What’s New in GameplayKit

Session 608

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

Your gameplay toolbox

Platformer

City Builder

RPG

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
- Agents
- Spatial Partitioning
- Procedural Generation
- Pathfinding
- Game AI
- Random Sources
- Rule Systems
- 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

Triangulates space between obstacles
Good quality paths
Fast and low overhead
Node location flexibility
• Centers, vertices, edges
/ 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
// 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

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
Agents
Agents

Refresher on agents

Autonomous entities controlled by goals
Goals are combined into behaviors
Realistic physical constraints
• Velocity
• Mass
• Obstacle avoidance
• Path following

Agent

Behavior

Seek
Intercept
Avoid
Target Speed
Wander
Cohere
Separate
Align
Flee
Follow Path
Agents

Agents in 3D

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

GKCompositeBehavior

- Subclass of GKBehavior
- Collection of weighted GKBehaviors
- Same relationship as GKBehavior → GKGoal
- Fully nestable
// 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
// 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 = GKAagent2D()
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
// 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 = GKA fixtures
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

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

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

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

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

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

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

GKR Tree

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

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

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

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

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

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

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

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

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

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

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)))

// Add some more enemies

// 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
let tree = GKQuadtree(boundingQuad: GKQuad(quadMin: float2(0.0,0.0),
                              quadMax: float2(1000.0,1000.0), minimumCellSize:100)!

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)
let tree = GKQuadtree(boundingQuad: GKQuad(quadMin: float2(0.0,0.0),
                              quadMax: float2(1000.0,1000.0), minimumCellSize: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
addElement(enemy1, with:GKQuad(quadMin:float2(40.0,40.0), quadMax:float2(50.0,50.0)))
addElement(enemy2, with:GKQuad(quadMin:float2(10.0,20.0), quadMax:float2(20.0,30.0)))
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)
addElements(in: GKQuad(quadMin:float2(0.0,0.0), quadMax:float2(100.0,100.0)))
Procedural Generation
Procedural Generation

Background

Premade Content
- Levels
- Textures
- Characters

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

Noise Sources
- Perlin
- Voronoi
- Billows
- Spheres
- Ridges
- Cylinders
- Checkerboard
- Constant

Transformations

Noise

Samples

Noise Map

Game Content
Procedural Generation
GKNoiseSource

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

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

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

Very Wet

Quite Dry

Extremely Cold

Very Warm
Procedural Generation

Biome example

- Arctic
- Tundra
- Temperate forest
- Savannah
- Desert
- Rain forest

Moisture vs Temperature
Procedural Generation

Biome example

Moisture

Temperature

Biomes
Procedural Generation

Biome example

Moisture
- Wet
- Dry

Temperature
- Cold
- Warm
- Cold

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
Monte Carlo strategist

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

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

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

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

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

Fast performance
Works on large state spaces
Only needs win/loss condition
Approximately optimal
Game AI

Elements

GKMonteCarloStrategist

- Budget
- Exploration parameter
- Game model

GKMonteCarloStrategist

budget
explorationParameter
bestMoveForActivePlayer()
// 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
// 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
// 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
// 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

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

Low overhead for determining action

Serializable

Uses branching for:

- Values
- Predicates
- Random weights
// 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,…]

let answers = ["nearButton?” : true,…]
tree.findActionForAnswers(answers: answers)
Game AI

GKDecisionTree

```swift
// 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)
```
// 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

<table>
<thead>
<tr>
<th>ourVertical</th>
<th>ourHorizontal</th>
<th>actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>404.9131</td>
<td>1396.555</td>
<td>Left</td>
</tr>
<tr>
<td>396.1226</td>
<td>1587.167</td>
<td>Left</td>
</tr>
<tr>
<td>741.8968</td>
<td>160.7707</td>
<td>Up</td>
</tr>
<tr>
<td>503.0585</td>
<td>1550.767</td>
<td>Left</td>
</tr>
<tr>
<td>436.9709</td>
<td>1472.766</td>
<td>Left</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Game AI

GKDecisionTree

<table>
<thead>
<tr>
<th>ourVertical</th>
<th>ourHorizontal</th>
<th>actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>404.9131</td>
<td>1396.555</td>
<td>Left</td>
</tr>
<tr>
<td>396.1226</td>
<td>1587.167</td>
<td>Left</td>
</tr>
<tr>
<td>741.8968</td>
<td>160.7707</td>
<td>Up</td>
</tr>
<tr>
<td>503.0585</td>
<td>1550.767</td>
<td>Left</td>
</tr>
<tr>
<td>436.9709</td>
<td>1472.766</td>
<td>Left</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

ourVertical?

<= 713.03
  <= 319.90
    Up
  > 319.90
    Left

> 713.03
  Left
Xcode Editor Integration

Sri Nair Game Technologies Engineer
Xcode Editor Integration

Background
Xcode Editor Integration

Background

```swift
class MovementComponent: GKComponent {
    var maxSpeed: Float = 6.2
    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)

addComponent(shadowComponent)

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

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

```swift
func initialize() {
    // Initialize Navigation Graph
    let points = [[-10.0, 50.0], [10.0, 50.0], [10.0, -50.0], [-10.0, -50.0], [0.0, 0.0]]
    var nodes = [GraphNode2D]()
    for pt in points {
        let pt = CGPoint(x: pt[0], y: pt[1])
        nodes.append(GraphNode2D(point: pt))
    }
    navGraph.addNodes(nodes)
}

override func update(deltaTime: TimeInterval) {
    // Find the next point to follow on path
    if (followRandomPath) {
        var index = GKRandomSource.sharedRandom().nextInt(upperBound: navGraph.nodes.count)
        while (lastIndexChosen == index) {
            index = GKRandomSource.sharedRandom().nextInt(upperBound: navGraph.nodes.count)
        }
        lastIndexChosen = index
        let targetNode = navGraph.nodes[lastIndexChosen as Int] as! GraphNode2D
        self.path = [targetNode.position]
    }

    // Move the agent based on the current speed
    let direction = Vector2(targetPosition.x - position.x, targetPosition.y - position.y)
    let distance = direction.length
    if distance > 0.0 { // If there is a target position
        let speed = min(maxSpeed, distance / 1.0)
        let directionNormalized = direction.normalized
        let newDirection = directionNormalized * speed
        position = position + newDirection * deltaTime
    }
}
```
Xcode Editor Integration

Background

class MovementComponent: GKComponent {
    var maxSpeed: Float = 5.2
    var friction: 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, offset: FlyingBot.shadowOffset)
addComponent(shadowComponent)

let animationComponent = AnimationComponent(textureSize: FlyingBot.animationSizes, animation: FlyingBot.animation)
addComponent(animationComponent)

let intelligenceComponent = IntelligenceComponent(states: [TaskBotAgentControlledState(entity: self), FlyingBotPreAttackState(entity: self), FlyingBotNextState(entity: self), TaskBotZippedState(entity: self)])
addComponent(intelligenceComponent)

func initialized() {
    // Initialize navigation graph
    let points = [[-10.0, 50.0], [10.0, 50.0], [-10.0, -50.0], [10.0, -50.0]]
    var nodes = [GKGraphNode2D]()
    for point in points {
        var pt = (Float(point[0]), Float(point[1]))
        nodes.append(GKGraphNode2D(position: 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) {
            lastIndexChosen = index
        }
        lastIndexChosen = index
        let targetNode = navGraph.nodes[(lastIndexChosen as! GKGraphNode2D).index]!
        self.path = [targetNode.position]
Xcode Editor Integration

Background
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

- Entity
  - Projectile
  - Tower
  - Archer
Component Editor

Entity and components

- Projectile
  - MoveComponent

- Tower
  - ShootComponent
  - TargetComponent

- Archer
  - MoveComponent
  - ShootComponent
  - TargetComponent
Component Editor
Component Editor
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
}
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 List:
- CollisionComponent
- MovementComponent
- PlayerInputComponent

Movement Component Properties:
- friction: 5
- acceleration: 4.4
- rollSpeed: 3.5
- maxSpeed: 8
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
override func update(_ currentTime: TimeInterval) {
    // Calculate the time change since the previous update.
    let timeSincePreviousUpdate = currentTime - previousUpdateTime

    // Update the state machine
    stateMachine.update(withDeltaTime: timeSincePreviousUpdate)

    /*
    Set previousUpdateTime to the current time, so the next update has
    accurate information.
    */
    previousUpdateTime = currentTime
}

State Machine Quick Look

```kotlin
override fun update(_ currentTime: TimeInterval) {
    // Calculate the time change since the previous update.
    let timeSincePreviousUpdate = currentTime - previousUpdateTime

    // Update the state machine
    stateMachine.update(withDeltaTime: timeSincePreviousUpdate)
}
```

- **State Machine**
- **Idle State**
- **Stand State**
- **Jump State**
- **Shoot State**
- **Walk State**
- **Crouch State**
State Machine Quick Look

```kotlin
override func update(_ currentTime: TimeInterval) {
    // Calculate the time change since the previous update.
    let timeSincePreviousUpdate = currentTime - previousUpdateTime
    // Update the state machine
    stateMachine.update(withDeltaTime: timeSincePreviousUpdate)
}
```

The `update` function is called with the current time. It calculates the time since the previous update and updates the state machine accordingly.

The state machine diagram shows the transitions between states:
- **IdleState**
- **ShootState**
- **JumpState**
- **WalkState**
- **CrouchState**

Each state has arrows pointing to and from other states, indicating the transitions that can occur.

The code snippet shows how the state machine is updated with the time difference since the last update.
State Machine Quick Look
State Machine Quick Look
Demo
GameplayKit – Editor based workflow
Wrap Up
GameplayKit – Bringing game ideas to life

Summary

Entities and Components
State Machines
Agents
Spatial Partitioning
Procedural Generation
Pathfinding
Game AI
Random Sources
Rule Systems
Xcode Integration
More Information

http://developer.apple.com/wwdc16/608
<table>
<thead>
<tr>
<th>Session</th>
<th>Location</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advances in SceneKit Rendering</td>
<td>Mission</td>
<td>Thursday 11:00AM</td>
</tr>
<tr>
<td>What’s New in SpriteKit</td>
<td>Presidio</td>
<td>Thursday 5:00PM</td>
</tr>
<tr>
<td>What’s New in Game Center</td>
<td>Mission</td>
<td>Friday 10:00AM</td>
</tr>
<tr>
<td>Game Technologies for Apple Watch</td>
<td>Mission</td>
<td>Friday 3:00PM</td>
</tr>
<tr>
<td>Lab</td>
<td>Location</td>
<td>Time</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>GameplayKit Lab</td>
<td>Graphics, Games, and Media Lab B</td>
<td>Thursday 10:00AM</td>
</tr>
<tr>
<td>macOS Graphics and Games Lab</td>
<td>Graphics, Games, and Media Lab B</td>
<td>Thursday 12:00PM</td>
</tr>
<tr>
<td>Metal Lab</td>
<td>Graphics, Games, and Media Lab A</td>
<td>Thursday 12:00PM</td>
</tr>
<tr>
<td>SceneKit Lab</td>
<td>Graphics, Games, and Media Lab A</td>
<td>Thursday 3:00PM</td>
</tr>
<tr>
<td>Model I/O Lab</td>
<td>Graphics, Games, and Media Lab B</td>
<td>Thursday 3:00PM</td>
</tr>
<tr>
<td>SpriteKit Lab</td>
<td>Graphics, Games, and Media Lab B</td>
<td>Friday 12:00PM</td>
</tr>
<tr>
<td>watchOS Graphics and Games Lab</td>
<td>Graphics, Games, and Media Lab B</td>
<td>Friday 4:00PM</td>
</tr>
</tbody>
</table>