IOS GAME PROGRAMMING

MATTHEW LIN

ABOUT ME

- 3rd Year Computer Science Major
- Incoming Software Engineering Intern at Apple (Summer 2016)
- Started learning Swift in Summer 2015 through Stanford 193P tutorials on iTunes U
- Released an iOS game to the App Store called "Keepy-Uppy"
- Avid video game fan since the age of 4 (1999)

IOS GAME DEVELOPMENT IS FUN!

- Play the games you develop!
  - iPhone
  - iPad
- Make money on the App Store
  - Purchase fee
  - Ads
  - In-app purchases
- Good resume booster
  - Apple loves iOS developers!

SPRITEKIT IS A GRAPHICS LIBRARY THAT MAKES IT EASY TO DEVELOP IOS GAMES!

- SpriteKit provides a plethora of useful features
  - Renders frames of animation efficiently using graphics hardware
  - Creates physics simulations such as gravity, collisions, etc.
  - Develop complex special effects and create various texture atlases (sprites)

LET'S GET STARTED!

- Tools you need:
  - Mac Laptop
  - Xcode 7 (for Swift 2.0)
- Optional Tools
  - iPhone
  - iPad
  - Lightning to USB cable
- Run this line in your Terminal:
  - git clone https://github.com/Darthpwner/iOS-Game-Programming-Skeleton-Code.git

SPRITEKIT BASICS

- import SpriteKit
- class <file name>: SKScene, SKPhysicsContactDelegate
- SpriteKit variable types:
    - SKSpriteNode: Creates a textured image
    - SKColor: Represents color and sometimes opacity (alpha value)
    - SKTexture: Manages the texture data and graphics that are needed to render the image
    - SKAction: Action that is executed by a node in the scene
    - SKLabelNode: Loads font and creates text for display
    - NSInteger: Used to describe an integer in SpriteKit
    - UInt32: A 32-bit unsigned integer that is used to handle collisions and contact

STEP 1: SETTING UP THE BIRD

- filteringMode is used when to make sprite size responsive
  - .Nearest means each pixel is drawn with the nearest point in the texture.
- animateWithTextures alternates between the two images every 0.2 seconds

```
func setupBird() {
    let birdTexture1 = SKTexture(imageNamed: "bird-01")
    birdTexture1.filteringMode = .Nearest
    let birdTexture2 = SKTexture(imageNamed: "bird-02")
    birdTexture2.filteringMode = .Nearest

    let anim = SKAction.animateWithTextures([birdTexture1, birdTexture2], timePerFrame: 0.2)
    let flap = SKAction.repeatActionForever(anim)

    bird = SKSpriteNode(texture: birdTexture1)
    bird.setScale(2.0)
    bird.position = CGPoint(x: self.frame.size.width * 0.35, y:self.frame.size.height * 0.6)
    bird.runAction(flap)

    self.addChild(bird)
}
```

STEP 2: SETTING UP THE PHYSICS

- physicsWorld applies physical properties to the entire game world
- bird.physicsBody applies physical properties to the bird node

```
func setupPhysics() {
    self.physicsWorld.gravity = CGVector( dx: 0.0, dy: -5.0 )
    self.physicsWorld.contactDelegate = self
}

func setupBirdPhysics() {
    bird.physicsBody = SKPhysicsBody(circleOfRadius: bird.size.height / 2.0)
    bird.physicsBody?.dynamic = true
    bird.physicsBody?.allowsRotation = false

    bird.physicsBody?.categoryBitMask = birdCategory
    bird.physicsBody?.collisionBitMask = worldCategory | pipeCategory
    bird.physicsBody?.contactTestBitMask = worldCategory | pipeCategory
}
```

STEP 3: SETTING UP THE BIRD'S ORIENTATION

- clamp helper function is used to prevent the bird from spinning around too much in the game world
- zRotation represents the Euler rotation around the vertical axis

```
// TODO: Move to utilities somewhere. There's no reason this should be a member function
func clamp(min: CGFloat, max: CGFloat, value: CGFloat) -> CGFloat {
    if( value > max ) {
        return max
    } else if( value < min ) {
        return min
    } else {
        return value
    }
}

override func update(currentTime: CFTimeInterval) {
    //Called before each frame is rendered
    bird.zRotation = self.clamp(-1, max: 0.5, value: bird.physicsBody!.velocity.dy * (bird.physicsBody!.velocity.dy < 0 ? 0.003 : 0.001 ) )
}
```

STEP 4: SETTING UP THE TAP RECOGNIZER

- setUpTaps makes the game world appear to be moving
- touchesBegan handles all the touch actions in this case when your finger taps the screen

```swift
func setupTaps() {
    moving = SKNode()
    self.addChild(moving)
    pipes = SKNode()
    moving.addChild(pipes)
}

override func touchesBegan(touches: Set<UITouch>, withEvent event: UIEvent?) {
    /* Called when a touch begins */
                if moving.speed > 0  {
                    for touch: AnyObject in touches {
                        let location = touch.locationInNode(self)

                        bird.physicsBody?.velocity = CGVector(dx: 0, dy: 0)
                        bird.physicsBody?.applyImpulse(CGVector(dx: 0, dy: 30))

                    }
                } else if canRestart {
                    self.resetScene()
                }
}
```

STEP 5: CREATING THE GROUND

- moveByX makes the ground appear to be moving
- for loop starts the action to move the ground on the screen

```
    func createGround() {
        groundTexture.filteringMode = .Nearest // shorter form for SKTextureFilteringMode.Nearest

        let moveGroundSprite = SKAction.moveByX(-groundTexture.size().width * 2.0, y: 0, duration: NSTimeInterval(0.02 *
groundTexture.size().width * 2.0))
        let resetGroundSprite = SKAction.moveByX(groundTexture.size().width * 2.0, y: 0, duration: 0.0)
        let moveGroundSpritesForever = SKAction.repeatActionForever(SKAction.sequence([moveGroundSprite,resetGroundSprite]))

        for var i:CGFloat = 0; i < 2.0 + self.frame.size.width / ( groundTexture.size().width * 2.0 ); ++i {
            let sprite = SKSpriteNode(texture: groundTexture)
            sprite.setScale(2.0)
            sprite.position = CGPoint(x: i * sprite.size.width, y: sprite.size.height / 2.0)
            sprite.runAction(moveGroundSpritesForever)
            moving.addChild(sprite)
        }
    }
```

STEP 6: CREATING THE INTERACTION WITH THE GROUND

- physicsBody initialization creates the physical dimensions of the ground's boundary
- dynamic means that the node will handle physical interactions i.e. forces and impulses
- categoryBitMask assigns a specific category to the ground to handle interaction with the bird later on

```
func createGroundInteraction(groundTexture: SKTexture) {
    var ground = SKNode()
    ground.position = CGPoint(x: 0, y: groundTexture.size().height)
    ground.physicsBody = SKPhysicsBody(rectangleOfSize: CGSize(width: self.frame.size.width, height: groundTexture.size().height * 2.0))
    ground.physicsBody?.dynamic = false
    ground.physicsBody?.categoryBitMask = worldCategory
    self.addChild(ground)
}
```

STEP 7: CREATING THE SKYLINE

- Similar to the ground, the skyline moves alongside the ground in the x-direction
- zPosition is the height of the skyline node relative to its parent (the game scene)

```
func createSkyline(groundTexture: SKTexture) {
    let skyTexture = SKTexture(imageNamed: "sky")
    skyTexture.filteringMode = .Nearest

    let moveSkySprite = SKAction.moveByX(-skyTexture.size().width * 2.0, y: 0, duration: NSTimeInterval(0.1 * skyTexture.size().width * 2.0))
    let resetSkySprite = SKAction.moveByX(skyTexture.size().width * 2.0, y: 0, duration: 0.0)
    let moveSkySpritesForever = SKAction.repeatActionForever(SKAction.sequence([moveSkySprite,resetSkySprite]))

    for var i:CGFloat = 0; i < 2.0 + self.frame.size.width / ( skyTexture.size().width * 2.0 ); ++i {
        let sprite = SKSpriteNode(texture: skyTexture)
        sprite.setScale(2.0)
        sprite.zPosition = -20
        sprite.position = CGPoint(x: i * sprite.size.width, y: sprite.size.height / 2.0 + groundTexture.size().height * 2.0)
        sprite.runAction(moveSkySpritesForever)
        moving.addChild(sprite)
    }
}
```

STEP 8: CREATING THE PIPES

- spawnPipes is an included function in this project that handles the selection of the pipe's height as well as detecting collisions with the bird.

```
func createPipes() {
    // create the pipes textures
    pipeTextureUp = SKTexture(imageNamed: "PipeUp")
    pipeTextureUp.filteringMode = .Nearest
    pipeTextureDown = SKTexture(imageNamed: "PipeDown")
    pipeTextureDown.filteringMode = .Nearest

    // spawn the pipes
    let spawn = SKAction.runBlock({() in self.spawnPipes()})
    let delay = SKAction.waitForDuration(NSTimeInterval(2.0))
    let spawnThenDelay = SKAction.sequence([spawn, delay])
    let spawnThenDelayForever = SKAction.repeatActionForever(spawnThenDelay)
    self.runAction(spawnThenDelayForever)


    // create the pipes movement actions
    let distanceToMove = CGFloat(self.frame.size.width + 2.0 * pipeTextureUp.size().width)
    let movePipes = SKAction.moveByX(-distanceToMove, y:0.0, duration:NSTimeInterval(0.01 * distanceToMove))
    let removePipes = SKAction.removeFromParent()
    movePipesAndRemove = SKAction.sequence([movePipes, removePipes])
}
```

STEP 9: SETTING UP THE BACKGROUND COLOR

- Red, green, blue have weights from 0 to 1
- Common practice is to have a ratio over 255 to set the color scheme
- Alpha is a measure of the opacity from 0 to 1. In this case, 1.0 is completely visible whereas 0.0 would be invisible.

```
func setupBackgroundColor() {
    skyColor = SKColor(red: 81.0/255.0, green: 192.0/255.0, blue: 201.0/255.0, alpha: 1.0)
    self.backgroundColor = skyColor
}
```

STEP 10: SETTING UP THE COLLISION BIT MASKS

- Collision bit masks between two nodes use bit-wise AND operations to indicate whether they can collide with each other

```
    func didBeginContact(contact: SKPhysicsContact) {
        if moving.speed > 0 {
            if ( contact.bodyA.categoryBitMask & scoreCategory ) == scoreCategory || ( contact.bodyB.categoryBitMask & scoreCategory ) == scoreCategory {
                // Bird has contact with score entity
                score++
                scoreLabelNode.text = String(score)

                // Add a little visual feedback for the score increment
                scoreLabelNode.runAction(SKAction.sequence([SKAction.scaleTo(1.5, duration:NSTimeInterval(0.1)), SKAction.scaleTo(1.0, 
duration:NSTimeInterval(0.1))]))
            } else {

                moving.speed = 0

                bird.physicsBody?.collisionBitMask = worldCategory
                bird.runAction(  SKAction.rotateByAngle(CGFloat(M_PI) * CGFloat(bird.position.y) * 0.01, duration:1), completion:{self.bird.speed = 0 })


                // Flash background if contact is detected
                self.removeActionForKey("flash")
                self.runAction(SKAction.sequence([SKAction.repeatAction(SKAction.sequence([SKAction.runBlock({
                    self.backgroundColor = SKColor(red: 1, green: 0, blue: 0, alpha: 1.0)
                }),SKAction.waitForDuration(NSTimeInterval(0.05)), SKAction.runBlock({
                    self.backgroundColor = self.skyColor
                }), SKAction.waitForDuration(NSTimeInterval(0.05))]), count:4), SKAction.runBlock({
                    self.canRestart = true
                })]), withKey: "flash")
            }
        }
    }
```

STEP 11: SETTING UP THE SCORE

- The SKLabelNode initializer passes in the font type
- Position sets up the x and y coordinates of the node
- zPosition is the height of the score node relative to its parent (the game scene)

```
func setUpScore() {
    score = 0
    scoreLabelNode = SKLabelNode(fontNamed:"MarkerFelt-Wide")
    scoreLabelNode.position = CGPoint( x: self.frame.midX, y: 3 * self.frame.size.height / 4 )
    scoreLabelNode.zPosition = 100
    scoreLabelNode.text = String(score)
    self.addChild(scoreLabelNode)
}
```

STEP 12: RESETTING THE SCENE

- For collision bit masks, bitwise OR means that the bird can collide with either the world (ground) or a pipe

```
func resetScene (){
    // Move bird to original position and reset velocity
    bird.position = CGPoint(x: self.frame.size.width / 2.5, y: self.frame.midY)
    bird.physicsBody?.velocity = CGVector( dx: 0, dy: 0 )
    bird.physicsBody?.collisionBitMask = worldCategory | pipeCategory
    bird.speed = 1.0
    bird.zRotation = 0.0

    // Remove all existing pipes
    pipes.removeAllChildren()

    // Reset _canRestart
    canRestart = false

    // Reset score
    score = 0
    scoreLabelNode.text = String(score)

    // Restart animation
    moving.speed = 1
}
```

FIN

- The complete repository is at https://github.com/Darthpwner/iOS-Game-Programming-Complete.git
  - git clone https://github.com/Darthpwner/iOS-Game-Programming-Complete.git
  - If you want the original repo…
    - git clone https://github.com/fullstackio/FlappySwift.git

Apple Inc.

THANK YOU!