Introduction to GameplayKit

Session 608

Bruno Sommer
Ross Dexter
Joshua Boggs

GameplayKit

GameplayKit

Mission

Provide gameplay solutions

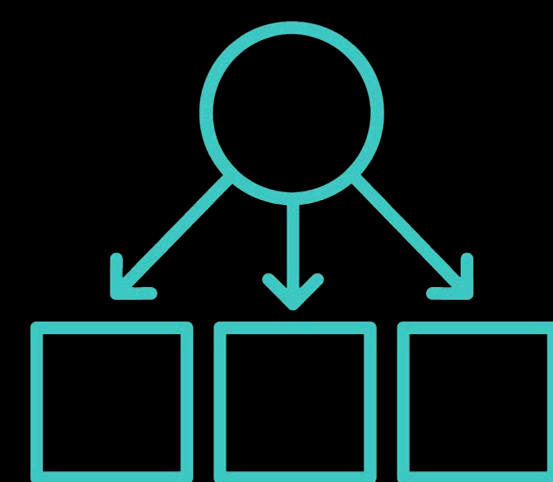
- Common design patterns and architecture
- Standard gameplay algorithms
- Applicable to many genres

Graphics- and engine-agnostic

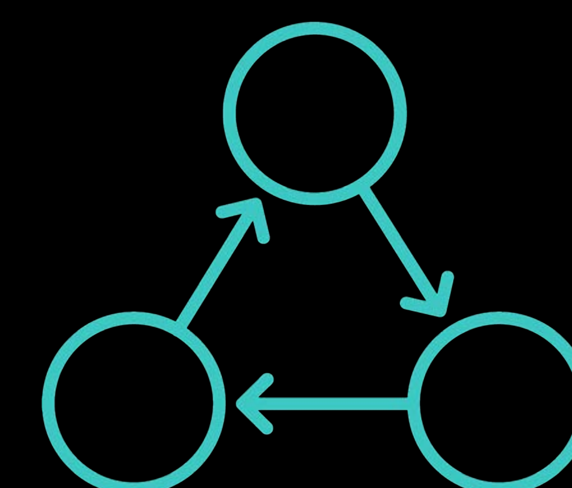
- SpriteKit, SceneKit, Metal, and more

GameplayKit

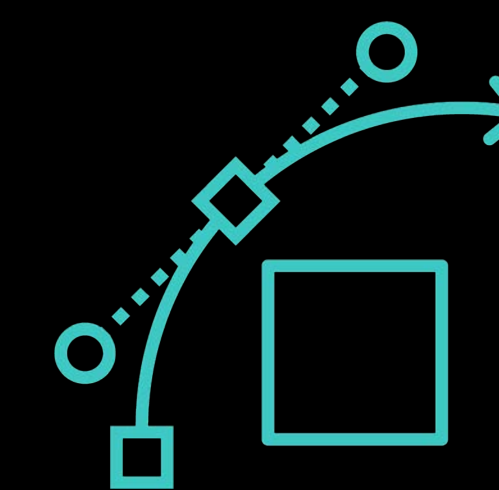
Bringing game ideas to life



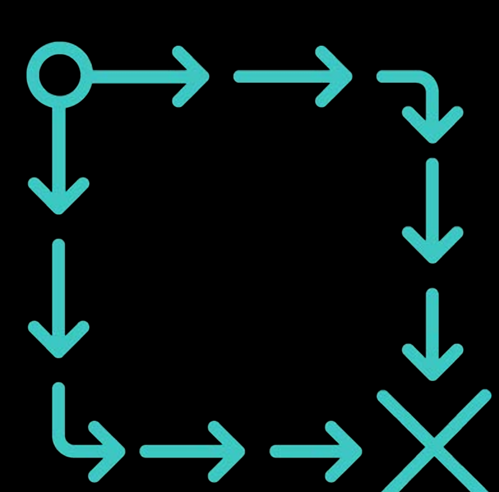
Entities &
Components



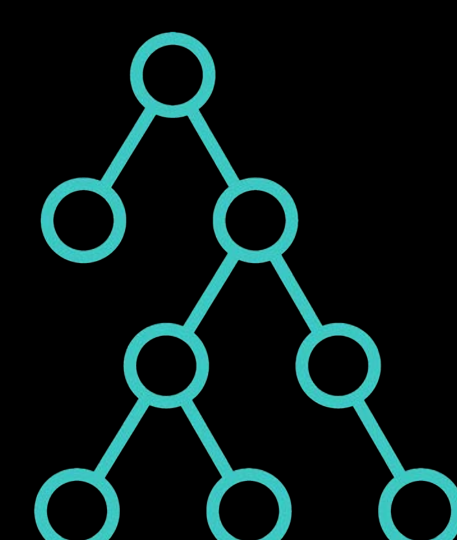
State Machines



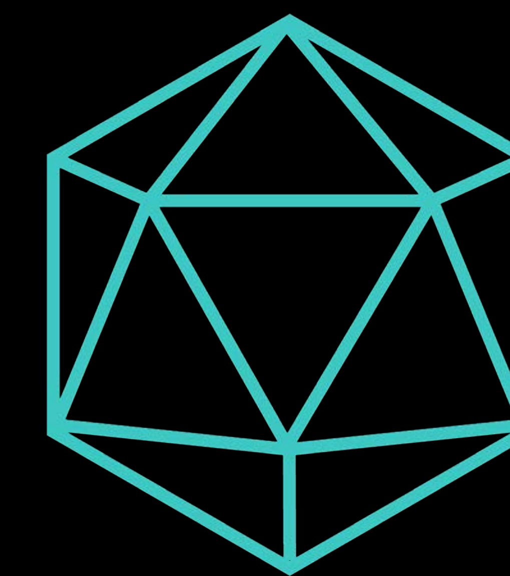
Agents



Pathfinding



MinMax AI



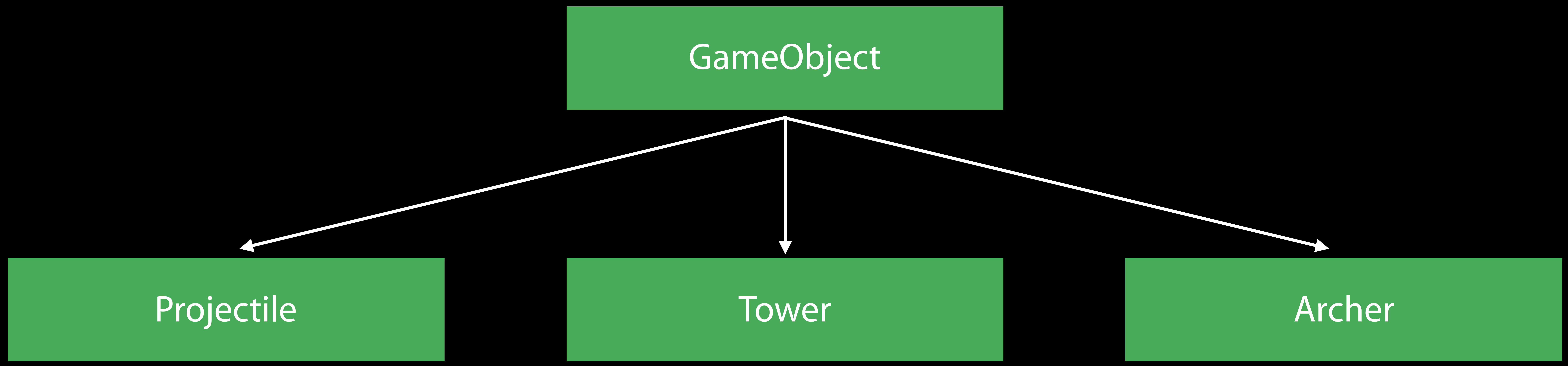
Random
Sources

Rule Systems

Entities and Components

Entities and Components

Background—Classic problem



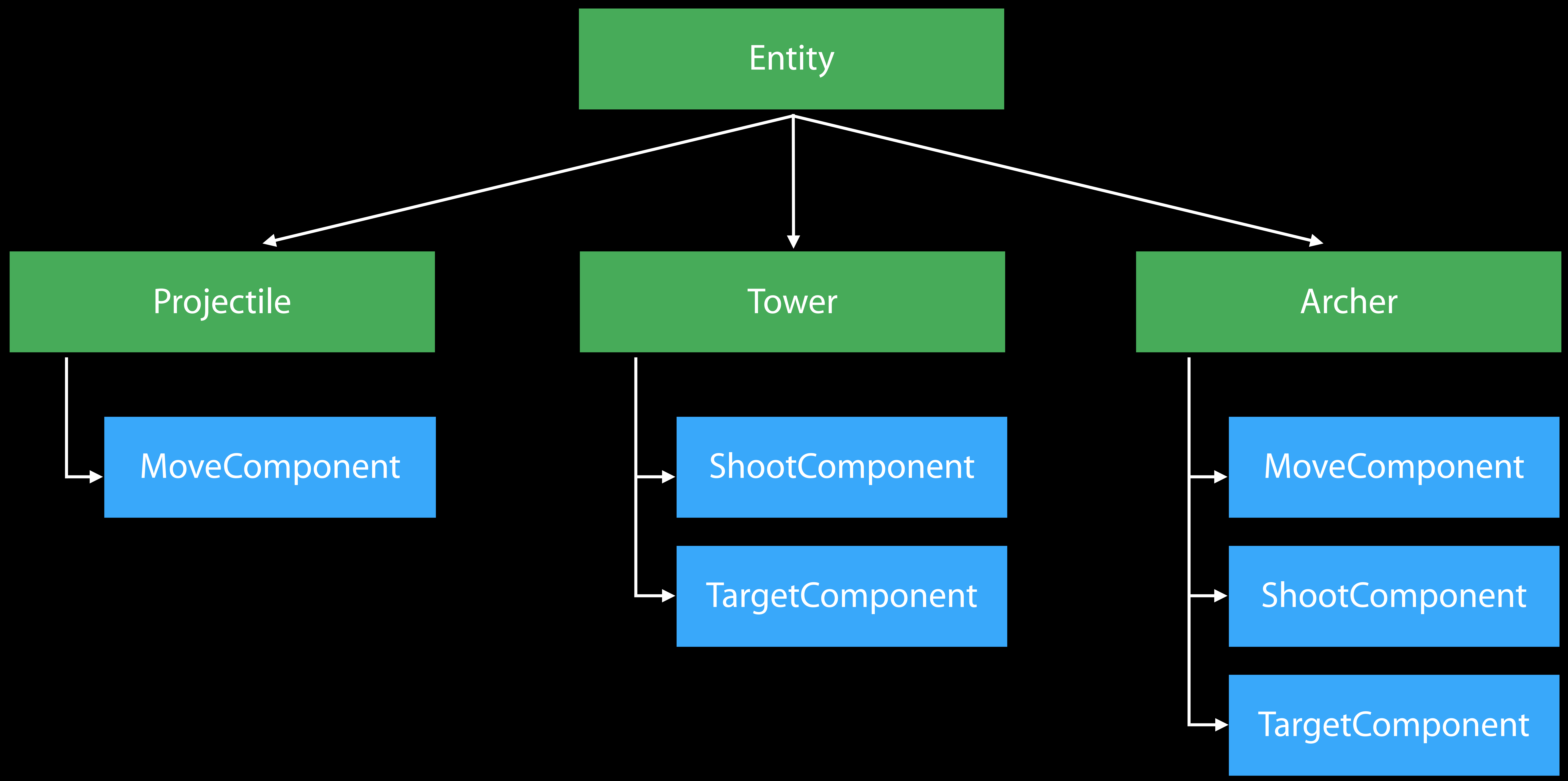
Where does `[shoot:]` go?

Where does `[move:]` go?

Where does `[isTargetable]` go?

Entities and Components

Background—Modern solution



Entities and Components

Background

Great way to organize game logic

Easy-to-maintain

Easy-to-collaborate

Scales with complexity

Dynamic behavior

Entities and Components

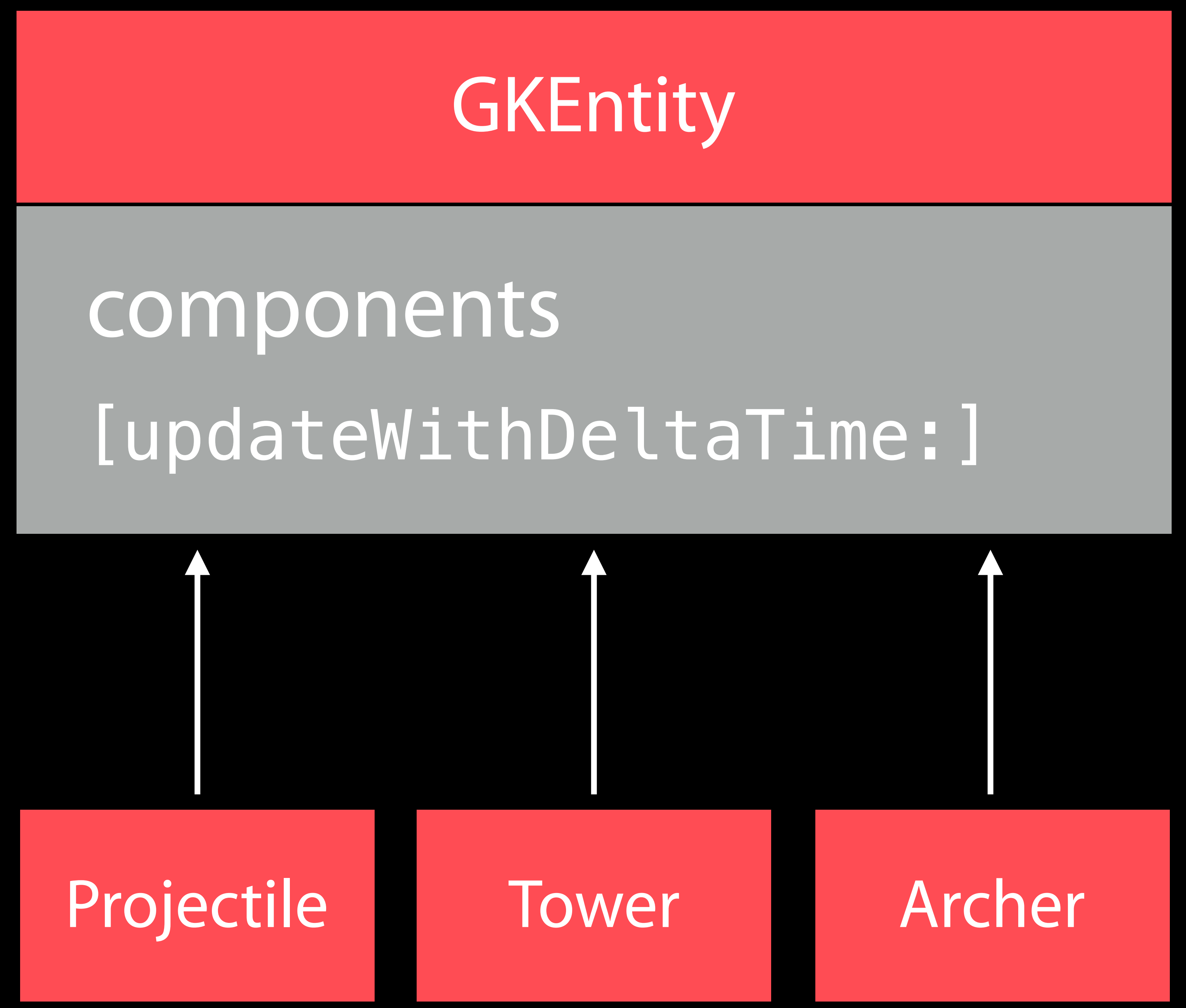
GKEntity

Collection of components

Dynamically add/remove components

Access components by class type

Update all components



Entities and Components

GKComponent

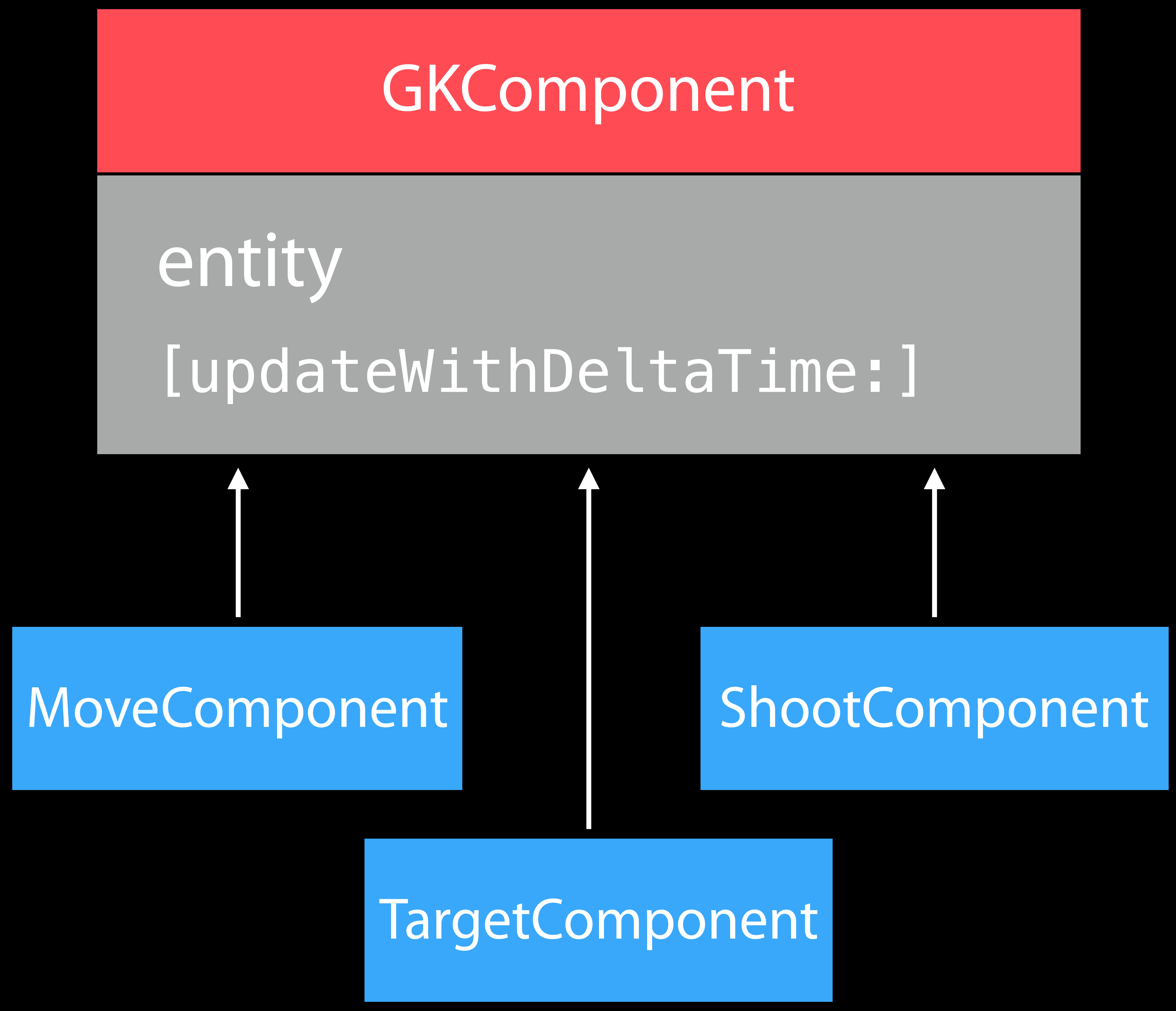
Subclass to add functionality

Store component data as properties

Custom selectors extend functionality

Updated by their entity's update

Implement logic in `[updateWithDeltaTime:]`



Entities and Components

GKComponentSystem

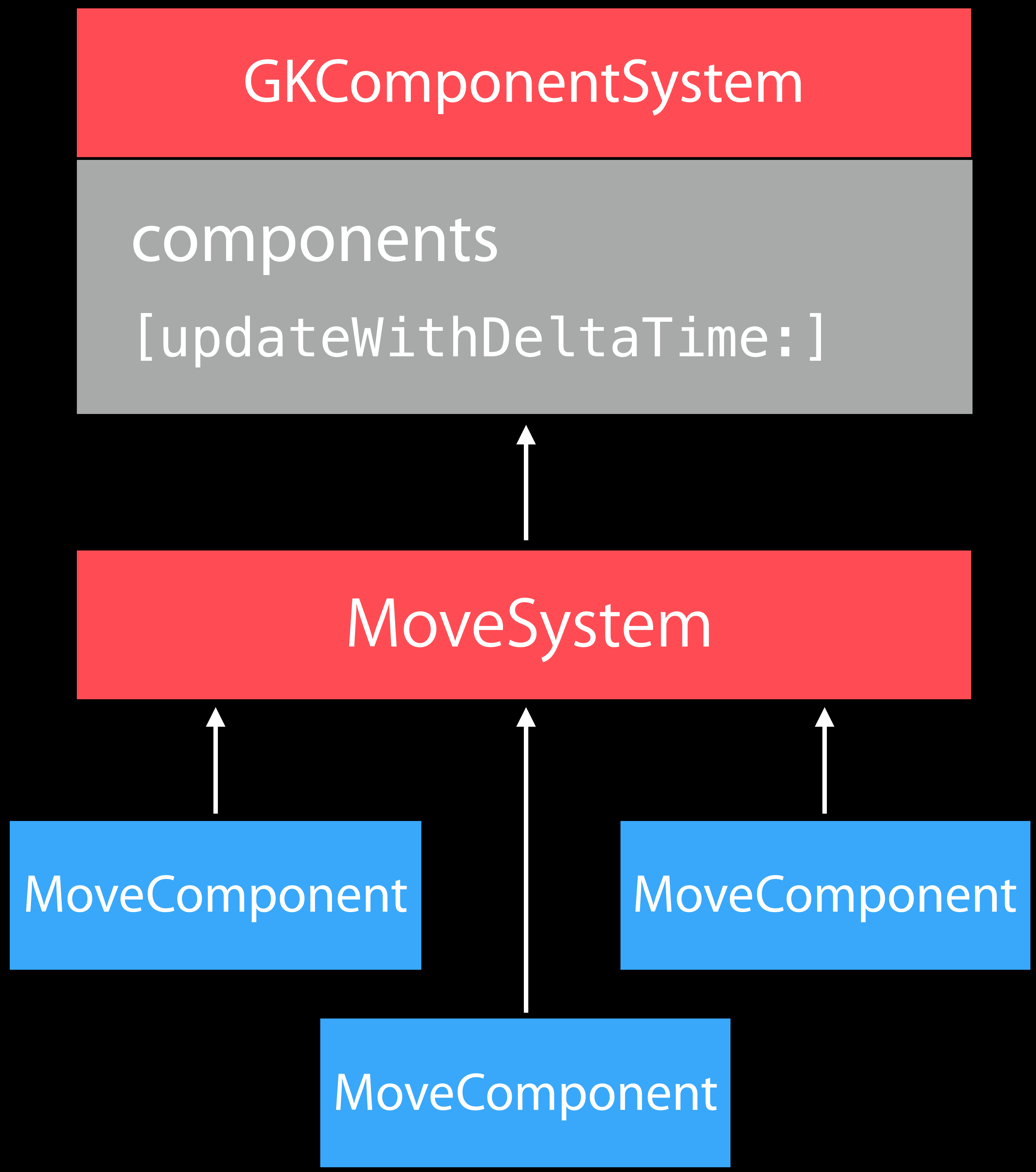
Collection of components from different entities

- All components are of the same class

Use when update order is important

- Update AI before movement, etc

Components in a system do not update with their entity's update



Entities and Components

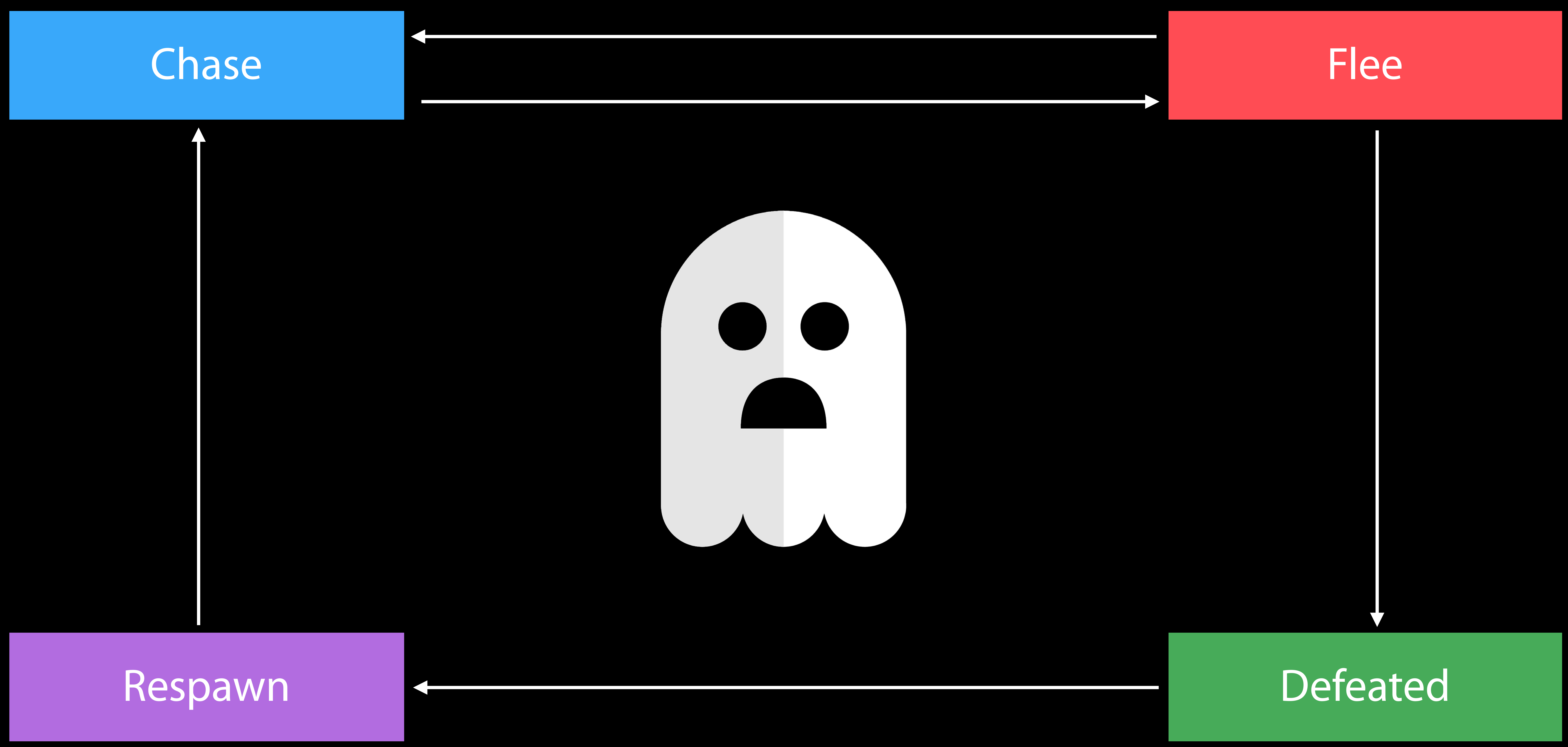
Example

```
/* Make our archer */
GKEntity *archer = [GKEntity entity];

/* Archers can move, shoot, be targeted */
[archer addComponent: [MoveComponent component]];
[archer addComponent: [ShootComponent component]];
[archer addComponent: [TargetComponent component]];

/* Create MoveComponentSystem */
GKComponentSystem *moveSystem =
          [GKComponentSystem systemWithComponentClass:MoveComponent.class];

/* Add archer's MoveComponent to the system */
[moveSystem addComponent: [archer componentForClass:MoveComponent.class]];
```

State Machines

State Machines

Example



State Machines

Background

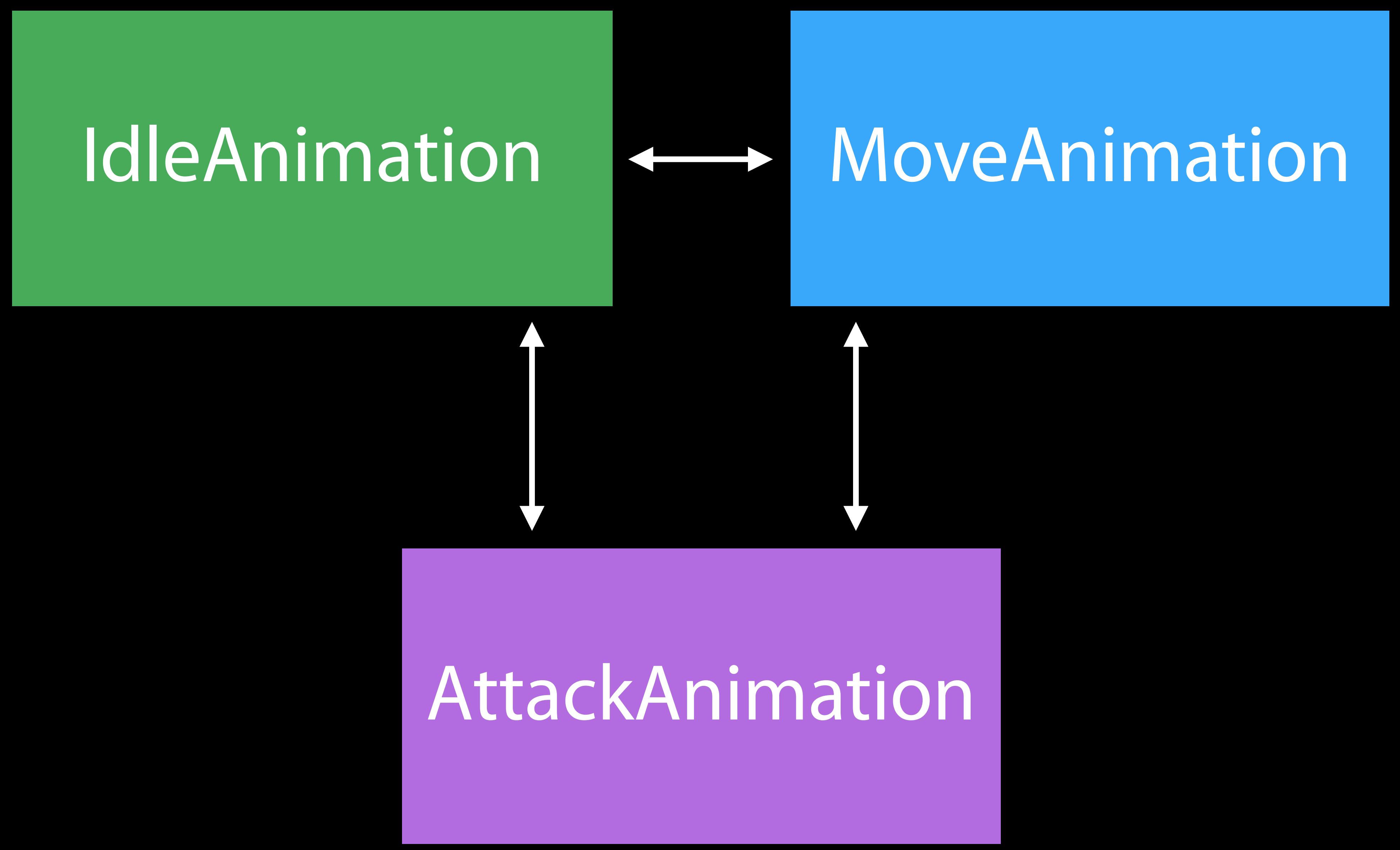
Backbone of many gameplay elements

Games are a collection of state machines

- Animation, AI, UI, levels, etc.

Common implementation removes boilerplate

States reused throughout your game



State Machines

GKStateMachine

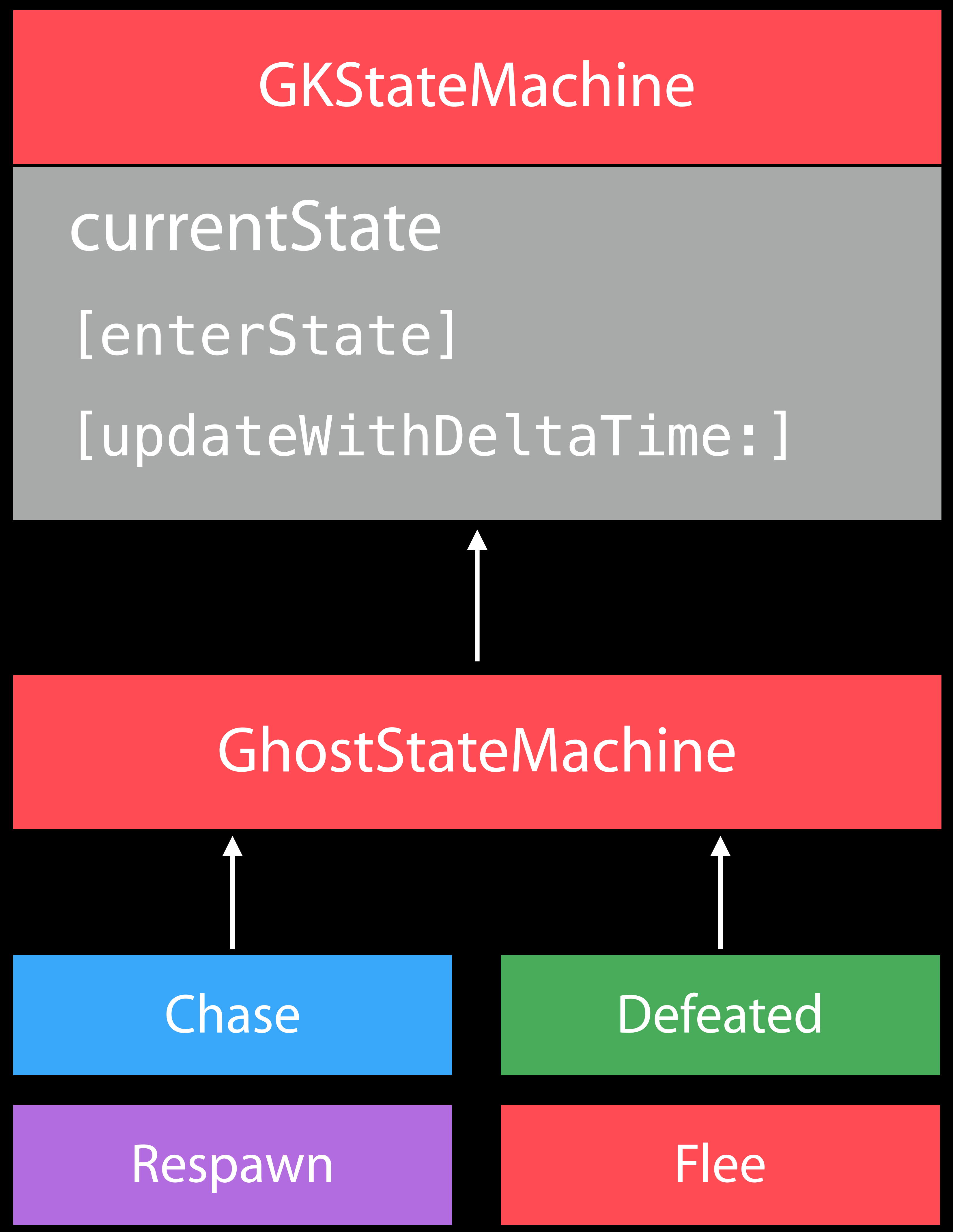
General purpose finite state machine

- Single current state
- All possible states

`[enterState]` causes state transition

- Checks if transition is valid
- Calls `[exit]` on previous, `[enter]` on next state

Updates `currentState`



State Machines

GKState

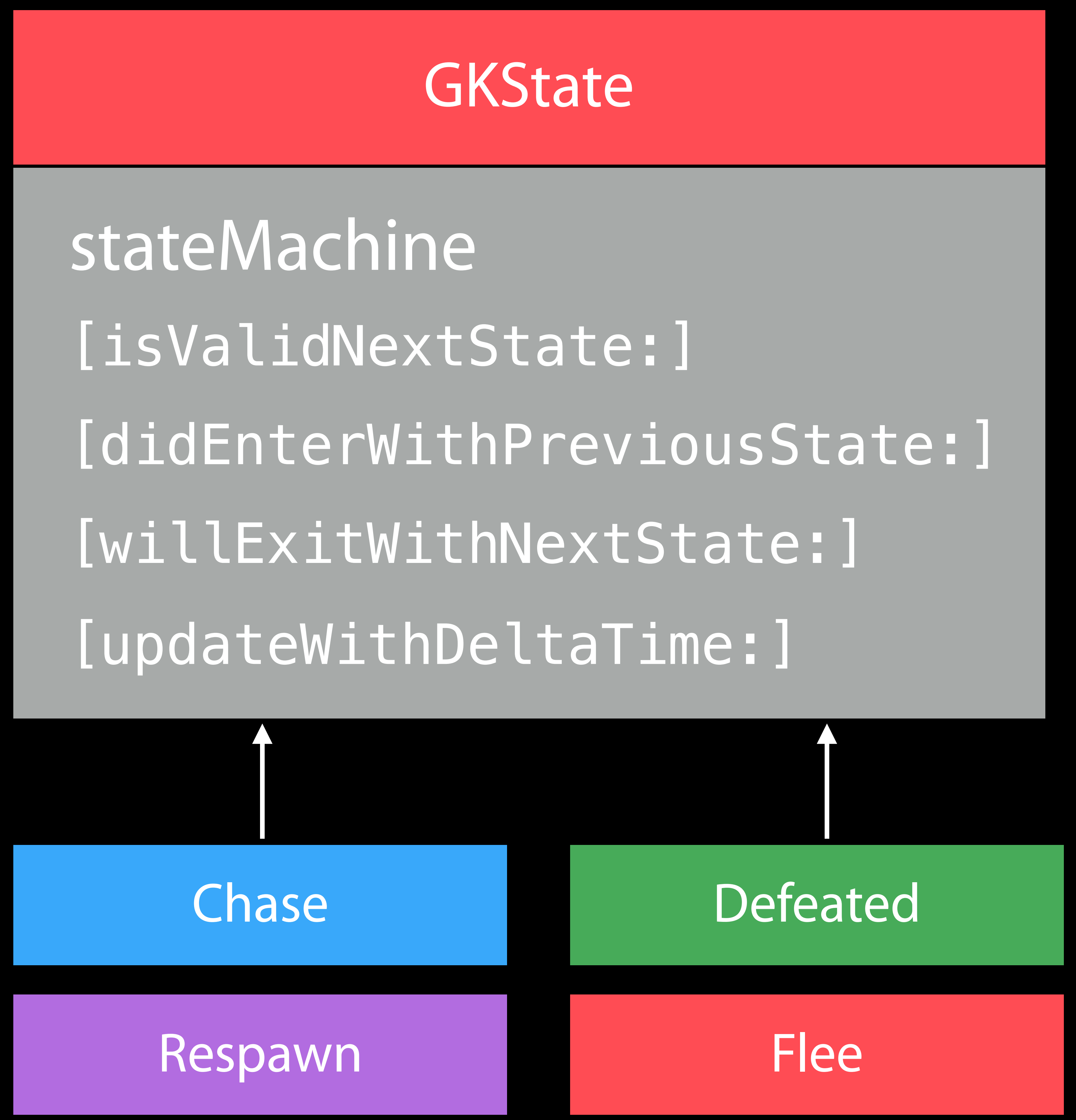
Abstract class

Implement logic in Enter/Exit/Update

- These are called by the state machine

Override `[isValidNextState:]` to control edges

- By default, all edges are valid
- Can be dynamic, based on internal state



State Machine

Example

```
/* Make some states - Chase, Flee, Defeated, Respawn */
ChaseState *chase = [ChaseState state];
FleeState *flee = [FleeState state];
DefeatedState *defeated = [DefeatedState state];
RespawnState *respawn = [RespawnState state];

/* Create a state machine */
GKStateMachine *stateMachine = [GKStateMachine stateMachineWithStates:
                                                    @[chase, flee, defeated, respawn]];

/* Enter our initial state - Chase */
[stateMachine enterState:chase];
```

Agents, Goals, and Behaviors

Agents, Goals, and Behaviors

Concepts

Agents are autonomously moving entities

- Driven by behaviors and goals
- Realistic constraints

Behaviors are made up of goals

- Goals combined via weights

Agents, Goals, and Behaviors

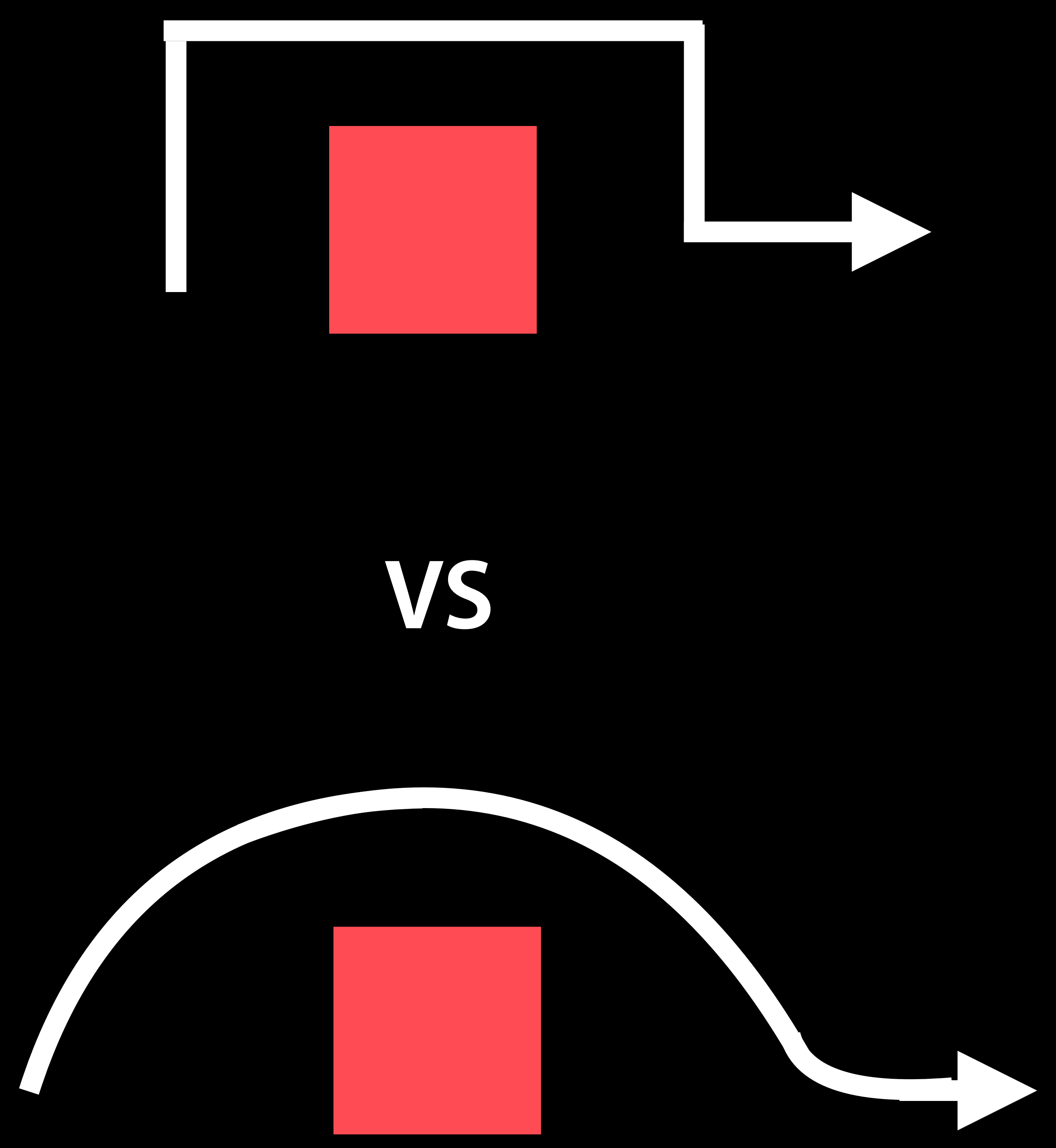
Background

Games need believable movements

Organic behaviors look intelligent

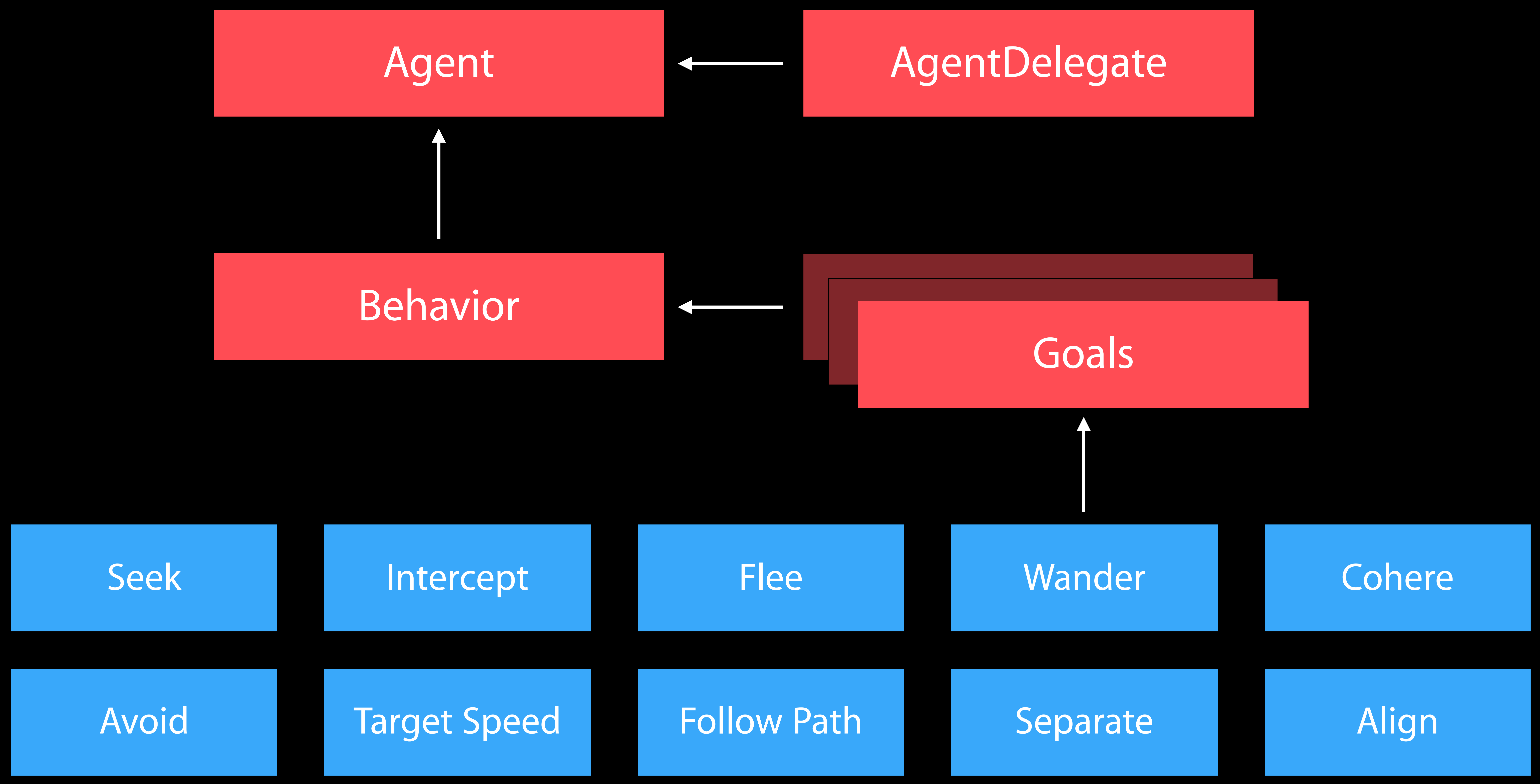
Realistic movement

- Has inertia
- Avoids obstacles
- Avoids other entities
- Follows paths



Agents, Goals, and Behaviors

Overview



Agents, Goals, and Behaviors

GKAgent

Simple autonomous point-mass

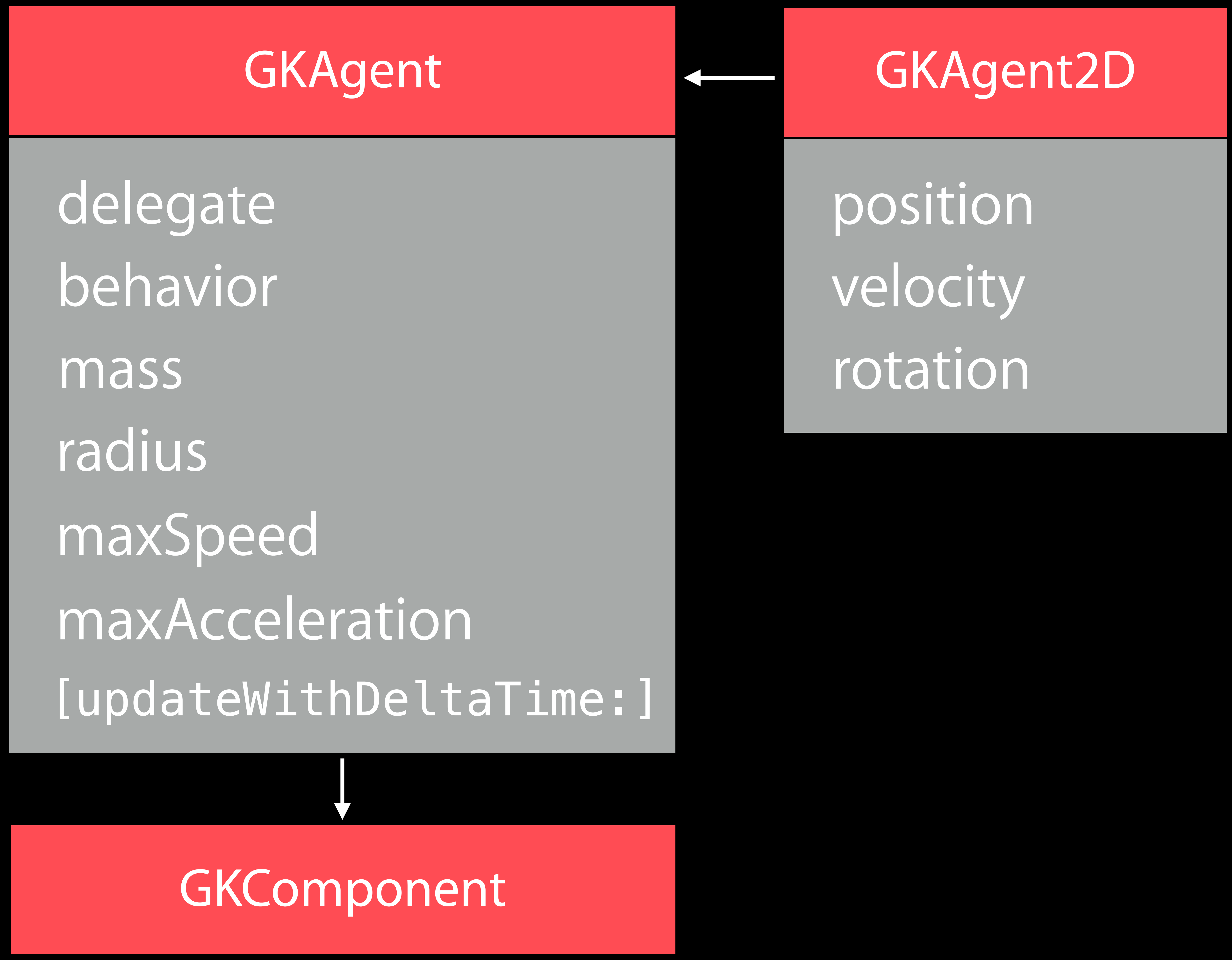
Is a GKComponent

Update applies behavior

- Goals change acceleration
- Velocity, position, rotation updated

Units are dimensionless

- Game-world specific



Agents, Goals, and Behaviors

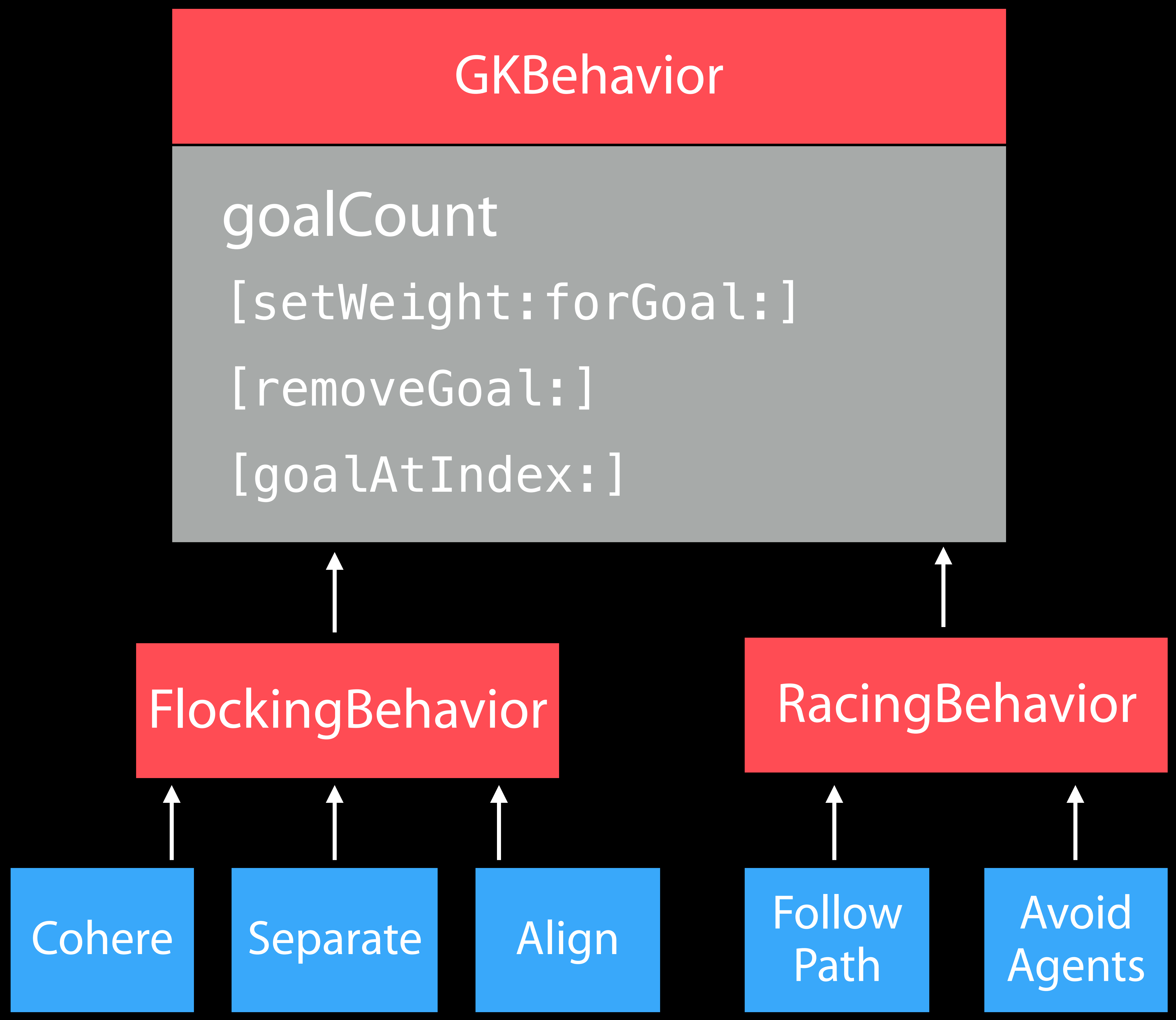
GKBehavior

Dictionary-like container of goals

Dynamically modify behavior

- Add/remove goals
- Modify weights

Set behavior on agent to use it



Agents, Goals, and Behaviors

Example

```
/* Make some goals, we want to seek the enemy, avoid obstacles, target speed */
GKGoal *seek = [GKGoal goalToSeekAgent:enemyAgent];
GKGoal *avoid = [GKGoal goalToAvoidObstacles:obstacles];
GKGoal *targetSpeed = [GKGoal goalToReachTargetSpeed:50.0f];

/* Combine goals into behavior
GKBehavior *behavior = [GKBehavior behaviorWithGoals:@[seek,avoid,targetSpeed]
                                          andWeights:@[@1.0,@5.0,@0.5]];

/* Make an agent - add the behavior to it */
GKAgent2D *agent = [[GKAgent2D* alloc] init];
agent.behavior = behavior;
```

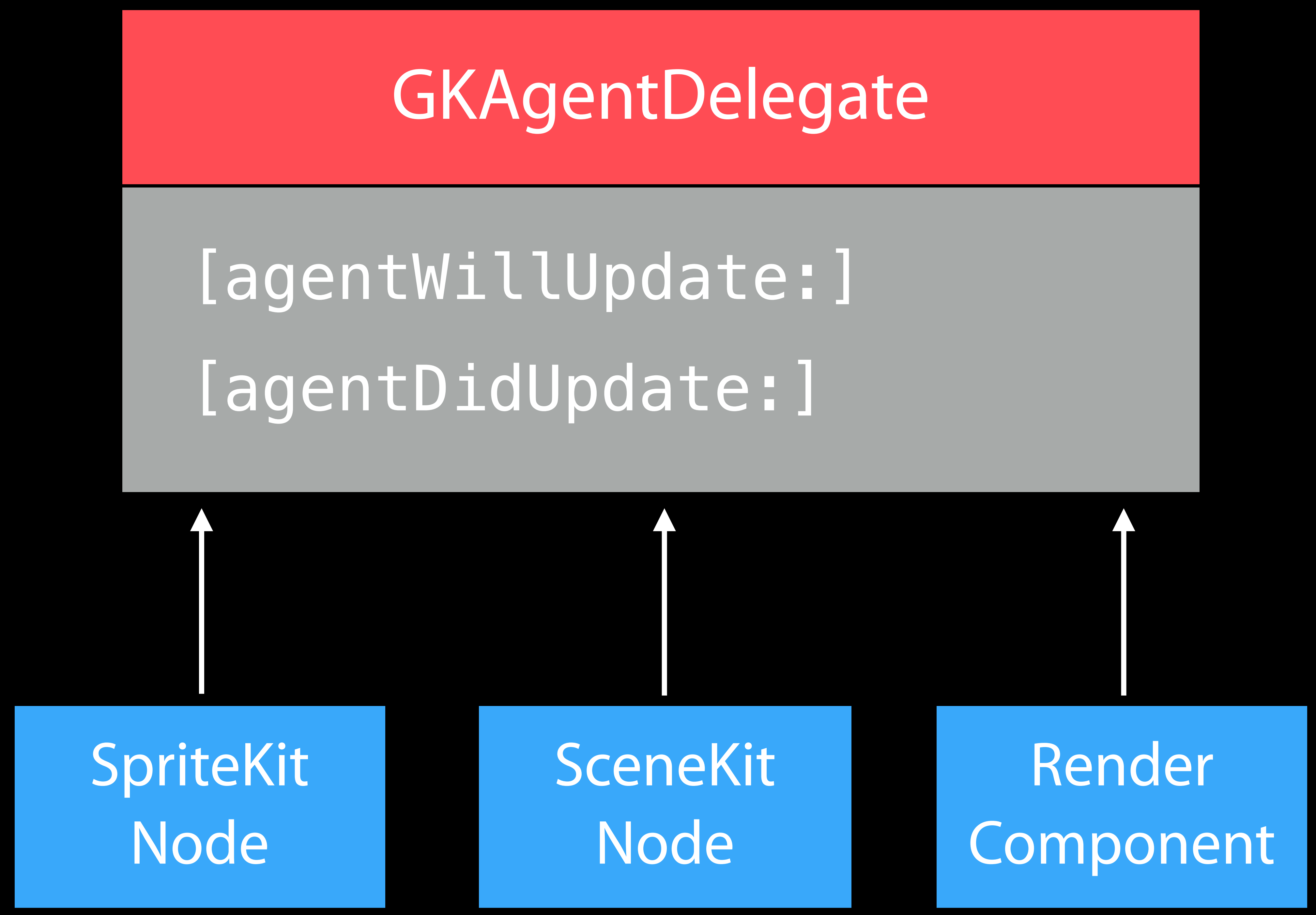
Agents, Goals, and Behaviors

GKAgentDelegate

Sync graphics, animations, physics, etc.

`[agentWillUpdate:]` called before updates

`[agentDidUpdate:]` called after updates



Agents, Goals, and Behaviors

SpriteKit delegate example

```
@implementation MyAgentSpriteNode
…
- (void)agentWillUpdate:(GKAgent2D *)agent {

    /* Position the agent to match our sprite */
    agent.position = self.position;
    agent.rotation = self.zRotation;
}

- (void)agentDidUpdate:(GKAgent2D *)agent {

    /* Update the sprite's position to match the agent */
    self.position = agent.position;
    self.zRotation = agent.rotation;
}
…
@end
```

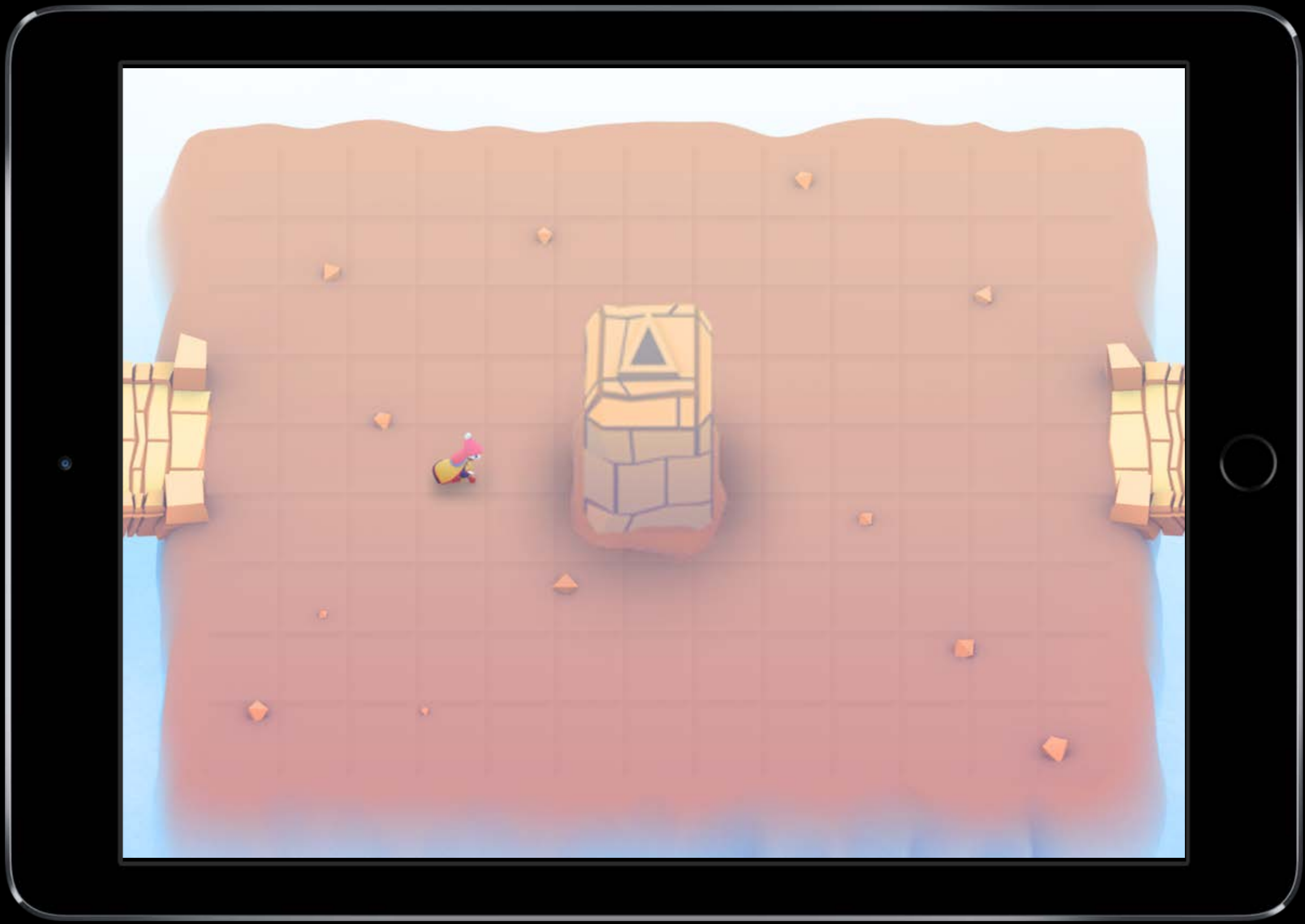
Demo

Agents and goals

Pathfinding

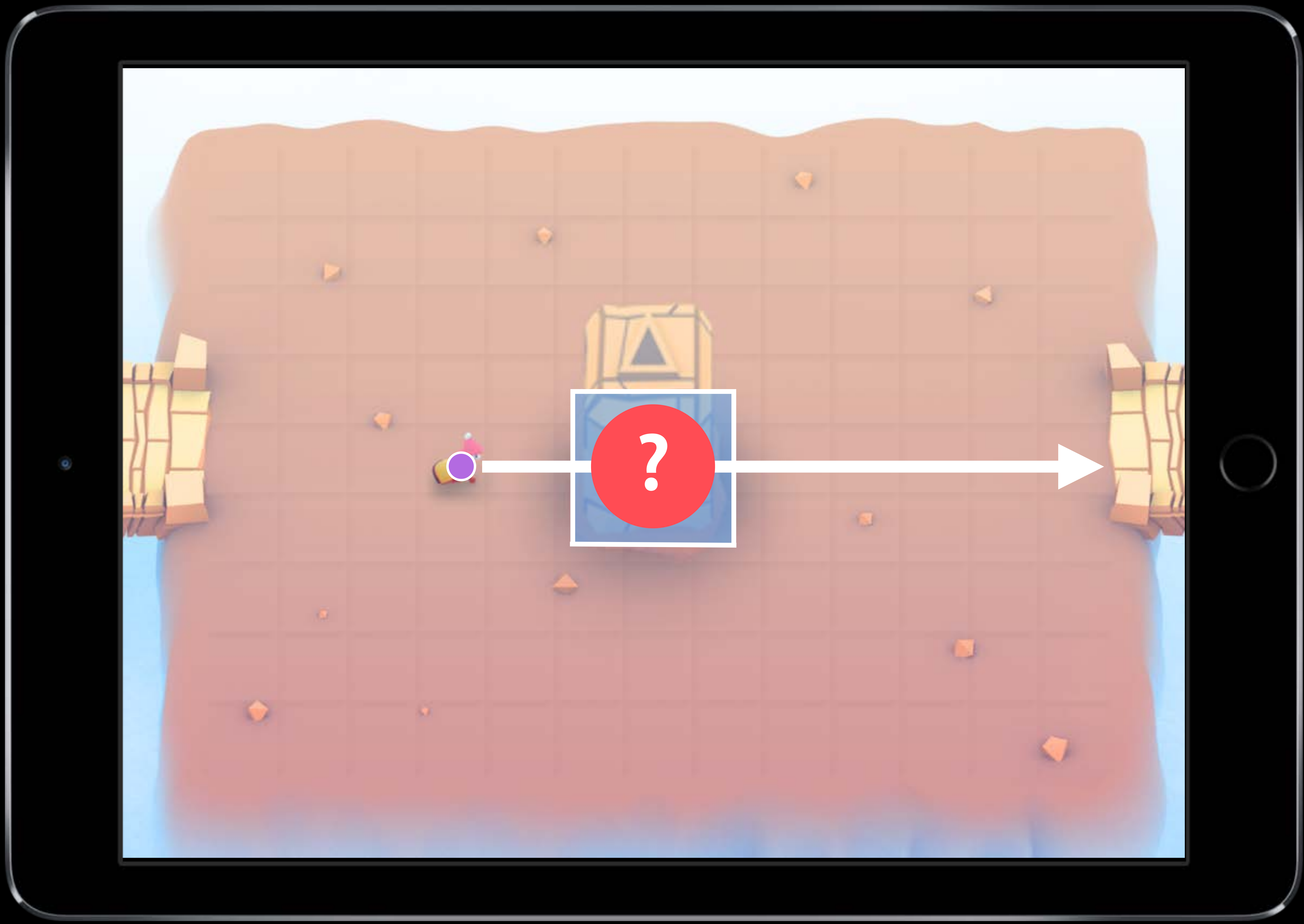
Pathfinding

The problem



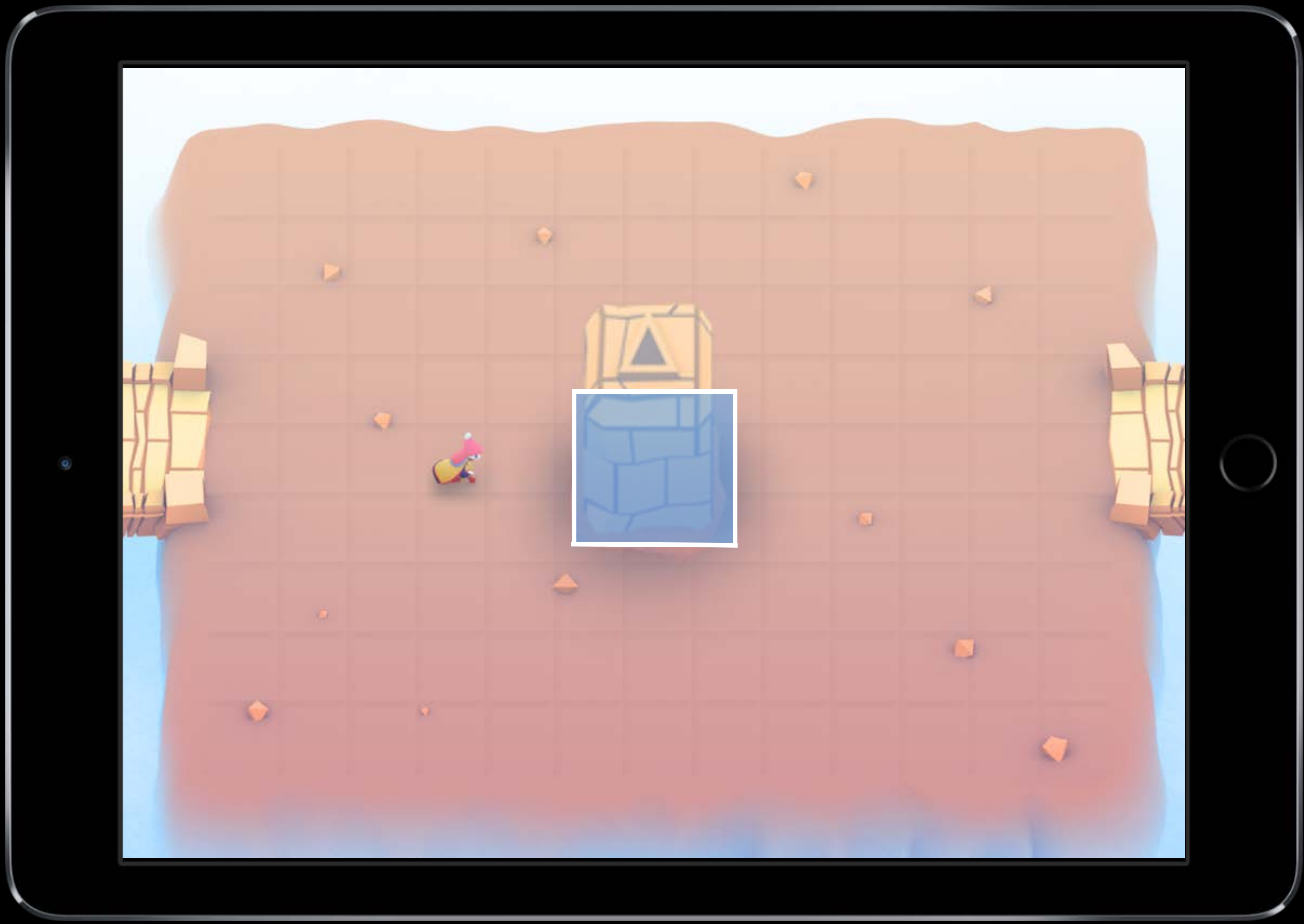
Pathfinding

The problem



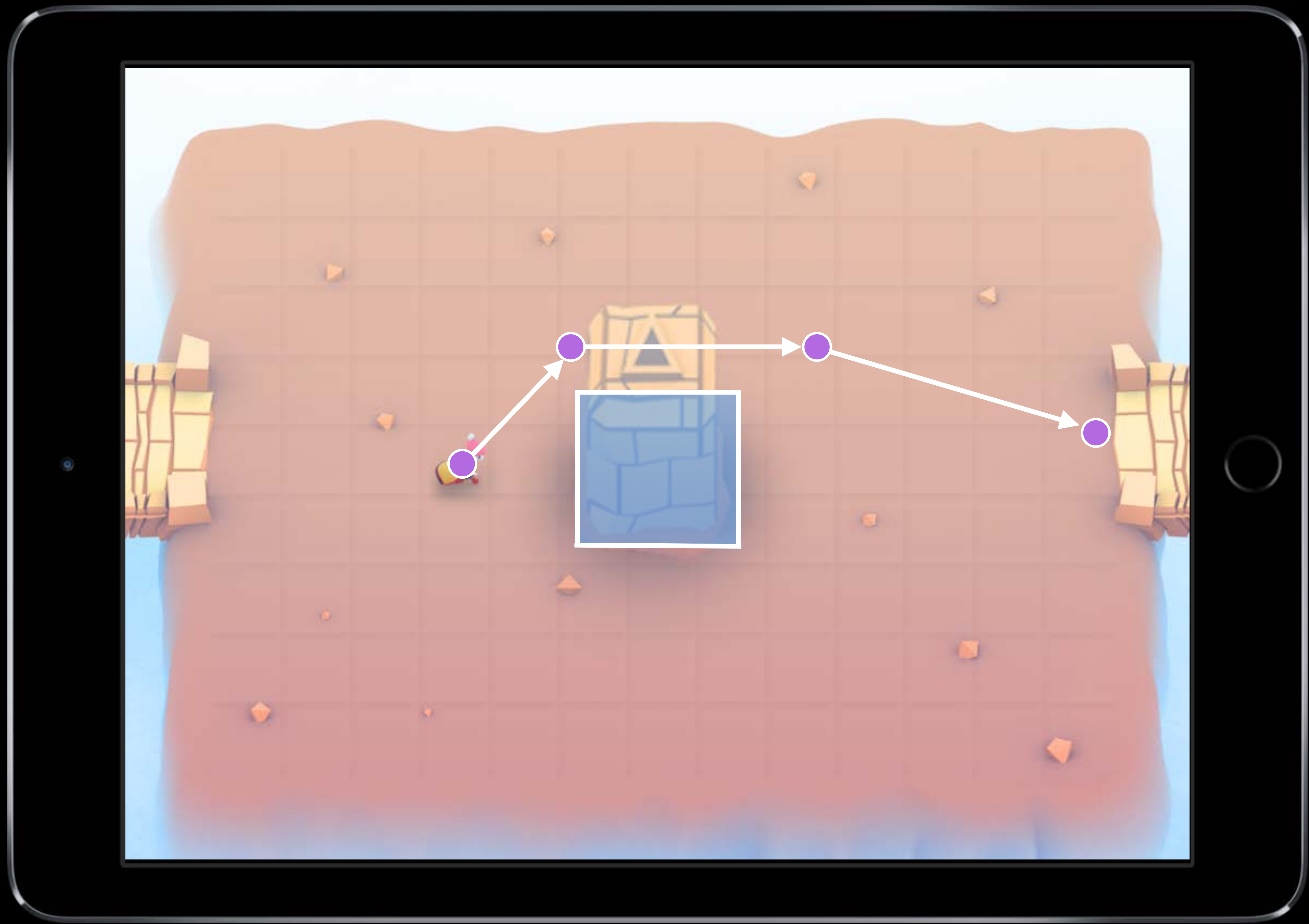
Pathfinding

The solution



Pathfinding

The solution



Pathfinding

Concepts

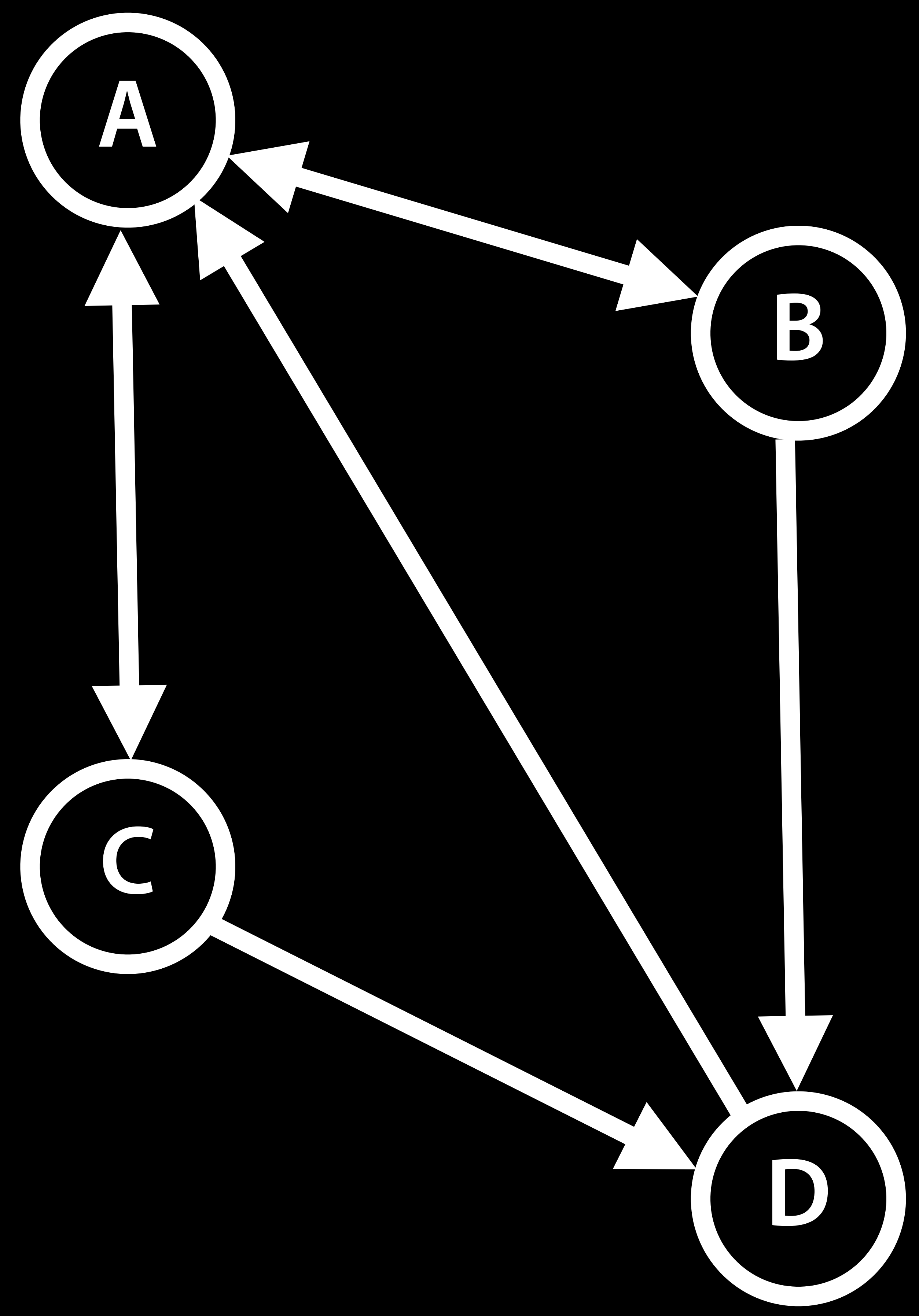
Pathfinding operates on a navigation graph

Graphs are collections of nodes

Nodes are joined by connections

Connections are directional

Optimal path exists between any two connected nodes



Pathfinding

GKGraph

Abstract graph base class

Container of graph nodes

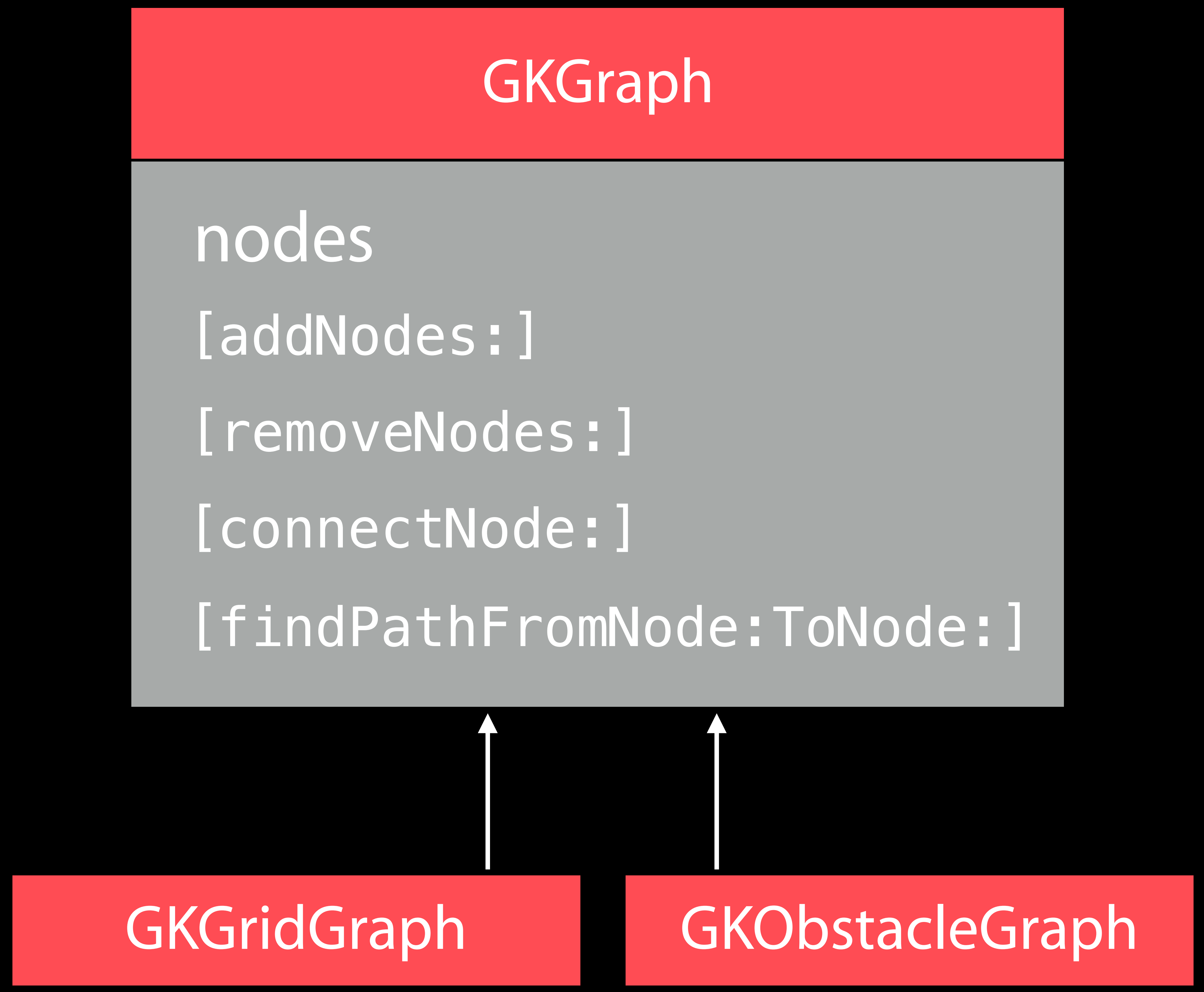
Dynamic add / remove nodes

Connect new nodes

Find paths between nodes

Two specializations

- Grid graphs
- Obstacle graphs



Pathfinding

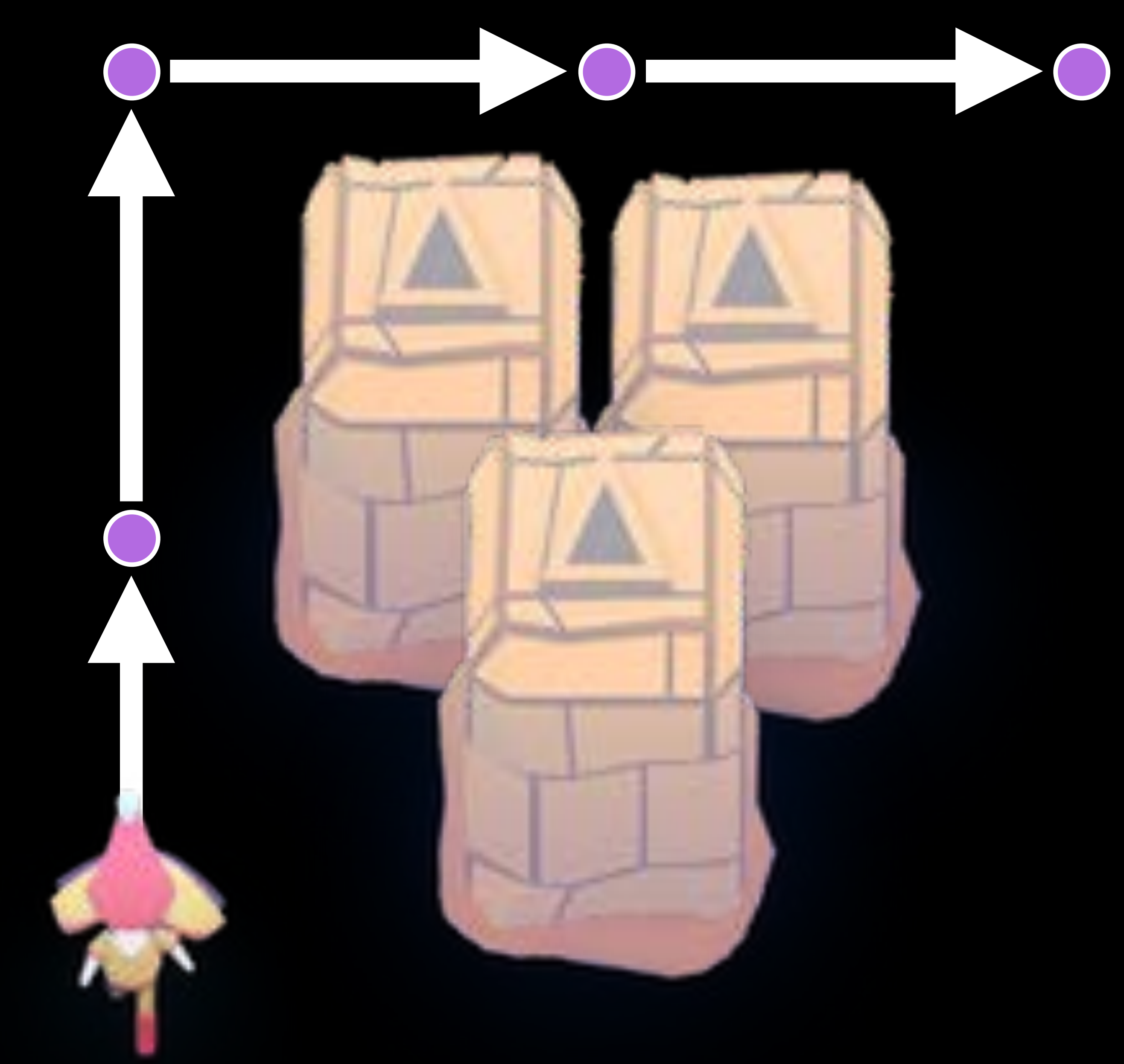
Overview

Find paths in navigation graphs

Generate navigation graphs from

- Obstacles
- Grids
- SpriteKit scenes

Dynamically modify graphs



Pathfinding

GKGridGraph

Specialized for a 2D Grid

Creates nodes on the grid

- Cardinal connections
- Optional diagonal connections

Easy add/remove of grid nodes

GKGridGraph

gridOrigin
gridWidth
gridHeight
diagonalsAllowed
`[nodeAtGridPosition:]`
`[connectNodeToAdjacentNodes:]`

GKGridGraphNode

gridPosition

Pathfinding

GKObstacleGraph

Specialized for pathing around obstacles

- Obstacles are arbitrary polygons

Dynamically add/remove obstacles

Dynamically connect nodes

Buffer radius

- “Safety zone” around obstacles
- Game-dependent size

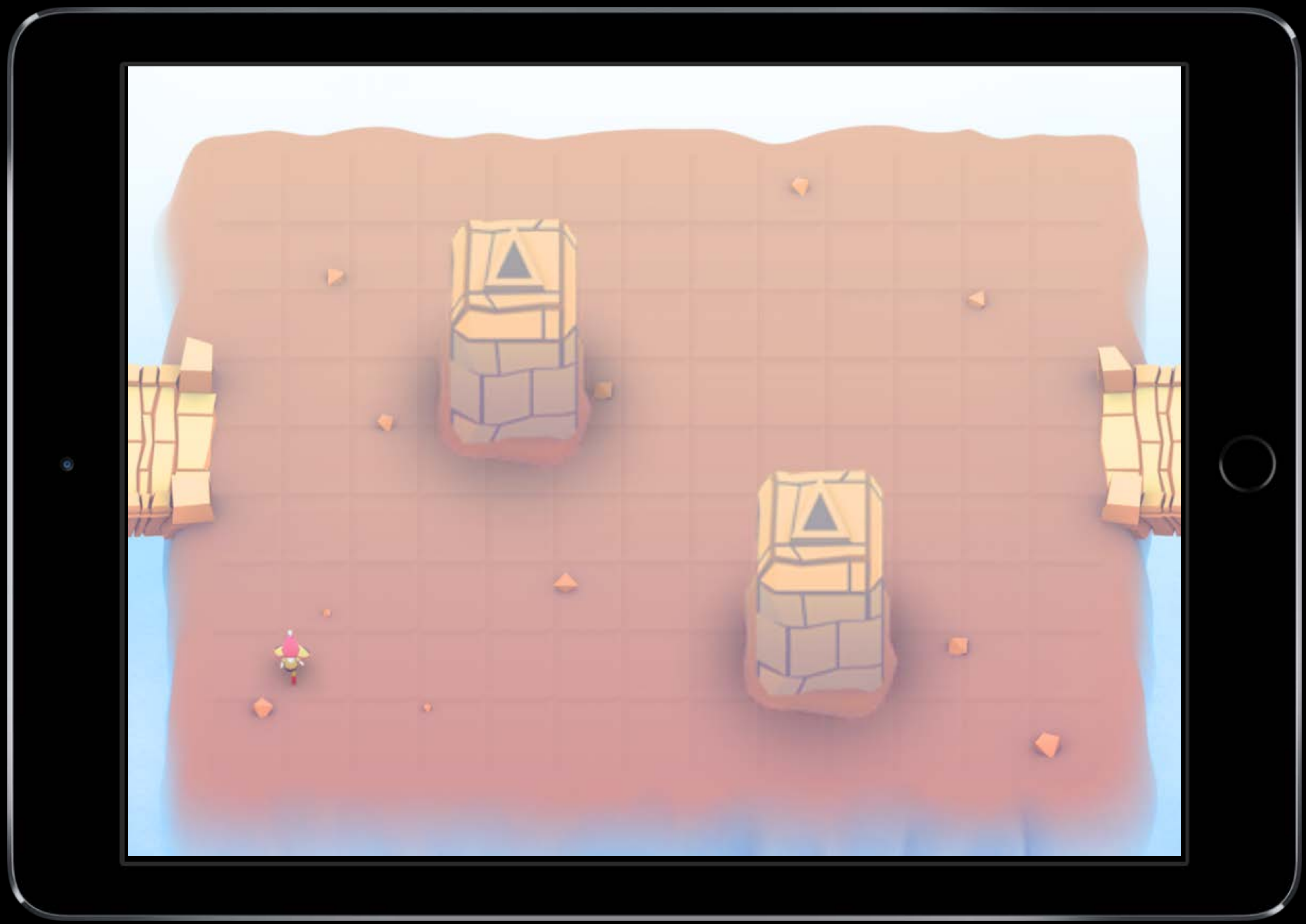
GKObstacleGraph

obstacles
bufferRadius
`[addObstacles:]`
`[removeObstacles:]`
`[connectNodeUsingObstacles:]`
`[lockConnectionFromNode:]`
`[unlockConnectionFromNode:]`

GKGraphNode2D

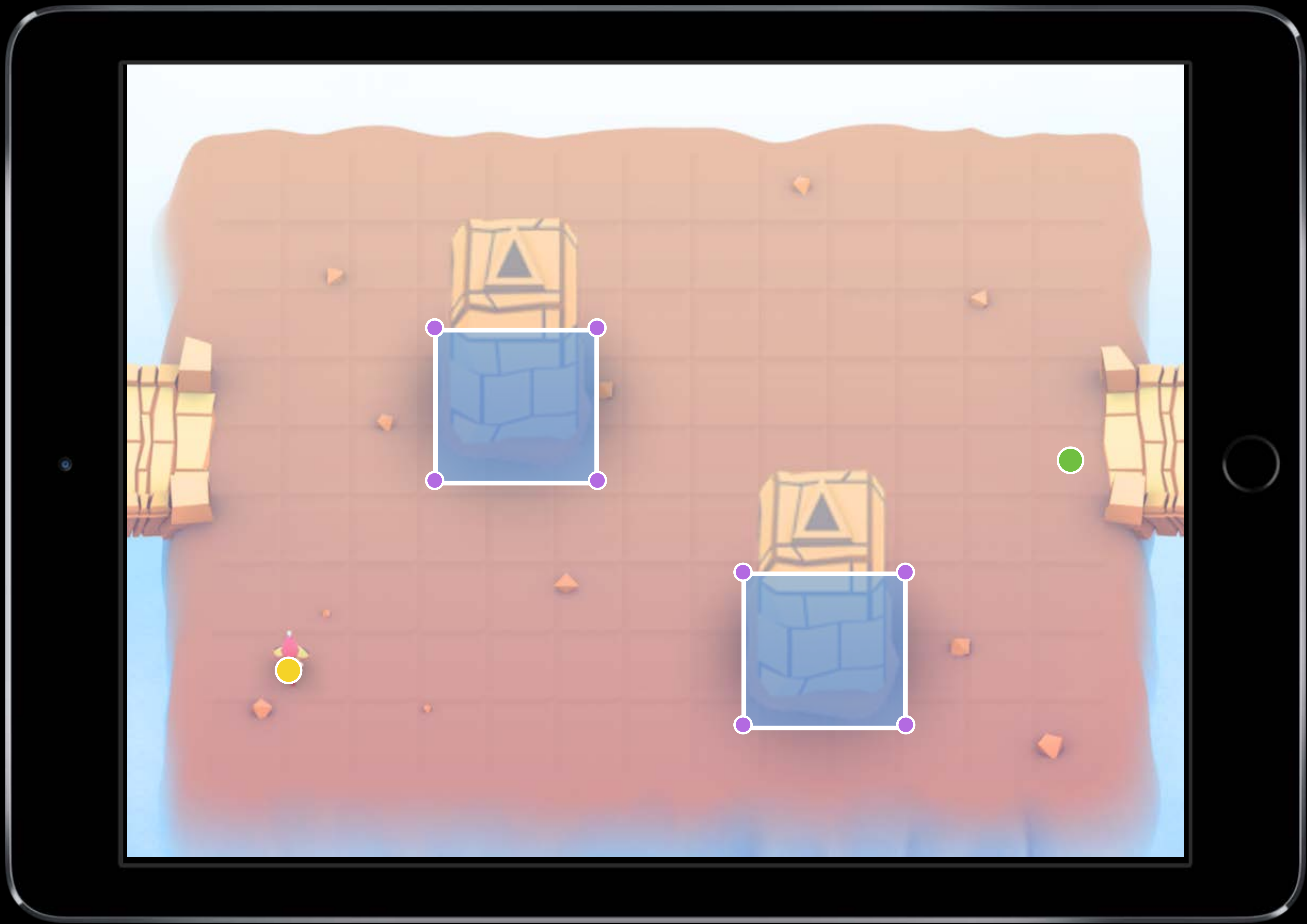
position

GKObstacleGraph Generation



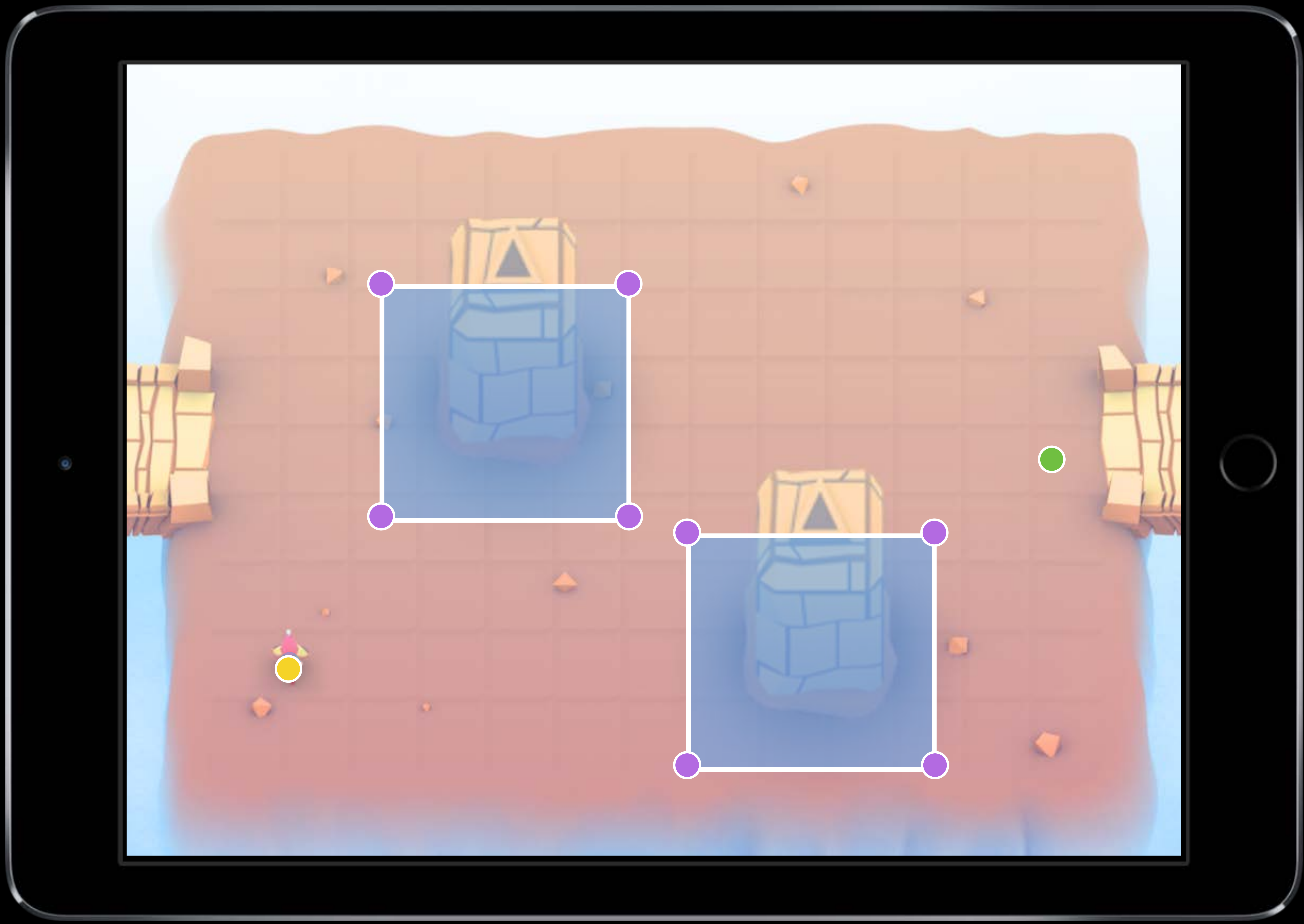
GKObstacleGraph Generation

GKObstacleGraph Generation

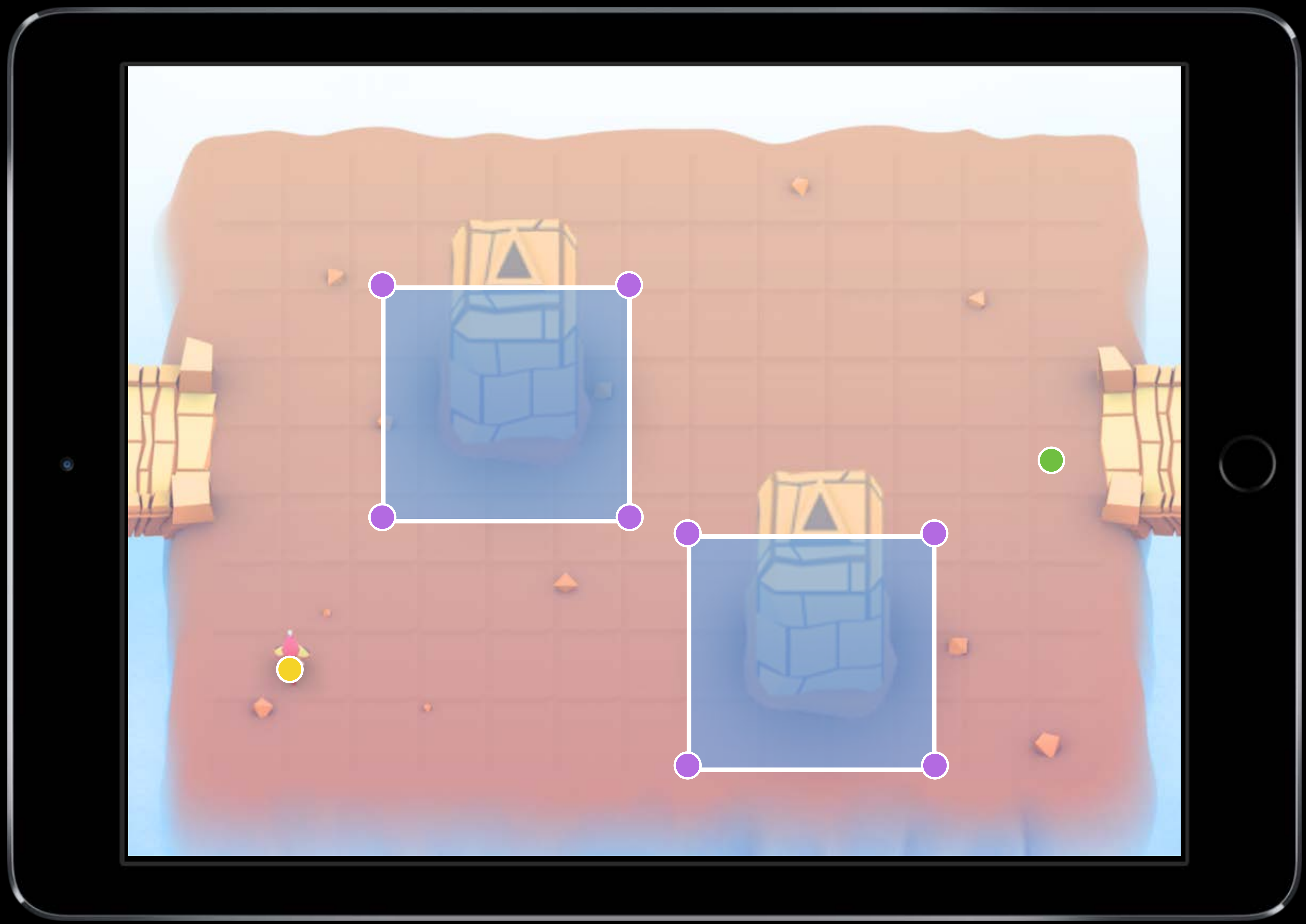


GKObstacleGraph Generation

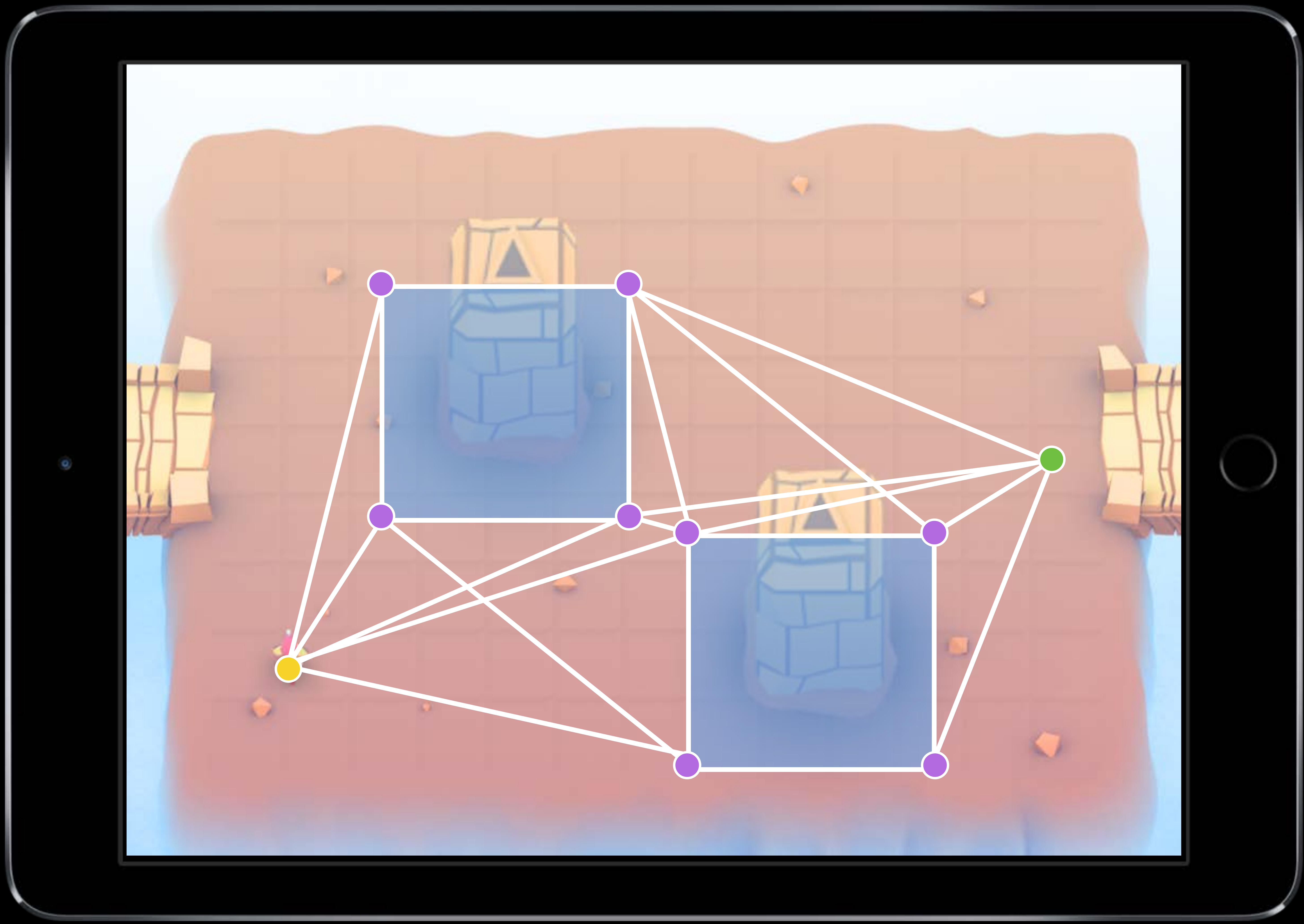
GKObstacleGraph Generation



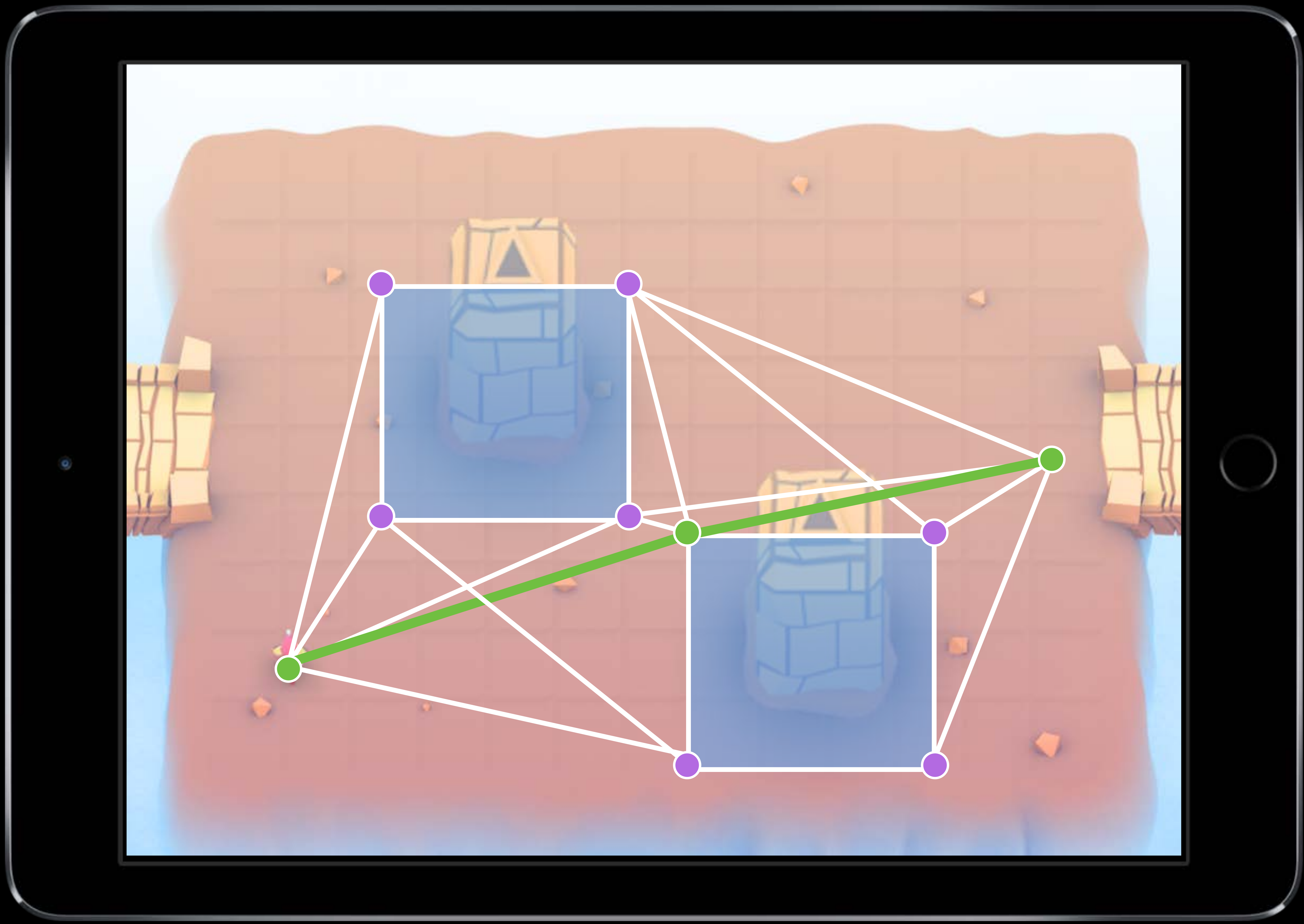
GKObstacleGraph Generation



GKObstacleGraph Generation



GKObstacleGraph Generation



Pathfinding

GKObstacleGraph example

```
/* Make an obstacle - a simple square */
vector_float2 points[] = {{400,400}, {500,400}, {500,500}, {400,500}};
GKPolygonObstacle *obstacle = [[GKPolygonObstacle alloc] initWithPoints:points count:4];

/* Make an obstacle graph */
GKObstacleGraph *graph = [GKObstacleGraph graphWithObstacles:@[obstacle] bufferRadius:10.0f];

/* Make nodes for hero position and destination */
GKGraphNode2D *startNode = [GKGraphNode2D nodeWithPoint:hero.position];
GKGraphNode2D *endNode = [GKGraphNode2D nodeWithPoint:goalPosition];

/* Connect start and end node to graph */
[graph connectNodeUsingObstacles:startNode];
[graph connectNodeUsingObstacles:endNode];

/* Find path from start to end */
NSArray *path = [graph findPathFromNode:startNode toNode:endNode];
```

Pathfinding

Advanced: GKGraphNode

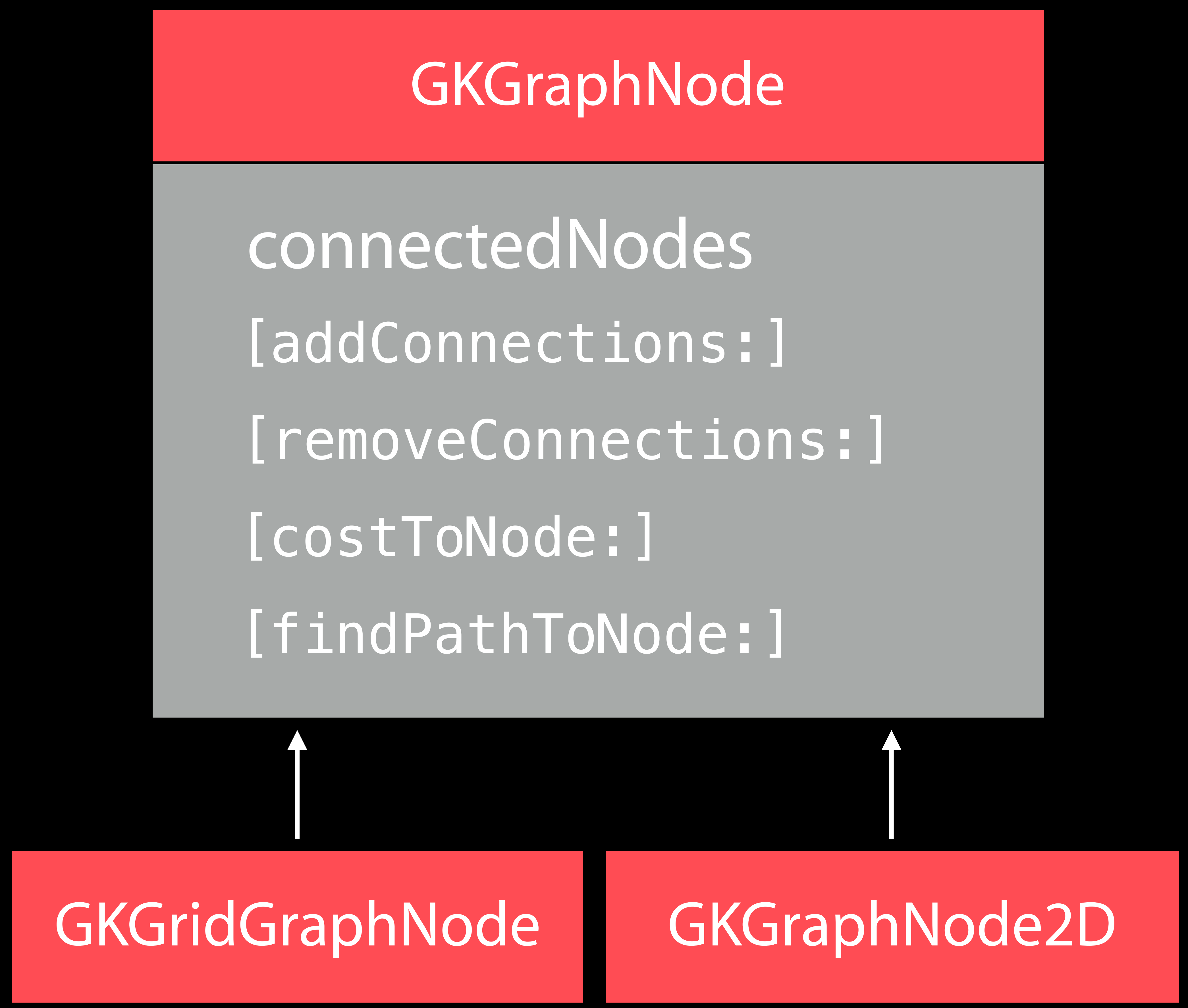
Graph node base class

Subclass for

- Advanced or non-spatial costs
- Control over pathfinding

Create your own graphs

- Manually manage connections
- Good for abstract or non-spatial graphs



Pathfinding

SpriteKit integration

Easily generate obstacles from SKNode bounds, physics bodies, or textures

```
/* Makes obstacles from sprite textures */
(NSArray*)obstaclesFromSpriteTextures:(NSArray*)sprites accuracy:(float)accuracy;

/* Makes obstacles from node bounds */
(NSArray*)obstaclesFromNodeBounds:(NSArray*)nodes;

/* Makes obstacles from node physics bodies */
(NSArray*)obstaclesFromNodePhysicsBodies:(NSArray*)nodes;
```

Demo

SpriteKit integration

MinMax AI

Ross Dexter

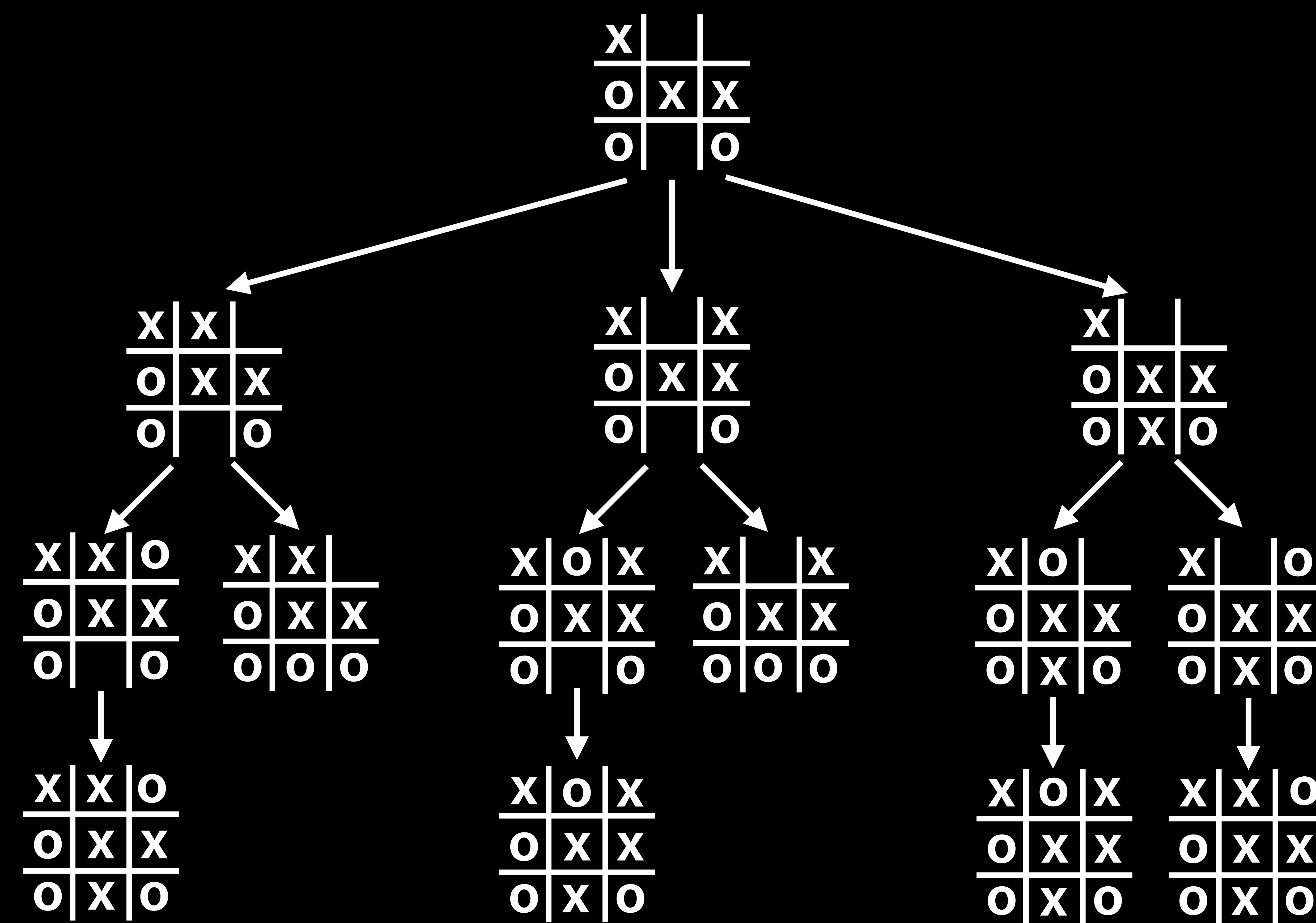
MinMax AI

Example

Many games need equal AI opponents

- Can play the entire game
- Play by the same rules as human players

Chess, Checkers, Tic-Tac-Toe, etc.



MinMax AI

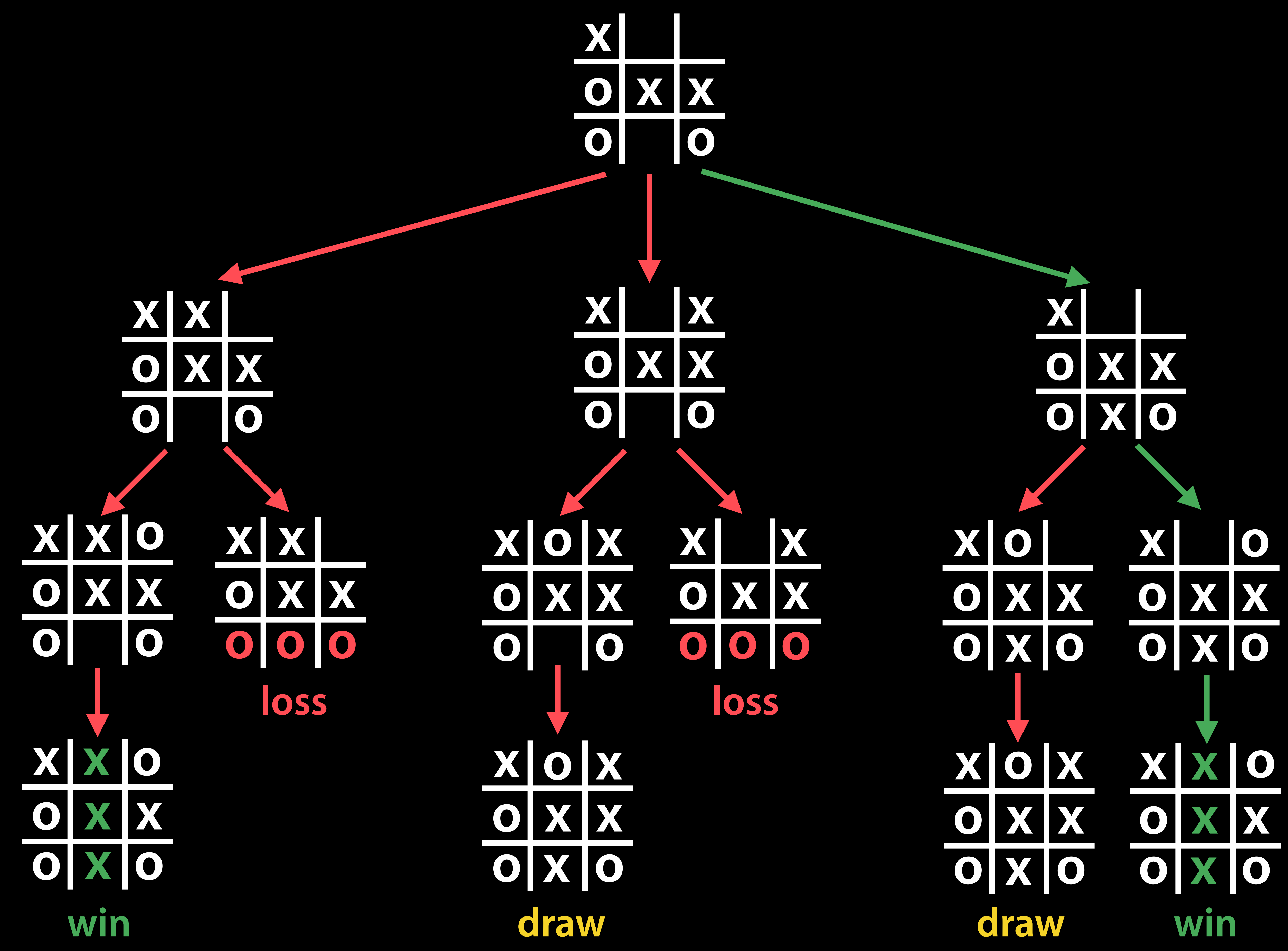
Example

MinMax AI

- Looks at player moves
- Builds decision tree
- Maximizes potential gain
- Minimizes potential loss

Tic-Tac-Toe example

- Right branch optimal
- Other branches lead to potential loss



MinMax AI

Features

AI-controlled opponents

Suggest move for human players

Best suited for turn-based games

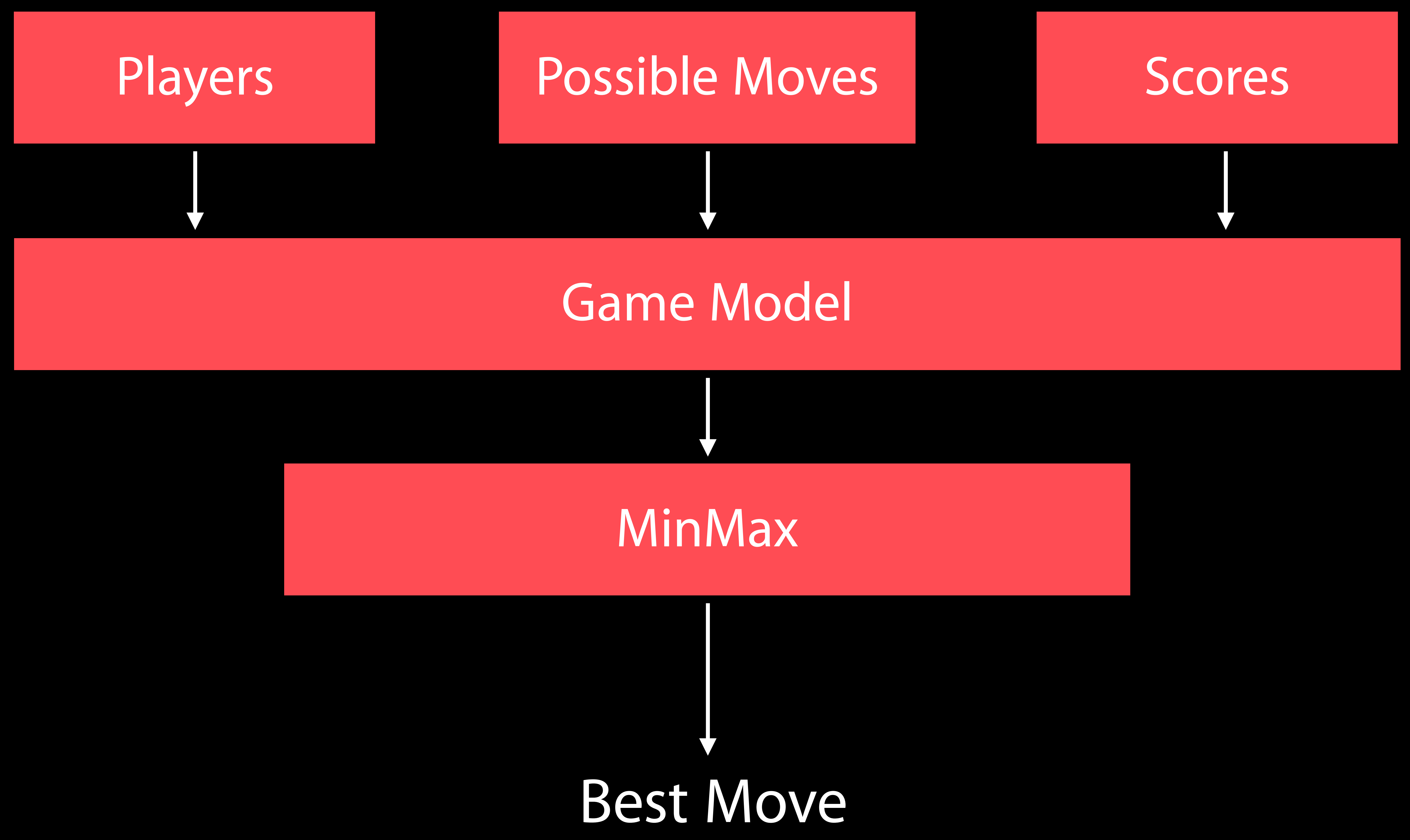
- Any game with discrete moves

Variable difficulty

- Adjust look ahead
- Select suboptimal moves

MinMax AI

Overview



MinMax AI

GKGameModel protocol

Abstract of the current game state

- List of players
- Currently active player
- Player scores
- Possible player moves

Apply moves for players

- Changes game state

GKGameModel

players
activePlayer
`[scoreForPlayer:]`
`[gameModelUpdatesForPlayer:]`
`[applyGameModelUpdate:]`
`[setGameModel:]`

MinMax AI

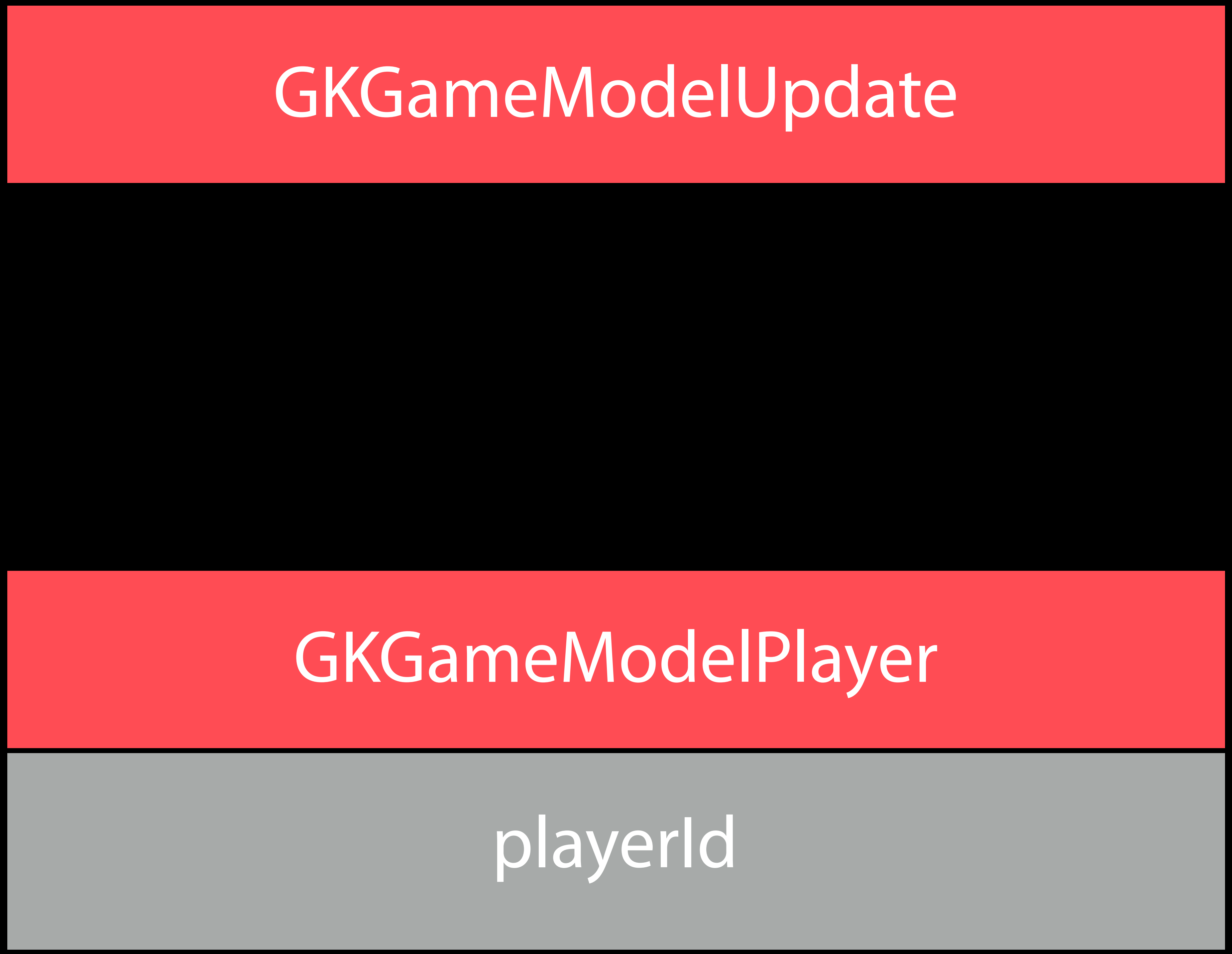
GKGameModel protocols

GKGameModelUpdate

- Abstract of a game move
- Used by MinMax to build decision tree
- Apply to GKGameModel to change state

GKGameModelPlayer

- Abstract for a player of the game
- Players make moves via GKGameModelUpdate



MinMax AI

GKMinmaxStrategist

Operates on a GKGameModel

maxLookAheadDepth is search depth

`[bestMoveForPlayer:]` for optimal outcome

- Ties can be broken at random

`[randomMoveForPlayer:]` for N best moves

Returns a GKGameModelUpdate

GKMinmaxStrategist

gameModel

maxLookAheadDepth

`[bestMoveForPlayer:]`

`[randomMoveForPlayer:]`

MinMax AI

GKMinmaxStrategist example

```
/* ChessGameModel implements GKGameModel */
ChessGameModel *chessGameModel = [ChessGameModel new];
GKMinmaxStrategist *minmax = [GKMinmaxStrategist new];

minmax.gameModel = chessGameModel;
minmax.maxLookAheadDepth = 6;

/* Find the best move for the active player */
ChessGameUpdate *chessGameUpdate =
            [minmax bestMoveForPlayer:chessGameModel.activePlayer];

/* Apply update to the game model */
[chessGameModel applyGameModelUpdate:chessGameUpdate];
```

Demo

Stone Flipper AI

Random Sources

Random Sources

Background

Games have unique random number needs

`rand()` gives us random numbers, but we need more

- Platform-independent determinism
- Multiple sources
- Number distribution

This is where random sources come in

Random Sources

Features

Game quality random sources

- Deterministic
- Serializable
- Industry-standard algorithms

Random distributions

- True random
- Gaussian
- Anti-clustering

NSArray shuffling

Random Sources

GKRandomSource

Base class for random sources

Adopts NSSecureCoding, NSCopying

Guaranteed determinism with same seed

- If no seed is given, one is drawn from a system source

`[sharedRandom]` is system's underlying shared random

- Not deterministic
- Desirable for card shuffling, etc.

Random Source

Random source algorithms

ARC4

- Low overhead, good characteristics

Linear Congruential

- Very low overhead

Mersenne Twister

- High-quality, but memory-intensive

Not suitable for cryptography

GKARC4RandomSource
NSData* seed

GKLinearCongruential…
uint64_t seed

GKMersenneTwister…
uint64_t seed

Random Sources

GKRandomDistribution

Base class for distribution

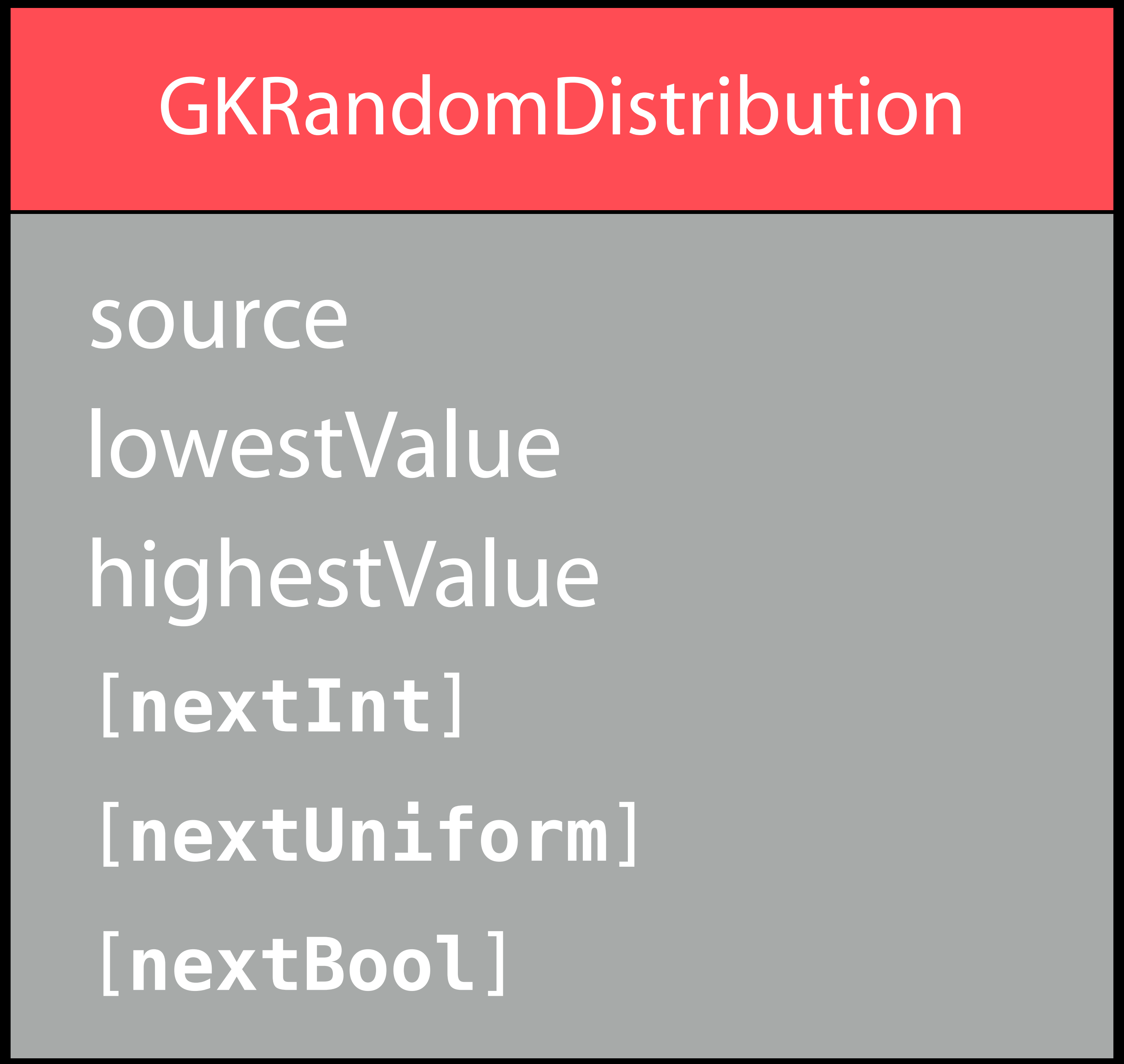
- Pure random distribution

Range between low and high value

`[nextInt]`, `[nextUniform]`, `[nextBool]`

Dice convenience constructors

- `[d6]`
- `[d20]`
- `[die:]`



Random Sources

GKGaussianDistribution

"Bell curve" distribution

- Biased toward mean value
- Decreasing probability away from mean

All values within three standard deviations

Outlying values culled

GKGaussianDistribution

mean
deviation

Range

Output

Random Sources

GKGaussianDistribution

"Bell curve" distribution

- Biased toward mean value
- Decreasing probability away from mean

All values within three standard deviations

Outlying values culled

GKGaussianDistribution
mean deviation

Range

1	2	3	4	5

Output

3	1	3	2	4
2	3	3	4	2
4	3	3	3	5

Random Sources

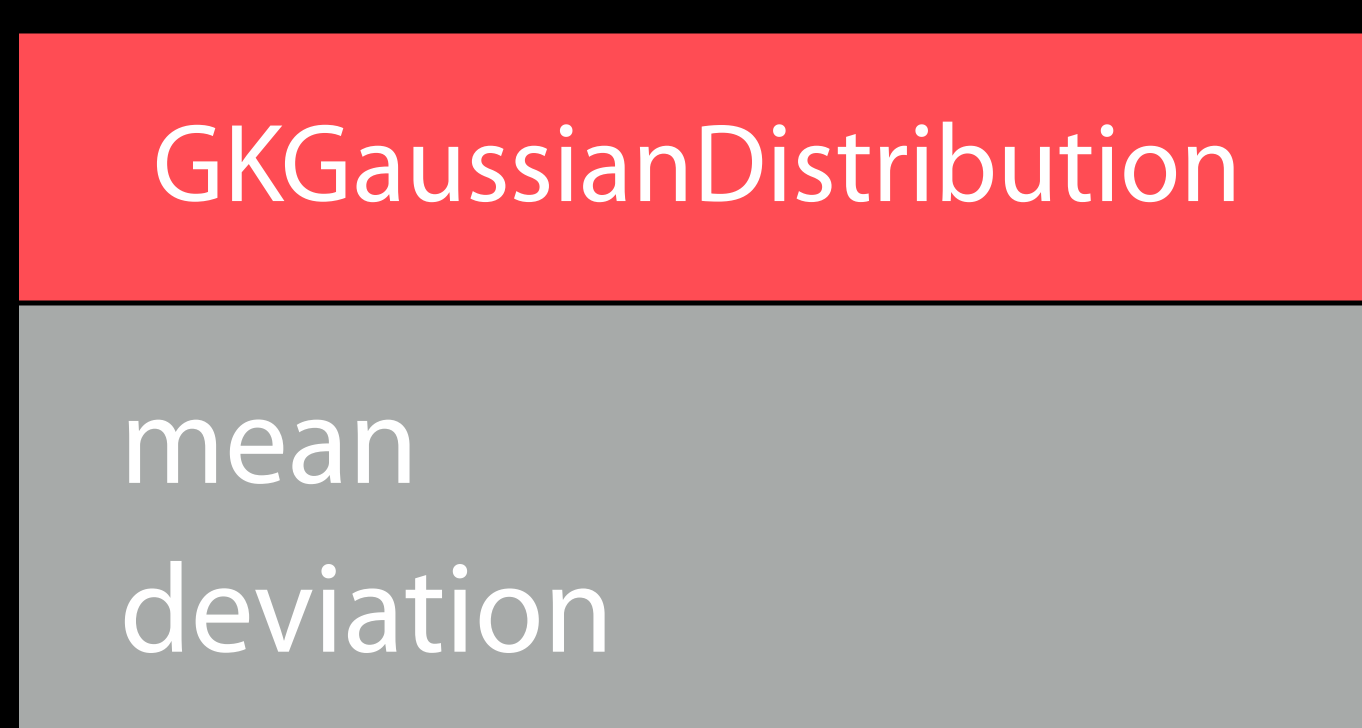
GKGaussianDistribution

"Bell curve" distribution

- Biased toward mean value
- Decreasing probability away from mean

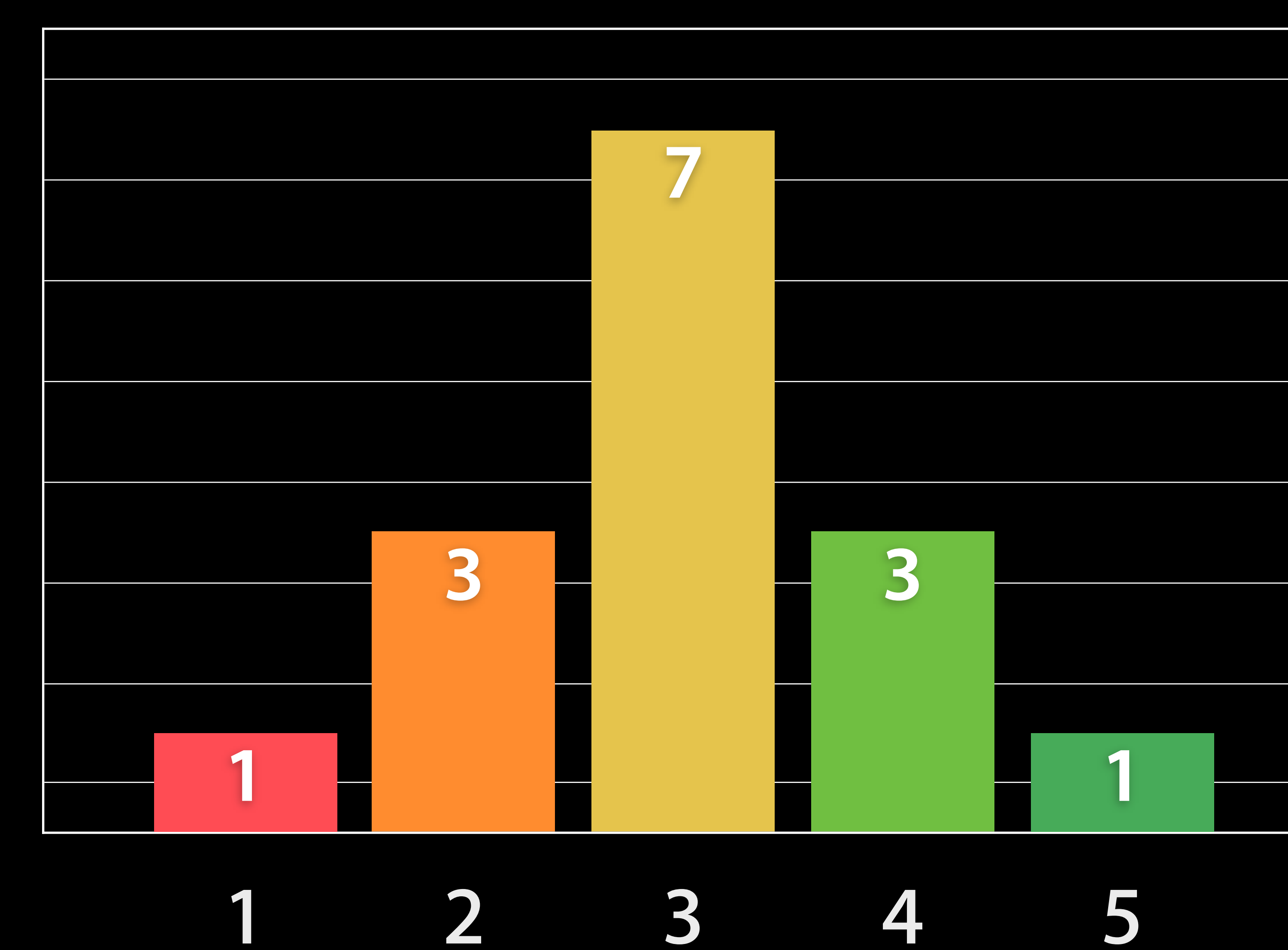
All values within three standard deviations

Outlying values culled



Range

Output



Random Sources

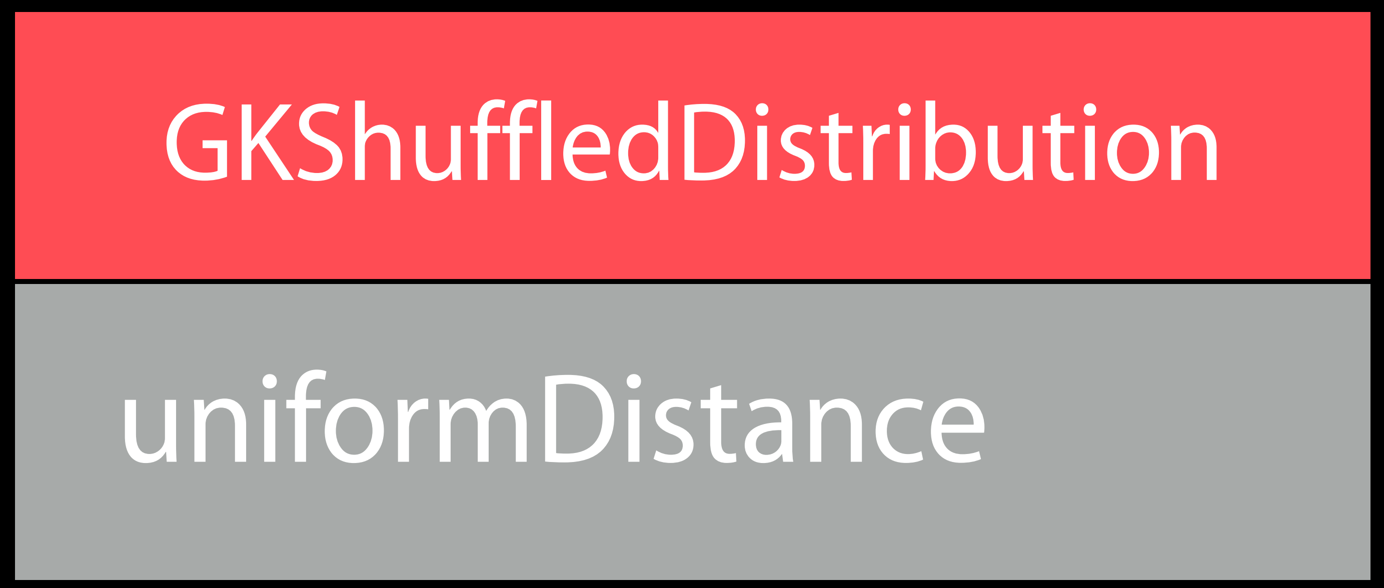
GKShuffledDistribution

Anti-clustering distribution

- Reduces or eliminates "runs"
- Still random over time

uniformDistance defines local reduction

- 0.0 = pure random
- 1.0 = all values different



Range

Output

Random Sources

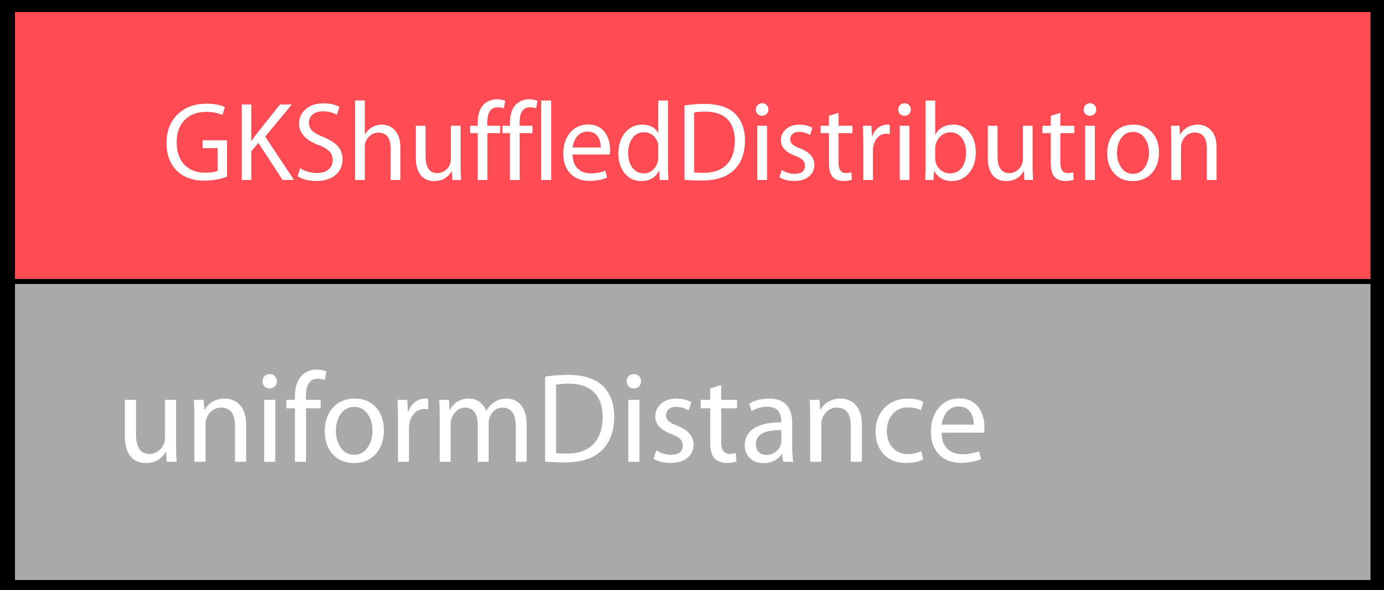
GKShuffledDistribution

Anti-clustering distribution

- Reduces or eliminates "runs"
- Still random over time

uniformDistance defines local reduction

- 0.0 = pure random
- 1.0 = all values different



Range

1	2	3	4	5

Output

3	4	1	2	5
1	3	5	4	2
4	2	1	5	3

Random Sources

GKShuffledDistribution

Anti-clustering distribution

- Reduces or eliminates "runs"
- Still random over time

uniformDistance defines local reduction

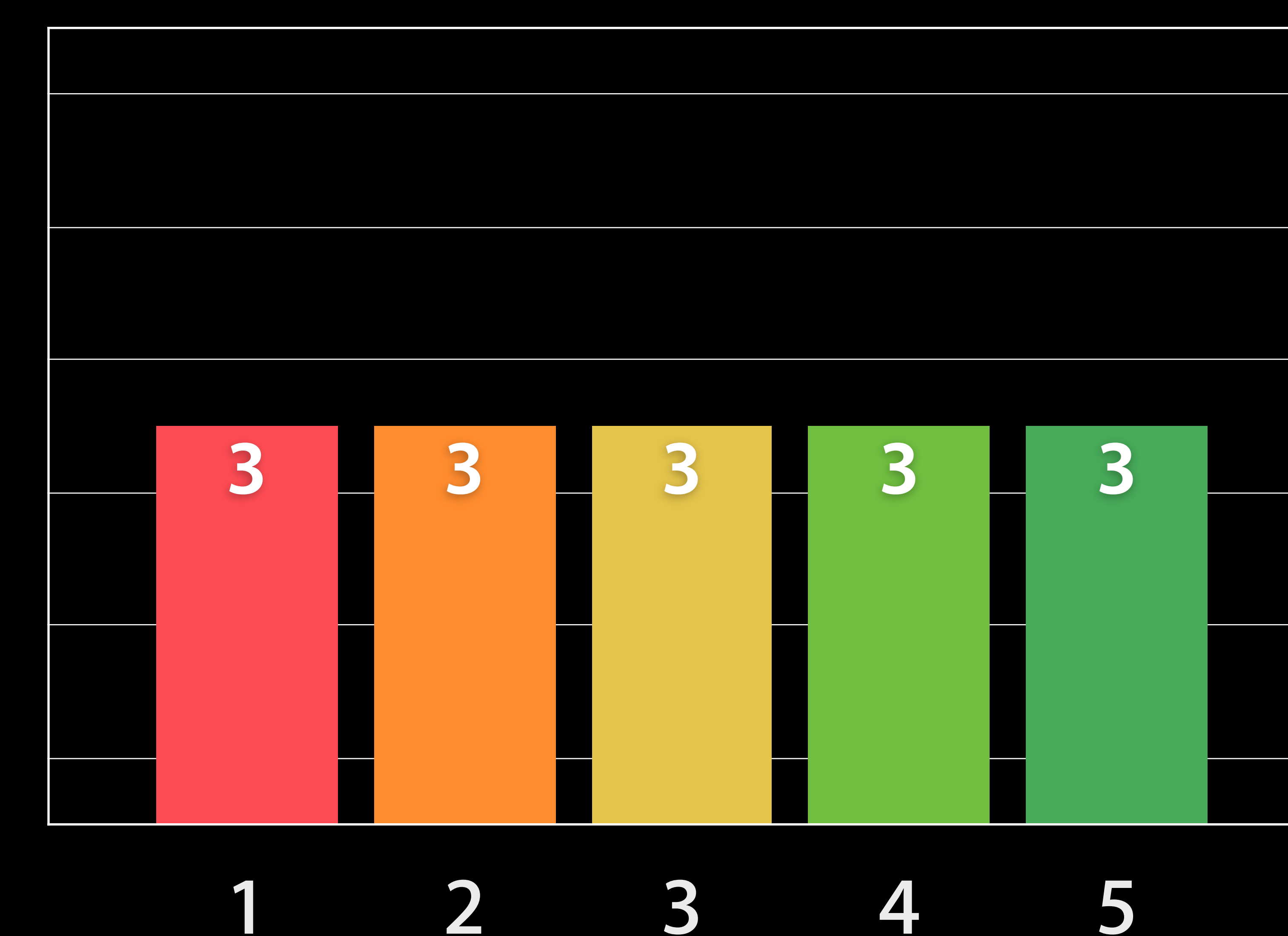
- 0.0 = pure random
- 1.0 = all values different

GKShuffledDistribution

uniformDistance

Range

Output



Random Sources

Simple usage

```
/* Create a six-sided die with its own random source */
let d6 = GKRandomDistribution.d6()

/* Get die value between 1 and 6 */
let choice = d6.nextInt()
```

Random Sources

Simple usage

```
/* Create a twenty-sided die with its own random source */
let d20 = GKRandomDistribution.d20()

/* Get die value between 1 and 20 */
let choice = d20.nextInt()
```

Random Sources

Simple usage

```
/* Create a custom 256-sided die with its own random source */
let d256 = GKRandomDistribution.die(lowest:1, highest:256)

/* Get die value between 1 and 256 */
let choice = d256.nextInt()
```

Random Sources

Intermediate usage

```
/* Create a twenty-sided die with a bell curve bias */
let d20 = GKGaussianDistribution.d20()

/* Get die value between 1 and 20 that is most likely to be around 11 */
let choice = d20.nextInt()
```

Random Sources

Intermediate usage

```
/* Create a twenty-sided die with no clustered values — fair random */
let d20 = GKShuffledDistribution.d20()

/* Get die value between 1 and 20 */
let choice = d20.nextInt()

/* Get another die value that is not the same as ‘choice’ */
let secondChoice = d20.nextInt()
```

Random Sources

Intermediate usage

```
/* Make a deck of cards */
var deck = [Ace, King, Queen, Jack, Ten]

/* Shuffle them */
deck = GKRandomSource.sharedRandom().shuffle(deck)
/* possible result - [Jack, King, Ten, Queen, Ace] */

/* Get a random card from the deck */
let card = deck[0]
```

Rule Systems

Joshua Boggs

Rule Systems

Game ingredients

A game consists of three elements:

Nouns (Properties)

- Position, speed, health, equipment, etc.

Verbs (Actions)

- Run, jump, use item, accelerate, etc.

Rules

- How your nouns and verbs interact

Rule Systems

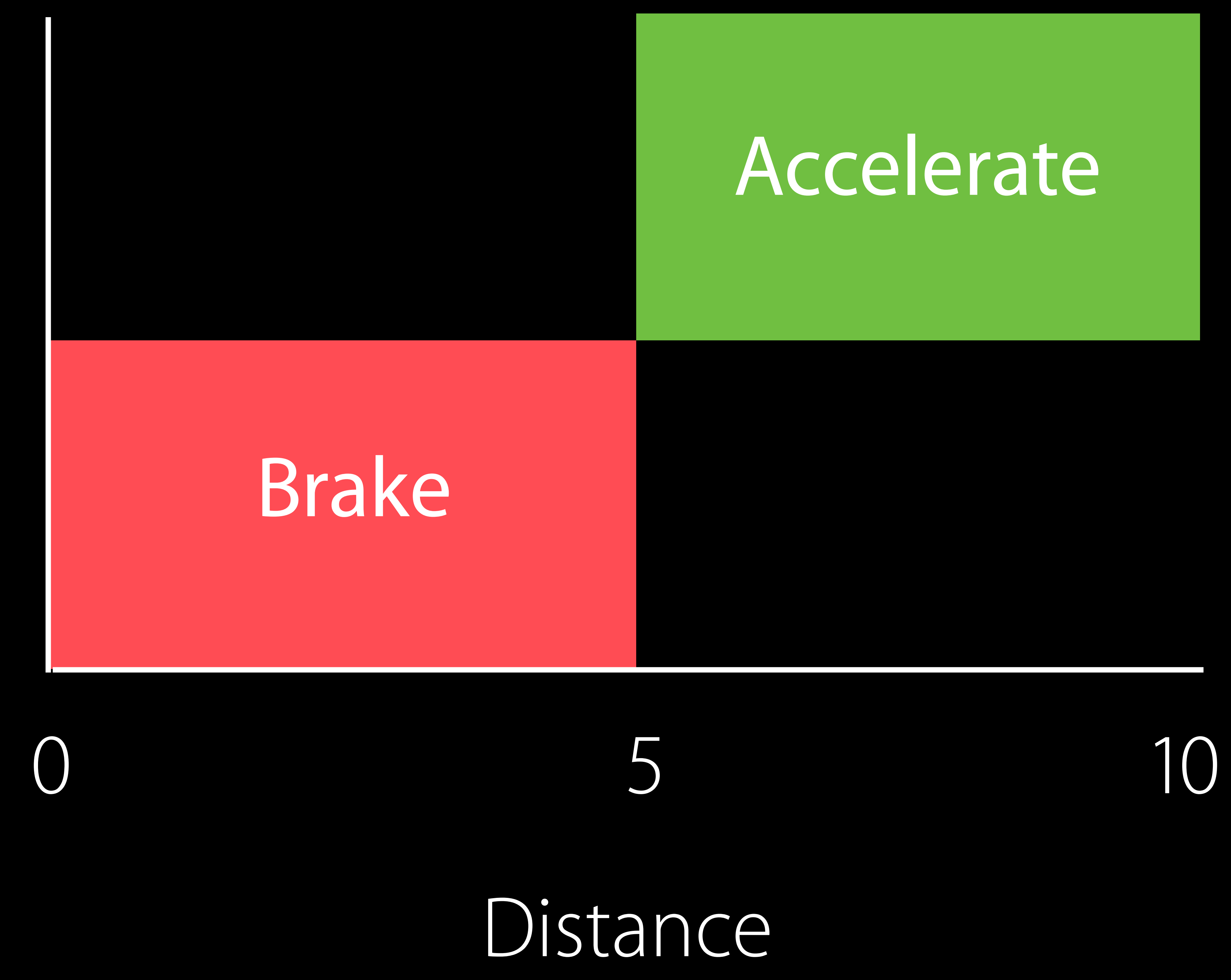
What is a rule system?

Binary Driver AI

- Input is distance
- Output is either `[slowDown]` or `[speedUp]`

Conditional

```
/* Test is distance */
if (car.distance < 5) {
  [car slowDown];
}
else if (car.distance >= 5) {
  [car speedUp];
}
```



Rule Systems

What is a rule system?

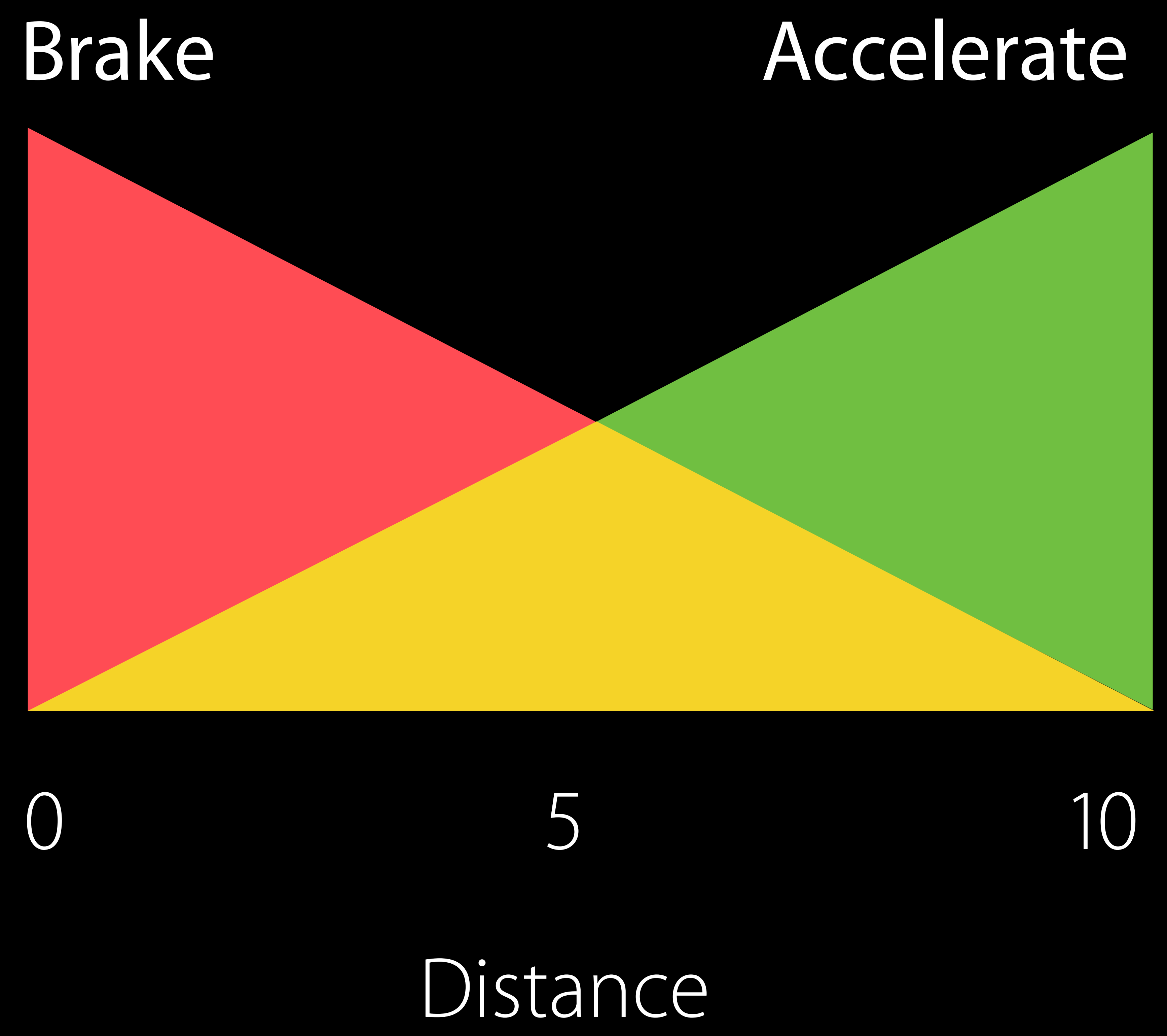
Fuzzy Driver AI

- Input is distance
- Rules output facts

Facts

```
closeness = 1.0f – distance / 10.0f;
farness = distance / 10.0f;
```

Can be both close and far



Rule Systems

Motivation

Complex reasoning with fuzzy logic

- Facts can be grades of true
- Fuzzy logic deals with approximations

Separate what we should do from how we should do it

- State facts about the world
- Take deferred actions based on those facts

Rule Systems

GKRule

A boolean predicate and an action

- Predicate matches against facts and state
- Action fires only if predicate is true

Action can be simple `[assertFact:]`

- Or complex block

Serializable

GKRule
`[ruleWithBlockPredicate:action:]`
`[ruleWithPredicate:assertingFact:grade:]`
`[performActionWithSystem:]`
`[evaluatePredicateWithSystem:]`
salience

Rule Systems

Approximation

Rule Systems provide approximate answers to questions

- How close am I to the car in front?
  - Very far

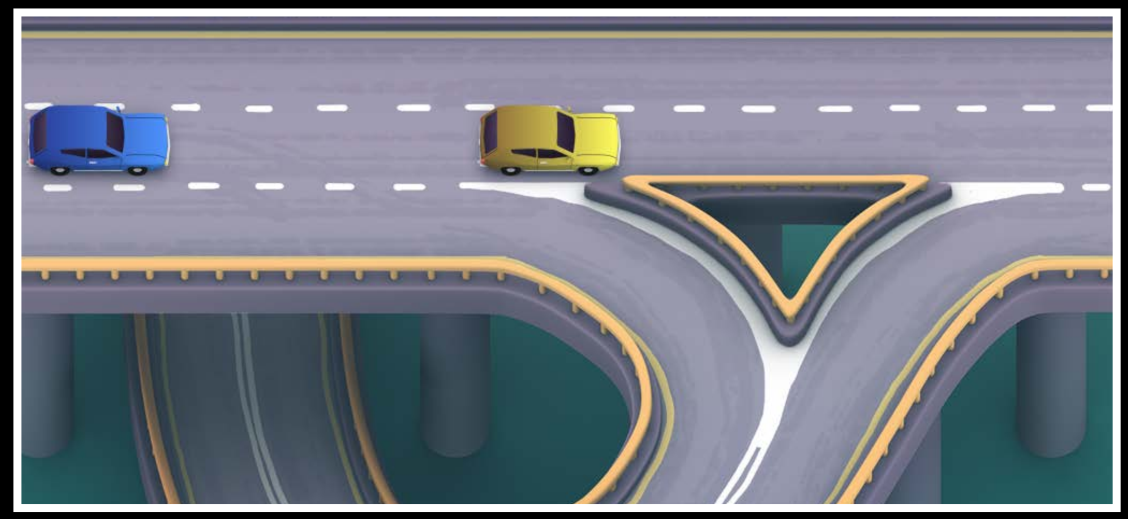
    ```
    farGrade = 1.0f;
    ```

  - Somewhere in between

    ```
    farGrade = closeGrade = 0.5;
    ```

  - ‘Close-ish’

    ```
    closeGrade = 0.75f;
    farGrade = 0.25f;
    ```



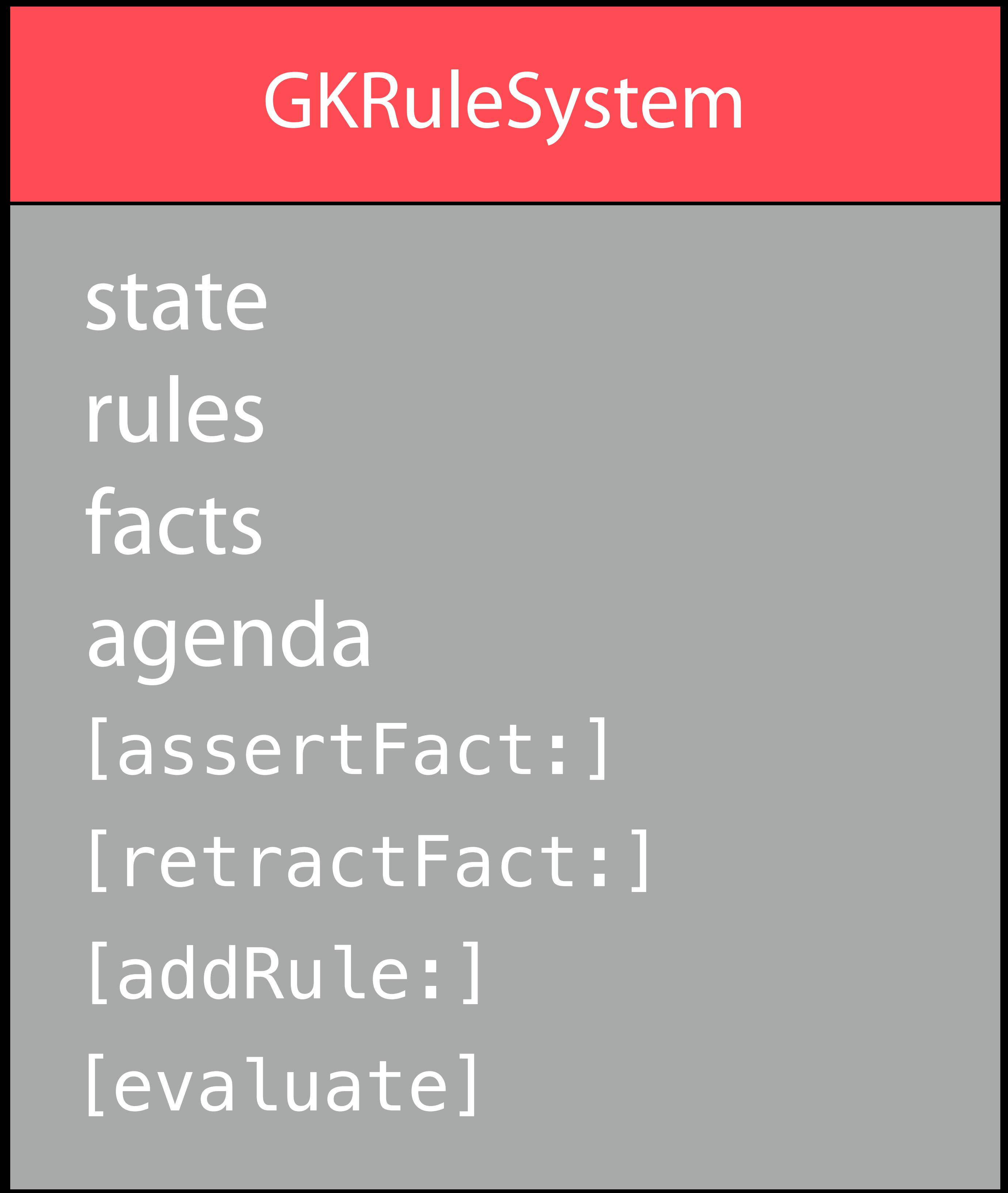
Rule Systems

GKRuleSystem

An ordered collection of rules and facts

Assert facts by calling `[evaluate]`

- Rules use the state dictionary as input
- Facts array holds the asserted output
- Repeat evaluation for each new fact
- `[reset]` and clear old facts to repeat



Rule Systems

Code example

```
/* Make a rule system */
GKRuleSystem* sys = [[GKRuleSystem alloc] init];

/* Getting distance and asserting facts */
float distance = sys.state[@“distance”];
[sys assertFact:@“close” grade:1.0f - distance / kBrakingDistance];
[sys assertFact:@“far” grade:distance / kBrakingDistance];

/* Grade our facts - farness and closeness */
float farness = [sys gradeForFact@“far”];
float closeness = [sys gradeForFact@“close”];

/* Derive Fuzzy acceleration */
float fuzzyAcceleration = farness - closeness;
[car applyAcceleration:fuzzyAcceleration withDeltaTime:seconds];
```

Demo

Traffic Toy

Rule Systems

Best practices

GKRuleSystem is an isolated system

- `state` is a snapshot of your game world
- Use many simple rules and assert facts about the game world

Facts are approximate, it's up to you to decide how to use them

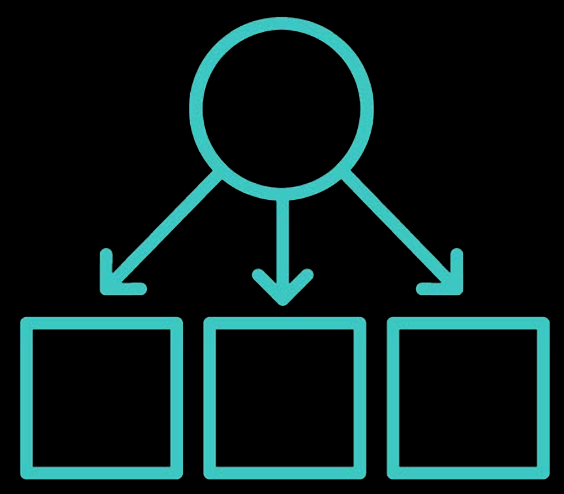
- Grade of a fact is the system's confidence in it
- Use fuzzy logic for more complex reasoning

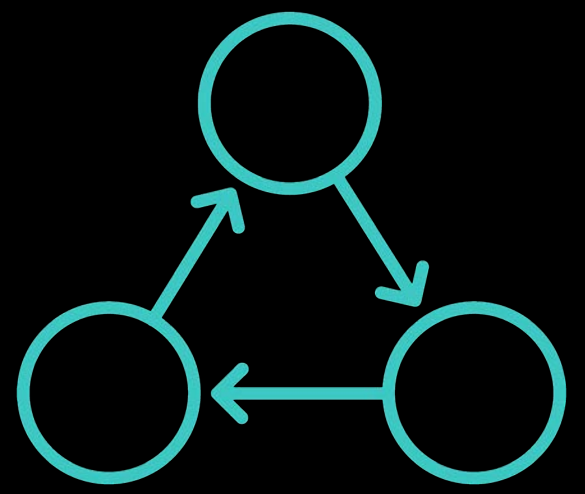
Wrap Up

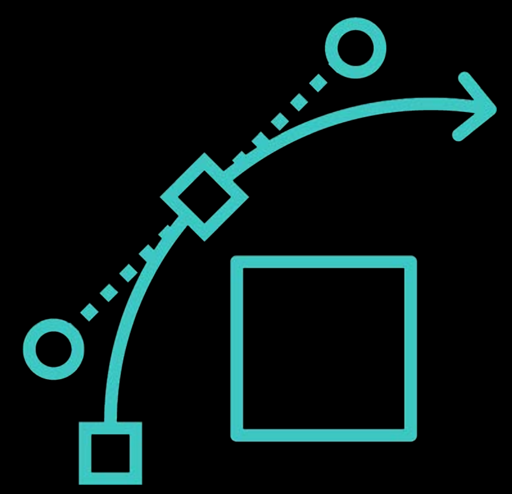
Bruno Sommer

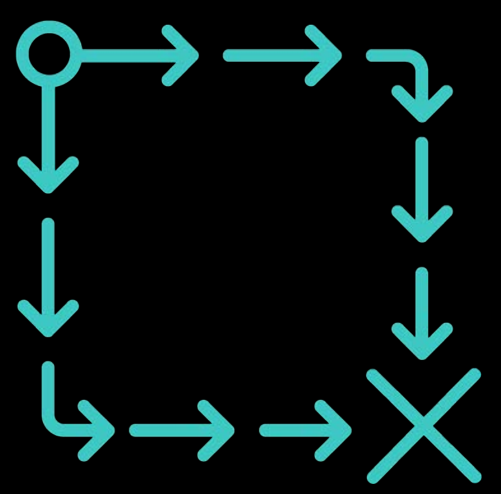
GameplayKit

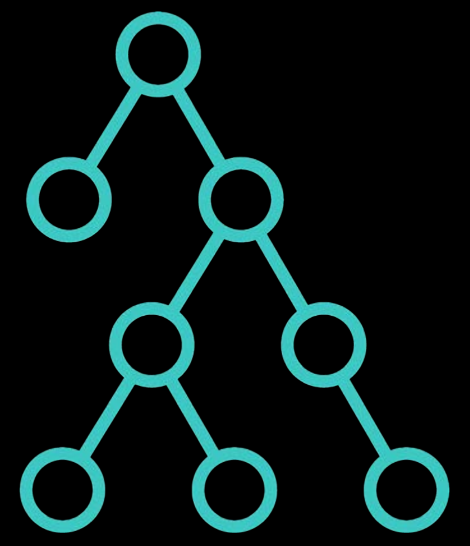
Bringing game ideas to life


Entities & Components


State Machines


Agents


Pathfinding


MinMax AI


Random Sources


Rule Systems

Code Samples

`DemoBots`

- SpriteKit game, lots of GameplayKit coverage

`FourInARow`

- Covers MinMaxAI

`AgentsCatalog`

- Covers agents, behaviors and goals

Related Sessions

What's New In SpriteKit	Mission	Wednesday 10:00 AM
Going Social with ReplayKit and Game Center	Mission	Wednesday 1:30 PM
Enhancements to SceneKit	Nob Hill	Wednesday 2:30 PM
Deeper into GameplayKit with DemoBots	Mission	Thursday 1:30 PM

Labs

Game Controllers Lab	Graphics D	Thursday 2:30 PM
Game Controllers Lab	Graphics D	Friday 9:00 AM
GameplayKit Lab	Graphics C	Thursday 2:30 PM
GameplayKit Lab	Graphics C	Friday 12:00 PM
SpriteKit Lab	Graphics C	Friday 9:00 AM

More Information

Documentation and Videos
http://developer.apple.com

Apple Developer Forums
http://developer.apple.com/forums

Developer Technical Support
http://developer.apple.com/support/technical

General Inquiries
Allan Schaffer, Game Technologies Evangelist
aschaffer@apple.com

WWDC15