

# Gems — *of* — Gameplay Kit



*Tobias  
Due  
Munk  
@tobiasdm*

A close-up photograph of a white ceramic cup filled with coffee. The coffee has a layer of frothy milk on top, featuring a delicate latte art design that resembles a heart or a stylized leaf. The cup is positioned on a dark, horizontally-grained wooden table. The background is blurred, showing warm, out-of-focus lights in shades of orange, yellow, and blue, creating a cozy atmosphere.

@sghhydgaeth

# Gems — *of* — Gameplay Kit

**ios**

**tvos**

**macos**

notchOS

**ios**

**tvos**

**macos**

**watchos**

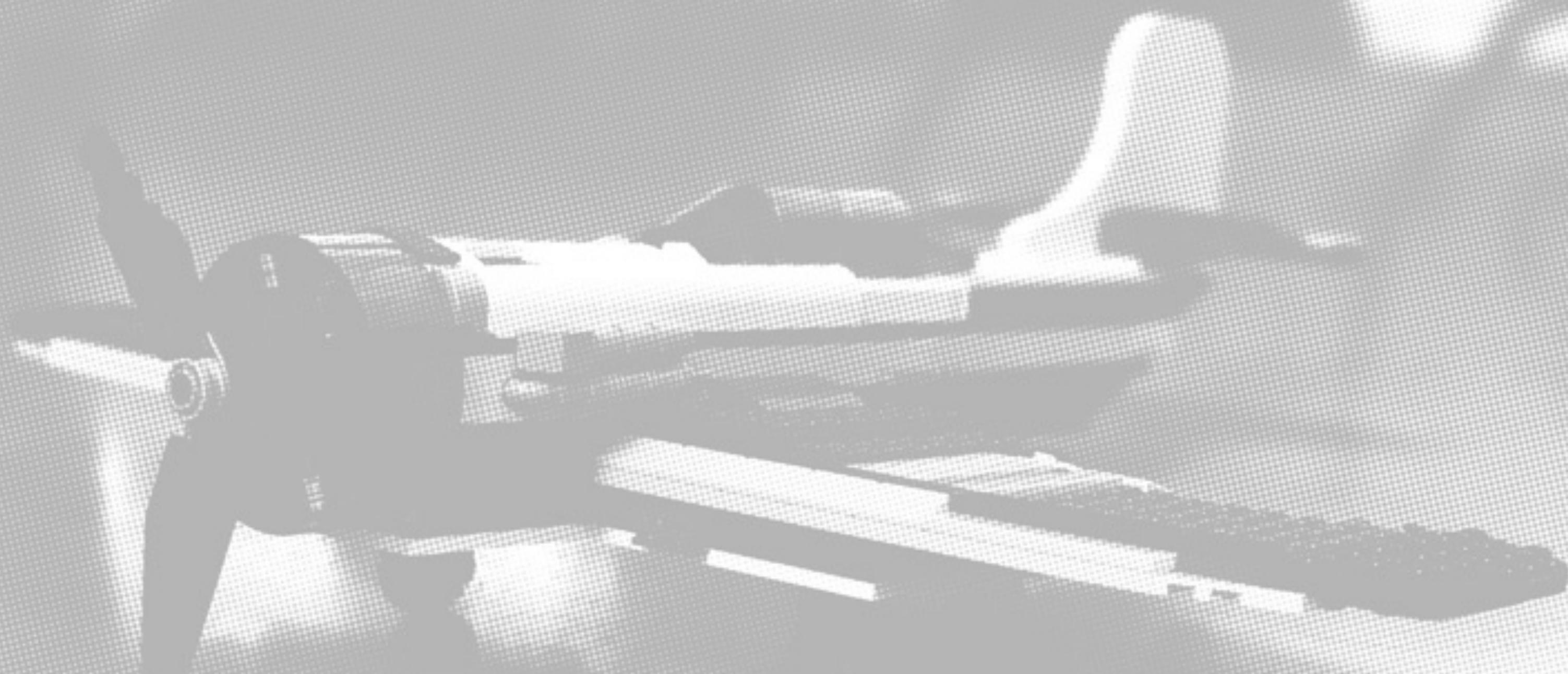
**“Unlike high-level game engines such as SpriteKit and SceneKit, GameplayKit is not involved in animating and rendering visual content. Instead, you ...”**

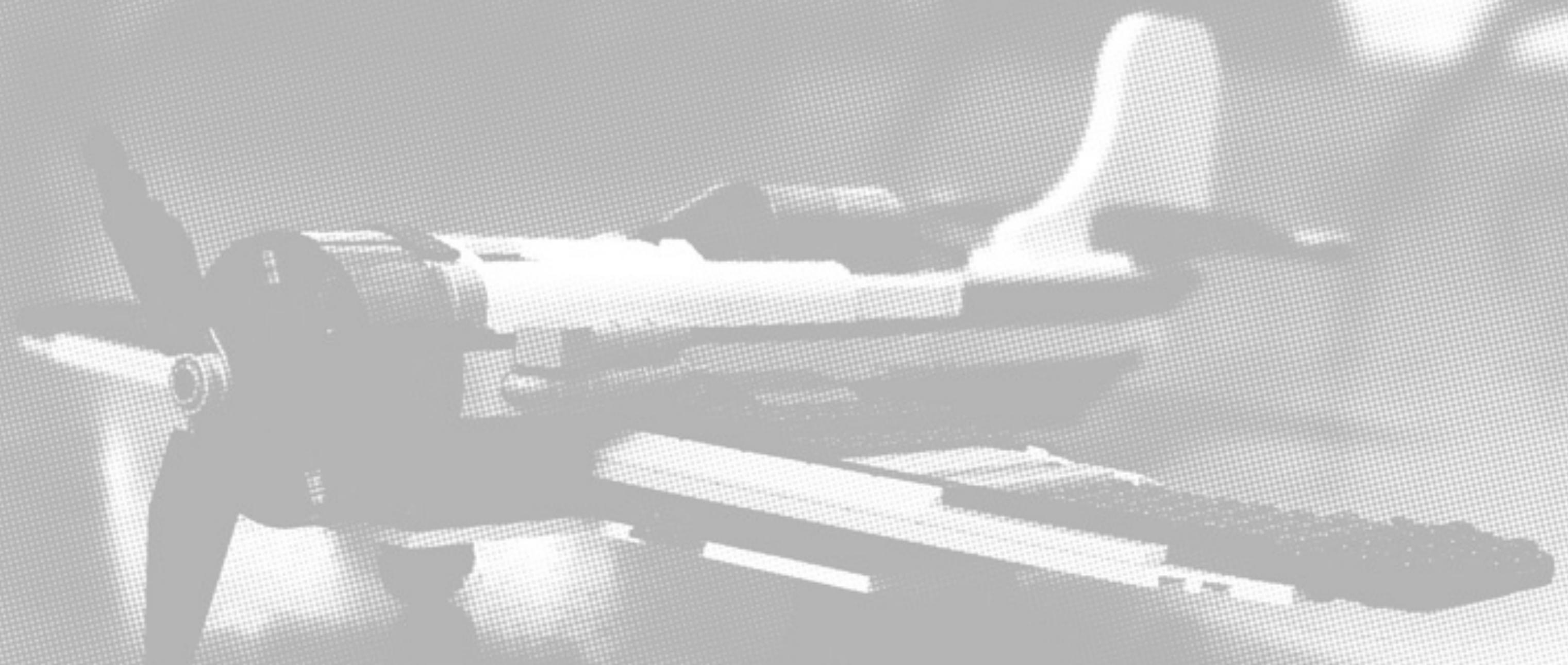
...

**use GameplayKit to develop  
your gameplay mechanics and  
to design modular, scalable  
game architecture with minimal  
effort.”**

...

**use GameplayKit to develop  
your gameplay mechanics and  
to design modular, scalable  
game architecture with minimal  
effort.”**





# shuffled

---

## *arrays*

# Naive

```
extension MutableCollection {
    mutating func shuffle() {
        guard count > 1 else {
            return
        }
        for (firstUnshuffled, unshuffledCount)
            in zip(indices, stride(from: count, to: 1, by: -1))
        {
            let d = Int(arc4random_uniform(Int(unshuffledCount)))
            let i = index(firstUnshuffled, offsetBy: d)
            swapAt(firstUnshuffled, i)
        }
    }
}
```

# Naive

```
extension MutableCollection {
    mutating func shuffle() {
        guard count > 1 else {
            return
        }
        for (firstUnshuffled, unshuffledCount)
            in zip(indices, stride(from: count, to: 1, by: -1))
        {
            let d = Int(arc4random_uniform(Int(unshuffledCount)))
            let i = index(firstUnshuffled, offsetBy: d)
            swapAt(firstUnshuffled, i)
        }
    }
}
```

# Naive

```
extension Sequence {  
    func shuffled() -> [Element] {  
        var result = Array(self)  
        result.shuffle()  
        return result  
    }  
}
```

# Naive

```
[0, 1, 2, 3].shuffled()
```

```
// [2, 1, 0, 3]
```

# Gem

```
import GameplayKit
```

# Gem

```
import GameplayKit  
let source = GKARC4RandomSource.sharedRandom()
```

# Gem

```
import GameplayKit  
  
let source = GKRandomSource.sharedRandom()  
  
source.arrayByShufflingObjects(  
    in: [0, 1, 2, 3]  
)
```

# Gem

```
import GameplayKit
```

```
let source = GKRandomSource.sharedRandom()  
([0, 1, 2, 3] as NSArray)  
.shuffled(using: source)
```

# Gem

```
import GameplayKit

extension Array {

    func shuffled(
        using source: GKRandomSource = .sharedRandom()
    ) -> [Element] {
        let nsArray = self as NSArray
        let shuffled = nsArray.shuffled(using: source)
        return shuffled as! [Element]
    }
}
```

# Gem

```
import GameplayKit

extension Array {

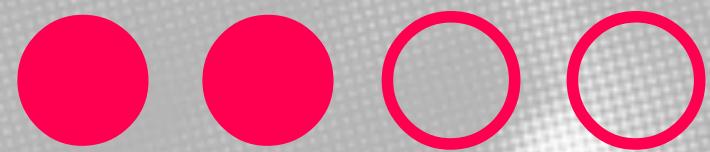
    func shuffled(
        using source: GKRandomSource = .sharedRandom()
    ) -> [Element] {
        let nsArray = self as NSArray
        let shuffled = nsArray.shuffled(using: source)
        return shuffled as! [Element]
    }
}
```

# Gem

```
[0, 1, 2, 3].shuffled()
```

```
// [2, 0, 1, 3]
```

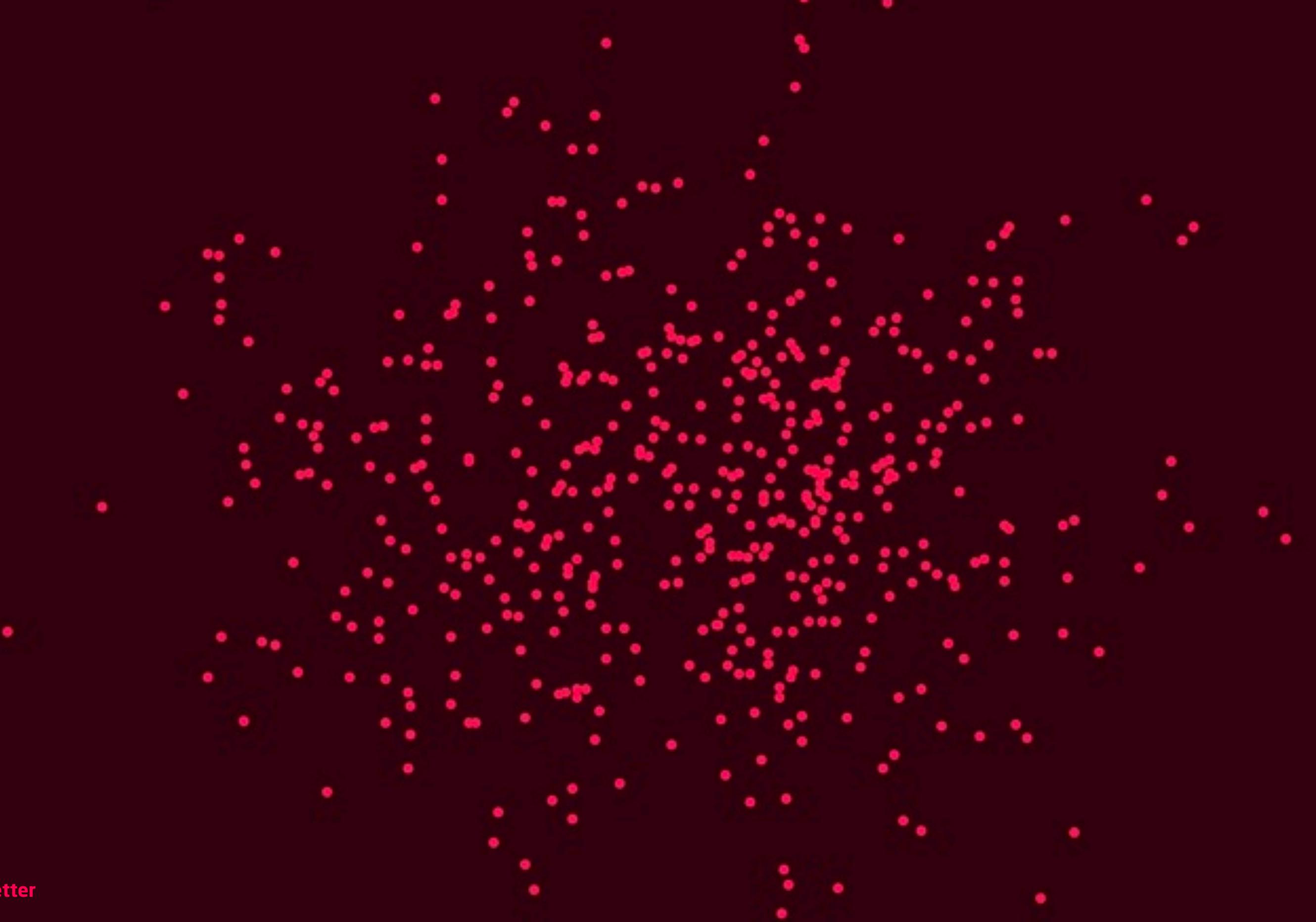




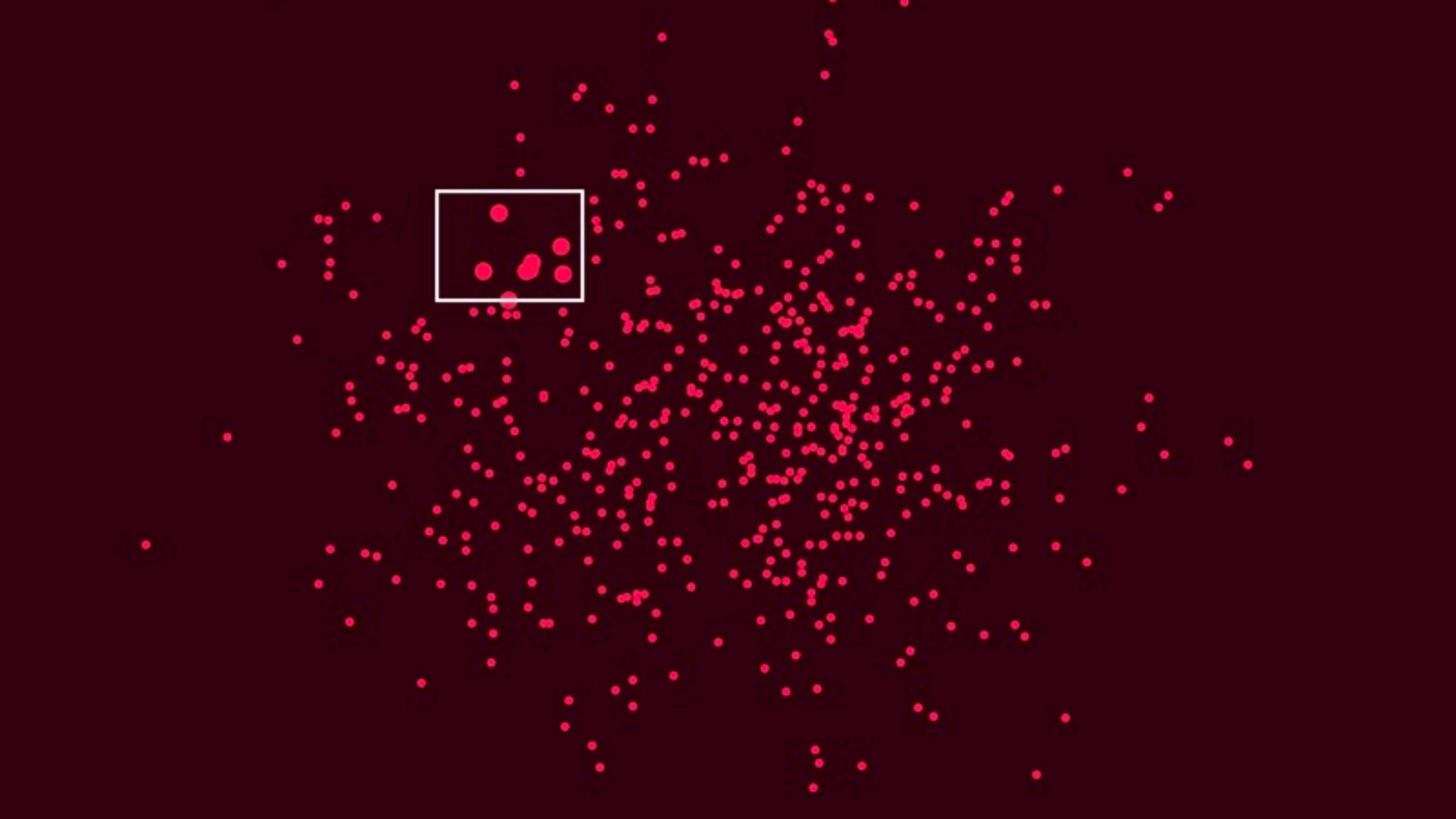
Performant

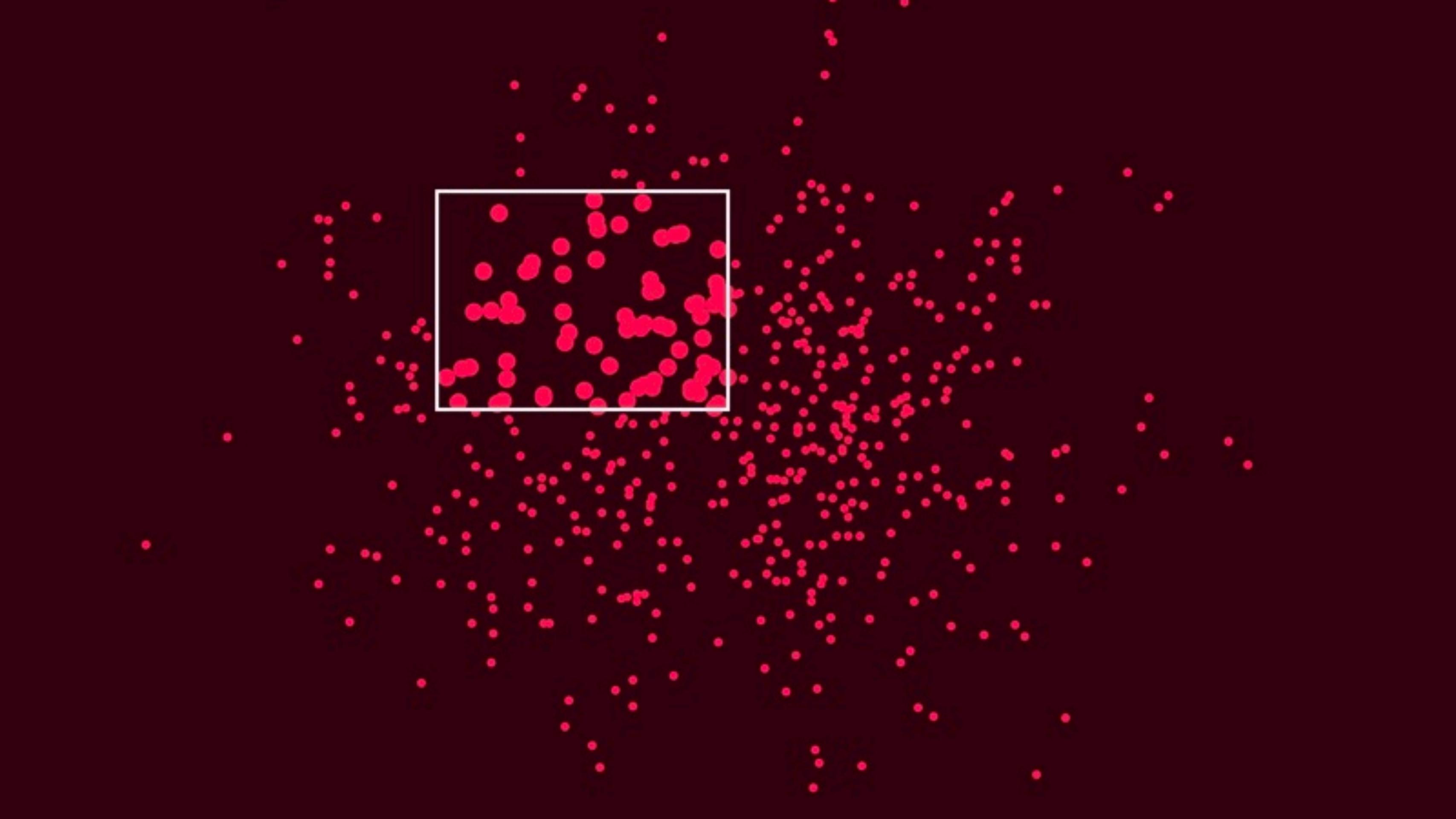
visual

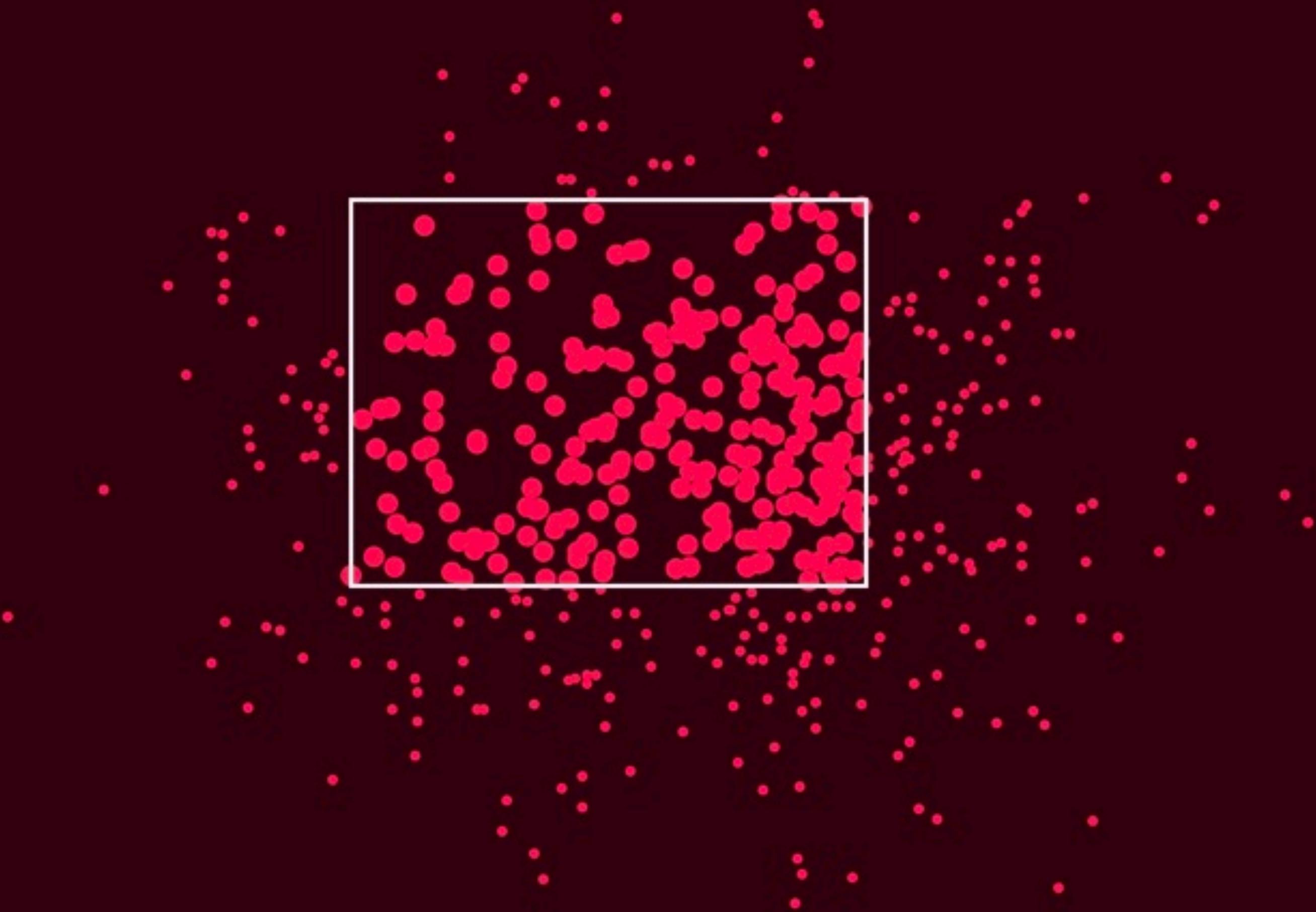
search



Matthias Tretter







# Naive

```
let dots = [  
    CGPoint(x: 0, y: 0),  
    CGPoint(x: 1, y: 1),  
    CGPoint(x: 2, y: 0)  
]
```

# Naive

```
let dots = [  
    CGPoint(x: 0, y: 0),  
    CGPoint(x: 1, y: 1),  
    CGPoint(x: 2, y: 0)  
]  
  
let selection = CGRect(  
    origin: .zero,  
    size: CGSize(width: 1, height: 1)  
)
```

# Naive

```
let dots = [  
    CGPoint(x: 0, y: 0),  
    CGPoint(x: 1, y: 1),  
    CGPoint(x: 2, y: 0)  
]  
  
let selection = CGRect(  
    origin: .zero,  
    size: CGSize(width: 1, height: 1)  
)
```

# Naive

```
dots.filter { dot in  
    return selection.contains(dot)  
}
```

# Gem

```
import GameplayKit
```

# Gem

```
import GameplayKit  
  
let tree = GKRTree<Point>(maxNumberOfChildren: 10)
```

# Gem

```
import GameplayKit

class Point: NSObject {
    let x: CGFloat
    let y: CGFloat
}
```

# Gem

```
import GameplayKit

let tree = GKRTree<Point>(maxNumberOfChildren: 10)
for point in points {
    let vector = vector_float2(
        x: Float(point.x),
        y: Float(point.y))
    tree.addElement(
        point,
        boundingRectMin: vector,
        boundingRectMax: vector,
    )
}
```

# Gem

```
import GameplayKit

let tree = GKRTree<Point>(maxNumberOfChildren: 10)
for point in points {
    let vector = vector_float2(
        x: Float(point.x),
        y: Float(point.y))
    tree.addElement(
        point,
        boundingRectMin: vector,
        boundingRectMax: vector,
    )
}
```

# Gem

```
import GameplayKit

let tree = GKRTree<Point>(maxNumberOfChildren: 10)
for point in points {
    let vector = vector_float2(
        x: Float(point.x),
        y: Float(point.y))
}
tree.addElement(
    point,
    boundingRectMin: vector,
    boundingRectMax: vector
)
}
```

# Gem

```
import GameplayKit

let tree = GKRTree<Point>(maxNumberOfChildren: 10)
for point in points {
    let vector = vector_float2(
        x: Float(point.x),
        y: Float(point.y))
    tree.addElement(
        point,
        boundingRectMin: vector,
        boundingRectMax: vector,
        splitStrategy: .reduceOverlap
    )
}
```

# Gem

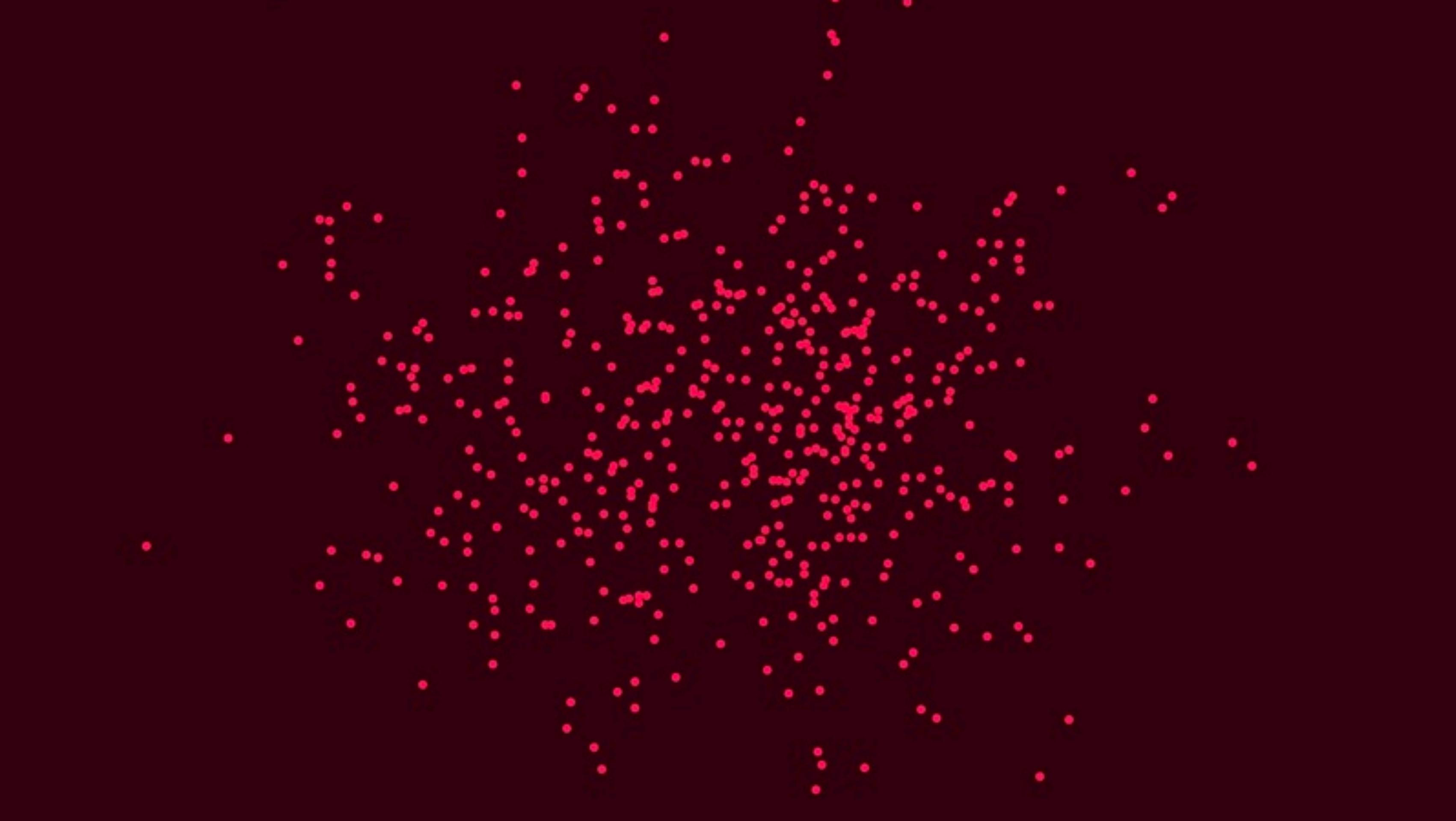
```
let selectionMin = vector_float2(  
    x: selection minX,  
    y: selection minY  
)  
let selectionMax = vector_float2(  
    x: selection maxX,  
    y: selection maxY  
)
```

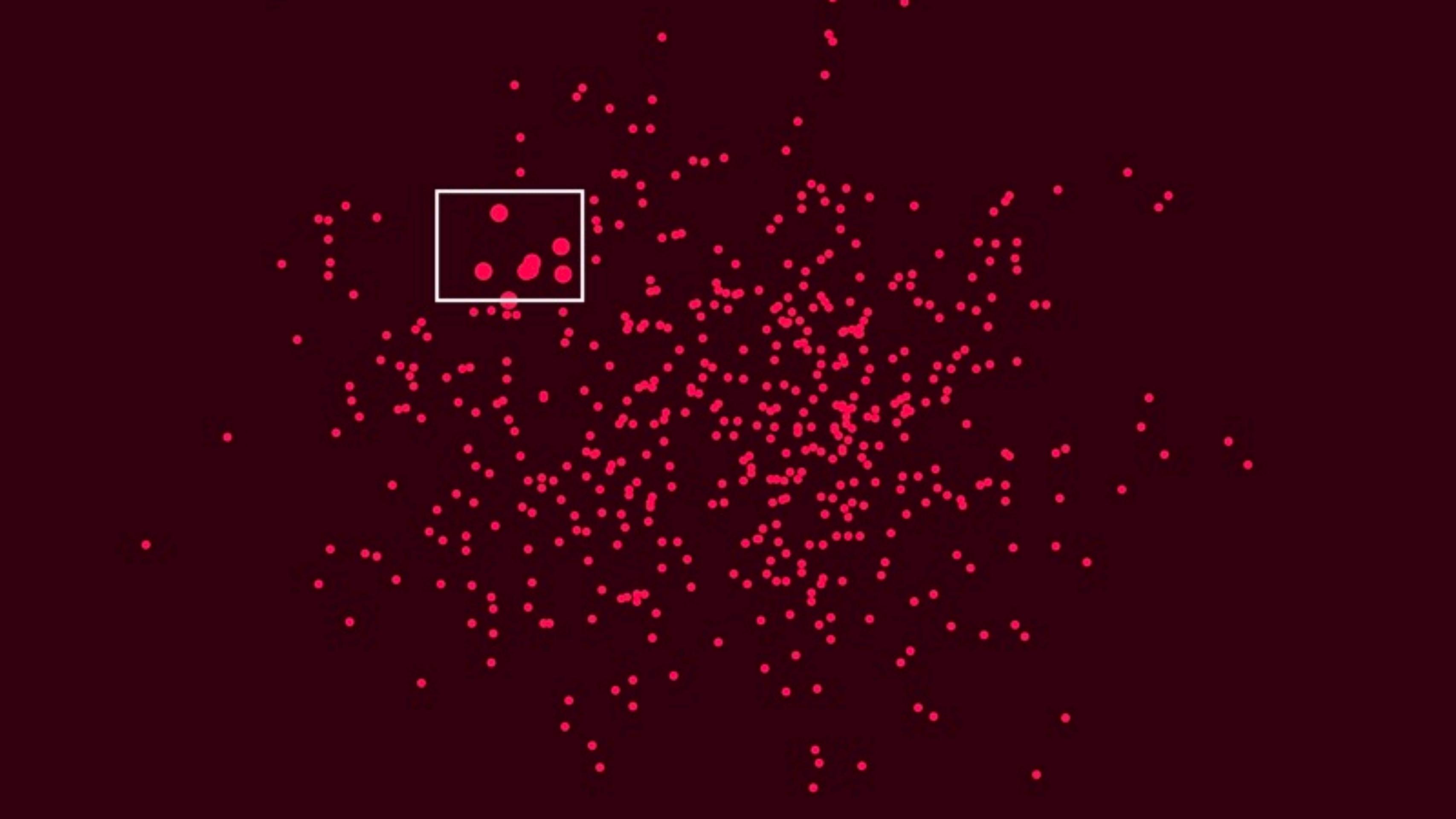
# Gem

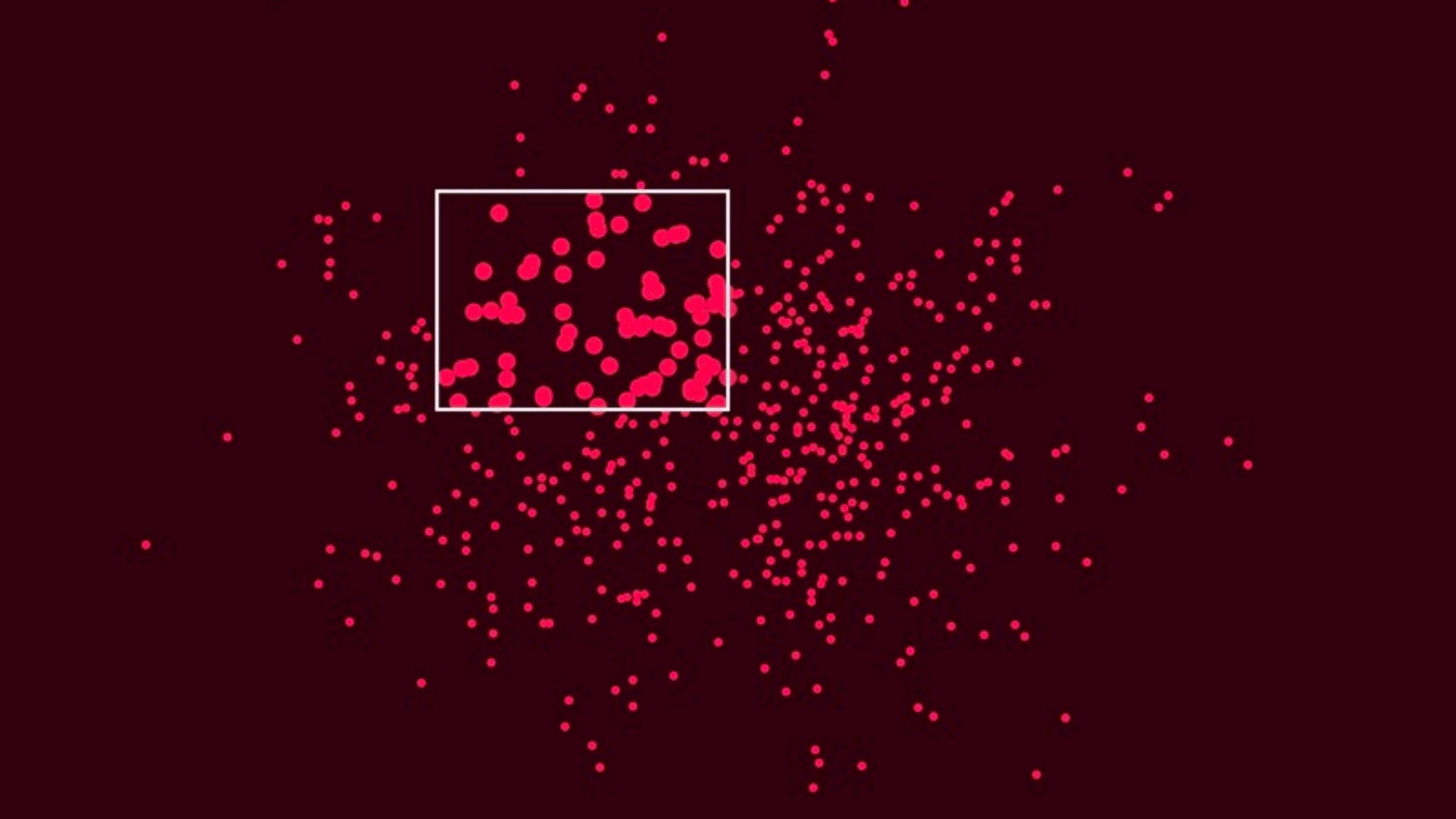
```
let selectionMin = vector_float2(  
    x: selection minX,  
    y: selection minY  
)  
let selectionMax = vector_float2(  
    x: selection maxX,  
    y: selection maxY  
)  
let selectedDots = tree.elements(  
    inBoundingRectMin: selectionMin,  
    rectMax: selectionMax  
)
```

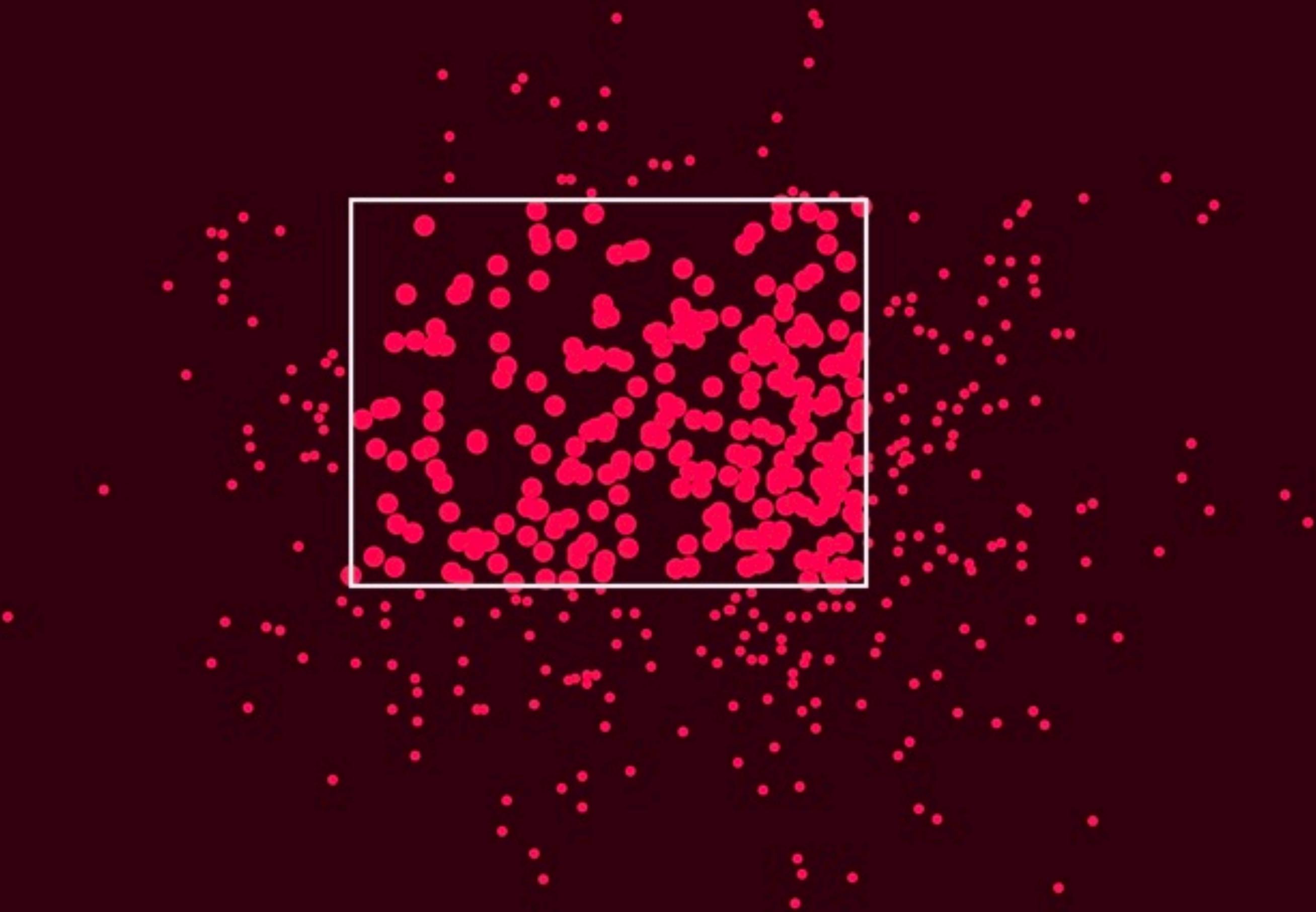
# Gem

```
import GameplayKit  
  
let tree = GKRTree<Point>(maxNumberOfChildren: 10)  
tree.queryReserve = 100
```













*natural*

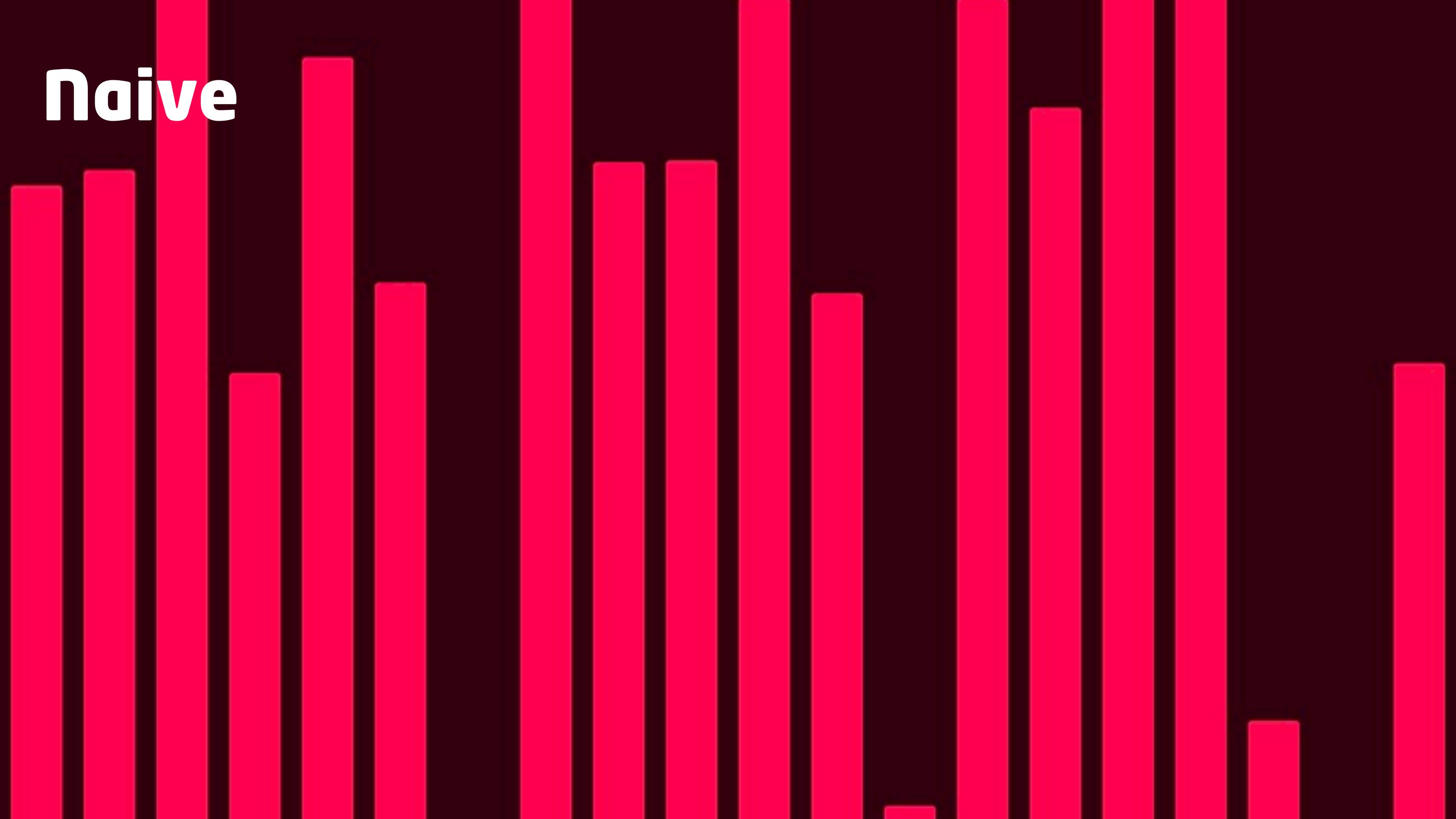
**Randomness**

---

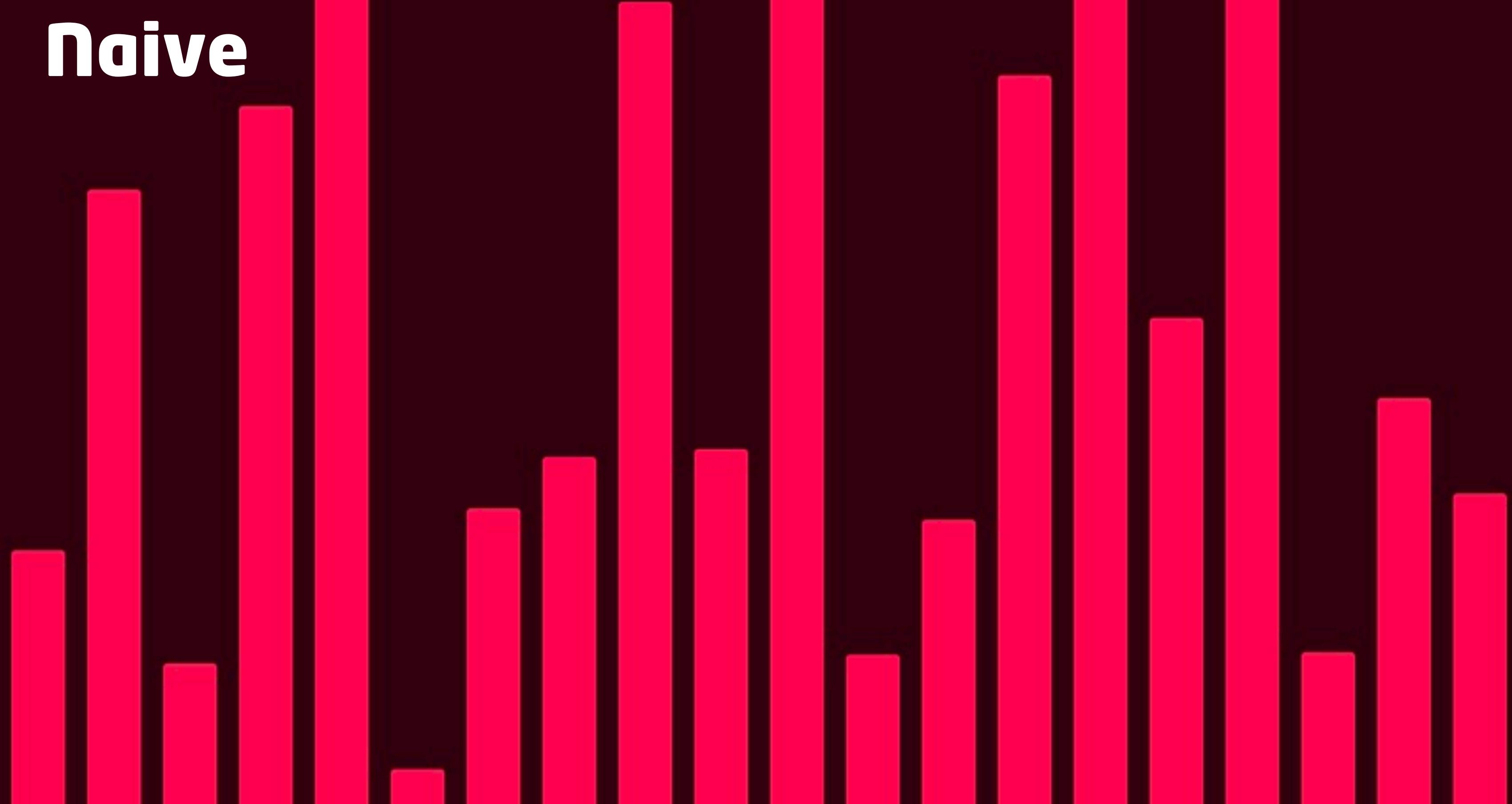
# Naive

**CGFloat(arc4random() ) / CGFloat(UINT32\_MAX)**

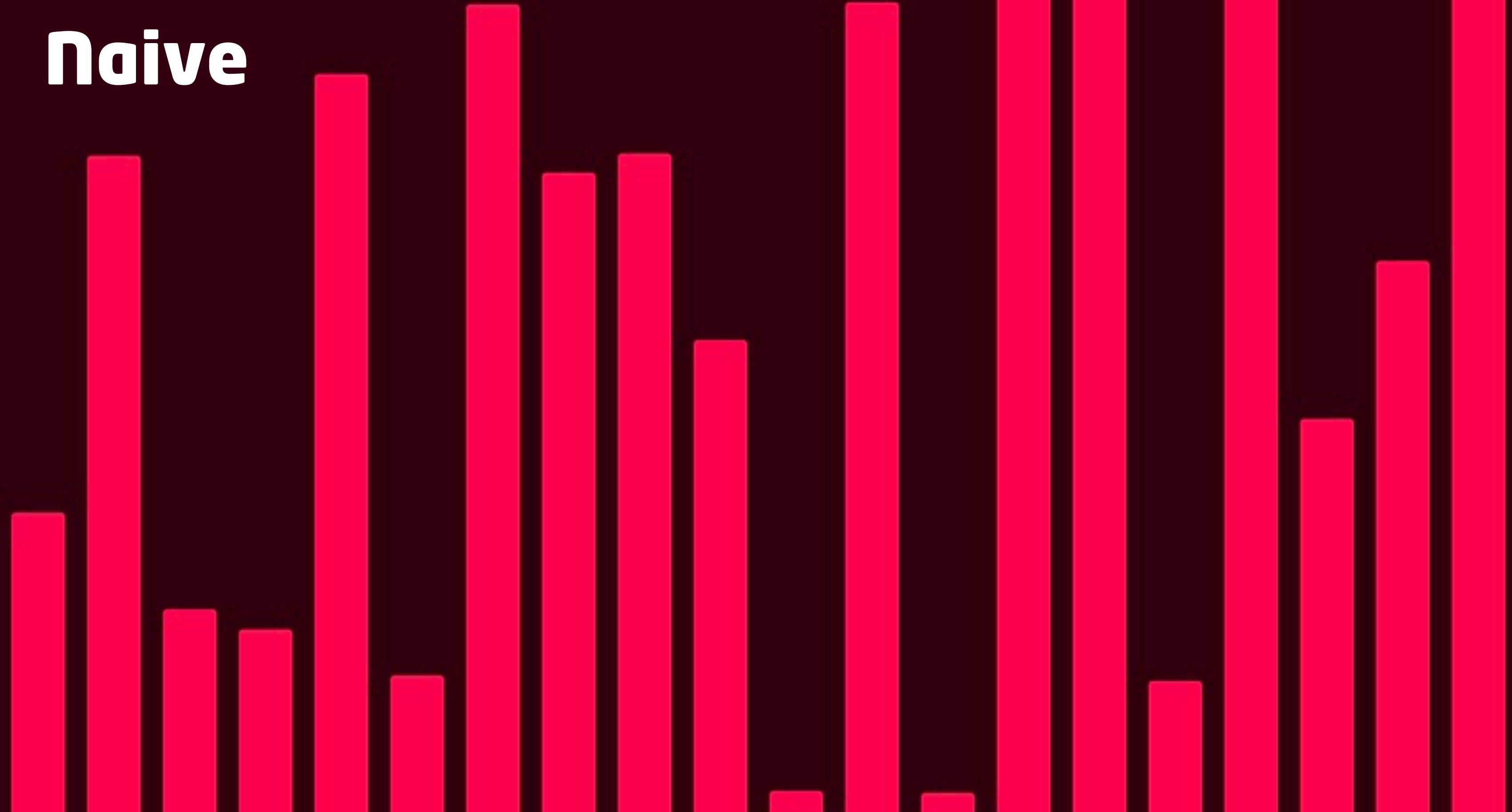
# Naive



# Naive



# Naive



# Gem

```
import GameplayKit
```

# Gem

```
import GameplayKit  
  
let source = GKPerlinNoiseSource(  
    frequency: 2,  
    octaveCount: 3,  
    persistence: 0.5,  
    lacunarity: 2  
)
```

# Gem

```
import GameplayKit

let source = GKPerlinNoiseSource(
    frequency: 2,
    octaveCount: 3,
    persistence: 0.5,
    lacunarity: 2
)
```

# Gem

```
let noise = GKNoise(source)
```

# Gem

```
let map = GKNoiseMap(  
    noise,  
    size: vector2(1, 1),  
    origin: vector2(0, 0),  
    sampleCount: vector2(3, 5),  
    seamless: true  
)
```

# Gem

**map.value(at: vector2(0, 0))**

# Gem

map.value(at: vector2(0, 0))

map.value(at: vector2(1, 0))

map.value(at: vector2(2, 0))

# Gem

map.value(at: vector2(0, 0))

map.value(at: vector2(1, 0))

map.value(at: vector2(2, 0))

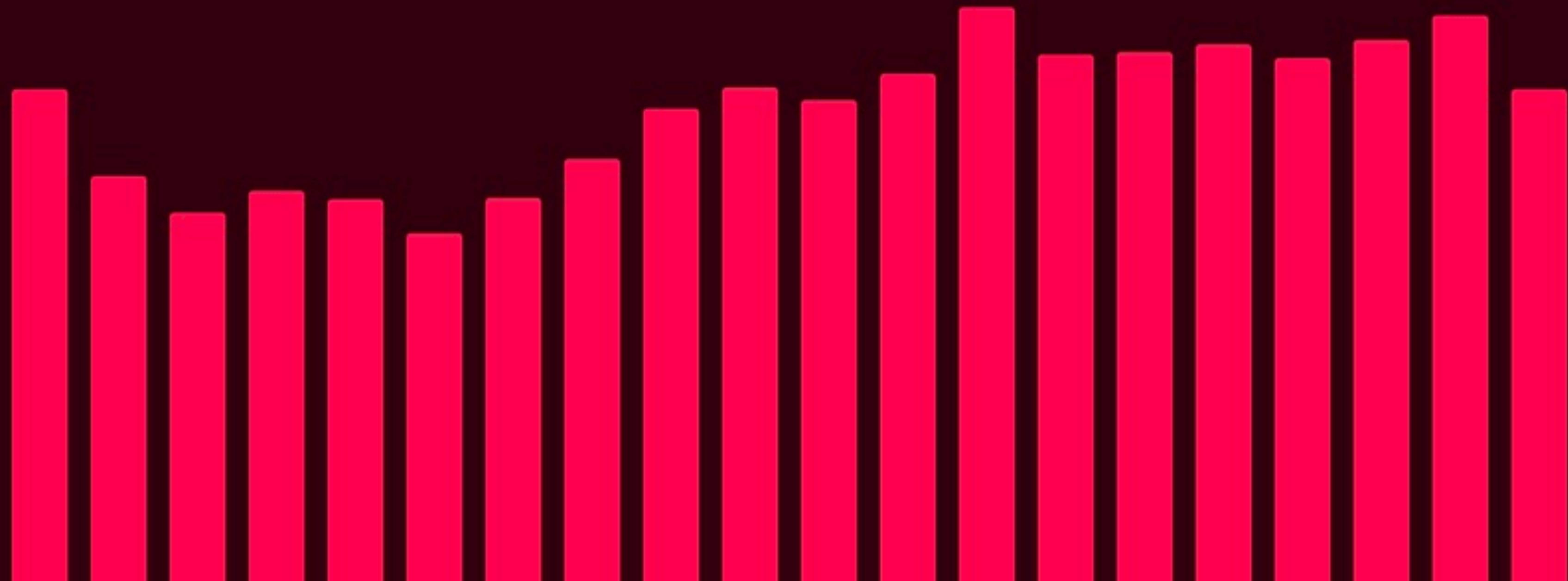
map.value(at: vector2(0, 1))

map.value(at: vector2(0, 2))

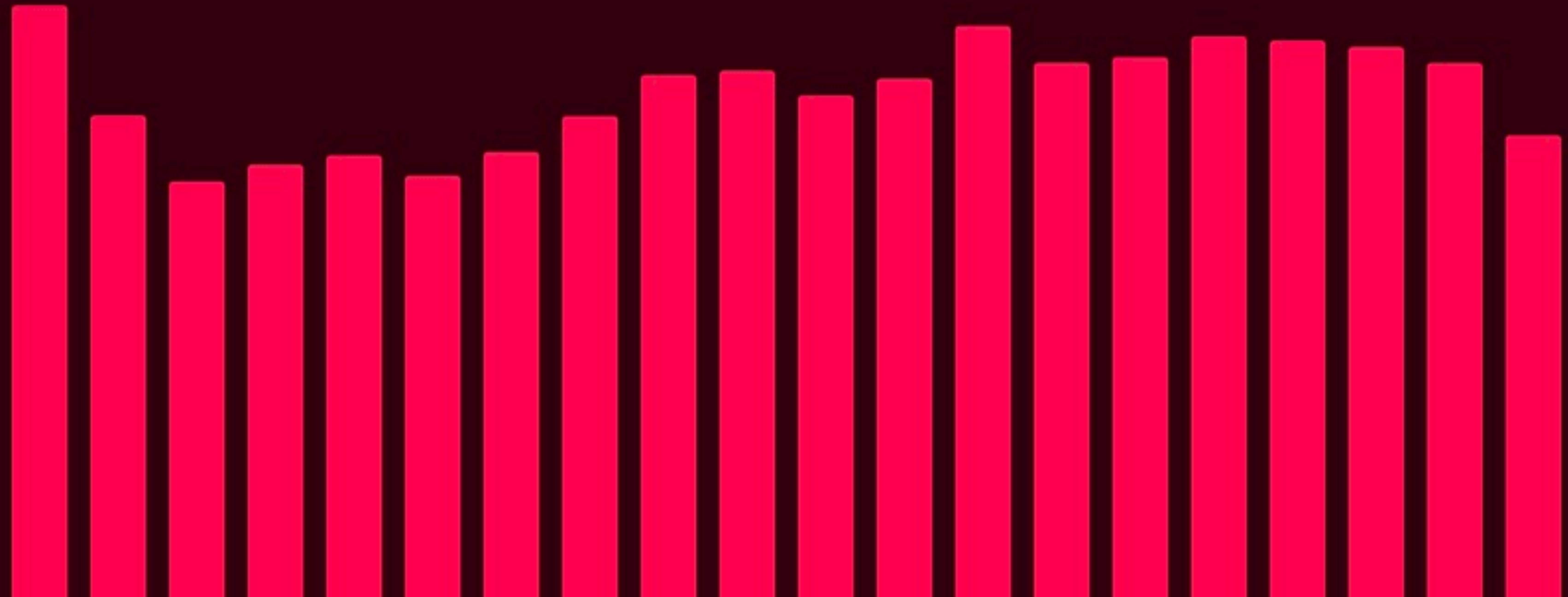
map.value(at: vector2(0, 3))

map.value(at: vector2(0, 4))

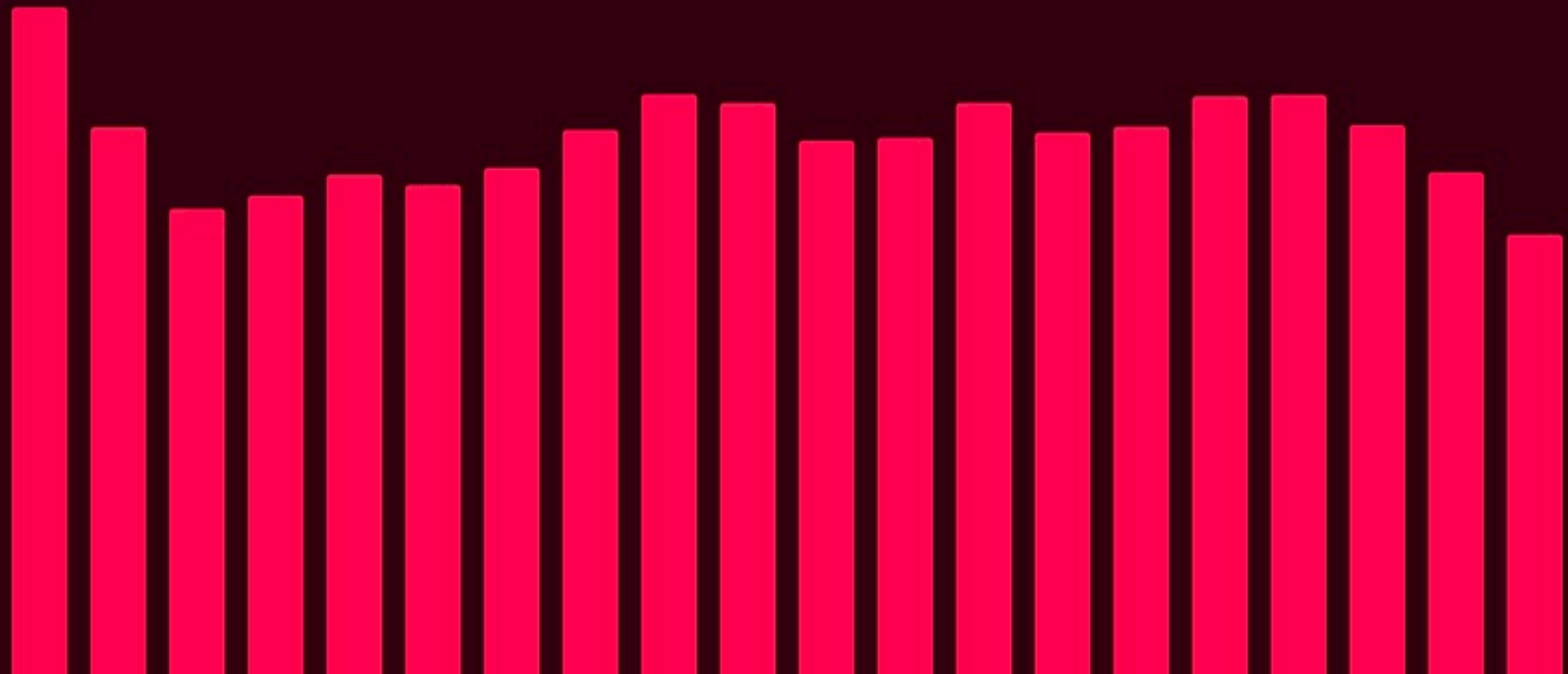
# Gem

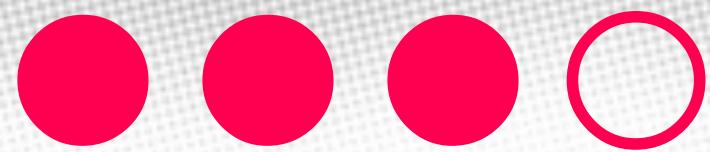


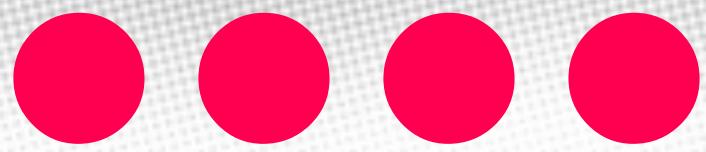
# Gem



# Gem







Path

---

*finding*



# Tivoli Gardens, Copenhagen



# Obstacles



Open Street Map data

# Obstacles

```
let obstacle =
```

```
GKPolygonObstacle(
```

```
    points: [
```

```
        float2(0, 0),
```

```
        float2(0, 2),
```

```
        float2(1, 2),
```

```
        float2(1, 0)
```

```
    ]
```

```
)
```



# Counterclockwise<sup>1</sup>

```
let obstacle =
```

```
GKPolygonObstacle(
```

```
    points: [
```

```
        float2(0, 0),
```

```
        float2(1, 0),
```

```
        float2(1, 2),
```

```
        float2(0, 2)
```

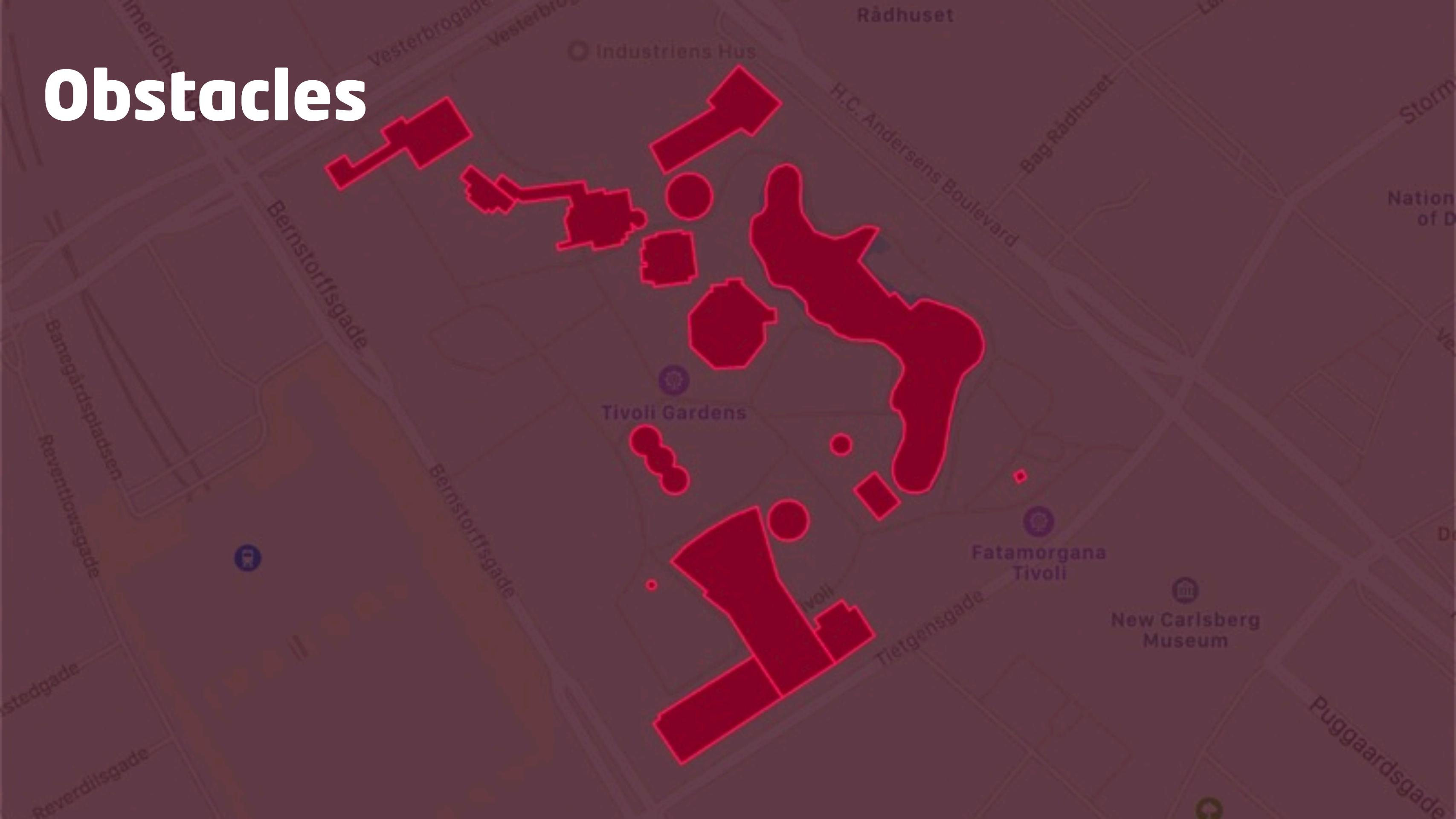
```
    ]
```

```
)
```

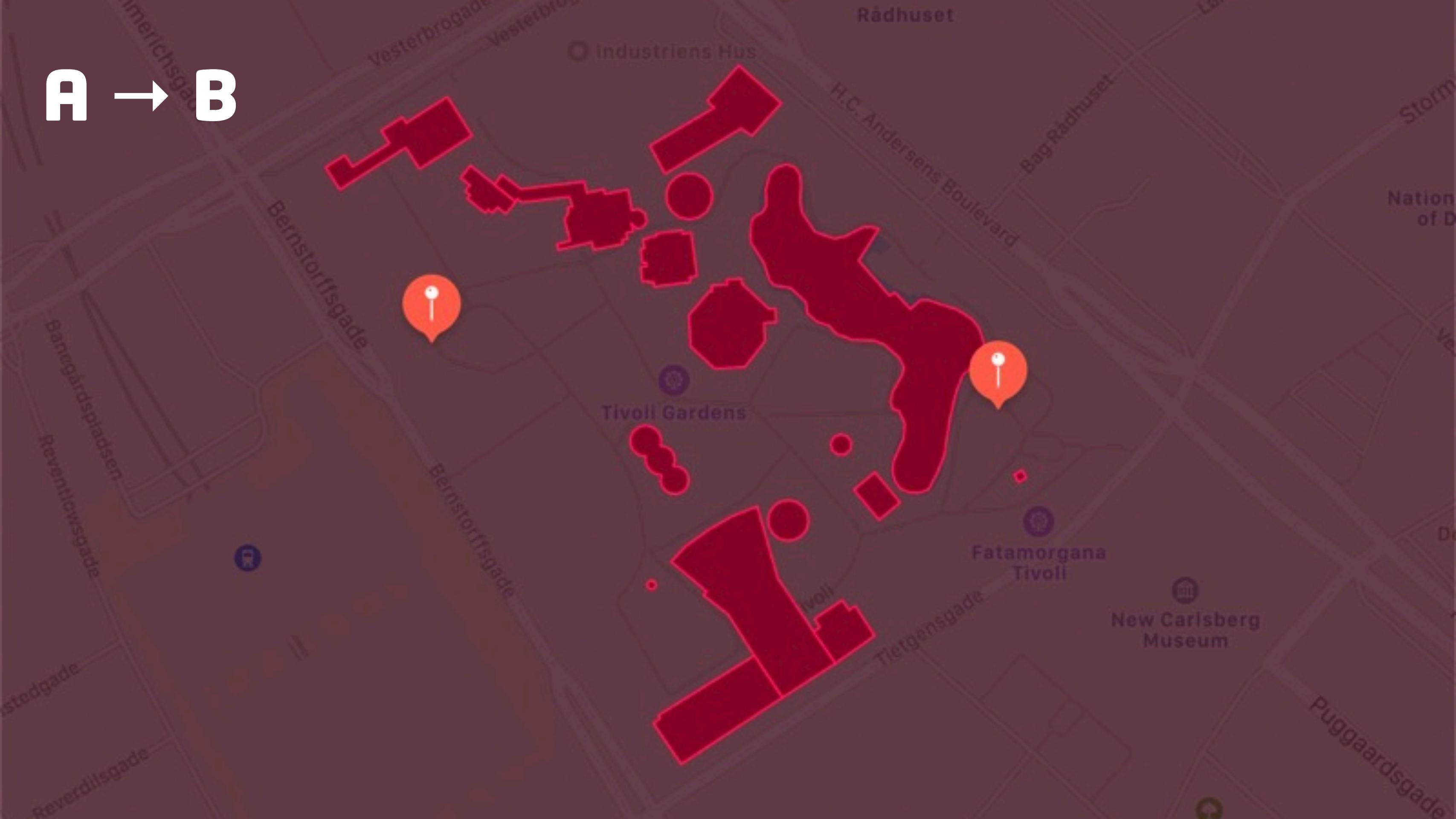
# Obstacle Graph

```
let graph =  
  
  GKObstacleGraph(  
    obstacles: [obstacle],  
    bufferRadius: 0  
)
```

# Obstacles



A → B



# Path

```
let a = GKGraphNode2D(point: float2(x: -1, y: 1))  
let b = GKGraphNode2D(point: float2(x: 2, y: 1))
```

# Path

```
let a = GKGraphNode2D(point: float2(x: -1, y: 1))
let b = GKGraphNode2D(point: float2(x: 2, y: 1))

graph.connectUsingObstacles(node: a)
graph.connectUsingObstacles(node: b)
```

# Path

```
let a = GKGraphNode2D(point: float2(x: -1, y: 1))  
let b = GKGraphNode2D(point: float2(x: 2, y: 1))
```

```
graph.connectUsingObstacles(node: a)  
graph.connectUsingObstacles(node: b)
```

```
let path = graph.findPath(from: a, to: b)
```

# Path

```
let from = GKGraphNode2D(point: float2(x: -1, y: 1))
let to   = GKGraphNode2D(point: float2(x:  2, y: 1))

graph.connectUsingObstacles(node: from)
graph.connectUsingObstacles(node: to)

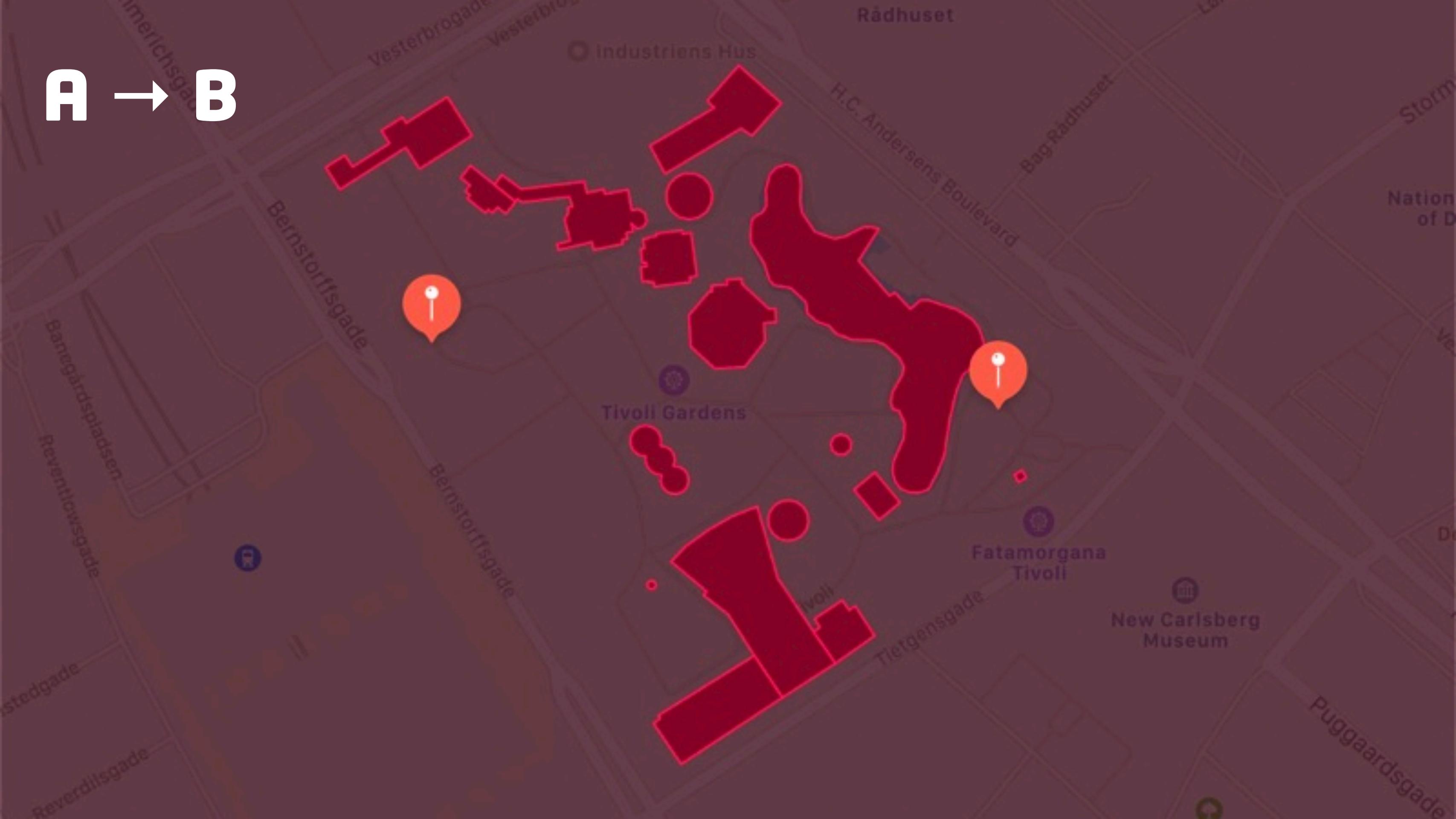
let path = graph.findPath(from: from, to: to)

[
    GKGraphNode2D: {-1, 1},
    GKGraphNode2D: { 0, 0},
    GKGraphNode2D: { 1, 0},
    GKGraphNode2D: { 2, 1}
]
```

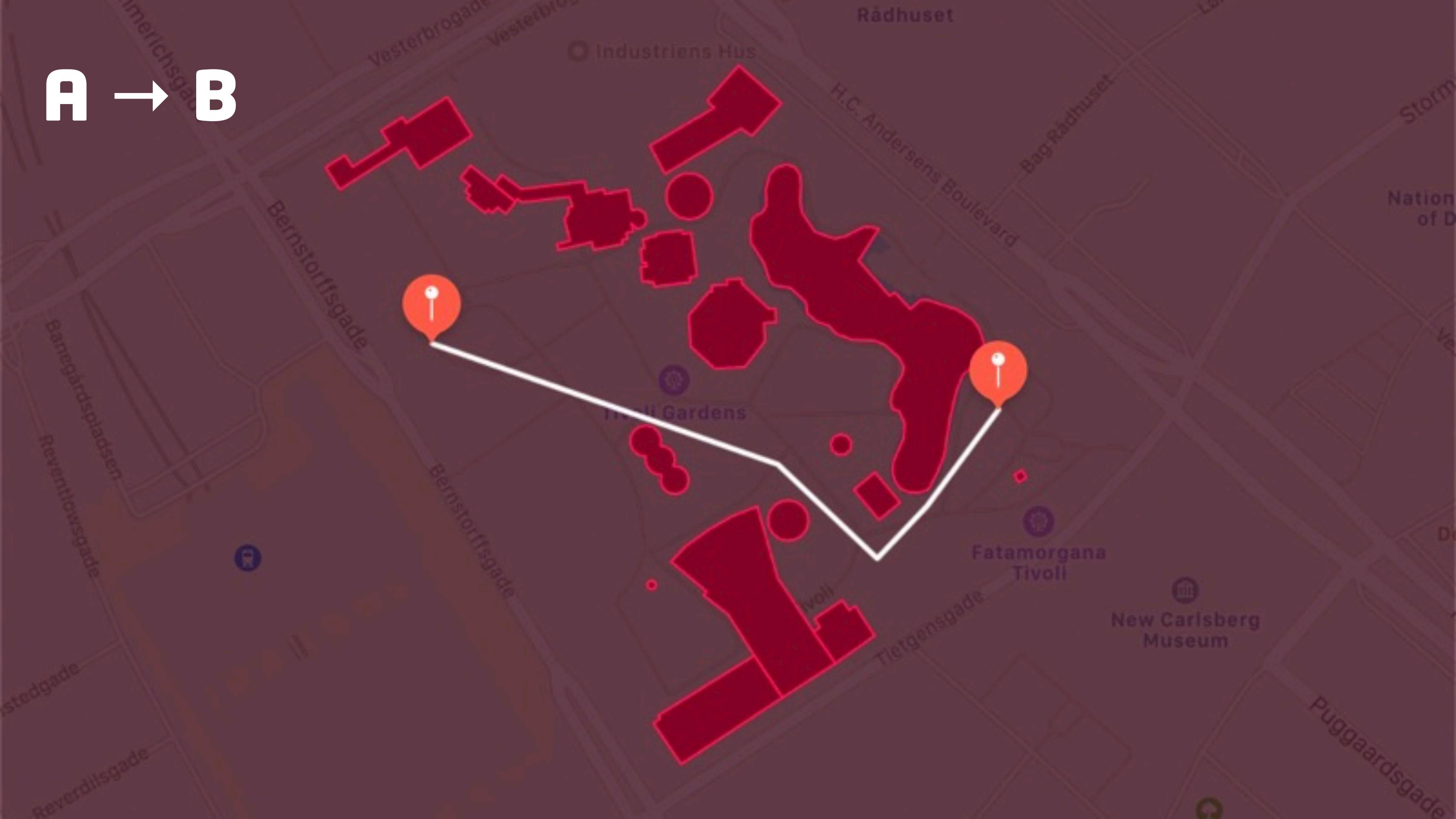
# Buffer Radius

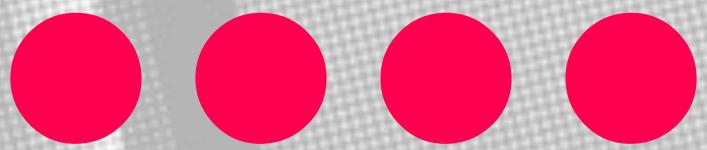
```
let graph = GKObstacleGraph(  
    obstacles: [obstacle],  
    bufferRadius: 0.5  
)  
  
[  
    GKGraphNode2D: {-1.0, 1.0},  
    GKGraphNode2D: {-0.5, 0.0},  
    GKGraphNode2D: { 1.5,-0.5},  
    GKGraphNode2D: { 2.0, 1.0}  
]
```

A → B



A → B





# *Shuffled Arrays*

Performant Visual

Natural Randomness

Pathfinding

shuffled arrays

*Performant Visual Search*

natural Randomness

Pathfinding

# Shuffled Arrays

## Performant Visual

### *natural Randomness*

# Pathfinding

Shuffled Arrays

Performant Visual

Natural Randomness

*Pathfinding*

# Links

- About GameplayKit *by Apple*
- Random Talk: The Consistent World of Noise *by Natalia Berdys*
- GameplayKit: Beyond Games *by Sash Zats*
- The Right Way To Write Dijkstra's Algorithm In Swift *by Federico Zanetello*
- Playground Examples

# Gems — *of* — Gameplay Kit