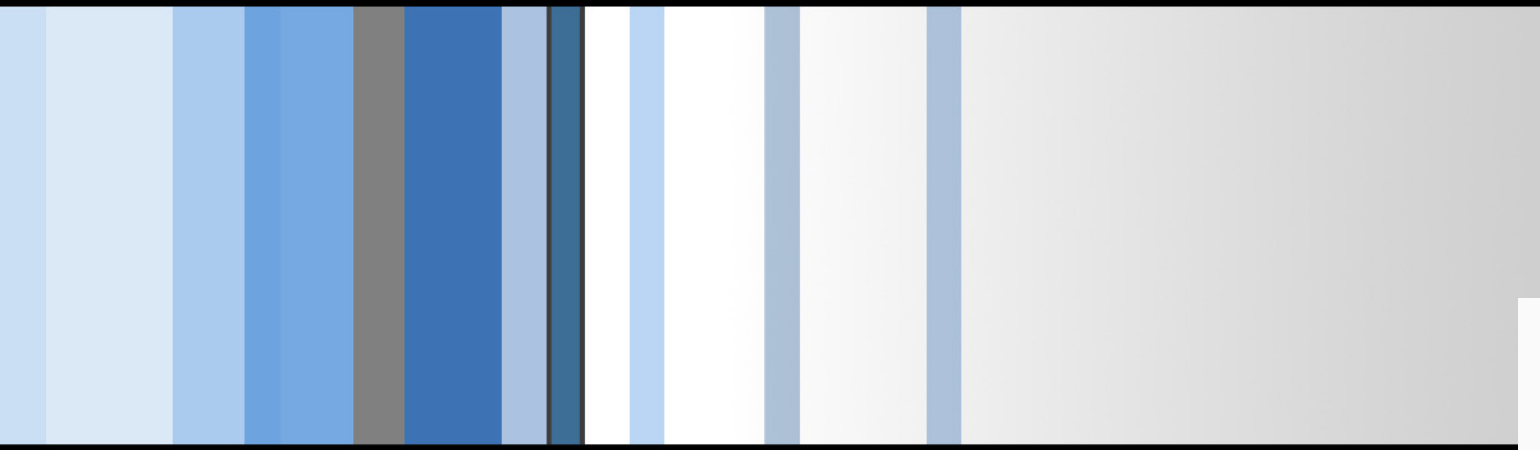
Class 23

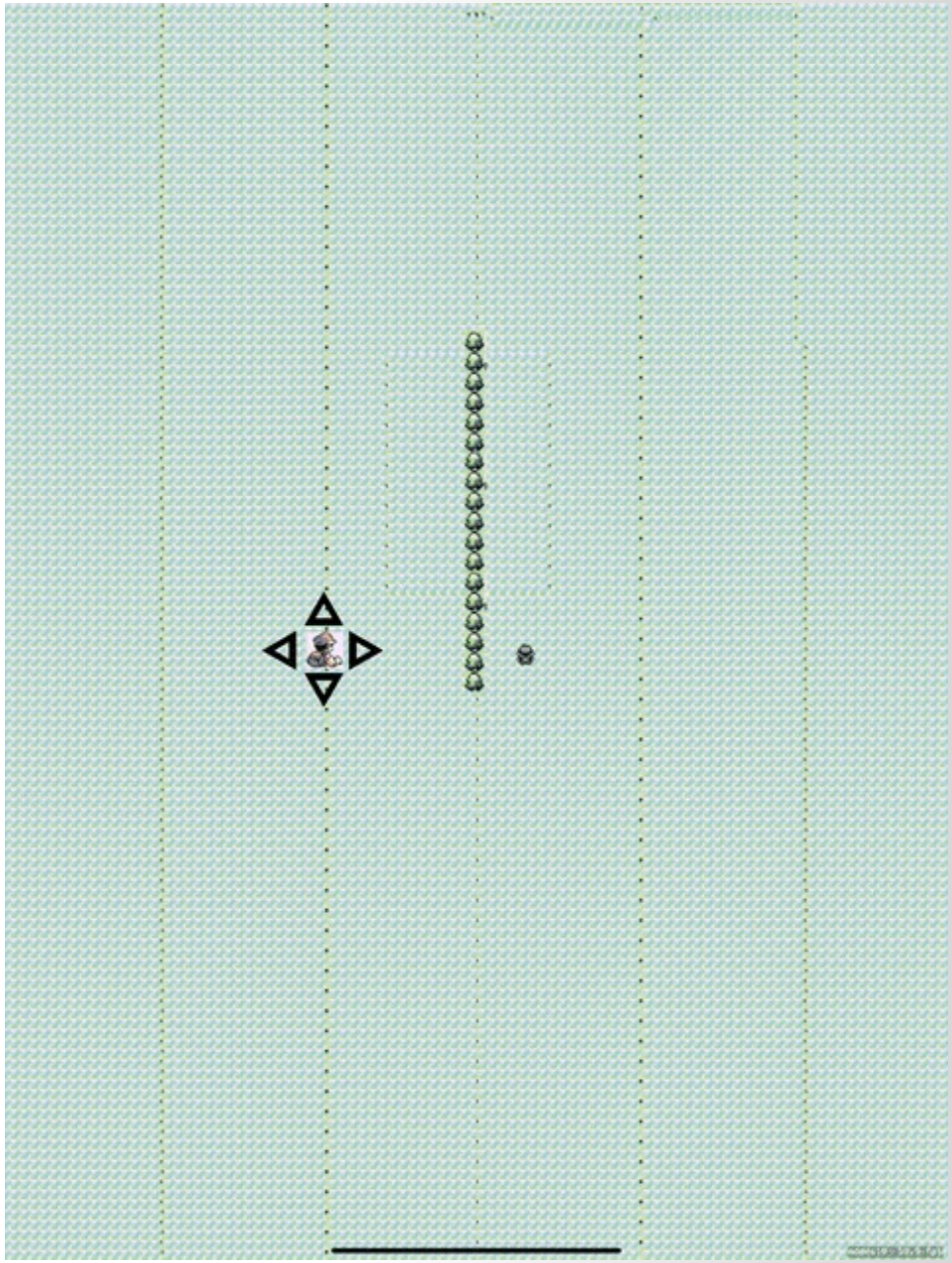
CSC 496: iOS App Development
SpriteKit (5): Handling Collisions and Contact events

Si Chen (schen@wcupa.edu)



SpriteKit

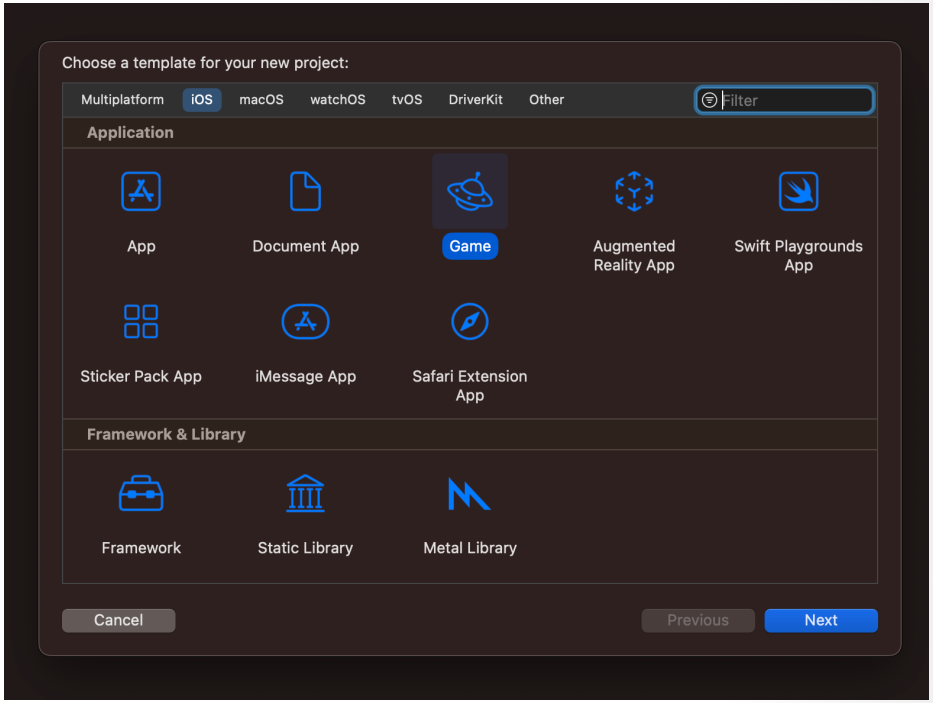
Pokemon2D game

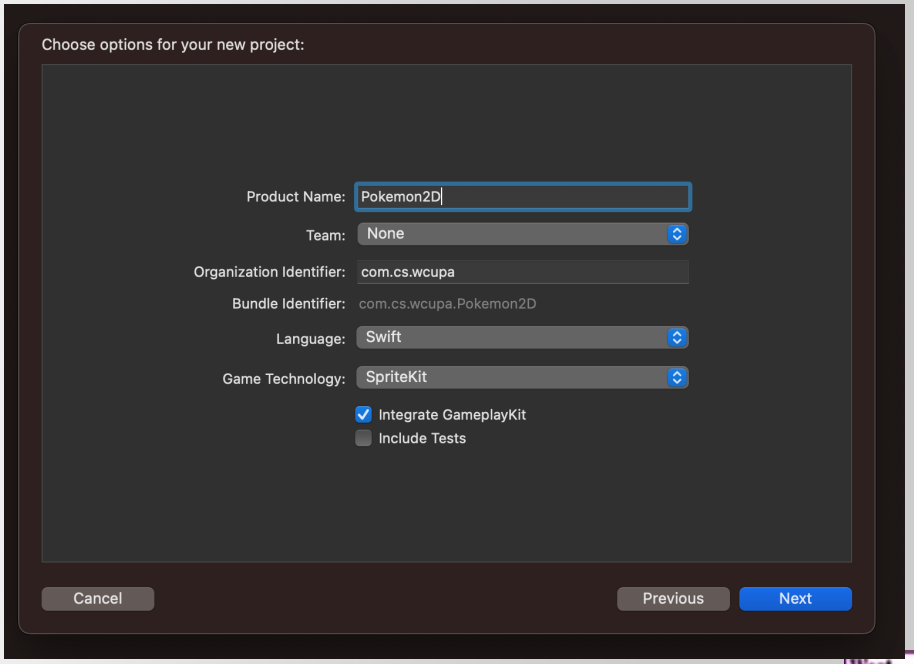


Use the Scene Editor to build 2D games

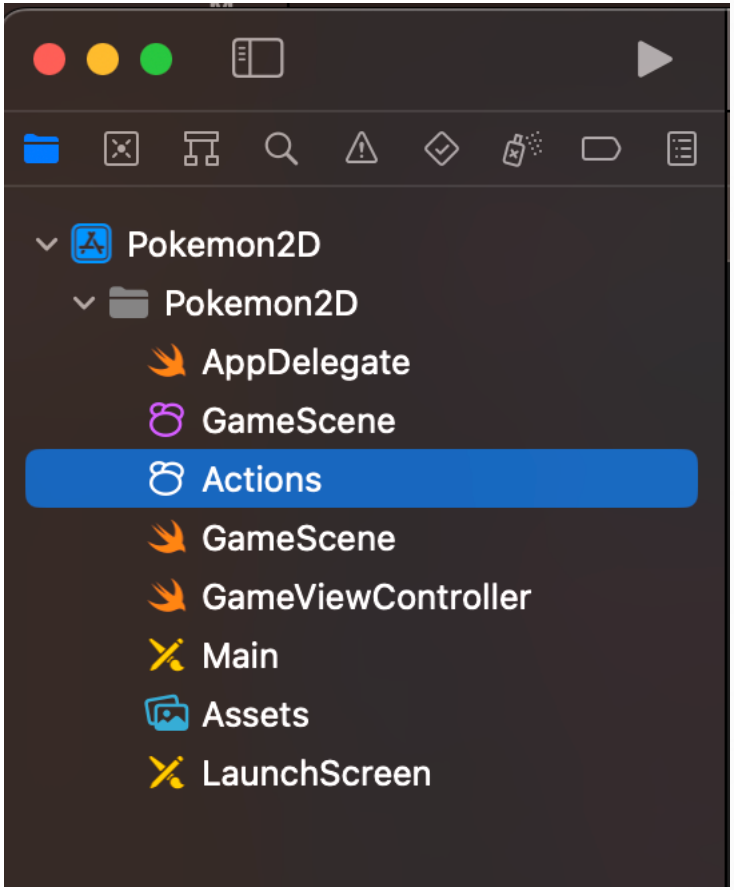
- In this project, we will use the Scene Editor to build a 2D game.
- The Scene Editor is a built-in feature of Xcode that allows us to create SpriteKit nodes and assign values to them.

Create a new Pokemon2D game project





Clean up the default template



Delete Action.sks
Do NOT delete the GameScene.sks

Clean up the default template

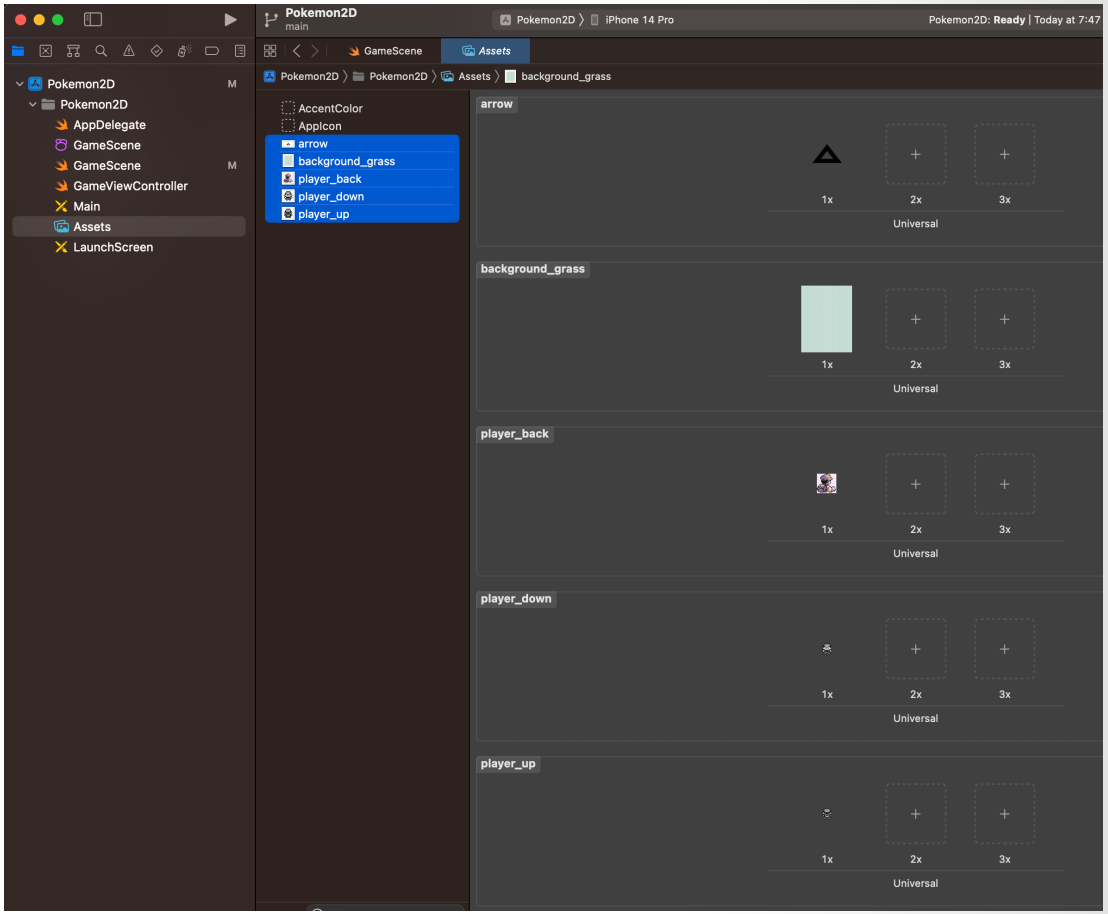
```
import SpriteKit
import GameplayKit

class GameScene: SKScene {

    var entities = [GKEntity]()
    var graphs = [String : GKGraph]()

    private var lastUpdateTime : TimeInterval = 0

    override func sceneDidLoad() {

        self.lastUpdateTime = 0

    }


    func touchDown(atPoint pos : CGPoint) {

    }

    func touchMoved(toPoint pos : CGPoint) {

    }

    func touchUp(atPoint pos : CGPoint) {

    }

    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)) }
    }
```

In GameScene.swift

1. Remove all of the code in the *sceneDidLoad()* method, leaving only the line *self.lastUpdateTime = 0*
2. Remove all of the code in the *touchdown()*, *touchMoved()* and *touchUp()* methods
3. Remove the If statement inside the *touchesBegan()* method.
4. Remove the *label* and *spinnyNode* properties on the top

<-- The final code should look like this

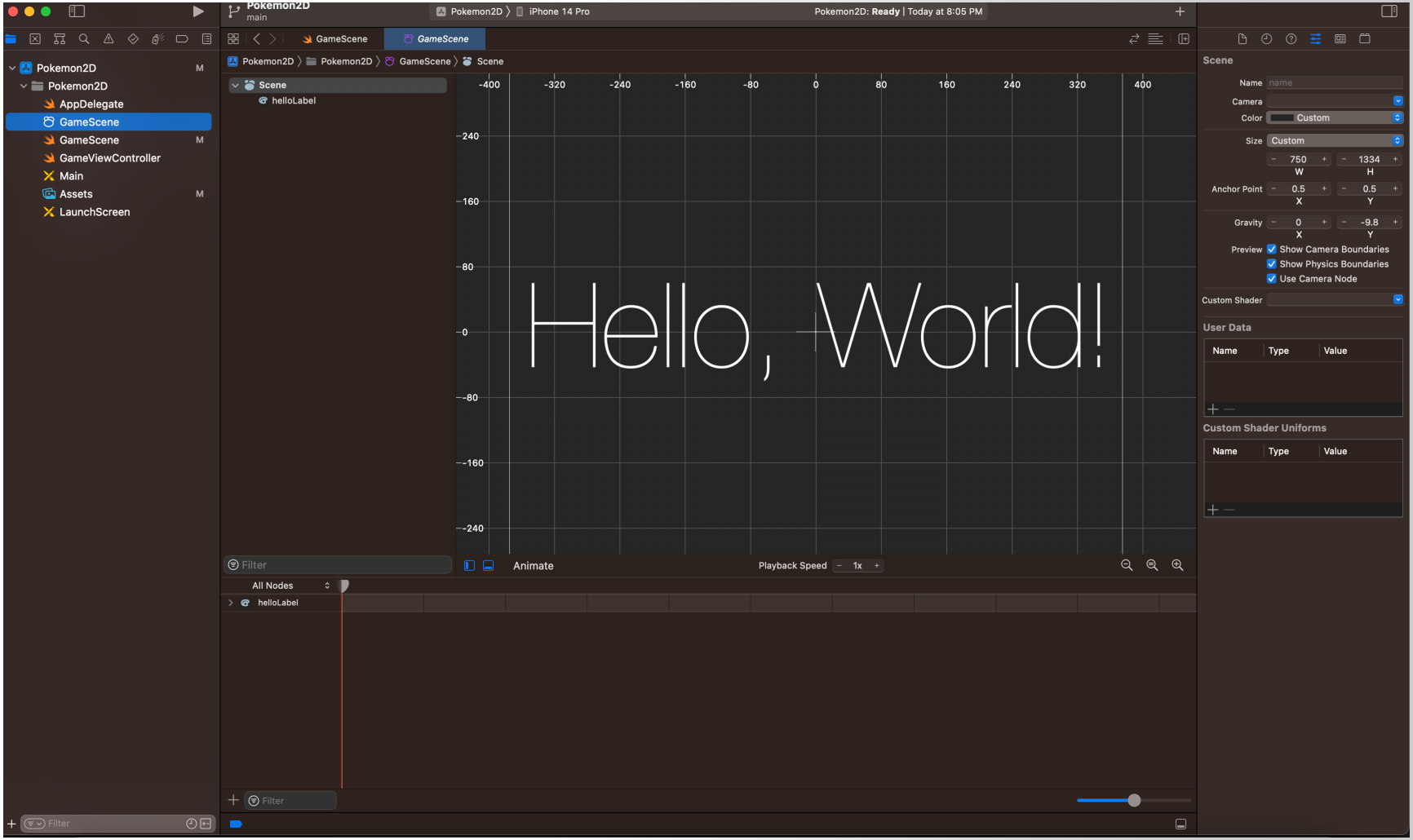
Adding the Assets



Download assets from class website,

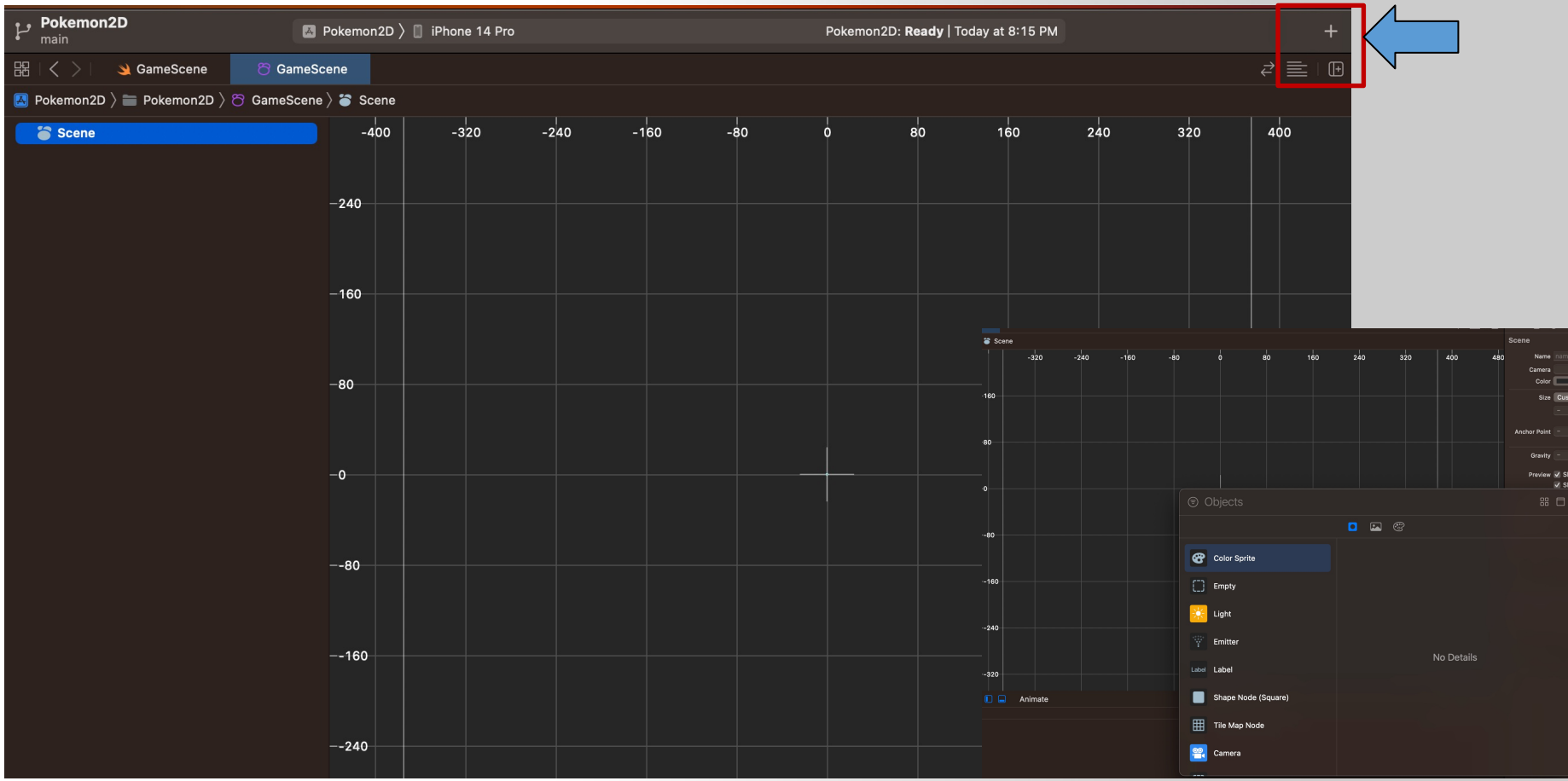
Drag and drop them into the Assets folder

Use Scene Editor to Add nodes



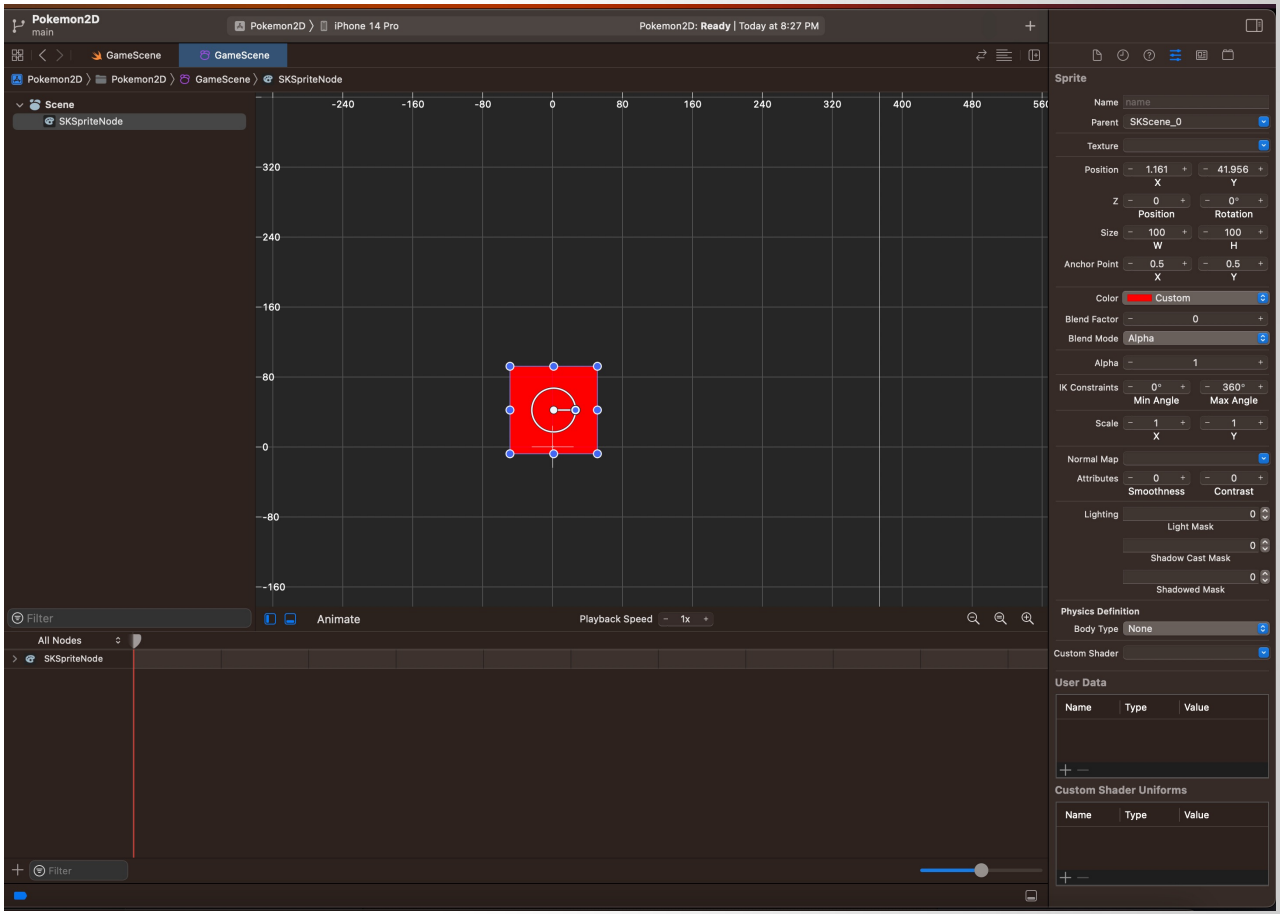
Click GameScene.sks

Use Scene Editor to Add nodes



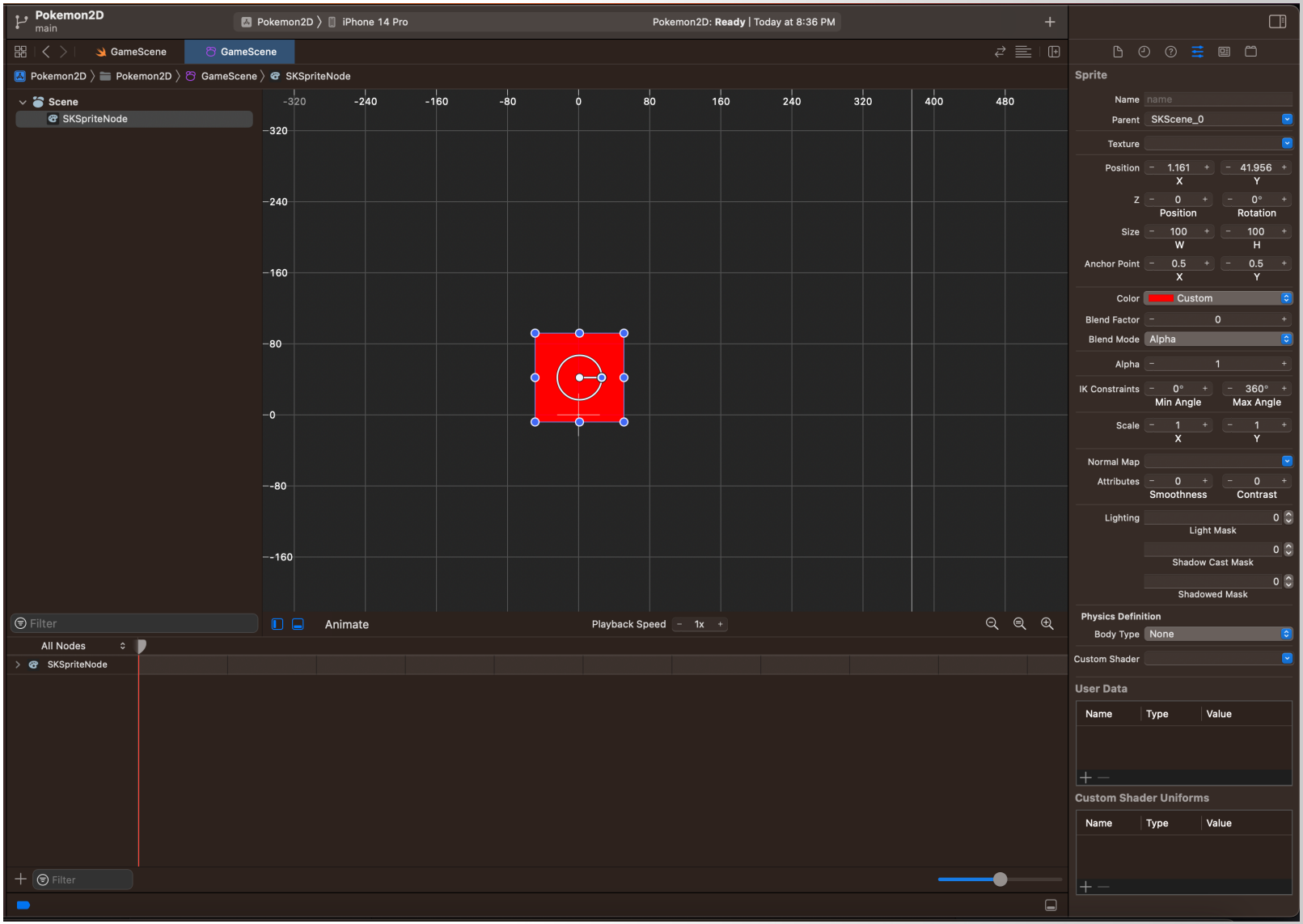
1. Delete the default “Hello world” Node
2. Click the “+” sign
3. Drag and drop the “Color Sprite” to the Scene

Use Scene Editor to Add nodes

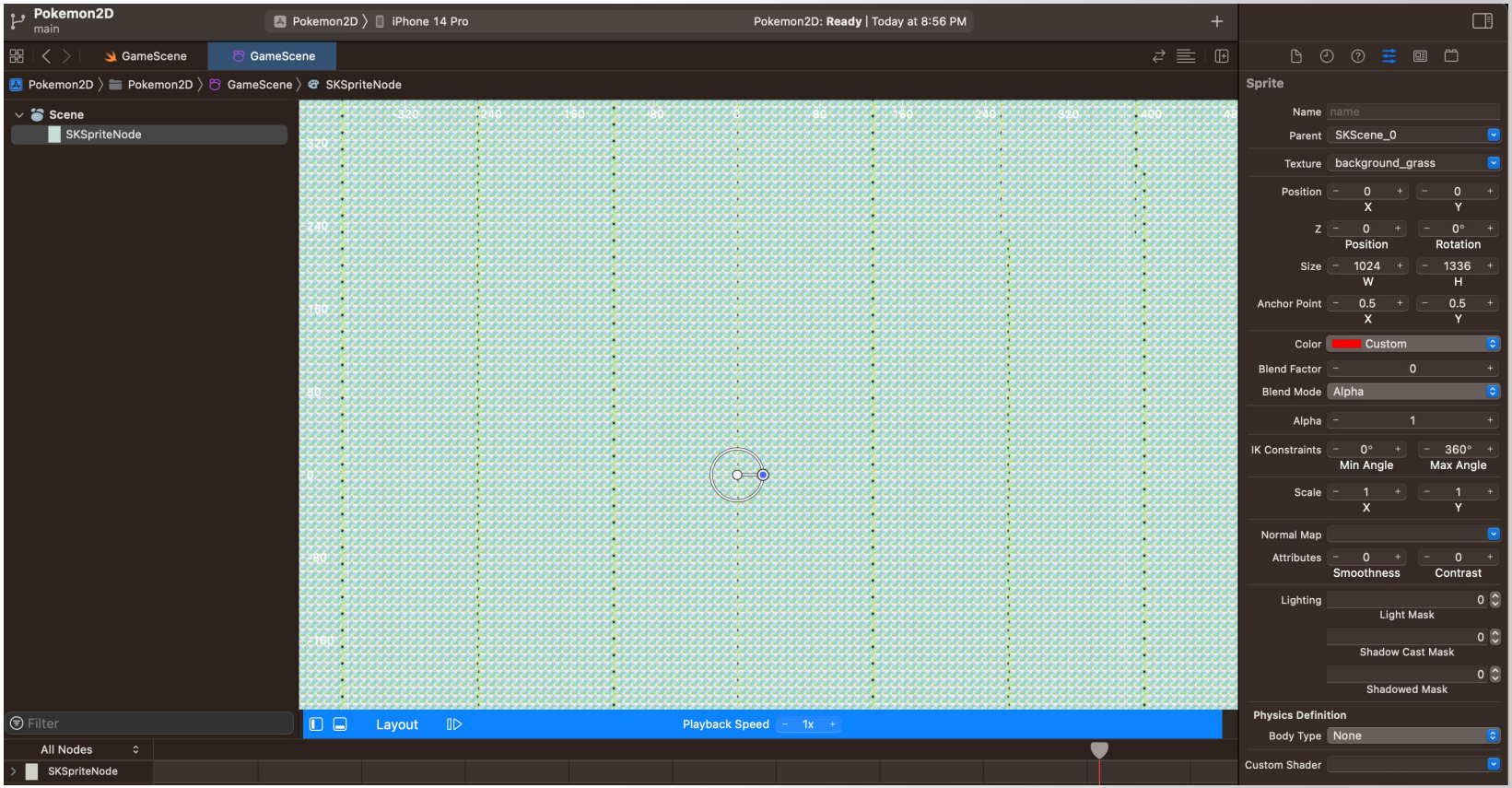


1. Delete the default “Hello world” Node
2. Click the “+” sign
3. Drag and drop the “Color Sprite” to the Scene

Use Scene Editor to Add nodes

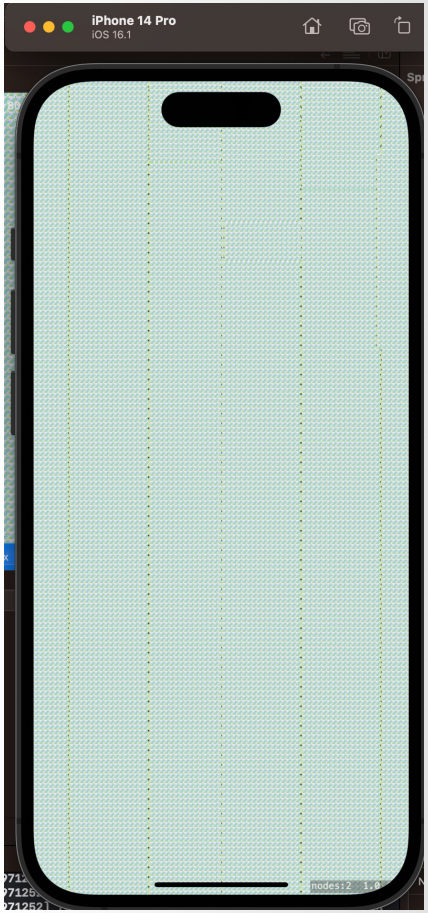


Use Scene Editor to Add nodes

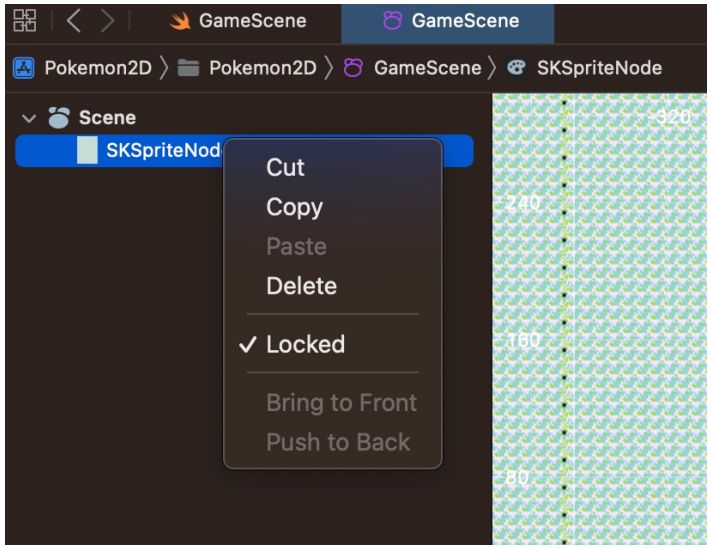


- Change texture to “background_grass”
- Set the position to x: 0 and y: 0

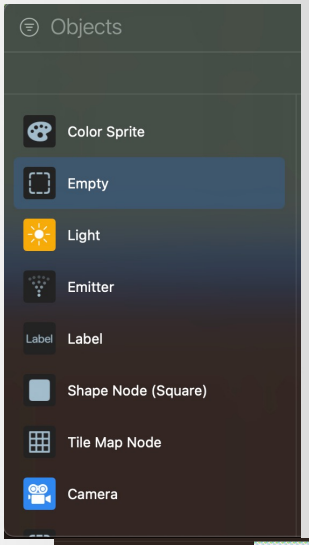
Running it



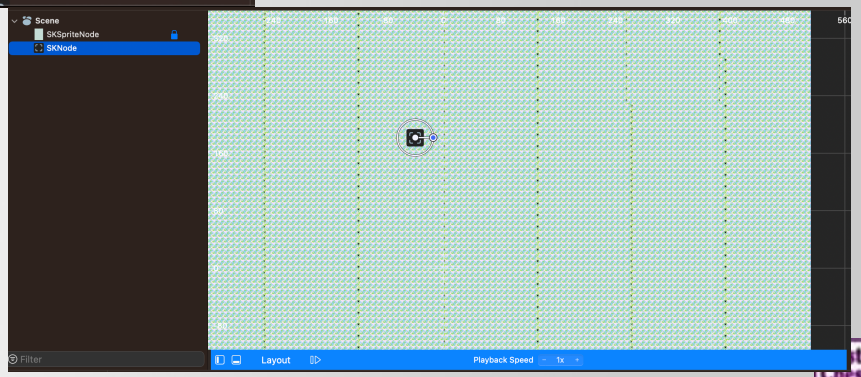
Adding other nodes



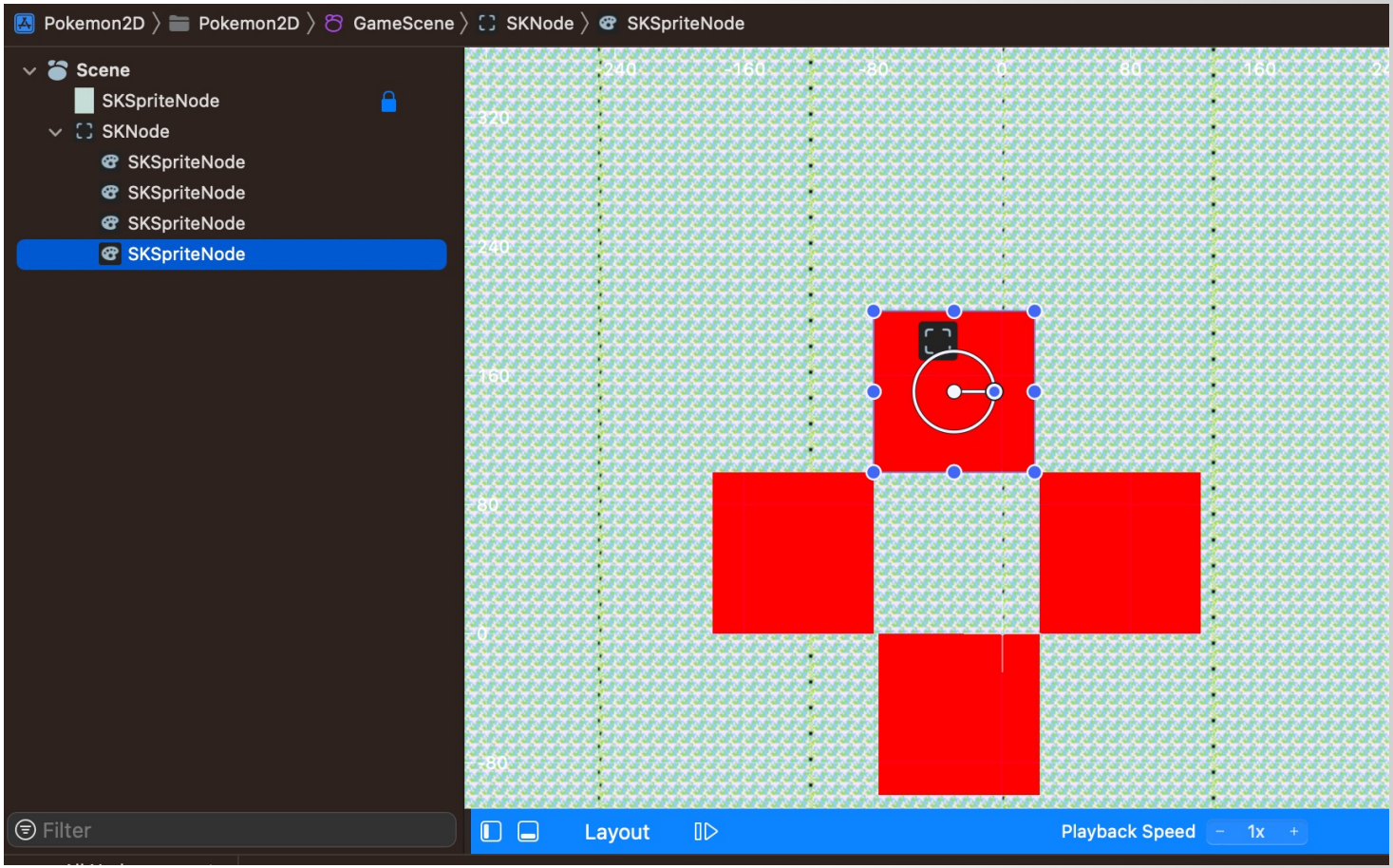
Right click the SKSpriteNode (background) and lock it



Drag “Empty” to the scene

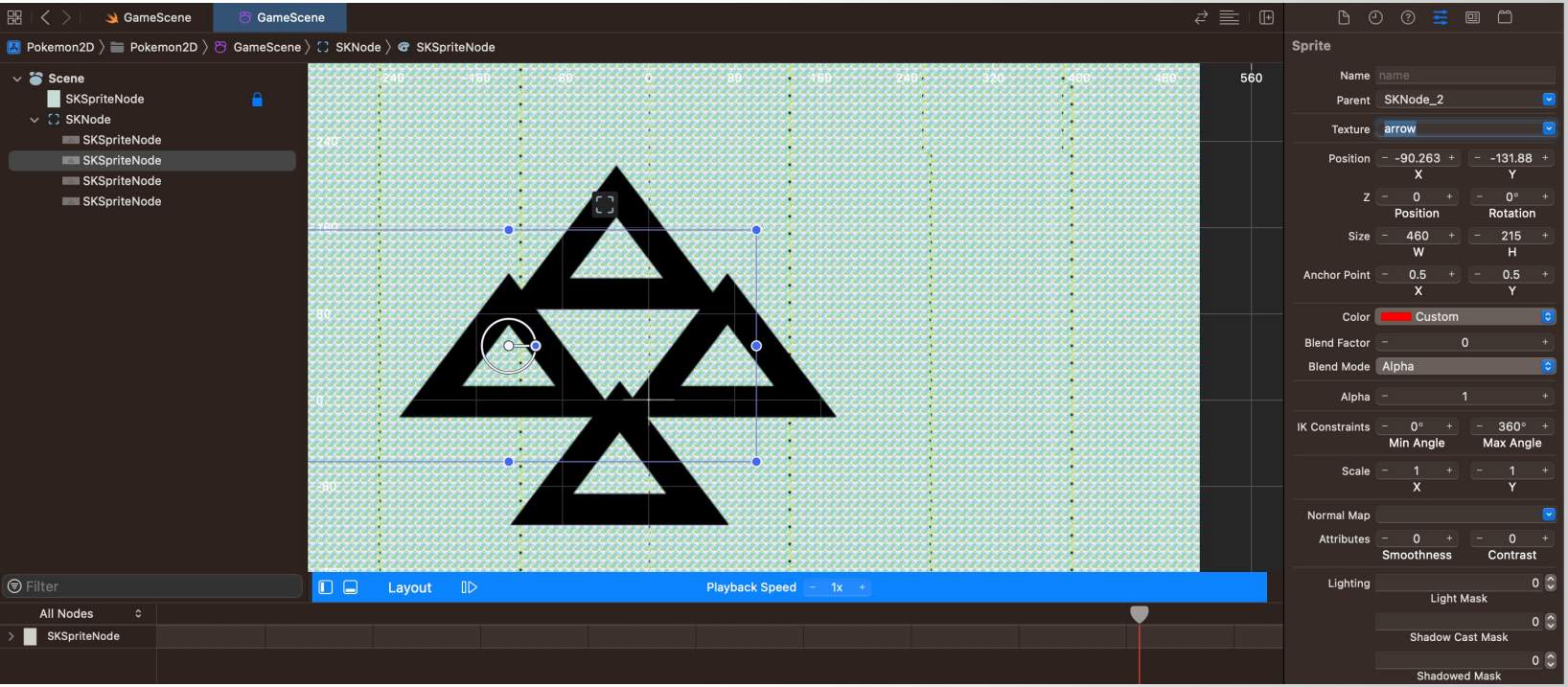


Adding other nodes



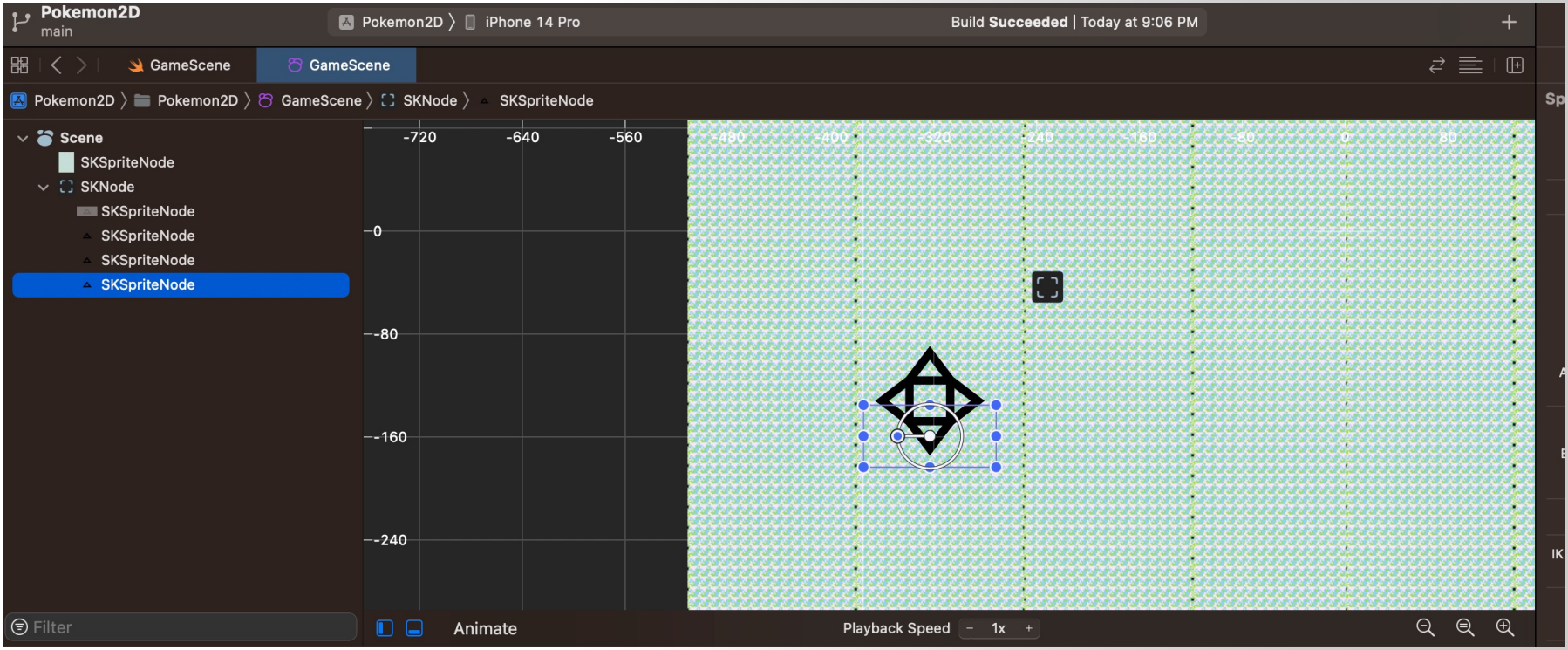
Add four more “Color Sprite” to the Scene and drag them under the SKNode (to group them up)

Adding other nodes



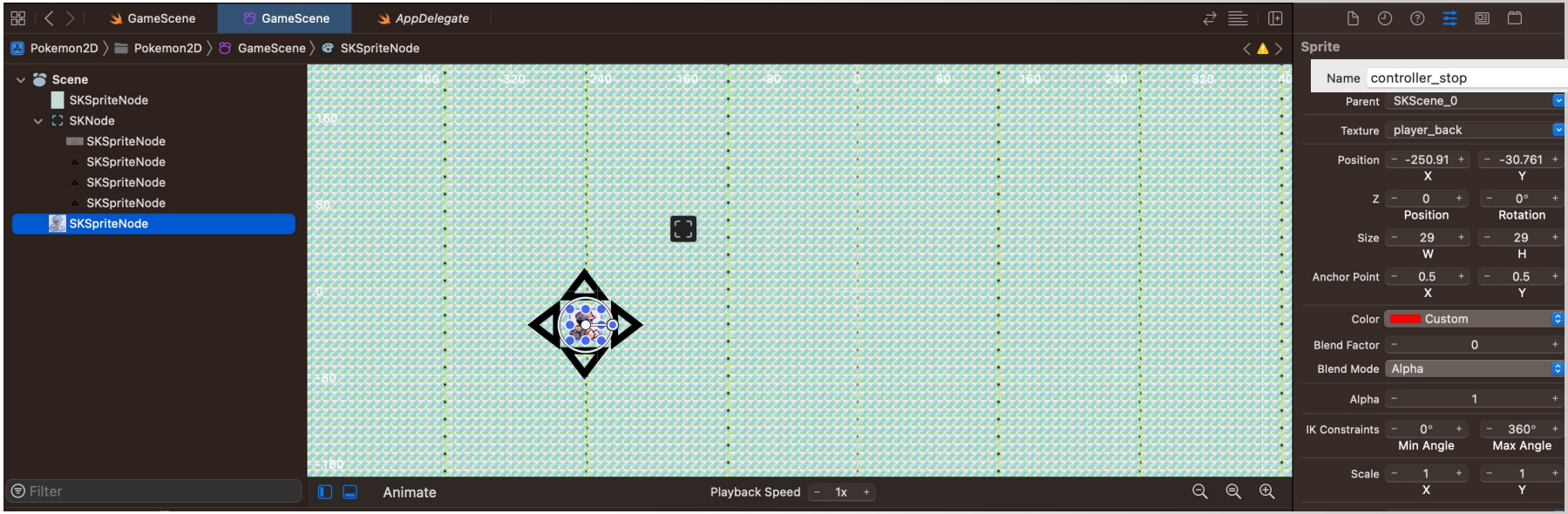
Change the texture of all four Color Sprite Nodes to “arrow”

Adding other nodes



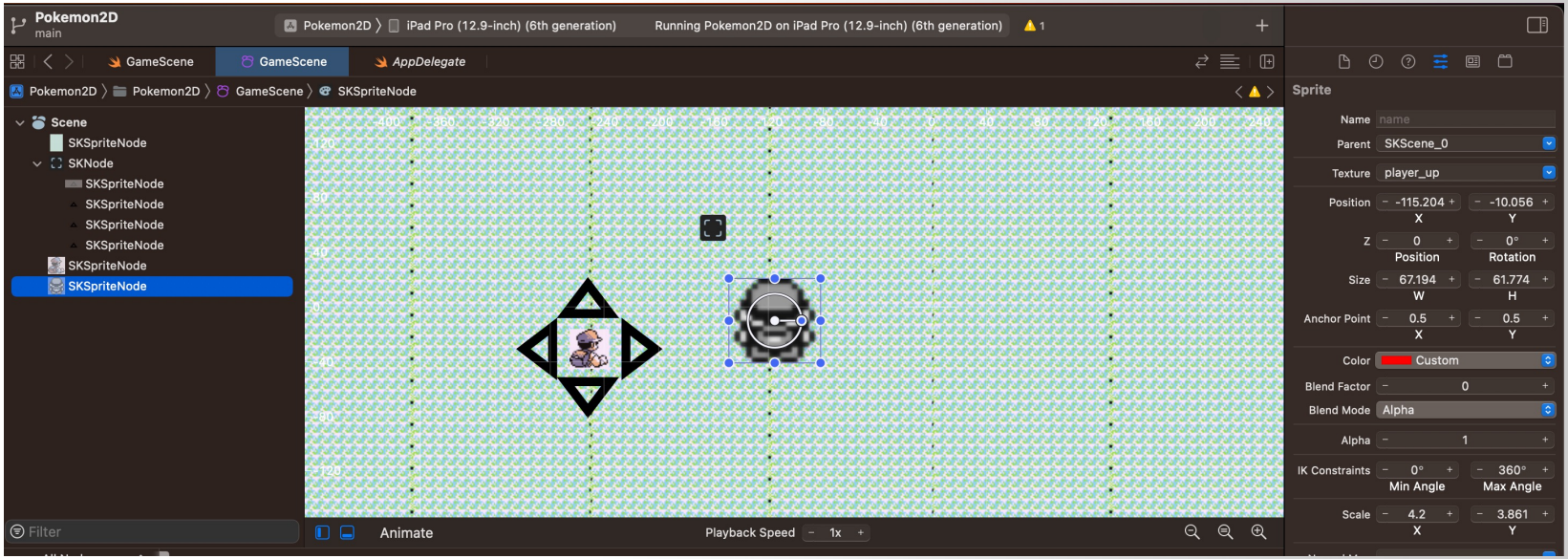
- adjust the size and rotation to make it like an on-screen controller (directional pad, aka D-Pad)

Adding other nodes



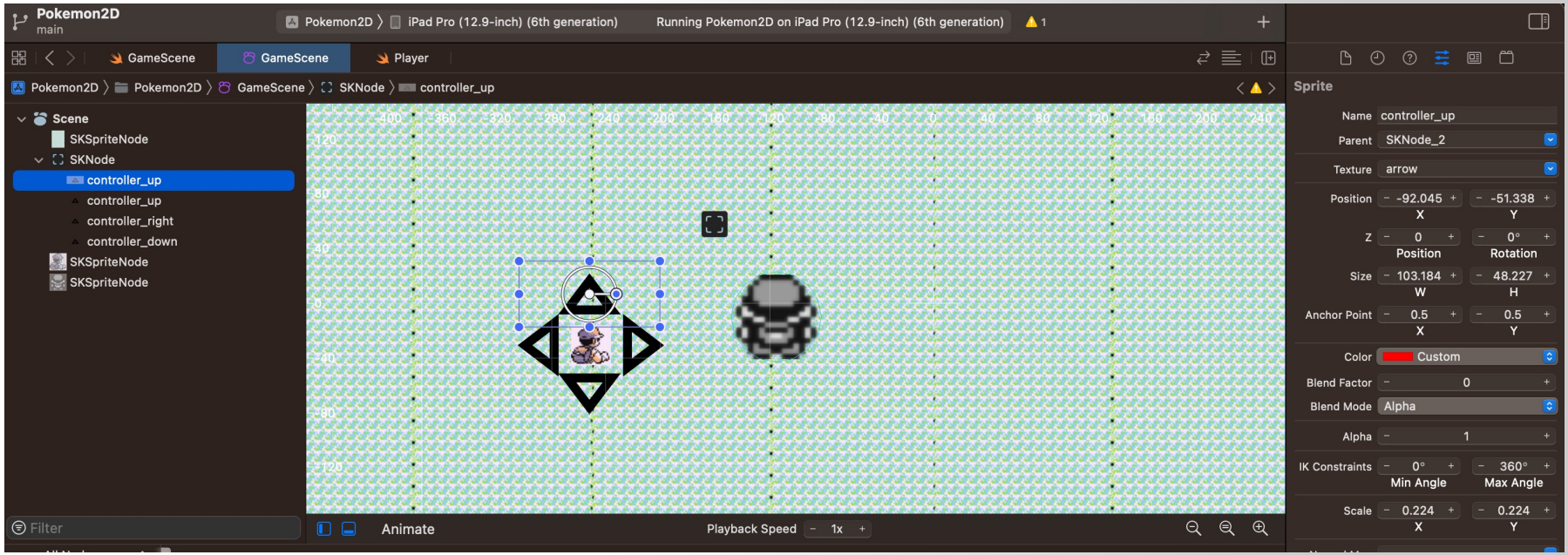
Add another Color Sprite Node and change to texture to “player_back”
And change it’s name to controller_stop

Adding other nodes



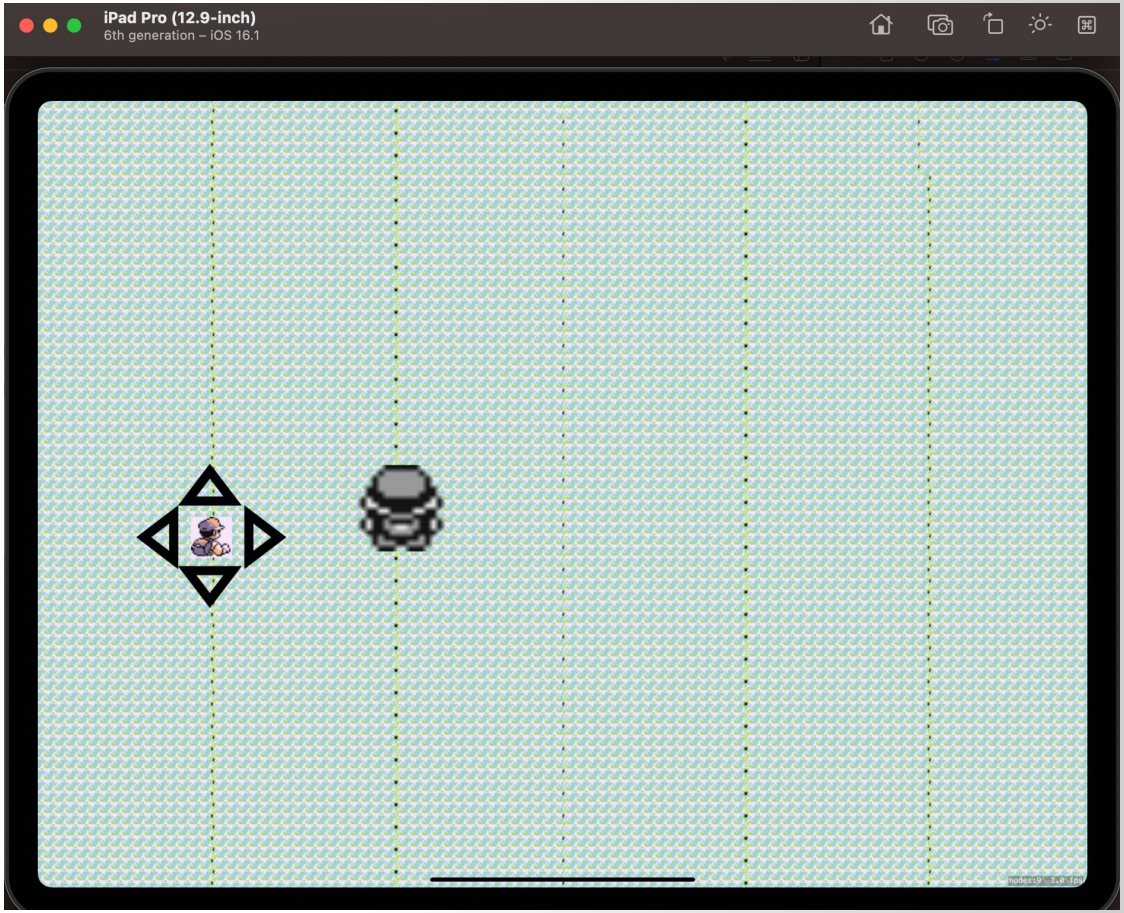
- Finally, add another Color Sprite Node (the player) and change the texture to “player_up”. You can adjust the size if you want.

Change D-pad nodes' name

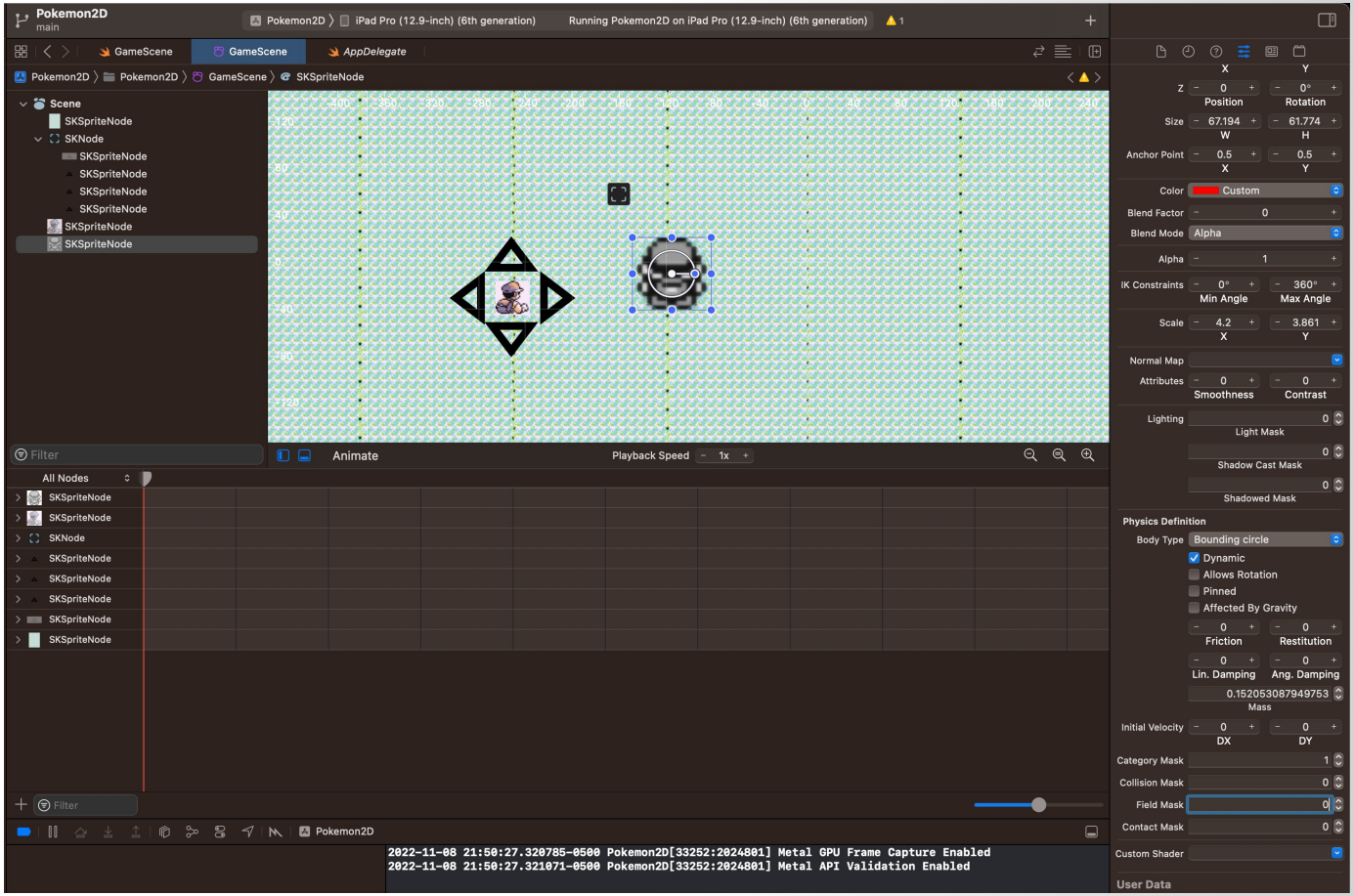


change the "name" of the d-pad button (arrow)'s name to controller_up, controller_down, controller_left, and controller_right accordingly

Run the game

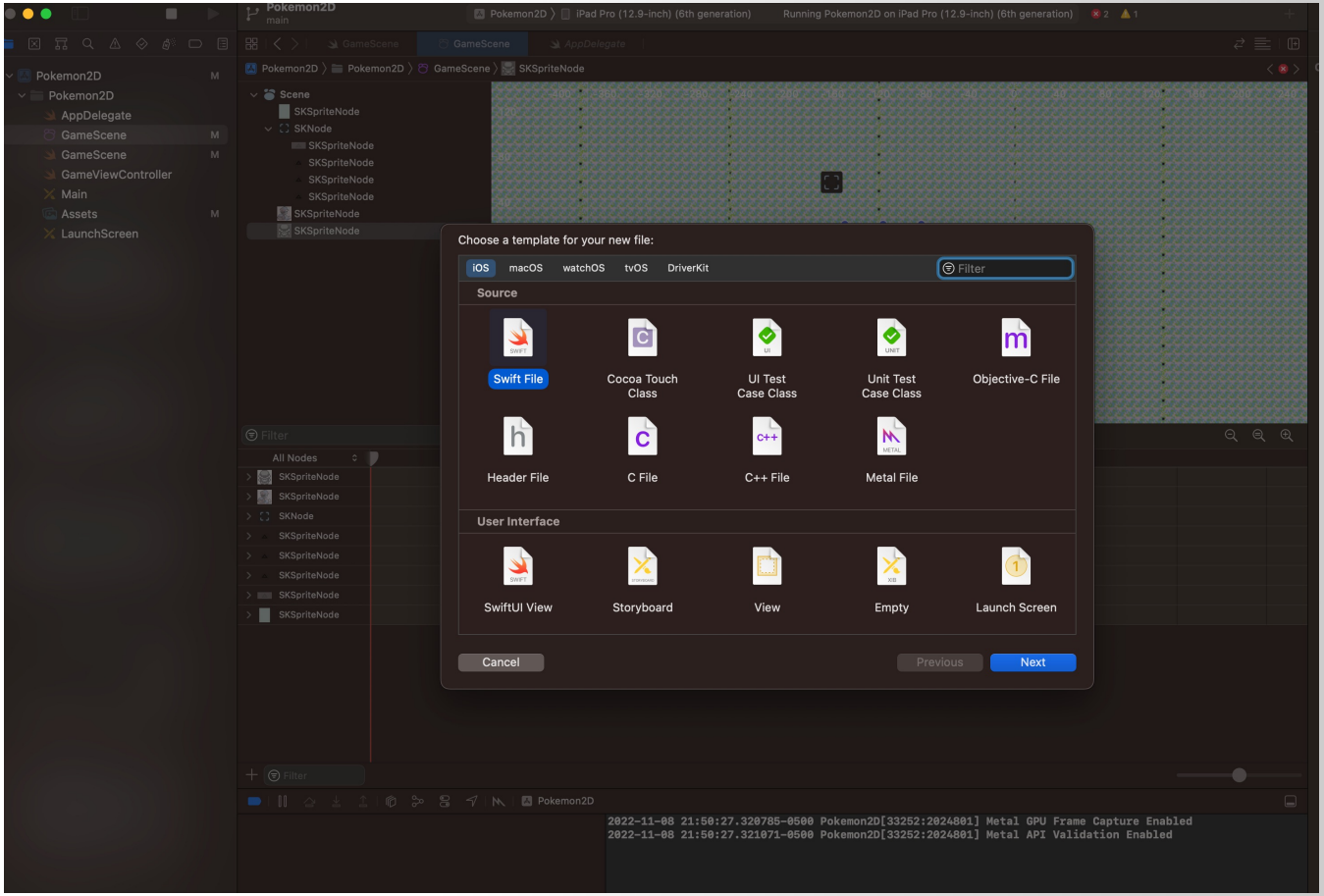


Use the Scene Editor to Add Physics



Click the Player node, change the “Body Type” to Bounding Circle, select “Dynamic” and deselect the allows rotation and affected by gravity options.
Change Fraction, restitution, Lin. Damping and Ang. damping to 0.
Set Category mask to 1, collision mask to 0, the field mask to 0 and the contact mask to 0

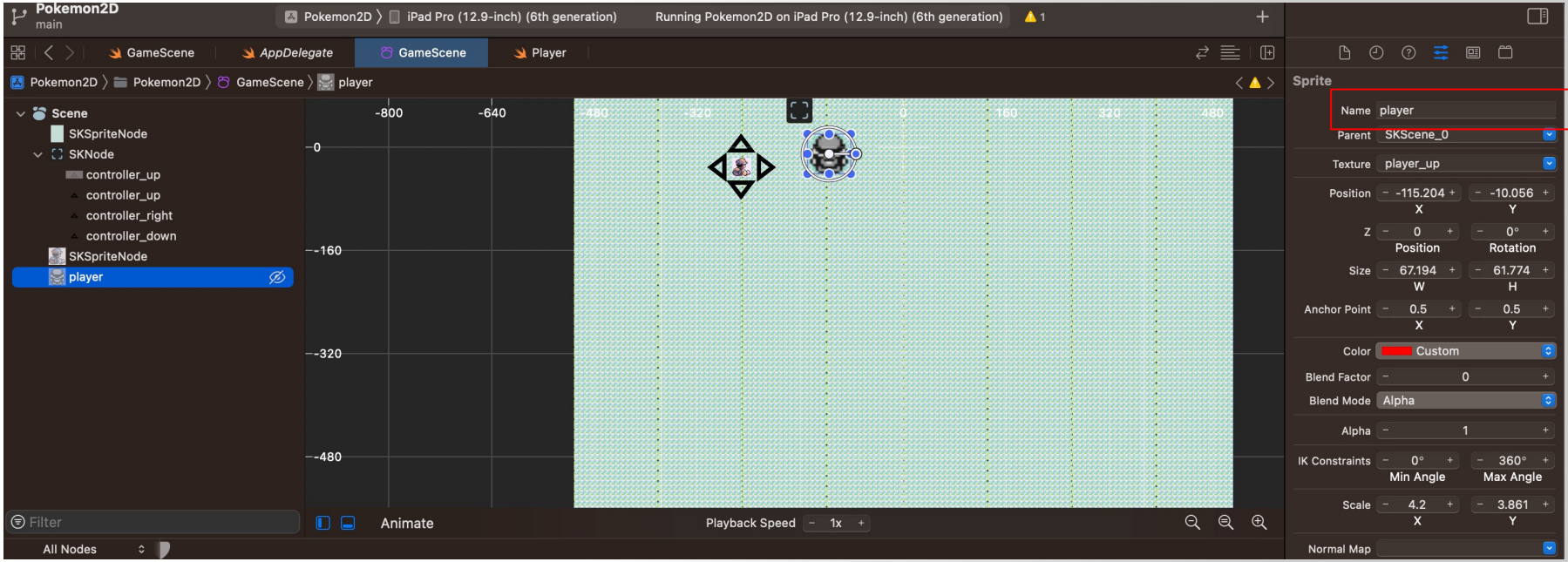
Create a Player class



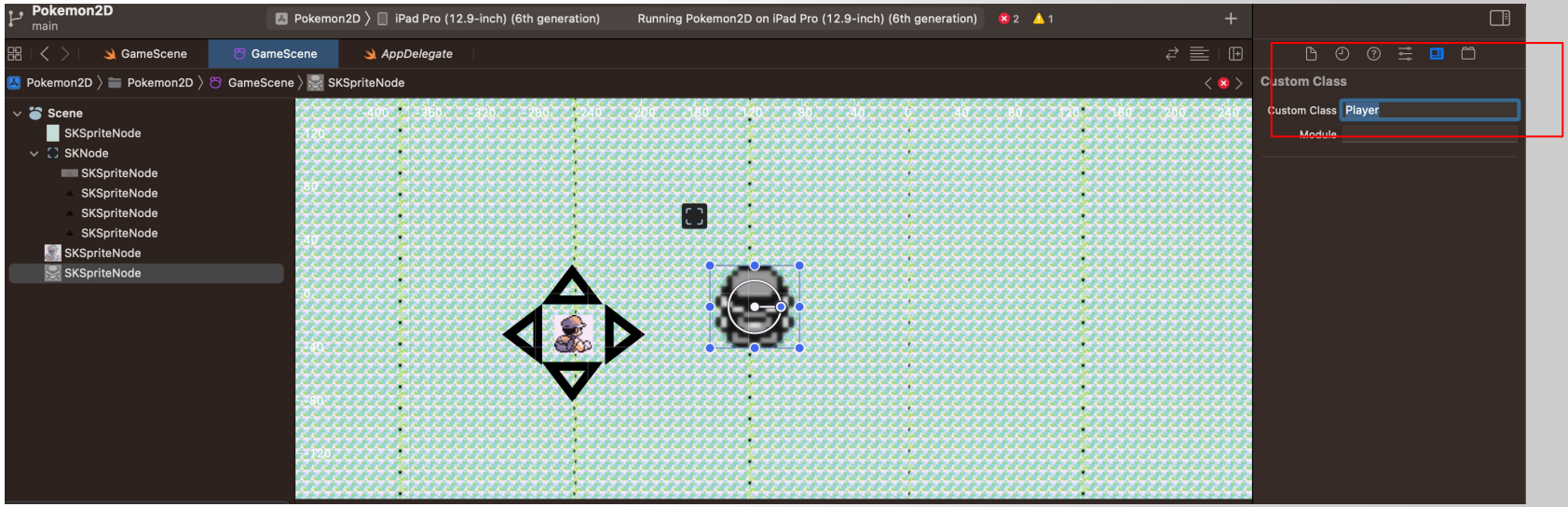
Create a Player class

```
import Foundation
import SpriteKit

enum Direction: String {
    case stop
    case left
    case right
    case up
    case down
}


class Player: SKSpriteNode{
    func move(_ direction: Direction){
        print("player move: \(direction.rawValue)")
    }


    func stop(){
        print("Stop")
    }
}
```

Assign Player node to the Player class



Assign Player node to the Player class

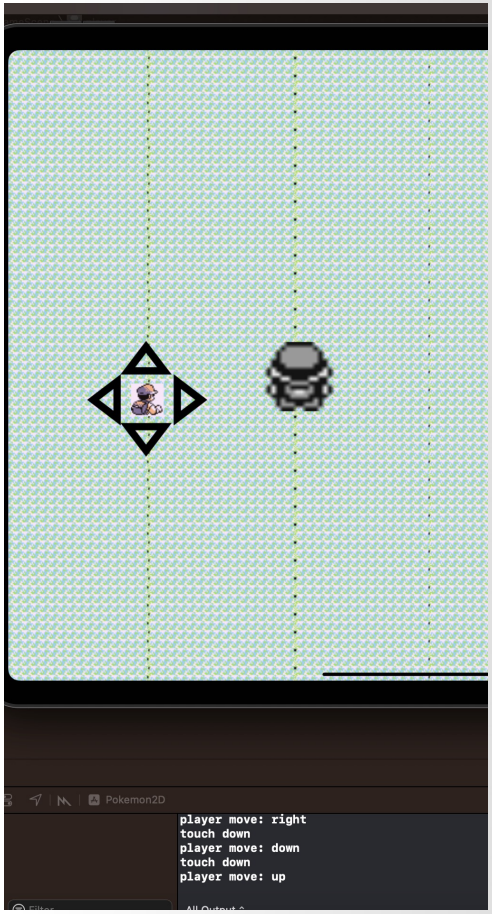


Move the player using Physics

- Open GameScene.swift file, override the didMove() method and update the touchdown() method:

```
import SpriteKit
import GameplayKit

class GameScene: SKScene {

    var entities = [GKEntity]()
    var graphs = [String : GKGraph]()

    private var lastUpdateTime : TimeInterval = 0
    private var player: Player?

    override func sceneDidLoad() {

        self.lastUpdateTime = 0

    }

    override func didMove(to view: SKView) {
        player = childNode(withName: "player") as? Player
        player?.move(.stop)
    }

    func touchDown(atPoint pos : CGPoint) {
        print("touch down")
        let nodeAtPoint = atPoint(pos)
        if let touchedNode = nodeAtPoint as? SKSpriteNode{
            if touchedNode.name?.starts(with: "controller_") == true{
                let direction = touchedNode.name?.replacingOccurrences(of: "controller_", with: "")
                player?.move(Direction(rawValue: direction ?? "stop")!)
            }
        }
    }
```

Run and test the game



```swift
import Foundation
import SpriteKit

enum Direction: String {
    case stop
    case left
    case right
    case up
    case down
}


class Player: SKSpriteNode{
    func move(_ direction: Direction){
        print("player move: \(direction.rawValue)")
        switch direction{
        case .up:
            self.physicsBody?.velocity = CGVector(dx: 0, dy: 100)
        case .down:
            self.physicsBody?.velocity = CGVector(dx:0, dy: -100)
        case .left:
            self.physicsBody?.velocity = CGVector(dx:-100, dy: 0)
        case .right:
            self.physicsBody?.velocity = CGVector(dx:100, dy: 0)
        case .stop:
            stop()
        }
    }


    func stop(){
        print("Stop")
        self.physicsBody?.velocity = CGVector(dx: 0, dy: 0)
    }
}
```

Exercise: 2D RPG Game

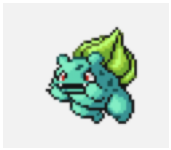
You can either start a new project or download Pokemon2DII from website

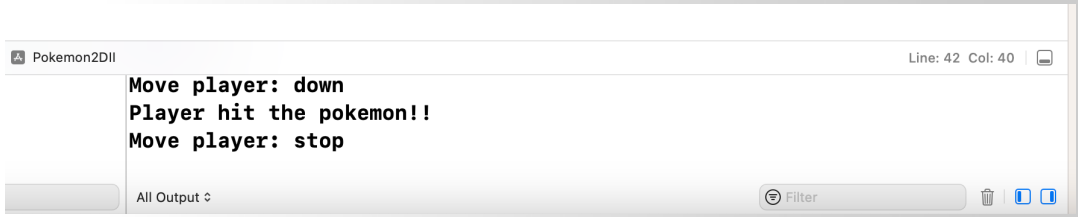
Tasks:

1. Add a new node – “tree” – to the game (the image can be downloaded from class website).

2. Use multiple tree nodes to create a maze that the player cannot pass through. (Hint: Set the Category Mask and Collision Mask for both the player and the tree node.)

3. Add a Pokémon node (node name: pokemon), and when the player hits the Pokémon, it will print "Player hit the Pokémon" in the console.



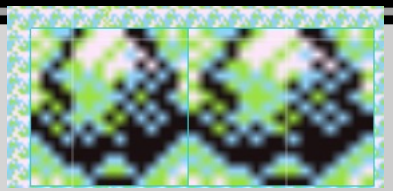


Solution

- In SpriteKit, handling collisions and contact events requires correctly setting the categoryBitMask, collisionBitMask, and contactMask properties.
- These properties need to be configured to ensure the correct response when the player node collides with the tree node.

Player Node:
•**Category Bit Mask**: This is the unique identifier used to identify the node. You have already set this to 1.
•**Collision Bit Mask**: Determines which other nodes it will collide with.
•**Contact Mask**: Determines which nodes to notify upon contact. This is often the same as the collisionBitMask but can be adjusted as needed.

Solution



Tree Node:

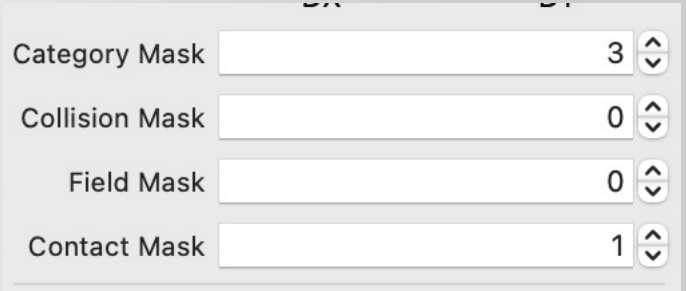
•**Category Bit Mask**: This **needs to match** the **collisionBitMask and/or contactTestBitMask of the Player Node**. So, you can set this to 3.

•**Collision Bit Mask**: This determines with which other nodes the tree node physically collides. To make the player stop upon collision, this should be set to 1 (matching the categoryBitMask of the Player Node).

•**Contact Test Bit Mask**: This determines which nodes to notify when it comes in contact with other nodes. You can set this based on whether you need to receive notifications upon contact. If you need to receive collision notifications, it should be set to 1.

Solution

- To implement a feature where the player node prints "player touched pokemon" in the terminal upon touching a new node named pokemon, follow these steps:
- **Setting Masks**
  - Firstly, you need to set appropriate categoryBitMask, collisionBitMask, and contactTestBitMask for the pokemon node. Assuming the player node's categoryBitMask is still 1, you can set them as follows:
    1. **pokemon's categoryBitMask**: Set this to a unique value, such as 3.
    2. **pokemon's collisionBitMask**: If you don't want the player to physically collide with the pokemon (i.e., pass through instead of stopping), this should be set to 0. Otherwise, set it to 1.
    3. **pokemon's contactTestBitMask**: To detect contact between the player and the pokemon, this mask should be set to 1 (the player's categoryBitMask).



Solution

Writing Collision Detection Code

- You need to implement the SKPhysicsContactDelegate protocol method to detect collisions and set the contactDelegate property of the physics world in your scene (usually a subclass of SKScene).
- **Setting the Delegate**: In the initialization method of your scene (GameScene.swift), set the contactDelegate of the physics world (similar to what we did in Pikachu Project):

```
class GameScene: SKScene {

    var entities = [GKEntity]()
    var graphs = [String : GKGraph]()

    private var lastUpdateTime : TimeInterval = 0
    private var player: Player?

    override func sceneDidLoad() {
        self.lastUpdateTime = 0
        physicsWorld.contactDelegate = self

    }
```

```
class GameScene: SKScene {
    override func update(_ currentTime: TimeInterval) {
        let dt = currentTime - self.lastUpdateTime

        // Update entities
        for entity in self.entities {
            entity.update(deltaTime: dt)
        }

        self.lastUpdateTime = currentTime
    }
}

extension GameScene:SKPhysicsContactDelegate{
    func didBegin(_ contact: SKPhysicsContact) {

    }
}
```

Solution

- **Implementing the didBegin(_:) Method**: Implement the didBegin(_:) method of the SKPhysicsContactDelegate protocol to detect contact events:

```
88  extension GameScene:SKPhysicsContactDelegate{
89      func didBegin(_ contact: SKPhysicsContact) {
90          let firstNode = contact.bodyA.node as? SKSpriteNode
91          let secondNode = contact.bodyB.node as? SKSpriteNode
92
93          if let firstNode = firstNode, let secondNode = secondNode {
94              if (firstNode.name == "player" && secondNode.name == "pokemon") ||
95                 (firstNode.name == "pokemon" && secondNode.name == "player") {
96                  print("player touched pokemon")
97              }
98          }
99      }
100 }
```

This code will be called whenever any two physics bodies begin contact. By checking the names of the nodes associated with these bodies, you can determine if it's the player and pokemon nodes that have made contact.

Make sure your nodes are properly named (e.g., the player node's name property is set to "**player**", the pokemon node's name property is set to "**pokemon**") for the above code to work correctly.

Final Project (40%)

- Create a 2D Role-Playing (RPG) Pokémon game using SpriteKit and demonstrate it on your iPad by recording a video.
- The player can move around in the game world and capture random Pokémon (auto-generated).
- Grading Criteria:
  - Use SpriteKit with Scenes and Nodes to design the game world (15%).
  - Include an on-screen controller to move the player and a camera to track the player (10%).
  - Utilize the Entity-Component Design Pattern to add Collectibles and Physics to your game (10%).
  - Add background music and sound effects (5%).
  - Track and display all Pokémon captured by the user (Bonus 5%).
- You need to submit the following on D2L:
  - Xcode Project (including both project settings file and source code).
  - A video of your game demo.

Q & A