LambdaSchool / **Contained**

No description, website, or topics provided.

1 commit | **1** branch | **0** releases | **1** contributor

Branch: **master** | **New pull request** | **Create new file** | **Upload files** | **Find file** | **Clone or download**

erica first commit — Latest commit `b171cae` 6 days ago

Art	first commit	6 days ago
Crab.xcassets	first commit	6 days ago
Images	first commit	6 days ago
README.md	first commit	6 days ago
SKSpriteNode+Utility.swift	first commit	6 days ago

README.md

Module Challenge: View Controller Containment - Contained

This challenge allows you to practice the concepts and techniques learned in today's guided lesson and apply them in a concrete project. Your lesson explored view controller containment using Navigation and Tab Bar controllers. You will demonstrate proficiency by creating an application that showcases the same features you learned in class.

Instructions

Read these instructions carefully. Understand exactly what is expected *before* starting this Challenge.

This is an individual assessment but you are permitted to consult with and support other members of your cohort. You are encouraged to follow the twenty-minute rule and seek support from your PM and Instructor in your cohort help channel on Slack.

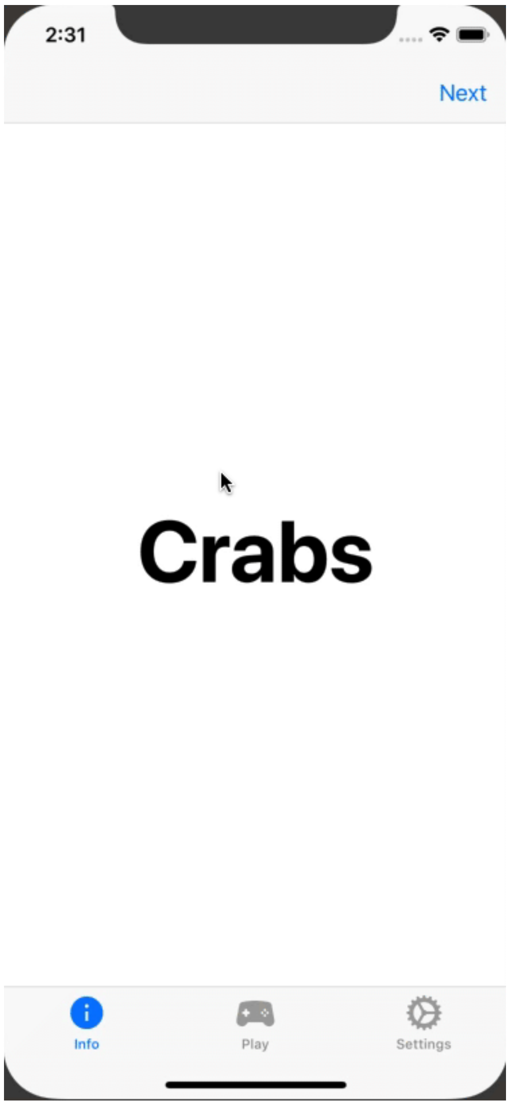
Commits

Commit your code regularly and meaningfully. This helps both you (in case you ever need to return to old code for any number of reasons) and your project manager.

Description

In this challenge, you build a multi-tabbed SpriteKit game.

Your completed application will look like the following GIF:



Questions

Demonstrate your understanding of today's concepts by answering the following free-form questions. Submit them as a text file along with your project.

- What is controller containment and what does it offer developers?
- What is the difference between navigation and tab controllers? Under what circumstances do you use each one?
- Name at least one Apple-supplied iOS application that uses each container class and explain how they're used in each app.

Project Setup

Follow these steps to set up your project:

- Create a new Single View application. Clean up the boilerplate in the application delegate.
- Set the project to be portrait only (or landscape only; either works).
- Delete the ViewController.swift class and delete the ViewController instance from your Main.storyboard
- Add the tab bar icons in the Art folder to your main asset catalog.
- Add the Crab asset catalog to your project.
- Add the SKSpriteNode+Utility.swift file to your project.

Building your Model

- Add a new Model.swift file to your project with the following contents:

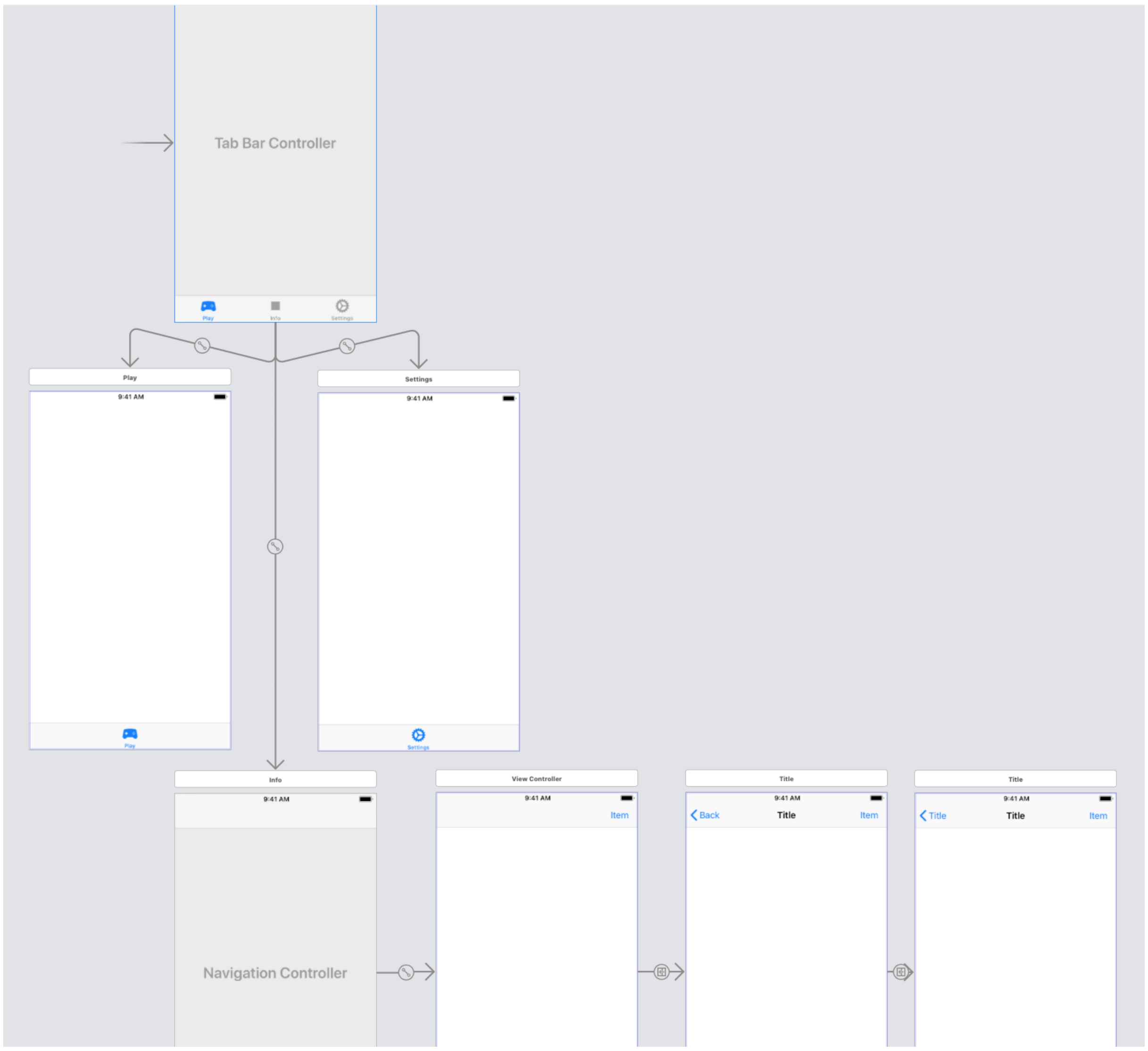
```
import Foundation

class Model {
    static let shared = Model()
    private init() {}

    var shouldRoll = false
    var shouldZoom = false

}
```

Building your Interface

Follow these steps in IB to lay out your interface

- Add three view controllers and embed all three into a tab bar controller. Ensure the tab bar controller is the storyboard entry point.
- Set up your tabs: Info, Play, and Settings along with the appropriate icons for each.
- Select your Info view controller in IB and embed it into a navigation controller.
- Add two more view controllers to IB. Add Navigation Items to each one, and then add bar button items to the top right of the info view controller.
- Connect the info view controller to the first of the two new controllers. Add a bar button to the top right. Connect that to the final view controller. Add a final bar button to the top right.
- Create three classes in three new files: GameViewController, InfoViewController, and SettingsViewController. Each descends from UIViewController but is otherwise empty. Assign these classes to your view controllers using the IB identity inspector.

Your interface should now look like this:



You're ready to start adding content to your interface.

Build the Info Sequence

Follow these steps to create content for your information tab.

- Name the first two bar button items "Next". Set the last bar button item to the system "Done" item using the attributes inspector.
- Add an IBAction from the Done button to the InfoViewController class and have it pop back to the root view controller:

```
@IBAction func done(_ sender: Any) {
    navigationController?.popToRootViewController(animated: true)
}
```

- Add a label to your first Info view controller. Set its font to System Bold 48. Center align the text in the label. Edit the text to Crabs.
- Center it horizontally and vertically and add two constraints
- Add a label to your second Info view controller. Set its font to System 24. Set the number of lines to 2 Center align the text in the label. Edit the text to "Tap to move the crab". Use option-return to add a new line between "move" and "the crab".
- Center it horizontally and vertically and add two constraints

- Copy the label from your second Info view controller to your third.
- Center it horizontally and vertically and add two constraints
- Edit the text to say "Let's play!"
- Edit the two titles on the second and third info controllers to remove the "Title" text.

Build the Settings Screen

Follow these steps to create content for your settings tab.

- Drag two labels and two switches onto the screen
- Set both switches from "on" to "off"
- Rename the labels to Roll and Zoom
- Embed one label and one switch into a horizontal stack view. Repeat for the other. Set both stack view spaces to 8
- Embed the two new stack views into a vertical stack view. Set the stack view space to 8.
- Make sure that the alignment and distribution for all three stack views are set to Fill.
- Center the stack view horizontally and vertically and add two constraints.
- Connect the switches to actions in SettingsViewController: `toggleRoll` and `toggleZoom` . Be sure to use a `UISwitch` type when connecting for the "sender". (Hand edit the functions if you get this wrong.)
- Have each method set the corresponding shared model property to the sender's `isOn` property.

Build the Game Screen

- Add an Sprite Kit View to the Game view controller. Use the tie fighter button to stretch the view to each side without spaces.
- Connect the view to an `skview` property in your Games view controller and import SpriteKit to get rid of the error message.
- Add the following code to your GamesViewController.swift file. You're about to create the CustomScene file so ignore the error for now.

```
class GamesViewController: UIViewController {
    @IBOutlet weak var skview: SKView!

    var skscene: CustomScene? = nil

    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)
        skscene = CustomScene(size: view.bounds.size)
        skview.presentScene(skscene)
    }
}
```

- Create a new CustomScene.swift file. This is the same project we built in class on Monday.

```
import SpriteKit

class CustomScene: SKScene {
    let crab = SKSpriteNode()

    // Add and center child, initializing animation sequence
    override func sceneDidLoad() {
        super.sceneDidLoad()
        addChild(crab)
        crab.loadTextures(named: "HappyCrab", forKey: SKSpriteNode.textureKey)
        crab.position = CGPoint(x: frame.midX, y: frame.midY)
    }

    // Move to touch
    public override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {

        // Fetch a touch or leave
        guard !touches.isEmpty, let touch = touches.first else { return }

        // Retrieve position
        let position = touch.location(in: self)

        // Create move action
        let actionDuration = 1.0
        let moveAction = SKAction.move(to: position, duration: actionDuration)

        let rollAction = SKAction.rotate(byAngle: CGFloat.pi * 2, duration: actionDuration)
        let zoomAction = SKAction.scale(by: 1.3, duration: 0.3)
        let unzoomAction = SKAction.scale(to: 1.0, duration: 0.1)

        switch Model.shared.shouldZoom {
        case false:
            crab.run(moveAction)
        case true:
            let sequenceAction = SKAction.sequence([zoomAction, moveAction, unzoomAction])
            crab.run(sequenceAction)
        }

        if Model.shared.shouldRoll {
            crab.run(rollAction)
        }
    }
}
```

Minimum Viable Product

Your finished project must include all of the following requirements:

- Three tabs that lead to fully working program elements.
- Clean layout that mimics the sample screenshots above.
- Well organized maintainable layout in Interface Builder.

Stretch Problems

After finishing your required elements, push your work further. These goals may or may not be things you learned in this module but they build on the material you just studied. Time allowing, stretch your limits and see if you can deliver on the following optional goals:

- The Crab reverts to a center position each time the user leaves from the game tab and returns. Expand the model to store the most recent touch point and use that value in `sceneDidLoad` .
- Add one or two more options, such as fade effects (a node's `alpha` property) or other features you can read about in the SpriteKit documentation.
- Allow the user to switch between the HappyCrab and the WaitingCrab textures via the settings screen.

Progress Dashboard

Download checked items