ABSTRACT

Report on the Development of Latin Pronunciation Tutorial Software

Kade Major

Director: David J. White, Ph.D.

This report contains information regarding the development of an iOS application titled VOX. This application is a piece of educational software for students of classical Latin pronunciation. The application was developed using the programming language Swift. The app provides users a means of practicing their aural recognition of vowels, consonants, and diphthongs. It implements an adaptive learning algorithm for the purpose of selecting upcoming questions. Future development aims to increase functionality, begin testing with students, and ultimately publish to app store.

APPROVED BY DIRECTOR OF HONORS THESIS:

_____

Dr. David J. White, Department of Classics

APPROVED BY THE HONORS PROGRAM:

_____

Dr. Elizabeth Corey, Director

DATE: _____

REPORT ON THE DEVELOPMENT OF LATIN TUTORIAL SOFTWARE

A Thesis Submitted to the Faculty of

Baylor University

In Partial Fulfillment of the Requirements for the

Honors Program

By

Kade Major

Waco, Texas

May 2017

TABLE OF CONTENTS

PROJECT SUMMARY

Traditionally Latin has been taught with a nearly exclusive focus on reading and translation in contrast with the dominant pedagogy of modern languages which also emphasize speaking and listening. Recently members of the community devoted to the study and teaching of the classics have begun to question this paradigm and begun to introduce more oral and aural exercises into their teaching. A separate trend within the broad area of language education has been the increasing availability and widespread use of educational software for language acquisition. The impetus behind this thesis project came from a recognition of the potential confluence of these two trends. The objective of this thesis project was to develop an iOS application that could serve as an aid to students of Latin pronunciation.

The project began with a survey of existing resources with an eye to identifying what space there might be for an innovative product. In the realm of iOS applications, existing products include Latin dictionaries, flashcard simulators, digital libraries, grammar quiz games, and parsing tools. The Perseus Project offers classicists free online access to an expansive, fully digitized library of extant Greco-Roman texts. Digital resources within the area of spoken Latin are, however, significantly less developed. A few audiobook recordings of seminal texts in the original language, like the Aeneid, are available for purchase online from a variety of sources. The editors of Wheelock's Latin textbook have published an audio CD as a companion to the textbook which includes

pronunciations of Latin vocabulary from each chapter as well as recitations of the Latin sentences from each chapter. There are also a small number of tutorial videos available on video hosting sites like YouTube aimed at those interested in studying Latin pronunciation. This research revealed the opportunity for the development of an interactive, educationally oriented application to assist in the study of Latin pronunciation.

After identifying this opportunity, the next step in the process was to identify a development platform. Currently the two most popular application platforms are iOS and Android smart phone apps. After consulting with experienced software developers and researching the pros and cons of each platform, I elected to develop for iOS using Swift, the programming language created by Apple specifically for writing iOS apps. This made the project more tractable in two respects: one, it didn't require me to purchase an Android phone for testing and two, it accelerated the development process as Swift is generally a more robust, better integrated programming language.

To learn Swift and the fundamental software design principles necessary to develop this app, I referenced video recordings of a junior level course from Stanford on iOS development. As necessary I also referenced the help manual published by Apple on Swift syntax as well as a variety of online resources. Simultaneously I drafted user interfaces for which I might be able to implement the functionality. These UI and functionality concepts included the following:

- Parse by Ear Quiz Game—the device plays the pronunciation of a Latin word and prompts the user to parse the word i.e. for a noun identify the number, gender, and case, for a verb, the mood, number, person, voice, and tense, etc.

- Learn Vocabulary with pictures—the device plays the pronunciation of a Latin word and presents the user with images of four items then prompts the user to select the image which corresponds to the word pronounced e.g. the user hears the word *tabula* (ENG: table), sees an image of a chair, a man, a table, and a cat, and then selects the image of the table.
- Poetic Meter Quiz—during my thesis presentation, one of the classics students in the audience suggested the implementation of a function to help students study the various meters found in Latin poetry.
- Latin Phonics Keyboard—the screen presents the user with a keyboard comprised of the consonants, diphthongs, and vowels, long and short, found in Latin. As the user presses each button on the keyboard, the device will play the corresponding pronunciation of the letter.
- Latin Phonics Quiz Game—the device plays the pronunciation of a Latin vowel, consonant, or diphthong, and prompts the user to enter the corresponding letter(s) on his keyboard.

For this project, I wrote the code necessary to implement the final functionality, the Latin phonics quiz game.

The app opens to a welcome screen with the name of the app and a button to begin playing the quiz game. After the user proceeds by pressing the button, the app plays the sound of letter, displays a field for the user to enter his response, a button to replay the sound, and a score for the round. Upon a correct answer, the score updates, and the app plays the next sound. Upon an incorrect answer, the score updates, and three buttons appear. One button displays the letter the user submitted as his response. The

second button displays the letter that was the correct answer. The user is able to press either button and hear the corresponding pronunciation. The third button advances the screen to the next question.

One key piece in developing this quiz game was designing and implementing an algorithm for determining the next question. A random number generator combined with an array containing the various characters would have been feasible, but the major shortcoming of such a method is its insensitivity to user input. A purely random selection would result in the user being quizzed equally using questions he consistently answers correctly and questions he consistently answers incorrectly—a less effective approach than prioritizing weak areas. To overcome this problem the app implements an adaptive learning algorithm that uses previous answers to semi-randomly determine the next question.

Upon startup an array is generated with a length equal to the total number of number of letters in the question bank. The array is filled with linearly spaced integer values which are interpreted as intervals where each interval corresponds to a letter in the question bank. To determine the next question, a random number generator produces a value between zero and the maximum value in the array. Once that value is produced, an iterative search through the array is conducted to locate the cell with the least upper bound to the randomly generated number. This number of this cell serves as the key to the array containing the questions and from there the processing necessary to play the selected sound goes on. When a correct answer is given, the interval in the array corresponding to that question is diminished which has the effect of diminishing the probability of its being selected again. When an incorrect answer is given, the interval is

extended effectively increasing the probability of that question being selected again. This adaptive learning algorithm is transferable and could be used in other parts of the app as development continues.

In addition to the anticipated difficulties associated with the research and development process, two other significant roadblocks intervened during the course of the thesis project. The primary one lied in finding a machine for development during the school year. As planned, the majority of the app was designed and implemented over the summer, however, because it was difficult to procure a Mac computer with the requisite administrative privileges and software configuration, I spent a significant amount of time overcoming this obstacle which I'd hoped to spend on making further improvements to the app itself. The secondary roadblock resulted from an unwitting mistake on my part early on the development process: the project was configured in such a way that seems to have prevented the addition of more functional pages to the app storyboard, consequently preventing the straightforward addition of more functionalities. Discovering and understanding this problem took a significant amount of time. Moving past it will require resetting and restructuring the app under a new project which I intend to do as I continue development.

Completion of this thesis project and the current state of the app represents a major milestone in the broader timeline of objectives for this project. At this point I've identified a niche functionality ripe with development potential, learned an unfamiliar programming language, written a basic prototype to prove the concept in this language, and outlined plans for future development. As I build out more functionality I hope to begin testing the app and receiving feedback from Latin students at The Ambrose School

where I received my K-12 education. This feedback and testing will provide guidance as I continue to developing and help me better understand one segment of the potential market for this app once it reaches product level quality.

APPENDIX

```
//
//  ViewController.swift
//  Vox Draft 1
//

import UIKit
import AVFoundation

class ViewController: UIViewController {



//self.view.addSubview(ViewControllerforAlphabetPronunciati
on)

    override func prepare(for segue: UIStoryboardSegue,
sender: Any?) {
        if segue.identifier == "RoundCompleteSegue" {
            let nextScene = segue.destination as!
ViewControllerforResults
            nextScene.resultsholder =
brain.GenerateFinalResultsString()
        }
    }

    fileprivate var brain = VoxBrain() // this object
houses all the functions for the business logic behind the
app
    var audioPlayerforQuestionsAndCorrect = AVAudioPlayer()
//these audioplayer instantiations play the vowel and
consonant sounds
    var audioPlayerforIncorrectSubmission = AVAudioPlayer()


    fileprivate var isSubmitView = true
    //isSubmitView is true while the user inputs a response
    //when the user presses submit, var becomes false to
display outcome until user moves to next question
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view,
typically from a nib.
        do{
            try audioPlayerforQuestionsAndCorrect =
AVAudioPlayer(contentsOf: brain.NewQuestion() as URL)
            audioPlayerforQuestionsAndCorrect.play()
        }
```

```
        catch{
            print("error")
        }
        ReplayIncorrectButton.isHidden = true
    }

    @IBOutlet weak var Question: UILabel! //this label
displays the character being quizzed on


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

    @IBOutlet weak var SubmitAdvanceButton: UIButton!
    @IBOutlet weak var Result: UILabel!

    @IBOutlet weak var ScoreMatrixDisplayTemp: UILabel!

    @IBOutlet weak var ScoreDisplay: UILabel!

    @IBAction func TouchSubmit(_ sender: UIButton) {

        if (isSubmitView)
        {
            let outcome =
brain.CheckAnswer(ResponseField.text!)
            Result.text = outcome.0
            ScoreDisplay.text = "Score: " +
String(brain.scoreCorrect) + "/" +
String(brain.scoreCorrect + brain.scoreIncorrect)
            ScoreMatrixDisplayTemp.text =
brain.GenerateProbArrayString()
            ReplayButton.setTitle("Replay", for:
UIControlState())
            if (outcome.1)
            {
                Result.backgroundColor = UIColor.green
                ReplayIncorrectButton.isHidden = true
                let seconds = 1.0
                let delay = seconds * Double(NSEC_PER_SEC)
// nanoseconds per seconds
                let dispatchTime = DispatchTime.now() +
Double(Int64(delay)) / Double(NSEC_PER_SEC)
```

```
                DispatchQueue.main.asyncAfter(deadline:
dispatchTime, execute: {
                    if
(self.brain.scoreCorrect+self.brain.scoreIncorrect >=
self.brain.numOfQuestionsInRound)
                    {
                        self.performSegue(withIdentifier:
"RoundCompleteSegue", sender:
self.brain.responseAnswerScoreMatrix)
                    }
                    else
                    {
                        self.ResponseField.text = ""
                        self.Result.backgroundColor =
UIColor.blue
                        self.Result.text = ""
                        do{
                            try
self.audioPlayerforQuestionsAndCorrect =
AVAudioPlayer(contentsOf: self.brain.NewQuestion() as URL)

self.audioPlayerforQuestionsAndCorrect.play()
                        }
                        catch{
                            print("error")
                        }
                    }
                    // here code performed with delay

                })                //end delay


            }
            else //incorrect answer submitted
            {
                ReplayIncorrectButton.isHidden = false

ReplayIncorrectButton.setTitle(brain.mostRecentSubmission,
for: UIControlState())
                Result.backgroundColor = UIColor.red
                SubmitAdvanceButton.setTitle("Next
Question", for: UIControlState())
                isSubmitView = false

ReplayButton.setTitle(brain.alphArrayStringsOnly[brain.curr
entVowelIndex], for: UIControlState())
```

```
            }


        }
        else
        {
            if (brain.scoreCorrect+brain.scoreIncorrect >=
brain.numOfQuestionsInRound)
            {
                performSegue(withIdentifier:
"RoundCompleteSegue", sender:
brain.responseAnswerScoreMatrix)
            }
            else
            {
                ReplayButton.setTitle("Replay", for:
UIControlState())
                ReplayIncorrectButton.isHidden = true
                Result.backgroundColor = UIColor.blue
                SubmitAdvanceButton.setTitle("Submit", for:
UIControlState())
                ResponseField.text = ""
                Result.text = ""
                do{
                    try audioPlayerforQuestionsAndCorrect =
AVAudioPlayer(contentsOf: brain.NewQuestion() as URL)
                    audioPlayerforQuestionsAndCorrect.play()
                }
                catch{
                    print("error")
                }

                isSubmitView = true
            }
        }
    }
    @IBOutlet weak var ReplayIncorrectButton: UIButton!
    @IBAction func PlayIncorrectSubmission(_ sender:
UIButton) {
        do{
            try audioPlayerforIncorrectSubmission =
AVAudioPlayer(contentsOf:
brain.ProvideURLforLastSubmission() as URL)
            audioPlayerforIncorrectSubmission.stop()
            audioPlayerforIncorrectSubmission.play()
        }
        catch{
```

```
            print("error")
        }

    }

    @IBOutlet weak var ReplayButton: UIButton!
    @IBAction func PressedReplay(_ sender: UIButton) {
        audioPlayerforQuestionsAndCorrect.stop()
        audioPlayerforQuestionsAndCorrect.play()
    }
    @IBOutlet weak var ResponseField: UITextField!
}
```

```
//
//  VoxBrain.swift
//  Vox Draft 1
//
//

import Foundation

class VoxBrain
{
    var currentVowelIndex = 0

    let alphArrayWithPaths = [("a",
Bundle.main.path(forResource: "a short", ofType: "wav")!),
("ā",Bundle.main.path(forResource: "a long", ofType:
"wav")!), ("e", Bundle.main.path(forResource: "e short",
ofType: "wav")!), ("ē", Bundle.main.path(forResource: "e
long", ofType: "wav")!), ("i", Bundle.main.path(forResource:
"i short", ofType: "wav")!), ("ī",
Bundle.main.path(forResource: "i long", ofType: "wav")!),
("o", Bundle.main.path(forResource: "o short", ofType:
"wav")!), ("ō", Bundle.main.path(forResource: "o long",
ofType: "wav")!), ("u", Bundle.main.path(forResource: "u
short", ofType: "wav")!), ("ū",
Bundle.main.path(forResource: "u long", ofType: "wav")!),
("y", Bundle.main.path(forResource: "y", ofType: "wav")!),
("ae", Bundle.main.path(forResource: "ae", ofType: "wav")!),
("au", Bundle.main.path(forResource: "au", ofType: "wav")!),
("ei", Bundle.main.path(forResource: "ei", ofType: "wav")!),
```

```
("oe", Bundle.main.path(forResource: "oe", ofType: "wav")!),
("eu", Bundle.main.path(forResource: "eu", ofType: "wav")!),
("ui", Bundle.main.path(forResource: "ui", ofType: "wav")!),
/*("b", Bundle.main.path(forResource: "b", ofType: "wav")!),
("c", Bundle.main.path(forResource: "c", ofType: "wav")!),
("d", Bundle.main.path(forResource: "d", ofType: "wav")!),
("f", Bundle.main.path(forResource: "f", ofType: "wav")!),
("h", Bundle.main.path(forResource: "h", ofType: "wav")!),
("l", Bundle.main.path(forResource: "L", ofType: "wav")!),
("m", Bundle.main.path(forResource: "M", ofType: "wav")!),
("n", Bundle.main.path(forResource: "n", ofType: "wav")!),
("p", Bundle.main.path(forResource: "p", ofType: "wav")!),
("q", Bundle.main.path(forResource: "qu", ofType: "wav")!),
("r", Bundle.main.path(forResource: "r", ofType: "wav")!),
("s", Bundle.main.path(forResource: "s", ofType: "wav")!),
("t", Bundle.main.path(forResource: "t", ofType: "wav")!),
("v", Bundle.main.path(forResource: "v", ofType: "wav")!),
("x", Bundle.main.path(forResource: "x", ofType: "wav")!),
("z", Bundle.main.path(forResource: "z", ofType: "wav")!)*/]

    let sizeOfQuestionSet : Int
    var mostRecentSubmission = String()


    var alphArrayWithURLs = [(String, URL)]()
    var alphArrayStringsOnly = [String]()

    var probabilityArray = [Int]()
    var responseAnswerScoreMatrix = [[Int]]()
    //[row: submitted answer] [column: correct answer]

    var scoreCorrect = 0
    var scoreIncorrect = 0
    let numOfQuestionsInRound = 5


    init()
    {
        sizeOfQuestionSet = alphArrayWithPaths.count
        responseAnswerScoreMatrix = Array(repeating:
Array(repeating: 0, count: sizeOfQuestionSet+1), count:
sizeOfQuestionSet)
        probabilityArray = Array(1...sizeOfQuestionSet)
//the number of cells in the probability array is equal to
the number of questions
        for (index, _) in probabilityArray.enumerated()
        {
```

```
            probabilityArray[index] *= 10 //prob array
looks like this: [0, 10, 20, 30, ... numofquestions*10]

        }
        for index in alphArrayWithPaths{
        alphArrayWithURLs.append((index.0,
URL(fileURLWithPath: index.1)))
        alphArrayStringsOnly.append(index.0)
        }
    }

    func GenerateMatrixString () -> String{
        var output = ""
        for row in responseAnswerScoreMatrix
        {
            for column in row
            {
                output.append(String(column))
            }
            output += "\n"
        }
        return output
    }

    func GenerateProbArrayString () -> String{
        var output = ""
        for item in probabilityArray
        {
            output.append(String(item))
            output += ", "
        }
        return output
    }

    func CheckAnswer(_ submission: String) -> (String,
Bool){

        var feedbackForUser: String
        var result: Bool
        mostRecentSubmission = submission

        responseAnswerScoreMatrix[currentVowelIndex][11] +=
1

        if(submission ==
alphArrayWithURLs[currentVowelIndex].0)
        {
```

```
            result = true
            UpdateProbArray(result)
            feedbackForUser = "correct"

responseAnswerScoreMatrix[currentVowelIndex][currentVowelIn
dex] += 1
            scoreCorrect += 1
        }
        else
        {
            result = false
            UpdateProbArray(result)
            feedbackForUser = "incorrect" + " Correct
Answer: " + alphArrayWithURLs[currentVowelIndex].0
            if let indexOfIncorrectResponse =
alphArrayStringsOnly.index(of: submission)
                {

responseAnswerScoreMatrix[indexOfIncorrectResponse][current
VowelIndex] += 1
                    print(indexOfIncorrectResponse)
            }
            scoreIncorrect += 1
        }
        return (feedbackForUser, result)
    }

    func NewQuestion() -> URL{

        currentVowelIndex = 0
        let maxPArrayVal = UInt32(probabilityArray.last!)
        let keyToProbArray =
Int(arc4random_uniform(maxPArrayVal))
        while (keyToProbArray >
probabilityArray[currentVowelIndex])
        {
            currentVowelIndex += 1
        }
        return alphArrayWithURLs[currentVowelIndex].1
    }

    func UpdateProbArray(_ answerCorrect: Bool)
    {
        var iterator = currentVowelIndex
        while (answerCorrect && iterator <
probabilityArray.count && (iterator == 0 ||
```

```
probabilityArray[iterator] - probabilityArray[iterator-1] >
2))
        {
                probabilityArray[iterator] -= 2
                iterator += 1
        }
        while (!answerCorrect && iterator <
probabilityArray.count) {
            probabilityArray[iterator] += 10
            iterator += 1
        }
    }

    func GenerateFinalResultsString() -> String
    {
        let output = "You scored " +
String(Int(Double(scoreCorrect)/Double(scoreCorrect+scoreIn
correct)*100)) + "%"
        return output
    }

    func ProvideURLforLastSubmission() -> URL{
        var indexOfRecentSubmission = Int()
        if let indexOfLastSubmission =
alphArrayStringsOnly.index(of: mostRecentSubmission)
        {
            indexOfRecentSubmission = indexOfLastSubmission
        }
        return alphArrayWithURLs[indexOfRecentSubmission].1
    }

}



//
//  AppDelegate.swift
//  Vox Draft 1
////

import UIKit
import CoreData

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?
```

```
    private func application(application: UIApplication,
didFinishLaunchingWithOptions launchOptions: [NSObject:
AnyObject]?) -> Bool {
        // Override point for customization after
application launch.
        return true
    }

    func applicationWillResignActive(_ application:
UIApplication) {
        // Sent when the application is about to move from
active to inactive state. This can occur for certain types
of temporary interruptions (such as an incoming phone call
or SMS message) or when the user quits the application and
it begins the transition to the background state.
        // Use this method to pause ongoing tasks, disable
timers, and throttle down OpenGL ES frame rates. Games
should use this method to pause the game.
    }

    func applicationDidEnterBackground(_ application:
UIApplication) {
        // Use this method to release shared resources,
save user data, invalidate timers, and store enough
application state information to restore your application
to its current state in case it is terminated later.
        // If your application supports background
execution, this method is called instead of
applicationWillTerminate: when the user quits.
    }

    func applicationWillEnterForeground(_ application:
UIApplication) {
        // Called as part of the transition from the
background to the inactive state; here you can undo many of
the changes made on entering the background.
    }

    func applicationDidBecomeActive(_ application:
UIApplication) {
        // Restart any tasks that were paused (or not yet
started) while the application was inactive. If the
application was previously in the background, optionally
refresh the user interface.
    }
```

```
    func applicationWillTerminate(_ application:
UIApplication) {
        // Called when the application is about to
terminate. Save data if appropriate. See also
applicationDidEnterBackground:.
        // Saves changes in the application's managed
object context before the application terminates.
        self.saveContext()
    }

    // MARK: - Core Data stack

    lazy var applicationDocumentsDirectory: NSURL = {
        // The directory the application uses to store the
Core Data store file. This code uses a directory named
"kademajor.Vox_Draft_1" in the application's documents
Application Support directory.
        let urls =
FileManager.default.urls(for: .documentDirectory,
in: .userDomainMask)
        return urls[urls.count-1] as NSURL
    }()

    lazy var managedObjectModel: NSManagedObjectModel = {
        // The managed object model for the application.
This property is not optional. It is a fatal error for the
application not to be able to find and load its model.
        let modelURL = Bundle.main.url(forResource:
"Vox_Draft_1", withExtension: "momd")!
        return NSManagedObjectModel(contentsOf: modelURL)!
    }()

    lazy var persistentStoreCoordinator:
NSPersistentStoreCoordinator = {
        // The persistent store coordinator for the
application. This implementation creates and returns a
coordinator, having added the store for the application to
it. This property is optional since there are legitimate
error conditions that could cause the creation of the store
to fail.
        // Create the coordinator and store
        let coordinator =
NSPersistentStoreCoordinator(managedObjectModel:
self.managedObjectModel)
```

```
        let url =
self.applicationDocumentsDirectory.appendingPathComponent("
SingleViewCoreData.sqlite")
        var failureReason = "There was an error creating or
loading the application's saved data."
        do {
            try coordinator.addPersistentStore(ofType:
NSSQLiteStoreType, configurationName: nil, at: url, options:
nil)
        } catch {
            // Report any error we got.
            var dict = [String: AnyObject]()
            dict[NSLocalizedDescriptionKey] = "Failed to
initialize the application's saved data" as AnyObject?
            dict[NSLocalizedFailureReasonErrorKey] =
failureReason as AnyObject?

            dict[NSUnderlyingErrorKey] = error as NSError
            let wrappedError = NSError(domain:
"YOUR_ERROR_DOMAIN", code: 9999, userInfo: dict)
            // Replace this with code to handle the error
appropriately.
            // abort() causes the application to generate a
crash log and terminate. You should not use this function
in a shipping application, although it may be useful during
development.
            NSLog("Unresolved error \(wrappedError),
\(wrappedError.userInfo)")
            abort()
        }

        return coordinator
    }()

    lazy var managedObjectContext: NSManagedObjectContext =
{
        // Returns the managed object context for the
application (which is already bound to the persistent store
coordinator for the application.) This property is optional
since there are legitimate error conditions that could
cause the creation of the context to fail.
        let coordinator = self.persistentStoreCoordinator
        var managedObjectContext =
NSManagedObjectContext(concurrencyType: .mainQueueConcurren
cyType)
        managedObjectContext.persistentStoreCoordinator =
coordinator
```

```
        return managedObjectContext
    }()

    // MARK: - Core Data Saving support

    func saveContext () {
        if managedObjectContext.hasChanges {
            do {
                try managedObjectContext.save()
            } catch {
                // Replace this implementation with code to
handle the error appropriately.
                // abort() causes the application to
generate a crash log and terminate. You should not use this
function in a shipping application, although it may be
useful during development.
                let nserror = error as NSError
                NSLog("Unresolved error \(nserror),
\(nserror.userInfo)")
                abort()
            }
        }
    }

}


//
//  ViewController.swift
//  Vox Draft 1
//

import UIKit
import AVFoundation

class ViewController: UIViewController {



//self.view.addSubview(ViewControllerforAlphabetPronunciati
on)

    override func prepare(for segue: UIStoryboardSegue,
sender: Any?) {
        if segue.identifier == "RoundCompleteSegue" {
            let nextScene = segue.destination as!
ViewControllerforResults
```

```
            nextScene.resultsholder =
brain.GenerateFinalResultsString()
        }
    }

    fileprivate var brain = VoxBrain() // this object
houses all the functions for the business logic behind the
app
    var audioPlayerforQuestionsAndCorrect = AVAudioPlayer()
//these audioplayer instantiations play the vowel and
consonant sounds
    var audioPlayerforIncorrectSubmission = AVAudioPlayer()


    fileprivate var isSubmitView = true
    //isSubmitView is true while the user inputs a response
    //when the user presses submit, var becomes false to
display outcome until user moves to next question
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view,
typically from a nib.
        do{
            try audioPlayerforQuestionsAndCorrect =
AVAudioPlayer(contentsOf: brain.NewQuestion() as URL)
            audioPlayerforQuestionsAndCorrect.play()
        }
        catch{
            print("error")
        }
        ReplayIncorrectButton.isHidden = true
    }

    @IBOutlet weak var Question: UILabel! //this label
displays the character being quizzed on


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

    @IBOutlet weak var SubmitAdvanceButton: UIButton!
    @IBOutlet weak var Result: UILabel!

    @IBOutlet weak var ScoreMatrixDisplayTemp: UILabel!
```

```
    @IBOutlet weak var ScoreDisplay: UILabel!

    @IBAction func TouchSubmit(_ sender: UIButton) {

        if (isSubmitView)
        {
            let outcome =
brain.CheckAnswer(ResponseField.text!)
            Result.text = outcome.0
            ScoreDisplay.text = "Score: " +
String(brain.scoreCorrect) + "/" +
String(brain.scoreCorrect + brain.scoreIncorrect)
            ScoreMatrixDisplayTemp.text =
brain.GenerateProbArrayString()
            ReplayButton.setTitle("Replay", for:
UIControlState())
            if (outcome.1)
            {
                Result.backgroundColor = UIColor.green
                ReplayIncorrectButton.isHidden = true
                let seconds = 1.0
                let delay = seconds * Double(NSEC_PER_SEC)
// nanoseconds per seconds
                let dispatchTime = DispatchTime.now() +
Double(Int64(delay)) / Double(NSEC_PER_SEC)

                DispatchQueue.main.asyncAfter(deadline:
dispatchTime, execute: {
                    if
(self.brain.scoreCorrect+self.brain.scoreIncorrect >=
self.brain.numOfQuestionsInRound)
                    {
                        self.performSegue(withIdentifier:
"RoundCompleteSegue", sender:
self.brain.responseAnswerScoreMatrix)
                    }
                    else
                    {
                        self.ResponseField.text = ""
                        self.Result.backgroundColor =
UIColor.blue
                        self.Result.text = ""
                        do{
                            try
self.audioPlayerforQuestionsAndCorrect =
AVAudioPlayer(contentsOf: self.brain.NewQuestion() as URL)
```

```
self.audioPlayerforQuestionsAndCorrect.play()
                        }
                        catch{
                            print("error")
                        }
                    }
                    // here code perfomed with delay

                })                //end delay


            }
            else //incorrect answer submitted
            {
                ReplayIncorrectButton.isHidden = false

ReplayIncorrectButton.setTitle(brain.mostRecentSubmission,
for: UIControlState())
                Result.backgroundColor = UIColor.red
                SubmitAdvanceButton.setTitle("Next
Question", for: UIControlState())
                isSubmitView = false

ReplayButton.setTitle(brain.alphArrayStringsOnly[brain.curr
entVowelIndex], for: UIControlState())

            }


        }
        else
        {
            if (brain.scoreCorrect+brain.scoreIncorrect >=
brain.numOfQuestionsInRound)
            {
                performSegue(withIdentifier:
"RoundCompleteSegue", sender:
brain.responseAnswerScoreMatrix)
            }
            else
            {
                ReplayButton.setTitle("Replay", for:
UIControlState())
                ReplayIncorrectButton.isHidden = true
                Result.backgroundColor = UIColor.blue
```

```
                SubmitAdvanceButton.setTitle("Submit", for:
UIControlState())
                ResponseField.text = ""
                Result.text = ""
                do{
                    try audioPlayerforQuestionsAndCorrect =
AVAudioPlayer(contentsOf: brain.NewQuestion() as URL)
                    audioPlayerforQuestionsAndCorrect.play()
                }
                catch{
                    print("error")
                }

                isSubmitView = true
            }
        }
    }
    @IBOutlet weak var ReplayIncorrectButton: UIButton!
    @IBAction func PlayIncorrectSubmission(_ sender:
UIButton) {
        do{
            try audioPlayerforIncorrectSubmission =
AVAudioPlayer(contentsOf:
brain.ProvideURLforLastSubmission() as URL)
            audioPlayerforIncorrectSubmission.stop()
            audioPlayerforIncorrectSubmission.play()
        }
        catch{
            print("error")
        }

    }

    @IBOutlet weak var ReplayButton: UIButton!
    @IBAction func PressedReplay(_ sender: UIButton) {
        audioPlayerforQuestionsAndCorrect.stop()
        audioPlayerforQuestionsAndCorrect.play()
    }
    @IBOutlet weak var ResponseField: UITextField!
}


//  ViewControllerforAlphabetPronunciation.swift
//  Vox Draft 1
//

import UIKit
```

```
class AlphaView: UIViewController {


    override func viewDidLoad() {
        super.viewDidLoad()

        // Do any additional setup after loading the view.
    }

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


    /*
     // MARK: - Navigation

     // In a storyboard-based application, you will often
want to do a little preparation before navigation
     override func prepareForSegue(segue: UIStoryboardSegue,
sender: AnyObject?) {
     // Get the new view controller using
segue.destinationViewController.
     // Pass the selected object to the new view controller.
     }
     */




//  ViewControllerforResults.swift
//  Vox Draft 1
//

import UIKit

class ViewControllerforResults: UIViewController {
    var resultsholder = String()

    override func viewDidLoad() {
        super.viewDidLoad()
        ResultsLabel.text = resultsholder

        // Do any additional setup after loading the view.
    }
```

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


    @IBOutlet weak var ResultsLabel: UILabel!
    /*
    // MARK: - Navigation

    // In a storyboard-based application, you will often
want to do a little preparation before navigation
    override func prepareForSegue(segue: UIStoryboardSegue,
sender: AnyObject?) {
        // Get the new view controller using
segue.destinationViewController.
        // Pass the selected object to the new view
controller.
    }
    */

}
```