



Basic Scripting

Introduction to Game Development in
Unity
Spring 2026, 98-127, Lecture 2

Instructors: Jingxuan Chen, Dario Quintero, Shangyi Zhu, Jeffrey Wang



Table of Contents

- How do we detect input?
- How can we use Unity's physics engine?
- How can we control the camera?
- Combining everything into a simple player controller

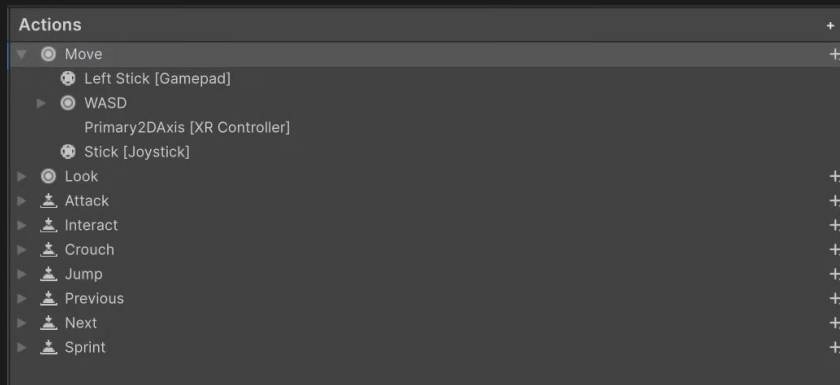


Detecting Input

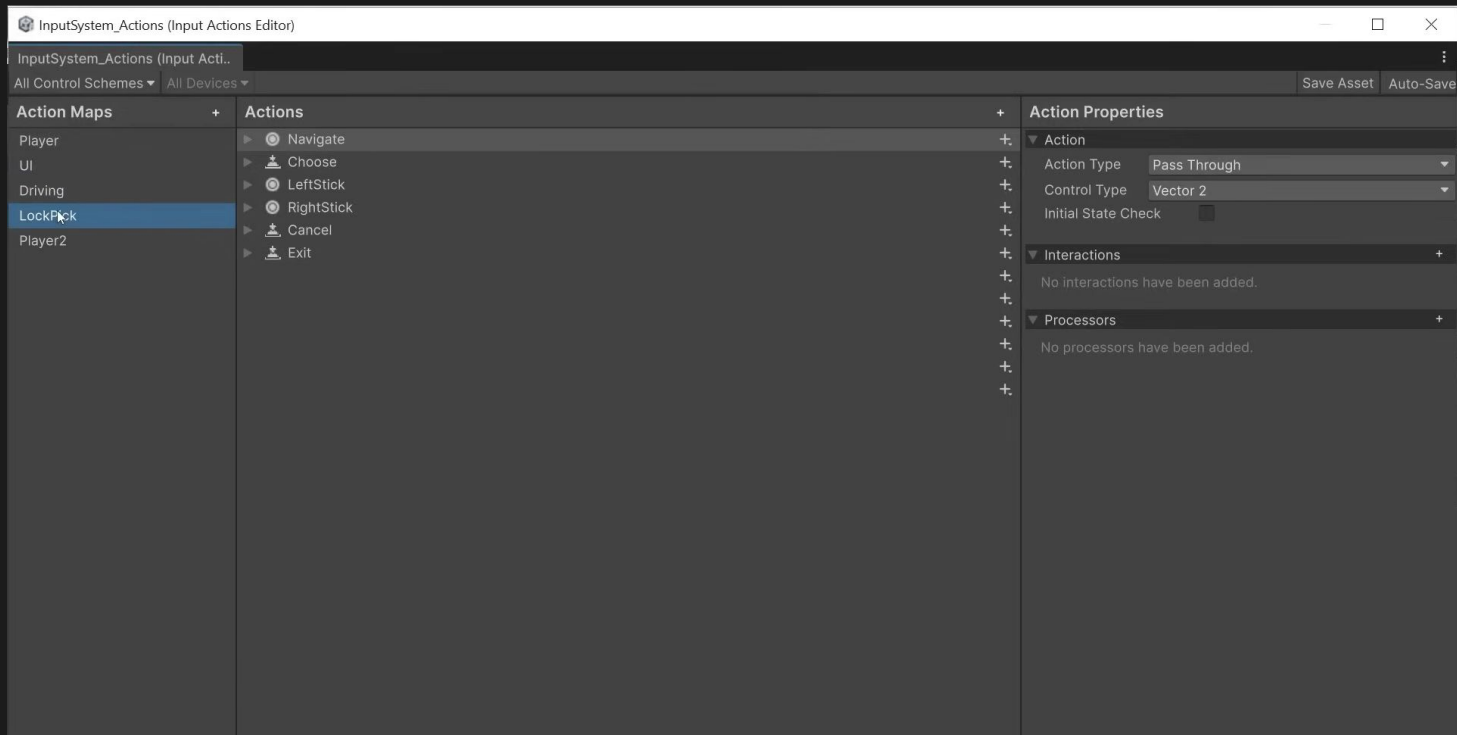


The Input Action Editor

- What is it?
 - Framework for handling input
 - Everything is categorized as an “Action”
- What is an Action?
 - Run, jump, shoot, interact, pause ...
 - Anything that is a consequence of player input
- Why Actions?
 - Organized
 - Modular
 - Easily support different input devices (keyboard and mouse, different controllers, etc)



The Input Action Editor

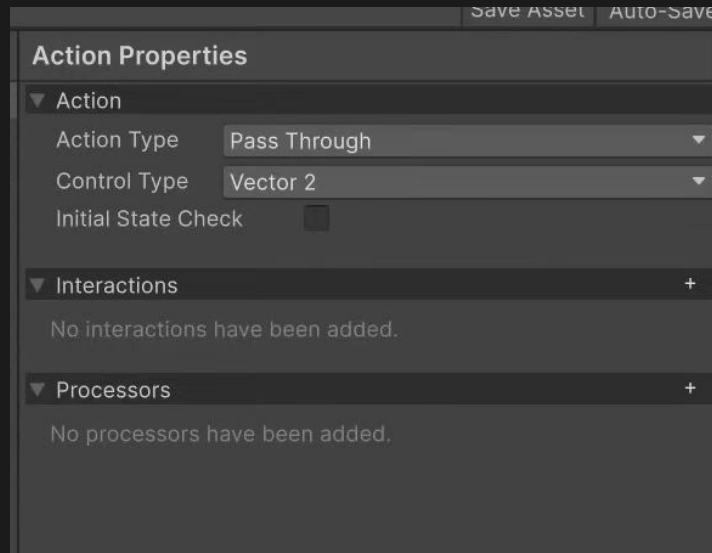


The Input Action Editor

- Action maps
 - Way to organize actions

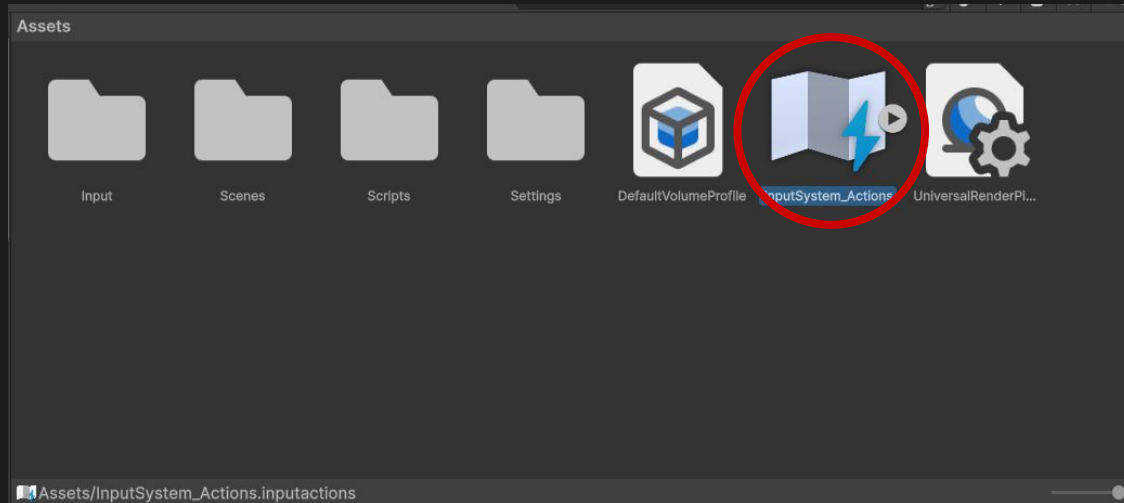


- Action properties
 - Used to modify what data an action returns
 - Used to modify how an action is triggered
 - Button Pressed? Tapped? Held?

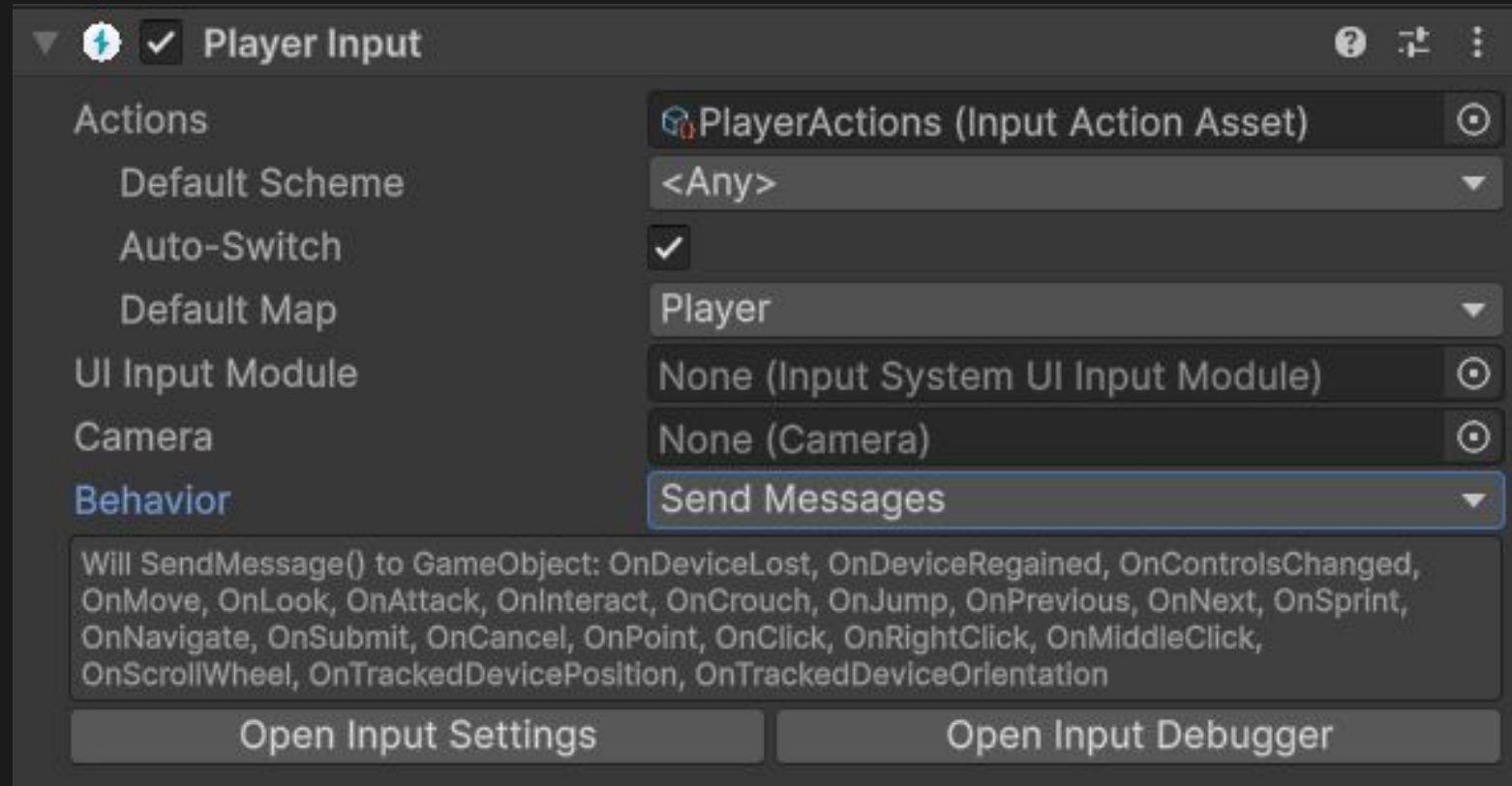


The Input Action Editor

- All your action maps, actions, and action properties are stored in a .inputactions file in your project.
 - Note: You can create new Input Action Assets at any time
- How can we use the data from this file in our code/in our game?



. The Player Input Component



The screenshot displays the Unity Inspector for the **Player Input** component. The component is active, as indicated by the checkmark icon. The settings are as follows:

- Actions:** PlayerActions (Input Action Asset)
- Default Scheme:** <Any>
- Auto-Switch:**
- Default Map:** Player
- UI Input Module:** None (Input System UI Input Module)
- Camera:** None (Camera)
- Behavior:** Send Messages

The **Behavior** dropdown is expanded, showing a list of events that will trigger `SendMessage()` to the `GameObject`:

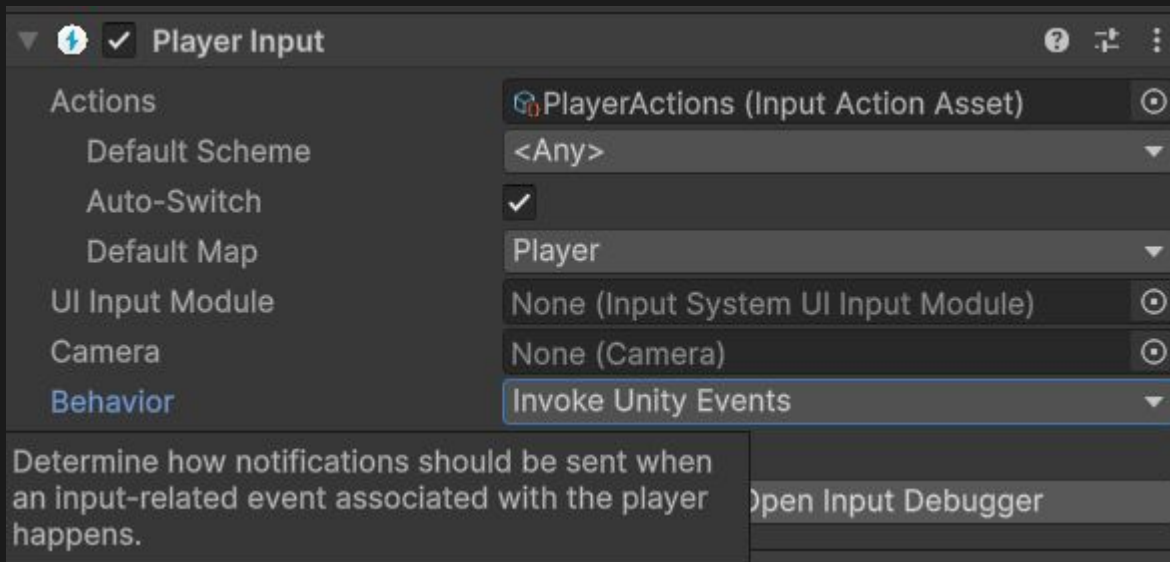
- OnDeviceLost
- OnDeviceRegained
- OnControlsChanged
- OnMove
- OnLook
- OnAttack
- OnInteract
- OnCrouch
- OnJump
- OnPrevious
- OnNext
- OnSprint
- OnNavigate
- OnSubmit
- OnCancel
- OnPoint
- OnClick
- OnRightClick
- OnMiddleClick
- OnScrollWheel
- OnTrackedDevicePosition
- OnTrackedDeviceOrientation

At the bottom of the Inspector, there are two buttons: **Open Input Settings** and **Open Input Debugger**.



. The Player Input Component

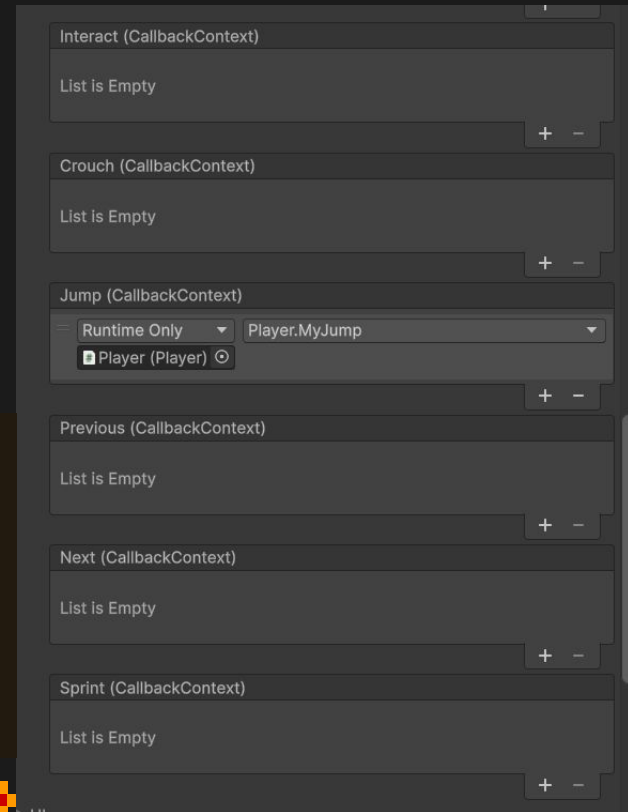
- What is it?
 - If a GameObject has a Player Input Component, the component will send information about Actions being performed to scripts on that GameObject
- How do we “send information”?
 - Several options, we will use Invoke Unity Events
 - Provides intuitive graphical way to run desired functions when Action are performed
 -
 -



Connecting to our Code

- In the Player Input Component, we can assign functions to be called to Actions
- These are functions that we write!

```
0 references  
public void MyJump(InputAction.CallbackContext context)  
{  
    if(context.performed)  
    {  
        Debug.Log("Jumped!");  
    }  
}
```



Reading Input Values

- Sometimes we only care about when a button is pressed, as in the previous example
- Other times, we would like more data from our input devices eg: Mouse position, joystick direction
- How do we read these values?
 - `context.ReadValue<T>()`

```
0 references  
public void MyWalk(InputAction.CallbackContext context)  
{  
    input = context.ReadValue<Vector2>();  
    Debug.Log(input);  
}
```



Demo/Lab Intermission!

Two Orders of Business:

1. Get Intellisense working for as many people as possible
2. Use what we now know about the input system to print a line in the console on button press



Intellisense

- Auto completion and error detection in VS Code
- Incredibly important and useful
- Sometimes working out the box, other times can be incredibly frustrating getting set up
 - Things can vary from computer to computer, we'll try our best to accommodate

```
transform.  
if(context  
{  
  Debug.  
}
```

▼ InverseTransformVectors
▼ IsChildOf
🔑 localEulerAngles
🔑 localPosition
🔑 localRotation
🔑 localScale
🔑 localToWorldMatrix
▼ LookAt
🔑 lossyScale
🔑 name
🔑 parent
🔑 position

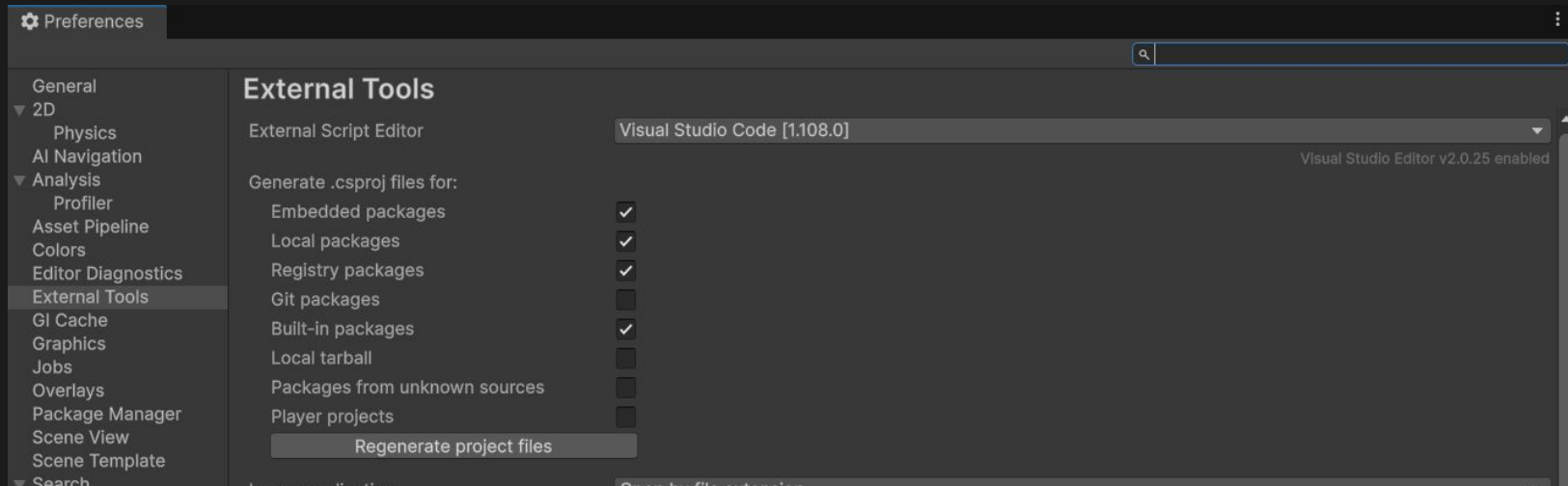
```
void Component.BroadcastMessage(string | ×  
(+ 3 overloads)  
Calls the method named methodName on  
every MonoBehaviour in this game object or  
any of its children.
```

Intellisense Checklist

1. Install .NET SDK

<https://dotnet.microsoft.com/en-us/download>

2. Go to **Edit->Preferences->External Tools** make sure matching boxes are checked and click **Regenerate Project Files**



Intellisense Checklist

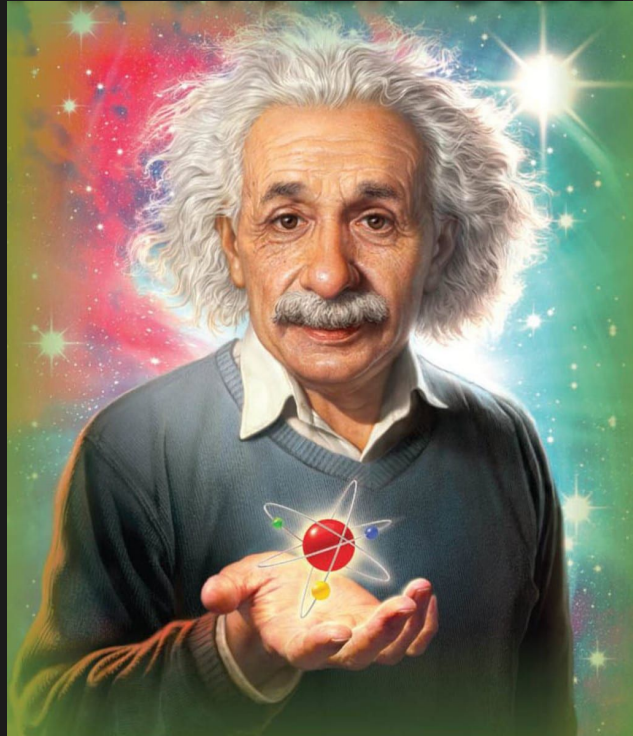
3. Click Assets -> Open C# Project
4. Test that Intellisense is working

Troubleshooting:

- Make sure you have `Assembly-CSharp.csproj`,
`Assembly-CSharp-Editor.csproj`, and `*project-name*.sln[x]`
files in project root directory, these should be generated when you
Regenerate Project Files

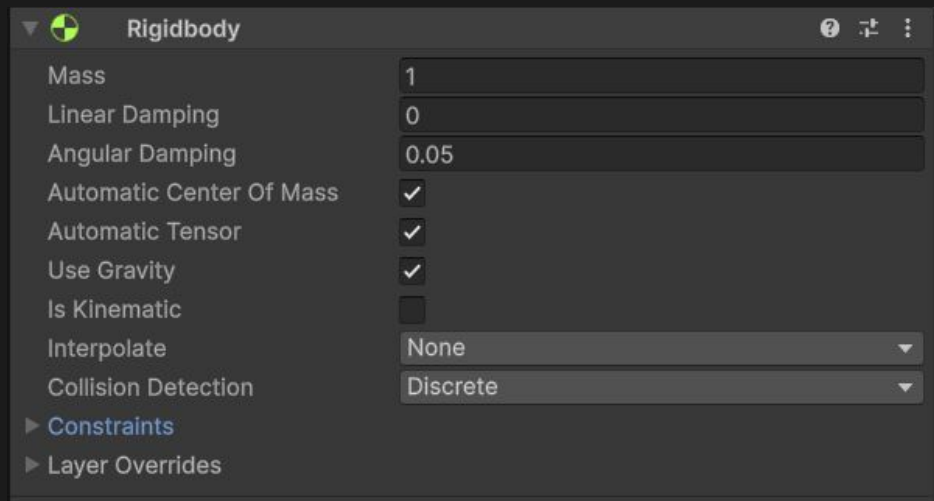


Physics

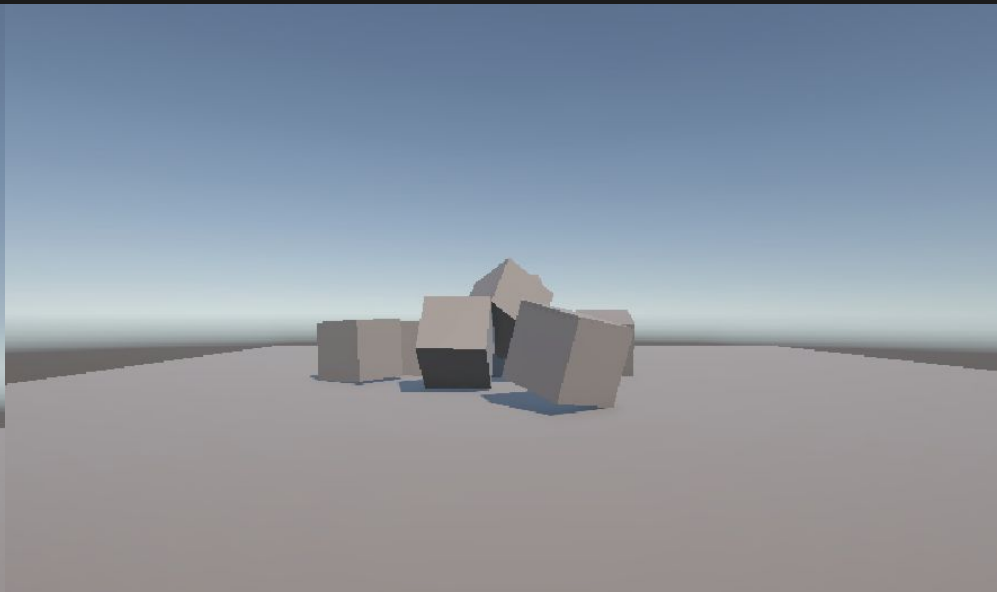
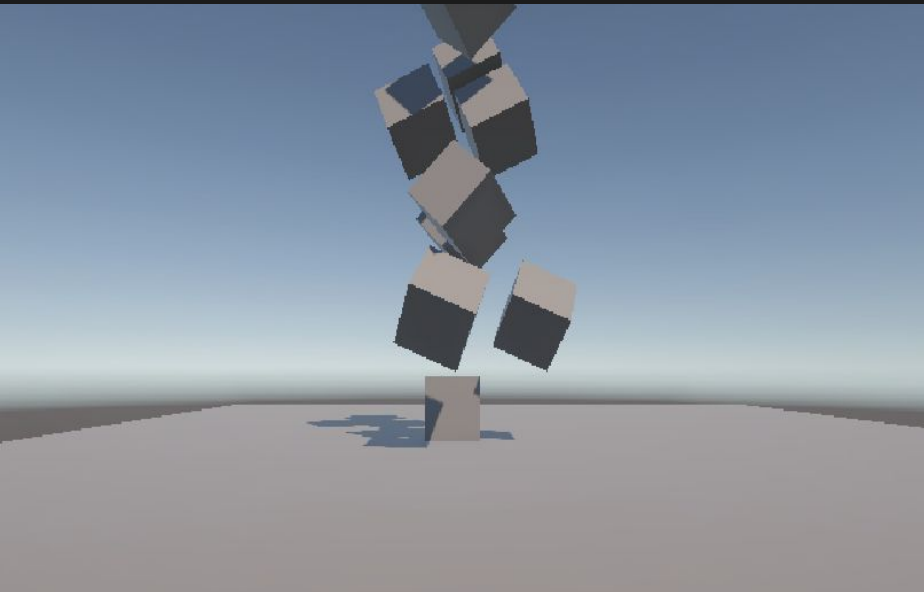


Rigidbody

- Rigidbody component
 - Allows object to be affected by physics
 - All info here:
<https://docs.unity3d.com/6000.3/Documentation/ScriptReference/Rigidbody.html>
- Notable Parameters:
 - Is Kinematic
 - Interpolate
 - Collision Detection
 - Constraints
 -



Rigidbody



- All you need is a rigidbody component and a collider component (box collider, sphere collider, mesh collider, etc..) and objects will behave like so



Controlling Rigidbodies

- `rb.AddForce(Vector3 force)`
 - `rb.AddForce(Vec3 force, Forcemode Mode)`

Properties

| Property | Description |
|--------------------------------|---|
| Force | Add a continuous force to the rigidbody, using its mass. |
| Acceleration | Add a continuous acceleration to the rigidbody, ignoring its mass. |
| Impulse | Add an instant force impulse to the rigidbody, using its mass. |
| VelocityChange | Add an instant velocity change to the rigidbody, ignoring its mass. |



Collision

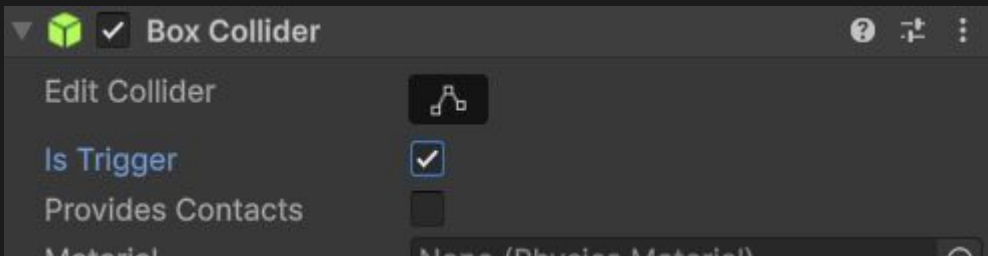
- Unity's physics system is tied to its collision system
- Make sure you have both a rigidbody and some type of collision component on your GameObject!
- Then use these 3 methods as pleased

```
0 references | ☑ Unity Message  
void OnCollisionEnter(Collision collision)  
{  
  
}  
  
0 references | ☑ Unity Message  
void OnCollisionStay(Collision collision)  
{  
  
}  
  
0 references | ☑ Unity Message  
void OnCollisionExit(Collision collision)  
{  
  
}
```



Collision Continued

- All collider components have the **Is Trigger** field
- Objects with Trigger collides do not need Rigidbodies
- This is because they do not physically collide with other objects
- Used for things like killzones, basketball hoops, cutscene triggers, AOE zones, etc



```
0 references | Unity Message  
void OnTriggerEnter(Collider other)  
{  
  
}  
  
0 references | Unity Message  
void OnTriggerStay(Collider other)  
{  
  
}  
  
0 references | Unity Message  
void OnTriggerExit(Collider other)  
{  
  
}
```



Physics.Raycast



- Use to detect if player is grounded, if a hitscan bullet hits a target, and in many other cases!
- Casts a ray, from point origin, in direction direction, of length maxDistance, against all colliders in the Scene.
- To select which layers a ray should collide with, use a LayerMask.

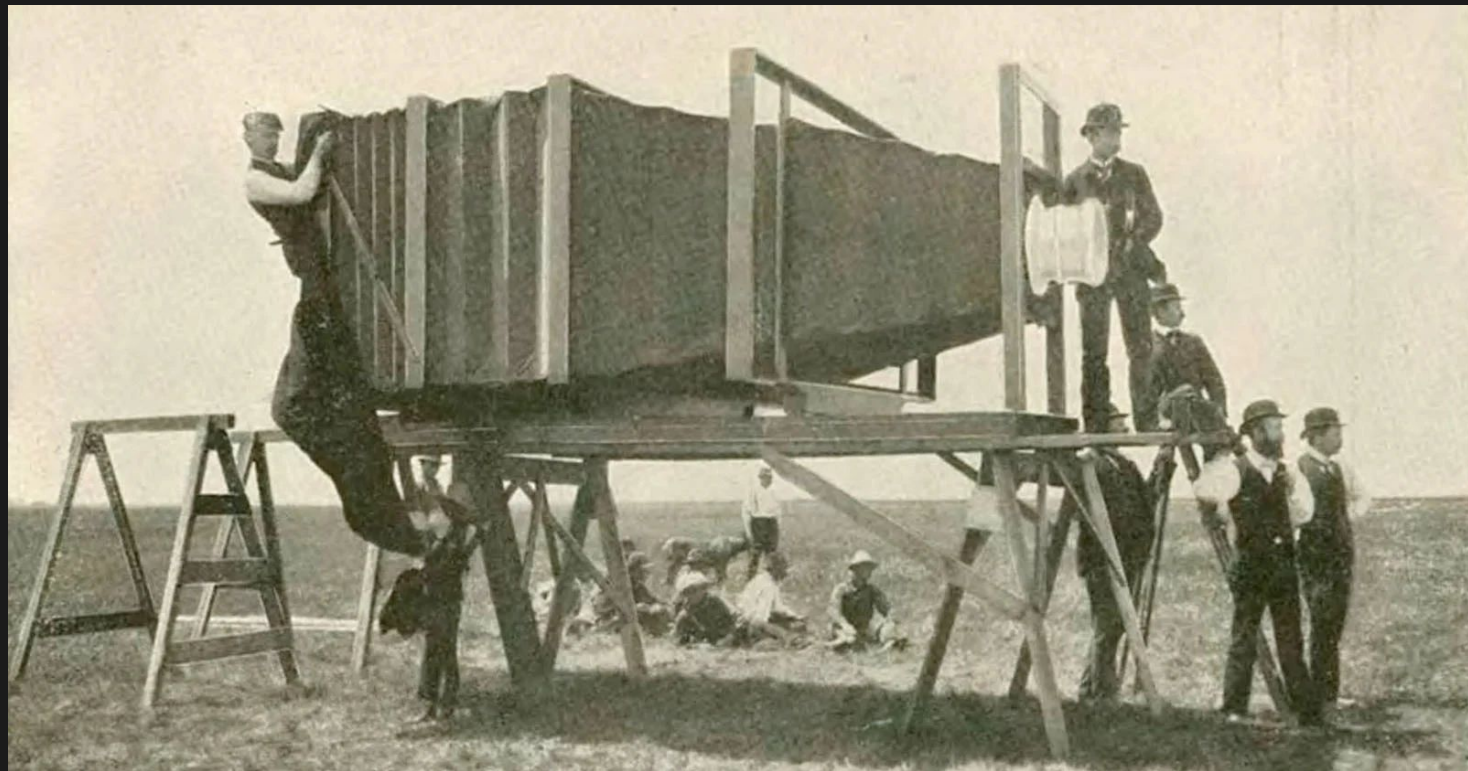
```
RaycastHit hit;
// Does the ray intersect any objects excluding the player layer
if (Physics.Raycast(transform.position, transform.TransformDirection(Vector3.forward), out hit, Mathf.Infinity, layerMask))
{
    Debug.DrawRay(transform.position, transform.TransformDirection(Vector3.forward) * hit.distance, Color.yellow);
    Debug.Log("Did Hit");
}
```

Demo 2

- Use what we've learned to make a basic character controller



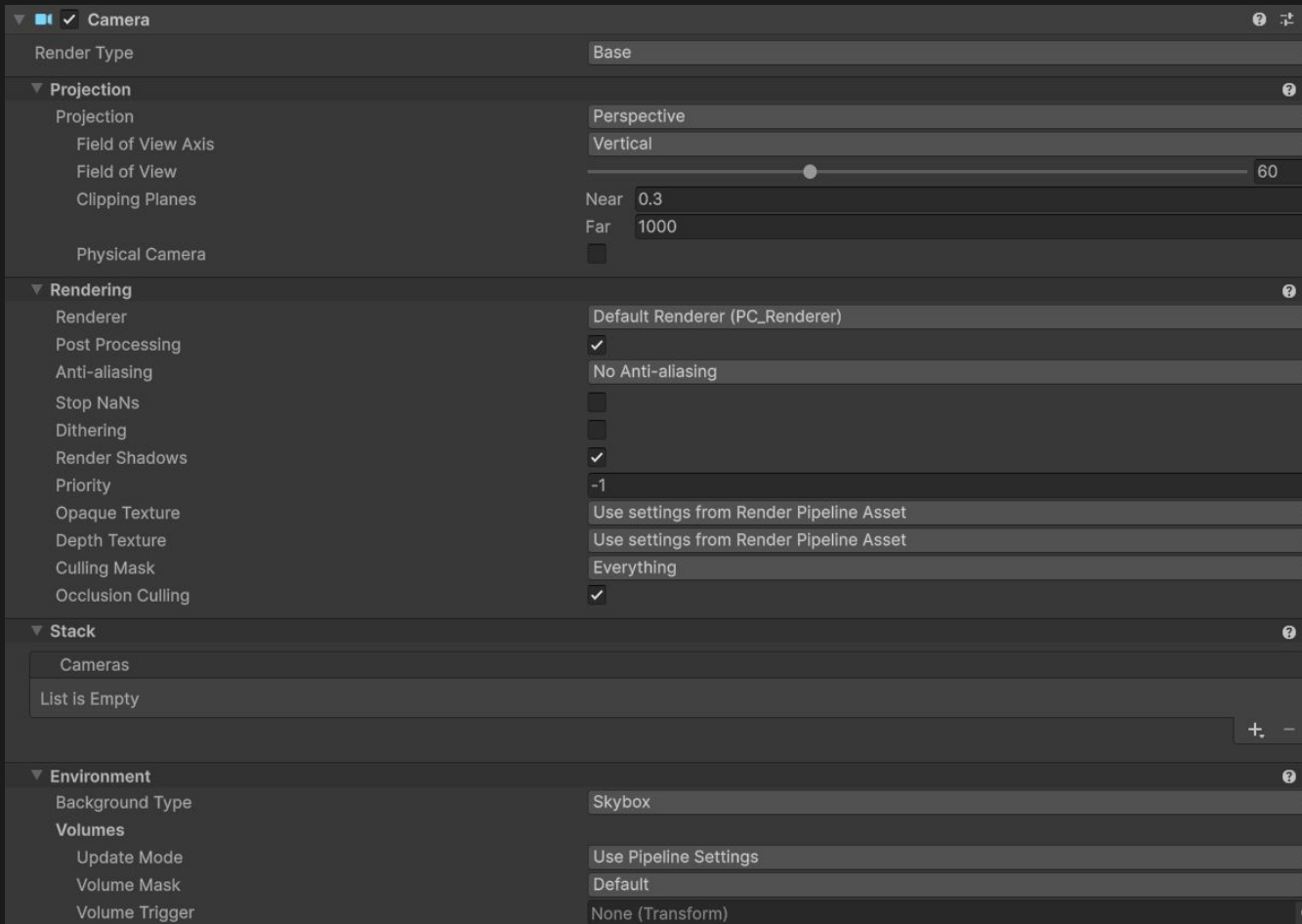
The Camera



The Camera Component

Many parameters, key ones are:

- Projection
- Field of View
- Clipping Planes
- Background Type



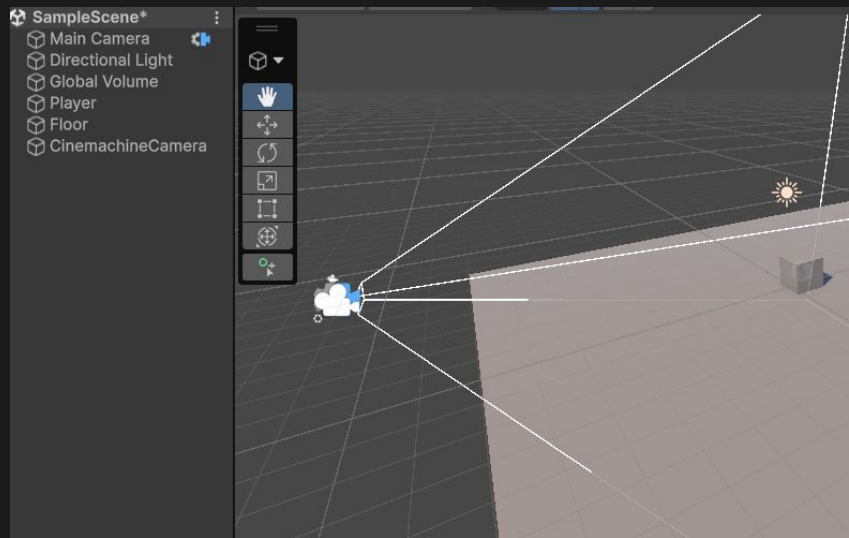
Controlling the Camera

- Many camera types
 - 3rd person, fps, side scroller, top down
- Each of these camera types have their own considerations
 - Preventing camera clipping
 - Scoping / zooming
 - Interpolation / smoothing
- We use the Cinemachine Package to handle all of this



Cinemachine Overview

- Solves the complex mathematics and logic of tracking targets, composing, blending, cutting between shots
- Dynamically adjusts its behavior to make the best shot given changes in animation, vehicle speed, terrain, etc
- Works in real time across all genres including FPS, third person, 2D, side-scroller, top down, and RTS.

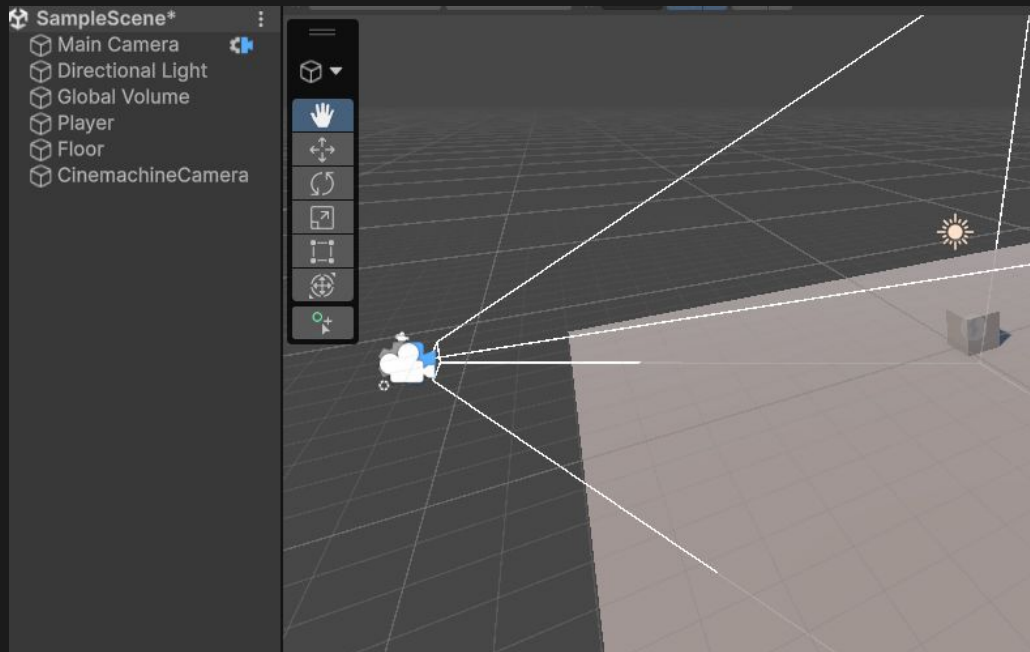


<https://docs.unity3d.com/Packages/com.unity.cinemachine@3.1/manual/index.html>



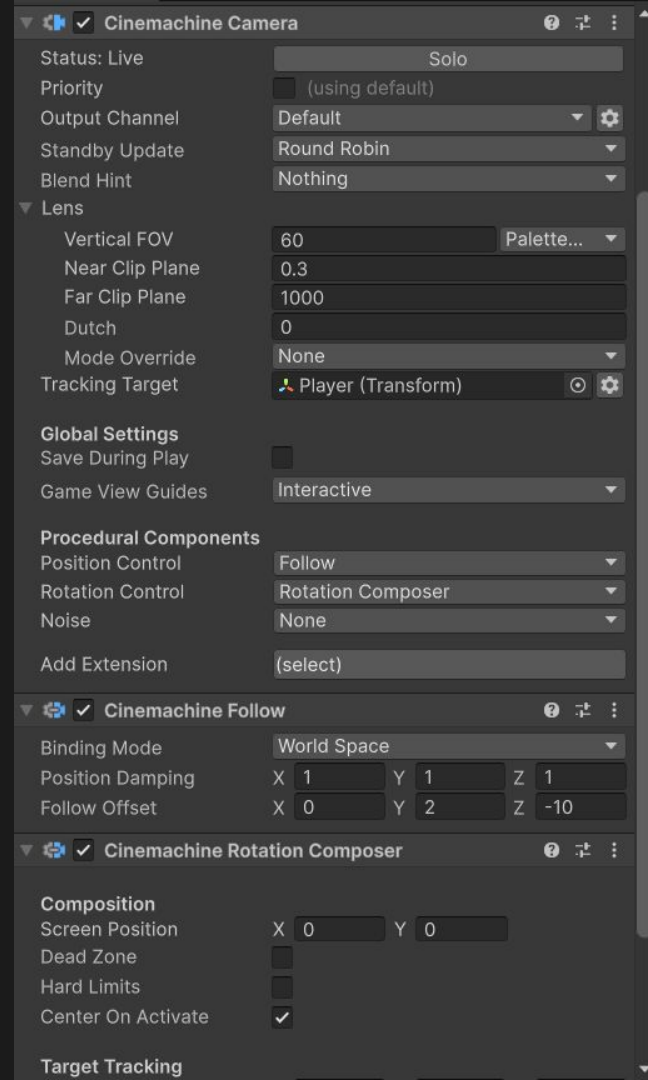
Cinemachine Overview

- Adds virtual camera(s), one of which (at a time) will control the movement and rotation of the main Unity Camera
- Provides modular system for changing how these virtual cameras behave
- Provides programming interface for enabling, disabling, and customizing behavior of virtual cameras



The Cinemachine Component

- Lens tab controls parameters of the Camera previously mentioned
- Procedural Components tab most important
 - Add position controls (following player)
 - Add rotation controls (looking at player)
- You can add extensions to add more camera effects. (Support pixel art, camera shake, and more)



■ Cinemachine Installation

- Window->Package Management->Package Manager
- Go to Unity Registry, look up Cinemachine, click install

The screenshot displays the Unity Package Manager window. On the left, the 'Window' menu is open, showing 'Package Management' selected, with a sub-menu containing 'Package Manager', 'My Assets', 'Services', and 'Asset Store'. The main interface shows the 'Package Manager' window with a search bar containing 'cinemachine'. Below the search bar, the 'Packages' list shows 'Cinemachine' version 3.1.5. The right-hand pane displays the details for 'Cinemachine 3.1.5', including the release date 'October 23, 2025', the source 'Unity Registry by Unity Technologies', and links for 'Documentation', 'Changelog', and 'Licenses'. An 'Install' button is visible. At the bottom, there is a 'Last refresh' timestamp of 'Jan 15, 17:28' and a refresh icon.

Window Help

Layouts >

Panels >

Next Window Ctrl+Tab

Previous Window Ctrl+Shift+Tab

General >

2D >

Accessibility >

AI >

Analysis >

Animation >

Audio >

Multiplayer >

Package Management >

Rendering >

Search >

Sequencing >

Text >

TextMeshPro >

UI Toolkit >

Version Control >

Visual Scripting >

Package Manager

My Assets

Services Ctrl+0

Asset Store

Package Manager

+ Sort: Name (asc) Filters Clear Filters

In Project

Updates

Unity Registry

My Assets

Built-in

Services

Search cinemachine

▼ Packages

Cinemachine 3.1.5

Cinemachine

3.1.5

Install

Details Version History Dependencies

Technical Name com.unity.cinemachine

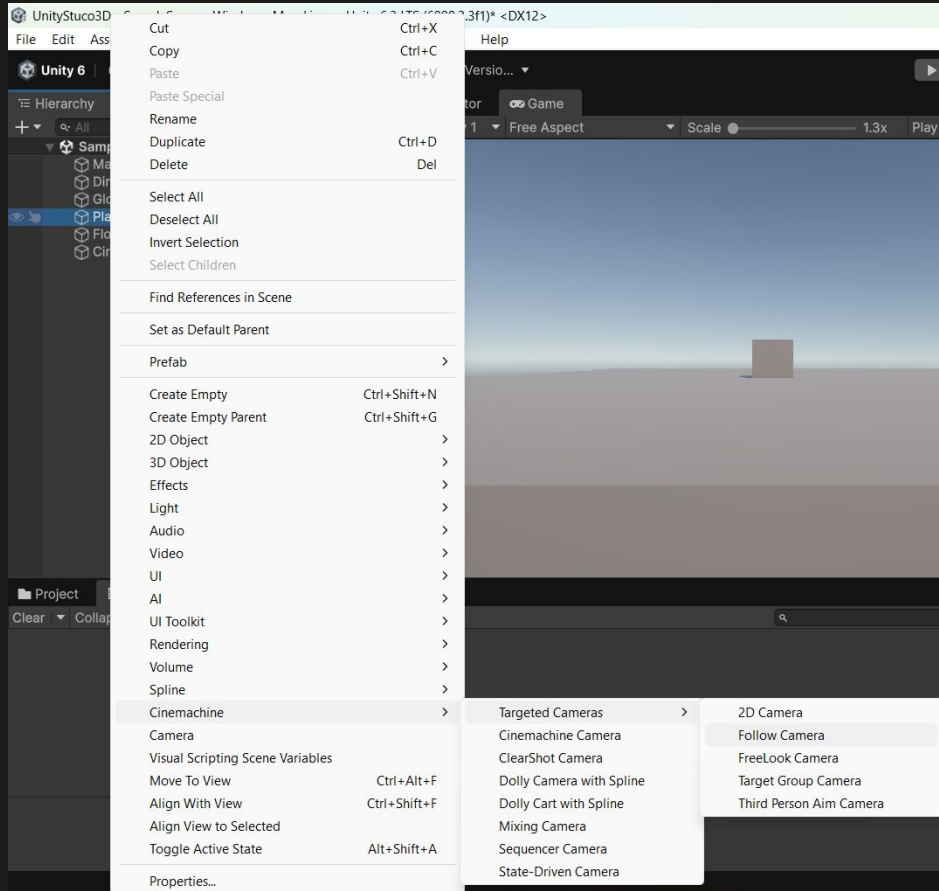
Minimum Editor Version 2022.3

Smart camera tools for passionate creators.

Cinemachine 3 is a newer and better version of Cinemachine, but upgrading an existing project from 2.X will likely require some effort. If you're considering upgrading an older project, please see our upgrade guide in the user manual.

Last refresh Jan 15, 17:28

Using Cinemachine



Right click your player in the hierarchy.

To set up from scratch, click:

(Cinemachine->Cinemachine Camera)

To choose a preset:

(Cinemachine->Targeted Camera->_)



Demo 3

- Set up Cinemachine Camera for our player controller
- Make sure everyone is caught up

