What's New in GameplayKit

Session 608

Bruno Sommer Game Technologies Engineer
Sri Nair Game Technologies Engineer
Michael Brennan Game Technologies Engineer

GameplayKit

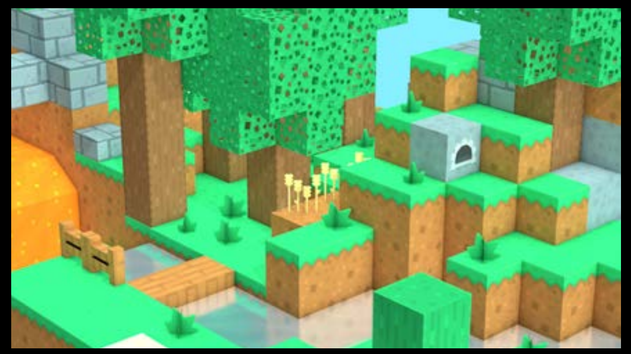
GameplayKit

Your gameplay toolbox

Platformer

RPG

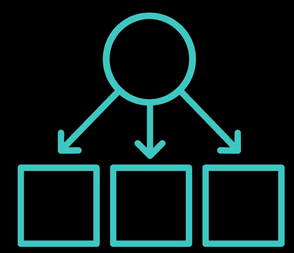
City Builder



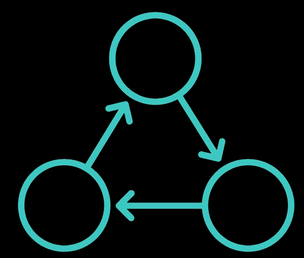
Sandbox

GameplayKit

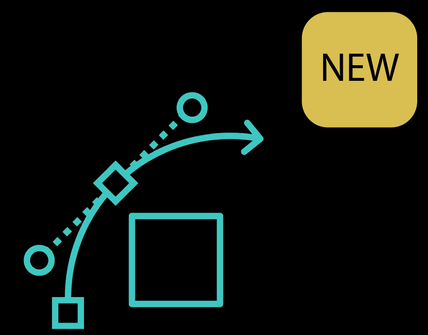
Bringing game ideas to life



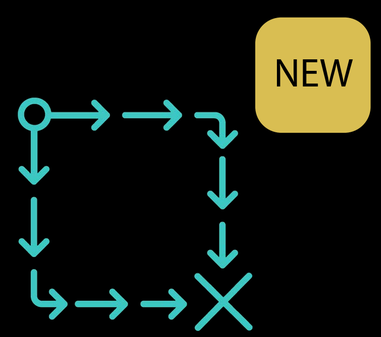
Entities and Components



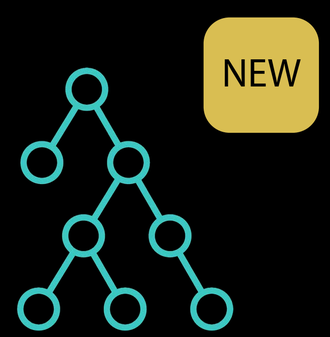
State Machines



Agents



Pathfinding



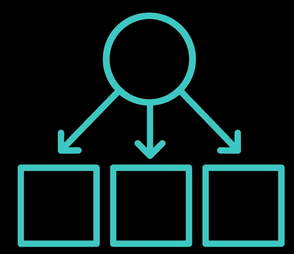
Game AI

Random Sources

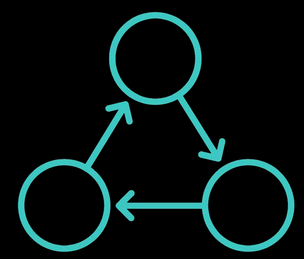
Rule Systems

GameplayKit

Bringing game ideas to life



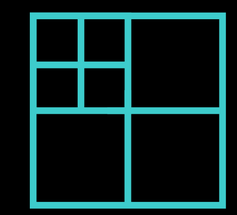
Entities and Components



State Machines

NEW

Agents

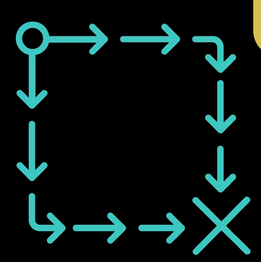


NEW

Spatial Partitioning

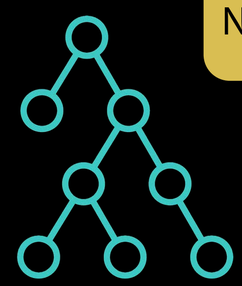
NEW

Procedural Generation



NEW

Pathfinding



NEW

Game AI

Random Sources

Rule Systems

NEW

Xcode Integration

Pathfinding

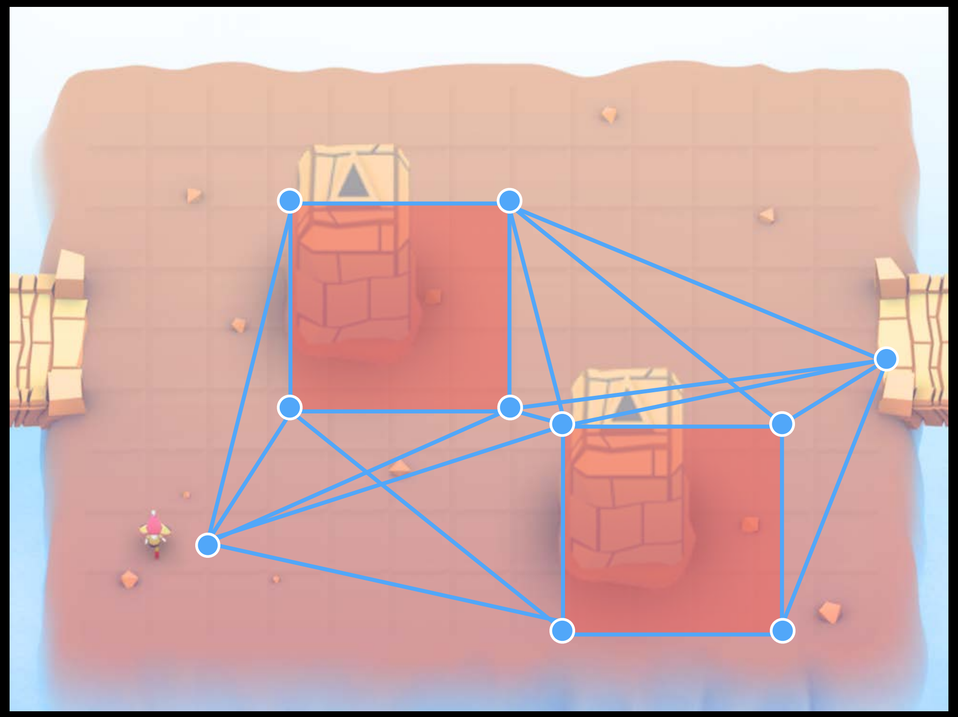
Pathfinding

GKObstacleGraph

Graphs the passable space between a set of obstacles

Very good quality paths but…

- Computationally expensive
- Memory-intensive



Pathfinding

GKMeshGraph

NEW

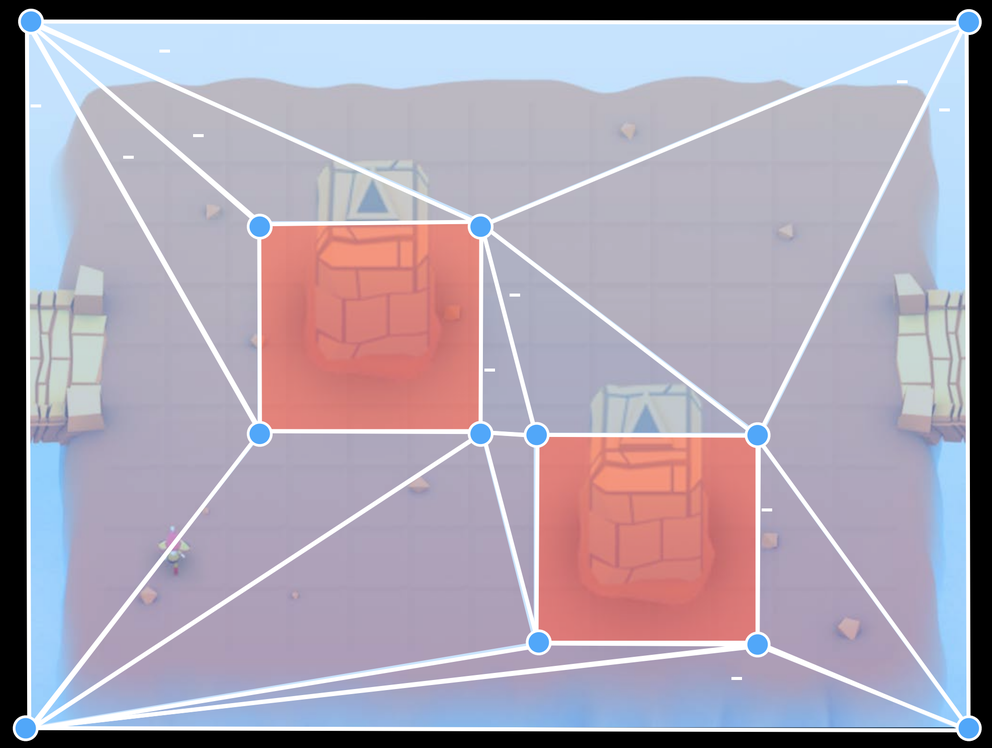
Triangulates space between obstacles

Good quality paths

Fast and low overhead

Node location flexibility

- Centers, vertices, edges



```swift
// Pathfinding
// Mesh graph example

// Create mesh graph of space between (0,0) and (1000,1000)
let meshGraph = GKMeshGraph(bufferRadius:10.0,
                            minCoordinate:float2(0.0,0.0), maxCoordinate:float2(1000.0,1000.0))

// Set triangulation mode - Nodes at triangle vertices and centers
meshGraph.triangulationMode = [.vertices, .centers]

// Add obstacles and triangulate
meshGraph.addObstacles(obstacles)
meshGraph.triangulate()
```

```
// Pathfinding
// Mesh graph example

// Create mesh graph of space between (0,0) and (1000,1000)
let meshGraph = GKMeshGraph(bufferRadius:10.0,
                            minCoordinate:float2(0.0,0.0), maxCoordinate:float2(1000.0,1000.0))

// Set triangulation mode - Nodes at triangle vertices and centers
meshGraph.triangulationMode = [.vertices, .centers]

// Add obstacles and triangulate
meshGraph.addObstacles(obstacles)
meshGraph.triangulate()
```

```swift
// Pathfinding
// Mesh graph example

// Create mesh graph of space between (0,0) and (1000,1000)
let meshGraph = GKMeshGraph(bufferRadius:10.0,
                            minCoordinate:float2(0.0,0.0), maxCoordinate:float2(1000.0,1000.0))

// Set triangulation mode - Nodes at triangle vertices and centers
meshGraph.triangulationMode = [.vertices, .centers]

// Add obstacles and triangulate
meshGraph.addObstacles(obstacles)
meshGraph.triangulate()
```

```
// Pathfinding
// Mesh graph example

// Create mesh graph of space between (0,0) and (1000,1000)
let meshGraph = GKMeshGraph(bufferRadius:10.0,
                            minCoordinate:float2(0.0,0.0), maxCoordinate:float2(1000.0,1000.0))

// Set triangulation mode - Nodes at triangle vertices and centers
meshGraph.triangulationMode = [.vertices, .centers]
```

```
// Add obstacles and triangulate
meshGraph.addObstacles(obstacles)
meshGraph.triangulate()
```

Pathfinding

Custom node classes

NEW

Can attach custom data or logic to nodes

Developer-supplied node class to instantiate

- GKGridGraph
- GKObstacleGraph
- GKMeshGraph

Appropriate init() is called on your nodes

Generics support; no casting required

GKGridGraph

GKObstacleGraph

GKMeshGraph

nodeClass:

Agents

Agents

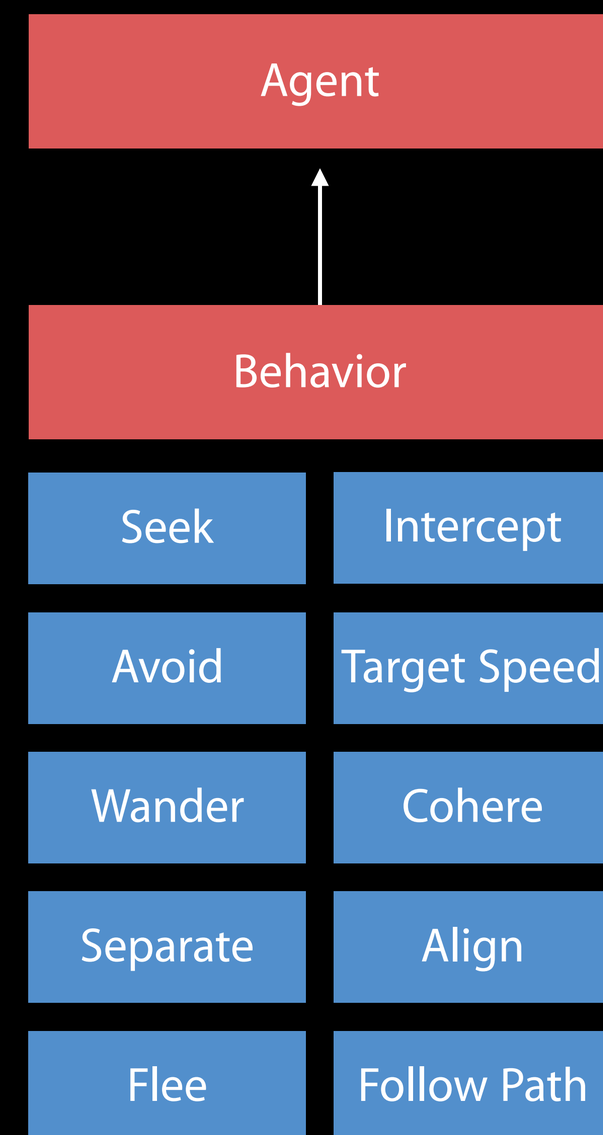
Refresher on agents

Autonomous entities controlled by goals

Goals are combined into behaviors

Realistic physical constraints

- Velocity
- Mass
- Obstacle avoidance
- Path following



Agents

Agents in 3D

NEW

GKAgent3D vs. GKAgent2D

- All goals supported
- Rotation is via matrix_float3x3

GKPath supports both 2D and 3D points

Obstacles constrained to single plane

GKAgent3D
vector_float3 position matrix_float3x3 rotation

GKPath
GKPath(float3Points:…) float3(at:)

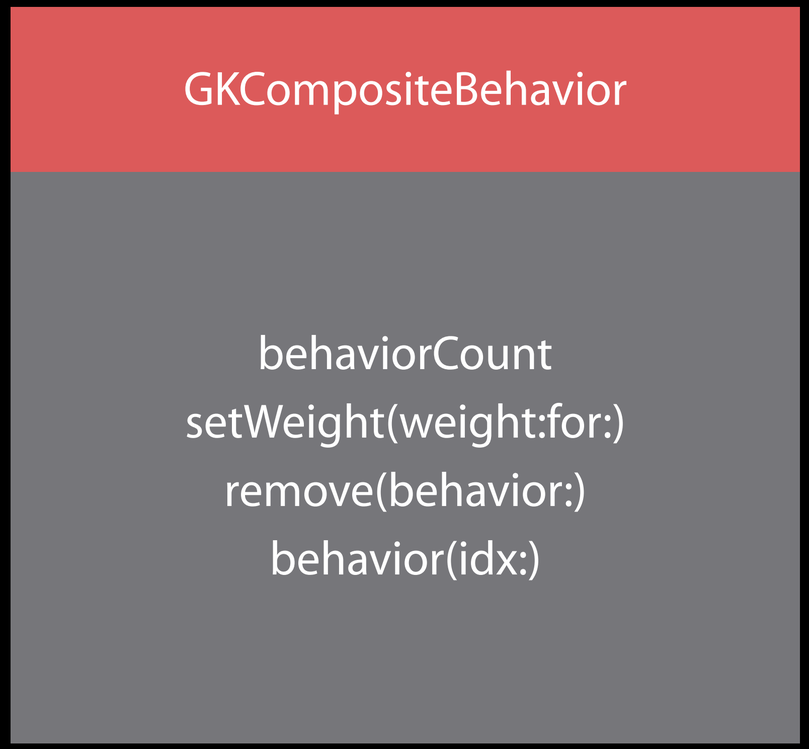
Agents

Behavior composition

NEW

GKCompositeBehavior

- Subclass of GKBehavior
- Collection of weighted GKBehaviors
- Same relationship as GKBehavior –> GKGoal
- Fully nestable



```swift
// Agents
// Composite behavior example

// Create a flocking behavior
let flocking = GKBehavior(goals: [align, cohere, separate])

// Create an avoid behavior
let avoid = GKBehavior(goals: [avoidObstacles, avoidEnemies])

// Make a composite behavior out of the flocking and avoid behaviors
let compBehavior = GKCompositeBehavior(behaviors: [flocking,avoid])

// Create an agent and set our composite behavior on it
let agent = GKAgent2D()
agent.behavior = compBehavior
```

```swift
// Agents
// Composite behavior example

// Create a flocking behavior
let flocking = GKBehavior(goals: [align, cohere, separate])

// Create an avoid behavior
let avoid = GKBehavior(goals: [avoidObstacles, avoidEnemies])

// Make a composite behavior out of the flocking and avoid behaviors
let compBehavior = GKCompositeBehavior(behaviors: [flocking,avoid])

// Create an agent and set our composite behavior on it
let agent = GKAgent2D()
agent.behavior = compBehavior
```

```swift
// Agents
// Composite behavior example

// Create a flocking behavior
let flocking = GKBehavior(goals: [align, cohere, separate])

// Create an avoid behavior
let avoid = GKBehavior(goals: [avoidObstacles, avoidEnemies])

// Make a composite behavior out of the flocking and avoid behaviors
let compBehavior = GKCompositeBehavior(behaviors: [flocking,avoid])

// Create an agent and set our composite behavior on it
let agent = GKAgent2D()
agent.behavior = compBehavior
```

```swift
// Agents
// Composite behavior example

// Create a flocking behavior
let flocking = GKBehavior(goals: [align, cohere, separate])

// Create an avoid behavior
let avoid = GKBehavior(goals: [avoidObstacles, avoidEnemies])

// Make a composite behavior out of the flocking and avoid behaviors
let compBehavior = GKCompositeBehavior(behaviors: [flocking,avoid])

// Create an agent and set our composite behavior on it
let agent = GKAgent2D()
agent.behavior = compBehavior
```

```
// Agents
// Composite behavior example

// Create a flocking behavior
let flocking = GKBehavior(goals: [align, cohere, separate])

// Create an avoid behavior
let avoid = GKBehavior(goals: [avoidObstacles, avoidEnemies])

// Make a composite behavior out of the flocking and avoid behaviors
let compBehavior = GKCompositeBehavior(behaviors: [flocking,avoid])
```

```
// Create an agent and set our composite behavior on it
let agent = GKAgent2D()
agent.behavior = compBehavior
```

Spatial Partitioning

Spatial Partitioning

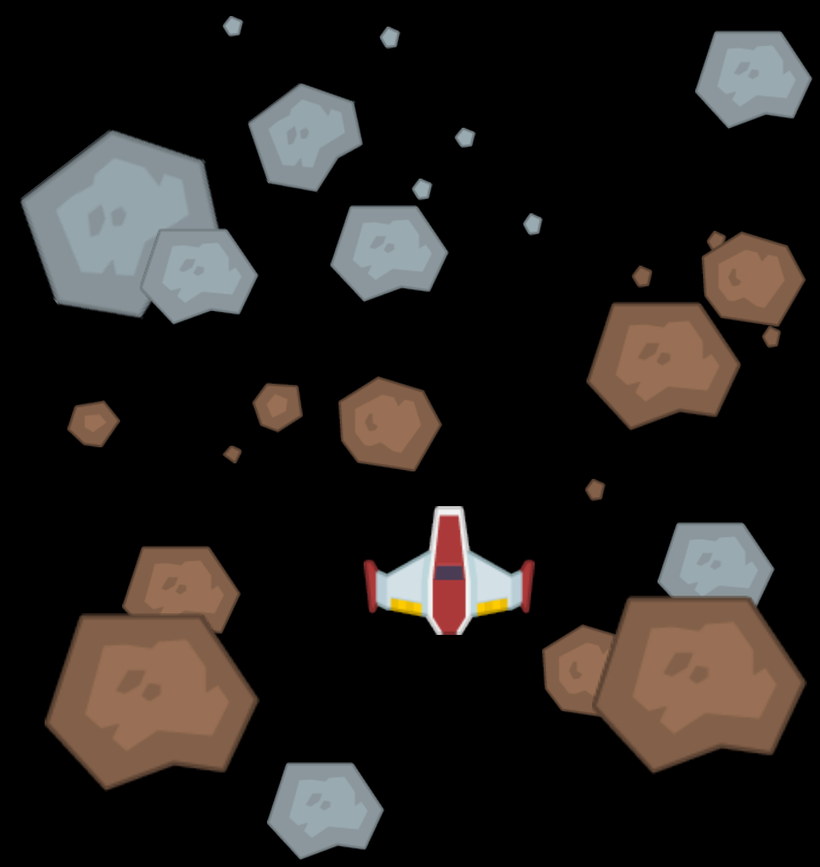
Background

We ask questions about our game world

- How many enemies are near the player?
- Where are all items in the my world?
- Which projectiles will hit player this frame?

Answers can be expensive

We can speed these queries up by caching



Spatial Partitioning

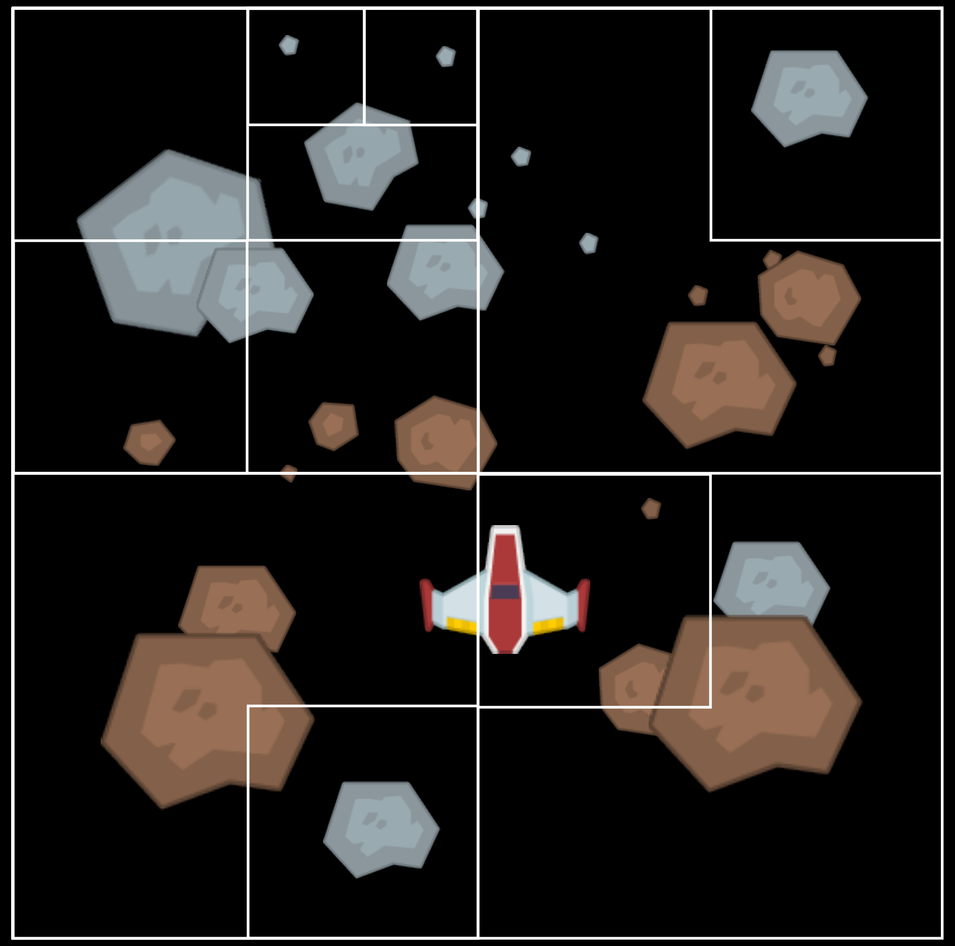
Overview

Cache game objects spatially

- Objects get grouped into hierarchies
- Queries are made faster

Tree data structures

- R-Tree
- Quadtree and Octtree



Spatial Partitioning

GKRTree

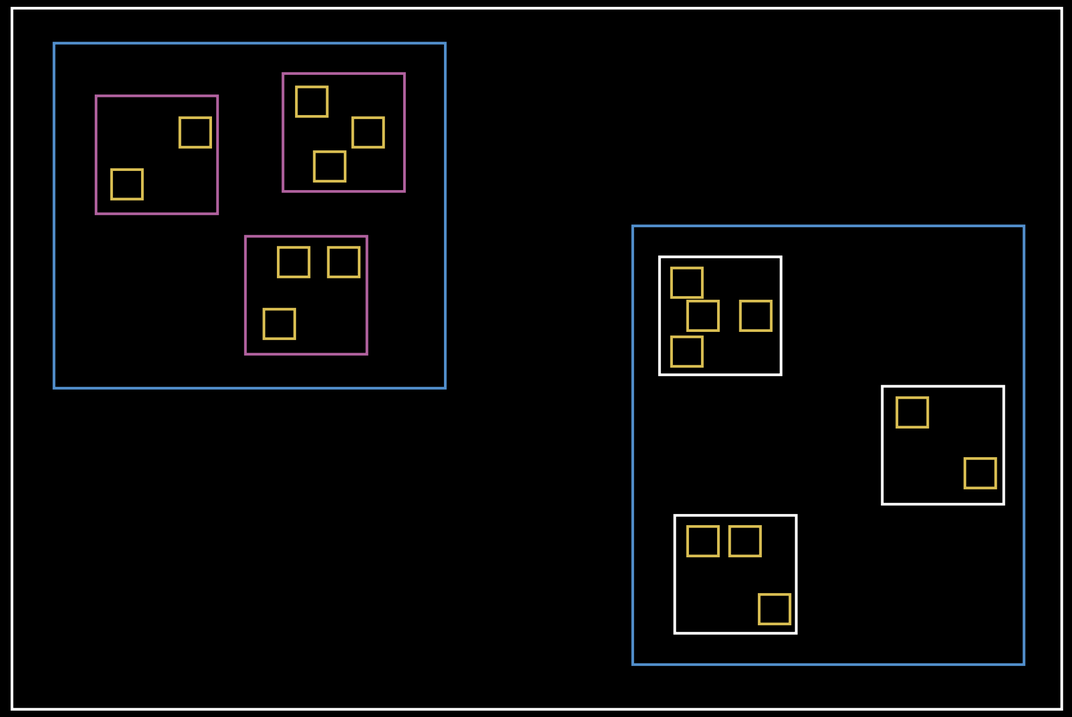
NEW

Tree data structure with "buckets"

- Objects are assigned to a bucket
- Bounding box is sum of all children

Boxes split when they grow too big

- Various strategies on how to split
- Halve, linear, quadratic, overlap reduction



Spatial Partitioning

GKRTree

NEW

Tree data structure with "buckets"

- Objects are assigned to a bucket
- Bounding box is sum of all children

Boxes split when they grow too big

- Various strategies on how to split
- Halve, linear, quadratic, overlap reduction

Spatial Partitioning

GKRTree

NEW

Tree data structure with "buckets"

- Objects are assigned to a bucket
- Bounding box is sum of all children

Boxes split when they grow too big

- Various strategies on how to split
- Halve, linear, quadratic, overlap reduction

Spatial Partitioning

GKRTree

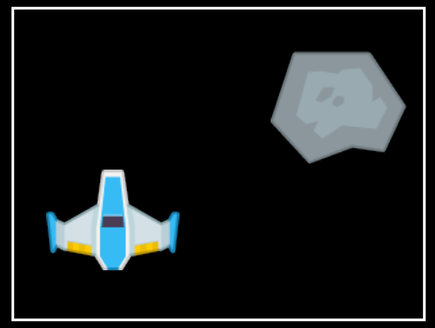
NEW

Tree data structure with "buckets"

- Objects are assigned to a bucket
- Bounding box is sum of all children

Boxes split when they grow too big

- Various strategies on how to split
- Halve, linear, quadratic, overlap reduction



Spatial Partitioning

GKRTree

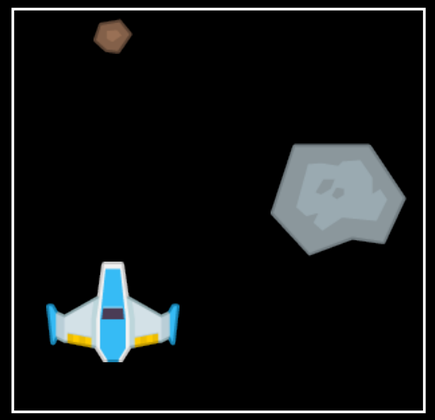
NEW

Tree data structure with "buckets"

- Objects are assigned to a bucket
- Bounding box is sum of all children

Boxes split when they grow too big

- Various strategies on how to split
- Halve, linear, quadratic, overlap reduction



Spatial Partitioning

GKRTree

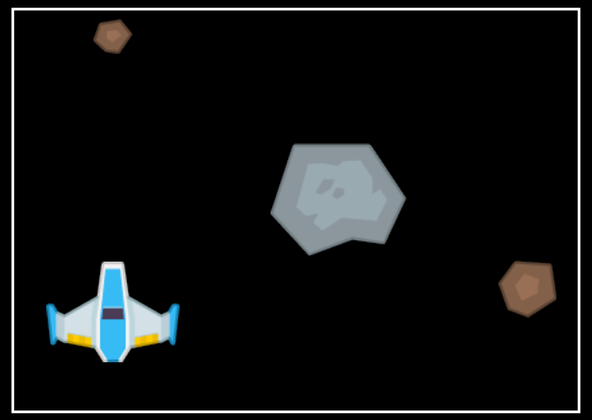
NEW

Tree data structure with "buckets"

- Objects are assigned to a bucket
- Bounding box is sum of all children

Boxes split when they grow too big

- Various strategies on how to split
- Halve, linear, quadratic, overlap reduction



Spatial Partitioning

GKRTree

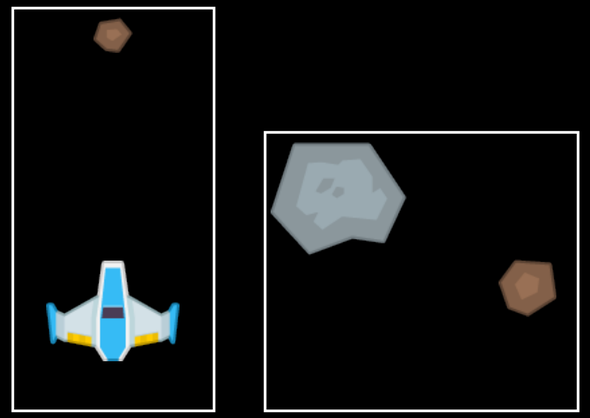
NEW

Tree data structure with "buckets"

- Objects are assigned to a bucket
- Bounding box is sum of all children

Boxes split when they grow too big

- Various strategies on how to split
- Halve, linear, quadratic, overlap reduction



Spatial Partitioning

GKRTree

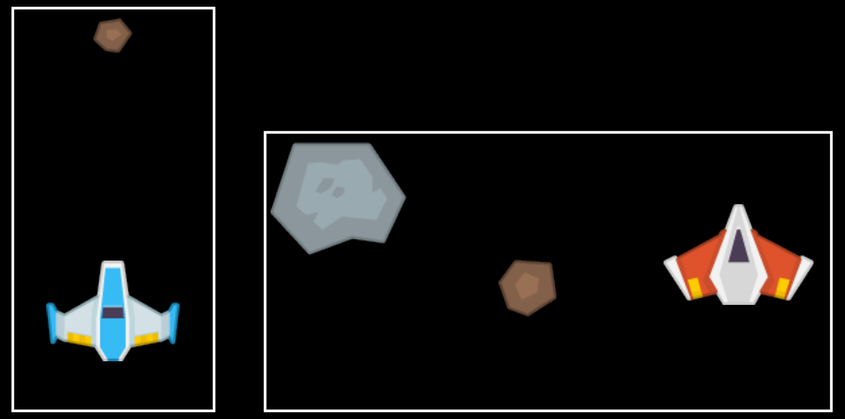
NEW

Tree data structure with "buckets"

- Objects are assigned to a bucket
- Bounding box is sum of all children

Boxes split when they grow too big

- Various strategies on how to split
- Halve, linear, quadratic, overlap reduction



Spatial Partitioning

GKRTree

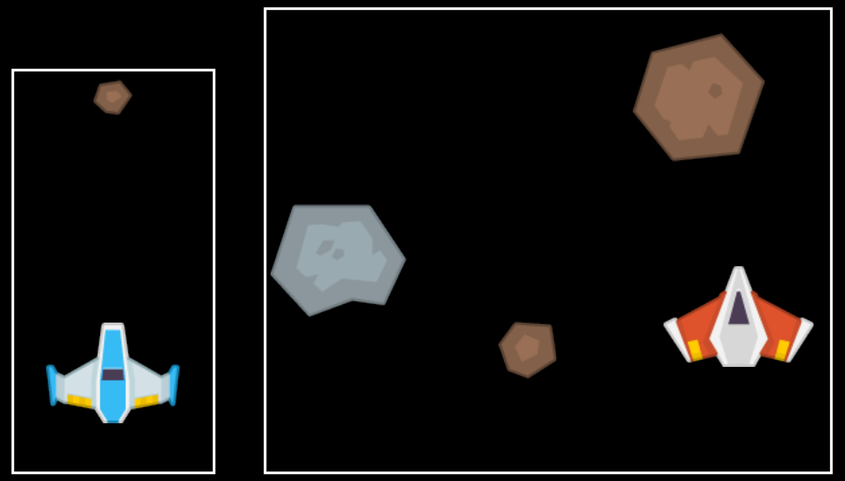
NEW

Tree data structure with "buckets"

- Objects are assigned to a bucket
- Bounding box is sum of all children

Boxes split when they grow too big

- Various strategies on how to split
- Halve, linear, quadratic, overlap reduction



Spatial Partitioning

GKRTree

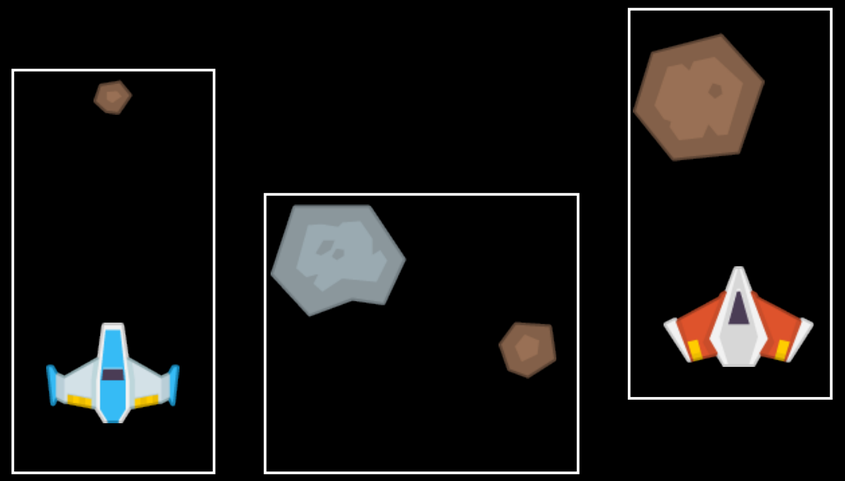
NEW

Tree data structure with "buckets"

- Objects are assigned to a bucket
- Bounding box is sum of all children

Boxes split when they grow too big

- Various strategies on how to split
- Halve, linear, quadratic, overlap reduction



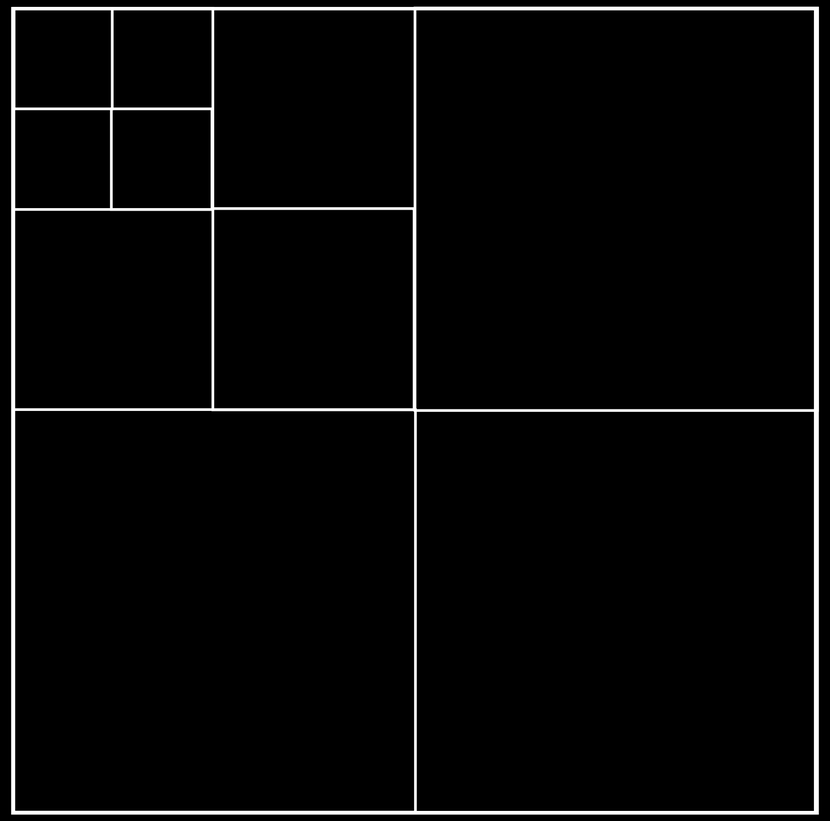
Spatial Partitioning

GKQuadtree and GKOctree

NEW

Tree data structure with levels

- Space is evenly divided at each level
- Max cell size controls depth
- Objects placed into smallest cell they fit in
- Cell size should make sense for your game



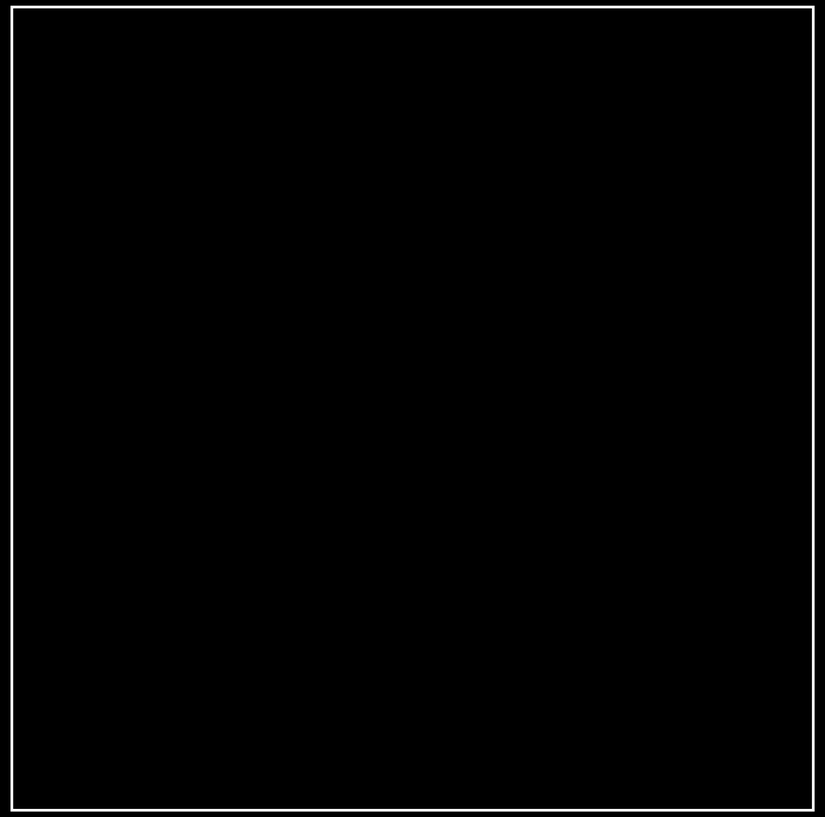
Spatial Partitioning

GKQuadtree and GKOctree

NEW

Tree data structure with levels

- Space is evenly divided at each level
- Max cell size controls depth
- Objects placed into smallest cell they fit in
- Cell size should make sense for your game



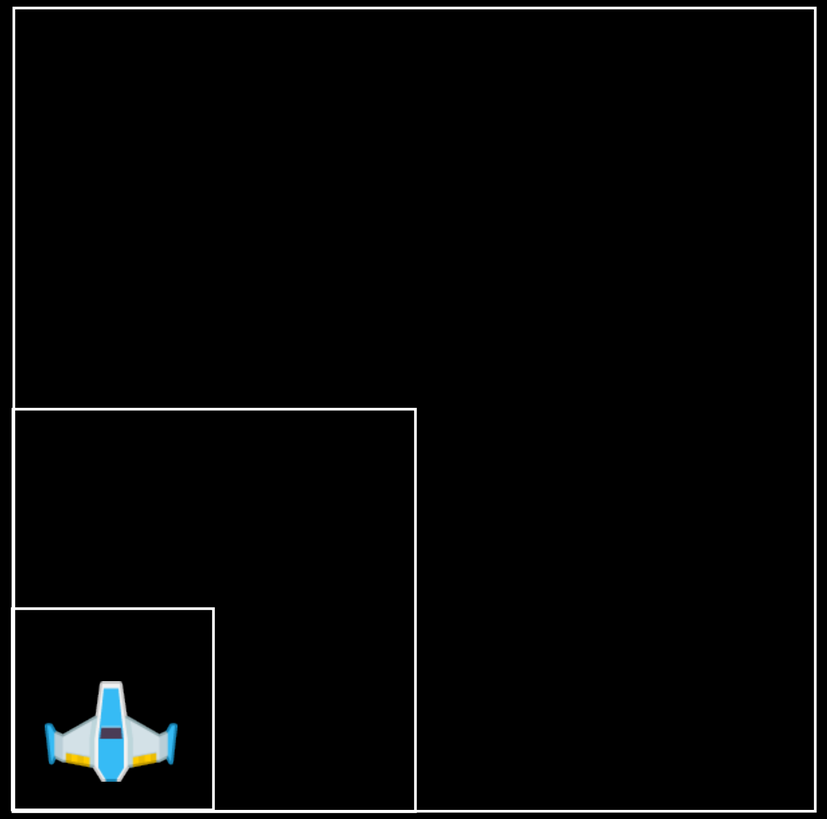
Spatial Partitioning

GKQuadtree and GKOctree

NEW

Tree data structure with levels

- Space is evenly divided at each level
- Max cell size controls depth
- Objects placed into smallest cell they fit in
- Cell size should make sense for your game



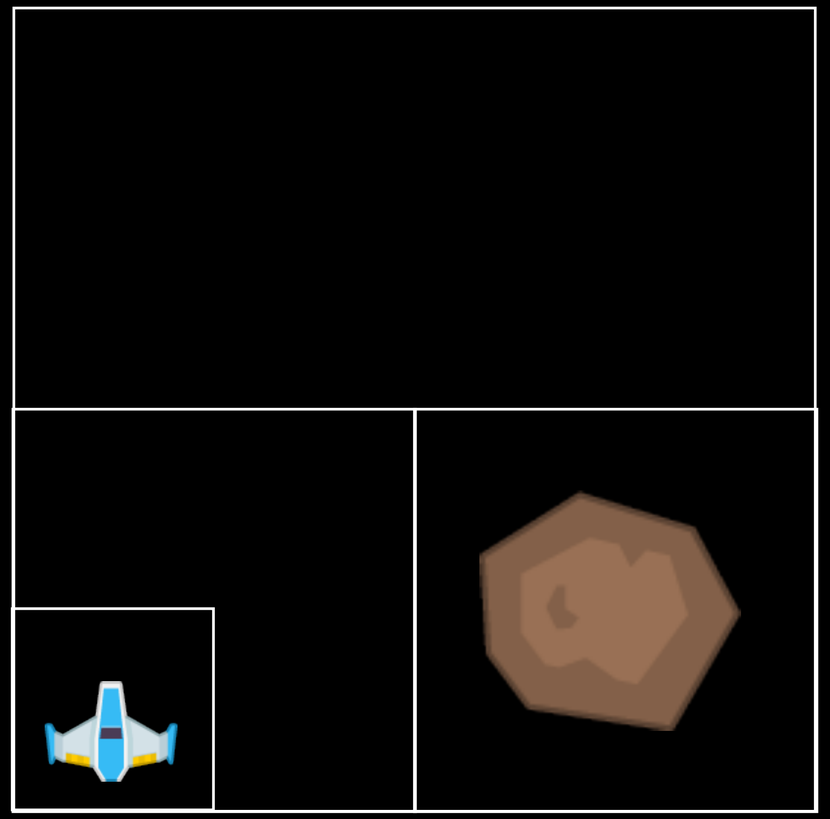
Spatial Partitioning

GKQuadtree and GKOctree

NEW

Tree data structure with levels

- Space is evenly divided at each level
- Max cell size controls depth
- Objects placed into smallest cell they fit in
- Cell size should make sense for your game



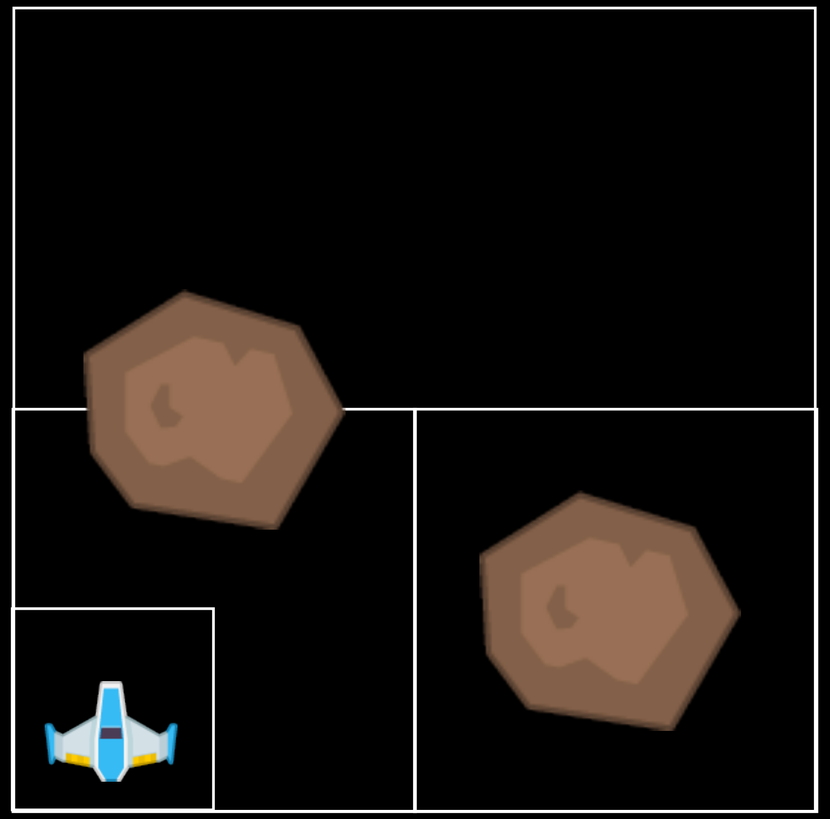
Spatial Partitioning

GKQuadtree and GKOctree

NEW

Tree data structure with levels

- Space is evenly divided at each level
- Max cell size controls depth
- Objects placed into smallest cell they fit in
- Cell size should make sense for your game



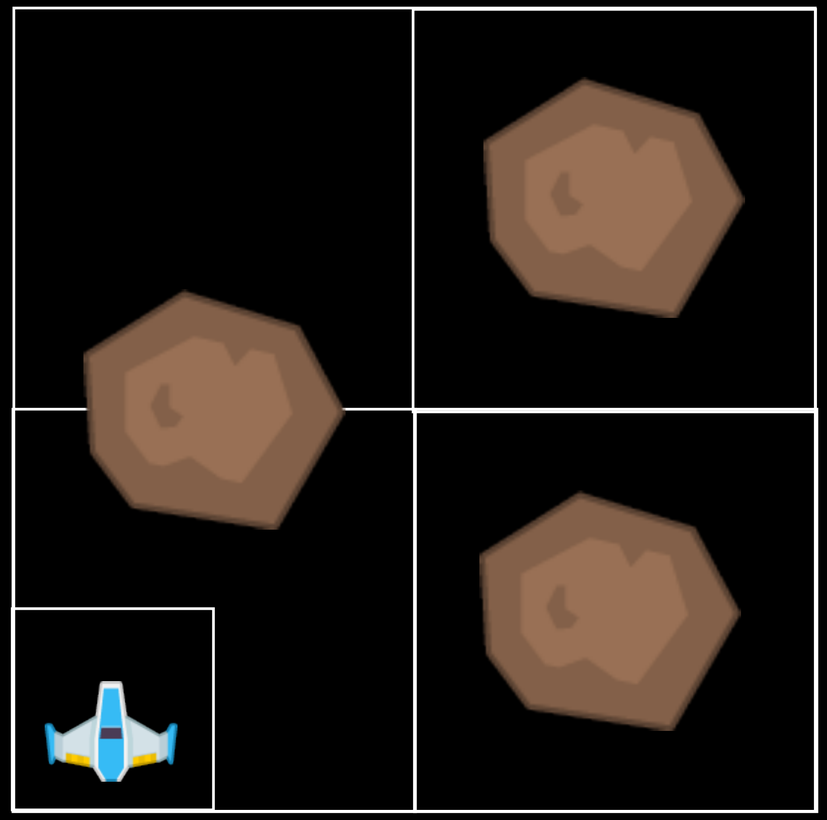
Spatial Partitioning

GKQuadtree and GKOctree

NEW

Tree data structure with levels

- Space is evenly divided at each level
- Max cell size controls depth
- Objects placed into smallest cell they fit in
- Cell size should make sense for your game



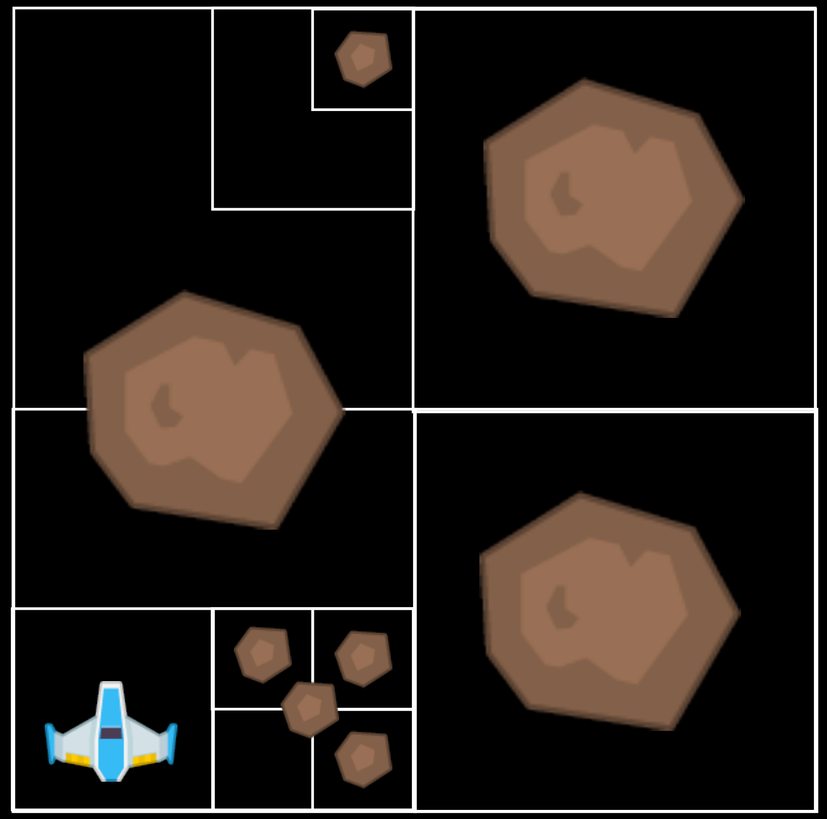
Spatial Partitioning

GKQuadtree and GKOctree

NEW

Tree data structure with levels

- Space is evenly divided at each level
- Max cell size controls depth
- Objects placed into smallest cell they fit in
- Cell size should make sense for your game



```
// Spatial Partitioning
// Quadtree example

// Create a quadtree spanning (0,0) to (1000,1000)
let tree = GKQuadtree(boundingQuad: GKQuad(quadMin: float2(0.0,0.0),
                                           quadMax: float2(1000.0,1000.0), minimumCellSize:100)!

// Add some enemies to the quadtree
tree.addElement(enemy1, with:GKQuad(quadMin:float2(40.0,40.0), quadMax:float2(50.0,50.0)))
tree.addElement(enemy2, with:GKQuad(quadMin:float2(10.0,20.0), quadMax:float2(20.0,30.0)))
tree.addElement(enemy3, with:GKQuad(quadMin:float2(0.0,0.0), quadMax:float2(10.0,10.0)))

// Query all the enemies in (0,0) to (100,100)
tree.elements(in: GKQuad(quadMin:float2(0.0,0.0), quadMax:float2(100.0,100.0)))
```

```
// Spatial Partitioning
// Quadtree example

// Create a quadtree spanning (0,0) to (1000,1000)
let tree = GKQuadtree(boundingQuad: GKQuad(quadMin: float2(0.0,0.0),
                                           quadMax: float2(1000.0,1000.0), minimumCellSize:100)!

// Add some enemies to the quadtree
tree.addElement(enemy1, with:GKQuad(quadMin:float2(40.0,40.0), quadMax:float2(50.0,50.0)))
tree.addElement(enemy2, with:GKQuad(quadMin:float2(10.0,20.0), quadMax:float2(20.0,30.0)))
tree.addElement(enemy3, with:GKQuad(quadMin:float2(0.0,0.0), quadMax:float2(10.0,10.0)))

// Query all the enemies in (0,0) to (100,100)
tree.elements(in: GKQuad(quadMin:float2(0.0,0.0), quadMax:float2(100.0,100.0)))
```

```
// Spatial Partitioning
// Quadtree example

// Create a quadtree spanning (0,0) to (1000,1000)
let tree = GKQuadtree(boundingQuad: GKQuad(quadMin: float2(0.0,0.0),
                                           quadMax: float2(1000.0,1000.0), minimumCellSize:100)!
```

```
// Add some enemies to the quadtree
tree.addElement(enemy1, with:GKQuad(quadMin:float2(40.0,40.0), quadMax:float2(50.0,50.0)))
tree.addElement(enemy2, with:GKQuad(quadMin:float2(10.0,20.0), quadMax:float2(20.0,30.0)))
tree.addElement(enemy3, with:GKQuad(quadMin:float2(0.0,0.0), quadMax:float2(10.0,10.0)))
```

```
// Query all the enemies in (0,0) to (100,100)
tree.elements(in: GKQuad(quadMin:float2(0.0,0.0), quadMax:float2(100.0,100.0)))
```

```
// Spatial Partitioning
// Quadtree example

// Create a quadtree spanning (0,0) to (1000,1000)
let tree = GKQuadtree(boundingQuad: GKQuad(quadMin: float2(0.0,0.0),
                                           quadMax: float2(1000.0,1000.0), minimumCellSize:100)!

// Add some enemies to the quadtree
tree.addElement(enemy1, with:GKQuad(quadMin:float2(40.0,40.0), quadMax:float2(50.0,50.0)))
tree.addElement(enemy2, with:GKQuad(quadMin:float2(10.0,20.0), quadMax:float2(20.0,30.0)))
tree.addElement(enemy3, with:GKQuad(quadMin:float2(0.0,0.0), quadMax:float2(10.0,10.0)))
```

```
// Query all the enemies in (0,0) to (100,100)
tree.elements(in: GKQuad(quadMin:float2(0.0,0.0), quadMax:float2(100.0,100.0)))
```

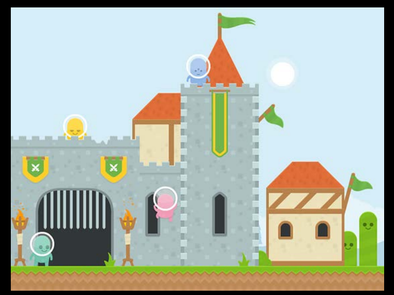
Procedural Generation

Procedural Generation

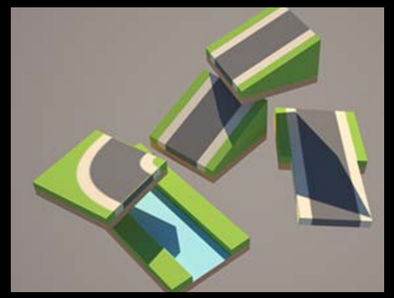
Background

Premade Content

Levels



Textures

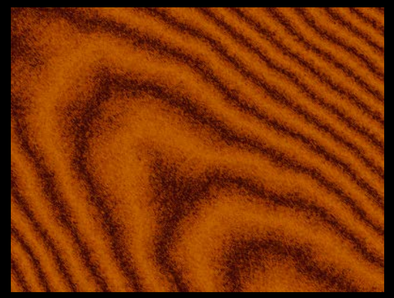


Characters

VS

Procedural Content

Worlds



Textures

Heightmaps

Procedural Generation

Procedural content in games

Need a source of "coherent" randomness

RNGs can be TOO random for content

- Values fluctuate wildly
- Difficult to have spatial relationships
- Challenging to have determinism

Procedural Generation

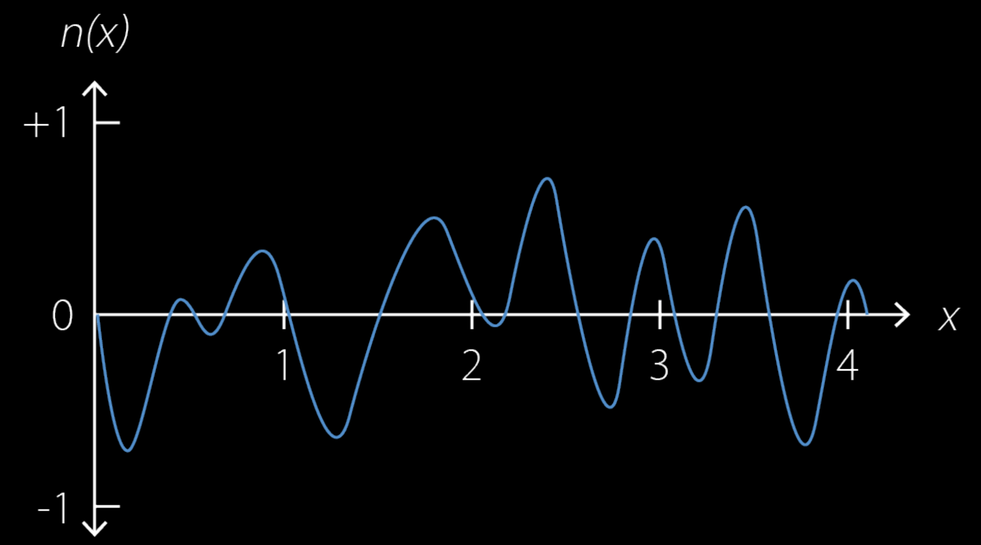
Noise theory

Noise is coherent randomness

- Small input change = small output change
- Big input change = random output change
- Infinite and deterministic

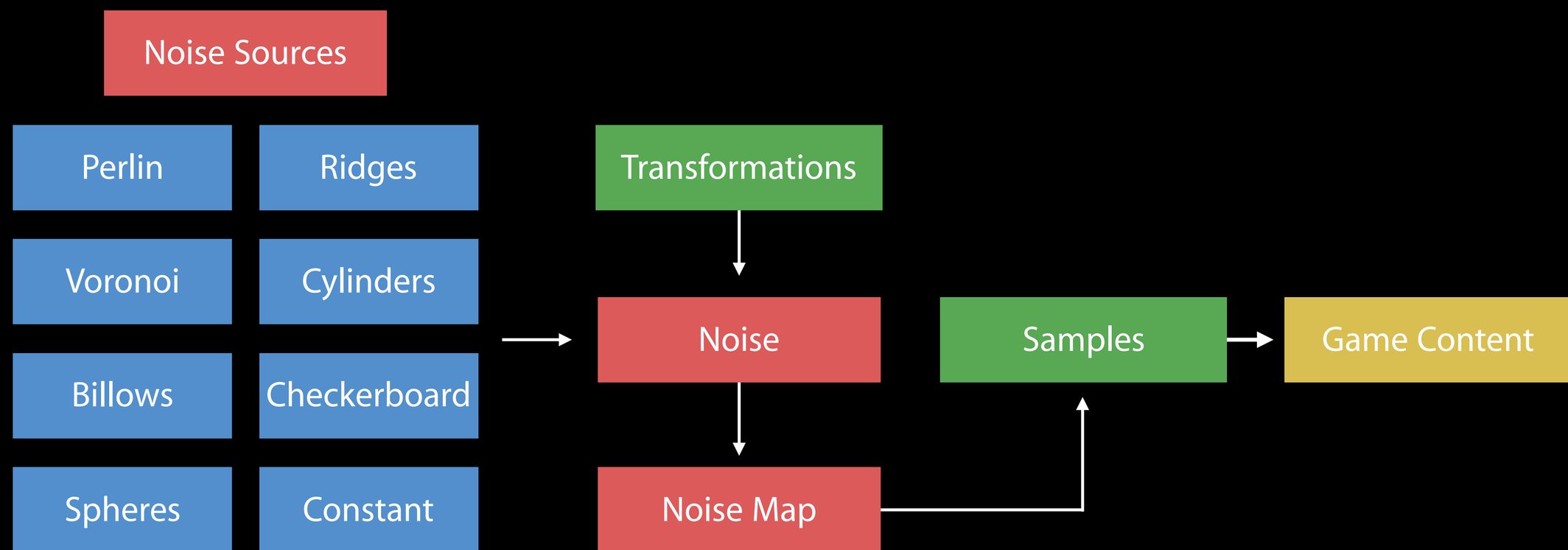
Sample at relevant intervals for your game

- Coordinates, tiles, biomes, texels and so on



Procedural Generation

Overview



Procedural Generation

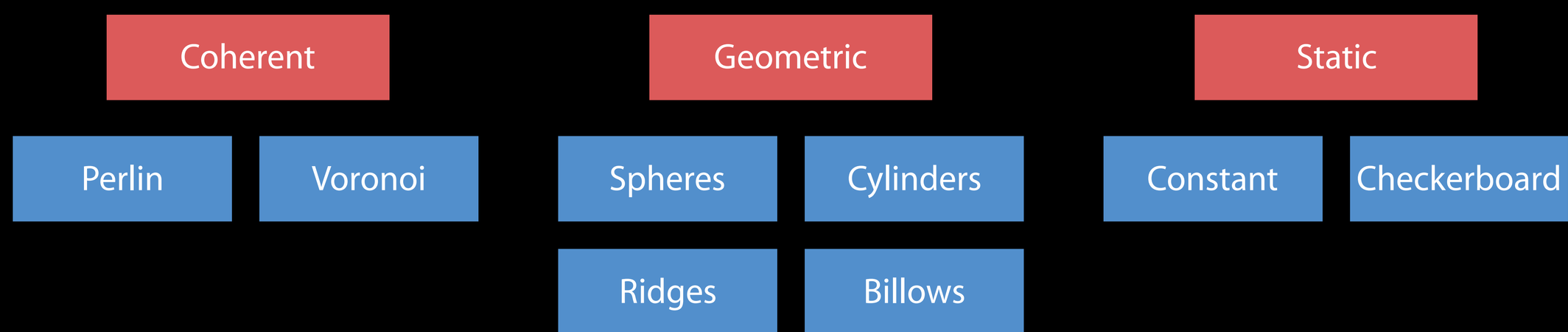
GKNoiseSource

NEW

Output are values between -1.0 and 1.0

Parameters to tweak noise output

- Coherent noise is seeded
- Geometric parameters alter shapes



Procedural Generation

GKNoise

NEW

Transform, modify or combine GKNoiseSources

Many common operations supported

Combiners

- Add
- Multiply
- Min
- Max
- Power
- Displace

Transformers

- Scale
- Rotate
- Translate
- Terrace Map
- Curve Map

Modifiers

- Abs
- Turbulence
- Clamp
- Invert

Procedural Generation

GKNoiseMap

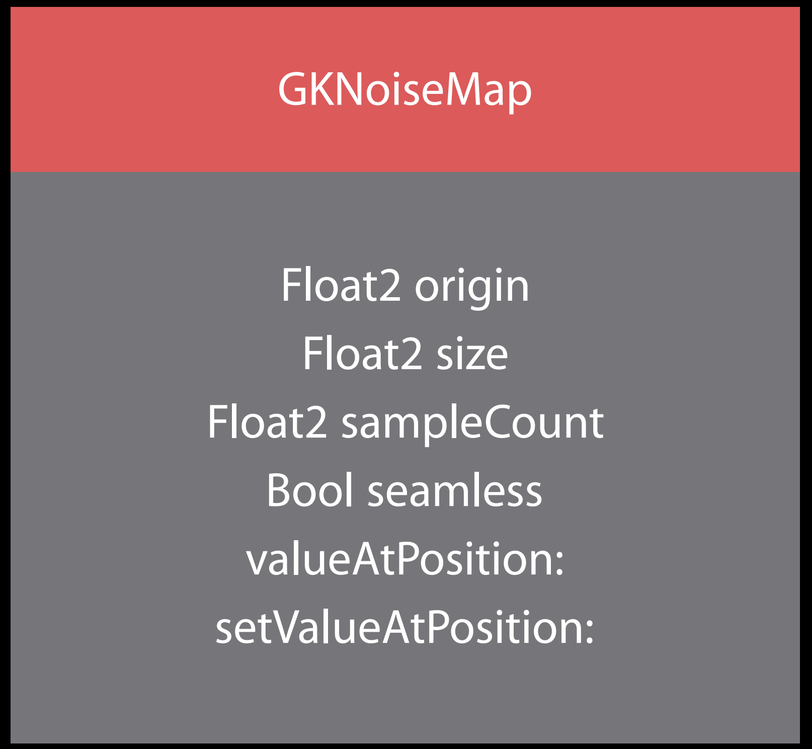
NEW

Samples a region of GKNoise

- Origin and size
- Sample count

Get value at a given position

- Range is [-1.0,1.0] like sources
- Can overwrite values when needed



Procedural Generation

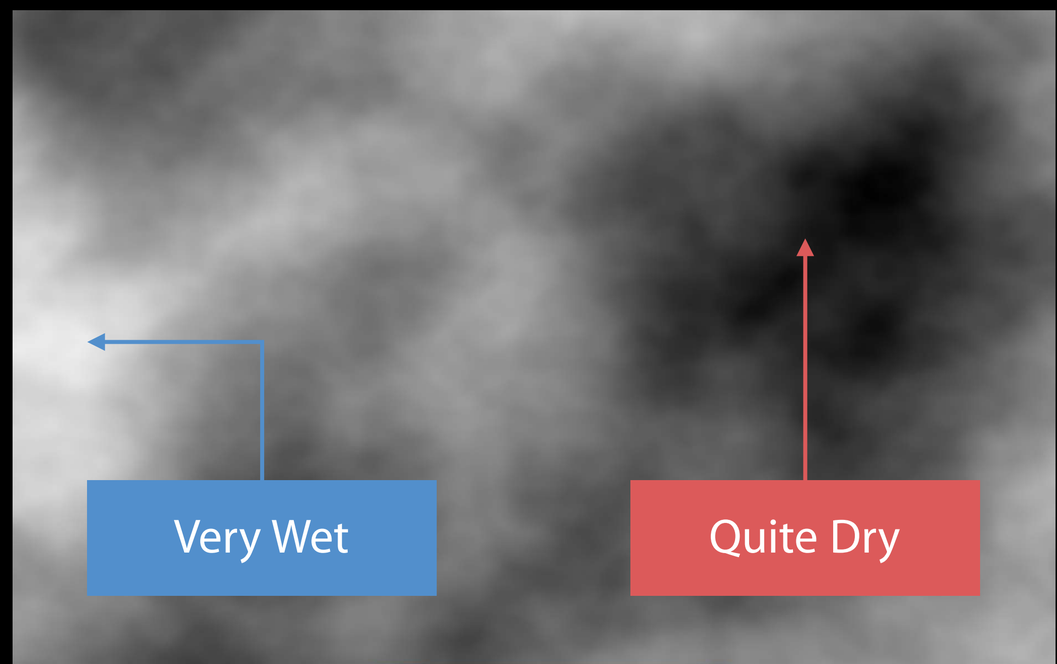
Biome example

Moisture

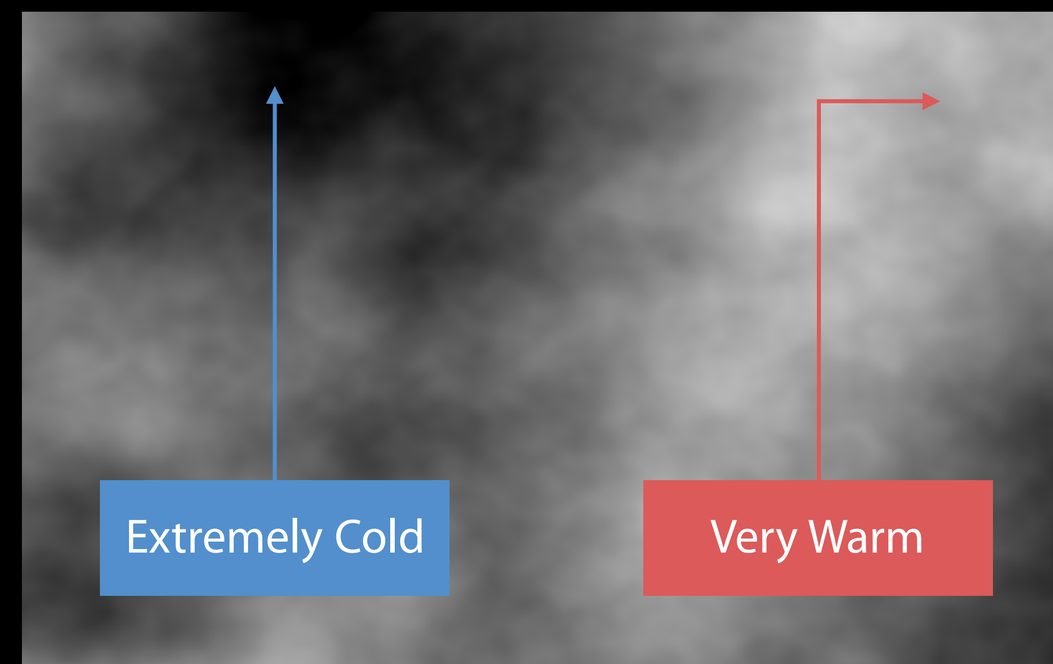
Temperature

Procedural Generation

Biome example



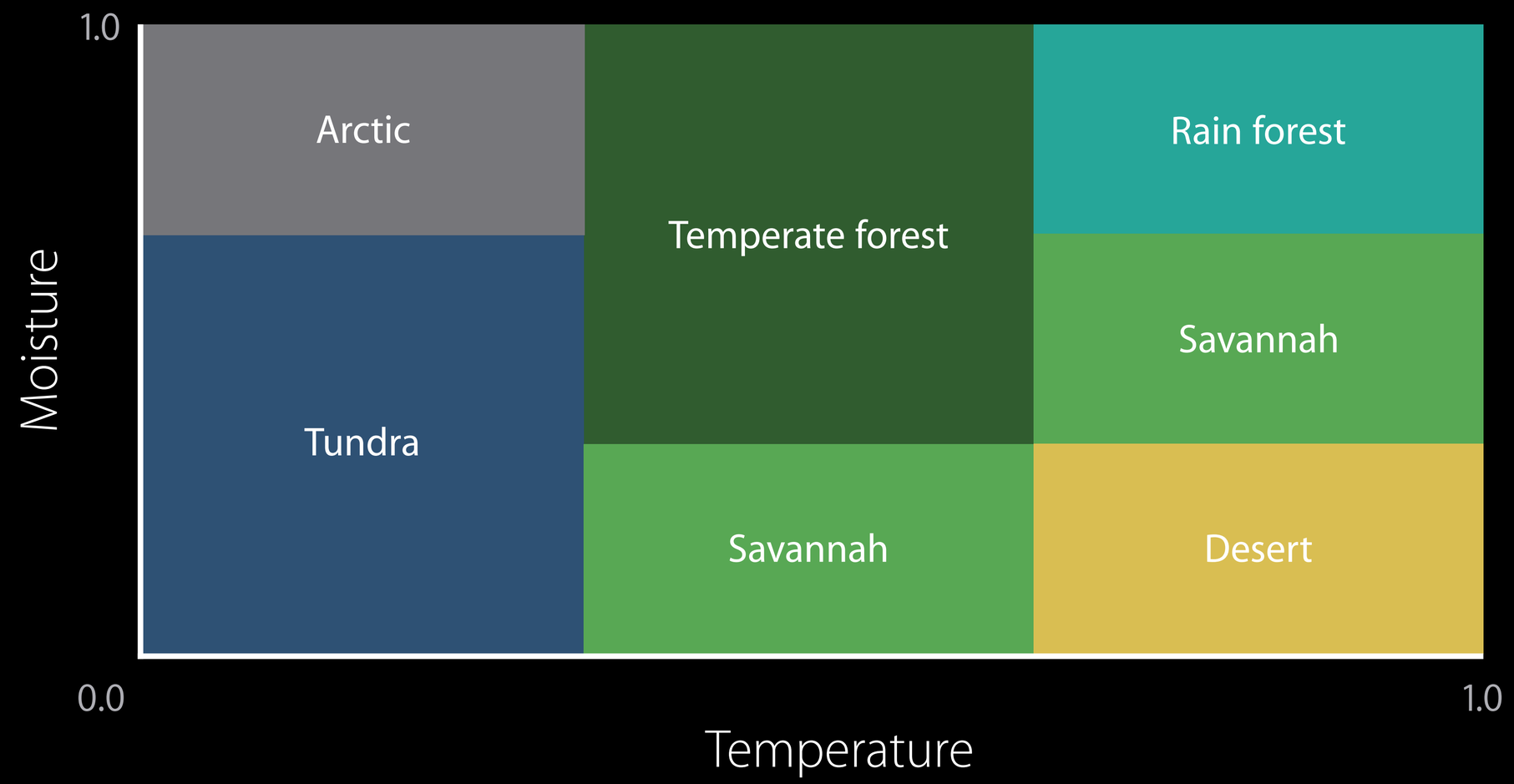
Moisture



Temperature

Procedural Generation

Biome example



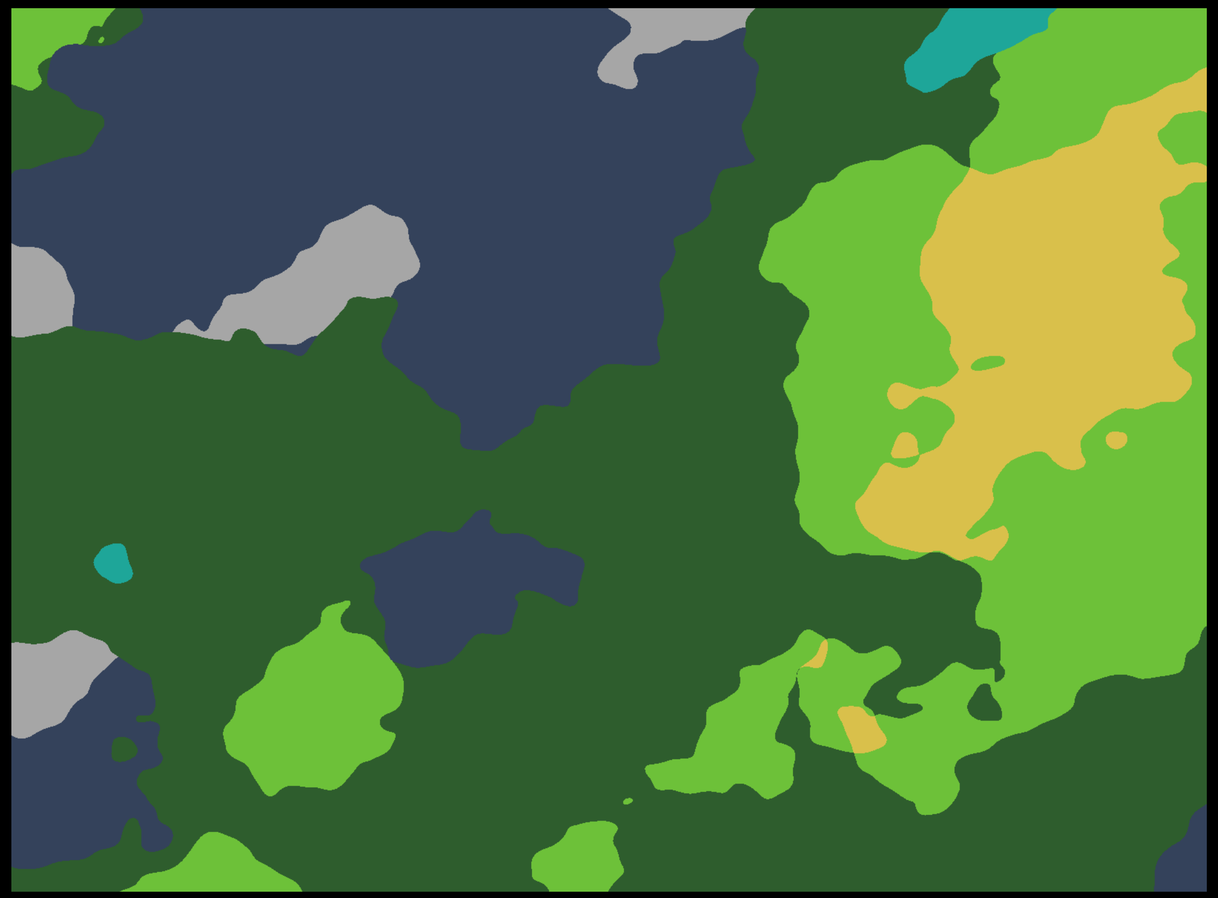
Procedural Generation

Biome example

Moisture

Temperature

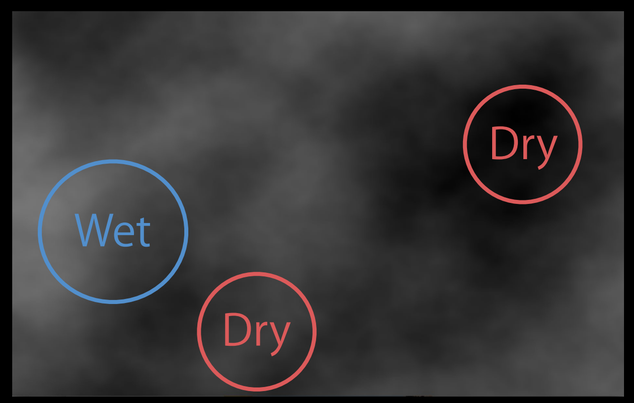
Biomes



Procedural Generation

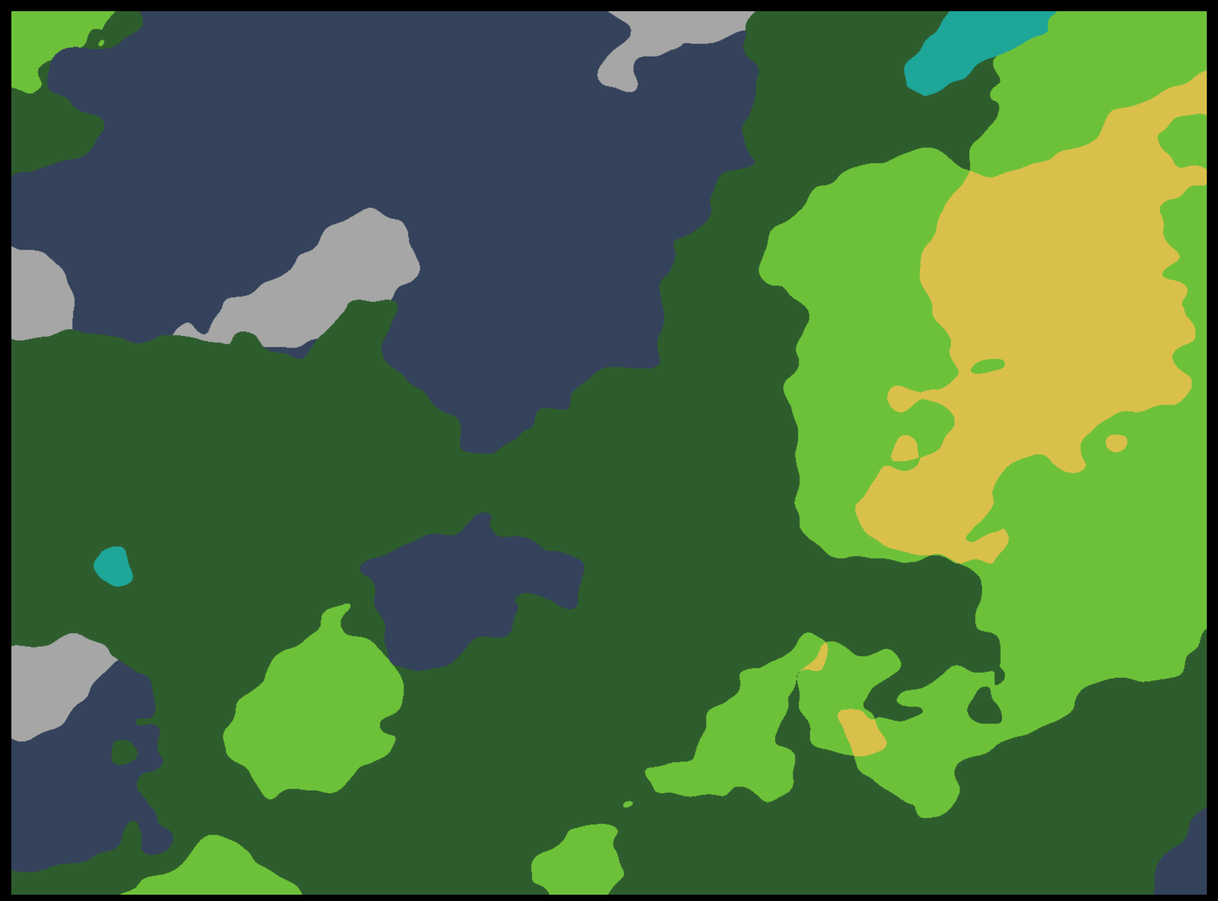
Biome example

Moisture



Temperature

Biomes



Game AI

Michael Brennan Game Technologies Engineer

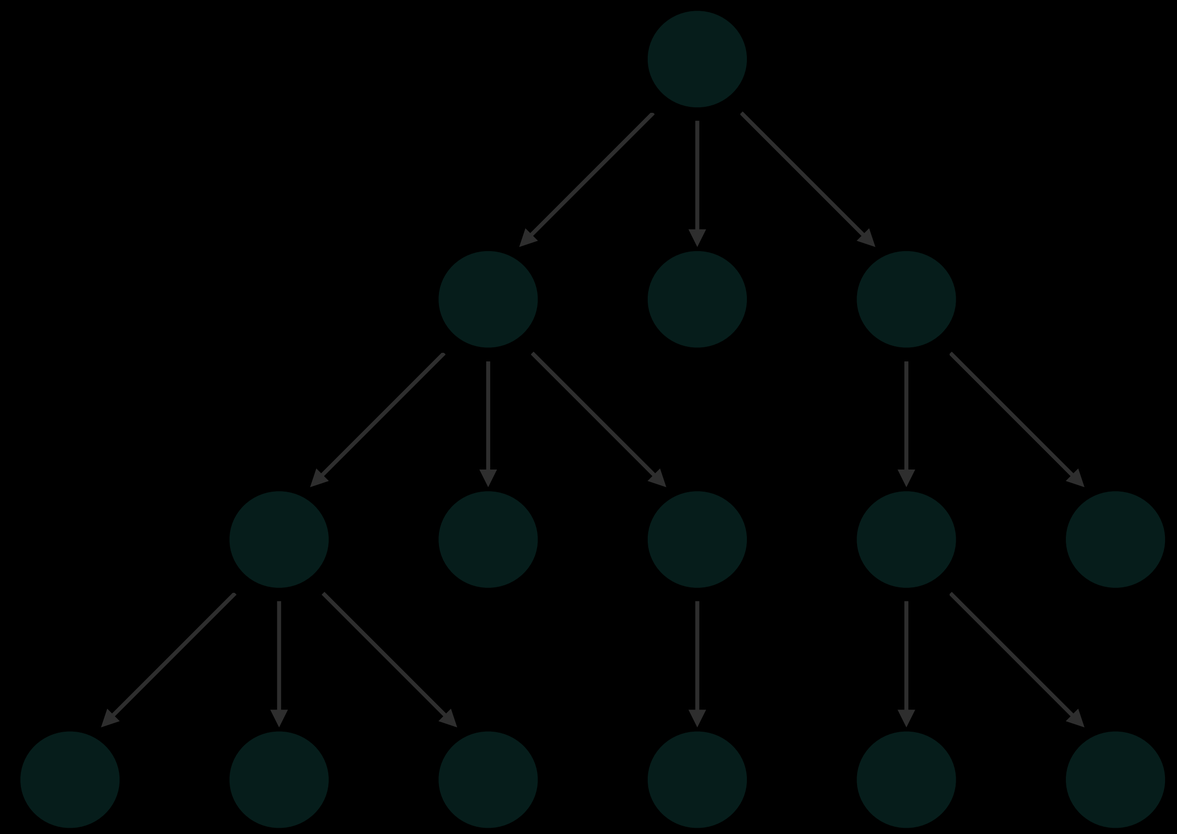
Game AI

Minmax strategist

Finds optimal move

- Exhaustive search
- Requires score for all states

Slower for larger state spaces



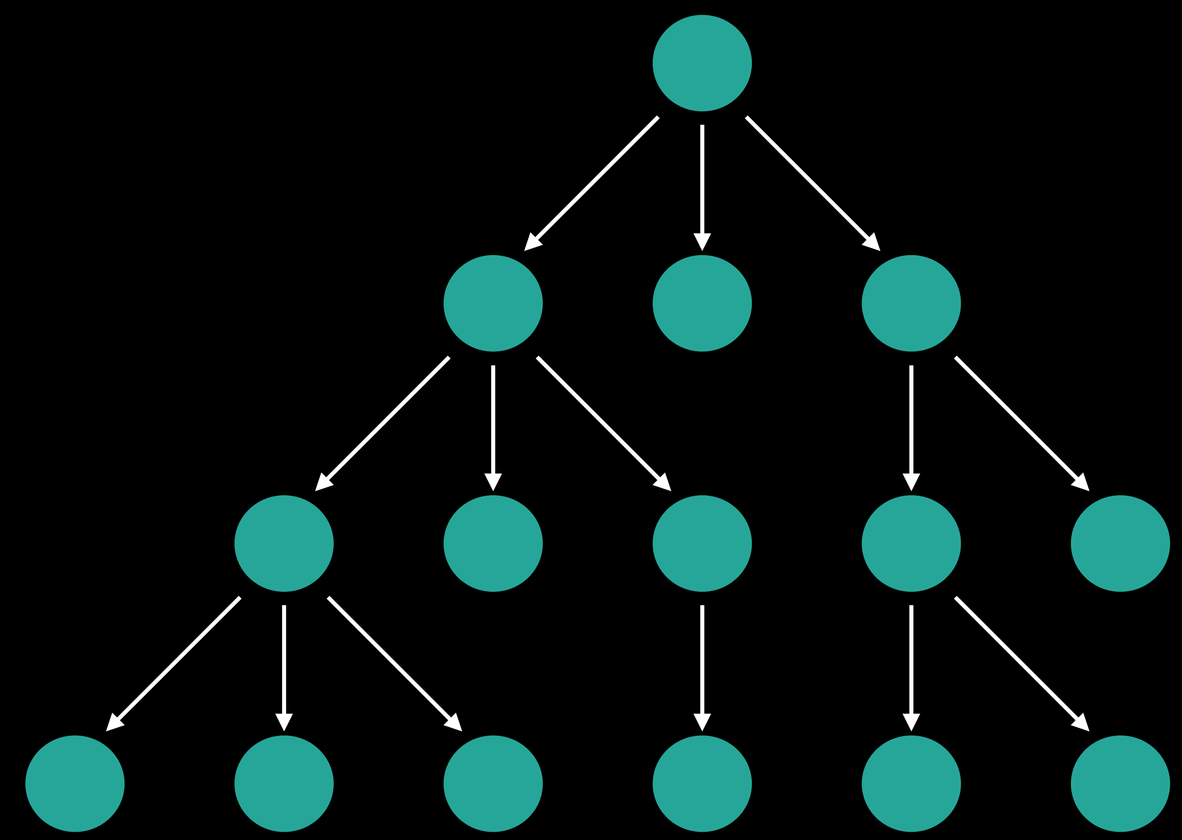
Game AI

Minmax strategist

Finds optimal move

- Exhaustive search
- Requires score for all states

Slower for larger state spaces



Game AI

Monte Carlo strategist

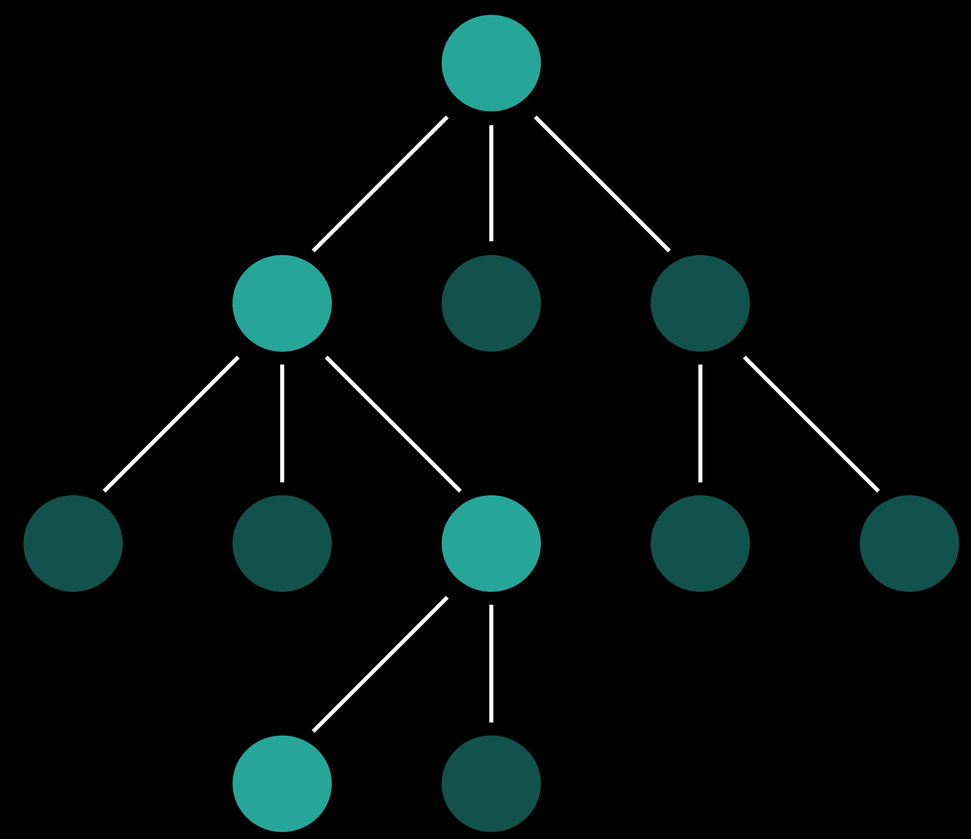
NEW

Best-first search

Random sampling of state space

- Selects player move
- Simulates out new games
- Finds win / loss
- Back-propagates win likelihood

Convergent on optimal move



Game AI

Monte Carlo strategist

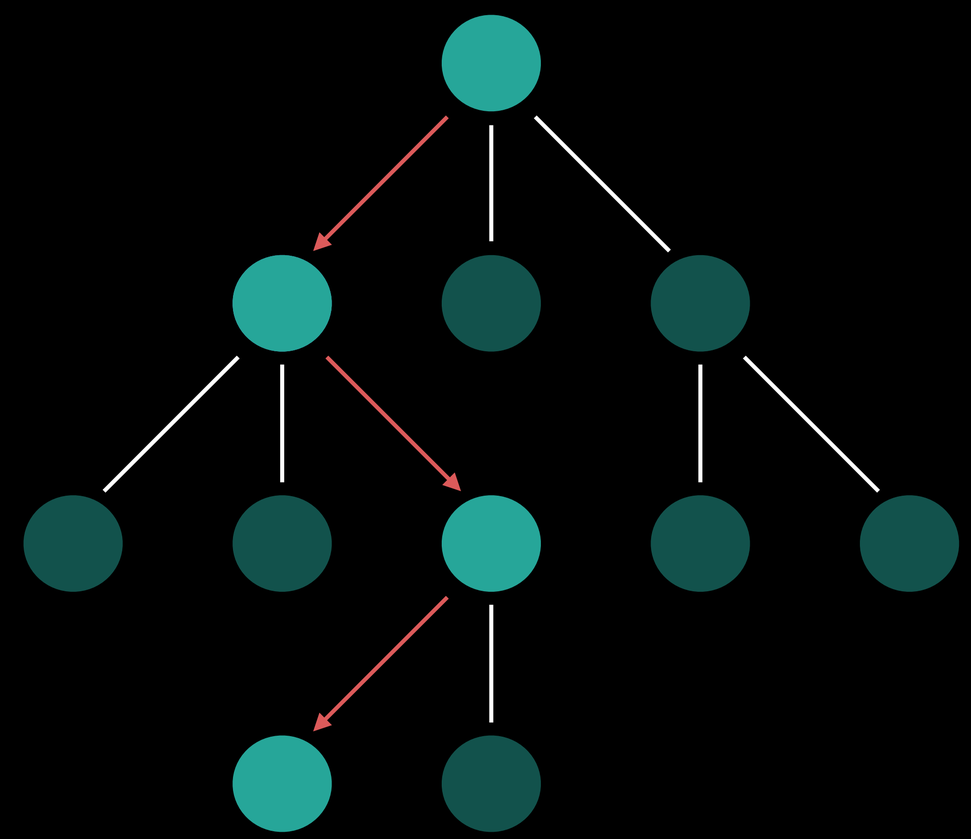
NEW

Best-first search

Random sampling of state space

- Selects player move
- Simulates out new games
- Finds win / loss
- Back-propagates win likelihood

Convergent on optimal move



Game AI

Monte Carlo strategist

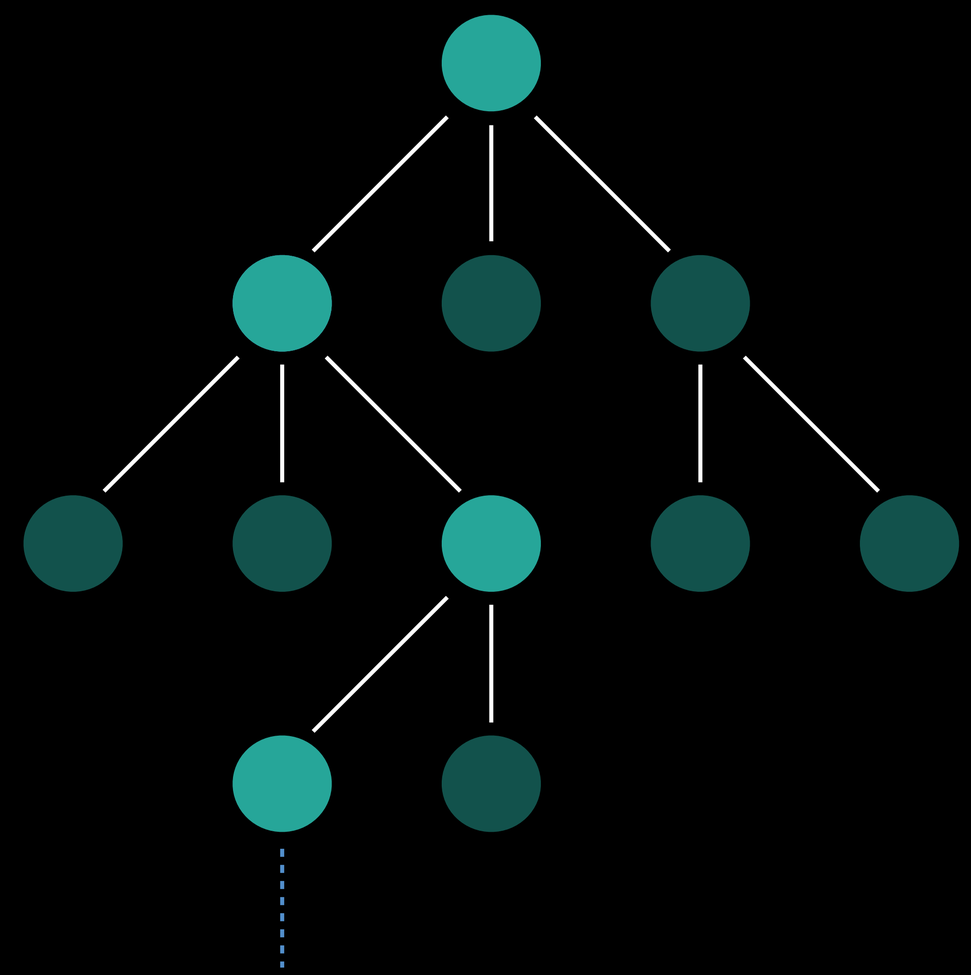
NEW

Best-first search

Random sampling of state space

- Selects player move
- Simulates out new games
- Finds win / loss
- Back-propagates win likelihood

Convergent on optimal move



Game AI

Monte Carlo strategist

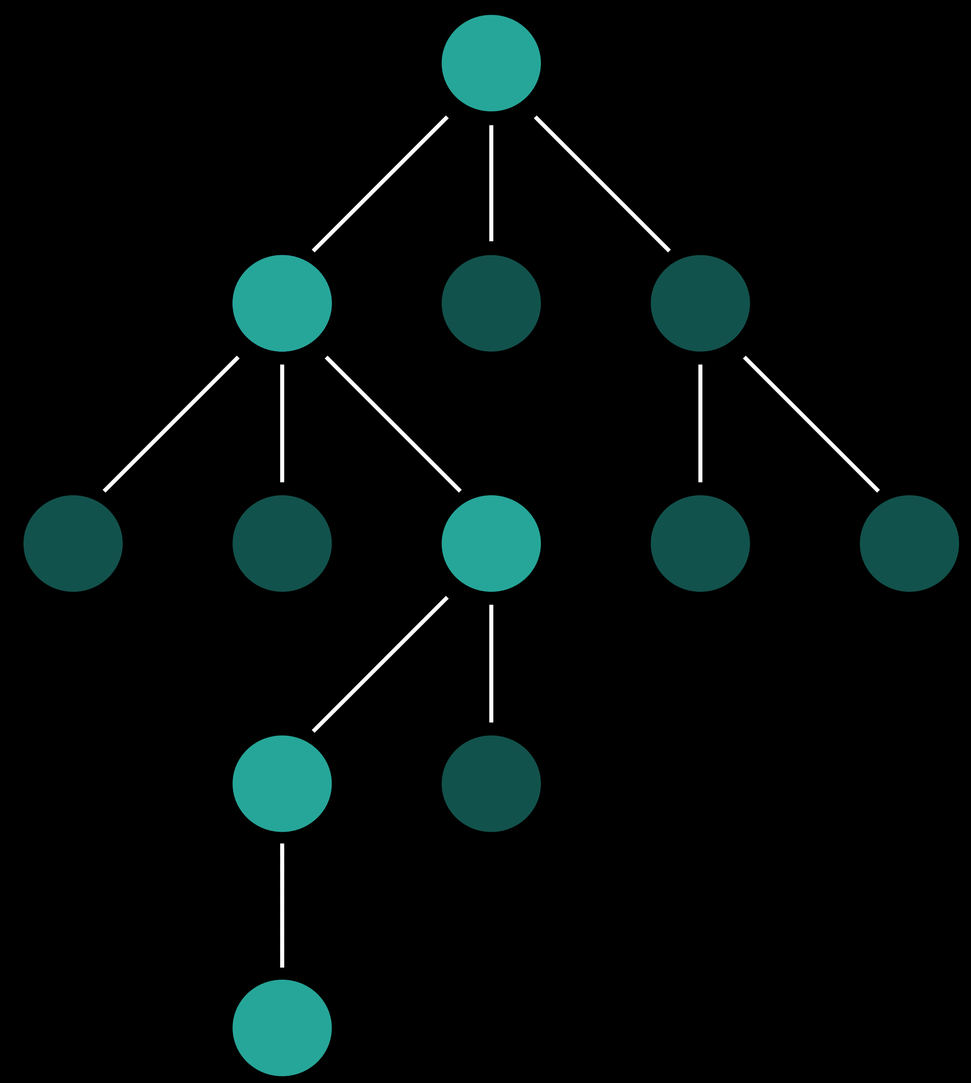
NEW

Best-first search

Random sampling of state space

- Selects player move
- Simulates out new games
- Finds win / loss
- Back-propagates win likelihood

Convergent on optimal move



Game AI

Monte Carlo strategist

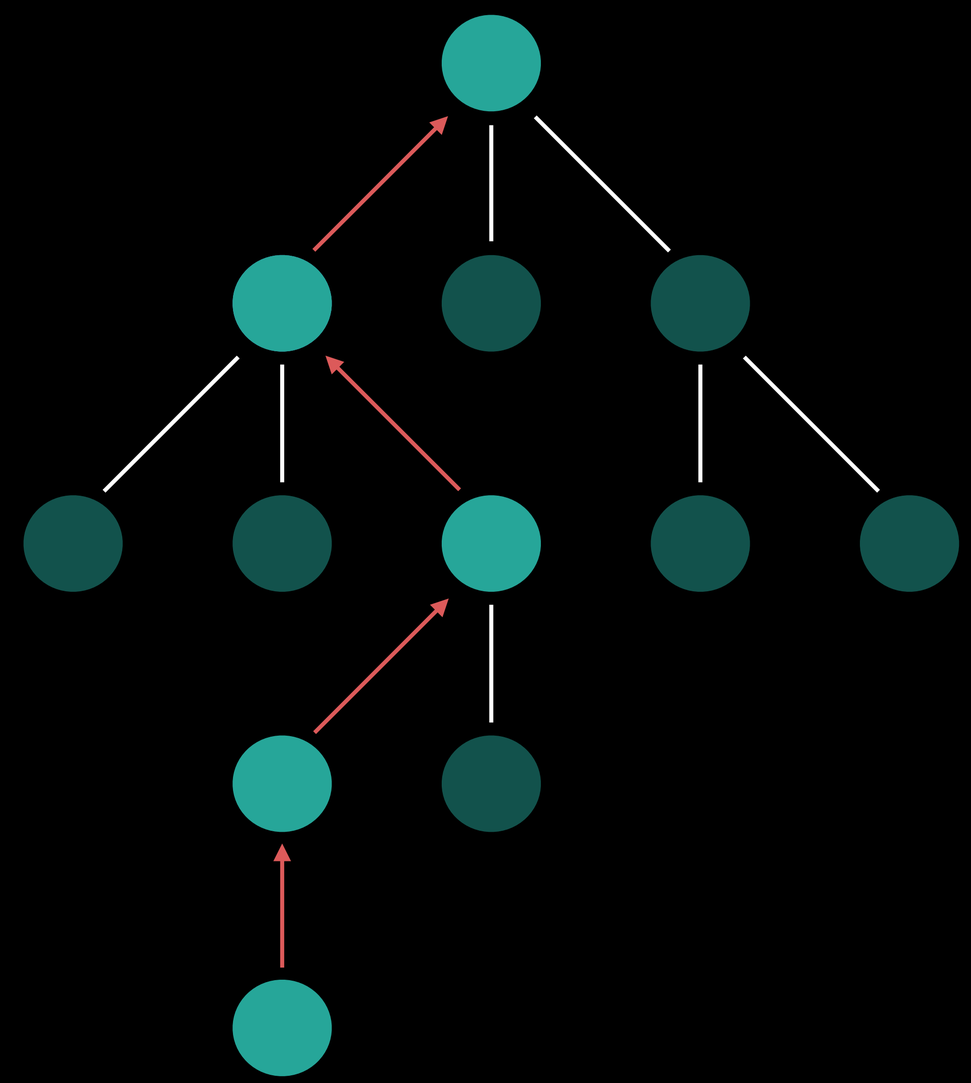
NEW

Best-first search

Random sampling of state space

- Selects player move
- Simulates out new games
- Finds win / loss
- Back-propagates win likelihood

Convergent on optimal move



Game AI

Monte Carlo strategist

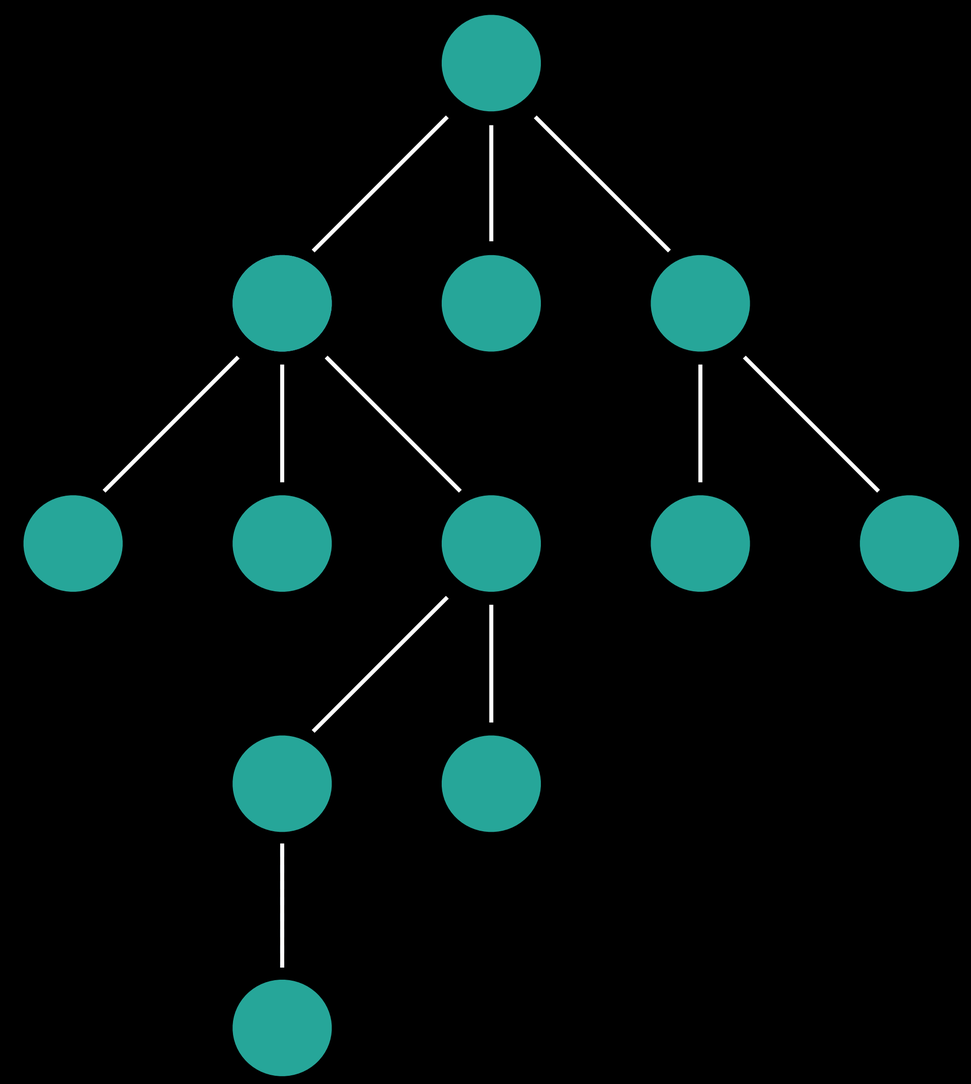
NEW

Fast performance

Works on large state spaces

Only needs win/loss condition

Approximately optimal



Game AI

Elements

GKMonteCarloStrategist

- Budget
- Exploration parameter
- Game model

```swift
// Game AI
// GKMonteCarloStrategist

// Create game model
let gameModel = GoGameModel()
let monteCarlo = GKMonteCarloStrategist()
monteCarlo.gameModel = gameModel

// Maximum number of samples to be processed
monteCarlo.budget = 100
monteCarlo.explorationParameter = 1

// Find best move for player
let modelUpdate = monteCarlo.bestMoveForActivePlayer()
gameModel.applyGameModelUpdate(update: modelUpdate)
```

```swift
// Game AI
// GKMonteCarloStrategist

// Create game model
let gameModel = GoGameModel()
let monteCarlo = GKMonteCarloStrategist()
monteCarlo.gameModel = gameModel

// Maximum number of samples to be processed
monteCarlo.budget = 100
monteCarlo.explorationParameter = 1

// Find best move for player
let modelUpdate = monteCarlo.bestMoveForActivePlayer()
gameModel.applyGameModelUpdate(update: modelUpdate)
```

```swift
// Game AI
// GKMonteCarloStrategist

// Create game model
let gameModel = GoGameModel()
let monteCarlo = GKMonteCarloStrategist()
monteCarlo.gameModel = gameModel

// Maximum number of samples to be processed
monteCarlo.budget = 100
monteCarlo.explorationParameter = 1

// Find best move for player
let modelUpdate = monteCarlo.bestMoveForActivePlayer()
gameModel.applyGameModelUpdate(update: modelUpdate)
```

```swift
// Game AI
// GKMonteCarloStrategist

// Create game model
let gameModel = GoGameModel()
let monteCarlo = GKMonteCarloStrategist()
monteCarlo.gameModel = gameModel

// Maximum number of samples to be processed
monteCarlo.budget = 100
monteCarlo.explorationParameter = 1

// Find best move for player
let modelUpdate = monteCarlo.bestMoveForActivePlayer()
gameModel.applyGameModelUpdate(update: modelUpdate)
```

Game AI

Custom strategists

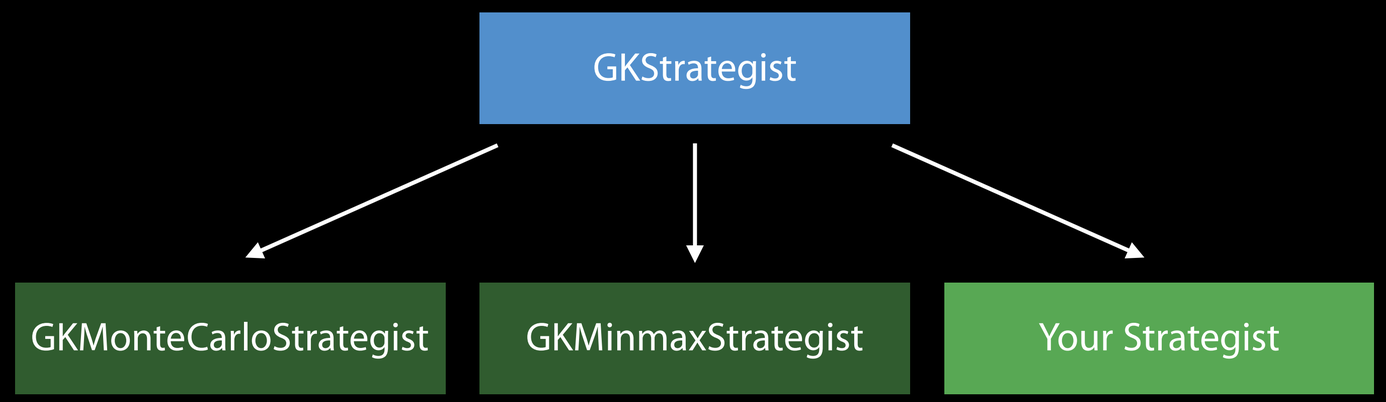
NEW

Define your own strategist

You provide:

- GKGameModel
- GKGameModelUpdate
- GKGameModelPlayer

And implement `bestMoveForActivePlayer`



Game AI

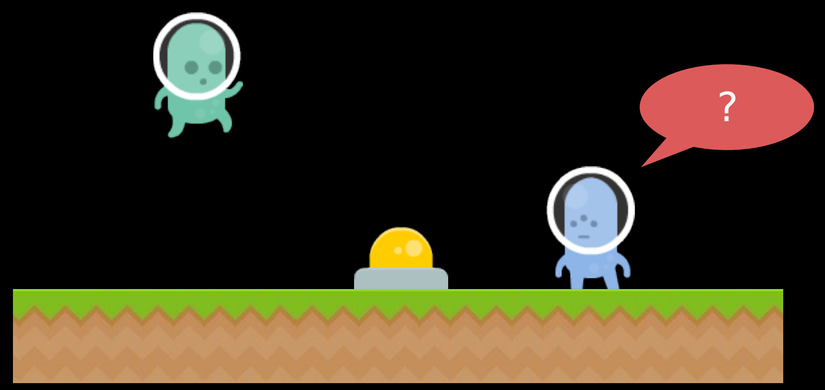
Decision making

Many ways to model logic

Entities need to make decisions

- Need to consider lots of state

Need to make decisions quickly



Decision Trees

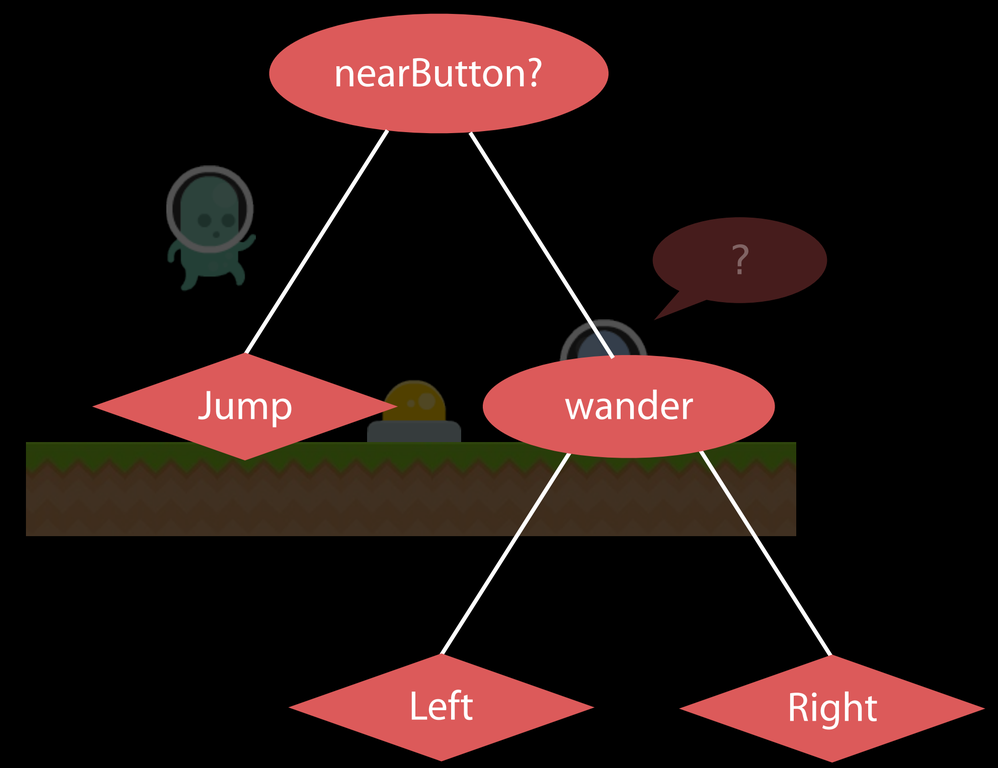
Overview

System for making decisions

Tree data structure

Easy to visualize and debug

Can be handmade or learned



Game AI

GKDecisionTree

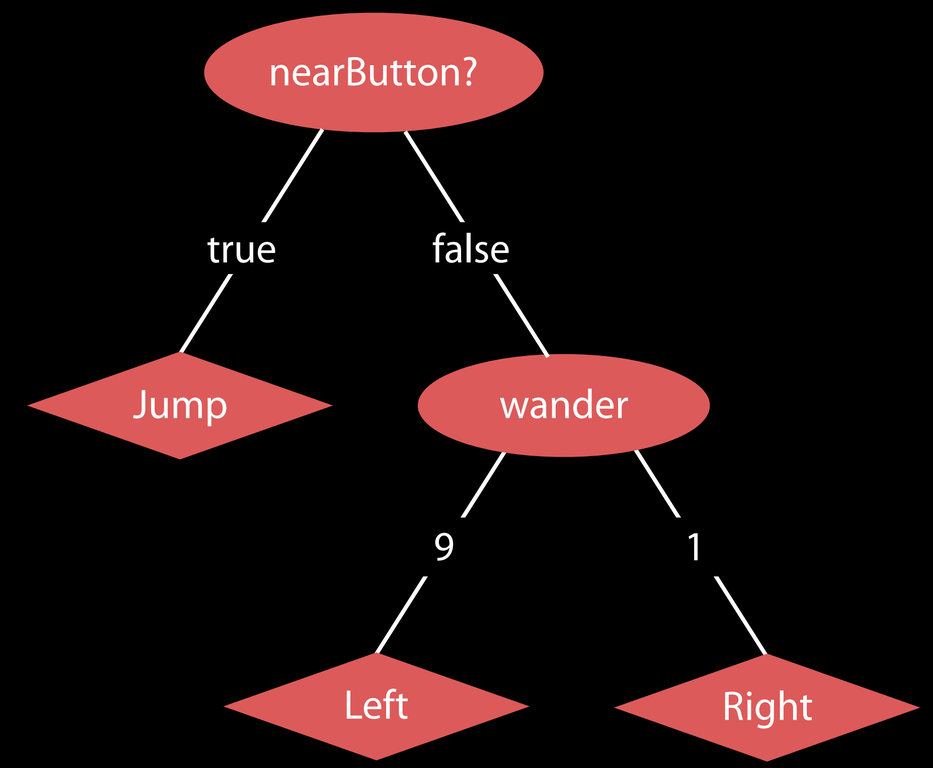
NEW

Low overhead for determining action

Serializable

Uses branching for:

- Values
- Predicates
- Random weights



Game AI

GKDecisionTree

```
// Create tree and get root
let tree = GKDecisionTree(attribute: “nearButton?”)
let root = tree?.rootNode
```

```
// Create branches
root.createBranch(value: true, attribute: “Jump”)
let wander = root.createBranch(value: false, attribute: “wander”)
```

```
// Create actions for when nearby
wander.createBranch(withWeight: 9, attribute: “Left”)
wander.createBranch(withWeight: 1, attribute: “Right”)
```

```
// Find action for answers
let answers = [“nearButton?” : true,…]
tree.findActionForAnswers(answers: answers)
```

Game AI

GKDecisionTree

```
// Create tree and get root
let tree = GKDecisionTree(attribute: “nearButton?”)
let root = tree?.rootNode
```

```
// Create branches
root.createBranch(value: true, attribute: “Jump”)
let wander = root.createBranch(value: false, attribute: “wander”)
```

```
// Create actions for when nearby
wander.createBranch(withWeight: 9, attribute: “Left”)
wander.createBranch(withWeight: 1, attribute: “Right”)
```

```
// Find action for answers
let answers = [“nearButton?” : true,…]
tree.findActionForAnswers(answers: answers)
```

Game AI

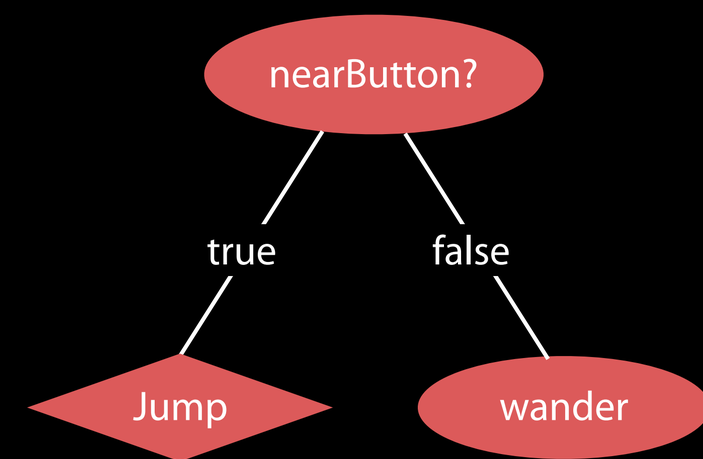
GKDecisionTree

```
// Create tree and get root
let tree = GKDecisionTree(attribute: “nearButton?”)
let root = tree?.rootNode
```

```
// Create branches
root.createBranch(value: true, attribute: “Jump”)
let wander = root.createBranch(value: false, attribute: “wander”)
```

```
// Create actions for when nearby
wander.createBranch(withWeight: 9, attribute: “Left”)
wander.createBranch(withWeight: 1, attribute: “Right”)
```

```
// Find action for answers
let answers = [“nearButton?” : true,…]
tree.findActionForAnswers(answers: answers)
```



Game AI

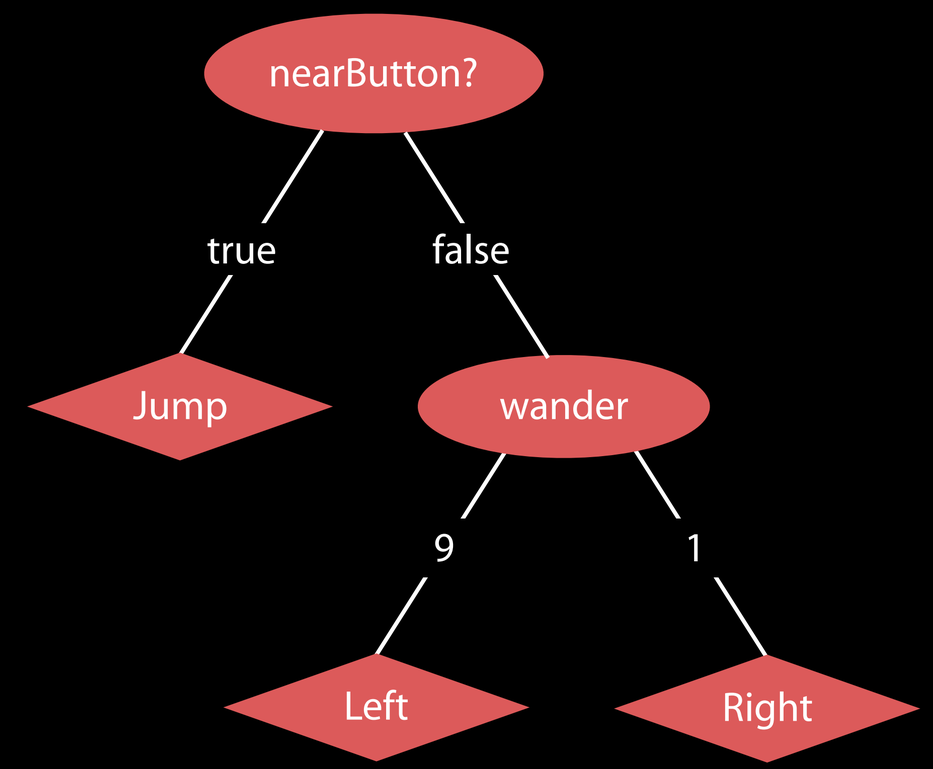
GKDecisionTree

```
// Create tree and get root
let tree = GKDecisionTree(attribute: “nearButton?”)
let root = tree?.rootNode
```

```
// Create branches
root.createBranch(value: true, attribute: “Jump”)
let wander = root.createBranch(value: false, attribute: “wander”)
```

```
// Create actions for when nearby
wander.createBranch(withWeight: 9, attribute: “Left”)
wander.createBranch(withWeight: 1, attribute: “Right”)
```

```
// Find action for answers
let answers = [“nearButton?” : true,…]
tree.findActionForAnswers(answers: answers)
```



Game AI

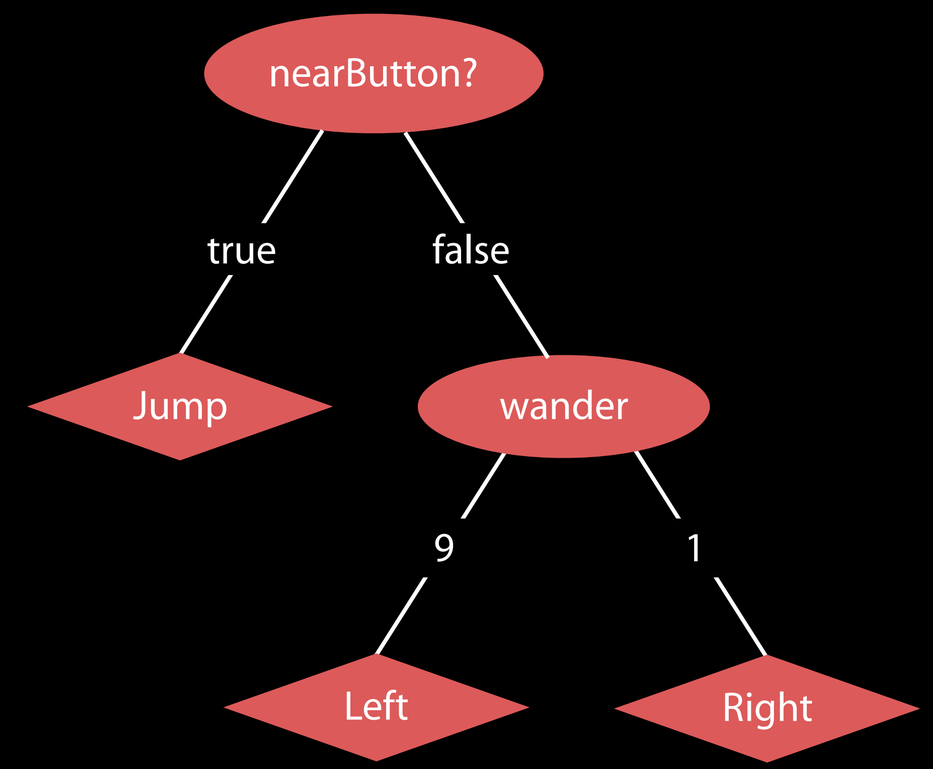
GKDecisionTree

```
// Create tree and get root
let tree = GKDecisionTree(attribute: “nearButton?”)
let root = tree?.rootNode
```

```
// Create branches
root.createBranch(value: true, attribute: “Jump”)
let wander = root.createBranch(value: false, attribute: “wander”)
```

```
// Create actions for when nearby
wander.createBranch(withWeight: 9, attribute: “Left”)
wander.createBranch(withWeight: 1, attribute: “Right”)
```

```
// Find action for answers
let answers = [“nearButton?” : true,…]
tree.findActionForAnswers(answers: answers)
```



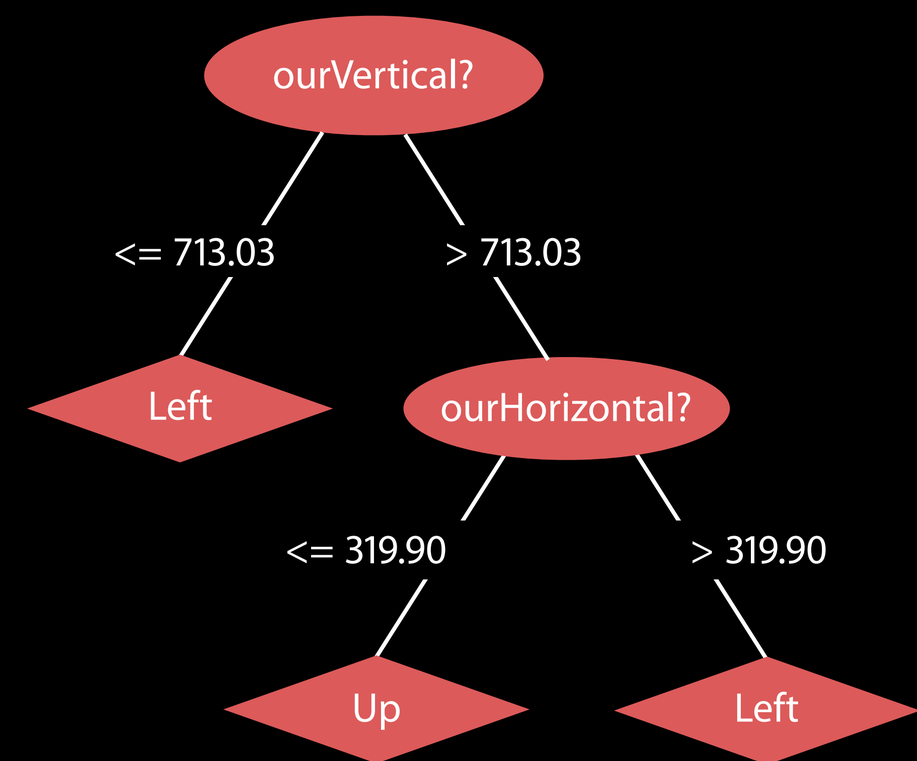
Game AI

GKDecisionTree

Decision trees can be modeled

Supply gameplay data:

- Finds decision-making behavior
- Fit decision tree to these behaviors



Game AI

GKDecisionTree

NEW

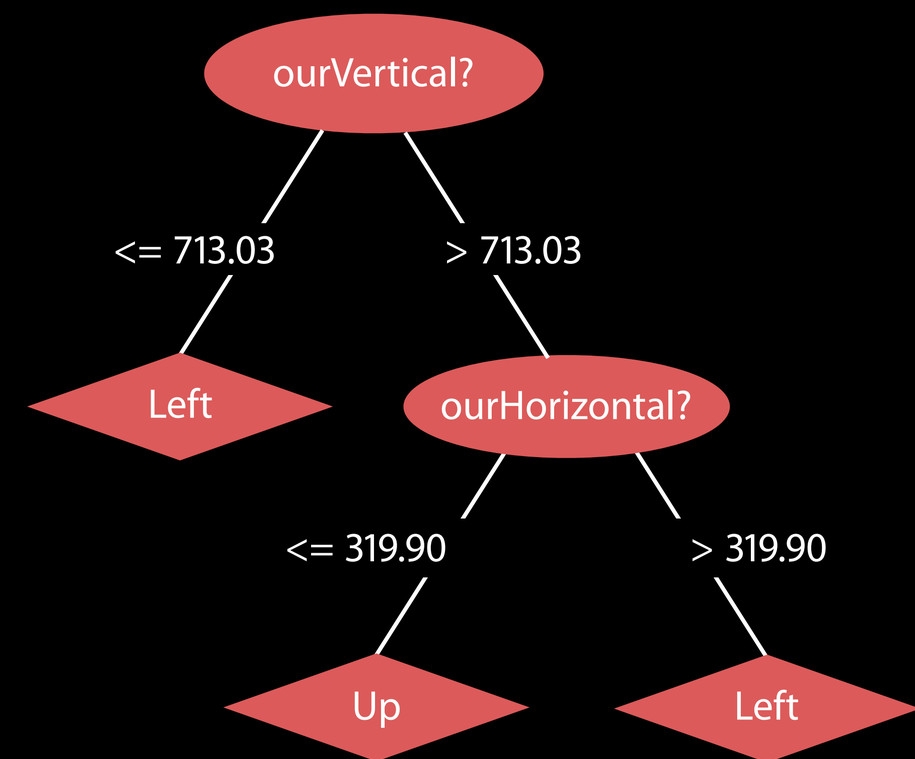
ourVertical	ourHorizontal	actions
404.9131	1396.555	Left
396.1226	1587.167	Left
741.8968	160.7707	Up
503.0585	1550.767	Left
436.9709	1472.766	Left
…	…	…

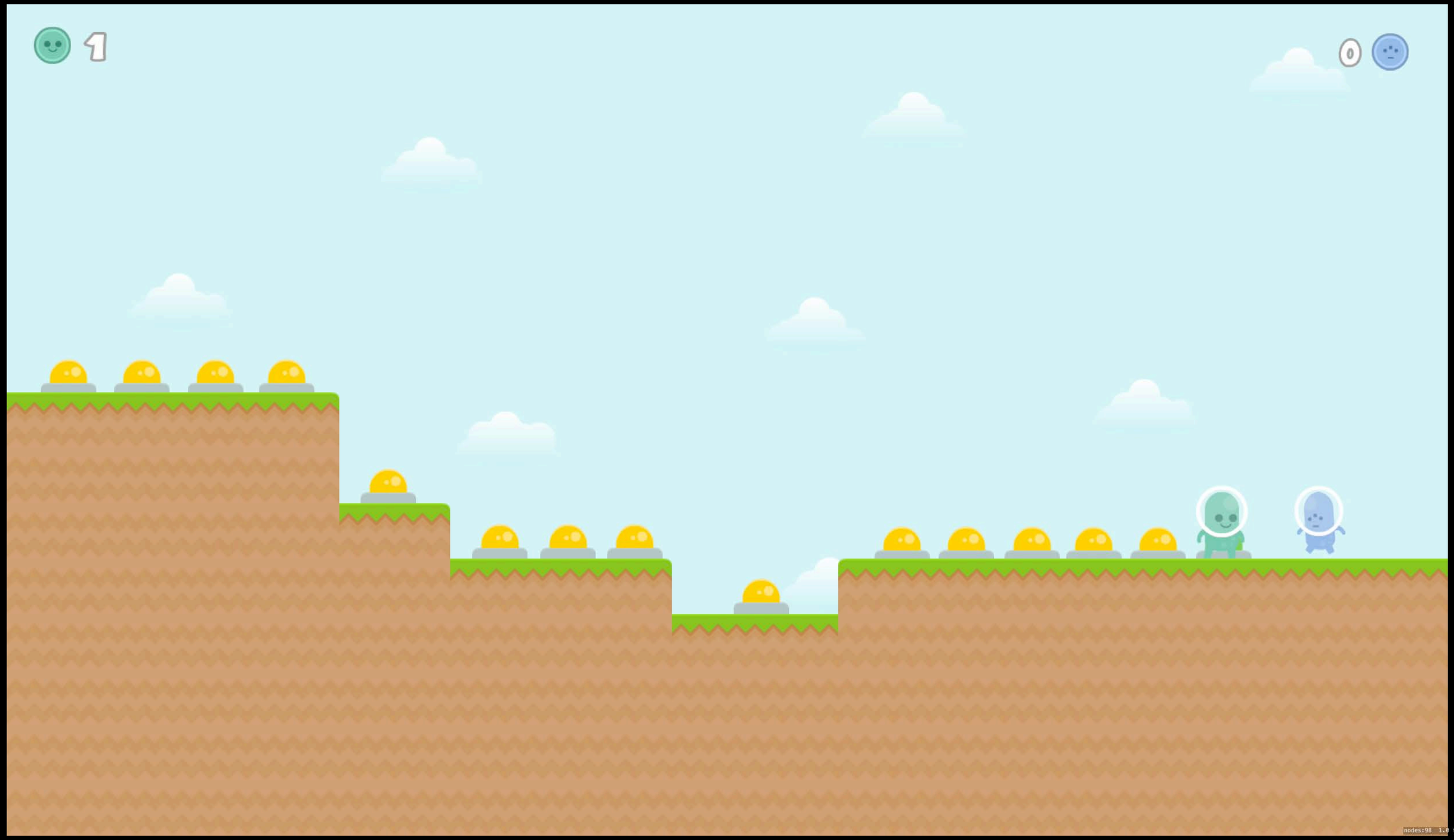
Game AI

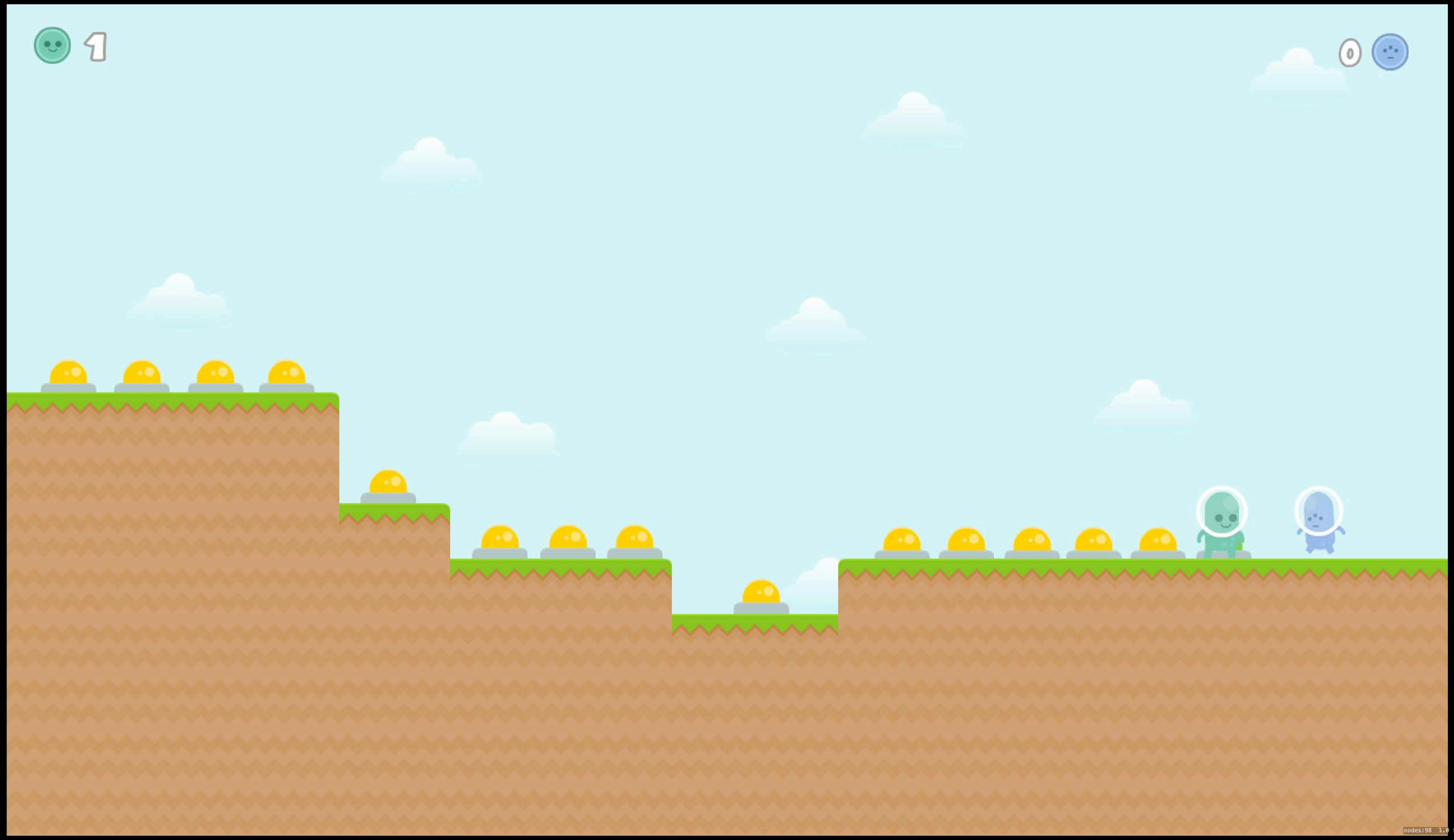
GKDecisionTree

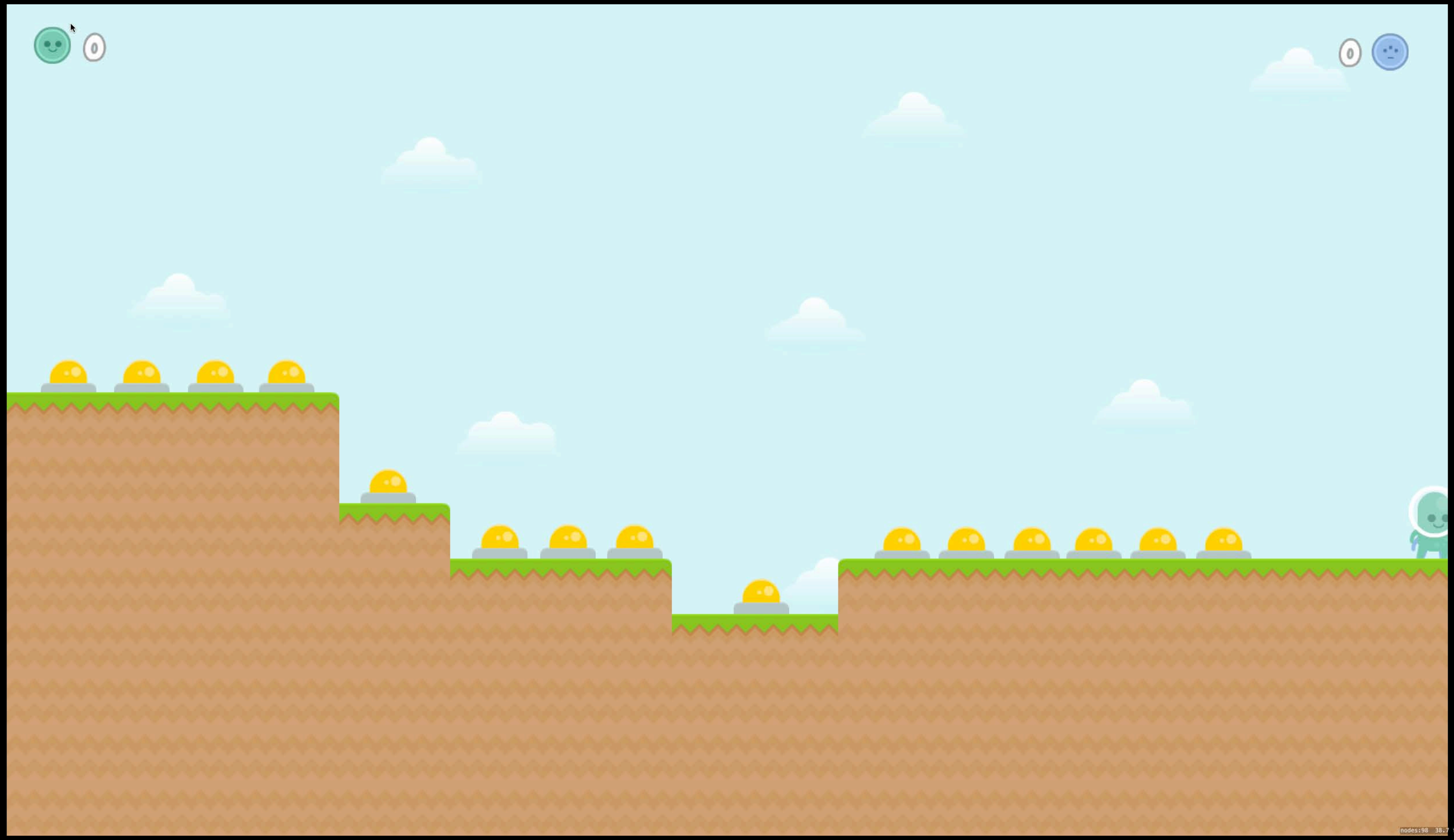
NEW

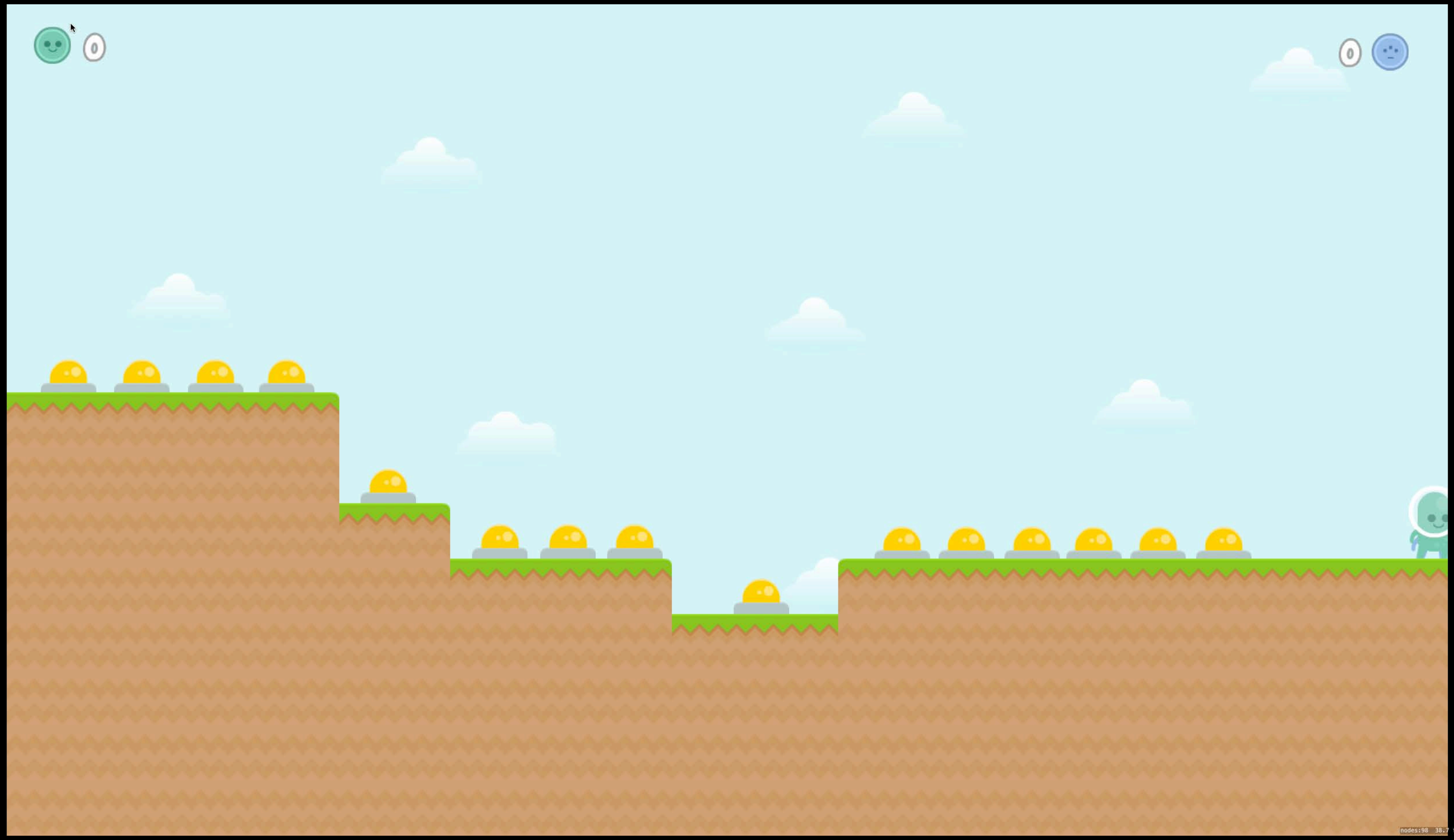
ourVertical	ourHorizontal	actions
404.9131	1396.555	Left
396.1226	1587.167	Left
741.8968	160.7707	Up
503.0585	1550.767	Left
436.9709	1472.766	Left
...	...	...




1
0
nodes:98 1.0 fps


1
0
nodes:98 1.0 fps


0
0
nodes:98 38.7 fps


0
0
nodes:98 38.7 fps

Xcode Editor Integration

Sri Nair Game Technologies Engineer

Xcode Editor Integration

Background

Xcode Editor Integration

Background

```swift
class MovementComponent: GKComponent {

    var maxSpeed: Float = 5.2
    var friction: Float = 1.0
    var acceleration: Float = 1.0
    var rollSpeed: Float = 9.0
    var velocity: Float = 0
    var frozen: Bool = false
    var carrying: Bool = false
    var state: MovementState = .idle {
        didSet {
            if let carriedObject = carriedObject {
                carriedObject.state = state
            }
        }
    }
```

```swift
// Create components that define how the entity looks and behaves.
let renderComponent = RenderComponent(entity: self)
addComponent(renderComponent)

let orientationComponent = OrientationComponent()
addComponent(orientationComponent)

let shadowComponent = ShadowComponent(texture: FlyingBot.shadowTex
    FlyingBot.shadowOffset)
addComponent(shadowComponent)

let animationComponent = AnimationComponent(textureSize: FlyingBot
addComponent(animationComponent)

let intelligenceComponent = IntelligenceComponent(states: [
    TaskBotAgentControlledState(entity: self),
    FlyingBotPreAttackState(entity: self),
    FlyingBotBlastState(entity: self),
    TaskBotZappedState(entity: self)
])
addComponent(intelligenceComponent)
```

```swift
func initialize() {
    // initialize Navigation Graph
    let points = [[-10.0, 50.0], [0.0, 50.0], [10.0, 50.0],
                  [-10.0, -50.0], [0.0, -50.0], [10.0, -50.0]]
    var nodes = [GKGraphNode2D]()
    for point in points {
        let pt = float2(Float(point[0]), Float(point[1]))
        nodes.append(GKGraphNode2D(point:pt))
    }
    navGraph.addNodes(nodes)
}

override func update(withDeltaTime seconds: TimeInterval) {
    // Find the next point to follow on path
    if (followRandomPath) {
        var index = GKRandomSource.sharedRandom().nextInt(withUpperBound:navGraph.nodes!.count)
        while (lastIndexChosen ==  index) {
            index = GKRandomSource.sharedRandom().nextInt(withUpperBound:navGraph.nodes!.count)
        }
        lastIndexChosen = index;
        let targetNode = navGraph.nodes![lastIndexChosen] as! GKGraphNode2D

        self.path = [targetNode.position]
    }
```

Xcode Editor Integration

Background

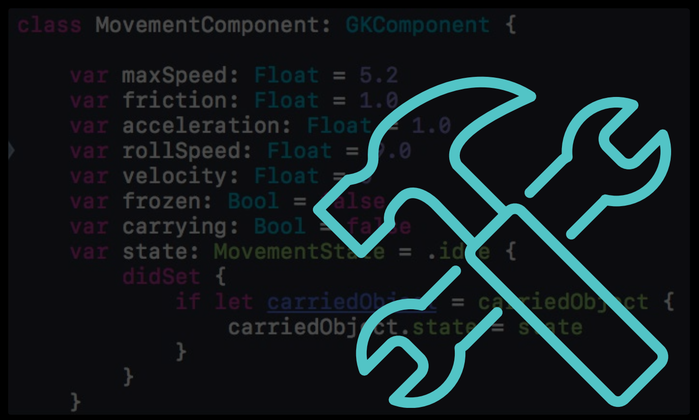
```
class MovementComponent: GKComponent {

    var maxSpeed: Float = 5.2
    var friction: Float = 1.0
    var acceleration: Float = 1.0
    var rollSpeed: Float = 9.0
    var velocity: Float = 0
    var frozen: Bool = false
    var carrying: Bool = false
    var state: MovementState = .idle {
        didSet {
            if let carriedObject = carriedObject {
                carriedObject.state = state
            }
        }
    }
```

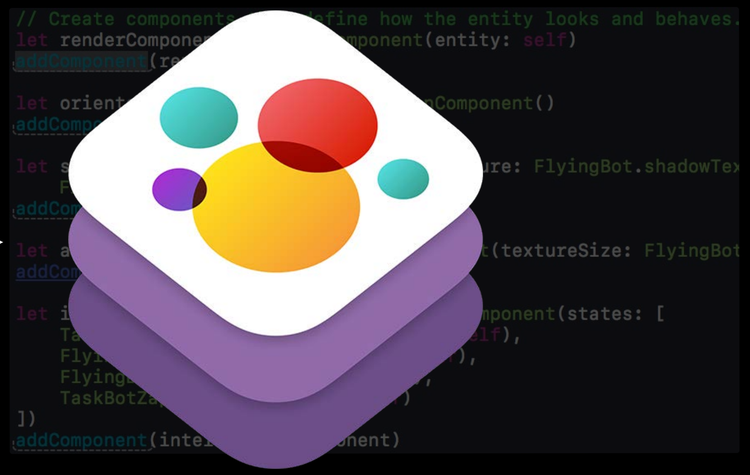
```
// Create components that define how the entity looks and behaves.
let renderComponent = RenderComponent(entity: self)
addComponent(renderComponent)

let orientationComponent = OrientationComponent()
addComponent(orientationComponent)

let shadowComponent = ShadowComponent(texture: FlyingBot.shadowTex
    FlyingBot.shadowOffset)
addComponent(shadowComponent)

let animationComponent = AnimationComponent(textureSize: FlyingBot
addComponent(animationComponent)

let intelligenceComponent = IntelligenceComponent(states: [
    TaskBotAgentControlledState(entity: self),
    FlyingBotPreAttackState(entity: self),
    FlyingBotBlastState(entity: self),
    TaskBotZappedState(entity: self)
])
addComponent(intelligenceComponent)
```

```
func initialize() {
    // initialize Navigation Graph
    let points = [[-10.0, 50.0], [0.0, 50.0], [10.0, 50.0],
                  [-10.0, -50.0], [0.0, -50.0], [10.0, -50.0]]
    var nodes = [GKGraphNode2D]()
    for point in points {
        let pt = float2(Float(point[0]), Float(point[1]))
        nodes.append(GKGraphNode2D(point:pt))
    }
    navGraph.addNodes(nodes)
}

override func update(withDeltaTime seconds: TimeInterval) {
    // Find the next point to follow on path
    if (followRandomPath) {
        var index = GKRandomSource.sharedRandom().nextInt(withUpperBound:navGraph.nodes!.count)
        while (lastIndexChosen ==  index) {
            index = GKRandomSource.sharedRandom().nextInt(withUpperBound:navGraph.nodes!.count)
        }
        lastIndexChosen = index;
        let targetNode = navGraph.nodes![lastIndexChosen] as! GKGraphNode2D

        self.path = [targetNode.position]
    }
```

Xcode Editor Integration

Background

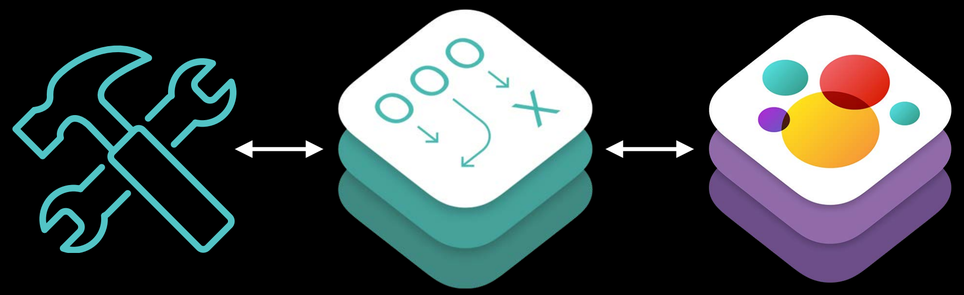





```
func initialize() {
    // initialize Navigation Graph
    let points = [[-10.0, 50.0], [0.0, 50.0], [10.0, 50.0],
                  [-10.0, -50.0], [0.0, -50.0], [10.0, -50.0]]
    var nodes = [GKGraphNode2D]()
    for point in points {
        let pt = float2(Float(point[0]), Float(point[1]))
        nodes.append(GKGraphNode2D(point:pt))
    }
    navGraph.addNodes(nodes)
}

override func update(withDeltaTime seconds: TimeInterval) {
    // Find the next point to follow on path
    if (followRandomPath) {
        var index = GKRandomSource.sharedRandom().nextInt(withUpperBound:navGraph.nodes!.count)
        while (lastIndexChosen ==  index) {
            index = GKRandomSource.sharedRandom().nextInt(withUpperBound:navGraph.nodes!.count)
        }
        lastIndexChosen = index;
        let targetNode = navGraph.nodes![lastIndexChosen] as! GKGraphNode2D

        self.path = [targetNode.position]
    }
```

Xcode Editor Integration

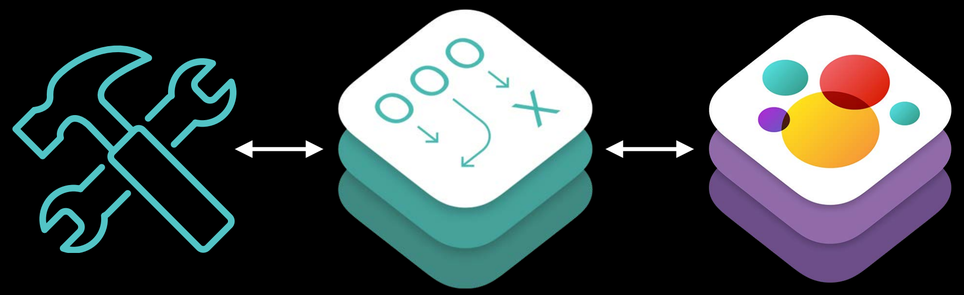
Overview



Xcode Editor Integration

Overview

Entity and Components Editor

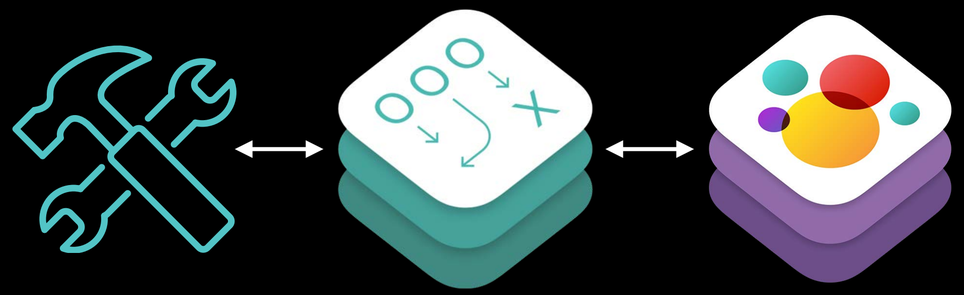


Xcode Editor Integration

Overview

Entity and Components Editor

Navigation Graph Editor



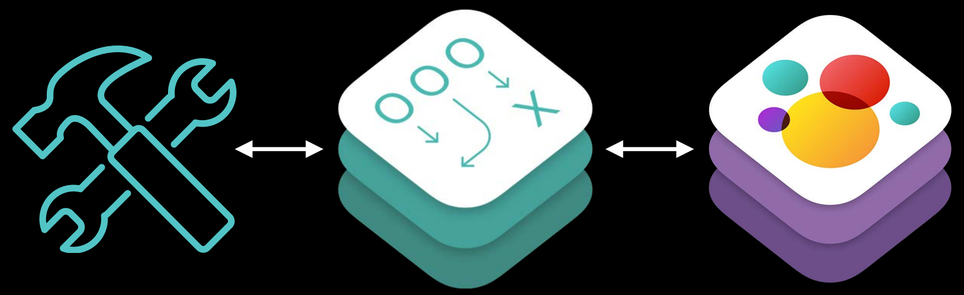
Xcode Editor Integration

Overview

Entity and Components Editor

Navigation Graph Editor

Scene Outline View



Xcode Editor Integration

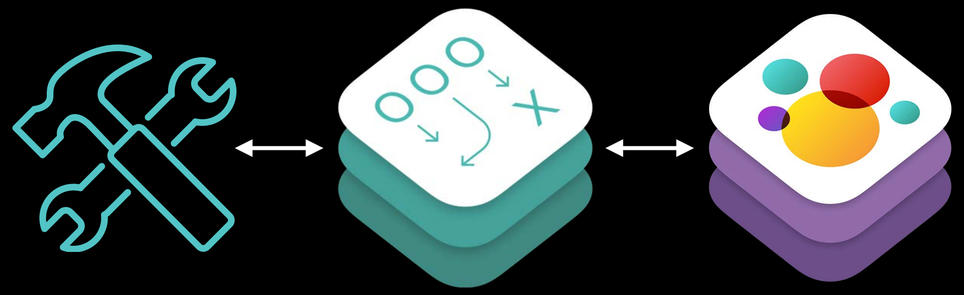
Overview

Entity and Components Editor

Navigation Graph Editor

Scene Outline View

State Machine Quick Look



Component Editor

Component Editor

Entity and components

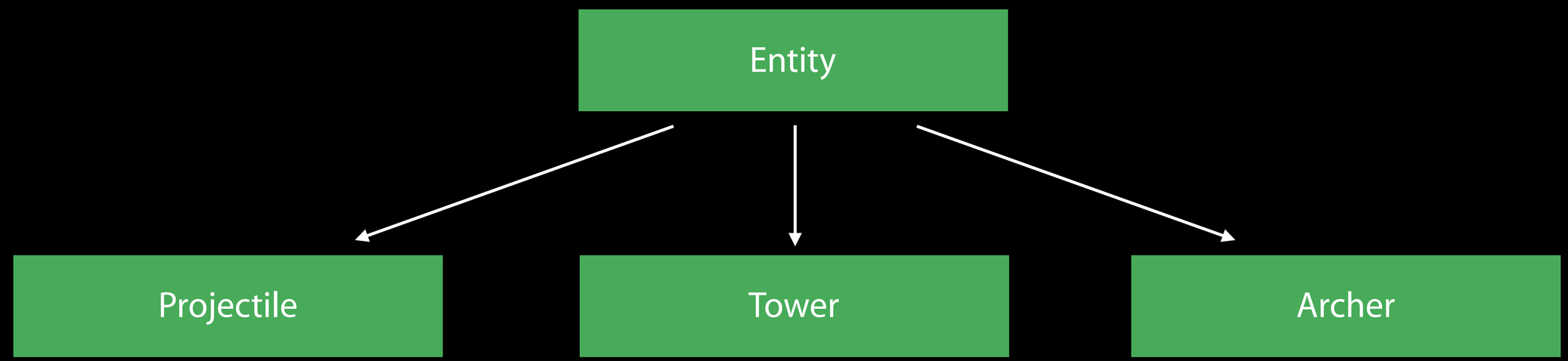
Component Editor

Entity and components

Entity

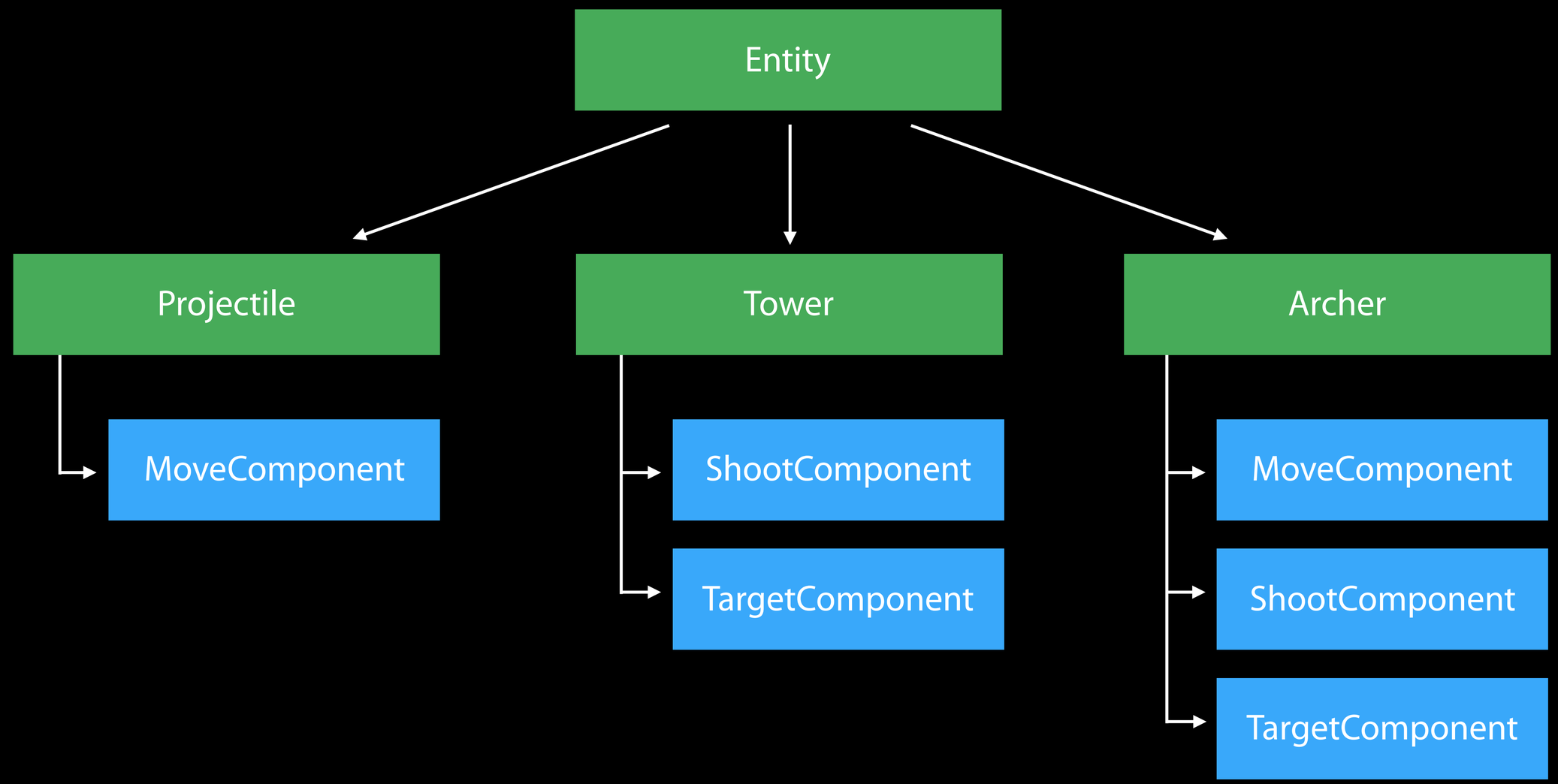
Component Editor

Entity and components

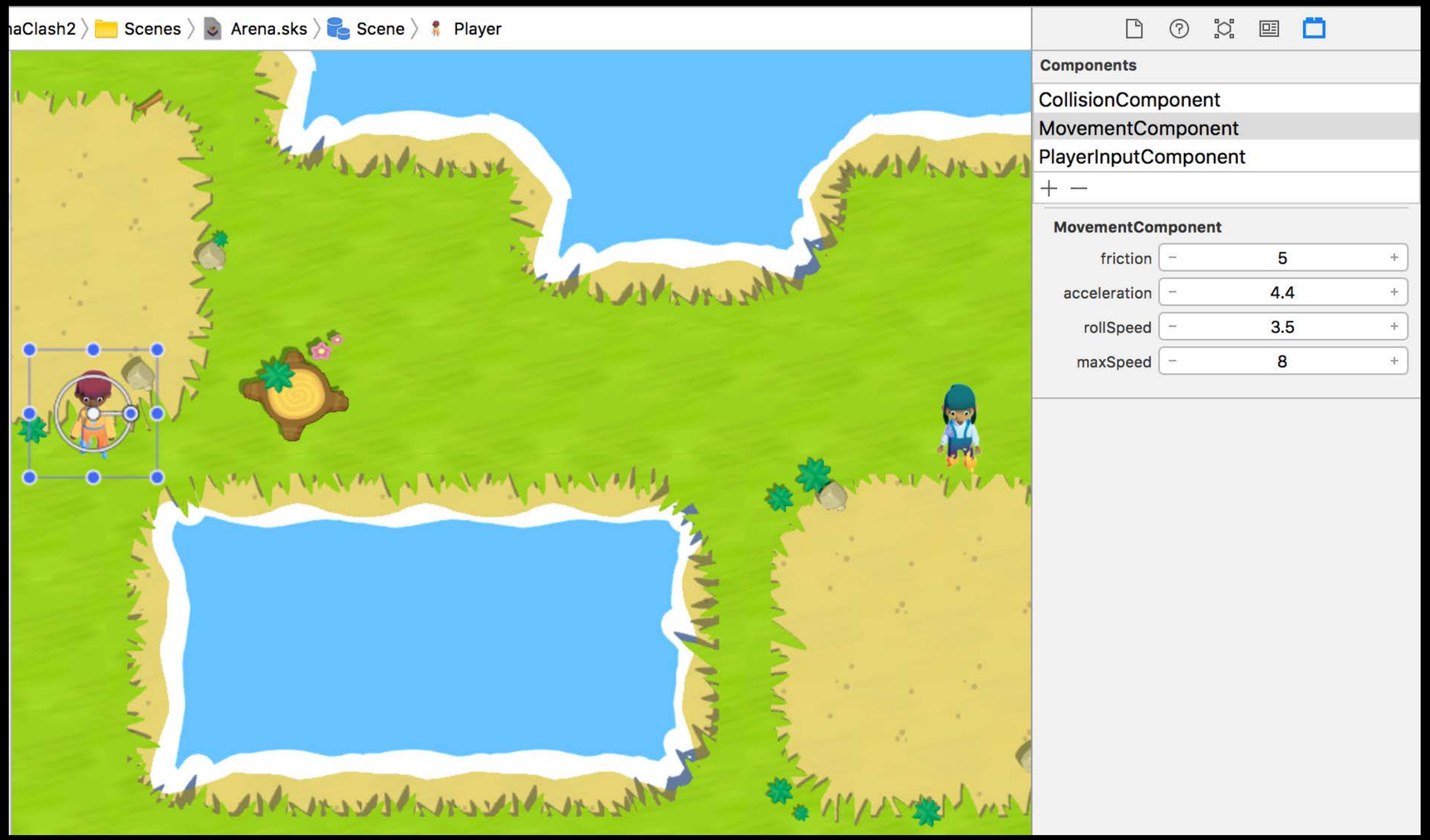


Component Editor

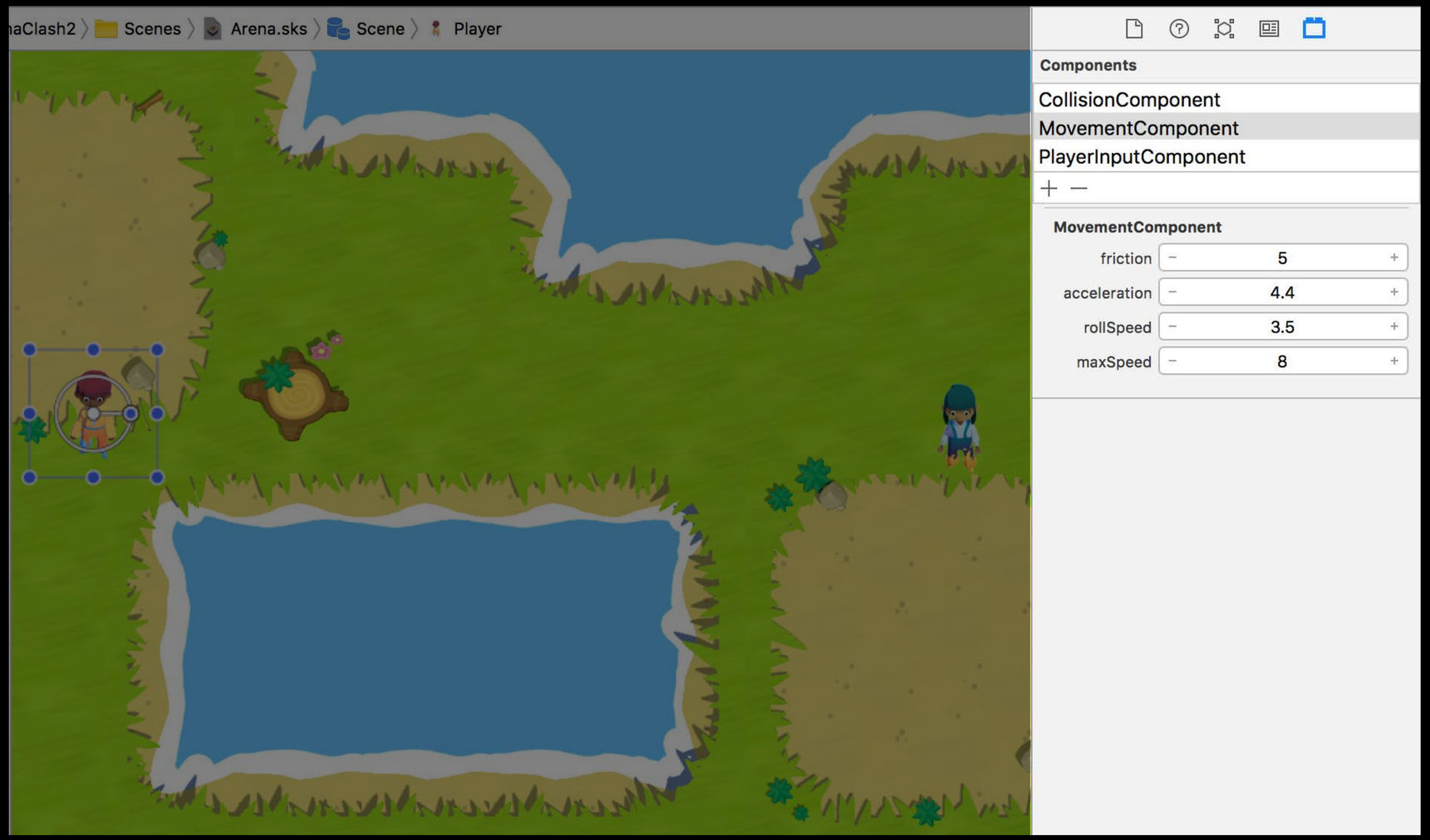
Entity and components



Component Editor

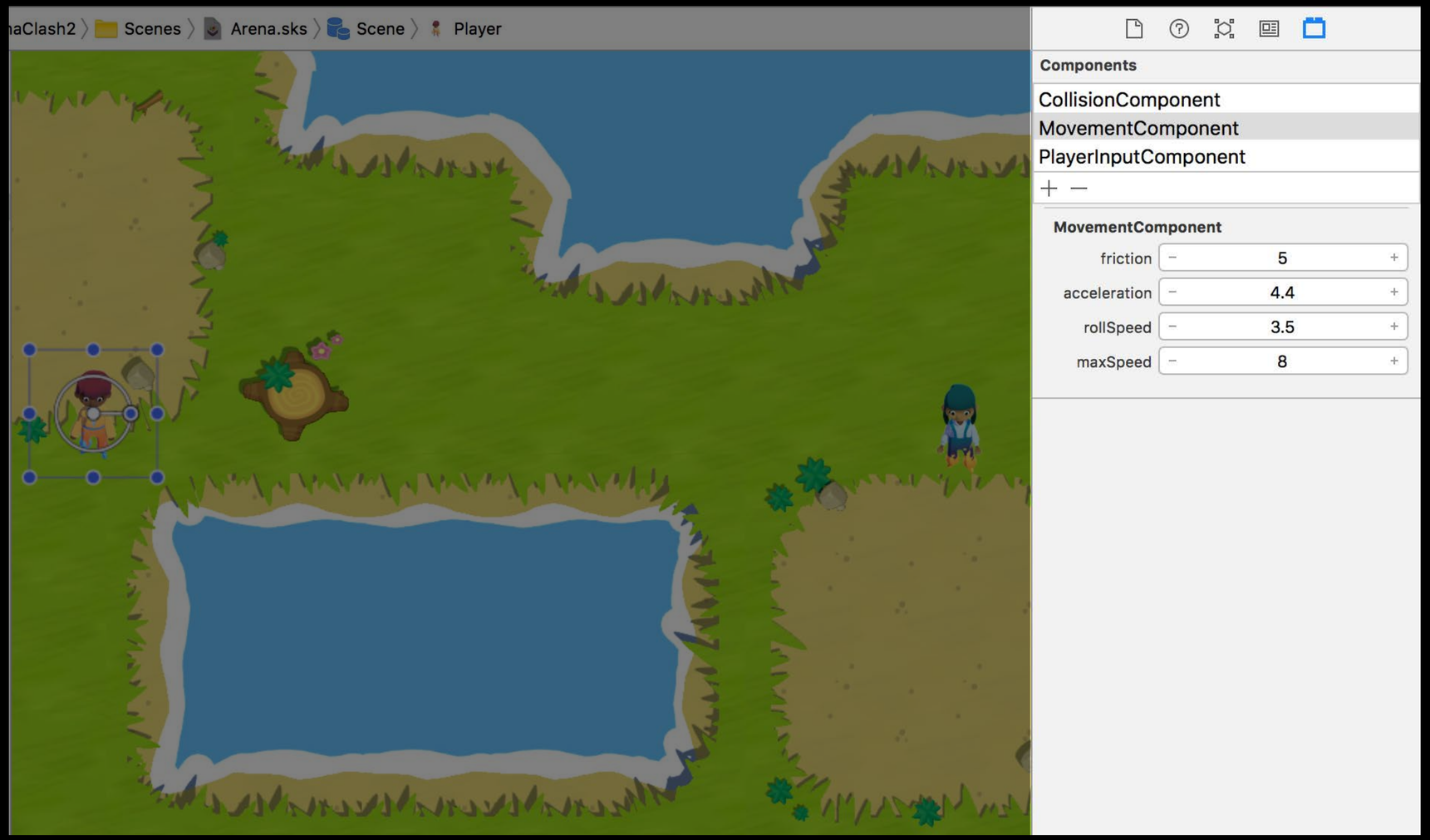


Component Editor



Component Editor

Autodiscovery

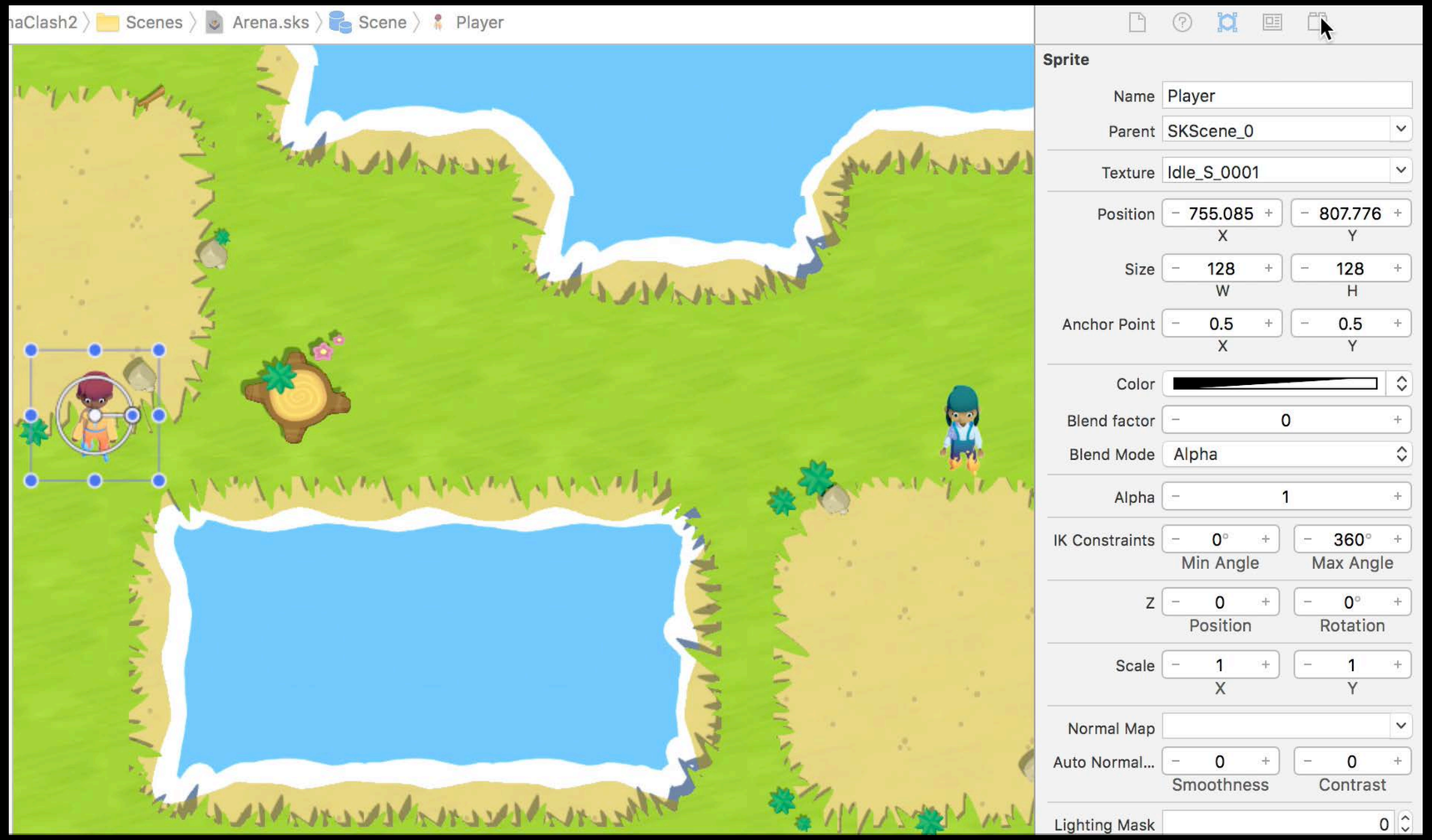


```
// Component Editor
// Autodiscovery


class MovementComponent: GKComponent {
    @GKInspectable var maxSpeed: Float = 5.2
    @GKInspectable var friction: Float = 1.0
    @GKInspectable var acceleration: Float = 1.0
    @GKInspectable var rollSpeed: Float = 9.0
}
```

```
// Component Editor
// Autodiscovery


class MovementComponent: GKComponent {
    @GKInspectable var maxSpeed: Float = 5.2
    @GKInspectable var friction: Float = 1.0
    @GKInspectable var acceleration: Float = 1.0
    @GKInspectable var rollSpeed: Float = 9.0
}
```

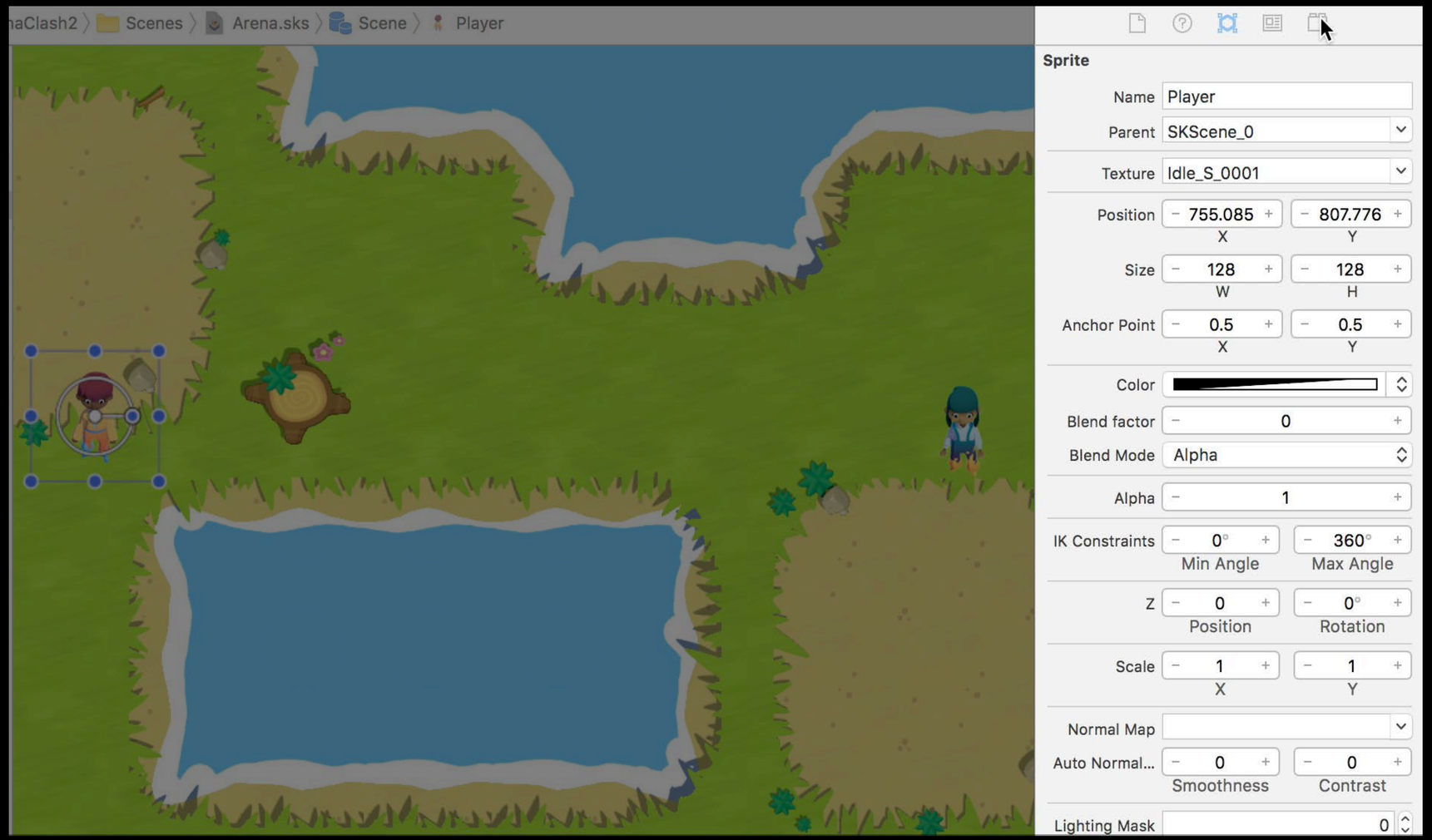
Component Editor

Add component



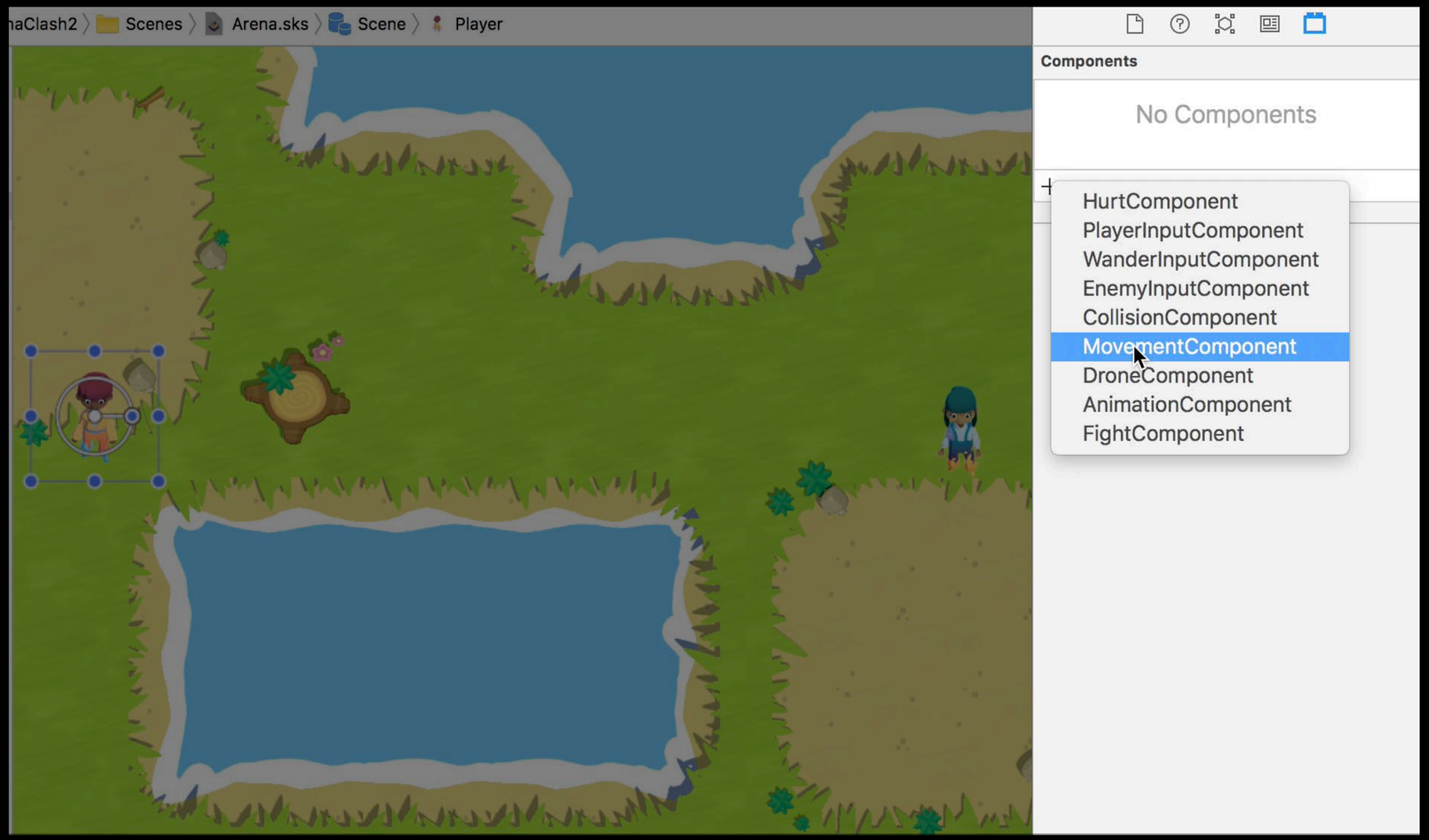
Component Editor

Add component



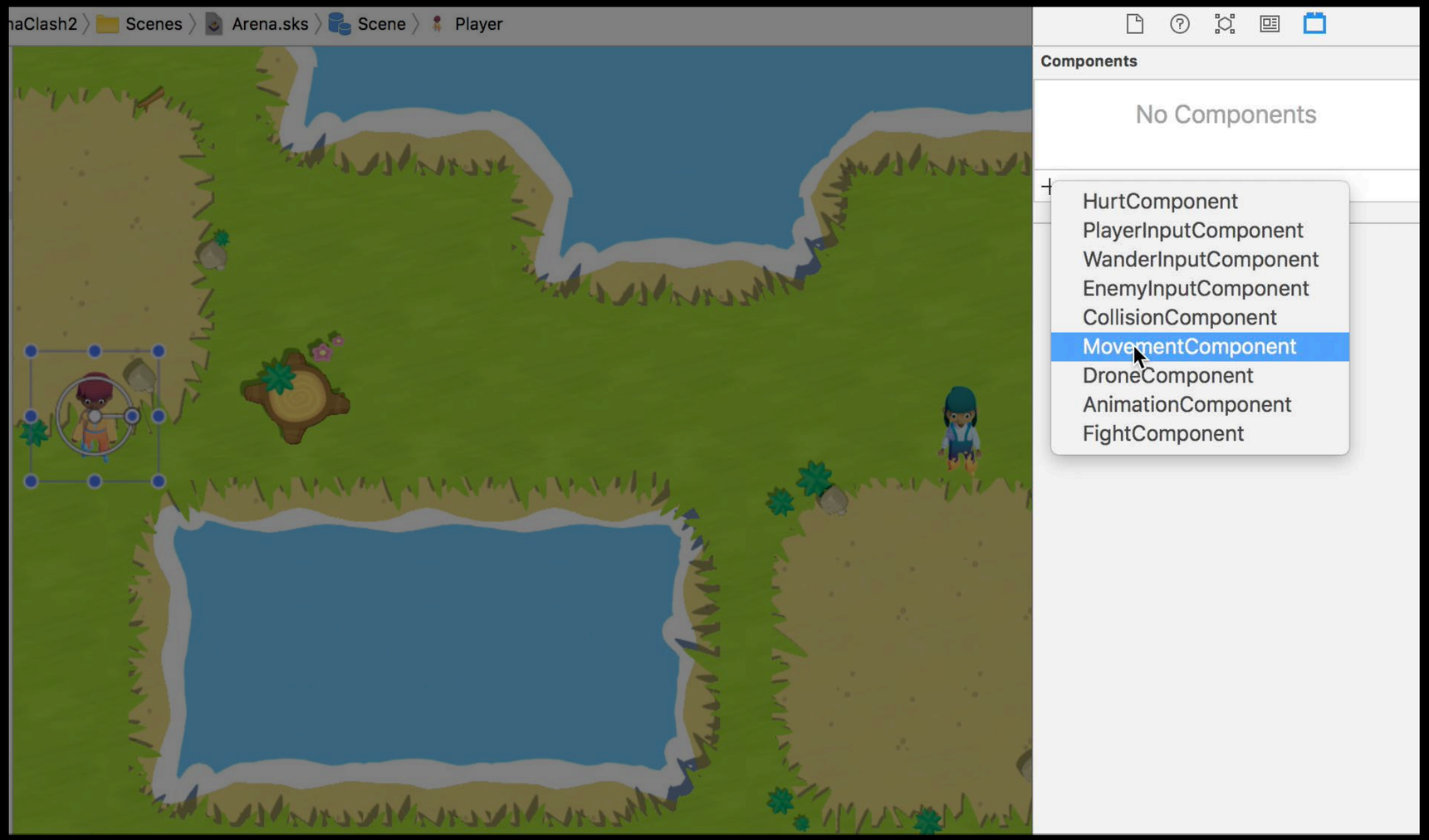
Component Editor

Add component



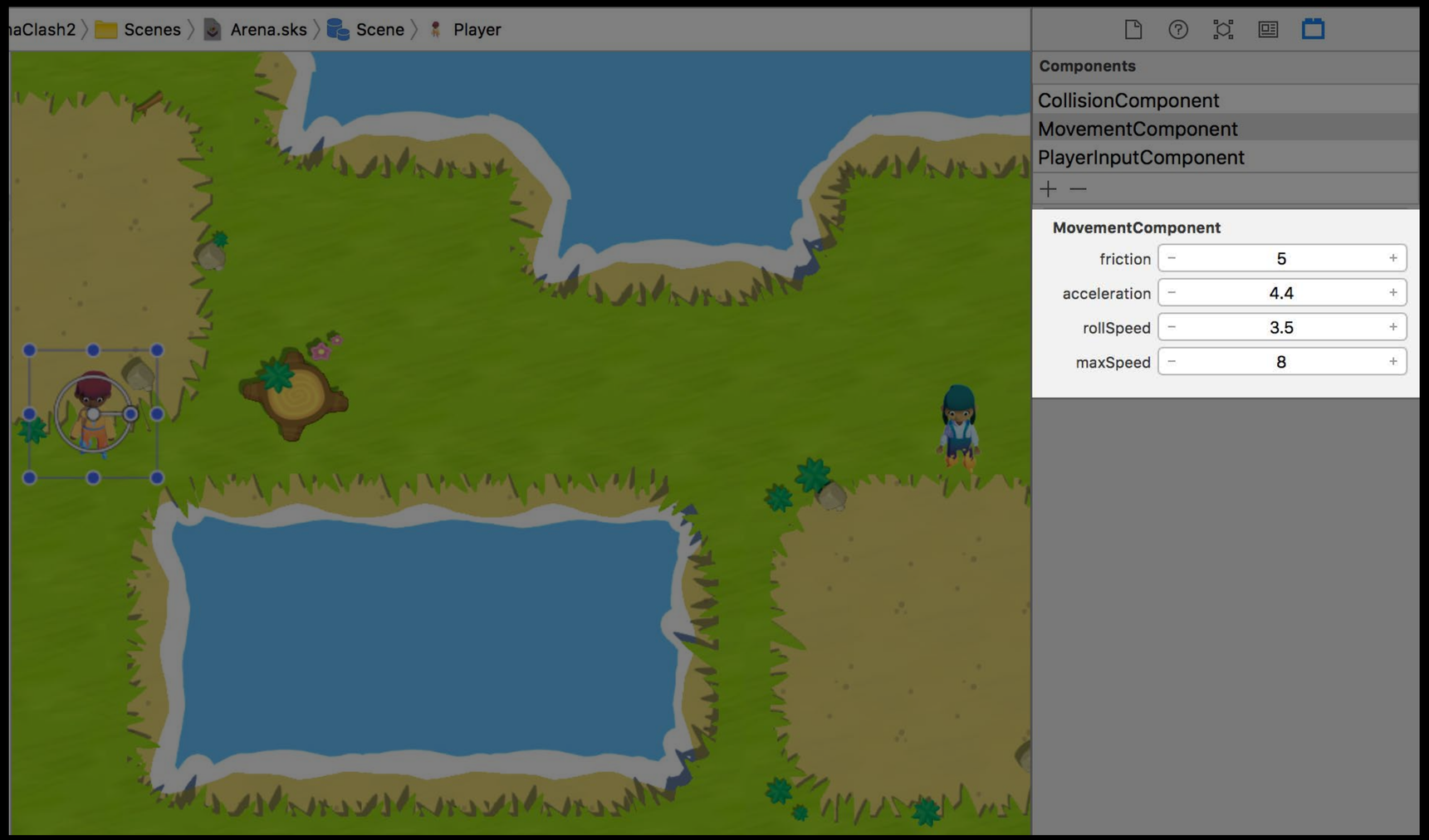
Component Editor

Add component



Component Editor

Tweak properties



Component Editor

Details

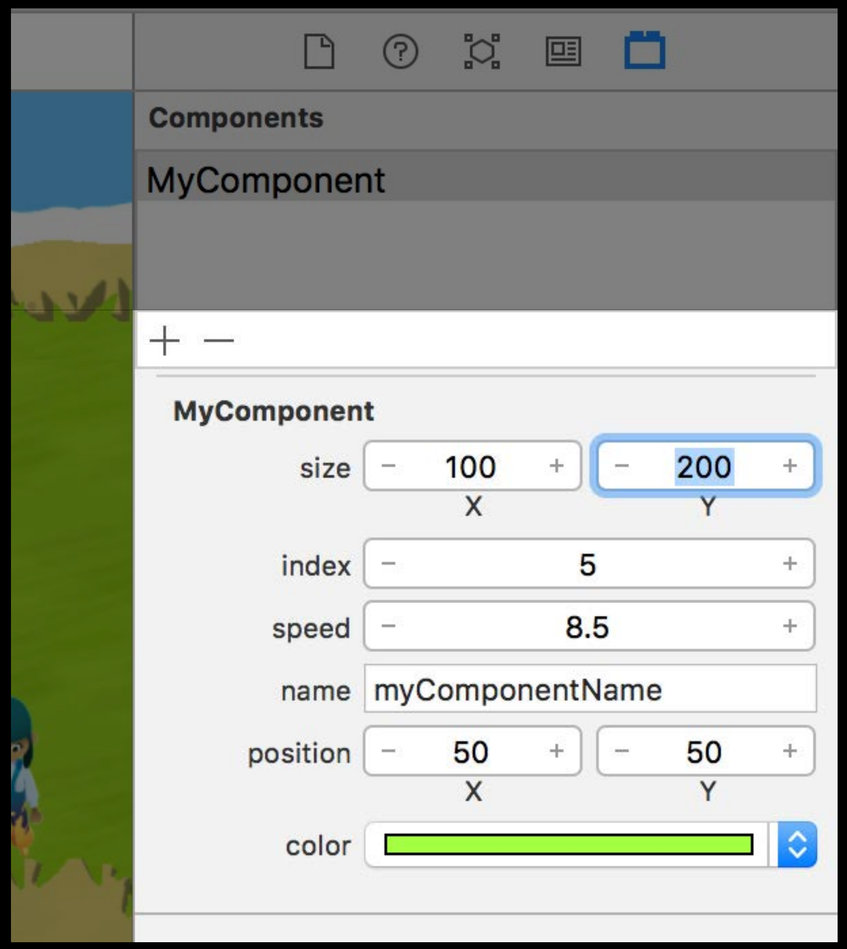
Updates stored under `GKScene` in the SKS file

- Unchanged properties use default values

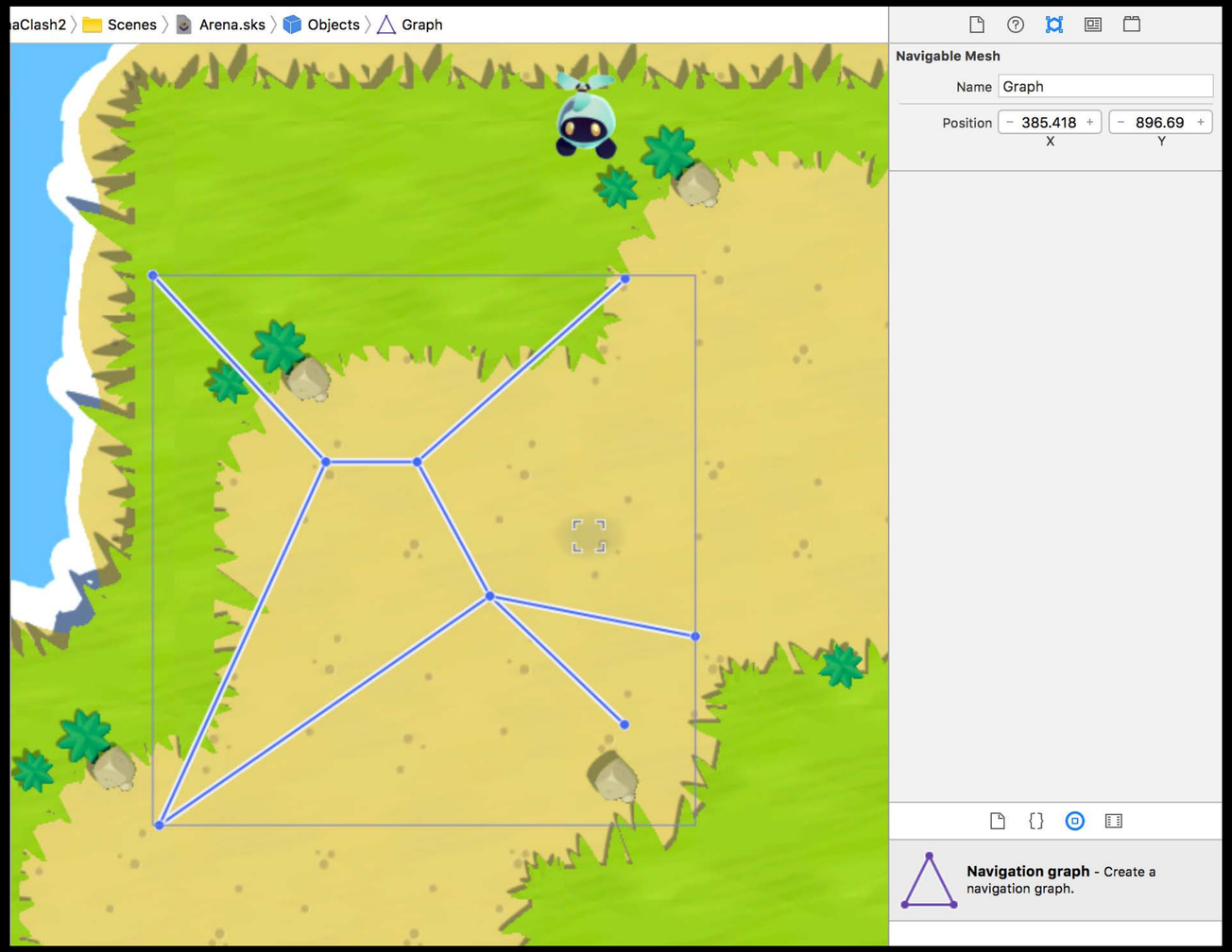
GKEntity linked to nodes via `GKSKComponent`

Common property types are supported

- Float, Int, Bool, Color, Point, Size, String



Navigation Graph Editor



Navigation Graph Editor

Navigation graph

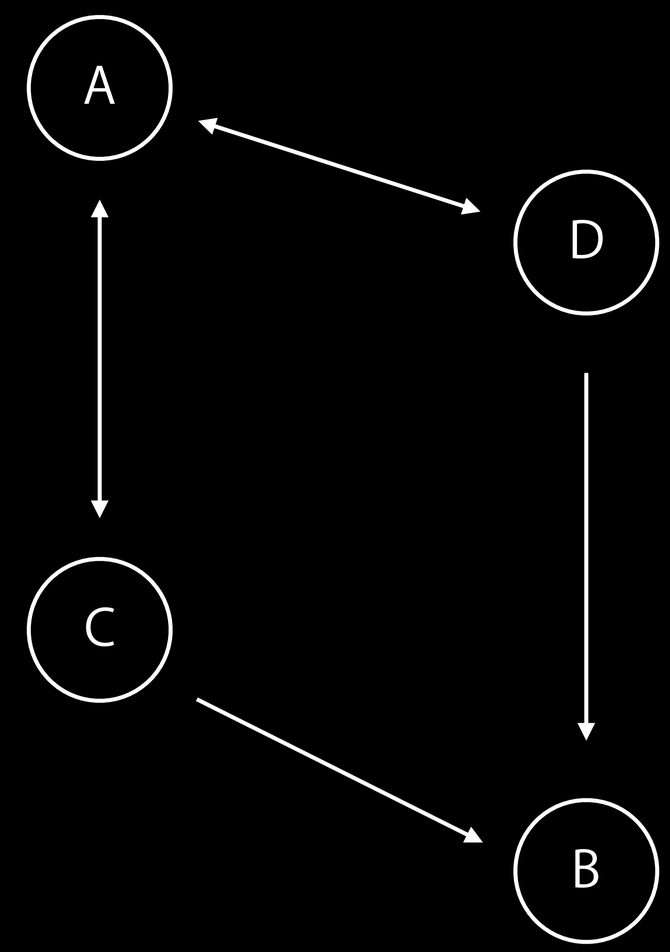
Used in pathfinding

Known as `GKGraph`

Graphs are collections of nodes

Nodes are joined by connections

- Connections are directional

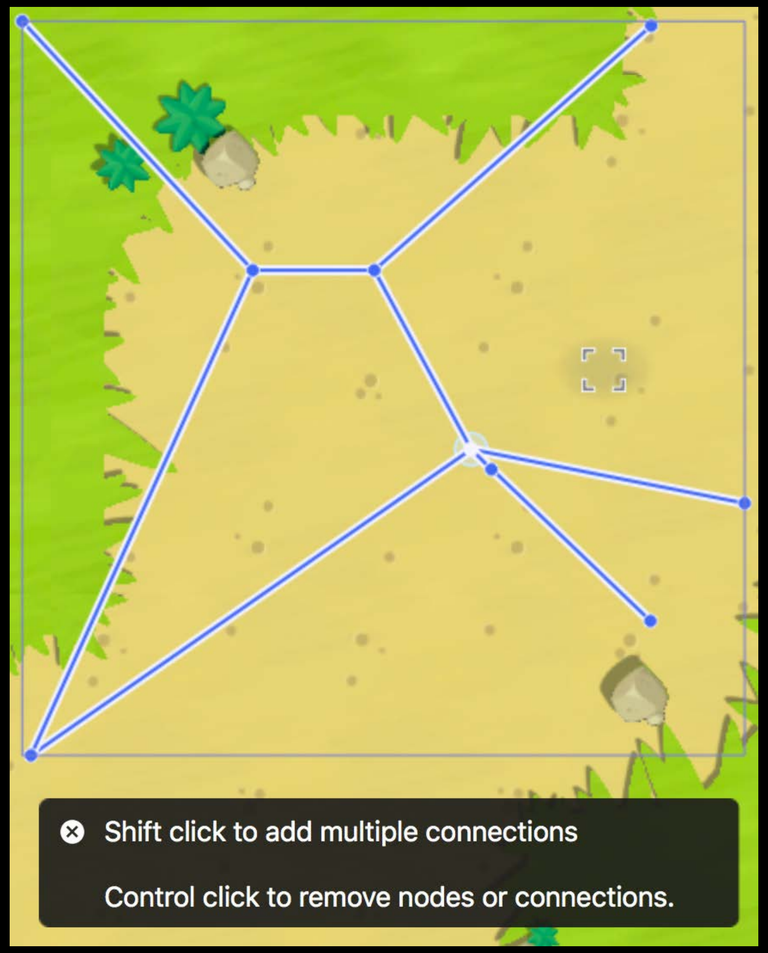


Navigation Graph Editor

Create GKGraphs in Editor

- Add or edit GKGraphs
- Nodes can be added or removed
- New connections can be made or adjusted

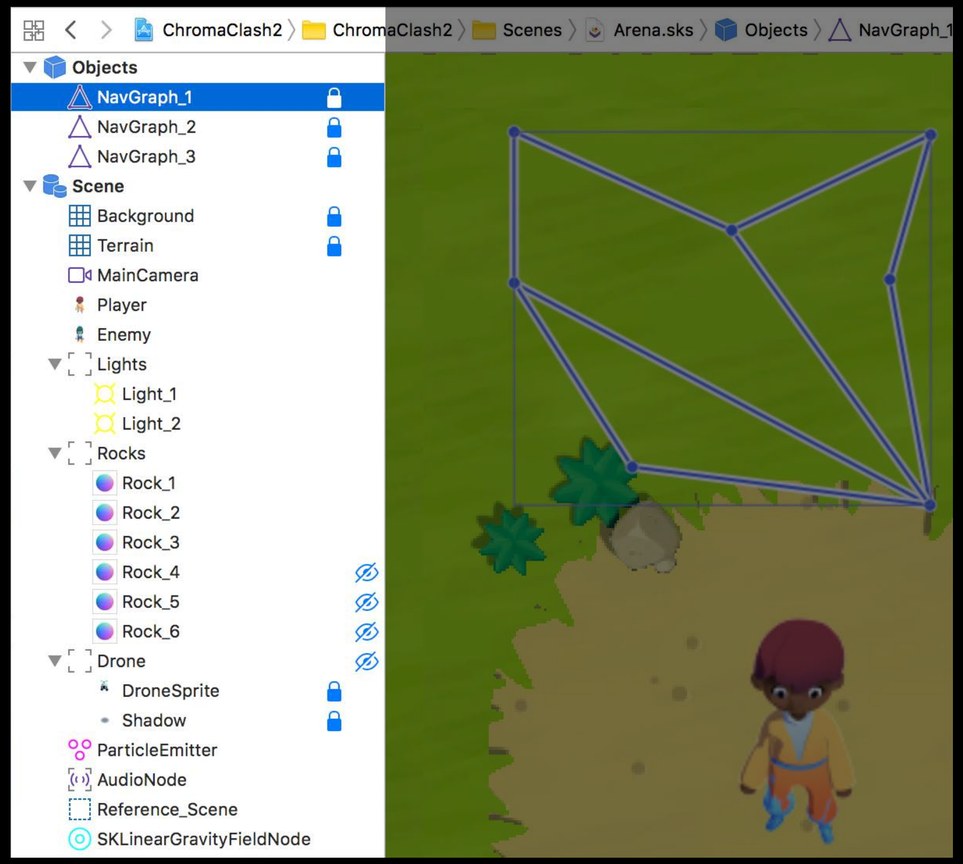
Saved as part of GKScene



Scene Outline View

Displays Scene elements and hierarchy

- Outline view and parent-child relationship
- Lists navigation graphs
- Operations – add, delete, rearrange, lock, visibility
- Other context menu operations



State Machine Quick Look

State Machine Quick Look

GKStateMachine

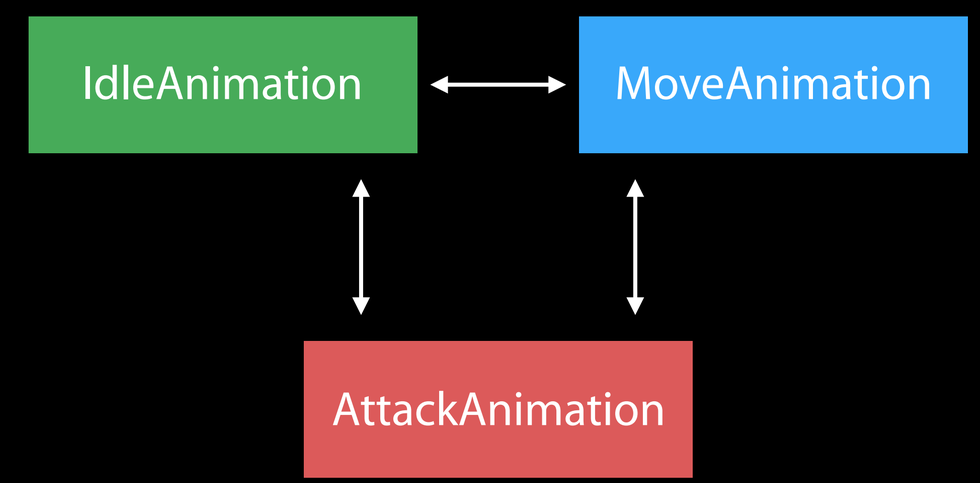
Represents an execution flow

Games use this in various ways:

- Animation, AI, UI, levels and so on

Represented by directed graph

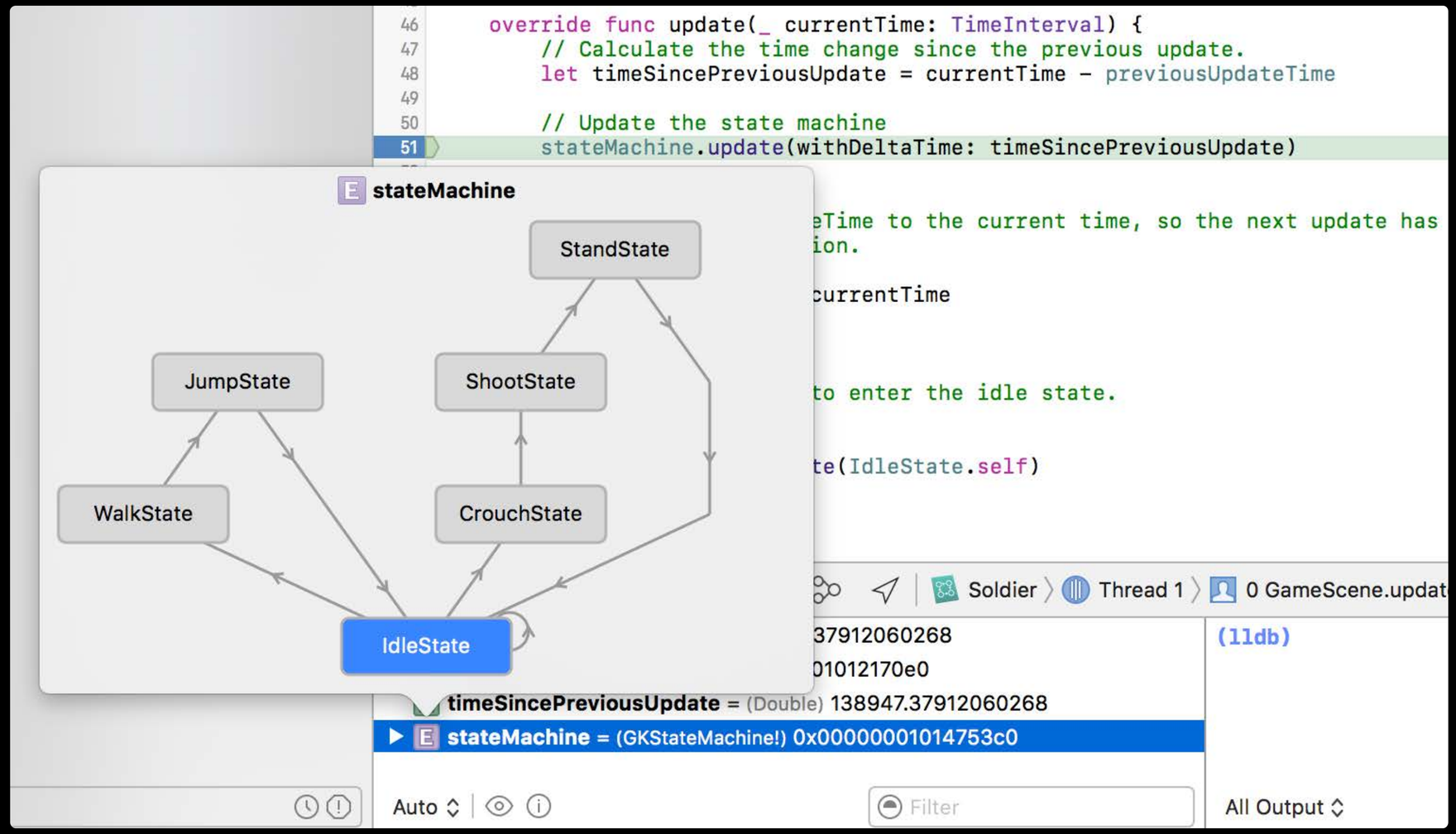
- Nodes = States
- Edges = Transitions



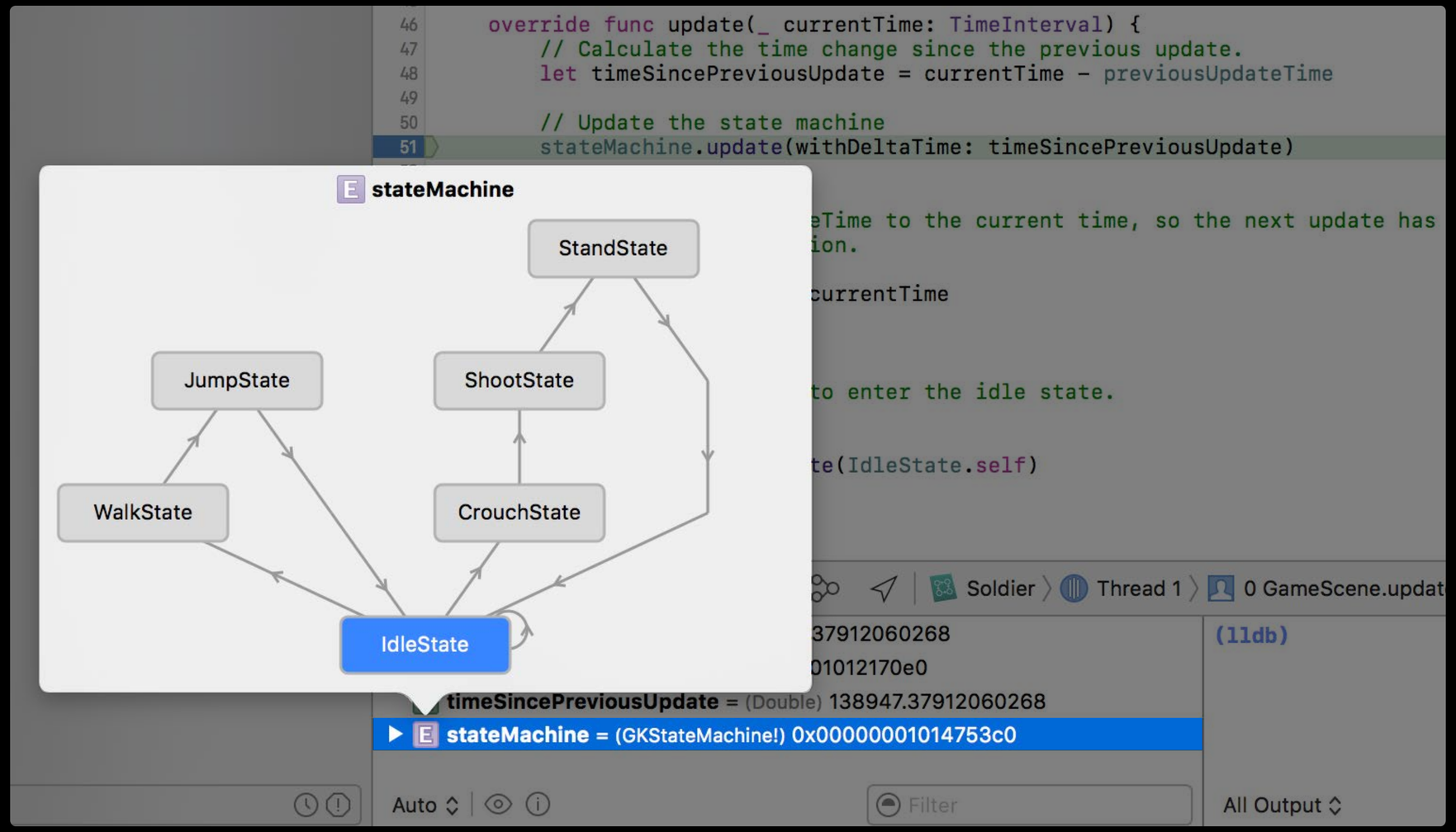
State Machine Quick Look

```
46      override func update(_ currentTime: TimeInterval) {
47          // Calculate the time change since the previous update.
48          let timeSincePreviousUpdate = currentTime - previousUpdateTime
49
50          // Update the state machine
51          stateMachine.update(withDeltaTime: timeSincePreviousUpdate)
52
53          /*
54              Set previousUpdateTime to the current time, so the next update has
55              accurate information.
56          */
57          previousUpdateTime = currentTime
58      }
59
```

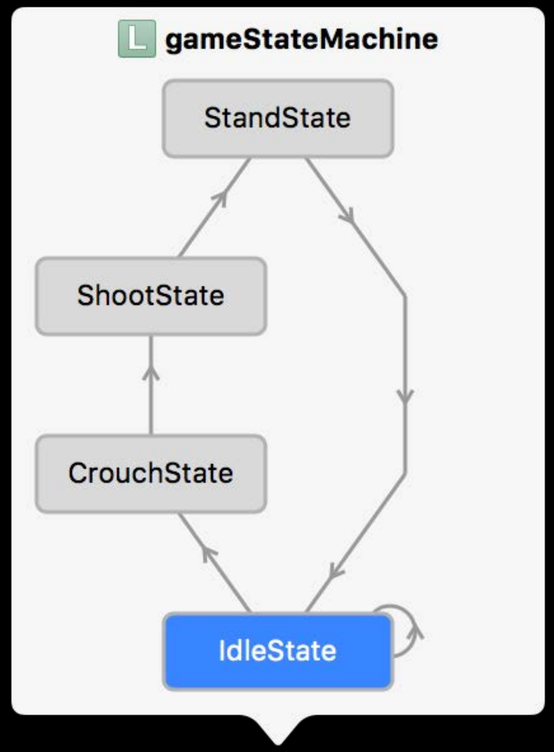
State Machine Quick Look

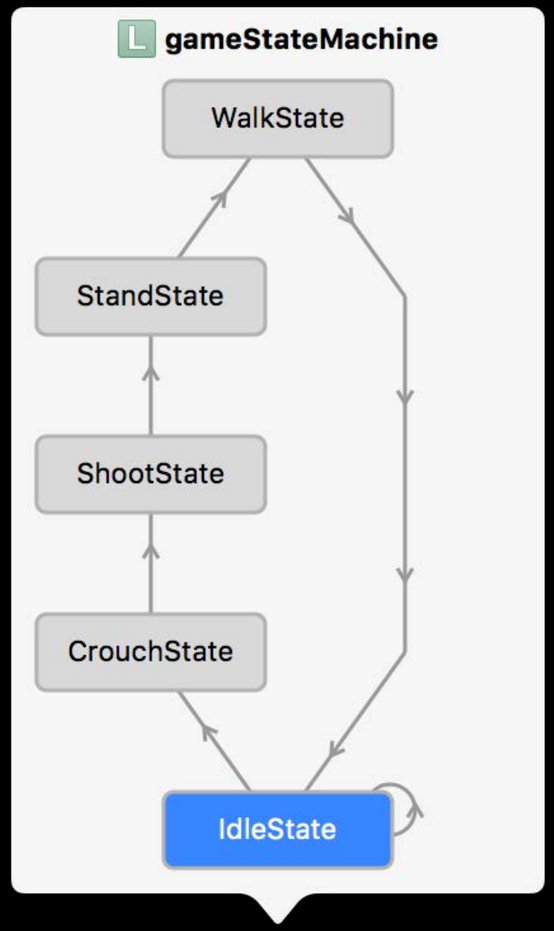


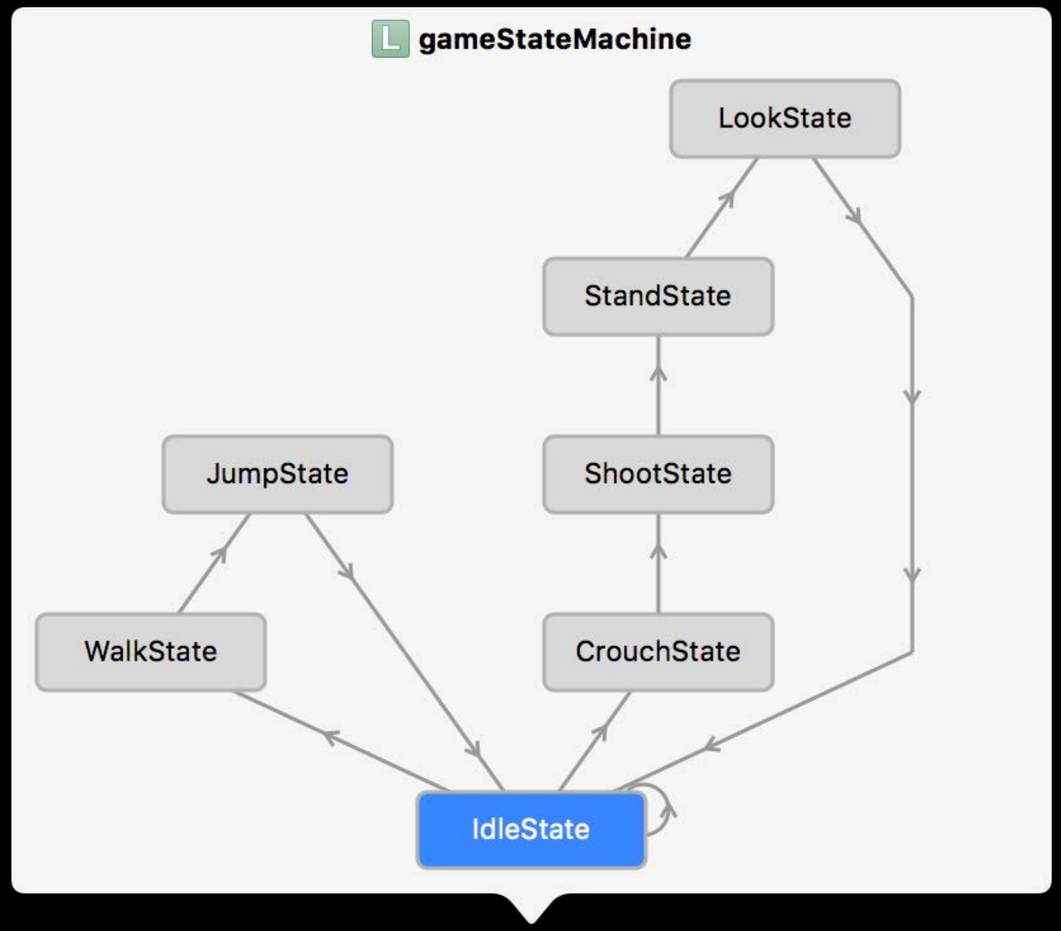
State Machine Quick Look



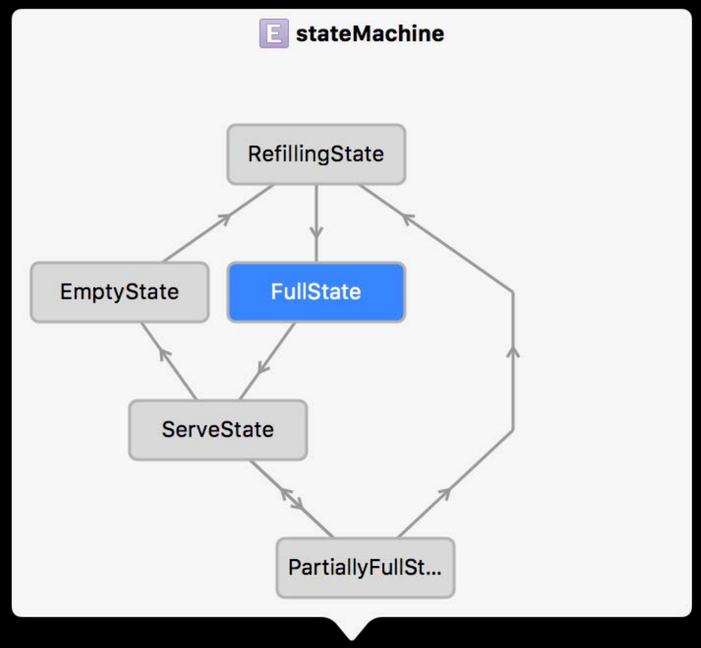
State Machine Quick Look

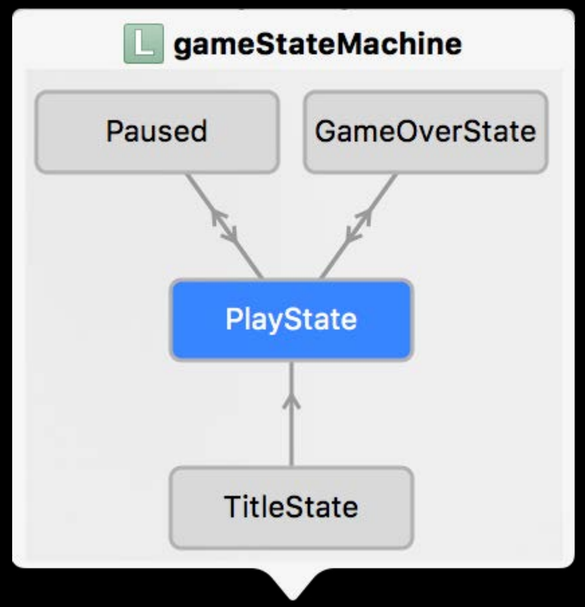


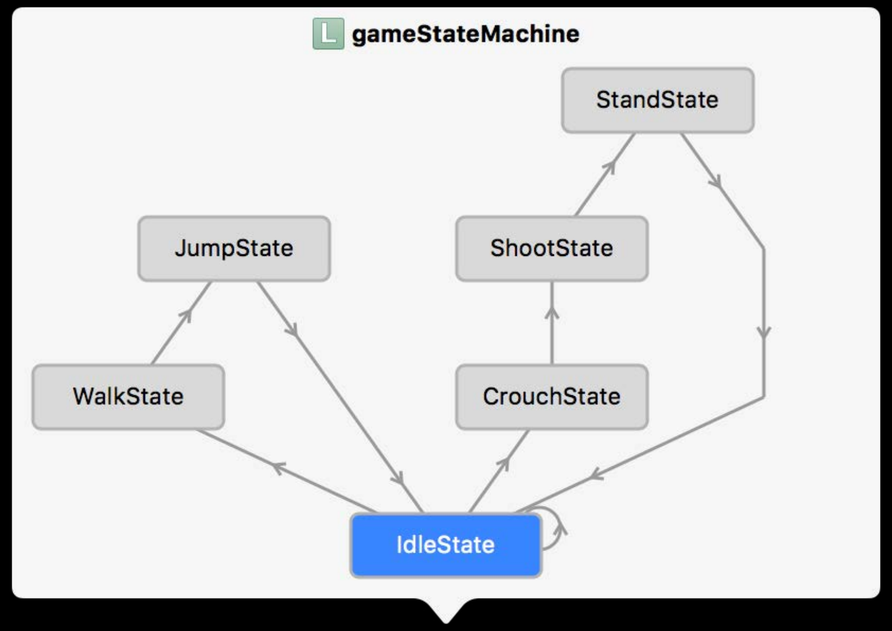




State Machine Quick Look







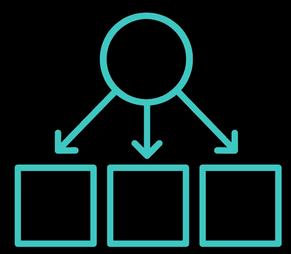
Demo

GameplayKit – Editor based workflow

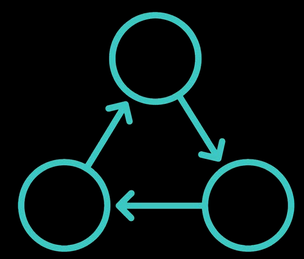
Wrap Up

Summary

GameplayKit – Bringing game ideas to life



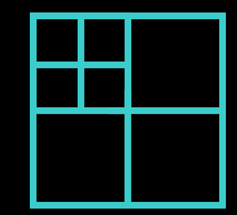
Entities and Components



State Machines

NEW

Agents

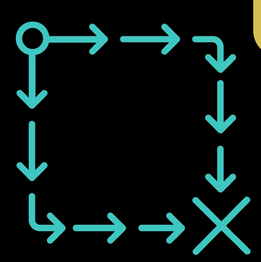


NEW

Spatial Partitioning

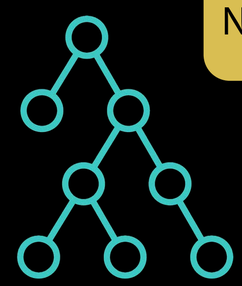
NEW

Procedural Generation



NEW

Pathfinding



NEW

Game AI

Random Sources

Rule Systems

NEW

Xcode Integration

More Information

http://developer.apple.com/wwdc16/608

Related Sessions

Advances in SceneKit Rendering	Mission	Thursday 11:00AM
What's New in SpriteKit	Presidio	Thursday 5:00PM
What's New in Game Center	Mission	Friday 10:00AM
Game Technologies for Apple Watch	Mission	Friday 3:00PM

Labs

GameplayKit Lab	Graphics, Games, and Media Lab B	Thursday 10:00AM
macOS Graphics and Games Lab	Graphics, Games, and Media Lab B	Thursday 12:00PM
Metal Lab	Graphics, Games, and Media Lab A	Thursday 12:00PM
SceneKit Lab	Graphics, Games, and Media Lab A	Thursday 3:00PM
Model I/O Lab	Graphics, Games, and Media Lab B	Thursday 3:00PM
SpriteKit Lab	Graphics, Games, and Media Lab B	Friday 12:00PM
watchOS Graphics and Games Lab	Graphics, Games, and Media Lab B	Friday 4:00PM

WWDC16