

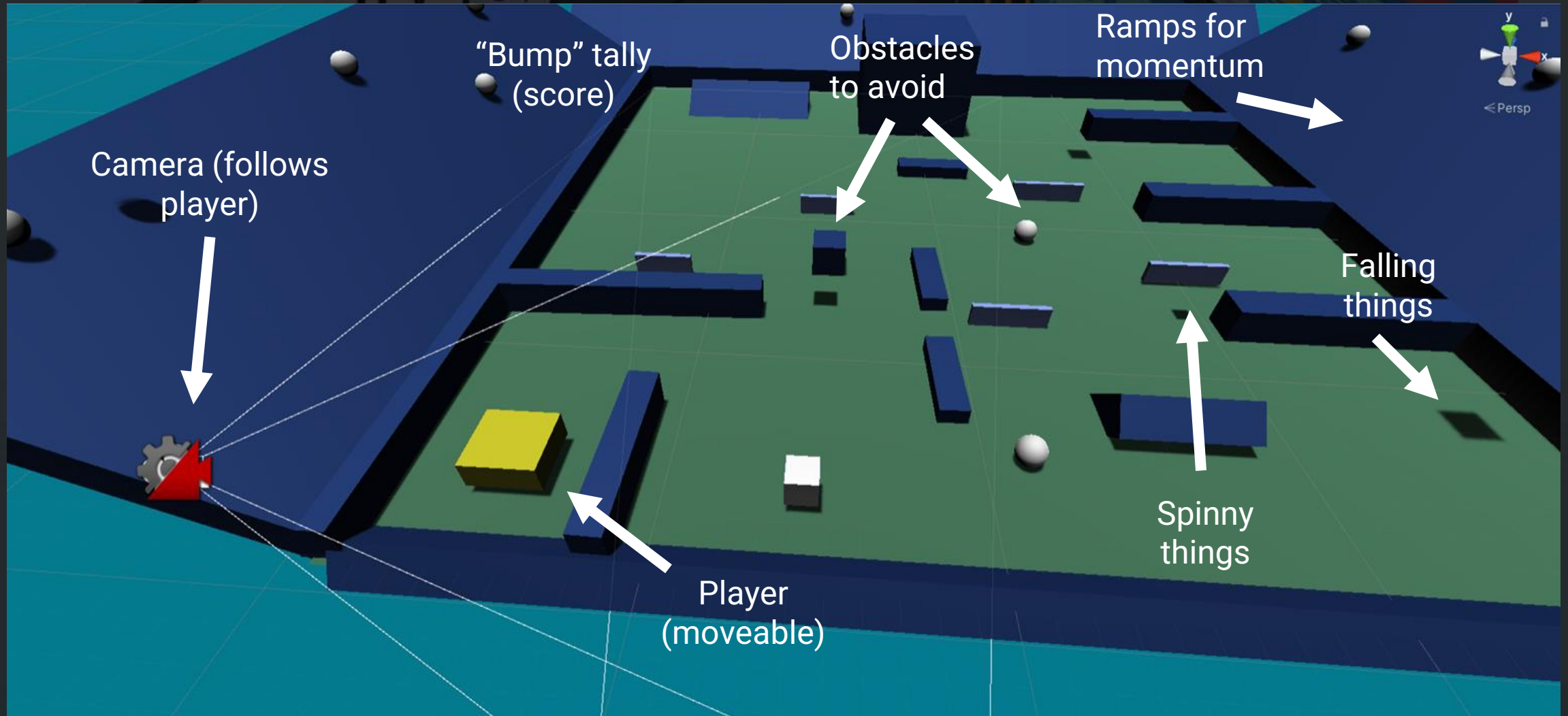
Obstacle Course

An isometric illustration depicting a digital obstacle course. In the center-left, a large laptop screen displays the Unity logo. To its right is a stack of dark grey rectangular blocks. Further right, a monitor shows three blue ghost-like icons, with a stack of colorful folders behind it. In the foreground, a large white and purple game controller sits on the floor. To the right of the controller is a stack of dark grey rectangular blocks with a green square on top. In the bottom left, a person in a purple shirt and blue pants stands next to the laptop. In the bottom center, a person in a green shirt and blue pants stands with arms raised. To the right of them, two people sit on the floor, one holding a blue tablet. In the bottom right, two more people sit on the floor, one holding a green tablet. The background is light blue with several small green rectangular blocks floating in the air.

Game Design

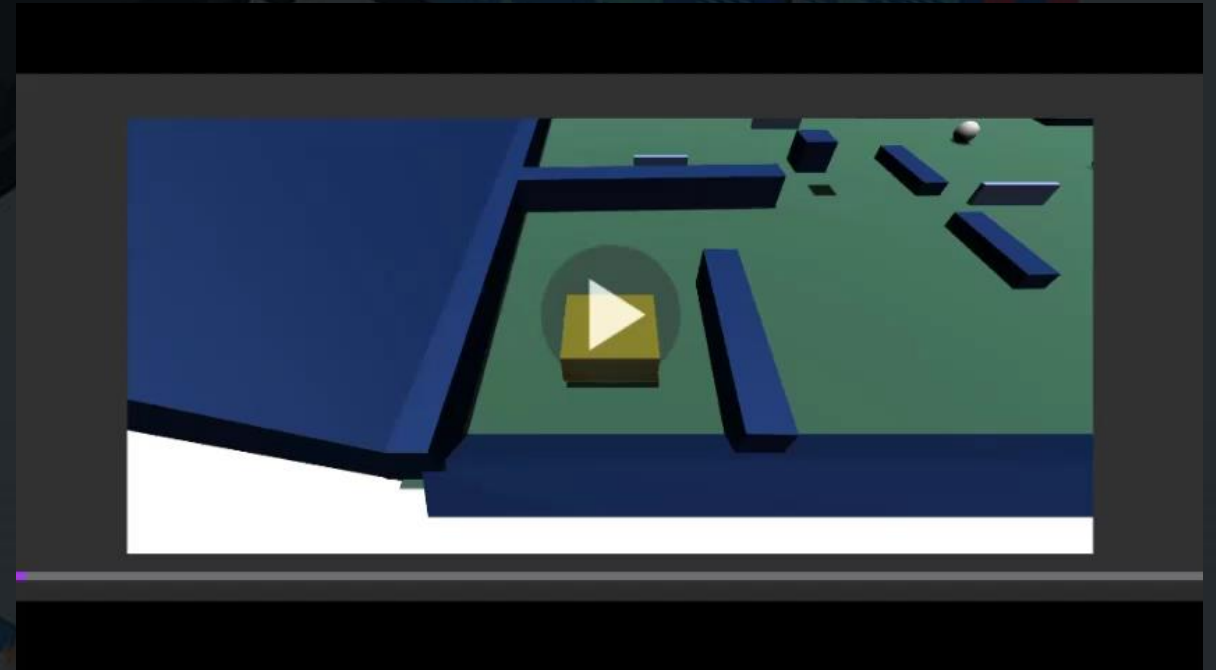
An isometric illustration depicting various aspects of game design. On the left, a large laptop displays the Unity logo. A person stands next to it, holding a smartphone. In the center, there are stacks of books or documents. To the right, a computer monitor shows a game interface with three green skulls. Below the monitor is a large, detailed joystick controller. In the foreground, several people are sitting on the floor, some holding tablets or books, appearing to be in a collaborative design session. The background is dark with some floating yellow cubes.

Core Gameplay Overview



Game Design

- ⚙️ Player Experience:
 - ⚙️ Nimble / agile
- ⚙️ Core Mechanic:
 - ⚙️ Move & dodge obstacles
- ⚙️ Game Loop:
 - ⚙️ Get from A to B

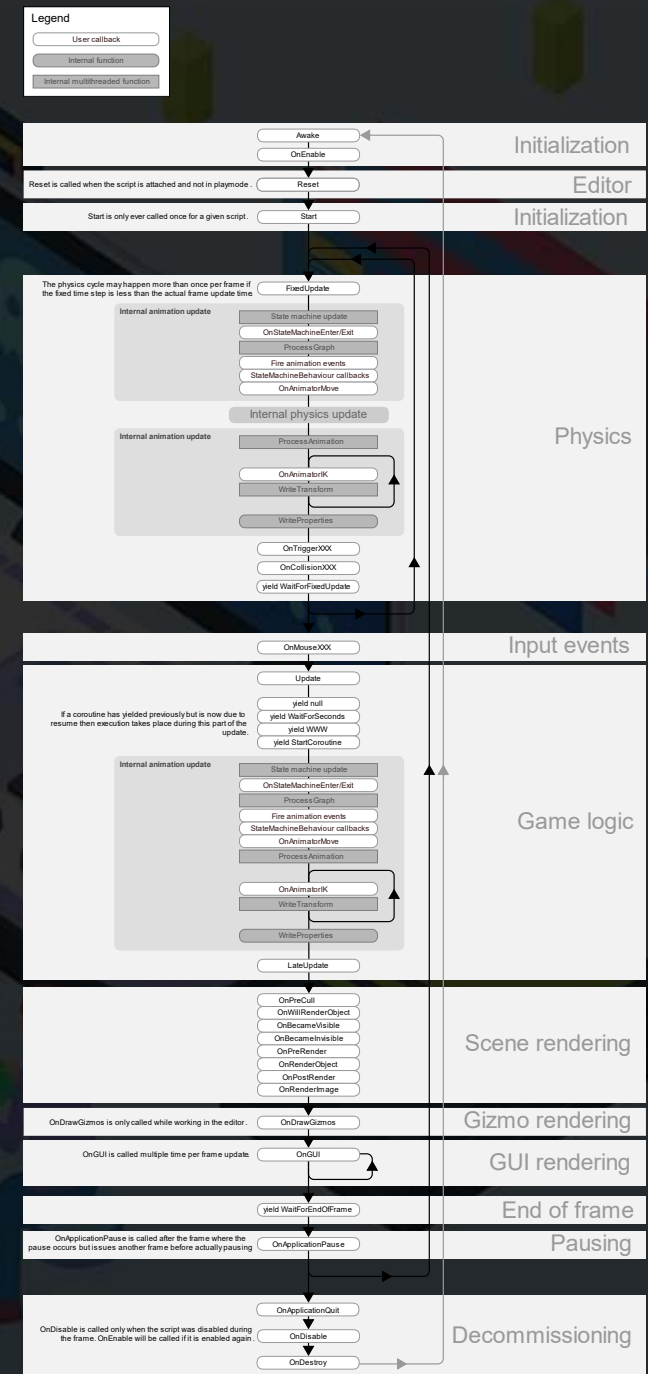


An isometric illustration of a game development event. On the left, a large laptop displays the Unity logo. A person stands next to it, holding a smartphone. In the center, there are stacks of books or documents. To the right, a large joystick controller is prominent, with a monitor displaying a game with three skulls. Further right, there are more stacks of books and a monitor showing a document. In the foreground, several people are sitting on the floor, some holding books or tablets. A person in the bottom center is standing with their arms raised. The background is dark with some floating yellow cubes.

Event Functions

Unity Order of Execution

- ☼ Awake
 - ☼ Called when a scene starts.
- ☼ Start
 - ☼ Called before the first frame update
- ☼ FixedUpdate
 - ☼ Called more frequently then Update
 - ☼ Can be called multiple times per frame
 - ☼ All physics calculations are made after Fixed Update
- ☼ Update
 - ☼ Called per frame
 - ☼ Main workhorse of a frame
- ☼ LateUpdate
 - ☼ Called once per frame after Update has finished.



Tutorial 1

- ✧ Create a new Unity 3D project (3D URP Core)
- ✧ Add a ground plane
- ✧ Create your “player”
- ✧ Rename your player
- ✧ Create a C# script “PlayerMovement”
- ✧ Add the script as a component to your “player” Gameobject

An isometric illustration on a dark background featuring various gaming and development elements. On the left, a large laptop displays the Unity logo. A person stands next to it holding a smartphone. In the center, there are stacks of dark rectangular blocks. To the right, a large game controller is positioned next to a monitor showing three blue skulls. Behind the monitor are several colorful document icons. In the foreground, a group of people are sitting on the floor, some holding tablets. A person stands with their arms raised near two large circular discs. Small yellow cubes are scattered throughout the scene.

Moving the Player

Transform.Translate

⚙ Declaration

- ⚙ `public void Translate(Vector3 translation);`
- ⚙ `public void Translate(float x, float y, float z);`

⚙ Description

- ⚙ Moves the transform in the direction and distance of translation.

Vector3

☀ Description

- ☀ Representation of 3D vectors and points.
- ☀ This structure is used throughout Unity to pass 3D positions and directions around.
- ☀ It also contains functions for doing common vector operations.

Tutorial 2

☀ Create three variables

- ☀ Create variables for the **x**, **y**, and **z** values.
- ☀ Change the values so your player move in the **x**, and **z** direction
- ☀ Change the value of **y** so the player flies up.
- ☀ Use **transform.Translate** to move the “player”
 - ☀ This method adds a value to the existing value of any **position variables** (x,y,z) of Transform component.

Tutorial 3

- ⚙️ Add some colors
- ⚙️ Create **Materials** for your Gameobjects
- ⚙️ Add the material to the **MeshRenderer** component



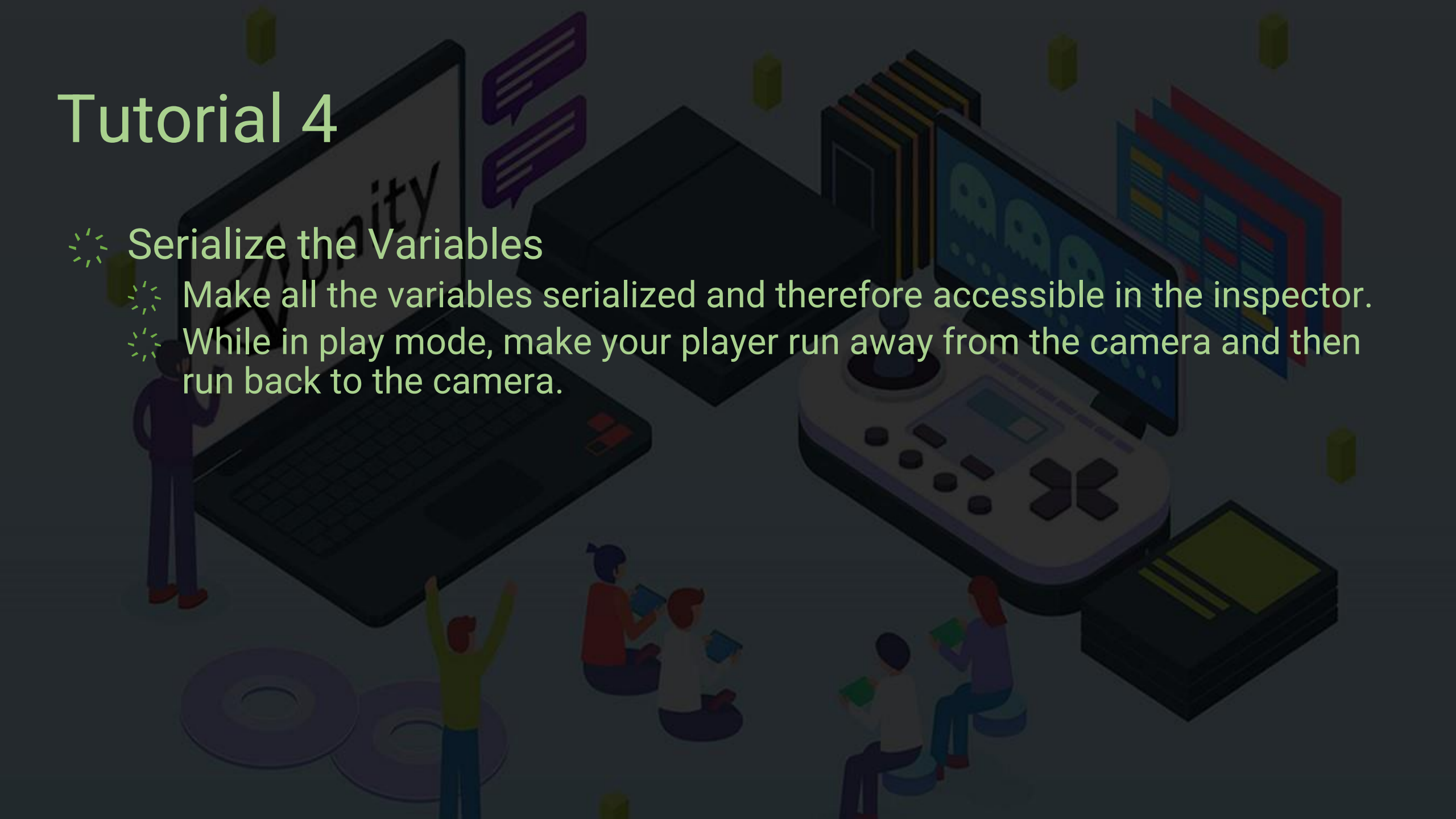
An isometric illustration of a game development environment. On the left, a large laptop displays the Unity logo. A person stands next to it, holding a smartphone. In the center, there are stacks of books or documents. To the right, a desktop monitor shows a game interface with three skulls, and a large game controller is in front of it. Another monitor to the right shows a data table. At the bottom, several people are sitting on the floor, some holding tablets. The background is dark with floating yellow cubes.

SerializeField

Tutorial 4

☀ Serialize the Variables

- ☀ Make all the variables serialized and therefore accessible in the inspector.
- ☀ While in play mode, make your player run away from the camera and then run back to the camera.



User Input



Tutorial 5

- ☀ Open Input Manager

- ☀ In file menu **Edit -> Project Settings -> Input Manager**

- ☀ Check the Vertical and Horizontal Axes

- ☀ Add Vertical Axis

- ☀ Update one of our variables so we are moving our player **forward** and **backward** (along the ground plane, not flying in the air).

- ☀ Add Horizontal Axis

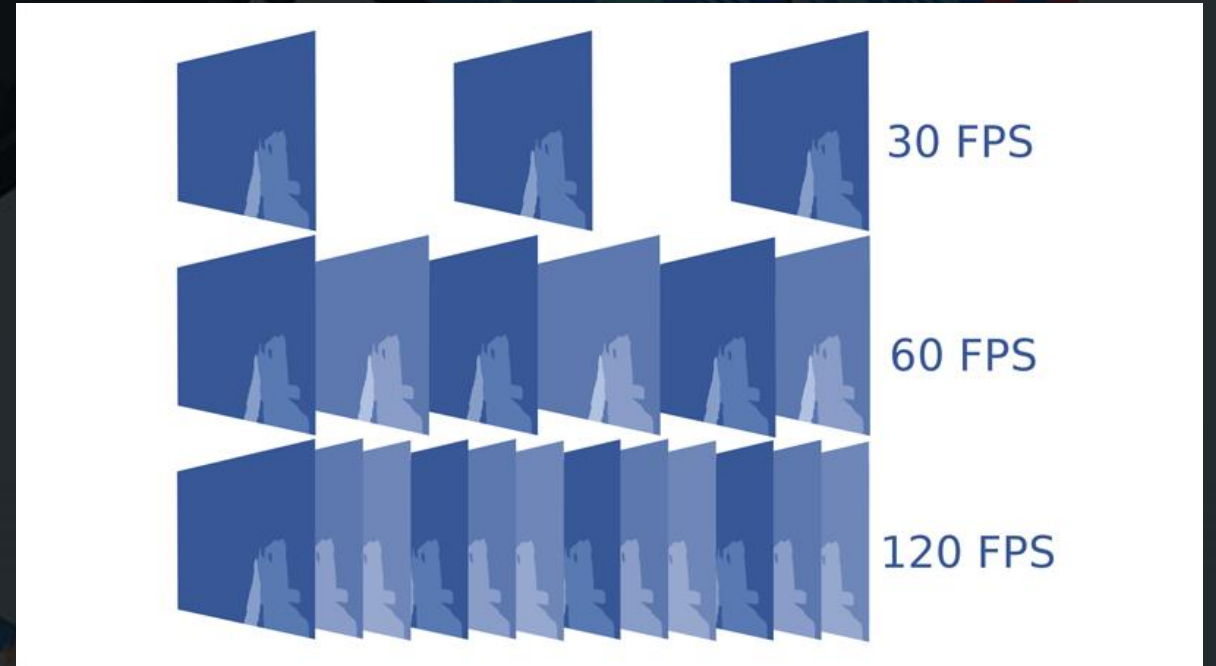
- ☀ Update one of our variables so we are moving our player **left** and **right** (along the ground plane, not flying in the air).

Framerate Independence

An isometric illustration on a dark background featuring various gaming and development elements. On the left, a large laptop displays the Unity logo and the word 'unity'. A person stands next to it. In the center, there are stacks of books or documents. To the right, a computer monitor shows three skull icons, with a stack of books behind it. In the foreground, a large game controller is prominent. At the bottom, several people are sitting on the floor, some holding books or tablets, and one person is standing with arms raised. There are also some circular objects resembling CDs or DVDs on the floor.

Framerate

- ⚙ Your frame rate, measured in frames per second (fps), describes how smoothly a given game runs on your PC.
- ⚙ The more frames you can pack into one second, the smoother on-screen motion will appear.
- ⚙ Lower frame rates—typically frame rates lower than 30fps or so—will appear choppy or slow.



Using Time.deltaTime

- ⚙ Using **Time.deltaTime** Unity can tell us how long each frame took to execute.
- ⚙ When we multiply something by **Time.deltaTime** it makes our game “frame rate independent”.
 - ⚙ i.e. The game behaves the same on fast and slow computers

On Update (each frame) move 1 unit to the left

Slow
Computer

Fast
Computer

Frames per second

10

100

Duration of frame

0.1s

0.01s

Distance per second

$1 \times 10 \times 0.1 = 1$

$1 \times 100 \times 0.01 = 1$

Tutorial 6

☀ Multiply By A Speed Variable

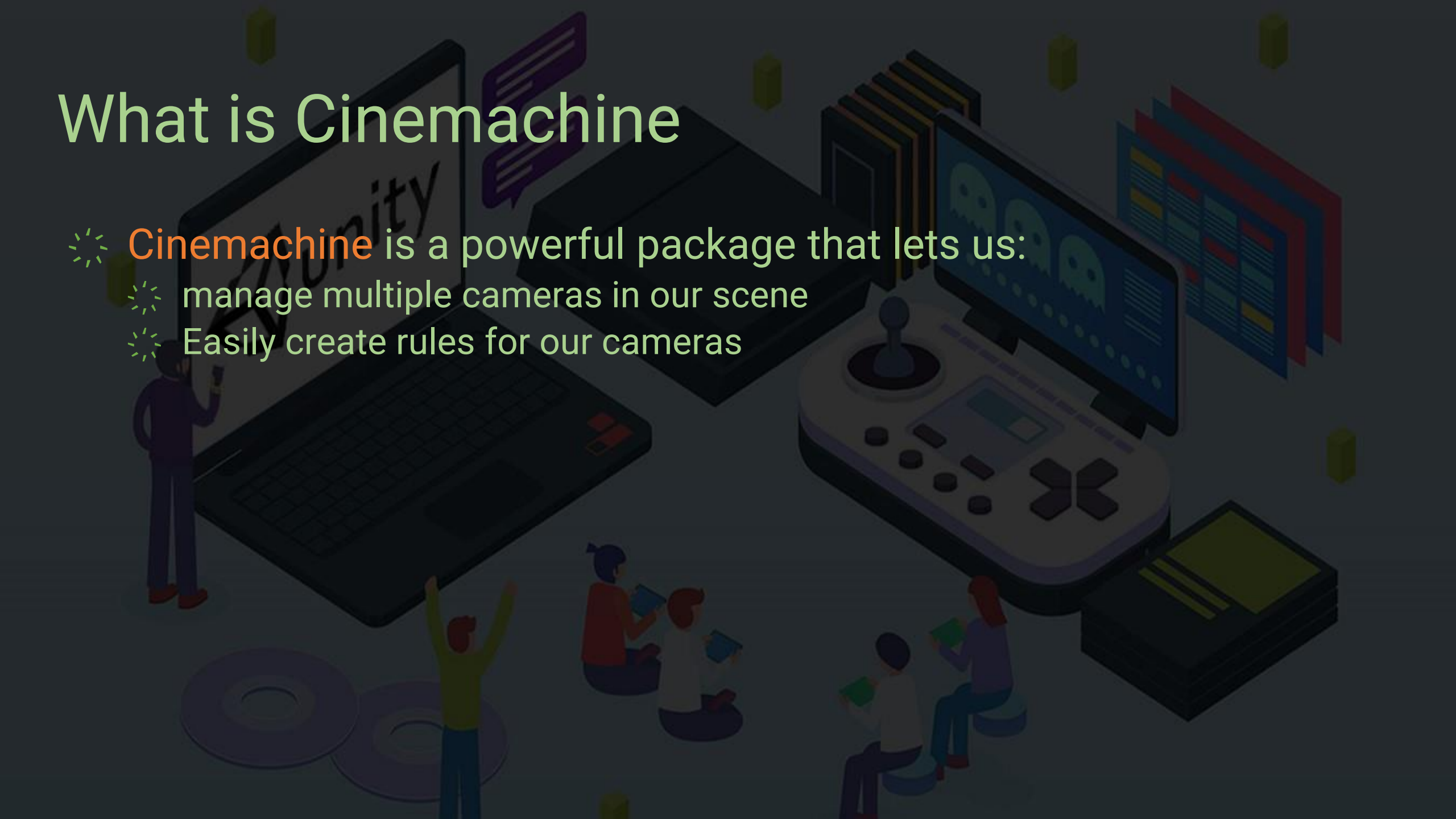
- ☀ Create a new variable called **moveSpeed**. The value of **moveSpeed** does not need to update each frame.
- ☀ Make it available in the inspector.
- ☀ Multiply your **xValue** and **zValue** by **moveSpeed**.
- ☀ Tune your player movement (as best you can for now).

An isometric illustration depicting a game development workflow. On the left, a large laptop displays the Unity logo. A person stands next to it, holding a smartphone. In the center, there are stacks of dark rectangular blocks. To the right, a large game controller is shown with a monitor attached, displaying three blue skull icons. Behind the controller are several colorful document icons. In the foreground, a group of people are sitting on the floor, some holding tablets or books. A person stands with their arms raised. The background is dark with several small yellow cubes floating in the air.

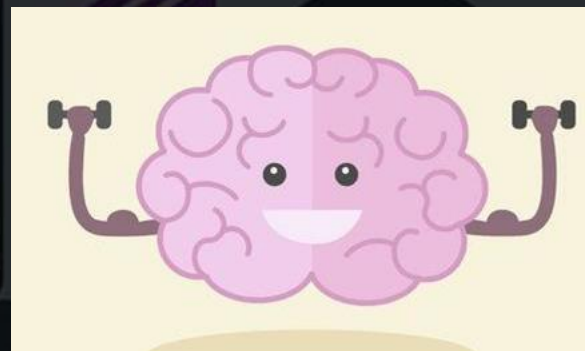
Cinemachine

What is Cinemachine

- ⚙️ **Cinemachine** is a powerful package that lets us:
 - ⚙️ manage multiple cameras in our scene
 - ⚙️ Easily create rules for our cameras



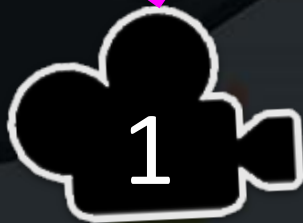
Cinemachine
Brain



Main Camera



Virtual
Cameras



Tutorial 7

- ⚙️ Open the **Package Manager** window
- ⚙️ Find and install **Cinemachine**
- ⚙️ Add **Cinemachine Brain** component to **main camera**
- ⚙️ Add a **Virtual Camera**
- ⚙️ Point it to follow the Player
- ⚙️ Tune the distance
- ⚙️ Feel free to play around with the other settings

Collisions

An isometric illustration on a dark background featuring various gaming and development elements. On the left, a large laptop displays the Unity logo. A person stands next to it holding a smartphone. In the center, there are stacks of books or documents. To the right, a large game controller is positioned next to a monitor showing three skull icons. Further right are more stacks of documents. In the foreground, several people are sitting on the floor, some holding tablets or books. A person stands with arms raised near two large circular objects. Small yellow cubes are scattered throughout the scene.

Basics of Collision

- ☀ Describes a collision.
- ☀ Collision information is passed to `Collider.OnCollisionEnter`, `Collider.OnCollisionStay` and `Collider.OnCollisionExit` events.
- ☀ `Collision other` parameter is used to get the collision data associated with the collision event.
 - ☀ The Collision class contains information, for example, about contact points and impact velocity.

Basics of Collision

☀ OnCollisionEnter(Collision other)

- ☀ OnCollisionEnter is called when this collider/rigidbody has begun touching another rigidbody/collider.

☀ OnCollisionExit(Collision other)

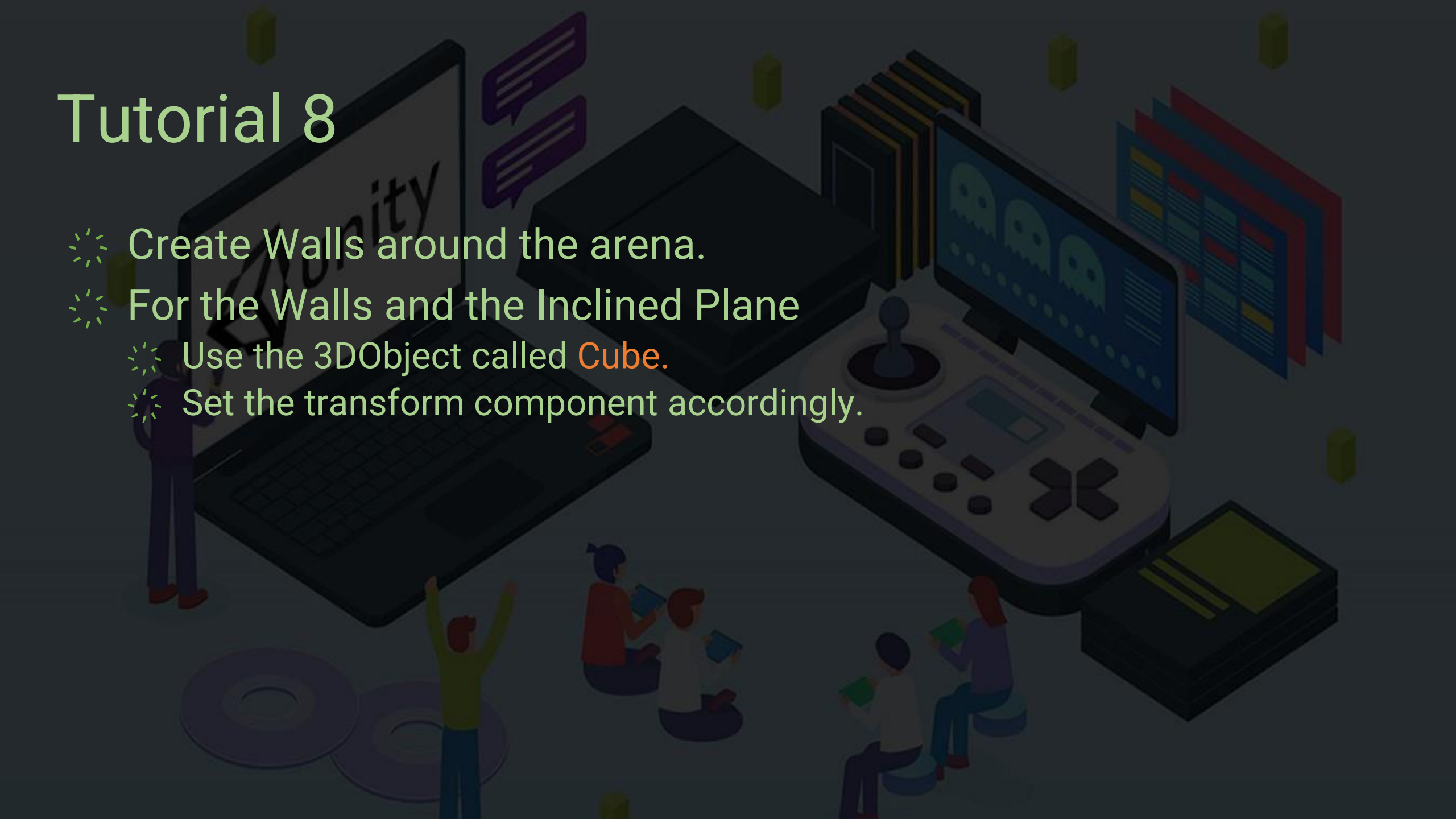
- ☀ OnCollisionExit is called when this collider/rigidbody has stopped touching another rigidbody/collider.

☀ OnCollisionStay(Collision other)

- ☀ OnCollisionStay is called once per frame for every Collider or Rigidbody that touches another Collider or Rigidbody.

Tutorial 8

- ✧ Create Walls around the arena.
- ✧ For the Walls and the Inclined Plane
 - ✧ Use the 3DObject called **Cube**.
 - ✧ Set the transform component accordingly.

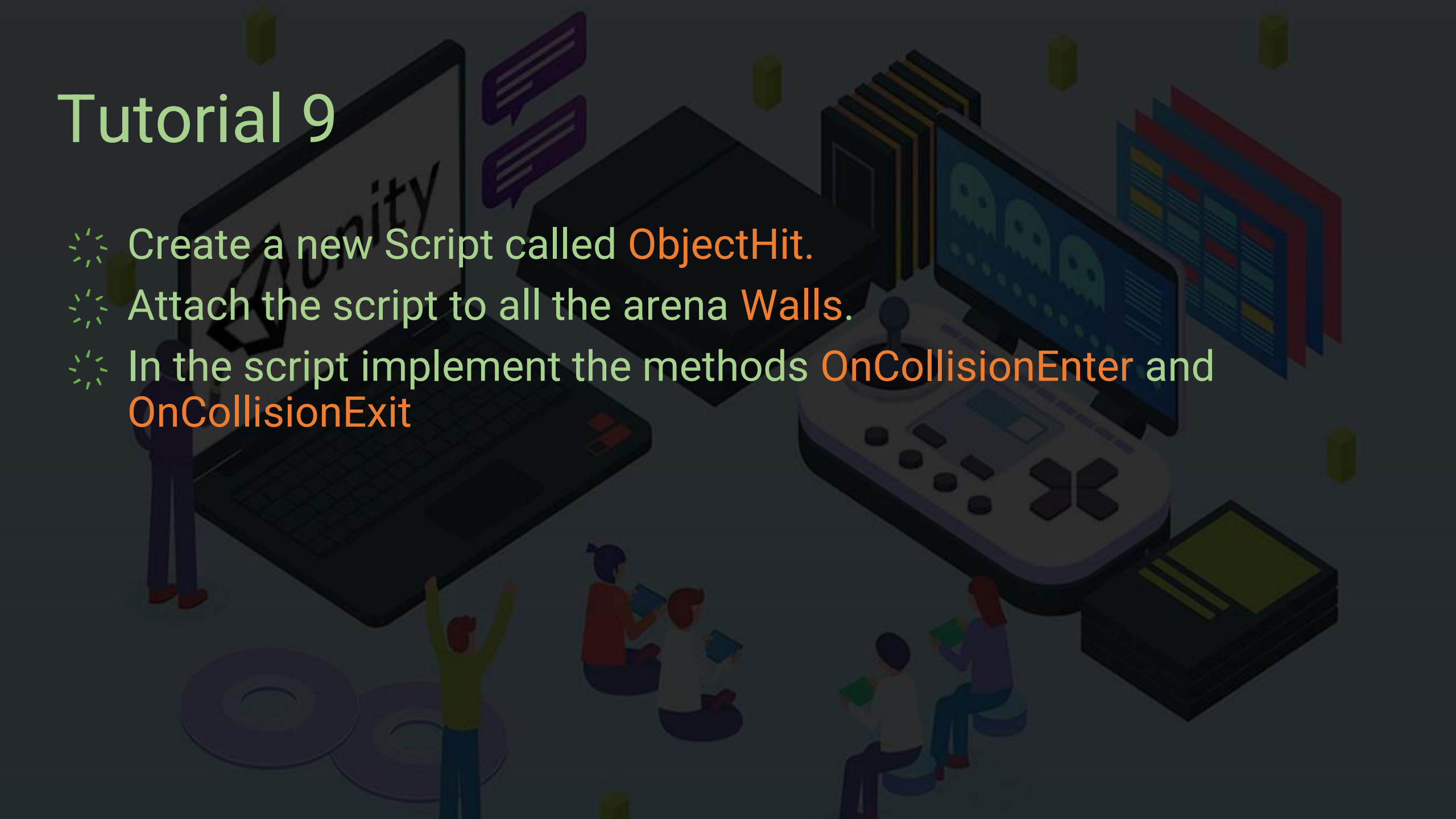


An isometric illustration on a dark background depicting various elements of game development. On the left, a large laptop displays the Unity logo and the word 'unity'. A person stands next to it, holding a smartphone. In the center, there are stacks of dark rectangular blocks. To the right, a large game controller is shown with a monitor displaying three blue skulls. Behind the controller are several colorful document icons. In the foreground, a group of people are sitting on the floor, some holding tablets or books. A person in the bottom center has their arms raised in celebration. The entire scene is surrounded by small, glowing yellow cubes.

Using Collision Events

Tutorial 9

- ⚙️ Create a new Script called **ObjectHit**.
- ⚙️ Attach the script to all the arena **Walls**.
- ⚙️ In the script implement the methods **OnCollisionEnter** and **OnCollisionExit**



Tags

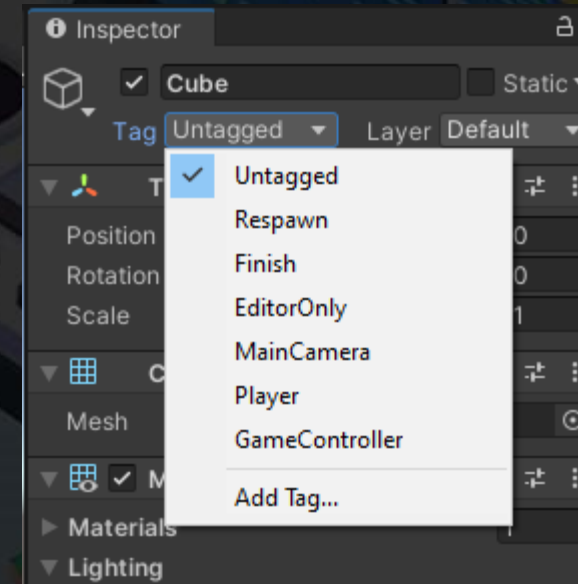
An isometric illustration of a gaming and development environment. On the left, a large laptop displays the Unity logo and the word 'unity'. A person stands next to it, holding a smartphone. In the center, there are stacks of books or documents. To the right, a large game controller is shown with a monitor displaying three skull icons. Further right, there are more stacks of books or documents. In the foreground, several people are sitting on the floor, some holding books or tablets. The background is dark with some floating yellow cubes.

Basics of Tags

- ⚙️ A tag is a reference word which you can assign to one or more GameObjects.
 - ⚙️ For example, you might define “Player” tags for player-controlled characters and an “Enemy” tag for non-player-controlled characters.
 - ⚙️ You might define items the player can collect in a Scene with a “Collectable” tag.
 - ⚙️ You can use any word OR phrase you want as a tag.
 - ⚙️ A GameObject can only have one tag assigned to it.
- ⚙️ Tags help you identify GameObjects for scripting purposes.
- ⚙️ Tags are useful in Collider control scripts.
 - ⚙️ For example, to determine if the player interacts with an enemy, a prop, or a collectable.

Tutorial 10

- ✨ Create new tags
 - ✨ The Inspector displays the Tag and Layer dropdown menus below the name of a GameObject.
 - ✨ Click **Add Tag...** to add list of new tags



- ✨ Create Tags for Walls, Boxy, Obstacle, Ball, Box

Caching a reference to a Component

An isometric illustration of a digital workspace. A large laptop on the left has the Unity logo and the word 'unity' on its screen. A person stands next to it holding a smartphone. To the right is a large game controller. In the background, there are stacks of books or documents and a monitor displaying three skull icons. In the foreground, several people are sitting on the floor, some holding tablets or books. The scene is dimly lit with small glowing cubes floating around.

GetComponent

⚙ Declaration

⚙ `public T GetComponent();`

⚙ Returns

⚙ `T` A reference to a component of the type `T` if one is found, otherwise null.

⚙ Description

⚙ Gets a reference to a component of type `T` on the specified `GameObject`.

Typical Usage

- ✧ The typical usage for this method is to call it on a reference to a different GameObject than the one your script is on.
 - ✧ For example:
 - ✧ `myResults = otherGameObject.GetComponent<ComponentType>()`
- ✧ However if you are writing code inside a MonoBehaviour class, you can omit the preceding GameObject reference to get a component from the same GameObject your script is attached to.
 - ✧ For example:
 - ✧ `myResults = GetComponent<ComponentType>()`

Tutorial 11

- ✧ Open the **ObjectHit** script
- ✧ Define variables
 - ✧ **MeshRenderer** meshRenderer;
 - ✧ **Color** originalColor;
- ✧ In **Start** method
 - ✧ Get a reference to the **MeshRenderer** component using the following statement.
 - ✧ **meshRenderer = GetComponent<MeshRenderer>();**
 - ✧ Save the current color of the GameObject
 - ✧ **originalColor = meshRenderer.material.color;**

Tutorial 12

- ✧ Open the **ObjectHit** script
- ✧ In **OnCollisionEnter** method do the following:
 - ✧ Check whether a collision has occurred with the Player.
 - ✧ `other.gameObject.CompareTag("Boxy")`
 - ✧ If Collision has occurred with the Player
 - ✧ Change the color to red.
 - ✧ `meshRenderer.material.color = Color.red;`

Tutorial 13

- ✧ Open the ObjectHit script
- ✧ In OnCollisionExit method do the following:
 - ✧ Check whether a collision has occurred with the Player.
 - ✧ `other.gameObject.CompareTag("Boxy")`
 - ✧ If Collision has exited with the Player
 - ✧ Change the color to red.
 - ✧ `meshRenderer.material.color = originalColor;`

An isometric illustration of a game development scene. On the left, a large laptop displays the Unity logo. A person stands next to it, holding a smartphone. In the center, there's a large, dark, rectangular block. To the right, a game controller is shown with a monitor displaying three skull icons. Behind the controller are several colorful document icons. In the foreground, a group of people are sitting on the floor, some holding tablets or books. A person in the bottom center is standing with arms raised. The background is dark with some floating yellow cubes.

Add Score

Tutorial 14

- ⚡ Create a new C# script called **Scorer**
- ⚡ Create a new **OnCollisionEnter()** method
- ⚡ When we hit something, print to the console, "You've bumped into a thing this many times:"
- ⚡ Create a variable **int hit = 0;**
- ⚡ Increment the variable when collision occurs.
- ⚡ Create some logic so the message only prints when player hits with Walls, Obstacles, Balls, and Boxes
 - ⚡ Remember the tags?
- ⚡ Don't forget to attach the script

Rigidbody

An isometric illustration on a dark background featuring various game development elements. On the left, a large laptop displays the Unity logo. A person stands next to it holding a smartphone. In the center, there are stacks of dark rectangular blocks. To the right, a large game controller is positioned next to a monitor showing three blue skull icons. Behind the controller are several colorful document icons. In the bottom left, two large circular discs are visible. In the bottom center, a group of five people are sitting on the floor, some holding tablets. The word 'Rigidbody' is written in a light green, sans-serif font across the middle of the image.

Rigidbody

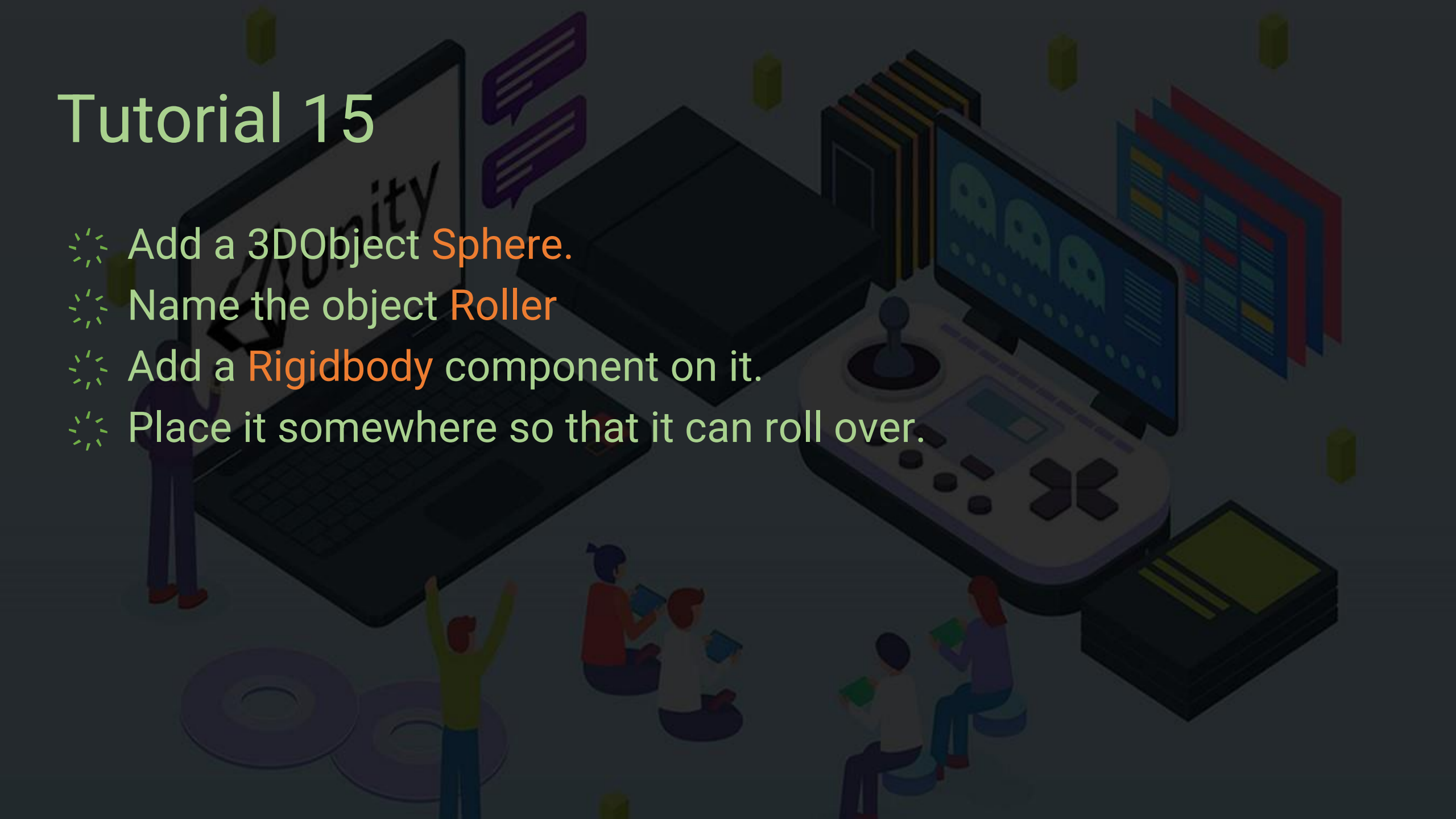
- ⚙ Control of an object's position through physics simulation.
- ⚙ Adding a **Rigidbody** component to an object will put its motion under the control of Unity's physics engine.
- ⚙ Even without adding any code, a **Rigidbody** object will be pulled downward by gravity and will react to collisions with incoming objects if the right Collider component is also present.
- ⚙ The **Rigidbody** also has a scripting API that lets you apply forces to the object and control it in a physically realistic way.
 - ⚙ For example, a car's behaviour can be specified in terms of the forces applied by the wheels.
- ⚙ In a script, the **FixedUpdate** function is recommended as the place to apply forces and change **Rigidbody** settings (as opposed to **Update**, which is used for most other frame update tasks).

Create Obstacles

An isometric illustration on a dark background depicting the game development process. On the left, a large laptop displays the Unity logo. A person stands next to it, holding a smartphone. In the center, there are stacks of books or documents. To the right, a desktop monitor shows a game with three blue skulls, and a large, detailed joystick controller sits in front of it. Further right, another monitor displays a data table. In the foreground, several people are sitting on the floor, some holding tablets or books, while one person stands with arms raised. The scene is decorated with small yellow cubes and purple speech bubbles.

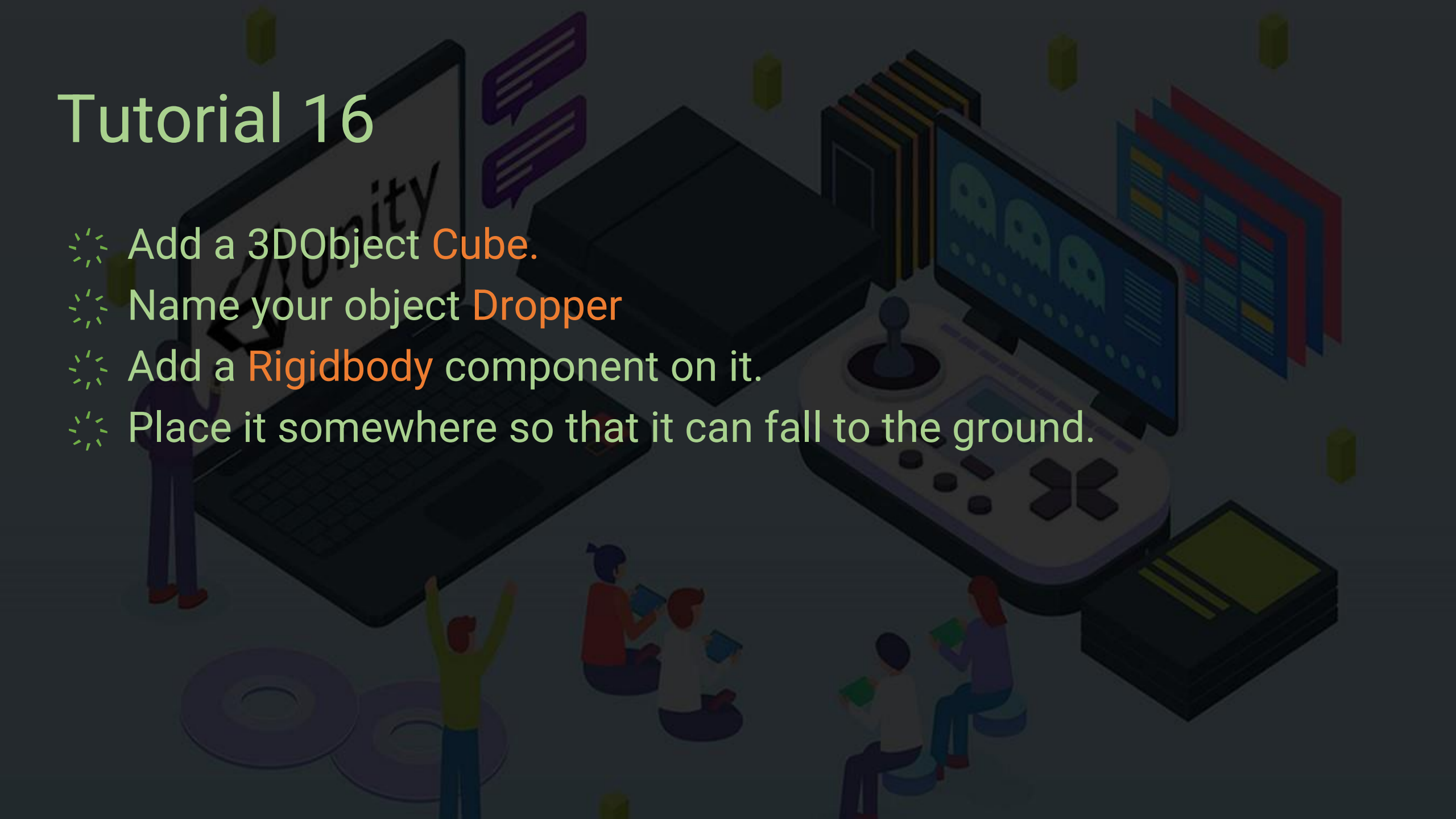
Tutorial 15

- ⚙️ Add a 3DObject **Sphere**.
- ⚙️ Name the object **Roller**
- ⚙️ Add a **Rigidbody** component on it.
- ⚙️ Place it somewhere so that it can roll over.



Tutorial 16

- ⚙️ Add a 3DObject **Cube**.
- ⚙️ Name your object **Dropper**
- ⚙️ Add a **Rigidbody** component on it.
- ⚙️ Place it somewhere so that it can fall to the ground.



An isometric illustration on a dark background depicting the game development process. On the left, a large laptop displays the Unity logo. A person stands next to it holding a smartphone. In the center, there are stacks of dark rectangular blocks. To the right, a large joystick controller sits in front of a monitor showing three blue skulls. Behind the monitor are several colorful document pages. In the foreground, a group of people are gathered: one person has their arms raised, while others are sitting on the floor looking at tablets. Two large, light-colored circular discs are on the floor to the left. Small yellow cubes are scattered throughout the scene.

Using Time.time

Problem & Solution

Problem to solve:

- ☀ Make an object fall after 3 seconds has passed

Solution:

1. A timer - **Time.time**
2. A mechanism to “do a thing if 3 seconds has elapsed” - **if statement**
3. A way to start the object falling after 3 seconds - **disable / enable gravity**

Tutorial 17

- ⚙️ Create a new C# script **Dropper**
- ⚙️ On every frame, print out to the console how much time has elapsed since the game started.
- ⚙️ HINT: Use **Time.time** within debug.log
- ⚙️ Add your script to the Box Game Object

Tutorial 18

- ⚙️ Update the C# script **Dropper**
- ⚙️ Use a variable for the time to wait that can be easily changed in the Inspector.
- ⚙️ Cache a reference of the **Rigidbody**.
 - ⚙️ `Rigidbody rBody = GetComponent<Rigidbody>();`
- ⚙️ Disable the **gravity**
 - ⚙️ `rBody.useGravity = false;`
- ⚙️ Enable the **gravity** if **timeToWait** has elapsed
 - ⚙️ `rBody.useGravity = true;`

An isometric illustration of a game development scene. On the left, a large laptop displays the Unity logo. A person stands next to it, holding a smartphone. In the center, there's a large, dark, rectangular block. To the right, a monitor shows three green skulls, and a joystick is positioned in front of it. Further right, a stack of colorful documents or codebooks is visible. In the foreground, several people are sitting on the floor, some holding tablets or books. The background is dark with some floating yellow cubes.

Rotate an Object

Transform.Rotate

⚙ Declaration

- ⚙ `public void Rotate(Vector3 eulers);`
- ⚙ `public void Rotate(float xAngle, float yAngle, float zAngle);`

⚙ Description

- ⚙ Use Transform.Rotate to rotate GameObjects in a variety of ways.
- ⚙ The rotation is often provided as an Euler angle.

Tutorial 19

- ✧ Create a 3DObject **Cube** and name it **Spinner**
- ✧ Create a new C# script **Rotator**.
- ✧ Create a variable for rotation speed so it can be updated from the inspector.
 - ✧ `float rotateSpeed = 3f;`
- ✧ In **Update** method
 - ✧ Rotate the Game Object using
 - ✧ `Transform.Rotate(0, rotateSpeed * Time.deltaTime, 0);`

An isometric illustration on a dark background depicting various elements of game development. On the left, a large laptop displays the Unity logo and the word 'unity'. A person stands next to it holding a smartphone. In the center, there are stacks of dark rectangular blocks. To the right, a large joystick controller sits in front of a monitor showing three green skull icons. Behind the monitor are several colorful document pages. In the bottom foreground, a group of people are sitting on the floor, some holding tablets or books, while one person stands with arms raised. Two large, light-colored circular discs are on the floor to the left. Small yellow cubes are scattered throughout the scene.

Trigger Collider

Basics of Trigger Collider

- ⚡ A trigger collider does not collide with other colliders; instead, other colliders pass through it.
- ⚡ OnTrigger events
 - ⚡ Trigger colliders don't cause collisions. Instead, they detect other colliders that pass through them, and call functions that you can use to initiate events.

Trigger Events

☀ `Collider.OnTriggerEnter`

- ☀ Unity calls this function on a trigger collider when it first makes contact with another collider.

☀ `Collider.OnTriggerStay`

- ☀ Unity calls this function on a trigger collider once per frame if it detects another Collider inside the trigger collider.

☀ `Collider.OnTriggerExit`

- ☀ Unity calls this function on a trigger collider when it ceases contact with another collider.

Tutorial 20

- ⚙️ Create a Finish Area with a Trigger
- ⚙️ Add a **Cube** to the specific location.
- ⚙️ Set Collider as **Trigger**
- ⚙️ Remove the **MeshRenderer** and **MeshFilter** components
- ⚙️ Create a new C# script **Finisher**
- ⚙️ In **OnTriggerEnter** method
 - ⚙️ Print a message when Player triggers the finish area

Prefabs

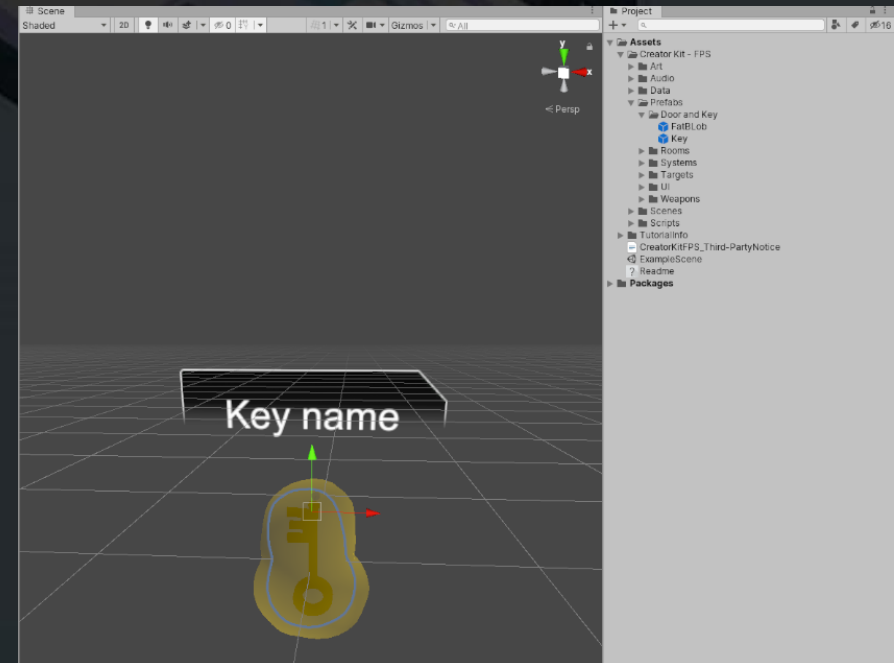
An isometric illustration on a dark background depicting various elements of game development. On the left, a large laptop displays the Unity logo and name. A person stands next to it, holding a smartphone. In the center, there are stacks of dark rectangular blocks. To the right, a large, stylized game controller is shown with a monitor on top displaying three green skull icons. Behind the controller are several colorful document icons. In the foreground, a group of people are sitting on the floor, some holding tablets or books. There are also two large circular discs on the floor. The word 'Prefabs' is written in a large, light green font across the center of the image.

Basics of Prefabs

- ✧ Unity's **Prefab** system allows you to create, configure, and store a **GameObject** complete with all its components, property values, and child **GameObjects** as a **reusable Asset**.
- ✧ The Prefab Asset acts as a **template** from which you can create new **Prefab instances** in the Scene.

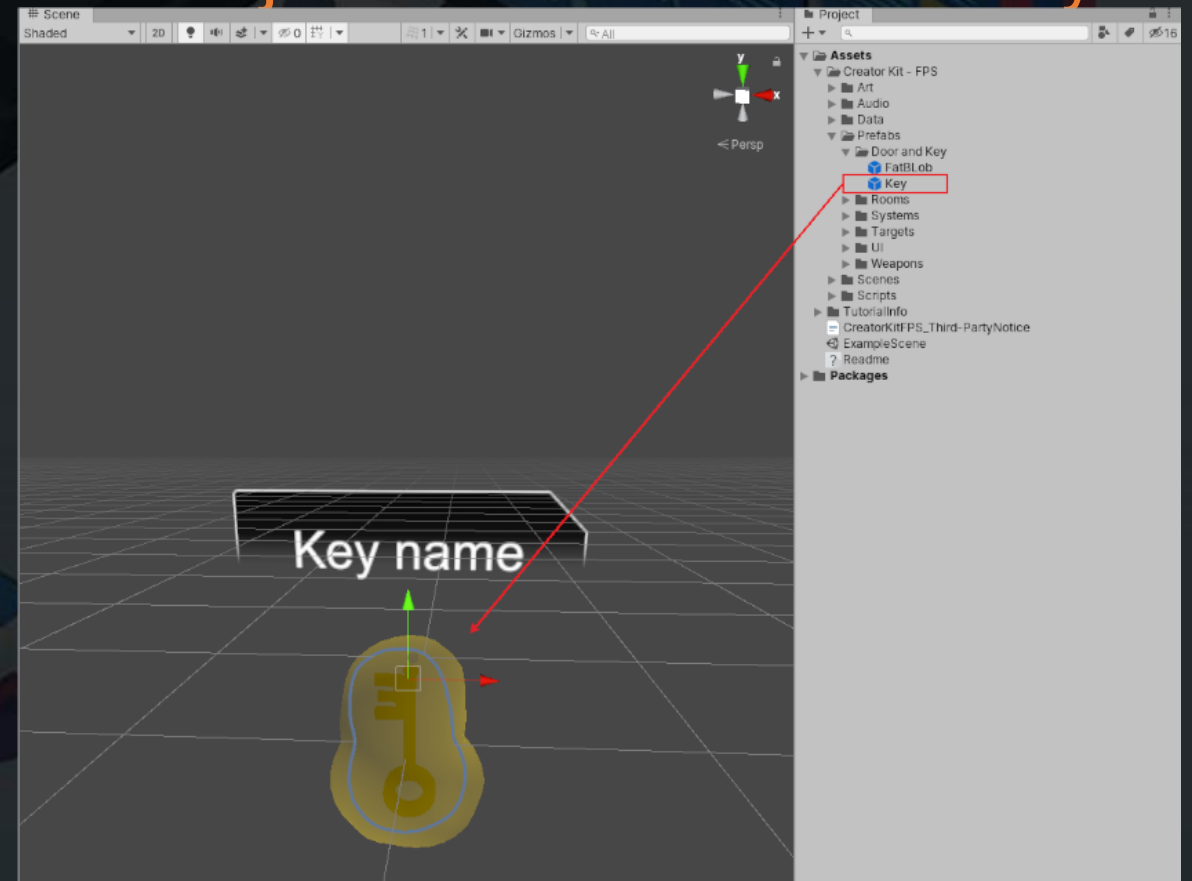
Create Prefab

- ⚙ To create a **Prefab Asset**, drag a GameObject from the **Hierarchy window** into the **Project window**.
- ⚙ Prefabs Assets in the Project window are shown with a **thumbnail view** of the GameObject, or the **blue cube Prefab icon**.



Create Prefab Instance

- ⚙ You can create instances of the Prefab Asset in the Editor by dragging the **Prefab Asset** from the **Project** view to the **Hierarchy** or **Scene** view.



Tutorial 21

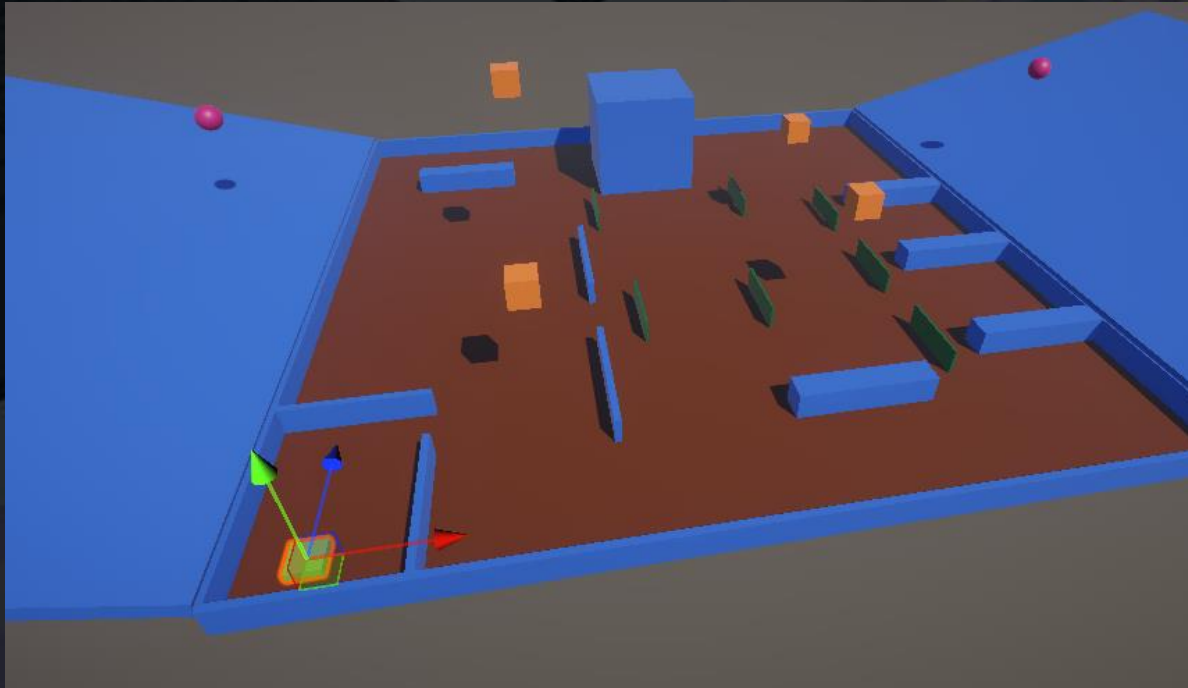
☼ Create Prefabs for the following GameObjects

- ☼ Boxy (Player)
- ☼ Wall
- ☼ Dropper
- ☼ Roller
- ☼ Spinner



Final Tutorial

- ✨ Create a fun Arena that makes the player go from A to B.
- ✨ Use your droppers, rollers and spinners to create interesting moments for the player.



Destroy Objects

An isometric illustration of a game development scene. On the left, a large laptop displays the Unity logo and the word 'unity'. A person stands next to it, holding a smartphone. In the center, there's a large, dark, rectangular object. To the right, a game controller is shown with a monitor displaying three skull icons. Further right, there are several colorful rectangular blocks. In the foreground, a group of people are sitting on the floor, some holding tablets or books. The background is dark with some floating yellow cubes.

Object.Destroy

⚙ Declaration

⚙ `public static void Destroy(Object obj, float t = 0.0F);`

⚙ Description

- ⚙ Removes a GameObject, component or asset.
- ⚙ The object obj is destroyed immediately after the current Update loop, or t seconds from now if a time is specified.
- ⚙ If obj is a Component, this method removes the component from the GameObject and destroys it.
- ⚙ If obj is a GameObject, it destroys the GameObject, all its components and all transform children of the GameObject.
- ⚙ Note: When destroying MonoBehaviour scripts, Unity calls OnDisable and OnDestroy before the script is removed.

Assignments

An isometric illustration depicting a game development workflow. On the left, a large laptop displays the Unity logo. A person stands next to it, holding a smartphone. In the center, there are stacks of dark blue blocks. To the right, a computer monitor shows a game interface with three green skulls and a progress bar. Next to the monitor is a stack of colorful documents. In the foreground, a large, detailed game controller is shown. To its right is a stack of yellow and green documents. At the bottom, several people are sitting on the floor, some holding tablets or books. One person is standing with their arms raised. The background is dark with some floating yellow cubes.

Lab Assignment

- ⚙ Complete this tutorial
- ⚙ Create a Finish Area
 - ⚙ User **Trigger Collider**
- ⚙ Take All the Balls to the Finish Area.
- ⚙ Score is given when ball is in the finish area.

Assignment

- ✨ Complete this Tutorial
- ✨ Add Pickups at the arena that boosts the player when picked up.
 - ✨ HINT:
 - ✨ Use **Trigger Collider**
 - ✨ Destroy the pickups using **Destroy()** method.
- ✨ Add a Hit Text as a child GameObject of Boxy (Player) that floats above the player and displays the current hits
 - ✨ Use **3dObject -> Text - TextMeshPro**