Building iOS 17 Apps with Xcode Storyboards

Neil Smyth

Payload publishing

Building iOS 17 Apps with Xcode Storyboards

Building iOS 17 Apps with Xcode Storyboards

ISBN-13: 978-1-951442-84-2

Rev: 1.0

Payload
publishing

https://www.payloadbooks.com

Chapter 93

93. An iOS 17 Sprite Kit Level Editor Game Tutorial

In this chapter, many of the Sprite Kit Framework features outlined in the previous chapter will be used to create a game-based app. In particular, this tutorial will demonstrate the practical use of scenes, textures, sprites, labels, and actions. In addition, the app created in this chapter will also use physics bodies to demonstrate the use of collisions and simulated gravity.

This tutorial will also demonstrate using the Xcode Sprite Kit Level, Live, and Action editors combined with Swift code to create a Sprite Kit-based game.

93.1 About the Sprite Kit Demo Game

The game created in this chapter consists of a single animated character that shoots arrows across the scene when the screen is tapped. For the game's duration, balls fall from the top of the screen, with the objective being to hit as many balls as possible with the arrows.

The completed game will comprise the following two scenes:

- **GameScene** – The scene which appears when the game is first launched. The scene will announce the game's name and invite the user to touch the screen to begin the game. The game will then transition to the second scene.

- **ArcheryScene** – The scene where the game-play takes place. Within this scene, the archer and ball sprites are animated, and the physics behavior and collision detection are implemented to make the game work.

In terms of sprite nodes, the game will include the following:

- **Welcome Node** – An SKLabelNode instance that displays a message to the user on the Welcome Scene.

- **Archer Node** – An SKSpriteNode instance to represent the archer game character. The animation frames that cause the archer to load and launch an arrow are provided via a sequence of image files contained within a texture atlas.

- **Arrow Node** – An SKSpriteNode instance used to represent the arrows as the archer character shoots them. This node has associated with it a physics body so that collisions can be detected and to make sure it responds to gravity.

- **Ball Node** – An SKSpriteNode represents the balls that fall from the sky. The ball has associated with it a physics body for gravity and collision detection purposes.

- **Game Over Node** – An SKLabelNode instance that displays the score to the user at the end of the game.

The overall architecture of the game can be represented hierarchically, as outlined in Figure 93-1:

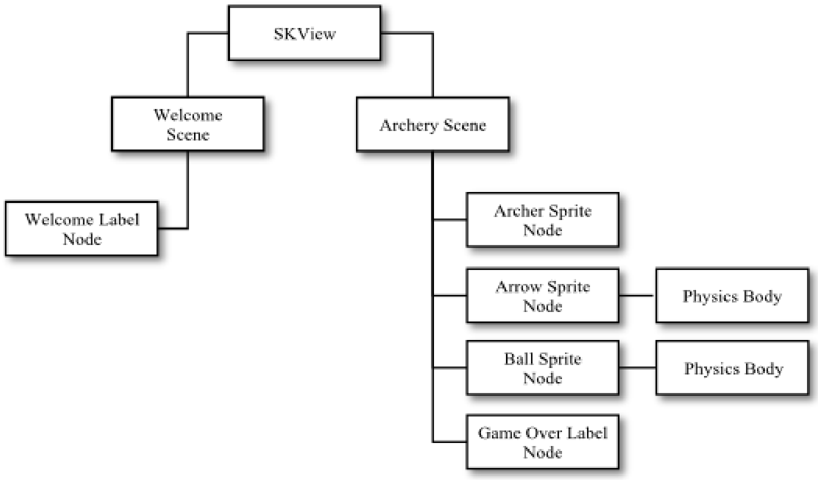


Figure 93-1

In addition to the nodes outlined above, the Xcode Live and Action editors will be used to implement animation and audio actions, which will be triggered from within the app's code.

93.2 Creating the SpriteKitDemo Project

To create the project, launch Xcode and select the *Create a new Xcode project* option from the welcome screen (or use the *File -> New -> Project…*) menu option. Next, on the template selection panel, choose the iOS *Game* template option. Click on the *Next* button to proceed and on the resulting options screen, name the product *SpriteKitDemo* and choose *Swift* as the language in which the app will be developed. Finally, set the Game Technology menu to *SpriteKit*. Click *Next* and choose a suitable location for the project files. Once selected, click *Create* to create the project.

93.3 Reviewing the SpriteKit Game Template Project

The selection of the SpriteKit Game template has caused Xcode to create a template project with a demonstration incorporating some pre-built Sprite Kit behavior. This template consists of a View Controller class (*GameViewController.swift*), an Xcode Sprite Kit scene file (*GameScene.sks*), and a corresponding GameScene class file (*GameScene.swift*). The code within the *GameViewController.swift* file loads the scene design contained within the *GameScene.sks* file and presents it on the view to be visible to the user. This, in turn, triggers a call to the *didMove(to view:)* method of the GameScene class as implemented in the *GameScene.swift* file. This method creates an SKLabelNode displaying text that reads "Hello, World!".

The GameScene class also includes a variety of touch method implementations that create SKShapeNode instances into which graphics are drawn when triggered. These nodes, in turn, are displayed in response to touches and movements on the device screen. To see the template project in action, run it on a physical device or the iOS simulator and perform tapping and swiping motions on the display.

As impressive as this may be, given how little code is involved, this bears no resemblance to the game that will be created in this chapter, so some of this functionality needs to be removed to provide a clean foundation on which to build. Begin the tidying process by selecting and editing the *GameScene.swift* file to remove the code to create and present nodes in the scene. Once modified, the file should read as follows:

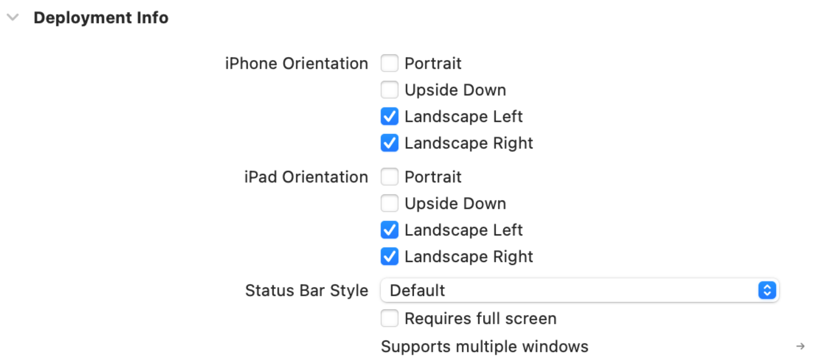
```
import SpriteKit
import GameplayKit
```

```
class GameScene: SKScene {

    override func didMove(to view: SKView) {

    }

    override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {

    }

    override func update(_ currentTime: TimeInterval) {
        // Called before each frame is rendered
    }
}
```

With these changes, it is time to start creating the SpriteKitDemo game.

93.4 Restricting Interface Orientation

The game created in this tutorial assumes that the device on which it is running will be in landscape orientation. Therefore, to prevent the user from attempting to play the game with a device in portrait orientation, the *Device Orientation* properties for the project need to be restricted. To achieve this, select the *SpriteKitDemo* entry located at the top of the Project Navigator and, in the resulting *General* settings panel, change the device orientation settings so that only the *Landscape* options are selected both for iPad and iPhone devices:

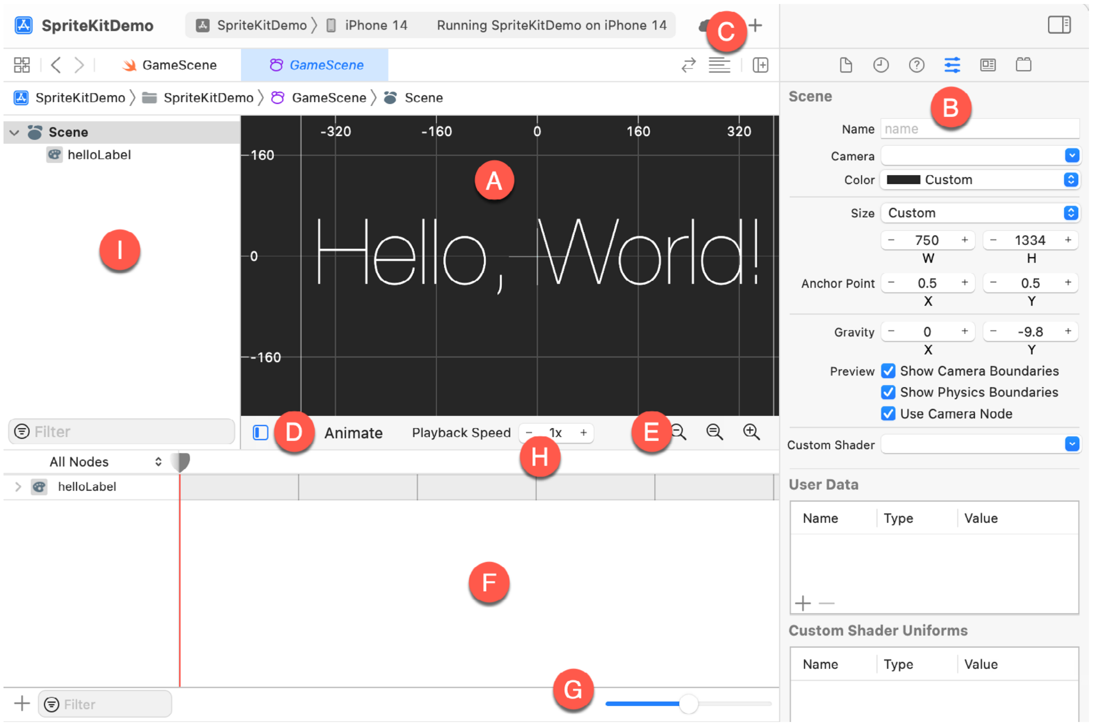

Deployment Info

iPhone Orientation: Portrait [ ], Upside Down [ ], Landscape Left [✓], Landscape Right [✓]
iPad Orientation: Portrait [ ], Upside Down [ ], Landscape Left [✓], Landscape Right [✓]
Status Bar Style: Default
Requires full screen [ ]
Supports multiple windows

Figure 93-2

93.5 Modifying the GameScene SpriteKit Scene File

As previously outlined, Xcode has provided a SpriteKit scene file (*GameScene.sks*) for a scene named GameScene together with a corresponding class declaration contained within the *GameScene.swift* file. The next task is to repurpose this scene to act as the welcome screen for the game. Begin by selecting the *GameScene.sks* file so that it loads into the SpriteKit Level Editor, as shown in Figure 93-3:



Scene
Name: name
Camera:
Color: Custom
Size: Custom
W: 750 H: 1334
Anchor Point: X: 0.5 Y: 0.5
Gravity: X: 0 Y: -9.8
Preview: ☑ Show Camera Boundaries ☑ Show Physics Boundaries ☑ Use Camera Node
Custom Shader:

User Data

Name	Type	Value

Custom Shader Uniforms

Name	Type	Value

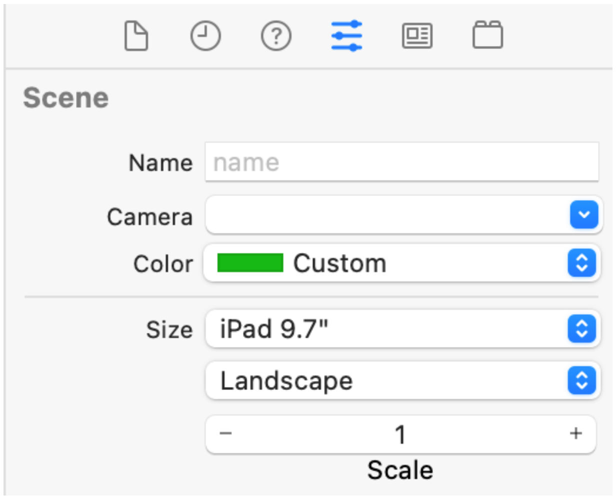
Figure 93-3

When working with the Level Editor to design SpriteKit scenes, there are several key areas of importance, each of which has been labeled in the above figure:

- **A – Scene Canvas** - This is the canvas onto which nodes may be placed, positioned, and configured.
- **B – Attribute Inspector Panel** - This panel provides a range of configuration options for the currently selected item in the editor panel. This allows SKNode and SKAction objects to be customized within the editor environment.
- **C – Library Button** – This button displays the Library panel containing a range of node and effect types that can be dragged and dropped onto the scene.
- **D – Animate/Layout Button** - Toggles between the editor's simulation and layout editing modes. Simulate mode provides a useful mechanism for previewing the scene behavior without compiling and running the app.
- **E – Zoom Buttons** – Buttons to zoom in and out of the scene canvas.
- **F – Live Editor** – The live editor allows actions and animations to be placed within a timeline and simulated within the editor environment. It is possible, for example, to add animation and movement actions within the live editor and play them back live within the scene canvas.
- **G – Timeline View Slider** – Pans back and forth through the view of the live editor timeline.
- **H – Playback Speed** – When in Animation mode, this control adjusts the playback speed of the animations and actions contained within the live editor panel.
- **I – Scene Graph View** – This panel provides an overview of the scene's hierarchy and can be used to select,

delete, duplicate and reposition scene elements within the hierarchy.

Within the scene editor, click on the "Hello, World!" Label node and press the keyboard delete key to remove it from the scene. With the scene selected in the scene canvas, click on the *Color* swatch in the Attribute Inspector panel and use the color selection dialog to change the scene color to a shade of green. Remaining within the Attributes Inspector panel, change the Size setting from *Custom* to *iPad 9.7"* in *Landscape* mode:



Scene

Name: name

Camera:

Color: 🟩 Custom

Size: iPad 9.7"

Landscape

– 1 +

Scale

Figure 93-4

Click on the button (marked C in Figure 93-3 above) to display the Library panel, locate the Label node object, and drag and drop an instance onto the center of the scene canvas. With the label still selected, change the *Text* property in the inspector panel to read "SpriteKitDemo – Tap Screen to Play". Remaining within the inspector panel, click on the T next to the font name and use the font selector to assign a 56-point *Marker Felt Wide* font to the label from the *Fun* font category. Finally, set the *Name* property for the label node to "welcomeNode". Save the scene file before proceeding.

With these changes complete, the scene should resemble that of Figure 93-5:

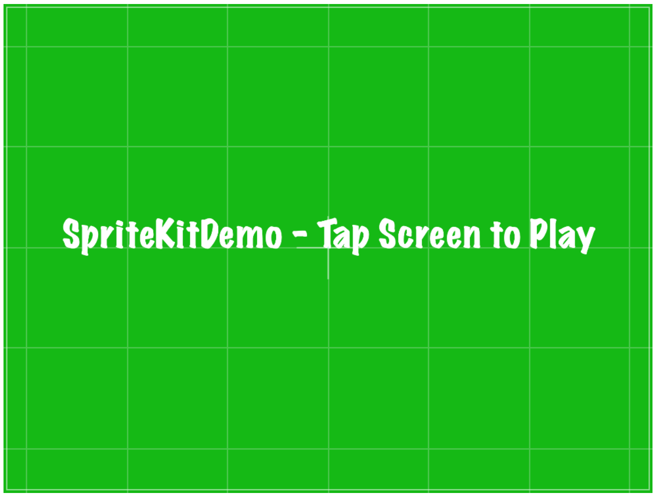


Figure 93-5

93.6 Creating the Archery Scene

As previously outlined, the game's first scene is a welcome screen on which the user will tap to begin playing within a second scene. Add a new class to the project to represent this second scene by selecting the *File -> New -> File…* menu option. In the file template panel, make sure that the *Cocoa Touch Class* template is selected in the main panel. Click on the *Next* button and configure the new class to be a subclass of *SKScene* named *ArcheryScene*. Click on the *Next* button and create the new class file within the project folder.

The new scene class will also require a corresponding SpriteKit scene file. Select *File -> New -> File…* once again, this time selecting *SpriteKit Scene* from the Resource section of the main panel (Figure 93-6). Click *Next*, name the scene *ArcheryScene* and click the *Create* button to add the scene file to the project.

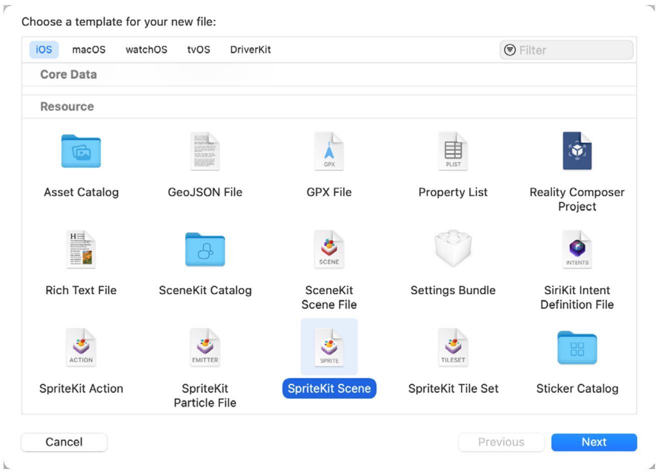


Figure 93-6

Edit the newly added *ArcheryScene.swift* file and modify it to import the SpriteKit Framework as follows:

```
import UIKit
import SpriteKit

class ArcheryScene: SKScene {

}
```

93.7 Transitioning to the Archery Scene

Clearly, having instructed the user to tap the screen to play the game, some code needs to be written to make this happen. This behavior will be added by implementing the *touchesBegan* method in the GameScene class. Rather than move directly to ArcheryScene, some effects will be added as an action and transition.

When implemented, the SKAction will cause the node to fade from view, while an SKTransition instance will be used to animate the transition from the current scene to the archery scene using a "doorway" style of animation. Implement these requirements by adding the following code to the *touchesBegan* method in the *GameScene.swift* file:

```
override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
    if let welcomeNode = childNode(withName: "welcomeNode") {
        let fadeAway = SKAction.fadeOut(withDuration: 1.0)
        welcomeNode.run(fadeAway, completion: {
            let doors = SKTransition.doorway(withDuration: 1.0)
            if let archeryScene = ArcheryScene(fileNamed: "ArcheryScene") {
                self.view?.presentScene(archeryScene, transition: doors)
            }
        })
    }
}
```

Before moving on to the next steps, we will take some time to provide more detail on the above code.

From within the context of the *touchesBegan* method, we have no direct reference to the *welcomeNode* instance. However, we know that when it was added to the scene in the SpriteKit Level Editor, it was assigned the name "welcomeNode". Using the *childNode(withName:)* method of the scene instance, therefore, a reference to the node is being obtained within the *touchesBegan* method as follows:

```
if let welcomeNode = childNode(withName: "welcomeNode") {
```

The code then checks that the node was found before creating a new SKAction instance configured to cause the node to fade from view over a one-second duration:

```
let fadeAway = SKAction.fadeOut(withDuration: 1.0)
```

The action is then executed on the welcomeNode. A completion block is also specified to be executed when the action completes. This block creates an instance of the ArcheryScene class preloaded with the scene contained within the *ArcheryScene.sks* file and an appropriately configured SKTransition object. The transition to the new scene is then initiated:

```
let fadeAway = SKAction.fadeOut(withDuration: 1.0)

welcomeNode.run(fadeAway, completion: {
    let doors = SKTransition.doorway(withDuration: 1.0)
    if let archeryScene = ArcheryScene(fileNamed: "ArcheryScene") {
        self.view?.presentScene(archeryScene, transition: doors)
    }
})
```

Compile and run the app on an iPad device or simulator in landscape orientation. Once running, tap the screen and note that the label node fades away and that after the transition to the ArcheryScene takes effect, we are presented with a gray scene that now needs to be implemented.

93.8 Adding the Texture Atlas

Before textures can be used on a sprite node, the texture images must first be added to the project. Textures take the form of image files and may be added individually to the project's asset catalog. However, for larger numbers of texture files, it is more efficient (both for the developer and the app) to create a texture atlas. In the case of the archer sprite, this will require twelve image files to animate an arrow's loading and subsequent shooting. A texture atlas will be used to store these animation frame images. The images for this project can be found in the sample code download, which can be obtained from the following web page:

https://www.payloadbooks.com/product/ios17xcode/

Within the code sample archive, locate the folder named *sprite_images*. Located within this folder is the *archer.atlas* sub-folder, which contains the animation images for the archer sprite node.

To add the atlas to the project, select the *Assets* catalog file in the Project Navigator to display the image assets panel. Locate the *archer.atlas* folder in a Finder window and drag and drop it onto the asset catalog panel so that it appears beneath the existing AppIcon entry, as shown in the following figure:

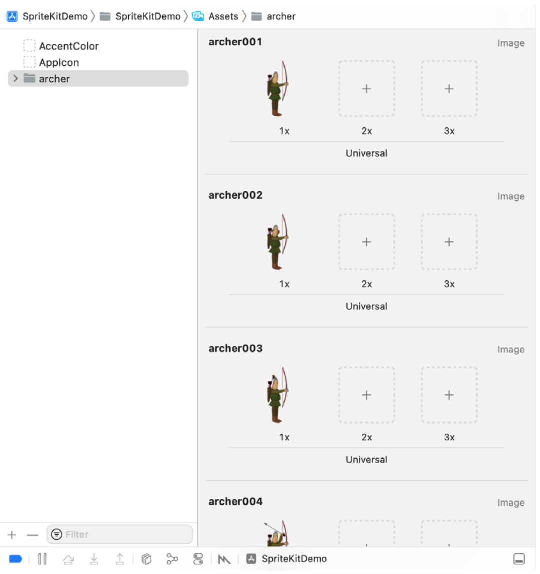


Figure 93-7

93.9 Designing the Archery Scene

The layout for the archery scene is contained within the *ArcheryScene.sks* file. Select this file so that it loads into the Level Editor environment. With the scene selected in the canvas, use the Attributes Inspector panel to change the color property to white and the Size property to landscape *iPad 9.7"*.

From within the SpriteKit Level Editor, the next task is to add the sprite node representing the archer to the scene. Display the Library panel, select the Media Library tab as highlighted in Figure 93-8 below, and locate the *archer001.png* texture image file:

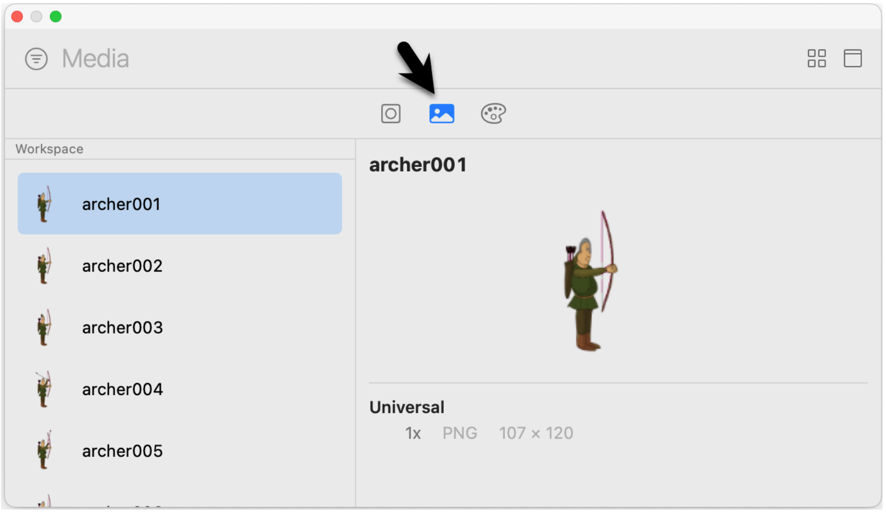


Figure 93-8

Once located, change the Size property in the Attributes Inspector to iPad 9.7", then drag and drop the texture onto the canvas and position it so that it is located in the vertical center of the scene at the left-hand edge, as shown in the following figure:

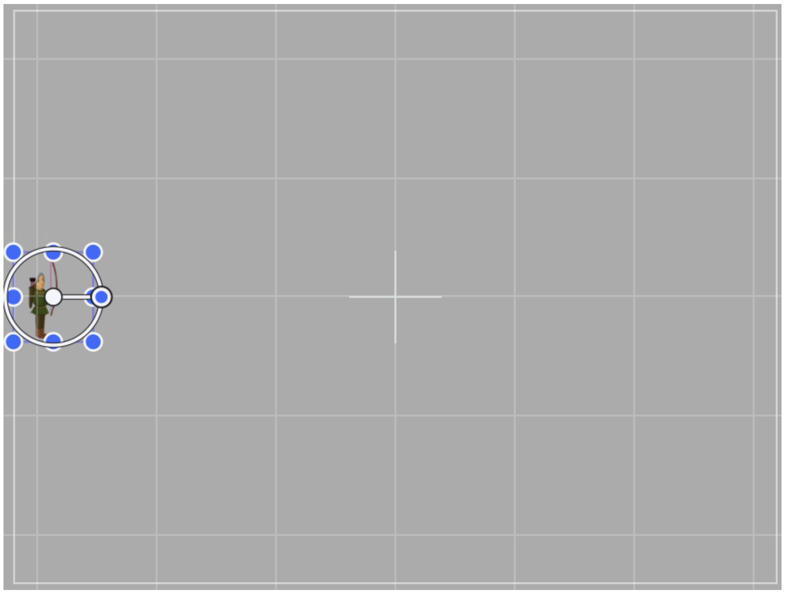
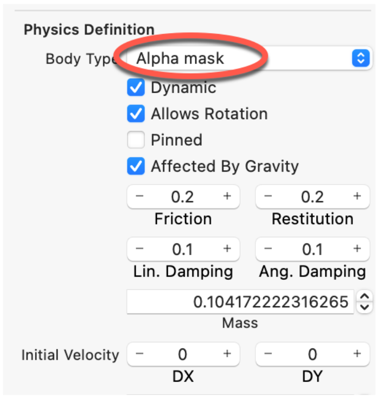


Figure 93-9

With the archer node selected, use the Attributes Inspector panel to assign the name "archerNode" to the sprite. The next task is to define the physical outline of the archer sprite. The SpriteKit system will use this outline when deciding whether the sprite has been involved in a collision with another node within the scene. By default, the physical shape is assumed to be a rectangle surrounding the sprite texture (represented by the blue boundary around the node in the scene editor). Another option is to define a circle around the sprite to represent the physical shape. A much more accurate approach is to have SpriteKit define the physical shape of the node based on the outline of the sprite texture image. With the archer node selected in the scene, scroll down within the Attribute Inspector panel until the *Physics Definition* section appears. Then, using the *Body Type* menu, change the setting to *Alpha mask*:



Physics Definition

Body Type: Alpha mask

Dynamic ✓

Allows Rotation ✓

Pinned

Affected By Gravity ✓

Friction	Restitution
0.2	0.2

Lin. Damping	Ang. Damping
0.1	0.1

Mass: 0.104172222316265

Initial Velocity: DX 0, DY 0

Figure 93-10

Before proceeding with the next phase of the development process, test that the scene behaves as required by clicking on the *Animate* button located along the bottom edge of the editor panel. Note that the archer slides