Graphics and Animation

Mobile Application Development in iOS

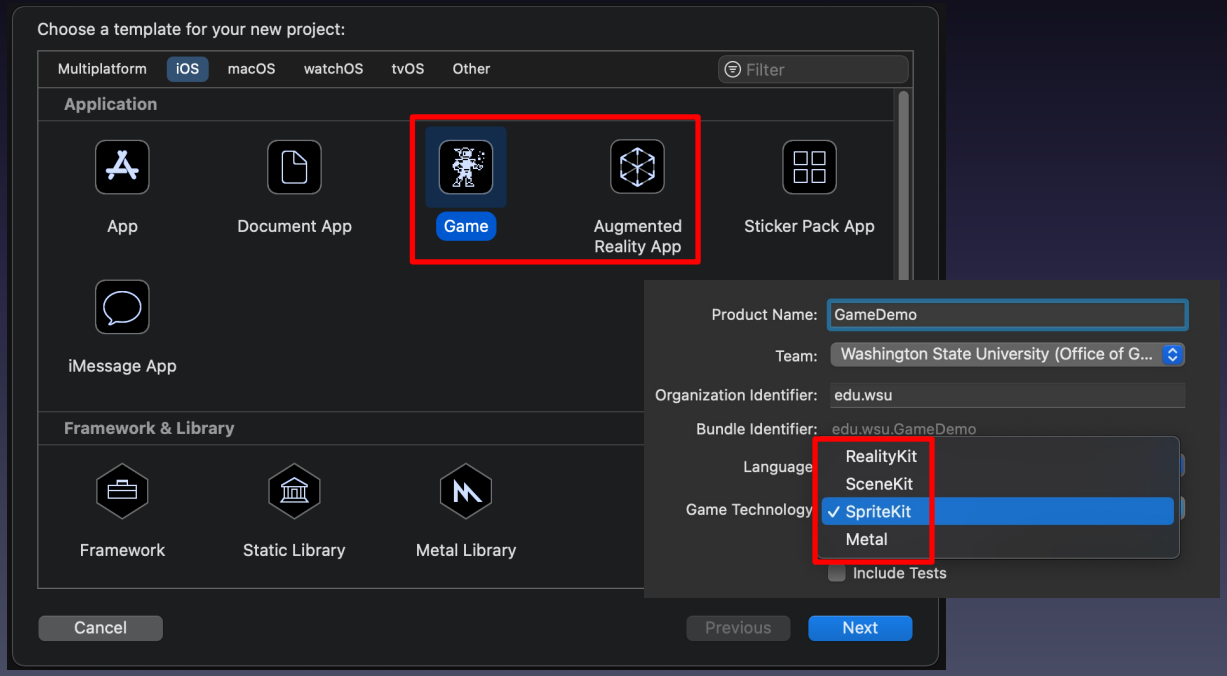
School of EECS

Washington State University

Instructor: Larry Holder

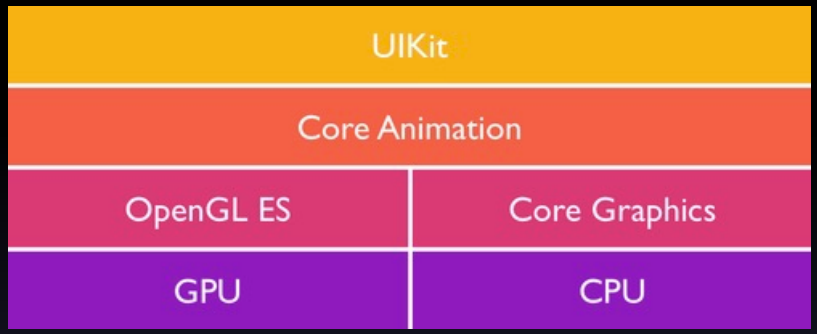
Outline

- iOS frameworks for graphics and animation
- Core Graphics and Core Animation
- SpriteKit 2d
- [SceneKit 3d]
- [ARKit]



iOS Frameworks (old)

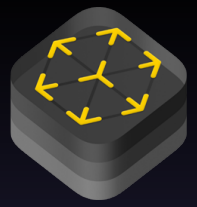
- UIKit graphics
  - Animate elements of view



- Core Graphics and Core Animation
  - 2D graphics and animation engine
  - Part of UIView
- OpenGL ES and GLKit
  - 2D and 3D rendering for GPUs on Embedded Systems (ES)

iOS Frameworks (new)

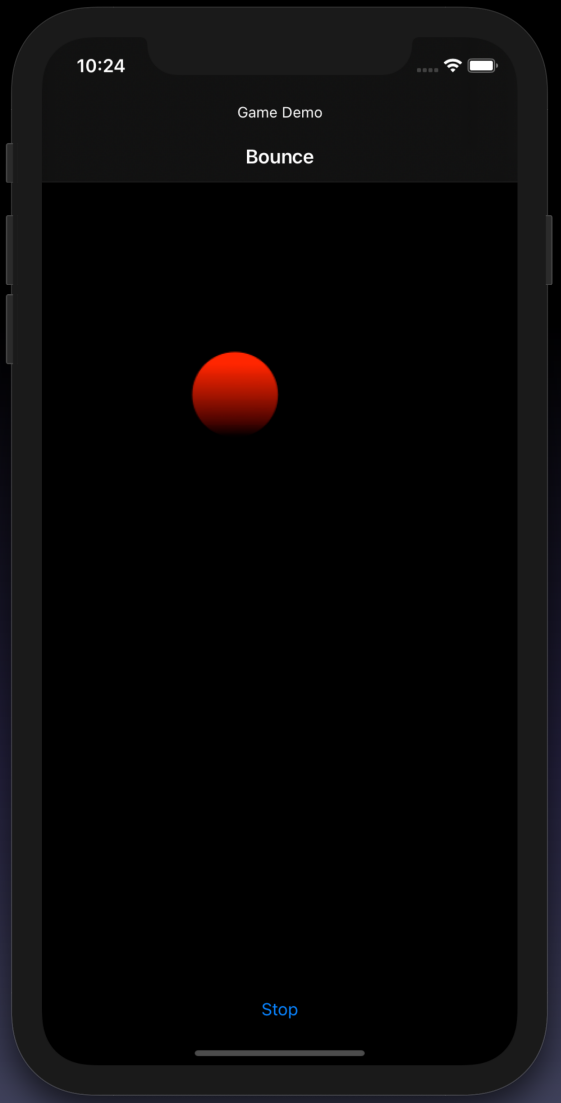
- SpriteKit: 2D game engine
- SceneKit: 3D game engine
- ARKit and RealityKit: Augmented reality
- MetalKit
  - More direct access to GPU for graphics and computation
- GameKit
  - Leaderboards, achievements, challenges, matchmaking
  - Access to Game Center
- GameplayKit
  - State machines, agents, goals, behaviors, rules



Cross Platform Games Engines

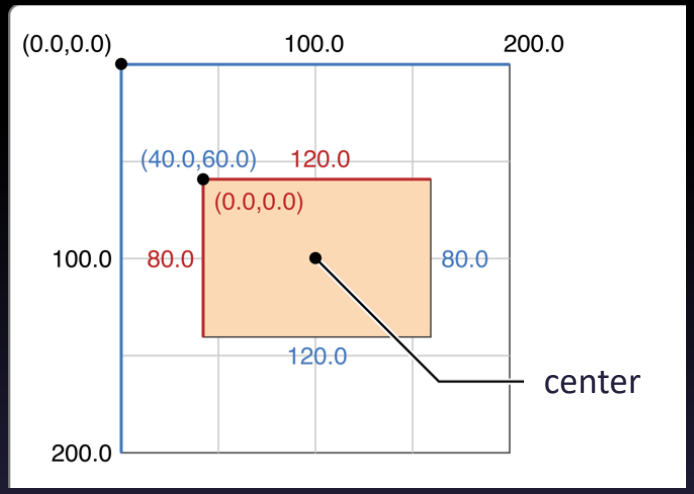
- Unity (unity3d.com) – C#, JavaScript
- Unreal (unrealengine.com) – C++, scripting

Core Graphics



Core Graphics Approach

- Coordinate system (upper-left origin)
- Points vs. pixels
  - Code uses points
  - Framework maps points to pixels
- Frames vs. bounds
  - CGRect = {origin.x, origin.y, size.width, size.height}
  - UIView.frame {40, 60, 120, 80}
  - UIView.bounds {0, 0, 120, 80}
  - UIView.center: CGPoint {100, 100}



Core Graphics Approach

- Add a UIView as a subView of the main view
- Implement gameUpdate() method
  - Modify subView's position
- Use Timer to call gameUpdate() method repeatedly

Core Graphics Approach

```
class ViewController: UIViewController {

    let frameRate = 30.0 // updates per second
    let ballSpeed = 200.0 // points per second
    var ballDirection = CGPoint(x: 1.0, y: -1.0)
    var ballImageView: UIImageView!
    var gameTimer: Timer!
    var gameRunning = false

    func initGame() {
        let ballImage = UIImage(named: "redball.png")!
        ballImageView = UIImageView()
        ballImageView.image = ballImage
        ballImageView.frame = CGRect(x: 0, y: 0,
            width: ballImage.size.width,
            height: ballImage.size.height)
        self.view.addSubview(ballImageView)
    }
```

Core Graphics Approach

```
func startGame () {
    gameTimer = Timer.scheduledTimer(withTimeInterval:
        (1.0 / frameRate), repeats: true, block: updateGame)
    gameRunning = true
}

func pauseGame () {
    gameTimer.invalidate()
    gameRunning = false
}
```

Core Graphics Approach

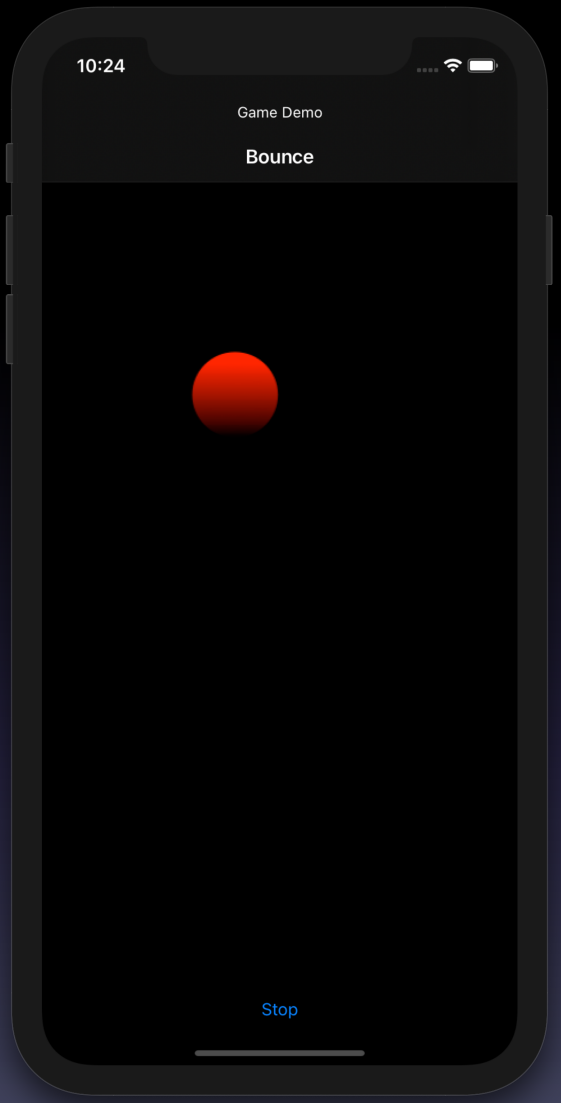
```
func updateGame (timer: Timer) {
    let x = ballImageView.frame.origin.x
    let y = ballImageView.frame.origin.y
    let width = ballImageView.frame.width
    let height = ballImageView.frame.height
    // If ball hits wall, then change direction
    if (x < 0) { // Hit left wall
        ballDirection.x = 1.0
    }
    if ((x + width) > self.view.frame.width) { // Hit right wall
        ballDirection.x = -1.0
    }
    // Handle top and bottom walls...
    // Update ball location
    let xOffset = CGFloat(ballSpeed / frameRate) * ballDirection.x
    let yOffset = CGFloat(ballSpeed / frameRate) * ballDirection.y
    ballImageView.frame.origin.x = x + xOffset
    ballImageView.frame.origin.y = y + yOffset
}
```

Watch for orientation change.

Core Graphics Extras

```
func centerBall() {
    ballImageView.center = CGPoint(x: self.view.frame.width / 2,
                                   y: self.view.frame.height / 2)
}

func topBarHeight() -> CGFloat {
    let statusBarHeight =
self.view.window?.windowScene?.statusBarManager?.statusBarFrame.height
?? 0.0
    let navBarHeight =
self.navigationController?.navigationBar.frame.height ?? 0.0
    return (statusBarHeight + navBarHeight)
}

func bottomMarginHeight() -> CGFloat {
    return startStopButton.frame.height + 20
}
```

Core Animation



UIView.animate

- UIView.animate (withDuration duration: TimeInterval,
  delay: TimeInterval,
  options: UIView.AnimationOptions = [],
  animations: @escaping () -> Void,
  completion: ((Bool) -> Void)?)

Core Animation

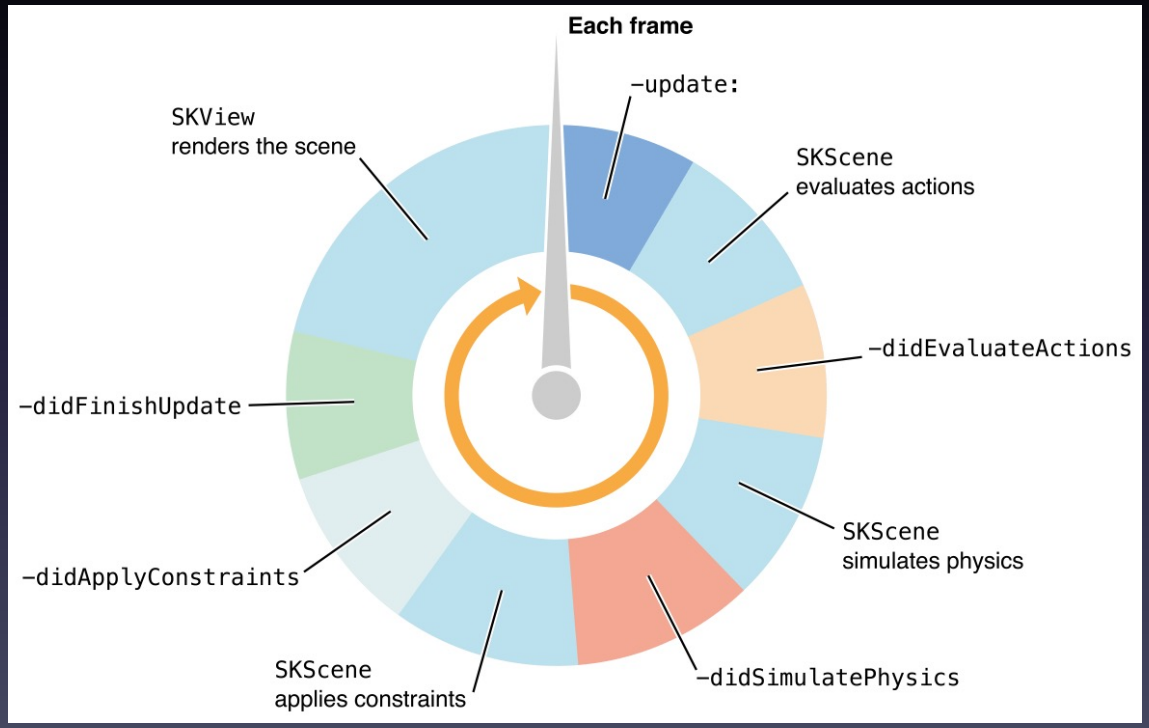
- Move ball to center at start of game

```
UIView.animate(withDuration: 1, delay: 0, animations: {
    self.ballImageView.center = CGPoint(
        x: self.view.frame.width / 2,
        y: self.view.frame.height / 2)
    }, completion: nil)
```

SpriteKit

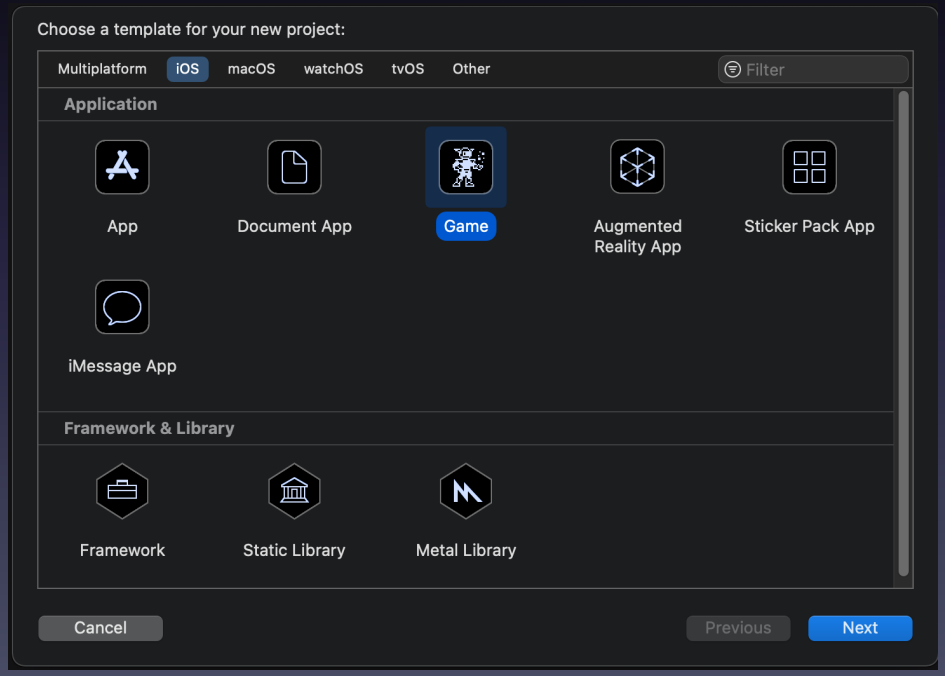
SpriteKit Approach

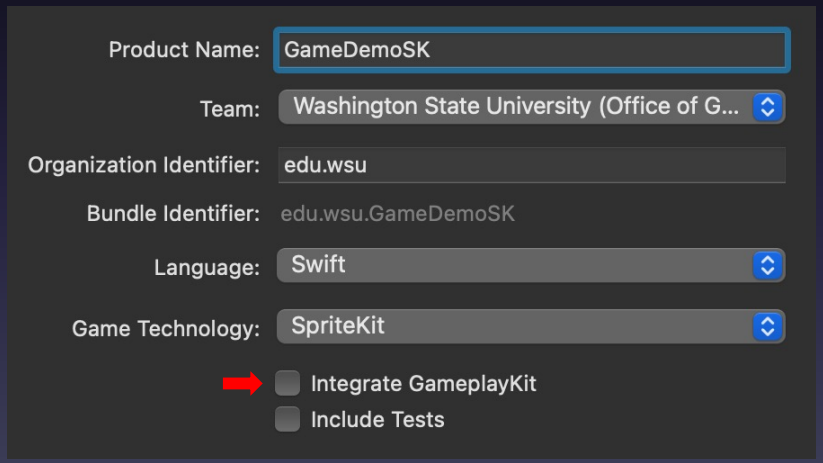
- Update/render loop



SpriteKit Approach

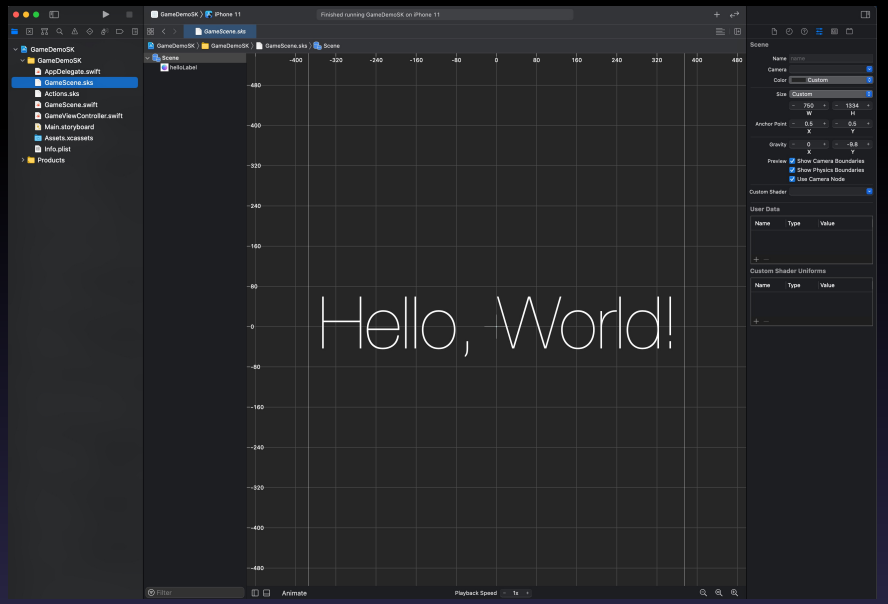
- Create new Game project
  - Game Technology: SpriteKit





SpriteKit Organization

- Scene(s) of type SKScene
  - GameScene.sks
  - Edit in Sprite Editor



- Scene controlled by SKScene class

GameScene.swift

```
import SpriteKit
import GameplayKit

class GameScene: SKScene {
```

SpriteKit Organization

- Main view of type SKView
- Present SKScene in SKView

GameViewController.swift

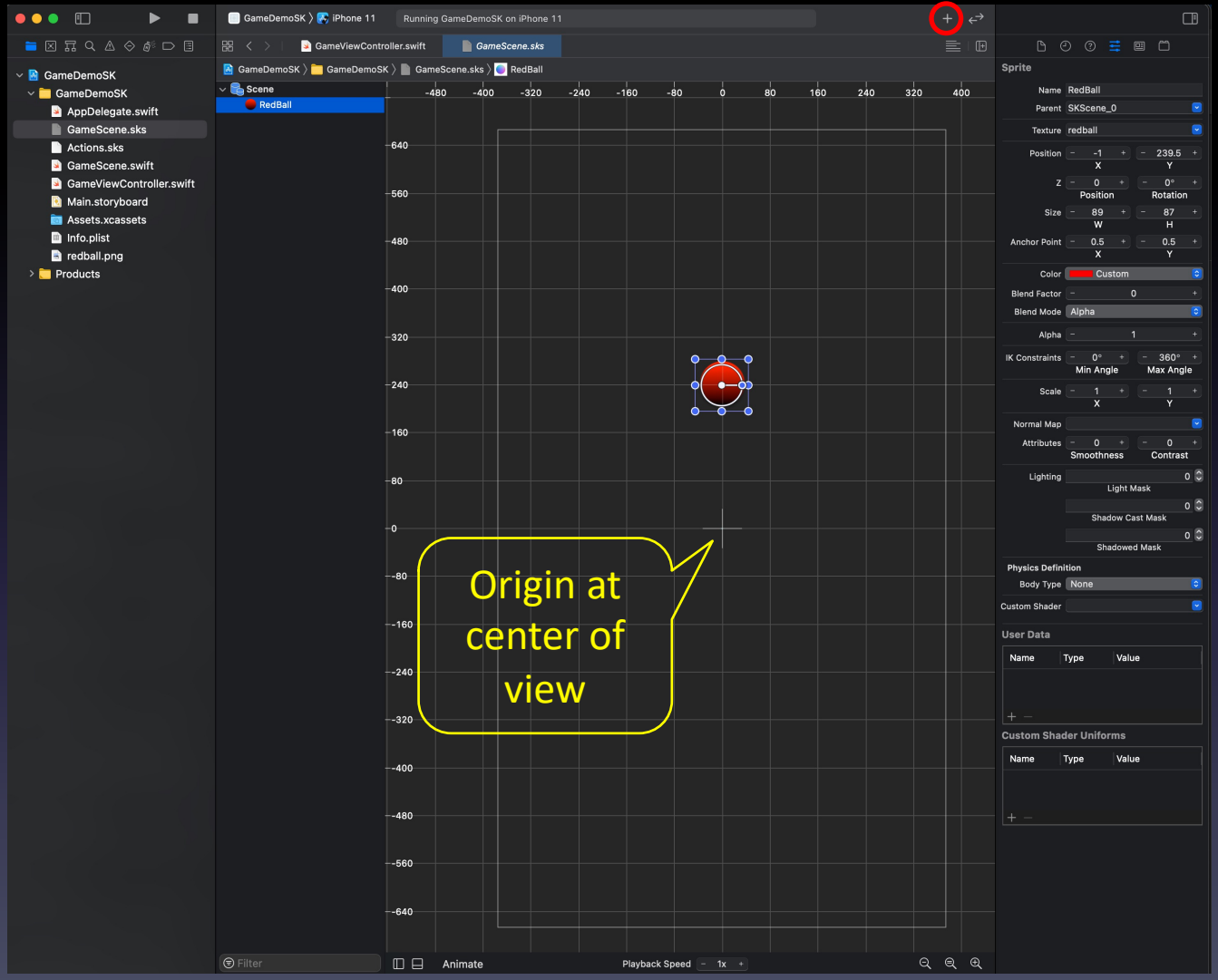
```
override func viewDidLoad() {
    super.viewDidLoad()
    if let view = self.view as! SKView? {
        // Load the SKScene from 'GameScene.sks'
        if let scene = SKScene(fileNamed: "GameScene") {
            // Set the scale mode to scale to fit the window
            scene.scaleMode = .aspectFill
            // Present the scene
            view.presentScene(scene)
        }
        view.ignoresSiblingOrder = true
        view.showsFPS = true
        view.showsNodeCount = true
    }
}
```

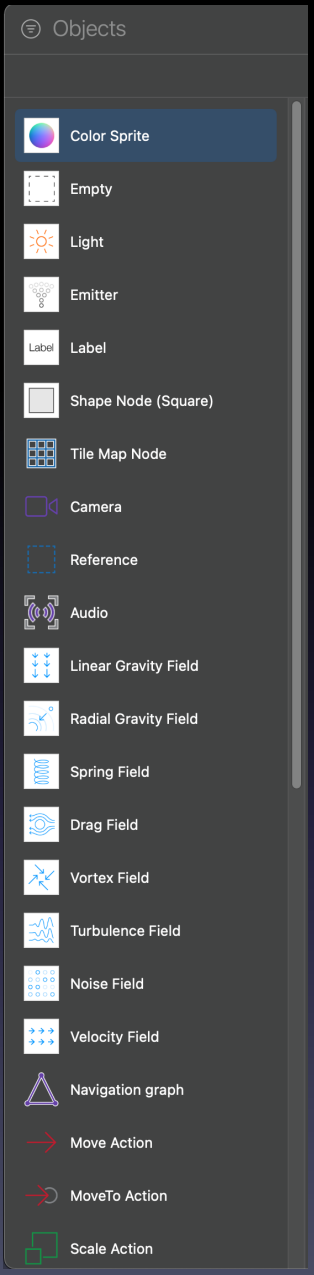
Sprites

redball.png

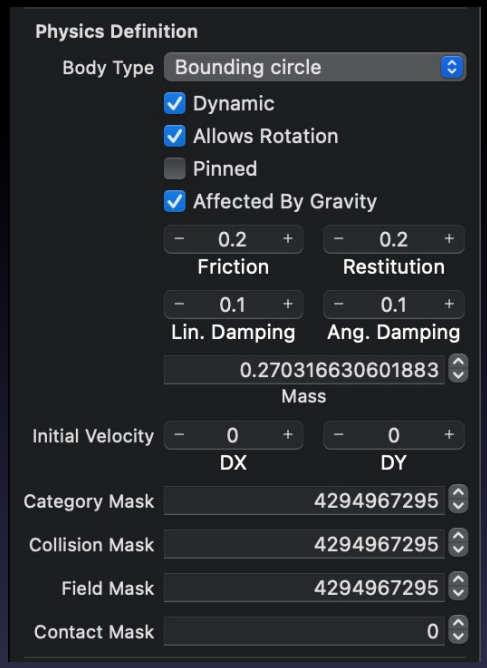
- Sprite is a shape with a texture (image)
- SKSpriteNode is a sprite with many properties
  - SKAction for actions to execute (e.g., fade in/out)
  - SKPhysicsBody for physical effects (e.g., gravity)
- Other types of SKNode's (e.g., SKLabelNode)
- SKScene is a collection of SKNode's

SpriteKit Scene Editor





SpriteKit Physics



- Body Type
- Dynamic
- Pinned (fixed to parent)
- Allows Rotation, Ang. Damping
- Affected By Gravity, Linear Damping, Mass
- Friction, Restitution (energy retained on bounce)

SpriteKit Physics

- Bounce off edge of screen (in SKScene class)

```
func didMove(to view: SKView) {
    // Set screen edge to bounce with no friction
    self.scaleMode = .aspectFit
    self.physicsBody = SKPhysicsBody(edgeLoopFrom: self.frame)
    // ...
}
```

- Get nodes by name (in SKScene class)

```
var redBallNode: SKSpriteNode!

redBallNode = self.childNode(withName: "RedBall") as? SKSpriteNode
```

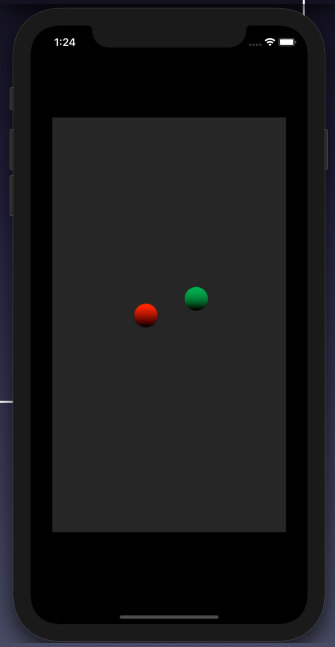
SpriteKit Physics

- Get objects moving
  - Set physics
  - Set velocity
  - Or, apply impulse


```
// Set velocity (ignores mass and current velocity)
redBallNode.physicsBody?.velocity = CGVector(dx: 400.0, dy: 400.0)

// Push (considers mass and current velocity)
redBallNode.physicsBody?.applyImpulse(CGVector(dx: 200.0, dy: 200.0))
```

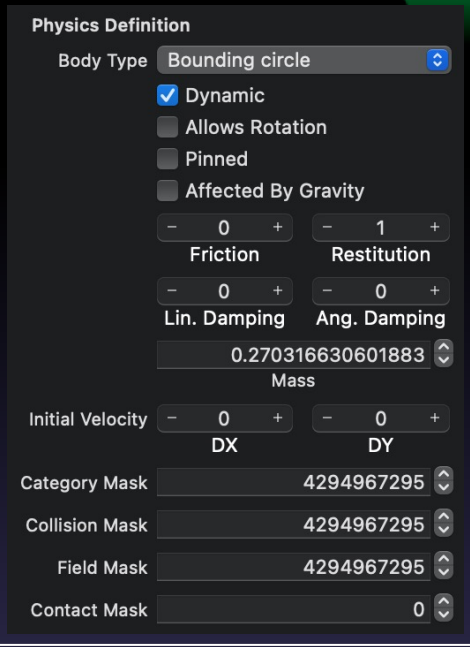
SpriteKit: Adding Nodes Programmatically

```
var greenBallNode: SKSpriteNode!

// Add green ball programmatically
greenBallNode = SKSpriteNode(imageNamed: "greenball.png")
greenBallNode.name = "GreenBall"
greenBallNode.physicsBody = SKPhysicsBody(circleOfRadius:
    greenBallNode.frame.size.width / 2.0)
greenBallNode.physicsBody?.affectedByGravity = false
greenBallNode.physicsBody?.friction = 0.0
greenBallNode.physicsBody?.restitution = 1.0
greenBallNode.physicsBody?.linearDamping = 0.0
self.addChild(greenBallNode)
```



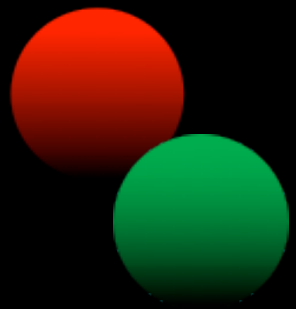
SpriteKit Physics: Collisions

- Mask is a bit string (4294967295 = 32 bits, all 1s)
- Category
  - Mask that is a unique power of 2 for each object type
  - E.g., ball: 0001, brick: 0010, wall: 0100
- Category Mask (SKPhysicsBody.categoryBitMask)
  - Categories this body belongs to
- Collision Mask (SKPhysicsBody.collisionBitMask)
  - Categories this body collides with
- Contact Mask (SKPhysicsBody.contactTestBitMask)
  - Categories generating Contact delegate call, if contacts this body

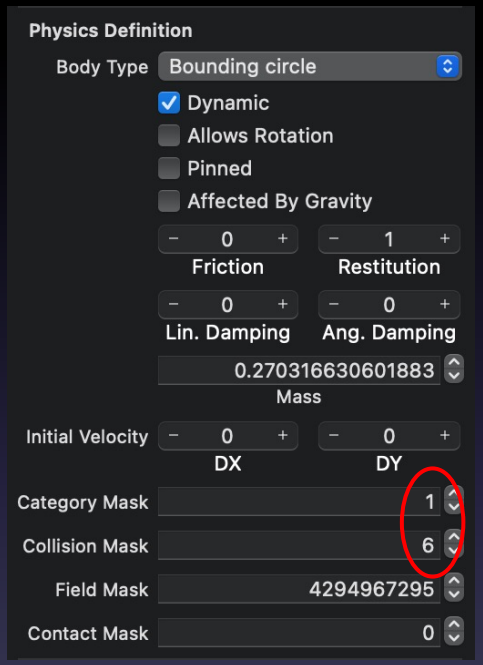



```
Body 1 Category Mask:  0010
Body 2 Collision Mask: 0011
Bitwise And:           0010 > 0
        Collision!
```

SpiteKit Physics: Collisions

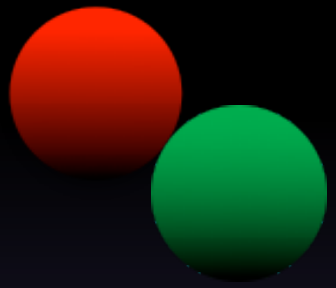


- Example
  - Balls shouldn't collide with each other
    - Ball category: 0b0001
  - But should collide with Wall and Brick
    - Wall category: 0b0100
    - Brick category: 0b0010



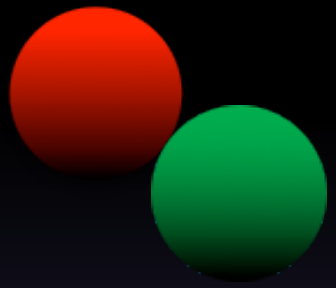

```
greenBallNode.physicsBody?.categoryBitMask = 0b0001 // Ball
greenBallNode.physicsBody?.collisionBitMask = 0b0110 // Wall & Brick
```

SpriteKit Physics: Contacts



- Delegate (in SKScene class)
  - SKPhysicsContactDelegate
  - self.physicsWorld.contactDelegate = self
- Delegate methods (in SKScene class)
  - didBegin(_ contact: SKPhysicsContact)
  - didEnd(_ contact: SKPhysicsContact)
    - contact.bodyA.node
    - contact.bodyB.node

SpriteKit Physics: Contacts



In SKScene class:

```
func didBegin(_ contact: SKPhysicsContact) {
    guard let nodeA = contact.bodyA.node else {return}
    guard let nodeB = contact.bodyB.node else {return}
    print("Contact: \(nodeA.name ?? "?") with \(nodeB.name ?? "?")")
}
```

SpriteKit Interaction

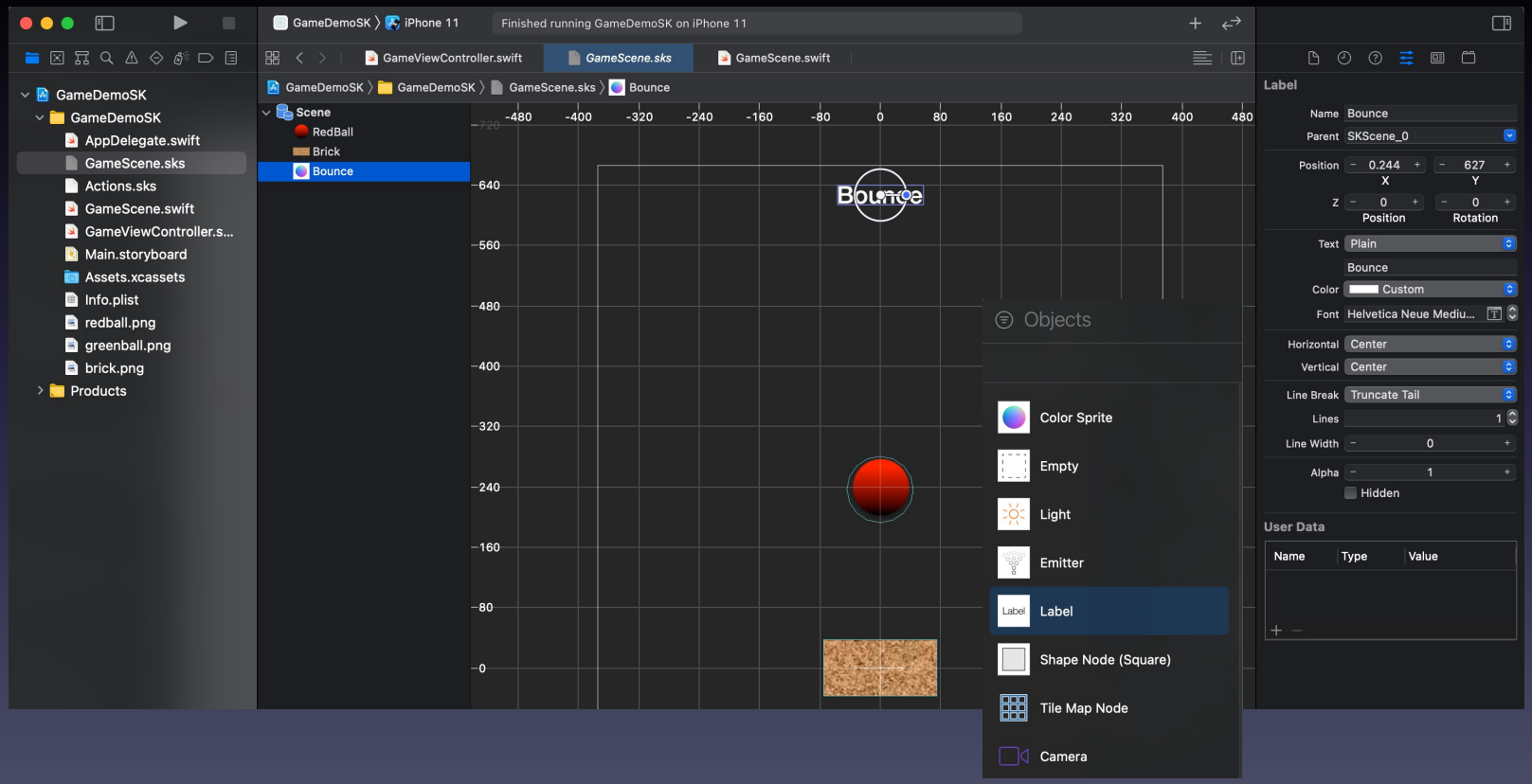
- Touches
  - func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?)
  - func touchesMoved(_ touches: Set<UITouch>, with event: UIEvent?)
  - func touchesEnded(_ touches: Set<UITouch>, with event: UIEvent?)

SpriteKit Interaction

```
override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
    for touch in touches {
        let point = touch.location(in: self)
        let nodeArray = nodes(at: point)
        for node in nodeArray {
            print("tapped \(node.name!)")
        }
    }
}
```

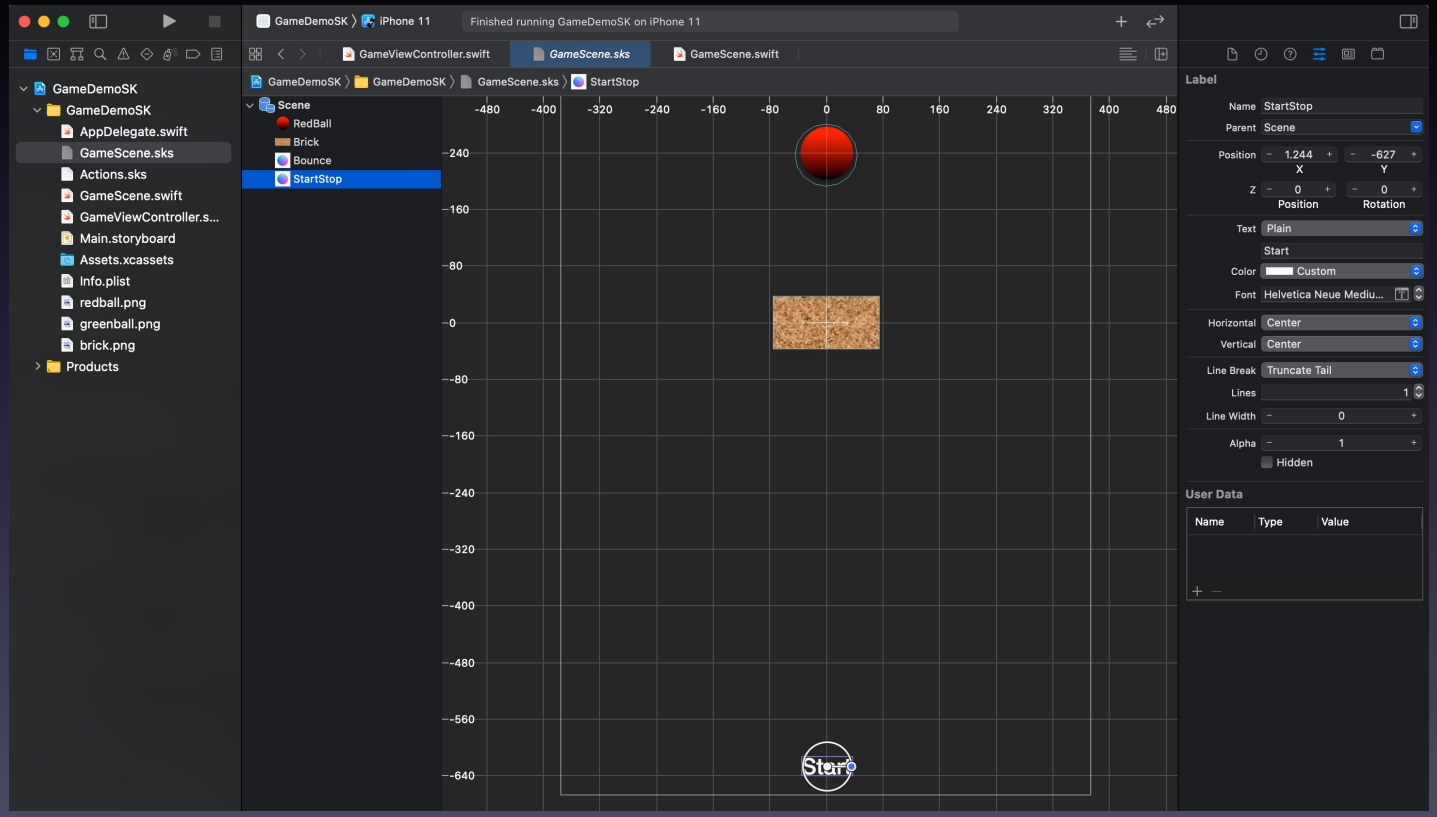
SpriteKit Labels and Buttons

- SKLabelNode



SpriteKit Labels and Buttons

- SKLabelNode (as button)

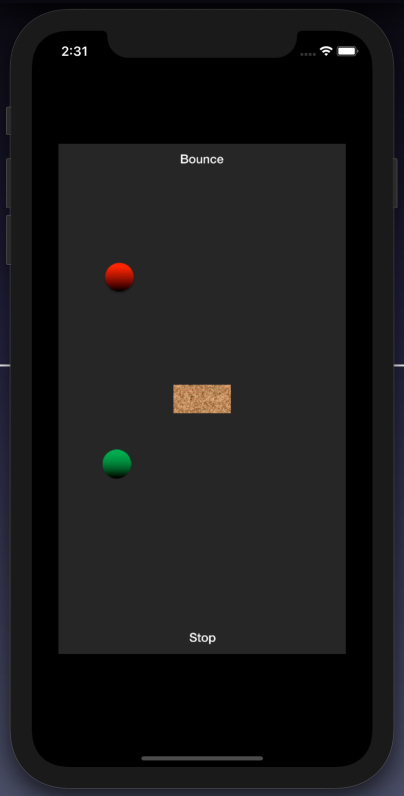


SpriteKit Label as Button

```
var gameRunning = false
var startStopNode: SKLabelNode!
startStopNode = self.childNode(withName: "StartStop") as? SKLabelNode

override func touchesEnded(_ touches: Set<UITouch>,with event: UIEvent?) {
    for touch in touches {
        let point = touch.location(in: self)
        let nodeArray = nodes(at: point)
        for node in nodeArray {
            if node.name == "StartStop" { // StartStop button tapped
                if gameRunning {
                    startStopNode.text = "Start"
                    gameRunning = false
                    pauseGame()
                } else {
                    startStopNode.text = "Stop"
                    gameRunning = true
                    startGame()
                }
            }
        }
    }
}
```

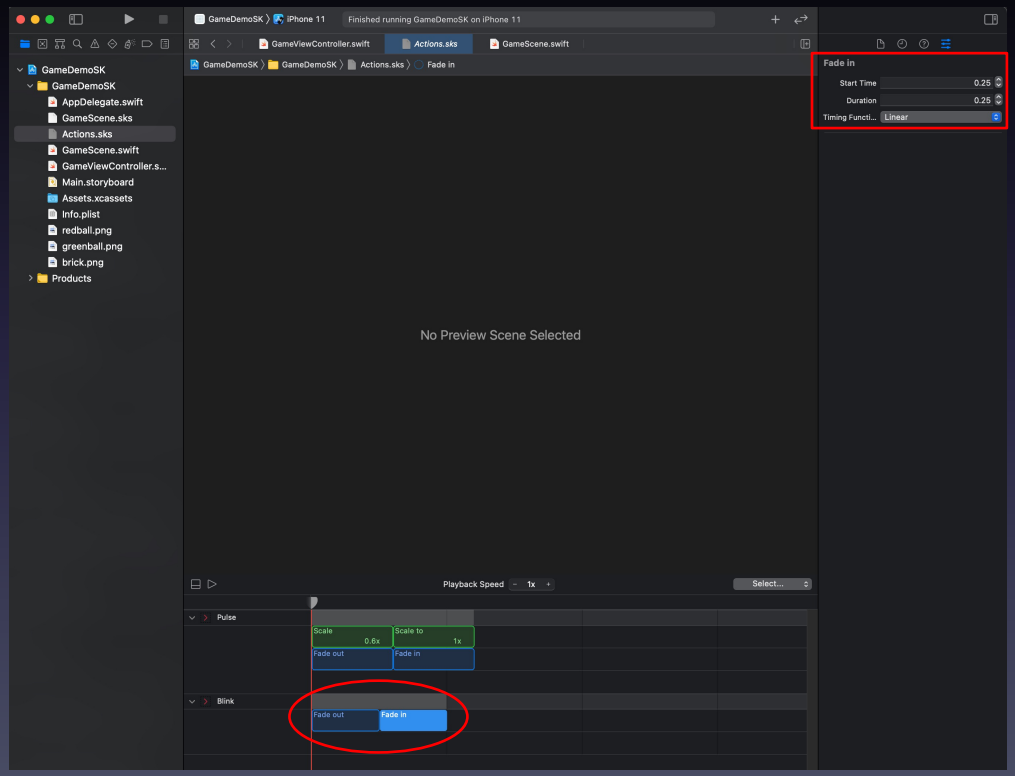
SpriteKit Label as Button

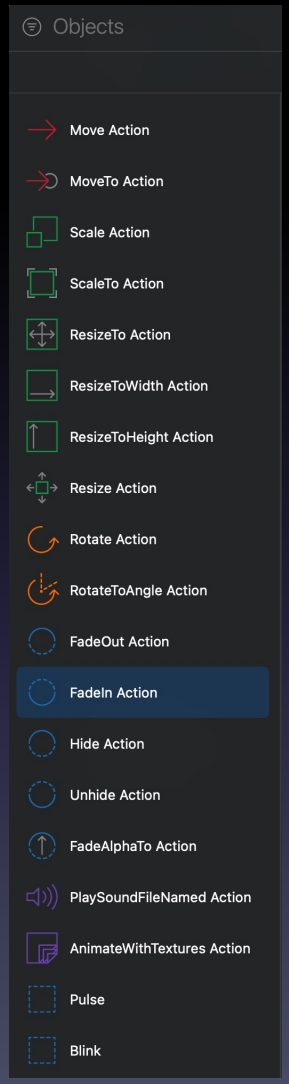
```
override func didMove(to view: SKView) {
    // ...
    self.pauseGame()
    redBallNode.physicsBody?.applyImpulse(CGVector(dx: 200.0, dy: 200.0))
}

func startGame() {
    self.isPaused = false
}

func pauseGame() {
    self.isPaused = true
}
```



SpriteKit Actions: Option 1

- Create SKAction in SpriteKit Action Editor
  - Execute SKNode.run(SKAction.init(named: "Blink"))





SpriteKit Actions: Option 1

```
func didBegin(_ contact: SKPhysicsContact) {
    let nodeA = contact.bodyA.node!
    let nodeB = contact.bodyB.node!
    print("Contact: \(nodeA.name ?? "?") with \(nodeB.name ?? "?")")
    let blinkAction = SKAction.init(named: "Blink")!
    nodeA.run(blinkAction)
    nodeB.run(blinkAction)
}
```

SpriteKit Actions: Option 2

- Create SKAction programmatically
- Execute SKNode.run(SKAction)

```
override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
    for touch in touches {
        let point = touch.location(in: self)
        let nodeArray = nodes(at: point)
        for node in nodeArray {
            print("tapped \(node.name!)")
            let action1 = SKAction.fadeOut(withDuration: 0.25)
            let action2 = SKAction.fadeIn(withDuration: 0.25)
            let blinkAction = SKAction.sequence([action1,action2])
            node.run(blinkAction)
        }
    }
}
```

SpriteKit Actions: Option 2

- Create SKAction programmatically
- Execute SKNode.run(SKAction)

```
func didBegin(_ contact: SKPhysicsContact) {
    let nodeA = contact.bodyA.node!
    let nodeB = contact.bodyB.node!
    print("Contact: \(nodeA.name ?? "?") with \(nodeB.name ?? "?")")
    let action1 = SKAction.fadeOut(withDuration: 0.25)
    let action2 = SKAction.fadeIn(withDuration: 0.25)
    let blinkAction = SKAction.sequence([action1,action2])
    nodeA.run(blinkAction)
    nodeB.run(blinkAction)
}
```

SpriteKit Audio

- Sound effects
  - SKAction.playSoundFileNamed
- Background music
  - AVAudioPlayer
- SKAudioNode
  - Positional
  - Effects, e.g., reverb

SpriteKit Audio: Sound Effects

- Create SKAction.playSoundFileNamed
- Execute SKScene.run(SKAction)

```
var bounceSoundAction: SKAction!

bounceSoundAction = SKAction.playSoundFileNamed("bounce.mp3",
        waitForCompletion: false)

func didBegin(_ contact: SKPhysicsContact) {
    let nodeA = contact.bodyA.node!
    let nodeB = contact.bodyB.node!
    // ...
    self.run(bounceSoundAction)
}
```

Background Music

- Import AVFoundation
- Create AVAudioPlayer from URL to music file
  - AVAudioPlayer(contentsOf: URL)
- Set volume, numberOfLoops (-1 = loop continuously), …
- Methods: play, pause, stop, …

Background Music

```
import AVFoundation

var audioPlayer: AVAudioPlayer?

let musicURL = Bundle.main.url(forResource: "WSU-Fight-Song.mp3",
        withExtension: nil)
do {
    audioPlayer = try AVAudioPlayer(contentsOf: musicURL!)
} catch {
    print("error accessing music")
}
audioPlayer?.volume = 0.25
audioPlayer?.numberOfLoops = -1 // loop forever

audioPlayer?.play() // In startGame()
audioPlayer?.pause() // In pauseGame()
```

Resources

- Core Graphics: developer.apple.com/documentation/coregraphics
- Core Animation: developer.apple.com/documentation/uikit/uiview/1622418-animate
- Sprite Kit: developer.apple.com/documentation/spritekit
- Scene Kit: developer.apple.com/documentation/scenekit
- GameplayKit: developer.apple.com/documentation/gameplaykit
- ARKit: developer.apple.com/documentation/arkit
- AVFoundation: developer.apple.com/documentation/avfoundation

Assets

redball.png

brick.png

greenball.png

blueball.png