesture, corresponding command would be applied. At the end of the code, some related variables were set to initial values.

6.3.3 CD view and animation

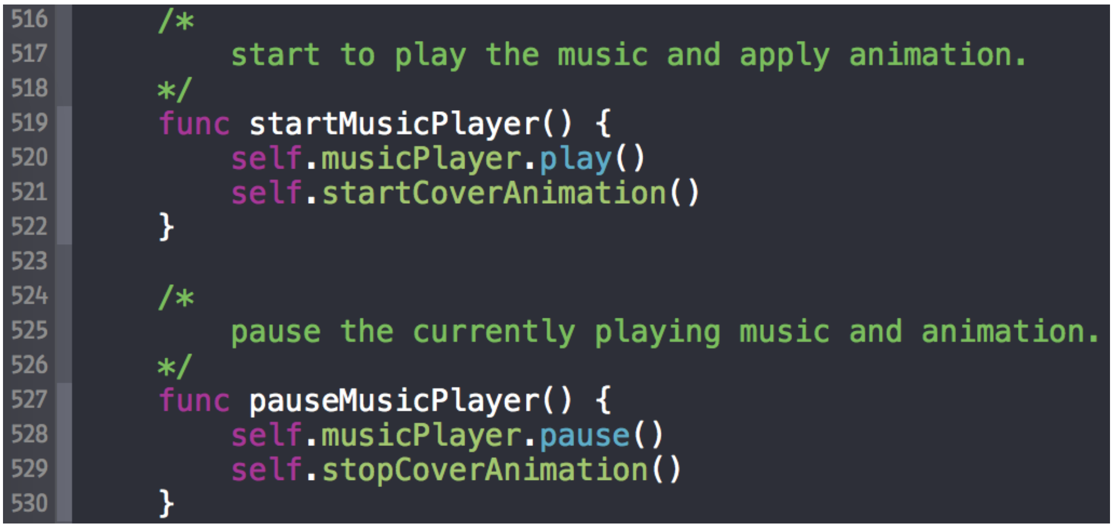
This section would introduce the methods related to the cd rotating animation.



```
    /*
        the action being called whent the CD image is tapped.
    */
    func coverImageViewTapped(sender:UITapGestureRecognizer) {
        if DEBUG_FLAG {
            println("coverImageViewTapped \(sender)")
        }
        // play something
        self.toggleMusicPlayer()
    }

    /*
        a concise function. If the music is playing, pause it. otherwise play it.
    */
    func toggleMusicPlayer() {
        if self.musicPlayer.playing {
            pauseMusicPlayer()
        } else {
            startMusicPlayer()
        }
    }
```

Graph6.3.2.2-5: CD Animation – part 5

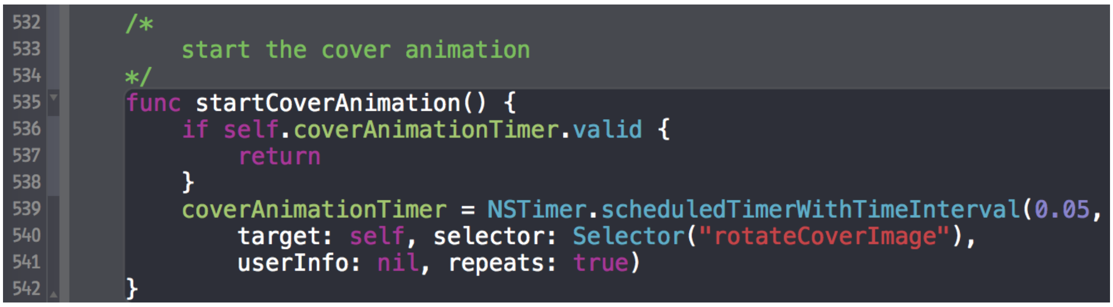
As presented in the graph6.3.2.2-4, the code would be executed when the user tapped on the CD in the interface. The function would simply call the “toggleMusicPlayer, which would automatically decide to play or pause the music depending on the current state.



```
    /*
        start to play the music and apply animation.
    */
    func startMusicPlayer() {
        self.musicPlayer.play()
        self.startCoverAnimation()
    }

    /*
        pause the currently playing music and animation.
    */
    func pauseMusicPlayer() {
        self.musicPlayer.pause()
        self.stopCoverAnimation()
    }
```

Graph6.3.2.2-6: CD Animation – part 6

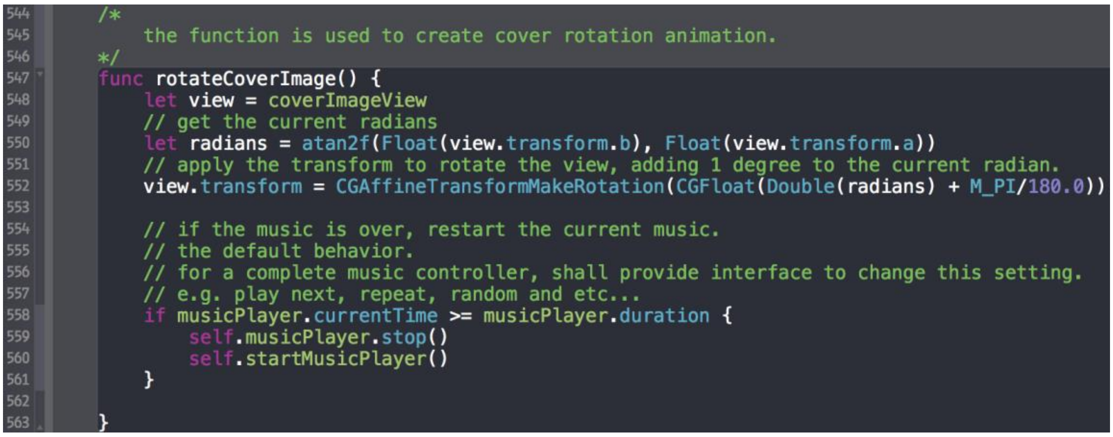
The function “startMusicPlayer” and “pauseMusicPlayer” were presented in the graph above. The structure was similar: start or pause the audio player, as well as the CD cover animation.



```swift
/*
    start the cover animation
*/
func startCoverAnimation() {
    if self.coverAnimationTimer.valid {
        return
    }
    coverAnimationTimer = NSTimer.scheduledTimerWithTimeInterval(0.05,
        target: self, selector: Selector("rotateCoverImage"),
        userInfo: nil, repeats: true)
}
```

Graph6.3.2.2-7: CD Animation – part 7

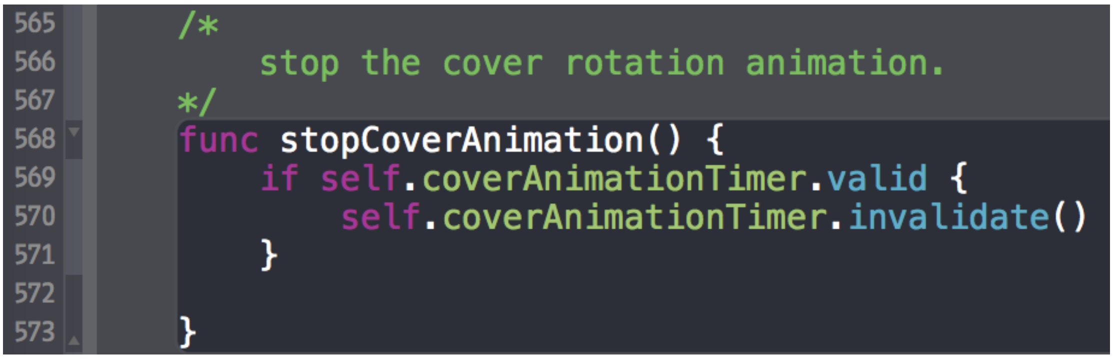
When the “startCoverAnimation” was called, a separate scheduled timer would be created to run function “rotateCoverImage” 20 times per second repeatedly.



```swift
/*
    the function is used to create cover rotation animation.
*/
func rotateCoverImage() {
    let view = coverImageView
    // get the current radians
    let radians = atan2f(Float(view.transform.b), Float(view.transform.a))
    // apply the transform to rotate the view, adding 1 degree to the current radian.
    view.transform = CGAffineTransformMakeRotation(CGFloat(Double(radians) + M_PI/180.0))

    // if the music is over, restart the current music.
    // the default behavior.
    // for a complete music controller, shall provide interface to change this setting.
    // e.g. play next, repeat, random and etc...
    if musicPlayer.currentTime >= musicPlayer.duration {
         self.musicPlayer.stop()
         self.startMusicPlayer()
    }

}
```

Graph6.3.2.2-8: CD Animation – part 8

The function “rotateCoverImage” simply calculated the current rotation angle, and then incremented the rotation a little bit every it got executed.



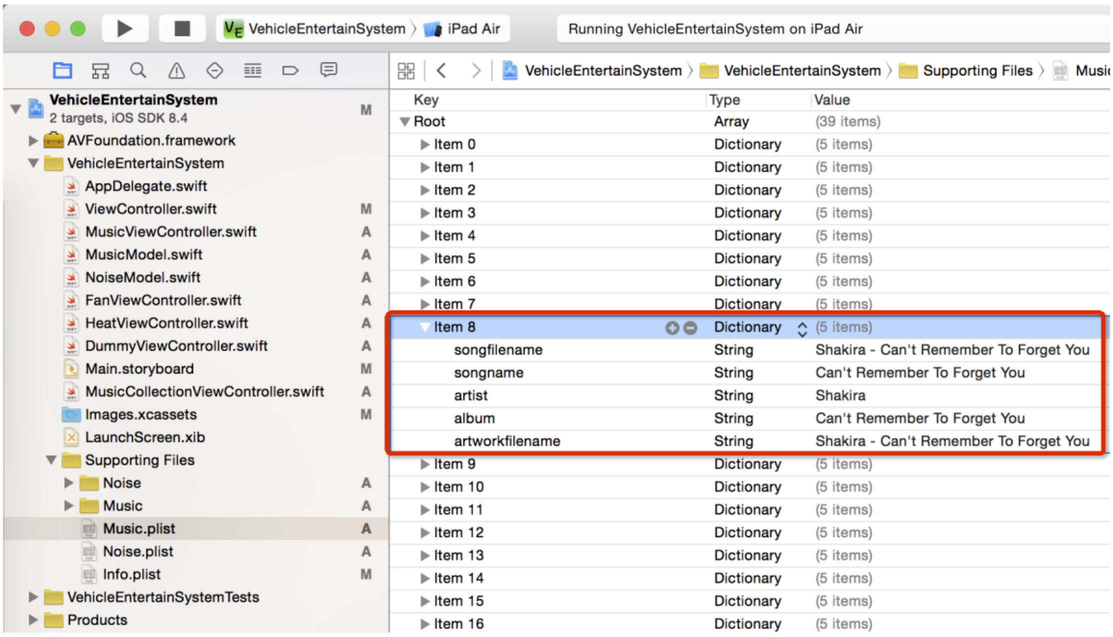
```swift
/*
    stop the cover rotation animation.
*/
func stopCoverAnimation() {
    if self.coverAnimationTimer.valid {
        self.coverAnimationTimer.invalidate()
    }

}
```

Graph6.3.2.2-9: CD Animation – part 9

The function “stopCoverAnimation” was relatively easy to implement. The code simply called “invalidate” to terminate the timer. Thus the animation would stop immediately.

6.3.4 Property list

The information of the resource files was stored in the property list: “Music.plist” and “Noise.plist”.

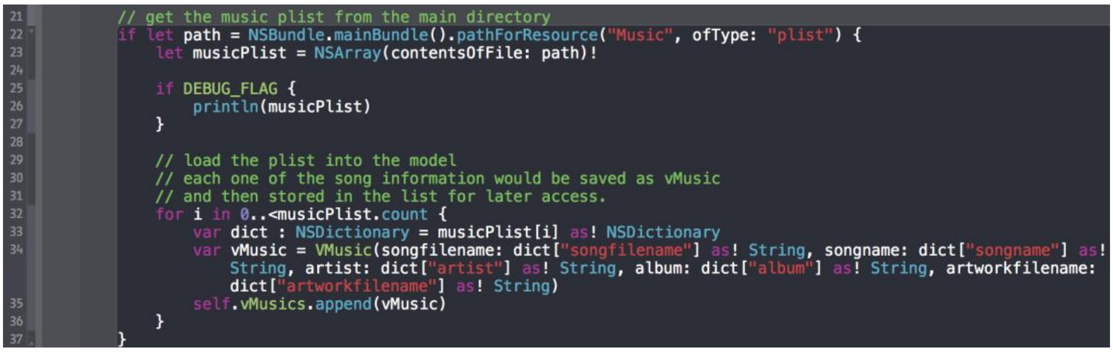

VehicleEntertainSystem 〉 iPad Air
Running VehicleEntertainSystem on iPad Air
VehicleEntertainSystem 〉 VehicleEntertainSystem 〉 Supporting Files 〉 Musi

VehicleEntertainSystem M
2 targets, iOS SDK 8.4
AVFoundation.framework
VehicleEntertainSystem
AppDelegate.swift
ViewController.swift M
MusicViewController.swift A
MusicModel.swift A
NoiseModel.swift A
FanViewController.swift A
HeatViewController.swift A
DummyViewController.swift A
Main.storyboard M
MusicCollectionViewController.swift A
Images.xcassets M
LaunchScreen.xib
Supporting Files
Noise A
Music A
Music.plist A
Noise.plist A
Info.plist M
VehicleEntertainSystemTests
Products

Key	Type	Value
Root	Array	(39 items)
Item 0	Dictionary	(5 items)
Item 1	Dictionary	(5 items)
Item 2	Dictionary	(5 items)
Item 3	Dictionary	(5 items)
Item 4	Dictionary	(5 items)
Item 5	Dictionary	(5 items)
Item 6	Dictionary	(5 items)
Item 7	Dictionary	(5 items)
Item 8	Dictionary	(5 items)
songfilename	String	Shakira - Can't Remember To Forget You
songname	String	Can't Remember To Forget You
artist	String	Shakira
album	String	Can't Remember To Forget You
artworkfilename	String	Shakira - Can't Remember To Forget You
Item 9	Dictionary	(5 items)
Item 10	Dictionary	(5 items)
Item 11	Dictionary	(5 items)
Item 12	Dictionary	(5 items)
Item 13	Dictionary	(5 items)
Item 14	Dictionary	(5 items)
Item 15	Dictionary	(5 items)
Item 16	Dictionary	(5 items)

Graph6.3.4-1: Property List

The graph above presented a section of the property in “Music.plist”. The property list is basically a highly formatted xml file. This property list contains an array of dictionaries. Each dictionary contains related information of the music, including the source file name, artist, album, artwork filename, and the name of the music.



```
// get the music plist from the main directory
if let path = NSBundle.mainBundle().pathForResource("Music", ofType: "plist") {
    let musicPlist = NSArray(contentsOfFile: path)!

    if DEBUG_FLAG {
        println(musicPlist)
    }

    // load the plist into the model
    // each one of the song information would be saved as vMusic
    // and then stored in the list for later access.
    for i in 0..<musicPlist.count {
        var dict : NSDictionary = musicPlist[i] as! NSDictionary
        var vMusic = VMusic(songfilename: dict["songfilename"] as! String, songname: dict["songname"] as!
            String, artist: dict["artist"] as! String, album: dict["album"] as! String, artworkfilename:
            dict["artworkfilename"] as! String)
        self.vMusics.append(vMusic)
    }
}
```

Graph6.3.4-2: Load Property List

The code above showed how the program loaded the "Music.plist" into the application. Firstly, an array was created to load the file; then, for each element, loaded the content from the dictionary given the correct keys; finally, create "VMusic" to stored the content and stored the "VMusic" into local list for later access.

7 Evaluation

7.1 Experiment

7.1.1 Participants

Seven participants were recruited for this experiment. All participants were students in University of Nottingham. The participants sample contains 4 males and 3 females, with the average age 26.0. They were all informed of the purpose of the experiment, and assigned the consent form before the experiment began. The consent form is available in the appendix.

7.1.2 Materials

Participants were individually invited to a discussion room located in the Computer Science Atrium in the third floor. Two chairs and one table were prepared in the room. The instruction form, which contains the basic usage of the application, were presented on the table. An iPad Air that installed the application was positioned at the left side of the table. A MacBook Pro was placed at the middle of the table with the front camera on. A video named “Need For Speed 2015 Full Movie” was loaded on YouTube from the Internet. During the experiment, the video would be played in full screen and the participants were required to watch the video whilst perform specific tasks on the iPad Air. An iPhone5 was used during the experiment at a video recorder to record the whole process of the experiment, mainly focus on the interaction between the participants and the interface on the iPad Air.

7.1.3 Design

There are totally 4 hypotheses to be examined in the experiment. In this experiment, the application named “Music” that comes with an iPad was chosen as the default Interface to be compared with.

a) Hypothesis 1: The new Interface does not cause distraction for the driver compared with the default Interface.

   The participants were required to watch the video whilst perform some tasks on the iPad Air. The camera of the MacBook Pro would record the eye movement during the whole experiment. The time that the participants leave their eye on the video would be calculated based on the recorded video.

b) Hypothesis 2: The new Interface does not affect the driver in terms of choosing to play next or previous song compared with the default Interface.

   The interaction between the participants and the interface on iPad Air would be recorded on an iPhone. The time that participants performed the commands “Go to Next Song” and “Go to Previous Song” would be calculated based on the recorded video.

c) Hypothesis 3: The new Interface does not affect the driver in terms of changing the volume of the music player compared with the default Interface.

The data required is similar to H2: The time that participants performed the commands “Increase the Volume” and “Decrease the Volume” would be calculated based on the recorded video.

d) Hypothesis 4: The new Interface does not affect the driver in terms of navigating through different sections compared with the default Interface.
   The purpose of this hypothesis is to test 1, if the large Icon design in the centre screen is suitable for driver to choose; 2, if the gesture “pinch to navigate back to center screen” is better than pressing the back button on the left top corner. To verify this hypothesis, the time consumed performing navigation commands in the experiments would be calculated and examined.

Basically, all the data would come from the recorded video. The measurement would be the time taken to complete corresponding tasks or some certain observed behaviours. The statistical analysis was conducted using SPSS 22 statistical software. Dependent variables were analysed using T-Test (Within-Subjects) with two different conditions (New Interface and Default Interface). The rejection level for all analyses were set at p=0.05 to determine statistical significance.

7.1.4 Procedure

This experiment took place in a discussion room on 3rd floor of Atrium in Computer Science in order to minimise the environment distraction. Participants were individually invited to the discussion room. On arrival, the participants were notified to complete a consent form and were informed about the tasks for them to perform.

Before the experiment began, the participants would have 5 minutes to read the instruction form on the table and interact with the interface to be tested accordingly. When the preparation time was over, the camera on the MacBook Pro would be turned on. The video prepared in the MacBook would be played when the experiment began. The participants were told to watch the video as much as possible so that they would have their eyes on the screen of the MacBook. The participants would interact with the default Interface at first. When it was over, the participants would interact with the new Interface. During the experiment, the interaction between the participants and the iPad Air were recorded using iPhone5.

When the experiment was over, the participants were asked to ask any questions and leave the room.

7.1.5 Tasks to Perform

There shall be a 30 seconds gap between each gap. The total time required to complete the tasks is estimated to be 12 minutes.

1. Start playing the song
2. Play next song
3. Decrease the volume to 0
4. Increase the volume to maximum
5. Play previous song
6. Pause the song
7. Navigate to Center Screen (for default interface, navigate to song-list interface)
8. Navigate to Music Player (for default interface, navigate to music collection interface)
9. Repeat from 1-8 for 3 times in total.

The data would be collected by examining the recorded video. The data to be evaluated would be the total time to perform each tasks. The results would present more details.

7.2 Results

7.2.1 Descriptive statistics

The data was measured by the time that the eyes of the participants off the screen in order to complete the tasks. The unit is in second.

The result of the descriptive statistics for the dependent variable is listed in the table below. The table 7.2.1-1 presented the data for completing each task with the default interface. The table 7.2.1-2 presented the data for completing each task with the new interface.

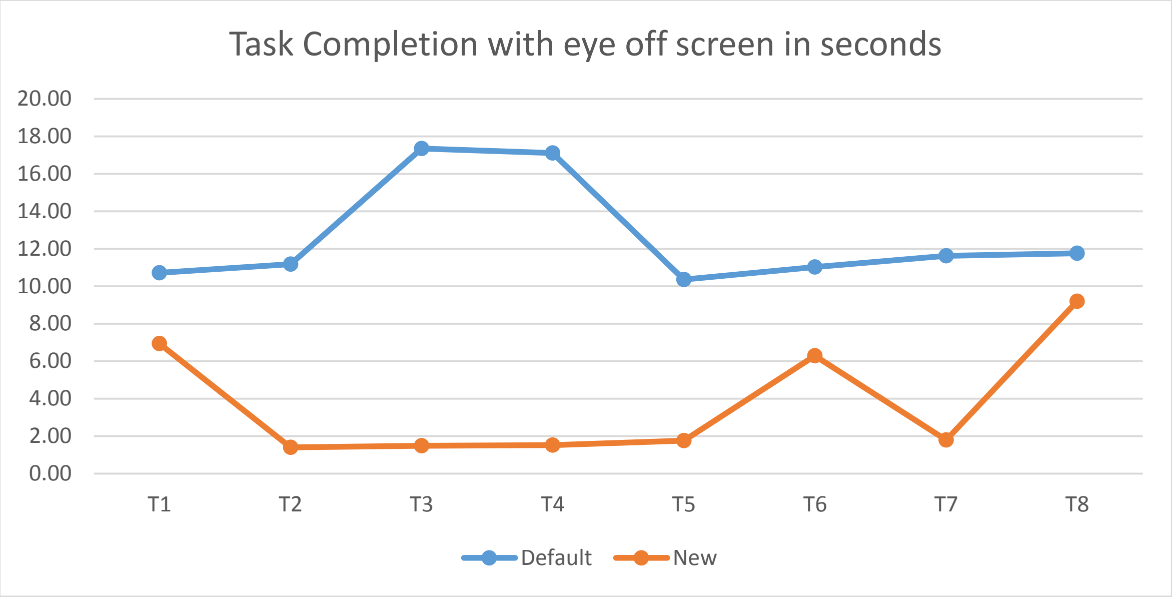
Task	Mean	Std. Deviation	Std. Error
T1	10.71	2.51149	0.35878
T2	11.17	2.00898	0.28700
T3	17.35	2.18778	0.31254
T4	17.10	2.88089	0.41156
T5	10.36	2.04348	0.29193
T6	11.03	2.23020	0.31860
T7	11.61	1.93083	0.27583
T8	11.76	2.81712	0.40245

Table 7.2.1-1: Default Interface

Task	Mean	Std. Deviation	Std. Error
T1	6.93	1.73370	0.24767
T2	1.40	0.79162	0.11309
T3	1.49	1.17534	0.16791
T4	1.51	0.78619	0.11231
T5	1.76	1.61334	0.23048
T6	6.29	2.89507	0.41358
T7	1.79	0.55506	0.07929
T8	9.19	1.83342	0.26192

Table 7.2.1-2: New Interface

Below is the graph that compares the mean time that the user left eyes from the screen of MacBook to iPad to complete tasks. From the descriptive statistics, it is obvious that the average time taken for completing tasks with new interface is less than with the default interface. It is noticeable that the task 2, 3, 4, 5, 7 took less time than the ones in the default interface. This difference may reflect that the new design of the interface improved the user interaction. Next section provides more details for understanding the statistics.



Graph 7.2.1-1: Interface Comparison in Mean time

7.2.2 Inferential Statistics

For each Hypothesis, a T-Test within subjects was applied.

7.2.2.1 Hypothesis 1

H1: The new Interface does not cause distraction for the driver compared with the default Interface.

Subject No.	Condition A (Default Interface)	Condition B (New Interface)
S1	88.2	21.6
S2	92.6	34.8
S3	102.0	35.4
S4	102.9	25.9
S5	121.3	26.3
S6	100.3	38.8
S7	100.3	29.6
Total	707.6	212.4
Average	101.1	30.3

Degree of freedom: df = 6

T_{obs}= 6.3669

t_{crit} (df = 6; $p<0.05$; two-tailed) = 2.447

As $t_{obs}>t_{crit}$ we reject the null hypothesis. Examination of the data shows that there was significant difference in observed distraction time between default interface (M = 101.1 seconds) and new interface (M = 30.3 seconds), t(6) = 6. 3669.

As the mean value of default interface (101.1) is greater than the new interface (30.3), it was concluded that the new interface cause less distraction than the default interface.

7.2.2.2 Hypothesis 2

H2: The new Interface does not affect the driver in terms of choosing to play next or previous song compared with the default Interface.

The related tasks are Task 2 and 5.

Subject No.	Condition A (Default Interface)	Condition B (New Interface)
S1	20.5	0.0
S2	17.3	3.6
S3	20.8	6.0
S4	21.1	4.5
S5	25.2	2.9
S6	20.8	3.3
S7	25.1	1.8
Total	150.7	22.1
Average	21.5	3.2

Degree of freedom: df = 6

T_{obs}= 6.1858

t_{crit} (df = 6; p<0.05; two-tailed) = 2.447

As $t_{obs}>t_{crit}$ we reject the null hypothesis. Examination of the data shows that there was significant difference in observed distraction time between default interface (M = 21.5 seconds) and new interface (M = 3.2 seconds), t(6) = 6.1857.

As the mean value of default interface (21.5) is greater than the new interface (3.2), it was concluded that the new interface causes less distraction than the default interface when switching to previous or next song.

7.2.2.3 Hypothesis 3

H3: The new Interface does not affect the driver in terms of changing the volume of the music player compared with the default Interface.

The related tasks are task 3 and 4.

Subject No.	Condition A (Default Interface)	Condition B (New Interface)
S1	29.3	0.0

S2	32.4	3.3
S3	38.5	5.5
S4	39.9	4.0
S5	37.8	3.3
S6	28.7	3.4
S7	34.6	1.5
Total	241.1	21.0
Average	34.4	3.0

Degree of freedom: df = 6

T_{obs}= 6.6881

t_{crit} (df = 6; p<0.05; two-tailed) = 2.447

As t_{obs}>t_{crit} we reject the null hypothesis. Examination of the data shows that there was significant difference in observed distraction time between default interface (M = 34.4 seconds) and new interface (M = 3.0 seconds), t(6) = 6.6881.

As the mean value of default interface (34.4) is greater than the new interface (3.0), it was concluded that the new interface causes less distraction than the default interface when changing volumes of the music player.

7.2.2.4 Hypothesis 4

H4: The new Interface does not affect the driver in terms of navigating through different sections compared with the default Interface.

Subject No.	Condition A (Default Interface)	Condition B (New Interface)
S1	19.2	14.5
S2	23.5	18.2
S3	23.3	12.8
S4	19.7	12.3
S5	33.0	16.9
S6	20.7	20.9
S7	17.9	17.2
Total	157.3	112.8
Average	22.5	16.1

Degree of freedom: df = 6

T_{obs}= 2.7521

t_{crit} (df = 6; p<0.05; two-tailed) = 2.447

As $t_{obs}>t_{crit}$ we reject the null hypothesis. Examination of the data shows that there was significant difference in observed distraction time between default interface (M = 22.5 seconds) and new interface (M = 16.1 seconds), t(6) = 2.7521.

As the mean value of default interface (22.5) is greater than the new interface (16.1), it was concluded that the new interface causes less distraction than the default interface when navigating through different sections of the application.

8 Discussion

In the inferential statistics, all 4 hypotheses were rejected, meaning that improvement in the new interface was significant, compared with the default interface.

Hypothesis 1 concerns the overall performance of the new interface compared with default interface. The statistical analysis showed the improvement in the new interface was significant.

Hypothesis 2 concerns the audio control, including playing next song and playing previous song. The result showed the significant improvement. In the default interface, there were two buttons for participants to tap to perform the corresponding operations. But in the new interface, participants could perform the same operations by swiping left or right, starting from anywhere (except the edges) inside the touch screen. These gestures significantly alleviated the visual demand from the participants and thus caused less distraction.

Hypothesis 3 also concerns the audio control, including increasing volume and decreasing volume of the music player. In the default interface, participants would require 3 steps to perform the volume control operation: 1, find the volume control slider; 2, find the current volume control icon (usually a circle in a slider); 3, change the volume by dragging the control icon to left or right. However, in the new interface, facilitated by the swiping gestures, participants only need to swipe up or down to increase or decrease the volume. This simplified procedure resulted in the improved performance that caused less visual distraction.

Hypothesis 4 concerns about the navigation between different functional sections in the application. Although it was clear that the statistically significant difference was found between the default interface and new interface in the experiment, the average distraction time for performing such operations was estimated to be about 2 seconds, which could possibly lead to dangerous repercussions in driving situations.

Finding: Large Icon

From the descriptive statistics it was obvious that the distraction time for completing task 1, 6, and 8 was much longer than the rest of the tasks in the new interface. The results showed that the performance were very close to the default interface. All the tasks mentioned in 1, 6 and 8 shared one similarity: participants needed to see the interface in order to perform certain interaction with the interface. E.g. touch a button or large icon.

Subject No.	Condition A (Default Interface)	Condition B (New Interface)
S1	28.2	19.4
S2	31.9	25.4
S3	33.5	22.8
S4	30.7	15.9
S5	45.1	19.0
S6	35.9	30.1
S7	29.1	24.2
Total	234.4	156.8
Average	33.5	22.4

Degree of freedom: df = 6

T_{obs}= 3.4641

t_{crit} (df = 6; p<0.05; two-tailed) = 2.447

As t_{obs}>t_{crit} we reject the null hypothesis. Examination of the data shows that there was significant difference in observed distraction time between default interface (M = 33.5 seconds) and new interface (M = 22.4 seconds), t(6) = 3.4641.

From the results, we know that there was an improvement compared with the default interface. However, this improvement was insufficient considering that the tasks that requires visual help would on average trigger a distraction that lasts from 2 to 3 seconds. In this case, even though the navigation icons were designed to take almost ¼ of the screen size, it still seemed not to be effective enough to avoid visual distraction.

9 Conclusion

In this project, a prototype has been designed and an application has been developed for iPad. Experiments have been conducted to evaluate the effectiveness of applied gestures to the new interface. The result suggested that the new interface, compared with default interface which was not designed specifically for the in-vehicle information system, required less visual demand. Further work should incorporate more gestures and more functionalities and the experiment shall be conducted in driving simulation.

10 Reference

Burnett, G., Crundall, E., Large, D., Lawson, G., & Skrypchuk, L. (2015). A Study of Unidirectional Swipe Gestures on In-Vehicle Touch Screens.

Kim, H., Kwon, S., Heo, J., Lee, H., & Chung, M. (2014). The effect of touch-key size on the usability of In-Vehicle Information Systems and driving safety during simulated driving. *Applied Ergonomics*, *45*(3), 379-388.

Kim, H., & Song, H. (2014). Evaluation of the safety and usability of touch gestures in operating in-vehicle information systems with visual occlusion. *Applied Ergonomics*, *45*(3), 789-798.

Large, D., Crundall, E., Burnett, G., & Skrypchuk, L. (2015). Predicting the Visual Demand of Finger-Touch Pointing Tasks in a Driving Context.

Salmon, P., Lenné, M., Triggs, T., Goode, N., Cornelissen, M., & Demczuk, V. (2011). The effects of motion on in-vehicle touch screen system operation: A battle management system case study. *Transportation Research Part F: Traffic Psychology And Behaviour*, *14*(6), 494-503.

11 Appendix

11.1 Coding

11.1.1 ViewController.swift

```
//
//  ViewController.swift
//  VehicleEntertainSystem
//
//  Created by Zongqi Wang on 8/31/15.
//  Copyright (c) 2015 Zongqi Wang. All rights reserved.
//

import UIKit
import AVFoundation
/*
    an extension for the images get shaped.
    the source code is available from the github.
*/
extension UIImage {
    var rounded: UIImage {
        let imageView = UIImageView(image: self)
        imageView.layer.cornerRadius = size.height < size.width ? size.height/2 : size.width/2
        imageView.layer.masksToBounds = true
        UIGraphicsBeginImageContext(imageView.bounds.size)
        imageView.layer.renderInContext(UIGraphicsGetCurrentContext())
        let result = UIGraphicsGetImageFromCurrentImageContext()
        UIGraphicsEndImageContext()
        return result
    }
    var circle: UIImage {
        let square = size.width < size.height ? CGSize(width: size.width, height: size.width) : CGSize(width: size.height, height: size.height)
        let imageView = UIImageView(frame: CGRect(origin: CGPoint(x: 0, y: 0), size: square))
        imageView.contentMode = UIViewContentMode.ScaleAspectFill
        imageView.image = self
        imageView.layer.cornerRadius = square.width/2
        imageView.layer.masksToBounds = true
        UIGraphicsBeginImageContext(imageView.bounds.size)
        imageView.layer.renderInContext(UIGraphicsGetCurrentContext())
        let result = UIGraphicsGetImageFromCurrentImageContext()
        UIGraphicsEndImageContext()
        return result
    }

}

class ViewController: UIViewController, NoisePlayerDelegate, MusicPlayerDelegate
{

    var noisePlayer : AVAudioPlayer!
    var musicPlayer : AVAudioPlayer!
    var musicFilename = "Alan Walker - Fade"

    @IBOutlet weak var centerScrollview: UIScrollView!
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.

        // there are three pages of icons to be displayed
        // user could swipe to explore.
        // the content and attributes of the UI elements were set in the storyboard.
        centerScrollview.contentSize.width = 1024 * 3

        let noiseUrl = NSURL(fileURLWithPath:
NSBundle.mainBundle().pathForResource("Fan 01", ofType: "mp3")!)
        noisePlayer = AVAudioPlayer(contentsOfURL: noiseUrl, error: nil)

        let musicUrl = NSURL(fileURLWithPath:
NSBundle.mainBundle().pathForResource(musicFilename, ofType: "mp3")!)
        musicPlayer = AVAudioPlayer(contentsOfURL: musicUrl, error: nil)
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

    @IBAction func iconTapped(sender: UITapGestureRecognizer) {
        performSegueWithIdentifier("dummySegue", sender: sender)
    }

    @IBAction func musicTapped(sender: UITapGestureRecognizer) {
        performSegueWithIdentifier("musicSegue", sender: sender)
    }
    @IBAction func heatTapped(sender: UITapGestureRecognizer) {
        performSegueWithIdentifier("heatSegue", sender: sender)
    }

    @IBAction func fanTapped(sender: UITapGestureRecognizer) {
        performSegueWithIdentifier("fanSegue", sender: sender)
    }

    /*
        This function is used to prepare for a segue to next view.
    */
    override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?)
{
        if let identifier = segue.identifier {
            switch identifier {
            case "dummySegue":
                    // for development purpose, the dummy interface is chosen
as default.
                    let controller = segue.destinationViewController as!
DummyViewController
                    let view = sender?.view as! UIImageView
                    controller.dummyImage = view.image
                    controller.dummyInfo = "This section is under
development."
            case "musicSegue":
                let controller = segue.destinationViewController as!
MusicViewController
                controller.musicPlayerDelegate = self
                controller.continueFromParent = true
                println("Prepare for music segue")
            case "heatSegue":
                println("Prepare for heat segue")
            case "fanSegue":
                //
                let controller = segue.destinationViewController as!
FanViewController
                controller.noisePlayerDelegate = self
                if noisePlayer.playing {
                    noisePlayer.pause()
                }
                println("Prepare for fan segue")
            default:
                break
            }
        }
    }

    func changeNoiseAudio(filename:String, type:String) {
        if noisePlayer.playing {
            noisePlayer.stop()
        }
        let noiseUrl = NSURL(fileURLWithPath:
NSBundle.mainBundle().pathForResource(filename, ofType: type)!)
        noisePlayer = AVAudioPlayer(contentsOfURL: noiseUrl, error: nil)
        noisePlayer.volume = 0.2
        noisePlayer.numberOfLoops = -1
        noisePlayer.play()
    }


    func getNoisePlayer() -> AVAudioPlayer {
        return self.noisePlayer
    }

    func changeMusicAudio(filename:String, type:String) {
        if musicPlayer.playing {
            musicPlayer.stop()
        }
        let musicUrl = NSURL(fileURLWithPath:
NSBundle.mainBundle().pathForResource(filename, ofType: type)!)
        musicPlayer = AVAudioPlayer(contentsOfURL: musicUrl, error: nil)
        self.musicPlayer.numberOfLoops = -1
        self.musicFilename = filename
    }

    func getMusicPlayer() -> AVAudioPlayer {
        return self.musicPlayer
    }

    func getMusicFilename() -> String {
        return musicFilename
    }

}
```

11.1.2 MusicViewController.swift

```
//
//  MusicViewController.swift
//  VehicleEntertainSystem
```

```
//
//    Created by Zongqi Wang on 8/31/15.
//    Copyright (c) 2015 Zongqi Wang. All rights reserved.
//

import Foundation
import UIKit
import AVFoundation

protocol MusicPlayerDelegate {
        func changeMusicAudio(filename:String, type:String)
        func getMusicPlayer() -> AVAudioPlayer
        func getMusicFilename() -> String
}

class MusicViewController : UIViewController, UIGestureRecognizerDelegate,
MusicCollectionDelegate {

        // external access to the audio player resource
        var musicPlayerDelegate :    MusicPlayerDelegate!

        let DEBUG_FLAG = false

        // the background for the interface, with blur affects applied to this background.
        @IBOutlet weak var backgroundImageView: UIImageView!

        // the text area for the interface to show related informaiton of the songs. e.g.
name.
        @IBOutlet weak var musicInfoTextView: UITextView!

        // the image holding the disc
        var discImageView    : UIImageView!

        // the image holding pecific song
        var coverImageView : UIImageView!

        // the timer to control the rotation animation of the cover image of the song.
        var coverAnimationTimer : NSTimer = NSTimer()

        // the model that holds all related information of a song.
        var musicModel : MusicModel!

        // the instance that is capable of playing audio files.
        var musicPlayer : AVAudioPlayer!
```

```
	// the instance of the music that is currently playing.
	var vMusic : VMusic!

	// the instance index for the song.
	var musicIndex = 1

	//

	/*
		The function to get the music model
	*/
	func getModel() -> MusicModel {
		return self.musicModel
	}

	/*
		if a song is selected from collection view, play the selected song.
	*/
	func setSelectedMusicFromCollection(music:VMusic) {
		if DEBUG_FLAG {
			println("change the selected music to:\(music)")
		}

		self.vMusic = music
		for i in 0..<self.musicModel.vMusics.count {
			if musicModel.vMusics[i].songfilename == vMusic.songfilename {
				self.musicIndex = i
			}
		}

		initMusicPlayer()
		resetUIViews()
		startMusicPlayer()

	}

	/*
		This function enables the system to recognize multiple gestures at the same
time.
	*/
	func gestureRecognizer(UIGestureRecognizer,
shouldRecognizeSimultaneouslyWithGestureRecognizer:UIGestureRecognizer) ->
Bool {
            return true
    }

    var continueFromParent = false

    /*
        The set of instructions called to create the initial view of this interface.
    */
    override func viewDidLoad() {
        super.viewDidLoad()
        if DEBUG_FLAG {
            println(backgroundImageView.bounds)
        }

        var currentTime = self.musicPlayerDelegate.getMusicPlayer().currentTime
        var volume = self.musicPlayerDelegate.getMusicPlayer().volume

        // initialize the model for the song lists.
        initModel()

        // initialize the music player to play the audio files.
        initMusicPlayer()

        // initialize the UI elements for the interface.
        initUI()

        // initialize the gestures that could be recgonized the interface.
        initGestures()

        if DEBUG_FLAG {
            println("viewDidLoad")
        }

        if continueFromParent {
            for i in 0..<self.musicModel.vMusics.count {
                if musicModel.vMusics[i].songfilename ==
self.musicPlayerDelegate.getMusicFilename() {
                    self.musicIndex = i
                    self.vMusic = musicModel.vMusics[i]
                }
            }
```

```
                resetUIViews()
                initMusicPlayer()
                self.musicPlayer.volume = volume
                //self.musicPlayer.currentTime = currentTime
                startMusicPlayer()
                continueFromParent = false
            }

        }
        /*
            initialize the gestures that could be recognized in this interface.
        */
        func initGestures() {
            // init the gestures associated with the specified views.
            //
            // 1, tap to play/pause the music.
            var tapGesture = UITapGestureRecognizer(target: self, action:
    Selector("coverImageViewTapped:"))
            coverImageView.addGestureRecognizer(tapGesture)
            coverImageView.userInteractionEnabled = true

            // pinch to go back
            var pinchGesture = UIPinchGestureRecognizer(target: self, action:
    "handlePinch:")
            self.view.addGestureRecognizer(pinchGesture)

            // pan gesture to adjust volume
            var panGesture = UIPanGestureRecognizer(target: self, action:
    "handlePan:")
            self.view.addGestureRecognizer(panGesture)

            // screen edge pan gesture to navigate the song list, may require new views
            var screenEdgePanGesture = UIScreenEdgePanGestureRecognizer(target:
    self, action: "handleScreenEdgePanGestureRight:")
            screenEdgePanGesture.edges = UIRectEdge.Right
            self.view.addGestureRecognizer(screenEdgePanGesture)

            // to restrict the pan gesture when screen edge pan gesture is recognized
    first.
            panGesture.requireGestureRecognizerToFail(screenEdgePanGesture)
        }

        /*
            Initialize the UI elements of the interface.
        */
        func initUI() {
            initCDViews()
        }

        /*
            initialize the model for the songs.
        */
        func initModel() {

            if DEBUG_FLAG {
                println("init MusicModel")
            }

            musicModel = MusicModel()
            vMusic = self.musicModel.vMusics[self.musicIndex]
        }

        /*
            initialize the audio player. Assuming all audio files are of type mp3.
        */
        func initMusicPlayer() {
            if !continueFromParent {
                self.musicPlayerDelegate.changeMusicAudio(vMusic.songfilename,
    type: vMusic.songfiletype)
            }

            musicPlayer = self.musicPlayerDelegate.getMusicPlayer()
            // keep the original volume of the songs.
            var volume : Float = 0.0
            if musicPlayer != nil {
                volume = musicPlayer.volume
            } else {
                volume = 1
            }

            musicPlayer.volume = volume
        }

        /*
            update all the UI elements according to the currently playing song.
        */
        func resetUIViews() {
            var imageName = currentMusic().songfilename
            var img = UIImage(named: imageName)!
            coverImageView.image = img.circle
            coverImageView.transform = CGAffineTransformMakeRotation(0)
            if DEBUG_FLAG {
                coverImageView.backgroundColor = UIColor.redColor()
            }

            // init musicInfoTextView
            // display the simple format of a song's related information.
            // Because this is not the core study of this project.
            musicInfoTextView.text = "Name: \(vMusic.songname)\nArtist:
    \(vMusic.artist)\nAlbum: \(vMusic.album)"

            // init the background
            self.backgroundImageView.image? = UIImage(named: imageName)!
        }

        /*
            Create the CD views.
        */
        func initCDViews() {
            // load image from file named "disc"
            discImageView = UIImageView(image: UIImage(named: "disc"))

            // set size and initial location
            discImageView.bounds = CGRect(x: 0, y: 0, width: 550, height: 550)

            // put it in the center of the view
            discImageView.center = self.view.center


            // load image for the current song
            var imageName = currentMusic().songfilename
            var img = UIImage(named: imageName)!

            // create image view for the image and initialize size and location
            coverImageView = UIImageView()
            coverImageView.bounds = CGRect(x: 0, y: 0, width: 450, height: 450)

            // change the image to a circle
            coverImageView.image = img.circle
            coverImageView.contentMode = UIViewContentMode.ScaleAspectFill
            // relocate the image to the center of the view
```

```
		coverImageView.center = self.view.center
		if DEBUG_FLAG {
			coverImageView.backgroundColor = UIColor.redColor()
		}

		self.view.addSubview(discImageView)
		self.view.addSubview(coverImageView)

		// init musicInfoTextView
		musicInfoTextView.text =
"Name:\t\(vMusic.songname)\nArtist:\t\(vMusic.artist)\nAlbum:\t\(vMusic.album)"

		// init the background
		// if the background is found the blur effect would be automatically applied.
		self.backgroundImageView.image? = UIImage(named: imageName)!
	}

	/*
		If the user performed an swipe from the right edge, go to the music
collectino interface.
	*/
	func handleScreenEdgePanGestureRight(sender :
UIScreenEdgePanGestureRecognizer) {

		if sender.state == UIGestureRecognizerState.Ended {
			//println("handleScreenEdgePanGestureRight present to the song list")
			performSegueWithIdentifier("MusicCollectionSegue", sender: sender)
		}
	}

	/*
		prepare for going to the music collection interface.
	*/
	override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?)
{
		if let identifier = segue.identifier {
			switch identifier {
			case "MusicCollectionSegue":
				let controller = segue.destinationViewController as!
MusicCollectionViewController
				controller.delegate = self
				if DEBUG_FLAG {
					println("moving to MusicCollection")
				}
```

```
                default:
                    break
                }
            }
        }

        /*
            If user performed pinch gesutre, with certain sensitivity, user would go back
to previous place.
        */
        func handlePinch(sender: UIPinchGestureRecognizer) {
            if sender.scale < 0.7 {
                self.navigationController?.popViewControllerAnimated(true)
            }
        }

        /*
            a set of variables handling the pan gesture.
        */
        // set the initial gesture direction
        var panGestureDirection = UISwipeGestureRecognizerDirection.allZeros

        // the limit in pixels that trigger the pan gesture recognition
        let panGestureDirectionLimit : CGFloat = 20

        // the sensitivity for each pixel to change the volume
        let panGestureVolumeSensitivity : Float = 0.05

        // to record the initial volume
        var panGestrueVolumeOriginal : Float = -100.0

        // a flag to record if the pan edge gesture is being performed
        var edgeGestureIsPending : Bool = false

        func handlePan(sender: UIPanGestureRecognizer) {

            // first, make sure the starting point is not around the edge!
            if sender.state == UIGestureRecognizerState.Began {
                // get the location of the gesture
                var location = sender.locationInView(self.view)
                if DEBUG_FLAG {
                    println("pan gesture began...\(location)")
                }
                // if the gesture begins close to the edge,
                // then the gesture shall be ingored,
                // because it shall be handled by pan edge gesture.
                // otherwise, the gesture would be handled in this function.
                if location.x > 1000 {
                    edgeGestureIsPending = true
                } else {
                    edgeGestureIsPending = false
                }
            }

            // if the gesture is starting from a point very close to the edge
            // terminate this function.
            if edgeGestureIsPending {
                return
            }

            /*
                the rest of the function deals with the pan gesture as desired.
            */
            // get the translation, including vertial and horizontal distance.
            let translation = sender.translationInView(self.view)
            // get the current speed of the gesture in vertical and horizontal.
            let velocity = sender.velocityInView(self.view)

            if DEBUG_FLAG {
                println("translation:\(translation)")
                println("velocity:\(velocity)")
            }

            // if no direction has been assigned, we want to decide the direction
            if panGestureDirection == UISwipeGestureRecognizerDirection.allZeros {

                // if the distance exceeds the limit, apply the direction assignment.
                if abs(translation.x) > panGestureDirectionLimit &&
abs(translation.x) > abs(translation.y) {

                    // could be left or right
                    if DEBUG_FLAG {
                        println("left or right \(translation.x)")
                    }
                    if translation.x > 0 {
                        // right
                        panGestureDirection =
UISwipeGestureRecognizerDirection.Right
                    } else {
                        // left
                        panGestureDirection =
UISwipeGestureRecognizerDirection.Left
                    }
                }

                // if the distance exceeds the limit, apply the direction assignment.
                if abs(translation.y) > panGestureDirectionLimit &&
abs(translation.y) > abs(translation.x) {

                    // could be up or down
                    if DEBUG_FLAG {
                        println("up or down \(translation.y)")
                    }

                    if translation.y > 0 {
                        // down
                        panGestureDirection =
UISwipeGestureRecognizerDirection.Down
                    } else {
                        // up
                        panGestureDirection =
UISwipeGestureRecognizerDirection.Up
                    }

                }
            }

            if DEBUG_FLAG {
                switch panGestureDirection {
                case UISwipeGestureRecognizerDirection.Up:
                    println("up")
                case UISwipeGestureRecognizerDirection.Down:
                    println("down")
                case UISwipeGestureRecognizerDirection.Left:
                    println("left")
                case UISwipeGestureRecognizerDirection.Right:
                    println("right")
                case UISwipeGestureRecognizerDirection.allZeros:
                    println("no direction")
                default:
                    println("error")
                }
            }

            // for up and down, change the volume
            if panGestureDirection == UISwipeGestureRecognizerDirection.Up ||
panGestureDirection == UISwipeGestureRecognizerDirection.Down {
                if panGestrueVolumeOriginal < 0 {
                    panGestrueVolumeOriginal = musicPlayer.volume
                }
                var volumeChange =    Float(-translation.y /
panGestureDirectionLimit) * panGestureVolumeSensitivity
                //println("\(volumeChange)")
                var result = panGestrueVolumeOriginal + volumeChange
                if result > 1 {
                    // the volume cannot exceed 1
                    musicPlayer.volume = 1
                } else if result < 0 {
                    // the volume cannot be negative
                    musicPlayer.volume = 0
                } else {
                    musicPlayer.volume = result
                }
                if DEBUG_FLAG {
                    println("setting volume:\(musicPlayer.volume)")
                }

            }

            // trigger the actions when the gesture is over.
            if sender.state == UIGestureRecognizerState.Ended {

                // for left and right, go to previous track or next track
                if panGestureDirection == UISwipeGestureRecognizerDirection.Left
{
                    if DEBUG_FLAG {
                        println("play next song")
                    }

                    playNextMusic()

                }
                if panGestureDirection == UISwipeGestureRecognizerDirection.Right
{
                    if DEBUG_FLAG {
                        println("play previous song")
                    }

                    playPreviousMusic()
                }

                // reset to initial value
                self.panGestrueVolumeOriginal = -100.0
                self.panGestureDirection =
UISwipeGestureRecognizerDirection.allZeros
            }

        }

        /*
            play the previous sound track
        */
        func playPreviousMusic() {
            self.musicPlayer.stop()
            vMusic = previousMusic()
            initMusicPlayer()
            resetUIViews()
            startMusicPlayer()
        }

        /*
            play the next sound track
        */
        func playNextMusic() {
            self.musicPlayer.stop()
            vMusic = nextMusic()
            initMusicPlayer()
            resetUIViews()
            startMusicPlayer()
        }

        /*
            the action being called whent the CD image is tapped.
        */
        func coverImageViewTapped(sender:UITapGestureRecognizer) {
            if DEBUG_FLAG {
                println("coverImageViewTapped \(sender)")
            }
            // play something
            self.toggleMusicPlayer()
        }

        /*
            a concise function. If the music is playing, pause it. otherwise play it.
        */
        func toggleMusicPlayer() {
            if self.musicPlayer.playing {
                pauseMusicPlayer()
            } else {
                startMusicPlayer()
            }
        }

        /*
            start to play the music and apply animation.
        */
        func startMusicPlayer() {
            self.musicPlayer.play()
            self.startCoverAnimation()
        }

        /*
            pause the currently playing music and animation.
        */
        func pauseMusicPlayer() {
            self.musicPlayer.pause()
            self.stopCoverAnimation()
        }

        /*
            start the cover animation
        */
        func startCoverAnimation() {
            if self.coverAnimationTimer.valid {
                return
            }
            coverAnimationTimer = NSTimer.scheduledTimerWithTimeInterval(0.05,
target: self, selector: Selector("rotateCoverImage"), userInfo: nil, repeats: true)
        }
        /*
            the function is used to create cover rotation animation.
        */
        func rotateCoverImage() {
            let view = coverImageView
            // get the current radians
            let radians = atan2f(Float(view.transform.b), Float(view.transform.a))
            // apply the transform to rotate the view, adding 1 degree to the current
radian.
            view.transform =
CGAffineTransformMakeRotation(CGFloat(Double(radians) + M_PI/180.0))

            // if the music is over, restart the current music.
            // the default behavior.
            // for a complete music controller, shall provide interface to change this
setting.
            // e.g. play next, repeat, random and etc...
            if musicPlayer.currentTime >= musicPlayer.duration {
                self.musicPlayer.stop()
                self.startMusicPlayer()
            }

        }

        /*
            stop the cover rotation animation.
        */
        func stopCoverAnimation() {
            if self.coverAnimationTimer.valid {
                self.coverAnimationTimer.invalidate()
            }

        }

        override func viewDidAppear(animated: Bool) {
            super.viewDidAppear(animated)

            println(self.musicPlayer.currentTime)

        }

        override func viewWillDisappear(animated: Bool) {
            super.viewWillDisappear(animated)
            // get rid of the background so the animation goes smoothly
        }

        /*
            get the current music that is being loaded by the controller.
        */
        func currentMusic() -> VMusic {
            return self.musicModel.vMusics[musicIndex]
        }

        /*
            load the next music from the list.
        */
        func nextMusic() -> VMusic {
            musicIndex++
            if musicIndex >= self.musicModel.vMusics.count {
                musicIndex = 0
            }

            return musicModel.vMusics[musicIndex]
        }

        /*
            load the previous music form the list.
        */
        func previousMusic() -> VMusic {
            musicIndex--
            if musicIndex < 0 {
                musicIndex = self.musicModel.vMusics.count - 1
            }
            return musicModel.vMusics[musicIndex]
        }

        /*
            the special function controls the data back from the unwind segue
        */
        @IBAction func unwindFromMusicCollection(segue:UIStoryboardSegue) {
            println("unwindFromMusicCollection")

            var found = false
            if let source = segue.sourceViewController as?
MusicCollectionViewController {
                var selectedMusic = source.selectedMusic!
                for i in 0..<self.musicModel.vMusics.count {
                    if musicModel.vMusics[i].songfilename ==
selectedMusic.songfilename {
                    self.musicIndex = i
                    self.vMusic = selectedMusic
                    found = true
                    }
                }
            }

            // if the music from selected song was not found
            if !found {
                println("unexpected song was selected. break here for debug.")
            }

            initMusicPlayer()
            resetUIViews()
            startMusicPlayer()
        }

}
```

11.1.3 FanViewController.swift

```
//
//  FanViewController.swift
//  VehicleEntertainSystem
//
//  Created by Zongqi Wang on 8/31/15.
//  Copyright (c) 2015 Zongqi Wang. All rights reserved.
//

import UIKit
import AVFoundation

protocol NoisePlayerDelegate {
    func changeNoiseAudio(filename:String, type:String)
    func getNoisePlayer() -> AVAudioPlayer
}

class FanViewController: UIViewController, UITableViewDataSource,
UITableViewDelegate  {

    @IBOutlet weak var tableView: UITableView!
    var noiseModel : NoiseModel!
    var noisePlayerDelegate : NoisePlayerDelegate!
    // the instance that is capable of playing audio files.
```

```
	var audioPlayer : AVAudioPlayer!

	override func viewDidLoad() {
		super.viewDidLoad()

		// Do any additional setup after loading the view.
		// pinch to go back
		var pinchGesture = UIPinchGestureRecognizer(target: self, action: 
"handlePinch:")
		self.view.addGestureRecognizer(pinchGesture)

		self.noiseModel = NoiseModel()
	}

	override func didReceiveMemoryWarning() {
		super.didReceiveMemoryWarning()
		// Dispose of any resources that can be recreated.
	}

	func numberOfSectionsInTableView(tableView: UITableView) -> Int {
		return 1
	}

	func tableView(tableView: UITableView, numberOfRowsInSection section: Int) 
-> Int {
		return self.noiseModel.vNoises.count
	}


	 func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: 
NSIndexPath) -> UITableViewCell {
		//NoiseCollectionCell
		var cell = tableView.dequeueReusableCellWithIdentifier("cell") as! 
UITableViewCell

		cell.textLabel?.text = noiseModel.vNoises[indexPath.row].songname

		return cell
	}

	func handlePinch(sender: UIPinchGestureRecognizer) {
		self.noisePlayerDelegate.getNoisePlayer().stop()

		if sender.scale < 0.7 {
```

```
                    self.navigationController?.popViewControllerAnimated(true)
                }

        }

        func tableView(tableView: UITableView, didSelectRowAtIndexPath indexPath:
NSIndexPath) {
                println("selected in \(indexPath.row)")


noisePlayerDelegate.changeNoiseAudio(self.noiseModel.vNoises[indexPath.row].son
gfilename,    type:self.noiseModel.vNoises[indexPath.row].songfiletype)
                noisePlayerDelegate.getNoisePlayer().play()

        }

}
```

11.1.4 DummyViewController.swift

```
//
//    DummyViewController.swift
//    VehicleEntertainSystem
//
//    Created by Zongqi Wang on 8/31/15.
//    Copyright (c) 2015 Zongqi Wang. All rights reserved.
//

import Foundation
import UIKit

class DummyViewController : UIViewController {

        var dummyInfo : String?
        var dummyImage : UIImage?

        @IBOutlet weak var dummyUIImageView: UIImageView!
        @IBOutlet weak var dummyUITextView: UITextView!


        override func viewDidLoad() {
                super.viewDidLoad()

                self.updateUI()

        }
```

```
    func updateUI() {
        println("updatingui: dummyinfo=\(dummyInfo)")
        if let text = dummyInfo {
            dummyUITextView.text = text
        }
        if let image = dummyImage {
            dummyUIImageView.image = image
        }
    }

    @IBAction func pinched(sender: UIPinchGestureRecognizer) {
        if sender.scale < 0.7 {
            self.navigationController?.popViewControllerAnimated(true)
        }

    }


}
```

11.1.5 MusicCollectionViewController.swift

```
//
//  MusicCollectionViewController.swift
//  VehicleEntertainSystem
//
//  Created by Zongqi Wang on 9/2/15.
//  Copyright (c) 2015 Zongqi Wang. All rights reserved.
//

import UIKit

// the defined protocal for data exchange between musicview and collection view.
protocol MusicCollectionDelegate {
    func getModel() -> MusicModel
    func setSelectedMusicFromCollection(music:VMusic)
}


class MusicCollectionViewController: UIViewController {

    let DEBUG_FLAG = false

    // the music model passed from music view.
    var delegate : MusicCollectionDelegate?
    // the currently selected music
    var selectedMusic : VMusic?
    // a scroll view to contain all the cd views.
    var scrollView : UIScrollView!
    // a list of cd views.
    var cdViews : [UIImageView] = [UIImageView]()


    override func viewDidLoad() {
        super.viewDidLoad()

        // Do any additional setup after loading the view.
        var length : Int = self.delegate!.getModel().vMusics.count
        var index : Int = Int(arc4random_uniform(UInt32(length)))
        selectedMusic = delegate!.getModel().vMusics[index]

        // pinch to go back
        var pinchGesture = UIPinchGestureRecognizer(target: self, action:
"handlePinch:")
        self.view.addGestureRecognizer(pinchGesture)

        updateMusicViews()
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

    func updateMusicViews() {
        // firstly, init as many image views as needed to hold for all songs
        scrollView = UIScrollView(frame: view.bounds)
        scrollView.backgroundColor = UIColor.blackColor()
        scrollView.contentSize = CGSize(width:
600*delegate!.getModel().vMusics.count, height: 768)
        scrollView.autoresizingMask = UIViewAutoresizing.FlexibleWidth
        self.view.addSubview(scrollView)

        if DEBUG_FLAG {
            var button = UIButton(frame: CGRect(x: 10, y: 10, width: 100, height:
100))
            button.setTitle("Test", forState: UIControlState.Normal)
            button.setTitleColor(UIColor.redColor(), forState:
UIControlState.Normal)
```

```
            button.addTarget(self, action: "buttonAction:", forControlEvents:
UIControlEvents.TouchUpInside)
            self.view.addSubview(button)
        }


        var width : CGFloat = 500
        var height : CGFloat = 500
        var offX : CGFloat = 100
        var offY : CGFloat = self.view.center.y
        var firstOffX : CGFloat = 400

        for i in 0..<delegate!.getModel().vMusics.count {
            var vMusic = delegate!.getModel().vMusics[i]
            var img = UIImage(named: vMusic.artworkfilename)
            var view = UIImageView()
            view.bounds = CGRect(x: 0, y: 0, width: width, height: height)
            view.image = img?.circle
            view.contentMode = UIViewContentMode.ScaleAspectFill
            view.center = CGPoint(x: firstOffX+(offX+width)*CGFloat(i), y:
offY)
            view.userInteractionEnabled = true
            self.scrollView.addSubview(view)
            cdViews.append(view)
            var tapGesture = UITapGestureRecognizer(target: self, action:
Selector("cdTapped:"))
            view.addGestureRecognizer(tapGesture)

        }
    }

    func cdTapped(sender : UITapGestureRecognizer) {
        //selectedMusic = delegate!.vMusics[index]
        if DEBUG_FLAG {
            println("cdTapped")
        }

        for i in 0..<self.cdViews.count {
            if sender.view == self.cdViews[i] {
                selectedMusic = delegate!.getModel().vMusics[i]
            }
        }
        if DEBUG_FLAG {
            println("the selected music is: \(selectedMusic!)")
        }
        delegate!.setSelectedMusicFromCollection(selectedMusic!)
        dismissViewControllerAnimated(true, completion: nil)
    }

    func buttonPressed(sender: UIButton) {
        dismissViewControllerAnimated(true, completion: nil)
    }


    func handlePinch(sender: UIPinchGestureRecognizer) {
        if sender.scale < 0.7 {
            dismissViewControllerAnimated(true, completion: nil)
        }

    }

}
```

11.1.6 MusicModel.swift

```
//
//  MusicModel.swift
//  VehicleEntertainSystem
//
//  Created by Zongqi Wang on 9/1/15.
//  Copyright (c) 2015 Zongqi Wang. All rights reserved.
//

import Foundation

public class MusicModel {

    let DEBUG_FLAG = false
    public var vMusics = [VMusic]()

    public init() {
        if DEBUG_FLAG {
            println("init Music Model")
        }

        // get the music plist from the main directory
        if let path = NSBundle.mainBundle().pathForResource("Music", ofType:
"plist") {
            let musicPlist = NSArray(contentsOfFile: path)!
            if DEBUG_FLAG {
                println(musicPlist)
            }

            // load the plist into the model
            // each one of the song information would be saved as vMusic
            // and then stored in the list for later access.
            for i in 0..<musicPlist.count {
                var dict : NSDictionary = musicPlist[i] as! NSDictionary
                var vMusic = VMusic(songfilename: dict["songfilename"] as!
String, songname: dict["songname"] as! String, artist: dict["artist"] as! String, album:
dict["album"] as! String, artworkfilename: dict["artworkfilename"] as! String)
                self.vMusics.append(vMusic)
            }
        }

        if DEBUG_FLAG {
            for i in 0..<self.vMusics.count {
                println("\(i):\t\(vMusics[i])")
            }
        }


    }

}

/*
    a class for storing the music information.
*/
public class VMusic : Printable {
    var album : String!
    var songfilename : String!
    var songname : String!
    var artist : String!
    var artworkfilename : String!
    // by default the type of the audio file would be mp3.
    var songfiletype : String = "mp3"

    public init(songfilename: String, songname: String, artist:String, album:String,
artworkfilename:String) {
        self.album = album
        self.artist = artist
        self.artworkfilename = artworkfilename
```

```
		self.songfilename = songfilename
		self.songname = songname
	}

	public var description: String {
		return "VMusic songfilename:\(songfilename), 
artworkfilename:\(artworkfilename), songname\(songname), artist\(artist), 
album\(album)"
	}
}
```

11.1.7 NoiseModel.swift

```
//
//  NoiseModel.swift
//  VehicleEntertainSystem
//
//  Created by Zongqi Wang on 9/13/15.
//  Copyright (c) 2015 Zongqi Wang. All rights reserved.
//

import Foundation
public class NoiseModel {

	let DEBUG_FLAG = true
	public var vNoises = [VNoise]()

	public init() {
		if DEBUG_FLAG {
			println("init Noise Model")
		}

		// get the music plist from the main directory
		if let path = NSBundle.mainBundle().pathForResource("Noise", ofType: 
"plist") {
			let noisePlist = NSArray(contentsOfFile: path)!

			if DEBUG_FLAG {
				println(noisePlist)
			}

			// load the plist into the model
			// each one of the song information would be saved as vNoise
			// and then stored in the list for later access.
			for i in 0..<noisePlist.count {
				var vNoise = VNoise(songfilename: noisePlist[i] as! String,
songname: noisePlist[i] as! String, songtype: "")
				self.vNoises.append(vNoise)
			}
		}

		if DEBUG_FLAG {
			for i in 0..<self.vNoises.count {
				println("\(i):\t\(vNoises[i])")
			}
		}

	}

}

/*
a class for storing the noise information.
*/
public class VNoise : Printable {
	var songfilename : String!
	var songname : String!
	// by default the type of the audio file would be mp3.
	var songfiletype : String = "mp3"
	var songtype : String!

	public init(songfilename: String, songname: String, songtype: String) {
		self.songfilename = songfilename
		self.songname = songname
		self.songtype = songtype
	}

	public var description: String {
		return "VNoise songfilename:\(songfilename), songname:\(songname),
songtype:\(songtype)"
	}
}
```

11.1.8 Music.plist

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<array>
	<dict>
```

```
        <key>songfilename</key>
        <string>李荣浩 - 不将就 (《何以笙箫默》电影片尾主题曲)</string>
        <key>songname</key>
        <string>不将就 (《何以笙箫默》电影片尾主题曲)</string>
        <key>artist</key>
        <string>李荣浩</string>
        <key>album</key>
        <string></string>
        <key>artworkfilename</key>
        <string>李荣浩 - 不将就 (《何以笙箫默》电影片尾主题曲)</string>
 </dict>
<dict>
      <key>songfilename</key>
      <string>Daydream - 泪花</string>
        <key>songname</key>
        <string>泪花</string>
        <key>artist</key>
        <string>Daydream</string>
        <key>album</key>
        <string>背景音乐之旅•感人之声</string>
        <key>artworkfilename</key>
        <string>Daydream - 泪花</string>
</dict>
 <dict>
        <key>songfilename</key>
        <string>李荣浩 - 模特</string>
        <key>songname</key>
        <string>模特</string>
        <key>artist</key>
        <string>李荣浩</string>
        <key>album</key>
        <string>模特</string>
        <key>artworkfilename</key>
        <string>李荣浩 - 模特</string>
 </dict>
 <dict>
        <key>songfilename</key>
        <string>Otokaze - 夏恋</string>
        <key>songname</key>
        <string>夏恋</string>
        <key>artist</key>
        <string>Otokaze</string>
        <key>album</key>
        <string>Otokaze / 夏恋</string>
        <key>artworkfilename</key>
        <string>Otokaze - 夏恋</string>
 </dict>
 <dict>
        <key>songfilename</key>
        <string>Timbaland - Apologize</string>
        <key>songname</key>
        <string>Apologize</string>
        <key>artist</key>
        <string>Timbaland</string>
        <key>album</key>
        <string>Timbaland Presents: Shock Value</string>
        <key>artworkfilename</key>
        <string>Timbaland - Apologize</string>
 </dict>
<dict>
      <key>songfilename</key>
      <string>Clifford White - A Blessing</string>
        <key>songname</key>
        <string>A Blessing</string>
        <key>artist</key>
        <string>Clifford White</string>
        <key>album</key>
        <string>The Healing Touch</string>
        <key>artworkfilename</key>
        <string>Clifford White - A Blessing</string>
</dict>
 <dict>
        <key>songfilename</key>
        <string>Lene Marlin - A Place Nearby</string>
        <key>songname</key>
        <string>A Place Nearby</string>
        <key>artist</key>
        <string>Lene Marlin</string>
        <key>album</key>
        <string>Playing My Game</string>
        <key>artworkfilename</key>
        <string>Lene Marlin - A Place Nearby</string>
 </dict>
 <dict>
        <key>songfilename</key>
        <string>艋舺 - Brotherhood</string>
        <key>songname</key>
        <string>Brotherhood</string>
        <key>artist</key>
        <string>艋舺</string>
        <key>album</key>
        <string>艋舺  电影原声带</string>
        <key>artworkfilename</key>
        <string>艋舺  - Brotherhood</string>
 </dict>
 <dict>
        <key>songfilename</key>
        <string>Shakira - Can&apos;t Remember To Forget You</string>
        <key>songname</key>
        <string>Can&apos;t Remember To Forget You</string>
        <key>artist</key>
        <string>Shakira</string>
        <key>album</key>
        <string>Can&apos;t Remember To Forget You</string>
        <key>artworkfilename</key>
        <string>Shakira - Can&apos;t Remember To Forget You</string>
 </dict>
 <dict>
        <key>songfilename</key>
        <string>Fall Out Boy - Centuries</string>
        <key>songname</key>
        <string>Centuries</string>
        <key>artist</key>
        <string>Fall Out Boy</string>
        <key>album</key>
        <string>Centuries</string>
        <key>artworkfilename</key>
        <string>Fall Out Boy - Centuries</string>
 </dict>
<dict>
      <key>songfilename</key>
      <string>Cara Dillon - Craigie Hill</string>
        <key>songname</key>
        <string>Craigie Hill</string>
        <key>artist</key>
        <string>Cara Dillon</string>
        <key>album</key>
        <string>Cara Dillon</string>
        <key>artworkfilename</key>
        <string>Cara Dillon - Craigie Hill</string>
</dict>
 <dict>
```

```
        <key>songfilename</key>
        <string>Basshunter - Dirty</string>
        <key>songname</key>
        <string>Dirty</string>
        <key>artist</key>
        <string>Basshunter</string>
        <key>album</key>
        <string>Calling Time</string>
        <key>artworkfilename</key>
        <string>Basshunter - Dirty</string>
    </dict>
    <dict>
        <key>songfilename</key>
        <string>Andemund Orchestra - Dream World</string>
        <key>songname</key>
        <string>Dream World</string>
        <key>artist</key>
        <string>Andemund Orchestra</string>
        <key>album</key>
        <string>减压音乐绿钢琴</string>
        <key>artworkfilename</key>
        <string>Andemund Orchestra - Dream World</string>
    </dict>
    <dict>
        <key>songfilename</key>
        <string>The Cranberries - Dying In The Sun</string>
        <key>songname</key>
        <string>Dying In The Sun</string>
        <key>artist</key>
        <string>The Cranberries</string>
        <key>album</key>
        <string>bury the hatchet</string>
        <key>artworkfilename</key>
        <string>The Cranberries - Dying In The Sun</string>
    </dict>
    <dict>
        <key>songfilename</key>
        <string>曲婉婷 - Everything in the World</string>
        <key>songname</key>
        <string>Everything in the World</string>
        <key>artist</key>
        <string>曲婉婷</string>
        <key>album</key>
        <string>我的歌声里</string>
        <key>artworkfilename</key>
        <string>曲婉婷 - Everything in the World</string>
</dict>
<dict>
        <key>songfilename</key>
        <string>Alan Walker - Fade</string>
        <key>songname</key>
        <string>Fade</string>
        <key>artist</key>
        <string>Alan Walker</string>
        <key>album</key>
        <string>Fade</string>
        <key>artworkfilename</key>
        <string>Alan Walker - Fade</string>
</dict>
<dict>
        <key>songfilename</key>
        <string>Dj Okawari - Flower Dance (纯音乐)</string>
        <key>songname</key>
        <string>Flower Dance (纯音乐)</string>
        <key>artist</key>
        <string>Dj Okawari</string>
        <key>album</key>
        <string>A Cup Of Coffee</string>
        <key>artworkfilename</key>
        <string>Dj Okawari - Flower Dance (纯音乐)</string>
</dict>
<dict>
        <key>songfilename</key>
        <string>Calvin Harris - I Need Your Love　(Extended Version)</string>
        <key>songname</key>
        <string>I Need Your Love　(Extended Version)</string>
        <key>artist</key>
        <string>Calvin Harris</string>
        <key>album</key>
        <string>I Need Your Love</string>
        <key>artworkfilename</key>
        <string>Calvin Harris - I Need Your Love　(Extended Version)</string>
</dict>
<dict>
        <key>songfilename</key>
        <string>Richard Clayderman - Lettre A Ma Mere</string>
        <key>songname</key>
        <string>Lettre A Ma Mere</string>
        <key>artist</key>
        <string>Richard Clayderman</string>
        <key>album</key>
        <string>背景音乐之旅•感人之声</string>
        <key>artworkfilename</key>
        <string>Richard Clayderman - Lettre A Ma Mere</string>
</dict>
<dict>
        <key>songfilename</key>
        <string>Passenger - Let Her Go</string>
        <key>songname</key>
        <string>Let Her Go</string>
        <key>artist</key>
        <string>Passenger</string>
        <key>album</key>
        <string>All The Little Lights</string>
        <key>artworkfilename</key>
        <string>Passenger - Let Her Go</string>
</dict>
<dict>
        <key>songfilename</key>
        <string>Lily Allen - Littlest Things</string>
        <key>songname</key>
        <string>Littlest Things</string>
        <key>artist</key>
        <string>Lily Allen</string>
        <key>album</key>
        <string>Alright, Still</string>
        <key>artworkfilename</key>
        <string>Lily Allen - Littlest Things</string>
</dict>
<dict>
        <key>songfilename</key>
        <string>邓福如 - Me And You</string>
        <key>songname</key>
        <string>Me And You</string>
        <key>artist</key>
        <string>邓福如</string>
        <key>album</key>
        <string>Me And You</string>
        <key>artworkfilename</key>
        <string>邓福如 - Me And You</string>
</dict>
<dict>
```

```
            <key>songfilename</key>
            <string>艋舺 - Monga-Memoir</string>
            <key>songname</key>
            <string>Monga-Memoir</string>
            <key>artist</key>
            <string>艋舺</string>
            <key>album</key>
            <string>艋舺 电影原声带</string>
            <key>artworkfilename</key>
            <string>艋舺 - Monga-Memoir</string>
      </dict>
      <dict>
            <key>songfilename</key>
            <string>Bandari - Mystica (Remix)</string>
            <key>songname</key>
            <string>Mystica (Remix)</string>
            <key>artist</key>
            <string>Bandari</string>
            <key>album</key>
            <string>New Age Top 2010</string>
            <key>artworkfilename</key>
            <string>Bandari - Mystica (Remix)</string>
      </dict>
      <dict>
            <key>songfilename</key>
            <string>Fall Out Boy - My Songs Know What You Did In The Dark Light
Em Up</string>
            <key>songname</key>
            <string>My Songs Know What You Did In The Dark Light Em
Up</string>
            <key>artist</key>
            <string>Fall Out Boy</string>
            <key>album</key>
            <string>Save Rock and Roll</string>
            <key>artworkfilename</key>
            <string></string>
      </dict>
      <dict>
            <key>songfilename</key>
            <string>Enya - Only Time</string>
            <key>songname</key>
            <string>Only Time</string>
            <key>artist</key>
            <string>Enya</string>
```

```
		<key>album</key>
		<string>The Very Best Of Enya</string>
		<key>artworkfilename</key>
		<string>Enya - Only Time</string>
	</dict>
	<dict>
		<key>songfilename</key>
		<string>Madeon - Pay No Mind</string>
		<key>songname</key>
		<string>Pay No Mind</string>
		<key>artist</key>
		<string>Madeon</string>
		<key>album</key>
		<string>Pay No Mind</string>
		<key>artworkfilename</key>
		<string>Madeon - Pay No Mind</string>
	</dict>
	<dict>
		<key>songfilename</key>
		<string>Calvin Harris - Pray to God (Calvin Harris vs Mike Pickering Hacienda Remix)</string>
		<key>songname</key>
		<string>Pray to God (Calvin Harris vs Mike Pickering Hacienda Remix)</string>
		<key>artist</key>
		<string>Calvin Harris</string>
		<key>album</key>
		<string>Pray to God (Remixes)</string>
		<key>artworkfilename</key>
		<string>Calvin Harris - Pray to God (Calvin Harris vs Mike Pickering Hacienda Remix)</string>
	</dict>
	<dict>
		<key>songfilename</key>
		<string>阿南亮子 - Refrain</string>
		<key>songname</key>
		<string>Refrain</string>
		<key>artist</key>
		<string>阿南亮子</string>
		<key>album</key>
		<string>Eternal Light</string>
		<key>artworkfilename</key>
		<string>阿南亮子 - Refrain</string>
	</dict>
	<dict>
		<key>songfilename</key>
		<string>Giorgio Moroder - Right Here, Right Now</string>
		<key>songname</key>
		<string>Right Here, Right Now</string>
		<key>artist</key>
		<string>Giorgio Moroder</string>
		<key>album</key>
		<string>Déjà Vu</string>
		<key>artworkfilename</key>
		<string>Giorgio Moroder - Right Here, Right Now</string>
	</dict>
	<dict>
		<key>songfilename</key>
		<string>Lorde - Royals</string>
		<key>songname</key>
		<string>Royals</string>
		<key>artist</key>
		<string>Lorde</string>
		<key>album</key>
		<string>2014 GRAMMY Nominees</string>
		<key>artworkfilename</key>
		<string>Lorde - Royals</string>
	</dict>
	<dict>
		<key>songfilename</key>
		<string>Wiz Khalifa - See You Again (《速度与激情 7》电影片尾
曲)</string>
		<key>songname</key>
		<string>See You Again</string>
		<key>artist</key>
		<string>Wiz Khalifa</string>
		<key>album</key>
		<string>Furious 7</string>
		<key>artworkfilename</key>
		<string>Wiz Khalifa - See You Again (《速度与激情 7》电影片尾
曲)</string>
	</dict>
	<dict>
		<key>songfilename</key>
		<string>Jami Soul;Mr. Bang;ㅅ ㅣㄴ ㅏㅇ ㅔ - She</string>
		<key>songname</key>
		<string>She</string>
```

```
        <key>artist</key>
        <string>Jami Soul;Mr. Bang;ㅅ ㅣㄴ ㅏㅇ ㅔ</string>
        <key>album</key>
        <string>여보세요</string>
        <key>artworkfilename</key>
        <string>Jami Soul;Mr. Bang;ㅅ ㅣㄴ ㅏㅇ ㅔ - She</string>
</dict>
<dict>
        <key>songfilename</key>
        <string>Fall Out Boy - The Phoenix</string>
        <key>songname</key>
        <string>The Phoenix</string>
        <key>artist</key>
        <string>Fall Out Boy</string>
        <key>album</key>
        <string>Save Rock and Roll</string>
        <key>artworkfilename</key>
        <string>Fall Out Boy - The Phoenix</string>
</dict>
<dict>
        <key>songfilename</key>
        <string>Fetty Wap - Trap Queen (Explicit)</string>
        <key>songname</key>
        <string>Trap Queen (Explicit)</string>
        <key>artist</key>
        <string>Fetty Wap</string>
        <key>album</key>
        <string>Trap Queen</string>
        <key>artworkfilename</key>
        <string>Fetty Wap - Trap Queen (Explicit)</string>
</dict>
<dict>
        <key>songfilename</key>
        <string>Austin Mahone - U</string>
        <key>songname</key>
        <string>U</string>
        <key>artist</key>
        <string>Austin Mahone</string>
        <key>album</key>
        <string>U</string>
        <key>artworkfilename</key>
        <string>Austin Mahone - U</string>
    </dict>
    <dict>
        <key>songfilename</key>
        <string>Shayne Ward - Until You</string>
        <key>songname</key>
        <string>Until You</string>
        <key>artist</key>
        <string>Shayne Ward</string>
        <key>album</key>
        <string>Until You</string>
        <key>artworkfilename</key>
        <string>Shayne Ward - Until You</string>
    </dict>
    <dict>
        <key>songfilename</key>
        <string>Baha Men - Who Let The Dogs Out (《四平青年》电影插曲)</string>
        <key>songname</key>
        <string>Who Let The Dogs Out</string>
        <key>artist</key>
        <string>Baha Men</string>
        <key>album</key>
        <string>Who Let The Dogs Out</string>
        <key>artworkfilename</key>
        <string>Baha Men - Who Let The Dogs Out (《四平青年》电影插曲)</string>
    </dict>
    <dict>
        <key>songfilename</key>
        <string>Lana Del Rey - Young And Beautiful</string>
        <key>songname</key>
        <string>Young And Beautiful</string>
        <key>artist</key>
        <string>Lana Del Rey</string>
        <key>album</key>
        <string>Young And Beautiful</string>
        <key>artworkfilename</key>
        <string>Lana Del Rey - Young And Beautiful</string>
    </dict>
</array>
</plist>
```

11.1.9 Noise.plist

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<array>
    <string>Blender 01</string>
    <string>Blender 02</string>
     <string>Dryer 01</string>
     <string>Dryer 02</string>
     <string>Dryer 03</string>
     <string>Fan 01</string>
     <string>Fan 02</string>
     <string>Fan 03</string>
     <string>Fan 04</string>
     <string>Fan 05</string>
     <string>Fan 06</string>
     <string>Heater 01</string>
     <string>Heater 02</string>
     <string>Heater 03</string>
     <string>Motor 01</string>
     <string>Ocean 01</string>
     <string>Ocean 02</string>
     <string>Ocean 03</string>
     <string>Ocean 04</string>
     <string>Ocean 05</string>
     <string>Ocean 06</string>
     <string>PinkNoise 01</string>
     <string>Rain 01</string>
     <string>Rain 02</string>
     <string>Rain 03</string>
     <string>Rain 04</string>
     <string>Rain 05</string>
     <string>Rain 06</string>
     <string>Rain 07</string>
     <string>Refrigerator 01</string>
     <string>Shower 01</string>
     <string>Storm 01</string>
     <string>Stream 01</string>
     <string>Stream 02</string>
     <string>Train 01</string>
     <string>Train 02</string>
     <string>Underwater 01</string>
     <string>Underwater 02</string>
     <string>Underwater 03</string>
     <string>Vacuum 01</string>
```

```
        <string>Water 01</string>
        <string>WaterBoiling 01</string>
        <string>WaterBoiling 02</string>
        <string>Waterfall 01</string>
        <string>Waves 01</string>
        <string>Waves 02</string>
        <string>White Noise 01</string>
        <string>White Noise 02</string>
        <string>White Noise 03</string>
        <string>White Noise 04</string>
</array>
</plist>
```

11.2 Consent Form

Research Consent Form

The University of Nottingham

Privacy and confidentiality. The participant's personal identity will not be made public in written work, discussions, or presentations. Where it is necessary to refer to the participant then it will be done anonymously in order to preserve the participant's privacy and confidentiality.

Objectives of the study: The purpose of the study is to understand the effects of using gestures in touch enabled devices. This course of research has been approved by the School of Computer Science's Ethics Committee at the University of Nottingham.

Risks of the study: The study involves the gathering of personal data. The data that is gathered will not be shared with anyone except the research supervisors. The data will not be placed on the Internet, and will be destroyed after project is completed.

Data to be captured: The study will gather a range of data to address its objectives. Specifically, where appropriate and with the participant's agreement the researcher will take notes, photos, audio or video recording (please tick as appropriate).

○ Video/audio ○ Photographs/screenshots ○ Fieldnotes

Use of the data: The data will be used by the researcher as to fulfill the coursework requirements for G64HCI and in meetings with the project supervisors to identify a range of topics that are relevant to research. Data may be used in supervision sessions to elaborate findings of the research. Written extracts may be used as examples in written reports.

Reuse of the data: The data will not be reused in any other research projects.

Who has access to the data: Direct access to the data is limited to the researcher and project supervisor. Copies of it will not be passed onto others.

Storage of the data: The data will be stored in secure digital environment at the University of Nottingham and on a password secured computer. The data will not be placed on the Internet at any time.

Your rights: Participants are free to withdraw from the research at any time without explanation and their personal data will be erased. Data collected will be held in a secure and safe manner in accordance with the Data Protection Act 1998. You have the right to request that your data be destroyed at any time.

No person under the age of 13 should take part on this research. The participation of any person under the age of 18 should also be approved by a parent or legal guardian.

Researcher's contact details: psxzw4@nottingham.ac.uk

Participant Name:

Signature:

Date:

Parent/legal guardian's signature (if applicable):