Abandon

User instructions:

Landing page:
To add a word into the database, submit the Chinese character(s) in the text field
–if the word is already in the database, an alert will pop up and say so
–if word can't be found, an alert will pop up and say so

WordBank page:
To see a word entry, touch the box containing that character.
To go to the menu, click menu on the upper right hand corner which will reveal all other menu options.

CharacterInfo page:
Click help button for info on available gestures.

Record page:
Click record to start recording. Will record consecutively for Chinese and then for English word.
When there are no more words left to record, will automatically dismiss view.

VocabExport page:
Tap boxes to select/deselect words to export. On finishing, will export concatenation of Chinese and English recordings of every word.
Exported m4a file is available in App tab when device is connected to iTunes. (Will be in a folder in Documents folder called "ExportedVocab")

The drawing game can be played by clicking on the menu button in the top-right hand corner and choosing "Practice Drawing." Once you enter this game, the goal is to correctly draw each Chinese word whose English definition is displayed at the top. When you think you've drawn the word correctly, hit "Done" on the toolbar at the bottom. If you make a mistake, hit "Start Over" to restart drawing the word from scratch. Once you hit "Done," it will give you another word to draw and it will display your last drawing and the correct Chinese word in the bottom left-hand corner. Once you've cycled through all of the words in your database, hit "Done" one last time to return to the main menu.

The memory game can be played by clicking on the menu button in the top-right hand corner and choosing "Play memory." Once the game it starts, your goal is to match each card. Each design represents a different type of card, the cards on the top contain the english words and the cards on bottom contain the chinese words. Click on one of each card to match them. Try to match them all before time runs out!

The bubble game can be played by clicking on the menu button in the top-right hand corner and choosing "Play bubble game." Once the game starts, it will give you some basic instructions on how to play. At the top of the screen is a bar with an english definition. Your goal is to find the bubble containing the chinese word that corresponds to that definition. Once you do, click on that bubble and drag it onto the bar where the definition is written. If you're correct, the bubble will pop! And then a new definition will be displayed. If not, you have to try again! Try to pop all the bubbles before time runs out!

Technical summary:

The in-app dictionary, character decomposition, and radical database is in txt files that are read in and stored inside CoreData (for faster querying) upon first launch of the app. Similarly, every word added into the word bank is also stored as an managed object model and placed in the queue of unrecorded words. CoreData methods are managed through convenience methods provided by external library called Magical Record.

On opening the Record page, all words in the unrecorded words queue are loaded. As each word is recorded with an AVAudioRecorder object, the word is deleted from the queue.

The WordBank page gets all word objects from the CoreData delegate (which for all view controllers is the AppDelegate) and loads them each as a cell in a UICollectionView. The UICollectionView methods are managed through convenience methods provided by an external library called MGBox.

On tapping a word on the WordBank page, app transitions modally to the CharacterView page, which takes the NSManagedObject of the cell that called it and places information into the fields accordingly. There are several UIGestureRecognizers on different labels, one for playing the recording for the word and one for revealing the character decomposition. If the UITextView is edited, the value is stored in the word object's 'mnemonic' field.

On the Vocabulary Export page, words are again stored as custom cells in a UICollectionView. If the word doesn't have a recording, then it will not appear in the collection. An array keeps track of words that are added or selected. Upon finishing selection, each recording for each word in the array will be loaded into an AVAsset, which is then added onto an AVMutableComposition. Upon completion of concatenation, the m4a file is saved to a folder called "ExportedVocab" in the app's Documents folder.

For the Navigation View Controller, the menu bar button and toggle function is managed by an external library called REMenu.

The drawing game (controlled by CGCViewController) is probably the simplest of the three. It uses a UIImageView as a drawing canvas for the user to draw characters on. To do this it calls a method UIGraphicsBeginImageContext that creates a bitmap-based graphics context to draw on. Each time a finger is moved on the screen or picked up, it draws onto this context and then sets the UIImageView to be image from the context. When the user hits the Done button on the toolbar at the bottom, it will then move the image into another UIImageView where the user can see how it compares to the correct drawing of the word. The user can hit Start Over if he/she makes a mistake while drawing.

The Memory game (controlled by CCRViewController) uses a UICollectionView with a custom cell called CCRMemoryCardCell that contains a UIButton. When the game is started, it grabs the information from the coreDataDelegate that Gwen has created and based on the number of words in there, it adjusts the size of each card to fit nicely on the screen. When a button/card is clicked on, the button uses NSNotificationCenter to send a notification that it has been flipped over.

CCRViewController deals with this notification in a method called respondToLooking. Here it looks and sees if a card has already been flipped over. If not, it leaves this card face up and stores the cell in a property called prevCell. If another card has been flipped over (ie prevCell != nil) then it checks to see if the word stored in prevCell is the same as the word stored in the cell that has just been flipped over. If they are, it runs an animation in a function called getRidOfCell (in CCRMemoryCardCell) that moves the cards off screen. If they're not, then it flips the cards over again. And after either of these two results, it resets prevCell to nil.

The Bubble game (controlled by BubbleGameViewController) first reads from Gwen's coreDataDelegate to see how many words the user has inputted. Then it creates a number of custom buttons, from my class that I created called BubbleView, equal to the number of words. Upon creating the bubble, it sets two timers to animate the bubble. The first timer, called animationTimer, makes the button move in a set direction. The second timer, switchDirectionTimer, switches the direction that the button is being moved in. I found that having two timers instead of one made the animation look smoother. When a bubble is clicked on, it automatically follows the finger around the screen. If the user drops it on the top bar, checked by seeing if the button has been dropped in a spot where the y-coordinate is less than 70 (the height of the definition bar) then it will send a notification through NSNotificationCenter to the BubbleGameViewController to see if the word in the definition is the same as the word in the bubble. If they are, it calls a method called animatePopping in the BubbleView class that removes the bubble from the superView. If they're not, it runs an animation, again in the bubbleView class, called animateRejection.