Lecture 14 Demo Code: Asteroids Dynamic Animation

Objective

Included below is the source code for the demo in lecture. It is provided under the same Creative Commons licensing as the rest of CS193p's course materials. Images for the asteroids and ship can be found here. Grayed out code was not typed in during lecture (it was either dragged in at the start or entered via a code snippet). And here is the complete project.

```
//
//  AsteroidsViewController.swift
//  Asteroids
//
//  Created by CS193p Instructor.
//  Copyright © 2017 Stanford University. All rights reserved.
//

import UIKit

class AsteroidsViewController: UIViewController
{
    private var asteroidField: AsteroidFieldView!
    private var ship: SpaceshipView!

    private lazy var animator: UIDynamicAnimator =
        UIDynamicAnimator(referenceView: self.asteroidField)

    private var asteroidBehavior = AsteroidBehavior()

    // MARK: View Controller Lifecycle

    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)
        initializeIfNeeded()
        animator.addBehavior(asteroidBehavior)
        asteroidBehavior.pushAllAsteroids()
    }

    override func viewWillDisappear(_ animated: Bool) {
        super.viewWillDisappear(animated)
        animator.removeBehavior(asteroidBehavior)
    }

    override func viewDidLayoutSubviews() {
        super.viewDidLayoutSubviews()
        asteroidField?.center = view.bounds.mid
        repositionShip()
    }
```

```
// MARK: Initializing and Positioning

private func initializeIfNeeded() {
    if asteroidField == nil {
        asteroidField = AsteroidFieldView(frame: CGRect(
            center: view.bounds.mid,
            size: view.bounds.size * Constants.asteroidFieldMagnitude
        ))
        view.addSubview(asteroidField)
        let shipSize = view.bounds.size.minEdge * Constants.shipSizeToMinBoundsEdgeRatio
        ship = SpaceshipView(frame: CGRect(squareCenteredAt: asteroidField.center, size: shipSize))
        view.addSubview(ship)
        repositionShip()
        asteroidField.addAsteroids(
            count: Constants.initialAsteroidCount,
            exclusionZone: ship.convert(ship.bounds, to: asteroidField)
        )
        asteroidField.asteroidBehavior = asteroidBehavior
    }
}

private func repositionShip() {
    if asteroidField != nil {
        ship.center = asteroidField.center
        asteroidBehavior.setBoundary(
            ship.shieldBoundary(in: asteroidField),
            named: Constants.shipBoundaryName
        ) {
            [weak self] in
            if let ship = self?.ship {
                if !ship.shieldIsActive {
                    ship.shieldIsActive = true
                    ship.shieldLevel -= Constants.Shield.activationCost
                    Timer.scheduledTimer(withTimeInterval: Constants.Shield.duration, repeats: false)
                    { timer in
                        ship.shieldIsActive = false
                        if ship.shieldLevel == 0 {
                            ship.shieldLevel = 100
                        }
                    }
                }
            }
        }
    }
}

// MARK: Firing Engines

@IBAction func burn(_ sender: UILongPressGestureRecognizer) {
    switch sender.state {
    case .began,.changed:
        ship.direction = (sender.location(in: view) - ship.center).angle
        burn()
    case .ended:
        endBurn()
    default: break
    }
}

private func burn() {
    ship.enginesAreFiring = true
    asteroidBehavior.acceleration.angle = ship.direction - CGFloat.pi
    asteroidBehavior.acceleration.magnitude = Constants.burnAcceleration
}

private func endBurn() {
    ship.enginesAreFiring = false
    asteroidBehavior.acceleration.magnitude = 0
}
```

```
    // MARK: Constants

    private struct Constants {
        static let initialAsteroidCount = 20
        static let shipBoundaryName = "Ship"
        static let shipSizeToMinBoundsEdgeRatio: CGFloat = 1/5
        static let asteroidFieldMagnitude: CGFloat = 10             // as a multiple of view.bounds.size
        static let burnAcceleration: CGFloat = 0.07                 // points/s/s
        struct Shield {
            static let duration: TimeInterval = 1.0       // how long shield stays up
            static let activationCost: Double = 15        // per activation
       }
   }
}
```

```
//
//  AsteroidBehavior.swift
//  Asteroids
//
//  Created by CS193p Instructor.
//  Copyright © 2017 Stanford University. All rights reserved.
//

import UIKit

class AsteroidBehavior: UIDynamicBehavior, UICollisionBehaviorDelegate
{
    // MARK: Child Behaviors

    private lazy var collider: UICollisionBehavior = {
        let behavior = UICollisionBehavior()
        behavior.collisionMode = .boundaries
//        behavior.translatesReferenceBoundsIntoBoundary = true
        behavior.collisionDelegate = self
        return behavior
    }()

    private lazy var physics: UIDynamicItemBehavior = {
        let behavior = UIDynamicItemBehavior()
        behavior.elasticity = 1
        behavior.allowsRotation = true
        behavior.friction = 0
        behavior.resistance = 0
        return behavior
    }()

    lazy var acceleration: UIGravityBehavior = {
        let behavior = UIGravityBehavior()
        behavior.magnitude = 0
        return behavior
    }()

    func pushAllAsteroids(by magnitude: Range<CGFloat> = 0..<0.5) {
        for asteroid in asteroids {
            let pusher = UIPushBehavior(items: [asteroid], mode: .instantaneous)
            pusher.magnitude = CGFloat.random(in: magnitude)
            pusher.angle = CGFloat.random(in: 0..<CGFloat.pi*2)
            pusher.action = { [unowned pusher] in
                pusher.dynamicAnimator?.removeBehavior(pusher)
            }
            addChildBehavior(pusher)
        }
    }
```

```
    // MARK: Adding and Removing Asteroids

    private var asteroids = [AsteroidView]()
    var speedLimit: CGFloat = 300.0

    override init() {
        super.init()
        addChildBehavior(collider)
        addChildBehavior(physics)
        addChildBehavior(acceleration)
        physics.action = { [weak self] in
            for asteroid in self?.asteroids ?? [] {
                let velocity = self!.physics.linearVelocity(for: asteroid)
                let excessHorizontalVelocity = min(self!.speedLimit - velocity.x, 0)
                let excessVerticalVelocity = min(self!.speedLimit - velocity.y, 0)
                self!.physics.addLinearVelocity(
                    CGPoint(x: excessHorizontalVelocity, y: excessVerticalVelocity),
                    for: asteroid
                )
            }
        }
    }

    func addAsteroid(_ asteroid: AsteroidView) {
        asteroids.append(asteroid)
        collider.addItem(asteroid)
        physics.addItem(asteroid)
        acceleration.addItem(asteroid)
        startRecapturingWaywardAsteroids()
    }

    func removeAsteroid(_ asteroid: AsteroidView) {
        if let index = asteroids.index(of: asteroid) {
            asteroids.remove(at: index)
        }
        collider.removeItem(asteroid)
        physics.removeItem(asteroid)
        acceleration.removeItem(asteroid)
        if asteroids.isEmpty {
            stopRecapturingWaywardAsteroids()
        }
    }
```

```
// MARK: Collision Handling

 private var collisionHandlers = [String:(Void)->Void]()

 // set a named UIBezierPath as a boundary for collisions
 // allows providing a closure to invoke when a collision occurs
 func setBoundary(_ path: UIBezierPath?, named name: String, handler: ((Void)->Void)?) {
     collider.removeBoundary(withIdentifier: name as NSString)
     collisionHandlers[name] = nil
     if path != nil {
         collider.addBoundary(withIdentifier: name as NSString, for: path!)
         collisionHandlers[name] = handler
     }
 }

 // UICollisionBehaviorDelegate
 func collisionBehavior(
     _ behavior: UICollisionBehavior,
     beganContactFor item: UIDynamicItem,
     withBoundaryIdentifier identifier: NSCopying?,
     at p: CGPoint
 ) {
     if let name = identifier as? String, let handler = collisionHandlers[name] {
         handler()
     }
 }
```

```
    // MARK: Recapturing Wayward Asteroids

    // inherited from UIDynamicBehavior
    // let's us know when our UIDynamicAnimator changes
    // we need to know so we can stop/start our wayward asteroid recapture
    override func willMove(to dynamicAnimator: UIDynamicAnimator?) {
        super.willMove(to: dynamicAnimator)
        if dynamicAnimator == nil {
            stopRecapturingWaywardAsteroids()
        } else if !asteroids.isEmpty {
            startRecapturingWaywardAsteroids()
        }
    }

    // every 0.5s
    // we look around for asteroids that are
    // outside an asteroid's superview
    // we wrap it around to the other side
    // we take care to notify the animator that we've moved the item
    // using updateItem(usingCurrentState:)

    var recaptureCount = 0
    private weak var recaptureTimer: Timer?

    private func startRecapturingWaywardAsteroids() {
        if recaptureTimer == nil {
            recaptureTimer = Timer.scheduledTimer(withTimeInterval: 0.5, repeats: true) { [weak self] timer in
                for asteroid in self?.asteroids ?? [] {
                    if let asteroidFieldBounds = asteroid.superview?.bounds, !asteroidFieldBounds.contains(asteroid.center) {
                        asteroid.center.x = asteroid.center.x.truncatingRemainder(dividingBy: asteroidFieldBounds.width)
                        if asteroid.center.x < 0 { asteroid.center.x += asteroidFieldBounds.width }
                        asteroid.center.y = asteroid.center.y.truncatingRemainder(dividingBy: asteroidFieldBounds.height)
                        if asteroid.center.y < 0 { asteroid.center.y += asteroidFieldBounds.height }
                        self?.dynamicAnimator?.updateItem(usingCurrentState: asteroid)
                        self?.recaptureCount += 1
                    }
                }
            }
        }
    }

    private func stopRecapturingWaywardAsteroids() {
        recaptureTimer?.invalidate()
    }
}
```

```
//
//  AsteroidFieldView.swift
//  Asteroids
//
//  Created by CS193p Instructor.
//  Copyright © 2017 Stanford University. All rights reserved.
//

import UIKit

class AsteroidFieldView: UIView
{
    // apply this behavior to all asteroids
    var asteroidBehavior: AsteroidBehavior? {
        didSet {
            for asteroid in asteroids {
                oldValue?.removeAsteroid(asteroid)
                asteroidBehavior?.addAsteroid(asteroid)
            }
        }
    }

    // get all of our asteroids
    // by converting our subviews array
    // into an array of all subviews that are AsteroidView
    private var asteroids: [AsteroidView] {
        return subviews.flatMap { $0 as? AsteroidView }
    }

    var scale: CGFloat = 0.002  // size of average asteroid (compared to bounds.size)
    var minAsteroidSize: CGFloat = 0.25 // compared to average
    var maxAsteroidSize: CGFloat = 2.00 // compared to average

    func addAsteroids(count: Int, exclusionZone: CGRect = CGRect.zero) {
        assert(!bounds.isEmpty, "can't add asteroids to an empty field")
        let averageAsteroidSize = bounds.size * scale
        for _ in 0..<count {
            let asteroid = AsteroidView()
            asteroid.frame.size =
                (asteroid.frame.size /
                (asteroid.frame.size.area / averageAsteroidSize.area)) *
                CGFloat.random(in: minAsteroidSize..<maxAsteroidSize)
            repeat {
                asteroid.frame.origin = bounds.randomPoint
            } while !exclusionZone.isEmpty && asteroid.frame.intersects(exclusionZone)
            addSubview(asteroid)
            asteroidBehavior?.addAsteroid(asteroid)
        }
    }
}
```

```
//
//  SpaceshipView.swift
//  Asteroids
//
//  Created by CS193p Instructor.
//  Copyright © 2017 Stanford University. All rights reserved.
//

import UIKit

class SpaceshipView: UIView
{
    // MARK: Public API

    var enginesAreFiring = false { didSet { if !exploding { resetShipImage() } } }
    var direction: CGFloat = 0 { didSet { updateDirection() } }
    var shieldLevel: Double = 100 { didSet { shieldLevel = min(max(shieldLevel, 0), 100); shieldLevelChanged() } }
    var shieldIsActive = false { didSet { setNeedsDisplay() } }

    func shieldBoundary(in view: UIView) -> UIBezierPath { return getShieldPath(in: view) }

    // MARK: Private Implementation

    private struct Constants {
        static let explosionDuration: TimeInterval = 1.5
        static let explosionToFadeRatio: Double = 1/4
        static let shieldActiveLinewidthRatio: CGFloat = 3
        static let shipImage = UIImage(named: "ship")
        static let shipWithEnginesFiringImage = UIImage(named: "shipfiring")
        static let explosionImage = UIImage.animatedImageNamed("explosion", duration: 1.5)
    }

    private var shieldLinewidth: CGFloat = 1.0 { didSet { setNeedsDisplay() } }
    private let imageView = UIImageView(image: Constants.shipImage)

    override init(frame: CGRect) {
        super.init(frame: frame)
        resetShipImage()
    }

    required init?(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)
        resetShipImage()
    }

    private func resetShipImage() {
        imageView.isHidden = (shieldLevel == 0)
        if imageView.superview == nil {
            isOpaque = false
            addSubview(imageView)
        }
        imageView.image = enginesAreFiring ? Constants.shipWithEnginesFiringImage : Constants.shipImage
        updateImageViewFrame()
        updateDirection()
        imageView.alpha = 1
    }

    override func layoutSubviews() {
        super.layoutSubviews()
        updateImageViewFrame()
    }

    private func updateImageViewFrame() {
        if !exploding && imageView.transform == CGAffineTransform.identity {
            imageView.frame = bounds
        }
    }

    private func updateDirection() {
        if !exploding {
            imageView.transform = CGAffineTransform.identity.rotated(by: direction)
        }
    }
```

```
    private func shieldLevelChanged() {
        if !exploding {
            if shieldLevel == 0 && !imageView.isHidden {
                explode()
            } else {
                imageView.isHidden = (shieldLevel == 0)
                setNeedsDisplay()
            }
        }
    }

    // MARK: Drawing

    private var shieldColor: UIColor {
        let red: CGFloat = shieldLevel < 50 ? 1 : 0
        let green: CGFloat = shieldLevel > 25 ? 1 : 0
        return UIColor(red: red, green: green, blue: 0, alpha: 1)
    }

    private func getShieldPath(level: Double = 100, in view: UIView? = nil) -> UIBezierPath {
        var middle = CGPoint(x: bounds.midX, y: bounds.midY)
        if view != nil { middle = self.convert(middle, to: view) }
        let radius = min(bounds.size.width, bounds.size.height) / 2 - shieldLinewidth
        let startAngle = -CGFloat.pi/2
        let endAngle = -CGFloat.pi/2 + CGFloat(level)/100 * CGFloat.pi*2
        let path = UIBezierPath(
            arcCenter: middle, radius: radius,
            startAngle: startAngle, endAngle: endAngle,
            clockwise: true
        )
        path.lineWidth = shieldLinewidth * (shieldIsActive ? Constants.shieldActiveLinewidthRatio : 1)
        return path
    }

    override func draw(_ rect: CGRect) {
        if shieldLevel > 0 && shieldLevel < 100 && !exploding {
            UIColor.lightGray.setStroke()
            getShieldPath().stroke()
            shieldColor.setStroke()
            getShieldPath(level: shieldLevel).stroke()
        }
    }

    // MARK: Exploding

    private var exploding: Bool {
        return imageView.image == Constants.explosionImage
    }

    private func explode() {
        imageView.image = Constants.explosionImage
        imageView.transform = CGAffineTransform.identity
        imageView.startAnimating()
        setNeedsDisplay()

        let smallerFrame = imageView.frame.insetBy(
            dx: imageView.bounds.size.width * 0.30,
            dy: imageView.bounds.size.height * 0.30
        )
        let biggerFrame = imageView.frame.insetBy(
            dx: -imageView.bounds.size.width * 0.15,
            dy: -imageView.bounds.size.height * 0.15
        )
        imageView.frame = smallerFrame
        let explodeTime = Constants.explosionDuration * Constants.explosionToFadeRatio
        UIView.animate(withDuration: explodeTime, animations: { [imageView = self.imageView] in
            imageView.frame = biggerFrame
        }, completion: { [imageView = self.imageView] finished in
            UIView.animate(withDuration: Constants.explosionDuration - explodeTime, animations: {
                imageView.alpha = 0
                imageView.frame = smallerFrame
            }, completion: { finished in
                self.resetShipImage()
            })
        })
    }
}
```

```
//
//  AsteroidView.swift
//  Asteroids
//
//  Created by CS193p Instructor.
//  Copyright © 2017 Stanford University. All rights reserved.
//

import UIKit

class AsteroidView: UIImageView
{
    convenience init() {
        self.init(frame: CGRect.zero)
    }

    override init(frame: CGRect) {
        super.init(frame: frame)
        setup()
    }

    required init?(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)
        setup()
    }

    private func setup() {
        image = UIImage(named: "asteroid\((arc4random()%9)+1)")
        frame.size = image?.size ?? CGSize.zero
    }
}
```

```
//
//  CoreGraphicsExtensions.swift
//  Asteroids
//
//  Created by CS193p Instructor.
//  Copyright © 2017 Stanford University. All rights reserved.
//

import CoreGraphics

extension CGFloat {
    static func random(in range: Range<CGFloat>) -> CGFloat {
        return CGFloat(arc4random())/CGFloat(UInt32.max)*(range.upperBound-range.lowerBound)+range.lowerBound
    }
    static let up = -CGFloat.pi/2
    static let down = CGFloat.pi/2
    static let left = CGFloat.pi
    static let right: CGFloat = 0
}

extension CGSize {
    static func square(_ size: CGFloat) -> CGSize {
        return CGSize(width: size, height: size)
    }

    static func *(_ size: CGSize, by: CGFloat) -> CGSize {
        return CGSize(width: size.width * sqrt(by), height: size.height * sqrt(by))
    }

    static func /(_ size: CGSize, by: CGFloat) -> CGSize {
        return CGSize(width: size.width / sqrt(by), height: size.height / sqrt(by))
    }

    var minEdge: CGFloat { return min(width, height) }

    var area: CGFloat { return width * height }
}

extension CGRect
{
    var mid: CGPoint { return CGPoint(x: midX, y: midY) }

    init(squareCenteredAt center: CGPoint, size: CGFloat) {
        let origin = CGPoint(x: center.x - size / 2, y: center.y - size / 2)
        self.init(origin: origin, size: CGSize.square(size))
    }

    init(center: CGPoint, size: CGSize) {
        self.init(origin: CGPoint(x: center.x-size.width/2, y: center.y-size.height/2), size: size)
    }

    var randomPoint: CGPoint {
        return CGPoint(x: CGFloat.random(in: 0..<size.width), y: CGFloat.random(in: 0..<size.height))
    }
}

extension CGPoint {
    func denormalized(in rect: CGRect) -> CGPoint {
        return CGPoint(x: rect.origin.x + x * rect.size.width, y: rect.origin.y + y * rect.size.height)
    }

    static func -(left: CGPoint, right: CGPoint) -> CGVector {
        return CGVector(dx: left.x-right.x, dy: left.y-right.y)
    }
}

extension CGVector
{
    var angle: CGFloat {
        let angle = atan(abs(dy)/abs(dx))
        if dx > 0 && dy < 0 {
            return -angle
        } else if dx < 0 && dy < 0 {
            return -CGFloat.pi + angle
        } else if dx > 0 && dy > 0 {
            return -CGFloat.pi*2 + angle
        } else if dx < 0 && dy > 0 {
            return -CGFloat.pi - angle
        } else if dx < 0 && dy == 0 {
            return -CGFloat.pi
        } else if dx == 0 && dy < 0 {
            return -CGFloat.pi/2
        } else if dx == 0 && dy > 0 {
            return -3*CGFloat.pi/2
        } else {
            return 0
        }
    }
}
```