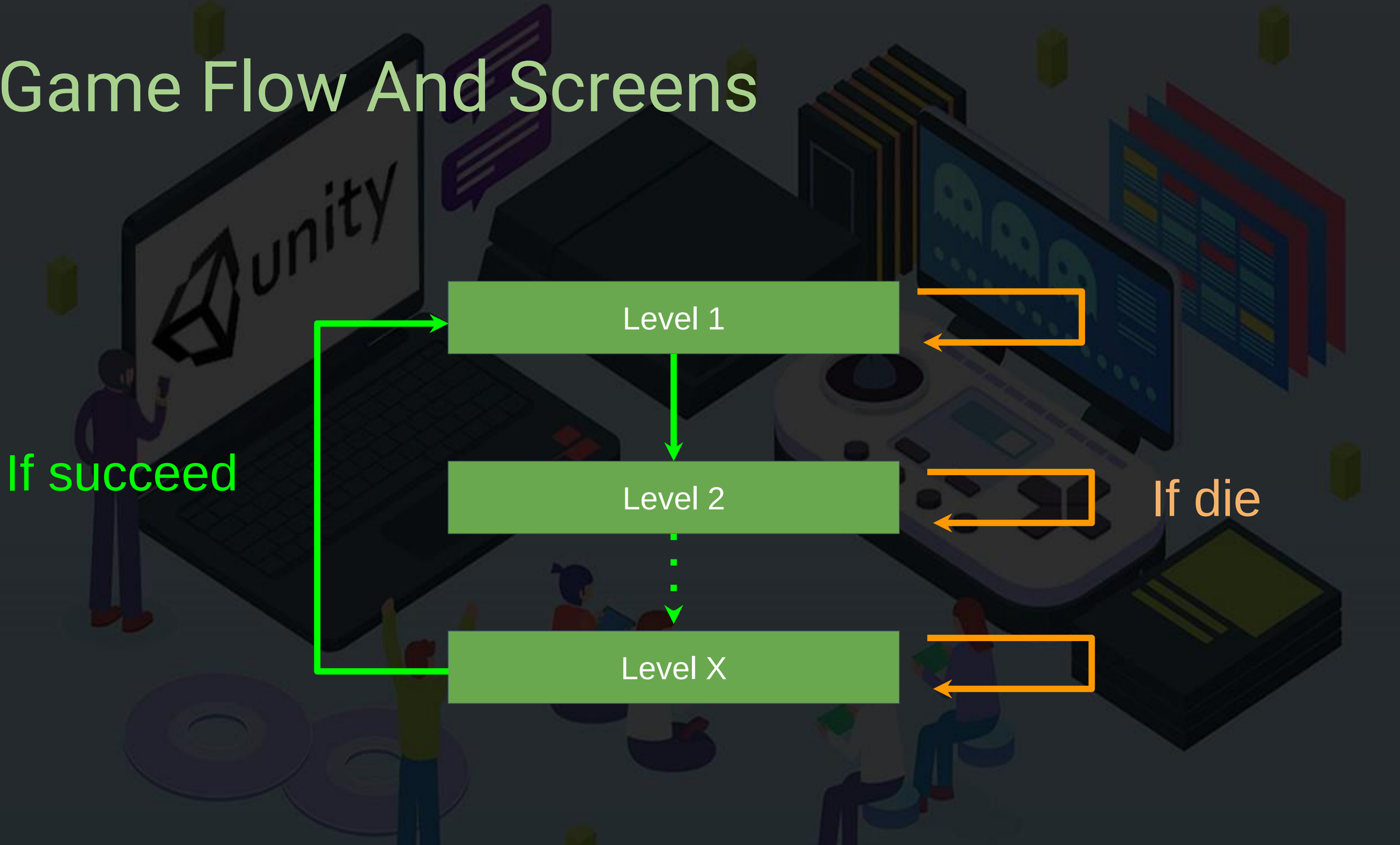
Section 5 – Sprooky the Rocket

Project Boost Game Design

Player Experience:
Precision. Skilful.
Core Mechanic:
Skilfully fly spaceship and avoid environmental hazards.
Core game loop:
Get from A to B to complete the level, then progress to the next level.

Game Flow And Screens

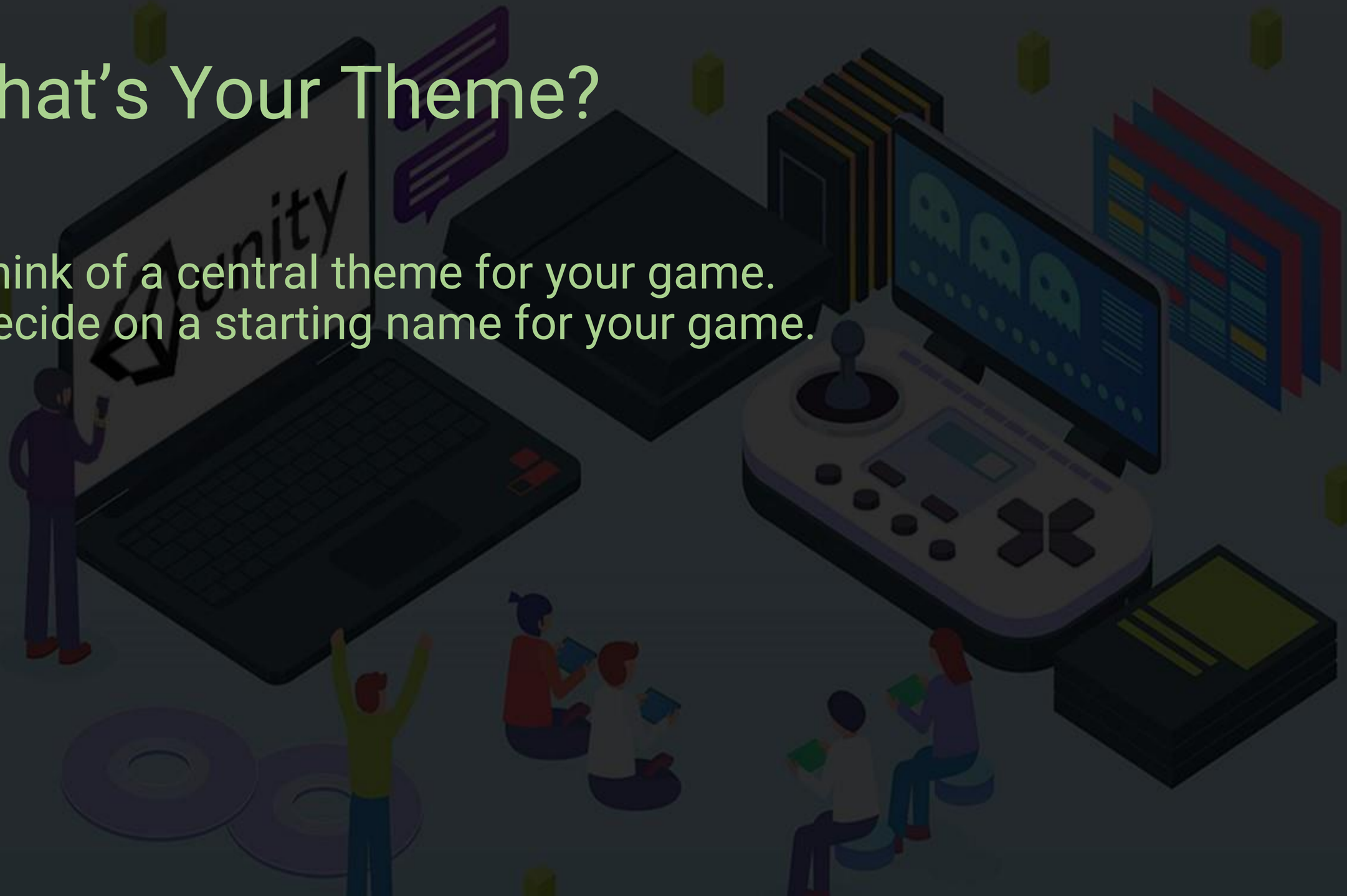


Game Theme (ie. Story & Visuals)

- Experimental early generation spacecraft.
  On an unknown planet, trying to escape.

What’s Your Theme?

Think of a central theme for your game.
Decide on a starting name for your game.



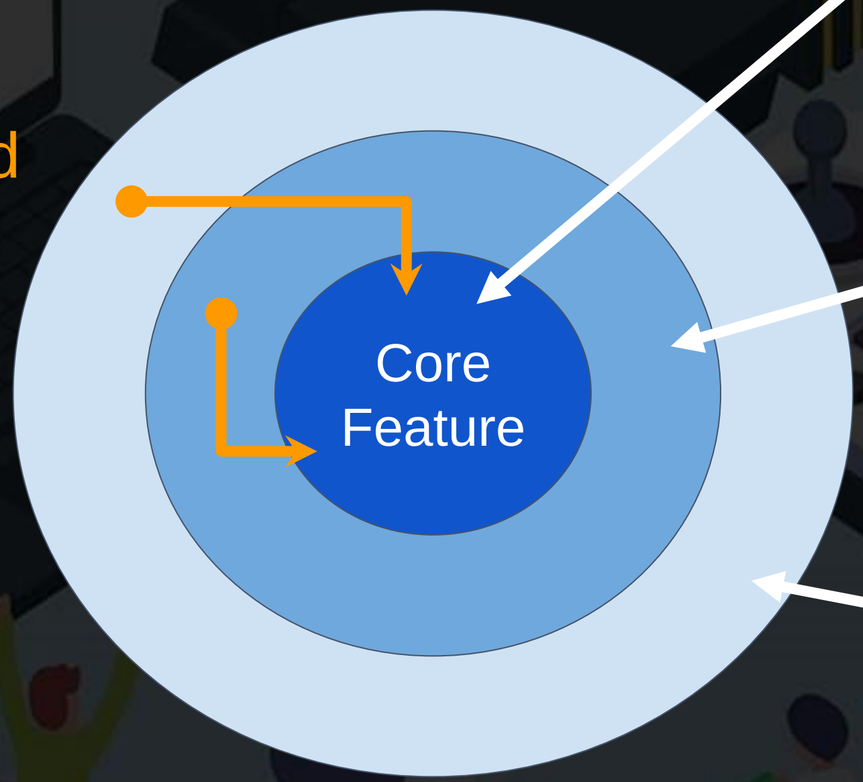
Onion Design

Common Development Questions

- What features should I include in my game?
- Where should I start development?
- What are my priorities?
- What if I run out of time?
- When should I stop?

Onion Design

All features need to feed the core and make it better



Most important feature

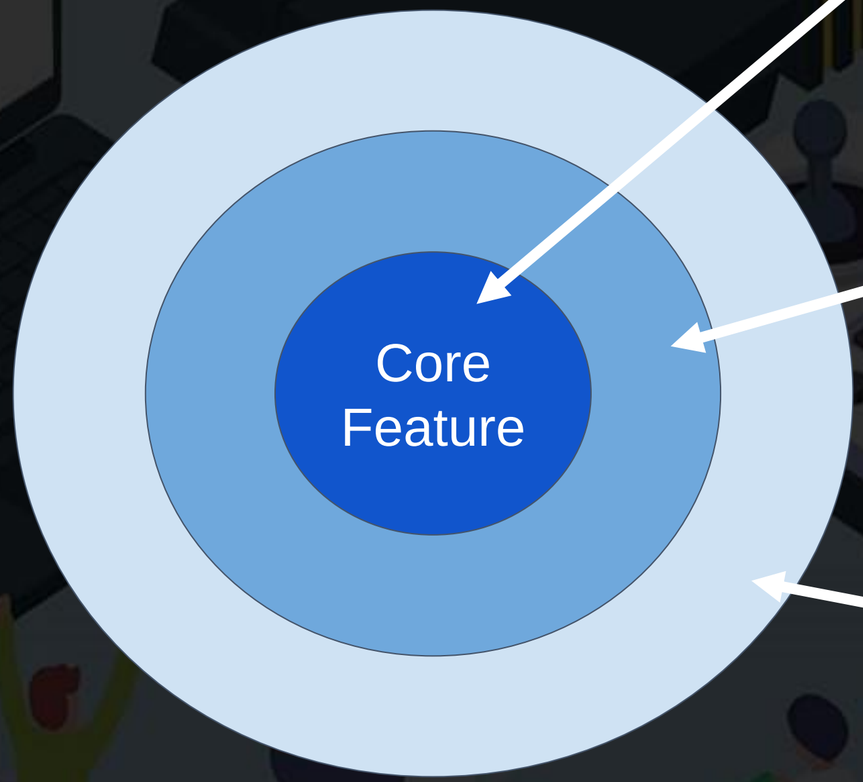
2nd most important feature

3rd most important feature

Sketch Our Onion Design

- What do you believe is the single most important feature of our game?
- What is next most important?

Onion Design For Project Boost

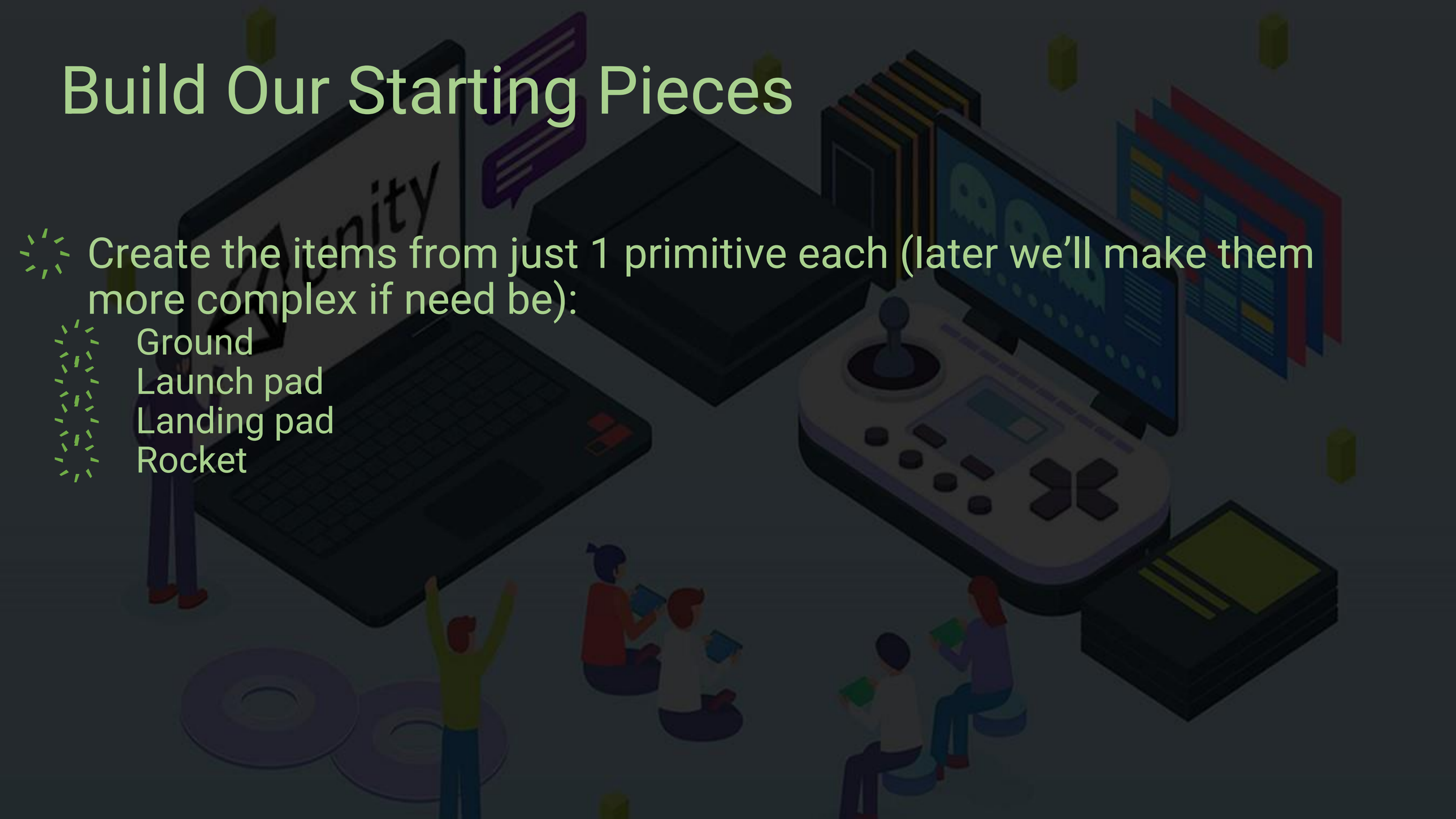


1. Movement / flying
2. World collision
3. Level progression

Which Axis Is Which...

+x = right
+y = up
+z = forward

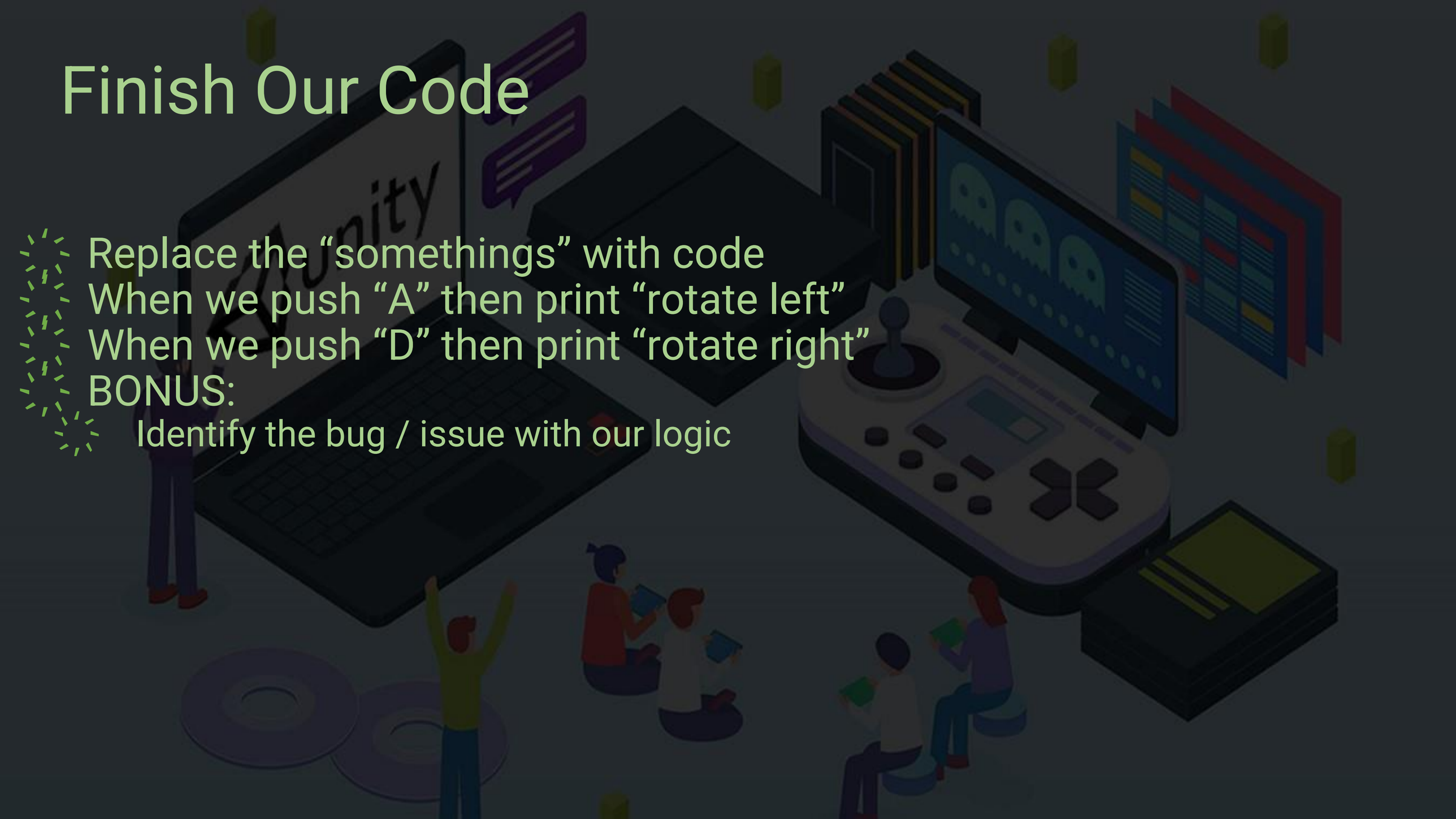
Build Our Starting Pieces



- Create the items from just 1 primitive each (later we'll make them more complex if need be):
  - Ground
  - Launch pad
  - Landing pad
  - Rocket

Basic Input Binding

Finish Our Code



- Replace the “somethings” with code
- When we push “A” then print “rotate left”
- When we push “D” then print “rotate right”
- BONUS:
  - Identify the bug / issue with our logic

Using AddRelativeForce()

Tuning Our Rocket

- Add a variable so we can tune the main rocket thrust
- Make the force applied frame rate independent

Transform.Rotate() Our Rocket

Tuning Our Rocket’s Rotation

- Add a variable so we can tune the rotation speed
- Make the rotation speed frame rate independent

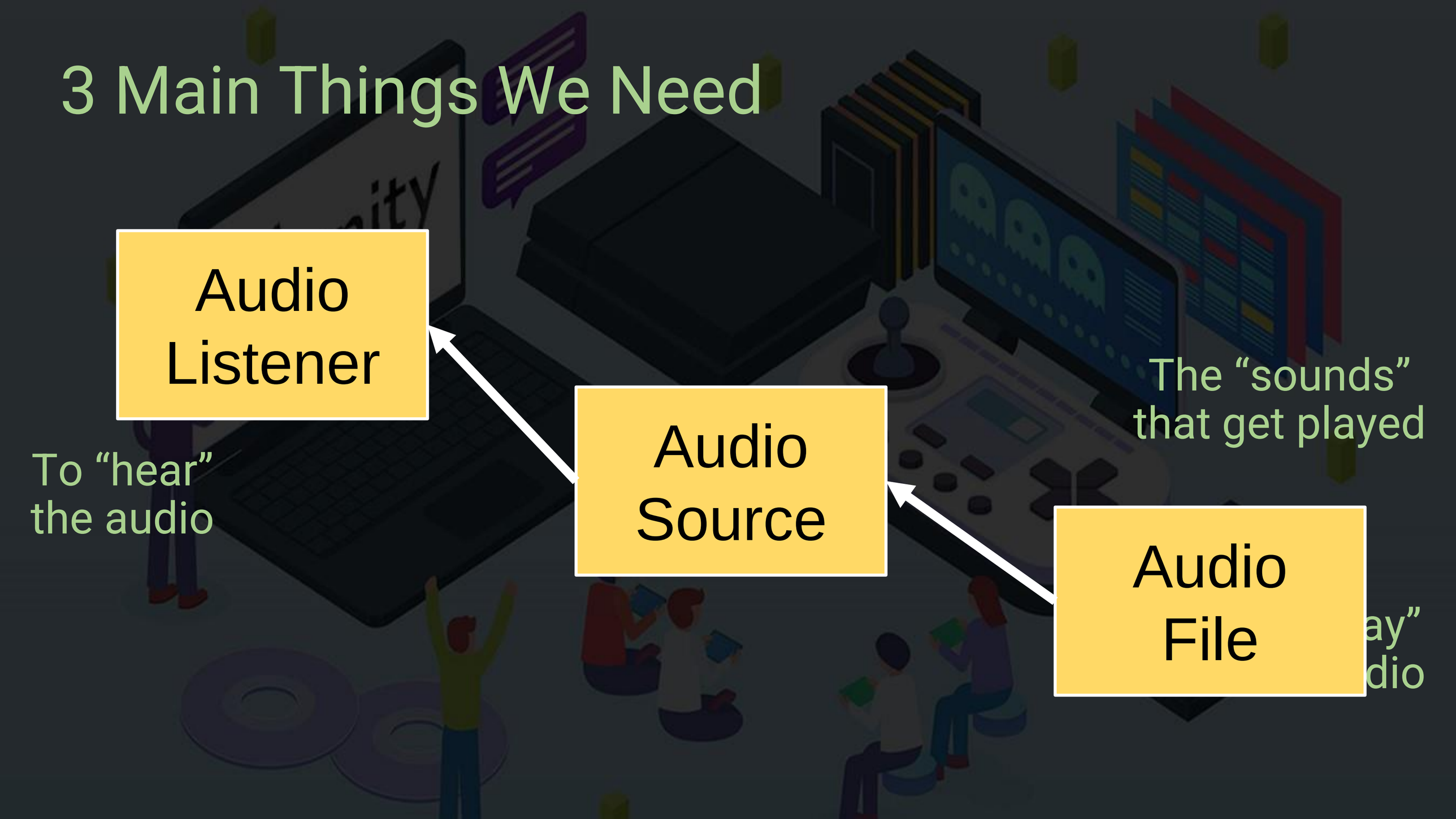
Rigidbody Constraints

Tidy Up Our Movement

- Apply changes to prefab
- Add an obstacle
- Freeze constraints for our rocket
- Fix our obstacle-hitting bug
- Set our drag value

Unity Audio Introduction

3 Main Things We Need



Audio Listener

To “hear” the audio

Audio Source

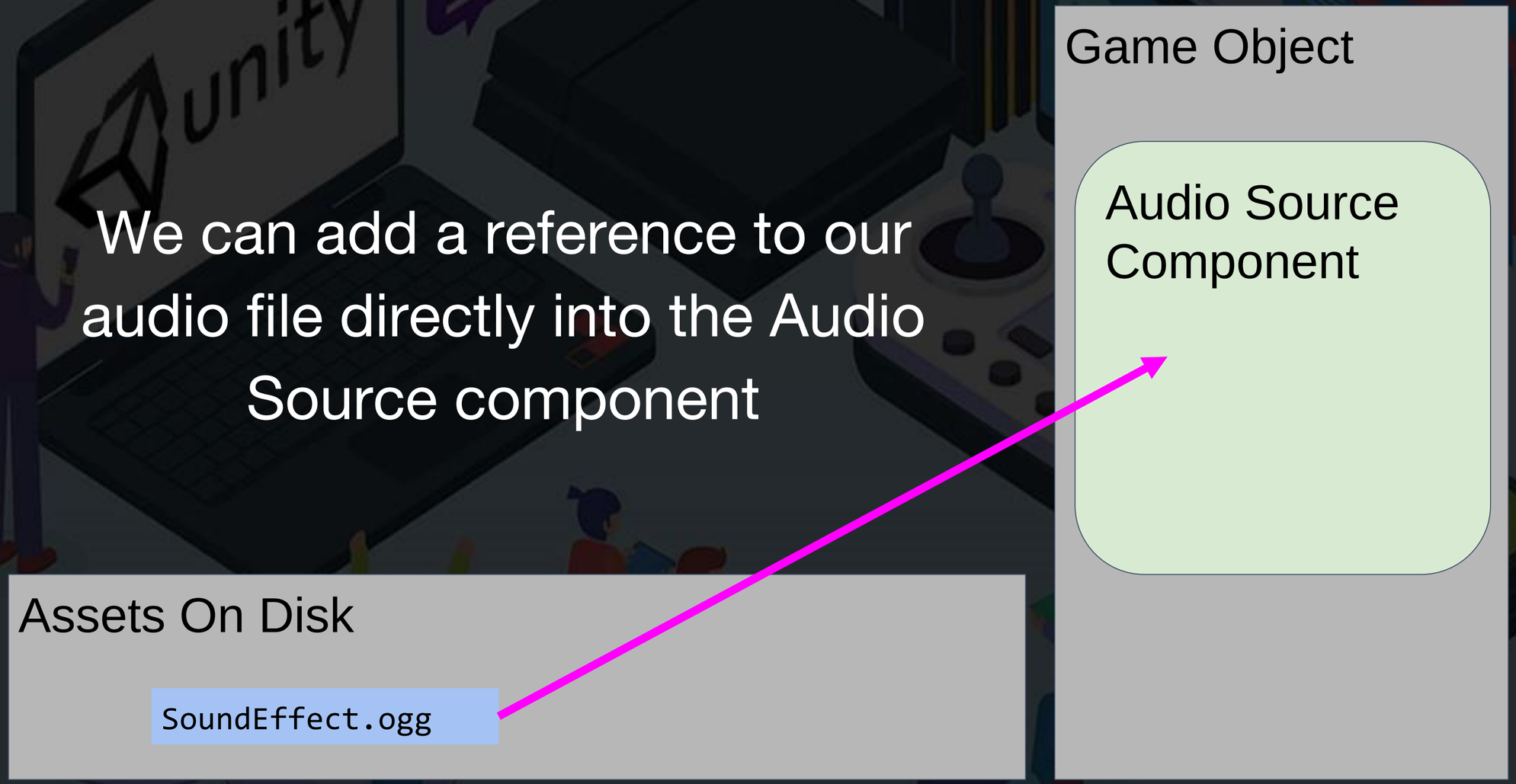
The “sounds” that get played

Audio File

ay”
dio

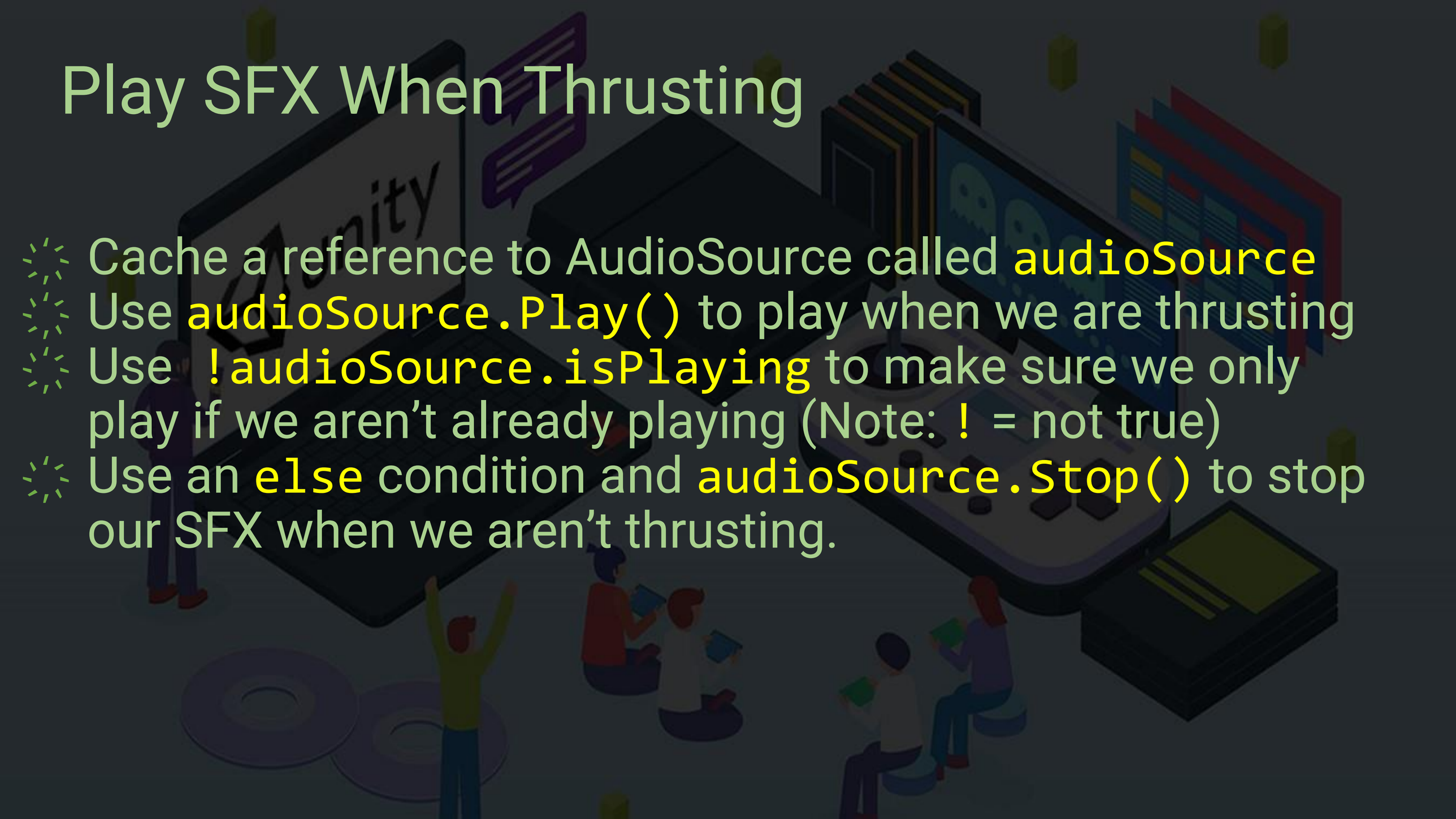
Linking Components To Assets

We can add a reference to our audio file directly into the Audio Source component



Play AudioSource SFX

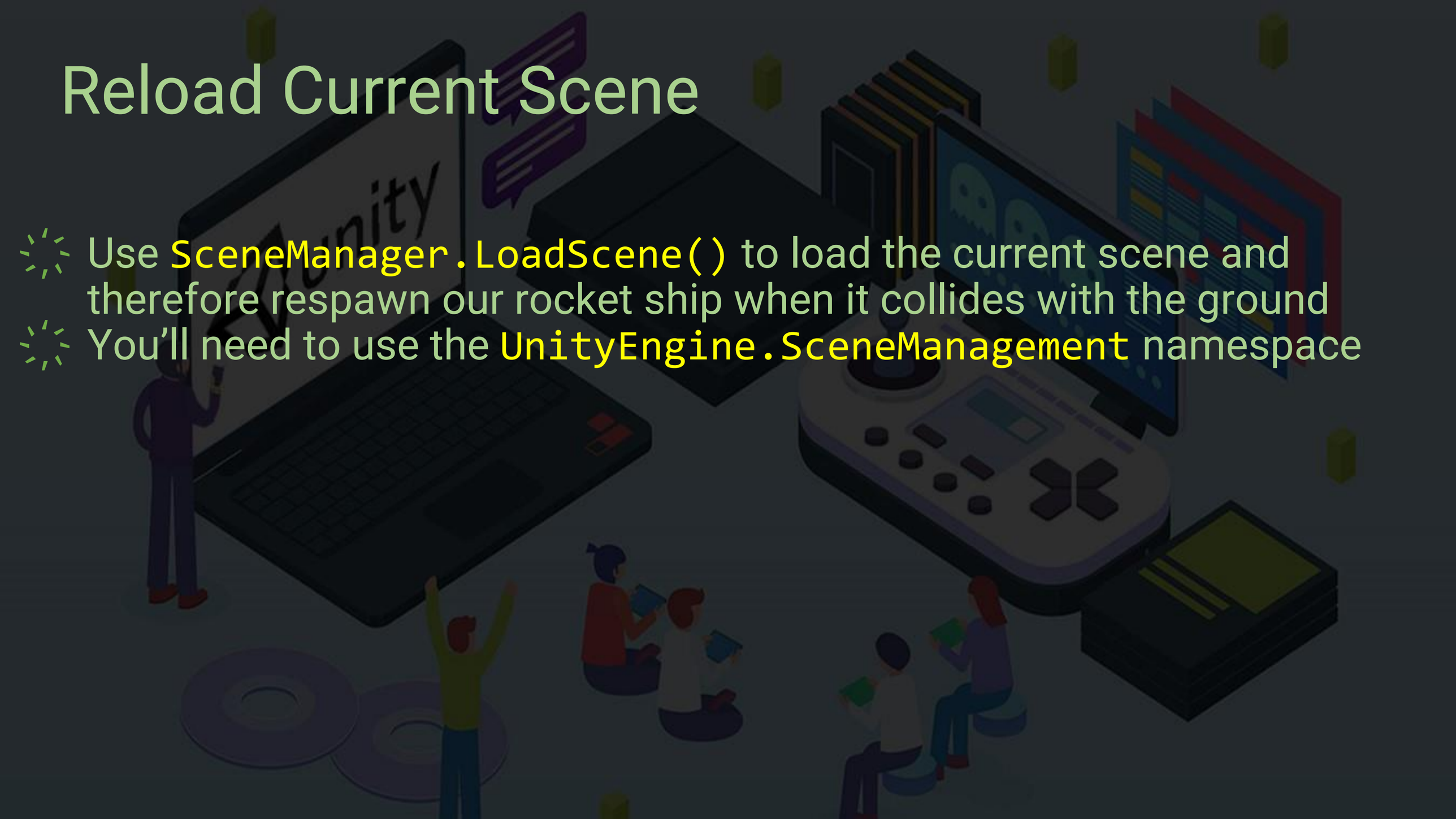
Play SFX When Thrusting



- Cache a reference to AudioSource called `audioSource`
- Use `audioSource.Play()` to play when we are thrusting
- Use `!audioSource.isPlaying` to make sure we only play if we aren't already playing (Note: `!` = not true)
- Use an `else` condition and `audioSource.Stop()` to stop our SFX when we aren't thrusting.

Respawn Using SceneManager

Reload Current Scene



- Use `SceneManager.LoadScene()` to load the current scene and therefore respawn our rocket ship when it collides with the ground
- You'll need to use the `UnityEngine.SceneManagement` namespace

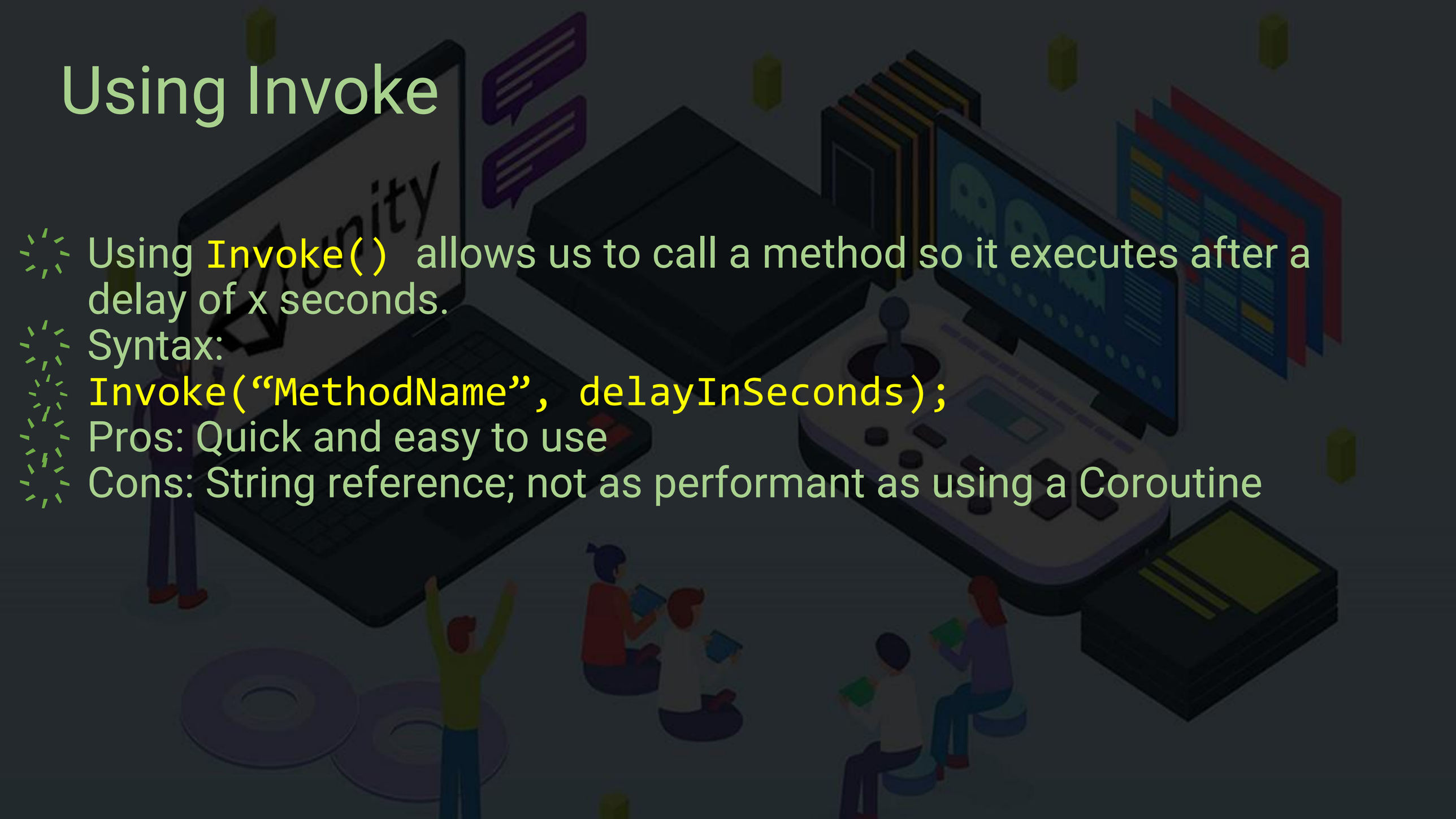
Load Next Level

Load Next Level

When we reach the Landing Pad, load the next level.
Hint: our scene index is an integer, therefore we can add a number to it.

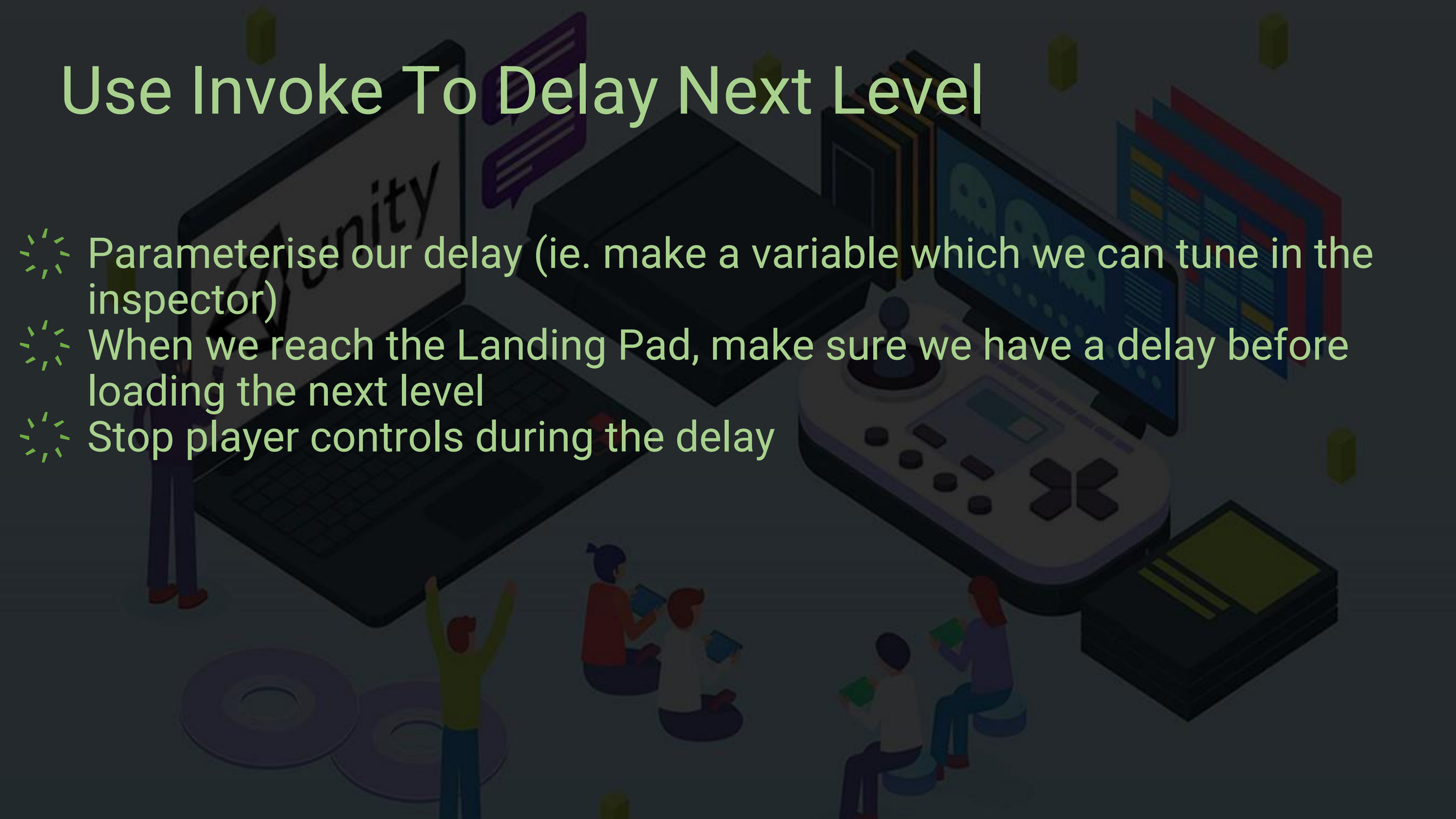
Using Invoke

Using Invoke



- Using `Invoke()` allows us to call a method so it executes after a delay of x seconds.
- Syntax:
- `Invoke(“MethodName”, delayInSeconds);`
- Pros: Quick and easy to use
- Cons: String reference; not as performant as using a Coroutine

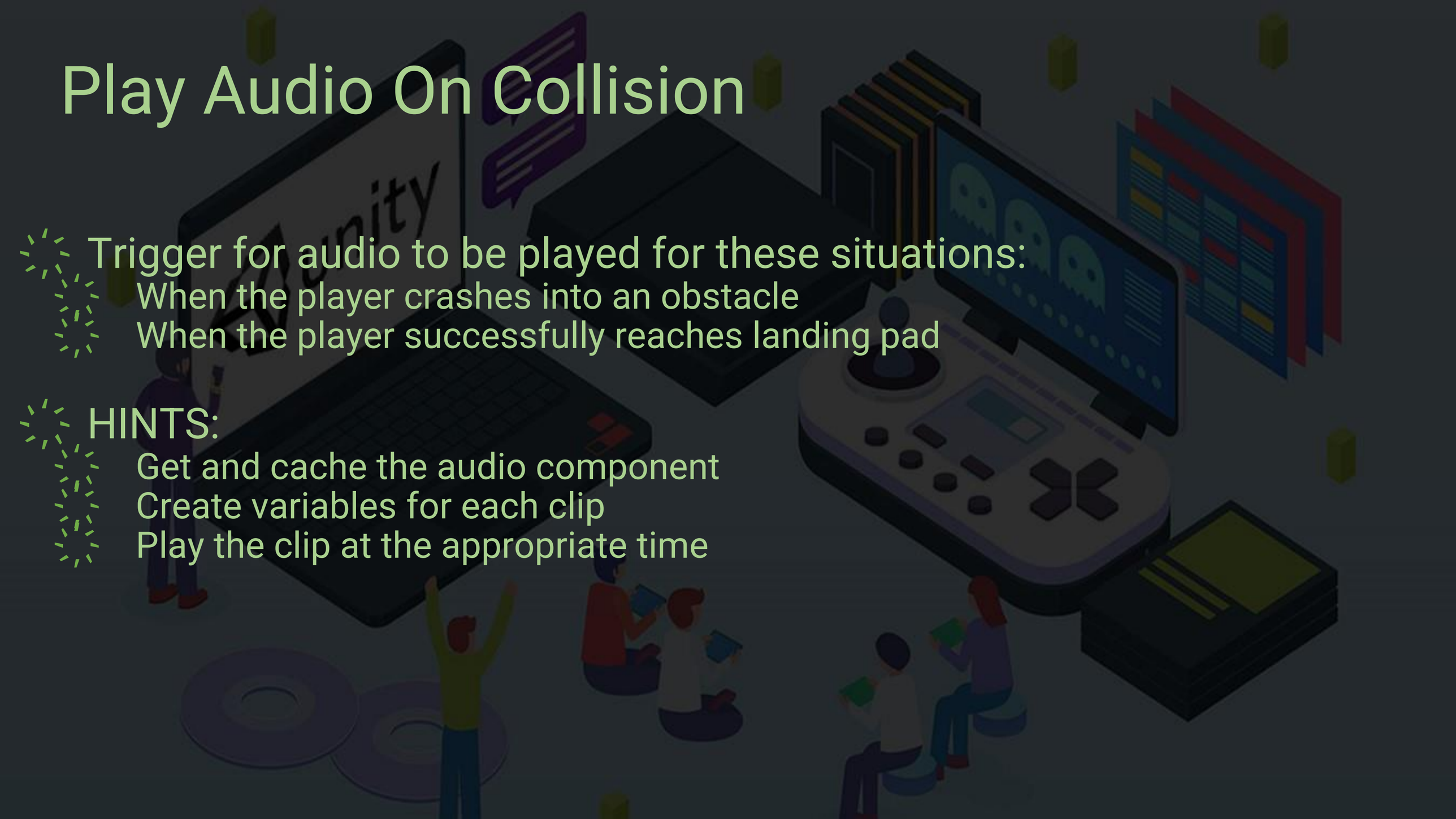
Use Invoke To Delay Next Level



- Parameterise our delay (ie. make a variable which we can tune in the inspector)
- When we reach the Landing Pad, make sure we have a delay before loading the next level
- Stop player controls during the delay

Multiple Audio Clips

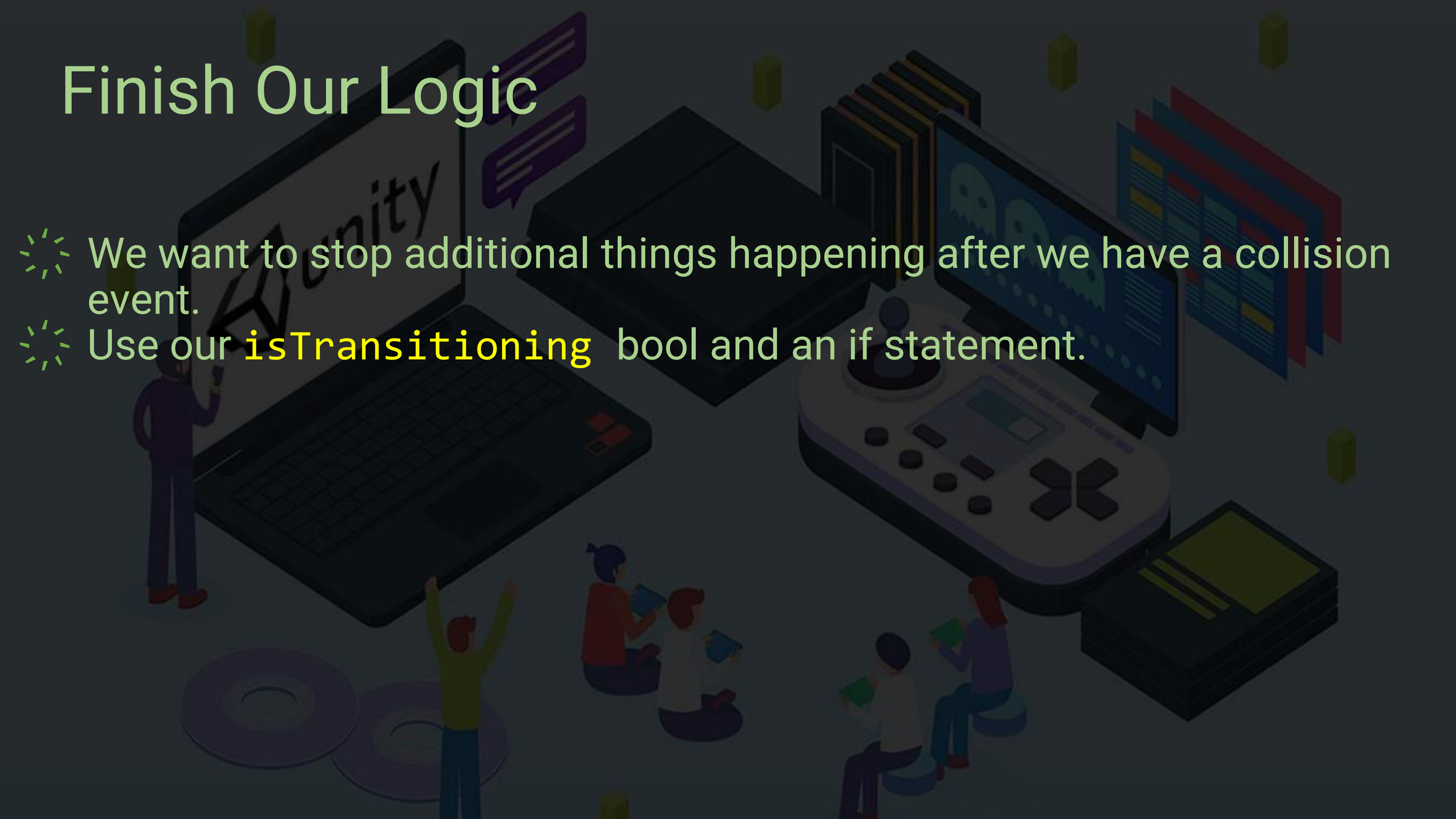
Play Audio On Collision



- Trigger for audio to be played for these situations:
  - When the player crashes into an obstacle
  - When the player successfully reaches landing pad
- HINTS:
  - Get and cache the audio component
  - Create variables for each clip
  - Play the clip at the appropriate time

Bool Variable For State

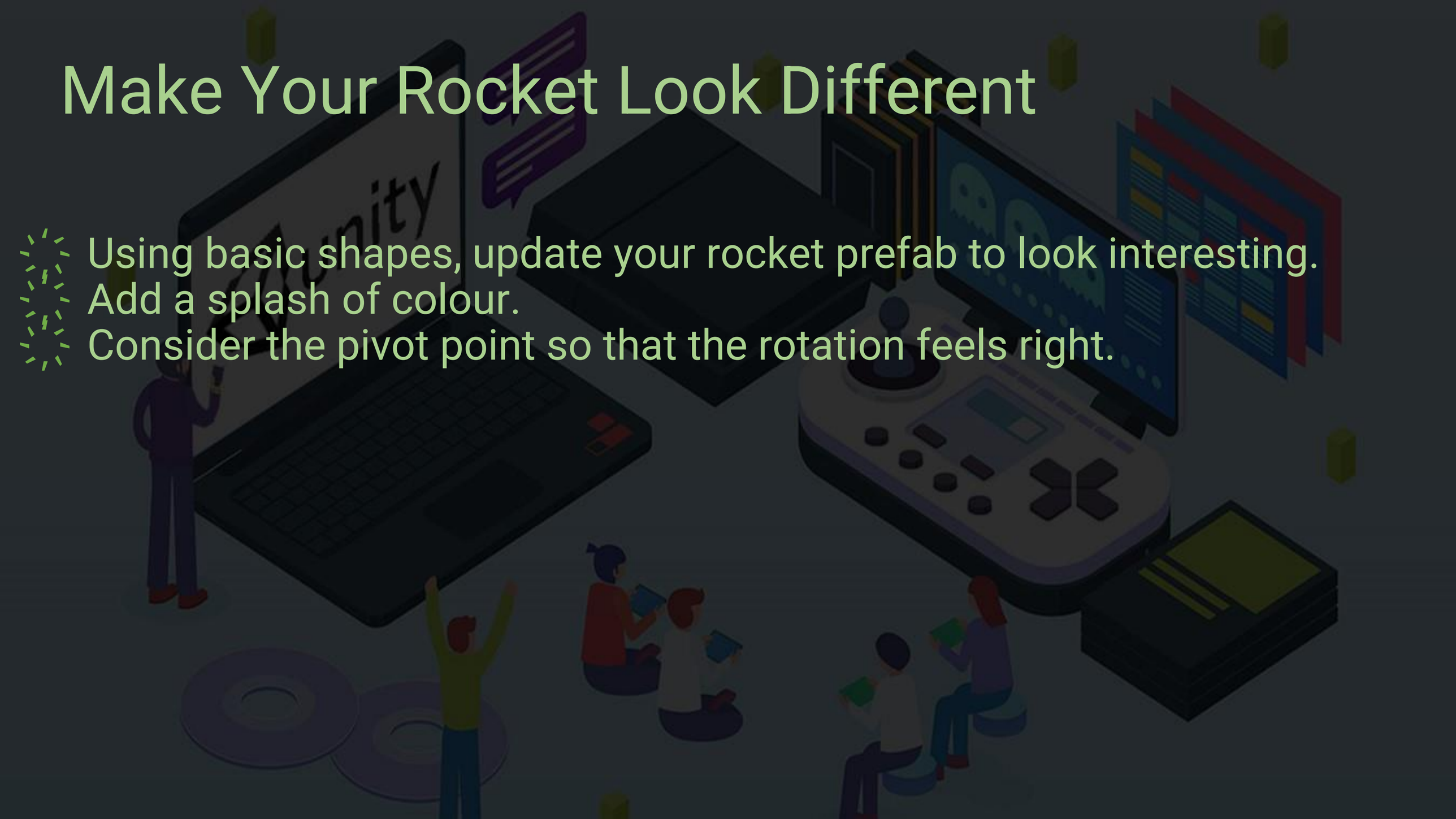
Finish Our Logic



- We want to stop additional things happening after we have a collision event.
- Use our `isTransitioning` bool and an if statement.

Make Rocket Look Spiffy

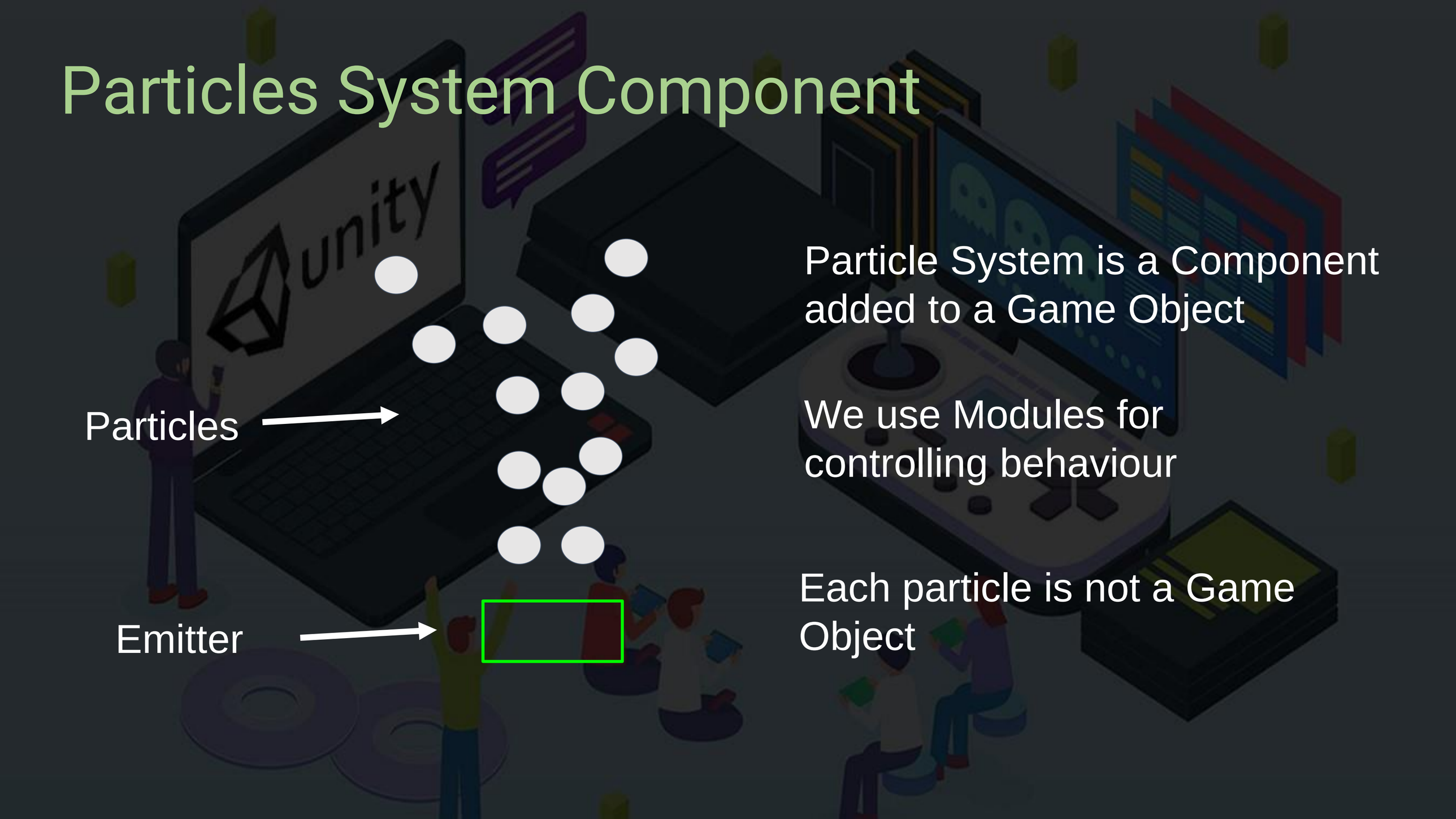
Make Your Rocket Look Different



- Using basic shapes, update your rocket prefab to look interesting.
- Add a splash of colour.
- Consider the pivot point so that the rotation feels right.

How To Trigger Particles

Particles System Component

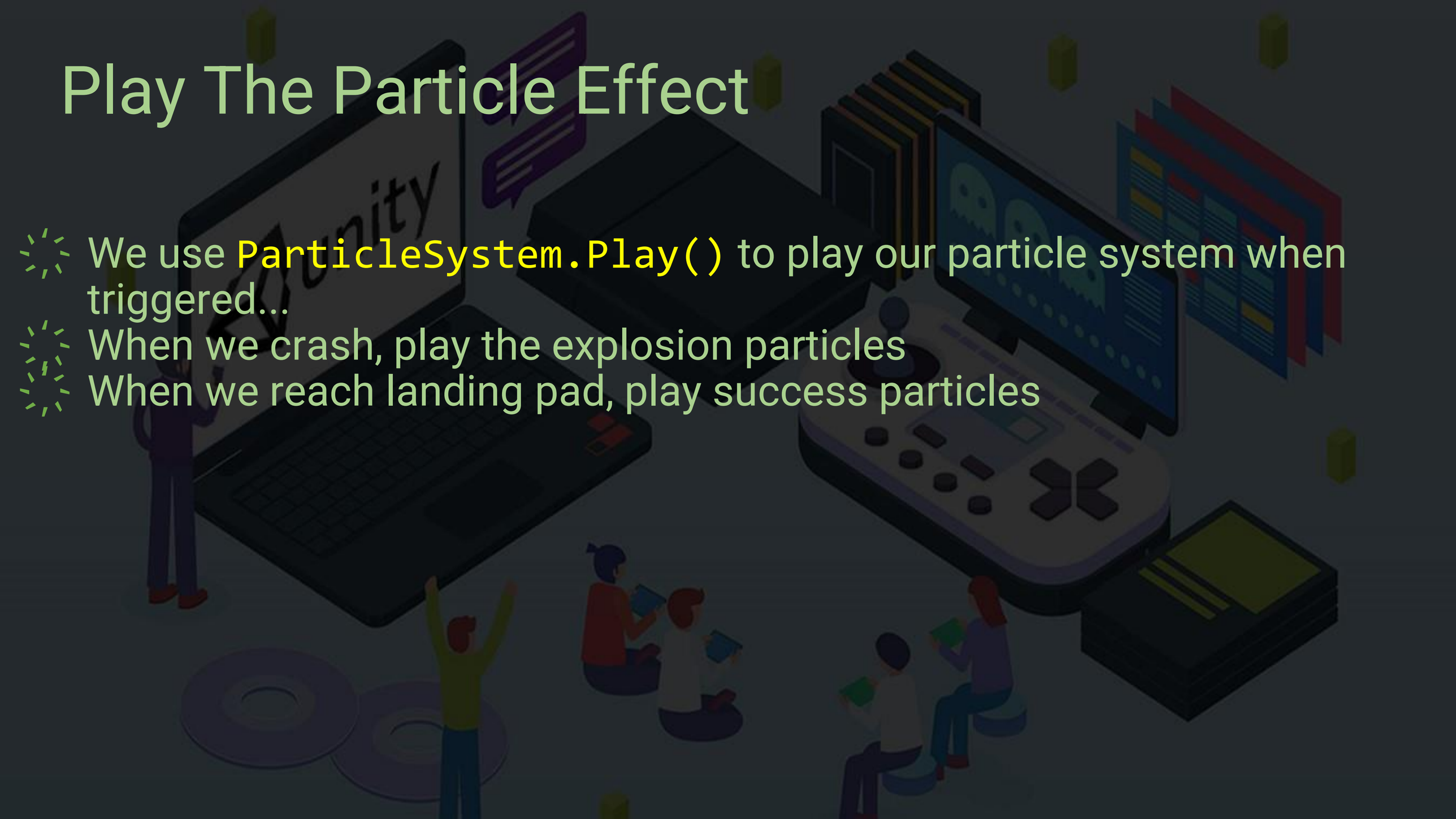


Particle System is a Component added to a Game Object

We use Modules for controlling behaviour

Each particle is not a Game Object

Play The Particle Effect



- We use `ParticleSystem.Play()` to play our particle system when triggered...
- When we crash, play the explosion particles
- When we reach landing pad, play success particles

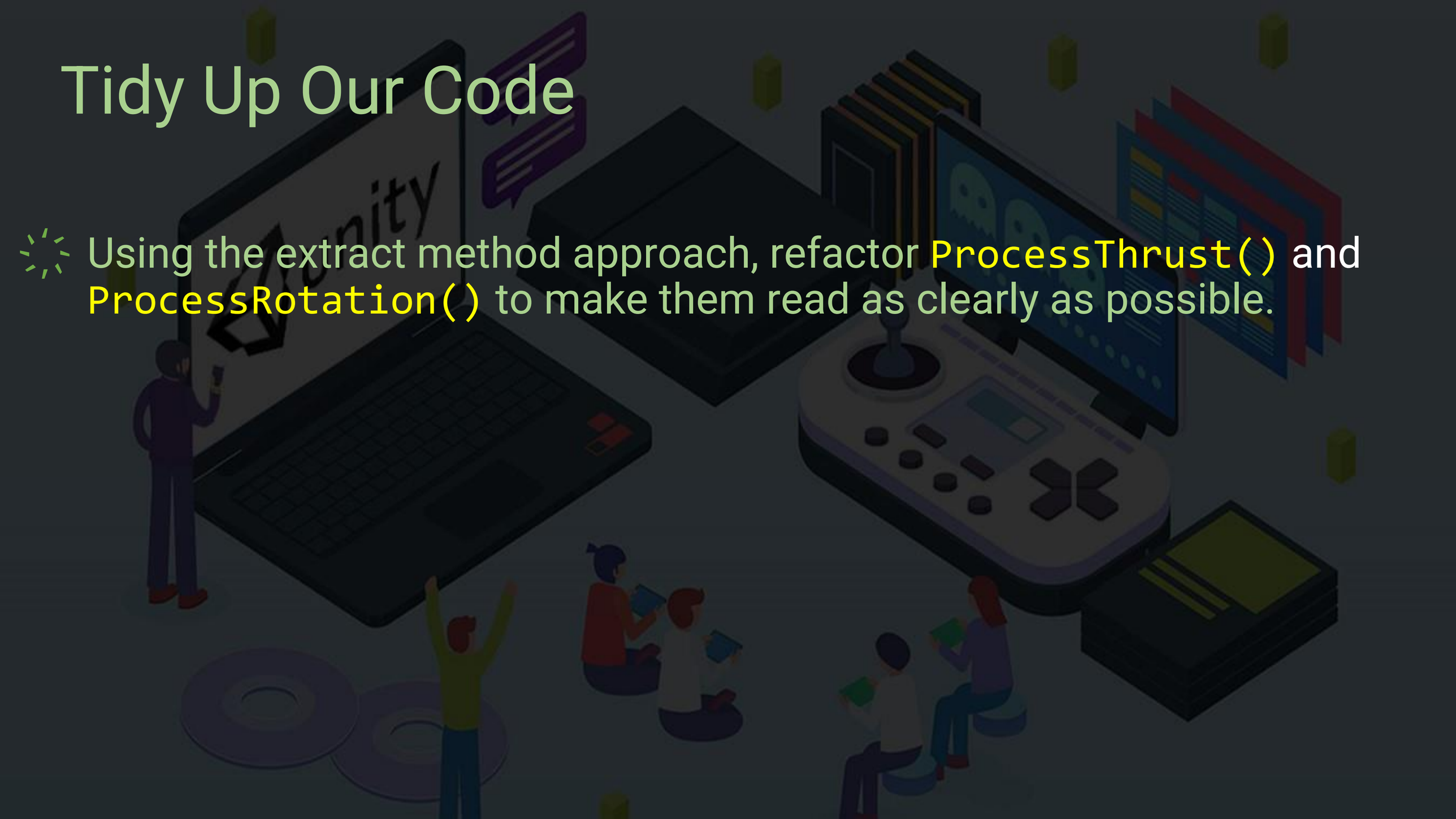
Particles For Rocket Boosters

Get Ourselves Ready To Trigger

- Set up your rocket so that it has 3 particle systems, ready for us to trigger in code:
  - Main booster
  - Left booster
  - Right booster
- Add them in your prefab and create variables for each

Refactor With Extract Method

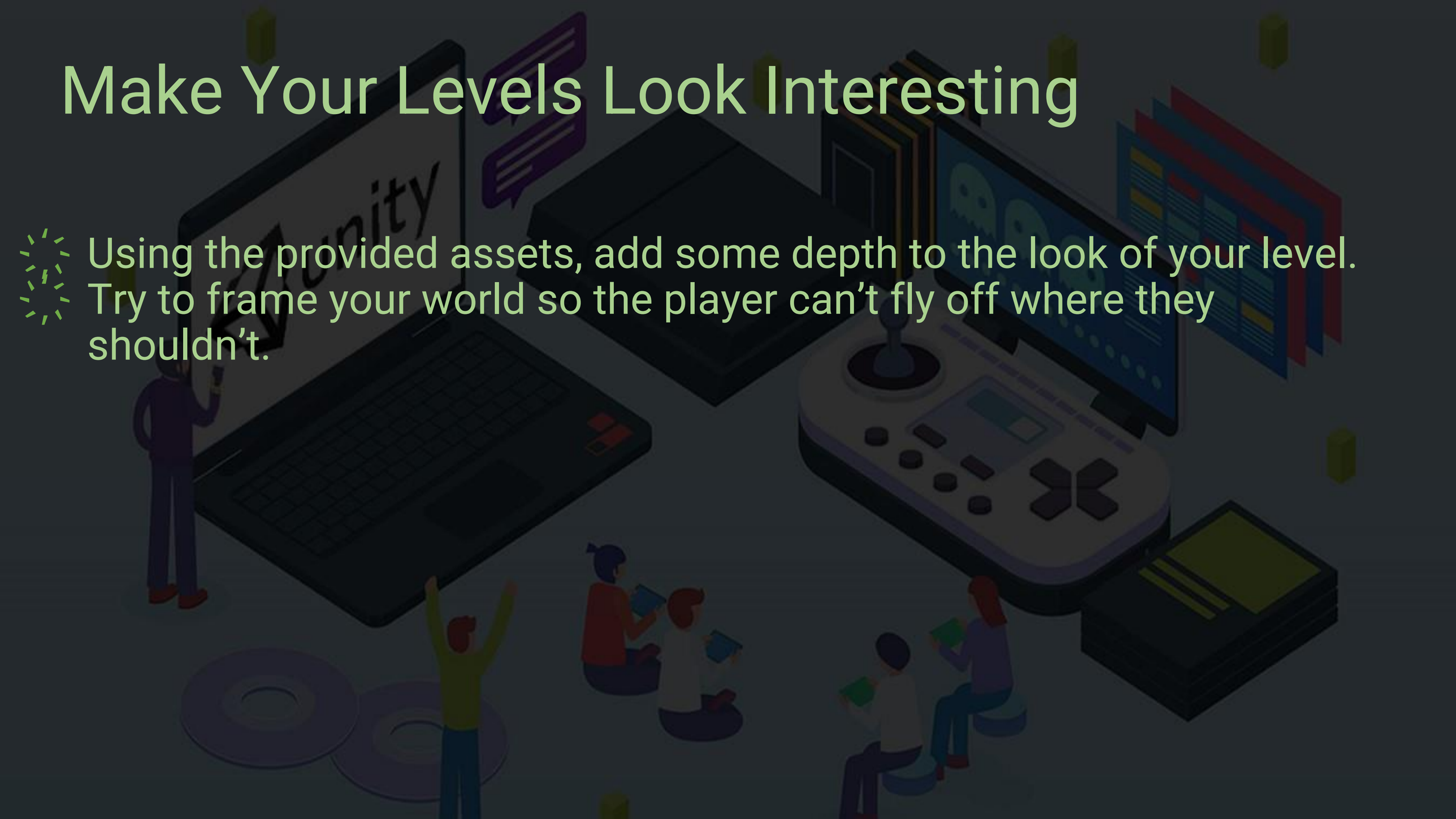
Tidy Up Our Code



- Using the extract method approach, refactor `ProcessThrust()` and `ProcessRotation()` to make them read as clearly as possible.

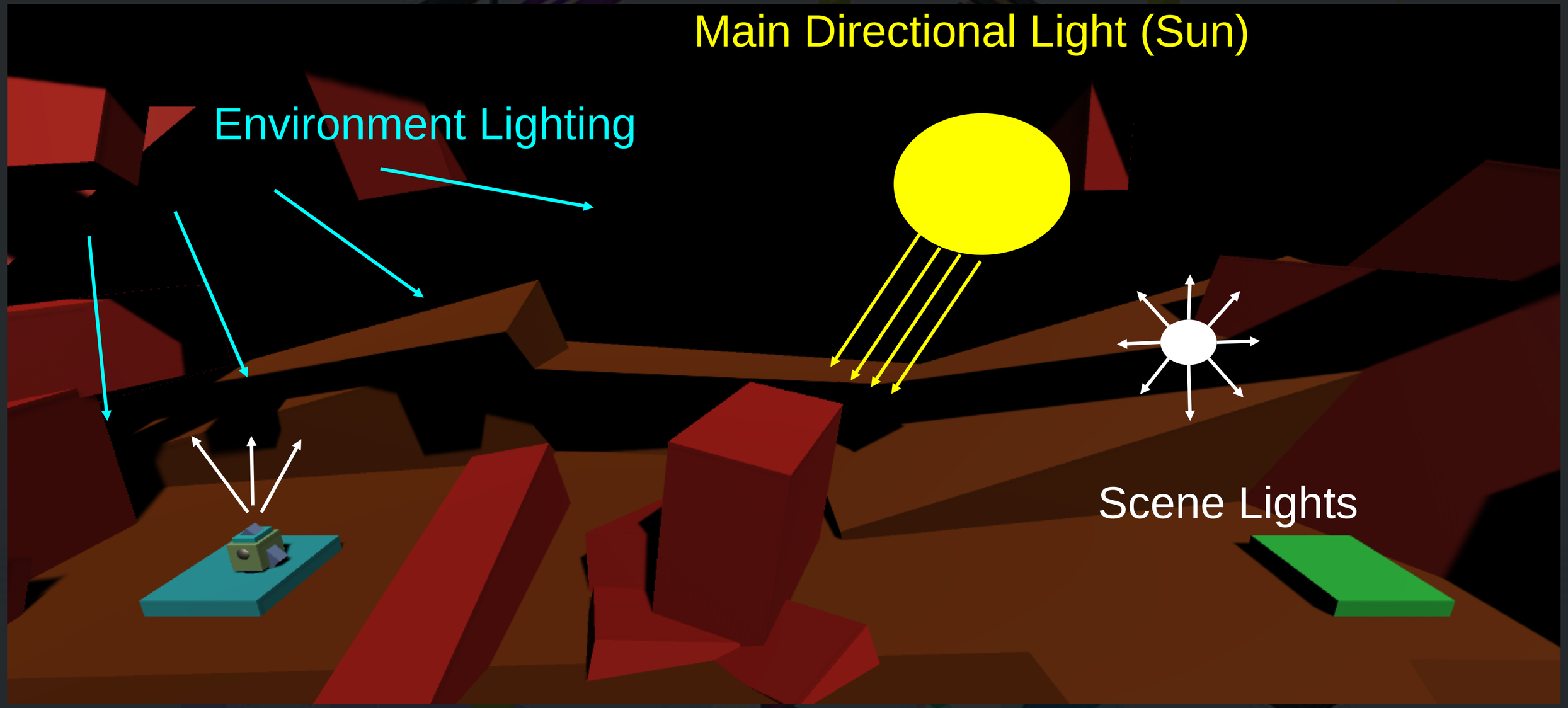
Make Environment

Make Your Levels Look Interesting

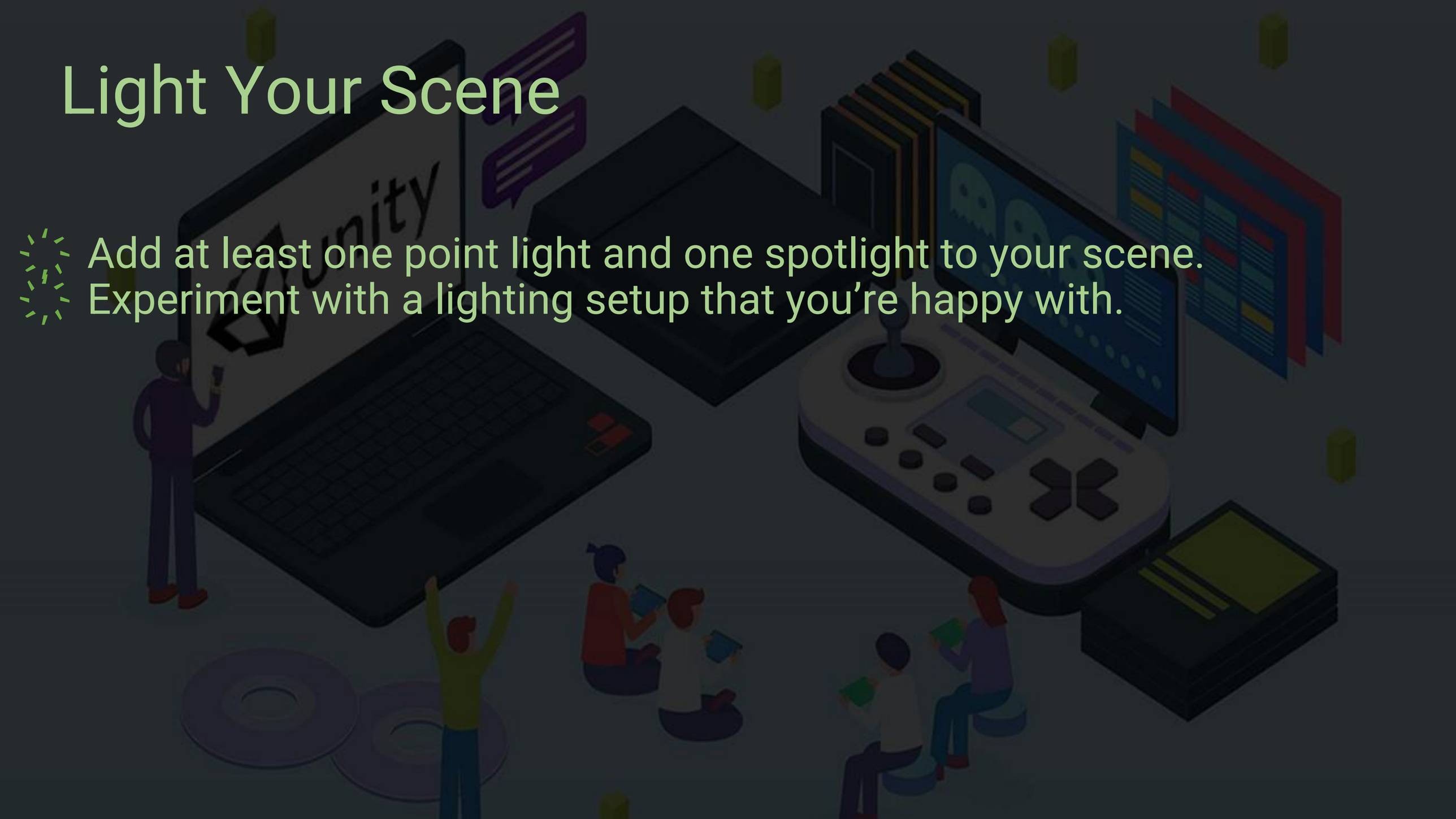


- Using the provided assets, add some depth to the look of your level. Try to frame your world so the player can’t fly off where they shouldn’t.

How To Add Lights In Unity


Main Directional Light (Sun)
Environment Lighting
Scene Lights

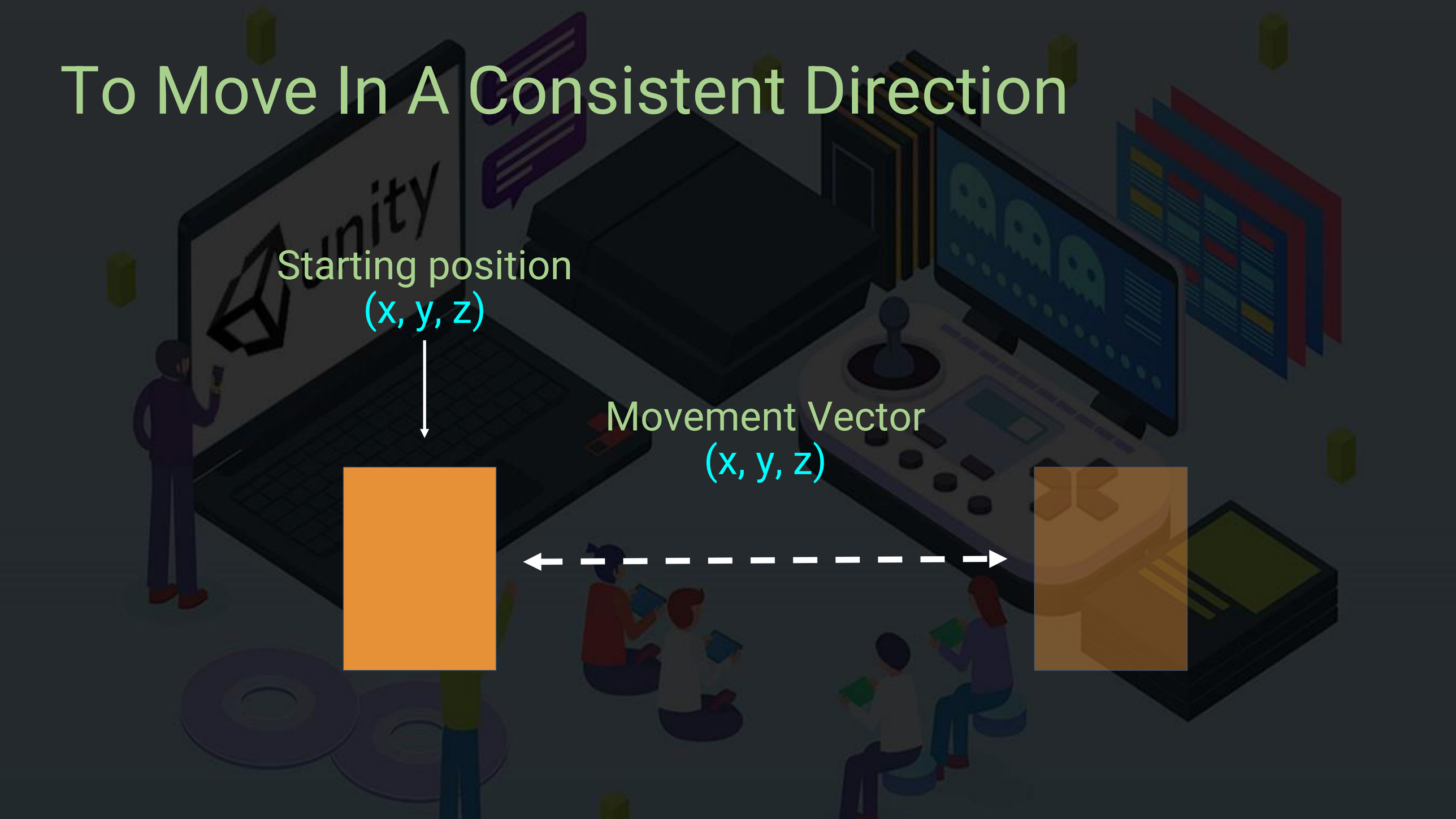
Light Your Scene



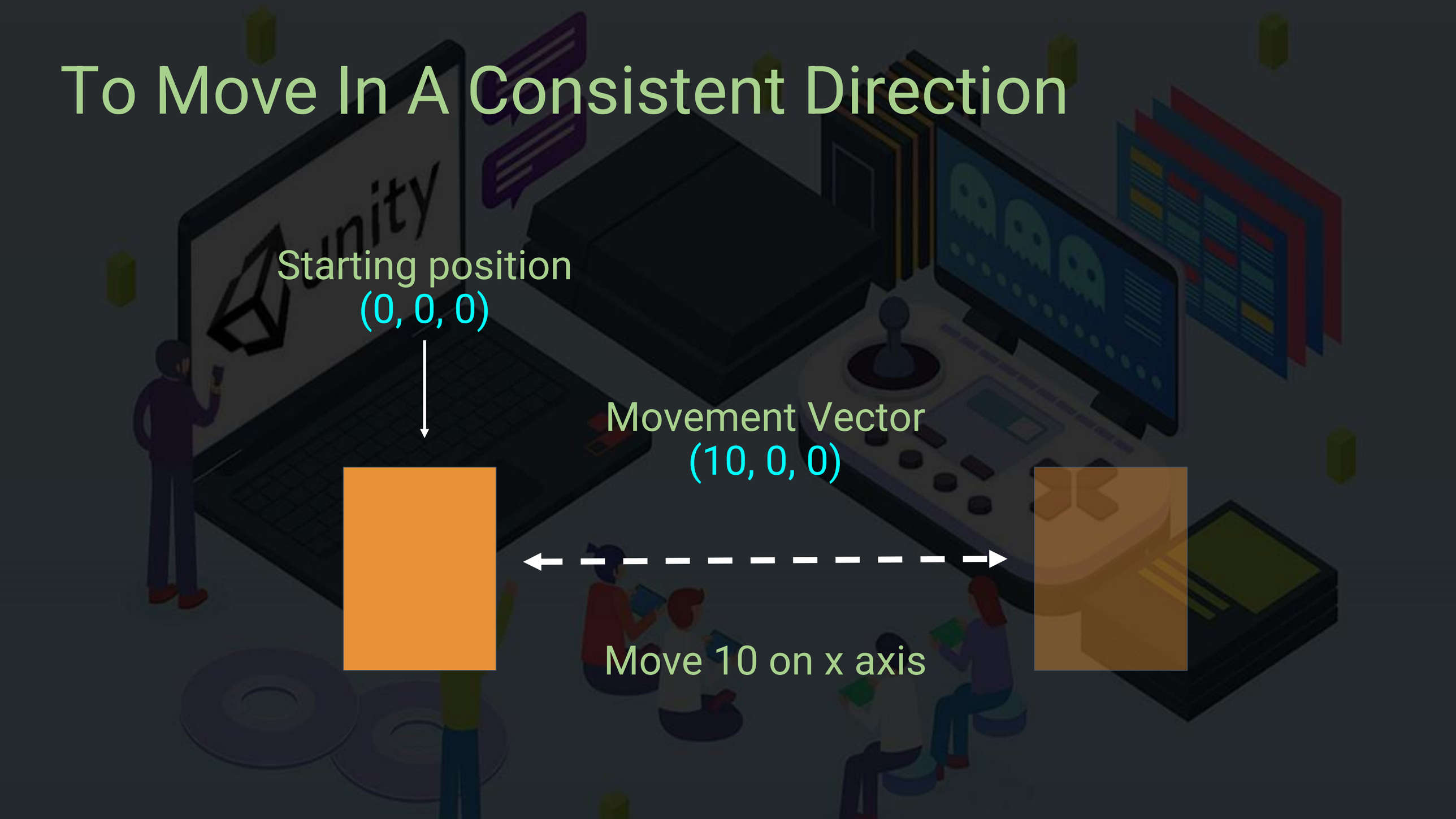
- Add at least one point light and one spotlight to your scene. Experiment with a lighting setup that you’re happy with.

Move Obstacle With Code

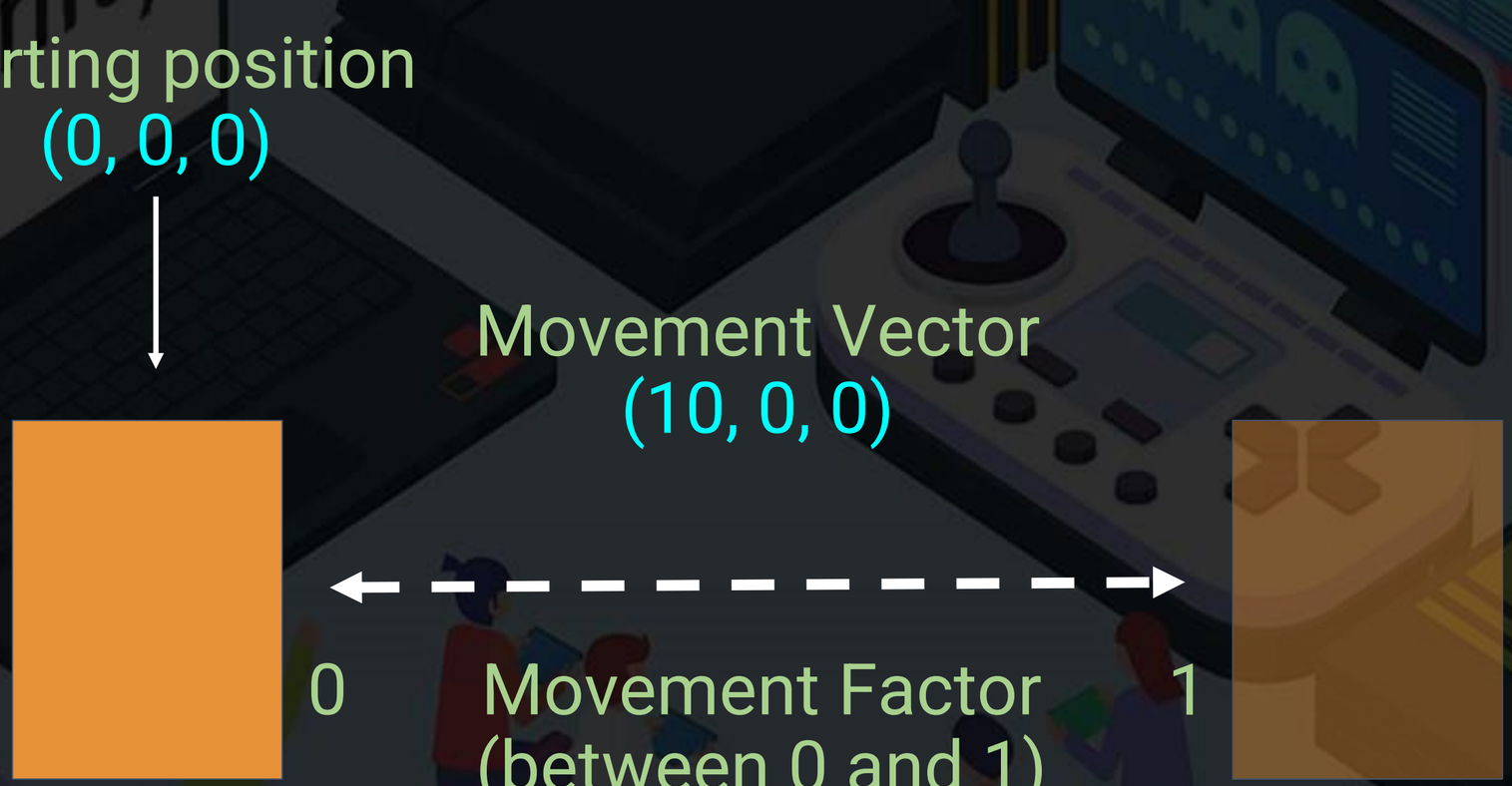
To Move In A Consistent Direction



To Move In A Consistent Direction



To Oscillate (Back & Forth)



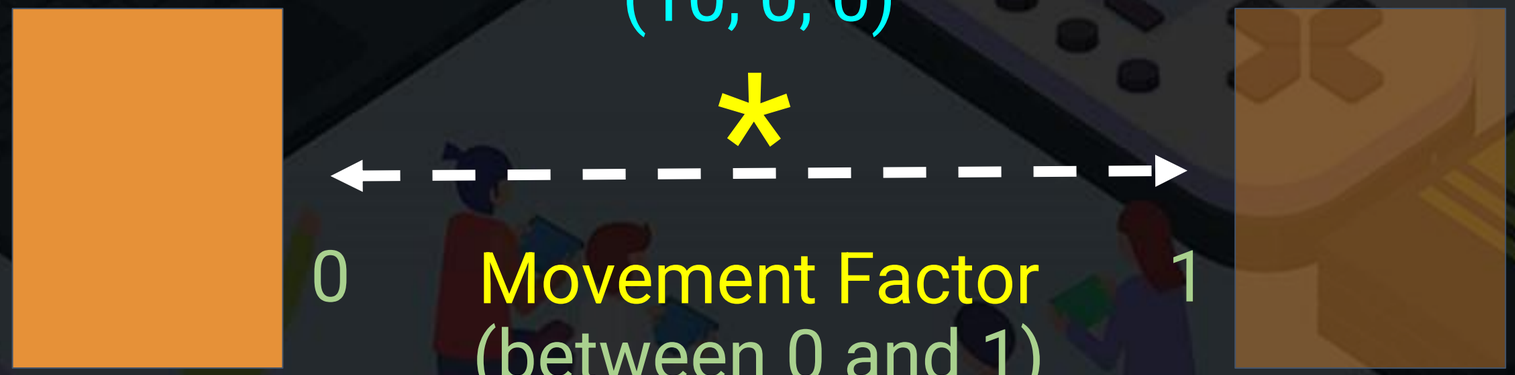
To Oscillate (Back & Forth)

Starting position
(0, 0, 0)

Offset =

Movement Vector
(10, 0, 0)

*



0

Movement Factor
(between 0 and 1)

1

Eg.
Movement Factor
is currently 0.5,
Offset is?...

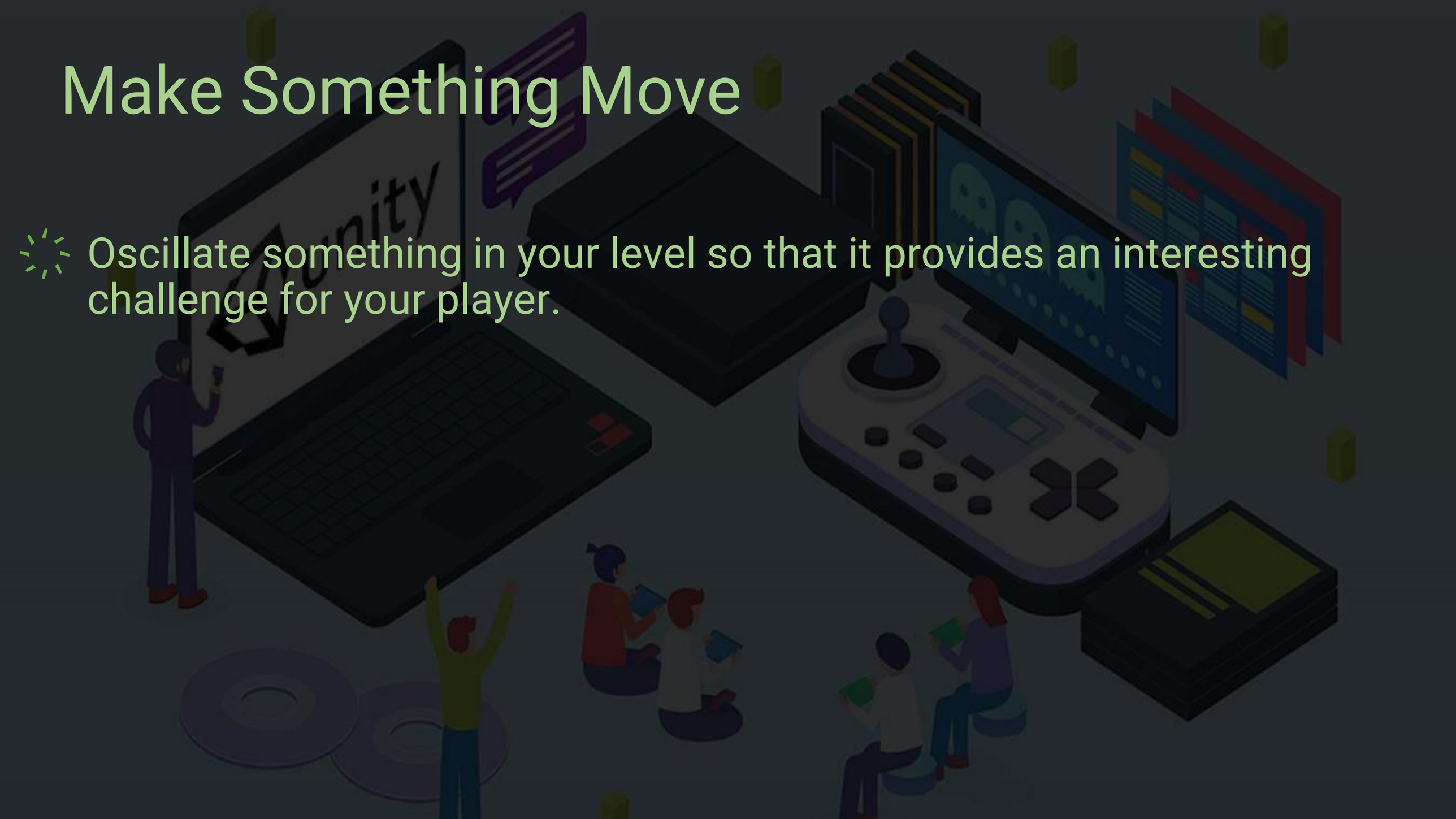
5, 0, 0

Finish The Last Line

- Add one more line that defines the new position of the object that is offset from its original position.

Mathf.Sin() For Oscillation

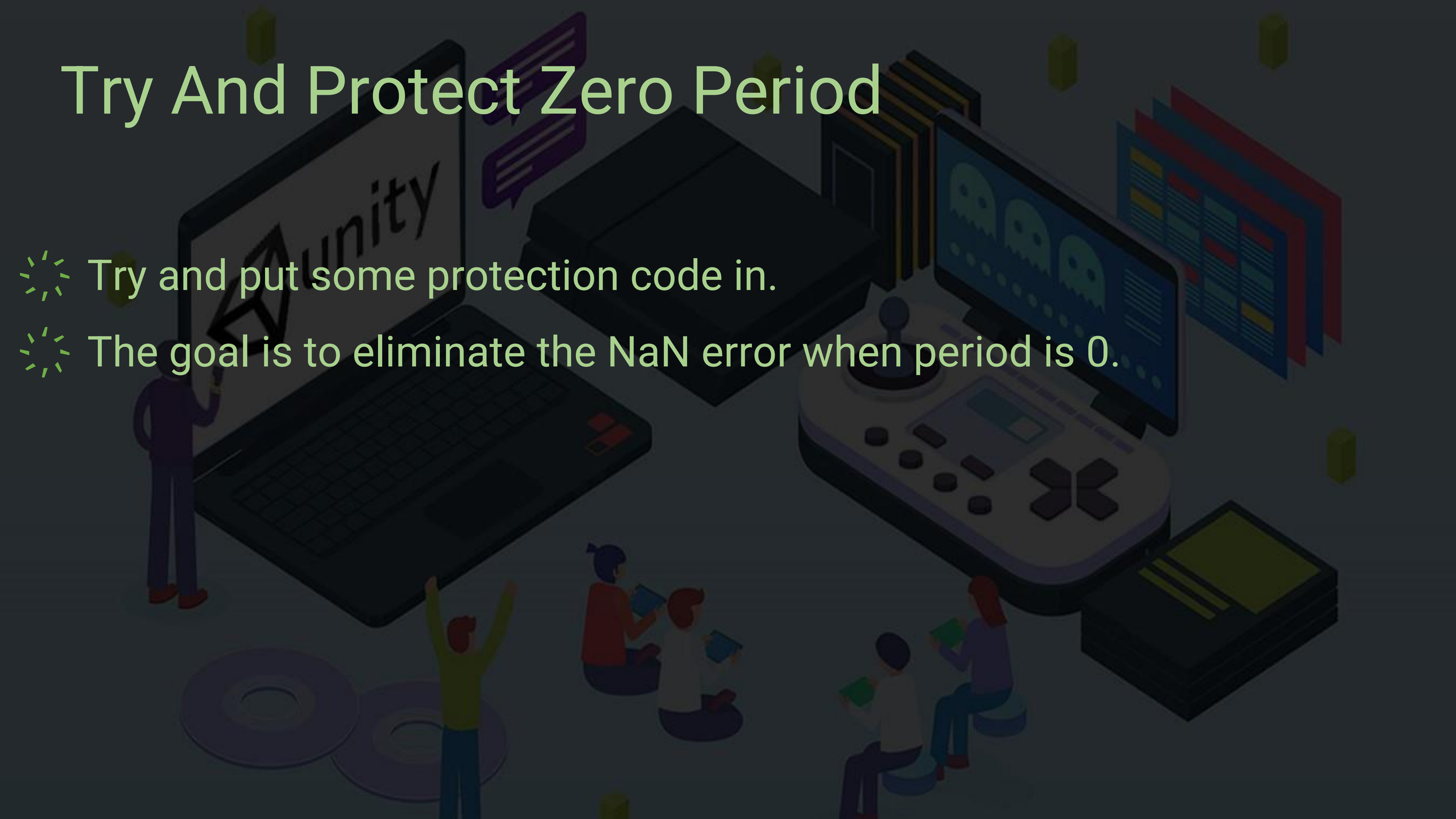
Make Something Move



- Oscillate something in your level so that it provides an interesting challenge for your player.

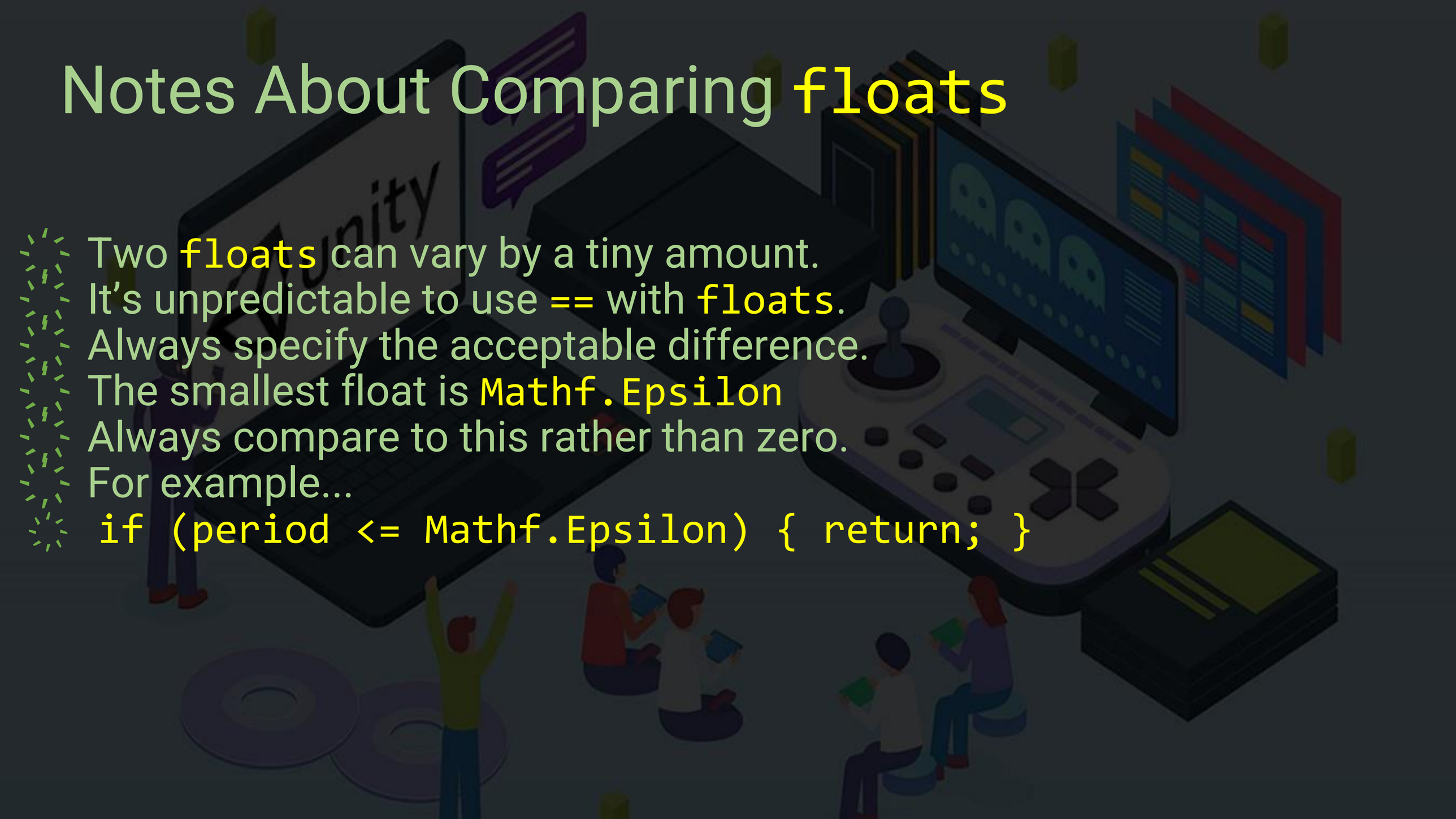
Protect Against NaN Error

Try And Protect Zero Period



- Try and put some protection code in.
- The goal is to eliminate the NaN error when period is 0.

Notes About Comparing `floats`



- Two `floats` can vary by a tiny amount.
- It's unpredictable to use `==` with `floats`.
- Always specify the acceptable difference.
- The smallest float is `Mathf.Epsilon`
- Always compare to this rather than zero.
- For example...

```
if (period <= Mathf.Epsilon) { return; }
```

Designing Level Moments

Useful Game Design Approach

- Design “moments” and then expand them into a level. Moments that use the environment:
  - Fly under
  - Fly over
  - Fly through a gap
  - Time your flight through moving obstacle
  - Land on moving platform
  - Fly through narrow tunnel

Useful Game Design Approach

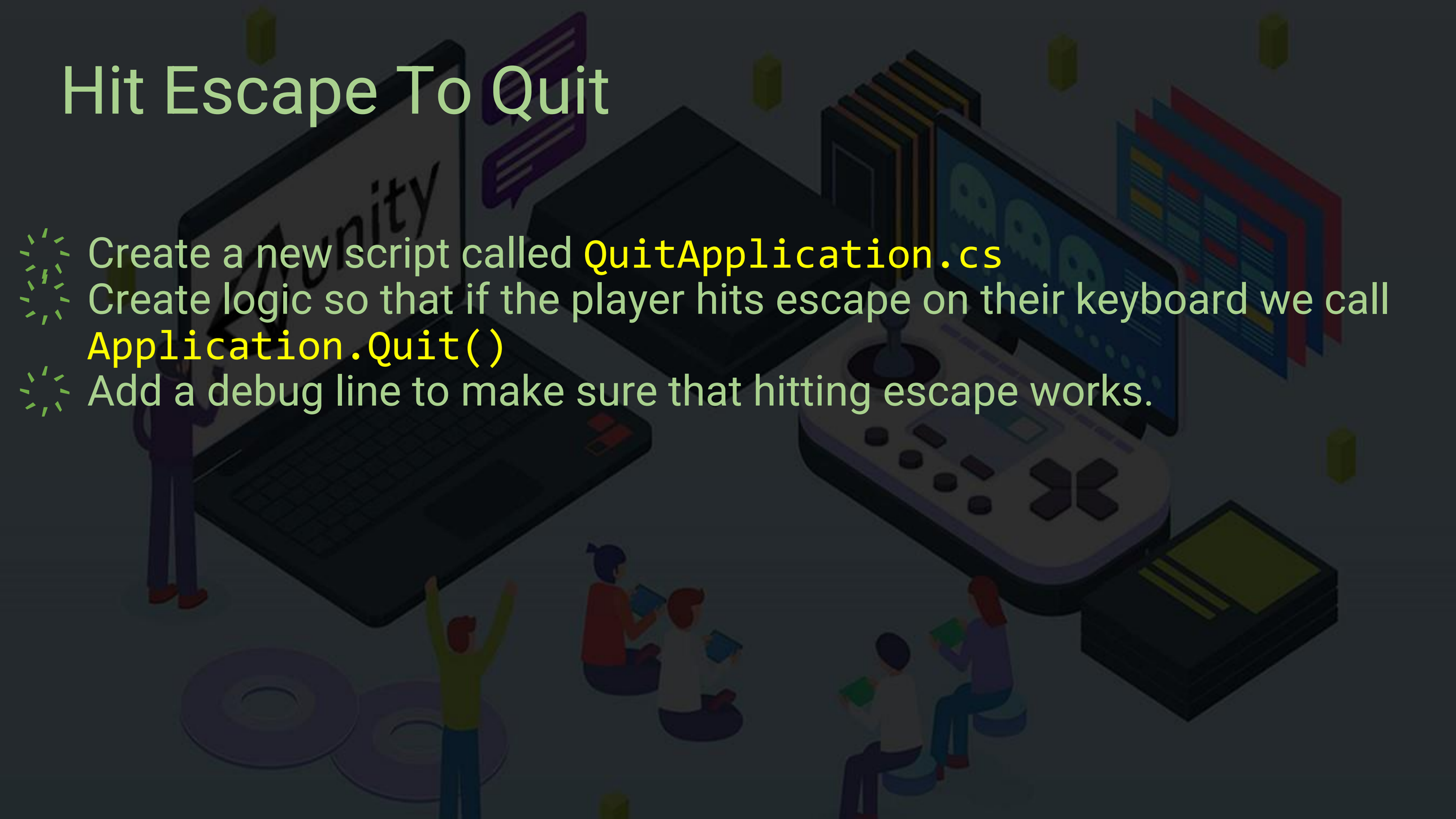
- Moments that use tuning of our existing game:
  - Slower rocket (eg. it got damaged)
  - Faster rocket (eg. got a boost)
  - Darker level
  - Closer camera
  - Bigger rocket (carrying something)
  - Reversed controls
  - And so on...

Level Design Challenge (If Interested)

- Create 5 (or more) different levels, each with a unique game moment
- Add the levels to the build settings so they are playable
- Share a video with us of your one best moment (OBS is a free recording package)

Quit Application

Hit Escape To Quit



- Create a new script called `QuitApplication.cs`
- Create logic so that if the player hits escape on their keyboard we call `Application.Quit()`
- Add a debug line to make sure that hitting escape works.

How To Build & Publish A Game

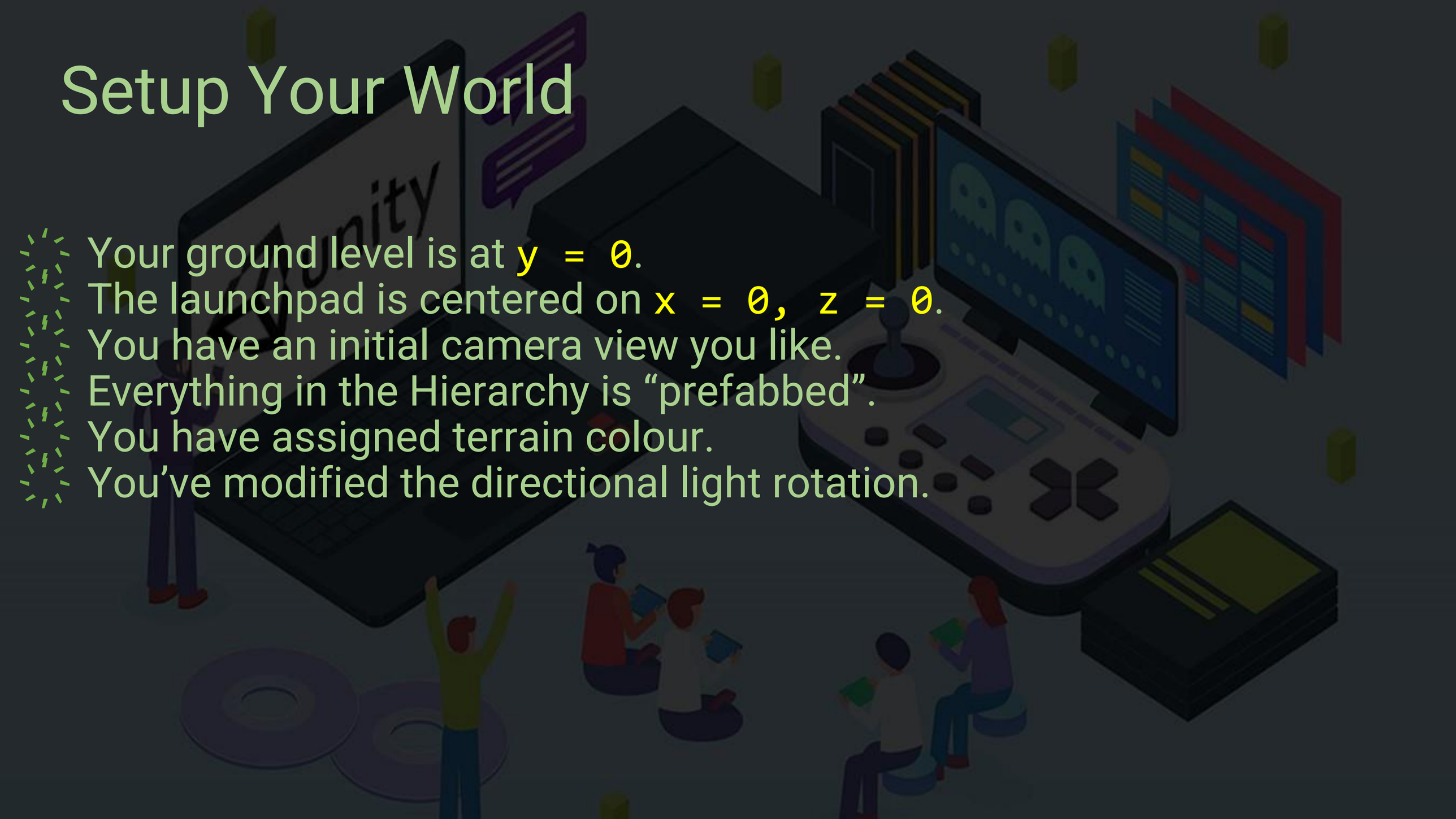
Tips & Tricks

The Origin Of Our World

Which Axis Is Which...

+x = right
+y = up
+z = forward

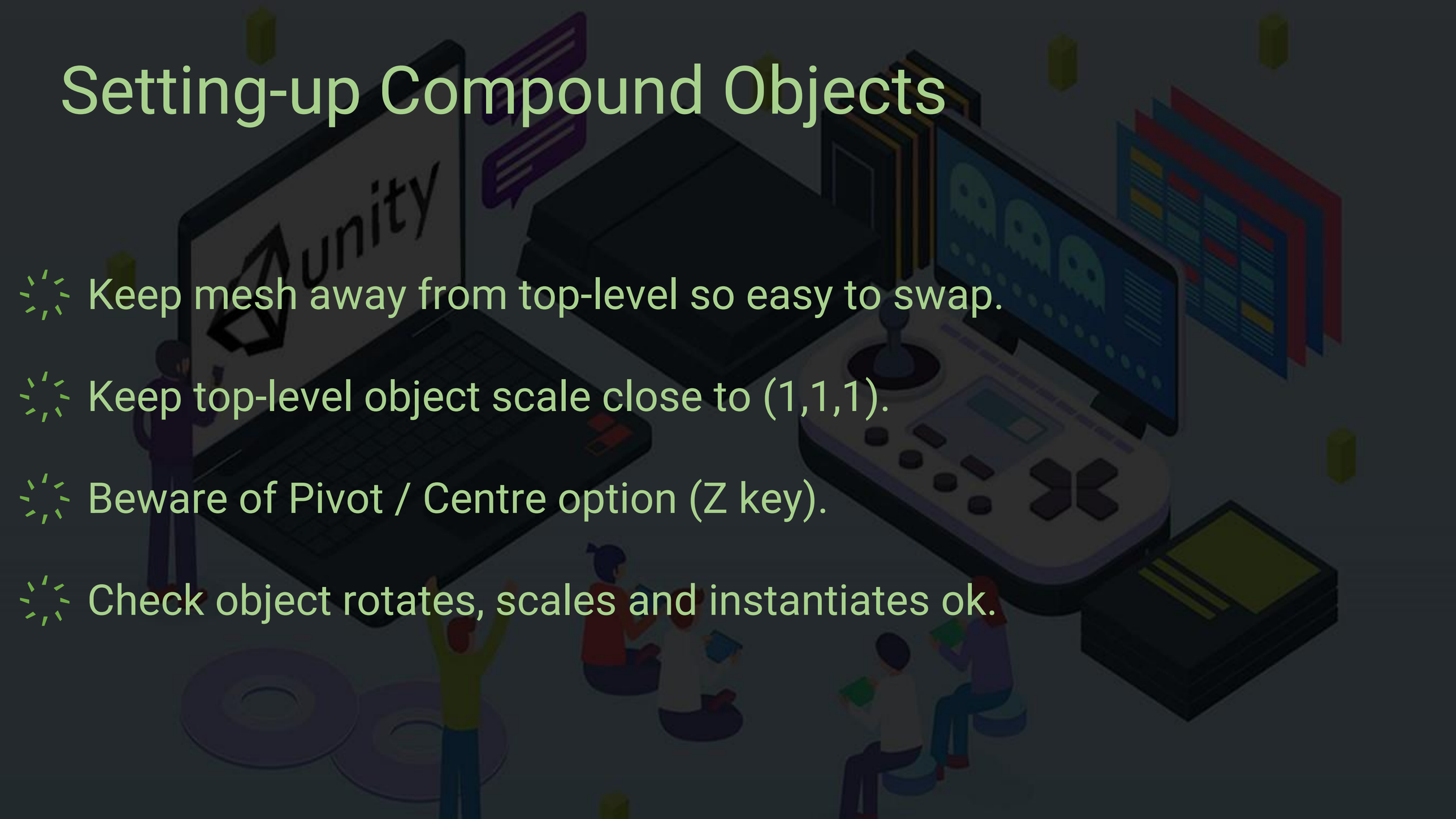
Setup Your World



- Your ground level is at `y = 0`.
- The launchpad is centered on `x = 0, z = 0`.
- You have an initial camera view you like.
- Everything in the Hierarchy is “prefabbed”.
- You have assigned terrain colour.
- You’ve modified the directional light rotation.

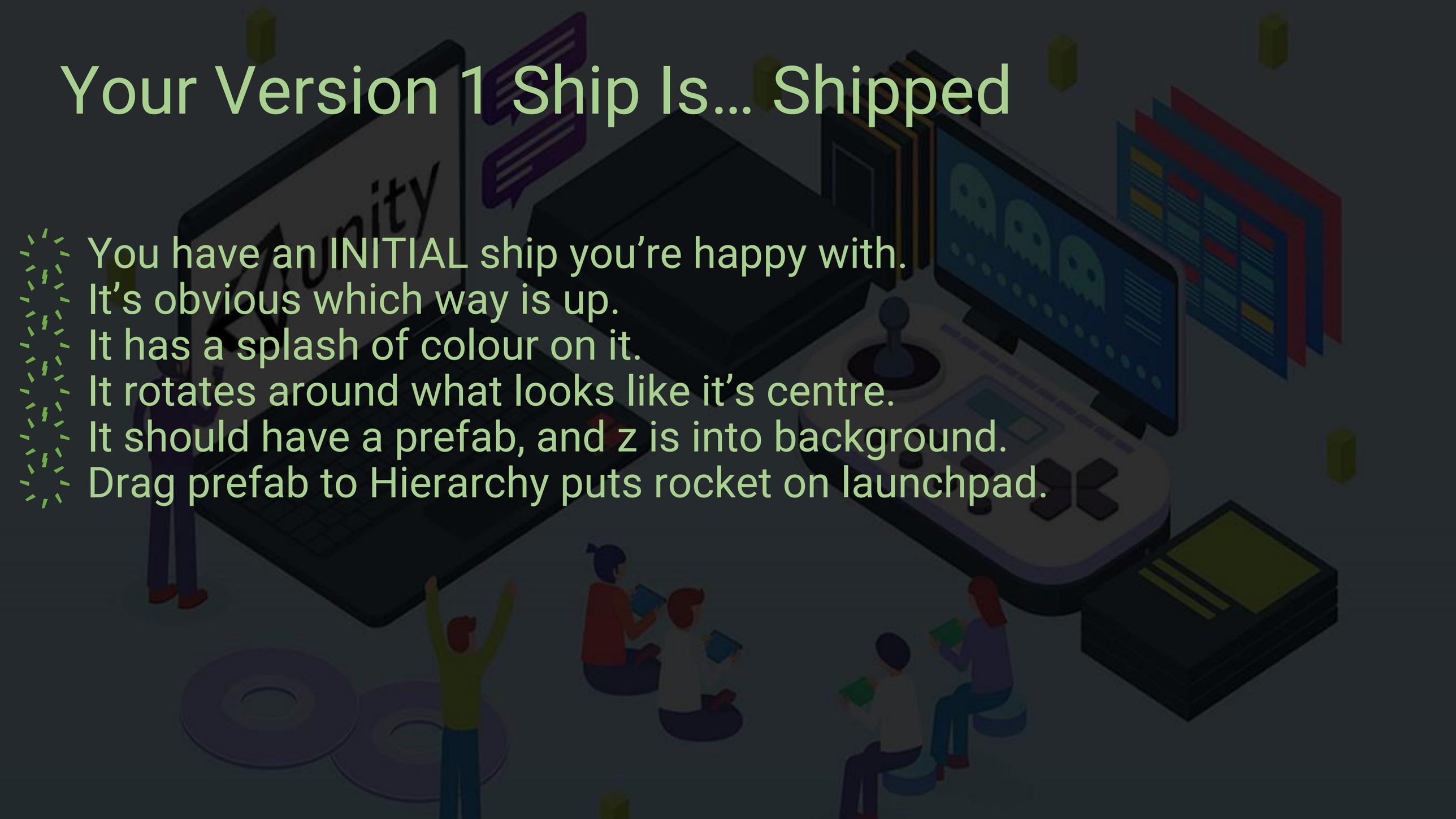
Placeholder Art From Primitives

Setting-up Compound Objects



- Keep mesh away from top-level so easy to swap.
- Keep top-level object scale close to (1,1,1).
- Beware of Pivot / Centre option (Z key).
- Check object rotates, scales and instantiates ok.

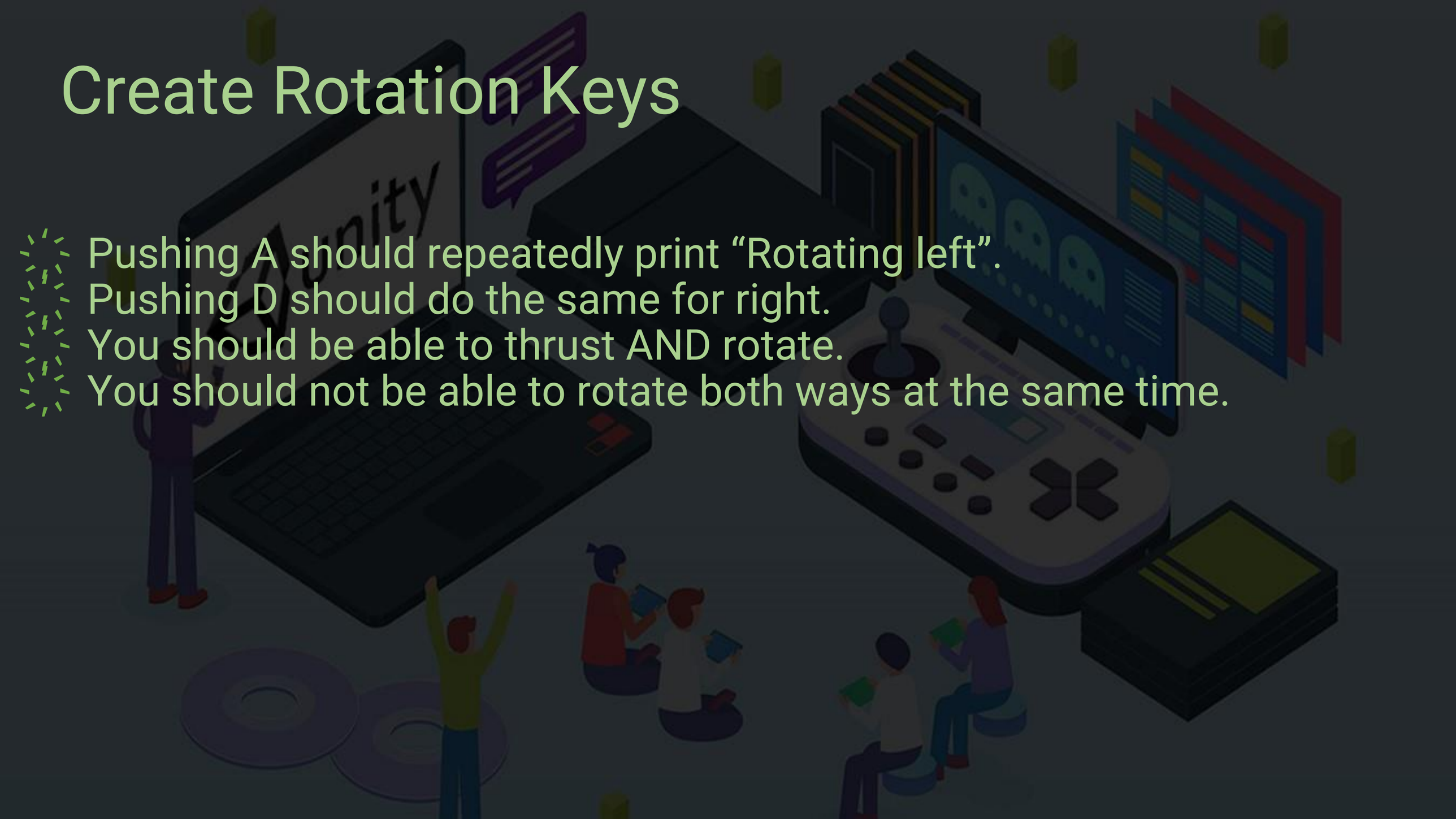
Your Version 1 Ship Is… Shipped



- You have an INITIAL ship you’re happy with.
- It’s obvious which way is up.
- It has a splash of colour on it.
- It rotates around what looks like it’s centre.
- It should have a prefab, and z is into background.
- Drag prefab to Hierarchy puts rocket on launchpad.

Basic Input Binding

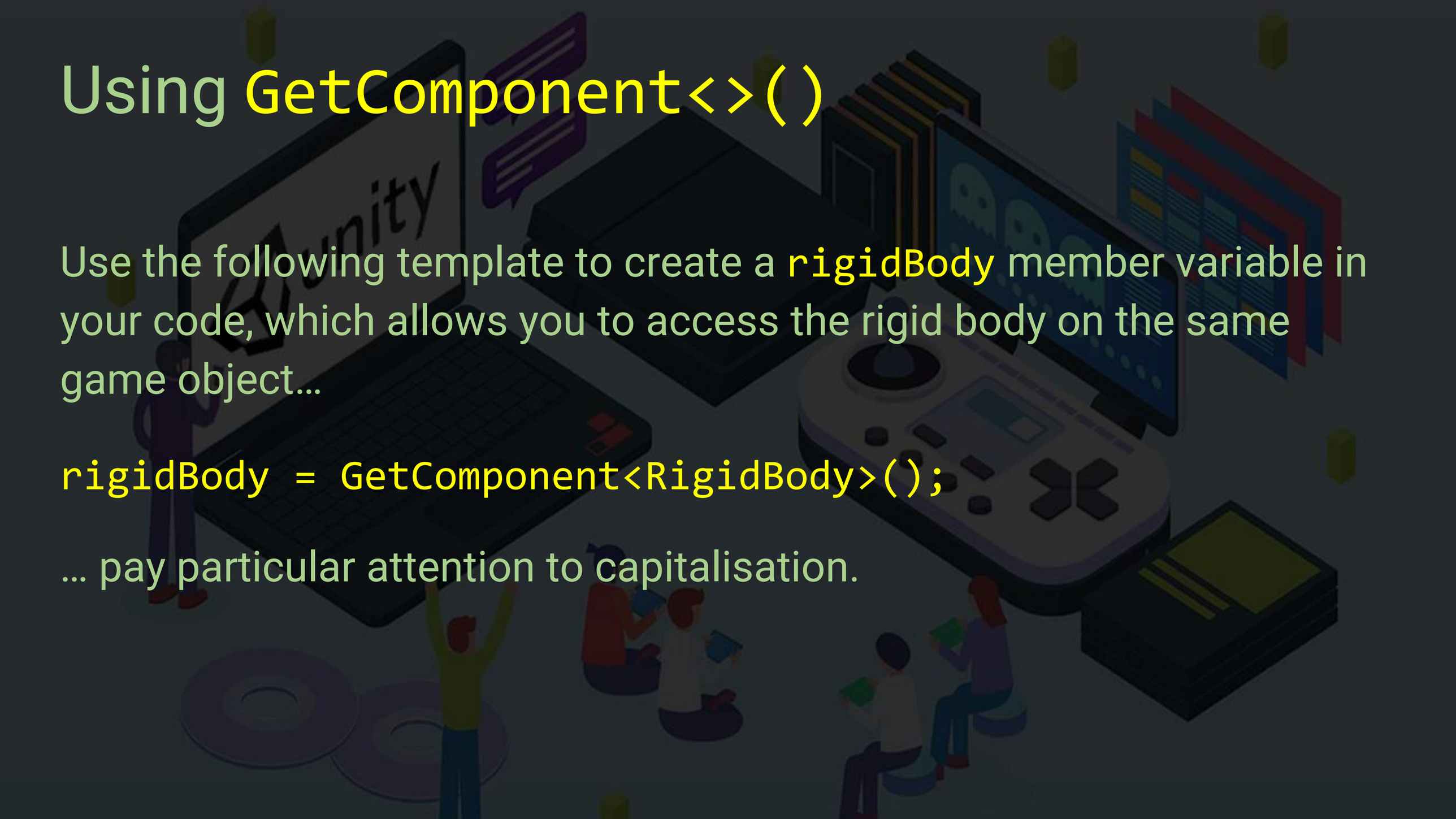
Create Rotation Keys



- Pushing A should repeatedly print “Rotating left”.
- Pushing D should do the same for right.
- You should be able to thrust AND rotate.
- You should not be able to rotate both ways at the same time.

Physics and Rigidbodies

Using `GetComponent<>()`

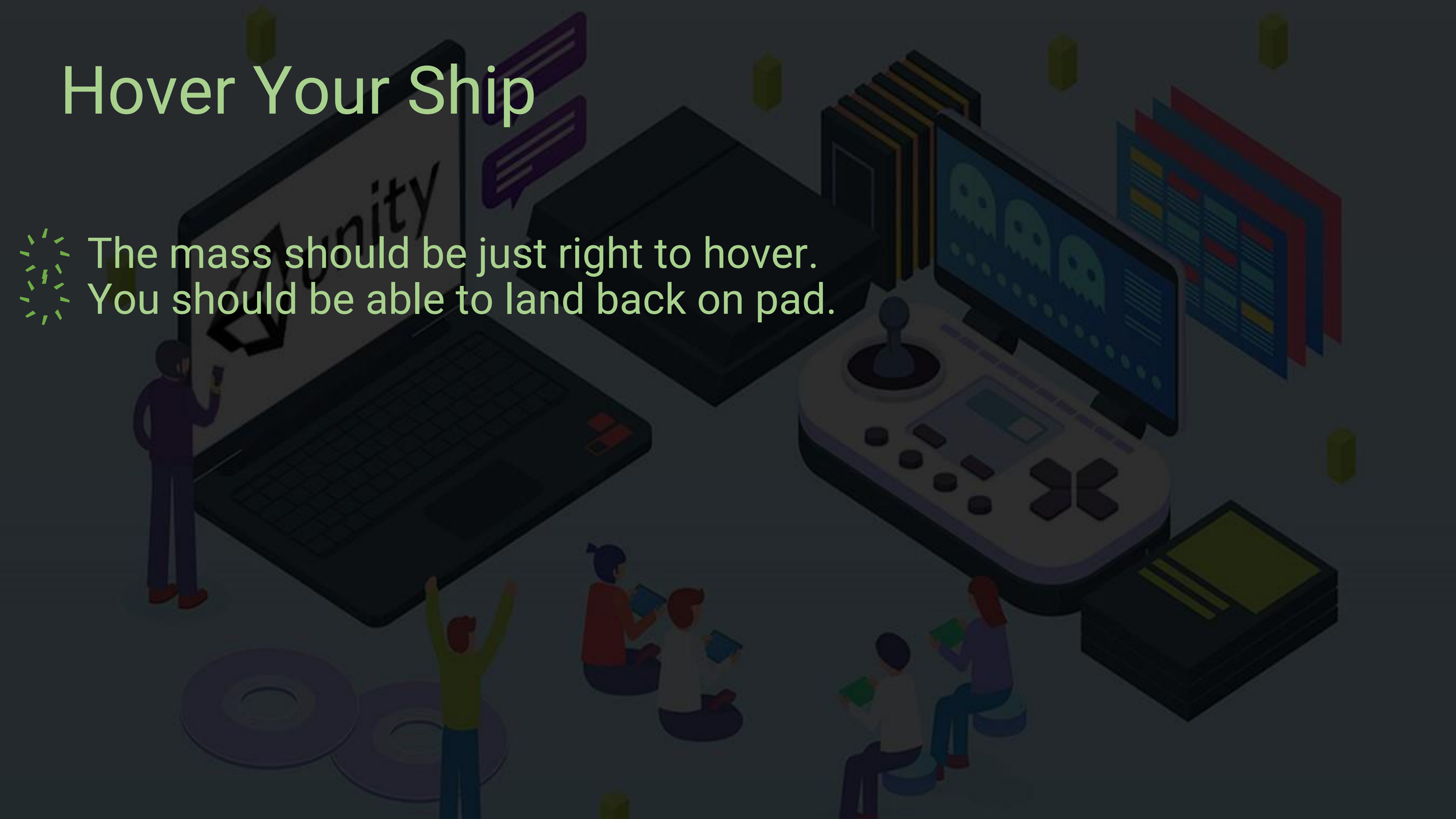


Use the following template to create a `rigidBody` member variable in your code, which allows you to access the rigid body on the same game object…

```
rigidBody = GetComponent<RigidBody>();
```

… pay particular attention to capitalisation.

Hover Your Ship

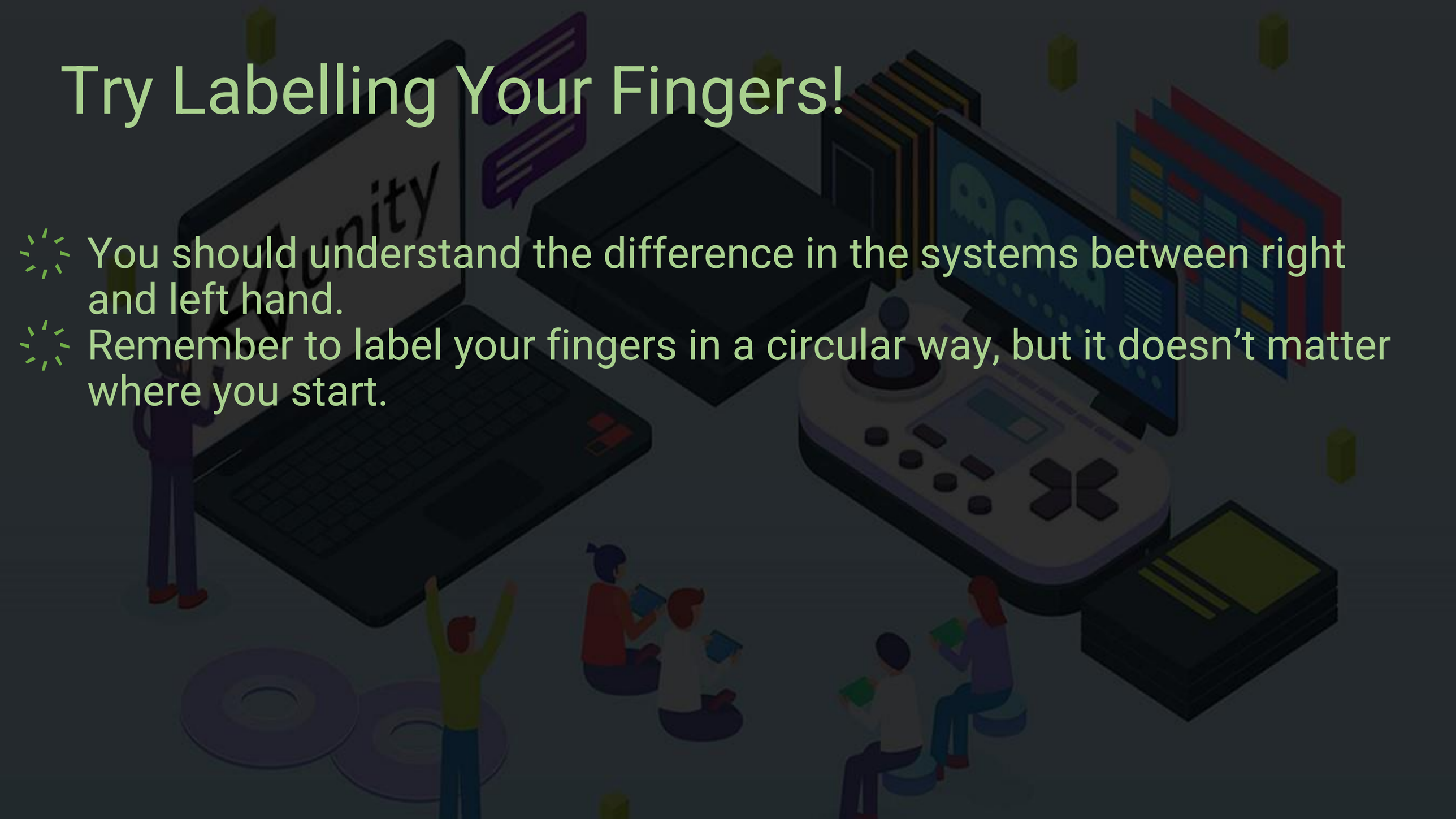


- The mass should be just right to hover.
- You should be able to land back on pad.

Coordinate System Handedness

Unity Uses A Left-Handed System

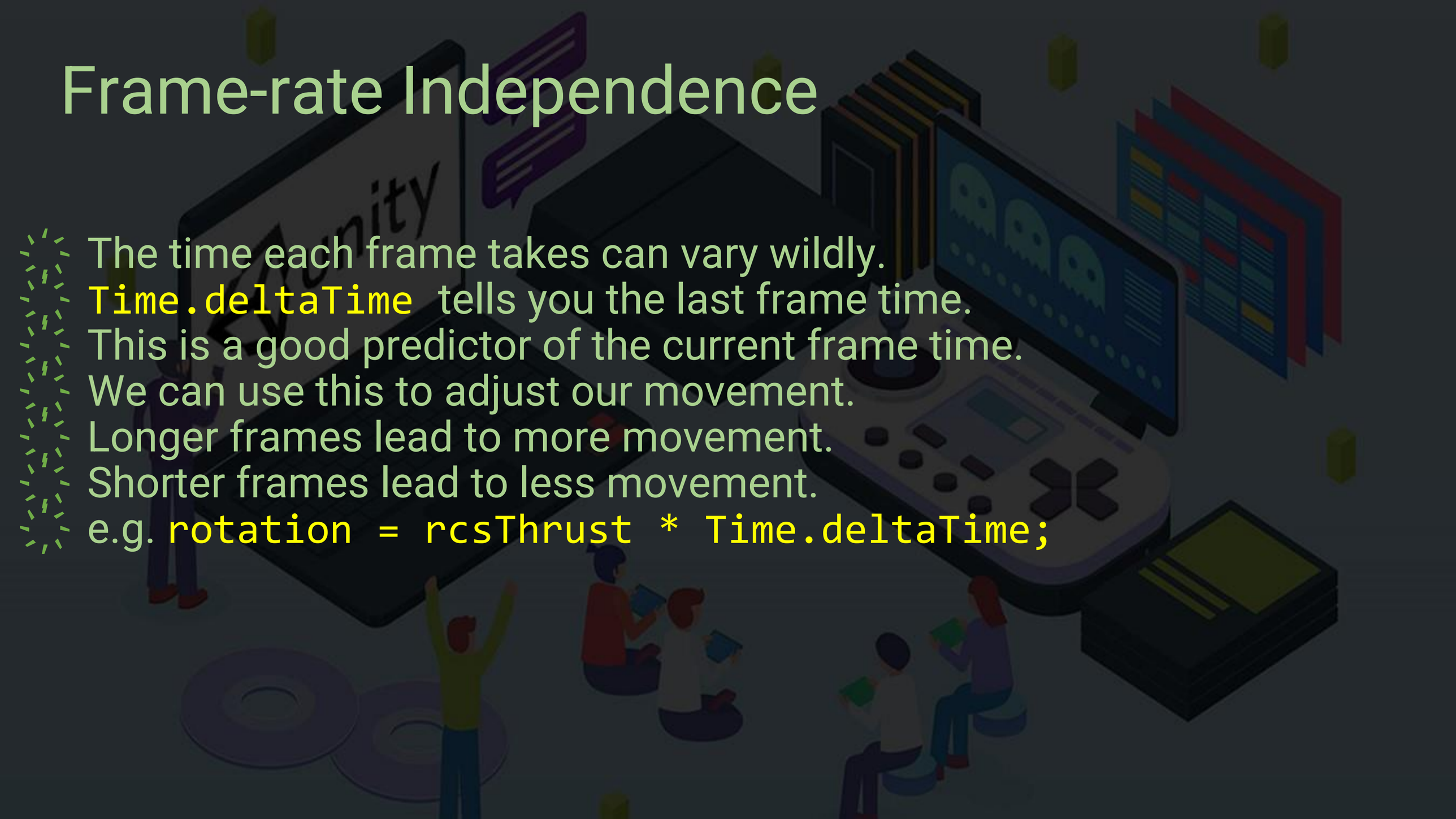
Try Labelling Your Fingers!



- You should understand the difference in the systems between right and left hand.
- Remember to label your fingers in a circular way, but it doesn’t matter where you start.

Using `Time.deltaTime`

Frame-rate Independence



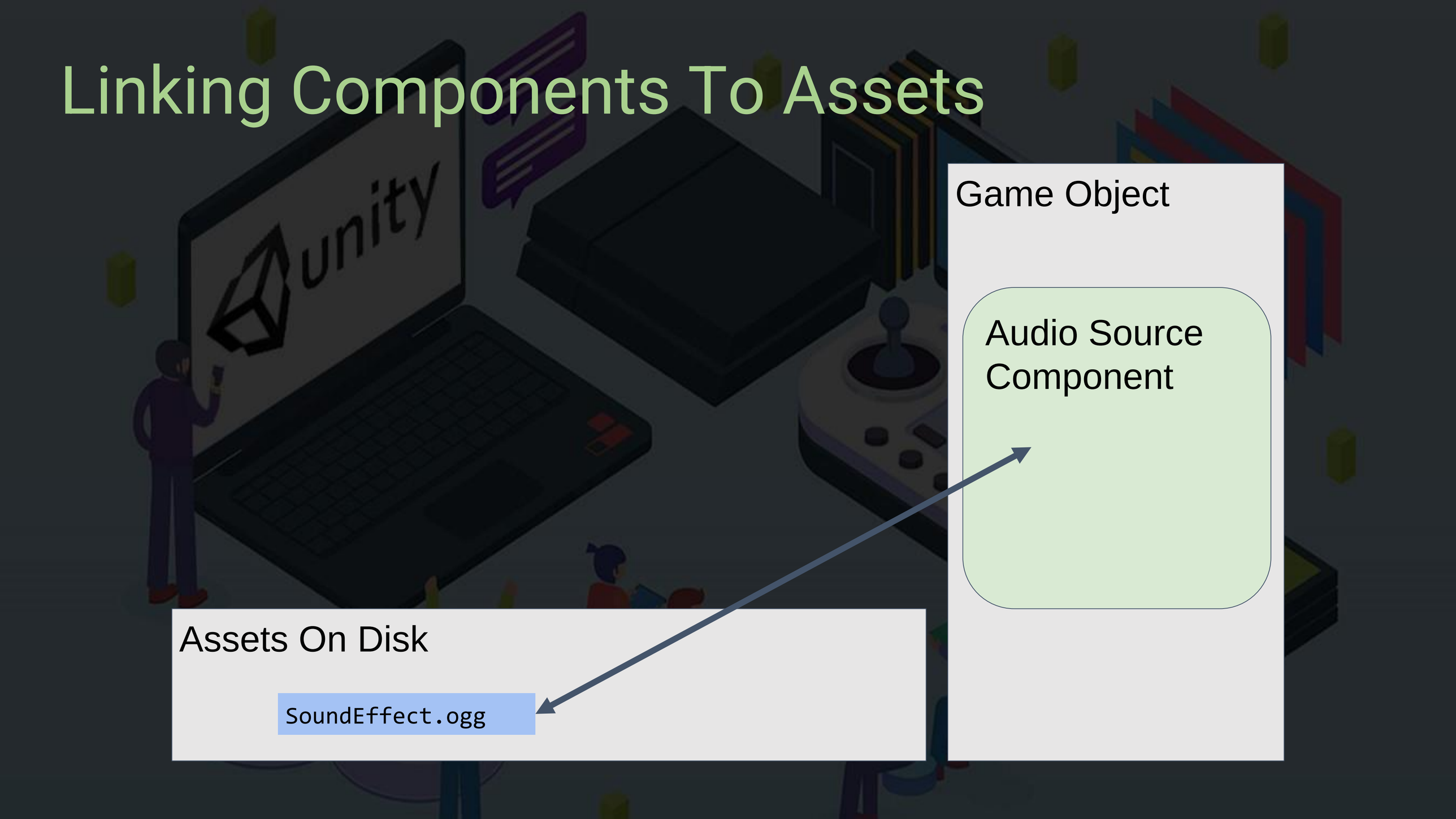
- The time each frame takes can vary wildly.
- `Time.deltaTime` tells you the last frame time.
- This is a good predictor of the current frame time.
- We can use this to adjust our movement.
- Longer frames lead to more movement.
- Shorter frames lead to less movement.
- e.g. `rotation = rcsThrust * Time.deltaTime;`

Words For Parts Of A Transform

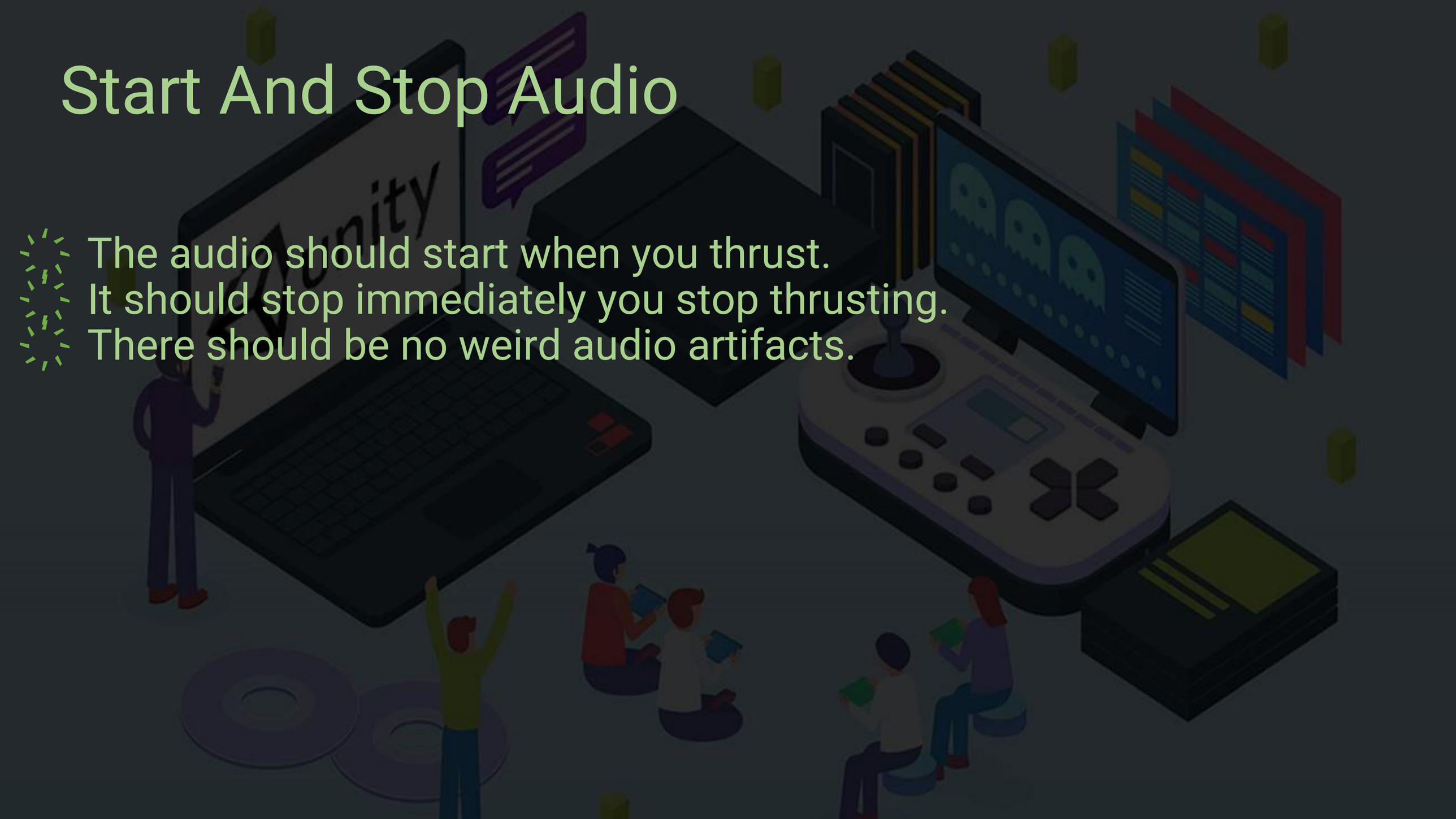
	As A Noun	As A Verb	Code Example
Transform	Position	Translate	`transform.Translate();`
	Rotation	Rotate	`transform.Rotate();`
	Scale	Scale	`transform.localScale;`

Adding A Touch Of Audio

Linking Components To Assets



Start And Stop Audio



- The audio should start when you thrust.
- It should stop immediately you stop thrusting.
- There should be no weird audio artifacts.

Resolving Movement Bugs

Your Movement Is Bug Free

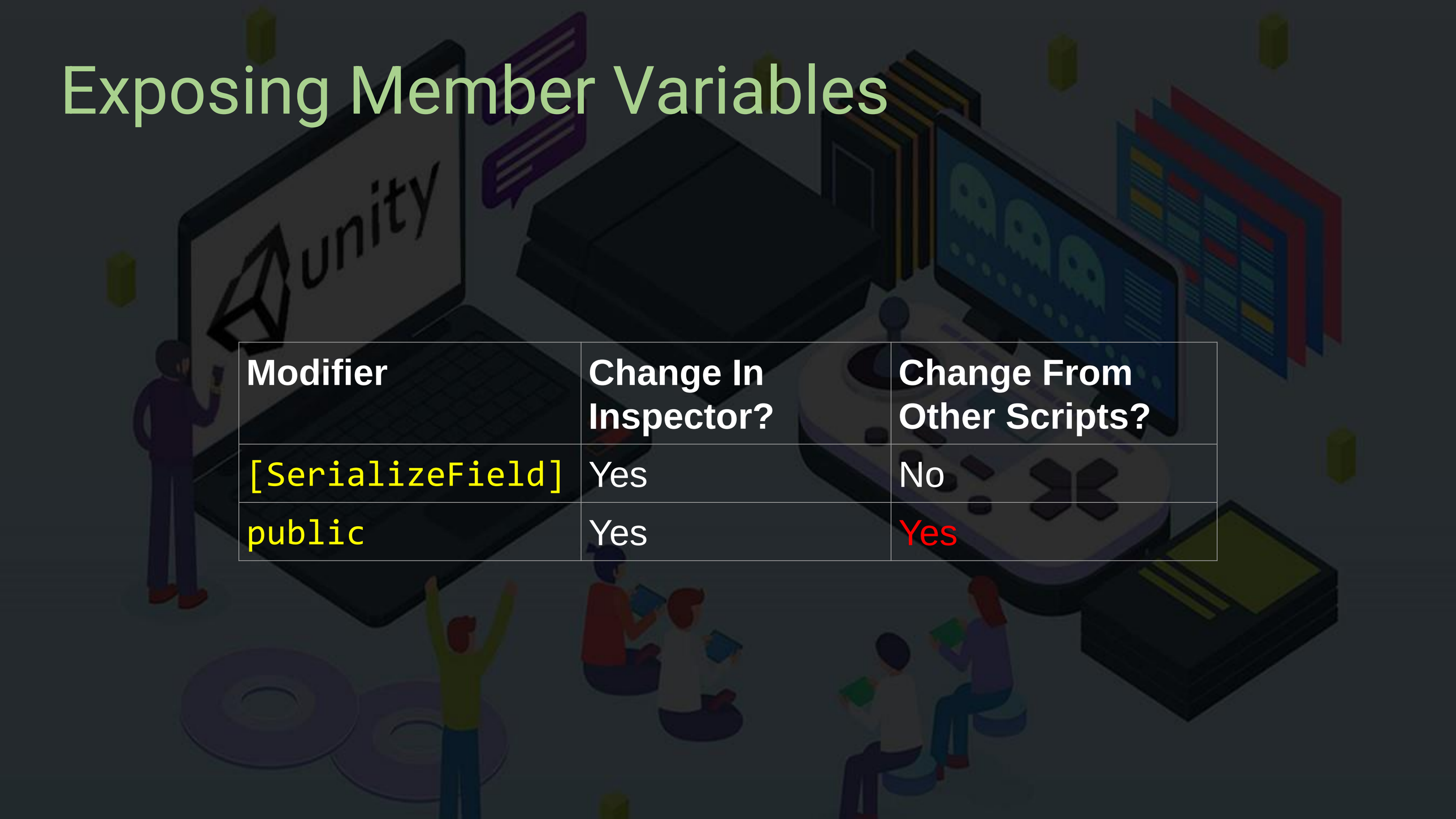
- You can move from one platform to another.
- A platform can induce spin but it stops on thrust.
- You have Drag value you’re happy with.
- Your code is beautiful.

Using `[SerializeField]` vs `public`

Multiplying Vectors

- Multiply a vector by a float.
- You end-up with a new vector.
- New vector is parallel.
- It’s a different length.
- Works for rotation and translation.

Exposing Member Variables



Modifier	Change In Inspector?	Change From Other Scripts?
`[SerializeField]`	Yes	No
`public`	Yes	Yes

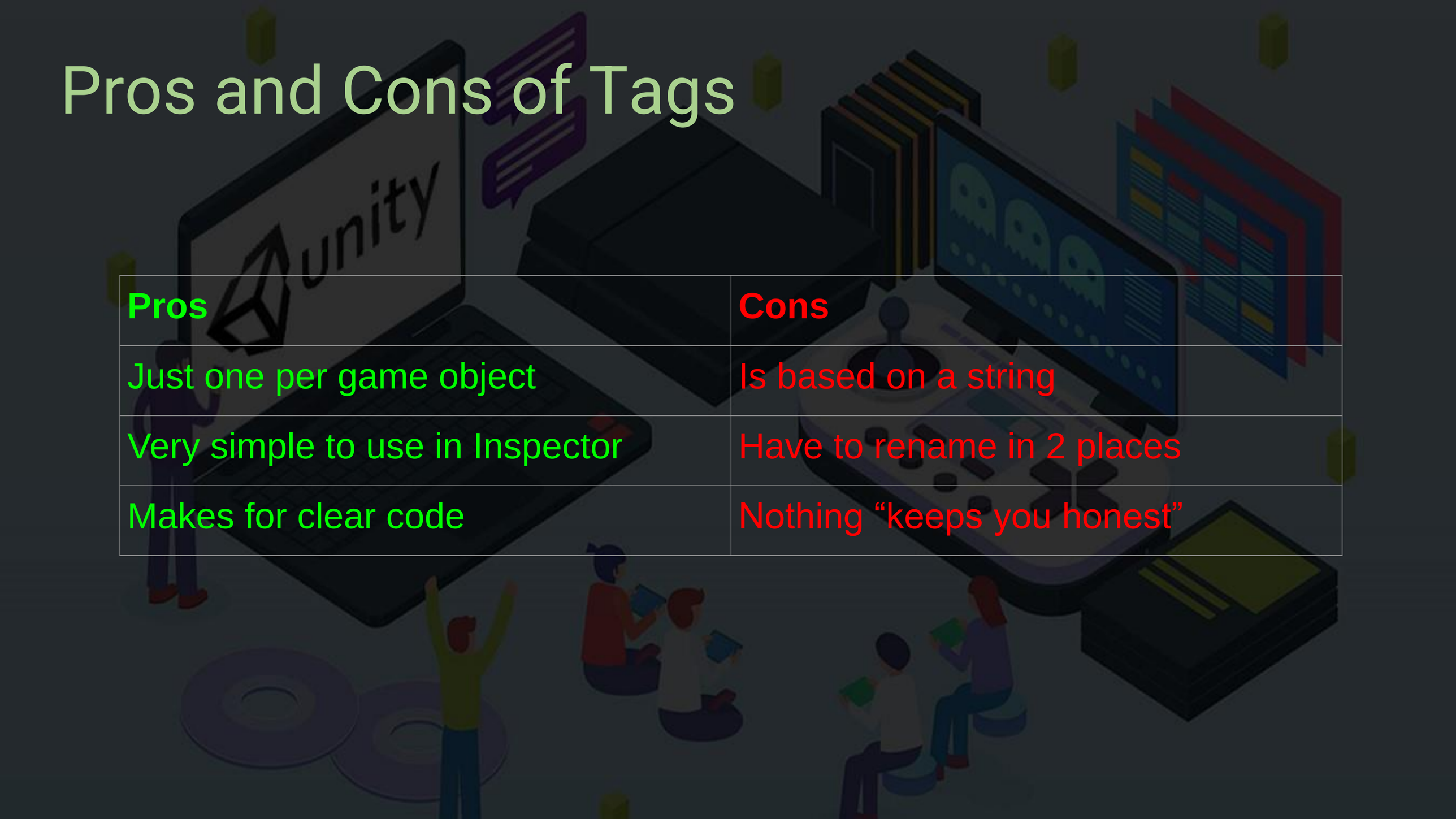
Serialise `mainThrust`

- Main Thrust should be adjustable in the inspector.
- The ship's Rigid Body component should be reset.
- The ship should handle similarly to before.

Enjoy fooling around with your ship!

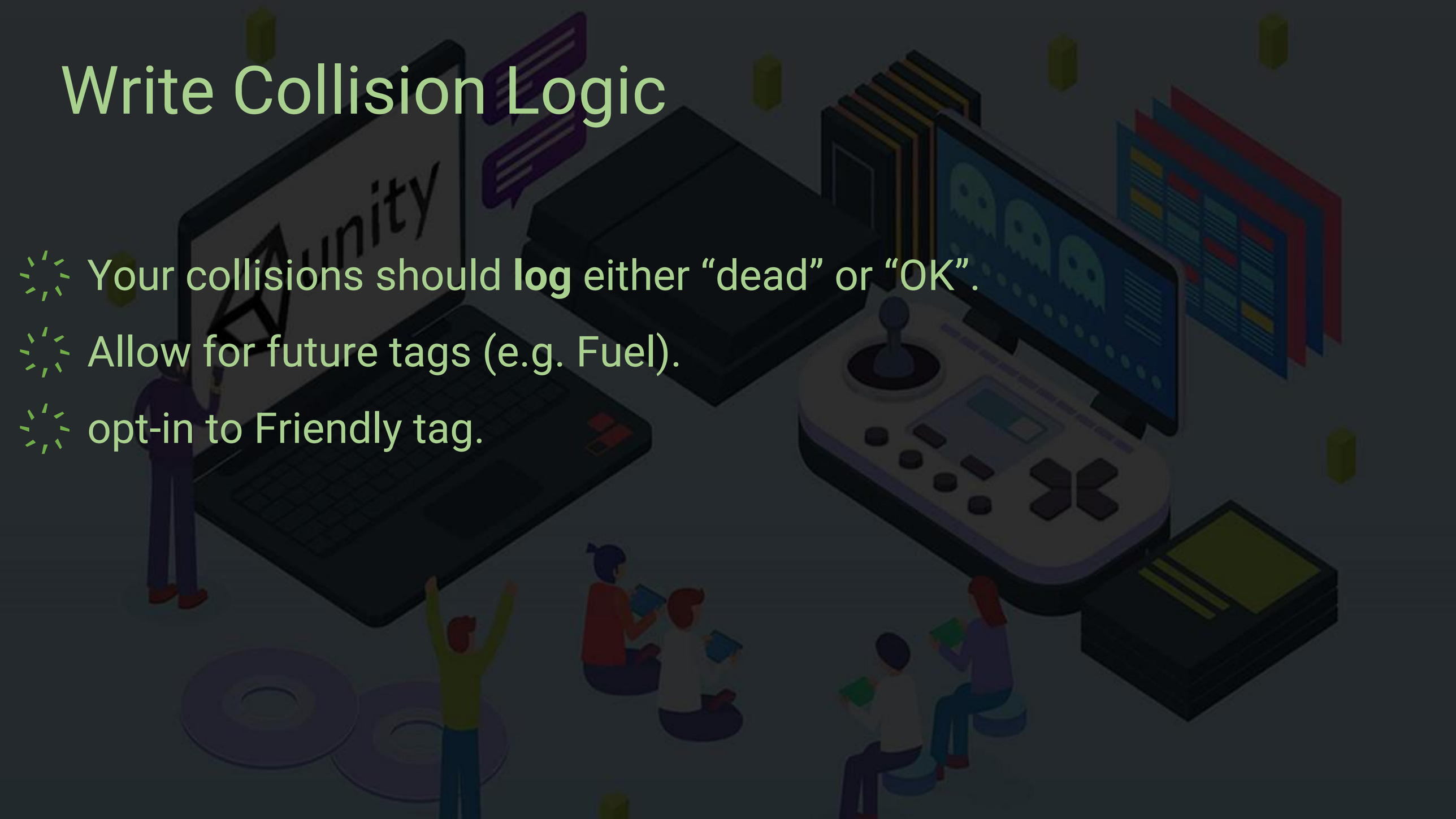
Tagging Game Objects As Friendly

Pros and Cons of Tags



Pros	**Cons**
Just one per game object	Is based on a string
Very simple to use in Inspector	Have to rename in 2 places
Makes for clear code	Nothing “keeps you honest”

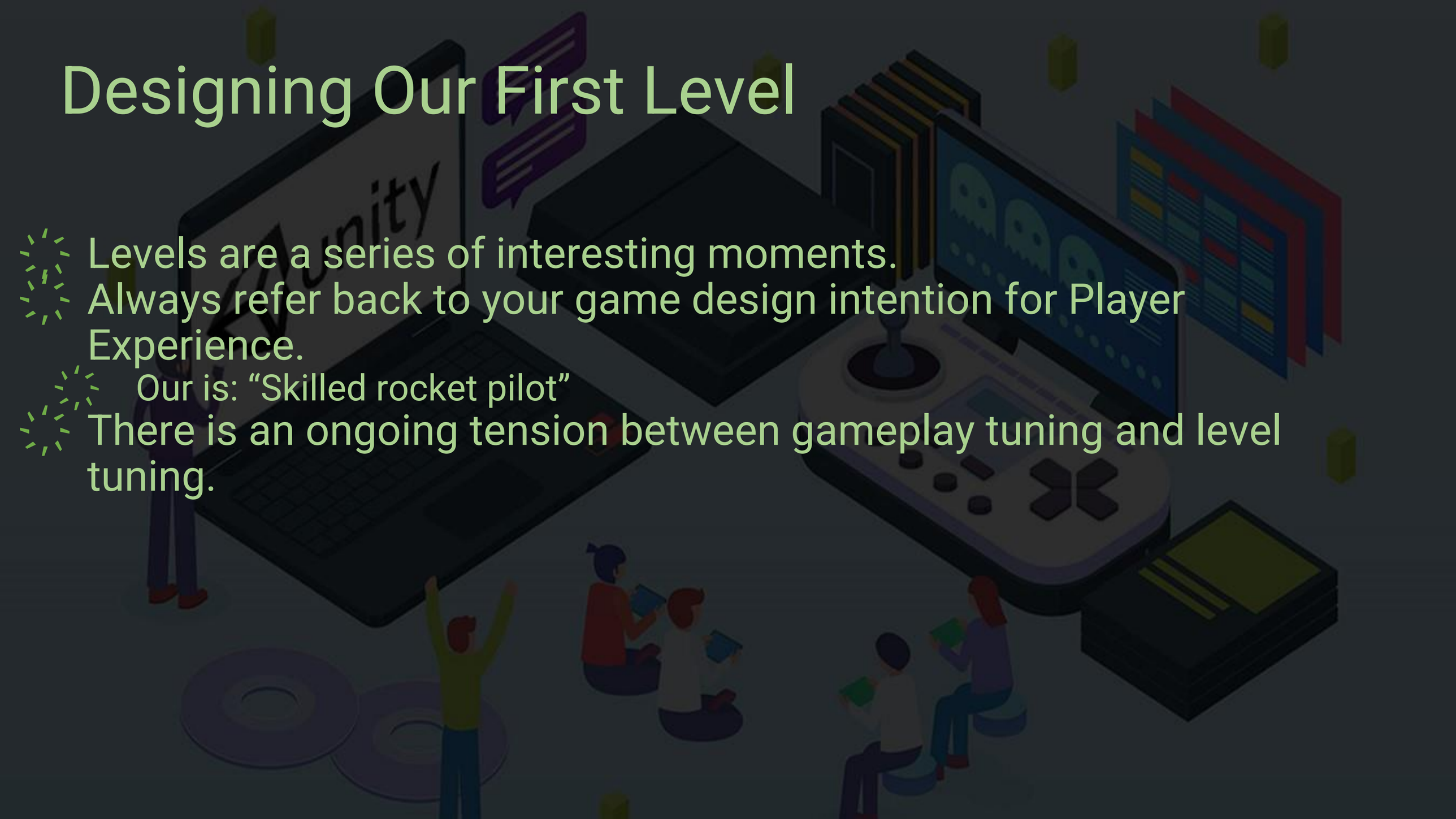
Write Collision Logic



- Your collisions should **log** either “dead” or “OK”.
- Allow for future tags (e.g. Fuel).
- opt-in to Friendly tag.

Basic Level Design

Designing Our First Level



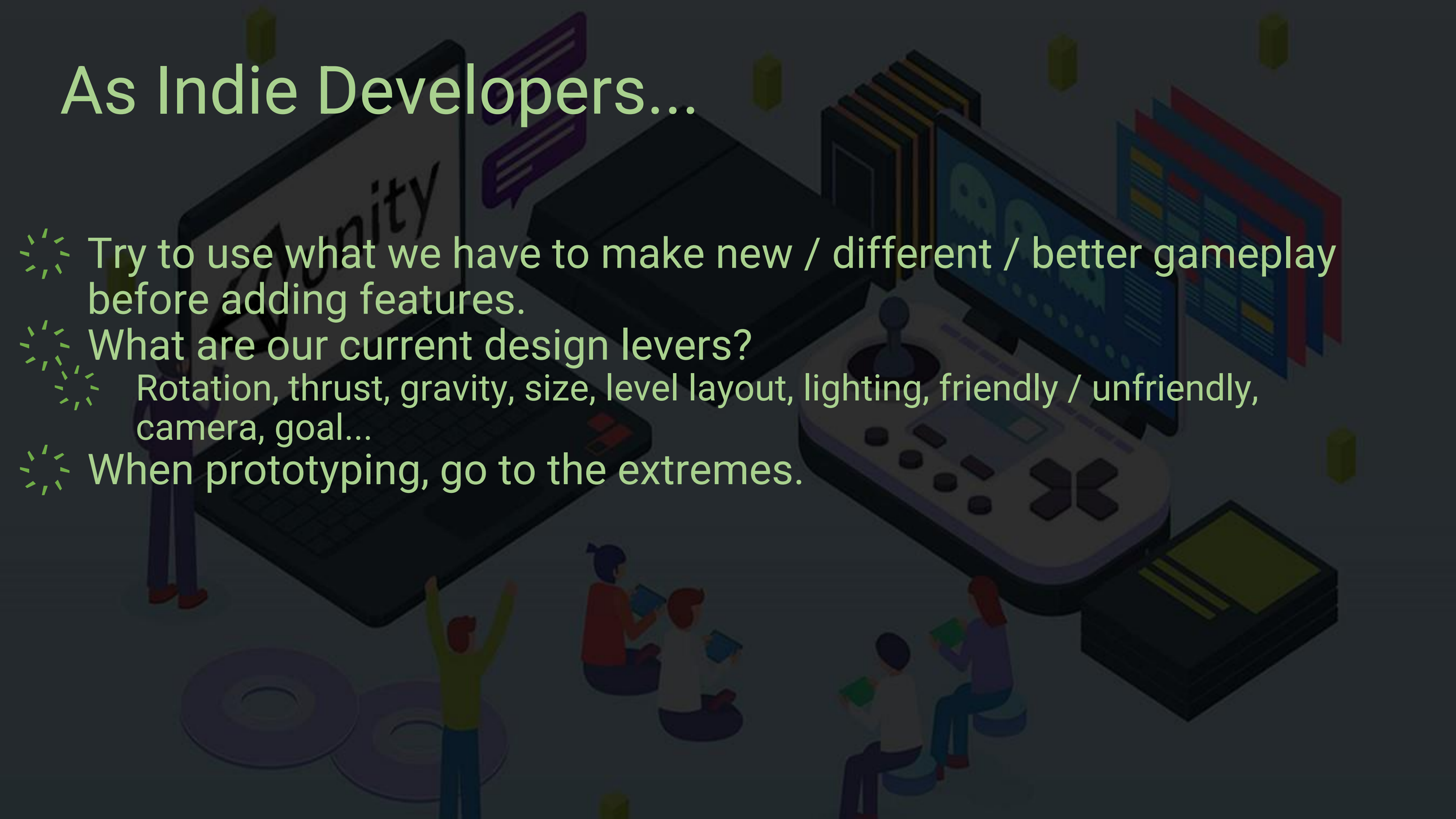
- Levels are a series of interesting moments.
- Always refer back to your game design intention for Player Experience.
  - Our is: “Skilled rocket pilot”
- There is an ongoing tension between gameplay tuning and level tuning.

Create An Interesting Moment

- Refine your camera if need be.
- Place start, finish and obstacles to create an interesting moment.
- Playtesting and refine.

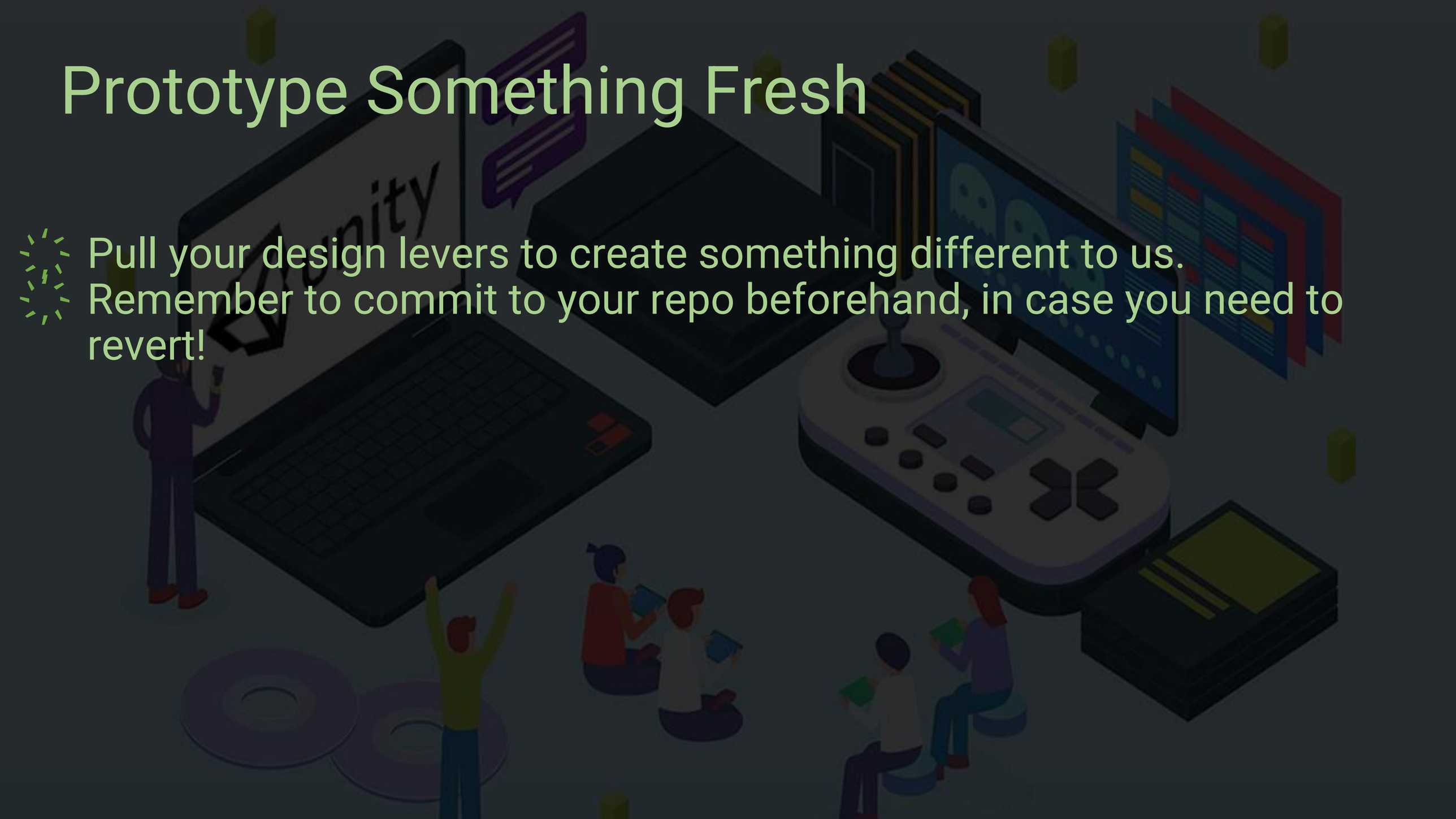
Design Levels And Variation

As Indie Developers...



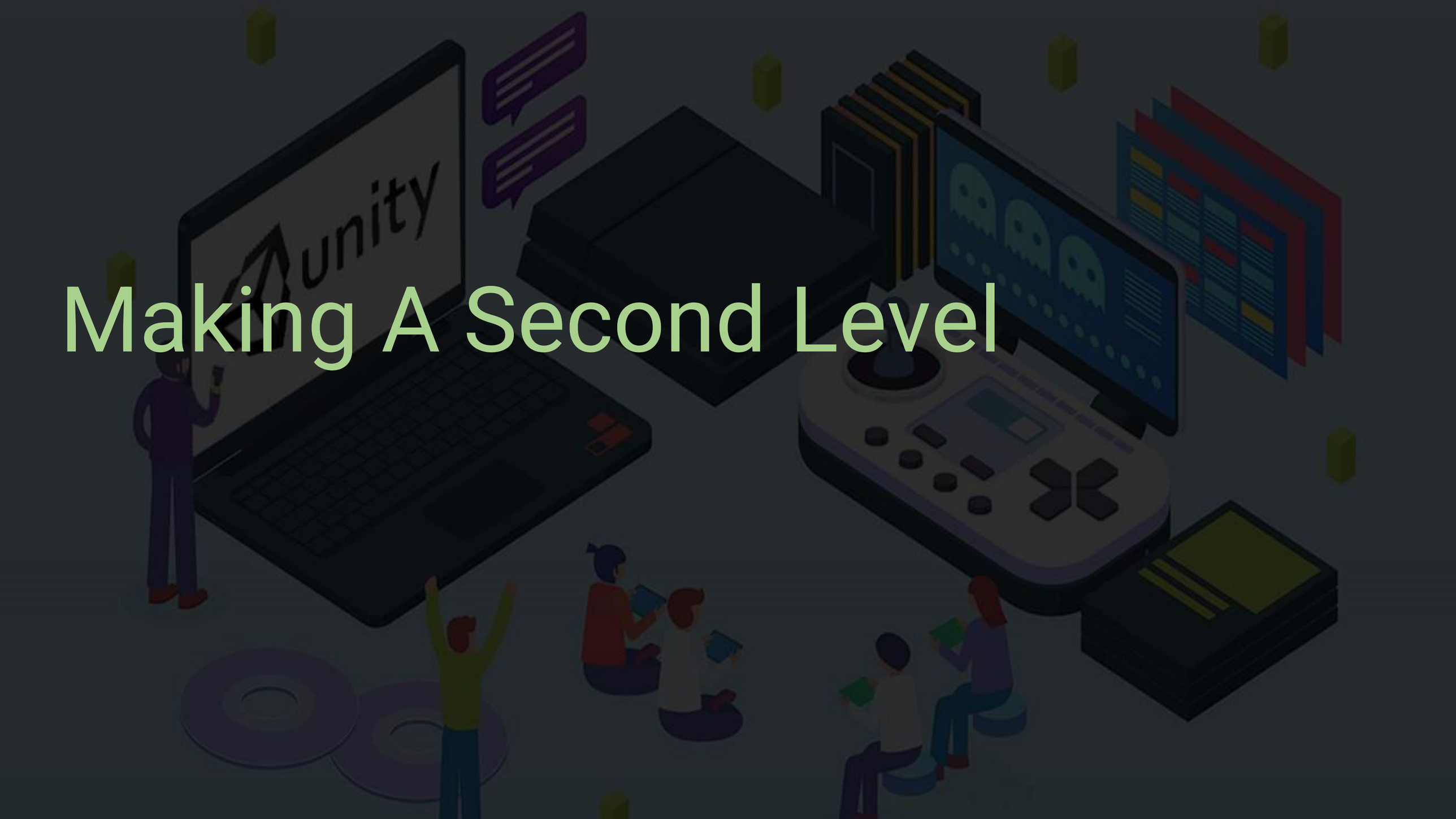
- Try to use what we have to make new / different / better gameplay before adding features.
- What are our current design levers?
  - Rotation, thrust, gravity, size, level layout, lighting, friendly / unfriendly, camera, goal...
- When prototyping, go to the extremes.

Prototype Something Fresh

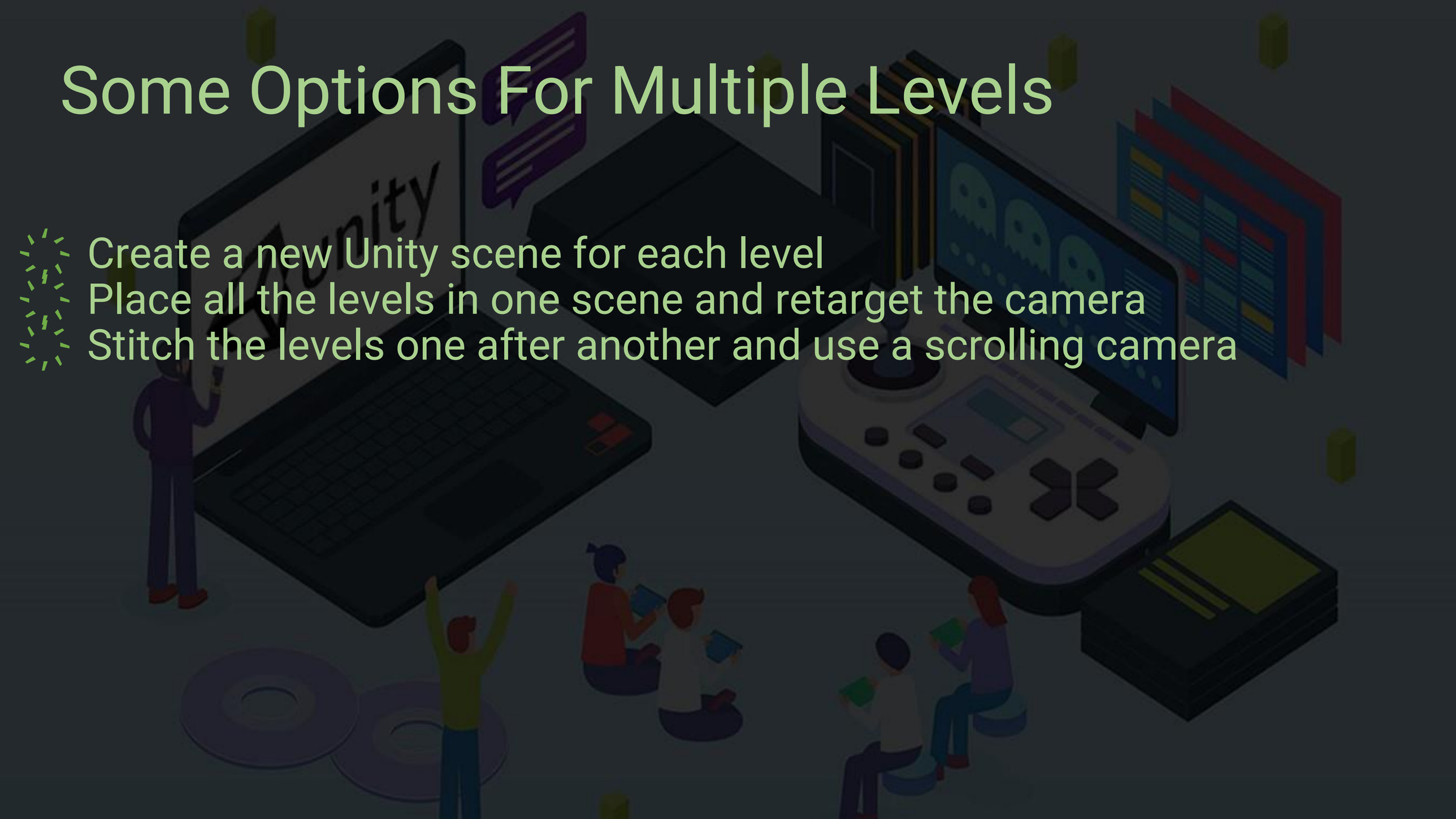


- Pull your design levers to create something different to us. Remember to commit to your repo beforehand, in case you need to revert!

Making A Second Level

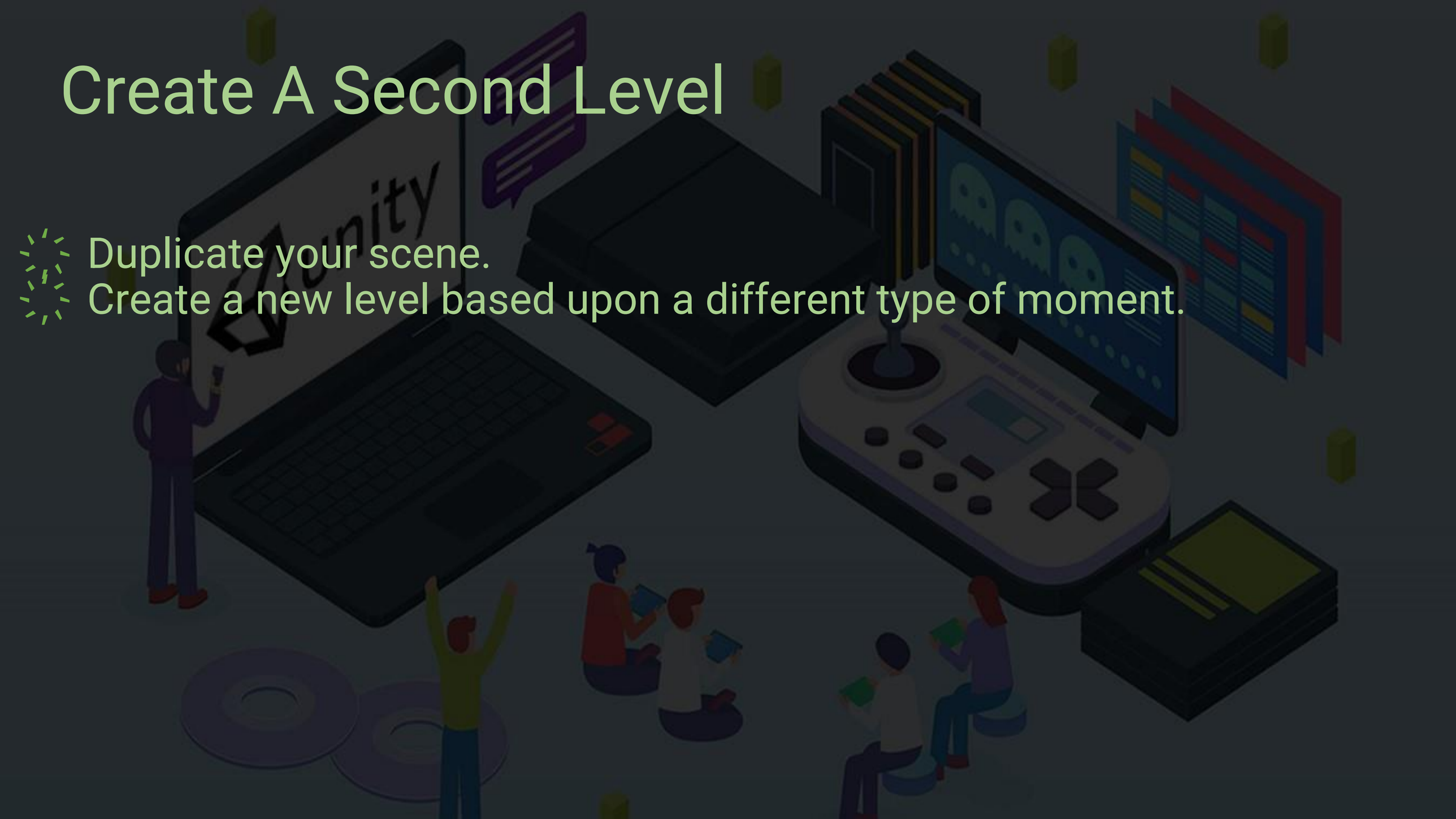


Some Options For Multiple Levels



- Create a new Unity scene for each level
- Place all the levels in one scene and retarget the camera
- Stitch the levels one after another and use a scrolling camera

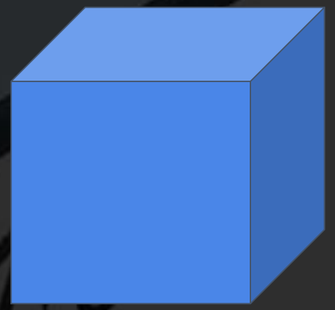
Create A Second Level



- Duplicate your scene.
- Create a new level based upon a different type of moment.

Prefabs In Detail

Create Object



Prefab & Save Scene

Position & Rotation

All other details

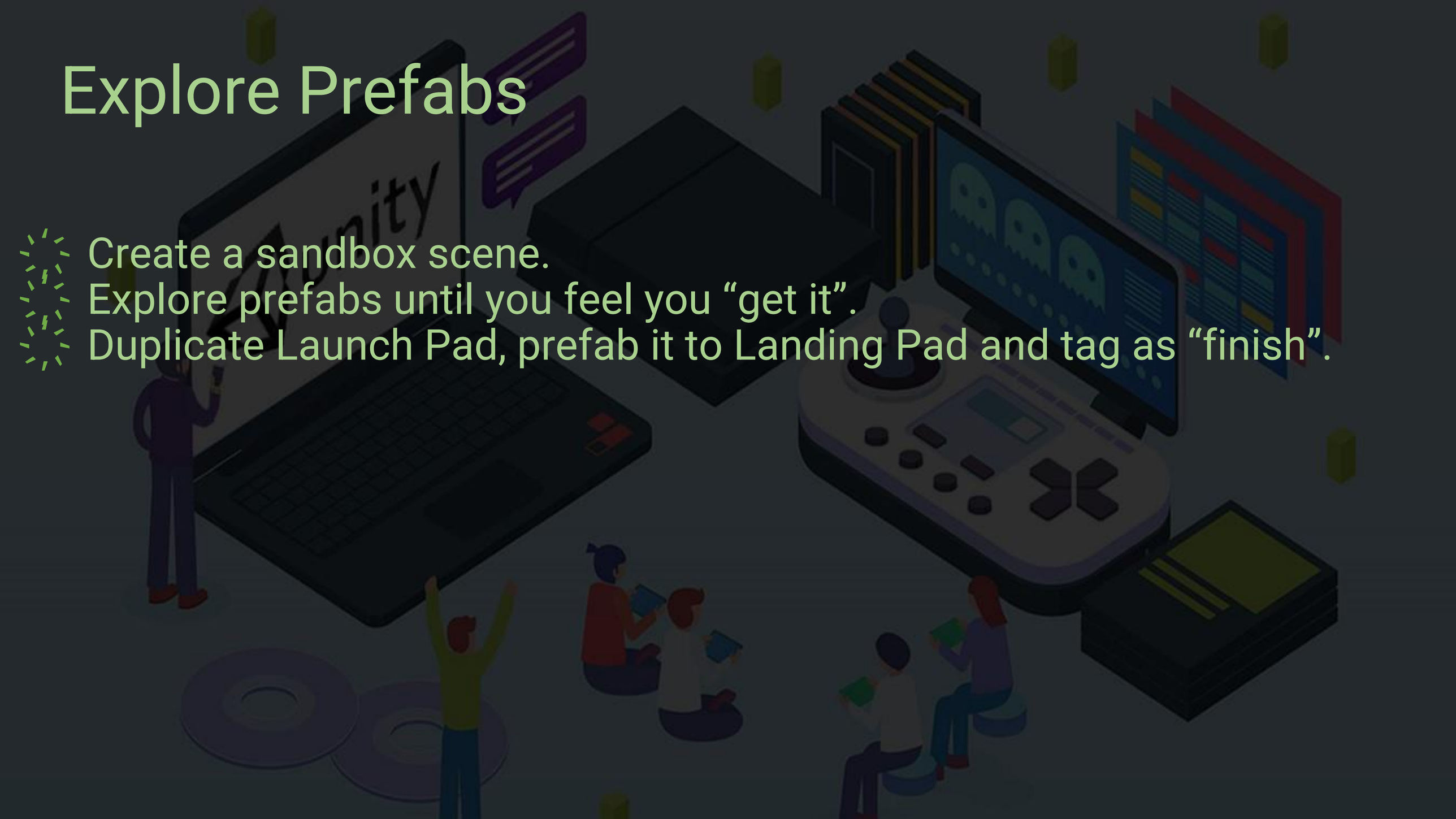
`.unity` scene file

Position & Rotation

All other details

New .prefab file

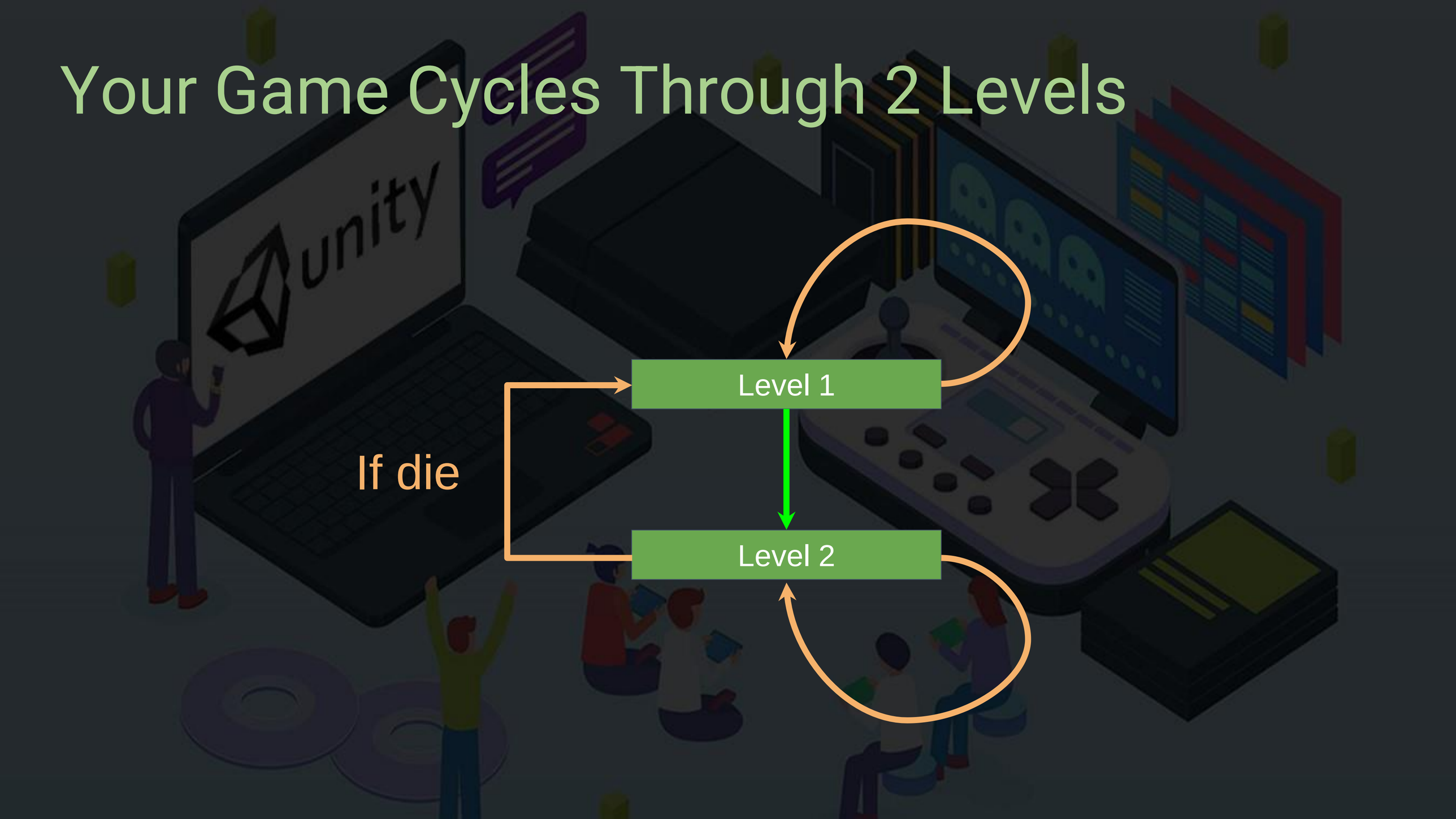
Explore Prefabs



- Create a sandbox scene.
- Explore prefabs until you feel you “get it”.
- Duplicate Launch Pad, prefab it to Landing Pad and tag as “finish”.

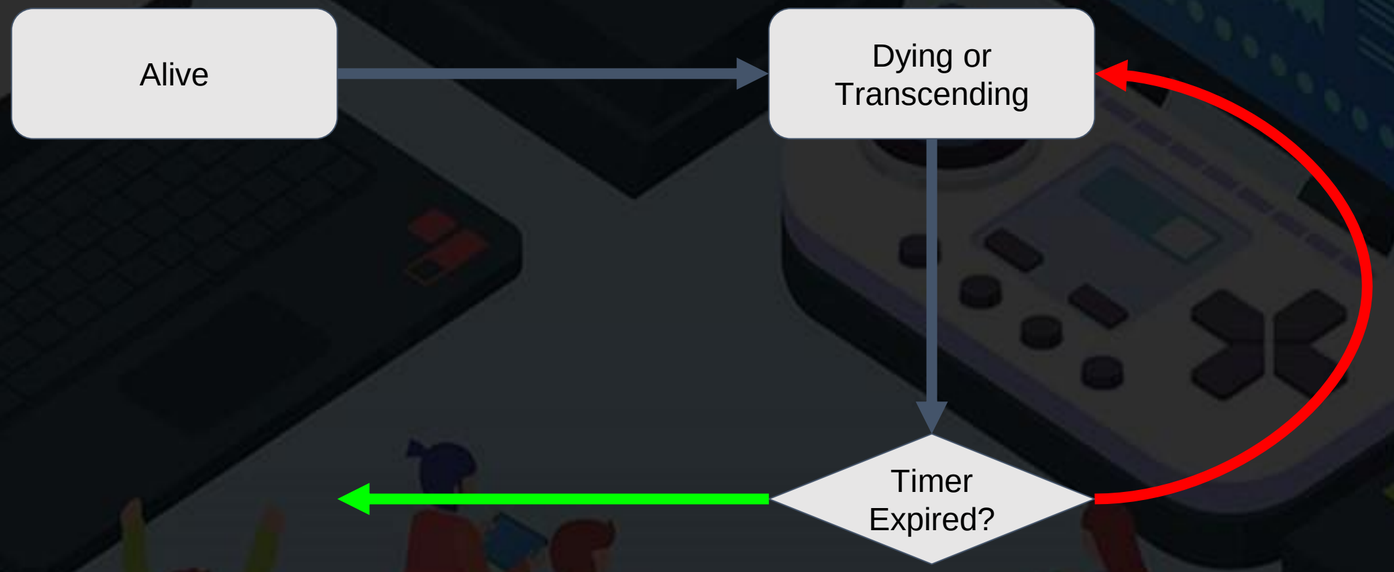
Level Loading & Scene Management

Your Game Cycles Through 2 Levels



`Invoke()` As A Coroutine Warm-up

Delaying Level Load



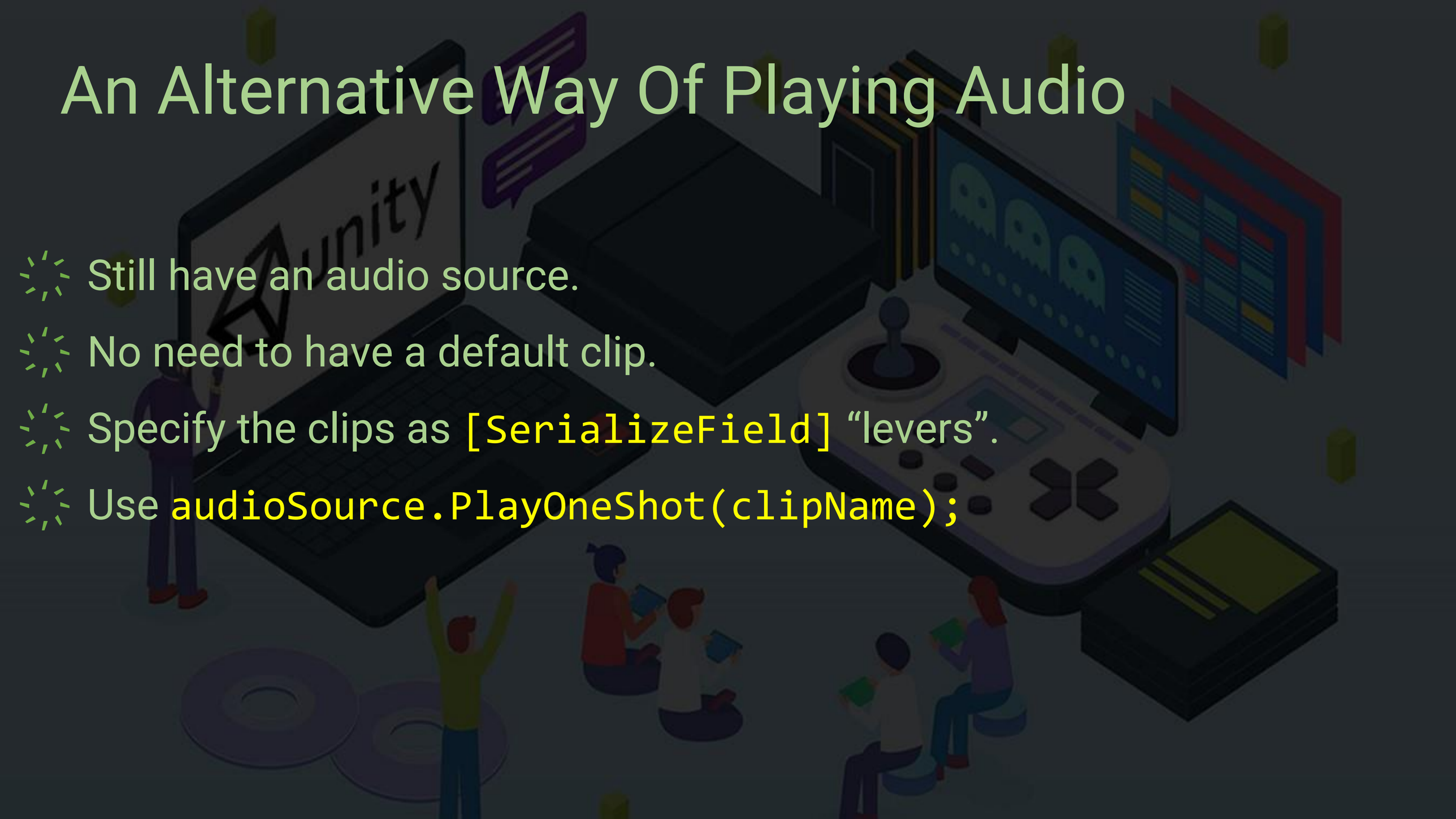
Remember other messages still arrive while waiting for timer

Delay Level Load On Death

- The first level should still load when you die.
- There should be a delay before it does so.
- Player controls should be disabled while dying.
- Create a new method.

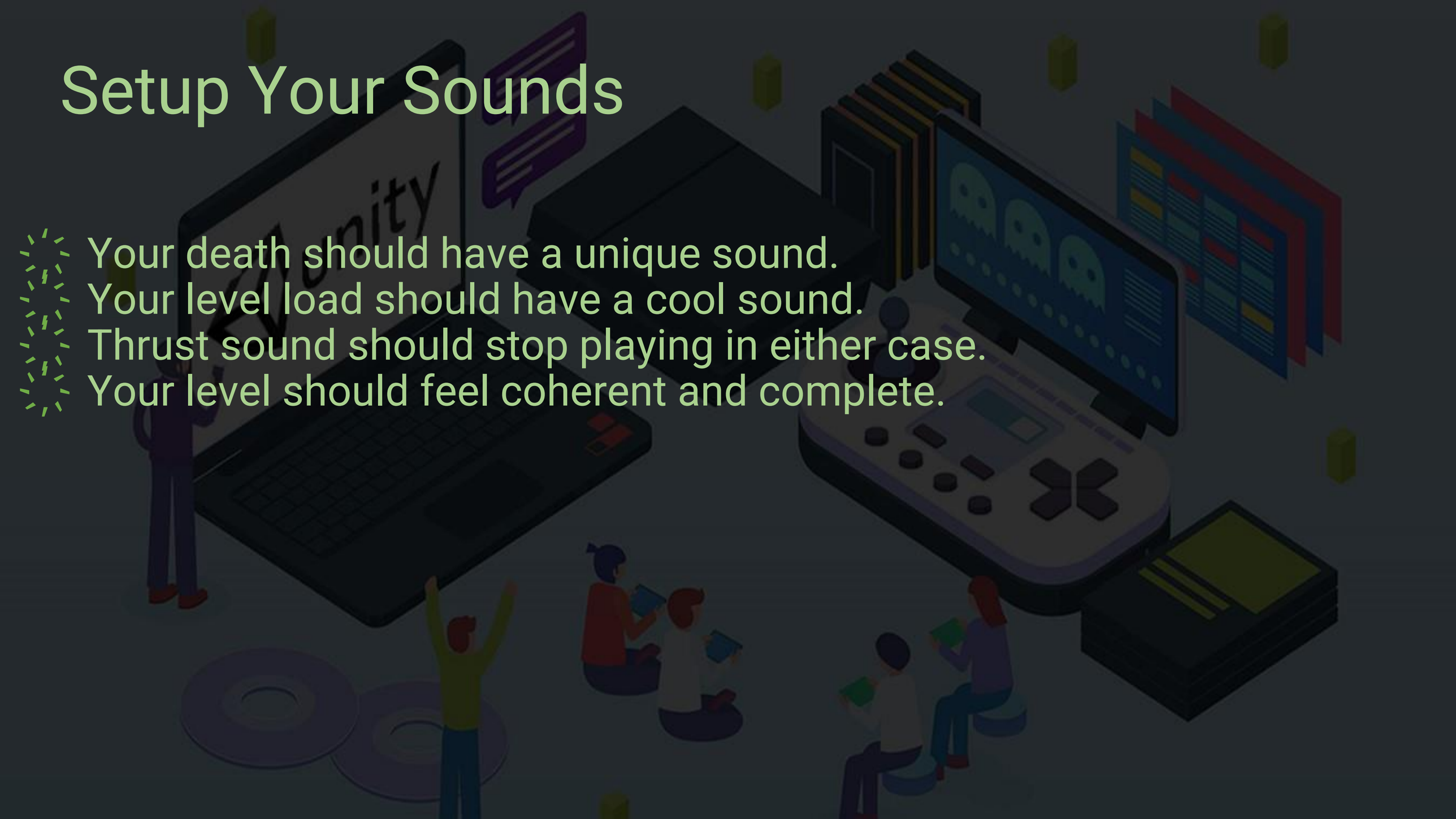
Playing Multiple Audio Clips

An Alternative Way Of Playing Audio



- Still have an audio source.
- No need to have a default clip.
- Specify the clips as `[SerializeField]` “levers”.
- Use `audioSource.PlayOneShot(clipName);`

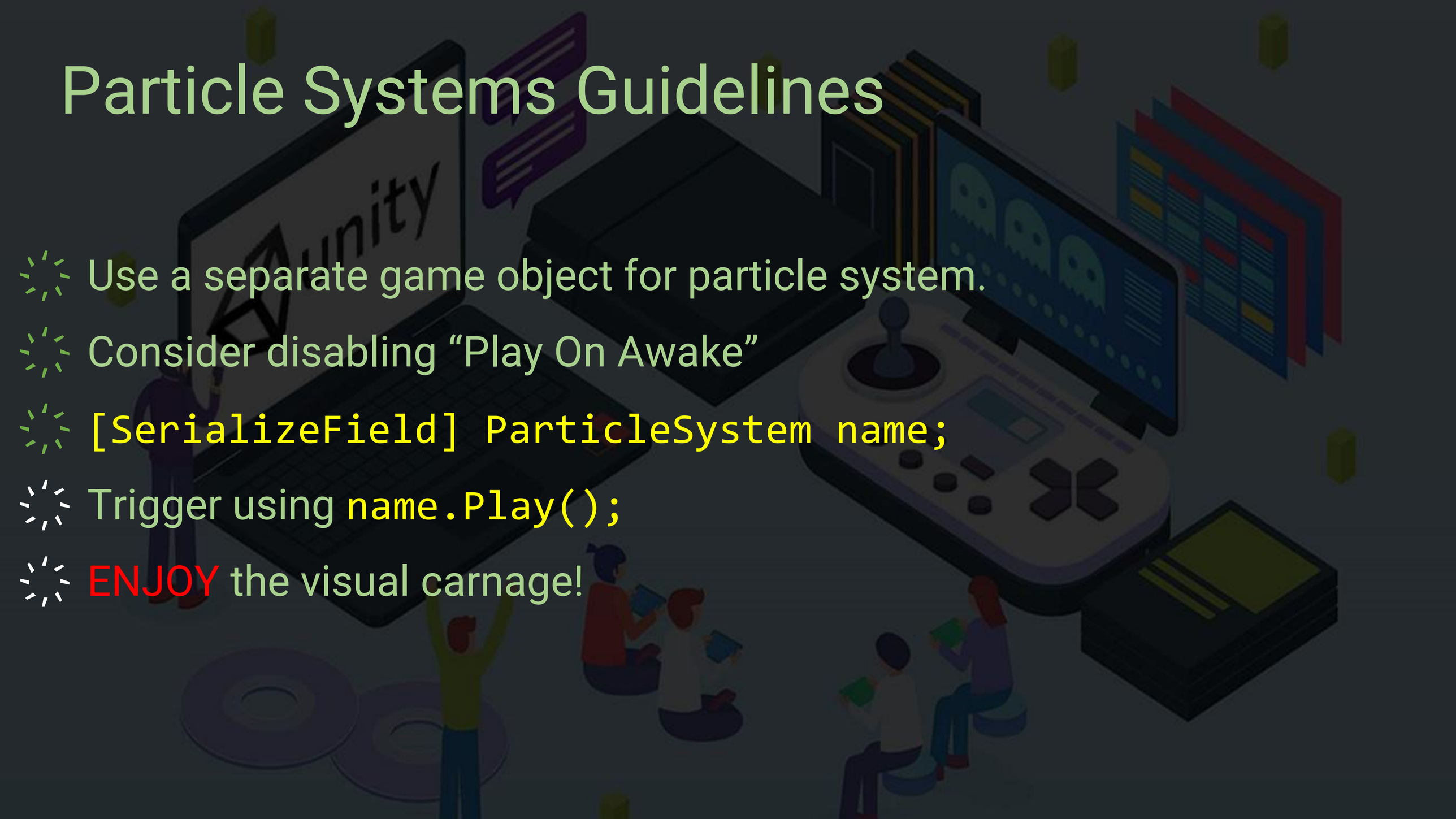
Setup Your Sounds



- Your death should have a unique sound.
- Your level load should have a cool sound.
- Thrust sound should stop playing in either case.
- Your level should feel coherent and complete.

Introducing Particle Effects

Particle Systems Guidelines



- Use a separate game object for particle system.
- Consider disabling “Play On Awake”
- `[SerializeField] ParticleSystem name;`
- Trigger using `name.Play();`
- ENJOY the visual carnage!

Trigger Particles On Death

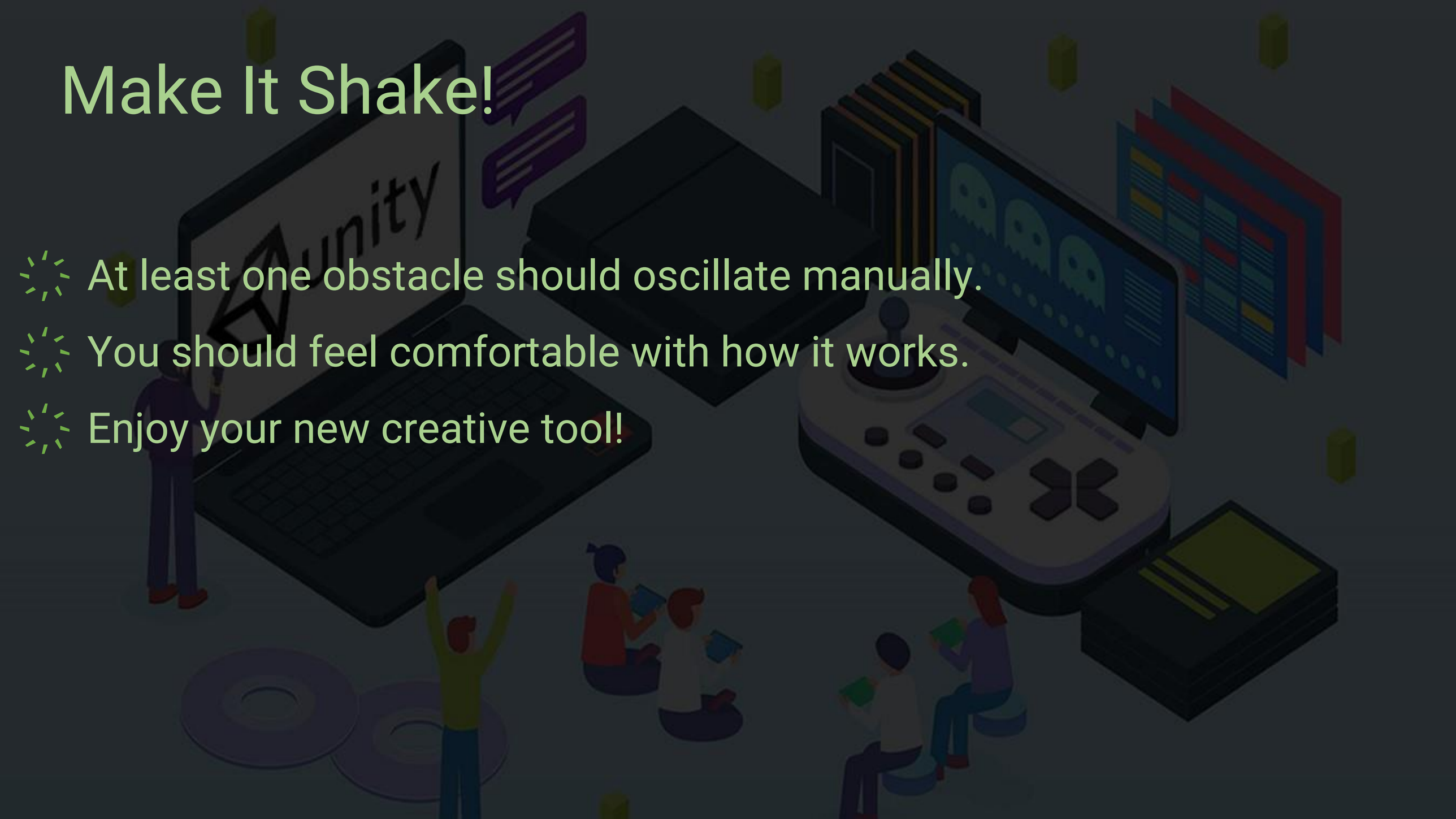
- There should be a spectacle on death.
- Your code should be super clean.
- Audio should still work fine.

Moving Platform Pattern

Manually Moving Platforms

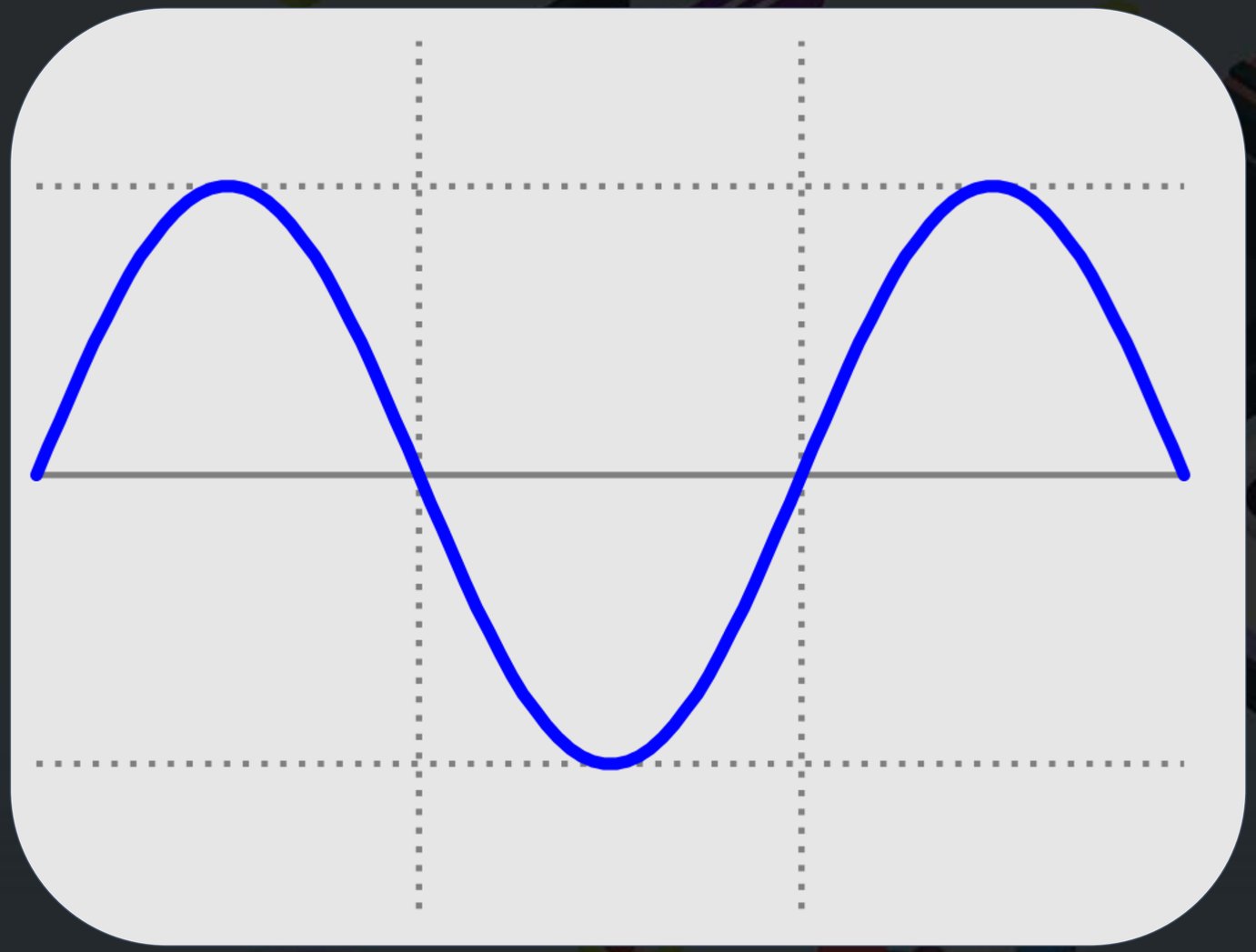
```
[SerializeField] Vector3 movementVector;
[Range(0, 1)] [SerializeField] float movementFactor;

Vector3 startingPos; // must be stored for absolute movement

void Start()
{
    startingPos = transform.position;
}

void Update()
{
    Vector3 offset = movementVector * movementFactor;
    transform.position = startingPos + offset;
}
```

Make It Shake!



- At least one obstacle should oscillate manually.
- You should feel comfortable with how it works.
- Enjoy your new creative tool!

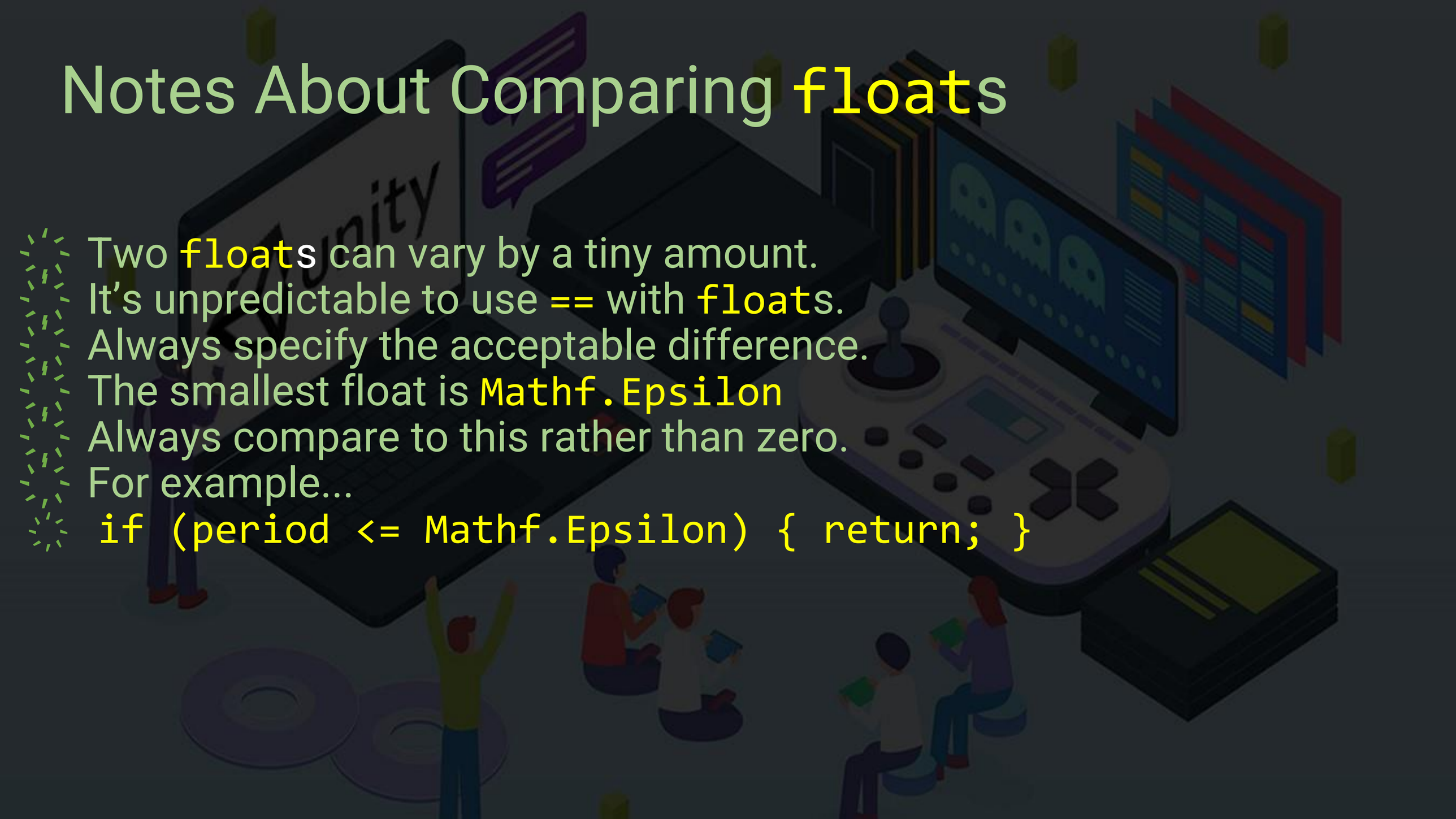
`Mathf.Sin()` For Movement Cycles


amplitude (m)
period (s)

Setup An Oscillator

- Setup at least one game object to oscillate.
- Have fun!

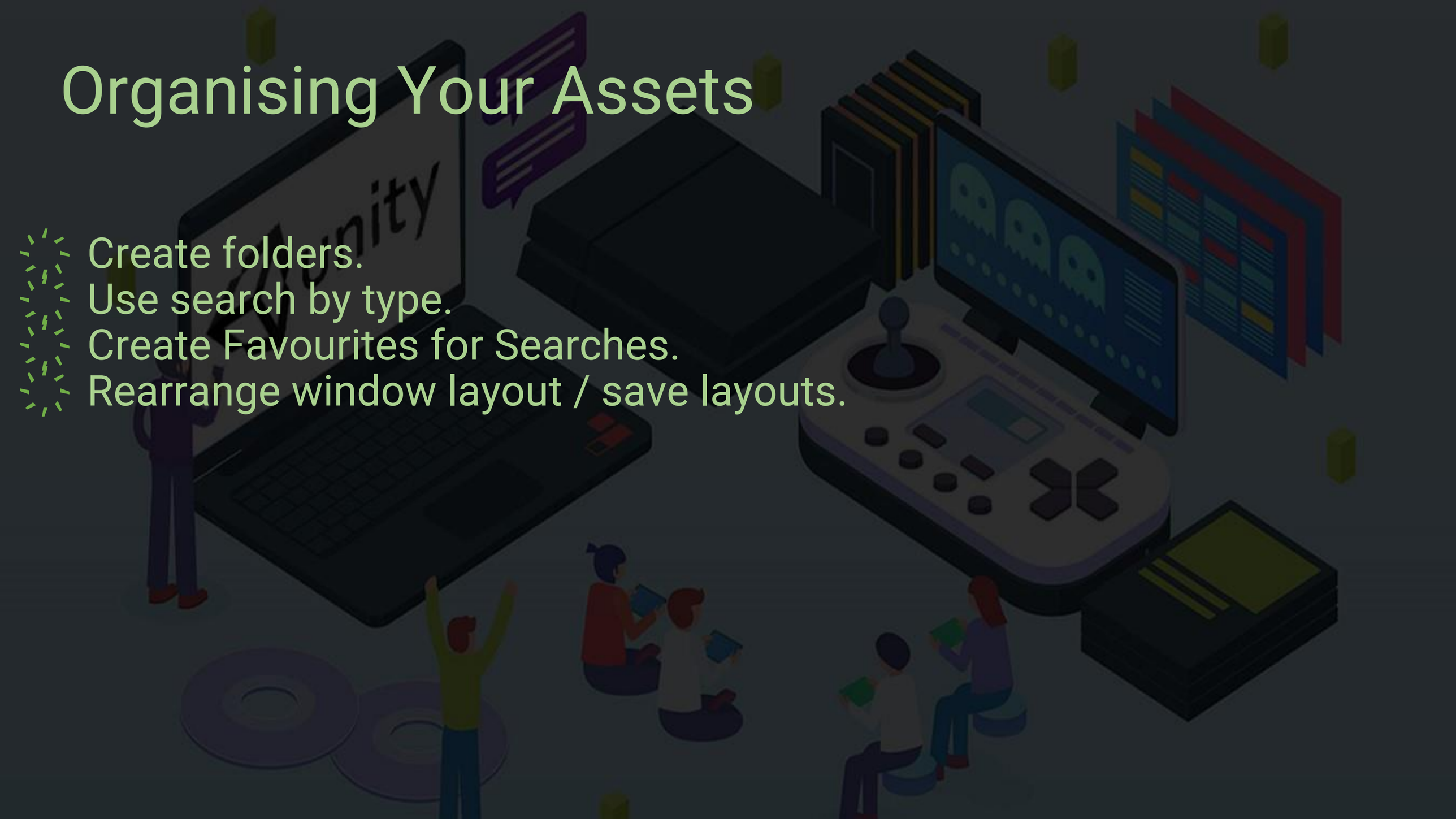
Notes About Comparing `floats`



- Two `floats` can vary by a tiny amount.
- It's unpredictable to use `==` with `floats`.
- Always specify the acceptable difference.
- The smallest float is `Mathf.Epsilon`
- Always compare to this rather than zero.
- For example...

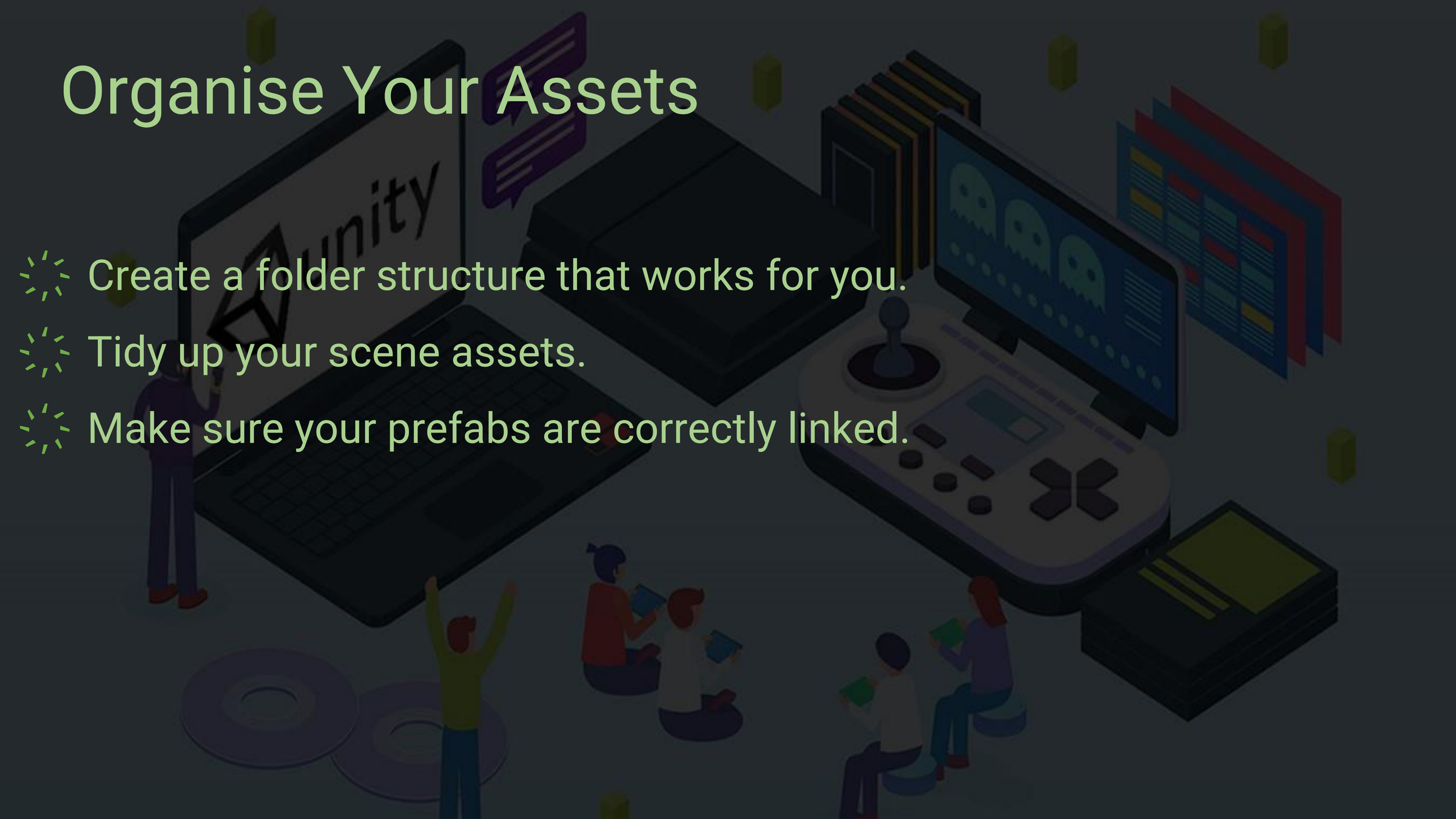
```
if (period <= Mathf.Epsilon) { return; }
```

Organising Your Assets



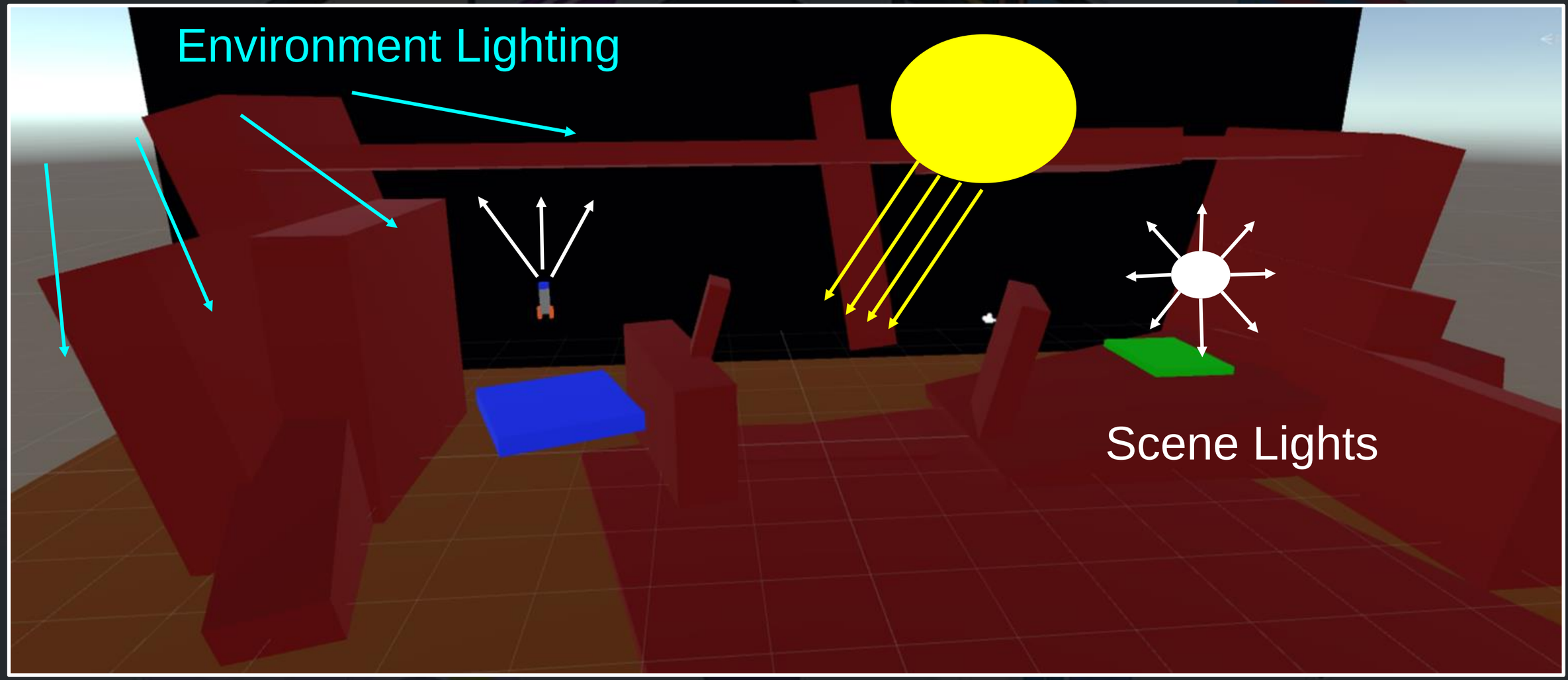
- Create folders.
- Use search by type.
- Create Favourites for Searches.
- Rearrange window layout / save layouts.

Organise Your Assets



- Create a folder structure that works for you.
- Tidy up your scene assets.
- Make sure your prefabs are correctly linked.

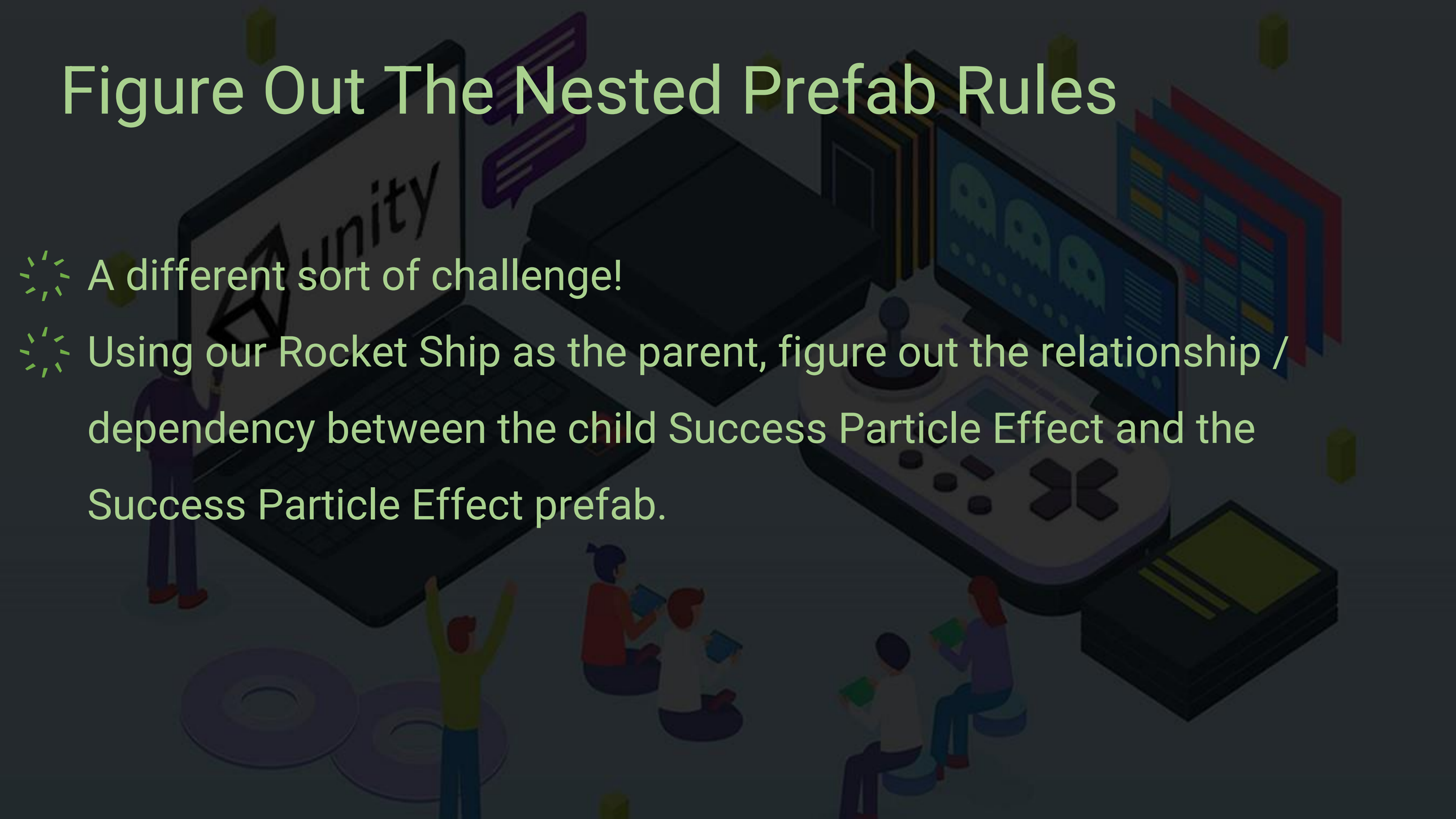
Main Directional Light (Sun)



Add Scene Lighting

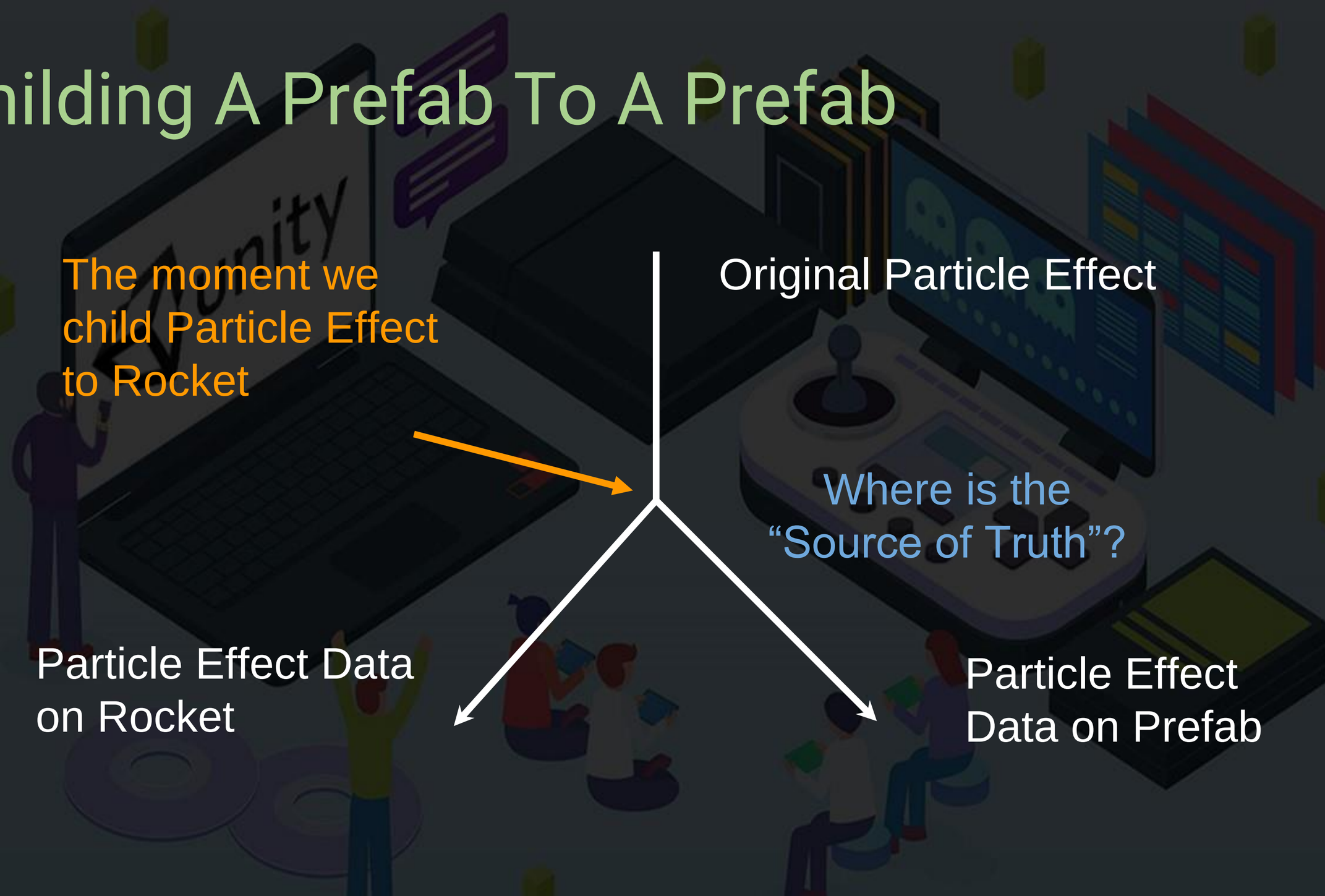
- Alter your scene’s main directional light to make your scene feel different.
- Add at least one scene light.

Figure Out The Nested Prefab Rules

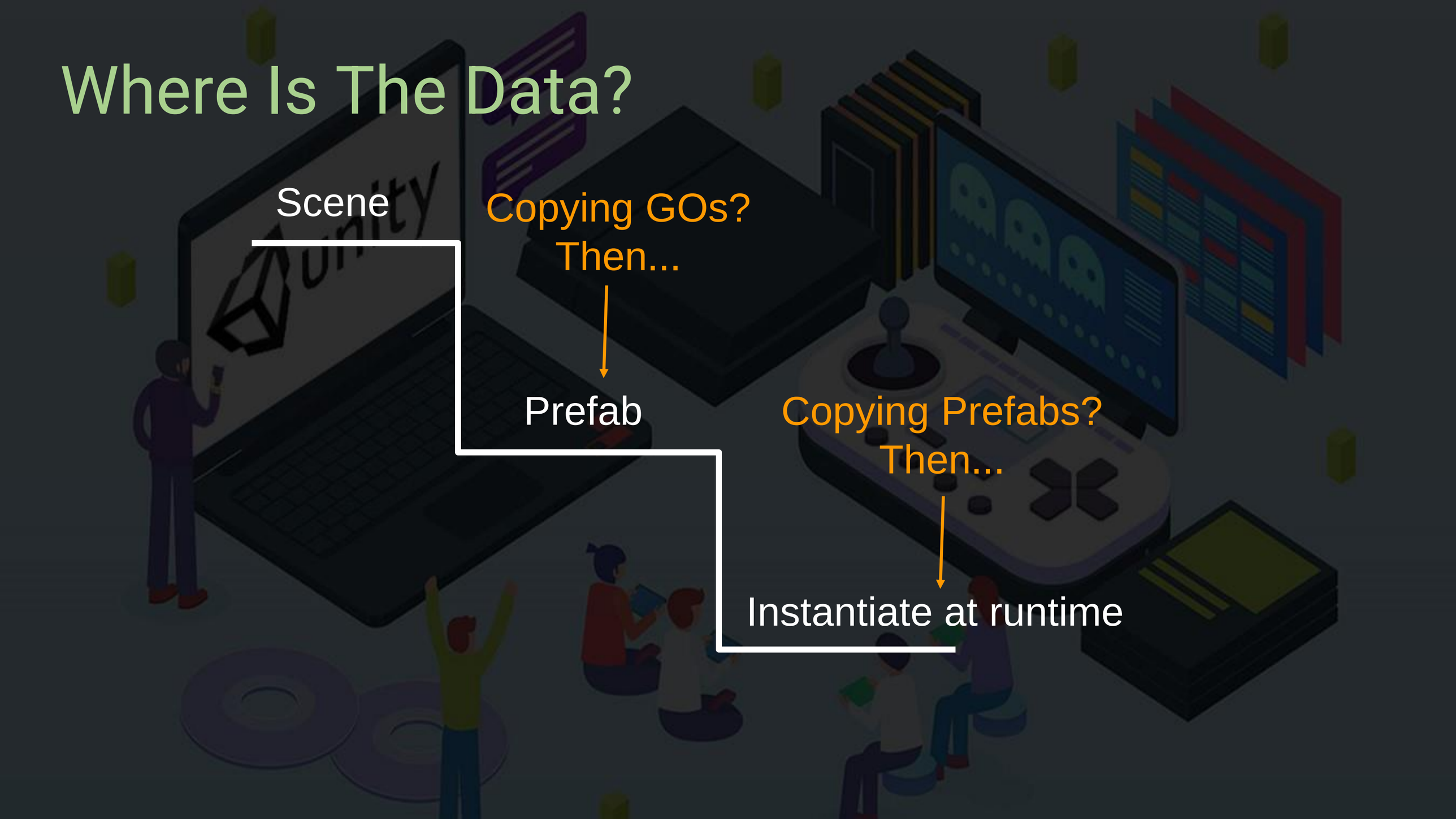


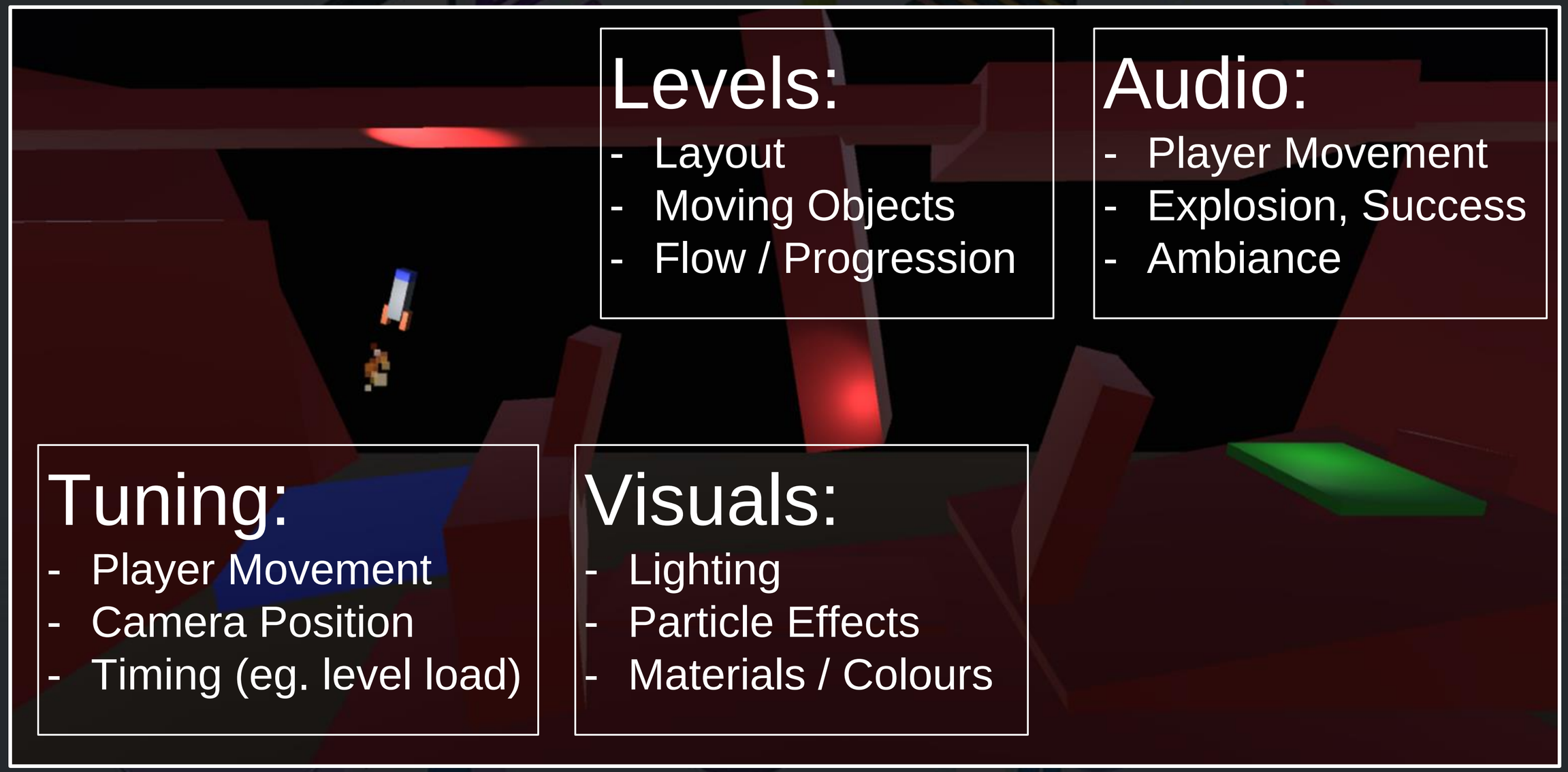
- A different sort of challenge!
- Using our Rocket Ship as the parent, figure out the relationship / dependency between the child Success Particle Effect and the Success Particle Effect prefab.

Childing A Prefab To A Prefab



Where Is The Data?





Levels:

- Layout
- Moving Objects
- Flow / Progression

Audio:

- Player Movement
- Explosion, Success
- Ambiance

Tuning:

- Player Movement
- Camera Position
- Timing (eg. level load)

Visuals:

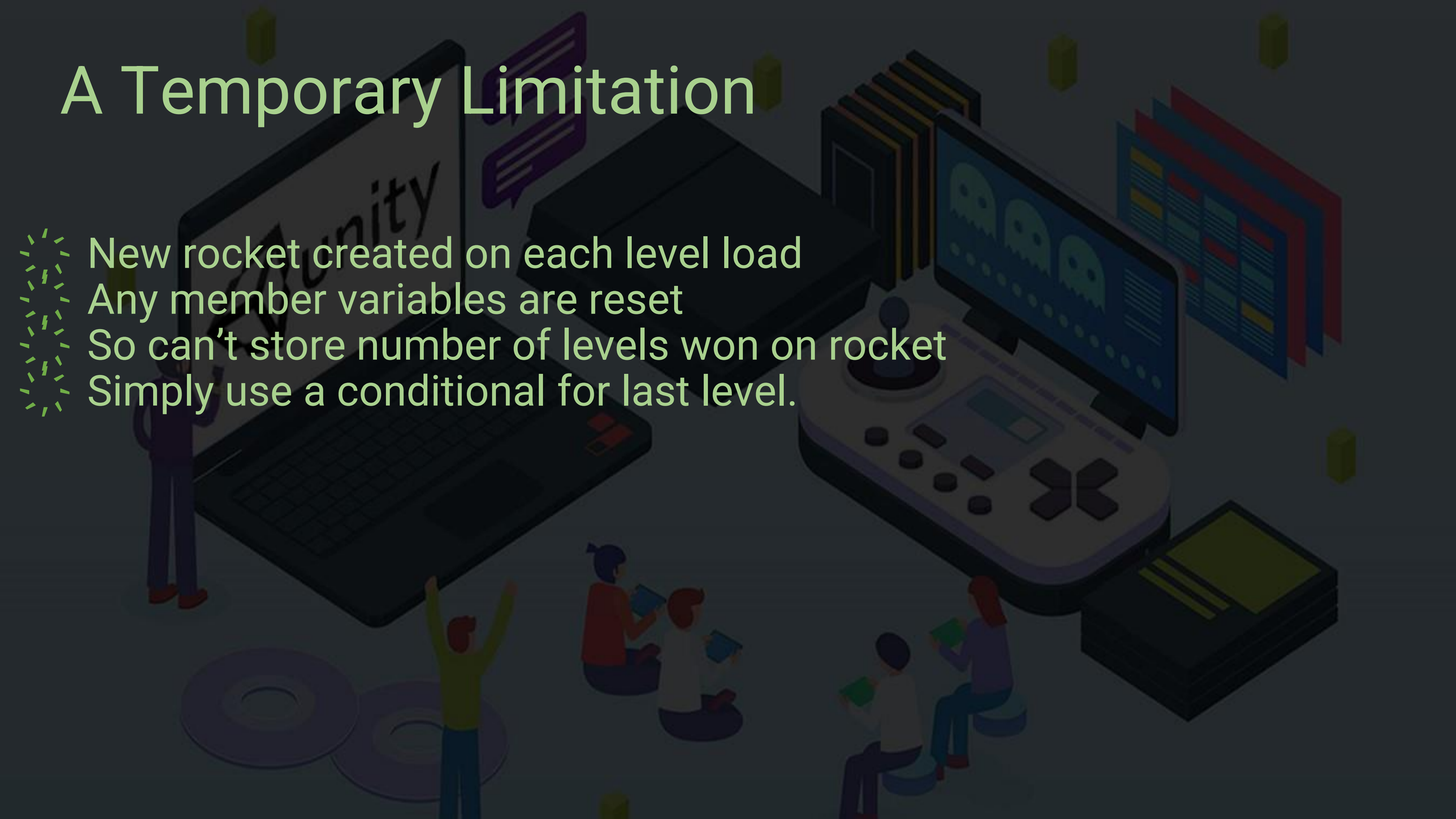
- Lighting
- Particle Effects
- Materials / Colours

Level Flow And Variety

- We need to keep our players engaged for as long as we can
- Two options with our current game:
  - Randomised levels of similar challenge level
  - Sequential levels of increasing challenge level

Looping Through Levels

A Temporary Limitation



- New rocket created on each level load
- Any member variables are reset
- So can’t store number of levels won on rocket
- Simply use a conditional for last level.

Get The Levels Cycling

- Your game should loop around all levels in the current build order. When you get to the last level return to the first.