```swift
//
//  GameViewController.swift
//  Cannon
//
//  Created by David Hincapie on 11/16/15.
//  Copyright (c) 2015 David Hincapie. All rights reserved.
//

import AVFoundation
import SpriteKit

// sounds defined once and reused throughout app
var blockerHitSound: AVAudioPlayer!
var targetHitSound: AVAudioPlayer!
var cannonFireSound: AVAudioPlayer!

class GameViewController: UIViewController {
    // called when GameViewController is displayed on screen
    override func viewDidLoad() {
        super.viewDidLoad()

        do{
        // load sounds when view controller loads
        blockerHitSound =  try AVAudioPlayer(contentsOfURL:
            NSURL(fileURLWithPath: NSBundle.mainBundle().pathForResource(
                "blocker_hit", ofType: "wav")!), fileTypeHint: nil)
        targetHitSound = try AVAudioPlayer(contentsOfURL:
            NSURL(fileURLWithPath: NSBundle.mainBundle().pathForResource(
                "target_hit", ofType: "wav")!), fileTypeHint: nil)
        cannonFireSound = try AVAudioPlayer(contentsOfURL:
            NSURL(fileURLWithPath: NSBundle.mainBundle().pathForResource(
                "cannon_fire", ofType: "wav")!), fileTypeHint: nil)
        }catch {
            print(error)
        }
        let scene = GameScene(size: view.bounds.size) // create scene
        scene.scaleMode = .AspectFill // resize scene to fit the screen

        let skView = view as! SKView // get GameViewController's SKView
        skView.showsFPS = true // display frames-per-second
        skView.showsNodeCount = true // display # of nodes on screen
        skView.ignoresSiblingOrder = true // for SpriteKit optimizations
        skView.presentScene(scene) // display the scene
    }
}
```

```swift
//
//  GameScene.swift
//  Cannon
//
//  Created by David Hincapie on 11/16/15.
//  Copyright (c) 2015 David Hincapie. All rights reserved.
//

import AVFoundation
import SpriteKit

// used to identify objects for collision detection
struct CollisionCategory {
    static let Blocker : UInt32 = 1
    static let Target: UInt32 = 1 << 1 // 2
    static let Cannonball: UInt32 = 1 << 2 // 4
    static let Wall: UInt32 = 1 << 3 // 8
}

// global because no type constants in Swift classes yet
private let numberOfTargets = 9

class GameScene: SKScene, SKPhysicsContactDelegate {
    // game elements that the scene interacts with programmatically
    private var secondsLabel: SKLabelNode! = nil
    private var cannon: Cannon! = nil

    // game state
    private var timeLeft: CFTimeInterval = 10.0
    private var elapsedTime: CFTimeInterval = 0.0
    private var previousTime: CFTimeInterval = 0.0
    private var targetsRemaining: Int = numberOfTargets

    // called when scene is presented
    override func didMoveToView(view: SKView) {
        self.backgroundColor = SKColor.whiteColor() // set background

        // helps determine game element speeds based on scene size
        var velocityMultiplier = self.size.width / self.size.height

        if UIDevice.currentDevice().userInterfaceIdiom == .Pad {
            velocityMultiplier = CGFloat(velocityMultiplier * 6.0)
        }

        // configure the physicsWorld
        self.physicsWorld.gravity = CGVectorMake(0.0, 0.0) // no gravity
        self.physicsWorld.contactDelegate = self

        // create border for objects colliding with screen edges
        self.physicsBody = SKPhysicsBody(edgeLoopFromRect: self.frame)
        self.physicsBody?.friction = 0.0 // no friction
        self.physicsBody?.categoryBitMask = CollisionCategory.Wall
        self.physicsBody?.contactTestBitMask = CollisionCategory.Cannonball

        createLabels() // display labels at scene's top-left corner

        // create and attach Cannon
        cannon = Cannon(sceneSize: size,
            velocityMultiplier: velocityMultiplier)
        cannon.position = CGPointMake(0.0, self.frame.height / 2.0)
```

```
self.addChild(cannon)

    // create and attach medium Blocker and start moving
    let blockerxPercent = CGFloat(0.5)
    let blockeryPercent = CGFloat(0.25)
    let blocker = Blocker(sceneSize: self.frame.size,
        blockerSize: BlockerSize.Medium)
    blocker.position = CGPointMake(self.frame.width * blockerxPercent,
        self.frame.height * blockeryPercent)
    self.addChild(blocker)
    blocker.startMoving(velocityMultiplier)

    // create and attach targets of random sizes and start moving
    let targetxPercent = CGFloat(0.6) // % across scene to 1st target
    var targetX = size.width * targetxPercent

    for _ in 1 ... numberOfTargets {
        let target = Target(sceneSize: self.frame.size)
        target.position = CGPointMake(targetX, self.frame.height * 0.5)
        targetX += target.size.width + 5.0
        self.addChild(target)
        target.startMoving(velocityMultiplier)
    }
}

// create the text labels
func createLabels() {
    // constants related to displaying text for time remaining
    let edgeDistance = CGFloat(20.0)
    let labelSpacing = CGFloat(5.0)
    let fontSize = CGFloat(16.0)

    // configure "Time remaining: " label
    let timeRemainingLabel = SKLabelNode(fontNamed: "Chalkduster")
    timeRemainingLabel.text = "Time remaining:"
    timeRemainingLabel.fontSize = fontSize
    timeRemainingLabel.fontColor = SKColor.blackColor()
    timeRemainingLabel.horizontalAlignmentMode = .Left
    let y = self.frame.height -
        timeRemainingLabel.fontSize - edgeDistance
    timeRemainingLabel.position = CGPoint(x: edgeDistance, y: y)
    self.addChild(timeRemainingLabel)

    // configure label for displaying time remaining
    secondsLabel = SKLabelNode(fontNamed: "Chalkduster")
    secondsLabel.text = "0.0 seconds"
    secondsLabel.fontSize = fontSize
    secondsLabel.fontColor = SKColor.blackColor()
    secondsLabel.horizontalAlignmentMode = .Left
    let x = timeRemainingLabel.calculateAccumulatedFrame().width +
        edgeDistance + labelSpacing
    secondsLabel.position = CGPoint(x: x, y: y)
    self.addChild(secondsLabel)
}

// test whether an SKPhysicsBody is the cannonball
func isCannonball(body: SKPhysicsBody) -> Bool {
    return body.categoryBitMask & CollisionCategory.Cannonball != 0
}
```

```
// test whether an SKPhysicsBody is a blocker
func isBlocker(body: SKPhysicsBody) -> Bool {
    return body.categoryBitMask & CollisionCategory.Blocker != 0
}

// test whether an SKPhysicsBody is a target
func isTarget(body: SKPhysicsBody) -> Bool {
    return body.categoryBitMask & CollisionCategory.Target != 0
}

// test whether an SKPhysicsBody is a wall
func isWall(body: SKPhysicsBody) -> Bool {
    return body.categoryBitMask & CollisionCategory.Wall != 0
}

// called when collision starts
func didBeginContact(contact: SKPhysicsContact) {
    var cannonball: SKPhysicsBody
    var otherBody: SKPhysicsBody

    // determine which SKPhysicsBody is the cannonball
    if isCannonball(contact.bodyA) {
        cannonball = contact.bodyA
        otherBody = contact.bodyB
    } else {
        cannonball = contact.bodyB
        otherBody = contact.bodyA
    }

    // cannonball hit wall, so remove from screen
    if isWall(otherBody) || isTarget(otherBody) ||
        isBlocker(otherBody) {
            cannon.cannonballOnScreen = false
            cannonball.node?.removeFromParent()
    }

    // cannonball hit blocker, so play blocker sound
    if isBlocker(otherBody) {
        let blocker = otherBody.node as! Blocker
        blocker.playHitSound()
        timeLeft -= blocker.blockerTimePenalty()
    }

    // cannonball hit target
    if isTarget(otherBody) {
        --targetsRemaining
        let target = otherBody.node as! Target
        target.removeFromParent()
        target.playHitSound()
        timeLeft += target.targetTimeBonus()
    }
}

// fire the cannon if there is not a cannonball on screen
override func touchesBegan(touches: Set<UITouch>, withEvent event: UIEvent?) {
    for touch in touches {
        let location = touch.locationInNode(self)
        cannon.rotateToPointAndFire(location, scene: self)
    }
```

```
    }
    }

    // updates to perform in each frame of the animation
    override func update(currentTime: CFTimeInterval) {
        if previousTime == 0.0 {
            previousTime = currentTime
        }

        elapsedTime += (currentTime - previousTime)
        timeLeft -= (currentTime - previousTime)
        previousTime = currentTime

        if timeLeft < 0 {
            timeLeft = 0
        }

        secondsLabel.text = String(format: "%.1f seconds", timeLeft)

        // check whether game is over
        if targetsRemaining == 0 || timeLeft <= 0 {
            runAction(SKAction.runBlock({self.gameOver()}))
        }
    }

    // display the game over scene
    func gameOver() {
        let flipTransition = SKTransition.flipHorizontalWithDuration(1.0)
        let gameOverScene = GameOverScene(size: self.size,
            won: targetsRemaining == 0 ? true : false,
            time: elapsedTime)
        gameOverScene.scaleMode = .AspectFill
        self.view?.presentScene(gameOverScene, transition: flipTransition)
    }
}
```

```swift
//
//  Blocker.swift
//  Cannon
//
//  Created by David Hincapie on 11/16/15.
//  Copyright © 2015 David Hincapie. All rights reserved.
//

import AVFoundation
import SpriteKit

enum BlockerSize: CGFloat {
    case Small = 1.0
    case Medium = 2.0
    case Large = 3.0
}

class Blocker : SKSpriteNode {
    // constants for configuring a blocker
    private let blockerWidthPercent = CGFloat(0.025)
    private let blockerHeightPercent = CGFloat(0.125)
    private let blockerSpeed = CGFloat(5.0)
    private let blockerSize: BlockerSize

    // initializes the Cannon, sizing it based on the scene's size
    init(sceneSize: CGSize, blockerSize: BlockerSize) {
        self.blockerSize = blockerSize
        super.init(
            texture: SKTexture(imageNamed: "blocker"),
            color: UIColor.clearColor(),
            size: CGSizeMake(sceneSize.width * blockerWidthPercent,
                sceneSize.height * blockerHeightPercent *
                    blockerSize.rawValue))

        // set up the blocker's physicsBody
        self.physicsBody =
            SKPhysicsBody(texture: self.texture!, size: self.size)
        self.physicsBody?.friction = 0.0
        self.physicsBody?.restitution = 1.0
        self.physicsBody?.linearDamping = 0.0
        self.physicsBody?.allowsRotation = true
        self.physicsBody?.usesPreciseCollisionDetection = true
        self.physicsBody?.categoryBitMask = CollisionCategory.Blocker
        self.physicsBody?.contactTestBitMask = CollisionCategory.Cannonball
    }

    // not called, but required if subclass defines an init
    required init?(coder aDecoder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    // applies an impulse to the blocker
    func startMoving(velocityMultiplier: CGFloat) {
        self.physicsBody?.applyImpulse(CGVectorMake(0.0,
            velocityMultiplier * blockerSpeed * blockerSize.rawValue))
    }

    // plays the blockerHitSound
    func playHitSound() {
        blockerHitSound.play()
```

```
 }

    // returns time penalty based on blocker size
    func blockerTimePenalty() -> CFTimeInterval {
        return CFTimeInterval(BlockerSize.Small.rawValue)
    }
}
```

```swift
//
//  Target.swift
//  Cannon
//
//  Created by David Hincapie on 11/16/15.
//  Copyright © 2015 David Hincapie. All rights reserved.
//

import SpriteKit
import AVFoundation


// enum of target sizes
enum TargetSize: CGFloat {
    case Small = 1.0
    case Medium = 1.5
    case Large = 2.0
}

// enum of target sprite names
enum TargetColor: String {
    case Red = "target_red"
    case Green = "target_green"
    case Blue = "target_blue"
}

// arrays of enum constants used for random selections;
// global because Swift does not yet support class variables
private let targetColors =
[TargetColor.Red, TargetColor.Green, TargetColor.Blue]
private let targetSizes =
[TargetSize.Small, TargetSize.Medium, TargetSize.Large]

class Target : SKSpriteNode {
    // constants for configuring a blocker
    private let targetWidthPercent = CGFloat(0.025)
    private let targetHeightPercent = CGFloat(0.1)
    private let targetSpeed = CGFloat(2.0)
    private let targetSize: TargetSize
    private let targetColor: TargetColor

    // initializes the Cannon, sizing it based on the scene's size
    init(sceneSize: CGSize) {
        // select random target size and random color
        self.targetSize = targetSizes[
            Int(arc4random_uniform(UInt32(targetSizes.count)))]
        self.targetColor = targetColors[
            Int(arc4random_uniform(UInt32(targetColors.count)))]

        // call SKSpriteNode designated initializer
        super.init(
            texture: SKTexture(imageNamed: targetColor.rawValue),
            color: UIColor.clearColor(),
            size: CGSizeMake(sceneSize.width * targetWidthPercent,
                sceneSize.height * targetHeightPercent *
                    targetSize.rawValue))

        // set up the target's physicsBody
        self.physicsBody =
            SKPhysicsBody(texture: self.texture!, size: self.size)
```

```
self.physicsBody?.friction = 0.0
        self.physicsBody?.restitution = 1.0
        self.physicsBody?.linearDamping = 0.0
        self.physicsBody?.allowsRotation = true
        self.physicsBody?.usesPreciseCollisionDetection = true
        self.physicsBody?.categoryBitMask = CollisionCategory.Target
        self.physicsBody?.contactTestBitMask =
            CollisionCategory.Cannonball
    }

    // not called, but required if subclass defines an init
    required init?(coder aDecoder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    // applies an impulse to the target
    func startMoving(velocityMultiplier: CGFloat) {
        self.physicsBody?.applyImpulse(CGVectorMake(0.0,
            velocityMultiplier * targetSize.rawValue * (targetSpeed +
                CGFloat(arc4random_uniform(UInt32(targetSpeed) + 5)))))
    }

    // plays the targetHitSound
    func playHitSound() {
        targetHitSound.play()
    }

    // returns time bonus based on target size
    func targetTimeBonus() -> CFTimeInterval {
        switch targetSize {
        case .Small:
            return 3.0
        case .Medium:
            return 2.0
        case .Large:
            return 1.0
        }
    }
}
```

```swift
//
//  Cannon.swift
//  Cannon
//
//  Created by David Hincapie on 11/16/15.
//  Copyright © 2015 David Hincapie. All rights reserved.
//

import AVFoundation
import SpriteKit

class Cannon : SKNode {
    // constants
    private let cannonSizePercent = CGFloat(0.15)
    private let cannonballSizePercent = CGFloat(0.075)
    private let cannonBarrelWidthPercent = CGFloat(0.075)
    private let cannonBarrelLengthPercent = CGFloat(0.15)
    private let cannonballSpeed: CGFloat
    private let cannonballSpeedMultiplier = CGFloat(0.25)
    private let barrelLength: CGFloat

    private var barrelAngle = CGFloat(0.0)
    private var cannonball: SKSpriteNode!
    var cannonballOnScreen = false

    // initializes the Cannon, sizing it based on the scene's size
    init(sceneSize: CGSize, velocityMultiplier: CGFloat) {
        cannonballSpeed = cannonballSpeedMultiplier * velocityMultiplier
        barrelLength = sceneSize.height * cannonBarrelLengthPercent
        super.init()

        // configure cannon barrel
        let barrel = SKShapeNode(rectOfSize: CGSizeMake(barrelLength,
            sceneSize.height * cannonBarrelWidthPercent))
        barrel.fillColor = SKColor.blackColor()
        self.addChild(barrel)

        // configure cannon base
        let cannonBase = SKSpriteNode(imageNamed: "base")
        cannonBase.size = CGSizeMake(sceneSize.height * cannonSizePercent,
            sceneSize.height * cannonSizePercent)
        self.addChild(cannonBase)

        // position barrel based on cannonBase
        barrel.position = CGPointMake(cannonBase.size.width / 2.0, 0.0)
    }

    // not called, but required if subclass defines an init
    required init?(coder aDecoder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    // rotate cannon to user's touch point, then fire cannonball
    func rotateToPointAndFire(point: CGPoint, scene: SKScene) {
        // calculate barrel rotation angle
        let deltaX = point.x
        let deltaY = point.y - self.position.y
        barrelAngle = CGFloat(atan2f(Float(deltaY), Float(deltaX)))

        // rotate the cannon barrel to touch point, then fire
```

```
func rotateToPointAndFire(point: CGPoint, scene: SKScene) {
        // calculate barrel rotation angle
        let deltaX = point.x
        let deltaY = point.y - self.position.y
        barrelAngle = CGFloat(atan2f(Float(deltaY), Float(deltaX)))

        // rotate the cannon barrel to touch point, then fire
        let rotateAction = SKAction.rotateToAngle(
            barrelAngle, duration: 0.25, shortestUnitArc: true)

        // perform rotate action, then call fireCannonball
        self.runAction(rotateAction, completion: {
            if !self.cannonballOnScreen {
                self.fireCannonball(scene)
            }
        })
    }

    // create cannonball, attach to scene and start it moving
    private func fireCannonball(scene: SKScene) {
        cannonballOnScreen = true

        // determine starting point for cannonball based on
        // barrelLength and current barrelAngle
        let x = cos(barrelAngle) * barrelLength
        let y = sin(barrelAngle) * barrelLength
        let cannonball = createCannonball(scene.frame.size)
        cannonball.position = CGPointMake(x, self.position.y + y)

        // create based on barrel angle
        let velocityVector =
        CGVectorMake(x * cannonballSpeed, y * cannonballSpeed)

        // put cannonball on screen, move it and play fire sound
        scene.addChild(cannonball)
        cannonball.physicsBody?.applyImpulse(velocityVector)
        cannonFireSound.play()
    }

    // creates the cannonball and configures its physicsBody
    func createCannonball(sceneSize: CGSize) -> SKSpriteNode {
        cannonball = SKSpriteNode(imageNamed: "ball")
        cannonball.size =
            CGSizeMake(sceneSize.height * cannonballSizePercent,
                sceneSize.height * cannonballSizePercent)

        // set up physicsBody
        cannonball.physicsBody =
            SKPhysicsBody(circleOfRadius: cannonball.size.width / 2.0)
        cannonball.physicsBody?.friction = 0.0
        cannonball.physicsBody?.restitution = 1.0
        cannonball.physicsBody?.linearDamping = 0.0
        cannonball.physicsBody?.allowsRotation = true
        cannonball.physicsBody?.usesPreciseCollisionDetection = true
        cannonball.physicsBody?.categoryBitMask =
            CollisionCategory.Cannonball
        cannonball.physicsBody?.contactTestBitMask =
            CollisionCategory.Target | CollisionCategory.Blocker |
            CollisionCategory.Wall
        return cannonball
```

```
    }
}

//
//  GameOverScene.swift
//  Cannon
//
//  Created by David Hincapie on 11/16/15.
//  Copyright © 2015 David Hincapie. All rights reserved.
//

import SpriteKit

class GameOverScene: SKScene {
    // configure GameOverScene
    init(size: CGSize, won: Bool, time: CFTimeInterval) {
        super.init(size: size)
        self.backgroundColor = SKColor.whiteColor()
        let greenColor =
        SKColor(red: 0.0, green: 0.6, blue: 0.0, alpha: 1.0)

        let gameOverLabel = SKLabelNode(fontNamed: "Chalkduster")
        gameOverLabel.text = (won ? "You Win!" : "You Lose")
        gameOverLabel.fontSize = 60
        gameOverLabel.fontColor =
            (won ? greenColor : SKColor.redColor())
        gameOverLabel.position.x = size.width / 2.0
        gameOverLabel.position.y =
            size.height / 2.0 + gameOverLabel.fontSize
        self.addChild(gameOverLabel)

        let elapsedTimeLabel = SKLabelNode(fontNamed: "Chalkduster")
        elapsedTimeLabel.text =
            String(format: "Elapsed Time: %.1f seconds", time)
        elapsedTimeLabel.fontSize = 24
        elapsedTimeLabel.fontColor = SKColor.blackColor()
        elapsedTimeLabel.position.x = size.width / 2.0
        elapsedTimeLabel.position.y = size.height / 2.0
        self.addChild(elapsedTimeLabel)

        let newGameLabel = SKLabelNode(fontNamed: "Chalkduster")
        newGameLabel.text = "Begin New Game"
        newGameLabel.fontSize = 24
        newGameLabel.fontColor = greenColor
        newGameLabel.position.x = size.width / 2.0
        newGameLabel.position.y =
            size.height / 2.0 - gameOverLabel.fontSize
        self.addChild(newGameLabel)
    }

    // not called, but required if you override SKScene's init
    required init?(coder aDecoder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    // present a new GameScene when user touches screen
    override func touchesBegan(touches: Set<UITouch>, withEvent event: UIEvent?) {
        let doorTransition =
        SKTransition.doorsOpenHorizontalWithDuration(1.0)
        let scene = GameScene(size: self.size)
```

```
        scene.scaleMode = .AspectFill
        self.view?.presentScene(scene, transition: doorTransition)
    }
}
```