Designing Games with Sprite Kit

Bringing your art to life

Session 503

Norman Wang

Sprite Kit Recap

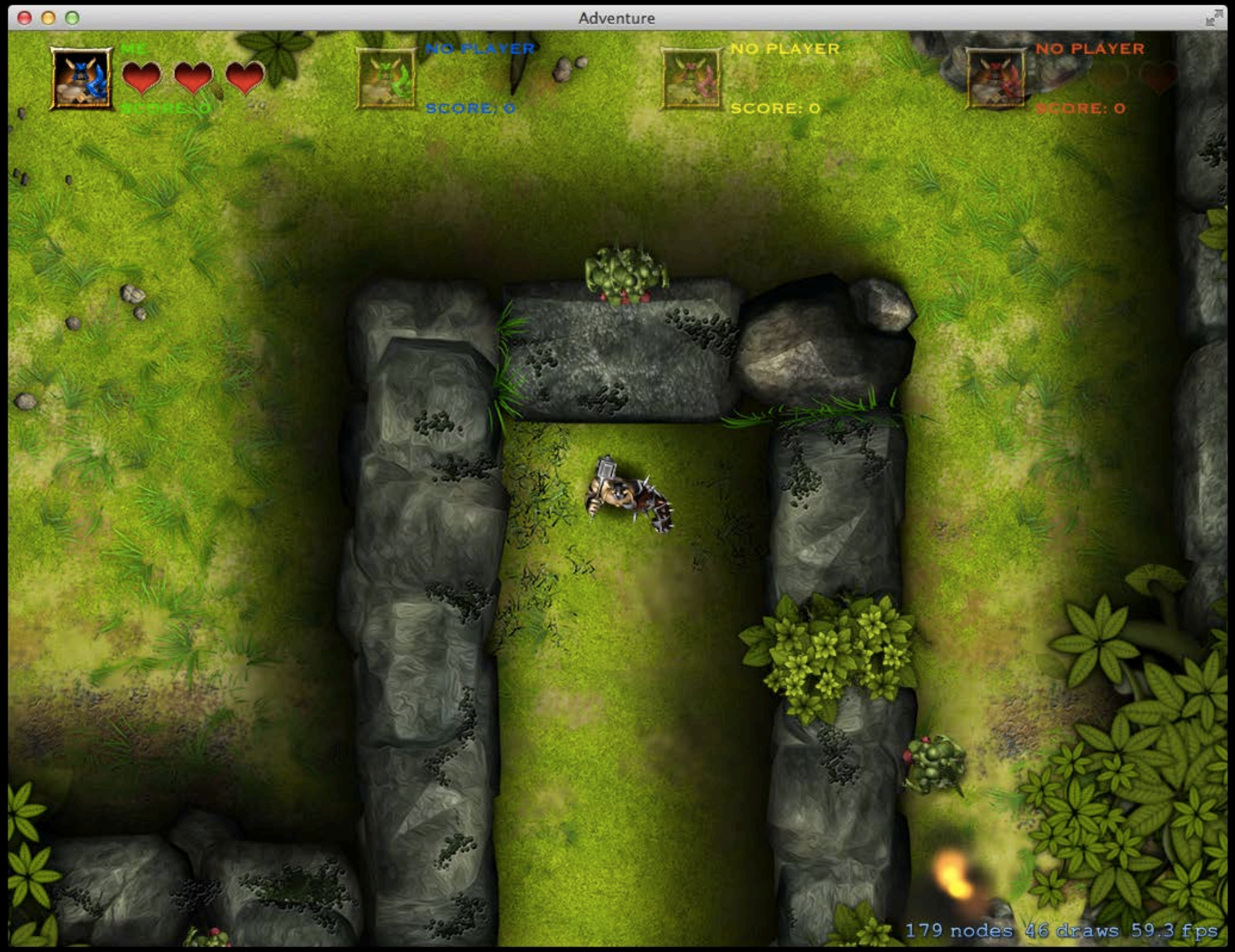
- High performance 2D rendering framework
- Built-in physics support
- Cross platform between OS X and iOS
- Packaged with runtime and tools
- Features games need
    - Sprites, shapes and particles
    - Non-linear animation
    - Audio, video, and visual effects

Agenda

- Adventure art pipeline
- Visual effects
- Building Adventure
- Developing custom tools
- Best practices

What Goes into Adventure?

- Adventure manages a lot of data
  - Artwork
  - Sounds
  - Particles
  - Physics
  - Visual effects
  - Collision and level maps

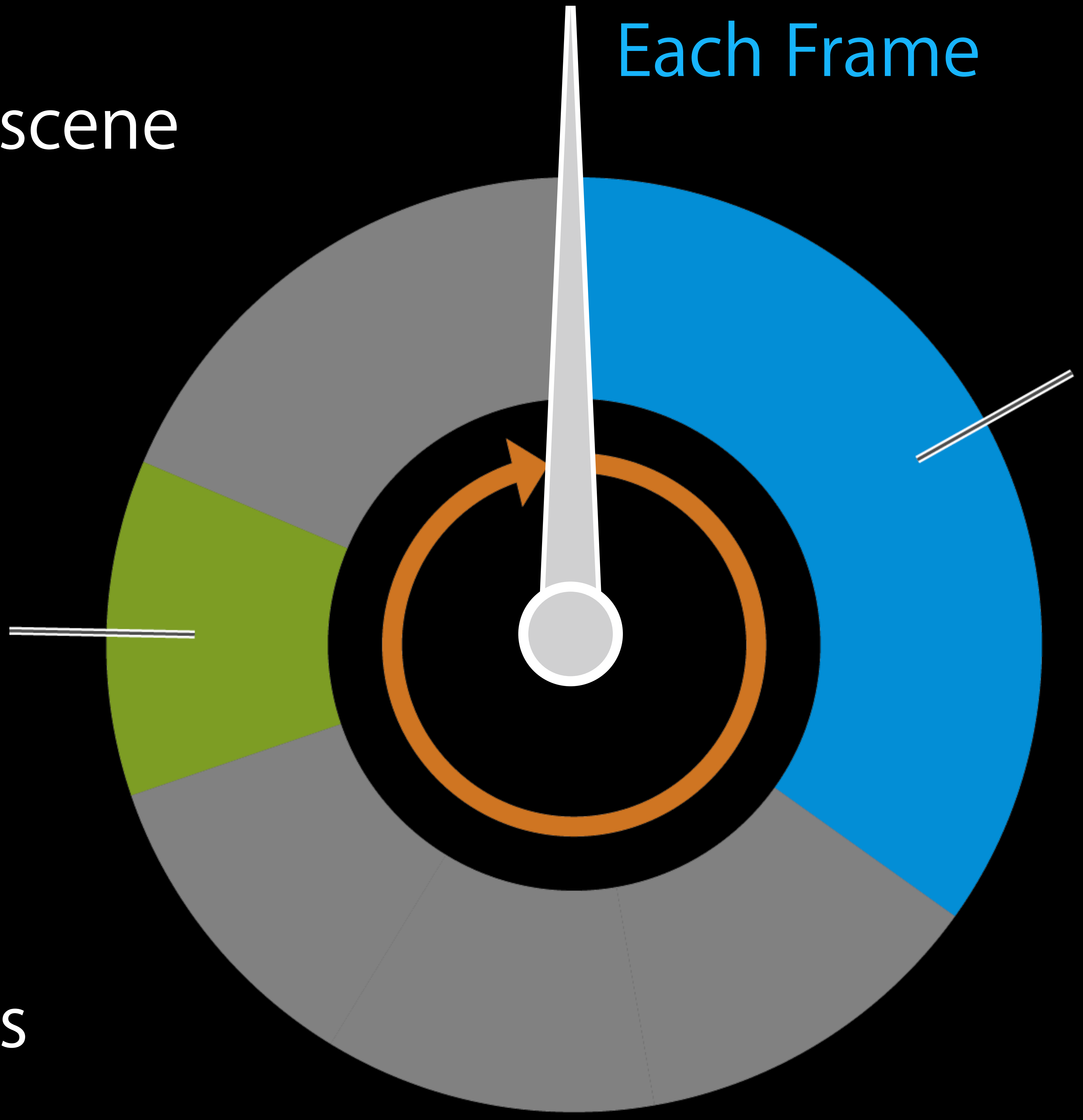


Adventure Startup Sequence

Building the world

- Load all the shared resources at app launch
    - Parallel async loading
- Create an instance of the scene
- Create and set initial positions for all the nodes
- Add the collision walls
- Present the scene using an `SKView`
- Register game controller notifications

The Adventure Game Loop



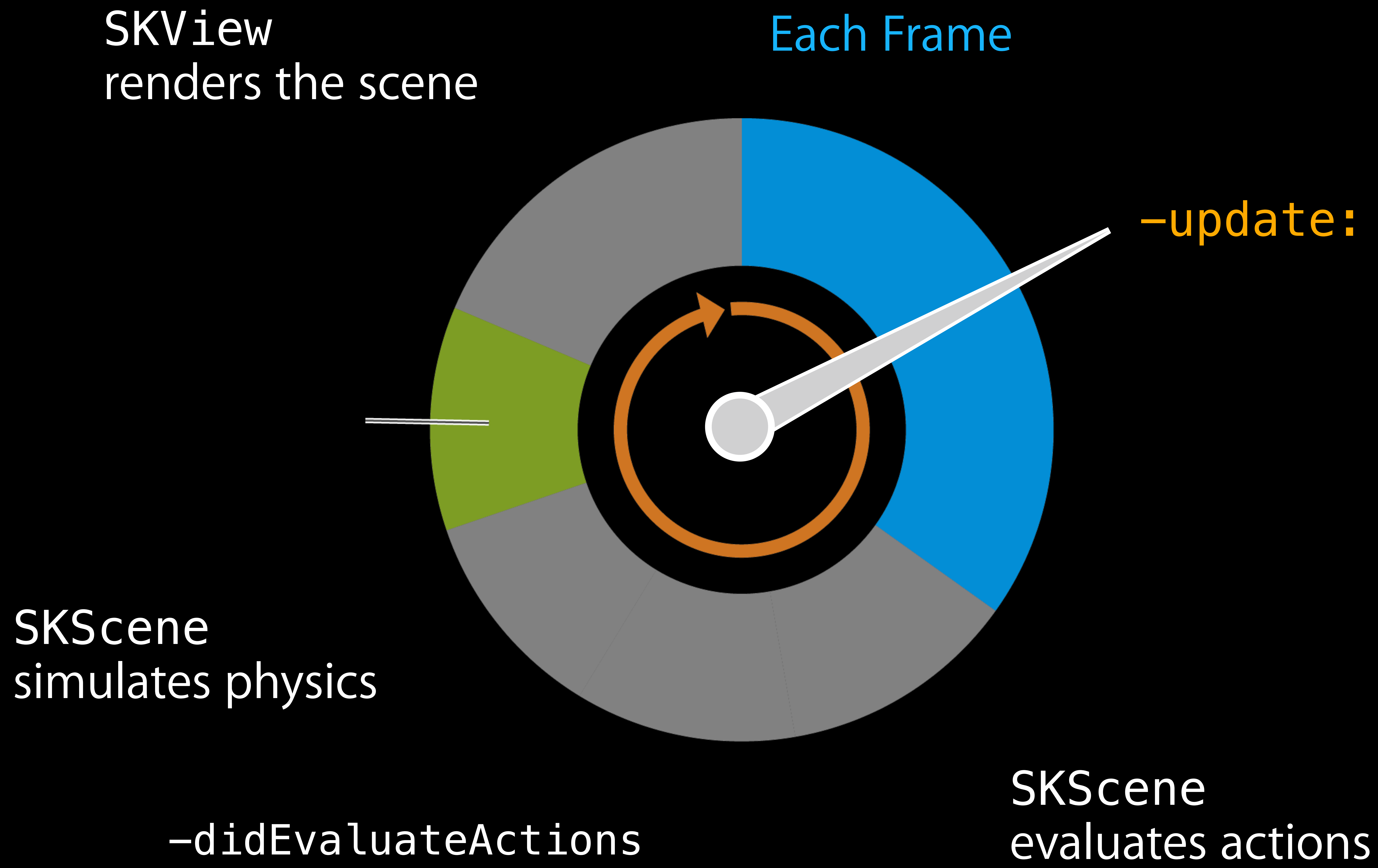
SKView
renders the scene

SKScene
simulates physics

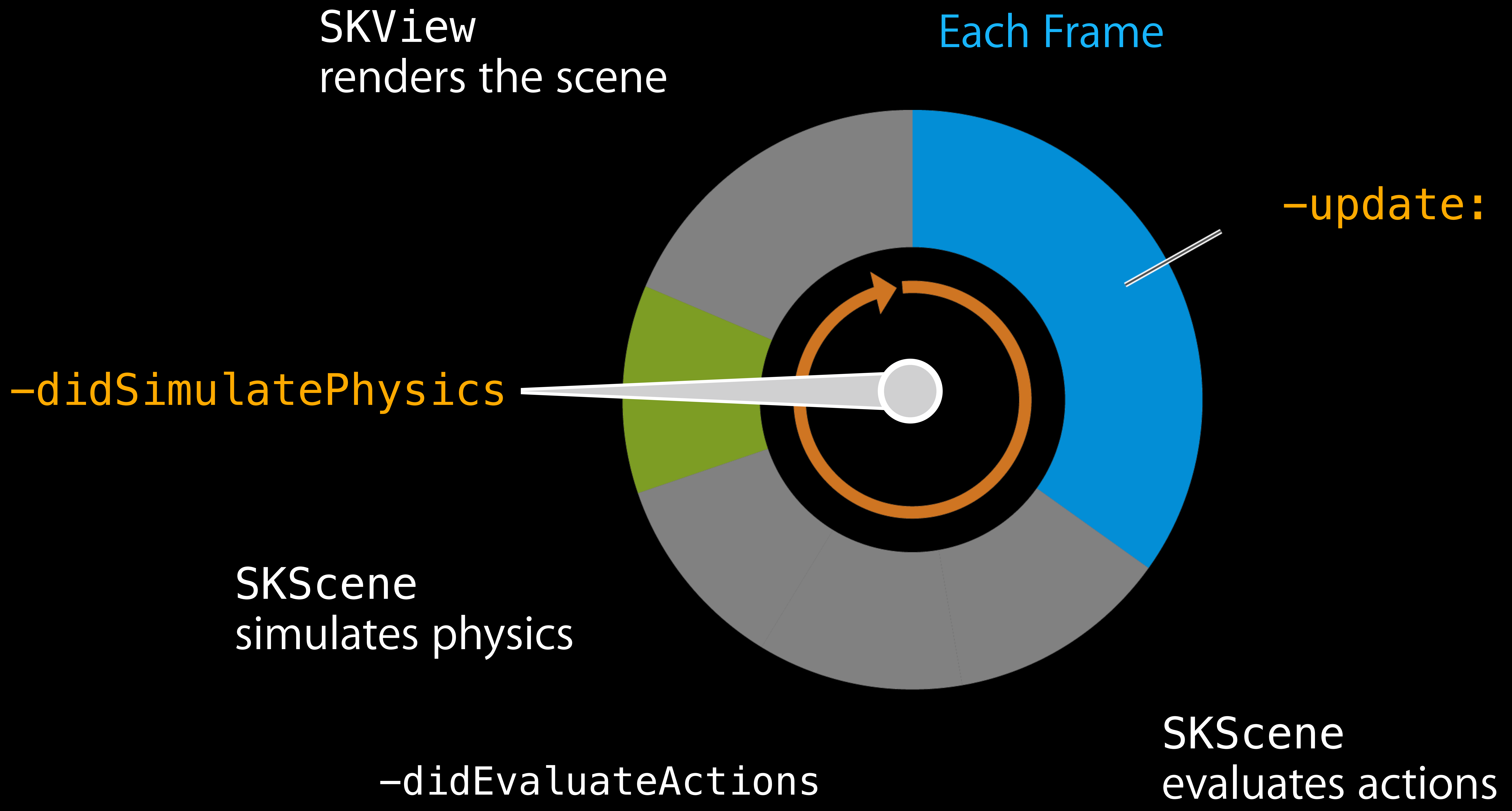
-didEvaluateActions

SKScene
evaluates actions

The Adventure Game Loop



The Adventure Game Loop



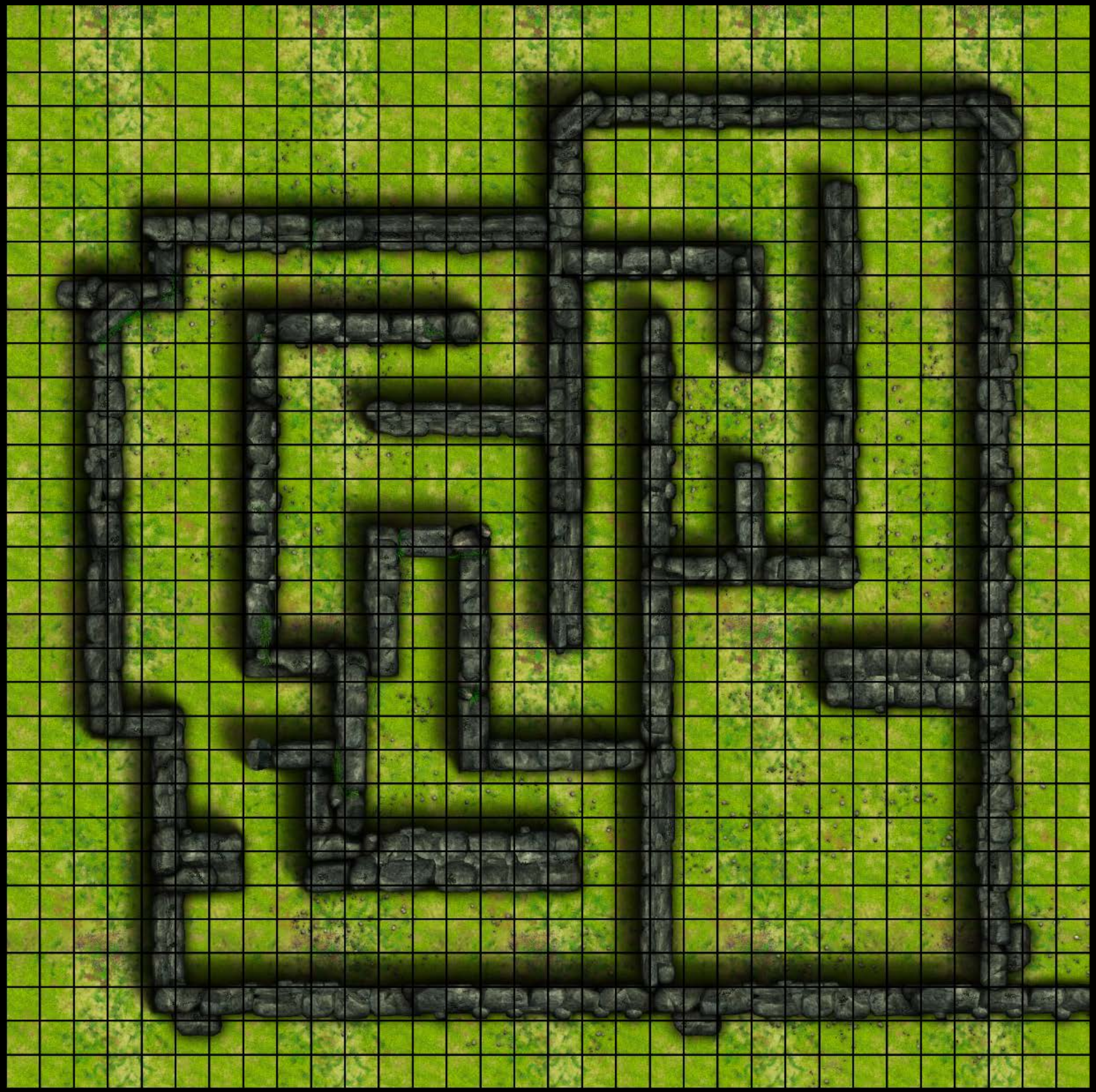
Adventure Assets Challenges

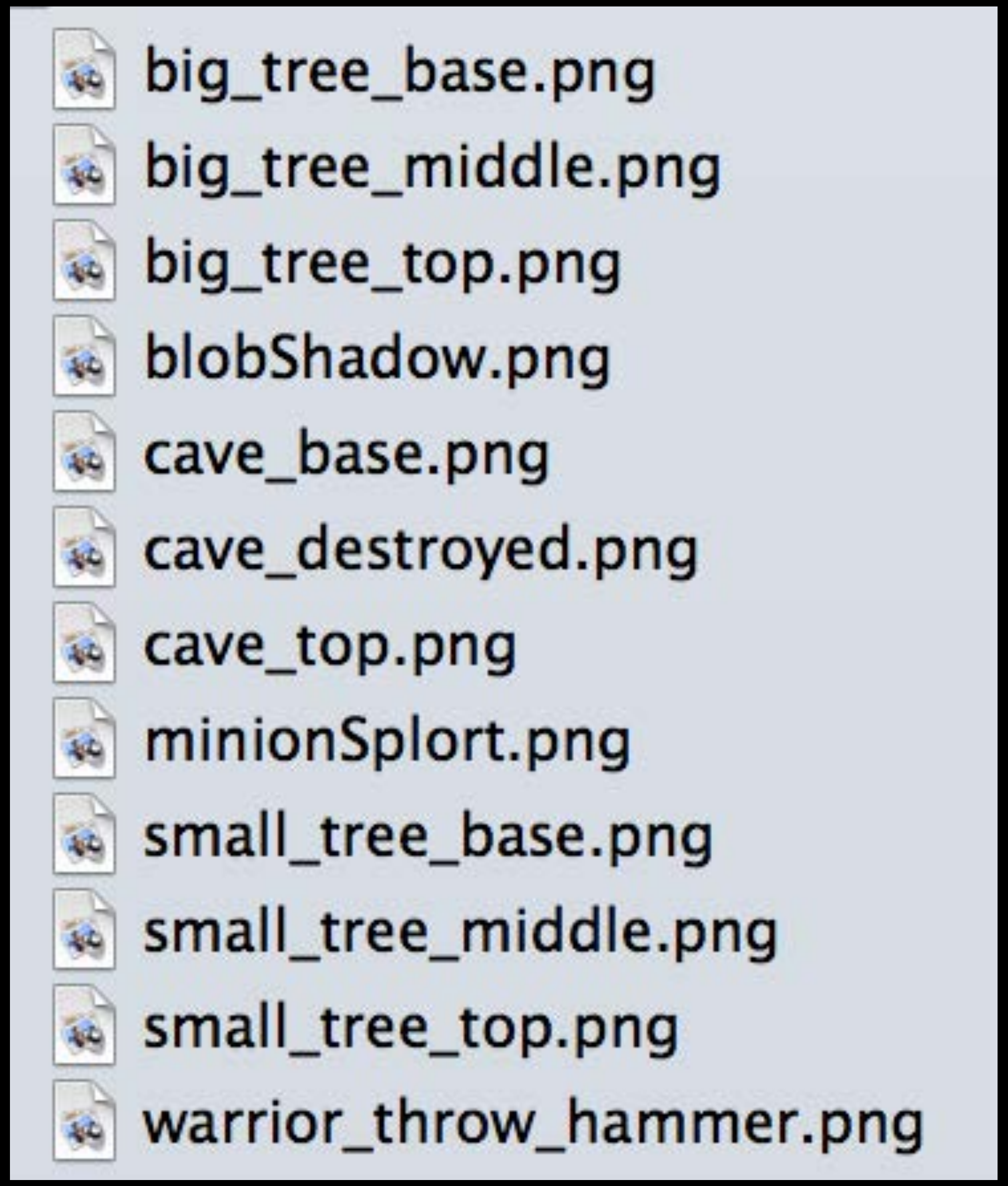
- Construction of scene depends on assets
- Huge data set for each art category
    - Tile based rendering
        - 1024 background tiles
    - Lots of animation frames for characters
    - Lots of visual effects
    - Complex level design with collision mapping

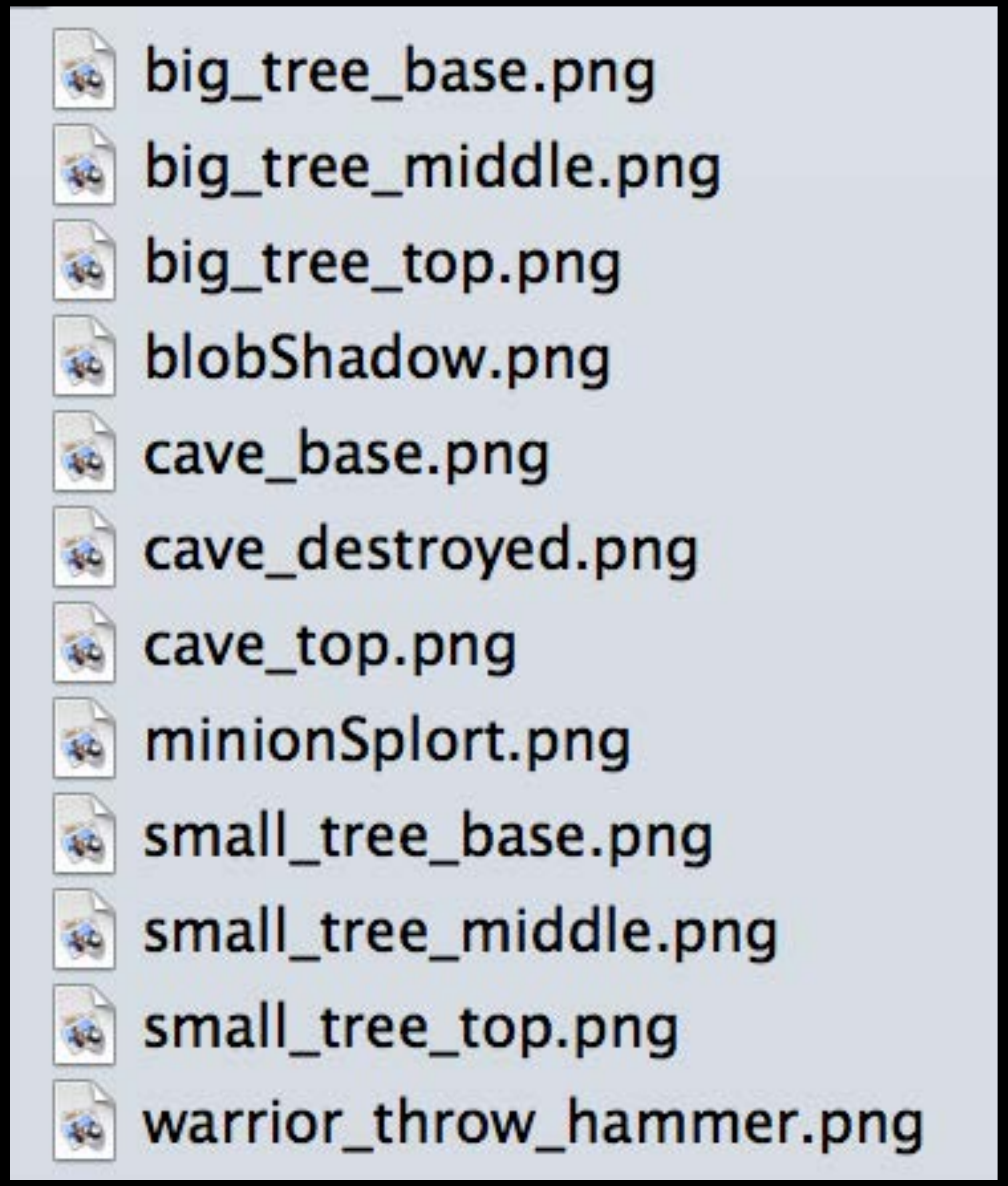
Adventure Artwork Pipeline

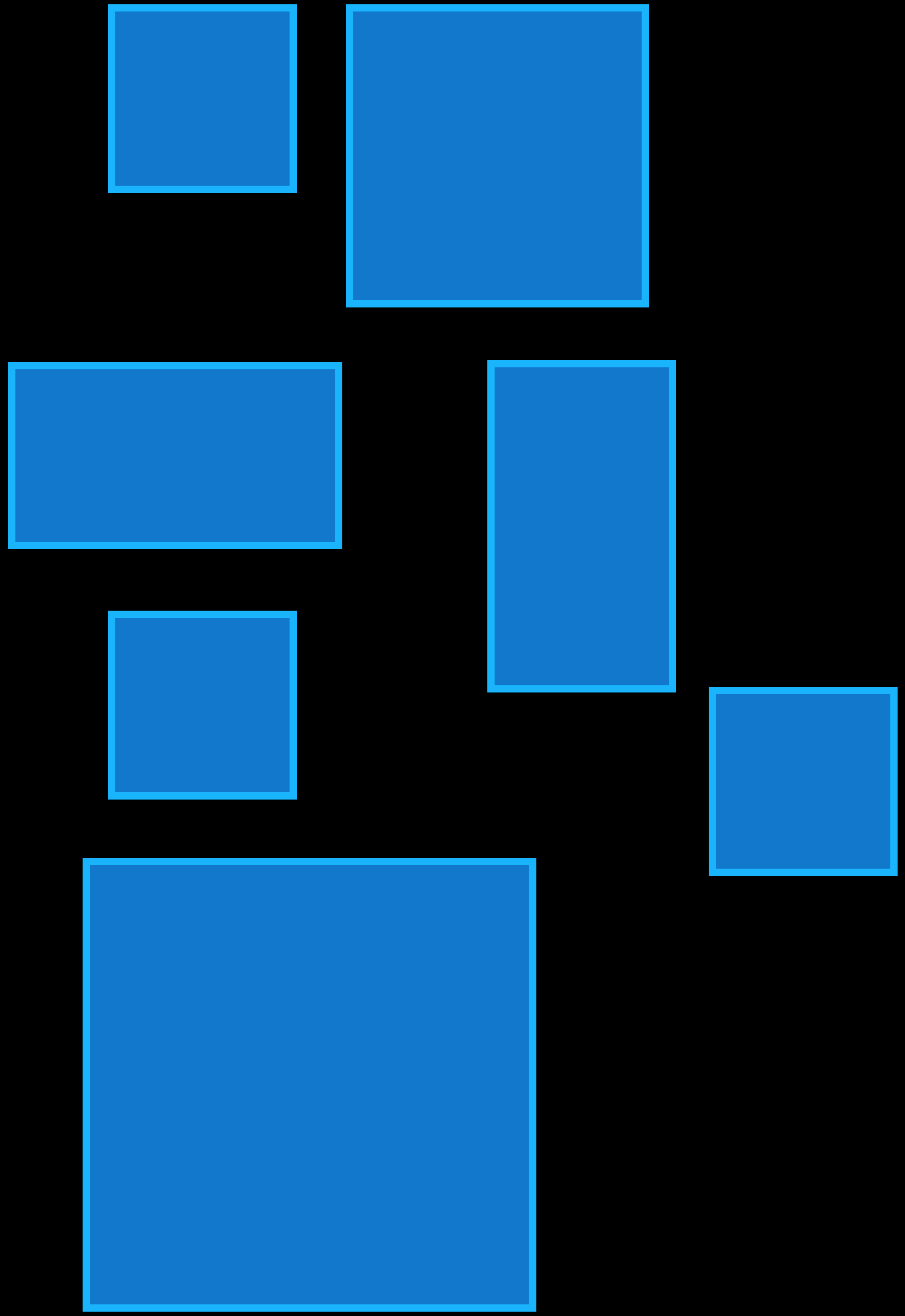
Adventure Textures

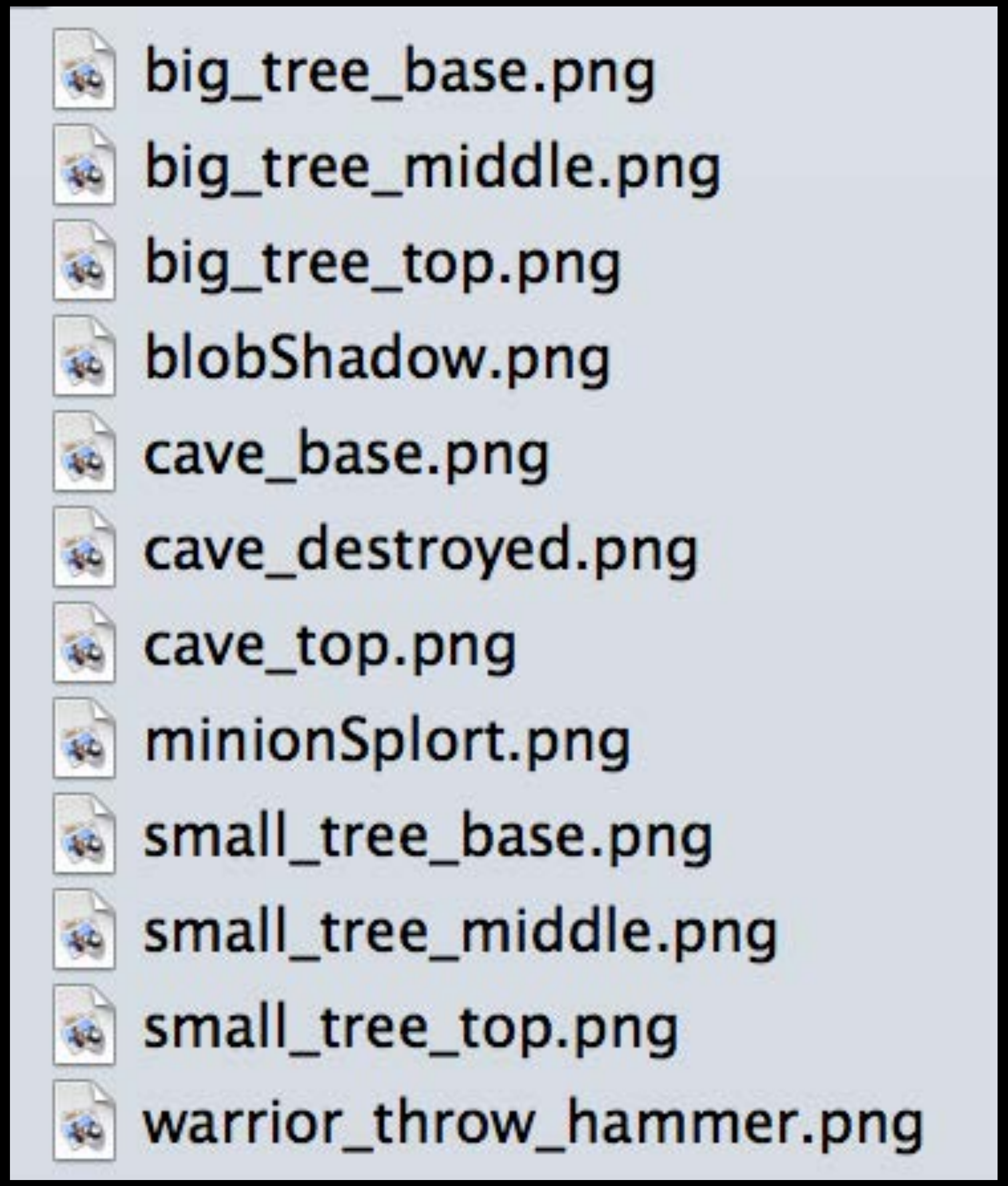
- Adventure has over 1600 texture files
- The game uses texture atlases for all textures
  - Character animation frames
  - Background tiles
  - Environmental elements
    - e.g. trees, caves, and projectiles

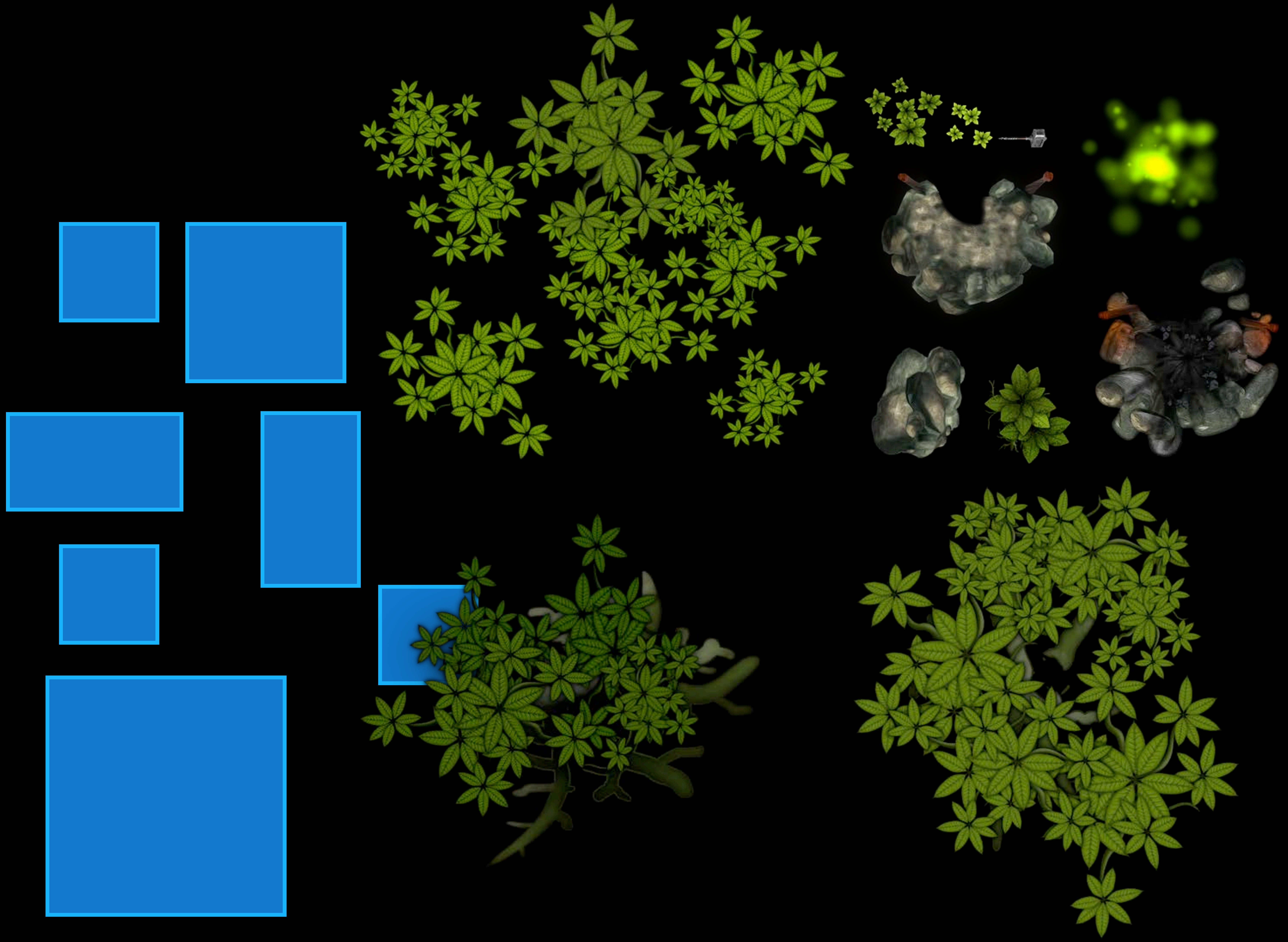



big_tree_base.png
big_tree_middle.png
big_tree_top.png
blobShadow.png
cave_base.png
cave_destroyed.png
cave_top.png
minionSplort.png
small_tree_base.png
small_tree_middle.png
small_tree_top.png
warrior_throw_hammer.png


big_tree_base.png
big_tree_middle.png
big_tree_top.png
blobShadow.png
cave_base.png
cave_destroyed.png
cave_top.png
minionSplort.png
small_tree_base.png
small_tree_middle.png
small_tree_top.png
warrior_throw_hammer.png




big_tree_base.png
big_tree_middle.png
big_tree_top.png
blobShadow.png
cave_base.png
cave_destroyed.png
cave_top.png
minionSplort.png
small_tree_base.png
small_tree_middle.png
small_tree_top.png
warrior_throw_hammer.png



Use a Texture Atlas

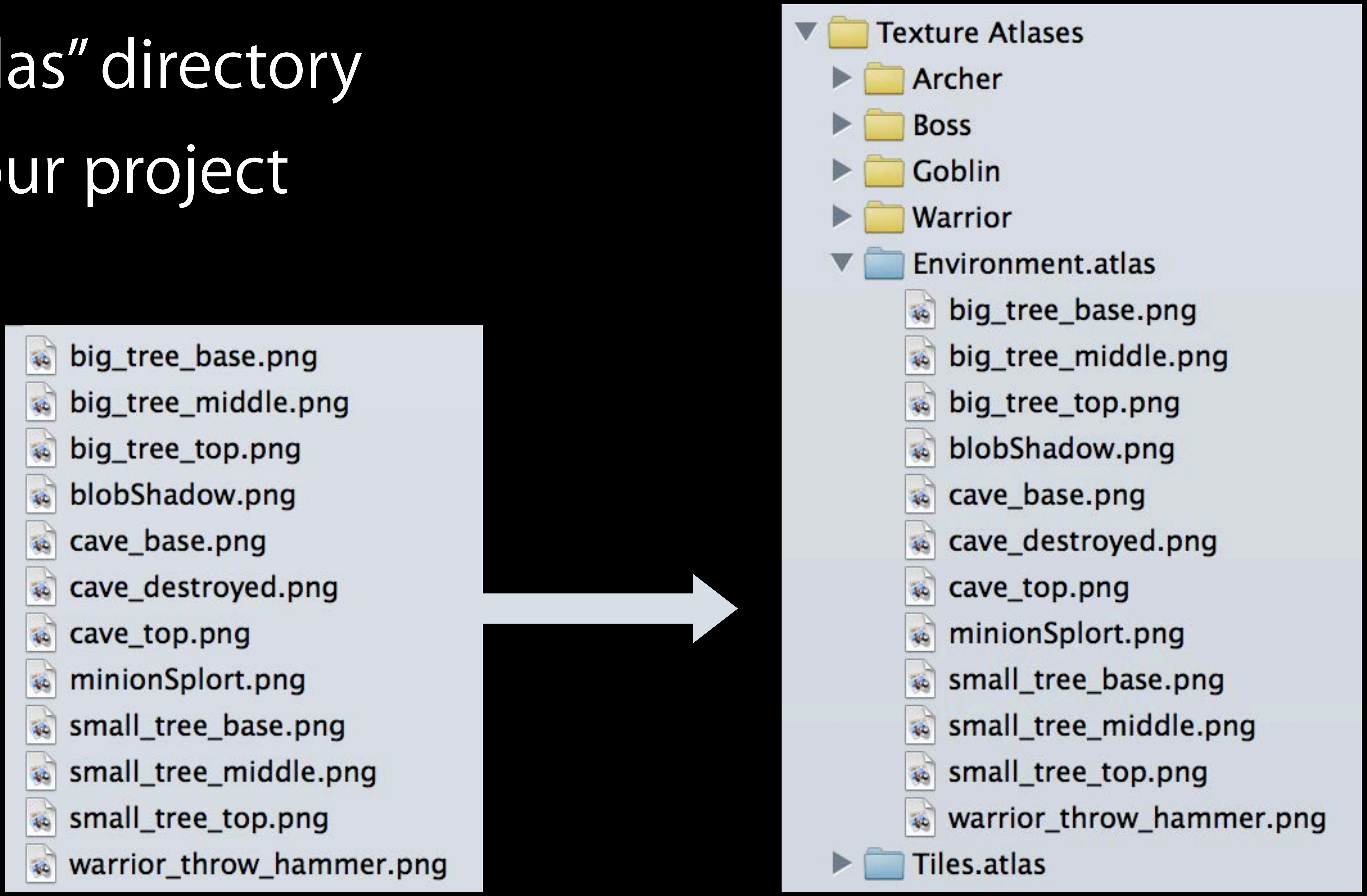
- Minimizes state changes
  - Enables Sprite Kit to batch draw calls
- Minimizes disk I/O
- Minimizes memory footprint and optimizes layout
- Can draw unusually shaped textures

Creating a Texture Atlas

- Integrated directly into Xcode
- Just put your files in a “.atlas” directory
- Drag the directory into your project
- That's it

Creating a Texture Atlas

- Integrated directly into Xcode
- Just put your files in a ".atlas" directory
- Drag the directory into your project
- That's it



```
big_tree_base.png
big_tree_middle.png
big_tree_top.png
blobShadow.png
cave_base.png
cave_destroyed.png
cave_top.png
minionSplort.png
small_tree_base.png
small_tree_middle.png
small_tree_top.png
warrior_throw_hammer.png
```

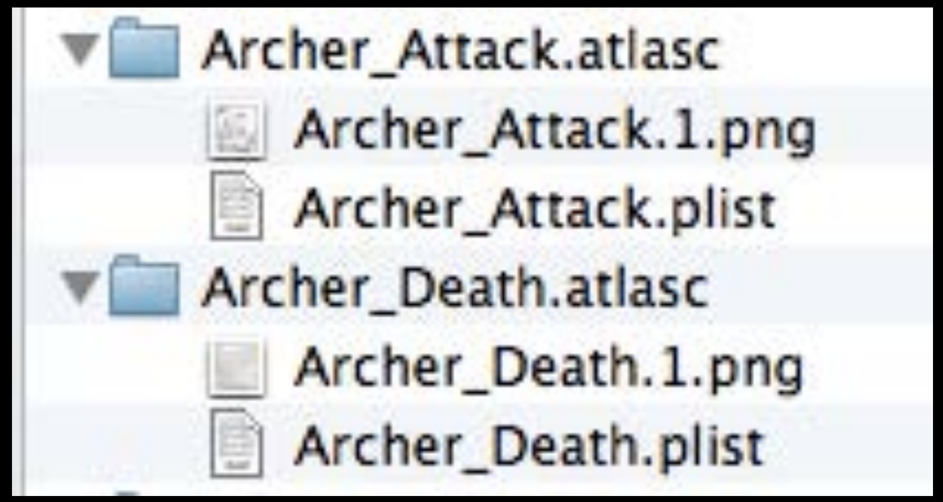
```
▼ Texture Atlases
  ▶ Archer
  ▶ Boss
  ▶ Goblin
  ▶ Warrior
  ▼ Environment.atlas
      big_tree_base.png
      big_tree_middle.png
      big_tree_top.png
      blobShadow.png
      cave_base.png
      cave_destroyed.png
      cave_top.png
      minionSplort.png
      small_tree_base.png
      small_tree_middle.png
      small_tree_top.png
      warrior_throw_hammer.png
  ▶ Tiles.atlas
```

Texture Atlas Generator

- Can be used in any iOS and OS X projects
  - Output format is OpenGL compatible, can be used in 3D games too
  - .atlasc output
- Remember to turn on texture atlas build setting

Texture Atlas Generator

- Can be used in any iOS and OS X projects
  - Output format is OpenGL compatible, can be used in 3D games too
  - .atlasc output

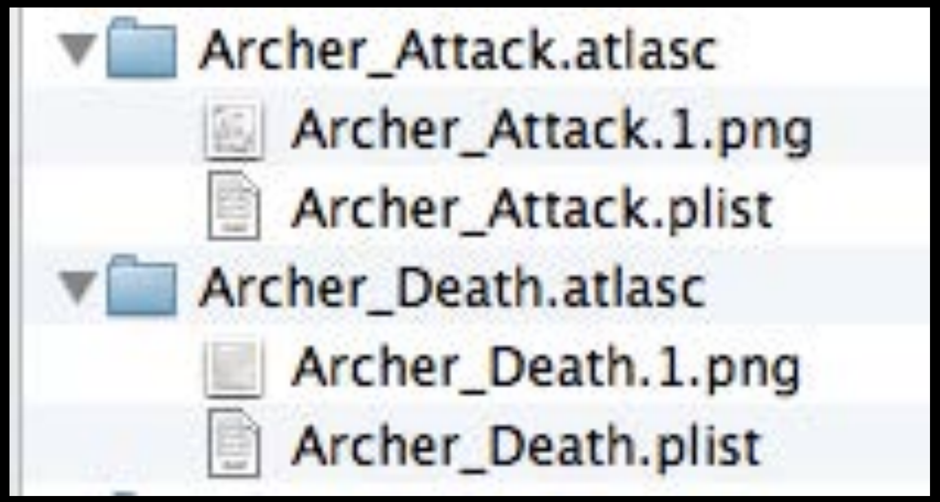


```
Archer_Attack.atlasc
    Archer_Attack.1.png
    Archer_Attack.plist
Archer_Death.atlasc
    Archer_Death.1.png
    Archer_Death.plist
```

- Remember to turn on texture atlas build setting

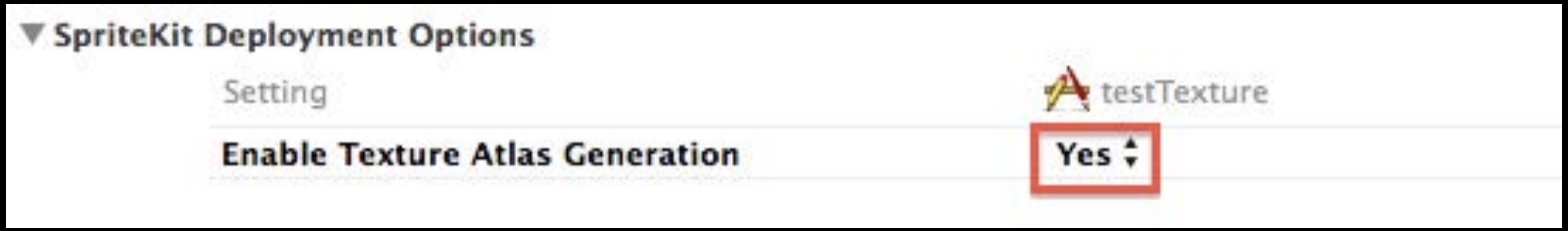
Texture Atlas Generator

- Can be used in any iOS and OS X projects
    - Output format is OpenGL compatible, can be used in 3D games too
    - .atlasc output



```
▼ Archer_Attack.atlasc
      Archer_Attack.1.png
      Archer_Attack.plist
▼ Archer_Death.atlasc
      Archer_Death.1.png
      Archer_Death.plist
```

- Remember to turn on texture atlas build setting



▼ SpriteKit Deployment Options

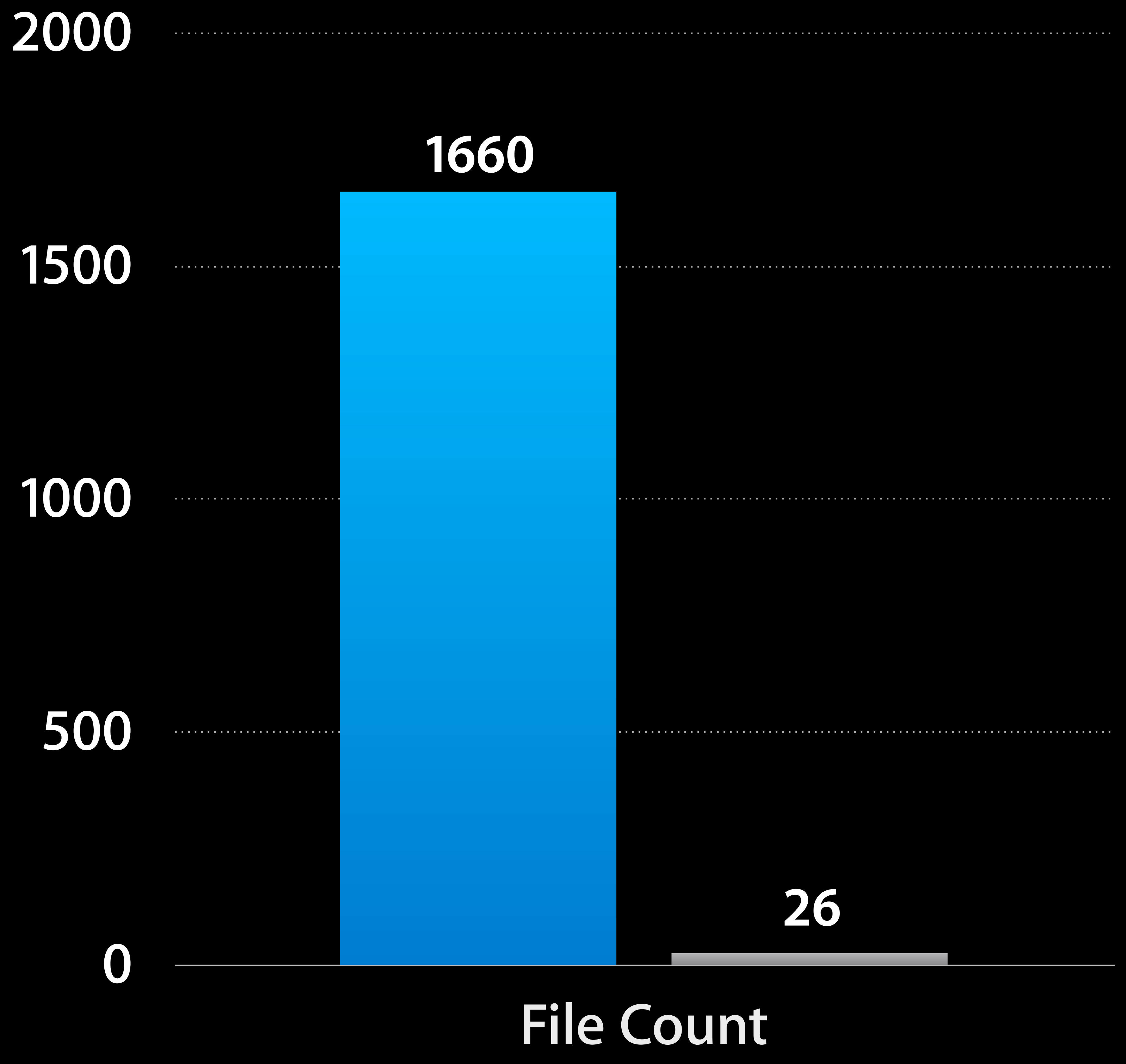
Setting	testTexture
Enable Texture Atlas Generation	Yes

Texture Atlases

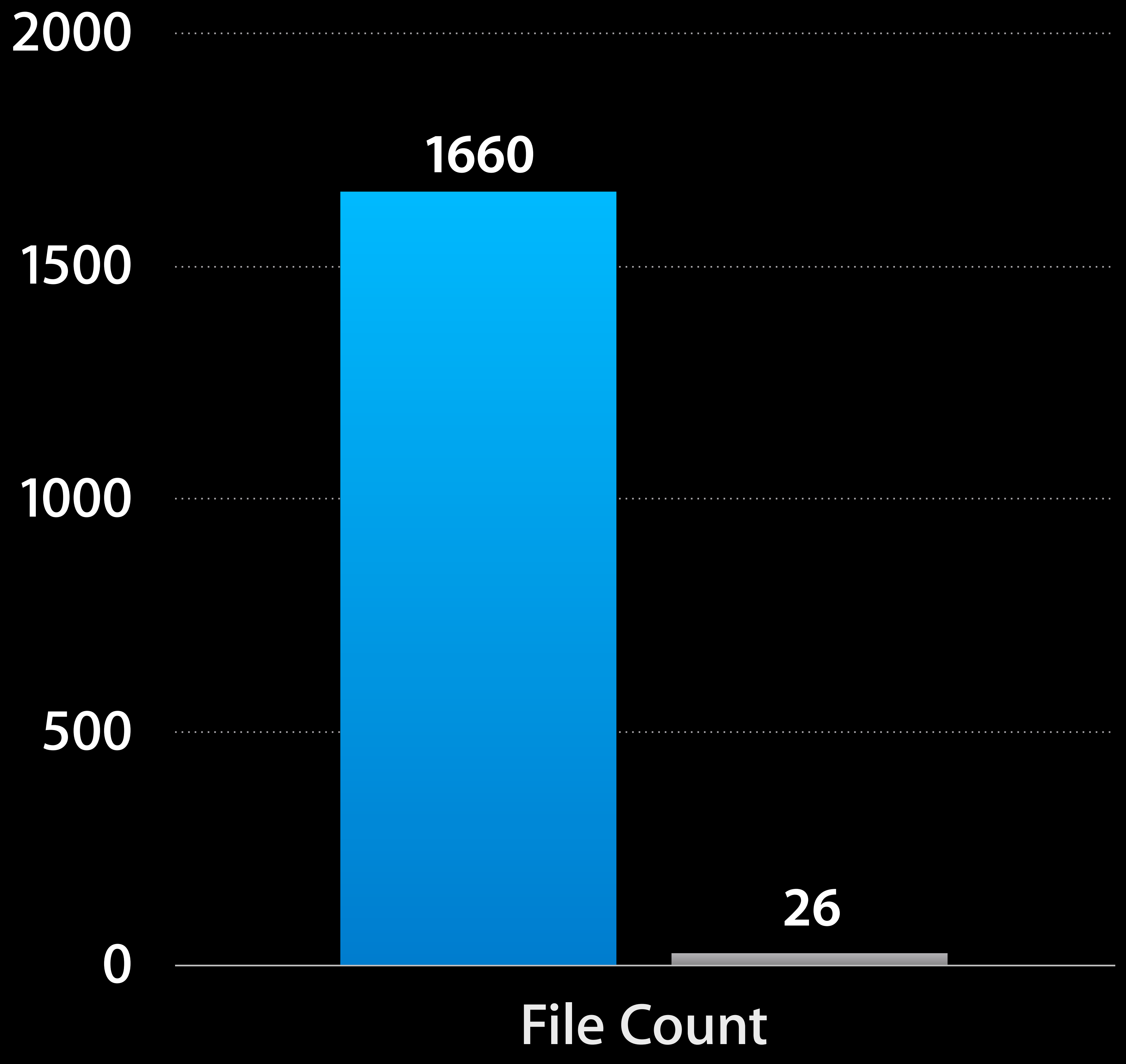
- Automatically combine textures
- Generate hardware specific atlases
- Max of 2048 x 2048 atlas size
- Source images will be processed and packed for maximum occupancy
  - Automatically rotation
  - Transparent edges trimming
  - Extrude opaque images
- Help improving your iteration time

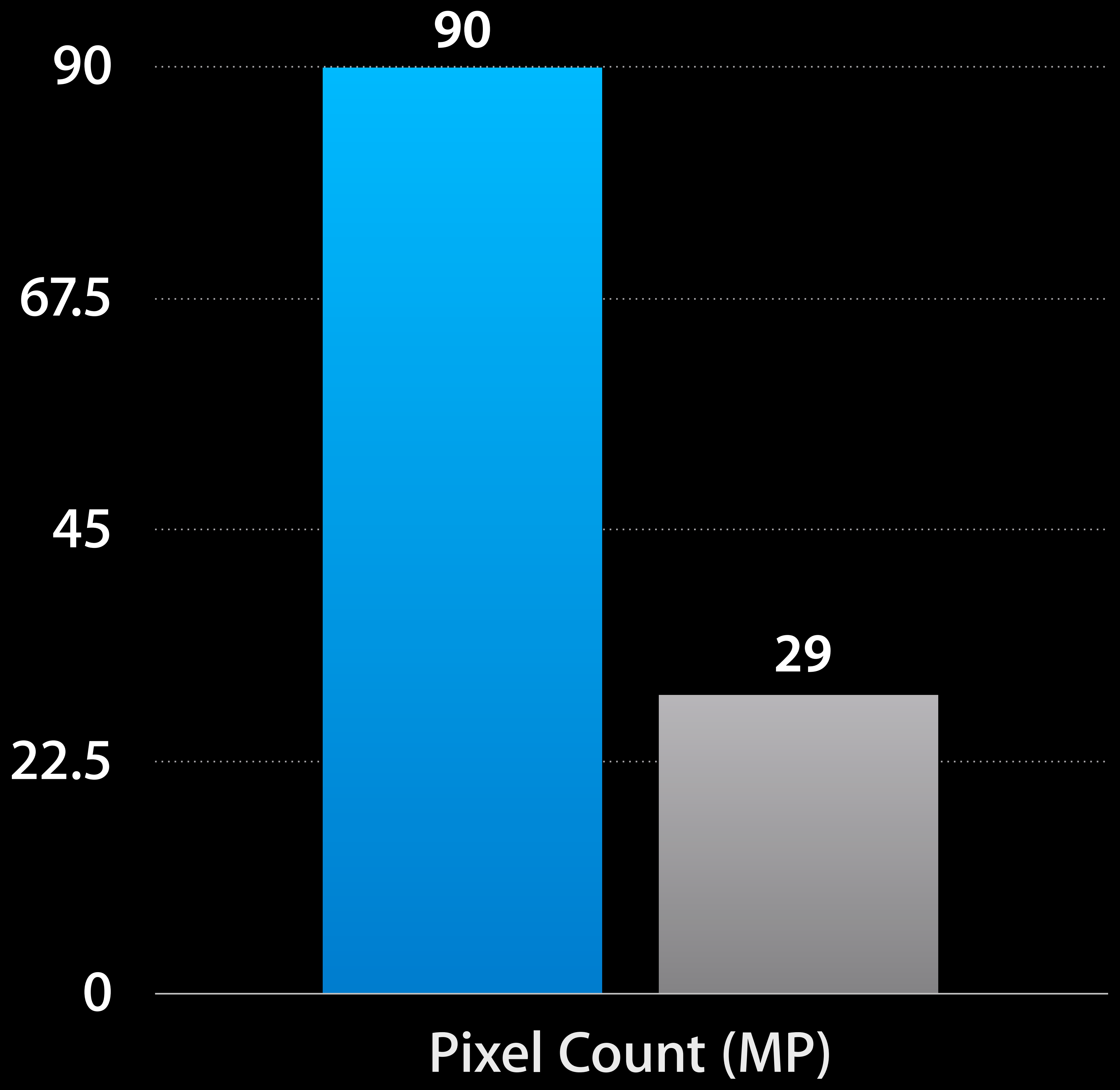
Adventure Textures Summary

Adventure Textures Summary

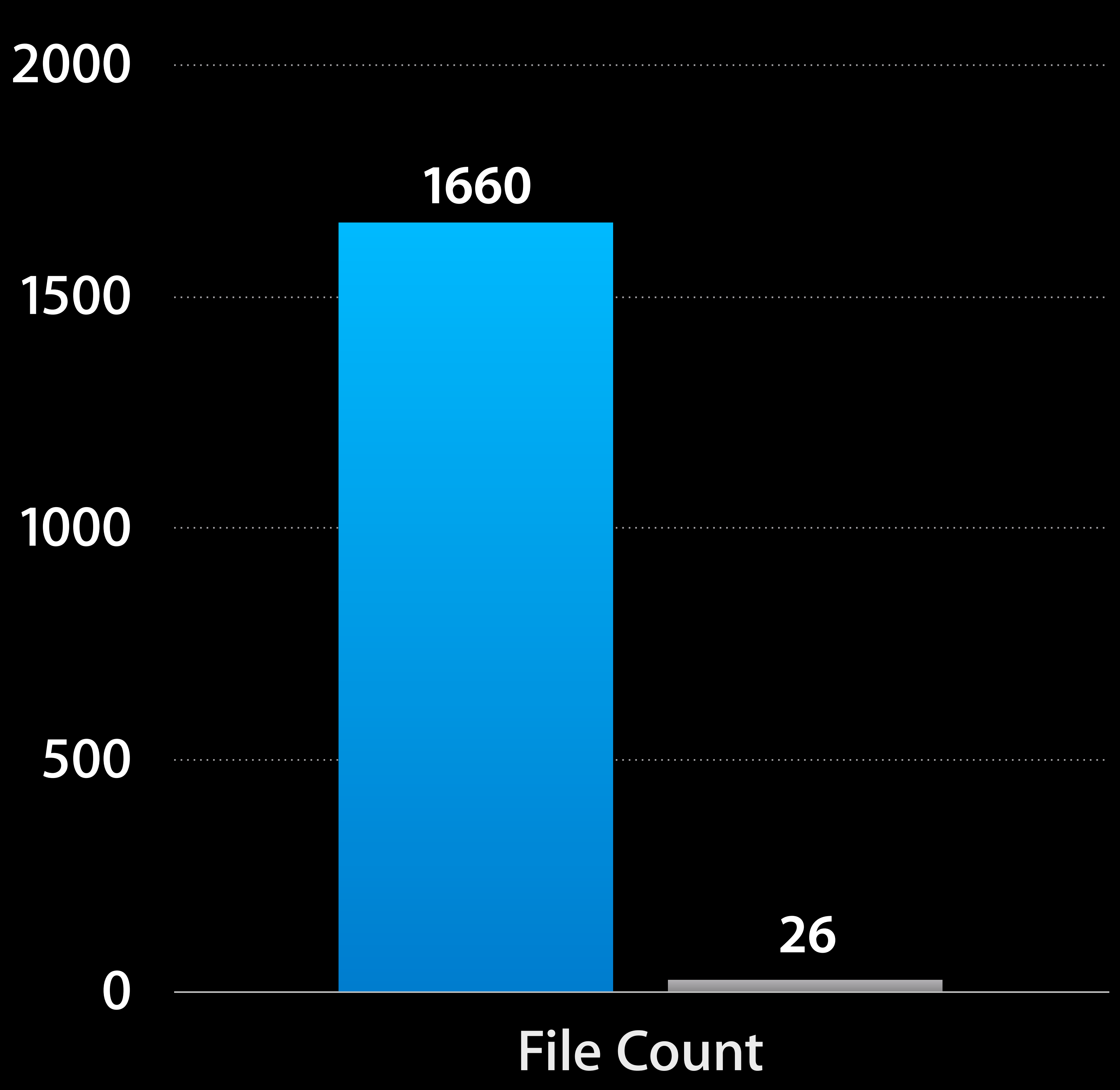


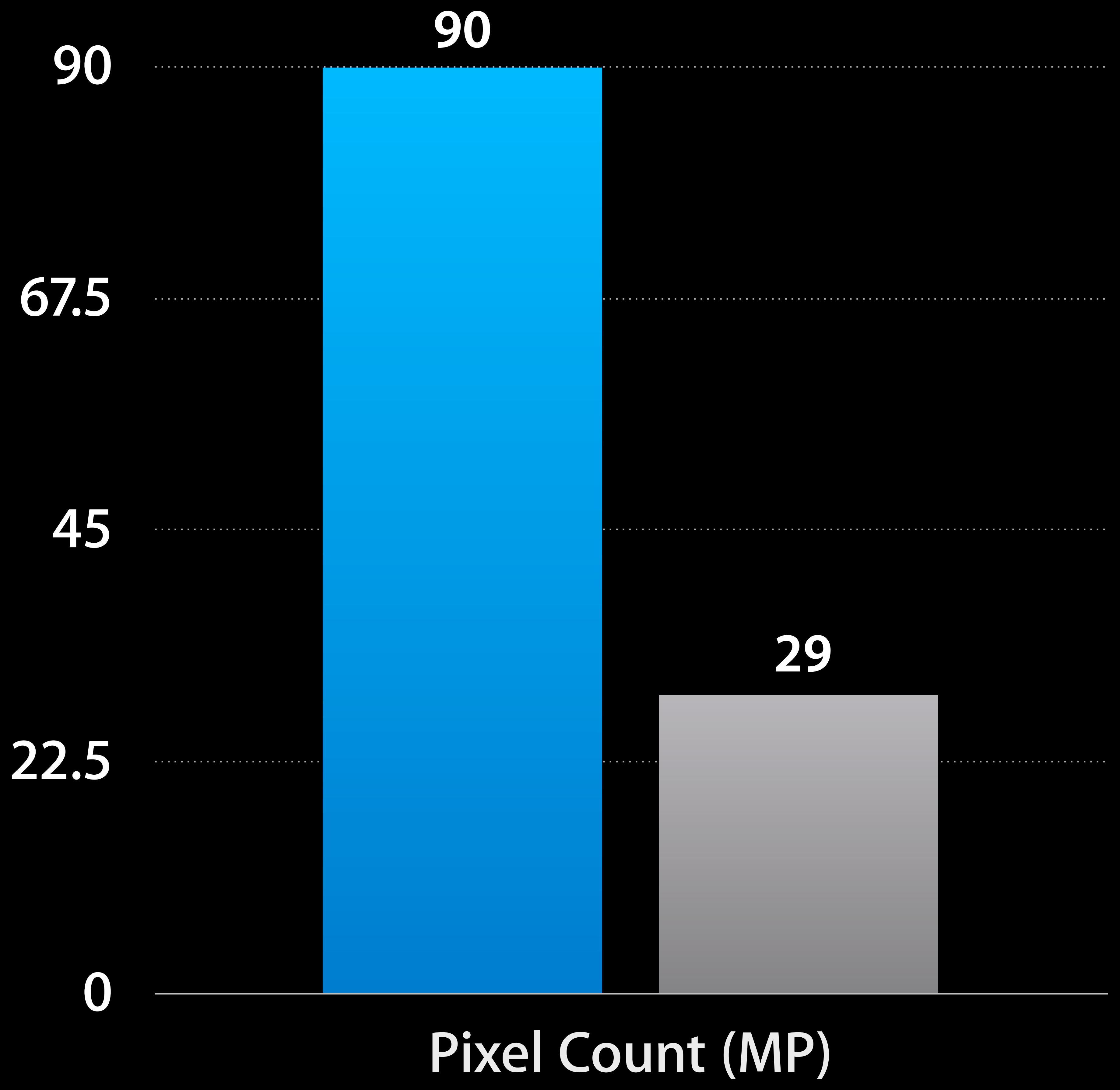
Adventure Textures Summary





Adventure Textures Summary





244MB memory savings!

Loading Textures

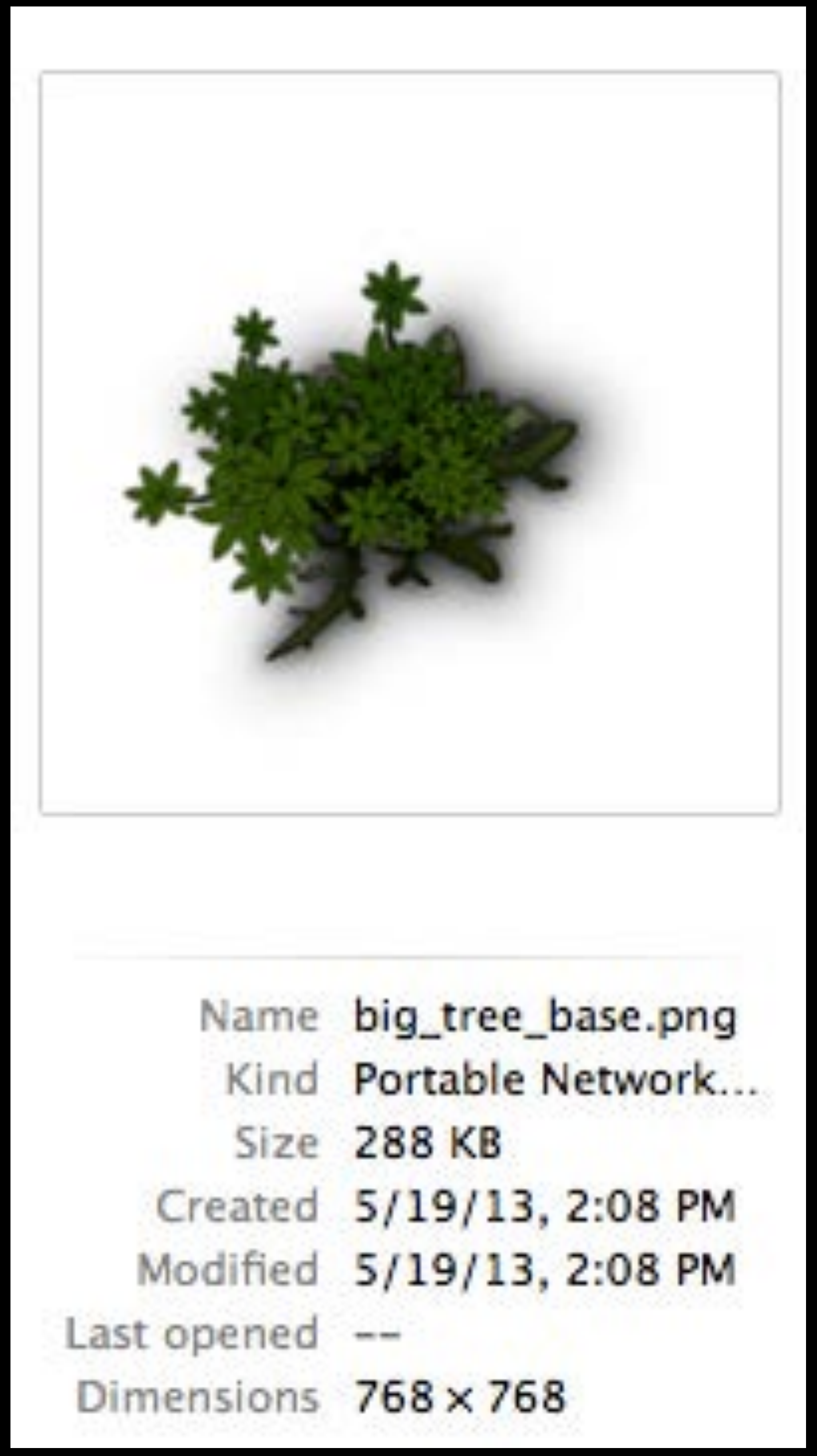
- Load a standalone texture

```
SKTexture *texture = [SKTexture textureWithImageNamed:imageName];
```

Loading Textures

- Load a standalone texture

```
SKTexture *texture = [SKTexture textureWithImageNamed:imageName];
```

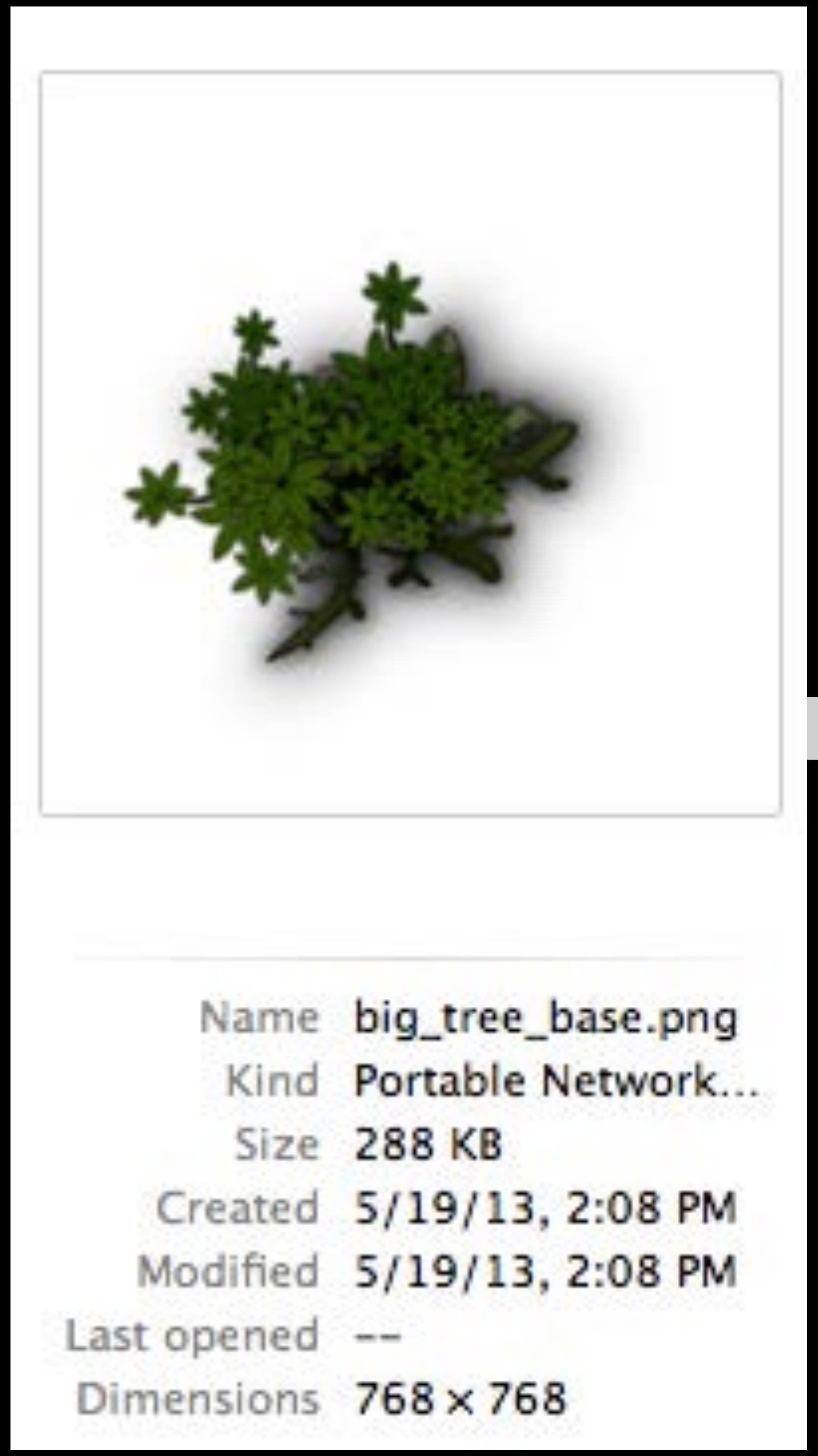


Name	big_tree_base.png
Kind	Portable Network...
Size	288 KB
Created	5/19/13, 2:08 PM
Modified	5/19/13, 2:08 PM
Last opened	--
Dimensions	768 × 768

Loading Textures

- Load a standalone texture

```
SKTexture *texture = [SKTexture textureWithImageNamed:imageName];
```



Name	big_tree_base.png
Kind	Portable Network...
Size	288 KB
Created	5/19/13, 2:08 PM
Modified	5/19/13, 2:08 PM
Last opened	--
Dimensions	768 × 768

Loading from a Texture Atlas

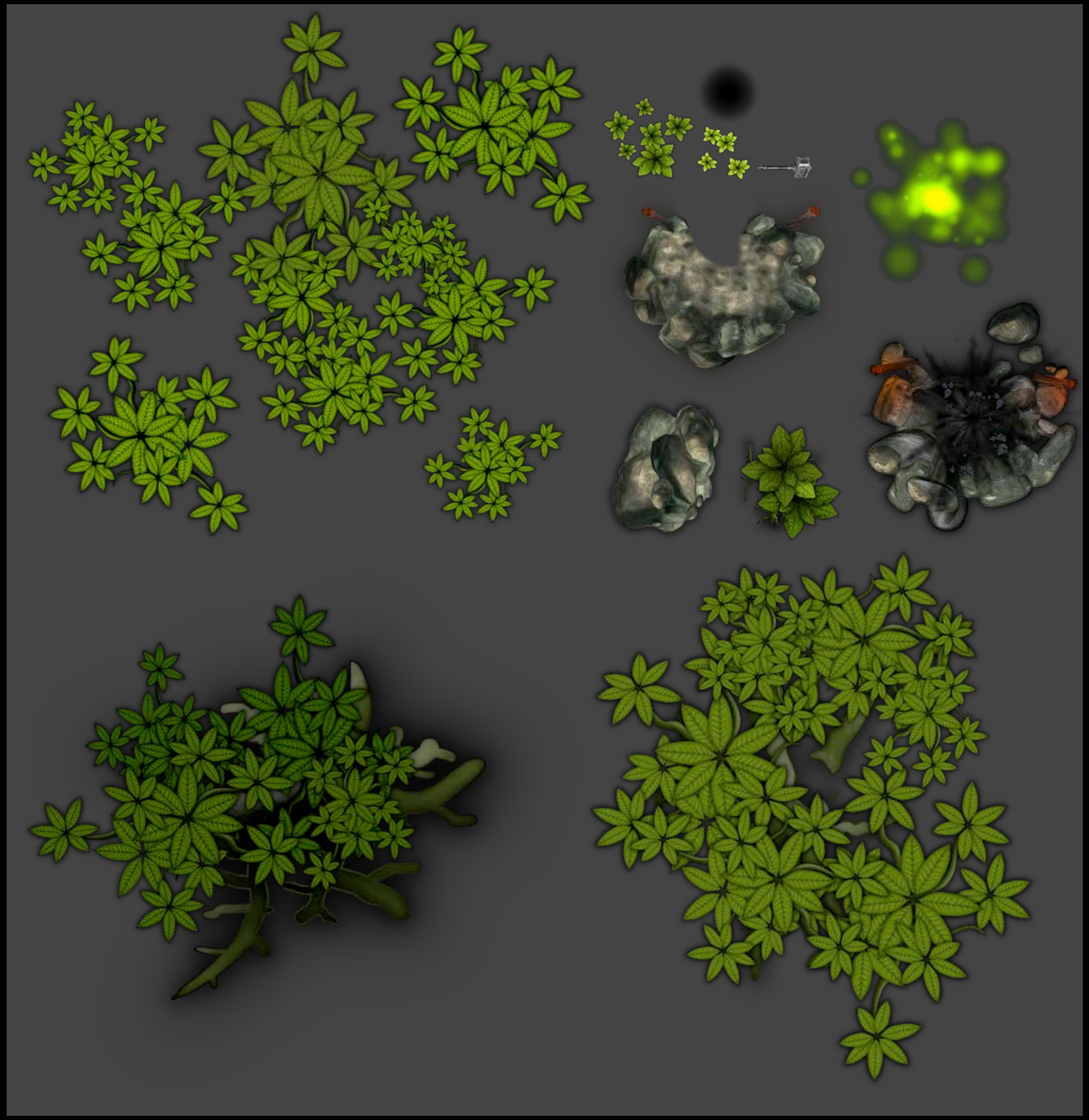
- Load a texture from a texture atlas

```
SKTexture *texture = [SKTexture textureWithImageNamed:imageName];
```

Loading from a Texture Atlas

- Load a texture from a texture atlas

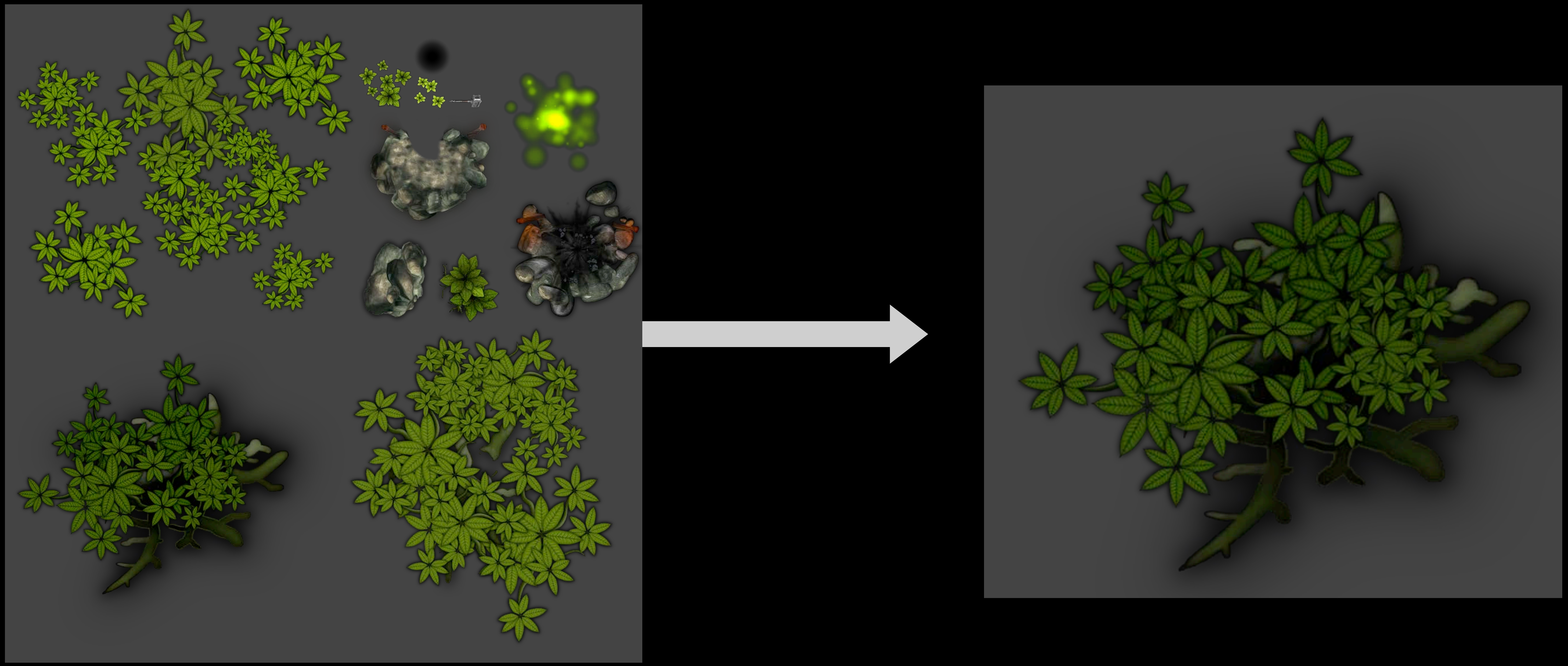
```
SKTexture *texture = [SKTexture textureWithImageNamed:imageName];
```



Loading from a Texture Atlas

- Load a texture from a texture atlas

```
SKTexture *texture = [SKTexture textureWithImageNamed:imageName];
```



Loose File vs. Texture Atlas

- Options for `SKTexture` creation
    - Standalone texture file
    - Sub-texture within an atlas
- Standalone file takes precedence over atlas
    - Easy to switch between both
    - Easy to iterate texture in atlas

Loading from a Texture Atlas

- Following UIKit/Appkit conventions for naming
- Load a texture from a texture atlas

```
SKTextureAtlas *atlas = [SKTextureAtlas atlasNamed:@"Environment"];
NSArray *textureNames = [atlas textureNames];

for (NSString *name in textureNames) {
    SKTexture *texture = [atlas textureNamed:name];
    ...
}
```

Demo

Creating and using a texture atlas

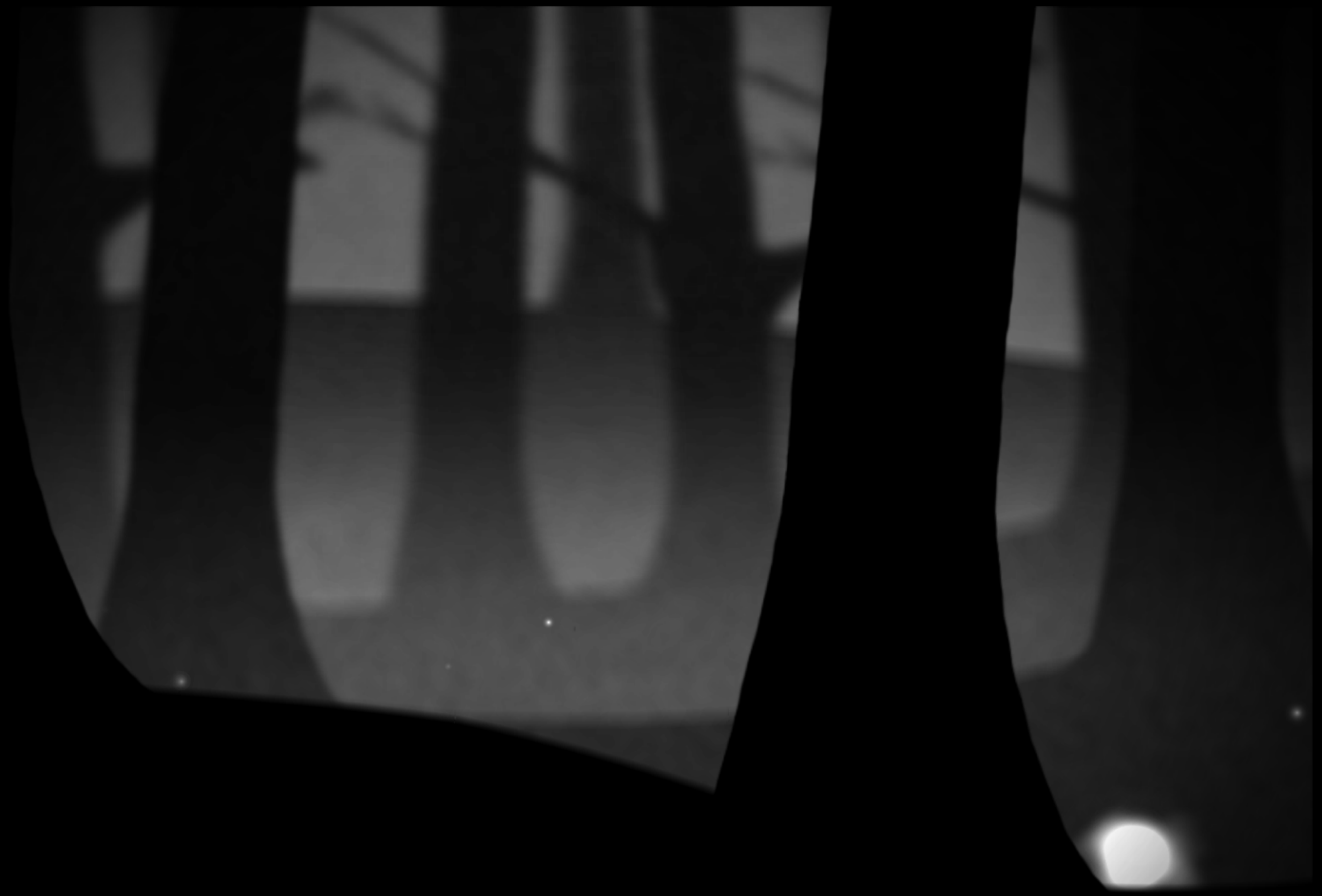
Adventure
Texture Atlases
FAVORITES
All My Files
AirDrop
Applicati...
Desktop
Documents
Downloads
DEVICES
Remote...
TAGS
Red
Orange
Yellow
Green
Blue
Purple
Gray

Name	Date Modified	Size	Kind
Archer	Today, 9:34 AM	--	Folder
Boss	Today, 9:34 AM	--	Folder
Environment.atlas	Today, 9:34 AM	--	Folder
Goblin	Today, 9:34 AM	--	Folder
Tiles.atlas	Today, 9:34 AM	--	Folder
Warrior	Today, 9:34 AM	--	Folder

Visual Effects

Visual Effects

- Post processing
  - Image processing on a given render target
  - Sprite Kit provides image processing effects via `CIFilters`
- Particle systems
  - Spawn a large number of very small sprites
  - `SKEmitterNode` can be used to generate particles



Post Processing with CIFilters

- `CIFilters` can be applied to any `SKEffectNode`
  - Effect will be applied to all children
  - Can cache via `shouldRasterize`

```
self.filter = [CIFilter filterWithName:@”CIGaussianBlur”];
```

- `CIFilters` can be applied to any `SKTextures`

```
SKTexture* texWithFilter = [texture textureByApplyingCIFilter:
                   [CIFilter filterWithName:@”CIGaussianBlur”]]
```

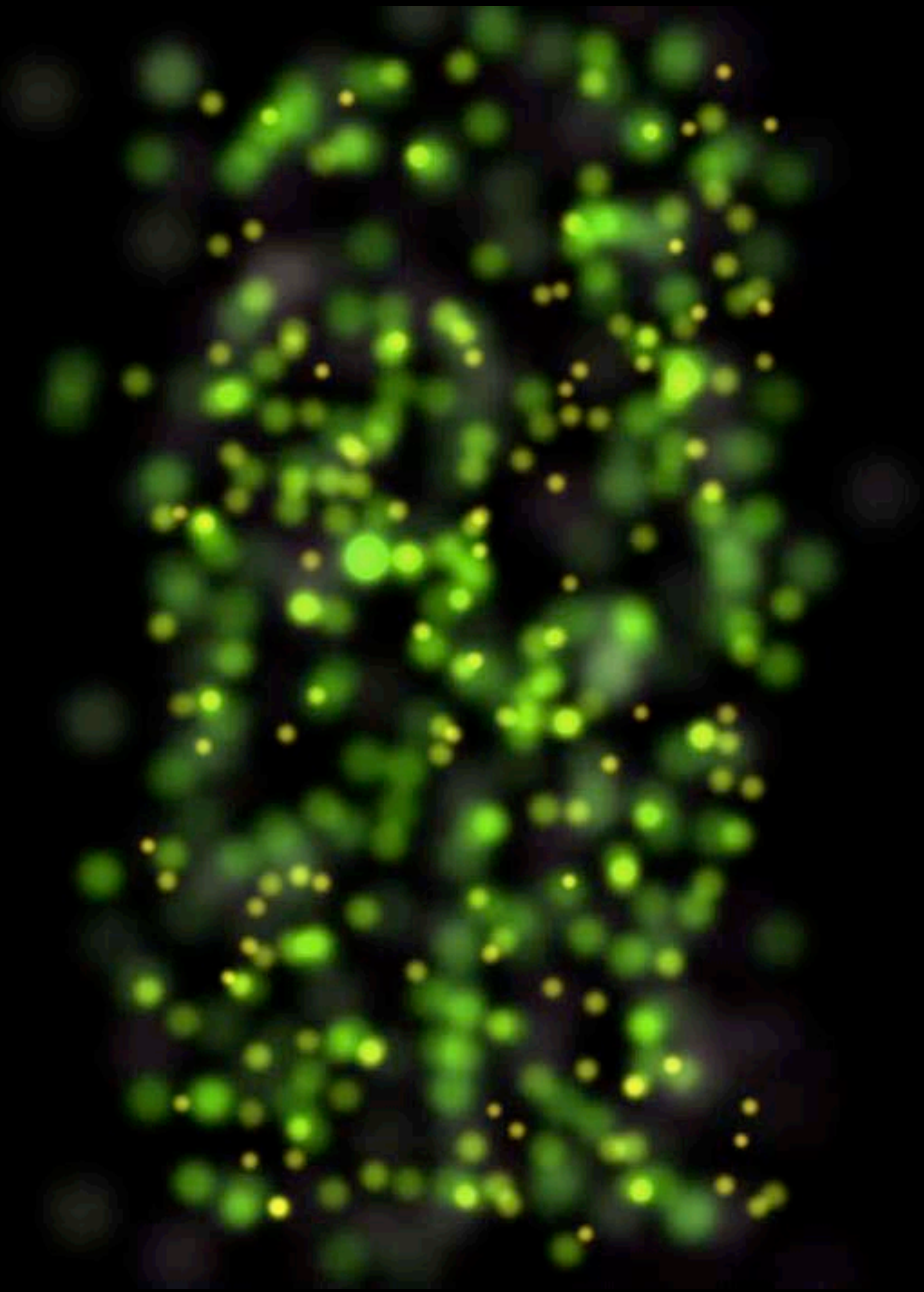
Demo

`CIFilters`

Trees

Particles

- Particle systems often used to generate special effects
- Extensive use in Adventure
    - Leaves
    - Damage
    - Flashes
    - Spawning
- Many of Adventure's particle emitters creating using a tool
    - Tool creates .sks files
    - Code unarchives into an `SKEmitterNode`



SKEmitterNode

Texture

TargetNode

ScaleSpeed

ZPositionRange

ZPosition

Scale

ScaleSequence

BlendMode

xAcceleration

Position

SKEmitterNode

yAcceleration

Alpha

EmissionAngle

Size

Color

Rotation

Color Sequence

Speed

SpeedRange

PositionRange

Action

Lifetime

Particle Editor

- Easy-to-use environment to edit particles
- Integrated directly into Xcode
- Editing all `SKEmitterNode` attributes visually
- Separates particle effect design from programming
- Tip—Best way to learn `SKEmitterNode` capabilities

Adventure.xcodeproj — Leaves_02.sks

Build Adventure: **Succeeded** | Today at 3:19 PM ⚠2

Adventure › Assets › Particles › Leaves_02.sks › SKEmitterNode

- Adventure — 2 targets, multiple platforms
  - Adventure – Shared
  - Adventure – OS X
  - Adventure – iOS
  - Assets
    - Sounds
    - UI
    - Texture Atlases
    - Environment
    - Particles
      - ArcherProjectile.sks
      - BossDamage.sks
      - CaveDamage.sks
      - CaveDeathSmoke.sks
      - CaveFireSmoke.sks
      - Damage.sks
      - Death.sks
      - CaveFire.sks
      - Leaves_01.sks
      - Leaves_02.sks
      - ProjectileSplat.sks
      - Spawn.sks
      - WarriorProjectile.sks
      - hardCircle.png
      - leaf_00.png
      - leaf_02.png
      - spark.png
      - star.png
  - Frameworks
  - Products
    - Adventure.app
    - Adventure.app

Emitter Node

Background			
Particle Texture	leaf_02.png		
Particles	5 (Birthrate)	0 (Maximum)	
Lifetime	4 (Start)	0 (Range)	
Position Range	500 (X)	500 (Y)	
Angle	191.941° (Start)	360.39° (Range)	
Speed	20 (Start)	0 (Range)	
Acceleration	0 (X)	-0 (Y)	
Scale	1 (Start)	0.25 (Range)	-0.2 (Speed)
Rotation	91.1° (Start)	316.2 (Range)	57.29 (Speed)
Color Blend	1 (Factor)	0 (Range)	0 (Speed)
Color Ramp			
Blend Mode	Alpha		

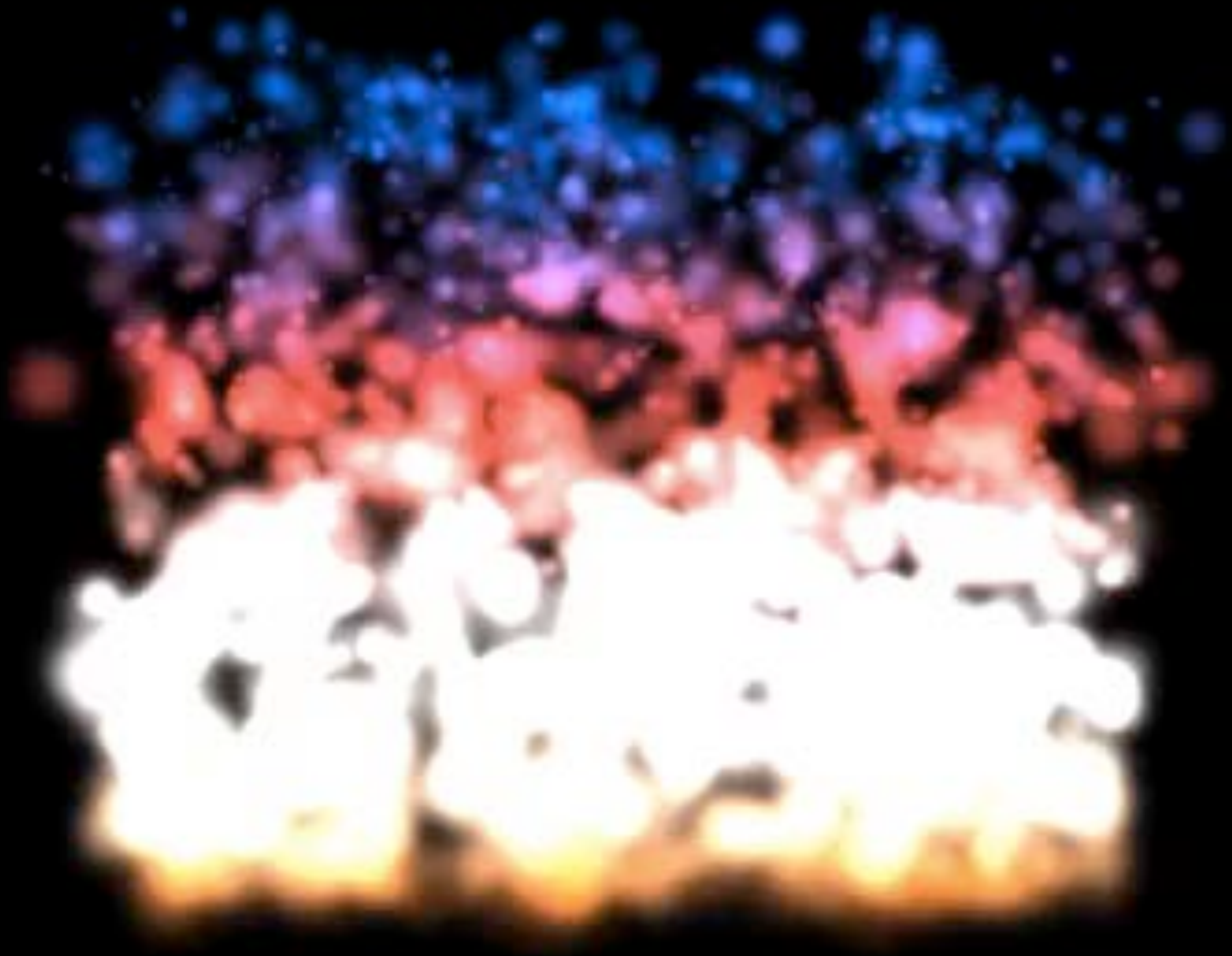
No Selection

Auto

All Output

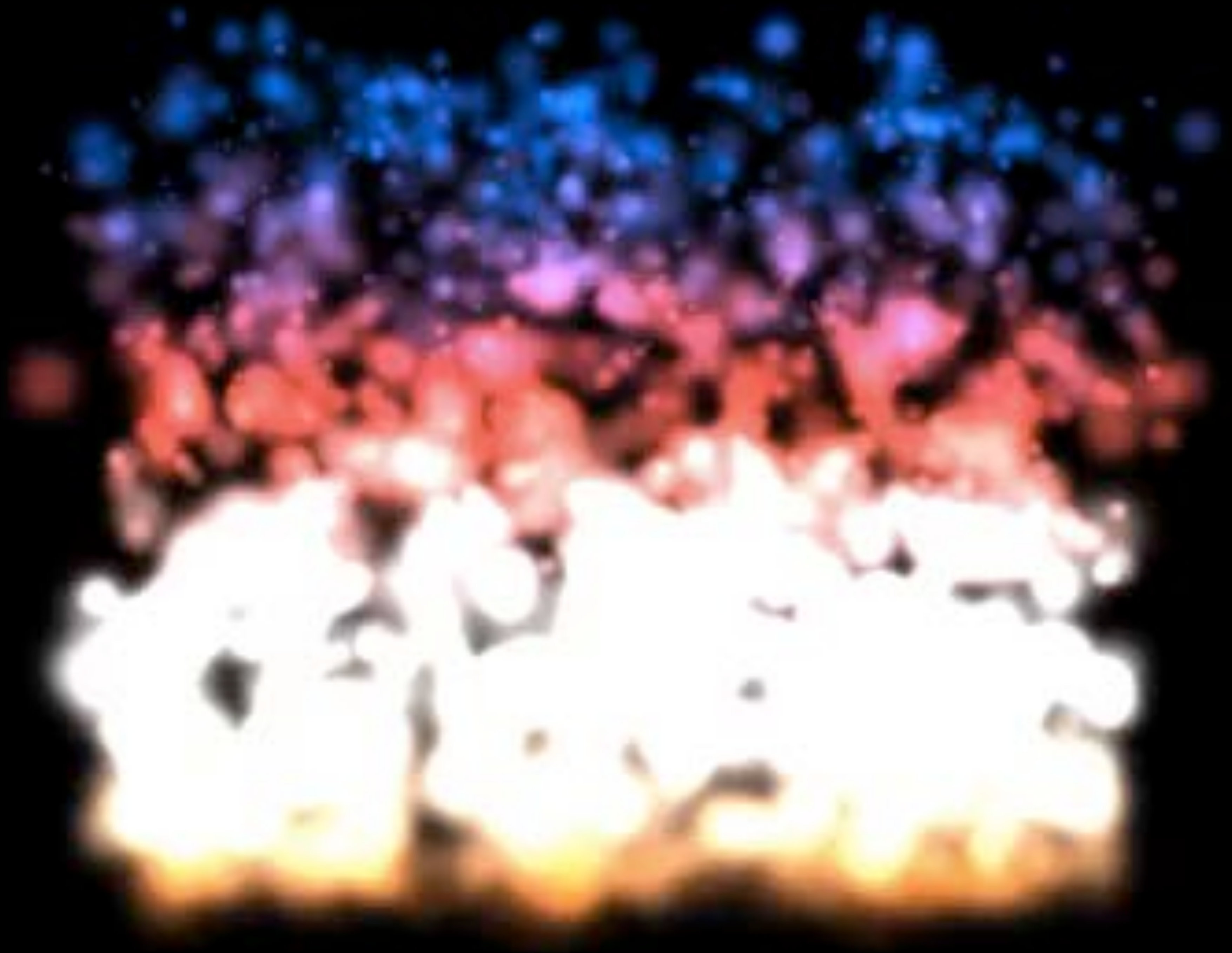
Using Keyframe Sequences

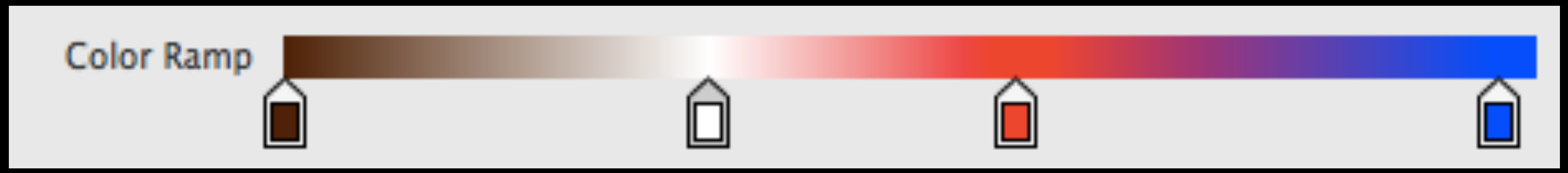
- Keyframe sequences provide more sophisticated behaviors
- Controls the lifetime color transition for each particle



Using Keyframe Sequences

- Keyframe sequences provide more sophisticated behaviors
- Controls the lifetime color transition for each particle





Using Keyframe Sequences

- Keyframe sequences can also be constructed in code
- Using a sequence to change a particle's scale property

```
SKKeyframeSequence *scaleSequence = [[SKKeyframeSequence alloc]
  initWithKeyframeValues:@[ @0.2, @0.7,   @0.1  ]
                   times:@[ @0.0, @0.250, @0.75 ]];
myEmitter.particleScaleSequence = scaleSequence;
```

Adding Actions to Particles

- Particles can execute actions
  - Enables more sophisticated behaviors
  - e.g. animating particle's textures
- Invoked by emitter
  - At time of particle creation
  - `particleAction` property

particleAction Example

```
emitter.particleAction = [SKAction animateWithTextures:attackFrames
                                          timePerFrame:1/22.0
                                                resize:YES
                                               restore:NO];
```

Loading an Emitter

- An emitter file is an archived `SKEmitterNode`
- Use `NSKeyedUnarchiver` to unarchive it at runtime

```
SKEmitterNode *emitter = [NSKeyedUnarchiver unarchiveObjectWithFile:
    [[NSBundle mainBundle] pathForResource:@"BossDamage" ofType:@"sks"]];
```

Particle Recommendations

Maximizing performance, minimizing iterations

- Keep birth rate down
- Iterate in Xcode particle editor
  - Then load archive into game
- Remove particle emitter if not visible
- Tip—A few particles is often enough to look great

Demo

Creating and loading particles

Building Adventure

Graeme Devine
GRL Games

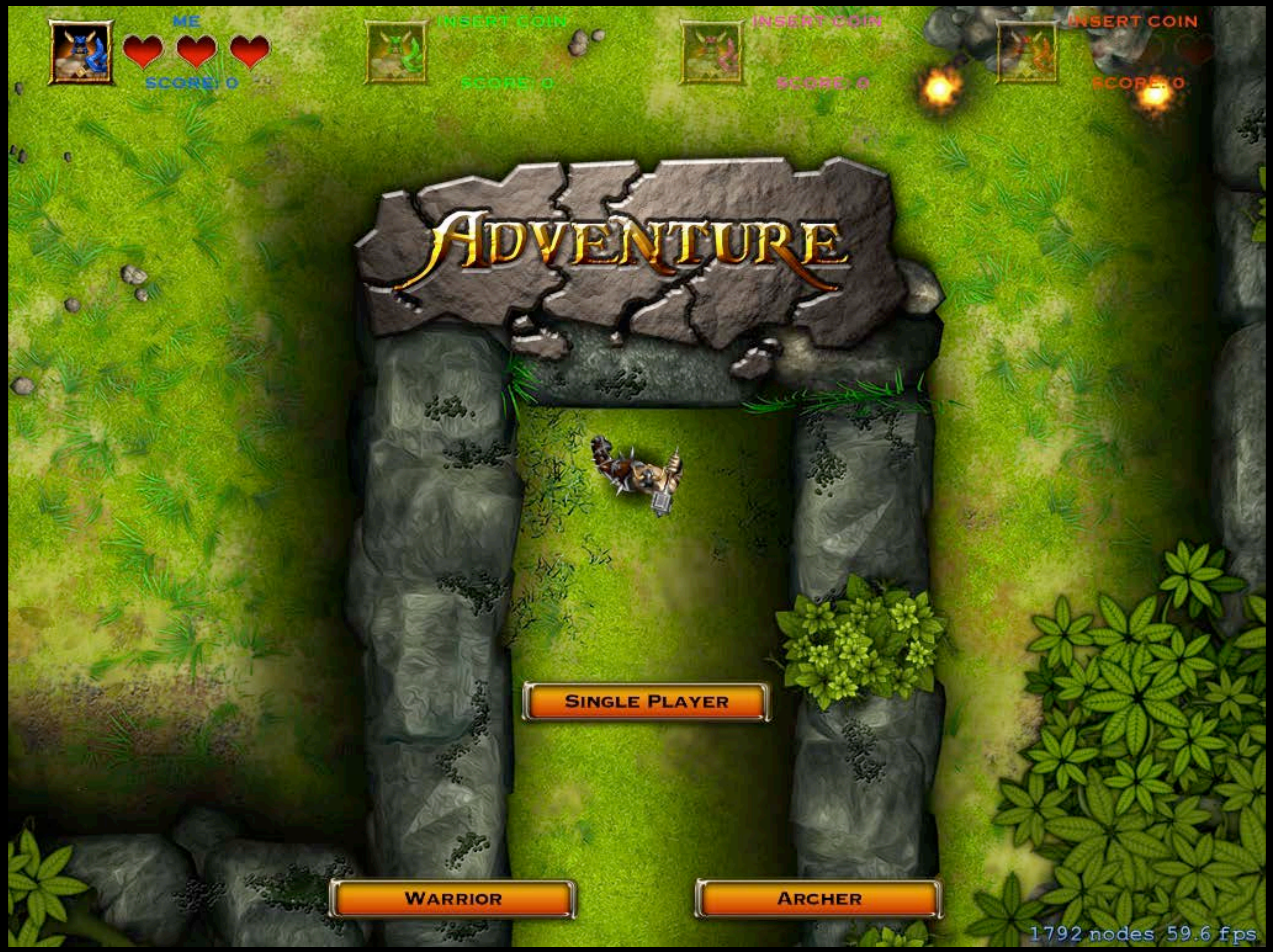
Spencer Lindsay
Lindsay Digital

Today

- Adventure demo production
- How we solved some of the technical challenges
- How we solved some of the art challenges

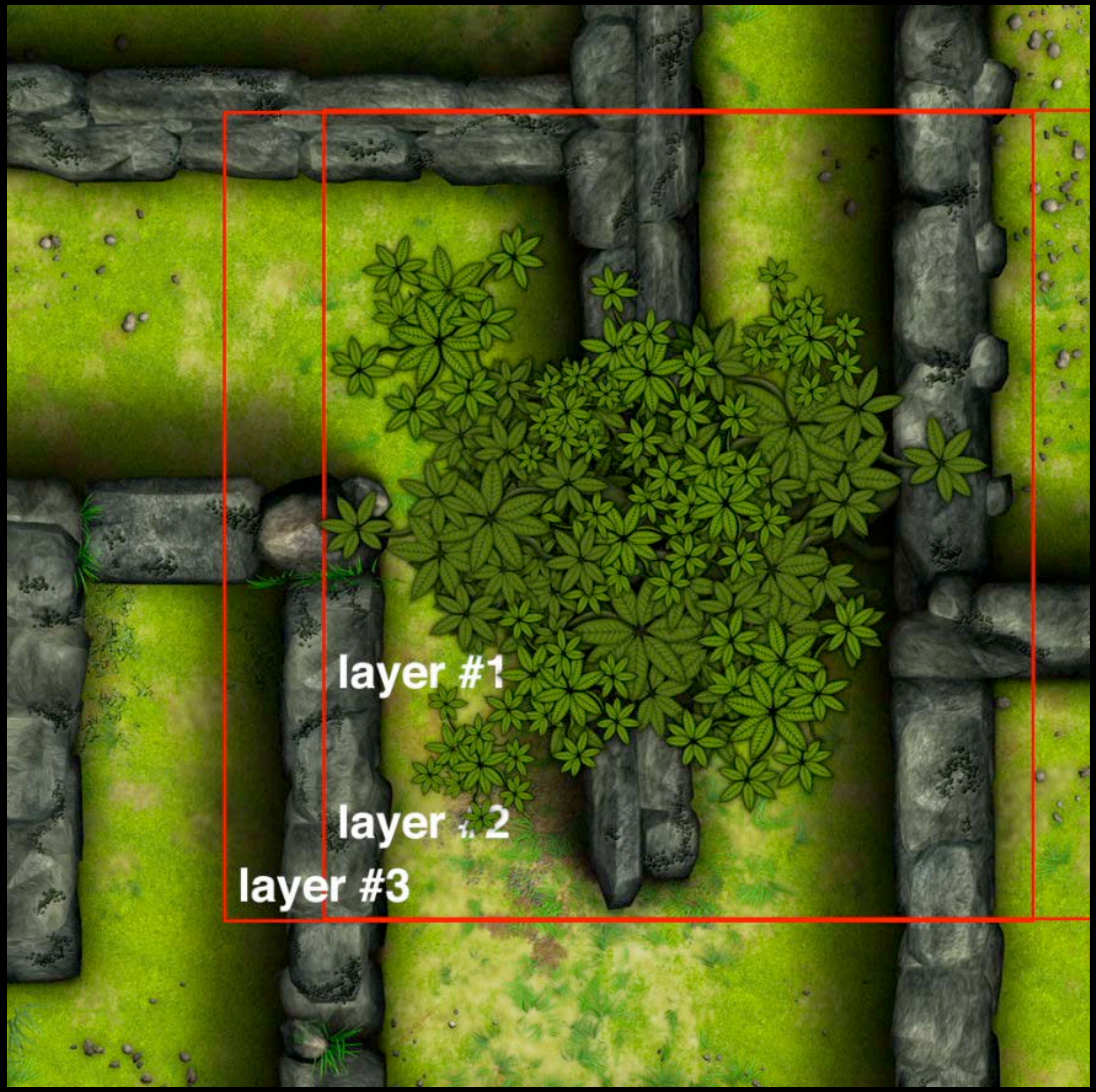
Demo

- Parallax
- Collisions
- Particles
- Sprites



Parallax

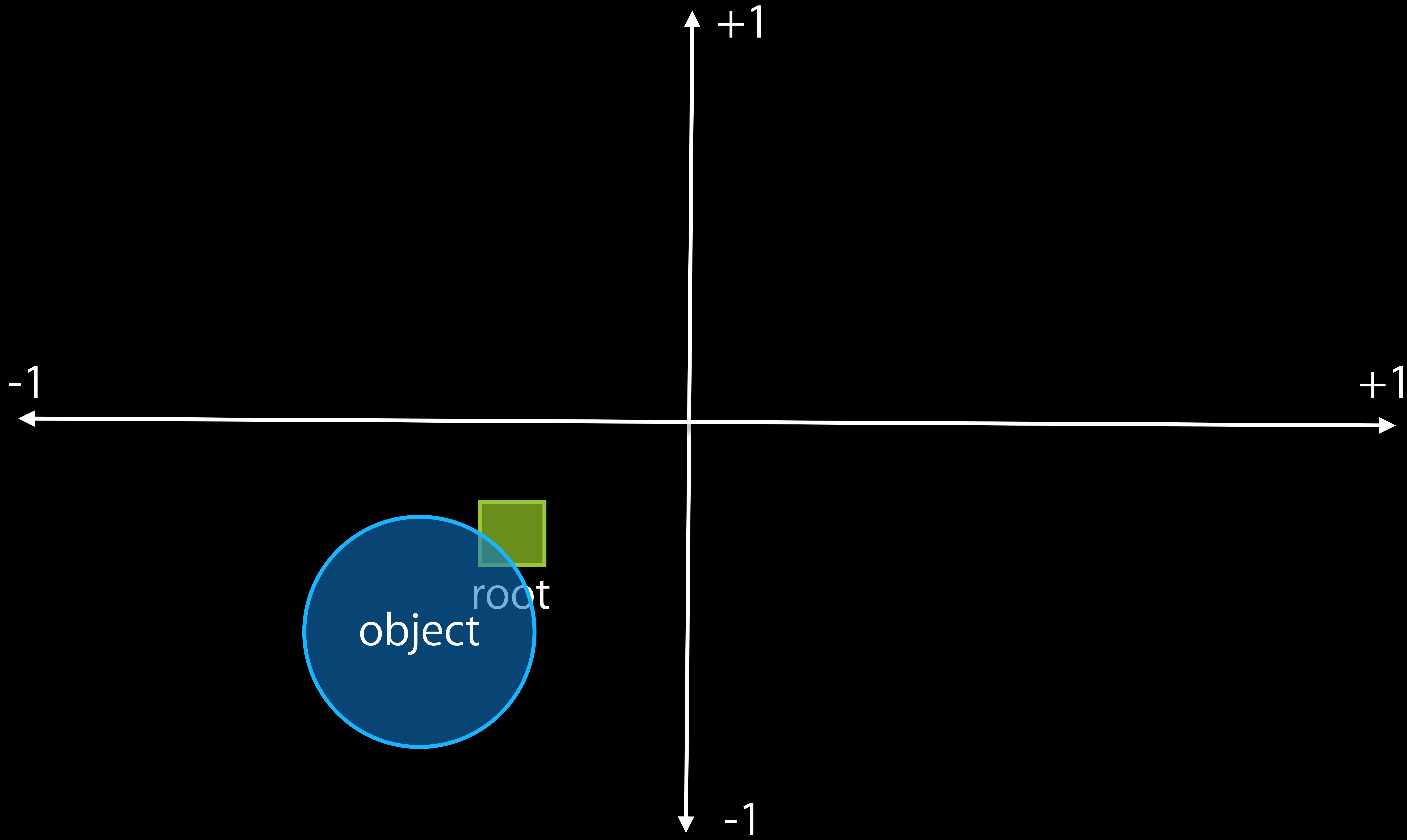
- z = w/(x+y) * 2.0 - 512
- sub classing SKNode
- camera
- movement



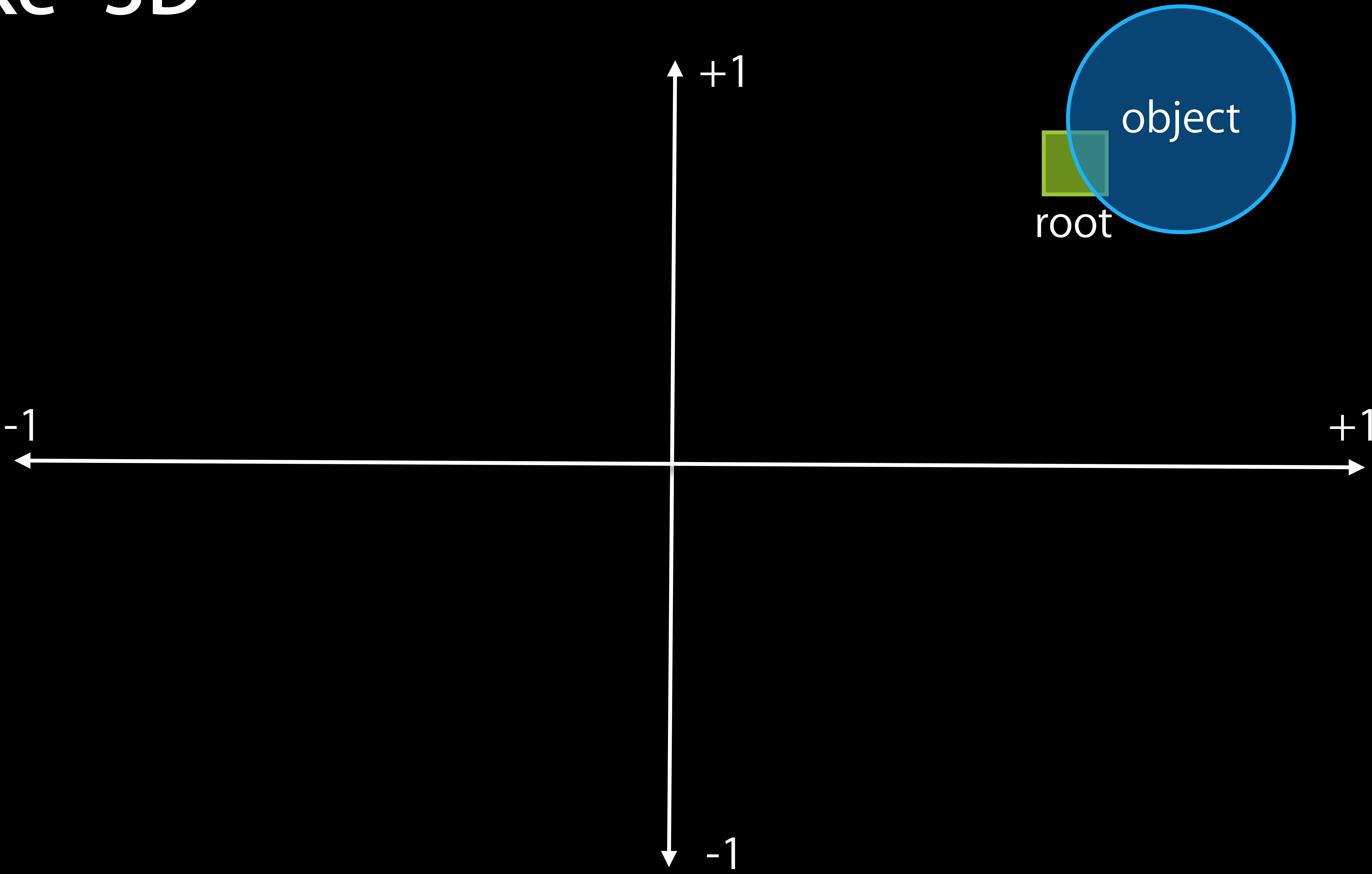
Lesson One

Fake It All

"Fake" 3D



"Fake" 3D



Parallax

```
@interface ParallaxSprite : SKNode
@end


ParallaxSprite  *ps = [[ParallaxSprite alloc] initWithSprites:@[
[SKSpriteNode spriteNodeWithImageNamed:@"tree_05a"],
[SKSpriteNode spriteNodeWithImageNamed:@"tree_05b"],
[SKSpriteNode spriteNodeWithImageNamed:@"tree_05c"] ] usingOffset:150.0f];

ps.fadeAlpha = YES;
```

Parallax

```
@interface ParallaxSprite : SKNode
@end
```

```
ParallaxSprite  *ps = [[ParallaxSprite alloc] initWithSprites:@[
[SKSpriteNode spriteNodeWithImageNamed:@"tree_05a"],
[SKSpriteNode spriteNodeWithImageNamed:@"tree_05b"],
[SKSpriteNode spriteNodeWithImageNamed:@"tree_05c"] ] usingOffset:150.0f];

ps.fadeAlpha = YES;
```

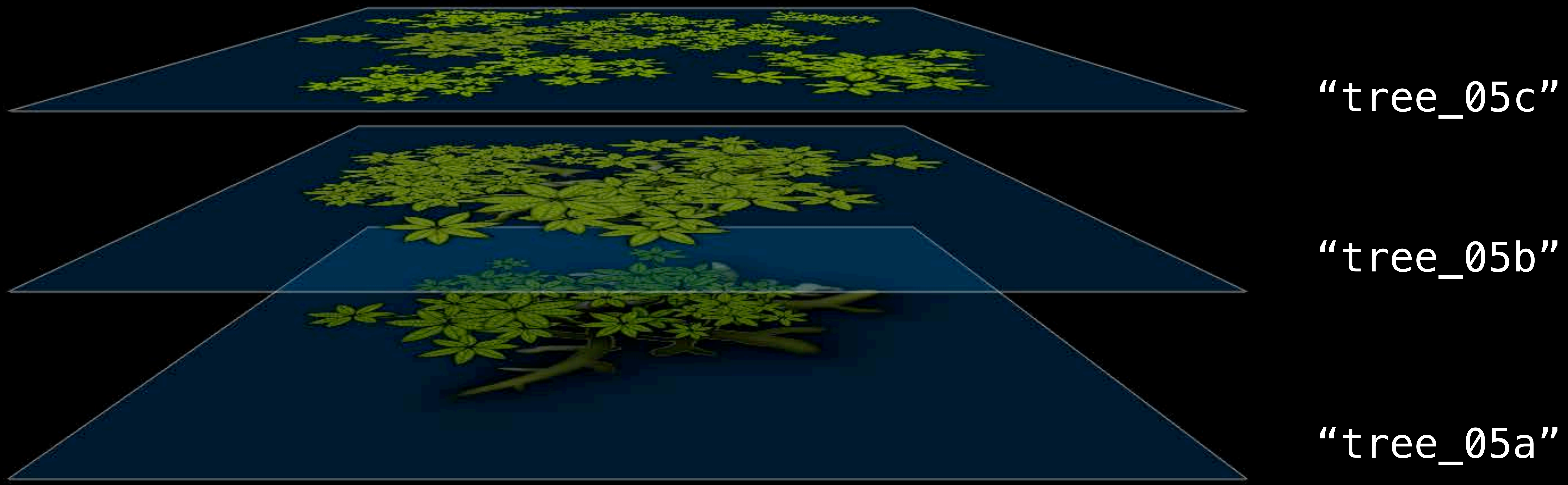
Parallax

```
@interface ParallaxSprite : SKNode
@end
```

```
ParallaxSprite  *ps = [[ParallaxSprite alloc] initWithSprites:@[
[SKSpriteNode spriteNodeWithImageNamed:@"tree_05a"],
[SKSpriteNode spriteNodeWithImageNamed:@"tree_05b"],
[SKSpriteNode spriteNodeWithImageNamed:@"tree_05c"] ] usingOffset:150.0f];

ps.fadeAlpha = YES;
```

Parallax



```
ParallaxSprite  *ps = [[ParallaxSprite alloc] initWithSprites:@[
                [SKSpriteNodeWithImageNamed:@”tree_05a”],
                [SKSpriteNodeWithImageNamed:@”tree_05b”],
                [SKSpriteNodeWithImageNamed:@”tree_05c”]] usingOffset:
150.0f],
```

Parallax

```
 -(void)updateOffset:(SKScene*)scene
{
    // Get the current scene position
    CGPoint scenePos = [scene convertPoint:self.position fromNode:self.parent];

    // Step 1, work out the -1 -> +1 of the X & Y
    CGFloat offsetX =  (-1.0f + (2.0 * (scenePos.x / scene.size.width)));
    CGFloat offsetY =  (-1.0f + (2.0 * (scenePos.y / scene.size.height)));


       // Step 2, apply offset multiplied by level to children
     for (int i = 0; i < self.children.count; i++)
     {
        pos.x = offsetX * (self.parallaxOffset * i);
        pos.y = offsetY * (self.parallaxOffset * i);

        SKNode* node = self.children[i];
        node.position = pos;
     }
     // Step 3, profit
}
```

Parallax

```
 -(void)updateOffset:(SKScene*)scene
{
    // Get the current scene position
    CGPoint scenePos = [scene convertPoint:self.position fromNode:self.parent];

    // Step 1, work out the -1 -> +1 of the X & Y
    CGFloat offsetX =  (-1.0f + (2.0 * (scenePos.x / scene.size.width)));
    CGFloat offsetY =  (-1.0f + (2.0 * (scenePos.y / scene.size.height)));


       // Step 2, apply offset multiplied by level to children
     for (int i = 0; i < self.children.count; i++)
     {
        pos.x = offsetX * (self.parallaxOffset * i);
        pos.y = offsetY * (self.parallaxOffset * i);

        SKNode* node = self.children[i];
        node.position = pos;
     }
     // Step 3, profit
}
```

Parallax

```
 -(void)updateOffset:(SKScene*)scene
{
    // Get the current scene position
    CGPoint scenePos = [scene convertPoint:self.position fromNode:self.parent];

    // Step 1, work out the -1 -> +1 of the X & Y
    CGFloat offsetX =  (-1.0f + (2.0 * (scenePos.x / scene.size.width)));
    CGFloat offsetY =  (-1.0f + (2.0 * (scenePos.y / scene.size.height)));


       // Step 2, apply offset multiplied by level to children
     for (int i = 0; i < self.children.count; i++)
     {
        pos.x = offsetX * (self.parallaxOffset * i);
        pos.y = offsetY * (self.parallaxOffset * i);

        SKNode* node = self.children[i];
        node.position = pos;
     }
     // Step 3, profit
}
```

Parallax

```
CGFloat maxDist = MAXFLOAT;
// Step 1, see if there's any heroes nearby
for (AdventureCharacter* hero in advscene.heroes)
{
    CGPoint theirPos = hero.mainSprite.position;
    CGFloat distance = DistanceBetweenPoints(self.position, theirPos);
    if (distance < maxDist)
    {
        maxDist = distance;
    }
}
// Step 2, if we’re close enough, apply alpha to sprite else
// make the sprite opaque
if (maxDist > kOpaqueDistance)
{
self.alpha = 1.0;
} else {
    CGFloat kalpha = 0.1 + ((maxDist / kOpaqueDistance) *
         (maxDist / kOpaqueDistance) ) * 0.9;
    self.alpha = kalpha;
}
```

Parallax

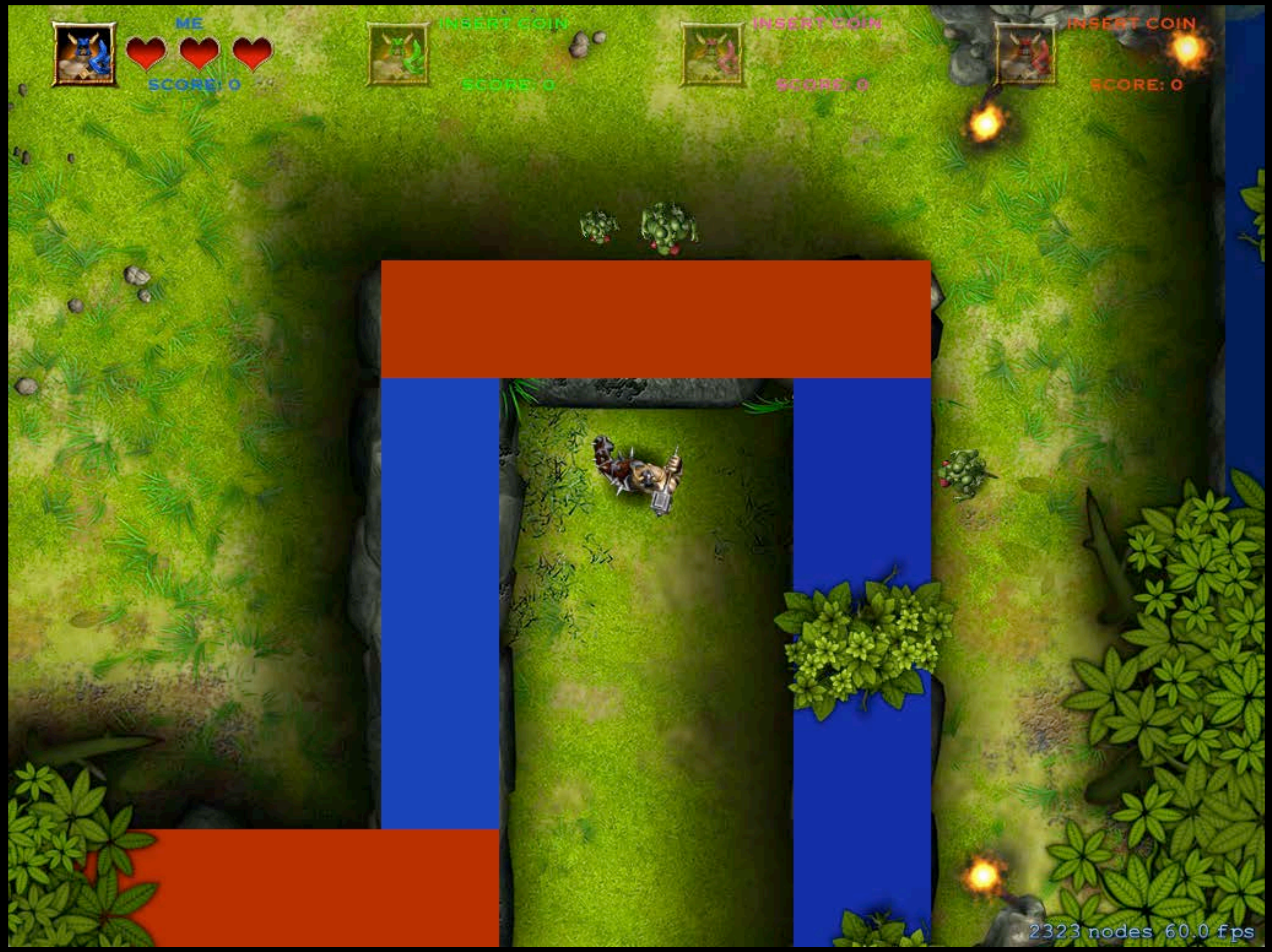
```
CGFloat maxDist = MAXFLOAT;
// Step 1, see if there's any heroes nearby
for (AdventureCharacter* hero in advscene.heroes)
{
    CGPoint theirPos = hero.mainSprite.position;
    CGFloat distance = DistanceBetweenPoints(self.position, theirPos);
    if (distance < maxDist)
    {
        maxDist = distance;
    }
}
// Step 2, if we’re close enough, apply alpha to sprite else
// make the sprite opaque
if (maxDist > kOpaqueDistance)
{
self.alpha = 1.0;
} else {
    CGFloat kalpha = 0.1 + ((maxDist / kOpaqueDistance) *
         (maxDist / kOpaqueDistance) ) * 0.9;
    self.alpha = kalpha;
}
```

Parallax

```
CGFloat maxDist = MAXFLOAT;
// Step 1, see if there's any heroes nearby
for (AdventureCharacter* hero in advscene.heroes)
{
    CGPoint theirPos = hero.mainSprite.position;
    CGFloat distance = DistanceBetweenPoints(self.position, theirPos);
    if (distance < maxDist)
    {
        maxDist = distance;
    }
}
// Step 2, if we’re close enough, apply alpha to sprite else
// make the sprite opaque
if (maxDist > kOpaqueDistance)
{
self.alpha = 1.0;
} else {
    CGFloat kalpha = 0.1 + ((maxDist / kOpaqueDistance) *
         (maxDist / kOpaqueDistance) ) * 0.9;
    self.alpha = kalpha;
}
```

Collision Mapping

- How we made it
- Using physics



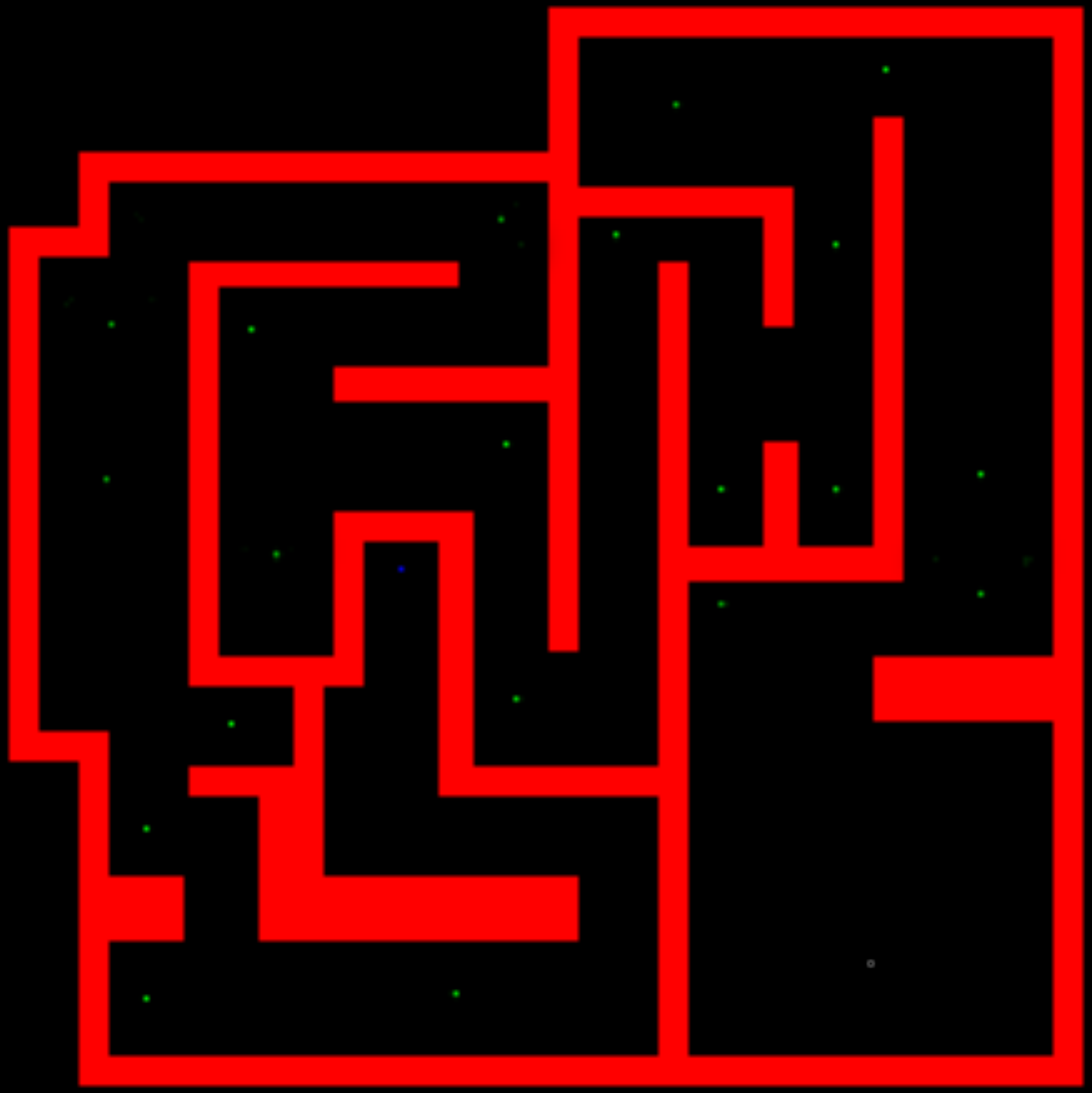
Collision Mapping

Collision Mapping

- The wrong way

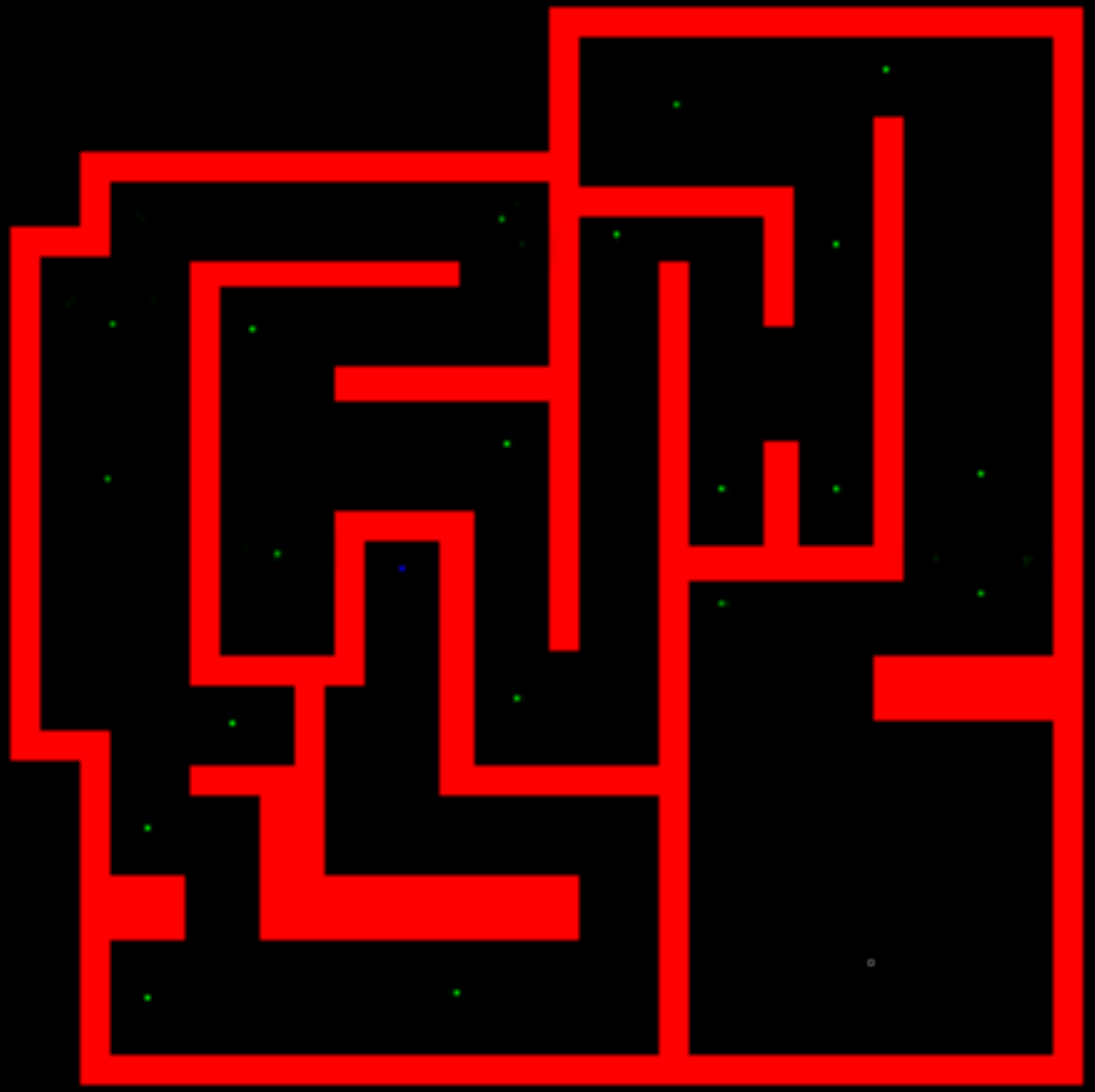
Collision Mapping

- The wrong way
- Wrong algorithm



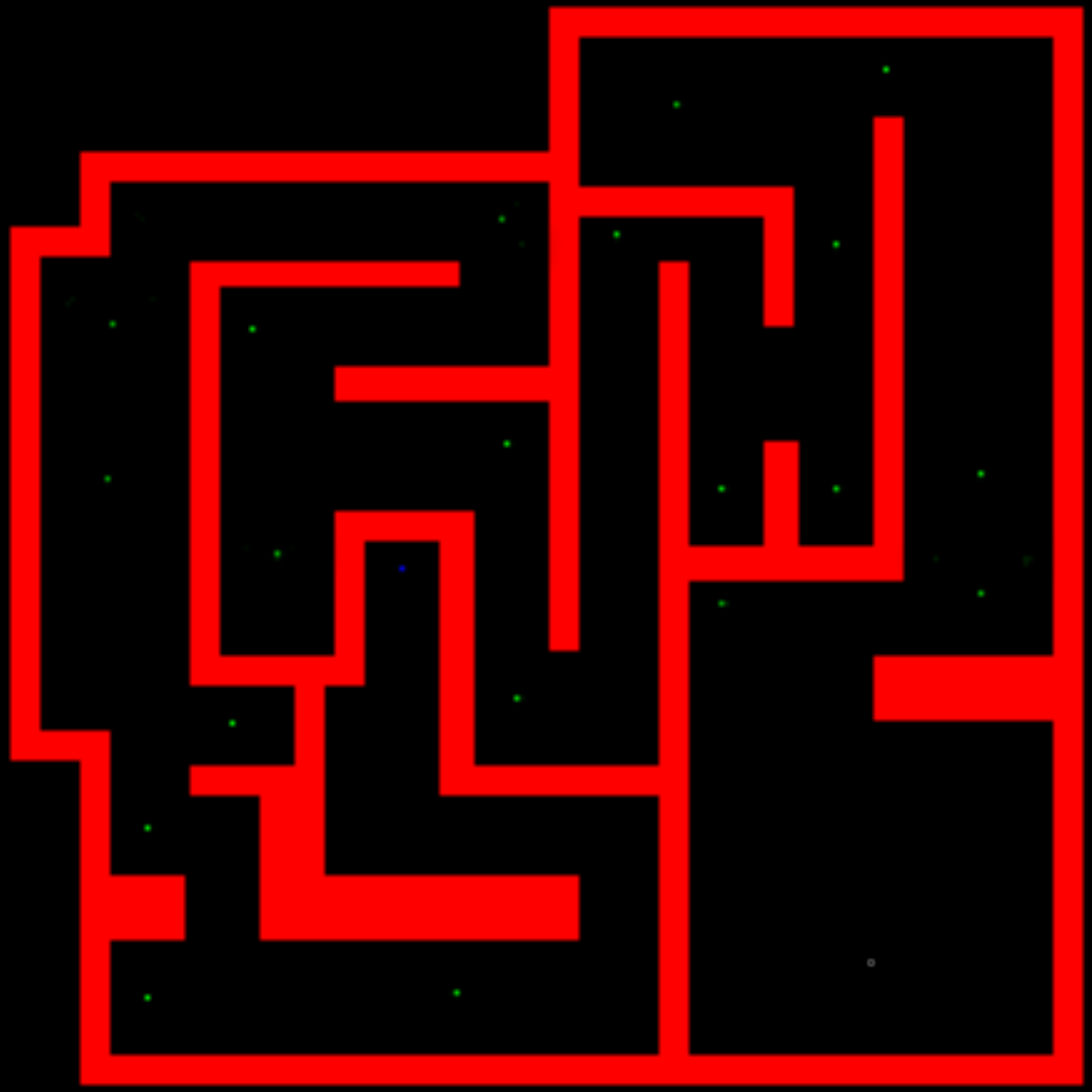
Collision Mapping

- The wrong way
- Wrong algorithm
- Too much code



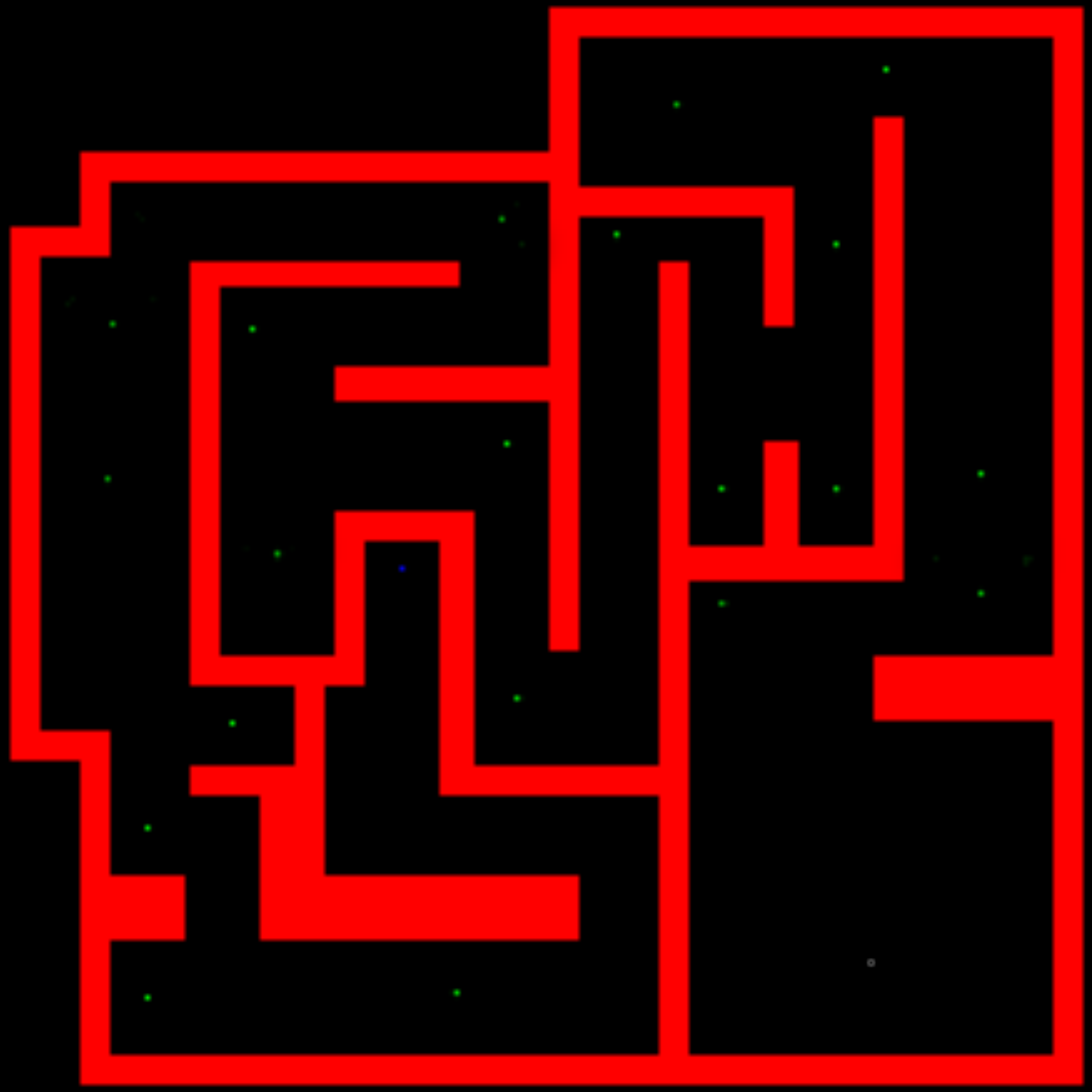
Collision Mapping

- The wrong way
- Wrong algorithm
- Too much code
- Worked terribly



Collision Mapping

- The wrong way
- Wrong algorithm
- Too much code
- Worked terribly
- Seemed obvious



Collision Mapping

Collision Mapping

- The right way

Collision Mapping

- The right way
- Actually turned out to be ZERO lines of code

Collision Mapping

- The right way
- Actually turned out to be ZERO lines of code

```
CGRect rect;
....
sprite.physicsBody = [SKPhysicsBody bodyWithRectangleOfSize:rect.size];
sprite.physicsBody.dynamic = NO;
sprite.physicsBody.categoryBitMask = kColliderWall;
[self addNodeToWorld:sprite atLayer:kLayerGround];
....
```

Lesson Two

Building Art

Art Pipeline

- Plan your art
- Limit use of system resources
- Build only what you need

Production

Production

Production

Production

Production

Production

Production

Production

Texture Atlas

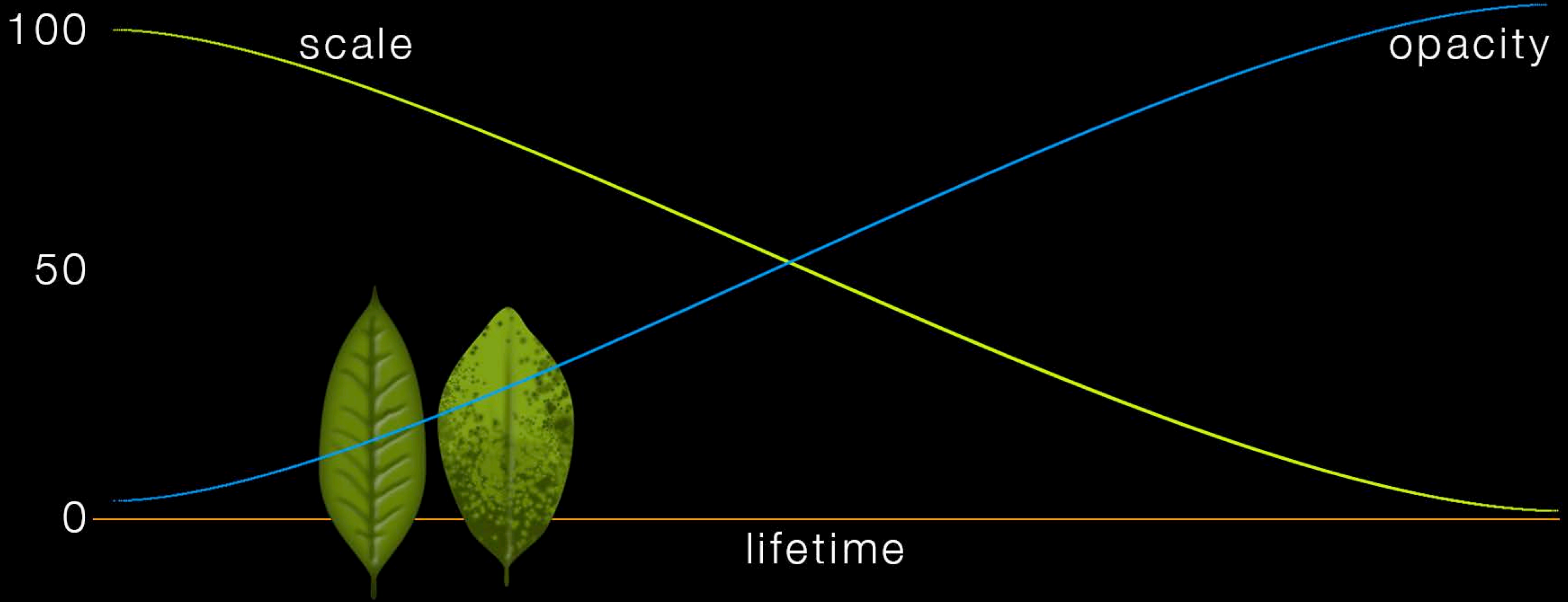
=

Texture Atlas

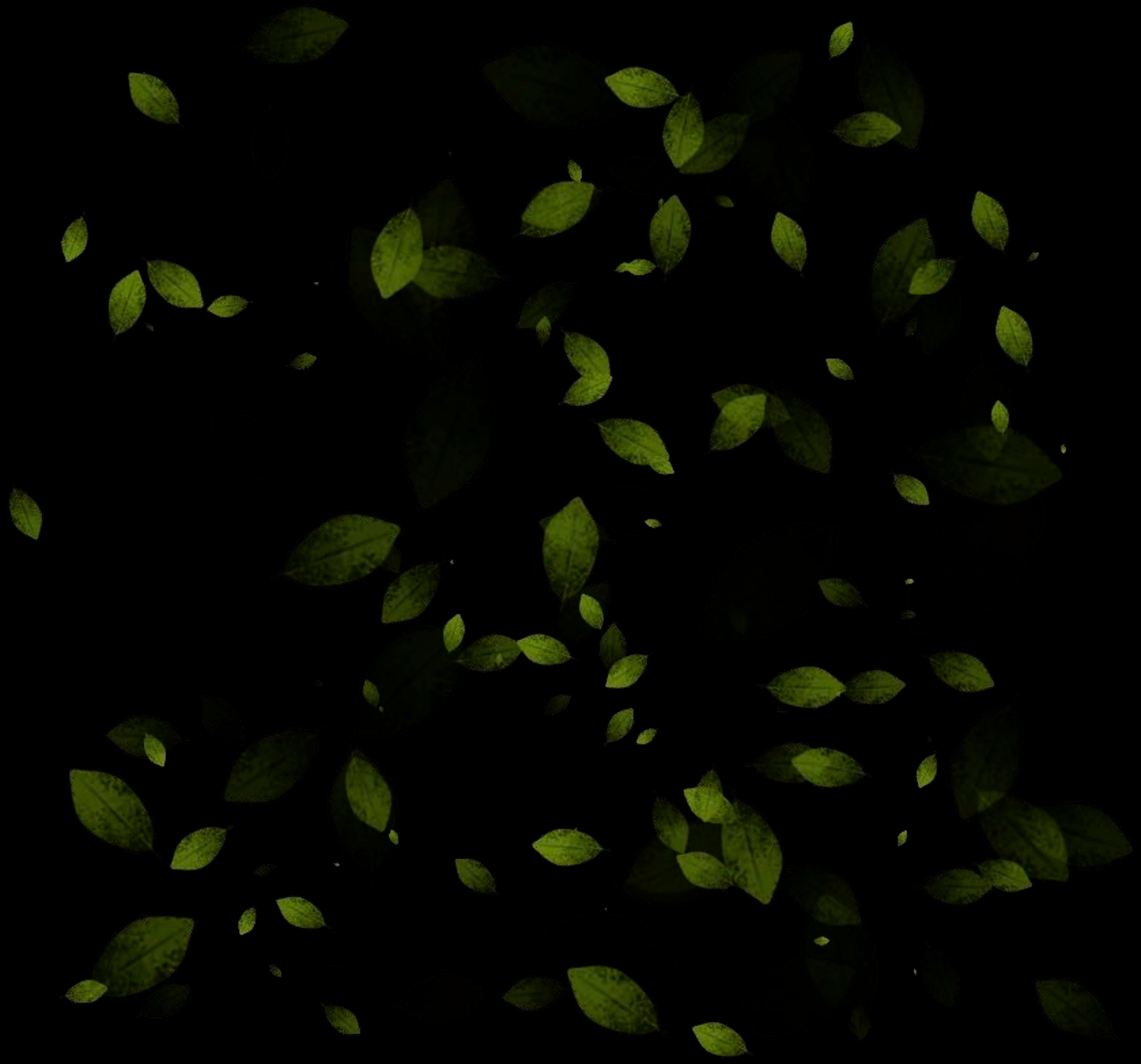
=

Particles

Particles



Result



Lesson Three

Agree on Stuff

Pipeline

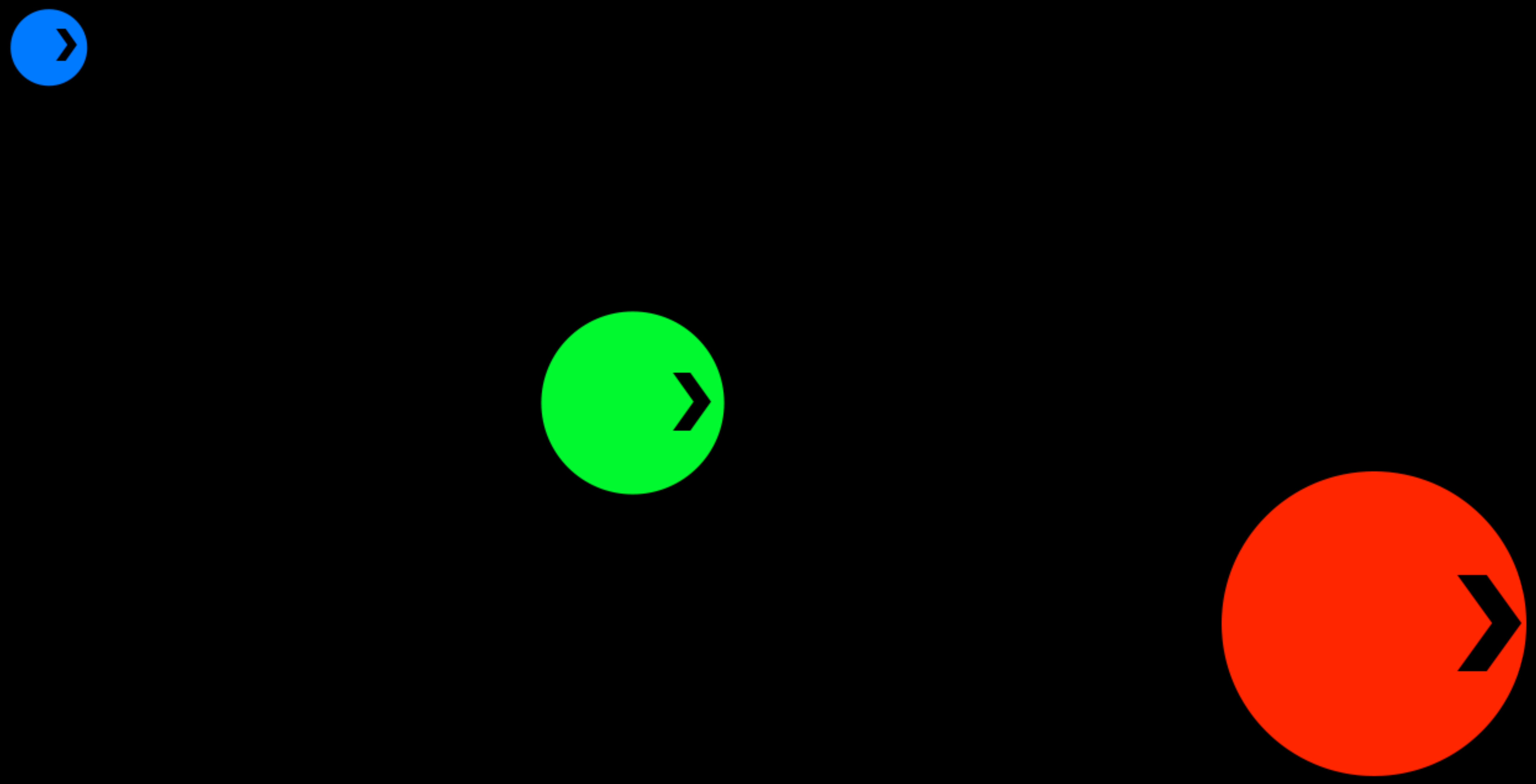
- Communication
- Naming scheme
- Folder structure
- Coordinate system
- Orientation

Production

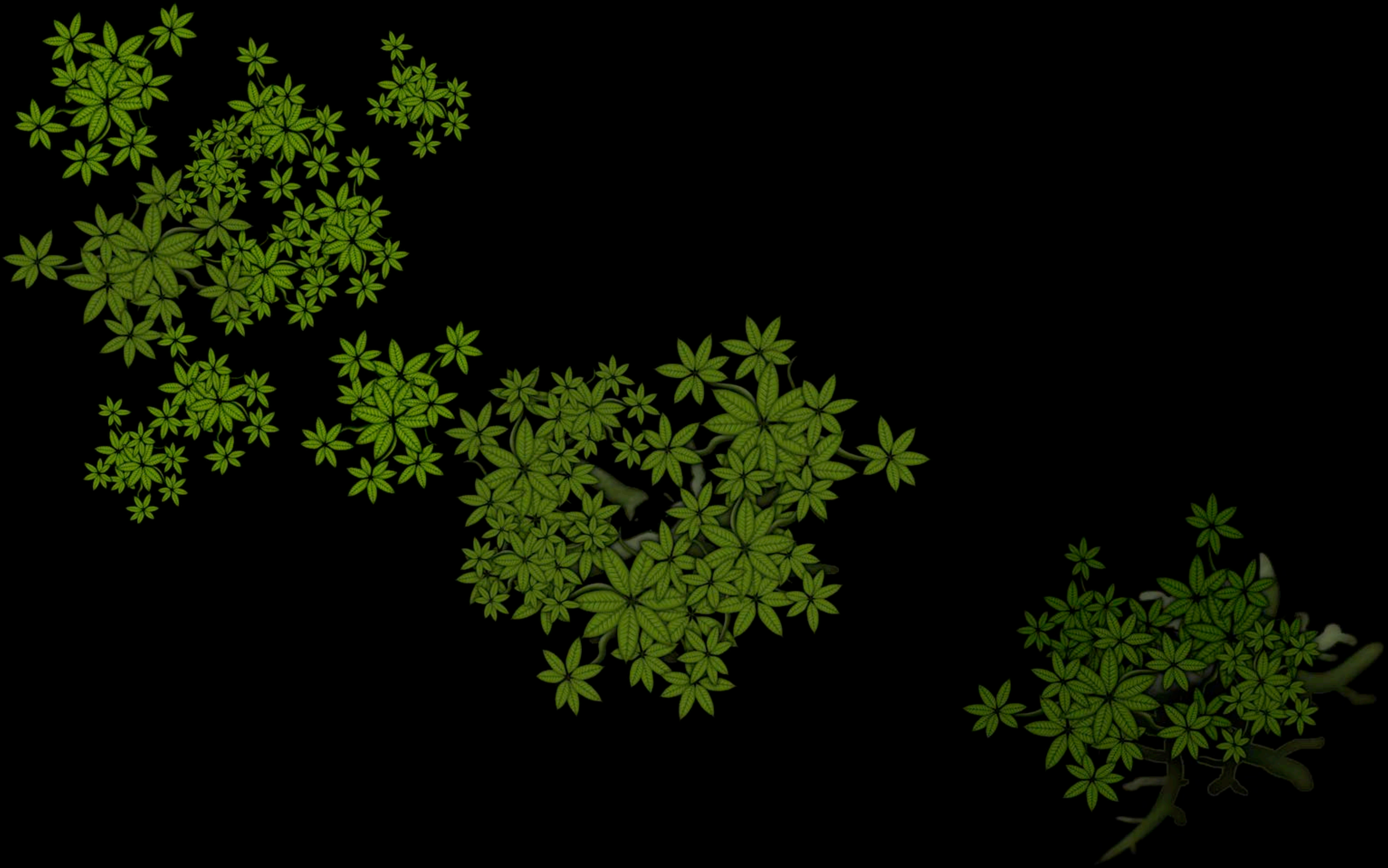
- Programmer art is good
- Acts as a map or guide for the art team to follow
- Helps start a conversation about visuals and code

Graeme’s Art

Graeme’s Art

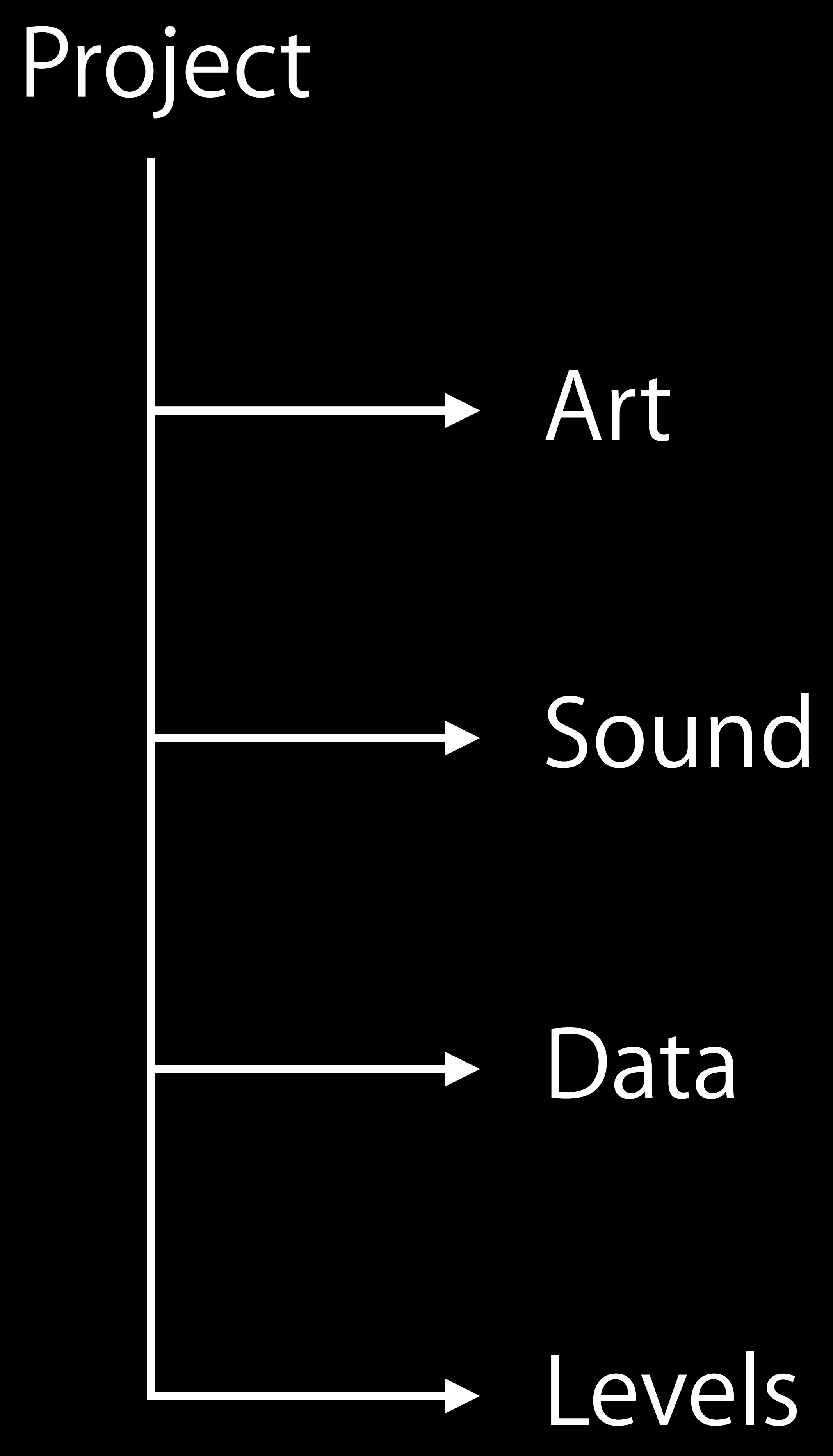


Spencer’s Art

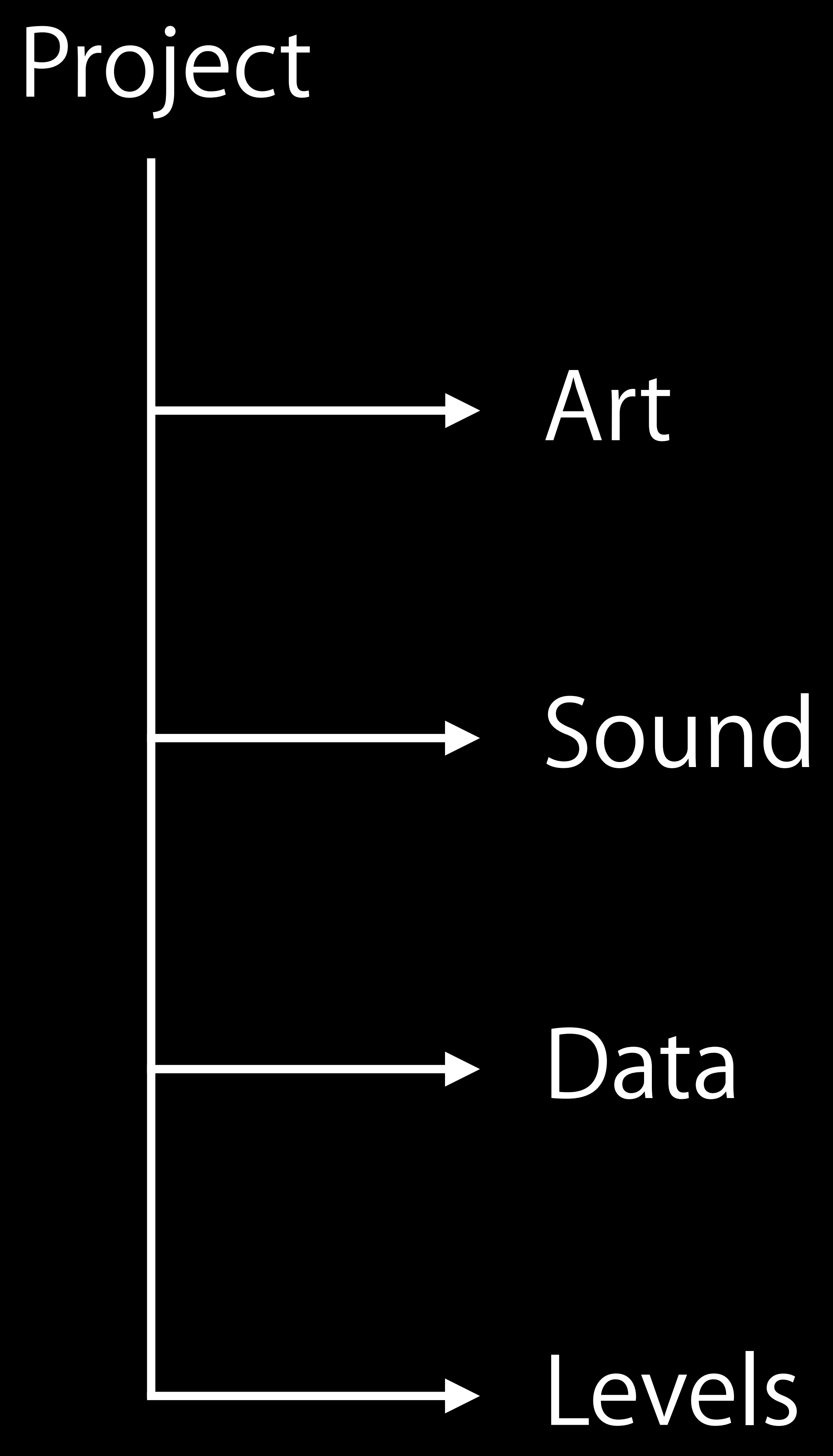


Folder Structure

Folder Structure



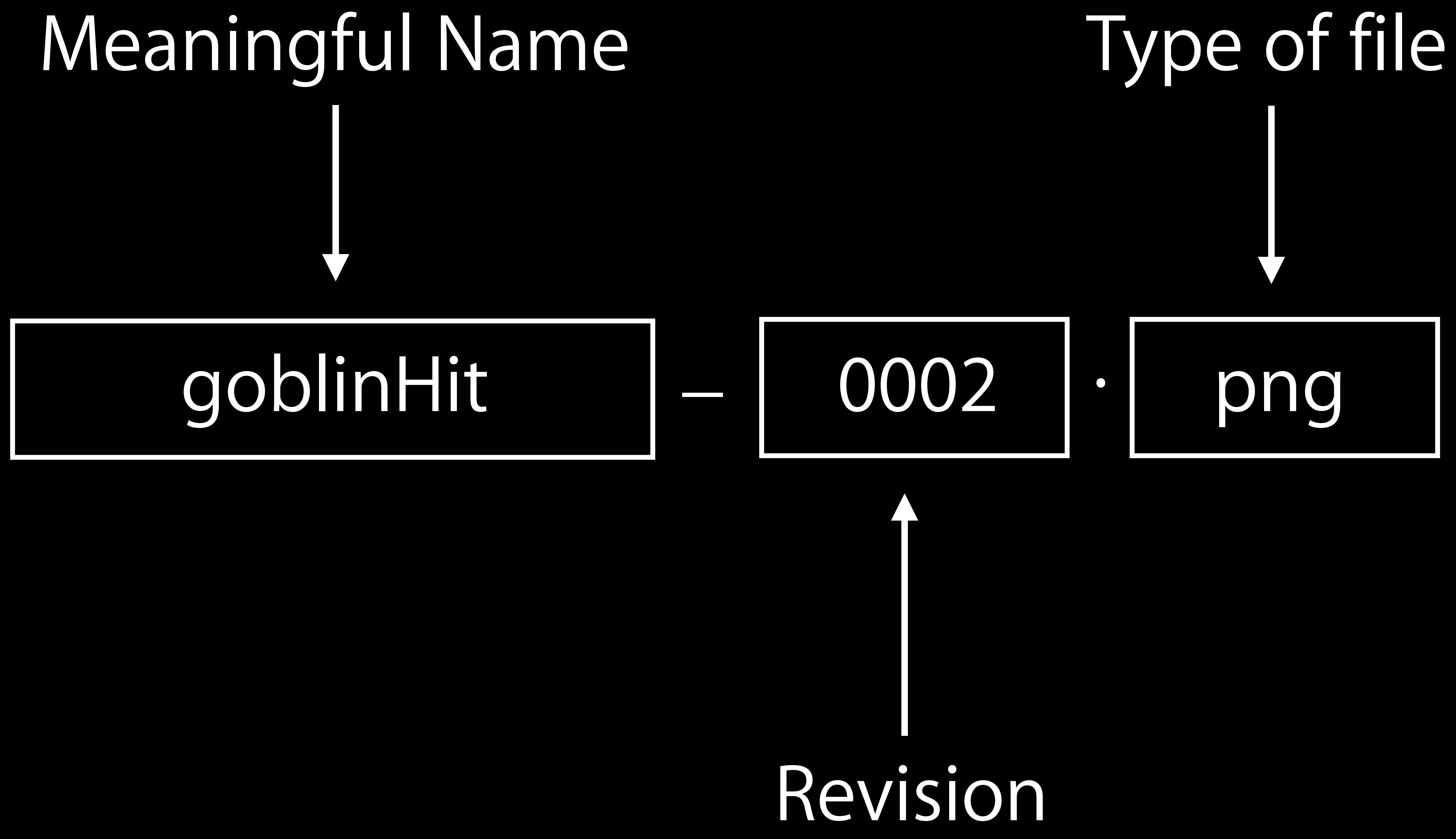
Folder Structure



Folders in Xcode and Project

File Names

File Names



Lesson Four

Have Fun

Lessons

- Just because it’s 2D doesn’t make it 2D
- Physics is useful for more than just physics
- Placeholder assets let the project run smoothly
- Art production—Less is more

Building Custom Tools

Extending Adventure

- Add multiple level support
    - And different collision map for each level
- More sophisticated and reusable `SKActions`
- Allow players to save and load game progress

Encoding and Decoding Nodes

- All SKNodes support the `NSCopying` and `NSCoding` protocols
- Sprite Kit nodes can be archived
  - Serialize/deserialize an entire scene with two lines of code
    - Quickly save/load game progress
    - Making a level editor for your game
  - Particle Editor is built on top of this

Serialize/Deserialize API

- To serialize any node, or tree of nodes, use `NSKeyedArchiver`

```
NSData *data = [NSKeyedArchiver archivedDataWithRootObject:node];
BOOL success = [data writeToURL:url options:... error:&anyError];
```

Serialize/Deserialize API

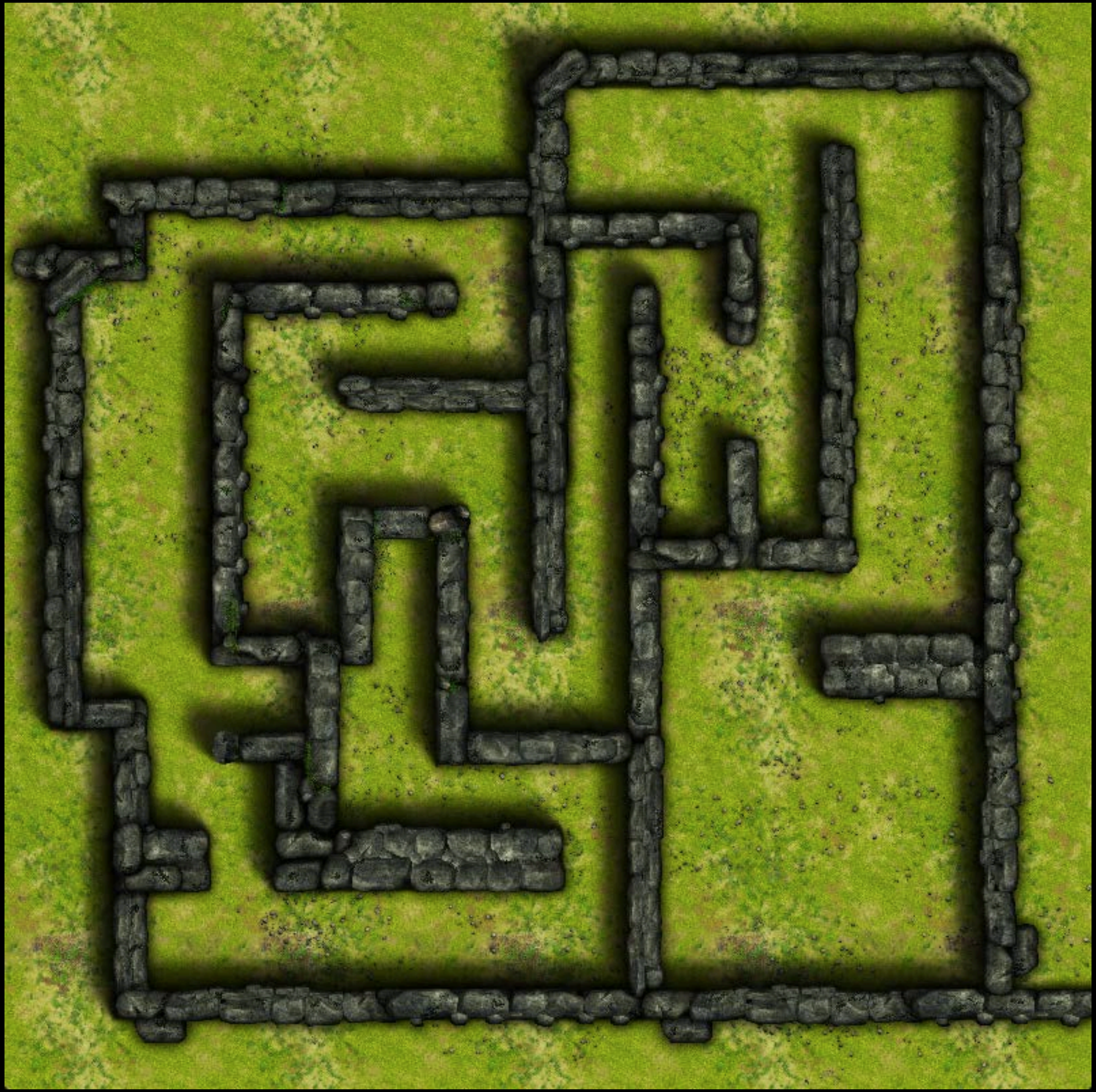
- To serialize any node, or tree of nodes, use NSKeyedArchiver

```
NSData *data = [NSKeyedArchiver archivedDataWithRootObject:node];
BOOL success = [data writeToURL:url options:... error:&anyError];
```

- To deserialize, use NSKeyedUnarchiver

```
NSData *data = [NSData dataWithContentsOfURL:url options:... error:&anyError];
SKNode *node = [NSKeyedUnarchiver unarchiveObjectWithData:data];
```

Custom Tool Implementation



Custom Tool Implementation

- Deserialize `SKScene` from saved data
- Use an overlay view to manipulate each `SKNode` in the scene
    - Supports add, remove, move, scale with overlay UI
    - Supports editing functionalities
        - Group select, copy, paste, undo and redo
- Ability to pause/unpause physics simulation
- Serialize the output to disk

Custom Actions

Custom Actions

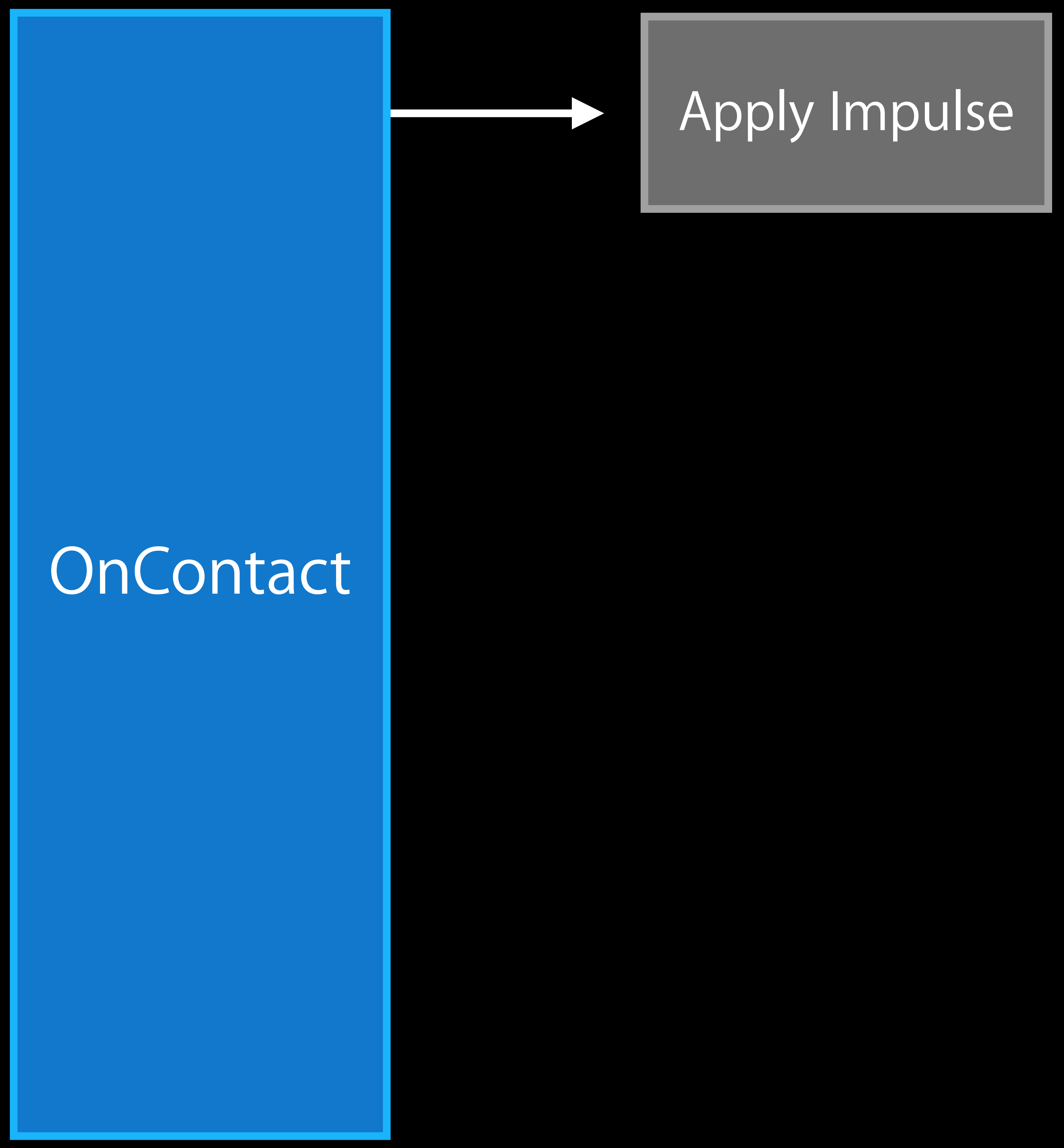
- To simulate a golf club hitting a golf ball

Custom Actions

- To simulate a golf club hitting a golf ball

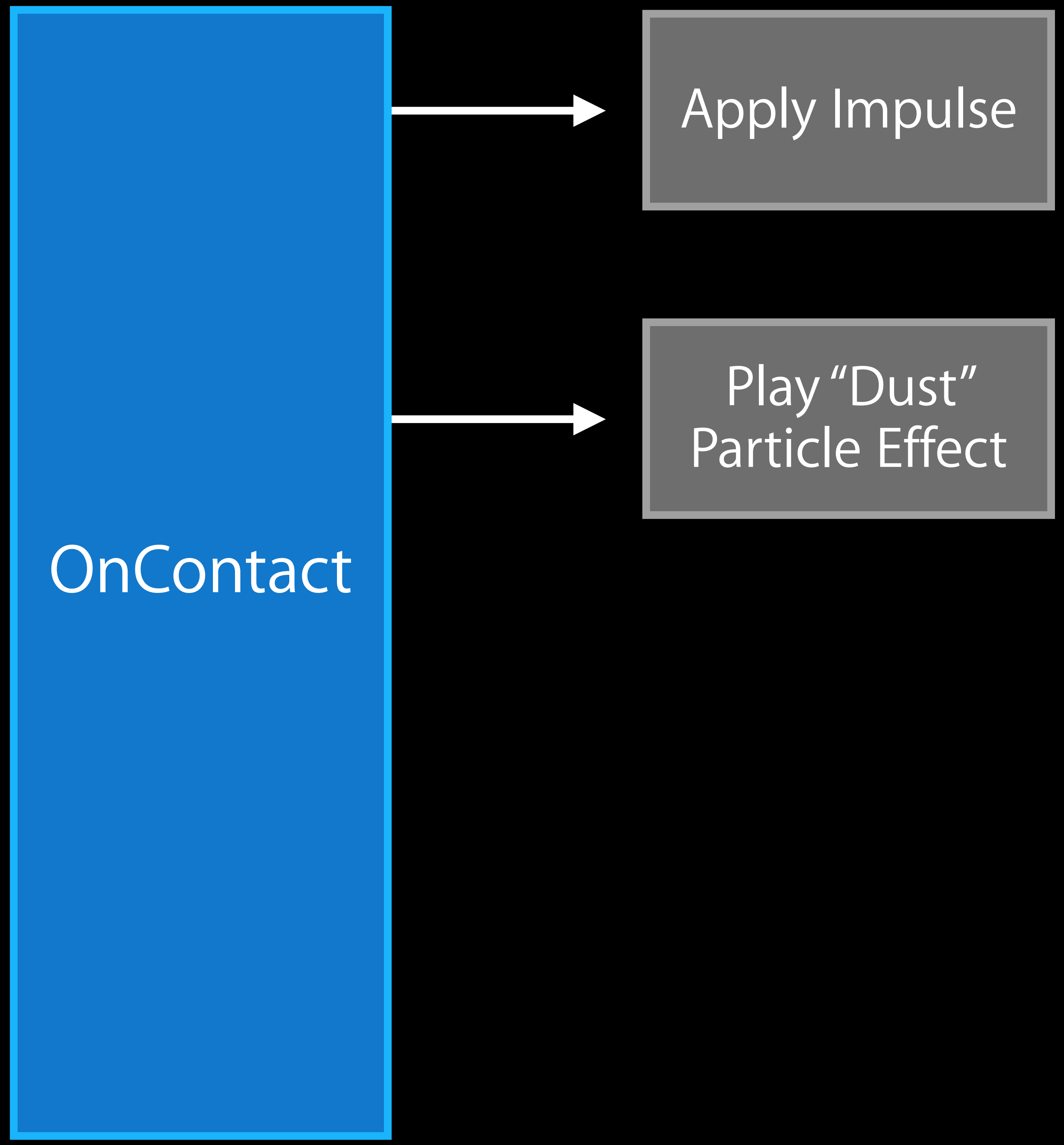
Custom Actions

- To simulate a golf club hitting a golf ball



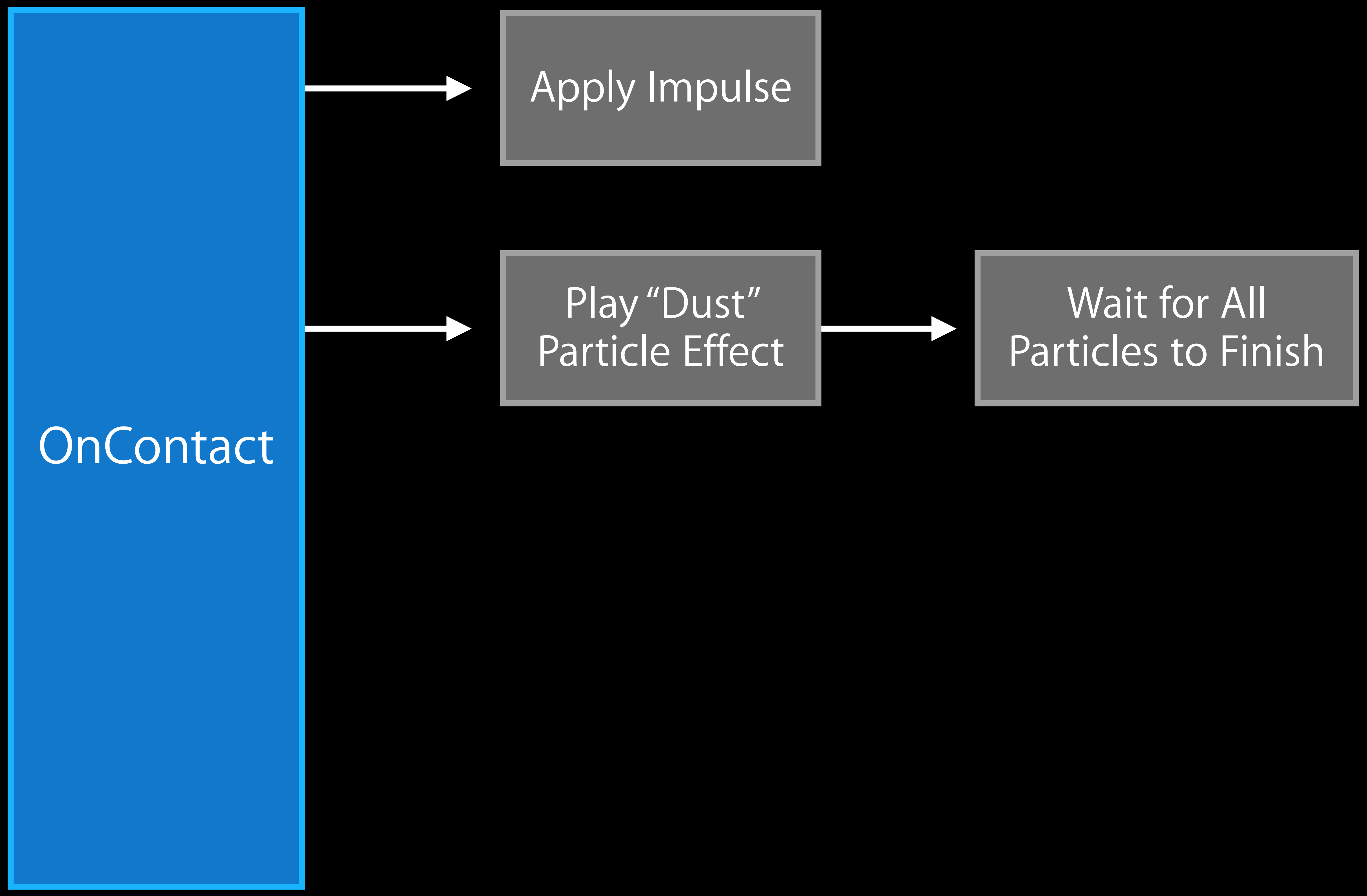
Custom Actions

- To simulate a golf club hitting a golf ball



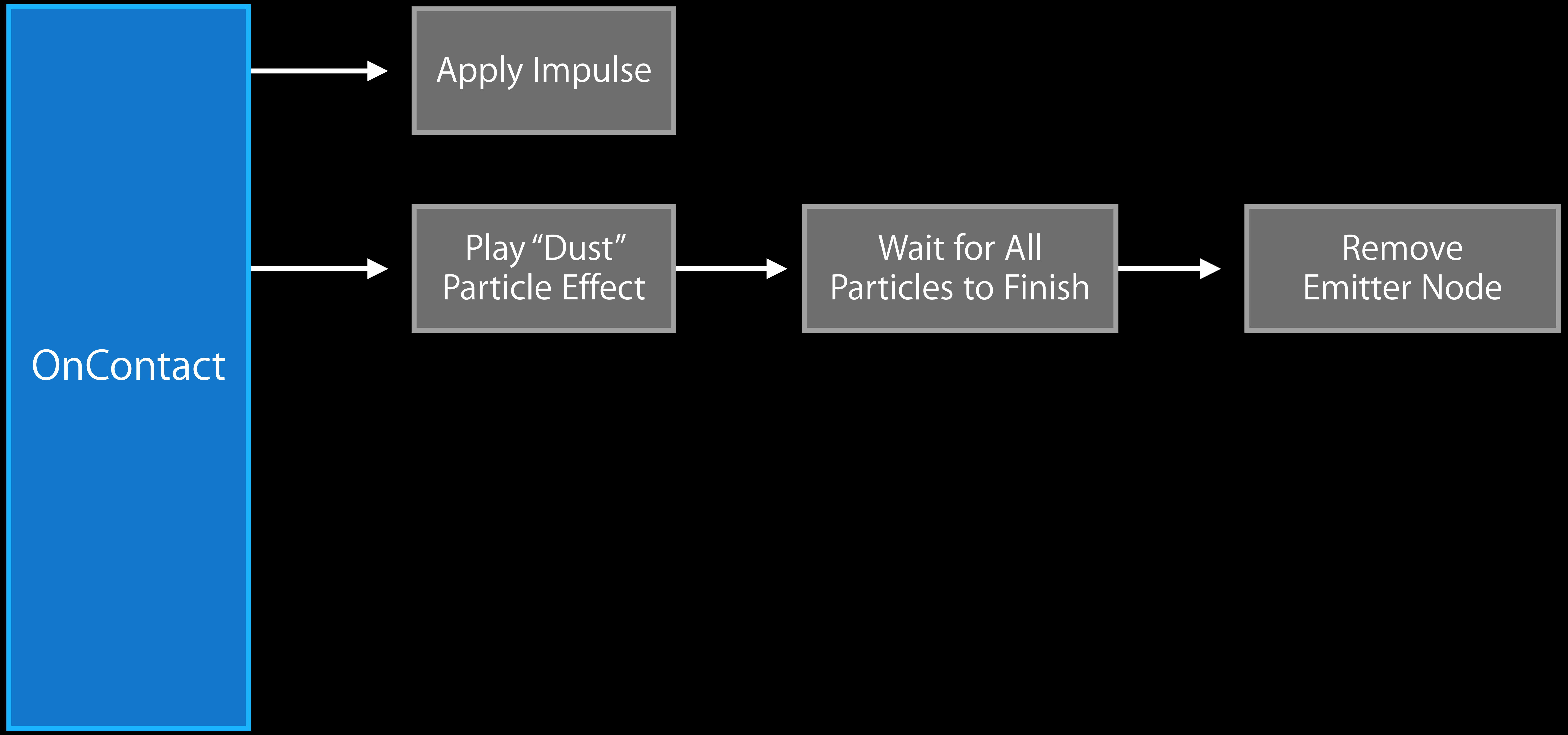
Custom Actions

- To simulate a golf club hitting a golf ball



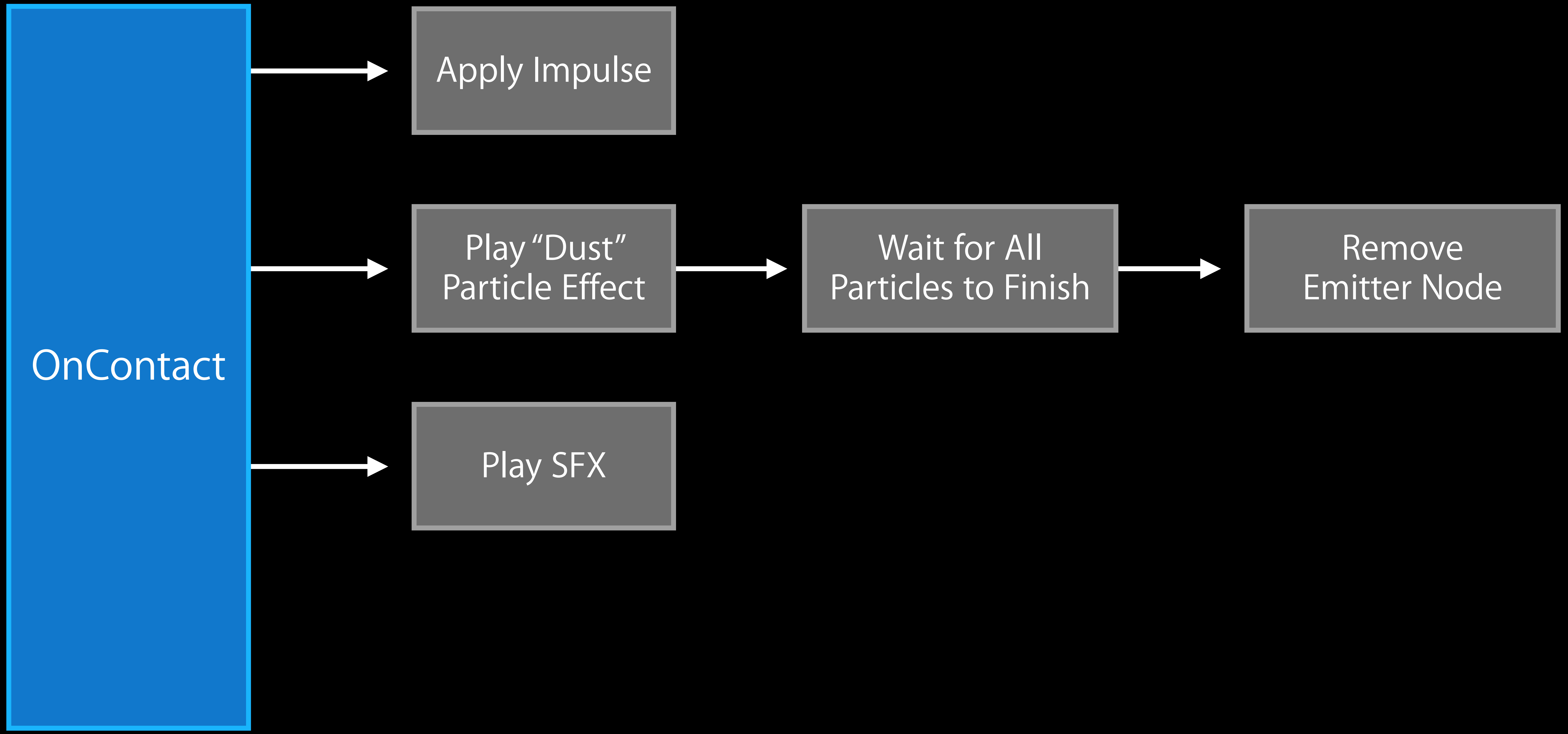
Custom Actions

- To simulate a golf club hitting a golf ball



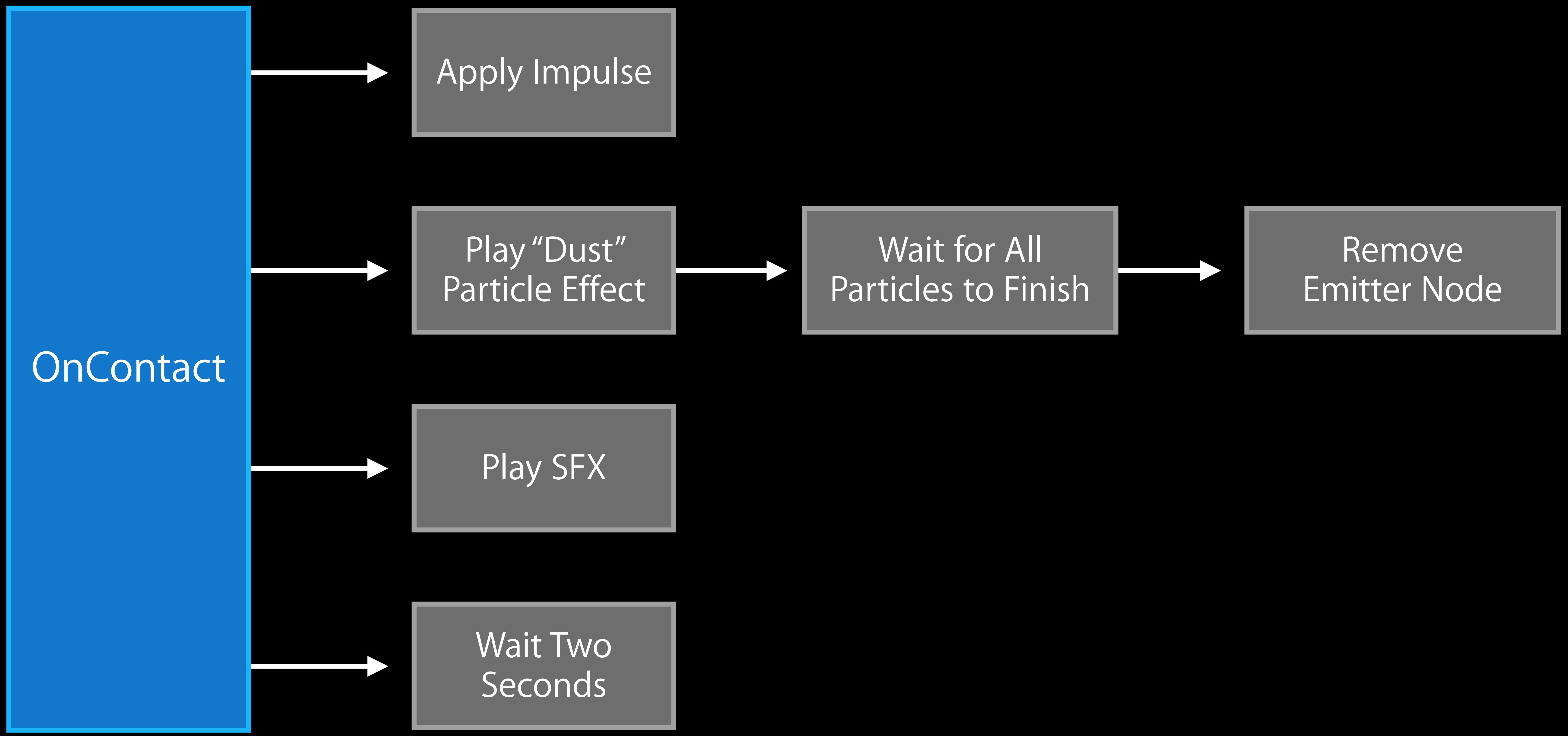
Custom Actions

- To simulate a golf club hitting a golf ball



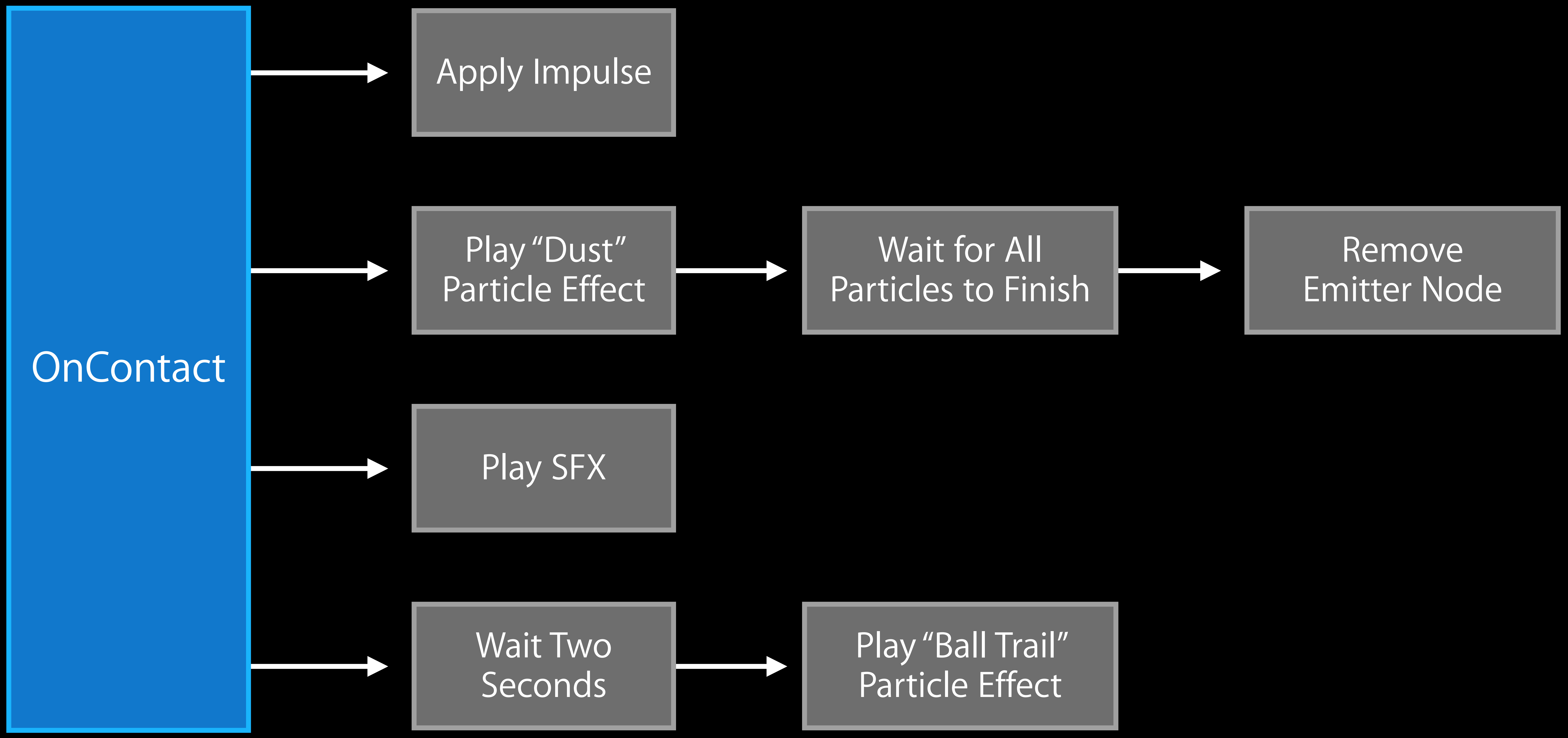
Custom Actions

- To simulate a golf club hitting a golf ball



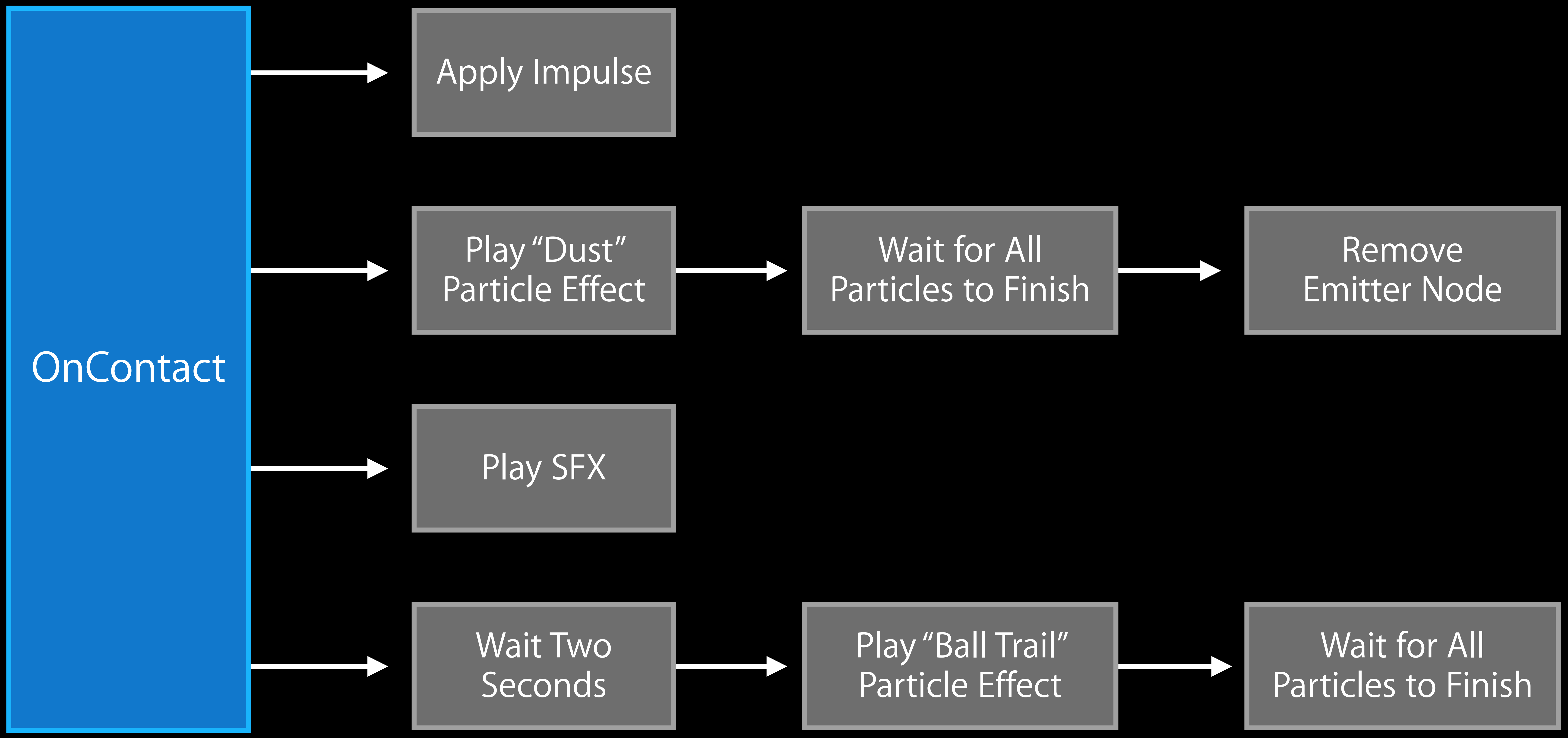
Custom Actions

- To simulate a golf club hitting a golf ball


OnContact
Apply Impulse
Play "Dust" Particle Effect
Wait for All Particles to Finish
Remove Emitter Node
Play SFX
Wait Two Seconds
Play "Ball Trail" Particle Effect

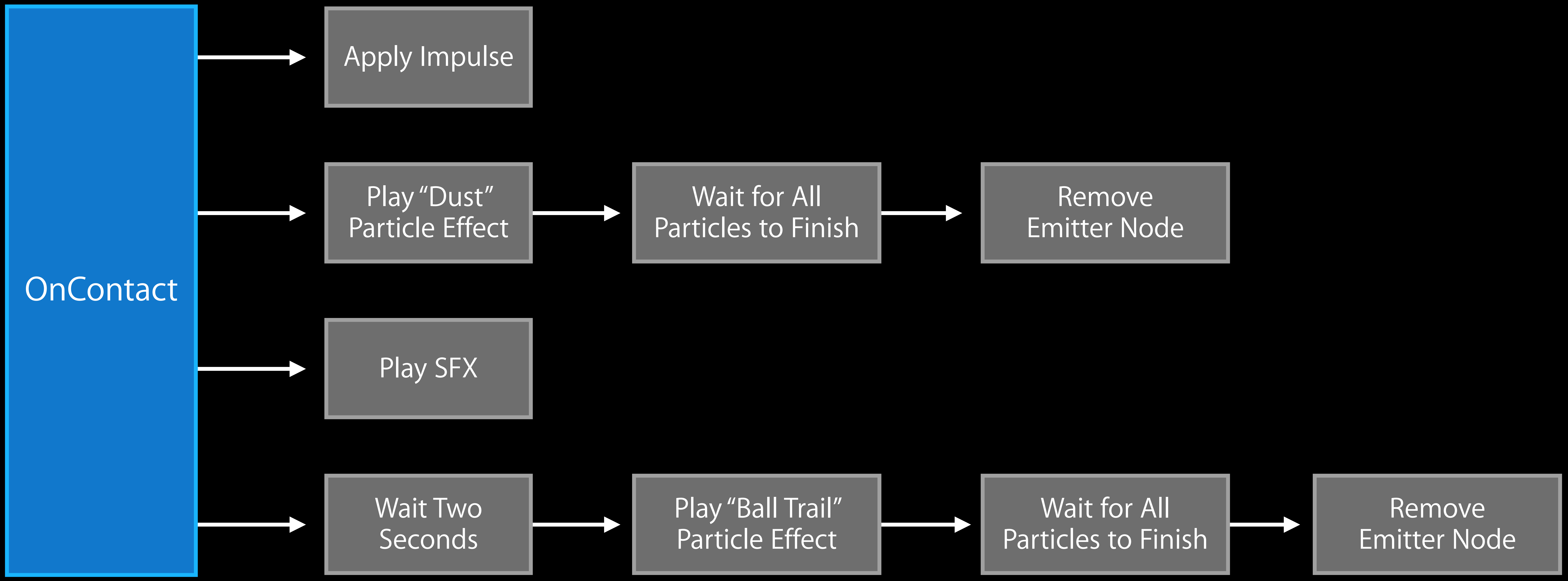
Custom Actions

- To simulate a golf club hitting a golf ball


OnContact
Apply Impulse
Play "Dust" Particle Effect
Wait for All Particles to Finish
Remove Emitter Node
Play SFX
Wait Two Seconds
Play "Ball Trail" Particle Effect
Wait for All Particles to Finish

Custom Actions

- To simulate a golf club hitting a golf ball


OnContact
Apply Impulse
Play "Dust" Particle Effect
Wait for All Particles to Finish
Remove Emitter Node
Play SFX
Wait Two Seconds
Play "Ball Trail" Particle Effect
Wait for All Particles to Finish
Remove Emitter Node

Custom SKActions

- `SKActions` are also `NSCoding` compliant
- Complicated `SKActions` can be serialized and loaded
- `SKNode` copies `SKAction` on write
- `SKAction` resets when assign to another `SKNode`
- Completed `SKActions` will be removed

Best Practices

Use `UIKit` or `AppKit` Controls

- Use standard controls as subviews of `SKView`
- As with other GL views, must be layer-backed on OS X

Adventure.xcodeproj — Main_iPhone.storyboard
Adventure
iPhone
Build Adventure: Succeeded | Today at 12:57 AM
No Issues
Adventure
Adventure
Main_iPhone.storyboard
Main_iPhone.storyboard (Base)
No Selection
Adventure
2 targets, iOS SDK 7.0
Adventure
AppDelegate.h
AppDelegate.m
Main_iPhone.storyboard
Main_iPad.storyboard
NewGameViewController.h
NewGameViewController.m
SplashScreenViewController.h
SplashScreenViewController.m
Images.xcassets
Supporting Files
AdventureTests
Frameworks
Products
ADVENTURE
New Game
Load Game
Splash Screen View Controller
Play!
Customize
View Controller
Tab Bar Controller
Tab Bar Controller – Customize
Playing a game!
View Controller – Game Play
Done
Customize
Share
Helmet
Viking
Body
Middle Ages Leather
Gloves
None
Durability
Heals with Spells
View Controller – Armor
Done
Customize
Share
Sword
Bow
Knife
Power
View Controller – Weapons
Done
Customize
Share
Healing
Damage
Invisibility
Power
View Controller – Spells

Improve Your Iteration Time

- Integrate Sprite Kit with game tools
    - Provide fast iteration on the assets
- Build the content in the tools offline
- Data-driven model allows everyone to collaborate in parallel

Performance Tips

- Use built-in stats from `SKView`
    - Show number of nodes, and number of draw calls
- Keep the node count low
    - Remove offscreen nodes
- If draw count is high, use texture atlas
- `CIFilters` are expensive
- Take advantage of `shouldRasterize` on `SKEffectNode`
- If game needs full screen filter
    - Consider raster to texture first

Organize Game Content into Scenes

- Scenes are the fundamental building block
- Define which scenes are needed
    - Similar to the role of view controllers
    - Easier to design transitions
    - How data is transferred from between scenes
- Sprite Kit culls out invisible nodes
- Add nodes to scene graph as necessary

More Information

Allan Schaffer
Graphics and Game Technologies Evangelist
aschaffer@apple.com

Apple Developer Forums
http://devforums.apple.com/

Developer Documentation
http://developer.apple.com/library/

Related Sessions

Integrating with Game Controllers	Pacific Heights Tuesday 3:15PM
Introduction to Sprite Kit	Presidio Wednesday 11:30AM

Labs

Sprite Kit Lab	Graphics and Games Lab B Wednesday 3:15PM
Sprite Kit Lab	Graphics and Games Lab B Thursday 9:00AM

WWDC2013