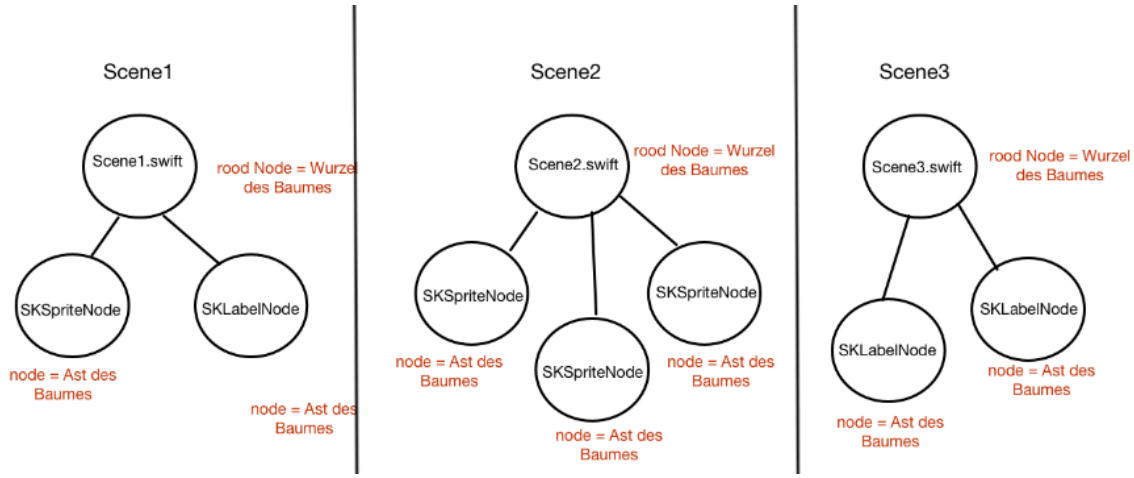
# Einführung in SpriteKit

**Was ist SpriteKit?**
SpriteKit ist ein Framework von Apple, welches uns die Möglichkeit gibt eigene 2D Spiele für das iPhone oder iPad zu entwickeln.
Dieser Artikel soll Dir einen kleinen Überblick in das Framework geben und den Aufbau des Frameworks näherbringen.

**Wie ist der Aufbau des Frameworks?**
Der Aufbau erfolgt mit Szenen, d.h. jede Spielszene erhält eine eigene Szene. Es gibt zum Beispiel das Spielmenü - das wäre dann Szene 1. Dann gibt es das Spiel selbst - das ist die Szene 2 und sollte man das Spiel verlieren gibt es Szene 3, welche die Game Over Szene wäre.


Im Bild 1 siehst Du 3 Szenen und im Bild 2 siehst Du, dass jede dieser Szene eine eigene Datei ist. Diese Datei ist eine Klasse, welche die Klasse SKScene aus dem Framework erbt. Das bedeutet jede dieser Dateien baut eine eigene Szene auf, wie im Bild 1 zu sehen.

**Wie ist der Aufbau einer Szene?**
Szenen machen SpriteKit aus. Doch wie ist der Aufbau einer Szene? Eine Szene kannst Du Dir vorstellen wie einen Baum. Es gibt immer eine Wurzel - das ist eure Swift Datei, welche eine Szene darstellt. Was macht einen Baum aus? Seine Äste - und genau diese Äste werden durch die Klasse SKNode realisiert. Die Klasse SKNode ist dabei die Superklasse und wie in einem echten Baum sieht jeder Ast ein wenig anders aus. Genauso ist das in einer Szene. Jede sichtbare Klasse erbt von der Klasse SKnode. Mit der Klasse SKSpriteNode, zum Beispiel, kannst Du Bilder in unser Spiel einfügen. Die Baumstruktur ist der Grund, warum man immer die Funtkion addchild() aufruft, um eben den Ast bzw. das Kind Element in den Baum bzw. in das Elternelement einzufügen.

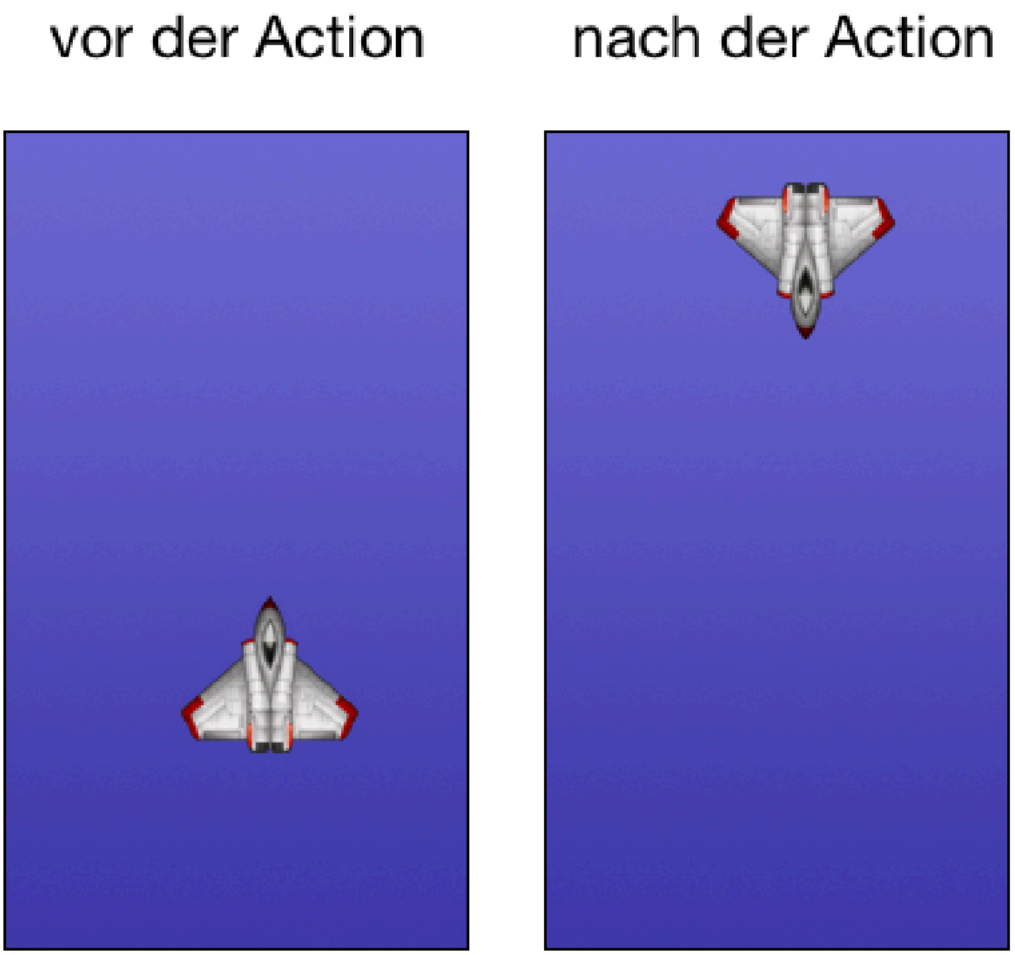



Im Bild 3 siehst Du die Baumstruktur jeder Szene. Dabei stellt jede Datei / jede Klasse die Wurzel des Baumes da. Der Baum wächst mit der hinzugefügten Node Klasse. Die Node Klassen sind zum Beispiel:

- SKSpriteNode -> Fügt Bilder in das Spiel hinzu
- SKShapeNode -> Fügt Formen in das Spiel hinzu
- SKLabelNode -> Fügt Text in das Spiel hinzu
- SKLightNode -> Fügt eine Lichtquelle in Spiel hinzu
- SKEmitterNode -> Fügt Effekte wie Regen oder Feuer in das Spiel hinzu

All diese Klassen haben die SKNode Klasse als Superklasse gemeinsam. Dies erkennst Du am Namen der Klassen. Jede dieser Klassen hat Node im Namen stehen.

**Wie wir das Spiel dynamisch?**
Nodes machen Dein Spiel aus. Denn nur Nodes werden auch im Spiel visualisiert. Diese Nodes kann man dynamisch machen. Das macht man mit Actions. Diese Actions werden mit Hilfe der Klasse SKAction bereitgestellt. Diese Klasse bietet Dir eine Menge Methoden, womit man Nodes dynamisch machen kann. Zum Beispiel kann man Nodes bewegen, rotieren oder springen lassen und vieles mehr. Wichtig ist zu wissen, dass diese Actions nur von Node Klassen ausgeführt werden können.




Hier im Bild siehst Du, dass sich das Raumschiff noch oben bewegt und gedreht hat. Dies kann schafft man mit Hilfe der Klasse SKAction.

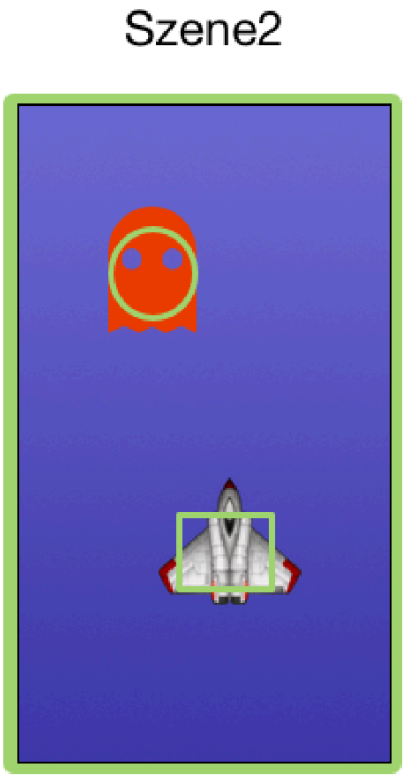
Hier wurden z.B. zwei Methoden der Klasse genutzt:

1. class func moveTo(y: CGFloat, duration sec: TimeInterval) -> SKAction : bewegt ein Node auf eine Position auf der y-Achse
2. class func rotate(toAngle radians: CGFloat, duration sec: TimeInterval) : SKAction -> lässt das Node rotieren

Dabei wurden beide Actions in einer Sequenz ausgeführt. Man kann mehrere Actions auf ein Node ausführen. Die können gleichzeitig oder in einer Sequenz passieren, ebenso kann man Actions auf ein Node sich wiederholen lassen. Dafür gibt es folgende Actions

1. class func group(_ actions: [SKAction]) -> SKAction : führt mehrere Actions gleichzeitig aus
2. class func sequence(_ actions: [SKAction]) -> SKAction : führt Actions nacheinander aus
3. class func repeatForever(_ action: SKAction) -> SKAction : wiederholt Actions für immer

**Wie funktioniert die Physikalische Simulation in SpriteKit?**
Damit es zur Interaktion kommen kann, muss man den Nodes mit physikalischen Bodys ummanteln. Das geht mit der Klasse SKPhysicsBody. Der physikalische Body wird um das Node gelegt und ist unsichtbar. Nicht die Nodes selbst treten in Interaktion, sondern die physikalischen Bodys, die um die Nodes gelegt worden.




In diesem Bild siehst Du die Nodes, welche der Klasse SKSpriteNode entsprechen. Diese Klasse gibt uns die Möglichkeit Objekte zu erstellen, welche Bilder aufnehmen, die wir im Spiel nutzen möchten. Die grünen Rahmen sind dabei die physikalischen Bodys, die mit Hilfe der Klasse SKPhysikBodys erstellt wurden. Es gibt verschiedene Formen, die man erstellen kann, auch die Größe ist variabel. Wichtig ist zu wissen, dass die physikalischen Bodys nur an Nodes angefügt werden können und dass die physikalische Interaktion nur anhand dieser physikalischen Bodys stattfindet, nicht anhand des Bildes. Daher ist die Bildgröße nicht entscheidend für die Interaktion. Diese wird nur anhand der Größe der physikalischen Bodys entschieden.

**Welche Arten von Interaktion gibt es?**
Es gibt zwei Arten von physikalischen Interaktionen. Es gibt die Kollision und den Kontakt. Der Unterschied ist, bei einer Kollision handelt es sich, wie der Name schon sagt, nur um einen Aufprall. Ein Ball prallt, zum Beispiel, von einer Wand ab. Diese Kollision ist dabei nur abhängig von den physikalischen Eigenschaften der physikalischen Bodys. Bei einem Kontakt kann man selbst entscheiden was passiert, wenn die physikalischen Bodys in Kontakt treten. Dieser Kontakt wird durch das Protokoll SKPhysicsContactDelegate erreicht, welches 2 Methoden besitzt die

1. func didBegin(_ contact: SKPhysicsContact) : wird aufgerufen wenn der Kontakt beginnt
2. func didEnd(_ contact: SKPhysicsContact) : wird aufgerufen wenn der Kontakt beendet ist

Diese Funktionen werden in der jeweiligen Szene implementiert und in dieser Szene kann man selbst entschieden, was passieren wird, sobald zwei physikalische Bodys in Kontakt treten.

Doch wie entscheidet man was stattfinden soll? Man kann den physikalischen Bodys Nummern zuweisen und dann entscheidet man, welche Nummern in Kollision oder in Kontakt treten können.
Dazu bietet die Klasse SKPhysiksbody mehrere Variablen an:

1. **var categoryBitMask: UInt32** -> damit weisen wir dem Physikalischem Body eine Nummer zu
2. **var collisionBitMask: UInt32** -> hier weisen wir die Nummern der anderen Physikalischen Bodys zu mit wem eine Kollision stattfinden soll
3. **var contactTestBitMask: UInt32** -> hier weisen wir die Nummern der anderen Physikalischen Bodys zu mit wem ein Kontakt stattfinden soll