BEGINNING METAL

HANDS-ON CHALLENGES

Beginning Metal

Caroline Begbie

Copyright ©2016 Razeware LLC.

Notice of Rights

All rights reserved. No part of this book or corresponding materials (such as text, images, or source code) may be reproduced or distributed by any means without prior written permission of the copyright owner.

Notice of Liability

This challenge and all corresponding materials (such as source code) are provided on an "as is" basis, without warranty of any kind, express of implied, including but not limited to the warranties of merchantability, fitness for a particular purpose, and noninfringement. In no event shall the authors or copyright holders be liable for any claim, damages or other liability, whether in action of contract, tort or otherwise, arising from, out of or in connection with the software or the use of other dealing in the software.

Trademarks

All trademarks and registered trademarks appearing in this book are the property of their own respective owners.

14 Challenge #14: Game Over

By Caroline Begbie

Congratulations! You've made it to the final challenge! In this challenge you're going to create a 3D Game Over scene.

You'll transition to this scene when you win or lose, and display the correct text.

Create a new file called **GameOverScene.swift** and replace the existing code with the basic scene code:

```
import MetalKit

class GameOverScene: Scene {

  override init(device: MTLDevice, size: CGSize) {
    super.init(device: device, size: size)
  }

  override func update(deltaTime: Float) {
  }
}
```

In this scene you're going to show a different model depending on whether the player wins or loses.

Create a property for the model:

```
var gameOverModel: Model!
```

When you transition to this scene from the game scene, you'll set a property to tell the scene which model to show.

Create this property:

```
var win: Bool = false {
  didSet {
  }
}
```

Here you'll load the appropriate model. Add this to `didSet`:

```
if win {
  gameOverModel = Model(device: device, modelName: "youwin")
  gameOverModel.materialColor = float4(0, 1, 0, 1)
} else {
  gameOverModel = Model(device: device, modelName: "youlose")
  gameOverModel.materialColor = float4(1, 0, 0, 1)
}
```

If `win` is true, you'll load the model that says **"You Win"**. You color the model green. If win is not true, you color the losing model red.

After that code add the model to the scene.

```
add(childNode: gameOverModel)
```

In this scene, the center of the screen is at (0, 0), so you don't need to reposition the model.

At the end of `init(device:size:)` add the lighting to the scene:

```
light.color = float3(1, 1, 1)
light.ambientIntensity = 0.3
light.diffuseIntensity = 0.8
light.direction = float3(0, -1, -1)
```

Position the camera way back:

```
camera.position.z = -30
```

That's created the Game Over scene. Now you need to transition to it in your game scene.

Scene transitions

You'll do this by means of a delegate. You should be loading scenes from the `ViewController`, not from other scenes. Your game scene will notify the `ViewController` of the scene to transition to, and the `ViewController` will then inform the renderer which scene to render.

In **Scene.swift**, before the class declaration, create a new protocol:

```
protocol SceneDelegate {
}
```

Add the transition method to the protocol:

```
func transition(to scene: Scene)
```

In `Scene`, add a new delegate property:

```
var sceneDelegate: SceneDelegate?
```

Add a new method to `GameScene` for the game over condition:

```
func endGame(win: Bool) {
}
```

Add this to the new method:

```
let gameOverScene = GameOverScene(device: device, size: size)
gameOverScene.win = win
sceneDelegate?.transition(to: gameOverScene)
```

Here you load the game over scene, and set its win status to true or false. You then inform `ViewController` to transition to this scene.

In `update(deltaTime:)`, find these lines:

```
if ball.position.y < 0 {
  ballVelocityY = -ballVelocityY
  bounced = true
}
```

Replace them with:

```
if ball.position.y < 0 {
  endGame(win: false)
}
```

When the ball hits the bottom of the scene, the game is lost.

To check the win state, add this to the end of `update(deltaTime:)`:

```
if bricks.nodes.count == 0 {
  endGame(win: true)
}
```

If all the bricks have been hit, then the player has won the game.

`ViewController` is going to be the `Scene`'s delegate. So in **ViewController.swift**, in `viewDidLoad()`, locate this line:

```
renderer?.scene = GameScene(device: device, size: view.bounds.size)
```

and replace it with:

```
let scene = GameScene(device: device, size: view.bounds.size)
scene.sceneDelegate = self
renderer?.scene = scene
```

The project won't compile now, because `ViewController` doesn't yet conform to the `Scene` protocol.

Add this to the end of **ViewController.swift**:

```
extension ViewController: SceneDelegate {
  func transition(to scene: Scene) {
    scene.size = view.bounds.size
    scene.sceneDelegate = self
    renderer?.scene = scene
  }
}
```

Here you create the scene delegate method to transition to the scene. `ViewController` tells the renderer which scene to render.

Build and run and win and lose. You can reduce the number of rows of bricks to test the win.

Now you need a way to transition back to the game to play again.

Transition back to GameScene

To transition back to `GameScene`, add this to `GameOverScene`:

```
override func touchesBegan(_ view: UIView, touches: Set<UITouch>,
                           with event: UIEvent?) {
 let scene = GameScene(device: device, size: size)
 sceneDelegate?.transition(to: scene)
}
```

Here you transition back to the game when you touch the screen.

Build and run and test out losing. When you return to the game scene, you'll notice that the game may be stretched and the paddle may be in the wrong place. The touch event that you registered before is affecting the position of the paddle.

If you were to move the touch event to `touchesEnded(_:touches:with:)`, the event will happen too quickly, because the player will have their finger already on the screen moving the paddle.

In `GameOverScene`, create a property to hold whether the scene has been touched or not.

```
var registerTouch = false
```

Change the touch methods to:

```
override func touchesBegan(_ view: UIView, touches: Set<UITouch>,
                           with event: UIEvent?) {
  registerTouch = true
}

override func touchesEnded(_ view: UIView, touches: Set<UITouch>,
                           with event: UIEvent?) {
  if registerTouch {
    let scene = GameScene(device: device, size: size)
    sceneDelegate?.transition(to: scene)
  }
}
```

Here you register the touch when the player taps. This tap won't occur until the player has lifted his finger from moving the paddle.

`touchesEnded(_:touches:with:)` then loads the new game.

There's still one small bug. When you transition between the scenes, the scene aspect ration is not being initialized properly, and that affects the projection matrix.

Whenever the scene size changes, `sceneSizeWillChange(to: size)` should be called to reset the camera's aspect ratio.

In `Scene` change:

```
  var size: CGSize
```

to:

```
  var size: CGSize {
    didSet {
      sceneSizeWillChange(to: size)
    }
  }
```

That will fix up the projection problem between scenes.

Message animation

As a finishing touch it would be nice if the message were animated. Using the sine function, you can make the model swing back and forth.

In `GameOverScene`, create a property to keep track of time:

```
var time: Float = 0
```

In `update(deltaTime:)` add `deltaTime` to the total time:

```
time += deltaTime
```

Now add the calculation for the rotation:

```
let amplitude: Float = 0.5
let period: Float = 2
let periodicAmount = sin(Float(time + 0.8) * period)
                        * amplitude * deltaTime
```

Sine is again useful to us - this will provide values that constantly increase then decrease.

And set the rotation and scale of the model:

```
gameOverModel.rotation.x -= π * periodicAmount  // π is Option+P
gameOverModel.scale += float3(periodicAmount/4)
```

Each frame, the game over model will rotate and scale up and down.

Build and run and you have a completed game with your own game engine!

You really are a winner! :] I can't wait to see what other games you create with your mini game engine.