CSE 438: Mobile Application Development

Lab 2: Virtual Pet App

Overview

In this lab, you will create an app to take care of your very own virtual pets! The app will only have one screen and simple logic, but we will use this as an opportunity to learn about creating custom layouts using Auto Layout. This will also be a great opportunity to learn how to program with the MVC programming paradigm and adhere to best practices in Swift.

Please **read this entire PDF** prior to starting work on your lab. Every section contains information relevant to how you will be graded, not just the Requirements section. To get 100% an app should meet the requirements in the best ways (as described in lecture and this document) and shouldn't crash.

Please test opening your zip file on another computer to make sure the files are included properly. TA's are not liable for broken submissions and the late penalty will apply.

Details

Due date: Monday, Sept 27th by 11:59 PM CT
Grading: This lab is out of 75 points total. The exact point distribution is described in the "Requirements" section below.
Submission: Zip the entire project folder and submit it on Canvas. Please name the Xcode project **"FirstNameLastName-Lab2".** *Describe* your creative feature as outlined in the "Creative Portions" section in a readme.txt file in your folder.

Description

This lab requires you to create your own app from scratch. Although many of the implementation details are left to you, there are some helpful guidelines and code snippets to get you started in the "Helpful Advice and Code Snippets" section below. Many of the topics covered in this lab have been discussed in lecture as well. Additional resources are available online in the Swift documentation and UIKit documentation that Apple provides.

The goal of this lab is to create an app that allows users to play with their virtual pets. There should be only one screen, but the layout should work correctly no matter what device size or orientation is used.

This lab will be graded quite different from most. We will be grading on a variety of device sizes and orientations (not just an iPhone 12 Pro in portrait), and we will grade the quality of your code. Points will be deducted if you programmatically insert and constrain view objects instead of using Storyboard and AutoLayout. In terms of code quality, you should adhere to the MVC pattern and follow modern Swift style guidelines. More information about this is provided later in this document.

Please detail your creative feature in the same way you did for lab 1. Tell us what you did (what it does), how you did it, and why it improves the app. Convince us that your creative work is worth 10 points. If you it difficult to suitably describe your creative project in this way, then your feature is probably not significant enough. For example, adding another pet is not creative enough because the only real difference is an additional option in the pet list. Please also document how to use the creative feature and what the additional behavior is.

Requirements

[25 points] The layout is correct no matter the device size or orientation.
[10 points] Five pets can be fed and played with, and their values update appropriately.
[5 points] Each pet has a custom image and color. Switching between pets updates the image, background color, and color of each display bar.
[5 points] The display bars animate when the pet is played or fed, but jump immediately to the correct values when switching between pets.
[10 points] The code adheres to the MVC design pattern.
[10 points] The code follows the Swift guidelines in the “Swift Guidelines” section.
[10 points] Creative portion: Add one other small feature. Be creative!

Extra Credit

[5 points] The background colors right now are pretty bright. Make the app usable in dark mode by creating custom color sets with a light and dark option for each color. Also make sure the rest of the interface looks good. You may NOT use system colors for the colored backgrounds (make a custom color set), but system colors are fine everywhere else. There should be nothing that looks weird in either dark or light mode

Helpful Advice and Code Snippets

Images and DisplayView

To help you out, a zip file containing all the pet images is included on the course website. To use these in your project, create new image sets in the blue Assets folder, and drag the images into the slots. Feel free to use your own images instead.

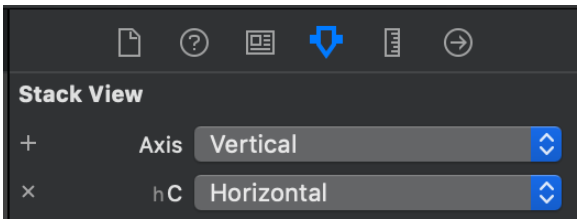
The colored display bar is a custom UIView subclass, and is included on the course website as DisplayView.swift. Simply drag the file into your project to use it! The DisplayView class has three public properties/functions: value, color, and animateValue. There should be no need to modify this file.

Layout

The main goal of this lab is to demonstrate proper UI design with storyboard and Auto Layout constraints instead of adding elements programmatically. Points will be docked for objects added using code statements.

In order to create a layout that works on any screen size, use Auto Layout in the storyboard (constraints, stack views, etc) in order to make the layout dynamic. Xcode has many great tools to test your layout including the storyboard sizes and multiple simulator options. Most of the time you won't even have to run the app to see your layout — just choose one of the devices from the menu at the bottom of the storyboard.

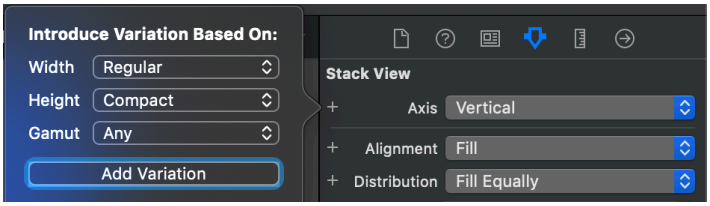
The app should be Universal (meaning that it works on both iPhones and iPads). The layout is optimized for smaller devices, but it should still work on a large iPad.



hC designates the special variation when the height is compact (such as when a device is rotated to landscape mode). In this case the Stack View's Axis is changed to Horizontal.

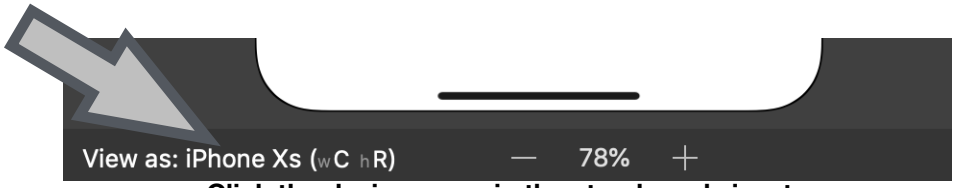
The layout should match the screenshots at the end of the document. The alignment, relative positioning between elements, and ratios should be the same, but the precise spacing and color values are up to you. It doesn't matter if the spacing between the text and display view is 5pt or 7pt, or if the gray is rgb(120,120,120) or rgb(130,130,130).

One convenient (and slightly hidden) feature is the ability to introduce variation to a property based on the size class of the device. To introduce variation, use the small "+" button next to a property in the storyboard. In the screenshot below, variation is added to a stack view's axis when the height is "compact".

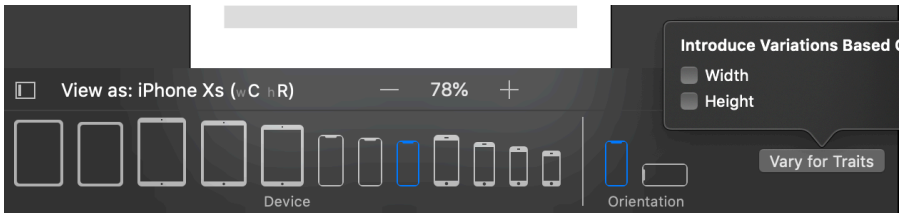


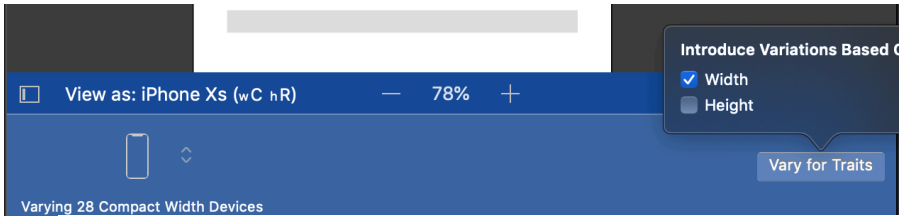
Adding a variation for compact height

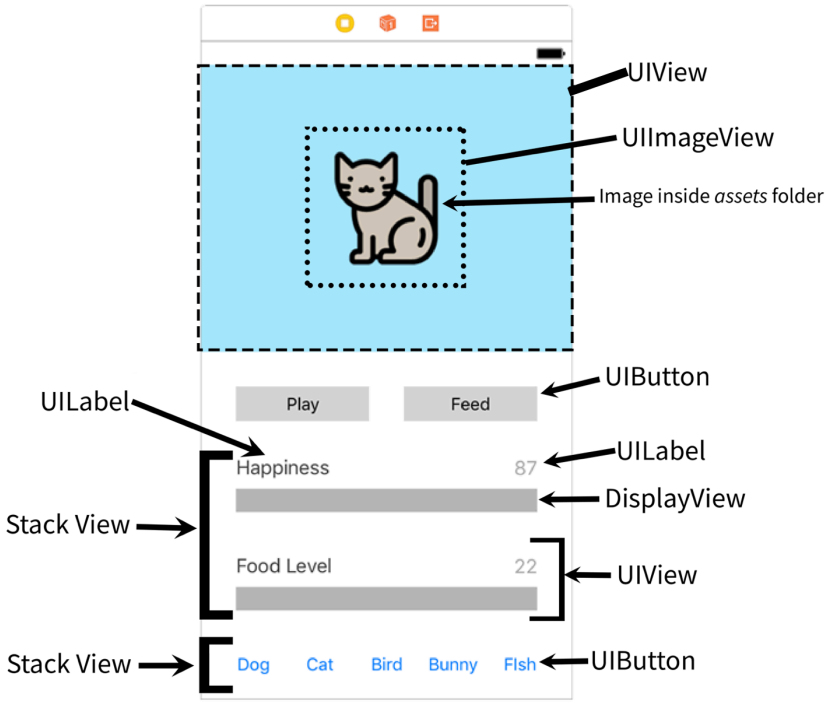
You can achieve the same effect with the “Vary for Traits” button in the bottom bar of the storyboard. When the bar is blue, any changes to the storyboard will only be applied to the devices listed.



Click the device name in the storyboard view to open up the dialog if you don’t see “Vary For Traits”







One of the most important layout challenges is deciding which UIKit objects to use for each element. There are many ways to achieve the same layouts, and each strategy has subtle benefits and drawbacks. You will get a feel for these as you build more apps, but for now here is an example of the elements to use to build the app. A labelled example layout is below:

Because the layout is quite complex, screenshots of various sizes and orientations are provided at the end of this document. If the layout is giving you trouble, don't give up! Take everything one step at a time and you'll get there eventually.

App Functionality

The user should be able to select a pet from a list and see information about that pet's current happiness and food level in addition to the total number of times fed and played with. When the user selects a new pet, that new pet's information should be shown. If they go back to a previous pet, the values should have been saved from before (values do not need to be saved between app launches however).

A user can interact with a pet in two ways: by playing and feeding. A pet can only be played with if its food level is above zero. If a pet is played with, its happiness increases by 1, and its food level decreases by 1. If a pet is fed, its current food level increases by 1. Display the data in the bars such that the respective bar is full if either the current happiness or current food level is **10**. The total number of times fed and played should be listed above the bar in labels. Those numbers should never decrease as they represent the total number of times for each pet, thus they can go higher than 10.

Here are two generalized usage cases with pseudo-code to better explain the desired behavior:

- User taps the **Play** button
    1. Check if the **food level** is above 0
    2. If it is above 0:
        1. Increment **happiness** variable and update displayView to match
        2. Increment **number of times played with** variable and update label to match
        3. Decrement **food level** variable and update displayView to match

- User taps the **Feed** button
    1. Check if **food level** is above 10
    2. If it is below 10:
        1. Increment **food level** variable and update displayView to match
        2. Increment **number of times fed** variable and update label to match

- User taps the **bird** button
    1. Check if **current pet** is not the bird
    2. If it is not the bird:
        1. Update pet image to be the **bird's image**
        2. Update the background and displayView colors
        3. Update the **happiness** displayView to display the **bird's happiness**
        4. Update the **number of times played with** label to display how many times the bird has been played with
        5. Update the **food level** displayView to display **bird's food level**
        6. Update the **number of times fed** label to display how many times the bird has been fed

Model View Controller (MVC)

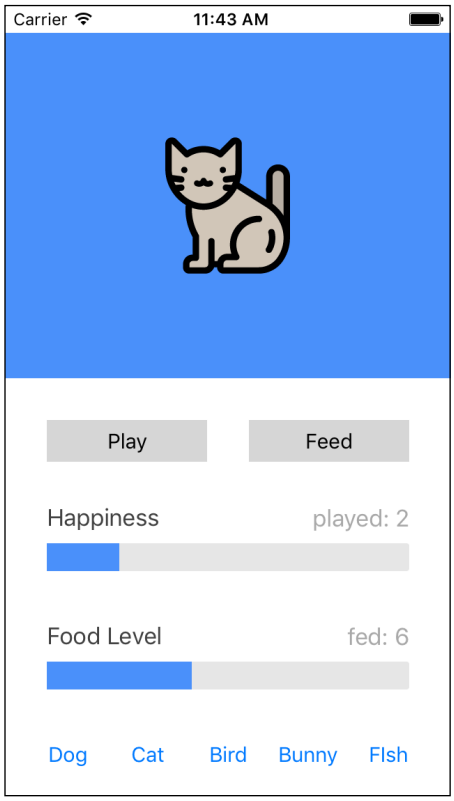
MVC is an object-oriented programming paradigm that separates the concerns of your code. There should be clear **model** objects with data-related abilities, **view** objects which exclusively display data, and **controller** objects, which operate between the model and the view. In the case of this lab, you should create a model object to represent a pet, and have this object handle all calculations. The view should be handled via Storyboard, and the controller via your UIViewController subclass. Think about the interaction between your objects and make each object's role clear.

Swift Guidelines

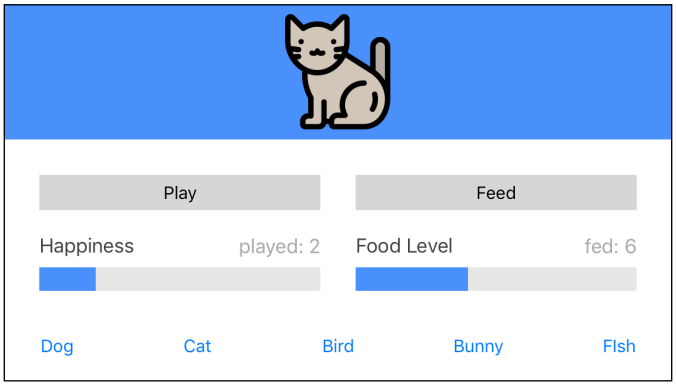
Swift is still an evolving language, and the true "best practices" haven't been cemented yet. Despite this, there are many basic conventions that will make your code easier to read. Your code should adhere to the following guidelines:

- Types begin with a capital letter. (ex: struct String)Properties and functions begin with a lowercase letter and use camel case. (ex: let myString = …, func sayHello() { …)
- Optionals are safely unwrapped. (using "if let", "guard", or "??")
- Compiler gives no warnings (yes, even storyboard warnings).

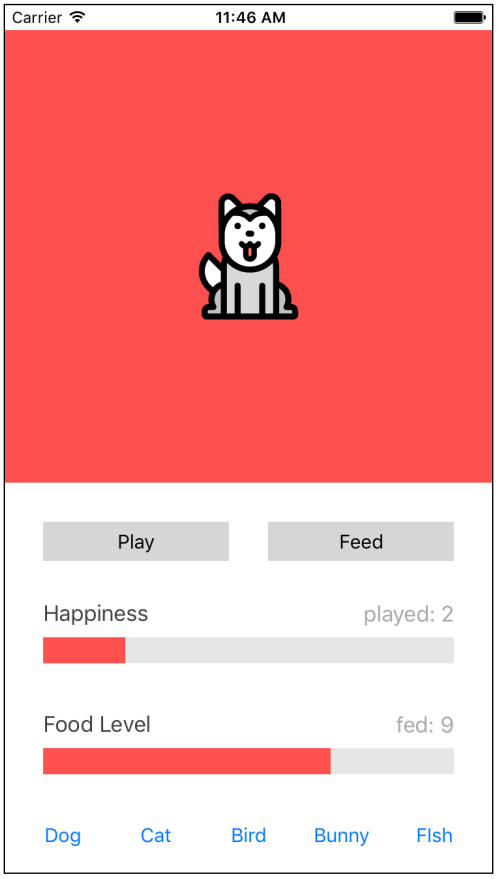
Example layout screenshots:

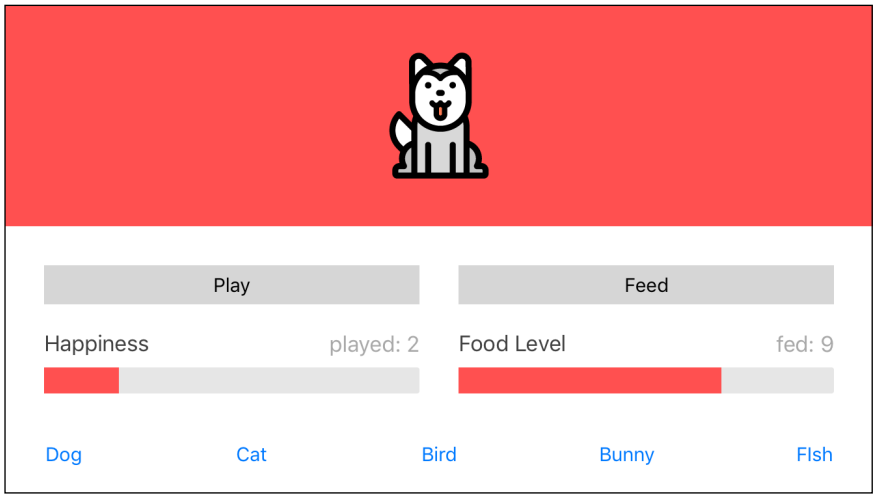


iPhone SE Portrait

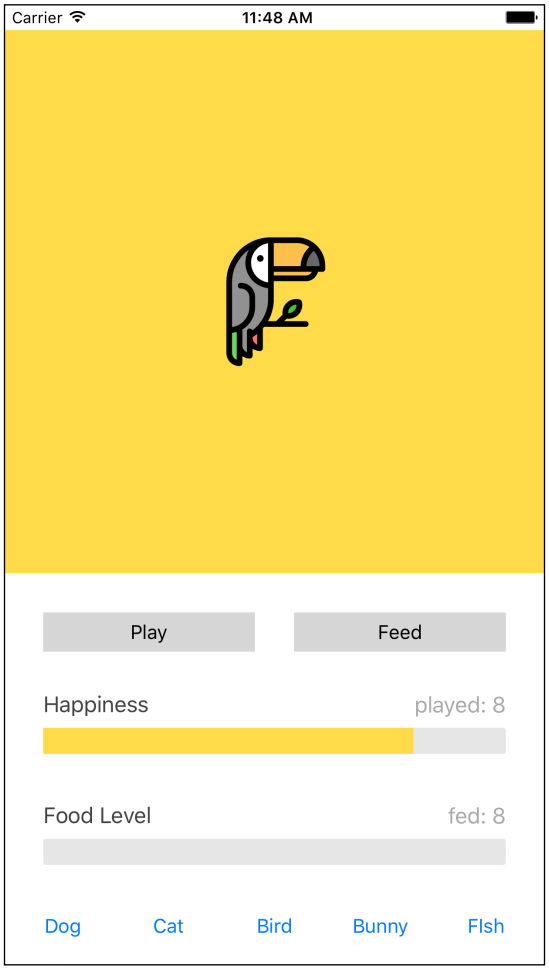


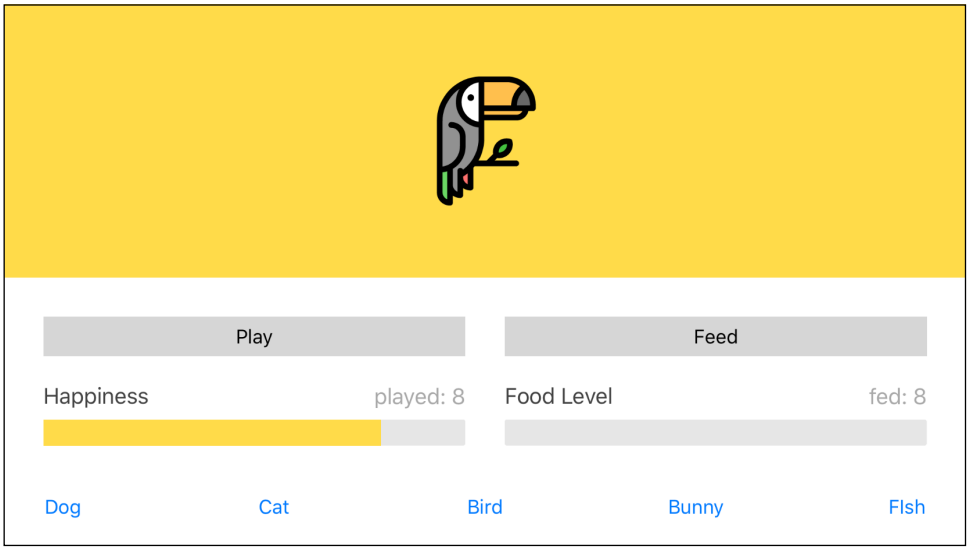
iPhone SE Landscape



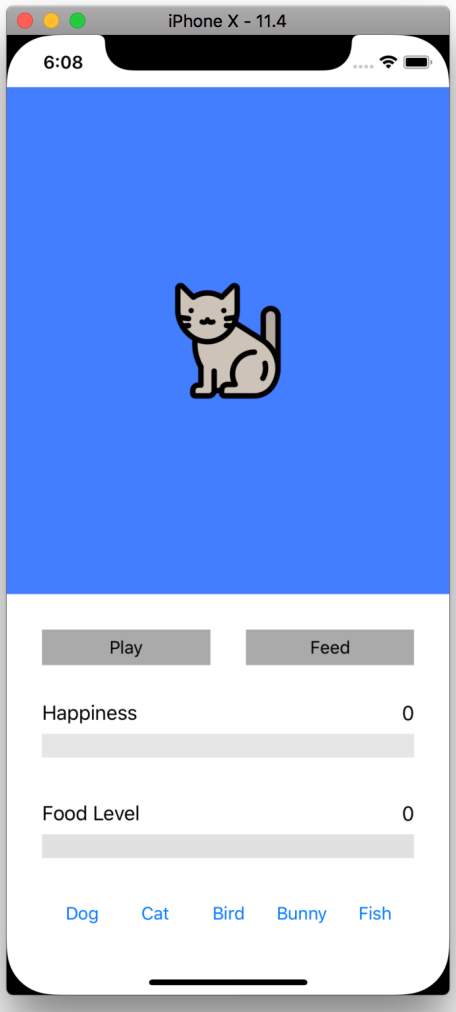


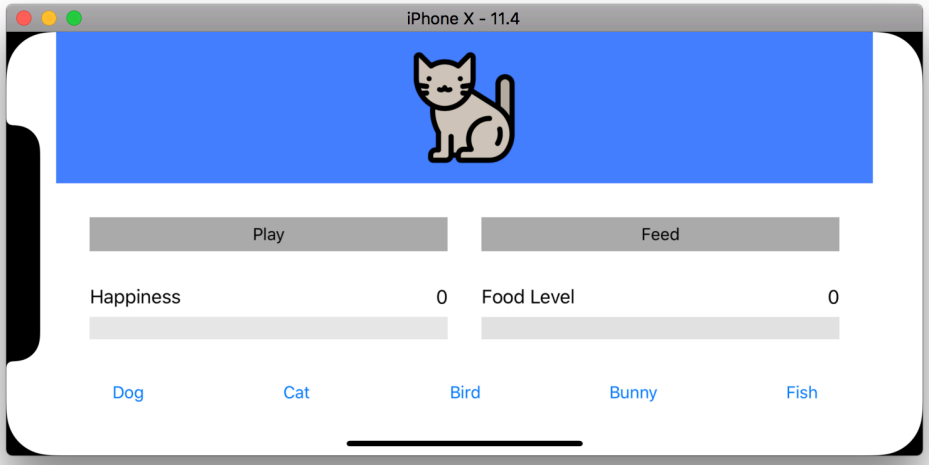
iPhone 8 Portrait & Landscape



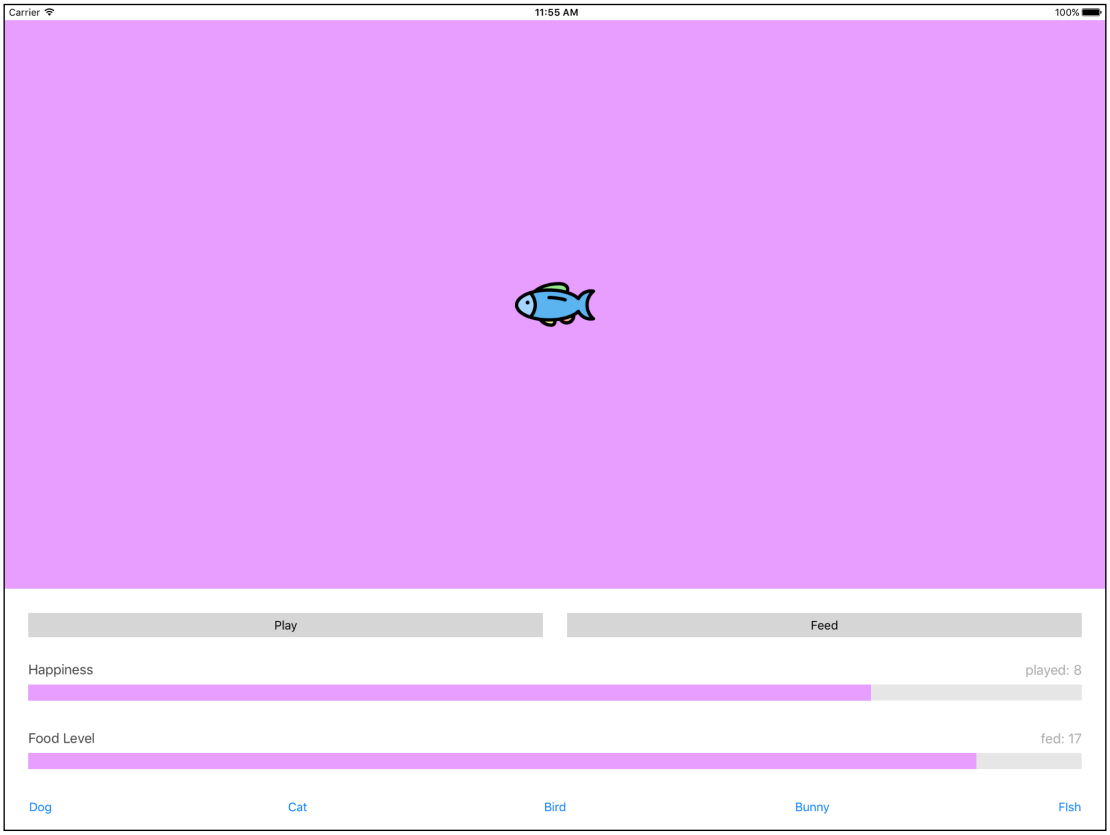


iPhone 8 Plus Portrait & Landscape

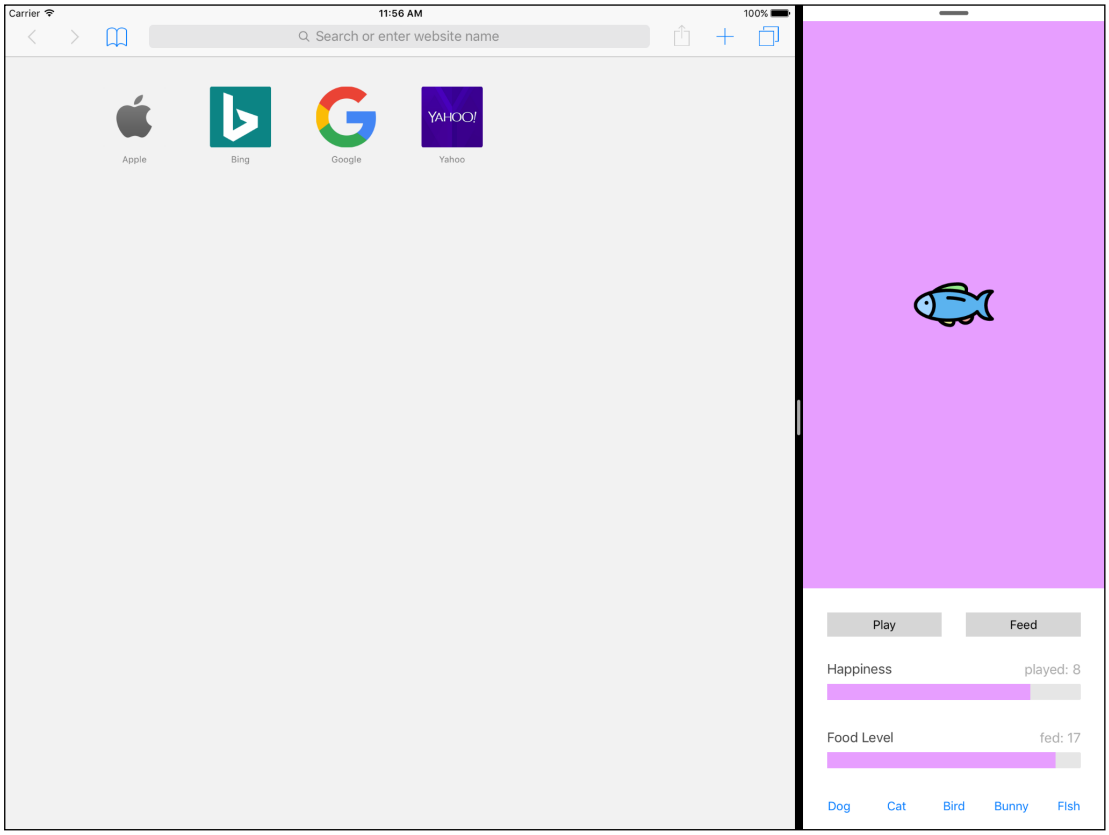




iPhone X Portrait & Landscape



iPad Pro (12.9 in) Landscape



iPad Pro (12.9 in) Landscape Split View