THE UNIVERSITY OF CHICAGO

ADVANCED iOS APPLICATION DEVELOPMENT

MPCS 51032 • SPRING 2020 • SESSION 6

PHYSICS

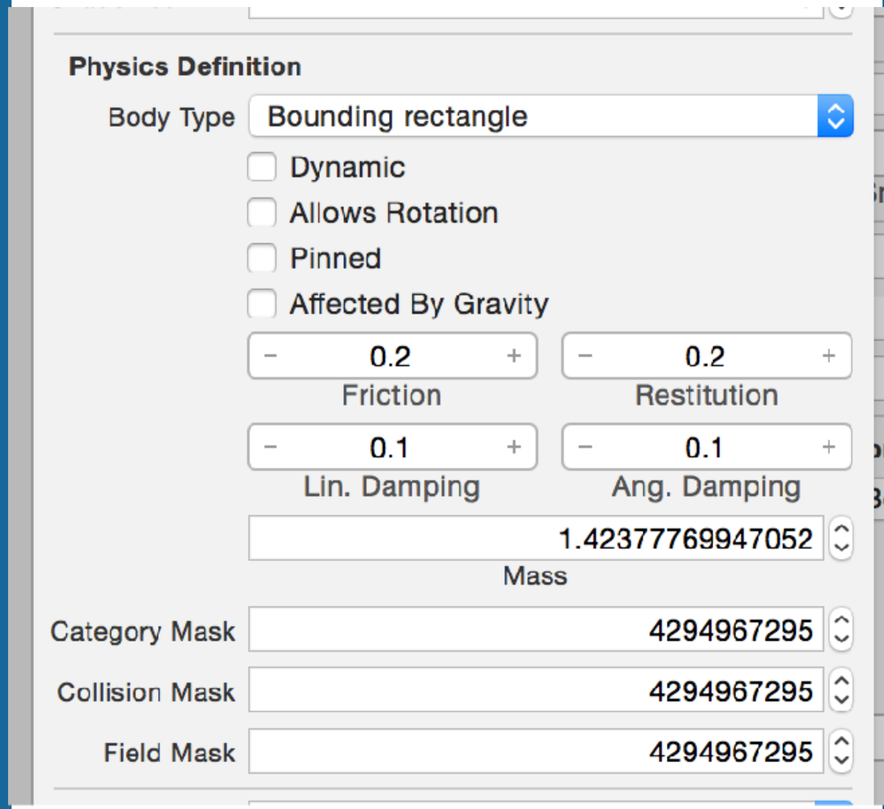
SUBTITLE

- Each scene has a `physicsWorld` property that is an instance of `SKPhysicsWorld` class
- Defines the rules of the system
  - It can be dynamic depending on the game play
  - For example, slow down gravity during a bonus round

```
self.path = CGPathCreateWithEllipseInRect(CGRectMake(
self.position = position
self.fillColor = UIColor.blueColor()
self.strokeColor = UIColor.blueColor()

self.physicsBody = SKPhysicsBody(circleOfRadius: radi
self.physicsBody!.allowsRotation = false
self.physicsBody!.friction = 0
self.physicsBody!.restitution = 1
self.physicsBody!.linearDamping = 0
self.physicsBody!.angularDamping = 0


let numberLabel = SKLabelNode(fontNamed: "Avenir-Next
numberLabel.fontSize = CGFloat(circumference - 1)
numberLabel.verticalAlignmentMode = .Center
numberLabel.horizontalAlignmentMode = .Center
numberLabel.text = "\(number)"


self.addChild(numberLabel)

let shrink = SKAction.scaleTo(0.0, duration: 0.0)
let grow = SKAction.scaleTo(1.0, duration: 0.5)
let tink = SKAction.playSoundFileNamed("Tink.caf", wa
self.runAction(SKAction.sequence([tink,shrink,grow]))
```

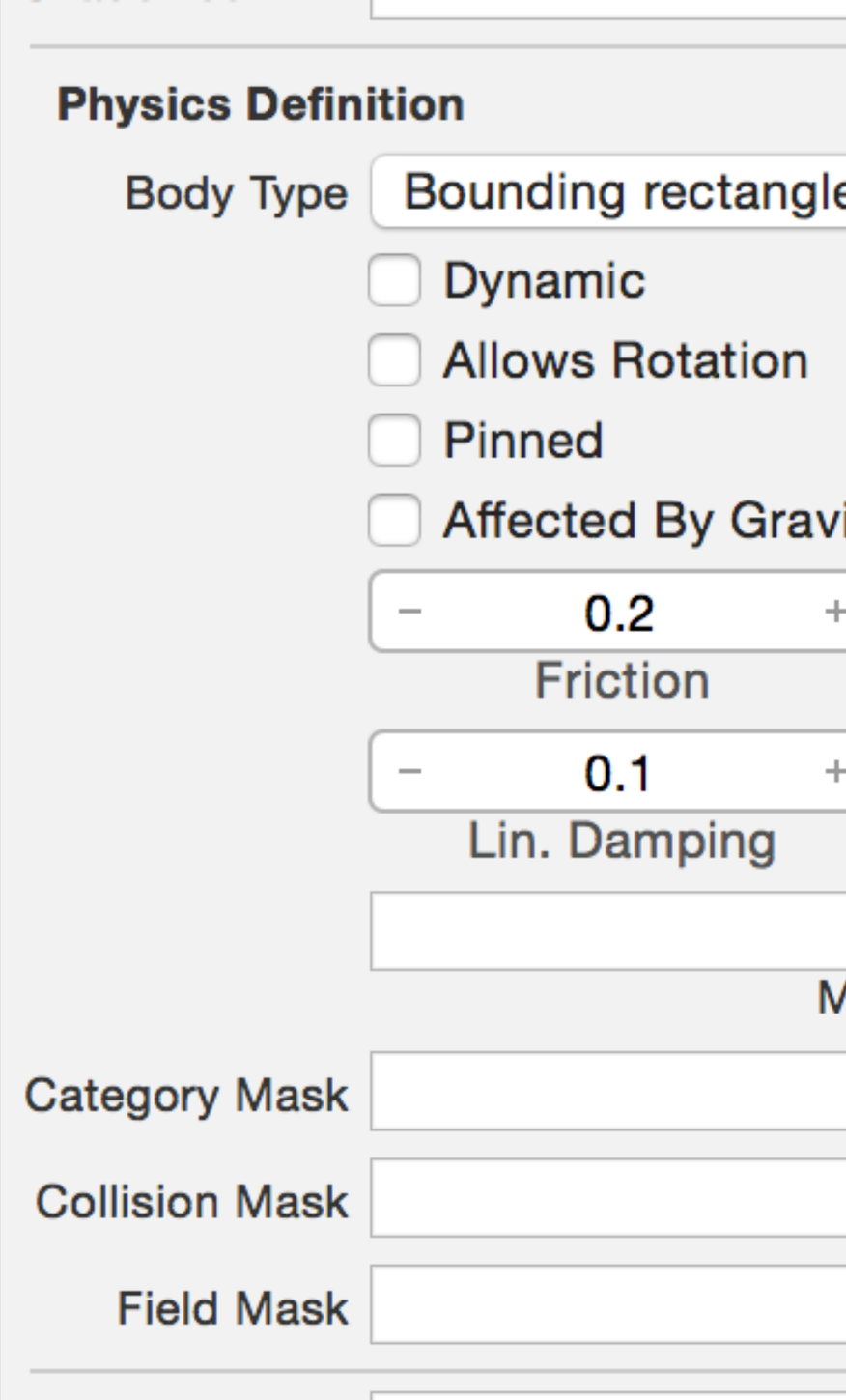
PHYSICS

- Each node within a scene has the option to have associated with it a physics body
- Physics bodies are represented by the SKPhysicsBody class
  - Mass, density, friction
- Not every node in scene must respond to physics

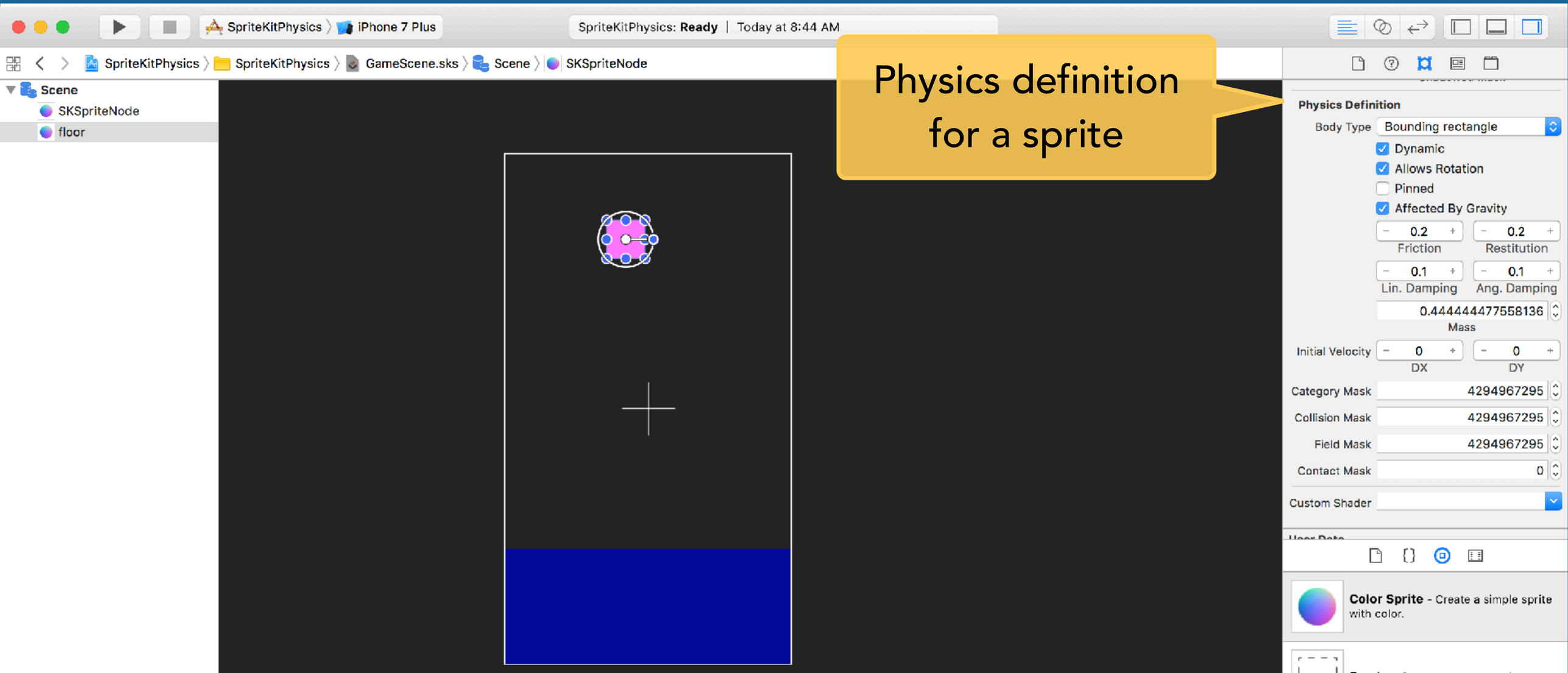


PHYSICS

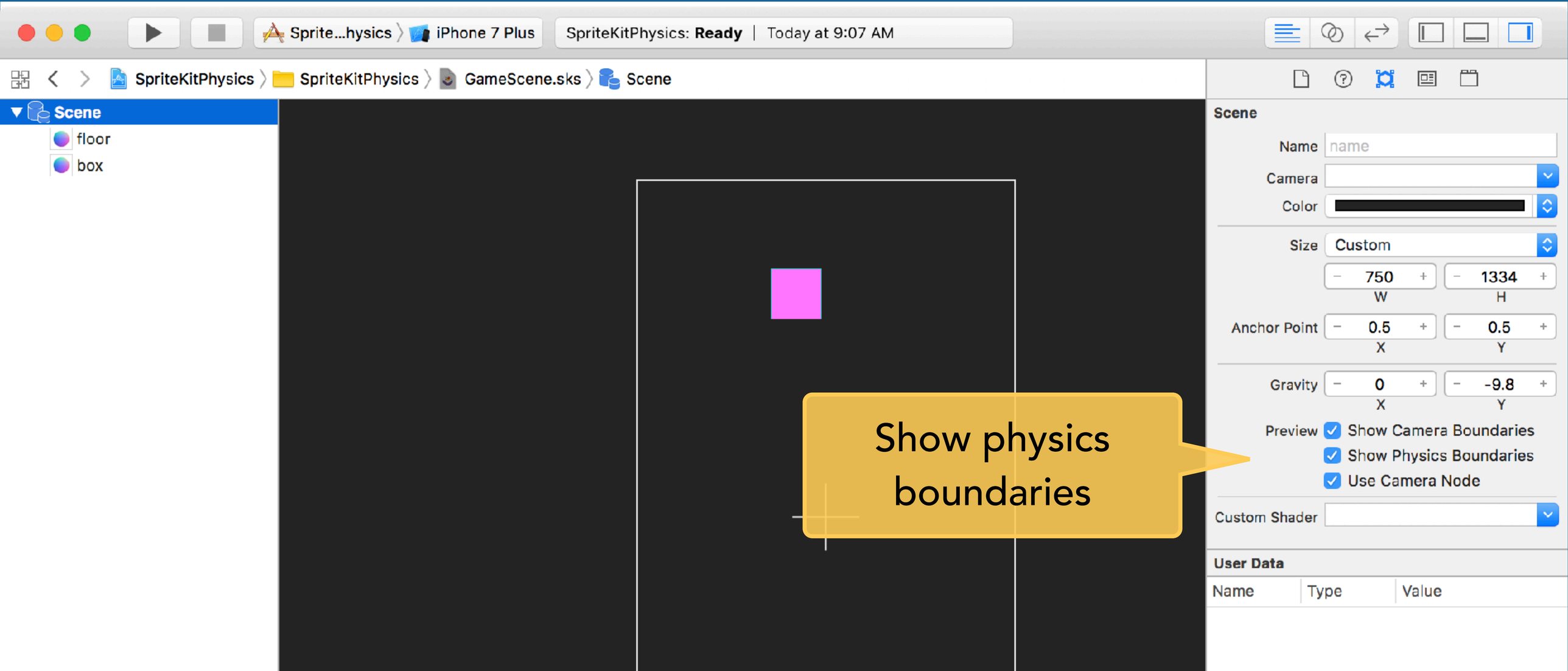
- Types of physics bodies
  - Dynamic volume
    - Can be moved and collide
  - Static volume
    - Can take part in collisions but doesn’t move
  - Edge
    - Boundaries



PHYSICS


SpriteKitPhysics 〉 iPhone 7 Plus
SpriteKitPhysics: Ready | Today at 8:44 AM
SpriteKitPhysics 〉 SpriteKitPhysics 〉 GameScene.sks 〉 Scene 〉 SKSpriteNode
Scene
SKSpriteNode
floor
Physics definition for a sprite
Physics Definition
Body Type
Bounding rectangle
Dynamic
Allows Rotation
Pinned
Affected By Gravity
0.2
Friction
0.2
Restitution
0.1
Lin. Damping
0.1
Ang. Damping
0.444444477558136
Mass
Initial Velocity
0
DX
0
DY
Category Mask
4294967295
Collision Mask
4294967295
Field Mask
4294967295
Contact Mask
0
Custom Shader
Color Sprite - Create a simple sprite with color.

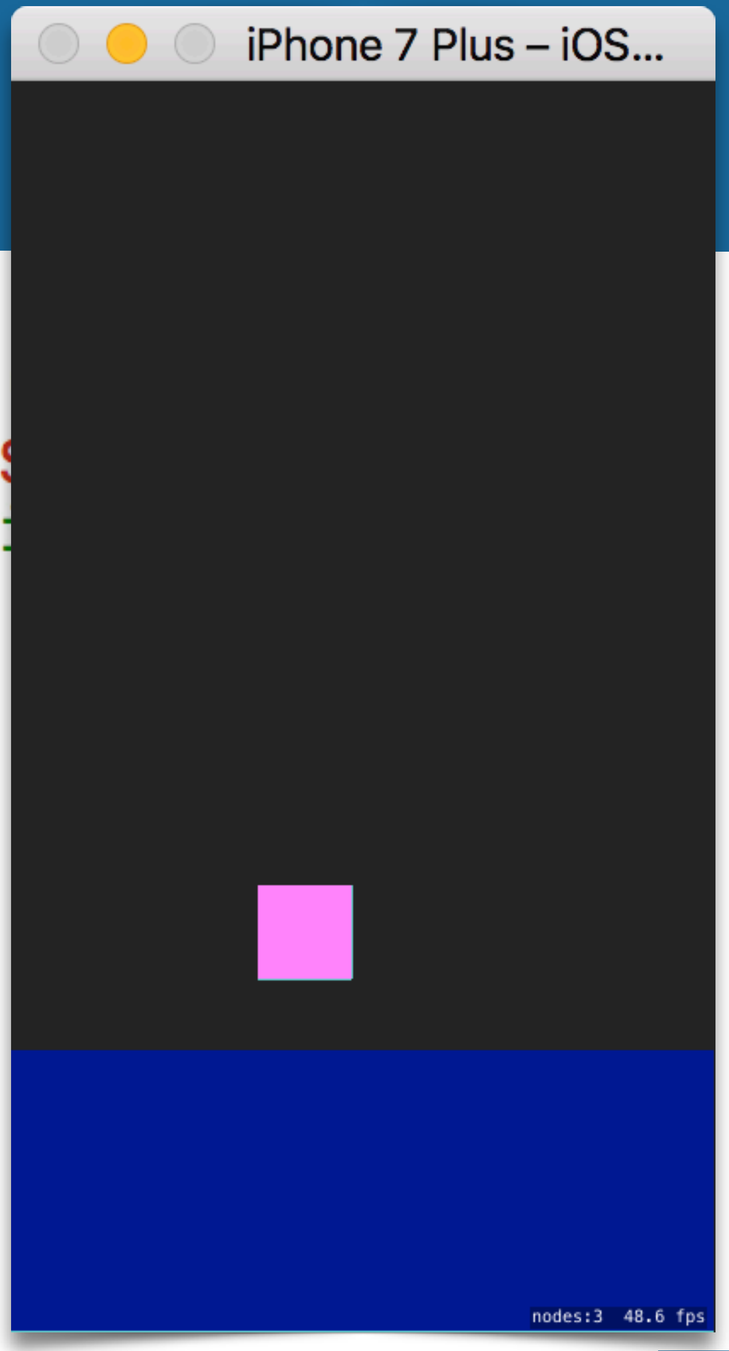
PHYSICS


Sprite...hysics
iPhone 7 Plus
SpriteKitPhysics: Ready | Today at 9:07 AM
SpriteKitPhysics
SpriteKitPhysics
GameSceneSks
Scene
Scene
floor
box
Show physics boundaries
Scene
Name
name
Camera
Color
Size
Custom
750
W
1334
H
Anchor Point
0.5
X
0.5
Y
Gravity
0
X
-9.8
Y
Preview
Show Camera Boundaries
Show Physics Boundaries
Use Camera Node
Custom Shader
User Data
Name
Type
Value

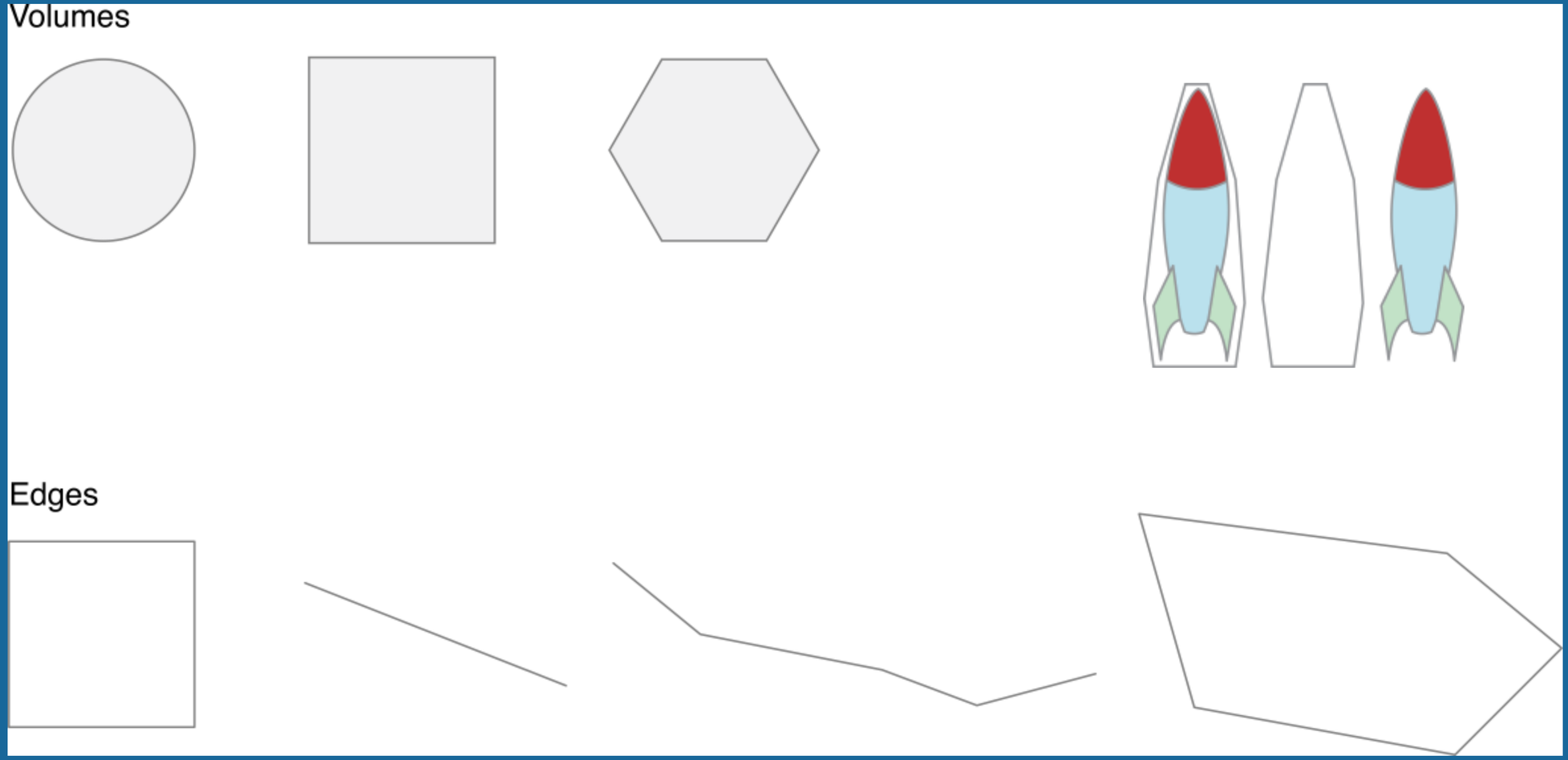
PHYSICS

```
if let view = self.view as! SKView? {
    // Load the SKScene from 'GameScene.sks
    if let scene = SKScene(fileNamed: "Game
        // Set the scale mode to scale to f
        scene.scaleMode = .aspectFill

        // Present the scene
        view.presentScene(scene)

    view.ignoresSiblingOrder = true
    view.showsPhysics = true
    view.showsFPS = true
    view.showsNodeCount = true
}
```

Show physics boundaries in simulator

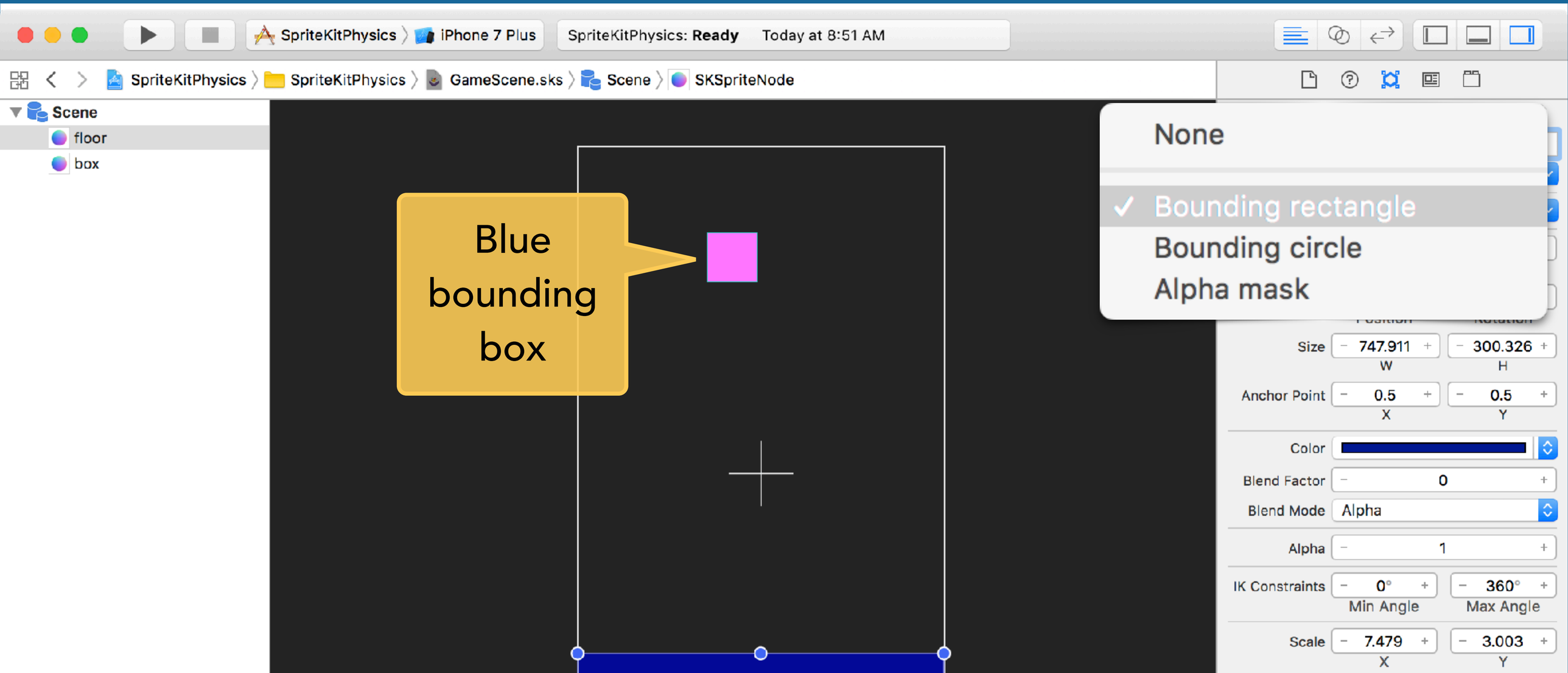


PHYSICS

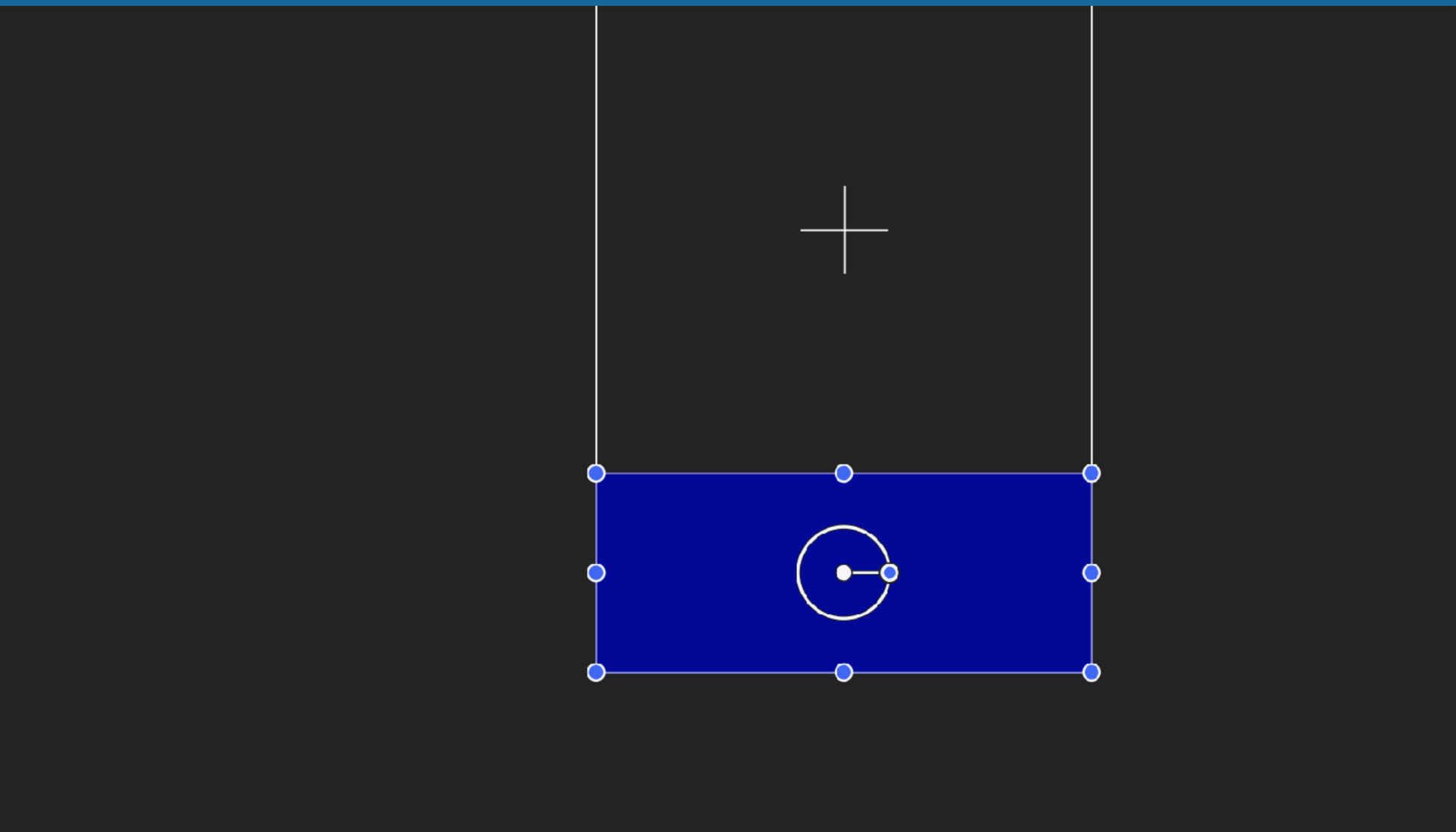


- Physics bodies can be standard shapes or paths (simpler is more efficient)

PHYSICS


SpriteKitPhysics 〉 iPhone 7 Plus
SpriteKitPhysics: Ready Today at 8:51 AM
SpriteKitPhysics 〉 SpriteKitPhysics 〉 GameScene.sks 〉 Scene 〉 SKSpriteNode
Scene
floor
box
Blue bounding box
None
Bounding rectangle
Bounding circle
Alpha mask
Size
747.911
W
300.326
H
Anchor Point
0.5
X
0.5
Y
Color
Blend Factor
0
Blend Mode
Alpha
Alpha
1
IK Constraints
0°
Min Angle
360°
Max Angle
Scale
7.479
X
3.003
Y

PHYSICS



0
Shadowed Mask

Physics Definition

Body Type: Bounding rectangle

- [ ] Dynamic
- [ ] Allows Rotation
- [ ] Pinned
- [ ] Affected By Gravity

Friction: 0.2 — Restitution: 0.2

Lin. Damping: 0.1 — Ang. Damping: 0.1

Mass: 9.98297309875488

Initial Velocity: DX: 0 — DY: 0

Category Mask: 4294967295

Collision Mask: 4294967295

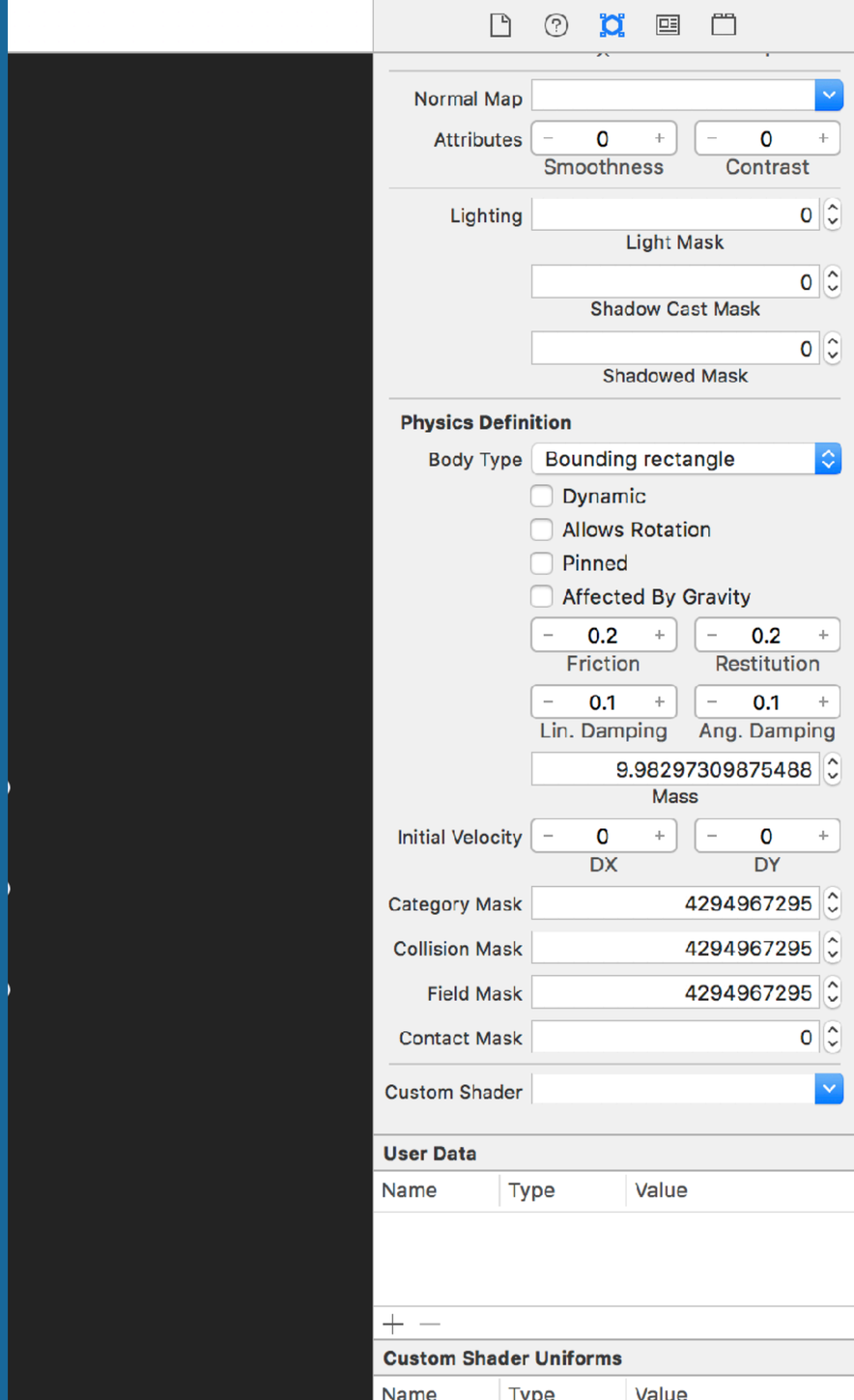
Field Mask: 4294967295

Contact Mask: 0

Custom Shader:

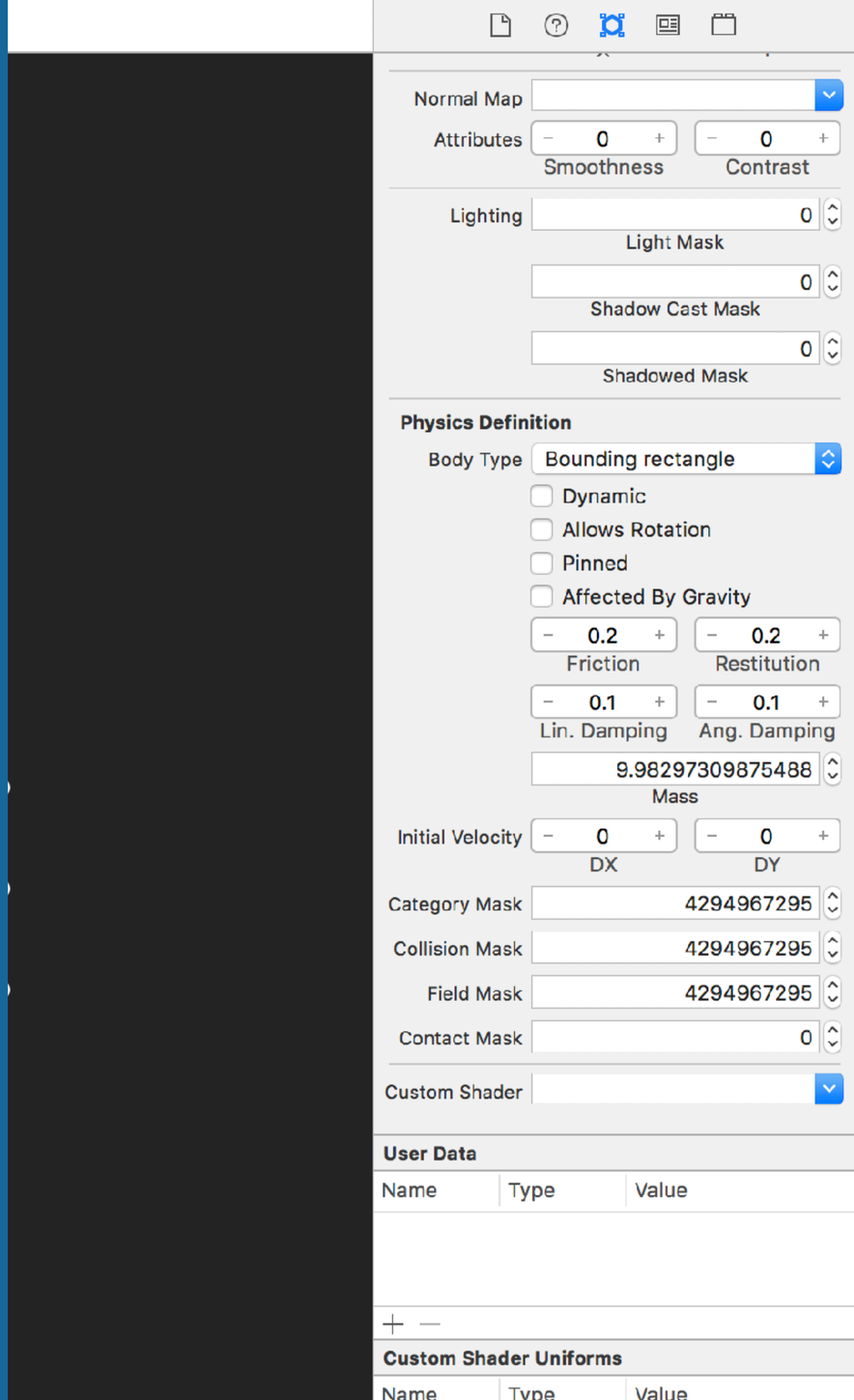
PHYSICS

- Physics node properties
  - Mass - how force and momentum affect body
  - Friction - body surface roughness
  - Linear damping - friction on body
  - Angular damping - friction on body
  - Restitution - energy maintained during a collision (bouncy-ness)

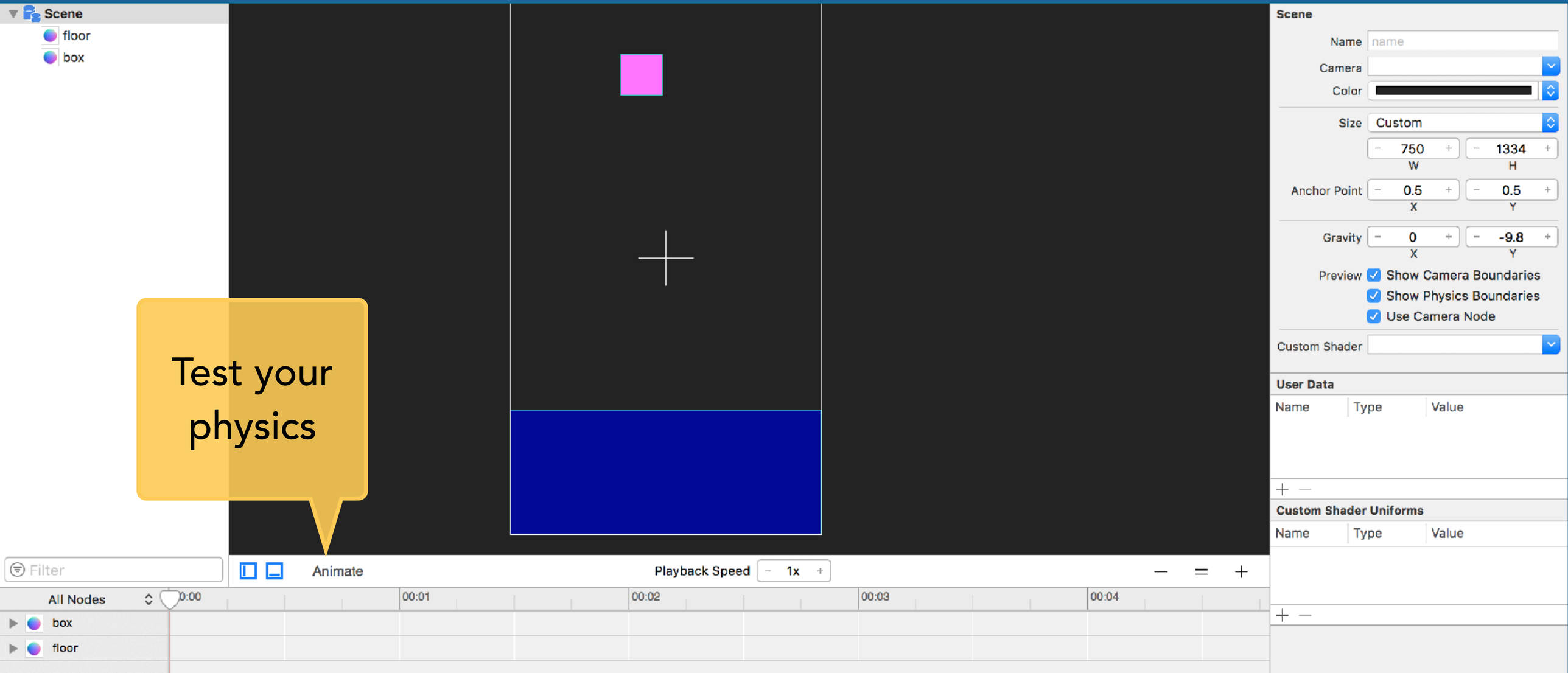


PHYSICS

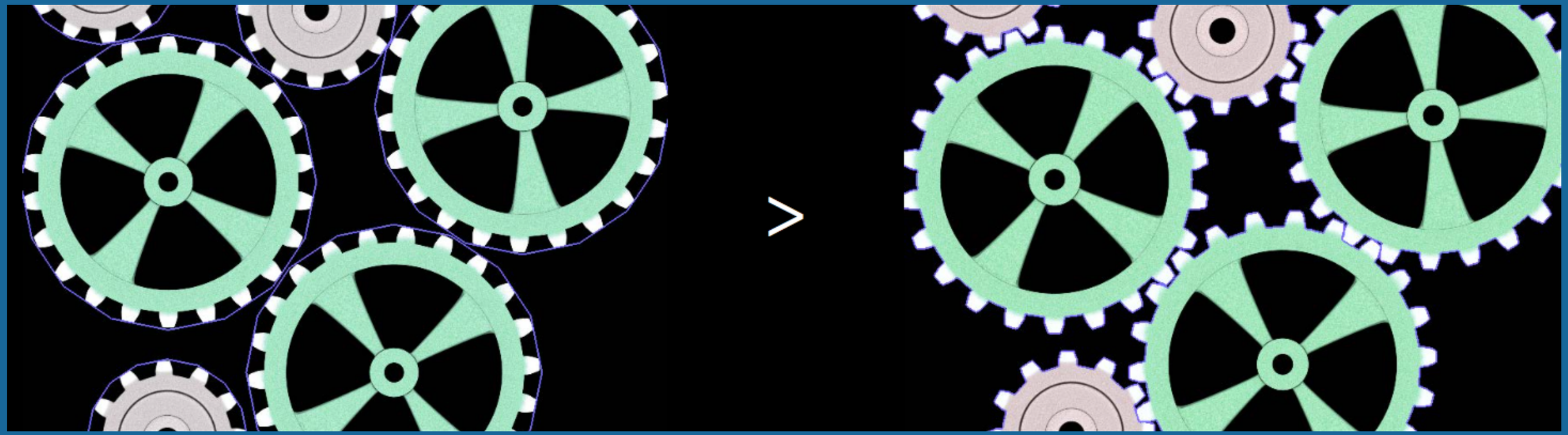
- Body properties
  - dynamic
  - affectedByGravity
  - allowsRotation
  - Pinned



PHYSICS

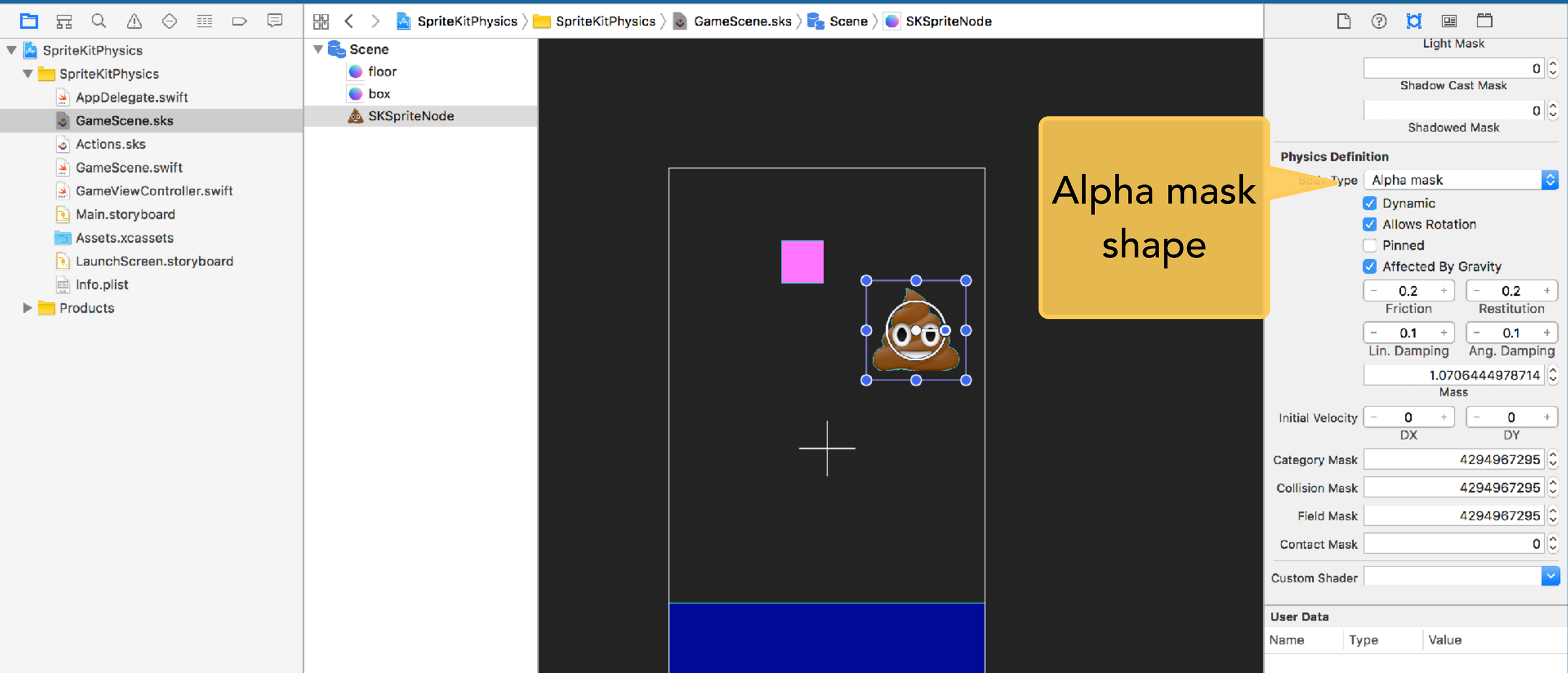

Scene
floor
box
Test your physics
Filter
Animate
Playback Speed
1x
All Nodes
0:00
00:01
00:02
00:03
00:04
box
floor
Scene
Name
name
Camera
Color
Size
Custom
750
W
1334
H
Anchor Point
0.5
X
0.5
Y
Gravity
0
X
-9.8
Y
Preview
Show Camera Boundaries
Show Physics Boundaries
Use Camera Node
Custom Shader
User Data
Name
Type
Value
Custom Shader Uniforms
Name
Type
Value

PHYSICS

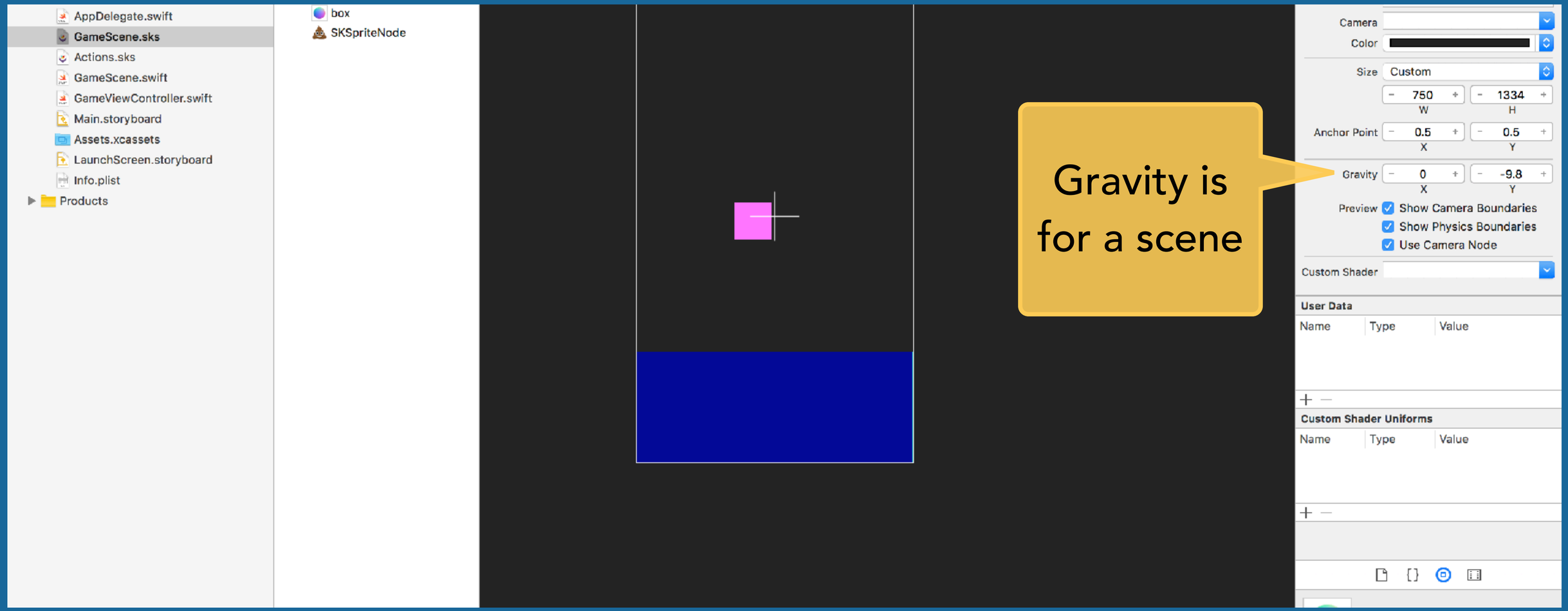


- A physics body for a circular sprite vs. using alpha property

PHYSICS


SpriteKitPhysics
SpriteKitPhysics
AppDelegate.swift
GameScene.sks
Actions.sks
GameScene.swift
GameViewController.swift
Main.storyboard
Assets.xcassets
LaunchScreen.storyboard
Info.plist
Products
SpriteKitPhysics
SpriteKitPhysics
GameScene.sks
Scene
SKSpriteNode
Scene
floor
box
SKSpriteNode
Alpha mask shape
Light Mask
0
Shadow Cast Mask
0
Shadowed Mask
Physics Definition
Body Type
Alpha mask
Dynamic
Allows Rotation
Pinned
Affected By Gravity
0.2
Friction
0.2
Restitution
0.1
Lin. Damping
0.1
Ang. Damping
1.0706444978714
Mass
Initial Velocity
0
DX
0
DY
Category Mask
4294967295
Collision Mask
4294967295
Field Mask
4294967295
Contact Mask
0
Custom Shader
User Data
Name
Type
Value

PHYSICS



- Objects can respond differently to gravity

MOVING PHYSICS BODIES

MOVING PHYSICS BODIES

SUBTITLE

- Moving physics bodies
  - Gravity (if enabled)
  - Explicit forces can be applied to nodes
    - Constant
    - Impulse

- Throwing a node:
  - Impulse in a direction
  - Other forces (friction, velocity are taken into account)

```
missile.physicsBody.velocity = self.physicsBody.velocity;
[missile.physicsBody applyImpulse:
CGVectorMake(missileLaunchImpulse*cosf(shipDirection),

missileLaunchImpulse*sinf(shipDirection))];
```

PHYSICS

- Receive UITouches and move sprites
- Remember touches sent as a set

```
func touchDown(atPoint pos : CGPoint) {
  player?.position = pos
}

func touchMoved(toPoint pos : CGPoint) {
  player?.position = pos
}

func touchUp(atPoint pos : CGPoint) {
  player?.position = pos
}

override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
  for t in touches { self.touchDown(atPoint: t.location(in: self)) }
}

override func touchesMoved(_ touches: Set<UITouch>, with event: UIEvent?) {
  for t in touches { self.touchMoved(toPoint: t.location(in: self)) }
}

override func touchesEnded(_ touches: Set<UITouch>, with event: UIEvent?) {
  for t in touches { self.touchUp(atPoint: t.location(in: self)) }
}

override func touchesCancelled(_ touches: Set<UITouch>, with event: UIEvent?) {
  for t in touches { self.touchUp(atPoint: t.location(in: self)) }
}
```

PHYSICS

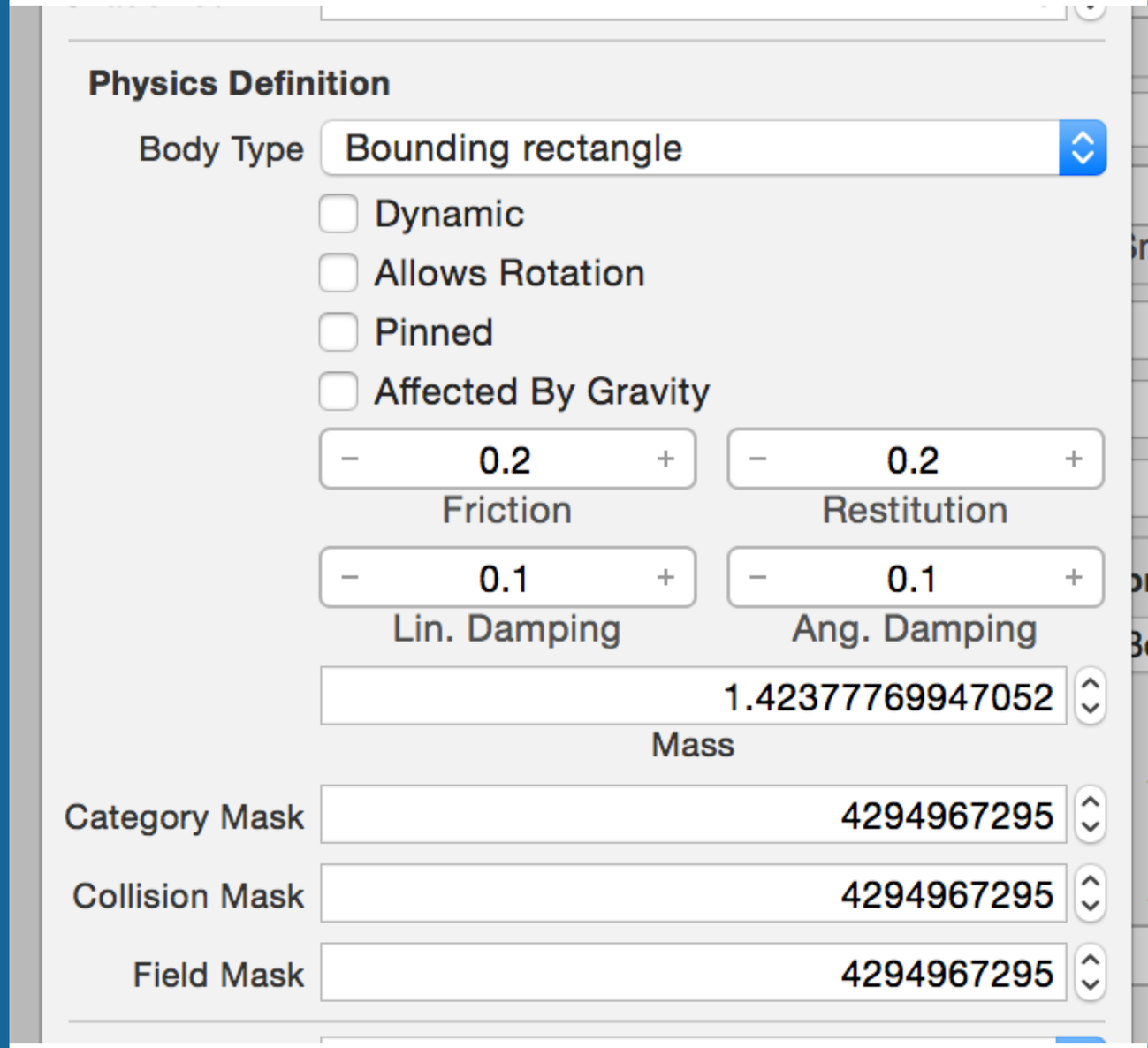
```
/**
 After touch is done, add back gravity and apply an impulse to "throw" node
 */
- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event {
    UITouch *touch = [touches anyObject];
    CGPoint pos = [touch locationInNode:self.parent];
    CGPoint prev = [touch previousLocationInNode:self.parent];

    CGFloat xMove = pos.x - prev.x;
    CGFloat yMove = pos.y - prev.y;
    CGFloat magnitude = MIN(sqrt((xMove * xMove) + (yMove * yMove)),5);
    //NSLog(@"Magnitude:%.2f",magnitude);

    self.strokeColor = [UIColor blueColor];
    self.physicsBody.affectedByGravity = YES;
    [self.physicsBody applyImpulse:CGVectorMake(xMove*magnitude, yMove*magnitude)];
    self.physicsBody.contactTestBitMask = CategoryTypePie;
}
```

BITMASKS

BITMASKS

SUBTITLE

- Masks are integers that handle physics relationships
  - Collisions
  - Contact
  - Logical groups
- Storing as integers allows for efficient storage and comparisons



BITMASKS

SUBTITLE

- Each physics object gets a category (binary number, 32 max)
- Number must be a power of 2 (bitshifting)
- Set interactions using bitwise OR operator
- A unique number is given for combinations

Name	Description
Category Mask	The type of object (player, enemy, item, etc.)
Collision Mask	The categories this object will physically collide with
Contact Mask	The categories that will trigger contact delegate methods upon contact
Field Mask	The gravity fields that affect the object

BITMASKS

- Bitmasked number allow logical operations
- UINT32 -> 32 bodies
- The default value is 0xFFFFFFFF (all bits set)

Binary	Value	Raw Code	Bit-Shifted Code
0001	1	0b0001	0b1
0010	2	0b0010	0b1 << 1
0011	3	0b0011	(does not apply)
0100	4	0b0100	0b1 << 2
0101	5	0b0101	(does not apply)

```
00000000000000000000000000000001 = 1 = 1 << 0
00000000000000000000000000000010 = 2 = 1 << 1
00000000000000000000000000000100 = 4 = 1 << 2
00000000000000000000000000001000 = 8 = 1 << 3
00000000000000000000000000010000 = 16 = 1 << 4
00000000000000000000000000100000 = 32 = 1 << 5
00000000000000000000000001000000 = 64 = 1 << 6
00000000000000000000000010000000 = 128 = 1 << 7
00000000000000000000000100000000 = 256 = 1 << 8
00000000000000000000001000000000 = 512 = 1 << 9
00000000000000000000010000000000 = 1024 = 1 << 10
```

PHYSICS

- Contact bit masks can be employed to specify the types of nodes for which contact notification is required
  - 00000000
  - 00000010
  - 00000100
- When a collision occurs the bit mask is tested between bodies
  - You can AND categories

```
/// Each physics body identifies a category tha
/// literals values but they need to be squares
/// http://stackoverflow.com/questions/24069703
enum PhysicsCategory: UInt32 {
    case Edge = 0              //0x1 << 0
    case Ball = 1              //0x1 << 1
    case Spoon = 2             //0x1 << 2
    case Hole = 4              //0x1 << 3
    case RainDrop = 8          //0x1 << 4
    case GravityField = 16     //0x1 << 5
    case Floor = 32            //0x1 << 6
}
```

```
00000010 // Edge

00000100 // Ball

00000110 // Edge and Ball
```

PHYSICS

```
/// Each physics body identifies a category tha
/// literals values but they need to be squares
/// http://stackoverflow.com/questions/24069703,
enum PhysicsCategory: UInt32 {
    case Edge = 0            //0x1 << 0
    case Ball = 1            //0x1 << 1
    case Spoon = 2           //0x1 << 2
    case Hole = 4            //0x1 << 3
    case RainDrop = 8        //0x1 << 4
    case GravityField = 16   //0x1 << 5
    case Floor = 32          //0x1 << 6
}
```

Define our physics bodies

```
self.physicsBody!.categoryBitMask = PhysicsCategory.ball.rawValue
self.physicsBody!.contactTestBitMask = PhysicsCategory.spoon.rawValue | PhysicsCategory.hole.rawValue
self.physicsBody!.collisionBitMask = PhysicsCategory.spoon.rawValue | PhysicsCategory.hole.rawValue | PhysicsCategory.floor.rawValue
```

Set our bit masks

PHYSICS COLLISIONS

COLLISIONS

```
self.physicsBody!.categoryBitMask = PhysicsCategory.Ball.rawValue
self.physicsBody!.contactTestBitMask = PhysicsCategory.Spoon.rawValue | PhysicsCategory.Hole.rawValue
self.physicsBody!.collisionBitMask = PhysicsCategory.Spoon.rawValue | PhysicsCategory.Hole.rawValue | PhysicsCategory.Floor.rawValue
```

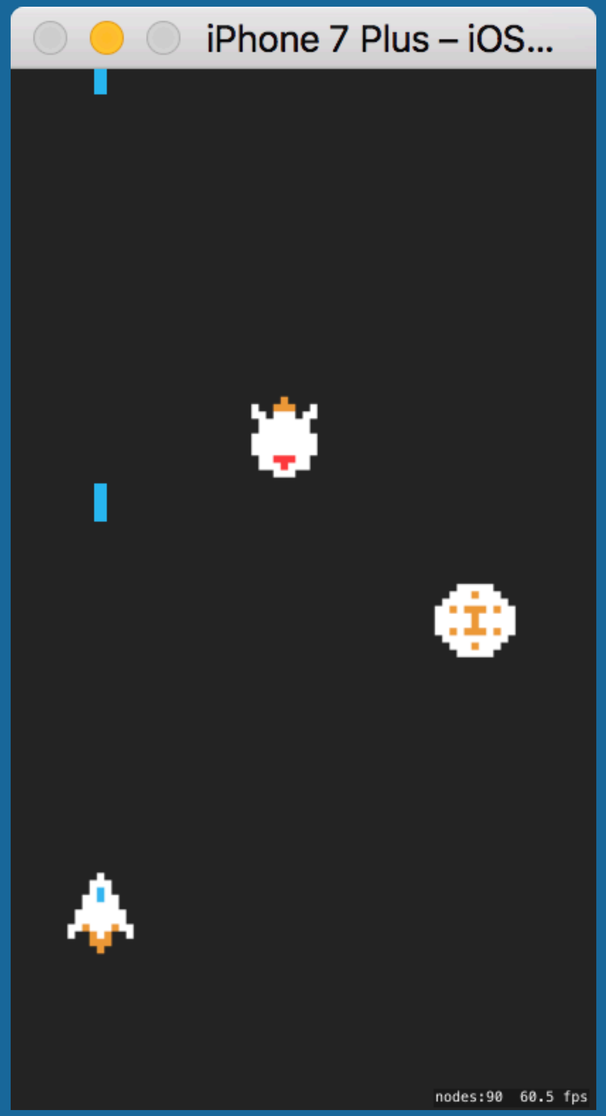
- We only are tracking Ball category
- We want to allows a Ball to contact a Spoon or a Hole
- Inform us when a collision occurs between a ball and Spoon, Hole or Floor

COLLISIONS

- Determine the rules of interactions in a physics world
- Example contact map for space shooter

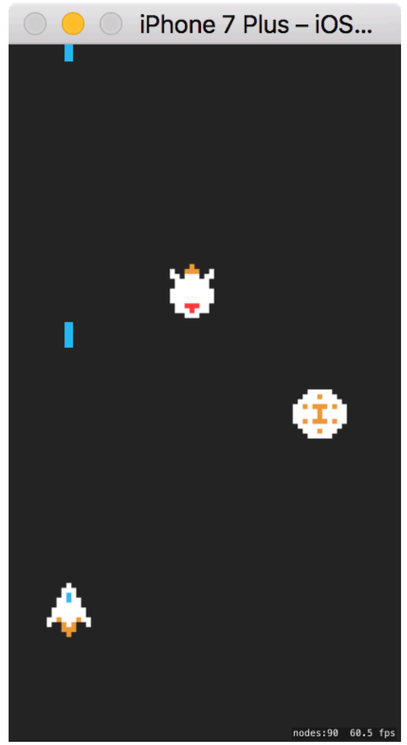
	Missile	Rocket ship	Asteroid	Planet
Missile	No	No	No	No
Rocket ship	No	Yes	Yes	Yes
Asteroid	No	Yes	Yes	No
Planet	No	No	No	Yes

COLLISIONS




```
class GameScene: SKScene,SKPhysicsContactDelegate {
  var player:SKSpriteNode?
  var enemy:SKSpriteNode?
  var item:SKSpriteNode?
  var fireRate:TimeInterval = 0.5
  var timeSinceFire:TimeInterval = 0
  var lastTime:TimeInterval = 0

  let noCategory:UInt32 = 0
  let laserCategory:UInt32 = 0b1
  let playerCategory:UInt32 = 0b1 << 1
  let enemyCategory:UInt32 = 0b1 << 2
  let itemCategory:UInt32 = 0b1 << 3


  override func didMove(to view: SKView) {
    self.physicsWorld.contactDelegate = self

    player = self.childNode(withName: "player") as? SKSpriteNode
    player?.physicsBody?.categoryBitMask = playerCategory
    player?.physicsBody?.collisionBitMask = noCategory
    player?.physicsBody?.contactTestBitMask = enemyCategory | itemCategory

    item = self.childNode(withName: "item") as? SKSpriteNode
    item?.physicsBody?.categoryBitMask = itemCategory
    item?.physicsBody?.collisionBitMask = noCategory
    item?.physicsBody?.contactTestBitMask = playerCategory

    enemy = self.childNode(withName: "enemy") as? SKSpriteNode
    enemy?.physicsBody?.categoryBitMask = enemyCategory
    enemy?.physicsBody?.collisionBitMask = noCategory
    enemy?.physicsBody?.contactTestBitMask = playerCategory | laserCategory
  }

  func didBegin(_ contact: SKPhysicsContact) {
    contact.bodyA.node?.removeFromParent()
    contact.bodyB.node?.removeFromParent()
  }
```

COLLISIONS



What category am I?

Which can I collide with?

Which triggers contact delegate

```
func spawnLaser() {
  let scene:SKScene = SKScene(fileNamed: "Laser")!
  let laser = scene.childNode(withName: "laser")
  laser?.position = player!.position
  laser?.move(toParent: self)
  laser?.physicsBody?.categoryBitMask = laserCategory
  laser?.physicsBody?.collisionBitMask = noCategory
  laser?.physicsBody?.contactTestBitMask = enemyCategory
}
```

COLLISIONS

Delegate to send contact info

Where to send...

```
class GameScene: SKScene,SKPhysicsContactDelegate {

    var player:SKSpriteNode?
    var enemy:SKSpriteNode?
    var item:SKSpriteNode?
    var fireRate:TimeInterval = 0.5
    var timeSinceFire:TimeInterval = 0
    var lastTime:TimeInterval = 0

    let noCategory:UInt32 = 0
    let laserCategory:UInt32 = 0b1
    let playerCategory:UInt32 = 0b1 << 1
    let enemyCategory:UInt32 = 0b1 << 2
    let itemCategory:UInt32 = 0b1 << 3


    override func didMove(to view: SKView) {
        self.physicsWorld.contactDelegate = self
```

COLLISIONS

- BodyA and BodyB involved in contact
  - Need to figure out which one is which
- Laser hits enemy both are removed

```
  enemy?.physicsBody?.contactTestBitMask = playerCategory | laserCategory
}

func didBegin(_ contact: SKPhysicsContact) {
  contact.bodyA.node?.removeFromParent()
  contact.bodyB.node?.removeFromParent()
}
```

Need to figure out who is A and B

COLLISIONS

```
    enemy?.physicsBody?.contactTestBitMask = playerCategory | laserCategory
}

func didBegin(_ contact: SKPhysicsContact) {
    let cA:UInt32 = contact.bodyA.categoryBitMask
    let cB:UInt32 = contact.bodyB.categoryBitMask

    if cA == playerCategory || cB == playerCategory {
        let otherNode:SKNode = (cA == playerCategory) ?
            contact.bodyB.node! : contact.bodyA.node!
        playerDidCollide(with: otherNode)
    }
    else {
        contact.bodyA.node?.removeFromParent()
        contact.bodyB.node?.removeFromParent()
    }
}

func playerDidCollide(with other:SKNode) {
    let otherCategory = other.physicsBody?.categoryBitMask
    if otherCategory == itemCategory {
        other.removeFromParent()
    }
    else if otherCategory == enemyCategory {
        other.removeFromParent()
        player?.removeFromParent()
    }
}
```

Did player contact anything?

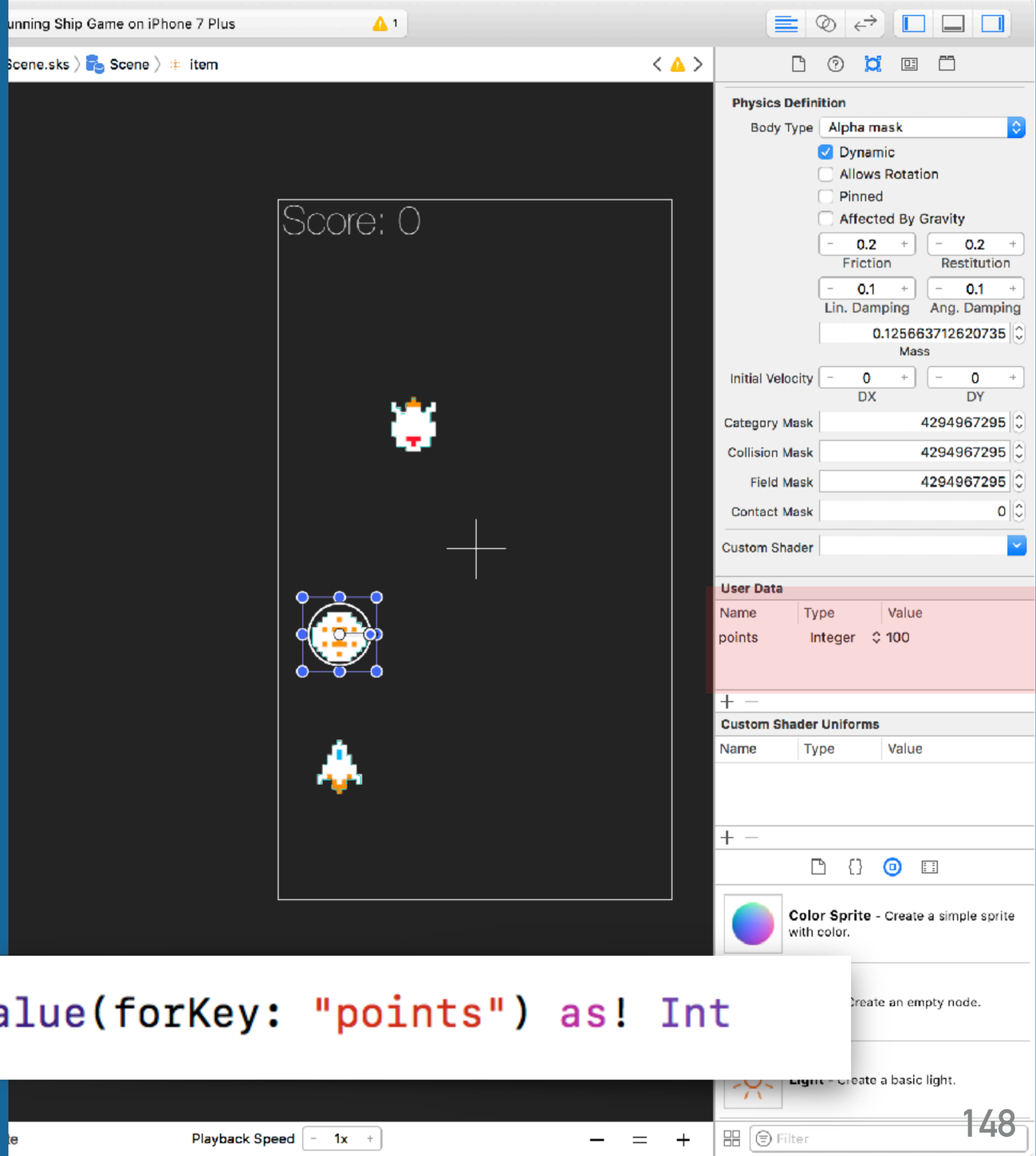
Did non-players contact (eg laser, enemy)?

Did player contact item or enemy?

COLLISIONS

SUBTITLE

- Store arbitrary data associated with a sprite using `userData`
- Store the point values for a given sprite




```
let points:Int = other.userData?.value(forKey: "points") as! Int
```

COLLISIONS

- didBegin() contact may be called multiple times in runloop
- Handle this as appropriate for your app

```
func didBegin(_ contact: SKPhysicsContact) {
    let cA:UInt32 = contact.bodyA.categoryBitMask
    let cB:UInt32 = contact.bodyB.categoryBitMask
    if cA == playerCategory || cB == playerCategory {
        let otherNode:SKNode = (cA == playerCategory) ? contact.bodyB.node! : contact.bodyA.node!
        playerDidCollide(with: otherNode)
    }
    else {
        contact.bodyA.node?.removeFromParent()
        contact.bodyB.node?.removeFromParent()
    }
}

func playerDidCollide(with other:SKNode) {
    if other.parent == nil {
        return
    }
    let otherCategory = other.physicsBody?.categoryBitMask
    if otherCategory == itemCategory {
        let points:Int = other.userData?.value(forKey: "points") as! Int
        score += points
        label?.text = "Score: \(score)"
        other.removeFromParent()
    }
    else if otherCategory == enemyCategory {
        other.removeFromParent()
        player?.removeFromParent()
    }
}
```

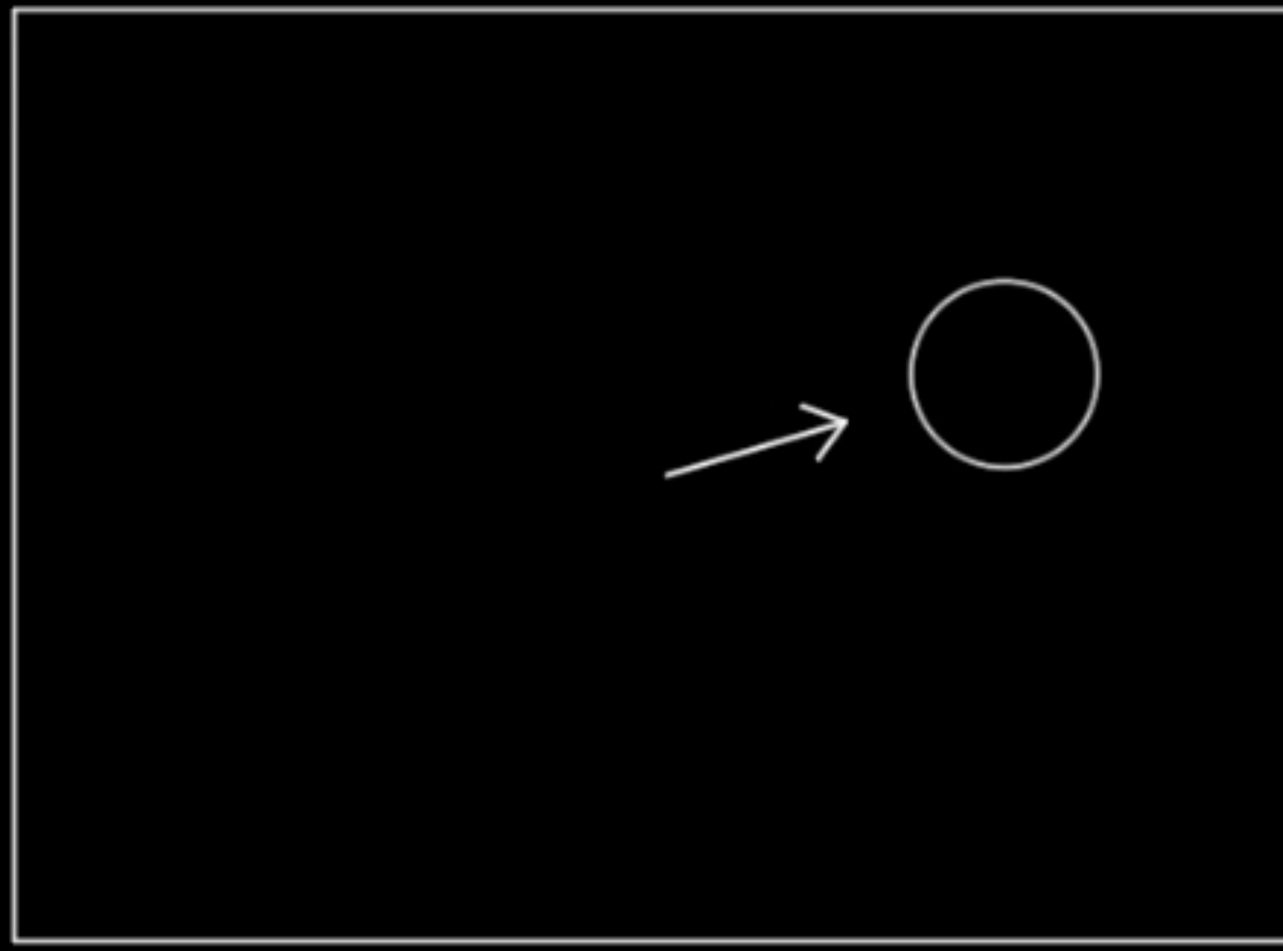
Check if it is already removed.

CONSTRAINTS (MORE COMPLEX PHYSICS BODIES)

CONSTRAINTS

SUBTITLE

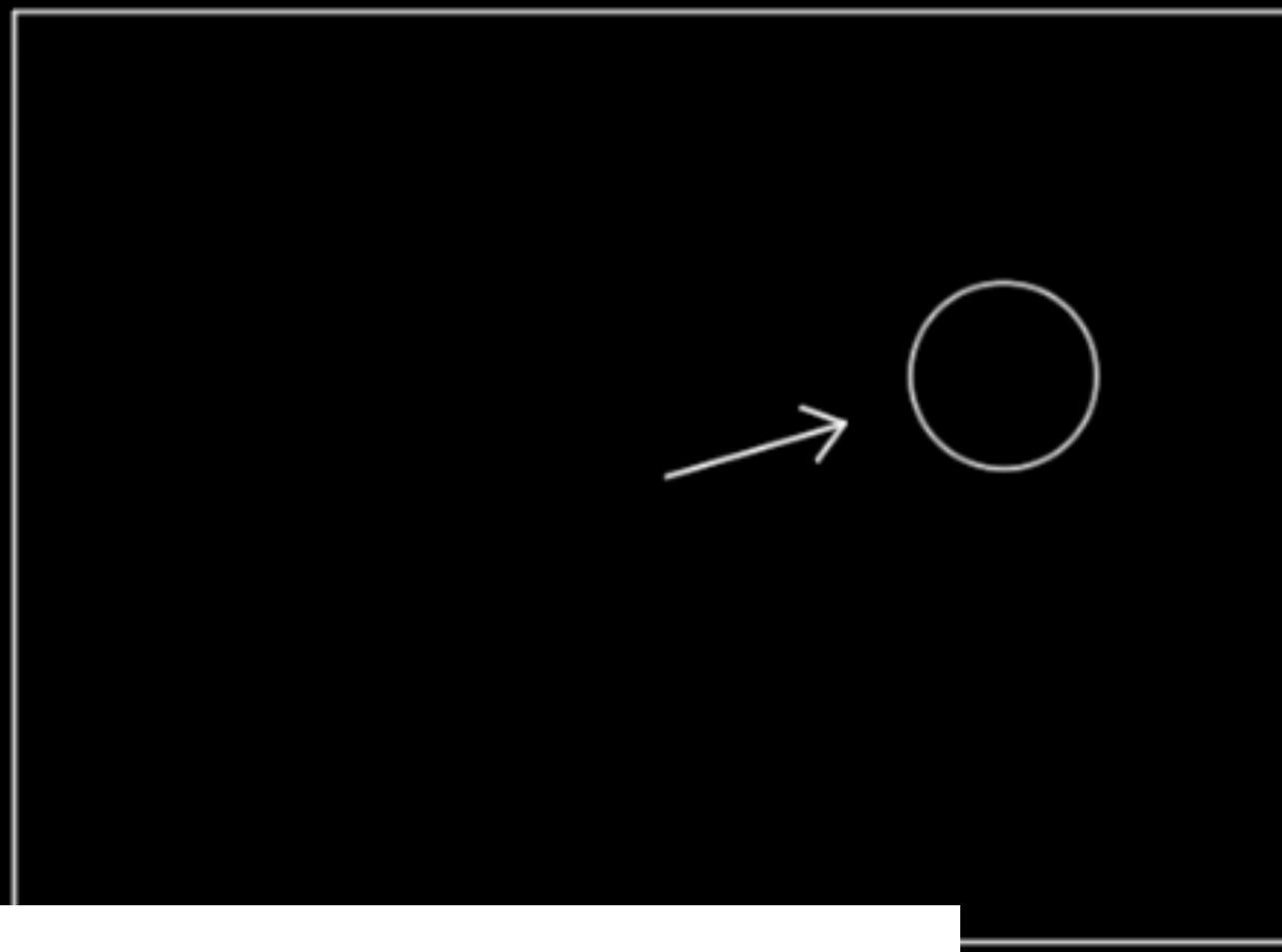
- Defines a mathematical constraint on one property of a node
- Constraints are attached to nodes
- Scene applies constraints attached to nodes



CONSTRAINTS

SUBTITLE

- Constraint properties
  - Position
  - Orientation
  - Distance
  - Enable/Disable

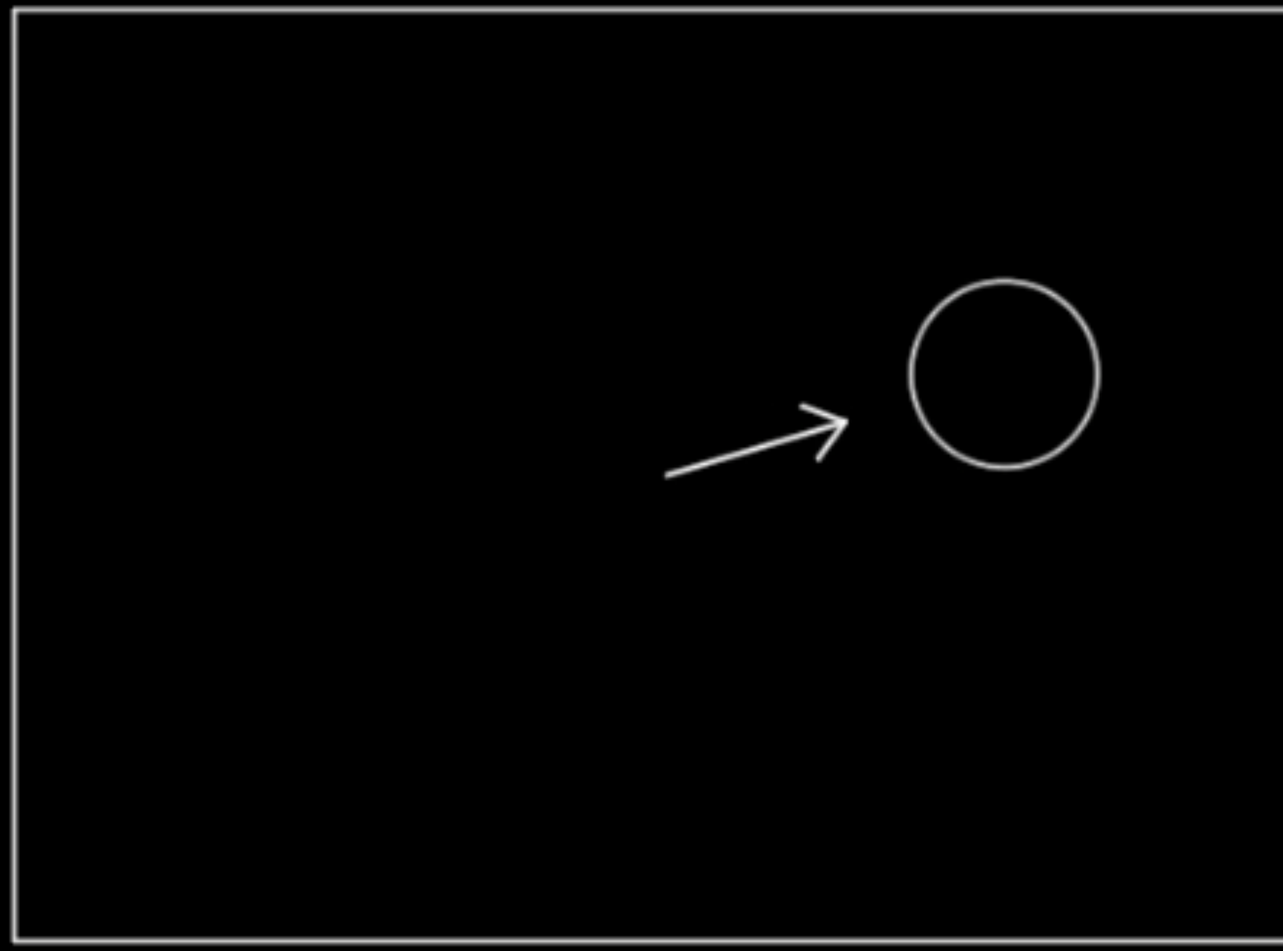


```
SKConstraint* orientConstraint = [SKConstraint orientToNode:targetNode];
node.constraints = @[orientConstraint];
```

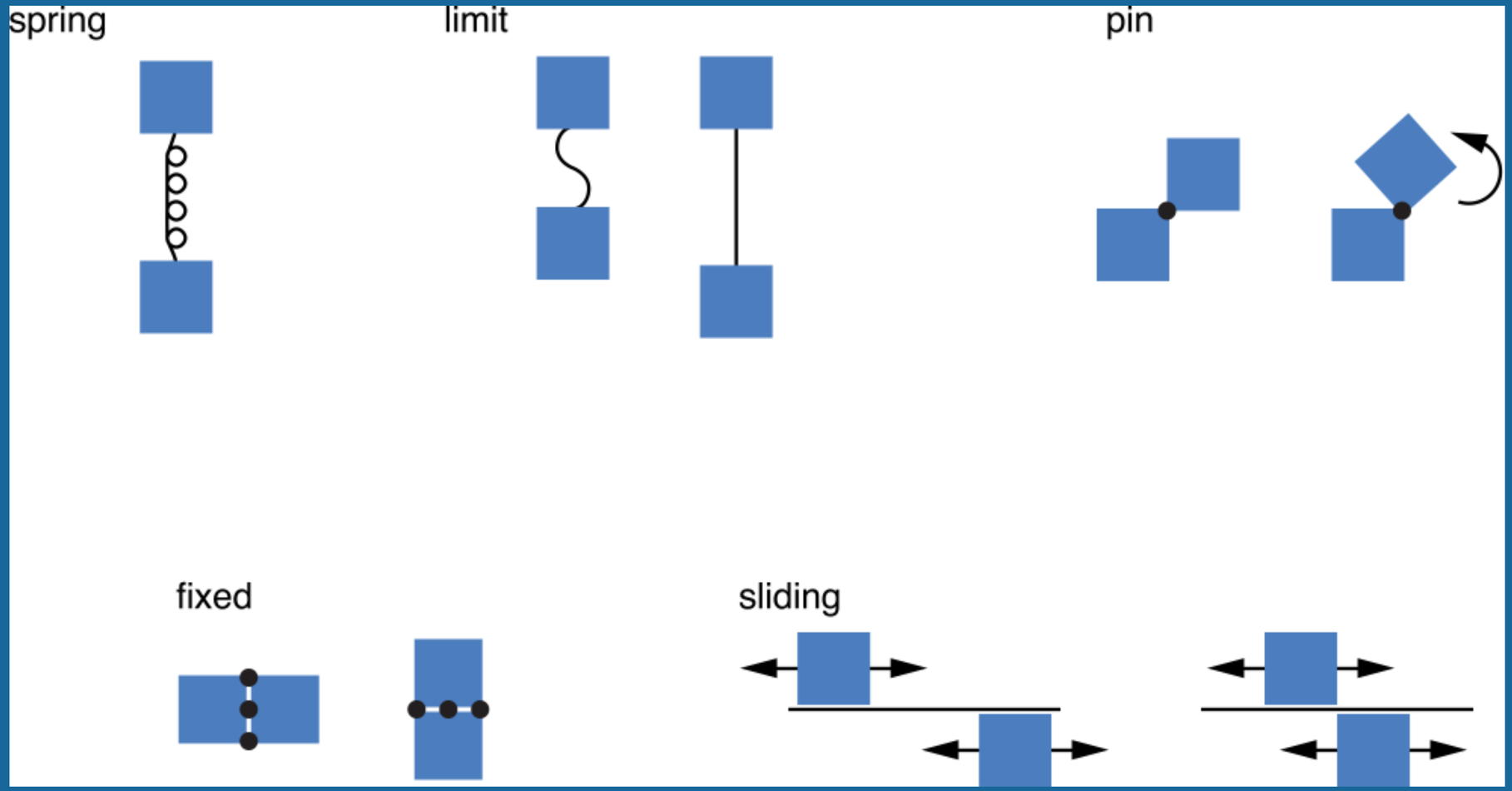
CONSTRAINTS

SUBTITLE

- Use cases
  - Follow an object
    - Point cannon at object
  - Group nodes
    - Health points always near a hero



CONSTRAINTS



- Connecting physics bodies

CONSTRAINTS

Class	Description
`SKPhysicsJointFixed`	A fixed joint fuses two bodies together at a reference point. Fixed joints are useful for creating complex shapes that can be broken apart later.
`SKPhysicsJointSliding`	A sliding joint permits the anchor points of two bodies to slide along a chosen axis.
`SKPhysicsJointSpring`	A spring joint acts as a spring whose length is the initial distance between two bodies.
`SKPhysicsJointLimit`	A limit joint imposes a maximum distance between two bodies, as if they were connected by a rope.
`SKPhysicsJointPin`	A pin joint pins two bodies together. The bodies rotate independently around the anchor point.

- Connecting physics bodies

THE UNIVERSITY OF CHICAGO

ADVANCED iOS APPLICATION DEVELOPMENT

MPCS 51032 • SPRING 2020 • SESSION 6

ACTIONS AND ANIMATIONS

ACTIONS AND ANIMATIONS

SUBTITLE

- Built actions and animations in scene editor or programmatically
- SKAction is class that alters the properties of an SKNode

Class

SKAction

An object that is executed by an SKNode to change its structure or content.

Language

Swift | Objective-C

SDKs

iOS 7.0+

macOS 10.9+

tvOS 9.0+

watchOS 3.0+

On This Page

Overview

Symbols

Relationships

Overview

An SKAction object is an action that is executed by a node in the scene (SKScene). Actions are most often used to change the structure and content of the node to which they are attached but can also make other changes to the scene. When the scene processes its nodes, actions associated with those nodes are evaluated.To create an action, call the class method for the action you are interested in. Then, configure the action's properties. Finally, to execute the action, call a node object's run(_:) method (or a similar method on the SKNode class) and pass it the action object.

Most actions allow you to change a node's properties, such as its position, rotation, or scale. Some actions specifically apply only to SKSpriteNode objects, allowing you to animate a sprite's color or texture properties. Many of these actions are animated by SpriteKit, meaning that they change the properties of the associated node over more than one frame of animation rendered by the scene. When an action is animated, the duration property states how

ACTIONS AND ANIMATIONS

- Very simple to use
  - Single action class—SKAction
  - One line creation
  - Chainable, reusable, human readable
- Like a scripting language for Sprite Kit
  - Actions directly affect the node it is run on
  - Actions run immediately
  - Removed on completion

ACTIONS AND ANIMATIONS

Creating Actions That Change a Node's Scale

+ `scaleBy:duration:`
+ `scaleTo:duration:`
+ `scaleXBy:y:duration:`
+ `scaleXTo:y:duration:`
+ `scaleXTo:duration:`
+ `scaleYTo:duration:`

Creating Actions That Change a Node's Transparency

+ `fadeInWithDuration:`
+ `fadeOutWithDuration:`
+ `fadeAlphaBy:duration:`
+ `fadeAlphaTo:duration:`

Creating Actions That Change a Sprite Node's Content

+ `resizeByWidth:height:duration:`
+ `resizeToHeight:duration:`
+ `resizeToWidth:duration:`
+ `resizeToWidth:height:duration:`
+ `setTexture:`
+ `setTexture:resize:`
+ `animateWithTextures:timePerFrame:`

Creating Actions That Change a Node's Transparency

+ `fadeInWithDuration:`
+ `fadeOutWithDuration:`
+ `fadeAlphaBy:duration:`
+ `fadeAlphaTo:duration:`

Creating Actions That Change a Sprite Node's Conten

+ `resizeByWidth:height:duration:`
+ `resizeToHeight:duration:`
+ `resizeToWidth:duration:`
+ `resizeToWidth:height:duration:`
+ `setTexture:`
+ `setTexture:resize:`
+ `animateWithTextures:timePerFrame:`
+ `animateWithTextures:timePerFrame:resize:restore:`
+ `colorizeWithColor:colorBlendFactor:duration:`
+ `colorizeWithColorBlendFactor:duration:`

Playing Sounds

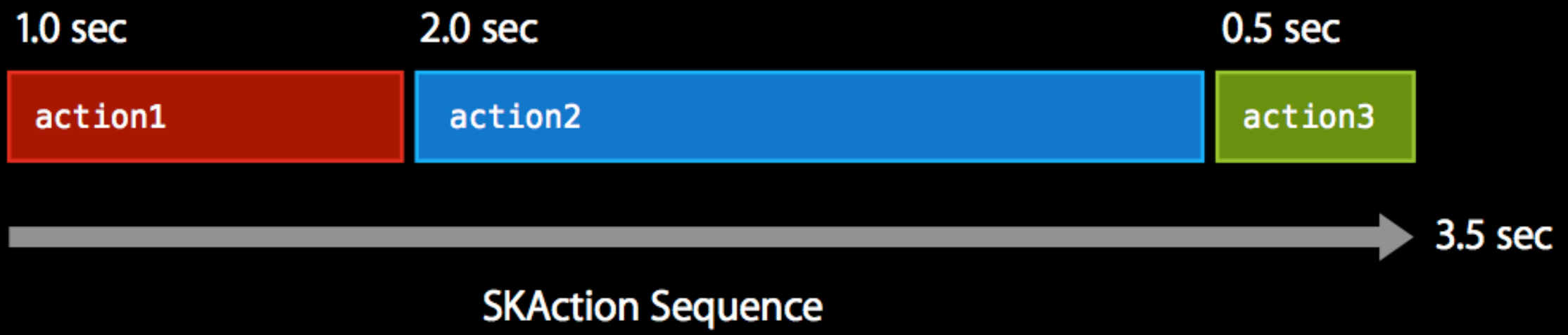
+ `playSoundFileNamed:waitForCompletion:`

Removing Nodes from the Scene

+ `removeFromParent`

ACTIONS AND ANIMATIONS

```
[myNode runAction: [SKAction sequence:@[action1, action2, action3]] ];
```



- Chain actions as sequence

ACTIONS AND ANIMATIONS

```
let shrink = SKAction.scale(to: 0.0, duration: 0.0)
let grow = SKAction.scale(to: 1.0, duration: 0.5)
let tink = SKAction.playSoundFileNamed("Tink.caf", waitForCompletion: false)
self.run(SKAction.sequence([tink,shrink,grow]))
```

- Chain actions to “appear”

ACTIONS AND ANIMATIONS

```
let grow = SKAction.scale(to: 5.0, duration: 0.1)
let tink = SKAction.playSoundFileNamed("Tink.caf", waitForCompletion: false)
let remove = SKAction.removeFromParent()
self.run(SKAction.sequence([tink,grow,remove]))
```

- Chain actions to “blow up”

Removing a node can be part of a sequence

ACTIONS AND ANIMATIONS

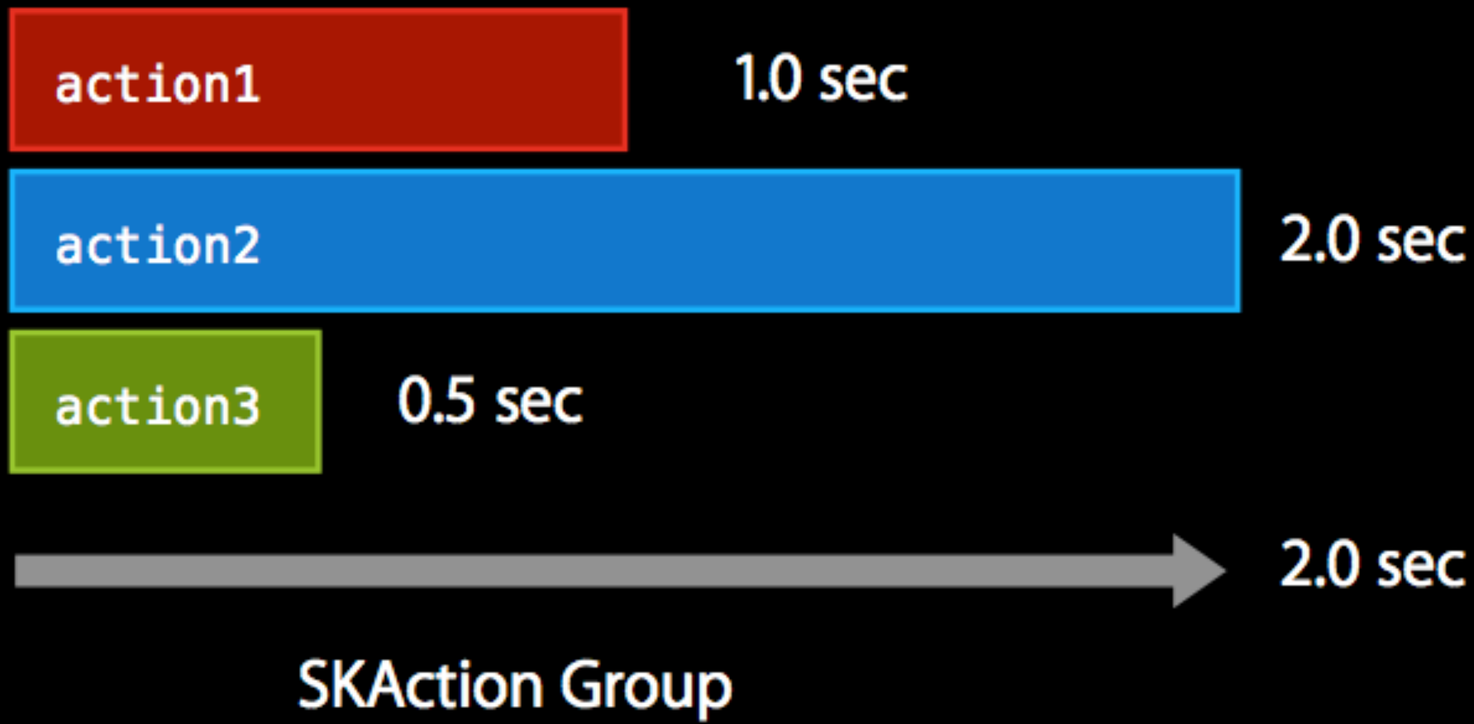
```
let shrink = SKAction.scale(to: 0.0, duration: 0.0)
let grow = SKAction.scale(to: 1.0, duration: 0.5)
let tink = SKAction.playSoundFileNamed("Tink.caf", waitForCompletion: false)
let sequence = SKAction.sequence([tink,shrink,grow])

self.run(sequence, withKey: "Appear-Action")
//self.removeAction(forKey: "Appear-Action")
```

- Named actions with a unique key

Name actions to access them later

ACTIONS AND ANIMATIONS

```
[myNode runAction: [SKAction group:@[action1, action2, action3]] ];
```



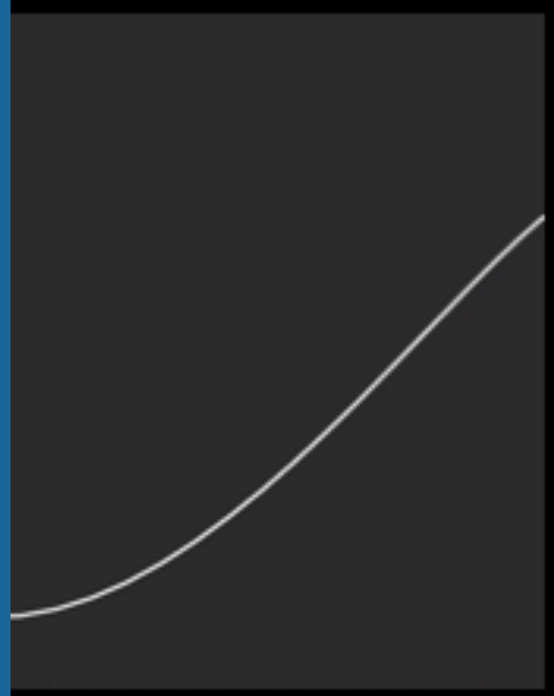
- Grouped actions perform simultaneously

ACTIONS AND ANIMATIONS

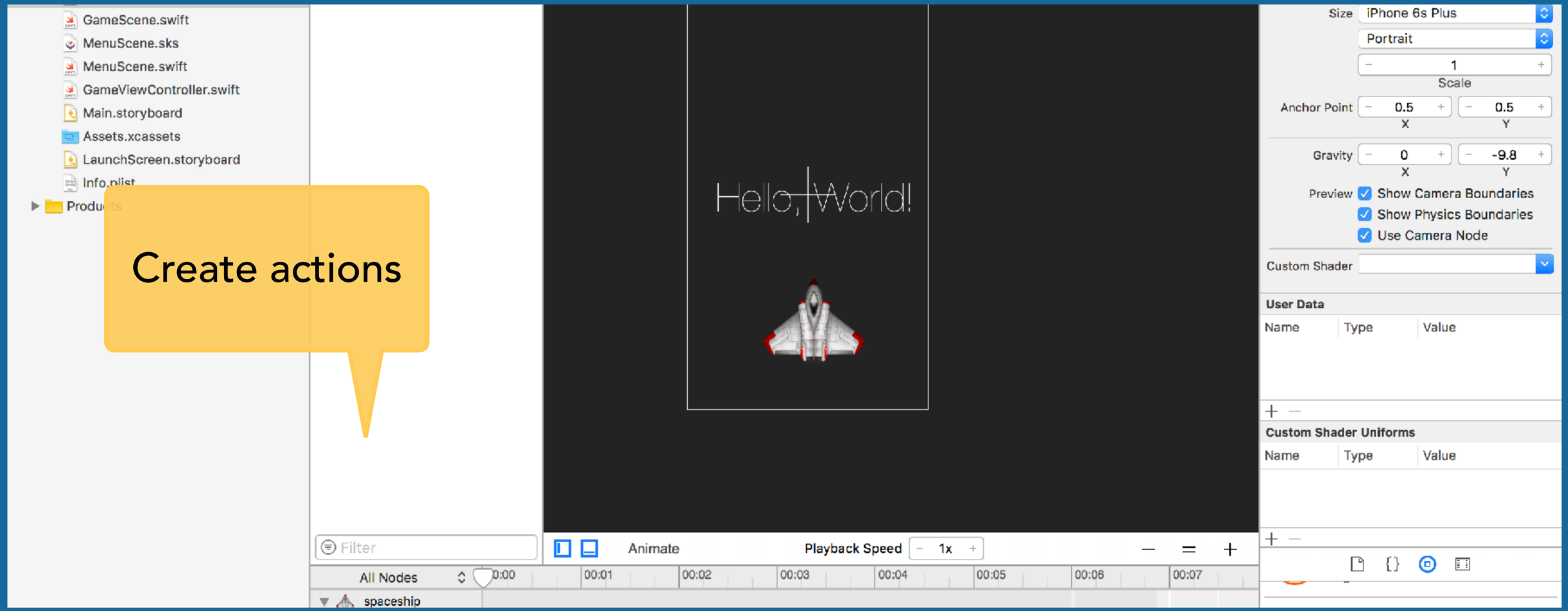
SUBTITLE

- Grouped actions perform simultaneously
- Groups of sequences
- Path following (UIBezierCurves)

```
ath duration:2.5]

ath asOffset:YES orientToPath:NO du
```



ACTIONS AND ANIMATIONS

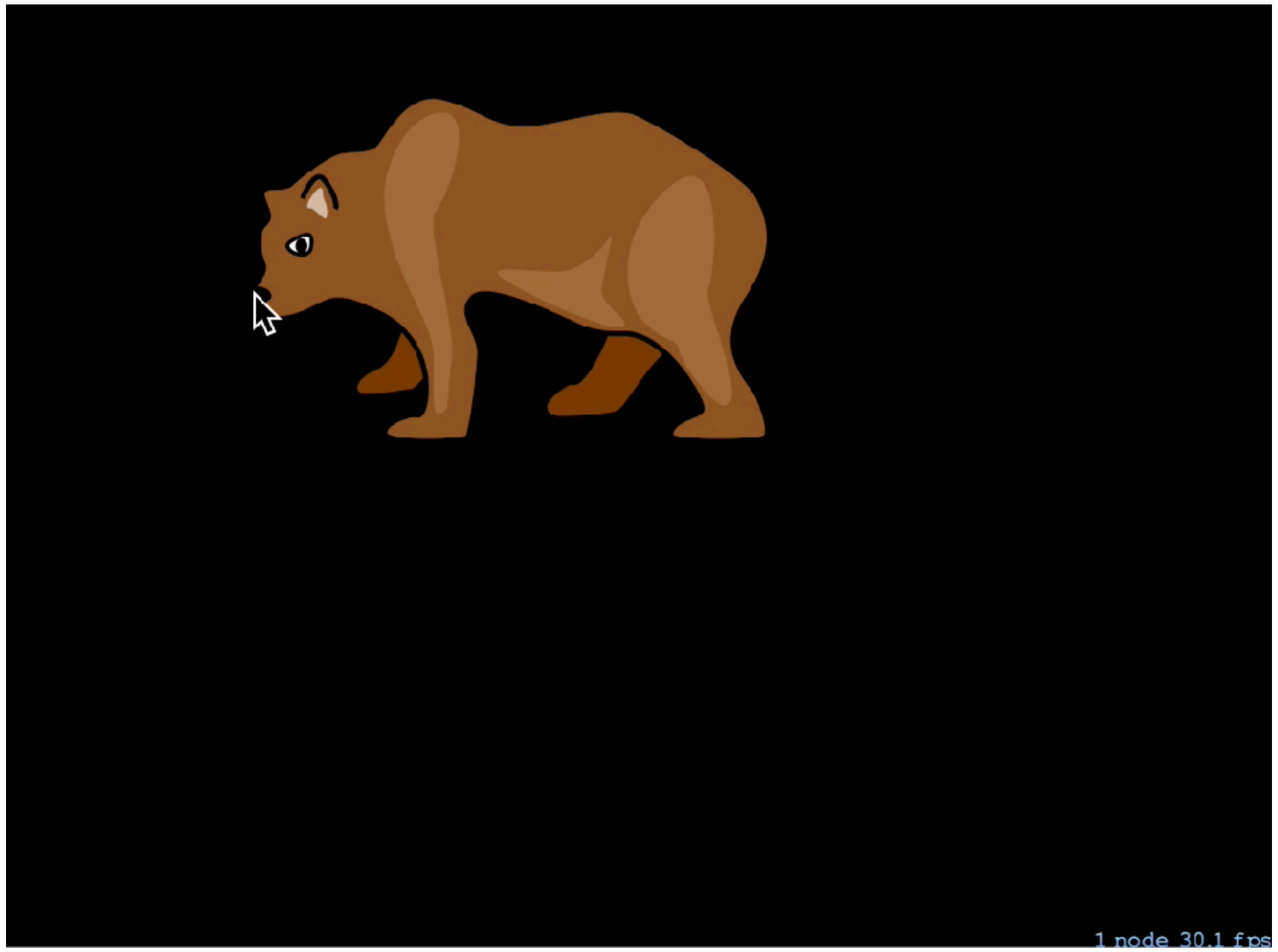


- Same call for atlas or stand alone image

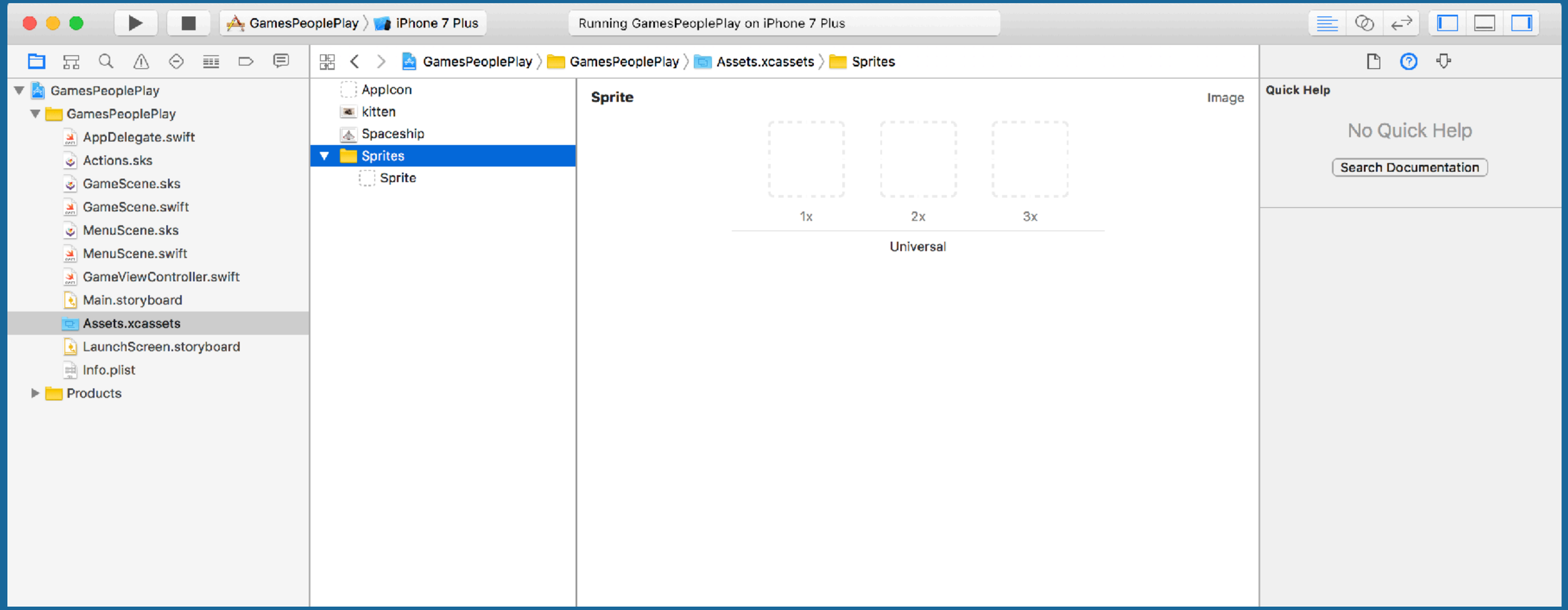
ACTIONS AND ANIMATIONS

SUBTITLE

- Frame animation plays animation sequence from an array of images



ACTIONS AND ANIMATIONS



- SpriteAtlas file organizes a images for more efficient access

ACTIONS AND ANIMATIONS

- Load each texture and pass them into a repeating action

```
let frame1:SKTexture = SKTexture(imageNamed: "player_frame1")
let frame2:SKTexture = SKTexture(imageNamed: "player_frame2")
let frame3:SKTexture = SKTexture(imageNamed: "player_frame3")
let frame4:SKTexture = SKTexture(imageNamed: "player_frame4")

let animation:SKAction = SKAction.animate(with: [frame1,frame2,frame3,frame4],
    timePerFrame: 0.1)
let repeatAction:SKAction = SKAction.repeatForever(animation)
player?.run(repeatAction)
```

ACTIONS AND ANIMATIONS

```
        /* Setup your scene here */
        backgroundColor = (UIColor.blackColor())

        //Load the TextureAtlas for the bear
        let bearAnimatedAtlas : SKTextureAtlas = SKTextureAtlas(named: "BearImages")

        //Load the animation frames from the TextureAtlas
        var walkFrames = [SKTexture]()
        let numImages : Int = bearAnimatedAtlas.textureNames.count
        for var i=1; i<=numImages/2; i++ {
            let bearTextureName = "bear\(i)"
            walkFrames.append(bearAnimatedAtlas.textureNamed(bearTextureName))
        }

        bearWalkingFrames = walkFrames

        //Create bear sprite, setup position in middle of the screen, and add to Scene
        let temp : SKTexture = bearWalkingFrames[0]
        bear = SKSpriteNode(texture: temp)
```

- Same call for atlas or stand alone image

http://www.raywenderlich.com/89222/sprite-kit-animations-texture-atlases-swift

ACTIONS AND ANIMATIONS

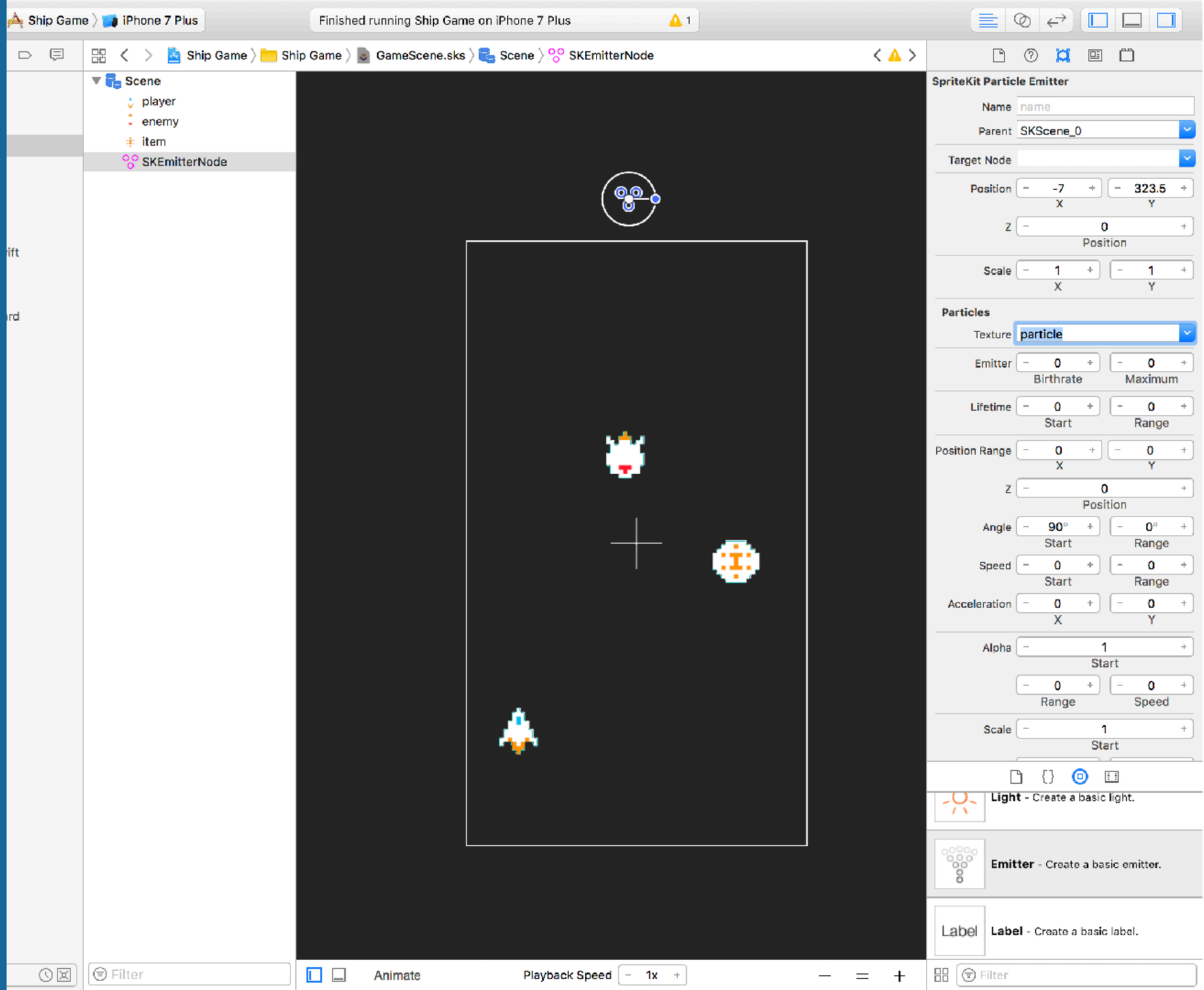
```
/* zero duration, fires once */

[SKAction runBlock:^{
  doSomething();
}]

/* show game over menu after character death animation */

SKAction *fade = [SKAction fadeOutWithDuration:1.0];
SKAction *remove = [SKAction removeFromParent];
SKAction *showMenu = [SKAction runBlock:^{
    [self showGameOverMenu];
    }];
[heroSprite runAction: [SKAction sequence:@[fade, showMenu, remove]] ];
```

Arbitrary block can be execute on node

PARTICLE EMITTERS

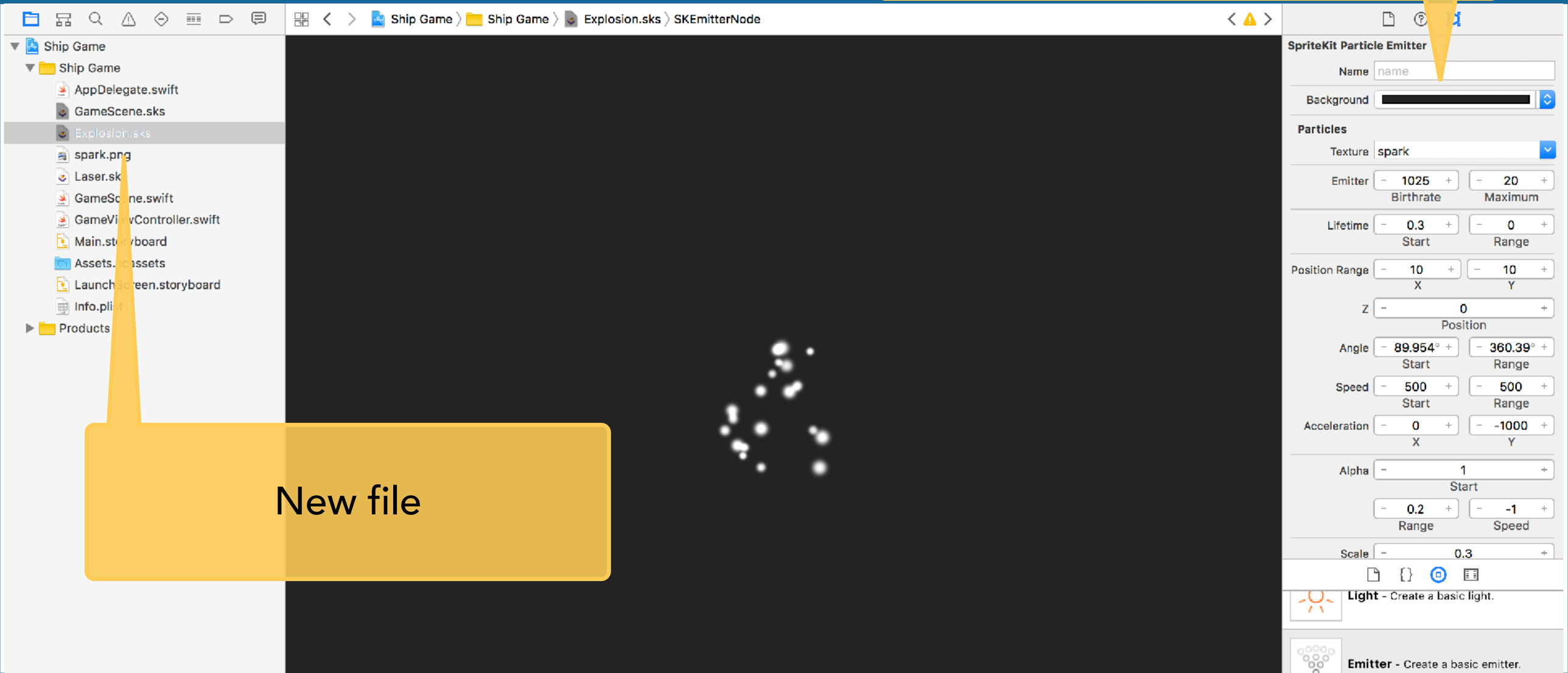
PARTICLE EMMITERS

- Particle emitter node generates images with adjustable visual properties



PARTICLE EMMITERS

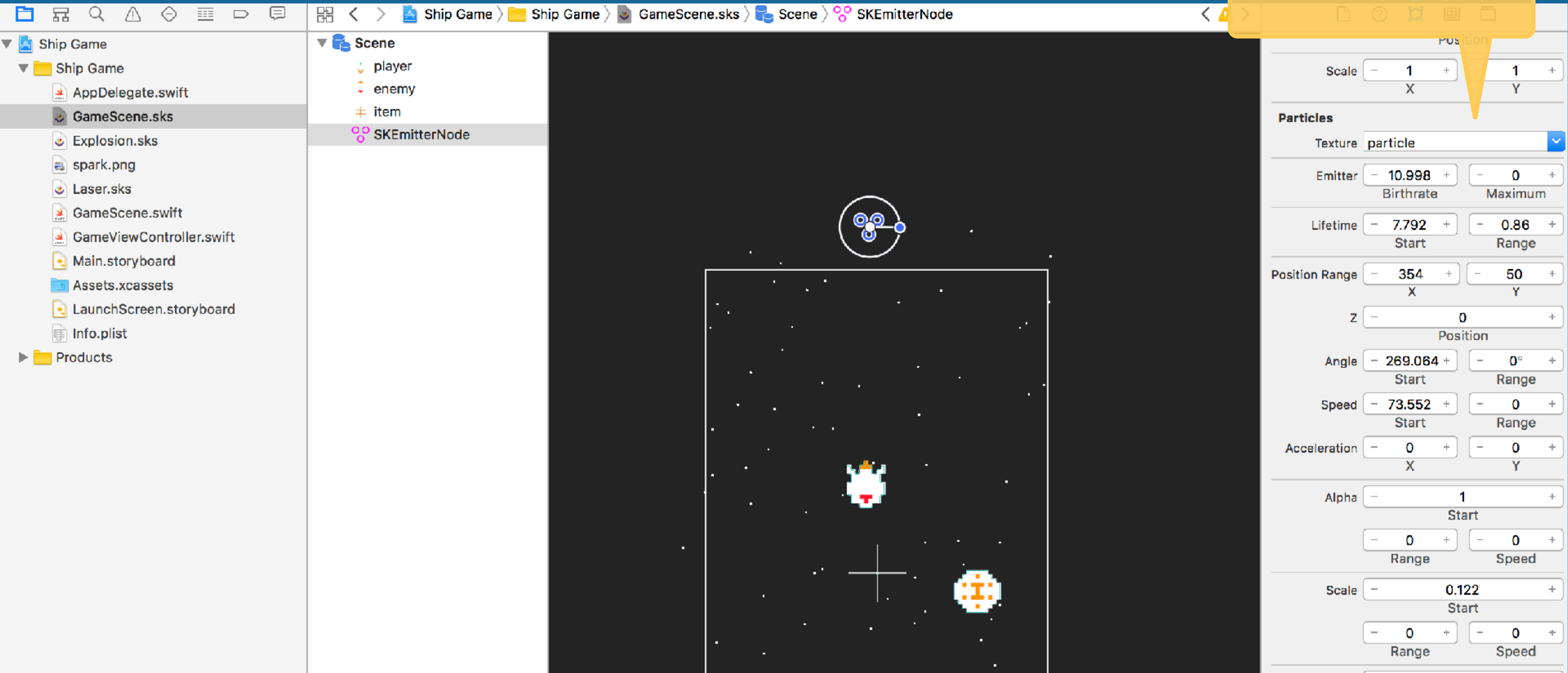
Create an emitter



New file

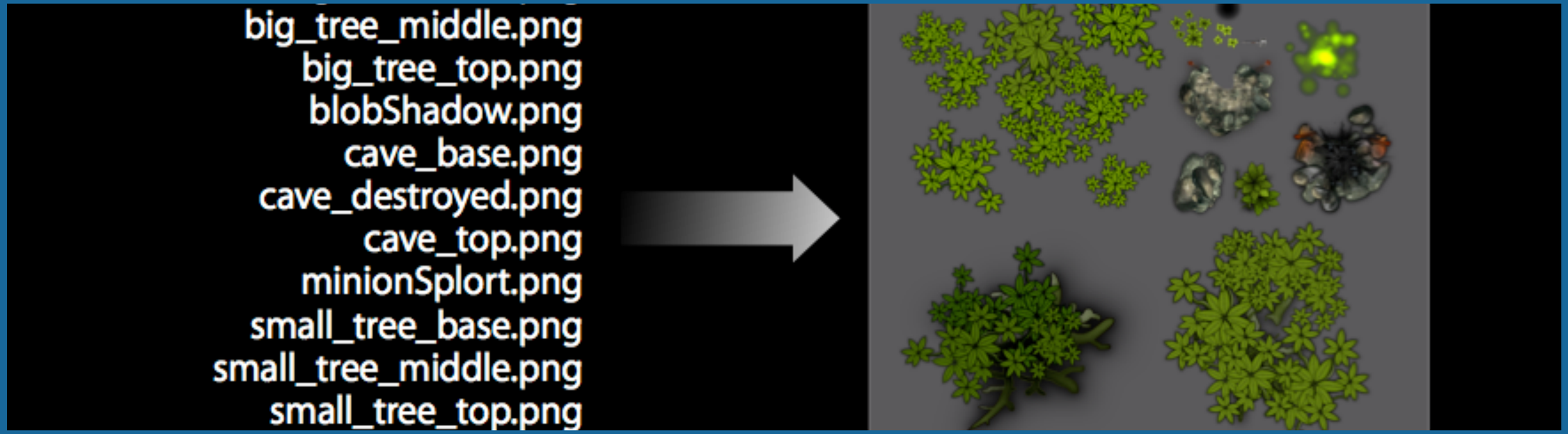
PARTICLE EMMITERS

Control the emitters



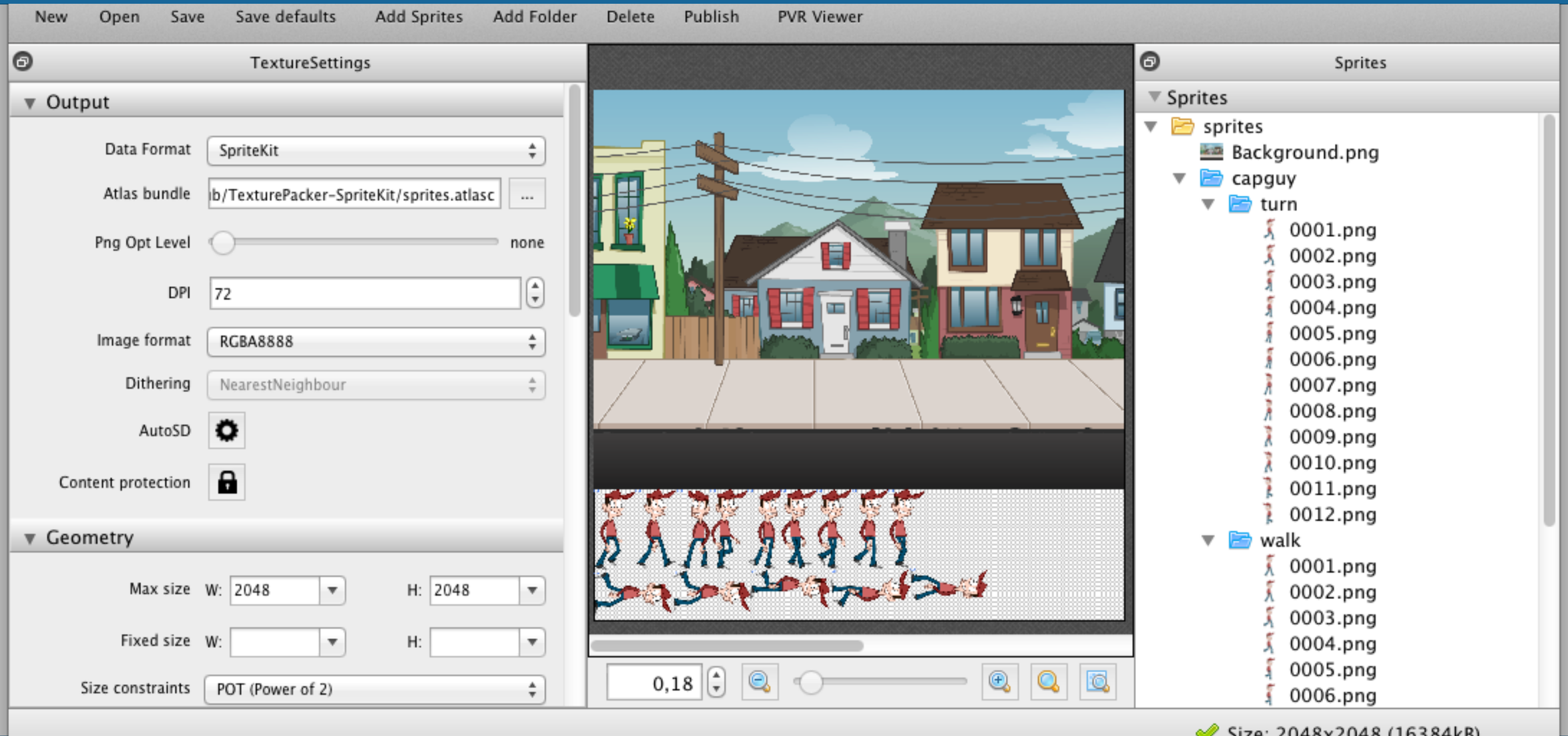
TEXTURES AND ATLASES

TEXTURES AND ALIASES



- Many images combined into a single image
  - Saves memory
  - Improves drawing efficiency

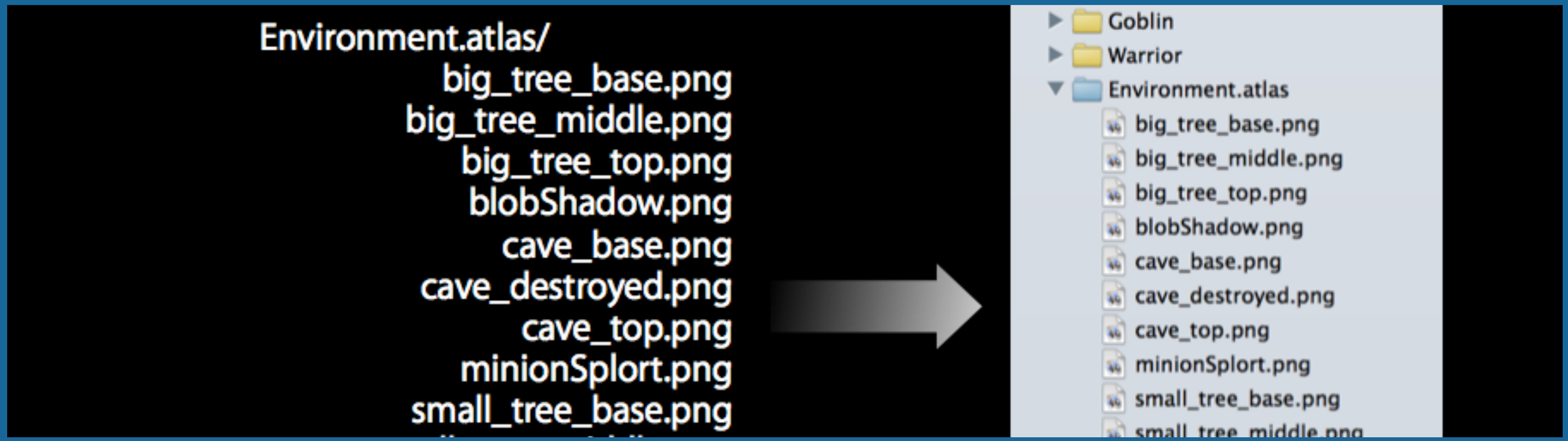
TEXTURES AND ALIASES



TEXTURES AND ALIASES

- Xcode generates atlases automatically
  - From individual image files
  - Retina and non-retina if provided
- Packed for maximum efficiency
  - Automatic rotation
  - Transparent edges trimmed
  - Up to 4096 x 4096

TEXTURES AND ALIASES



- Put your files in a “.atlas” folder
- Drag the folder into your project
- Xcode magic

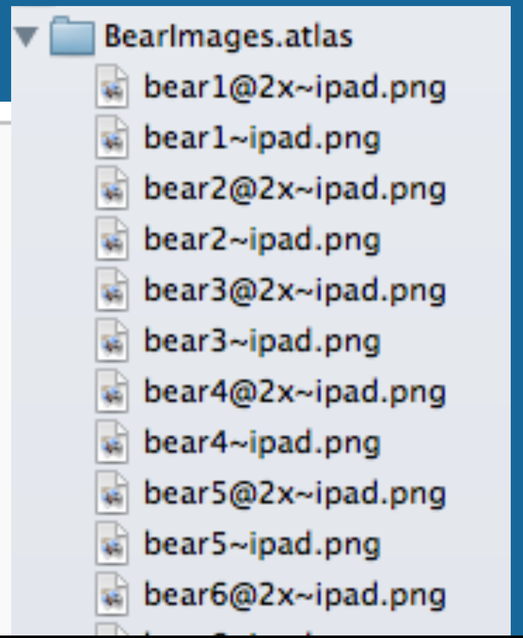
TEXTURES AND ATLASES

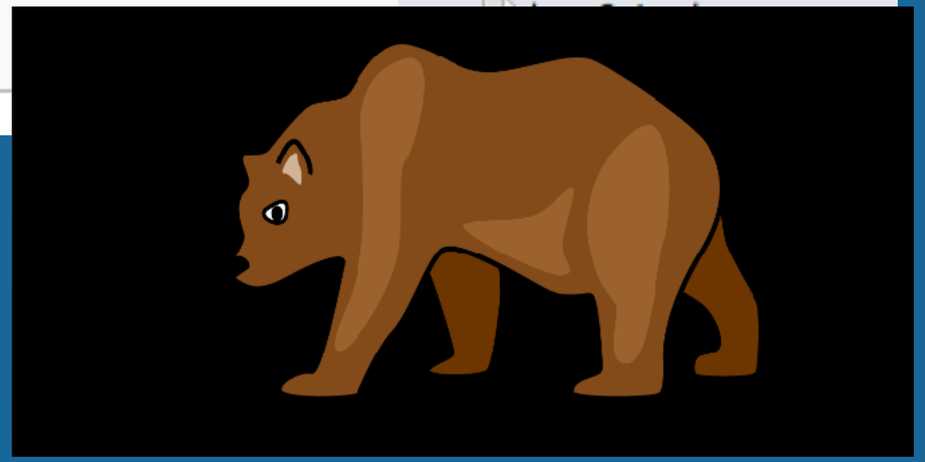
```
let texture = SKTexture(imageNamed: "BadGuy")
```

- Same call for atlas or stand alone image

TEXTURES AND ATLASES

```
let bearAnimatedAtlas = SKTextureAtlas(named: "BearImages")
var walkFrames = [SKTexture]()

let numImages = bearAnimatedAtlas.textureNames.count
for var i=1; i<=numImages/2; i++ {
  let bearTextureName = "bear\(i)"
  walkFrames.append(bearAnimatedAtlas.textureNamed(bearTextureName))
}

bearWalkingFrames = walkFrames
```





- Same call for atlas or stand alone image

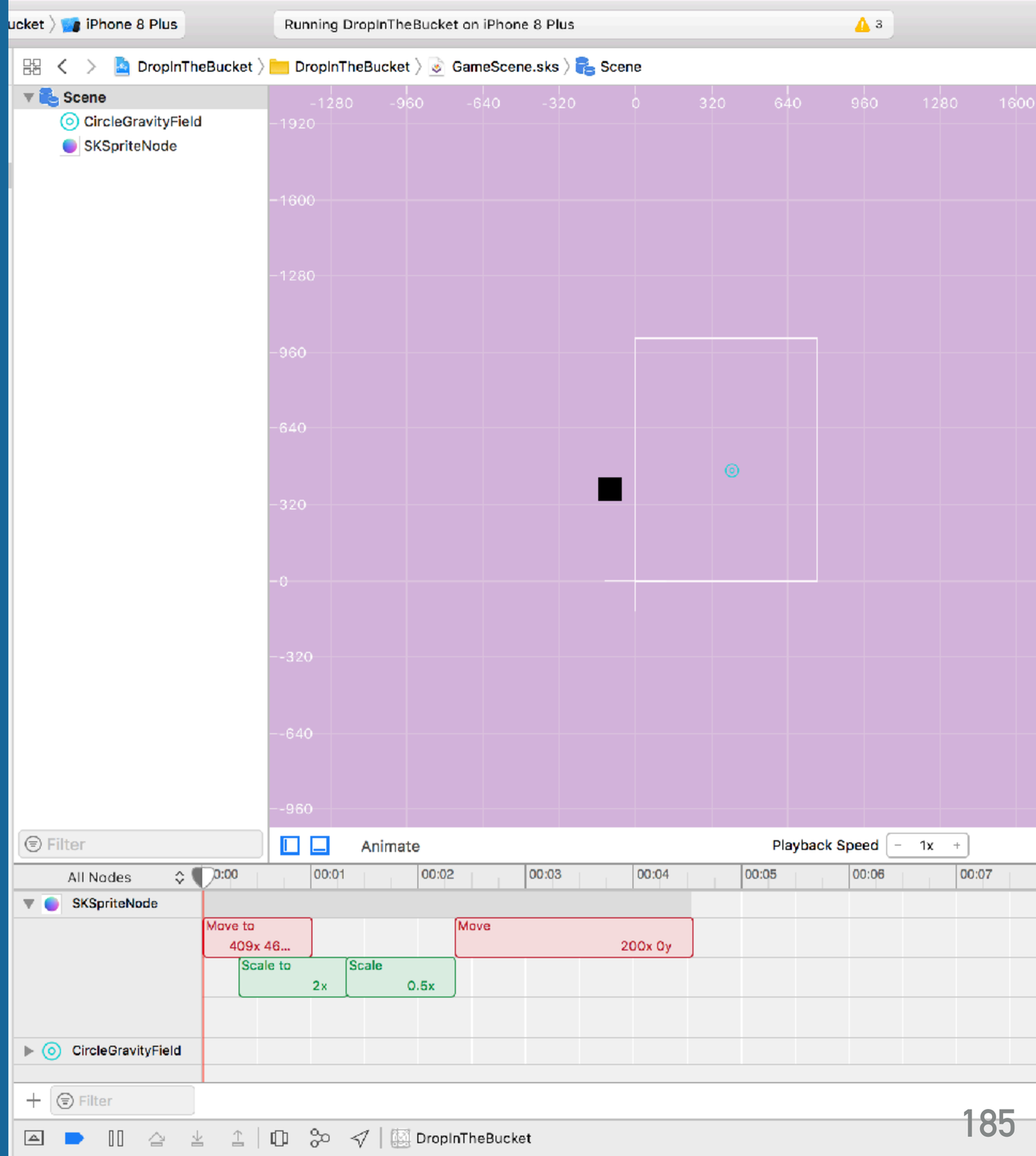
http://www.raywenderlich.com/89222/sprite-kit-animations-texture-atlases-swift

SOUND EFFECTS

SOUND EFFECT

SUBTITLE

- Play static sound through scene editor
- Play dynamic sounds programmatically



SOUND EFFECT

```
let shrink = SKAction.scale(to: 0.0, duration: 0.0)
let grow = SKAction.scale(to: 1.0, duration: 0.5)
let tink = SKAction.playSoundFileNamed("Tink.caf", waitForCompletion: false)
let sequence = SKAction.sequence([tink,shrink,grow])

self.run(sequence, withKey: "Appear-Action")
//self.removeAction(forKey: "Appear-Action")
```

Sounds are an action

Wait until its done before going on to next action

SOUND EFFECT

SUBTITLE

- Sounds should be preloaded for better performance
- Use AVAudioPlayer to call `prepareToPlay`

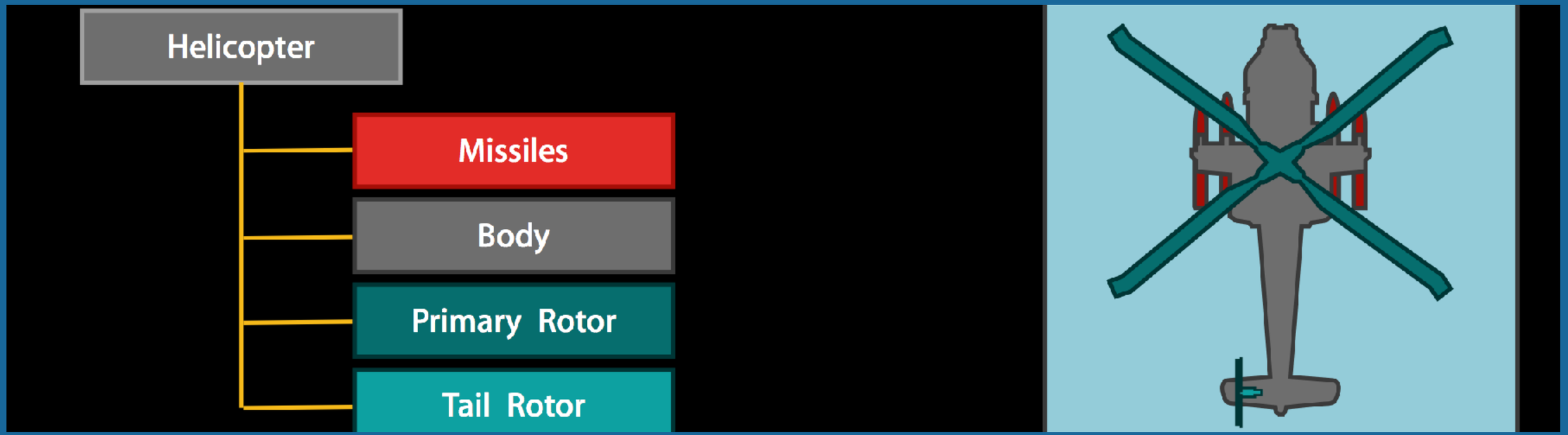
Sound node for background music

```
let bg:SKAudioNode = SKAudioNode(fileNamed: "music.m4a")
bg.autoplayLooped = true
self.addChild(bg)
```

Loop?

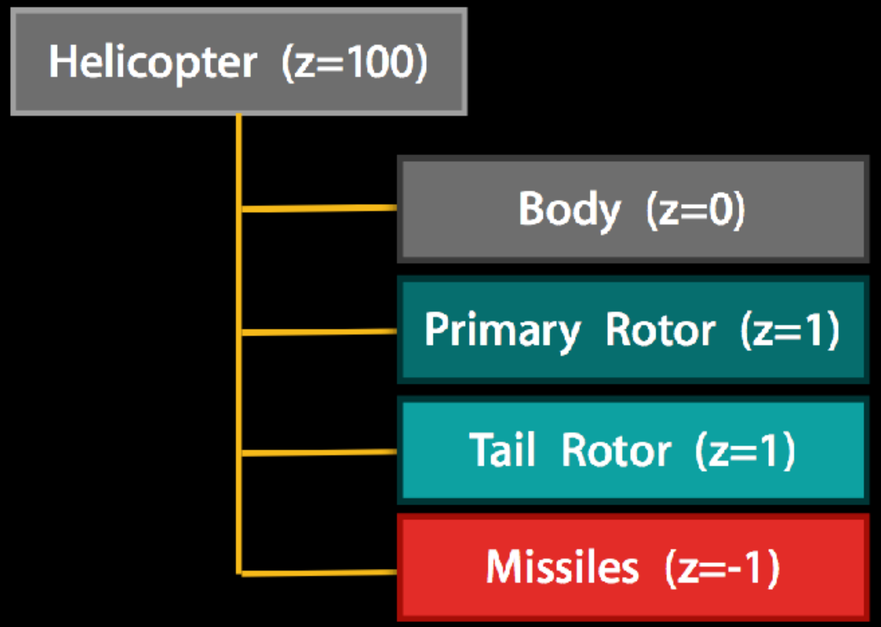
DRAWING ORDER AND PERFORMANCE

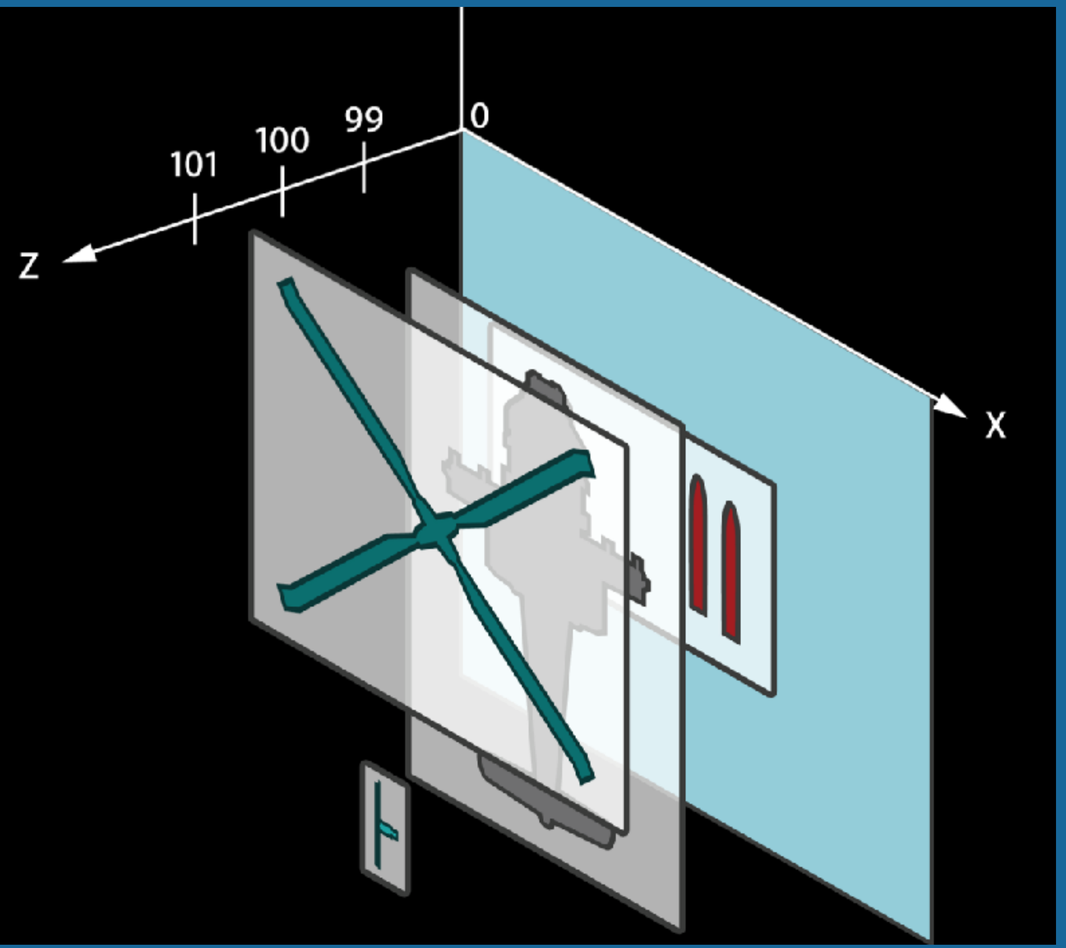
DRAWING ORDER



- Standard behavior
  - Parent draws content before rendering its children
  - Children are rendered in the order they appear in the child array

DRAWING ORDER





- zPosition property gives the node an explicit height (relative to parent)
- Nodes drawn in global Z order (more efficient across many nodes)

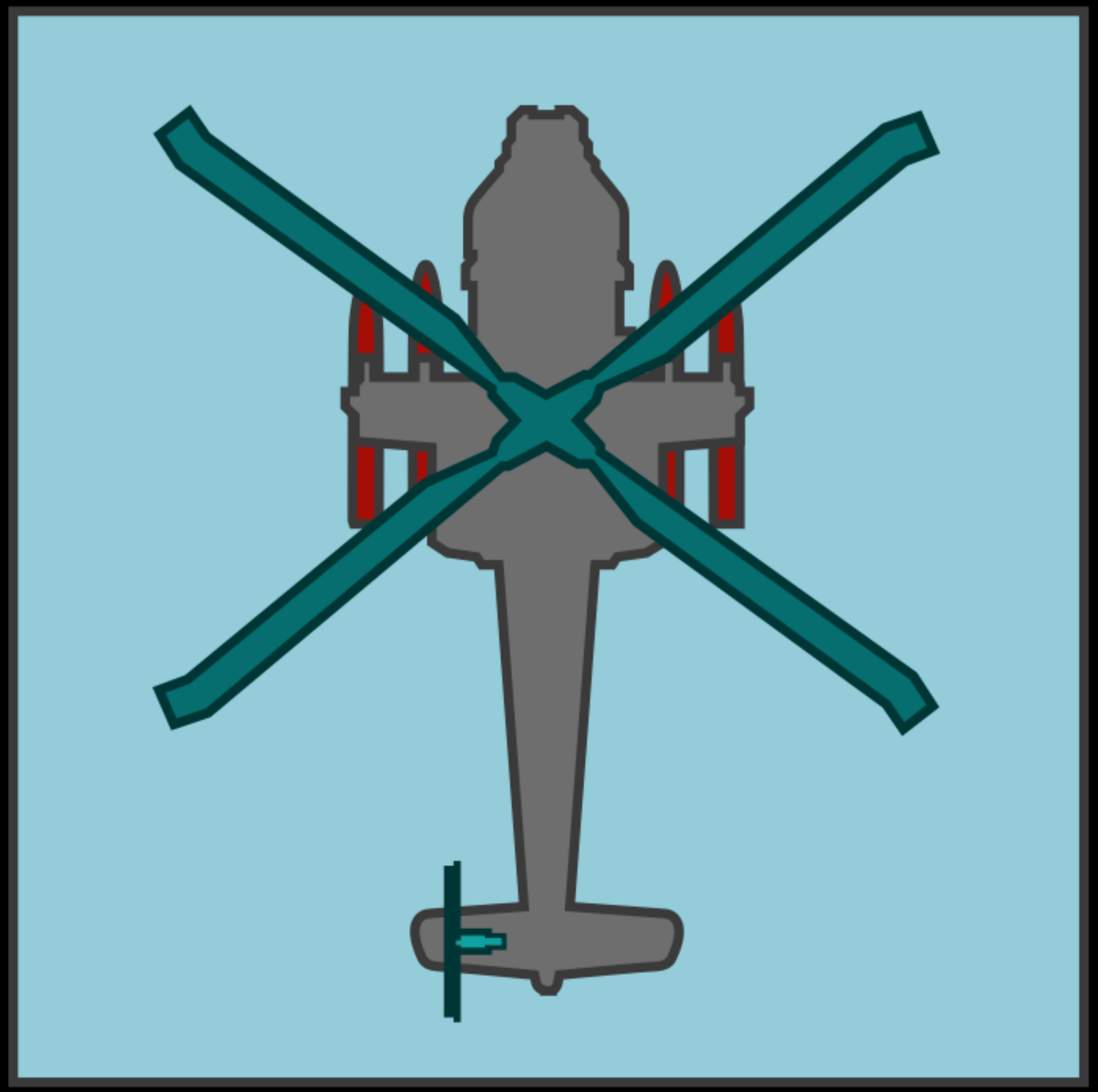
DRAWING ORDER

- SKScene `ignoresSiblingOrder` property
  - Allows nodes to be drawn in any order
  - Sprite Kit batches scene by state (fewer draw calls)
- Combine `ignoresSiblingOrder` with and `zPosition`
  - Each nodes global Z position is calculated
  - Drawn for lowest Z to highest Z
  - Sprite Kit sorts nodes at equal Z for optimal batching and determines optimal drawing

Only draw what you see

DRAWING ORDER

- Best practices
  - Compose scene as layers
    - Give objects common Z value per layer
    - Place overlapping objects in different layers



THE UNIVERSITY OF CHICAGO

ADVANCED iOS APPLICATION DEVELOPMENT

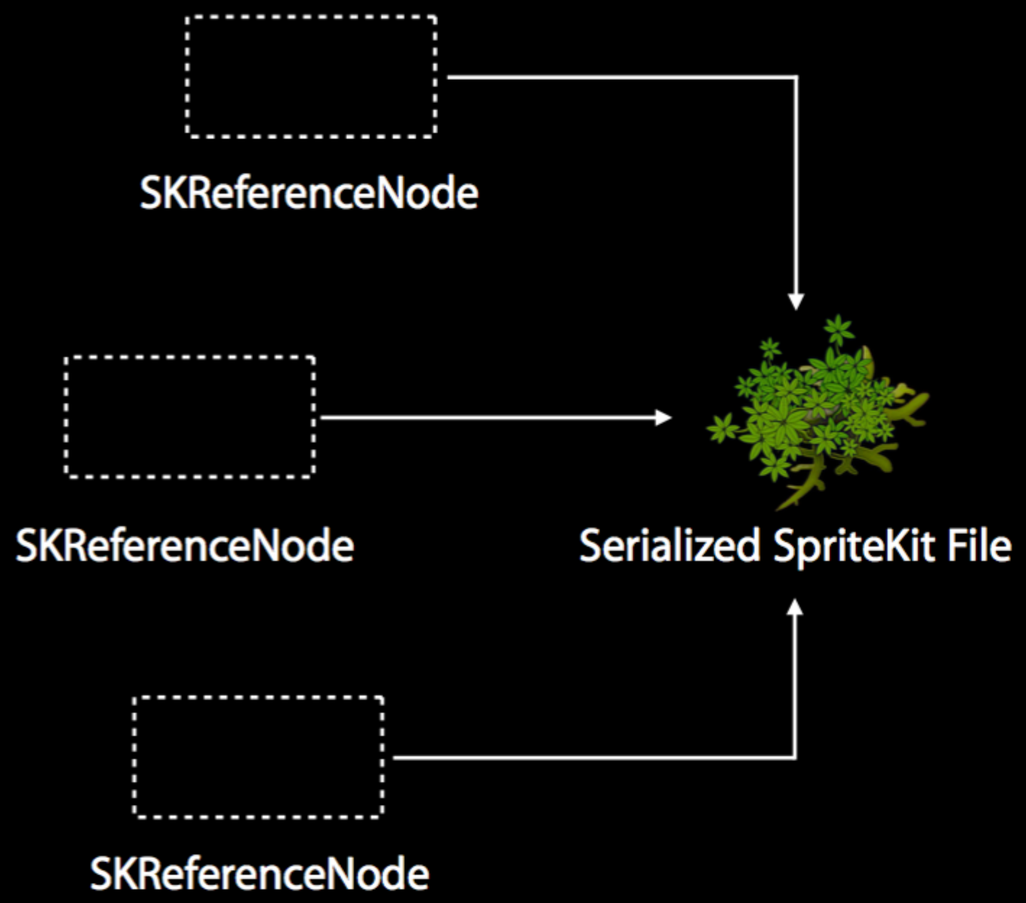
MPCS 51032 • SPRING 2020 • SESSION 6

"ADVANCED" SPRITEKIT TOPICS

“ADVANCED” SPRITEKIT TOPICS

SUBTITLE

- Referencing and Instancing
  - Leverage the NSCoding capability
  - Instances of SKAction and SKNode
  - Reusable components
  - Share across multiple projects



“ADVANCED” SPRITEKIT TOPICS

SUBTITLE

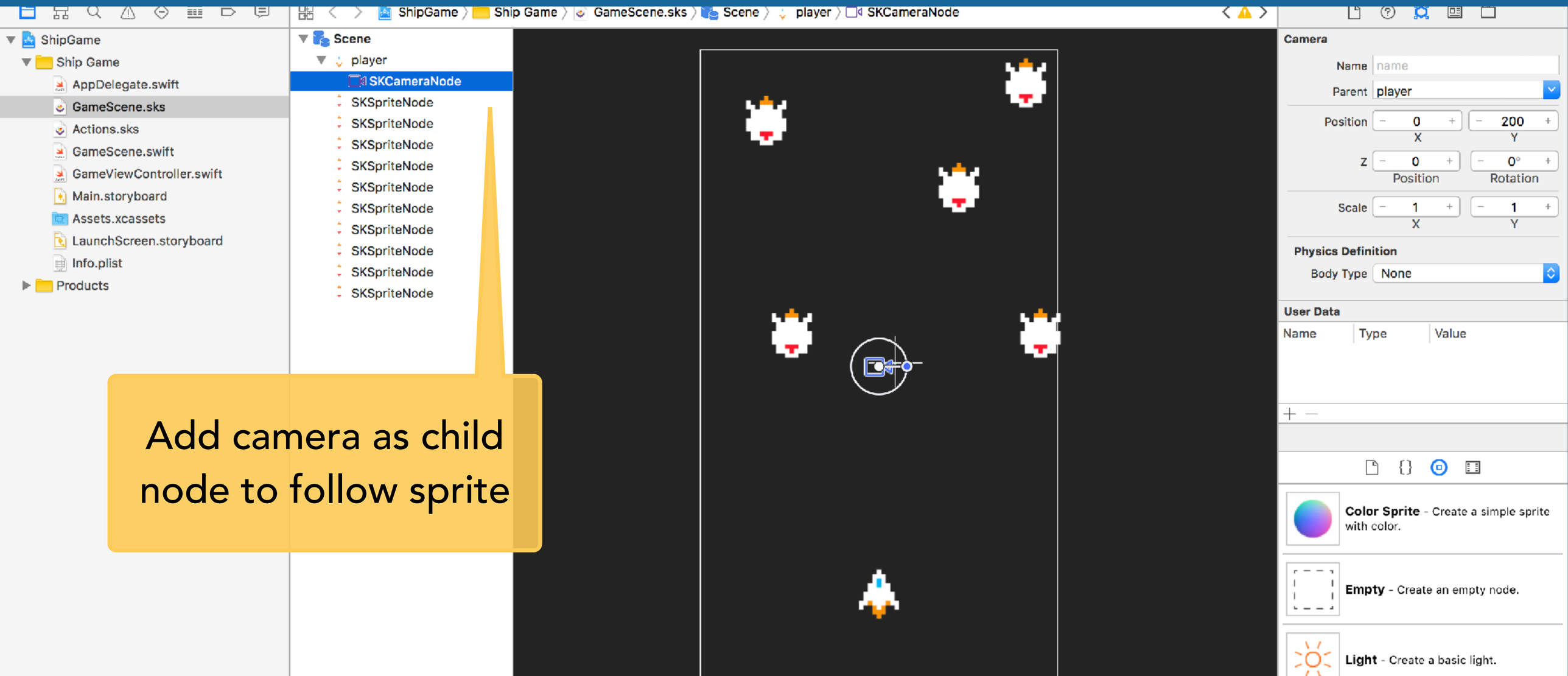
- Metal backed on devices that support it OpenGL on systems that don’t
- Zero action required for developers
- Better performance and battery life

“ADVANCED” SPRITEKIT TOPICS

SUBTITLE

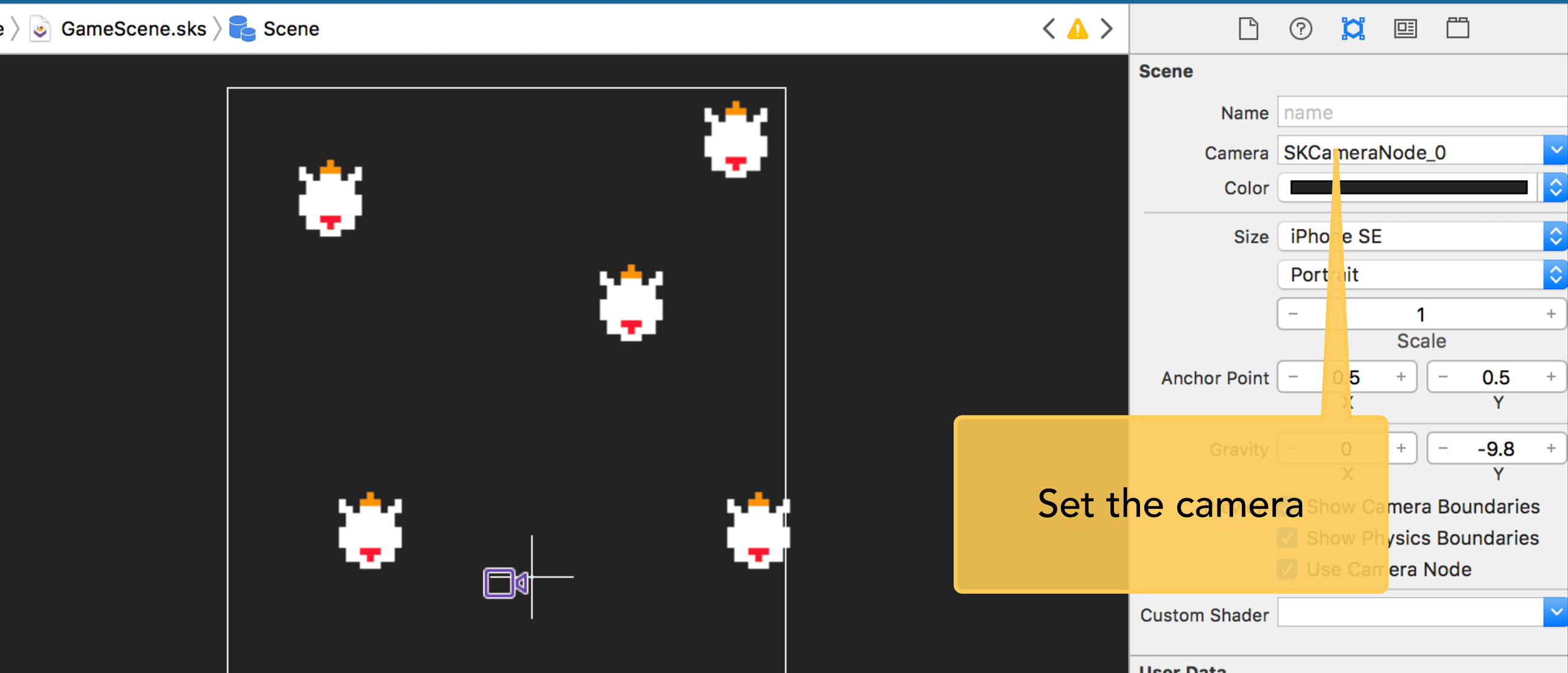
- SceneKit
  - High-level 3D graphics framework that helps you create 3D animated scenes and effects
  - Contains
    - Physics engine, a particle generator, and easy ways to script the actions of 3D objects
  - Renders 3D assets in 2D games

CAMERA NODES

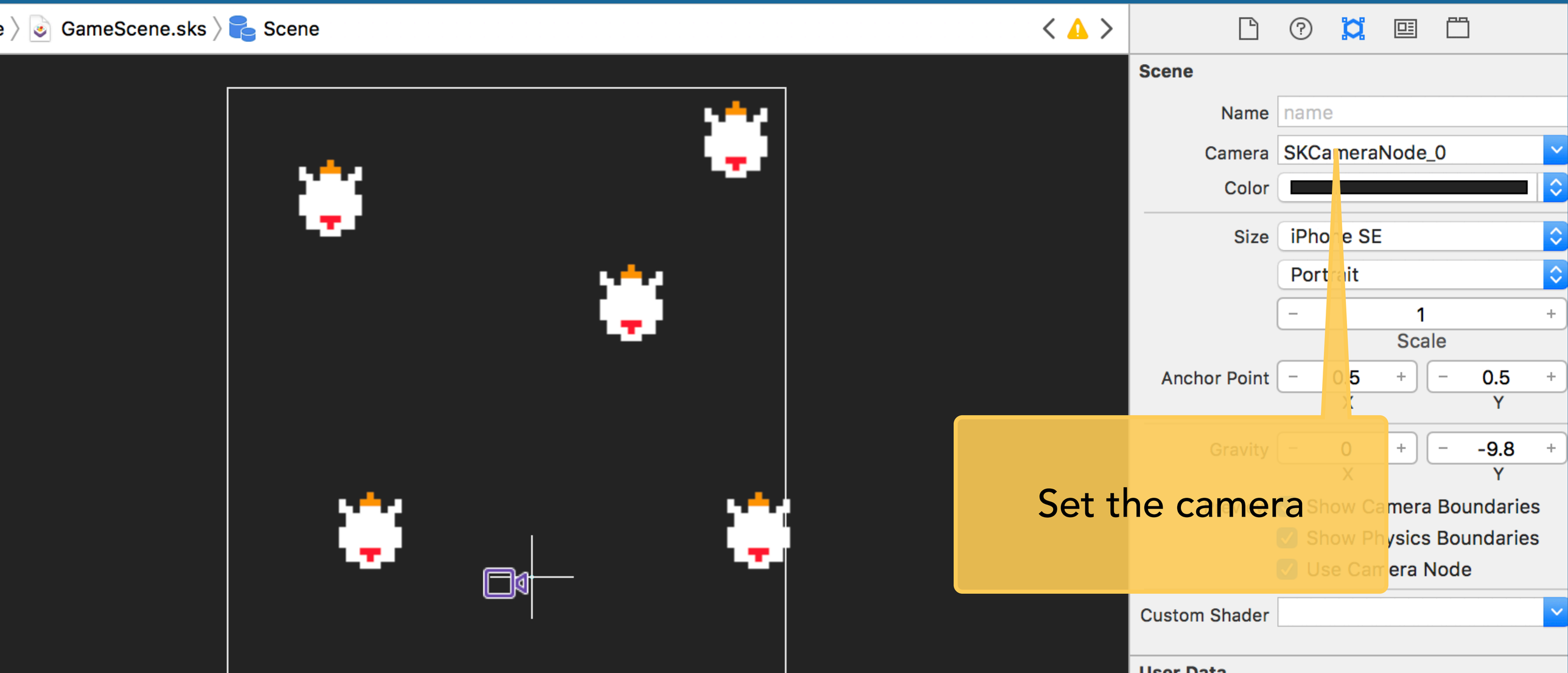


Add camera as child node to follow sprite

CAMERA NODES

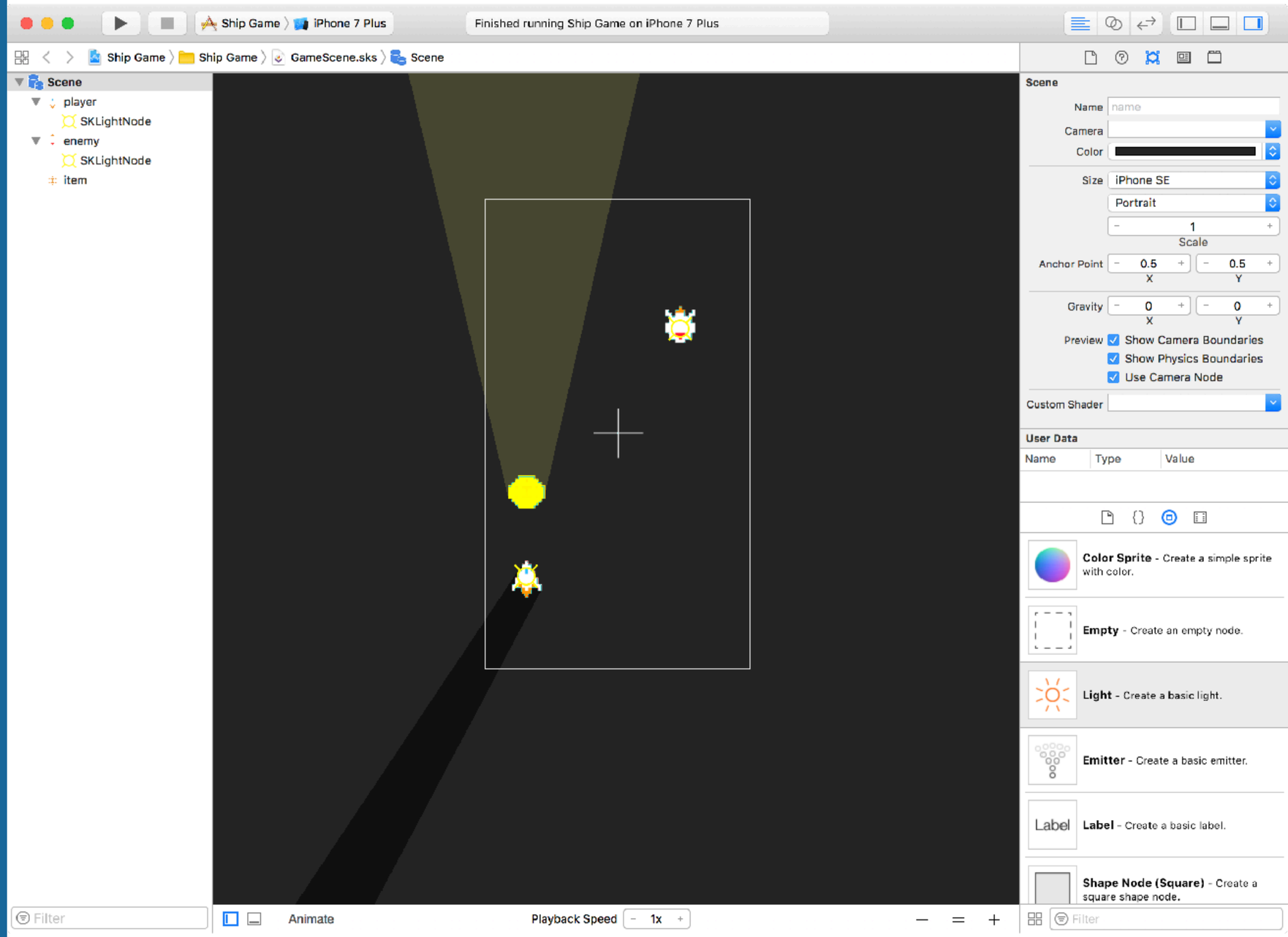


CAMERA NODES



CAMERA NODES

- Add light sources as
- Selectively add to nodes
- Modify in scene kit



CAMERA NODES

```
/// Called before each frame is rendered
override func update(currentTime: CFTimeInterval) {
  print("\(redBox?.position) --> \(camera?.position)")
  updateCamera()
}

/// Update the camera position by following around the red box.  There is only
/// a single camera assumed to be on the scene for this example.
func updateCamera() {
  if let camera = camera {
    camera.position = CGPoint(x: redBox!.position.x, y: redBox!.position.y)
  }
}
```

- Follow the `redBox` node around (update the camera position each run loop)
- You have to work out the scaling on the different nodes that will be visible

“ADVANCED” SPRITEKIT TOPICS

- GameplayKit
  - Entity/Component systems
  - State machines
  - Agents and behaviors
  - Pathfinding
  - AI strategists
  - Random sources
  - Rule systems

“ADVANCED” SPRITEKIT TOPICS

On This Page

About GameplayKit

GameplayKit is a collection of foundational tools and technologies for building games in iOS, OS X, and tvOS. Building, evolving, and maintaining a sophisticated game requires a well-planned design—GameplayKit provides architectural tools to help you design modular, scalable game architecture with minimal effort. Creating great games also requires deploying complex algorithms to solve the problems underlying common game mechanics—GameplayKit also provides standard implementations of such algorithms, allowing you to spend more time on the features that make your gameplay unique.

Because GameplayKit is independent of high-level game engine technologies, you can combine it with any of those technologies to build a complete game: SpriteKit for 2D games, SceneKit for 3D games, or a custom or third-party game engine using Metal or OpenGL ES. For games with less demanding graphics needs, you can even use GameplayKit with UIKit (in iOS or tvOS) or AppKit (in OS X).

GameplayKit provides seven core areas of functionality, which you can combine or use independently to create your game:

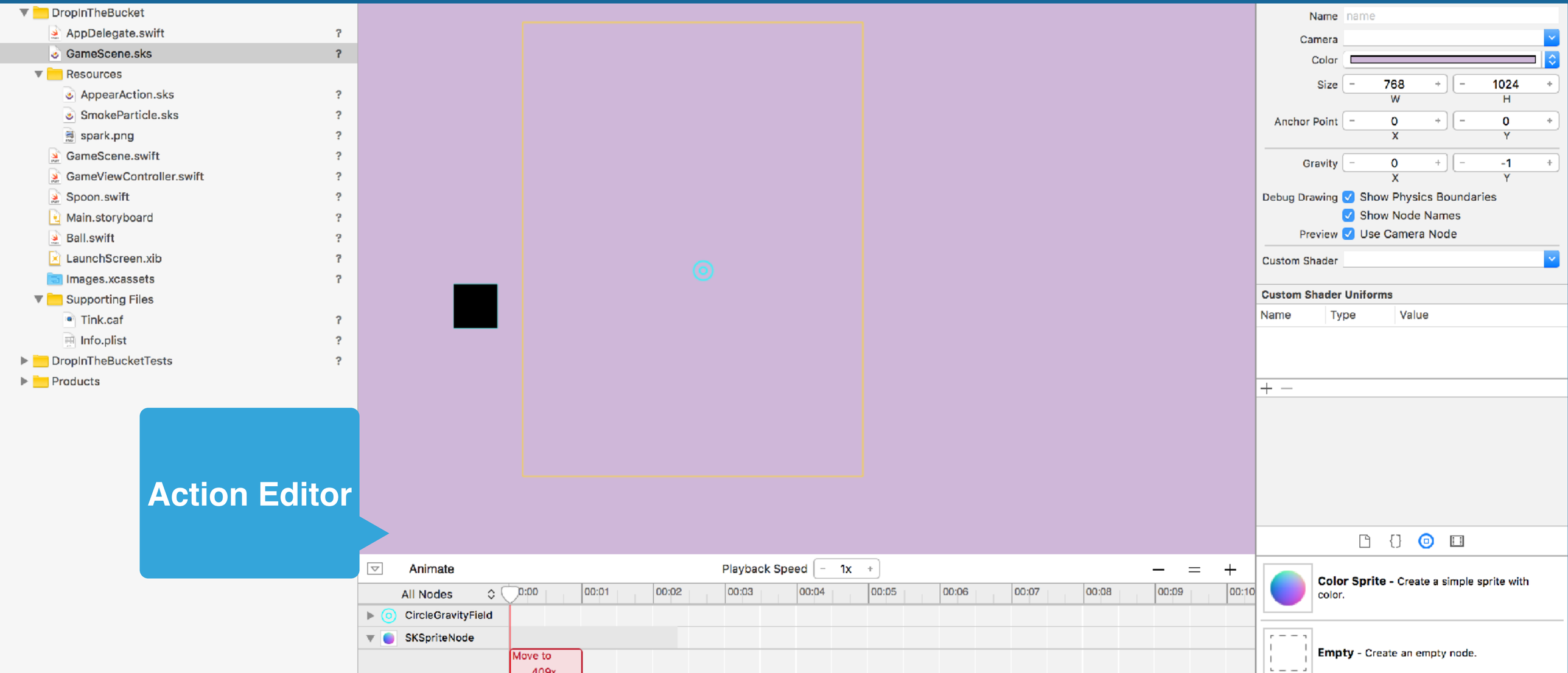
“ADVANCED” SPRITEKIT TOPICS

Component system, discussed in the Entities and Components chapter.

- *Dispenser: GameplayKit State Machine Basics* — A basic demonstration of GameplayKit’s state machine system, discussed in the State Machines chapter.
- *Pathfinder: GameplayKit Pathfinding Basics* — A basic demonstration of GameplayKit’s pathfinding system, discussed in the Pathfinding chapter.
- Maze: Getting Started with GameplayKit — A simplified classic arcade game, using many of the design and gameplay features in GameplayKit. This project is discussed in the Entities and Components, State Machines, Pathfinding, and Rule Systems chapters.
- *FourInARow: Using the GameplayKit Minmax Strategist for Opponent AI* — A simple board game built using UIKit (iOS only), illustrating use of the `GKMinmaxStrategist` class and related protocols, discussed in The Minmax Strategist.
- *AgentsCatalog: Using the Agents System in GameplayKit* — A demonstration of the `GKAgent` class and several of the individual goals an agent can follow, as well as how to combine goals into complex behaviors. The Agents, Goals, and Behaviors chapter discusses this project.

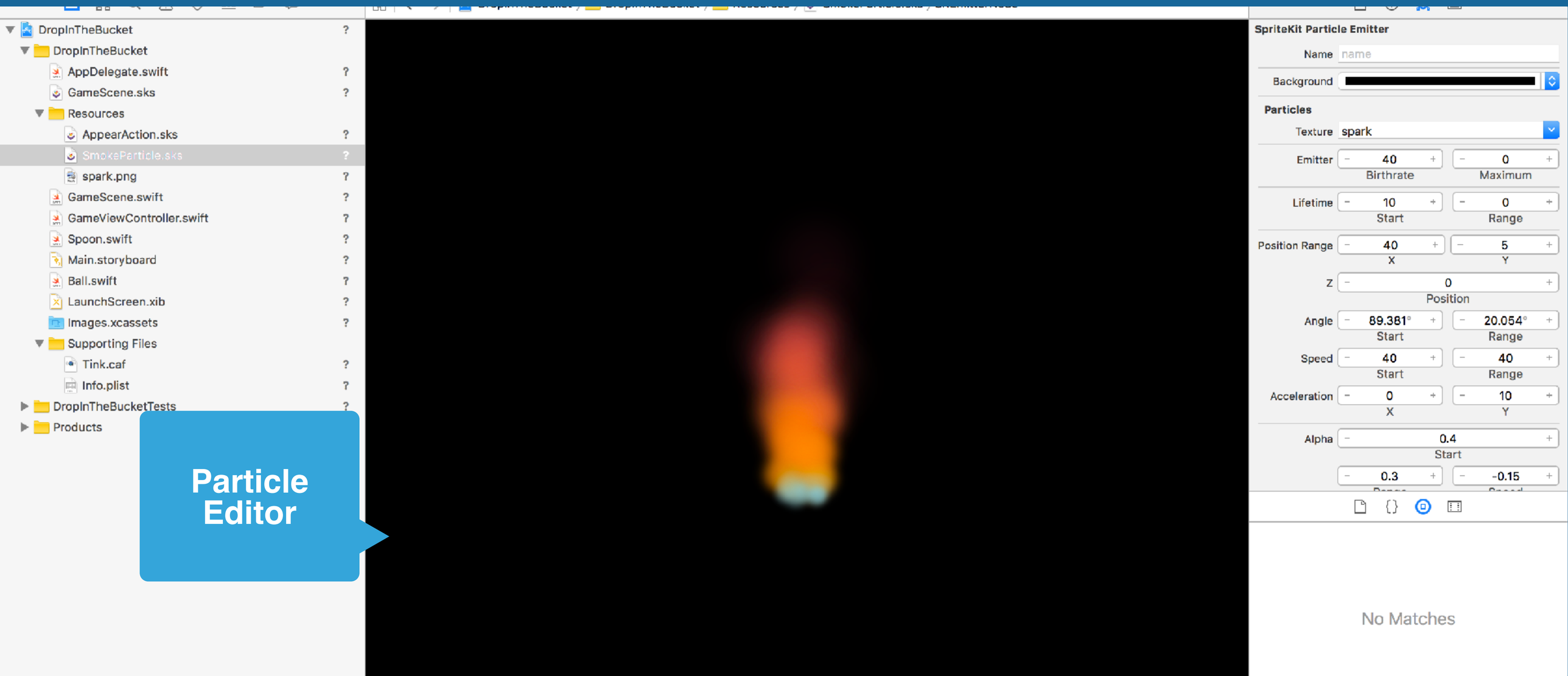
DROP IN THE BUCKET

DROP IN THE BUCKET



Action Editor

DROP IN THE BUCKET



DROP IN THE BUCKET

```
/// The "names" of the nodes in my scene
enum Category: String {
    case RainDropCategoryName = "RainDrop"
    case BallCategoryName = "Ball"
    case SpoonCategoryName = "Spoon"
    case HoleCategoryName = "Hole"
}

/// Each physics body identifies a category that it belongs to.  You can use
/// literals values but they need to be squares.  Check out this SO answer
/// http://stackoverflow.com/questions/24069703/how-to-define-category-bit-mask-enumeration-for-spritekit-in-swift
enum PhysicsCategory: UInt32 {
    case Edge = 0            //0x1 << 0
    case Ball = 1            //0x1 << 1
    case Spoon = 2           //0x1 << 2
    case Hole = 4            //0x1 << 3
    case RainDrop = 8        //0x1 << 4
    case GravityField = 16   //0x1 << 5
    case Floor = 32          //0x1 << 6
}
```

DROP IN THE BUCKET

```
override func didMoveToView(view: SKView) {
    setupPhysics()
    setupFloor()
    setupSpoon()

     // Add a gesture recognizer to the scene so a double tap makes it rain
    let doubleTap: UITapGestureRecognizer = UITapGestureRecognizer(target: self, action: #selector(GameScene.doubleTap(_:)))
    doubleTap.numberOfTapsRequired = 2
    view.addGestureRecognizer(doubleTap)

}
```

Scene is in responder chain

DROP IN THE BUCKET

PHYSICS

```
//
// MARK: - Physics Management
//

/// Add a gray rectangle that shows us where the physics body is located
/// to make it easier to visualize during demonstration.
func setupPhysics() {

    let physicsFrame = CGRectInset(frame, 50, 50)
    let gamePlayBackground = SKShapeNode(rect: physicsFrame)
    gamePlayBackground.fillColor = UIColor.lightGrayColor()
    gamePlayBackground.zPosition = -1
    addChild(gamePlayBackground)

    // Bounding edge of the screen in the physics simulation
    // This is commented out so you can see how the floor only interacts with
    // the green balls, not the blue rain drops
    let borderBody = SKPhysicsBody(edgeLoopFromRect: physicsFrame)
    borderBody.friction = 0
    physicsBody = borderBody

    // Pass our phyics contacts to this scene
    physicsWorld.contactDelegate = self
}
```

DROP IN THE BUCKET

PHYSICS

```
//
// MARK: - Node Management
//
func setupFloor() {
    let floor = SKShapeNode(rect: CGRectMake(0, 0, frame.width, 10))
    floor.fillColor = UIColor.orangeColor()
    floor.physicsBody = SKPhysicsBody(edgeLoopFromRect: floor.frame)
    floor.physicsBody!.categoryBitMask = PhysicsCategory.Floor.rawValue
    floor.physicsBody!.collisionBitMask = PhysicsCategory.Ball.rawValue
    addChild(floor)
}

func setupSpoon() {
    let spoon = Spoon(position:CGPointMake(100, 100))
    addChild(spoon)
}
```

DROP IN THE BUCKET

RAIN DROPS

```
//
// MARK: - Nodes and Effects
//

/// Drop a bunch of dots
func rain() {
    // let's create 300 bouncing cubes
    for i in 1..<100 {
        let randomX: CGFloat = CGFloat(arc4random_uniform(375))
        let randomY: CGFloat = CGFloat(600+i)
        let shape = SKShapeNode(circleOfRadius: 10)

        shape.name = Category.RainDropCategoryName.rawValue
        shape.fillColor = UIColor.blueColor()
        shape.lineWidth = 10
        shape.position = CGPoint(x: randomX, y: randomY)
        print("\(i) position:\(shape.position.x) \(shape.position.y)")

        shape.physicsBody = SKPhysicsBody(circleOfRadius: shape.frame.size.width/2)
        shape.physicsBody!.dynamic = true

        // Allow the balls to bounce off eachother
        // to see how the collision bit mask works uncomment PhysicsCategory.Floor
        shape.physicsBody!.collisionBitMask = PhysicsCategory.RainDrop.rawValue //| PhysicsCategory.Floo
        shape.physicsBody!.categoryBitMask = PhysicsCategory.RainDrop.rawValue
        shape.physicsBody!.contactTestBitMask = 0 // Don't report any collisions to the delegate

        self.addChild(shape)
    }
```

Double tap rain

DROP IN THE BUCKET

SPOON

```
class Spoon: SKSpriteNode {

    init(position: CGPoint) {
        let texture = SKTexture(imageNamed: "Spoon")
        super.init(texture: texture, color: UIColor(), size: texture.size())
        self.position = position
        self.userInteractionEnabled = true
        self.zRotation = CGFloat(DegreesToRadians(0))

        self.physicsBody = SKPhysicsBody(texture: texture, size: texture.size())
        self.physicsBody!.friction = 0.8
        self.physicsBody!.dynamic = false
        self.physicsBody!.categoryBitMask = PhysicsCategory.Spoon.rawValue
    }

    required init(coder aDecoder: NSCoder) {
        fatalError("NSCoding not supported")
    }


    override func touchesMoved(touches: Set<UITouch>, withEvent event: UIEvent?) {
        let touch = touches.first

        for touch in (touches ) {
            let location = touch.locationInNode(self.parent!)
            let touchedNode = nodeAtPoint(location)
            touchedNode.position = location
        }
    }

}
```

Create from texture (with alpha)

Move to touch point

DROP IN THE BUCKET

BALL

```
class Ball: SKShapeNode {

    let number: Int = Int(arc4random_uniform(10) + 1)
    var radius: CGFloat!
    var circumference: CGFloat!

    override init() {
        super.init()
        circumference = CGFloat(number * 10)
        radius = CGFloat(circumference / 2)
    }

    required init?(coder aDecoder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    convenience init(position: CGPoint) {
        self.init()
        self.path = CGPathCreateWithEllipseInRect(CGRectMake(-radius, -radius, circumference, circumference), nil)
        self.position = position
        self.fillColor = UIColor.greenColor()
        self.strokeColor = UIColor.greenColor()

        self.physicsBody = SKPhysicsBody(circleOfRadius: radius)
        self.physicsBody!.allowsRotation = false
        self.physicsBody!.friction = 0
        self.physicsBody!.restitution = 0.2
        self.physicsBody!.linearDamping = 0
        self.physicsBody!.angularDamping = 0

        self.physicsBody!.categoryBitMask = PhysicsCategory.Ball.rawValue
        self.physicsBody!.contactTestBitMask = PhysicsCategory.Spoon.rawValue | PhysicsCategory.Hole.rawValue
        self.physicsBody!.collisionBitMask = PhysicsCategory.Spoon.rawValue | PhysicsCategory.Hole.rawValue | PhysicsCategory.Floor.rawValue

        self.physicsBody!.fieldBitMask = PhysicsCategory.GravityField.rawValue


        let numberLabel = SKLabelNode(fontNamed: "Avenir-Next")
        numberLabel.fontSize = CGFloat(circumference - 1)
        numberLabel.verticalAlignmentMode = .Center
        numberLabel.horizontalAlignmentMode = .Center
        numberLabel.text = "\(number)"


        self.addChild(numberLabel)

        let shrink = SKAction.scaleTo(0.0, duration: 0.0)
        let grow = SKAction.scaleTo(1.0, duration: 0.5)
        let tink = SKAction.playSoundFileNamed("Tink.caf", waitForCompletion: false)
        self.runAction(SKAction.sequence([tink,shrink,grow]))

    }
```

Shape node with a label node inside

Appear animation

DROP IN THE BUCKET

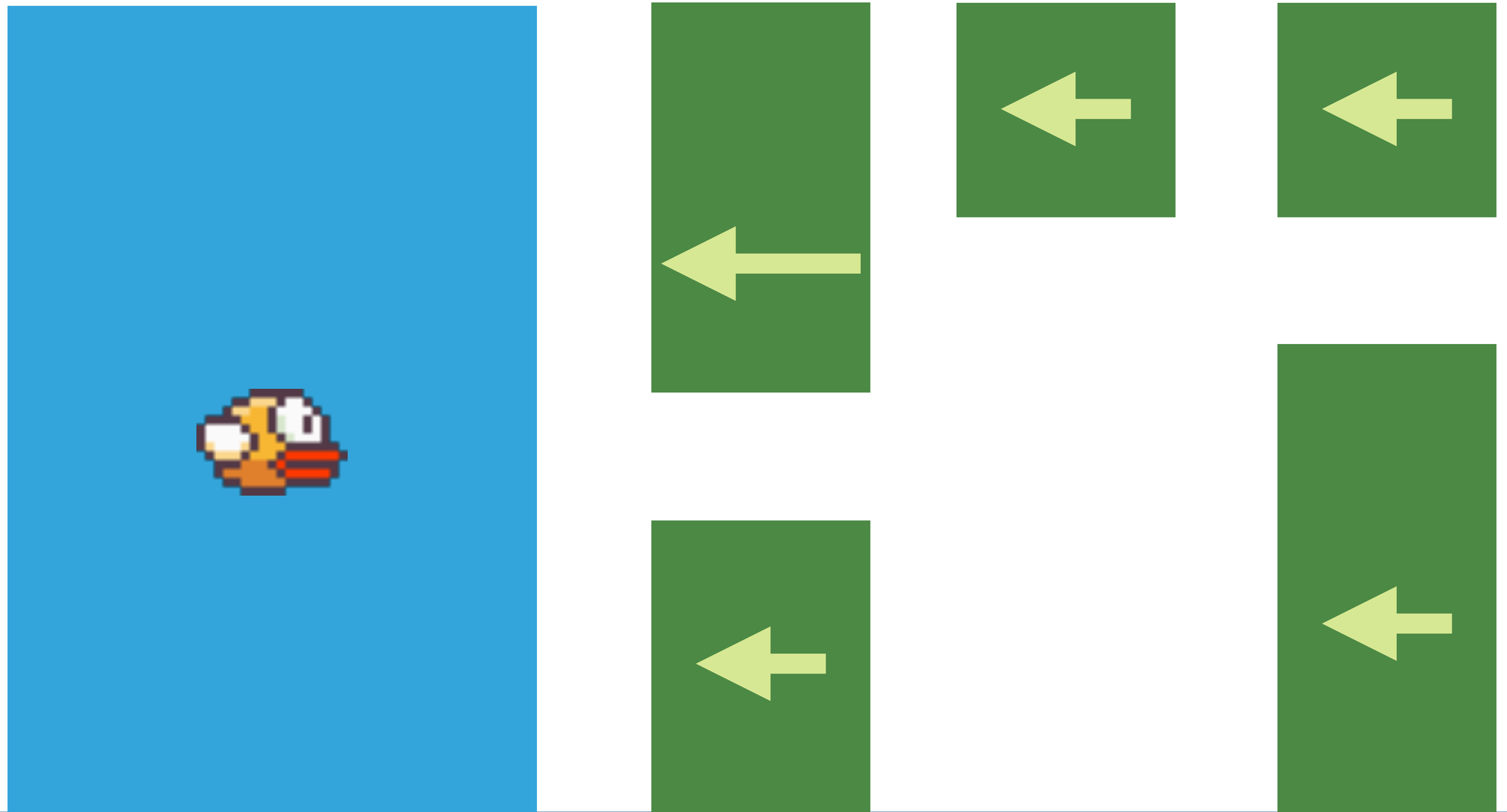
```
func didBeginContact(contact: SKPhysicsContact) {
    // Create local variables for two physics bodies
    var firstBody: SKPhysicsBody
    var secondBody: SKPhysicsBody

    // Assign the two physics bodies so that the one with the lower category
    // is always stored in firstBody.  Why?  This could be useful in your
    // logic.
    if contact.bodyA.categoryBitMask < contact.bodyB.categoryBitMask {
        firstBody = contact.bodyA
        secondBody = contact.bodyB
    } else {
        firstBody = contact.bodyB
        secondBody = contact.bodyA
    }
    print("Contact between \(firstBody.node?.name) and \(secondBody.node?.name)")

    // React to the contact between ball and the black hole (square).
    if firstBody.categoryBitMask == PhysicsCategory.Ball.rawValue &&
       secondBody.categoryBitMask == PhysicsCategory.Hole.rawValue {
        // TODO: Replace the log statement with display of Game Over Scene
        print("Hit bottom. First contact has been made.")
        let ball: Ball = (firstBody.node as? Ball)!
        ball.removeWithActions()
    }
```

Identify the contacts

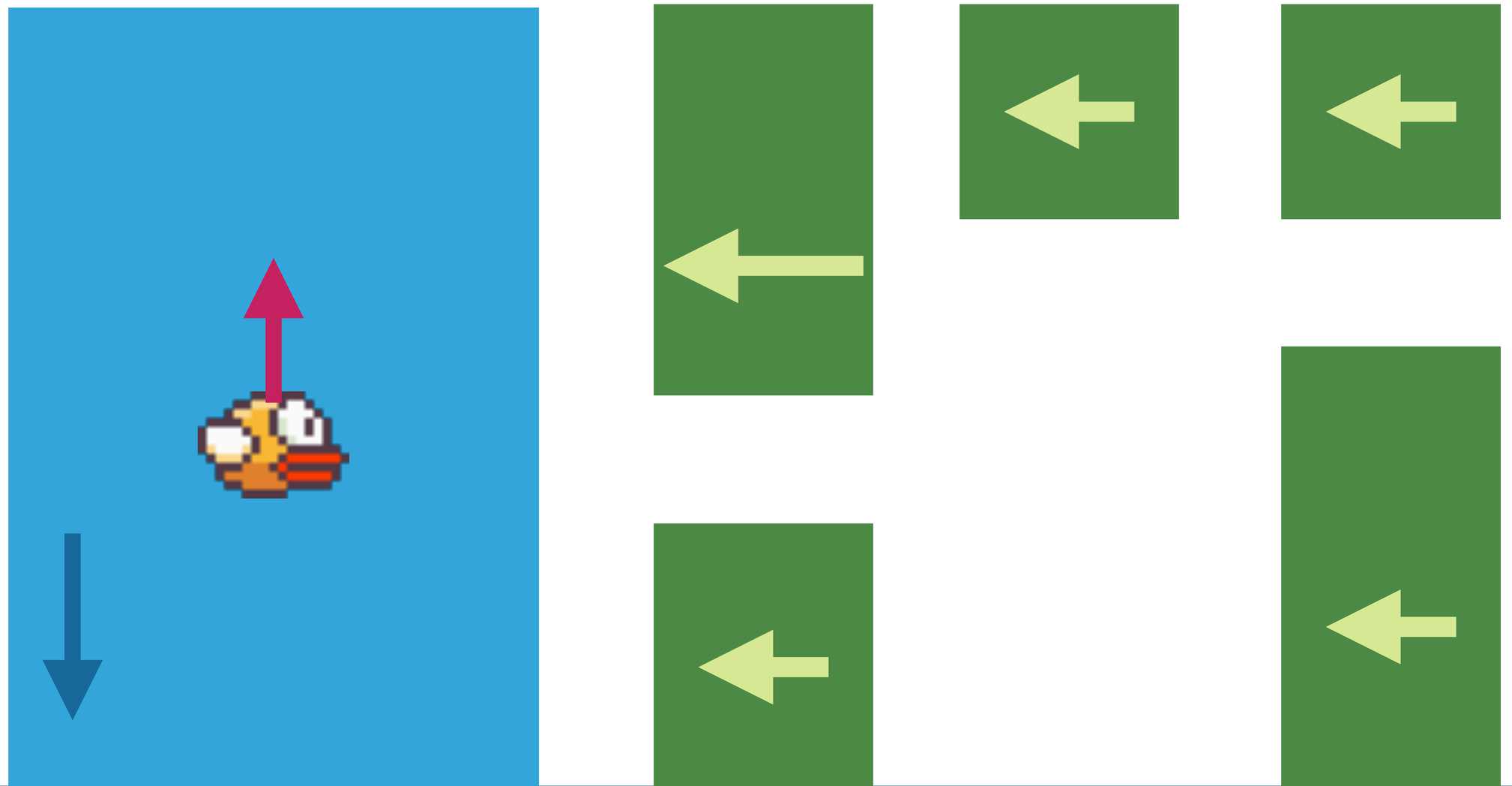
If we put the ball in the hole we win

FLAPPY BIRD

FLAPPY BIRD



FLAPPY BIRD



FLAPPY BIRD

```
class GameScene: SKScene {
  let verticalPipeGap = 150.0

  var bird:SKSpriteNode!
  var skyColor:SKColor!
  var pipeTextureUp:SKTexture!
  var pipeTextureDown:SKTexture!
  var movePipesAndRemove:SKAction!
  var moving:SKNode!
  var pipes:SKNode!
  var canRestart = Bool()
  var scoreLabelNode:SKLabelNode!
  var score = NSInteger()

  // Set our category bit masks
  let birdCategory: UInt32 = 1 << 0
  let worldCategory: UInt32 = 1 << 1
  let pipeCategory: UInt32 = 1 << 2
  let scoreCategory: UInt32 = 1 << 3
```

The nodes

FLAPPY BIRD

GROUND

Move Action

```
//
// Setup ground and skyline and use different timer intervals to create a
// simple parallax animation
//
let groundTexture = SKTexture(imageNamed: "land")
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 = CGPointMake(i * sprite.size.width, sprite.size.height / 2.0)
  sprite.runAction(moveGroundSpritesForever)
  moving.addChild(sprite)
}

//
// Skyline
//
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)
```

Create Node, add Action

FLAPPY BIRD

GROUND

```
//
// Skyline
//
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 = CGPointMake(i * sprite.size.width, sprite.size.height / 2.0 + groundTexture.size().height * 2.0)
  sprite.runAction(moveSkySpritesForever)
  moving.addChild(sprite)
}
```

Move Action

Create Node, add Action

FLAPPY BIRD

PIPES

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

// 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])

// 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 Node

Move Action

Spawn in scene

FLAPPY BIRD

Create a pipe at random distances that contacts the bird

```
func spawnPipes() {
  let pipePair = SKNode()
  pipePair.position = CGPointMake( self.frame.size.width + pipeTextureUp.size().width * 2, 0 )
  pipePair.zPosition = -10

  let height = UInt32( UInt(self.frame.size.height / 4) )
  let y = arc4random() % height + height

  let pipeDown = SKSpriteNode(texture: pipeTextureDown)
  pipeDown.setScale(2.0)
  pipeDown.position = CGPointMake(0.0, CGFloat(Double(y)) + pipeDown.size.height + CGFloat(verticalPipeGap))


  pipeDown.physicsBody = SKPhysicsBody(rectangleOfSize: pipeDown.size)
  pipeDown.physicsBody?.dynamic = false
  pipeDown.physicsBody?.categoryBitMask = pipeCategory
  pipeDown.physicsBody?.contactTestBitMask = birdCategory
  pipePair.addChild(pipeDown)

  let pipeUp = SKSpriteNode(texture: pipeTextureUp)
  pipeUp.setScale(2.0)
  pipeUp.position = CGPointMake(0.0, CGFloat(Double(y)))

  pipeUp.physicsBody = SKPhysicsBody(rectangleOfSize: pipeUp.size)
  pipeUp.physicsBody?.dynamic = false
  pipeUp.physicsBody?.categoryBitMask = pipeCategory
  pipeUp.physicsBody?.contactTestBitMask = birdCategory
  pipePair.addChild(pipeUp)

  let contactNode = SKNode()
  contactNode.position = CGPointMake( pipeDown.size.width + bird.size.width / 2, CGRectGetMidY( self.frame ) )
  contactNode.physicsBody = SKPhysicsBody(rectangleOfSize: CGSizeMake( pipeUp.size.width, self.frame.size.height ))
  contactNode.physicsBody?.dynamic = false
  contactNode.physicsBody?.categoryBitMask = scoreCategory
  contactNode.physicsBody?.contactTestBitMask = birdCategory
  pipePair.addChild(contactNode)

  pipePair.runAction(movePipesAndRemove)
  pipes.addChild(pipePair)

}
```

FLAPPY BIRD

```
// setup our bird
let birdTexture1 = SKTexture(imageNamed: "bird-01")
birdTexture1.filteringMode = .Nearest
let birdTexture2 = SKTexture(imageNamed: "bird-02")
birdTexture2.filteringMode = .Nearest

//
// Flap the wings using a texture animation
//
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)

// configure physics body for our flappybird
bird.physicsBody = SKPhysicsBody(circleOfRadius: bird.size.height / 2.0)
bird.physicsBody?.dynamic = true
bird.physicsBody?.allowsRotation = false

// check if our flappybird collides with a pipe or the world (aka ground) and exclude collision with the score
bird.physicsBody?.categoryBitMask = birdCategory
bird.physicsBody?.collisionBitMask = worldCategory | pipeCategory
bird.physicsBody?.contactTestBitMask = worldCategory | pipeCategory

self.addChild(bird)
```

Create bird with a texture

FLAPPY BIRD

CONTACTS

```
didBeginContact(contact: SKPhysicsContact) {

moving.speed > 0 {

// check if our flappybird made it through a gap and increase score
if ( contact.bodyA.categoryBitMask & scoreCategory ) == scoreCategory || ( contact.bodyB.categoryBitMask & scoreCategory ) == scoreCategory {
  // Bird has contact with score entity
  score = score + 1
  scoreLabelNode.text = String(score)

  // Briefly scale up the score counter to provide snazzy visual feedback
  scoreLabelNode.runAction(SKAction.sequence([SKAction.scaleTo(1.5, duration:NSTimeInterval(0.1)), SKAction.scaleTo(1.0, duration:NSTimeInterval(0.1)
} else {

  // whoops, our flappybird hit something else, uh oh
  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")
}
```

Score zone

Pipe

THE UNIVERSITY OF CHICAGO

ADVANCED iOS APPLICATION DEVELOPMENT

MPCS 51032 • SPRING 2020 • SESSION 6