SpriteKit

CS193W - Spring 2016 - Lecture 9

SpriteKit

- A framework for creating 2D games on iOS and tvOS
- Can be used in conjunction with UIKit
- There is an analogous framework called SceneKit for creating 3D games

SKTexture

Rather than use `UIImage`s, in `SpriteKit` you use `SKTexture`s.

```
init(imageNamed name: String)
```

This will look for images with the given name in the same way that `UIImage` does. If it does not find an image, it will look for the image in any available *texture atlases*.

You create a texture atlas by creating a folder with an `.atlas` extension in your project and placing images into it.

Using a texture atlas is more efficient than using individual images, both computationally and due to memory usage.

Texture Altases

- When loaded into memory, images are always padded to be a factor of 2 in size. (i.e. 512x512 pixels.)
- In a texture atlas, padding is stripped away and several images are combined into one.
- In addition to the space savings, the renderer can combine passes if images are in the same texture atlas, speeding things up.

Basic Concepts

- A SpriteKit *scene* consists of *nodes*, which represent *sprites* and other game elements
- *Actions* are run on nodes to animate them and otherwise modify them

SKNode and its Subclasses

```
SKNode
  SKSpriteNode
  SKLabelNode
  SKShapeNode
  SKVideoNode
  SKLightNode
  SKCameraNode
```

and few more…

SKSpriteNode

- *sprite* - a computer graphic that may be moved on-screen and otherwise manipulated as a single entity.
- A sprite can be given an appearance via a `SKTexture`

```
init(texture texture: SKTexture?)
```

For convenience, you can create the texture implicitly and just call:

```
init(imageNamed name: String)
```

SKLabelNode

- Can be used to make a node with a single line of text
- Can set the text, font, alignment, color, etc.

SKShapeNode

- Can be used to create nodes that are circles, squares, eclipses, or defined by arbitrary paths
- Lower performance than SKSpriteNode though, so use SKSpriteNodes if you can.

SKVideoNode

- A node that plays a video

SKNode

- The superclass of `SKSpriteNode`, `SKLabelNode`, etc.
- All nodes have the following modifiable properties:
- `position –` the (x,y) position in the parent node's coordinate system

  `zPosition –` the z position in the parent node's coordinate system (higher z-values are on top of lower ones)

  `xScale –` a multiplier to the node's width
  `yScale –` a multiplier to the node's height

  `zRotation –` a rotation angle (in radians)

  `alpha –` the transparency of the node

  `hidden –` true / false

Grouping with SKNode

- SKNode has no visual rendering, but can often be used to group together child nodes
- e.g. an avatar might be composed of several sprite nodes (body, head, weapon, etc.) all of which are children of the same SKNode

Nodes and their Children

```
func addChild(_ node: SKNode )

func removeFromParent()

func removeAllChildren()


var parent: SKNode ? { get }

var children: [ SKNode ] { get }
```

Node Names

- Nodes can be assigned `name`s. The names can be unique or not.

- You can use `childNodeWithName` or `enumerateChildNodesWithName(_:usingBlock:)` to access the child(ren) with a given name

SKAction

- Actions can be run by nodes to change their properties
- For example:

```
class func scaleBy(_ scale: CGFloat,
          duration sec: NSTimeInterval) -> SKAction
```

is used to animate the scale of a node over a number of seconds.

A sampling of SKActions

```
moveBy(_:duration:)
moveTo(_:duration:)

rotateByAngle(_:duration:)
rotateToAngle(_:duration:)

scaleBy(_:duration:)
scaleTo(_:duration:)

unhide()
hide()

fadeInWithDuration(_:)
fadeOutWithDuration(_:)
```

Reversing Actions

```
func reversedAction() -> SKAction
```

Note: not all actions can be reserved, see the documentation

Repeating Actions

You can run an action multiple times or forever

```
class func repeatAction(_ action: SKAction,
                  count count: Int) -> SKAction

class func repeatActionForever(_ action: SKAction ) -> SKAction
```

Sequencing Actions

You can create a composite action composed of executing several actions in sequence

```
class func sequence(_ actions: [ SKAction ]) -> SKAction
```

To pause between actions create a wait action:

```
class func waitForDuration(_ sec: NSTimeInterval) ->
SKAction
```

Grouping Actions

You can also run actions in parallel by creating groups:

```
class func group(_ actions: [ SKAction ]) -> SKAction
```

Custom Actions

You can run arbitrary code as part of an action:

```
class func runBlock(_ block: dispatch_block_t ) -> SKAction
```

SKScene

- Controls the rendering of the graphics in the SKView that presented the `SKScene`
- An `SKScene` consists of `SKNode`s, of which the SKScene is the root node.
- You subclass `SKScene` to create new scenes

SKScene Loop
(Once per frame)

1 The scene calls its `update:` method.

2 The scene executes actions on its children.

3 The scene calls its `didEvaluateActions` method.

4 The scene executes any physics simulations on physics bodies in the scene.

5 The scene calls its `didSimulatePhysics` method.

6 The scene applies any constraints associated with nodes in the scene.

7 The scene calls its `didApplyConstraints` method.

8 The scene calls its `didFinishUpdate` method.

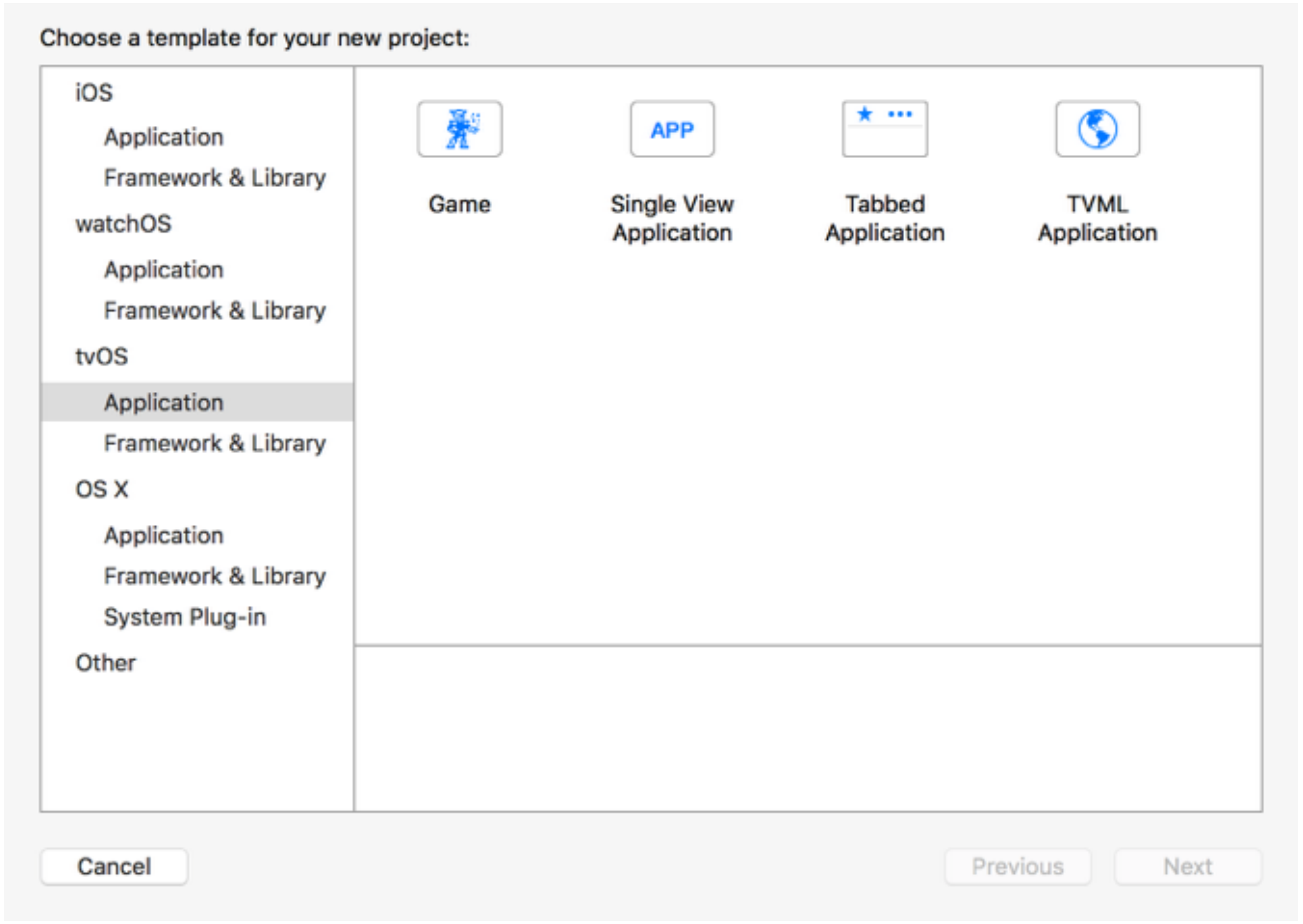
9 The scene renders all of its nodes and updates the view to display the new contents

SKView

- A subclass of UIView
- Has a bunch of properties, but we'll ignore these for now
- All we care about it is the method `presentScene`, which takes a `SKScene`

Creating a SpriteKit Project

File -> New -> Project…



Autogenerated Code

```swift
class GameViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        if let scene = GameScene(fileNamed: "GameScene") {
            // Configure the view.
            let skView = self.view as! SKView
            skView.showsFPS = true
            skView.showsNodeCount = true

            /* Sprite Kit applies additional optimizations to improve rendering performance */
            skView.ignoresSiblingOrder = true

            /* Set the scale mode to scale to fit the window */
            scene.scaleMode = .AspectFill

            skView.presentScene(scene)
        }
    }
```

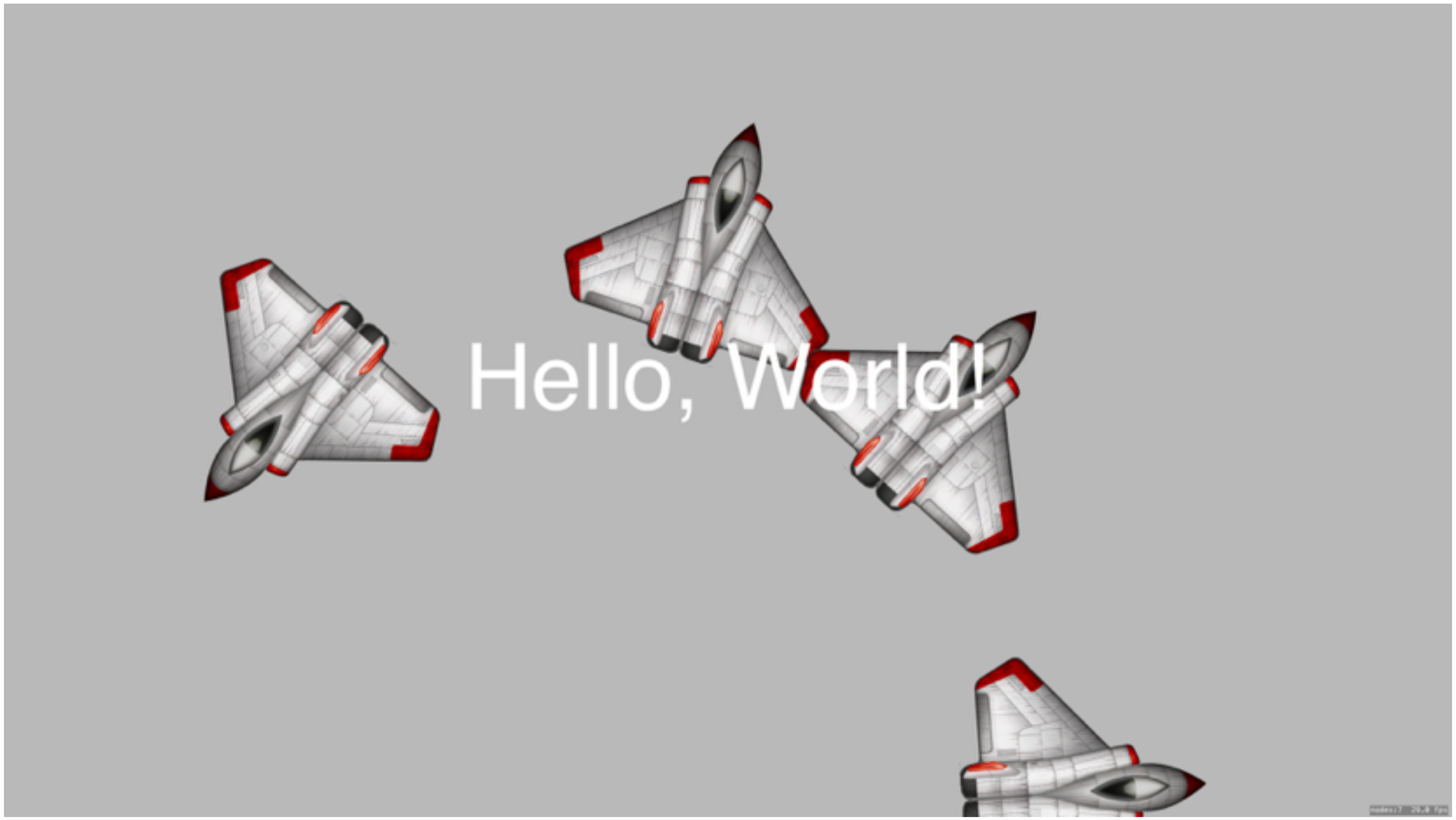
More Autogenerated Code

```
class GameScene: SKScene {
    override func didMoveToView(view: SKView) {
        /* Setup your scene here */
        let myLabel = SKLabelNode(fontNamed:"Chalkduster")
        myLabel.text = "Hello, World!"
        myLabel.fontSize = 65
        myLabel.position = CGPoint(x:CGRectGetMidX(self.frame), y:CGRectGetMidY(self.frame))

        self.addChild(myLabel)
    }

    override func update(currentTime: CFTimeInterval) {
        /* Called before each frame is rendered */
    }

    override func touchesBegan(touches: Set<UITouch>, withEvent event: UIEvent?) {
        /* Called when a touch begins */

        for touch in touches {
            let location = touch.locationInNode(self)

            let sprite = SKSpriteNode(imageNamed:"Spaceship")

            sprite.xScale = 0.5
            sprite.yScale = 0.5
            sprite.position = location

            let action = SKAction.rotateByAngle(CGFloat(M_PI), duration:1)

            sprite.runAction(SKAction.repeatActionForever(action))

            self.addChild(sprite)
        }
    }

}
```

Let’s change touchesBegan to touchesEnded

```swift
class GameScene: SKScene {
    override func didMoveToView(view: SKView) {
        /* Setup your scene here */
        let myLabel = SKLabelNode(fontNamed:"Chalkduster")
        myLabel.text = "Hello, World!"
        myLabel.fontSize = 65
        myLabel.position = CGPoint(x:CGRectGetMidX(self.frame), y:CGRectGetMidY(self.frame))
        myLabel.name = "helloLabel"

        self.addChild(myLabel)
    }

    override func update(currentTime: CFTimeInterval) {
        /* Called before each frame is rendered */
    }

    override func touchesEnded(touches: Set<UITouch>, withEvent event: UIEvent?) {
        /* Called when a touch ends */

        for touch in touches {
            let location = touch.locationInNode(self)

            let sprite = SKSpriteNode(imageNamed:"Spaceship")

            sprite.xScale = 0.5
            sprite.yScale = 0.5
            sprite.position = location

            let action = SKAction.rotateByAngle(CGFloat(M_PI), duration:1)

            sprite.runAction(SKAction.repeatActionForever(action))

            self.addChild(sprite)
        }
    }

}
```

After a few touches



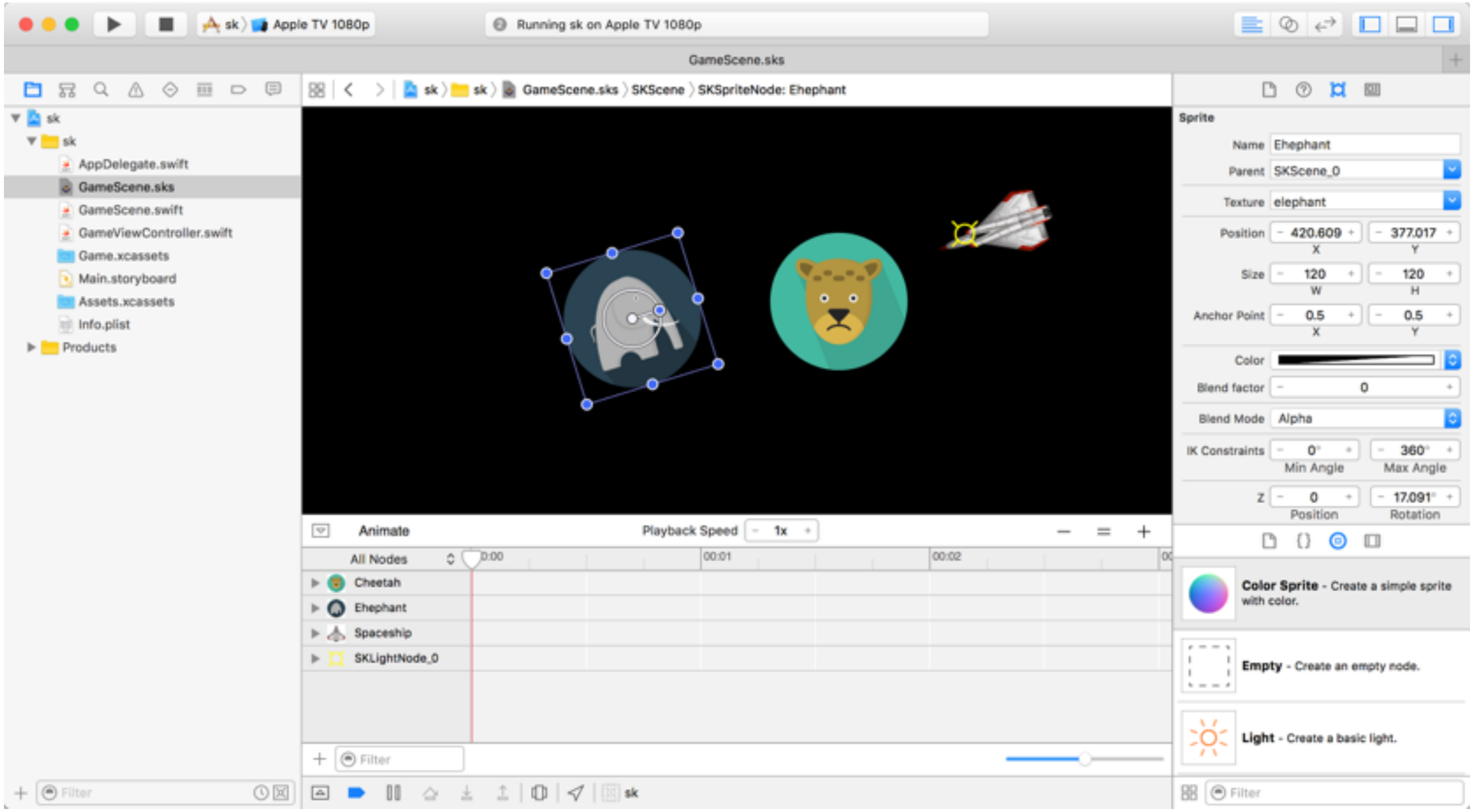
The SpriteKit Scene Editor

- Recall the line:

  ```
  scene = GameScene(fileNamed: “GameScene”)
  ```

- This refers to a file called GameScene.sks
- Great for creating levels where the positioning of the objects and bad guys changes from level to level

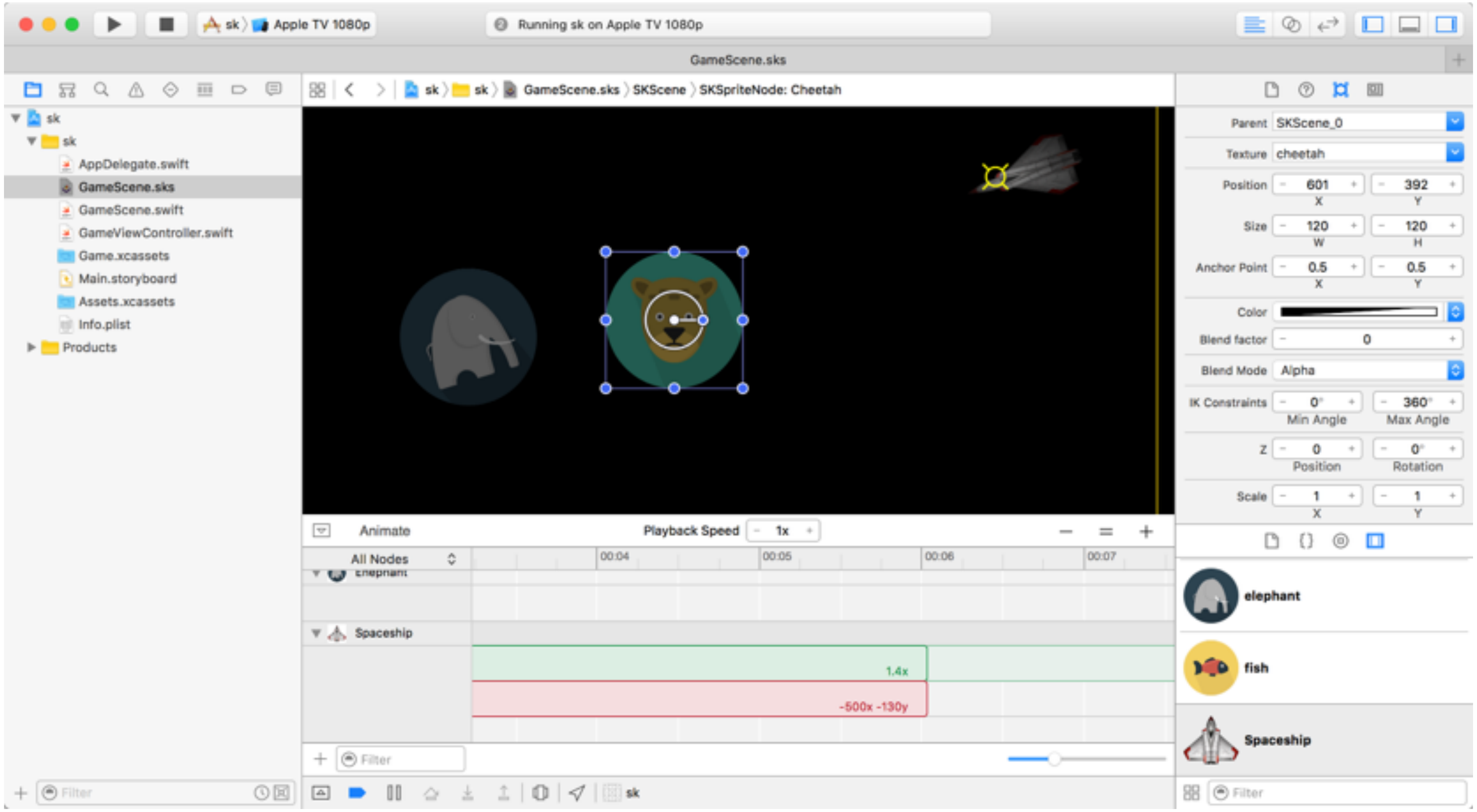
Using the Editor



Sprite

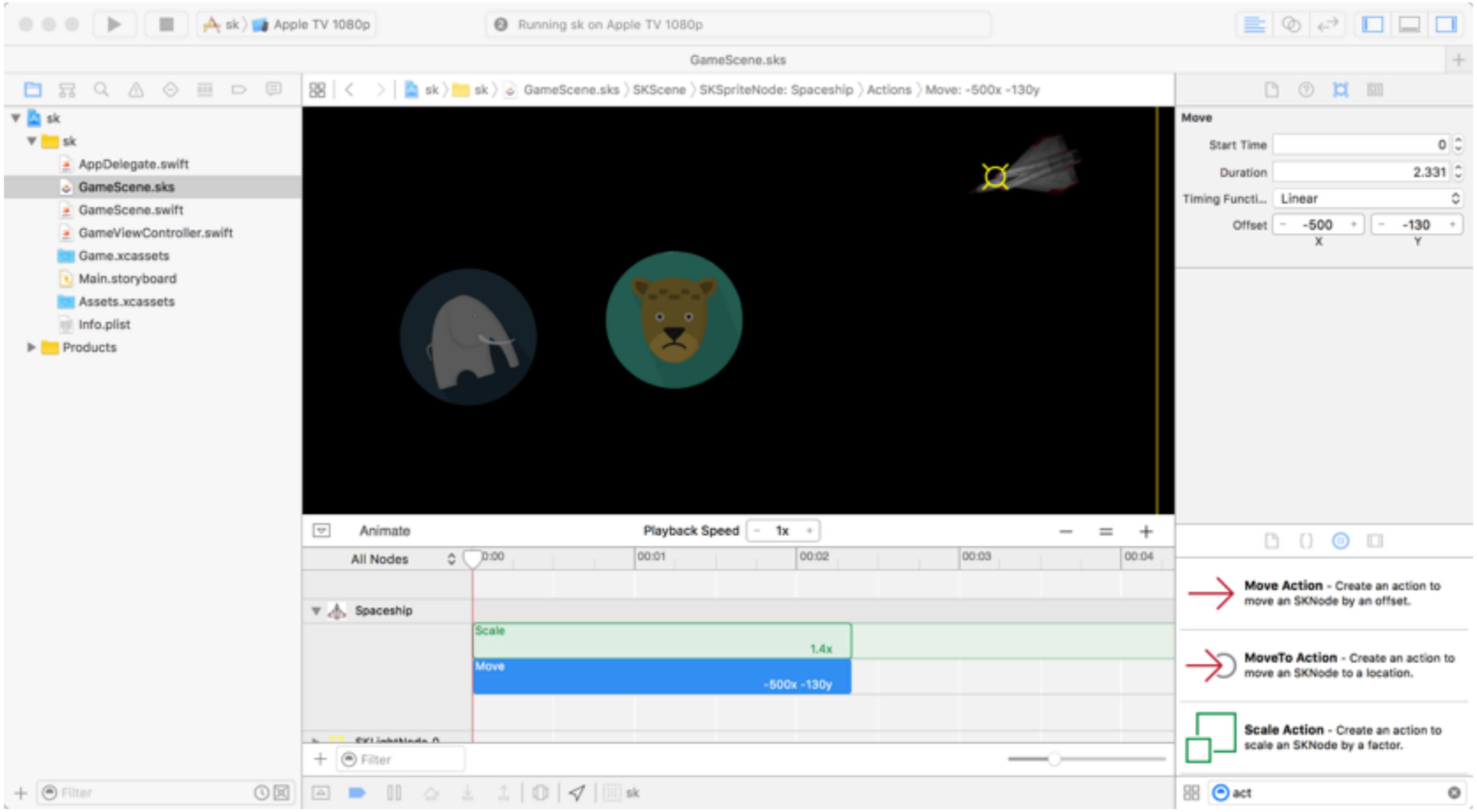
Name: Ehephant
Parent: SKScene_0
Texture: elephant
Position: 420.609 (X), 377.017 (Y)
Size: 120 (W), 120 (H)
Anchor Point: 0.5 (X), 0.5 (Y)
Color
Blend factor: 0
Blend Mode: Alpha
IK Constraints: 0° (Min Angle), 360° (Max Angle)
Z: 0 (Position), 17.091° (Rotation)

Using the Editor



Parent SKScene_0
Texture cheetah
Position 601 X 392 Y
Size 120 W 120 H
Anchor Point 0.5 X 0.5 Y
Color
Blend factor 0
Blend Mode Alpha
IK Constraints 0° Min Angle 360° Max Angle
Z 0 Position 0° Rotation
Scale 1 X 1 Y

Using the Editor



Move

Start Time 0

Duration 2.331

Timing Functi... Linear

Offset -500 X -130 Y

Playback Speed 1x

Move Action - Create an action to move an SKNode by an offset.

MoveTo Action - Create an action to move an SKNode to a location.

Scale Action - Create an action to scale an SKNode by a factor.

Simulating Physics

- In addition to applying actions to nodes, you can define physical characteristics of nodes and simulate their interactions
- Nodes can have shape, mass, density, velocity, etc.

SKPhysicsBody

- To give a node physical properties, assign its `physicsBody` property a `SKPhysicsBody` object

- Physics bodies are `dynamic` by default, meaning they are affected by the physical simulation. Static bodies (i.e. `dynamic = false`) are stationary but do interact with dynamic bodies. Good for e.g. walls in a maze.

- There are two types of Physics bodies - volumes and edges. Edges are static and are infinitely thin.

Defining Physics Bodies

`Volume-based bodies`

```
    init(circleOfRadius:)

    init(rectangleOfSize:)

    init(polygonFromPath:)
```

`Edge-based bodies`

```
    init(edgeLoopFromRect:)

    init(edgeFromPoint:toPoint:)

    init(edgeLoopFromPath:)

    init(edgeChainFromPath:)
```

Making the Screen Edge a Physical Boundary

```
[SKPhysicsBody bodyWithEdgeLoopFromRect:self.frame];
```

Physical Properties

```
var mass: CGFloat
```

The mass of the body in Kilograms. The default is the area of the object times the density.

```
var density: CGFloat
```

The density of the object in Kilograms per square meters. The default is 1.0.

```
var friction: CGFloat
```

A value between 0 and 1, used to apply a frictional force to objects that are in contact with the body. The default is 0.2.

```
var restitution: CGFloat
```

A value between 0 and 1, used to determine how much energy the body loses when it bounces off another object. The default is 0.2.

```
var linearDamping: CGFloat
```

A value between 0 and 1, used to simulate air or fluid resistance. The default is 0.1.

Some Important Properties

```swift
var affectedByGravity: Bool

var allowsRotation: Bool

var dynamic: Bool
```

Applying Force

```
func applyForce(_ force: CGVector)
```

Applies force in both the x and y directions.

```
func applyTorque(_ torque: CGFloat )
```

Applies torque (rotational velocity).

Contacts and Collisions

- When two physics bodies touch, they can either *collide* (and interact) with each other and/or trigger a *contact* (and create an event)

- You specify groups of physics bodies and specify which bodies can contact / collide with other bodies

Bitmasks

- You can define up to 32 categories of objects using bit masks

```
var categoryBitMask: UInt32
```

The categories of this SKPhysicsNode (default 0xFFFFFFFF)

```
var collisionBitMask: UInt32
```

The categories this body can collide with (default 0xFFFFFFFF)

```
var contactTestBitMask: UInt32
```

The categories this body can contact (default 0x00000000)

Contact Callbacks

- Assign the `physicsWorld.contactDelegate` property of an `SKScene` object to a `SKPhysicsContactDelegate`.
- Then implement the callbacks:

```
didBeginContact(_ contact: SKPhysicsContact)
didEndContact(_ contact: SKPhysicsContact)
```

That’s the basics

- There’s more to know of course, but that should be enough to get you started!