RandomsChoice

Turning Dreams into Reality

ROLL A VIRTUAL DIE FOR FUN!

Table of Contents

Intro (pg. 2)

1. The Story
    a. The Dream (pg. 3)
    b. The Idea (pg. 4)
    c. The Learning (pg. 5)
    d. The Programming (pg. 6)
    e. Simple! (pg. 7)
    f. The Timeline (pg. 8-11)
    g. Submit (pg. 12)
    h. User Feedback (pg. 13)
    i. The End? (pg. 14)
2. How Did I Make It?
    a. Jumping in... (pg. 15)
    b. Variables (pg. 16 & 17)
    c. Text & Previews (pg. 18)
    d. Images & Tapping (pg. 19)
    e. Stacks (pg. 20)
    f. The Structure (pg. 21)
    g. Adding Style (pg. 22)
    h. Light & Dark Mode (pg. 23)
    i. Snipping Code & Animations (pg. 24)

Updates & The Future (pg. 25)

The Code (pg. 26-31)

RandomsChoice

Roll a virtual die for fun!

Now available for Apple download.

What is RandomsChoice?

Rolling a die off a table can be frustrating, but RandomsChoice allows families to play games without rolling the die onto the floor!

Introducing an app that the whole family can enjoy! With this app, you can easily generate a random number by rolling a virtual die on your screen. Say goodbye to dice rolling off the table; now you can roll a die even during long car rides! Use the app for any situation that requires a random selection. Download RandomsChoice today to help you make random decisions for your next family game!

To use the app, simply tap the die to roll it as many times as you like.

- Enjoy a fantastic rolling animation
- Roll random numbers from 1 to 6
- Supports both dark and light modes
- Accessible offline
- Contains no ads or in-app purchases

The Story

Every journey starts with a dream.

I enjoy programming various projects. So far, I have worked with the following programming languages:

- Python
- Java
- Swift
- HTML, JavaScript, and CSS

I believe the most rewarding aspect of developing something is sharing it with others so they can enjoy it.

Wouldn't it be awesome to make my very own app that anyone can download?

The simple Flappy Bird game reached over 50 million downloads!

I showcase my projects on GitHub and keep them public. Often, I see apps and think I could create something similar or at least learn how to do it.

Before the developer of the iOS app "Flappy Bird" removed it due to its addictive nature, the app reached millions of users. If such a simple app could achieve that level of success, maybe I can create apps that people will enjoy. Simplicity is what people are looking for, after all.

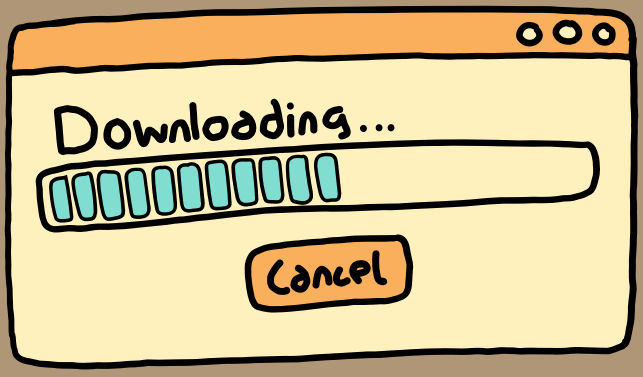
I can!

It started with a dream, but I came to a point when a light bulb came on. I decided that I wanted this to be more than just a dream.

The Before Process

I have an Apple phone, but I couldn't program apps for it because I didn't own an Apple computer. Instead, I decided to use Android Studio to create a program for the Android tablet that our family has. This allowed me to jump into this project as soon as possible. I developed a math app and installed it on my younger sister's tablet.

For Christmas, my dad bought me a mini Mac computer. After that, I went straight away and downloaded Xcode.



Where To Start?

Clearly, I couldn't just dive in and start developing an app. I had a vision, but I lacked the knowledge to execute it.

Step 1: Install Xcode

I learned that Xcode is Apple's official IDE (Integrated Development Environment for programming iOS apps using Swift.

I had to install Xcode before I could program with it.

Step 2: Watch and Follow

Books are nice, but the best way I learn is by doing. To learn how to program using Xcode, I would watch YouTube videos. My dad also bought me a subscription to LinkedIn Learning, so I could access all the Swift programming videos on the site.

Step 3: Make Things

I had fun playing around with what I learned and making calculator programs and running them with the Xcode simulator and downloading them onto my phone!

Later, my dad set me up with a developer account so I could eventually submit apps.

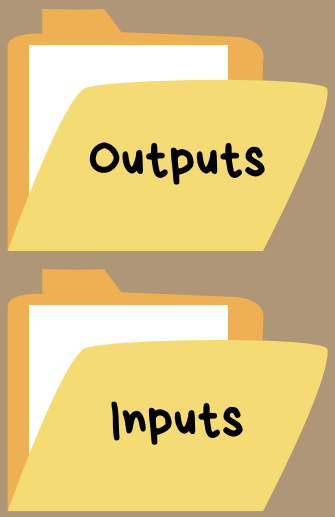
Programming VS Coding

There came a point in time when I was ready to start programming. Before we dive into that, what is programming? Programming and Coding are used interchangeably, but are they the same thing? The answer is no.

So, what's the difference?

Coding is a subset of programming, focusing on the language, syntax, and writing instructions for computers. In contrast, programming encompasses a broader scope, including documentation, review, and analysis.

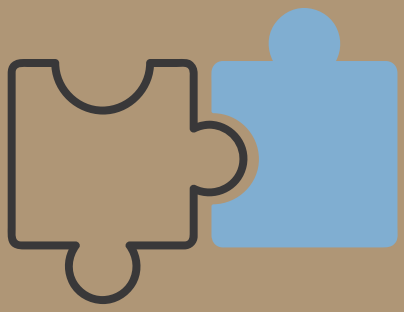
Commenting and organizing your code is important when programming. When I program, I organize things into folders.



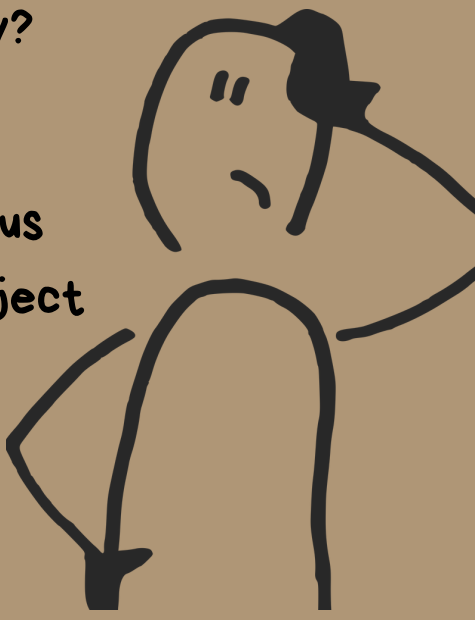
What do I make?

You may be familiar with the acronym KISS, which means Keep It Simple.

I had one of those moments where I told myself. Is there a simpler way?



After watching numerous tutorials, I began a project that I named FullCount.



Why FullCount?

My brother plays baseball, so I thought it would be a great idea to create an app that keeps track of the game's score. I knew our family would enjoy it!

The app was simple, allowing users to keep score, track outs, and count strikes and balls. It was effective, but I wanted to do more.

CountFusion

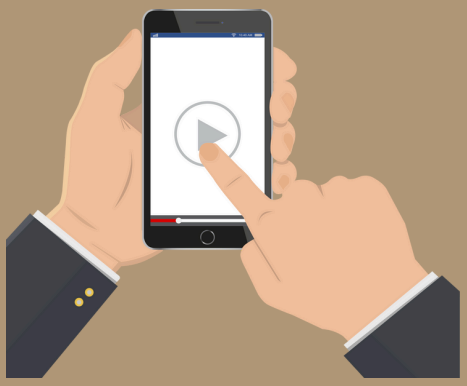
I wanted to do too much. I kept expanding the app and eventually decided to rename it CountFusion instead of FullCount, as it became a counter for various activities, not just baseball. However, this expansion introduced more bugs into the code.

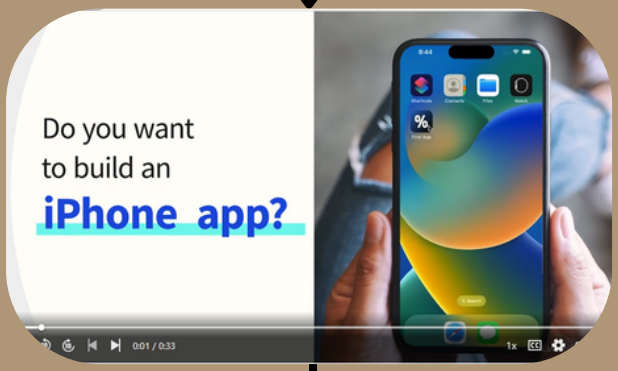
Finally Set

I finally decided to create a die-rolling app for my first submission, but I may revisit the ideas for FullCount or CountFusion later.

Tutorials

LinkedIn and Youtube

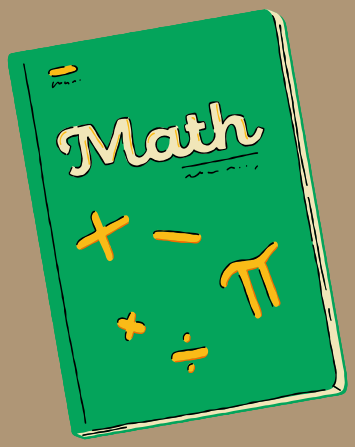


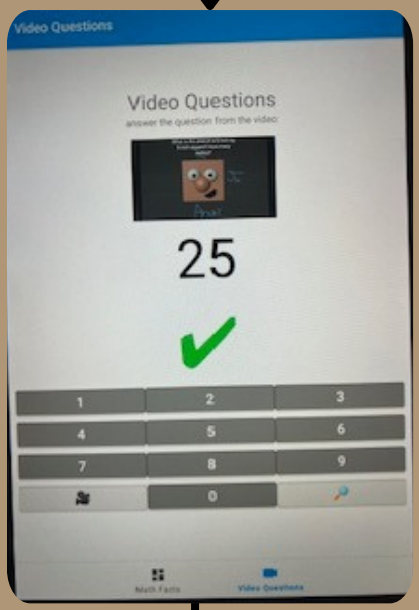


I watched tutorials to learn how to program in various languages and develop apps.

First App

Android Math App

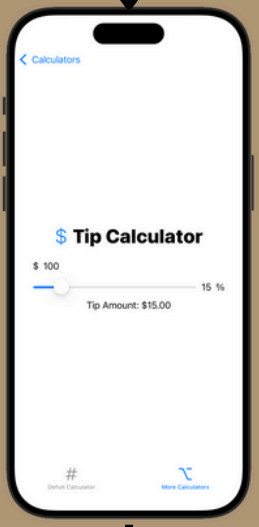




The first app I created was called "Math Matter." It is an Android app designed for my younger sister, aimed at helping her master math facts. The app provides practice problems and engaging video-based story problems to enhance her learning experience.

Tip Calculator

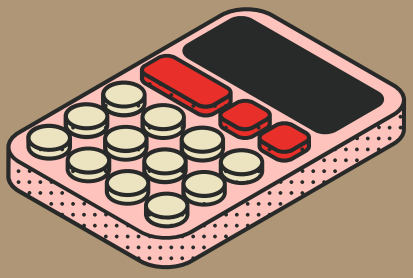
Following Tutorials

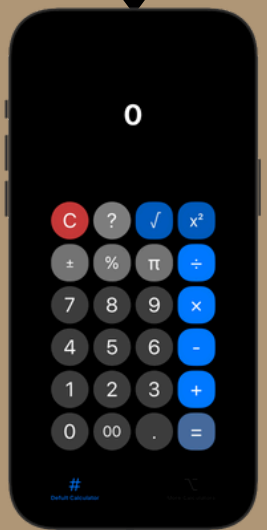


After a LinkedIn tutorial on SwiftUI, I created a tip calculator app with a sliding bar, but it was never released to the App Store.

Computation

All in One Calculator

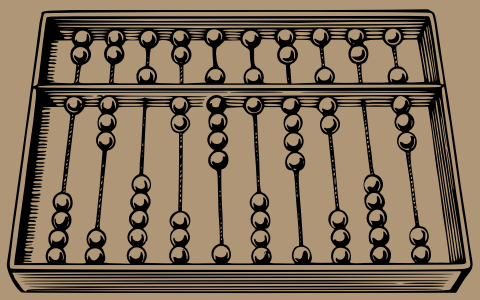


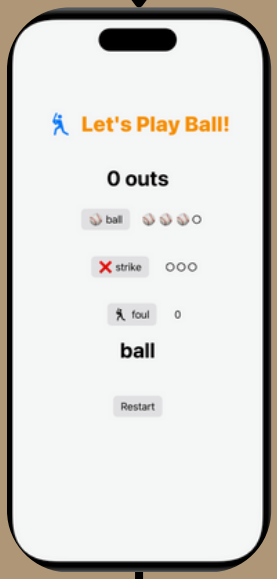


I explored advanced topics and created my calculator for Apple, called Computation. It was a fun project, but it never made it to the App Store.

FullCount

A Baseball Scorer

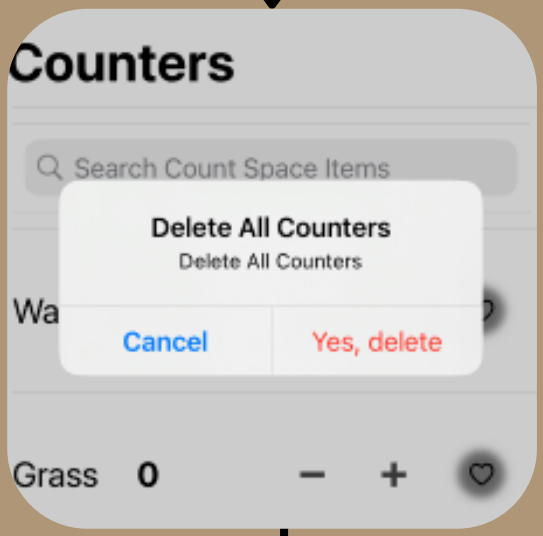




I decided to create my app without any tutorials and developed a fun baseball scoring app for my brother, who plays the sport. I named it FullCount.

CountFusion

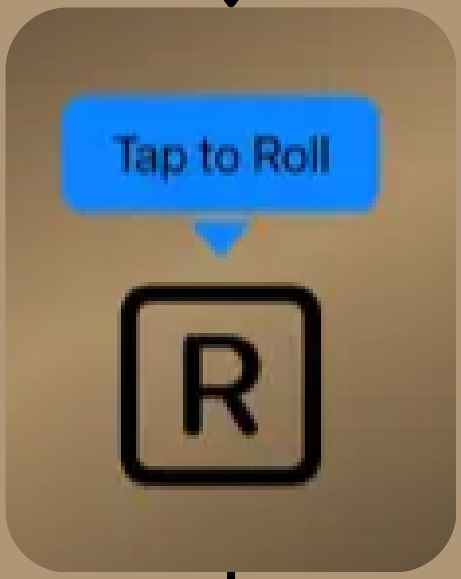
All in one Counter



I intended to release FullCount, but as I added features, it formed into CountFusion, a multi-use counter. It became too complex for a first release, teaching me to introduce features gradually.

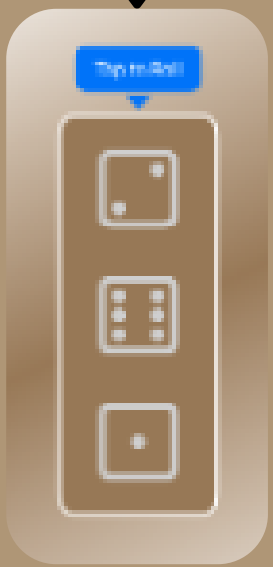
RandomsChoice

Roll a virtual die



Future

RandomsChoice Updates

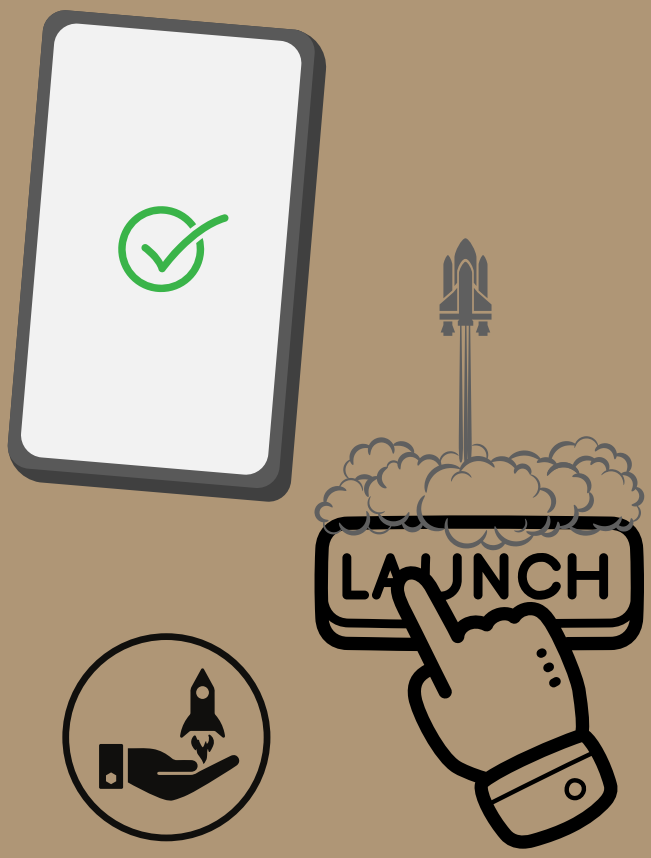


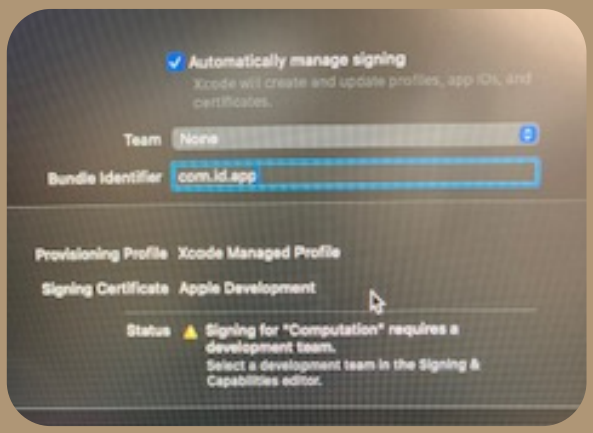
I aimed to create a simple app and developed a digital die called RandomsChoice. Initially named RandomChoice, I changed it due to availability. While unique, the added "s" made it harder to find in searches. The app has room for improvement and updates.

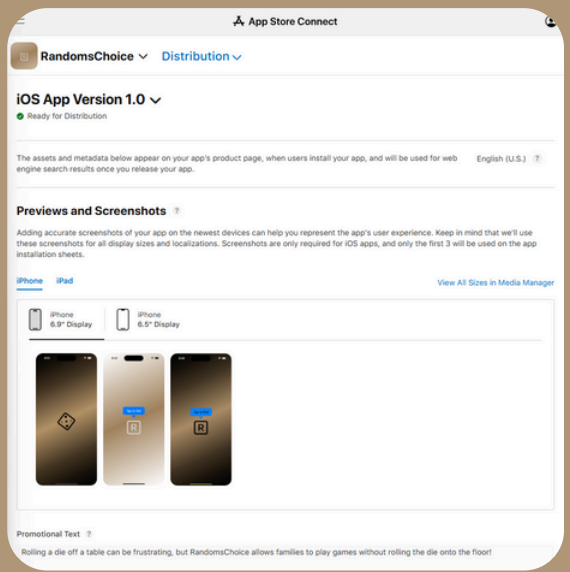
I plan to update RandomsChoice with multiple dice and sound for a better experience, adding features gradually to avoid overcomplication. I also aim to create more apps for both Apple and Android.

I Am Ready

I decided the app was ready for submission and researched the process. My dad set up an Apple developer account, and since I'm under 18, he will "sell" the free app until I turn 18. I hold the copyrights since I created it.







I learned to submit an app by entering my bundle identifier in Xcode and clicking "Archive" to upload it. I then filled out the app description, pricing, and media for the App Store, and created a privacy policy. Finally, I submitted the app to Apple for review. The process is straightforward, though it can be lengthy.

What are People Saying?

I aim to improve my app marketing as I release updates and new apps. I've received user feedback, with many requesting enough dice for Yahtzee and sound effects. One user felt it lacked randomness, but I explained that adding more dice would enhance that experience.




Ratings and Reviews

5.0 out of 5 — 5 Ratings

★★★★★

A 5-yo named Refrigerator, 04/21/2025

Great app!

I love the animations. It is very useful when you don't have a die. I sometimes use it to decide what kind of homework I want to do first, which expands its use beyond board games

★★★★★

HoneyBear1977, 03/31/2025

Really useful

Really useful for when you lose a die or have some crazy rollers. Vey simple. Thank you so much to whoever made this.

While my app isn't very popular yet, over 30 downloads is still a positive achievement despite limited ratings and reviews.

Is This The End?

Of course not! The app journey doesn't end here. I submitted my first app to Apple, but that's just the beginning. I have big plans for the future!

I plan to update the RandomsChoice app and develop more apps for a variety of devices, including both Apple and Android.

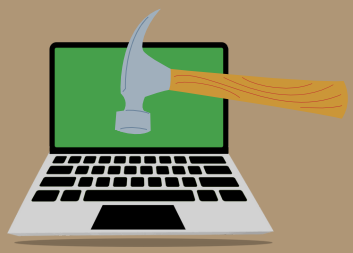
This is the essence of computer science: to dream today and act tomorrow, continually improving and making the best even better at every step. The science of computer science.

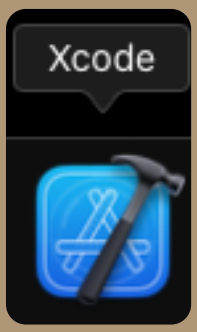
How Did I Make It?

What are you waiting for? Download Now!

How to Install Xcode

Earlier, I discussed how to install Xcode, now, I'll explain more in-depth how that was done.





First, Xcode is found in the Mac App Store. To put it on my Mac computer, I downloaded it there.

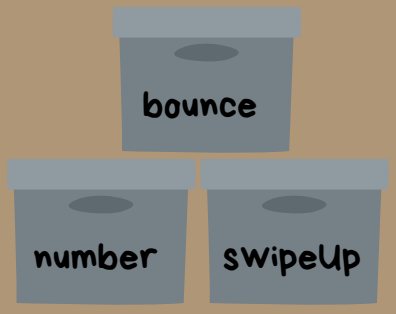
After opening Xcode, it gives the option to create a new project, clone a project from GitHub, which I have also done before. The third option is the classic open button, which allows me to open a project from my computer.

How does it work?

Using the programming language Swift and an importation called SwiftUI, I can program using variables, functions, structures, and organize my code using programming practices. Let's talk about those practices and how my code comes together.

What is a variable?

A variable stores information that can later be used in the program.



Each part of the program works as one whole. There isn't one file doing all the work, instead there are multiple structures that communicate with each other to get the job done.

```
bounce = true
```

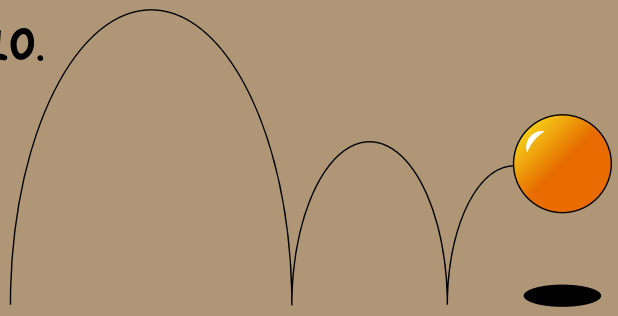
Here is an example of a variable being used in my app. I am setting bounce to true. That can later be used to tell the animation to bounce.

Doing things with Variables

After a variable is stored, it is used in some way. Below is an example of me using the bounce variable to offset the instruction text up and down.

```
.offset(y: bounce ? -10 : 10)
```

If bounce is true, then the text offsets to -10; if the bounce is false, it offsets to 10.



```
if !hideInstructions { //
    NOT hidden
    Text("Tap to Roll")
        .font(.title2)
```

The simplest form of a conditional statement is an if statement. If statements are what check to see if a condition is true or false. In the example above, the statement checks to see if the hideInstruction boolean is NOT true. The "!" symbol means not in programming.

Did you know?

The variable that contains a true or false condition is called a Boolean variable. A boolean can only hold a value of true, false, or nothing. Booleans are similar to binary.

Variables are an essential part of programming. Another example of a variable in my app is using one for the color scheme.

```
.foregroundColor(colorScheme == .light ? .accentColor :
    .black)
```

```
Text("Tap to Roll")
```

Text() is one of the simplest SwiftUI commands. In the code above, "Tap to Roll" is displayed on the app as text.

```
Text("Hamlet")
    .font(.title)
```

Displaying a Preview

The preview is what displays the content of the app. This way, I can review my changes without compiling the application.

```
#Preview {
    ContentView()
}
```

I realize I never mentioned why I used SwiftUI instead of Storyboards. I like SwiftUI for the following reasons:

Displaying Text

What would an app be without words to direct the users and explain what the app does?

Did you know that you can change the font of the Text just by adding a .font to the bottom as shown on the left photo?

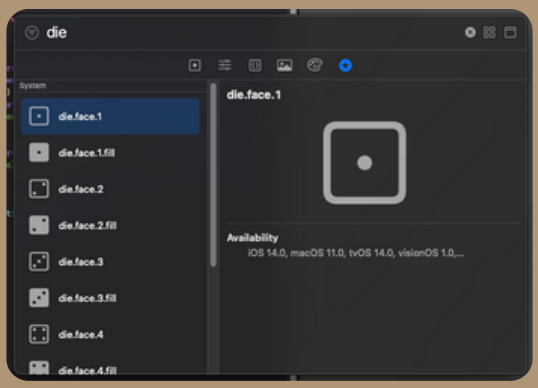
The ability to view a preview on Xcode is one of the main reasons why I decided to use SwiftUI instead of Swift Storyboards.

Why SwiftUI?

- SwiftUI's preview feature is awesome!
- It is simple and easy to pick up.
- I'm better at designing using code, not the storyboard interface.

Images

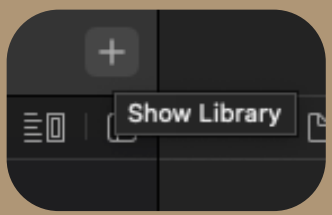
If all an app had as an output was text, the app would be boring. Images make the visual aspects of the app come alive!



Displaying an image on the app using SwiftUI is almost as easy as displaying text. To display an image, you simply type: Image(). In the parentheses is where I put the image path or an Xcode system image.

Xcode's Image Library

Xcode has an image library that contains a list of Apple's system images that developers can use for their apps. My app uses the die images for my app. This way, I can focus more on the creation of the app rather than the images.



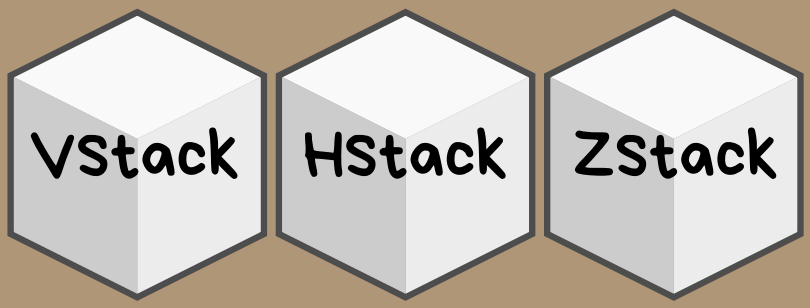
Eventually I plan to use my own images for future updates instead of the Xcode library.

The tap gesture

Buttons are sometimes necessary in an app, but other times, tapping an image works well. In my die app, a tap increments the counter and triggers a rolling animation. I also plan to add a feature that lets users shake their phone to roll the die.

Stacks

When programming in SwiftUI, 3 crucial User Interface organization techniques are used. These are called stacks. The three types of stacks are the...



In my code, I use a VStack and some ZStacks

VStack

V stands for vertical. The VStack is what stacks all the content of the app within the braces into a vertical stack. I use a VStack in my code, here is what a VStack looks like:

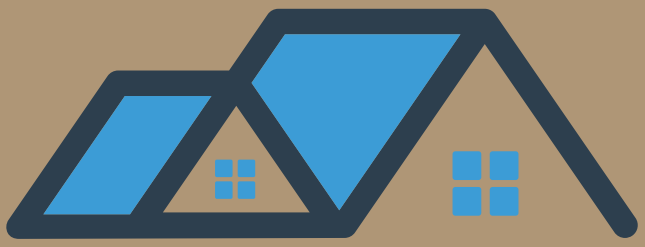
```
VStack {
```

HStack

H stands for horizontal. The HStack is what stacks all the content of the app within the braces into a horizontal stack. I don't use a HStack in my code, although I will in the future updates.

ZStack

In this context, "H" stands for horizontal, "V" stands for vertical, and "Z" refers to the Z-axis on a plane. This indicates that content can be stacked in layers.



The Structure

Every building must have a foundation, every structure must have a frame. Did you know that in SwiftUI, there is a command called the structure?

Below is an example of a structure in my code. Making a structure called "Die" and setting that structure to a view type, I can create a view interface that allows someone to roll the die on the screen, but first, I must make the structure.

```
struct Die: View {
    @Environment(\.colorScheme) var c
    @State var number = 0 // what num
    @EnvironmentObject var rotation:
```

In the code to your right, there is an introduction to SwiftUI. Without this importation, there can be no programming in SwiftUI. So that is important.

```
import SwiftUI

struct BouncingTextView: View {
```

The photo above shows a snippet of my code. You can see another example of the struct command. I made a structure called "bouncingTextView"

Adding Style

As you might expect, using "Text()" by itself or "Image()" by itself is boring. It is bland, there is no style or excitement. How do we make this less boring? How do we add style?

Modifiers

To add more style to the buttons, text, images, or other user elements, I added something called a modifier. A modifier is an element in SwiftUI that adds a new modification to that element. Some examples of modifications are text size, element color, touch effects, and more!

Other modifier examples include:

- font()
- foregroundColor()
- padding()
- background()
- cornerRadius()
- And more!

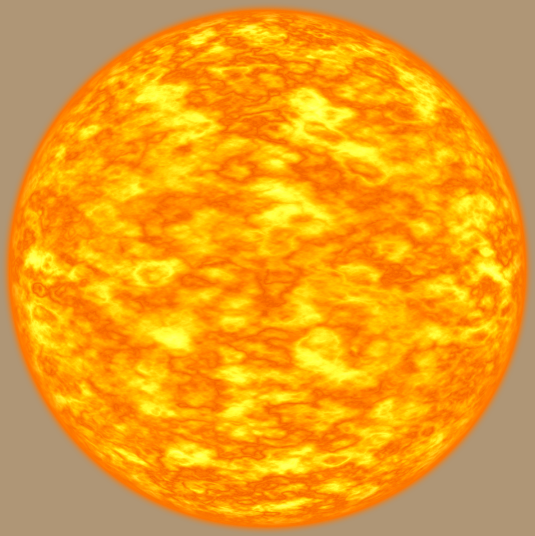
```
.font(.system(size: 100))
```

Above is an example of a font modifier. A font modifier adds a text font to the text element, or in this case, it is assigning the font size of the die image. The system size of 100 is set to how big the image is.

Modifiers can also me custom made, which is something I plan to play around with more in the future.

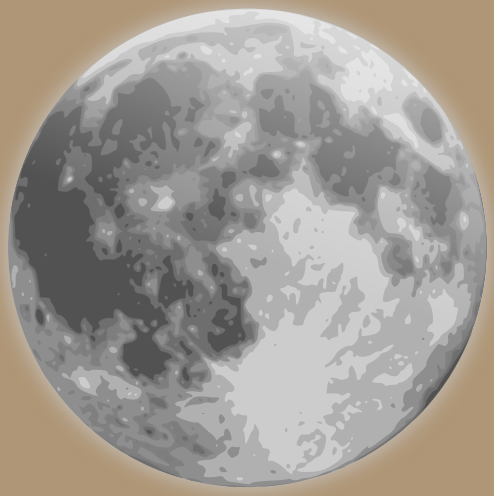
```
Image(systemName: number == 1 ? "die.face.1" : number == 2 ?
    "die.face.2" : number == 3 ? "die.face.3" : number == 4
    ? "die.face.4" : number == 5 ? "die.face.5" : number ==
    6 ? "die.face.6" : "r.square")
    .font(.system(size: 100))
```

Light Mode



The mode that is set to default is called light mode. Often shown with bright lights and white screens.

Dark Mode



Most people choose dark mode over light mode because it is better for the eyes. Dark mode includes dark screens.

Setting the Mode

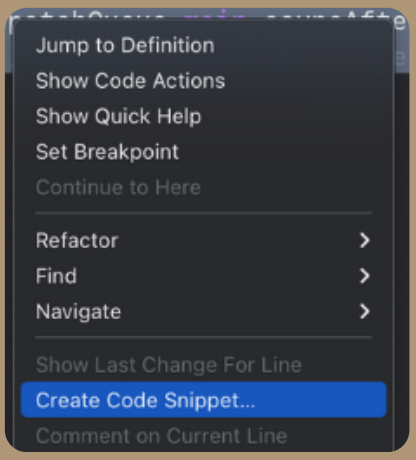
To set the mode to dark or light, I detect the user's settings and store them in an environment variable called "colorScheme". This allows me to check if the phone is in light or dark mode, displaying lighter colors for light mode and darker colors for dark mode.

```
@Environment(\.colorScheme) var colorScheme
```

I enjoy using inputs and outputs in programming and the possibilities they create. For instance, I made an app that displays a moon icon in dark mode and a sun icon in light mode. I can also detect the phone's rotation and track steps!

A Code Snippet

Xcode offers a great functionality that gives developers the ability to reuse their code easily. This is called a Code Snippet.



I saved a sound function in SwiftUI for future projects. Now, when I update RandomsChoice with sound, I can easily implement it because I stored the code snippet under the keyword "sound," allowing for efficient reuse.

To add my code to a code snippet, I can right-click on the selected code and left-click on "Create Code Snippet..." I can assign a word to this snippet.

Animation

Adding life to an app is something else that makes the whole UX (user experience) so much better. This is done by adding animation. In SwiftUI, this can be done using the modifier: ".animation"

Here is an animation example:

```
struct ContentView: View {
    @State private var scale = 1.0

    var body: some View {
        Button("Press here") {
            scale += 1
        }
        .scaleEffect(scale)
        .animation(.linear(duration: 1), value: scale)
    }
}
```

Below is me using .animation in my code:

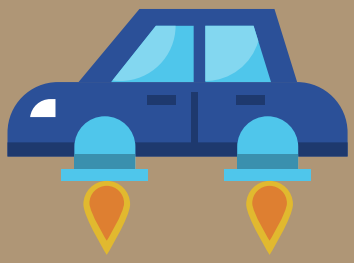
```
.animation(
    Animation
        .easeInOut(duration: 0.5)
        .repeatForever(autoreverses: true),
    value: bounce
)
```

In the future....

I have several exciting updates planned and new apps I want to develop. My goals include enhancing existing features for better user experience and exploring innovative ideas that meet user needs. Below, I'll outline my development roadmap and upcoming updates.

- More Dice to RandomsChoice

- Adding sound to RandomsChoice

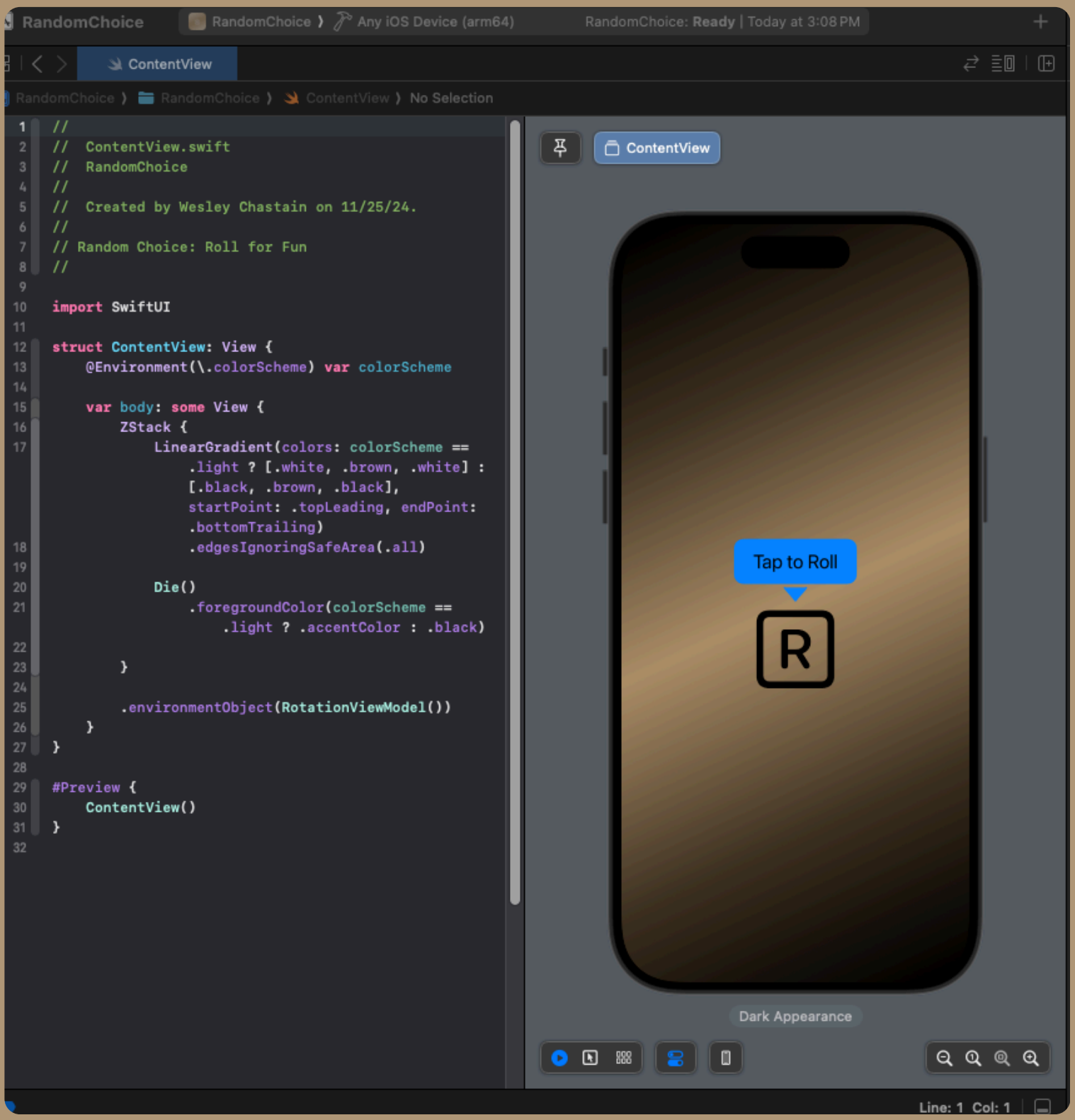


- Shake the screen to roll the die
- Make more apps, such as iMessage apps and more!
- Add ads and in-app purchases to the apps.

- Make these apps for not just Apple, but also Android and even Windows!

The Code

Now I'll show you the complete Swift code. You'll get to see the entire software and organization of RandomsChoice.



The app was going to be called RandomChoice, but that name had already been taken to the App Store. This is why I changed the name to RandomsChoice

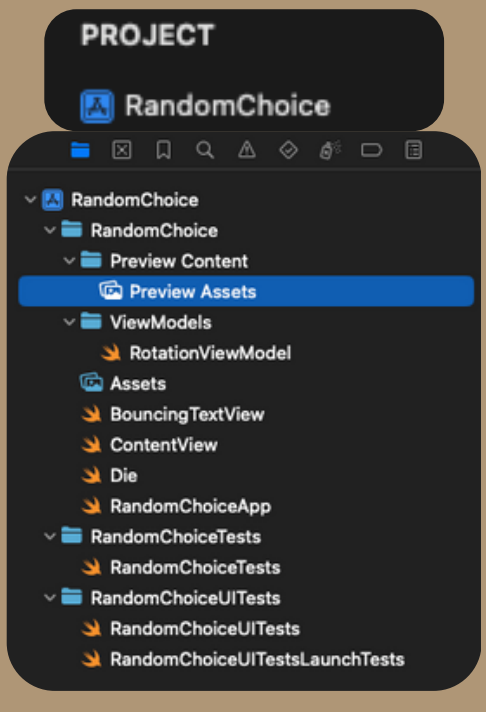
RandomChoiceApp

```swift
//
//  RandomChoiceApp.swift
//  RandomChoice
//
//  Created by Wesley Chastain on 11/25/24.
//

import SwiftUI

@main
struct RandomChoiceApp: App {
    var body: some Scene {
        WindowGroup {
            ContentView() // call content view, this is what runs the app
        }
        .environmentObject(RotationViewModel())
    }
}
```

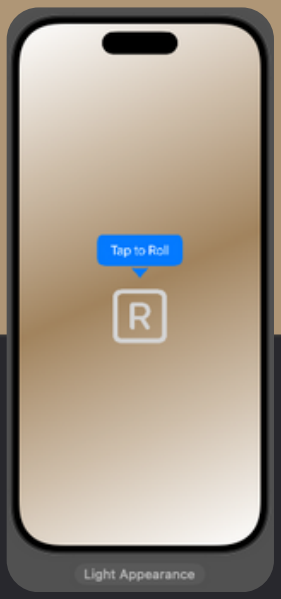
ViewModels

RotationViewModel

```swift
//
//  RotationViewModel.swift
//  RandomChoice
//
//  Created by Wesley Chastain on 11/25/24.
//


import SwiftUI
import Combine
class RotationViewModel: ObservableObject {
    @Published var rotation: Double = 0
    // setting the rotation variable for the die animation
}
```



The ContentView gets called in the RandomChoiceApp structure. The ContentView structure is what holds the makings of the app.

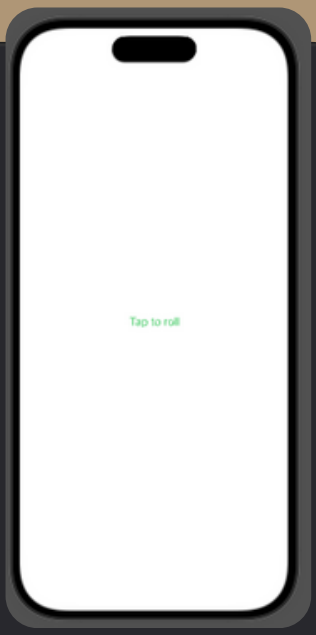


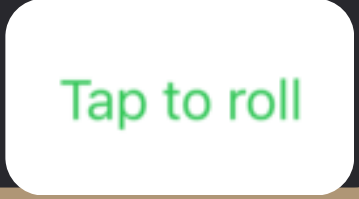
ContentView

```swift
//
//  ContentView.swift
//  RandomChoice
//
//  Created by Wesley Chastain on 11/25/24.
//
// Random Choice: Roll for Fun
//

import SwiftUI

struct ContentView: View {
    @Environment(\.colorScheme) var colorScheme

    var body: some View {
        ZStack {
            LinearGradient(colors: colorScheme == .light ? [.white, .brown,
                .white] : [.black, .brown, .black], startPoint:
                .topLeading, endPoint: .bottomTrailing)
                .edgesIgnoringSafeArea(.all)
            // background gradient

            Die()
                .foregroundColor(colorScheme == .light ? .accentColor :
                    .black)
            // change the die color if the mode is dark or light

        }

        .environmentObject(RotationViewModel())
    }
}

#Preview {
    ContentView()
}
```

BouncingTextView

```swift
//
//  BouncingTextView.swift
//  RandomChoice
//
//  Created by Wesley Chastain on 11/25/24.
//


import SwiftUI

struct BouncingTextView: View {
    @State private var hideInstructions = false
    @State private var bounce = false

    var body: some View {
        VStack {
            // bouncing text "Tap to roll" for instructions
            Text(hideInstructions ? "" : "Tap to roll")
                .foregroundColor(.green)
                .offset(y: bounce ? -10 : 10)
                .animation(
                    Animation
                        .easeInOut(duration: 0.5)
                        .repeatForever(autoreverses: true),
                    value: bounce
                    // bounce text animation
                )
            // show instructions when the app opens
                .onAppear {
                    bounce = true
                }
                .onTapGesture {
                    withAnimation {
                        hideInstructions = true
                        // hide instructions after the first tap gesture
                    }
                }
        }
        .padding()
    }
}

struct BouncingTextView_Previews: PreviewProvider {
    static var previews: some View {
        BouncingTextView()
    }
}
```





Die


```swift
//
//  Die.swift
//  RandomChoice
//
//  Created by Wesley Chastain on 11/25/24.
//

import SwiftUI

struct ShakeEffect: GeometryEffect {
    var amount: CGFloat = 10
    var shakesPerUnit: CGFloat = 3
    var animatableData: CGFloat
    // manage shake effect in the animation

    func effectValue(size: CGSize) -> ProjectionTransform {
        ProjectionTransform(CGAffineTransform(translationX: amount *
        sin(animatableData * .pi * shakesPerUnit), y: 0))
    }
}

struct PointedBubble: Shape {
    // Create the information instruction box that points to the die
    func path(in rect: CGRect) -> Path {
        var path = Path()

        let cornerRadius: CGFloat = 10
        let pointerSize: CGFloat = 20

        // Create the main rounded rectangle
        path.addRoundedRect(in: CGRect(x: 0, y: 0, width: rect.width, height:
            rect.height - pointerSize), cornerSize: CGSize(width: cornerRadius,
            height: cornerRadius))

        // Add the pointer
        path.move(to: CGPoint(x: rect.width / 2 - 3 - pointerSize / 2, y:
            rect.height - pointerSize + 5))
        path.addLine(to: CGPoint(x: rect.width / 2, y: rect.height))
        path.addLine(to: CGPoint(x: rect.width / 2 + 3 + pointerSize / 2, y:
            rect.height - pointerSize + 5))

        return path
    }
}
```

Die

```
struct Die: View {
    @Environment(\.colorScheme) var colorScheme // dark or light mode variable
    @State var number = 0 // what number did I roll?
    @EnvironmentObject var rotation: RotationViewModel
    @State var swipeUp: Int = 0
    @State var numRotate = 0
    @State var hideInstructions: Bool = false // instructions hidden boolean:
        yes or no
    @State private var shakes: CGFloat = 0
    @State var bounce = false

    var body: some View {
        VStack {


            if !hideInstructions { // explaining the instructions if they are
                NOT hidden
                Text("Tap to Roll")
                    .font(.title2)
                    .foregroundColor(colorScheme == .dark ? .black : .white)
                    .padding()
                    .offset(y: -10)
                    .frame(width: 140, height: 70)
                    .background(PointedBubble().fill(Color.blue))
                    .overlay(PointedBubble().stroke(Color.blue, lineWidth: 2))
                    .offset(y: bounce ? 10 : 20)
                    .animation(
                        Animation
                            .easeInOut(duration: 0.5)
                            .repeatForever(autoreverses: true),
                        value: bounce
                    )
                    .onAppear { // add animation to the instructions to make it
                        feel real
                        bounce = true
                    }
                    .onTapGesture { // hide instructions after the die is
                        activiated
                        withAnimation {
                            hideInstructions = true
                        }
                    }
            }
```

Die

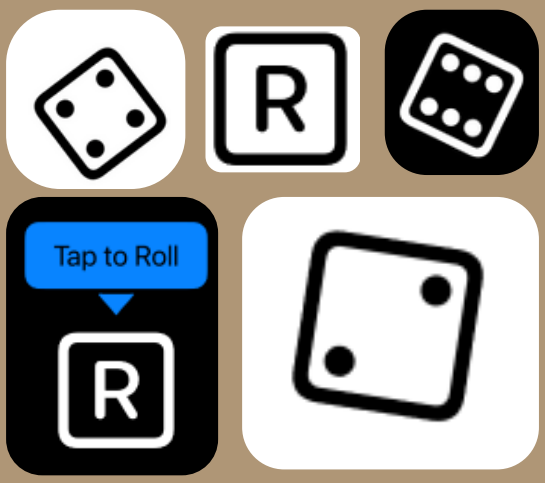
```
struct Die: View {
    @Environment(\.colorScheme) var colorScheme // dark or light mode variable
    @State var number = 0 // what number did I roll?
    @EnvironmentObject var rotation: RotationViewModel
    @State var swipeUp: Int = 0
    @State var numRotate = 0
    @State var hideInstructions: Bool = false // instructions hidden boolean:
        yes or no
    @State private var shakes: CGFloat = 0
    @State var bounce = false

    var body: some View {
        VStack {


            if !hideInstructions { // explaining the instructions if they are
                NOT hidden
                Text("Tap to Roll")
                    .font(.title2)
                    .foregroundColor(colorScheme == .dark ? .black : .white)
                    .padding()
                    .offset(y: -10)
                    .frame(width: 140, height: 70)
                    .background(PointedBubble().fill(Color.blue))
                    .overlay(PointedBubble().stroke(Color.blue, lineWidth: 2))
                    .offset(y: bounce ? 10 : 20)
                    .animation(
                        Animation
                            .easeInOut(duration: 0.5)
                            .repeatForever(autoreverses: true),
                        value: bounce
                    )
                    .onAppear { // add animation to the instructions to make it
                        feel real
                        bounce = true
                    }
                    .onTapGesture { // hide instructions after the die is
                        activiated
                        withAnimation {
                            hideInstructions = true
                        }
                    }
            }
```

Die

```
ZStack {

    VStack {

        // display a different number according to the random
            variable number
        Image(systemName: number == 1 ? "die.face.1" : number == 2 ?
            "die.face.2" : number == 3 ? "die.face.3" : number == 4
            ? "die.face.4" : number == 5 ? "die.face.5" : number ==
            6 ? "die.face.6" : "r.square")
            .font(.system(size: 100))
    }
}
.onTapGesture {
    hideInstructions = true

    withAnimation {
        self.shakes += CGFloat(Int.random(in: 1...3)) // Trigger the
            shake animation
    }

    DispatchQueue.main.asyncAfter(deadline: .now() + 0.5)
    {
        withAnimation
        {
            // randomly generate a number between 1 and 6
            // use these numbers because the probibility on a die is
                1/6
            number = Int.random(in: 1...6)
            rotation.rotation += Double(360*Double.random(in:
                0.5...2))
            numRotate += 360*Int.random(in: 1...3)

            // the rotation of the die will be different every time
            // the number that is rolled is not the only thing that
                is random

        }
    }
}
.font(.largeTitle)
.modifier(ShakeEffect(animatableData: shakes))
.rotationEffect(Angle(degrees: rotation.rotation))
.animation(.easeInOut(duration: 1), value: rotation.rotation)
.padding()
```

Die

```
        }

    }
}

#Preview {
    Die()
        .environmentObject(RotationViewModel())
}
```



That's all the code that runs the RandomsChoice app on the Apple iOS store. Thanks for reading!

There are other files included on the project, but they are not responsible for running the app. They are called test files.

RandomChoiceUITests

RandomChoiceUITests

RandomChoiceUITestsLaunchTests

```
//
//  RandomChoiceTests.swift
//  RandomChoiceTests
//
//  Created by Wesley Chastain on 11/25/24.
//

import Testing
@testable import RandomChoice

struct RandomChoiceTests {

    @Test func example() async throws {
        // Write your test here and use APIs like `#expect(...)` to check expected conditions.
    }

}
```

Resources

HACKINGWITHSWIFT.COM

DEVELOPER.APPLE.COM

XCODE

LINKEDIN.COM/LEARNING

SWIFT.ORG

GEEKSFORGEEKS.ORG

YOUTUBE.COM/C/$WIFTFULTHINKING

MEDIUM.COM