Praktikum – iOS-Entwicklung

Sommersemester 2019

Prof. Dr. Linnhoff-Popien

Markus Friedrich, Christoph Roch

Games

SpriteKit, GameplayKit

SpriteKit

SpriteKit ist ein Framework zur Erstellung von 2D Spielen.

Es stellt Funktionalität für die folgenden Bereiche zur Verfügung:

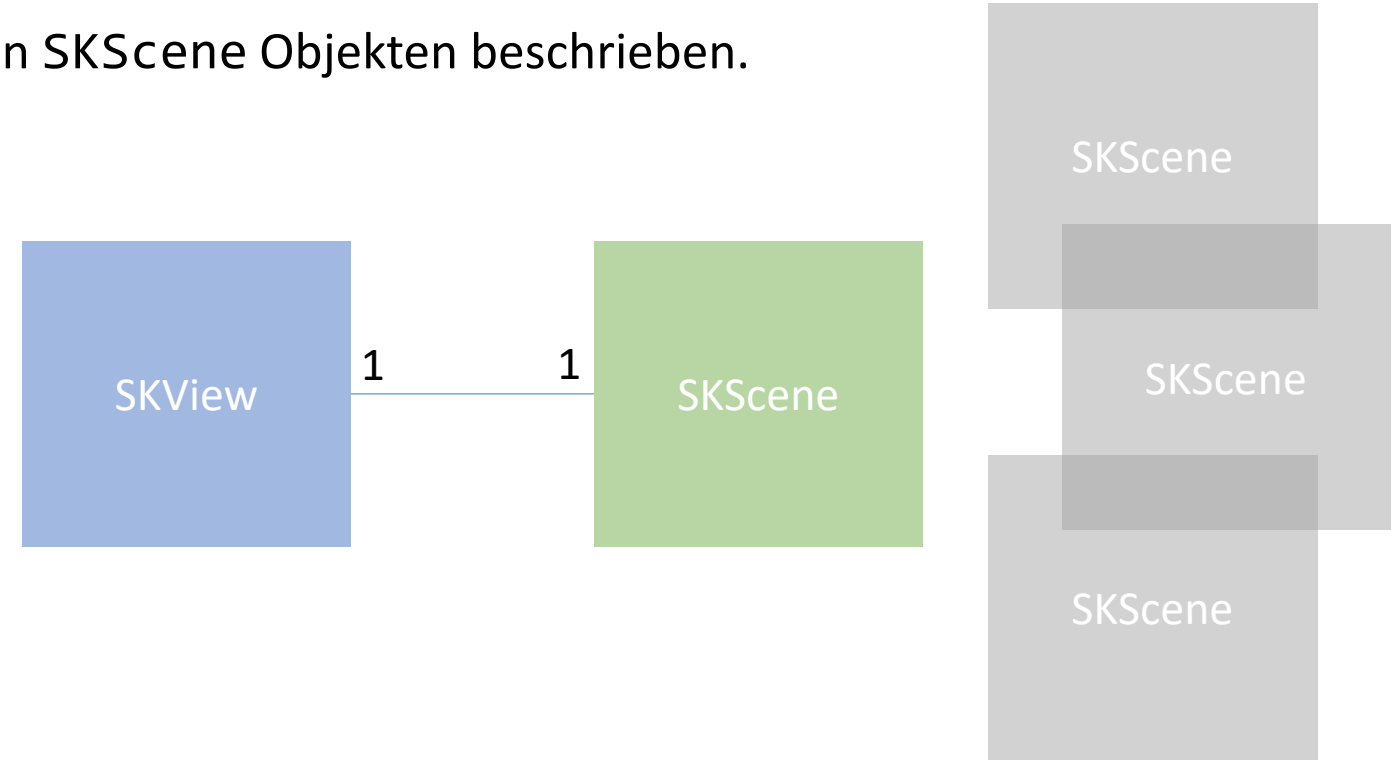
- Darstellen von Sprites und Spriteanimationen
- Partikeleffekte
- Font Rendering
- 2D Physikengine
- Audioeffekte

- Unterstützte Plattform: iOS, macOS, tvOS

SpriteKit – SKView und SKScene

- Ein `SKView` Objekt bietet den Rahmen, in dem der Spielinhalt dargestellt wird.

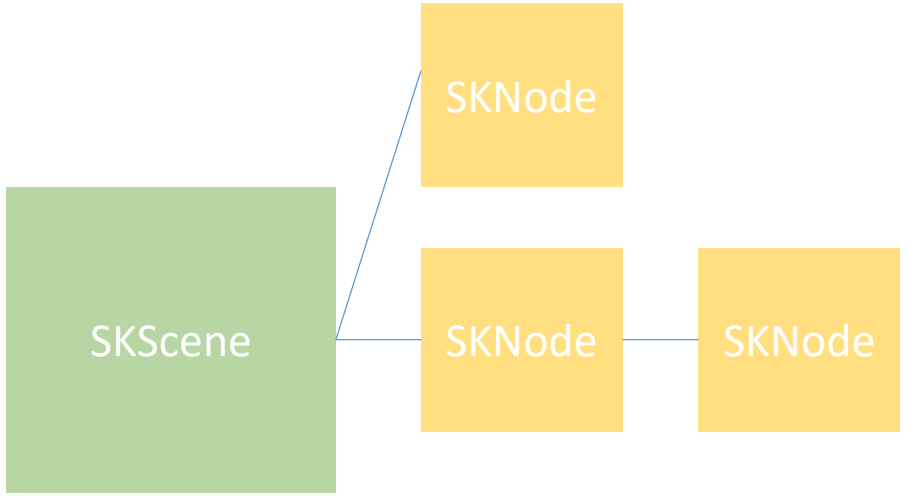
- Die Spielinhalte werden von `SKScene` Objekten beschrieben.



SpriteKit – SKView

- Ein `SKView` Objekt kann für folgende Dinge verwendet werden:
    - Konfiguration: Bildwiederholrate, Automatisches Entfernen nicht-sichtbarer Bildelemente,…
    - Debuginformationen: Bildwiederholrate , Physikinfos, Anzahl Objekte,…
    - Simulation pausieren
    - Rendern von Szeneninhalten in eine Textur
    - …
- Die Szene, die dargestellt werden soll, kann mit `presentScene(SKScene?)` geändert werden.

SpriteKit – SKScene

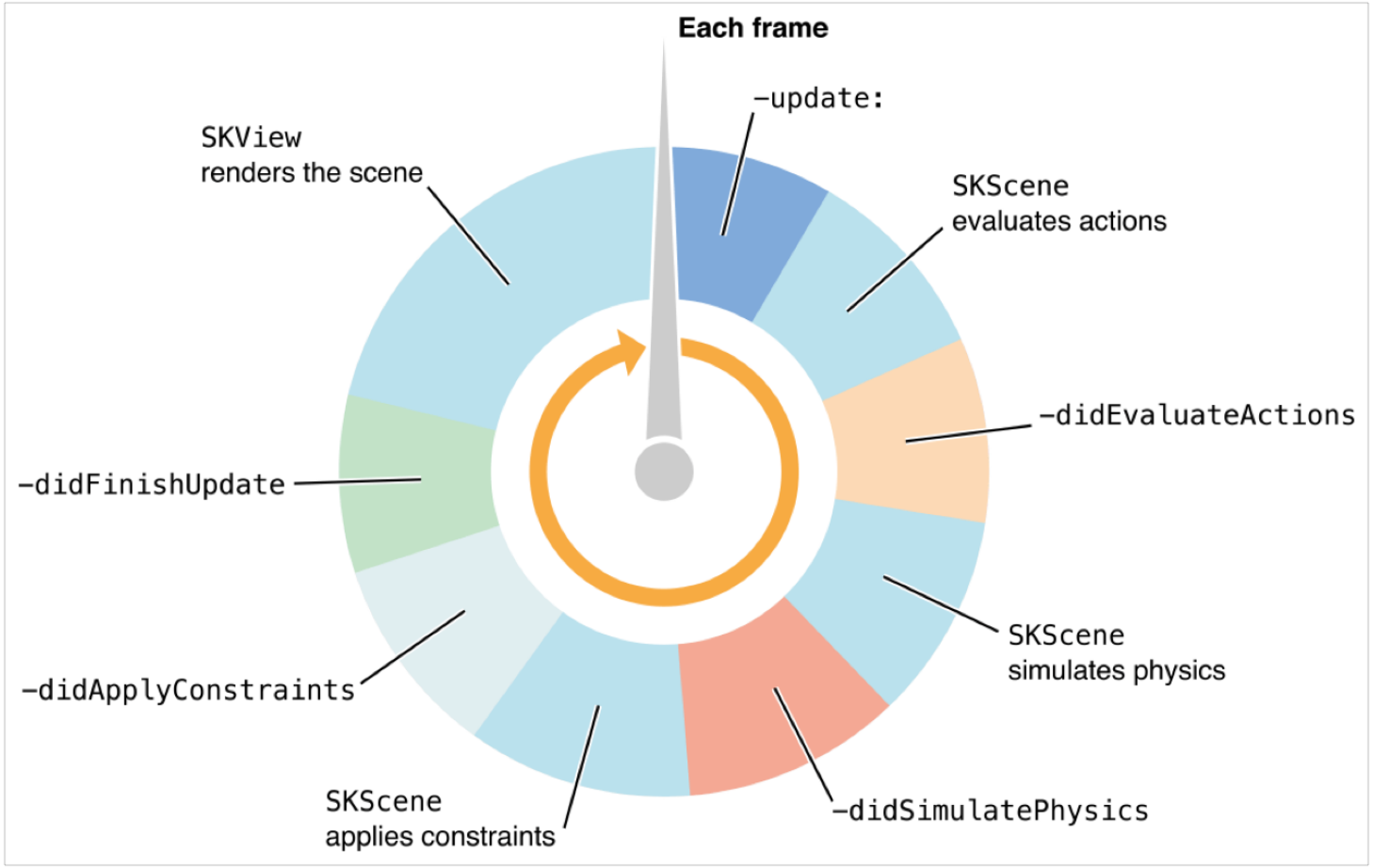


- Ein `SKScene` Objekt kann folgendes enthalten:
  - Visueller Inhalt (Sprite, Text, Partikel, Animationen)
  - Spiellogik

- Um eine neue Szene zu implementieren, kann man entweder von `SKScene` erben (in den meisten Fällen bevorzugt), oder das `SKSceneDelegate` implementieren.

- Mögliche Aktionen:
  - Hintergrundfarbe ändern
  - Viewport ermitteln
  - Die Physiksimulation verwalten (z.B. Schwerkraft konfigurieren, siehe `SKPhysicsWorld`)
  - In die Szenenrepräsentation und den Animation/Simulationsablauf eingreifen
  - (Positional) Audio konfigurieren (siehe `SKAudioNode` und `listener` Property von `SKScene`)

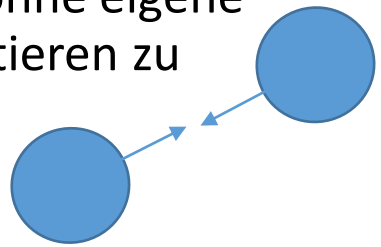
SpriteKit – Game Loop



https://developer.apple.com/documentation/spritekit/skscene

`update:` Wird einmal pro Frame aufgerufen. Kann für Spielelogik verwendet werden.

`didApplyConstraints:` Wird aufgerufen, nachdem alle Constraints angewandt wurden. Constraints (`SKConstraint`) werden verwendet, um Beziehungen zwischen Szenenelementen auszudrücken (bzgl. Rotation, oder Position), ohne eigene Logik implementieren zu müssen.

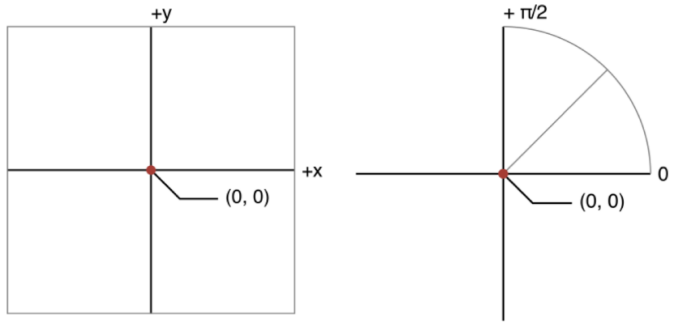


SpriteKit – SKNode

- Alle (visuellen) Elemente einer Szene werden mit Objekten von Klassen, die von SKNode ableiten dargestellt:
  - `SKSpriteNode`: Darstellung von 2D Texturen auf einem Quad.
  - `SKLightNode`: Licht und Schatten für ausgewählte Sprites.
  - `SKAudioNode`: (Positionsbasiertes) Abspielen von Audiosamples.
  - `SKEmitterNode`: Darstellung und Animation von Partikeln.
  - `SKLabelNode`: Darstellung von Text
  - …
- Jedes SKNode Objekt verfügt über eine Position (x,y), Skalierung (x,y), Rotation (z-Achse) und Bounding Box
- SKNode Objekte sind nicht sichtbar, können aber benutzt werden, um andere Szenenelemente zu gruppieren (z.B. um mehrere Layer zu gruppieren in Parallax Games) => Position + Rotation + Skalierung werden auch auf Kindelemente angewandt

SpriteKit – Koordinatensystem & Darstellung

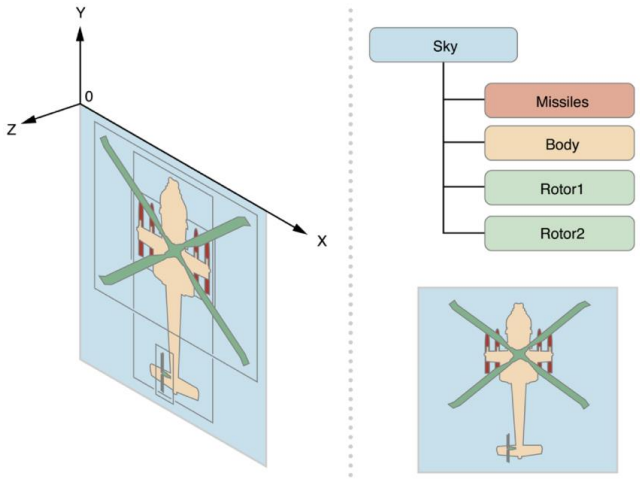
- Koordinatensystem:



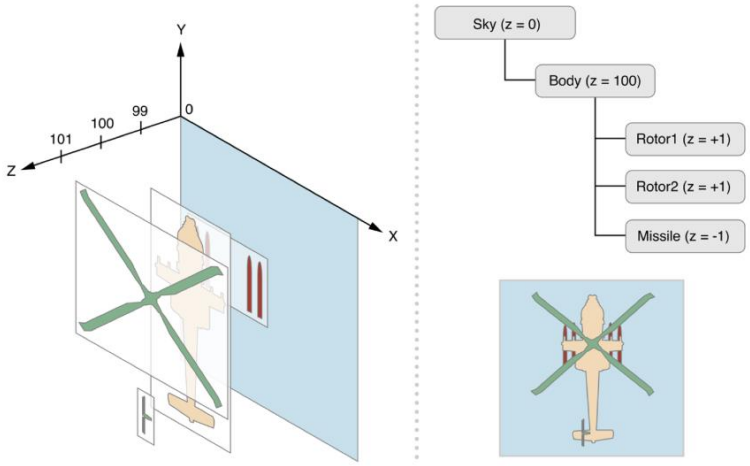
Performance: Reihenfolge gemäß `zPosition` erlaubt dem Renderer, nach Textur zu sortieren (wenn `ignoreSiblingsOrder == true`).

- Darstellungsreihenfolge:

Reihenfolge gemäß **Hierarchie & Ordnung**



Reihenfolge gemäß **`zPosition`**



https://developer.apple.com/documentation/spritekit/sknode

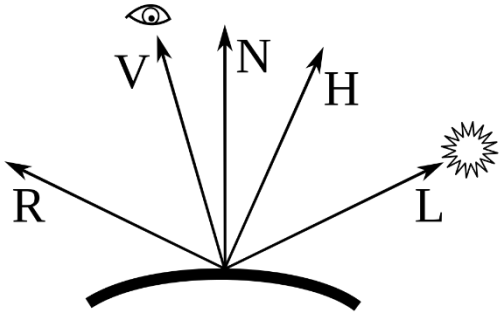
SpriteKit – Erzeugung

- Beispiel:

```
class GameScene : SKScene {
  var label : SKLabelNode!
  var sprite : SKSprite!

  override func didMove(to view: SKView) {
    label = SKLabelNode(fontNames: „Courier“)
    label.text = „Hello, world!“
    label.position = CGPoint(x:100, y:100)
    addChild(label)

    sprite = SKSpriteNode(imageNamed: „sprite.jpg“)
    sprite.position = CGPoint(x:400, y:300)
    sprite.zPosition = -1
    addChild(sprite)
  }
}
```

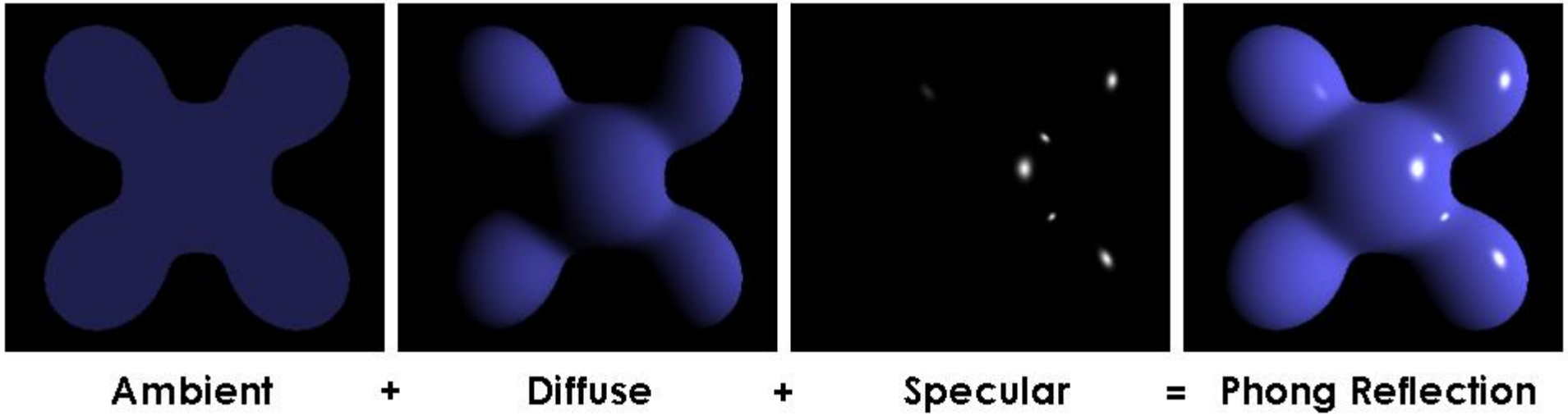
Aus dem „Content“ Ordner in XCode.

SpriteKit – Licht & Schatten



https://en.wikipedia.org/wiki/Phong_reflection_model#/media/File:Blinn_Vectors.svg

Exkurs: Phong-Beleuchtungsmodell



https://de.wikipedia.org/wiki/Phong-Beleuchtungsmodell#/media/File:Phong_components_version_4.png

`SKLightNode.ambientColor` | `lightColor`

Lichtstärken

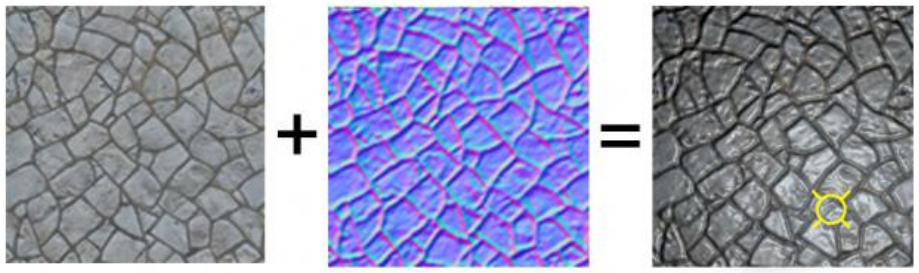
Reflexionsfaktoren

$$I_{\text{out}} = I_{\text{a}}\, k_{\text{ambient}} + I_{\text{in}} \left[k_{\text{diffus}} \left(\vec{L} \cdot \vec{N}\right) + k_{\text{specular}} \frac{n+2}{2\pi} \left(\vec{R} \cdot \vec{V}\right)^n \right]$$

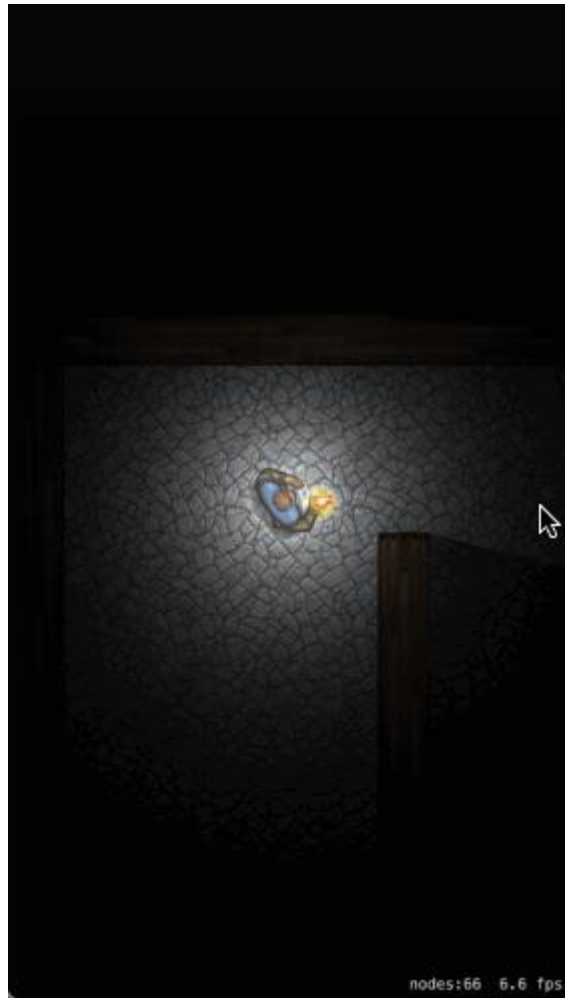
SpriteKit – Licht & Schatten

SpriteKit verfügt über ein System zur Berechnung von Licht & Schatteneffekten.

- Licht:
  - Hinzufügen eines `SKLightNode` Objekts zu einem `SKNodeSprite` Objekt.
  - Licht hat Category Mask
  - Sprite hat Light Mask, Shadow Cast Mask, Shadowed Mask
  - If **Category Mask AND Light/Shadow/Shadowed Mask** == true wird Lichtberechnung auf Sprite angewendet.
- Normal Maps:



https://www.raywenderlich.com/158961/introduction-spritekit-scene-editor

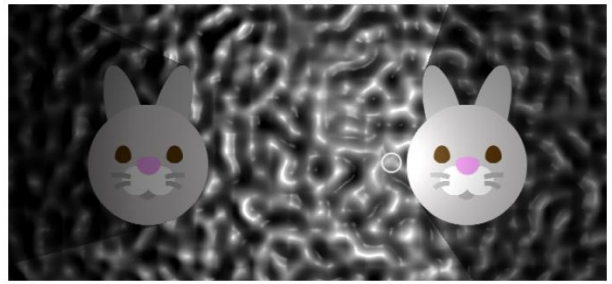


https://www.raywenderlich.com/158961/introduction-spritekit-scene-editor

SpriteKit – Licht & Schatten

Beispiel:

https://developer.apple.com/documentation/spritekit/skspritenode



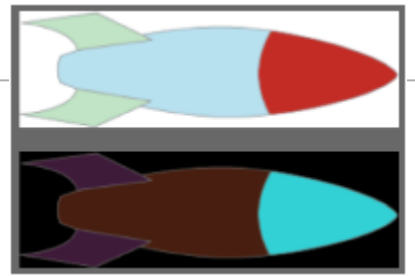
```
let lightNode = SKLightNode()
lightNode.categoryBitMask = 0b0001
lightNode.lightColor = .white
scene.addChild(lightNode)

let background = SKSpriteNode(texture: noiseTexture,
                              normalMap: noiseTexture.generatingNormalMap())
background.lightingBitMask = 0b0001
scene.addChild(background)

for position in [CGPoint(x: -10, y: 0), CGPoint(x: 10, y: 0)] {
  let rabbit = SKSpriteNode(imageNamed: "rabbit")
  rabbit.position = position
  spriteKitViewController.scene.addChild(rabbit)
  rabbit.lightingBitMask = 0b0001
  rabbit.shadowCastBitMask = 0b0001
}
```

SpriteKit – Shader Programmierung

Ein SKSpriteNode Objekt hat ein shader Property, mit dem ein OpenGL ES fragment shader definiert werden kann.

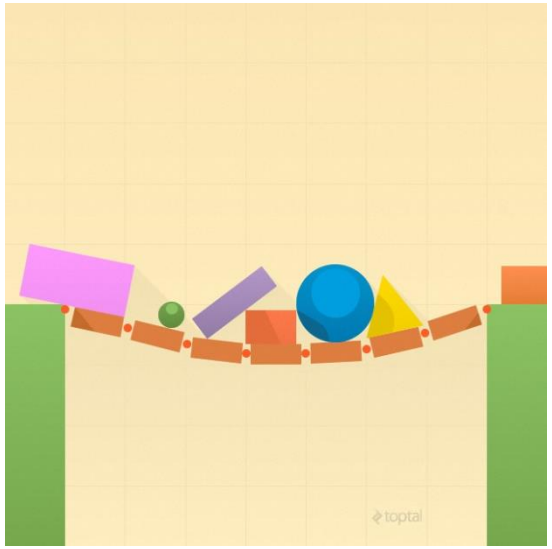
```
sprite = SKSpriteNode(imageNamed: „rocket.jpg“)
  sprite.position = CGPoint(x:400, y:300)
  sprite.zPosition = -1
  sprite.shader = SKShader(source:
    "void main() { " +
    "    gl_FragColor = vec4(1.0 - SKDefaultShading().rgb, SKDefaultShading().a); " +
    "}")

  addChild(sprite)
```

https://developer.apple.com/documentation/spritekit/skspritenode



SpriteKit - Physik

Das SpriteKit enthält eine einfach zu verwendende 2D-Physik-Engine, die Starrkörperbewegungen simulieren kann.



https://www.toptal.com/game/video-game-physics-part-iii-constrained-rigid-body-simulation

Das SpriteKit enthält eine einfach zu verwendende 2D-Physik-Engine, die Starrkörperbewegungen simulieren kann.

```
let box = SKSpriteNode(color: UIColor.red, size:
CGSize(width: 64, height: 64))
addChild(box)
//Physics body for box
box.physicsBody = SKPhysicsBody(rectangleOf:
CGSize(width: 64, height: 64))
box.physicsBody?.restitution = 0.4 //”Bounciness”
//Physics body for the whole scene
physicsBody = SKPhysicsBody(edgeLoopFrom: frame)
```

SpriteKit - SKPhysicsWorld

- Jede Szene enthält ein Property des Typs `SKPhysicsWorld`, welches die komplette Physiksimulation übernimmt.

- Folgendes lässt sich damit bewerkstelligen:

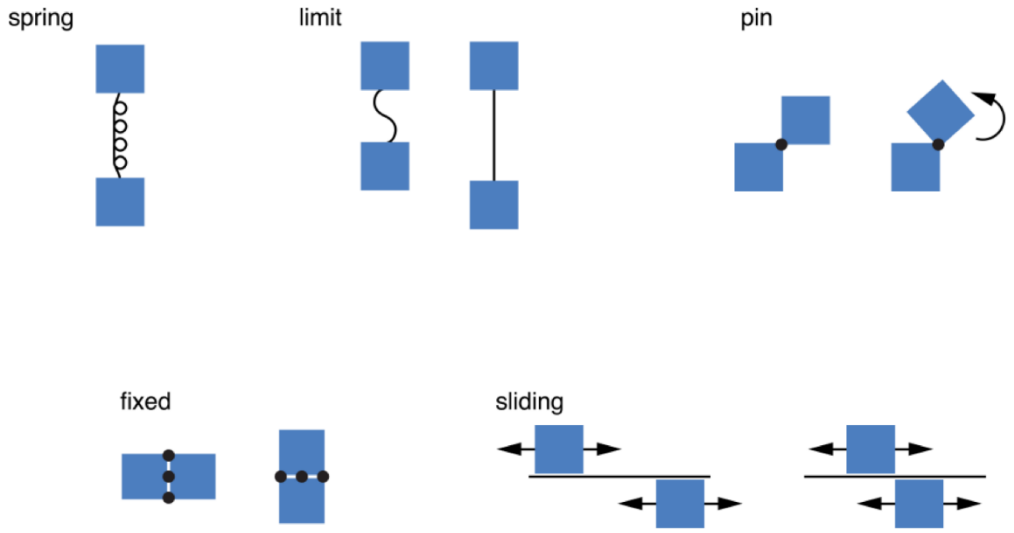
    - Setzen von Simulationsparametern Schwerkraft (`gravity`, als 3D Vektor) und Simulationsgeschwindigkeit (`speed`, 1.0: normal, 0.0: pausiert)

    - Suche von Körpern (`SKPhysicsBody`) basierend auf Punkten, Regionen, Strahlen zwischen Körpern.

Einfluss der Schwerkraft lässt sich durch die Eigenschaft `affectedByGravity` ausschalten.

SpriteKit – Physikkörper verbinden

SKPhysicsBody Objekte lassen sich verbinden:



https://developer.apple.com/documentation/spritekit/skphysicsjoint

Verschiedene Verbindungstypen werden von Klassen, die von `SKPhysicsJoint` erben, dargestellt.

```
let pinJoint =
SKPhysicsJointPin.joint(withBodyA:
pin.physicsBody!,
bodyB: piston.physicsBody!,
anchor: pin.position)

scene.physicsWorld.add(pinJoint)
```

SpriteKit - Kollisionserkennung

Auf Kollisionen, die von der Physik-Engine gemeldet werden, kann durch die Implementierung des `SKPhysicsContactDelegate` reagiert werden:

```
class GameScene: SKScene, SKPhysicsContactDelegate {

  override func didMove(to view: SKView) {
    physicsWorld.contactDelegate = self
    let box1 = SKSpriteNode(/*…*/)
    box1.physicsBody = SKPhysicsBody(/*…*/)
    box1.physicsBody!.categoryBitMask = 0b0001
    box1.physicsBody!.contactTestBitMask = 0b0001
    box1.physicsBody!.collisionBitMask = 0b0001
    box1.physicsBody?.isDynamic = false //No motion
  }
  func collisionBetween(ball: SKNode, object: SKNode){/*…*/}
}
```

`collisionTestBitMask`

Welche anderen Objekte sollen mit diesem Objekt kollidieren?

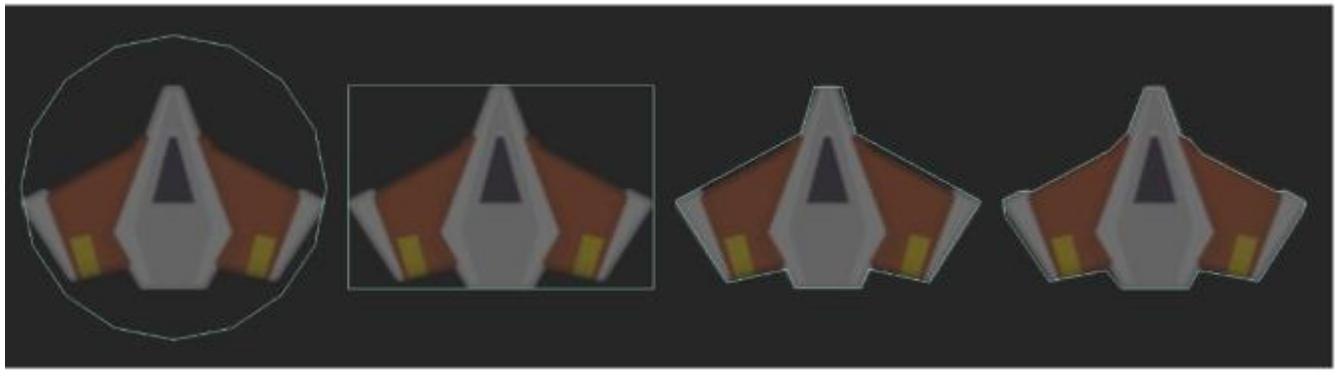
`contactTestBitMask`

Über welche Kollisionen soll benachrichtigt werden?

`usesPreciseCollisionDetection = true` für schnelle, kleine Objekte

SpriteKit - Kollisionserkennung

Beim Erstellen eines SKPhysicsBody Objekt lässt sich festlegen, wie die für die Kollisionsprüfung benötigte geometrische Form des Objekts aussehen soll:



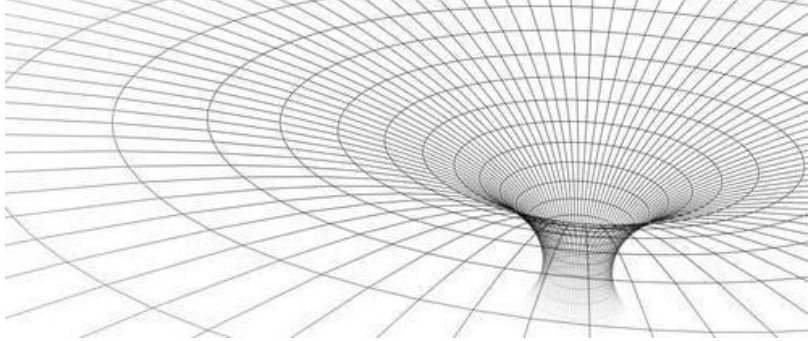
https://developer.apple.com/documentation/spritekit/skphysicsbody

Wichtig: Ein Körper hat Masse, Dichte, Fläche und Ladung

Es existieren drei Arten physikalischer Körper:

- Dynamic Volumes: Volumen + Masse, Beeinflusst durch Kräfte und andere Objekte
- Static Volumes: Volumen + Masse, Nicht beeinflusst durch Kräfte und andere Objekte
- Static, Volume-less: „Kanten“ ohne Volumen (z.B. als Begrenzung der Szene)

SpriteKit - Kraftfelder



http://www.ipodphysics.com/forces-gravitational-field.php

Es ist möglich, einer Szene Kraftfelder (`SKFieldNode`) hinzuzufügen, die dann Nodes mit `SKPhysicsBody` Objekt beeinflussen können:

```
physicsWorld.gravity = CGVector(dx:0, dy: 0)

let gravity = SKFieldNode.linearGravityField(withVector: vector_float3(0,-1,0))
gravity.categoryBitMask = 0x1
gravity.strength = 9.8
gravity.region = SKRegion(radius: 100)

let ship = SKSpriteNode(/*…*/)
ship.physicsBody = SKPhysicsBody(/*…*/)
ship.physicsBody?.fieldBitMask = 0x1

addChild(gravity)
addChild(ship)
```

Interessant: Es gibt sogar elektrische Felder uvm. (auch eigene Ideen implementierbar)!

SpriteKit - Partikeleffekte

- Partikeleffekte lassen sich mit einem `SKEmitterNode` realisieren.
- Anwendungsgebiete: Rauch, Feuer(-werk), Funken, …
- Neben der zu verwendenden Textur lassen sich viele Parameter festlegen:
  - Partikelskalierung, -skalierungsgeschwindigkeit, -skalierungsweite
  - „Geburtsrate“
  - Anzahl Partikel
  - Startposition (Variation und zeitliche Veränderung)
  - …

http://sound-of-silence.com/?article=20170205

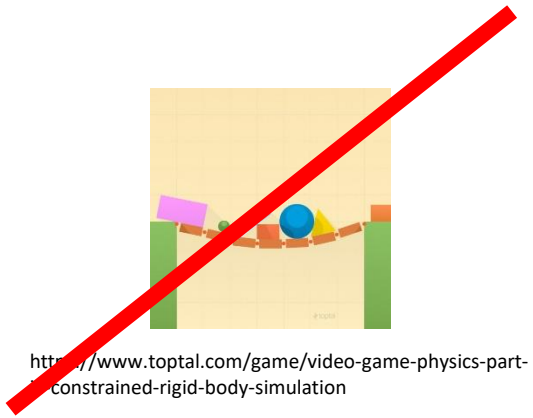
- Auch wenn man jeden Parameter programmatisch anpassen kann, ist es ratsam den **XCode Particle Emitter Editor** zu verwenden:

```
var emitter = SKEmitterNode(fileNamed: "smoke.sks")
```

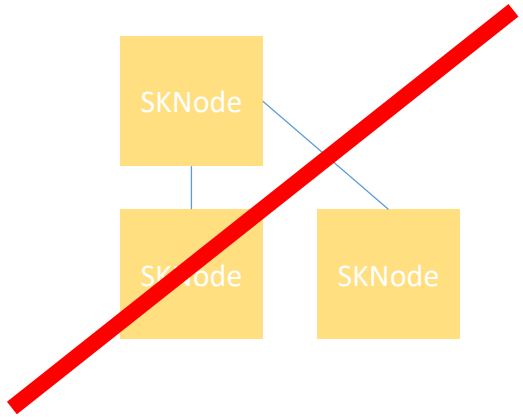
SpriteKit – PE - Partikelobjekte

Einzelne Partikel lassen sich **nicht** wie normale SKNode Objekte in der Szene ansprechen. Das bedeutet:

- Einzelnen Partikeln kann kein `SKPhysicsBody` Objekt zugewiesen werden.
- Einzelne Partikel können keine SKNode spezifischen Operationen ausführen.



http://www.toptal.com/game/video-game-physics-part-constrained-rigid-body-simulation



SpriteKit – PE – Keyframe-Sequenzen

- Das Aussehen und Verhalten von Partikeln lässt sich über Keyframe-Sequenzen festlegen:

```
var emitter = SKEmitterNode(fileNamed: "smoke.sks")

var sequence = SKKeyframeSequence(keyframeValues: [0.2, 0.7, 0.1],
                                  times: [0.0, 0.250, 0.75])

Emitter.particleScaleSequence = sequence
```

- Neben dem `particleScaleSequence` Property stehen auch noch folgende Properties zur Verfügung:
  - `particleColorSequence`
  - `particleColorBlendFactorSequence` (Blendfaktor zwischen Partikeltextur und -farbe)
  - `particleAlphaSequence`

SpriteKit – PE – Partikel und Aktionen

- Es ist möglich, dem Emitter eine Aktion zuzuweisen, die dann für alle Partikel ausgeführt wird:

```
var emitter = SKEmitterNode(fileNamed: "smoke.sks")

var action = SKAction.fadeOut(withDuration: 1)
Emitter.particleAction = action
```

Wichtig: Aktionen von bereits emittierten Partikeln lassen sich nicht entfernen.

Wichtig: Komplexe Aktionen haben großen Einfluss auf die Performance.

SpriteKit – PE – Partikel und Target-Nodes

- Partikel sind „Kinder“ des Emitter-Nodes => Wenn Emitter-Node rotiert (oder verschoben) wird, werden alle zugehörigen (also auch bereits emittierte) Partikel ebenso transformiert.

- In manchen Fällen ist das nicht das gewünschte Verhalten. Vielmehr sollen emittierte Partikel unabhängig von ihrem Emitter sein.

- Das kann man mit `targetNode` Property eines `SKEmitterNode` Objekts bewerkstelligen:

```
//Assumption: scene object exists.

var emitter = SKEmitterNode(fileNamed: "smoke.sks")

emitter.targetNode = scene;
```

SpriteKit – PE - Tips

- Partikel sind leichtgewichtiger als Sprites, müssen aber trotzdem berechnet werden. Deswegen:
  - So wenig wie möglich
  - Geringe Geburtenrate, kurze Lebensdauer
  - So groß wie möglich

- Die Anwendung von Aktionen auf Partikel ist sehr aufwendig => Nur, wenn nicht anders möglich.

- Partikelemitter, die nicht sichtbar sind, sollten von der Szene entfernt werden.

SpriteKit - Aktionen

- Aktionen können Struktur und Inhalt eines SKNode Objekts ändern.

- Aktionen (SKAction) sind dabei als **Kommando-** und **Kompositum**-Entwurfsmuster realisiert:

  - Kompositum: Aktionen können „Kind“-Aktionen beinhalten (Sequenz, Gruppe, Wiederholung).
  - Kommando: Aktionen sind in sich geschlossen und haben keinen Status (State).

- Aktionen sind entweder **instantan** (z.B. Node Entfernung) oder **nicht-instantan** (z.B. Animation).

- Aktionen werden auf Nodes angewendet:

```
let action = SKAction.moveBy(x: 0.0,
                             y: 100.0,
                             duration: 1.0)
node.run(action)
```

- Aktionen können aus *.sks-Dateien geladen werden.

SpriteKit - Aktionen

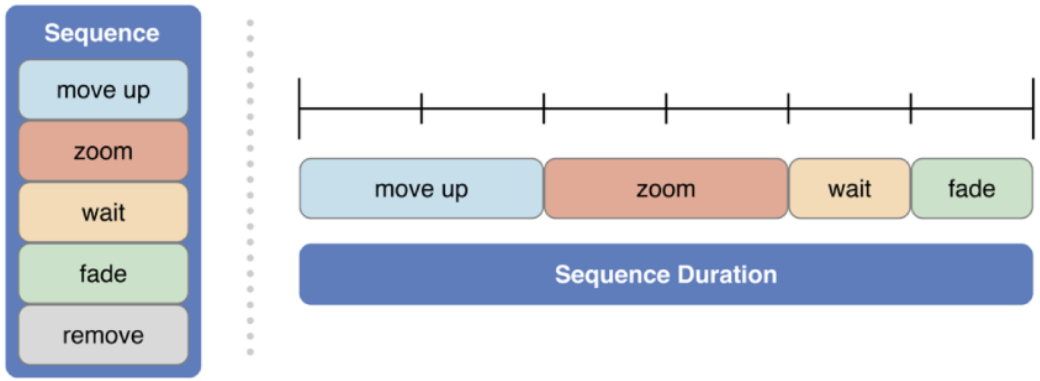
- Viele Aktionen implementieren die `reversed()`-Methode, welche eine Aktion zurückgibt, die den Effekt der Aktion rückgängig macht.

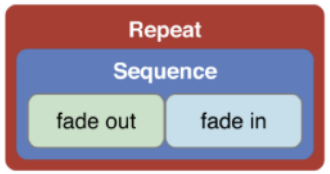
- Aktionen sind unveränderlich (keinen State, einmal konfiguriert, können die meisten Properties nicht mehr verändert werden). Vorteil:
  - Effiziente Ausführung.
  - Aktionen können von vielen Objekten verwendet werden (Wiederverwendbarkeit).

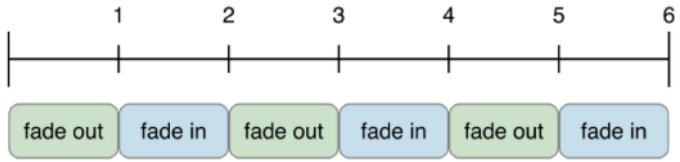
- Aktionen abbrechen: `SKNode.removeAllActions()`

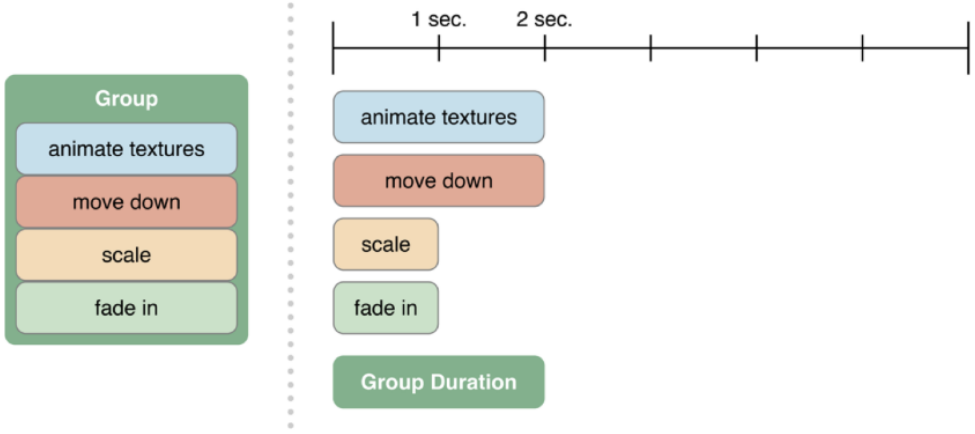
- Callback, wenn Aktion beendet wurde: `SKNode.run(_:completion:)`

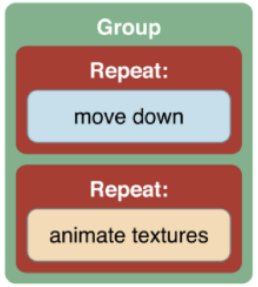
SpriteKit - Aktionen

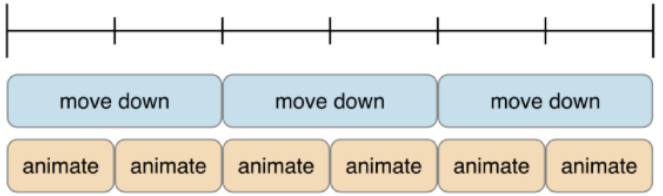












SpriteKit - Kameras

- Das SKCameraNode Objekt erlaubt das festlegen einer Position, Rotation und Skalierung, aus der eine Szene gerendert werden soll:

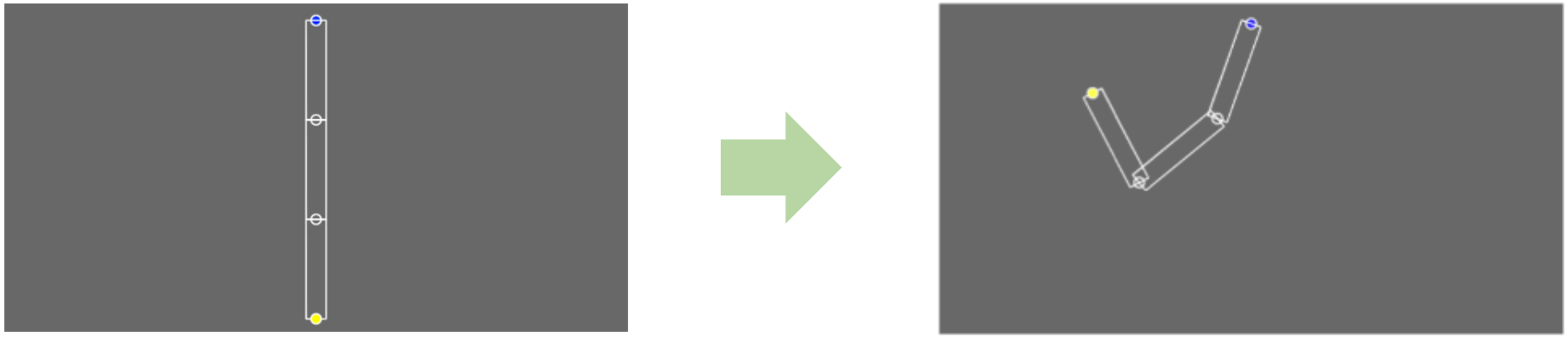
```
override func didMove(to view: SKView) {
  super.didMove(to: view)
  camera = SKCameraNode()
  self.camera = camera
  self.addChild(camera)
}
```

Wichtig: SKCameraNode Objekte haben alle Möglichkeiten der SKNode Klasse (Physik, Aktionen, …)

- Die Szene wird so gerendert, das der Ursprung der Kamera im Szenenmittelpunkt liegt.
- Dabei wird auf Szenenelemente die inverse Skalierung und Rotation angewandt.

SpriteKit – Aktionen – Weitere Möglichkeiten

- Aktionen können verwendet werden, um Inverse Kinematik Simulationen zu erstellen:



https://developer.apple.com/documentation/spritekit/skaction#1655635

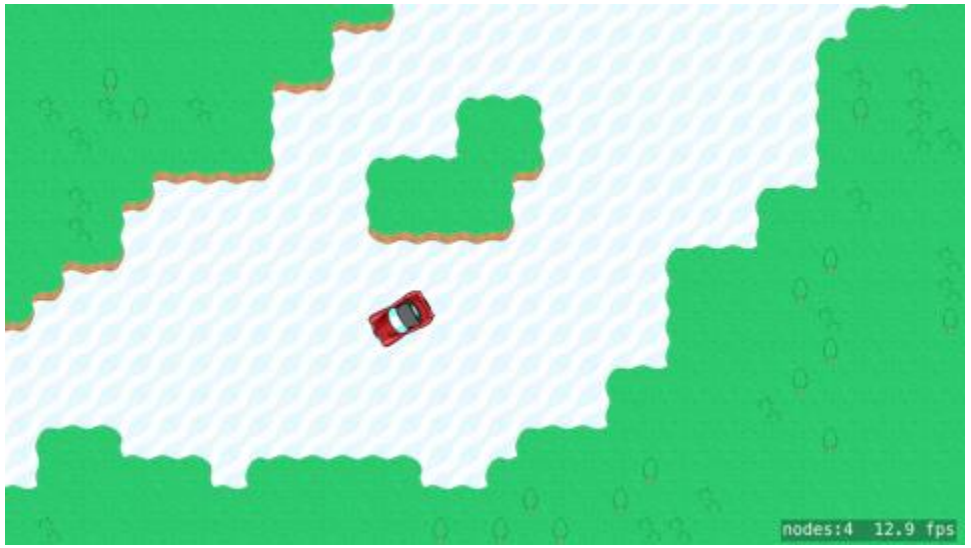
- Für die Erstellung eigener Aktionen, wird **nicht** von `SKAction` abgeleitet, sondern z.B mit…

```
SKAction.customAction(withDuration duration: TimeInterval,
                      actionBlock block: @escaping (SKNode, CGFloat) -> Void) -> SKAction
```

…ein neues `SKAction` Objekt erzeugt, die ein Closure aufruft.

SpriteKit – Tile Maps

Tile Maps (`SKTileMapNode` Objekte) können verwendet werden um 2D Spielwelten (Rechtecke, Hexagons, Iso) aus kleinen „Tiles“ aufzubauen:

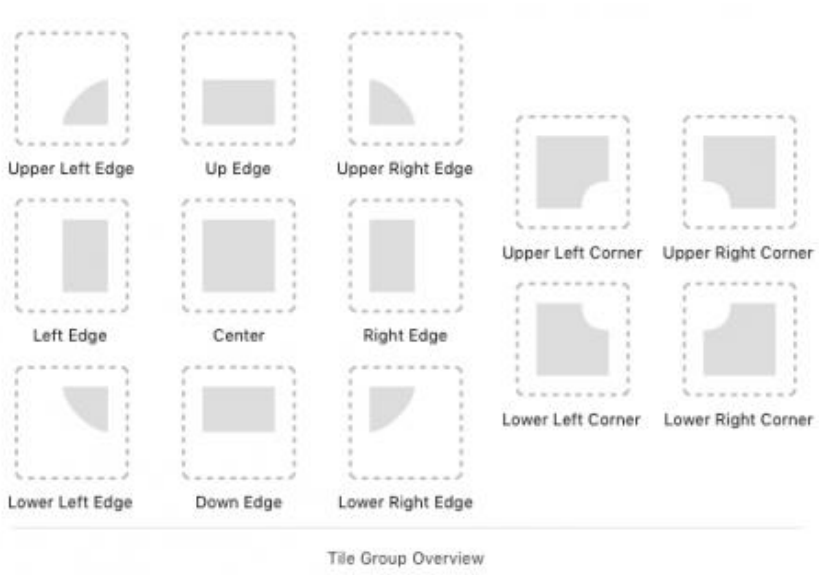


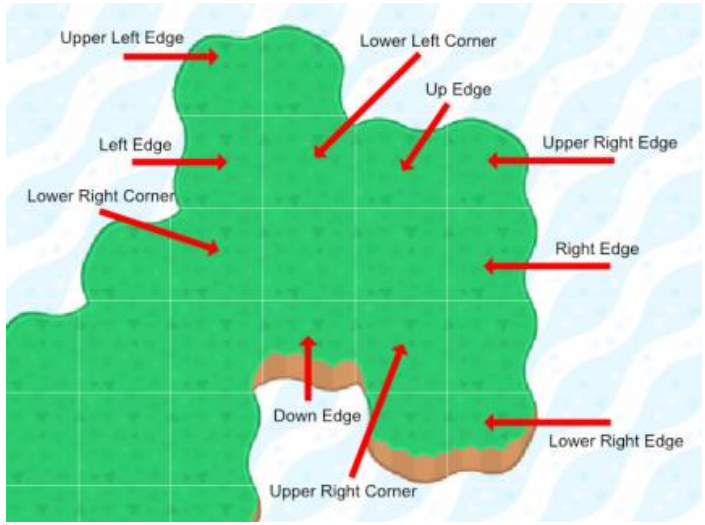
https://www.raywenderlich.com/137216/whats-new-spritekit-ios-10-look-tile-maps

- Für jede Rasterposition, wird ein Tile festgelegt (bzw. eine Tile-Gruppe, siehe nächste Folie), welches aus einer Menge an Tiles ausgewählt werden kann.
- Die Erstellung von Tile Maps kann im Editor erledigt werden.

SpriteKit – Tile Maps

- Tile Sets (`SKTileSet`) bestehen aus einer Menge von Tile-Gruppen (`SKTileGroup`), z.B. für Gras.
- Jede Tile-Gruppe besteht aus einer Menge von Regeln (`SKTileGroupRule`) und einer Menge von Tile-Definitionen (`SKTileDefinition` => enthält Textur).





https://www.raywenderlich.com/137216/whats-new-spritekit-ios-10-look-tile-maps

- Mehrere Tile-Definitionen pro Regel => zu platzierendes Tile wird per Zufall ausgewählt.

SpriteKit – XCode Szeneneditor

- Praktisches Tool, um Szenen (nicht-programmatisch) zu erstellen.
- Es können Node-Hierarchien angelegt und konfiguriert werden.
- Resultat wird in *.sks Dateien gespeichert (wird von SKScene Konstruktur geladen).
- Zugriff auf Nodes aus dem Code heraus:

```
var spriteNode:SKSpriteNode!

override func didMove(to view: SKView) {
    spriteNode = self.childNode(withName: "//sprite1") as? SKSpriteNode
}
```

SpriteKit – Links

- https://www.hackingwithswift.com/read/11/overview
- https://github.com/twostraws/HackingWithSwift
- https://github.com/jVirus/ios-learning-materials

- https://www.raywenderlich.com/137216/whats-new-spritekit-ios-10-look-tile-maps

GameplayKit

Framework, das unabhängig von verwendeten Präsentationsframeworks high-level Bausteine zur Spieleentwicklung bereitstellt.

Zufallszahlen

Strategien

Regelsysteme

Entitäten & Komponenten

Agentensystem

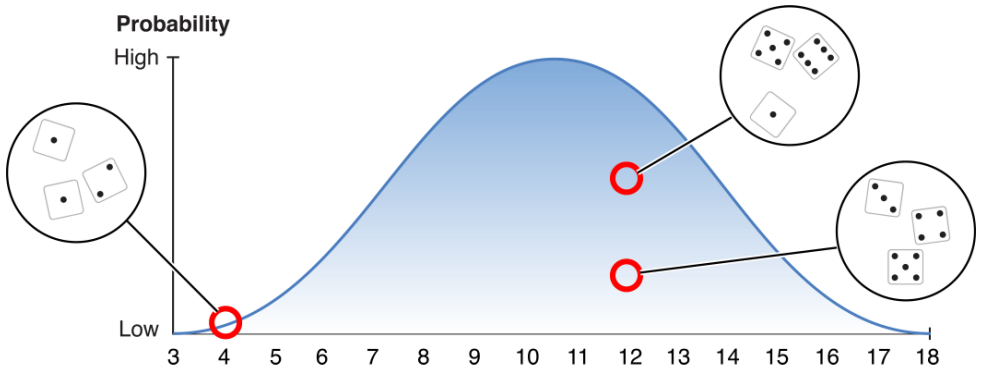
Pfadfindung

Zustandsautomaten

GameplayKit - Zufallszahlen

- Performante Pseudozufallszahlen aus verschiedenen Quellen (Mersenne Twister, ARC4, …)

- Wahrscheinlichkeitsverteilungen (z.B. Normalverteilung)



https://developer.apple.com/library/content/documentation/General/Conceptual/GameplayKit_Guide/RandomSources.html

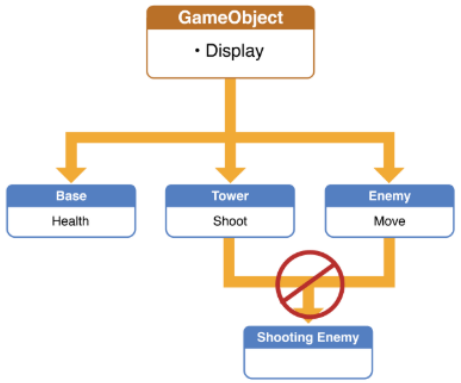
- Determinismus: Es ist möglich, gleiche Sequenzen von Zufallszahlen zu produzieren (für Testing, Multiplayer).

- Vermeidung von Clustern gleicher Werte: `GKShuffledDistribution`

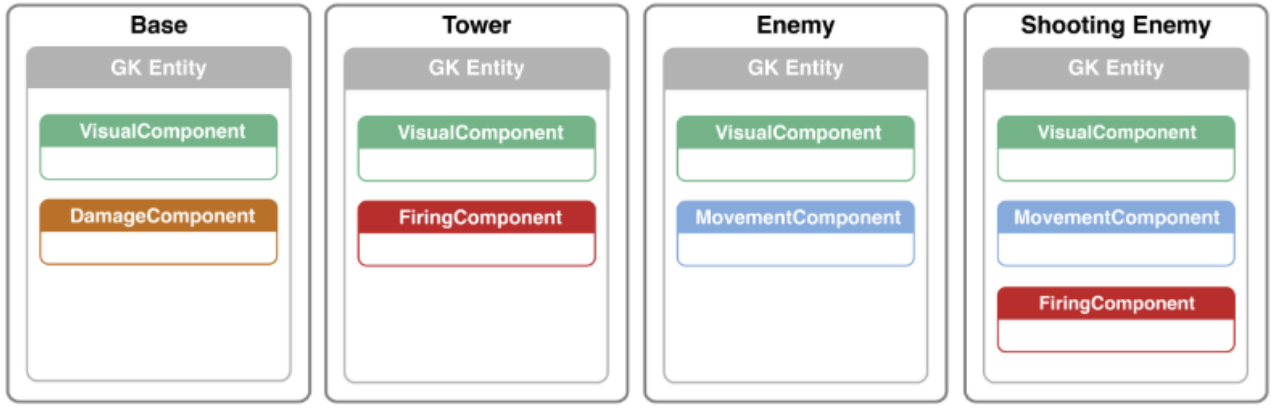
GameplayKit – Entitäten und Komponenten

Elemente einer Architektur, die **nicht auf Vererbung**, sondern auf dem **Entity-Component Entwurfsmuster** basiert.

Problem:



Lösung:



https://developer.apple.com/library/content/documentation/General/Conceptual/GameplayKit_Guide/EntityComponent.html

- `GKEntity.update()` wird dann in der `update()` Methode der Szene aufgerufen (welche dann die `update()` Methode der verwendeten `GKComponent` Objekte aufruft.
- Auch möglich: update() Aufruf pro `GKComponent`. Besser bei großen Entity-Mengen.

GameplayKit - Zustandsautomaten

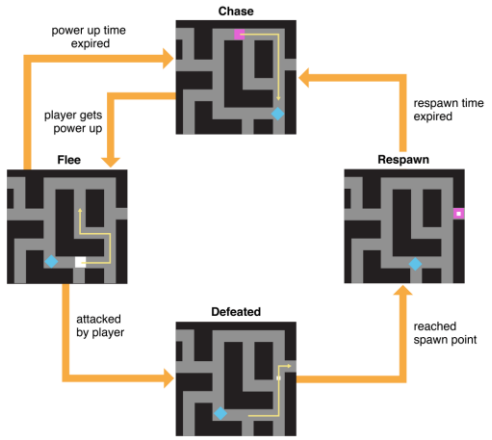
Zustandsautomaten werden mithilfe des State Patterns mit der abstrakten Klasse `GKState` und der Klasse `GKStateMachine` realisiert:

`GKStateMachine:`

- `currentState`
- `canEnterState(AnyClass)`
- `enter(AnyClass) -> bool`
- `update(TimeInverval)`

`GKState:`

- `stateMachine`
- `isValidNextState(AnyClass)->bool`
- `didEnter(from: GKState? )`
- `update(TimeInverval`
- `willExit(to: GKState)`



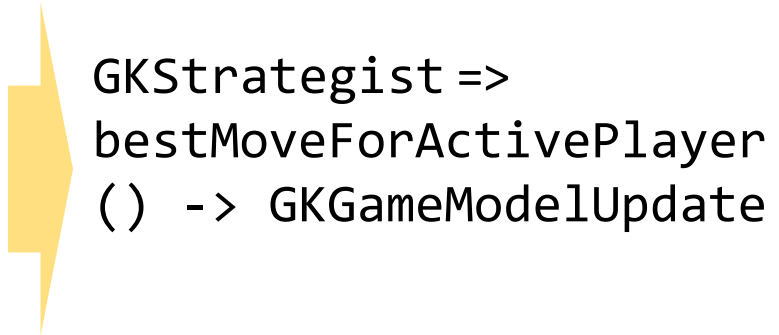
https://developer.apple.com/library/content/documentation/General/Conceptual/GameplayKit_Guide/StateMachine.html

```
Override func isValidNextState(_ stateClass : AnyClass) -> Bool {
  return stateClass is AttackState.Type
}
```

Wo Einsetzen? Spiel UI, Verhalten von Spielaktoren, Animation,…

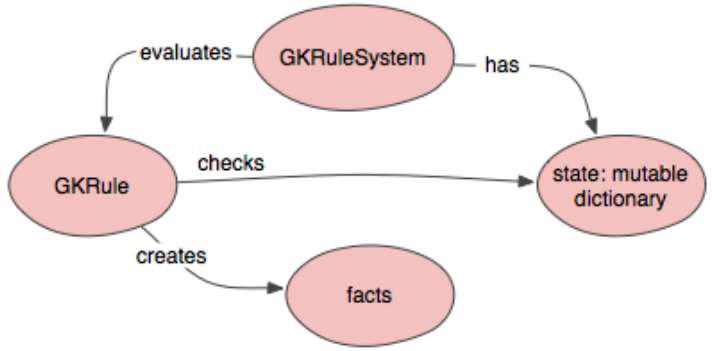
GameplayKit – Strategien

- KI für die Ermittlung der optimalen Strategie (`GKStrategist`) für rundenbasierte Spiele basierend auf dem MinMax Algorithmus (https://de.wikipedia.org/wiki/Minimax-Algorithmus) oder MonteCarlo Sampling (Schneller bei großen State-Spaces)
- Voraussetzungen zur Anwendbarkeit:
  - Rundenbasiert
  - Wenn ein Spieler gewinnt, verlieren automatisch alle anderen
  - Deterministisch
  - Perfekte Information
- Zentrale Protokolle
  - `GKGameModel:` Repräsentiert den aktuellen Zustand des Spiels (Spieler, Mögliche Spielzüge und Score für S.)
  - `GKGameModelPlayer`: Repräsentiert einen Spieler im Spiel (Id, Status)
  - `GKGameModelUpdate`: Modelliert eine Spielaktion (Move), die auf das Spielmodell angewendet werden kann.



GameplayKit - Regelsystem

- Ein Regelsystem (`GKRuleSystem`) besteht aus einer Agenda von Regeln (`GKRule`), State-Information (`Dictionary`) und Fakten (`Any`).
- Beim Auswerten des Regelsystems (`.evaluate()`) passiert folgendes für jede Regel:
  - Das Regelprädikat wird auf Basis des States ausgewertet (input)
  - Wenn true => Fakten (irgendein Objekt) werden definiert (output).
  - Input kann auch aus Fakten bestehen, Output kann State verändern.
- Definierte Fakten haben einen „Grade“-Wert zwischen 0 und 1, der abgefragt werden kann.
- Erlaubt den Einsatz von **Fuzzylogik** (min, max anstelle von and, or).



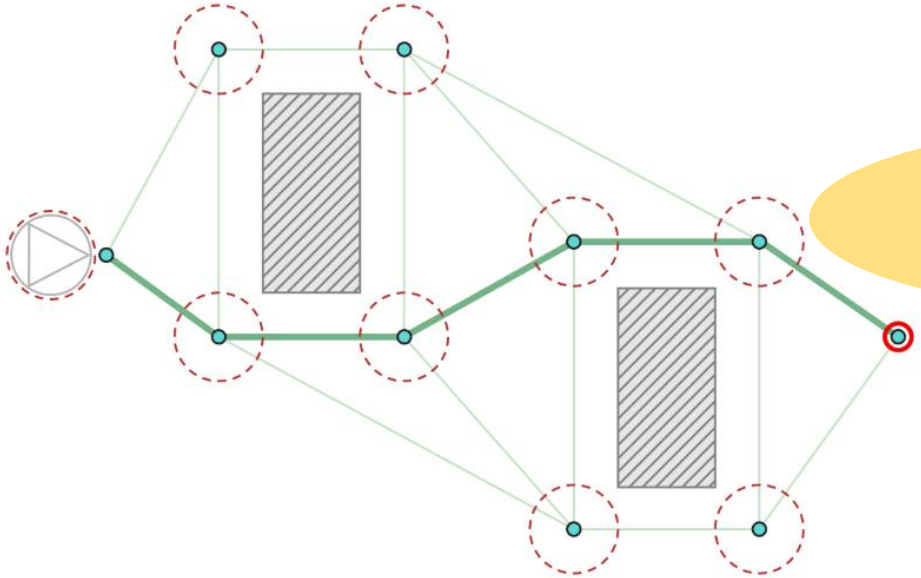
https://www.smashingmagazine.com/2017/04/ios-game-logic-gameplaykit/

GKRule
evaluatePredicate()
performAction()

```
//a data-driven rule
let healthTest = NSPredicate(format: "$player.health < 25")
let rule = GKRule(predicate: healthTest, assertingFact:
"player_weak", grade: 0.5)
```

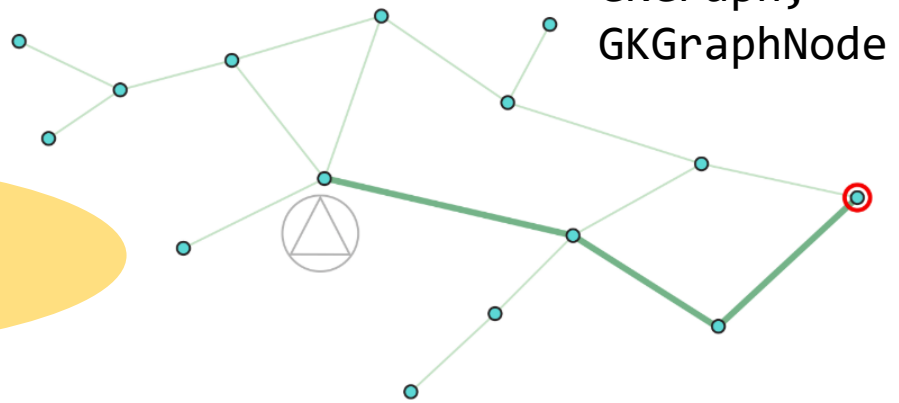
GameplayKit - Pfadfindung

Finden von kürzesten Wegen in Graphen (GKGraph Objekten). Drei Möglichkeiten der Modellierung:

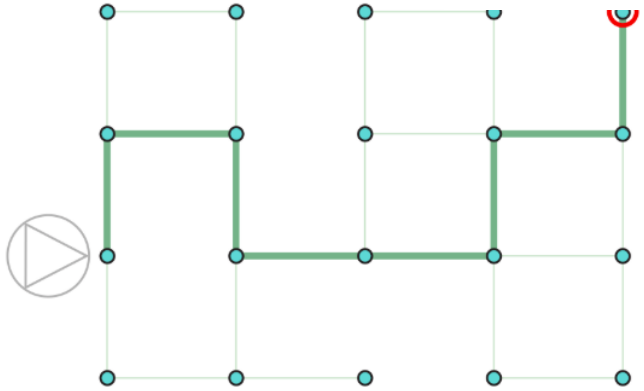


GKObstacleGraph (effizient),
GKMeshGraph (smooth),
GKGraphNode2D (PoIs), GKObstacle (Polygon,…)

GKGraph.findPath()



GKGraph,
GKGraphNode

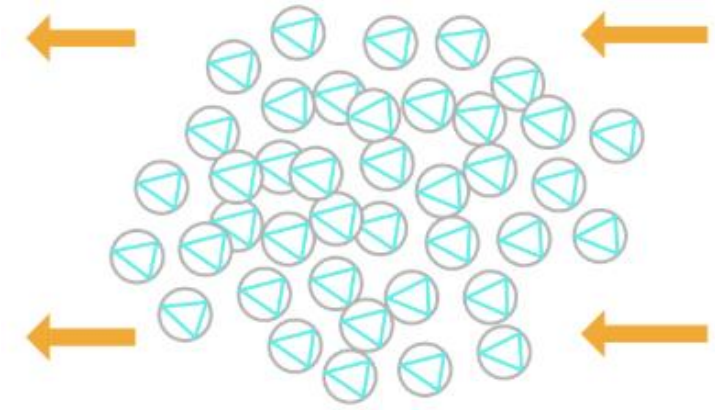


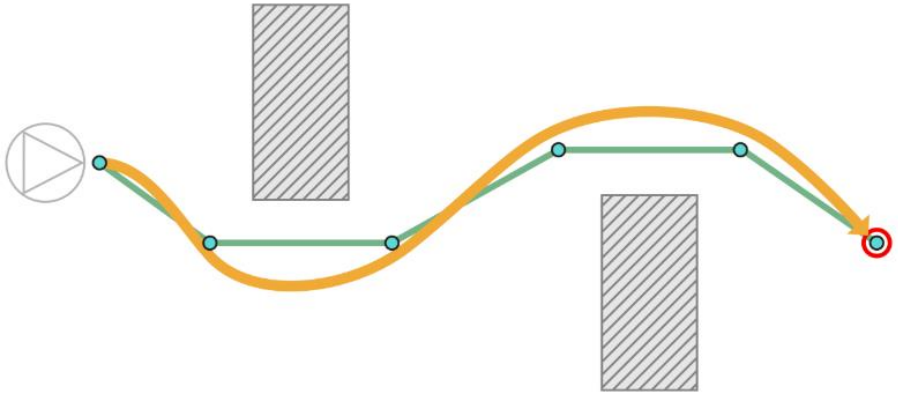
GKGridGraph,
GKGridGraphNode

https://developer.apple.com/library/content/documentation/General/Conceptual/GameplayKit_Guide/Pathfinding.html

GameplayKit – Agentensystem

- Das Agentensystem wird für die Implementierung **autonomer Bewegung** verwendet.
- Ein Agent hat physikalische **Constraints** (Größe, p, v, a) und **ein bestimmtes Verhalten** (`GKBehavior`).
- Verhalten besteht aus einer **gewichteten Menge von Zielen** (`GKGoal`).
- Beispiele für Ziele (Stecken alle in den `GKGoal.init`-Methoden):





https://developer.apple.com/library/content/documentation/General/Conceptual/GameplayKit_Guide/Agent.html

- Agenten & Komponenten: Relevanten Spielentitäten einfach eine `GKAgent`-Komponente hinzufügen.

GameKit – Links

- https://developer.apple.com/library/content/documentation/General/Conceptual/GameplayKit_Guide/index.html