Tutorial of unity3d

> Designed by ZHU Yueming, Haoran Wang, Yucheng Wang, Zexuan JIA in 2023 Aug 21th
>
> Thanks for Haoran Wang and Yuchen Wang provides a teaching demo for this tutorial.

Learning Objectives：

- Can download and install unity3d software
- Have register unity ID
- Understand basic window of unity3d software
- Can import game from existing package.
- Can build game into exe file
- Understand basic winodw of unity 3d.

Part 1. Download and installation

Download Address

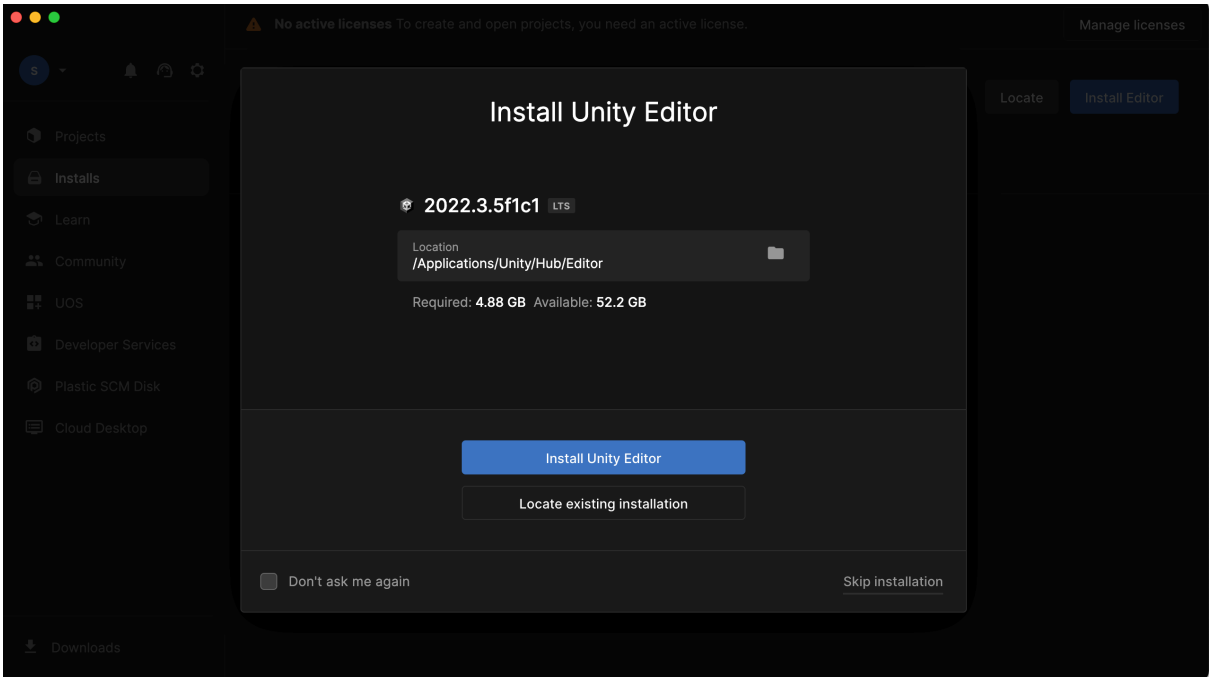
https://unity.cn/releases

Install Steps:

Mac & Windows

- **Register an unity ID**
- **Download Unity Hub**

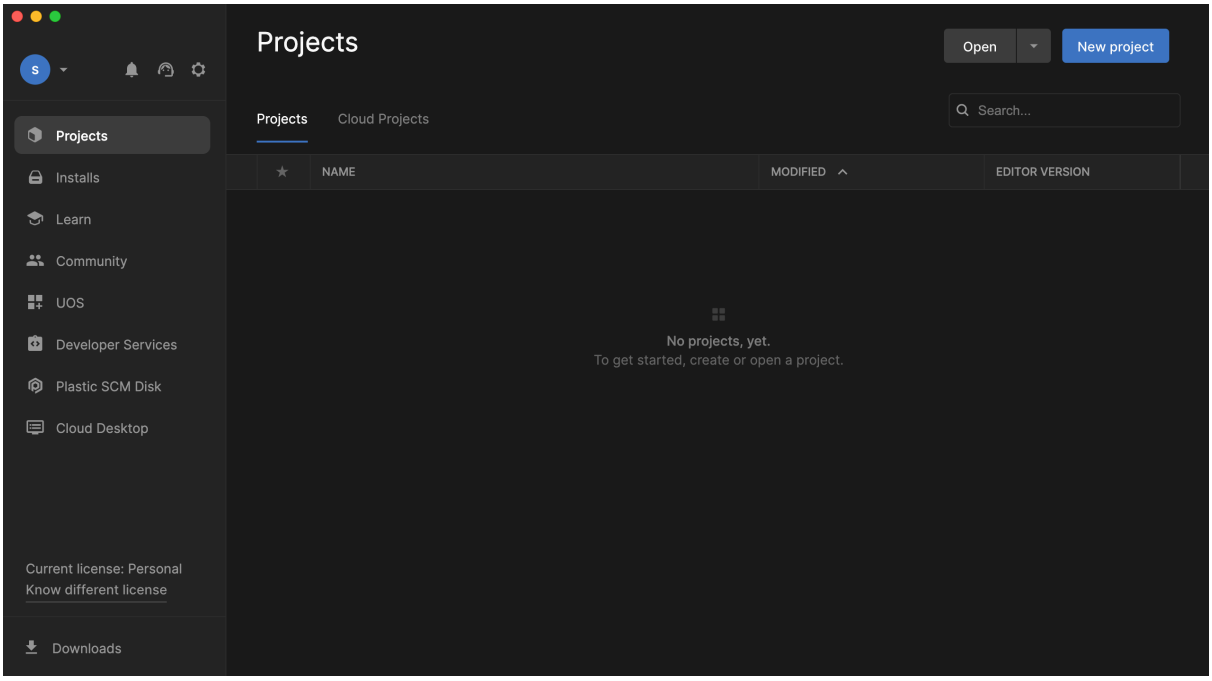
  After download Unity Hub, Sign in with your unity ID, and then open it.



- **Download unity3d**

You can download unity3d software from unity hub or official website.

After install the unity3d software, you can open it.

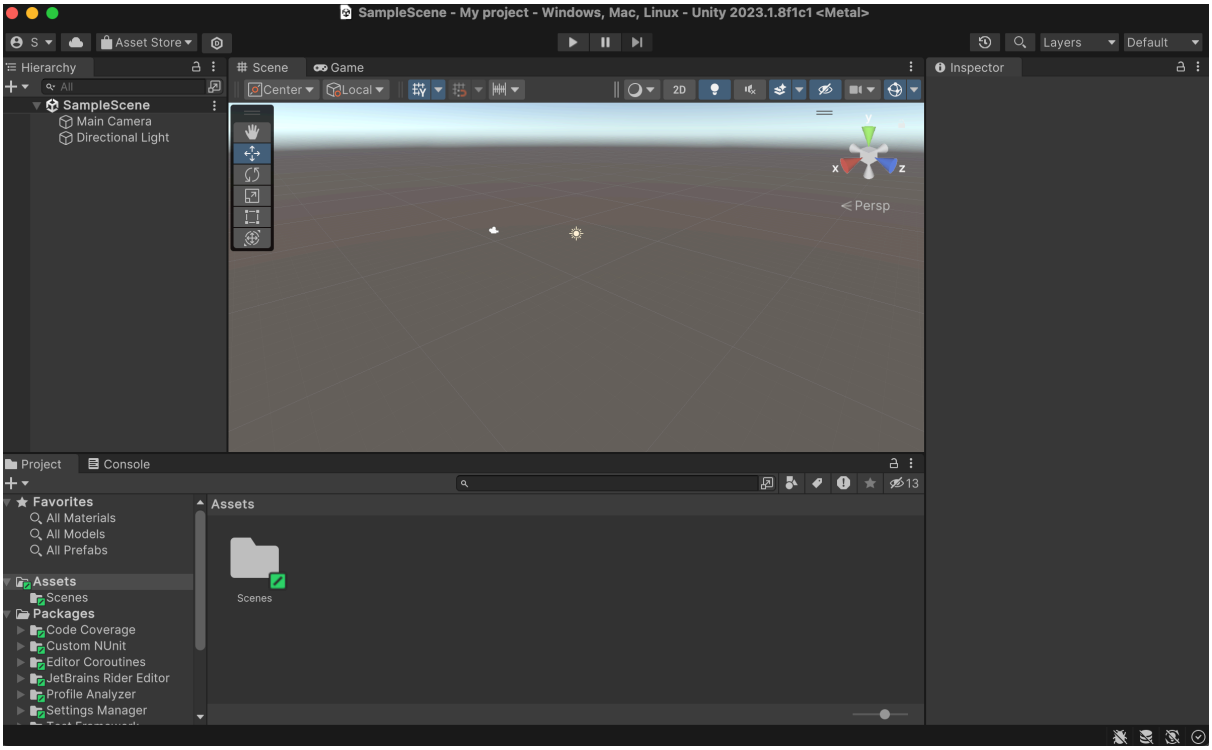


Create a new 3d project

- **Create a new project and download PlasticSCM**

  https://unity.cn/plasticscm

- **A new project would be**



Linux (Ubuntu)

- **Register an unity ID**
- **Download Unity Hub**
  1. Add public signing key.

     ```
     wget -qO - https://hub.unity3d.com/linux/keys/public | gpg --dearmor |
     sudo tee /usr/share/keyrings/Unity_Technologies_ApS.gpg > /dev/null
     ```

  2. Add UnityHub repository in `/etc/apt/sources.list.d`.

     ```
     sudo sh -c 'echo "deb [signed-
     by=/usr/share/keyrings/Unity_Technologies_ApS.gpg]
     https://hub.unity3d.com/linux/repos/deb stable main" >
     /etc/apt/sources.list.d/unityhub.list'
     ```

  3. Update package cache and install.

     ```
     sudo apt update
     sudo apt-get install unityhub # or download specific version
     # sudo apt-get install unityhub=<version>
     ```

- **Install dotnet SDK**
  1. Download script.

     ```
     wget https://dot.net/v1/dotnet-install.sh -O dotnet-install.sh
     ```

  2. Update script permissions.

     ```
     sudo chmod +x ./dotnet-install.sh
     ```

  3. Install dotnet.

     ```
     ./dotnet-install.sh --version latest # LTS 6.0.413, or you could
     install .NET 7.0 SDK
     # ./dotnet-install.sh --channel 7.0
     ```

  4. Update environment variables.

```
vim ~/.bashrc
# insert
export DOTNET_ROOT=$HOME/.dotnet
export PATH=$PATH:$HOME/.dotnet:$HOME/.dotnet/tools
# save
source ~/.bashrc
```

- **Install Mono**

  1. Add the Mono repository to your system.

```
sudo apt install ca-certificates gnupg
sudo gpg --homedir /tmp --no-default-keyring --keyring
/usr/share/keyrings/mono-official-archive-keyring.gpg --keyserver
hkp://keyserver.ubuntu.com:80 --recv-keys
3FA7E0328081BFF6A14DA29AA6A19B38D3D831EF
# Ubuntu 20.04
echo "deb [signed-by=/usr/share/keyrings/mono-official-archive-
keyring.gpg] https://download.mono-project.com/repo/ubuntu vs-focal
main" | sudo tee /etc/apt/sources.list.d/mono-official-vs.list
# Ubuntu 18.04
echo "deb [signed-by=/usr/share/keyrings/mono-official-archive-
keyring.gpg] https://download.mono-project.com/repo/ubuntu vs-bionic
main" | sudo tee /etc/apt/sources.list.d/mono-official-vs.list
sudo apt update
```

  2. Install Mono.

```
sudo apt install mono-devel
# if it not works, use this command, then update and install
# echo "deb [arch=amd64] https://download.mono-project.com/repo/ubuntu
vs-focal main"
```

- **IDE**

  1. **Jetbrains Rider**

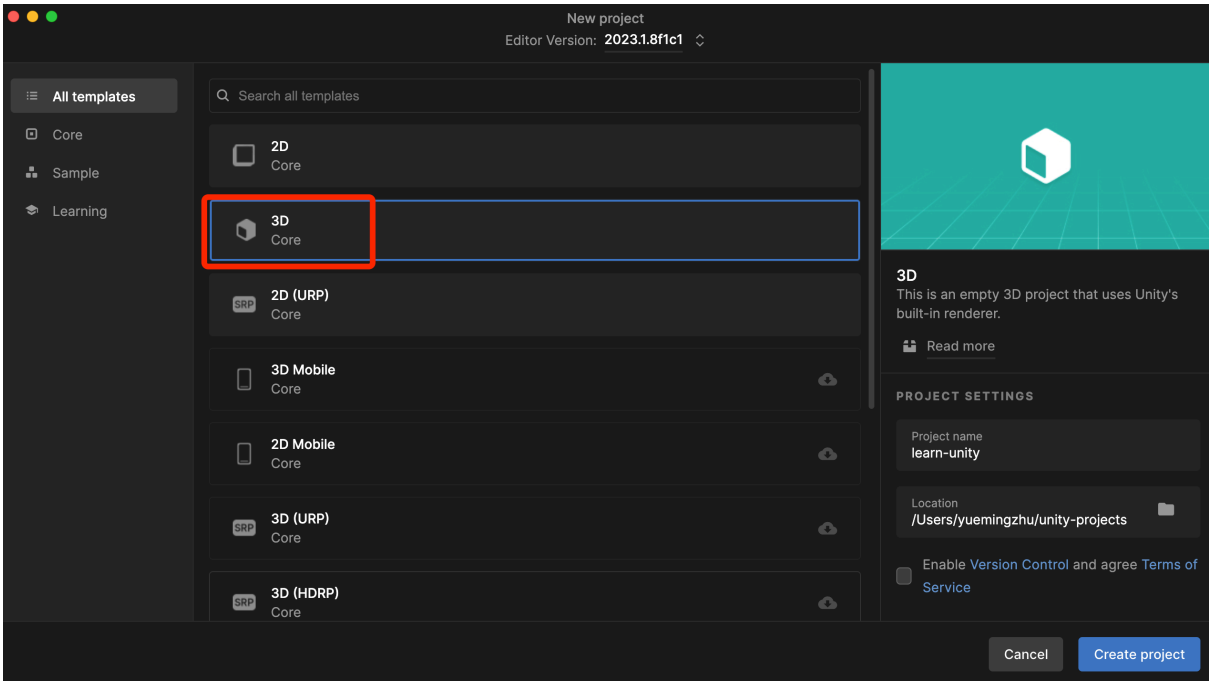
```
# download from offical website
xdg-open https://www.jetbrains.com/rider/download/#section=linux

# extract the tarball to a directory that supports file execution
sudo tar -xzf Rider.tar.gz -C <your directory>

# excute script
bash Rider.sh
```

```
# or download from snap
sudo snap install rider --classic
```

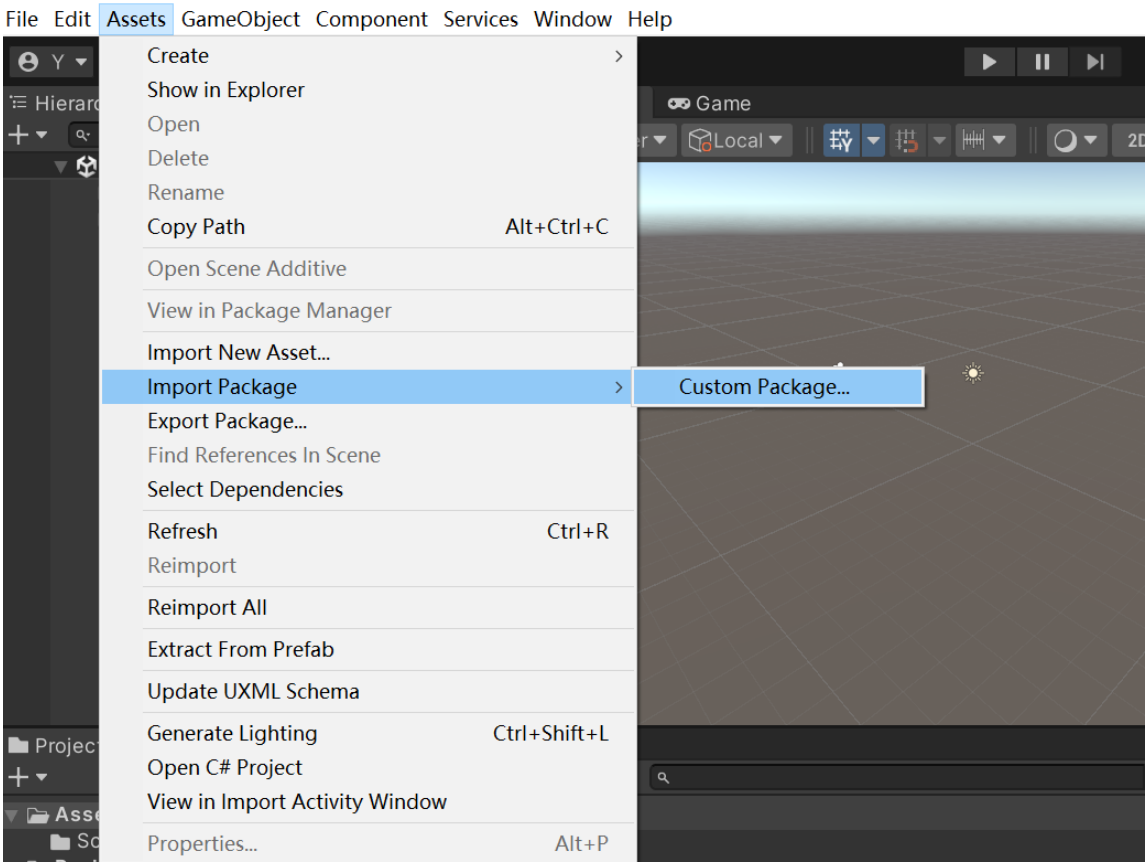
2. **VScode**
    1. Open VScode, go to menu File - Preferences - Settings - Extension - C# configuration.
    2. Look for "**Omnisharp: Use Global Mono**", and set it to "**always**", then look for "**Omnisharp: use Modern Net**"" and **uncheck** it.
    3. Restart VScode.
3. **Connect with Unity**
    1. Go to Unity menu Edit - Preferences - External Tools - External Script Editor, and pick "Visual Studio Code" or "JetBrains Rider".
    2. In the same window, check "Embedded packages", "Local packages", "Built-in packages", "Git packages" and after click on "**Regenerate project files**".

Part 2. Import teaching demo

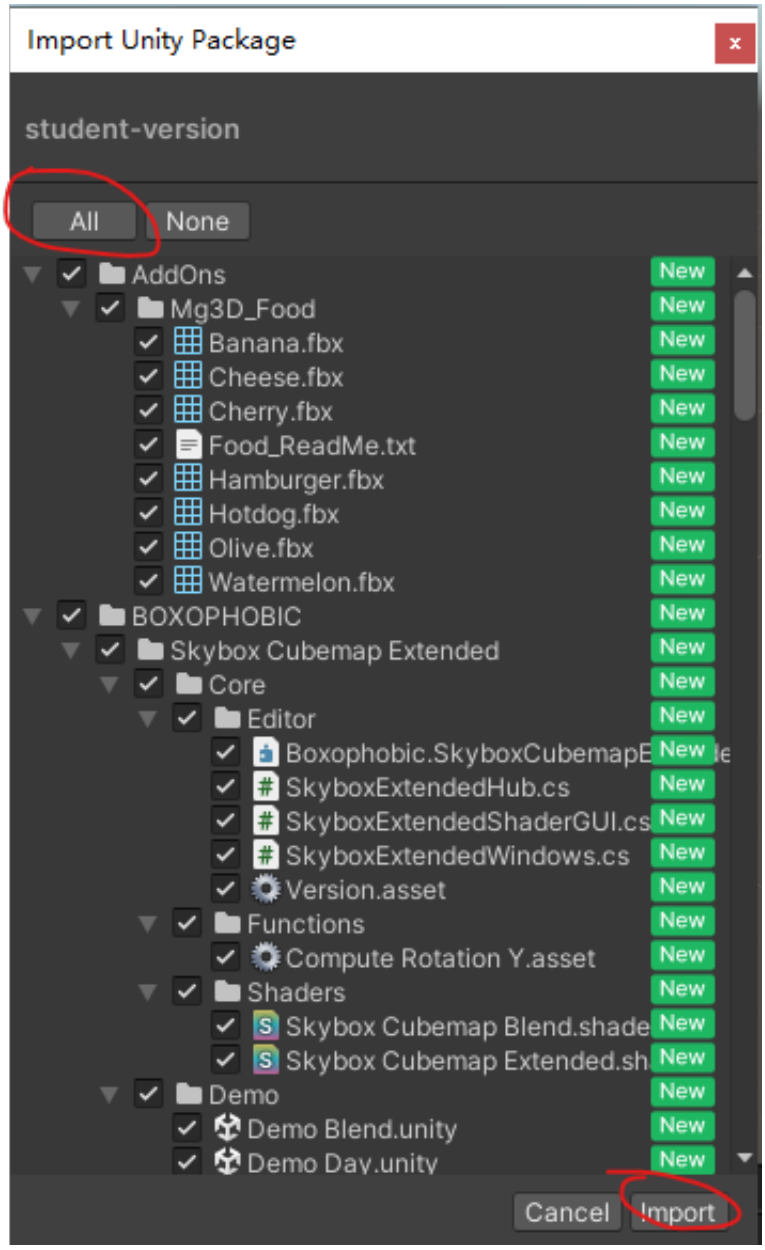
1. **Create a new 3D project**



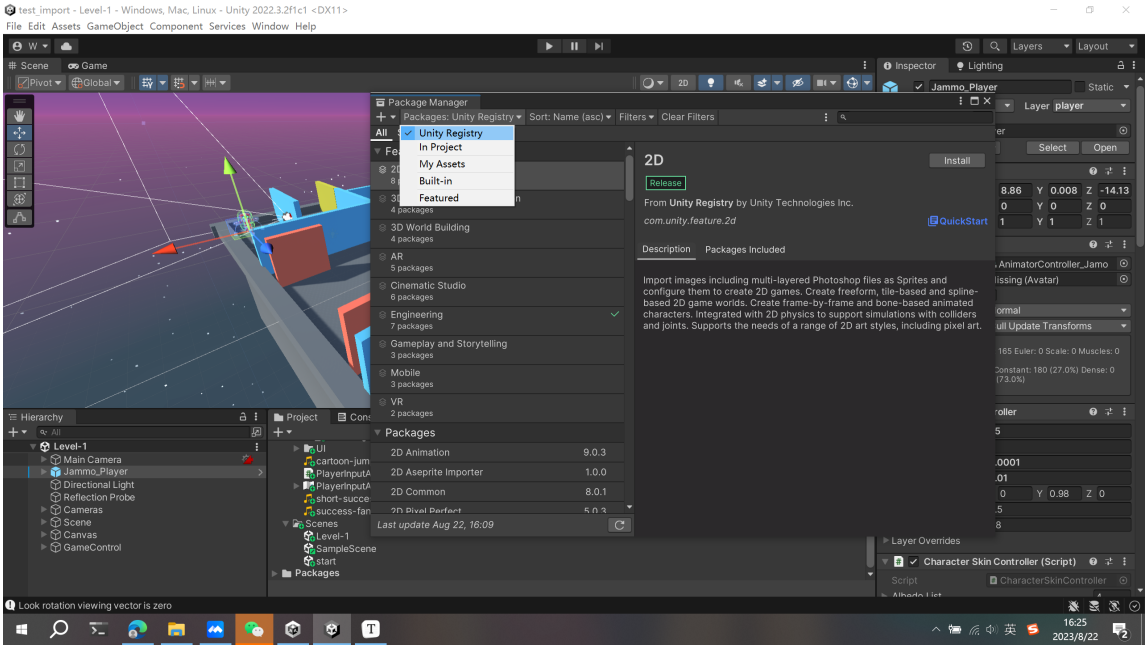
2. **Import demo package**
    - Assets -> Import Package -> Custom Package



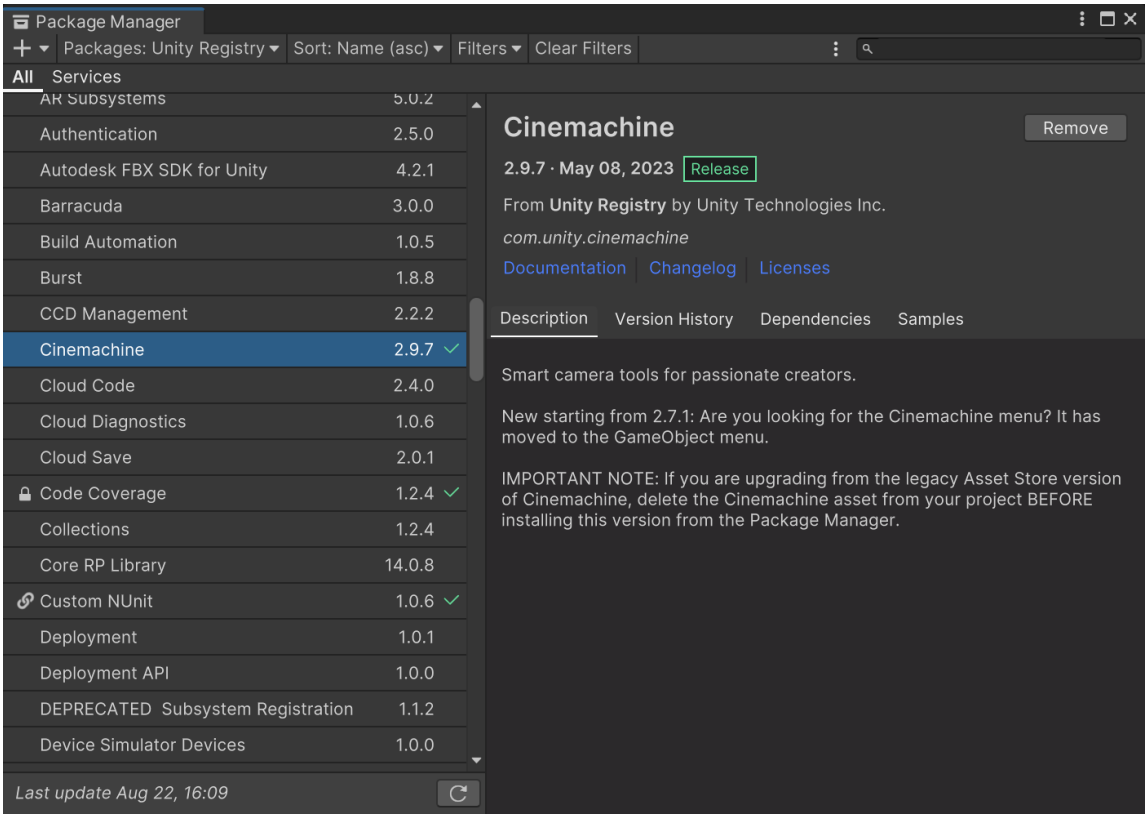
- Select student-version.unitypackage, then click Import.



3. **Download Packages**
    1. Click `Window -> Package Manager`
    2.



3. Find `Cinemachine` and `Input System` . Download.



4. The Editor will inform you that the editor should be restart. Click "Yes".

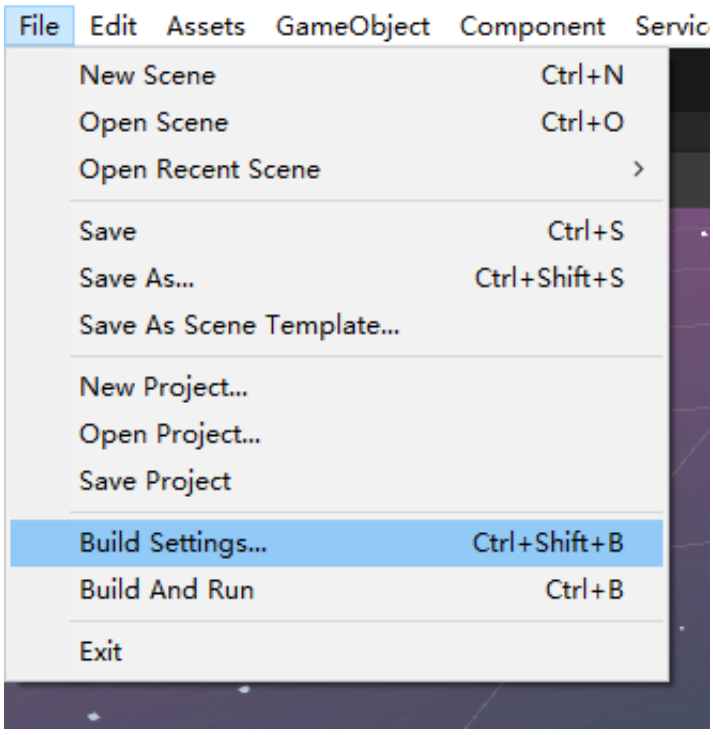
4. **Add Layers**

1. Go to the folder of your unity project, and then replace `{Project Name}/ProjectSettings/TagManager.asset` with our provided `TagManager.asset`

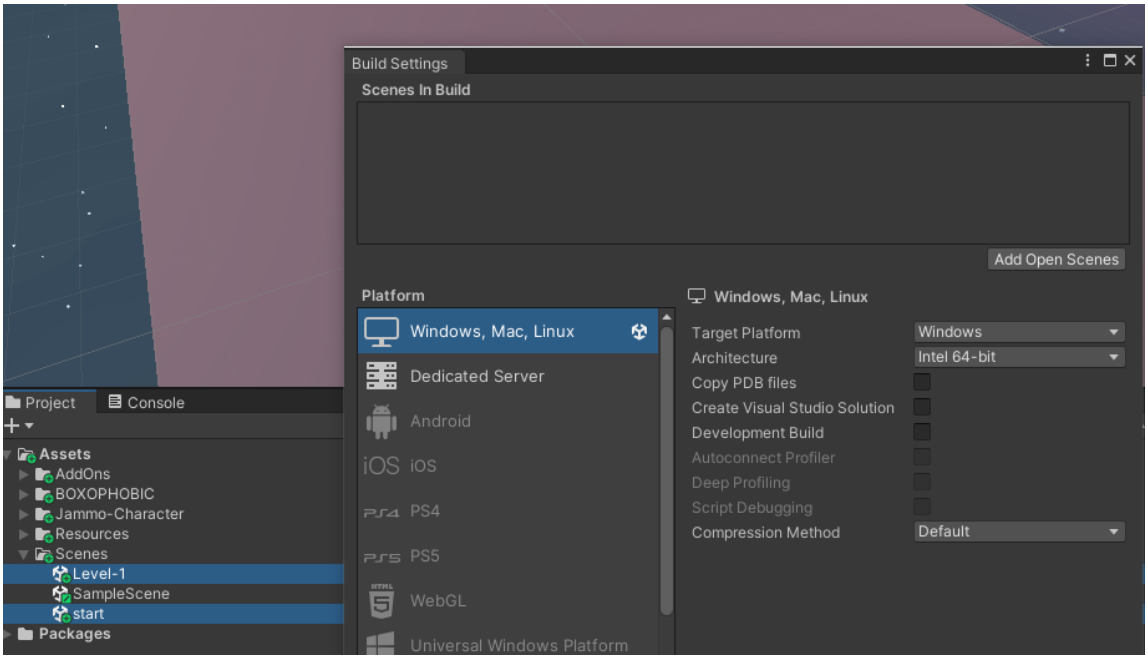
5. **Build Setting**

   In this part, we need add all scenes into one game.

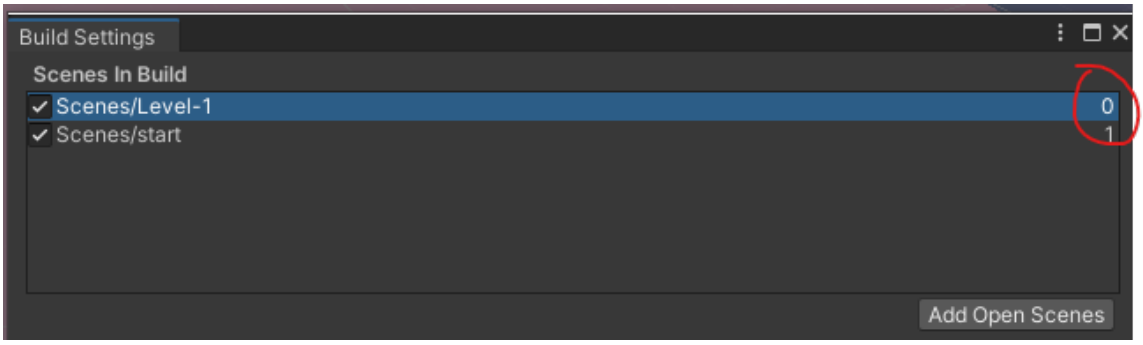
   1. Open `File->Build Setting`

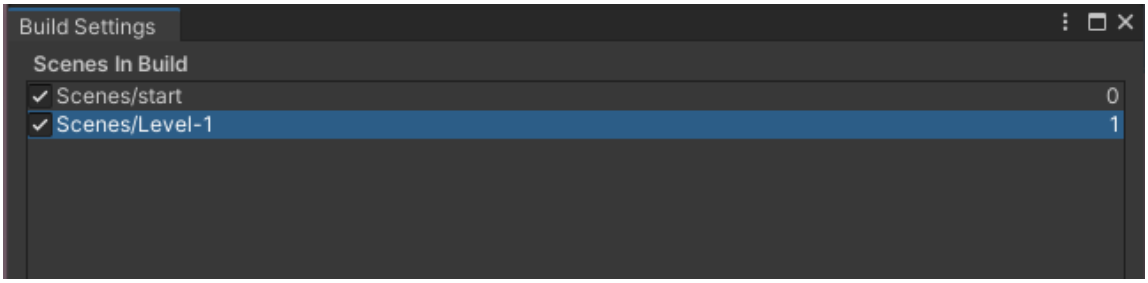


   2. Drag `start` and `Level-1` to `Scenes In Build`



3. Drag `Level-1` down, making `start` ranking 0





4. Click Build button

Part 3. Systems Introduce

1. Game object

In Unity, a GameObject is a fundamental building block used to represent entities in your game world. It's the basic unit of interaction and manipulation within the Unity game engine. GameObjects are used to create and manage various elements in your game, such as characters, items, enemies, cameras, lights, and more.

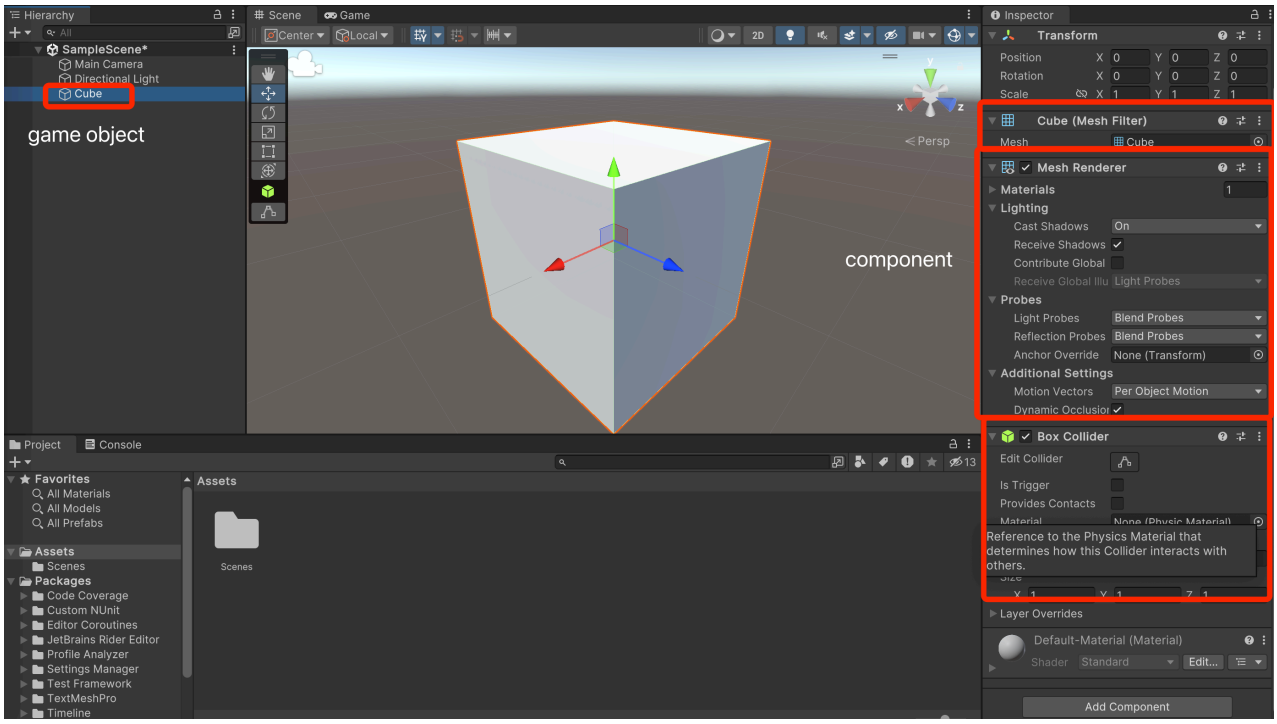
Here are some key points about GameObjects in Unity:

Representation: GameObjects are containers that hold components, which define the behavior, appearance, and functionality of the object. Components can be scripts, colliders, renderers, audio sources, and more.

Hierarchy: In the Unity Editor, GameObjects are organized in a hierarchical structure called the Hierarchy window. This hierarchy represents the relationships between different objects in the scene.

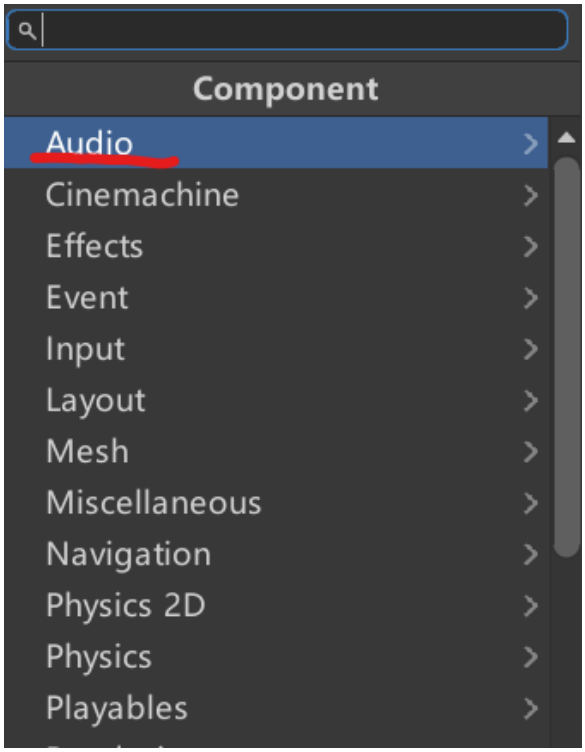
Components: Components are attached to GameObjects to give them specific properties or behaviors. For example, a "Mesh Renderer" component defines how a GameObject should be visually rendered, while a "Rigidbody" component adds physics behavior.

For example: In `Hierarchy` window, there are three game objects: `Main Camera`, `Directional Light`, `Cube`, and click Cube object, we can find several components in `Inspector` window.

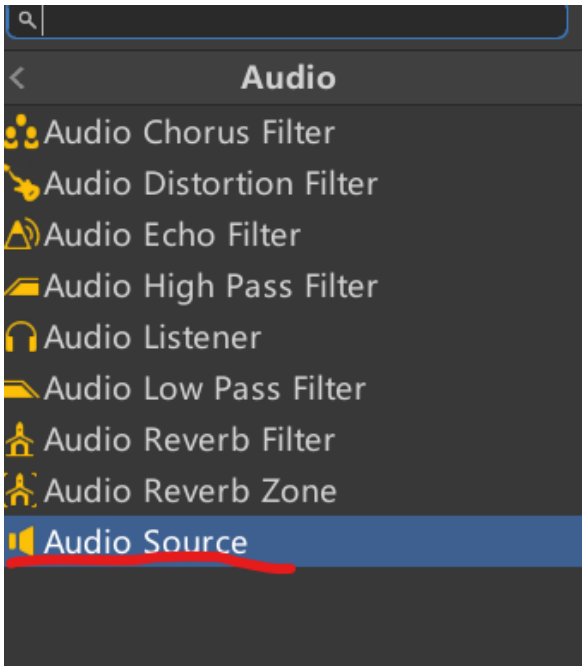


1.1 Add Audio Listener:

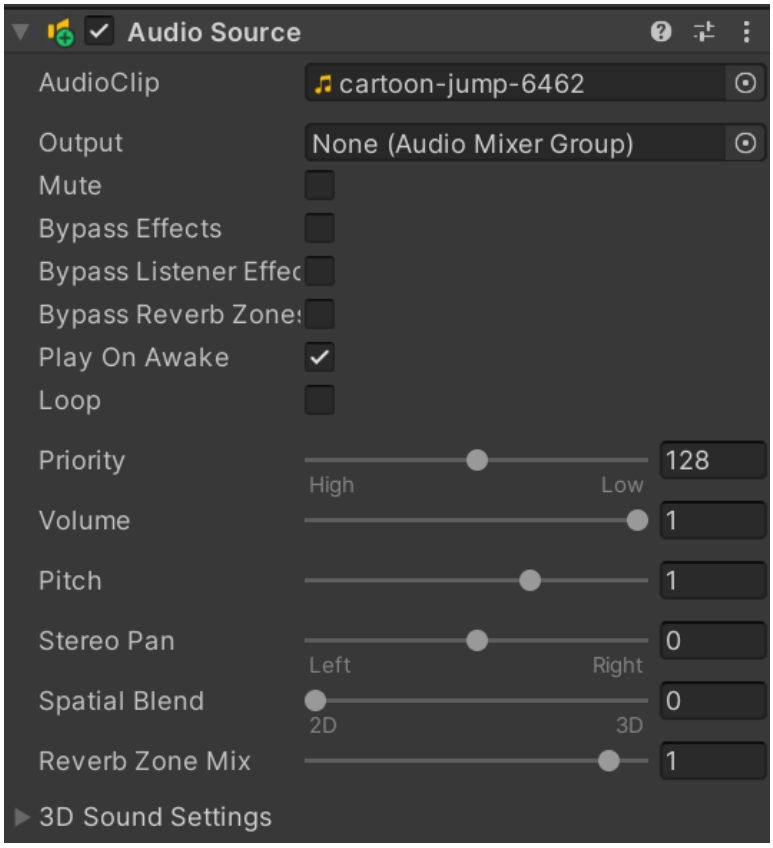
- Select `Jammo_Player` object, then click add component. Then click `Audio`
- Select Audio



- Click Audio Source



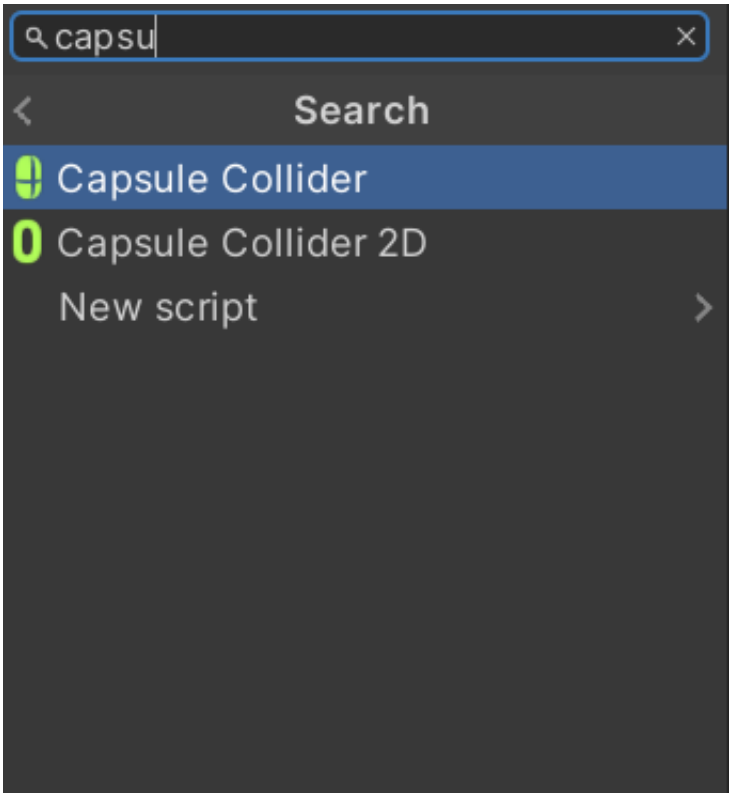
- Add AudioClip



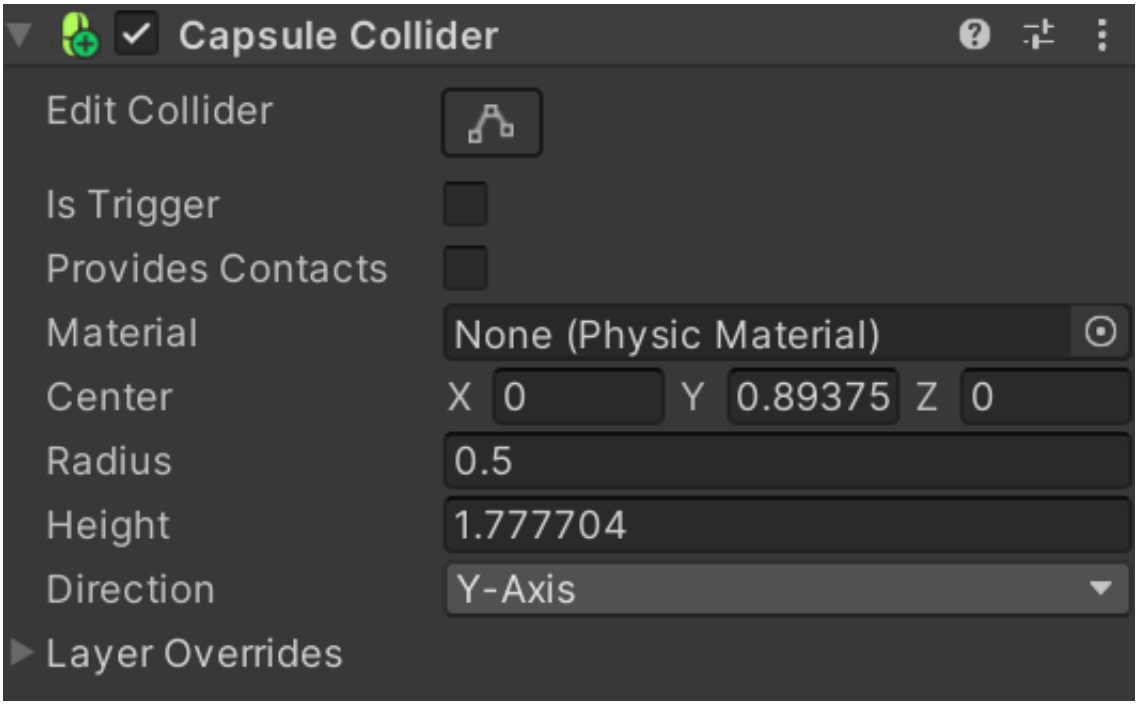
And then you can add the music you like on the AudioClip.

1.2 Collider

- A collider in Unity is a component that defines the shape of a GameObject for the purposes of collision detection. It determines how other objects can interact with and collide against the GameObject. Colliders can be simple shapes like boxes, spheres, or capsules, or more complex meshes. When two colliders intersect, Unity's physics engine can detect the collision and trigger appropriate events or reactions, allowing for realistic interactions and gameplay mechanics. Colliders are essential for creating interactive and dynamic environments in games.
- Select `Jammo_Player` object, then click add component, and search capsule collider

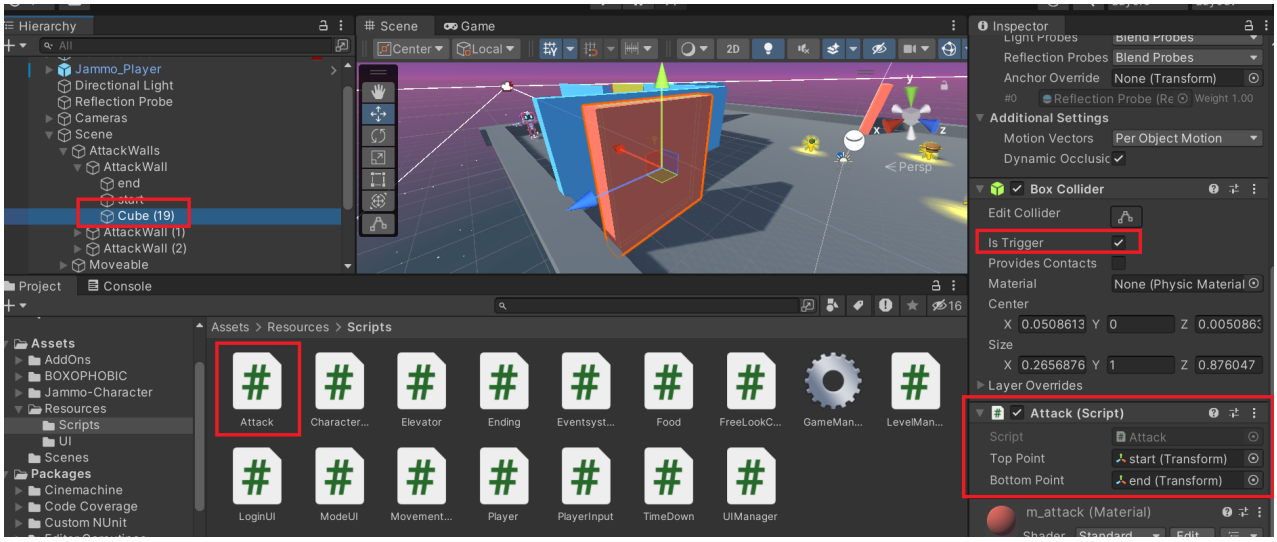


- After adding Collider, it would be:



1.3 Trigger

The trigger part of a Collider in Unity is used to detect trigger events between objects without causing actual physical collisions. When two Colliders with triggers overlap, specific event callbacks are triggered, allowing developers to implement interactive actions such as entering areas or triggering events within the game.

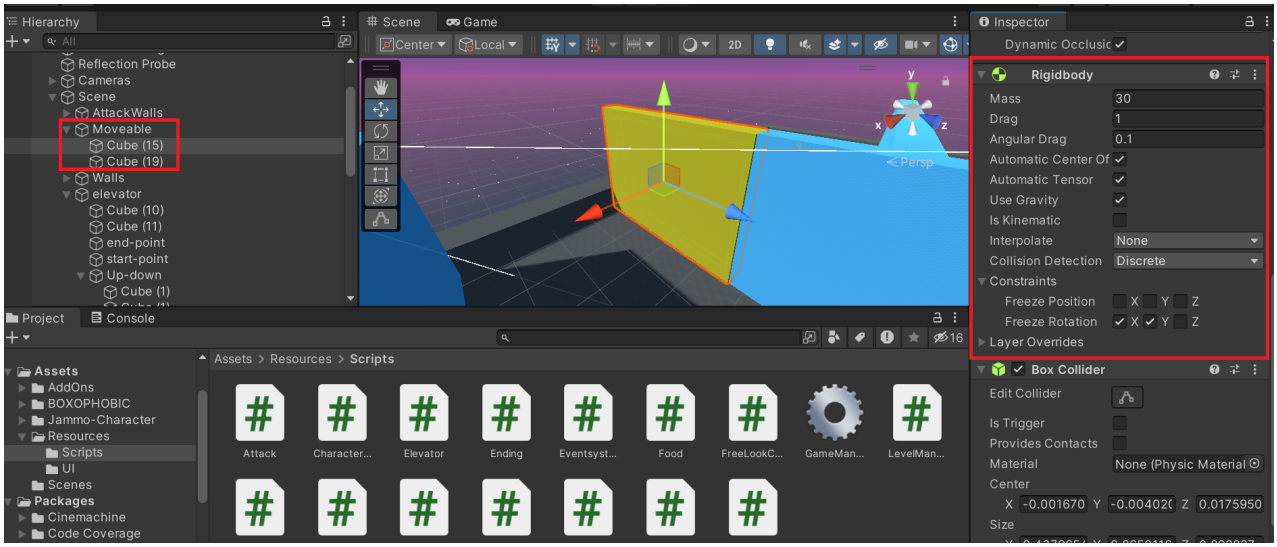


Code:

```
// In Attack.cs
private void OnTriggerEnter(Collider other)
  {
      if (other.gameObject.layer == LayerMask.NameToLayer("player"))
      {
          LevelManager.Instance.Loss();
      }
  }
```

1.4 Rigidbody

When a collision occurs, the rigid body simulates physical motion according to the defined parameter values.



Example: Collider and play sound

In script `Food.cs`:

- the method below detect the player entering the collider.
- invork the `CollectOne()` method to play the sound.
- make gameObject is inactived.

```
private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.layer == LayerMask.NameToLayer("player"))
        {
            LevelManager.Instance.CollectOne();
            gameObject.SetActive(false);
        }
     }
```

In script `LevelManager`:

- getting the audioSource in `start()` method.
- Define `CollectOne()` method to play the sound and increase the `collectedCnt`.

```
public void CollectOne()
    {
        collectedCnt += 1;
        getting.PlayOneShot(getting.clip);
    }
```

Additional Introduce

Transform: Every GameObject has a Transform component, which defines its position, rotation, and scale in the game world.

Scripts: You can attach scripts (written in languages like C#) to GameObjects to control their behavior. These scripts can manipulate components and respond to events like input, collisions, and more.

Prefab: Prefabs are a way to create reusable templates for GameObjects. You can create a prefab from a GameObject and then instantiate it multiple times in your scene.

Instantiation: Creating an instance of a GameObject in your scene is called instantiation. You can do this programmatically or by dragging a prefab into the scene.

Scene Management: GameObjects are essential for organizing and managing your game scenes. Scenes are comprised of various GameObjects that make up the visual and interactive elements of the game level.

Interaction: GameObjects can interact with each other through physics simulations, collision detection, raycasting, and events.

Game Logic: Through scripts and components, GameObjects can contain the logic that drives gameplay, character movement, AI behavior, and more.

Here's a simple example of creating a GameObject in Unity using C# script:

```
using UnityEngine;

public class ExampleScript : MonoBehaviour
{
    private void Start()
    {
        // Create a new GameObject
        GameObject cube = new GameObject("Cube");

        // Add a Mesh Renderer component to the GameObject
        cube.AddComponent<MeshRenderer>();

        // Add a Box Collider component to the GameObject
        cube.AddComponent<BoxCollider>();
    }
}
```

Remember that Unity's GameObject system is at the core of building interactive 3D and 2D games, allowing you to create complex game worlds and interactions through a combination of GameObjects and their components.

2. Event Function (Lifecycle Function)

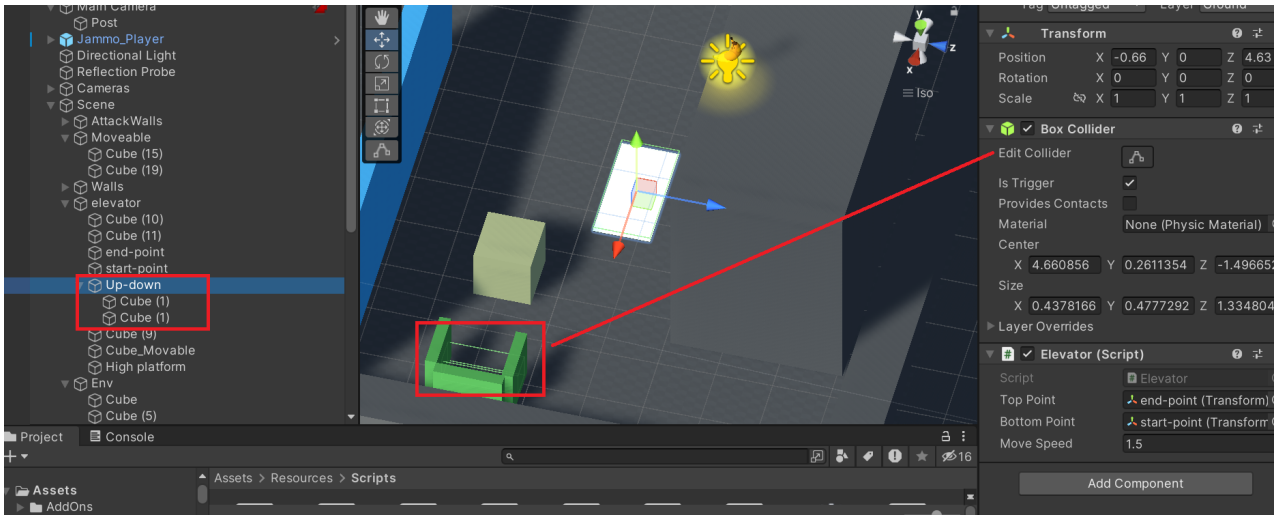
In Unity, event functions are special functions that are automatically called by the engine at specific times during the lifecycle of a game object or a scene. These functions allow developers to respond to various events and perform custom actions.

1. **Initial Function**
    - `Awake()` The first called function, would be called only once. Called when loading the scene; called when GameObject is activated; called when Instantiate is used to instantiate GameObject
    - `Start()` Default initial function. Called once before Update before the first frame
    - `OnEnable()` Repeatedly assign values to attributes and change them to the initial state. It will be called once every time the game object is activated or the script is activated
2. **Update Function**
    - `Update()` Called every frame(帧), so if you need to read the input of the player, this is the perfect function to handle it so you don’t miss any event.
    - `LateUpdate()` Executes after all the `Updates` functions have been called.
    - `FixedUpdate()` Executed at a specific rate defined in the editor, before `LateUpdate`. The method is controlled by the item's Fixed Timestep value. This value defaults to 0.02 (50Hz). To change this timestep, see "Project setting --> Time --> Fixed Timestep".
3. **End Function**

- `OnDisable()` This function is called when the behaviour becomes disabled. This is also called when the object is destroyed and can be used for any cleanup code. When scripts are reloaded after compilation has finished, OnDisable will be called.
- `OnDestory()` The scene or game ends; stopping the play mode will terminate the application; called when the webview is closed.
- `OnApplicationQuit` Exit game safely, used to handle some logic after exiting the game. Use the following function (marco) to exit safely both in editor and game.

```
private void OnExitClick(){
#if UNITY_EDITOR
        UnityEditor.EditorApplication.isPlaying = false;
#elif UNITY_STANDALONE
        Application.Quit();
#endif
}
```

Example:



In script `Elevator.cs`

- When start game, set elevator position

```
void Start()
    {
        targetPosition = topPoint.position;
    }
```

- After start the game, the update method would be called.

```
void Update()
    {
        if (flag)
        {
            MoveElevator();
        }
    }
```

- When any cube goes into the trigger(Box Collider), the method would be invoked.

```
private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.layer == LayerMask.NameToLayer("cube"))
        {
            flag = true;
        }
    }
```

3. Switch Scenes

As the following code:

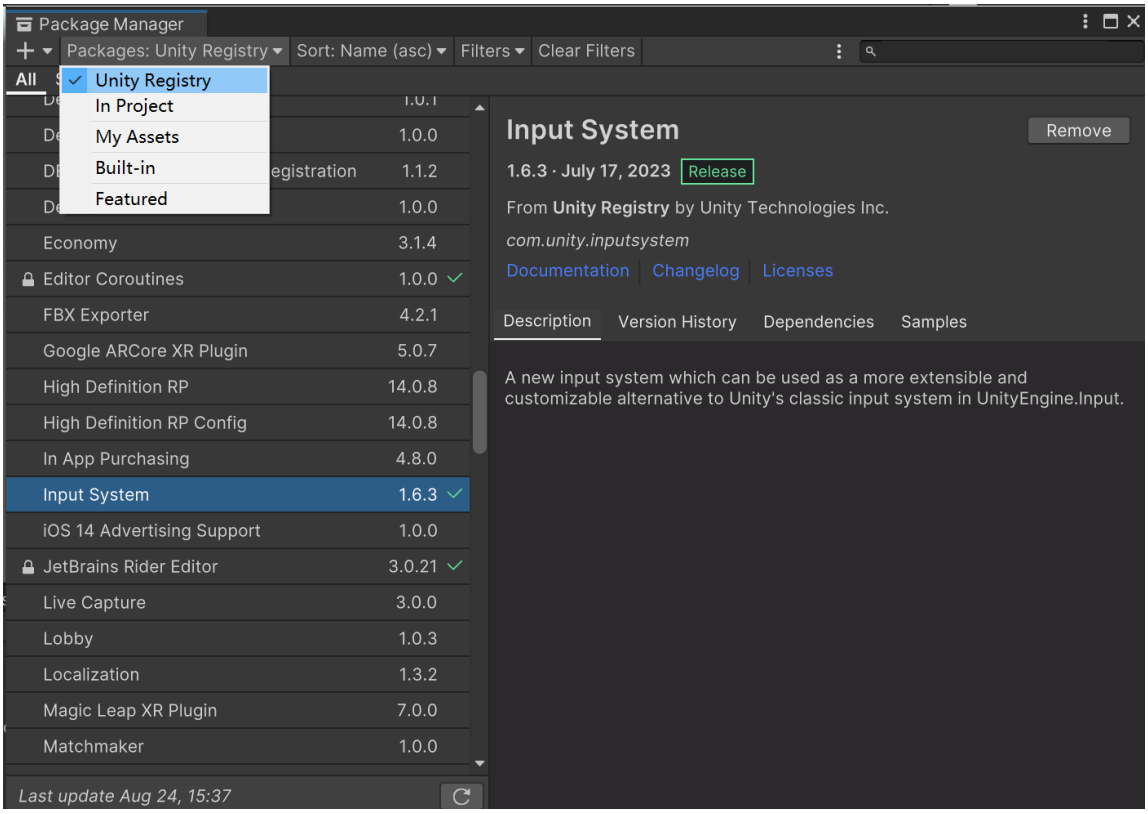
```
SceneManager.LoadScene("Your scene name");
```

Put it at where you need to change scene.

4. Input system

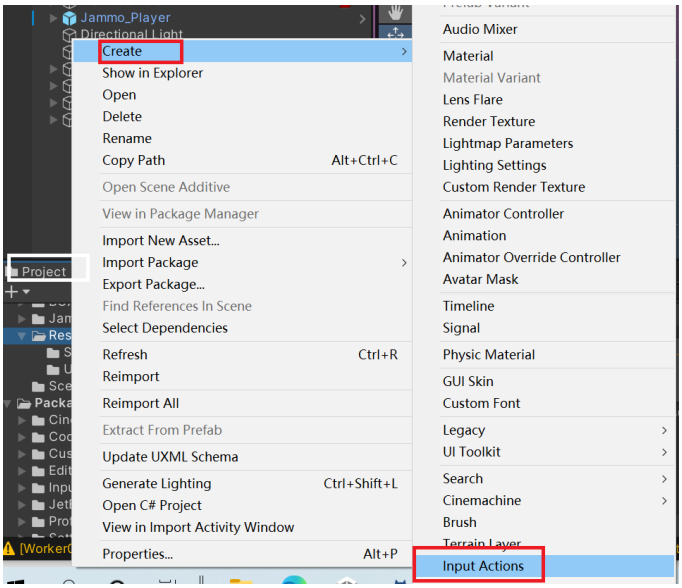
> Reference : Unity Input System 新输入系统的功能及用法介绍*unity 新输入系统*ZeryChen的博客-CSDN博客
>
> Video In Youtube: Learn Unity Beginner/Intermediate 2023 (FREE COMPLETE Course - Unity Tutorial) - YouTube Chapter-Input System Refactor Also, you can watch Chapter-Interact Actions, C# Events
>
> Video In Bilibili: 创建及配置新输入系统 New Input System | Unity2022.2 最新教程《勇士传说》入门到进阶 | 4K*哔哩哔哩*bilibili

- **Install Input System**
    1. `Window -> Package Manager -> (Unity Registry)` Find `Input System` and install. The Editor may restart.

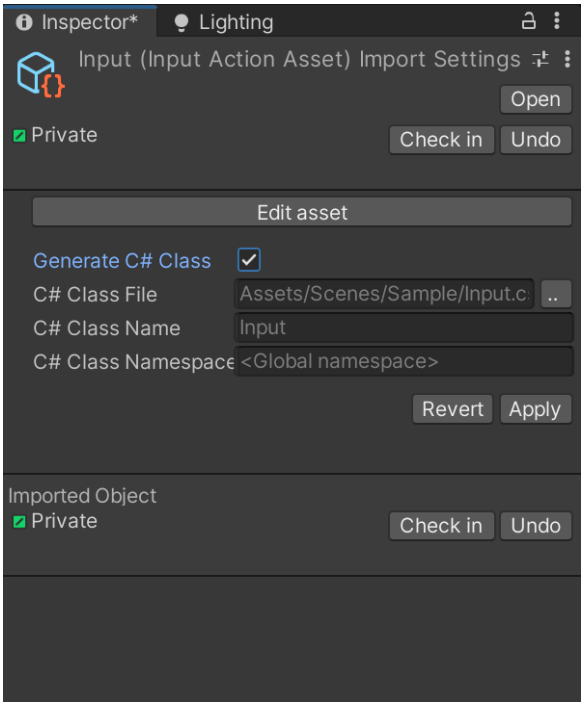


- **Create Input Actions**

1. In `Project` window, if you want to create an Input Action under any folder, right click the folder, select `Create -> Input Actions`

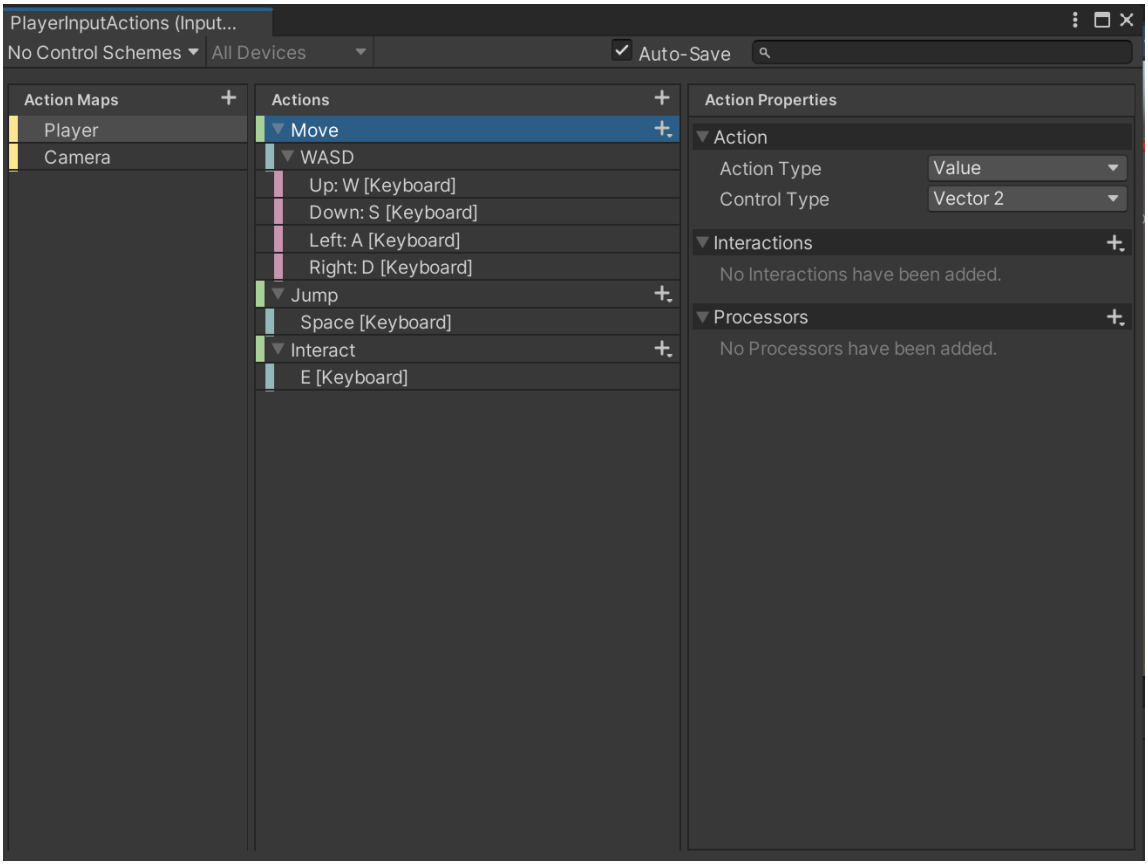


2. After rename the Input Action, click it. Look at Inspector, mark `Generate C# class` and click Apply



3. Double click the Input Action you just created (mark Auto-Save or you may forget to save). See Reference for more information about Input Action. Watch the video in Bilibili or Youtube and you can get more familiar with Input System.

   For example: In `Project` window, find `Resources` , then double click `PlayerInput`



   We can find that, there are three types of Action, in this project we use two of them:

Actions Maps	Actions	Action Type	Return Type
Player	Move	Value	Vector 2
Player	Jump	Button	Void
Player	Interact	Button	Void
Camera	Rotate	Value	Delta

4. Create a script to manage the global inputs. In Demo we use the script: `Resources -> Scripts -> PlayerInput.cs`
5. Conbination scripts:

   Value Type:

   In `PlayerInput.cs`: Always return a value.

```
 public Vector2 GetMovementVectorNormalized()
    {
        // get the value of Player.Move
        Vector2 moveVector =
inputActions.Player.Move.ReadValue<Vector2>();
        moveVector = moveVector.normalized;
        return moveVector;
    }
```

   In `Player.cs`: The returned value can be got by `playerInput.GetMovementVectorNormalized();`, and it can use the value to change player's location.

```
 private void HandleMovement()
    {
        Vector2 inputVector =
playerInput.GetMovementVectorNormalized();

        Transform mainCamera = Camera.main.transform;
        Vector3 forward = mainCamera.forward;
        Vector3 right = mainCamera.right;
        ......
```

   Button Type

   In `PlayerInput.cs` :

   Add button performed listener

```
inputActions.Player.Jump.performed += Jump_performed;
```

   Add method:

```
private void
Jump_performed(UnityEngine.InputSystem.InputAction.CallbackContext obj)
{
        OnJumpPerformed?.Invoke(this, EventArgs.Empty);
    }
```

In `Player.cs`: All actions that occur when clicking space are implemented in this method.

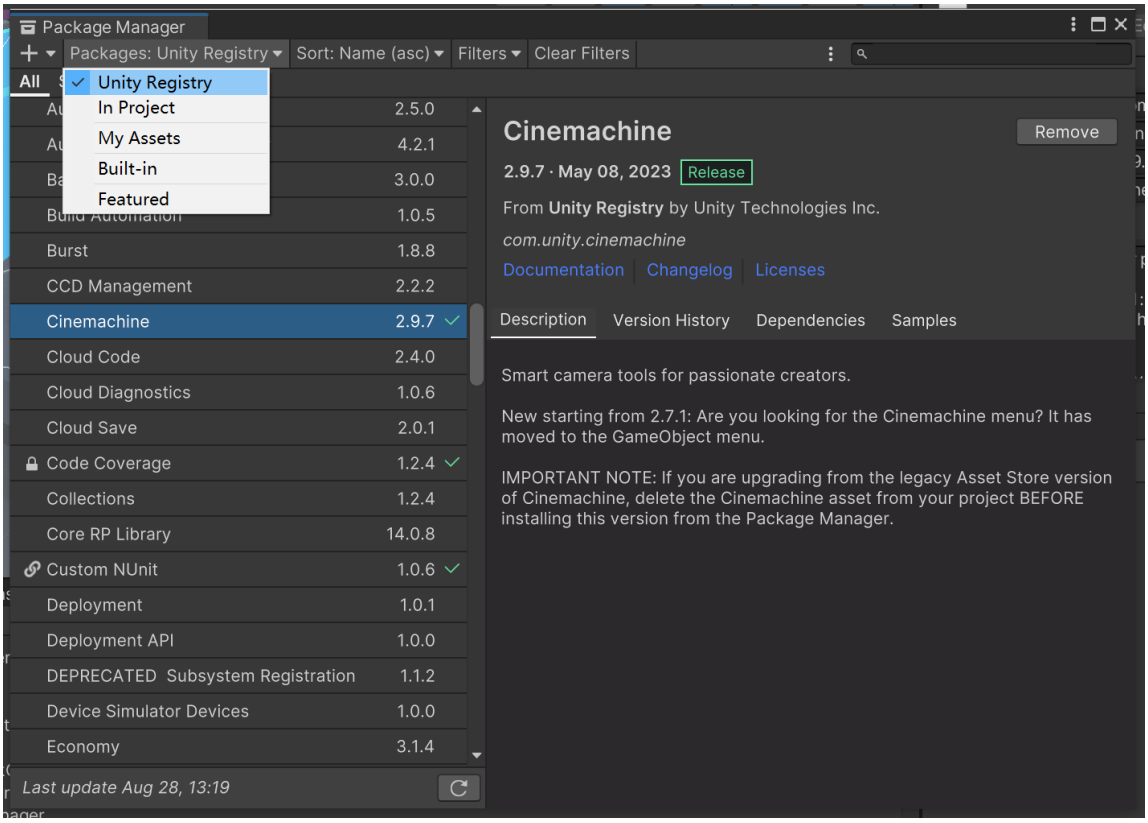
```
private void PlayerInput_OnJumpPerformed(object sender,
System.EventArgs e){
        // TODO: Implement Jump, when you press key 'Space'.
        // In this function, you're noticed by your Input System that
key 'Space' is pressed.
        // The animator is prepared. When you handle jump, you should
set a trigger named 'Jump' by animator.SetTrigger() function
        // Remember to check isGrounded

    }
```

5. Third Person (第三人称)

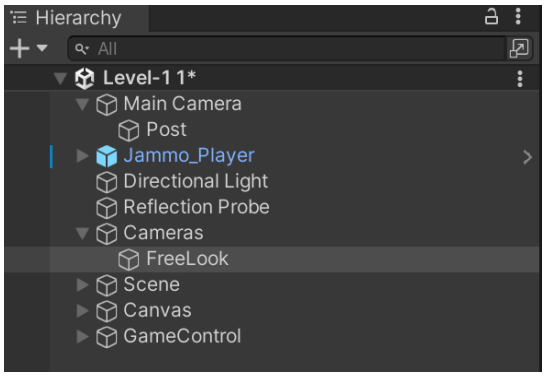
Cinemachine

> Reference: 【游戏开发教程】Unity Cinemachine快速上手，详细案例讲解（虚拟相机系统 | 新发出品 | 良心教程）_林新发的博客-CSDN博客
>
> Unity Official Reference: About Cinemachine | Cinemachine | 2.9.7 (unity3d.com)
>
> Video in Youtube: How to use Cameras in Unity: Cinemachine Overview and Brain Explained! - YouTube / How to use Cinemachine's Free Look Camera | 3rd Person Camera in Unity - YouTube

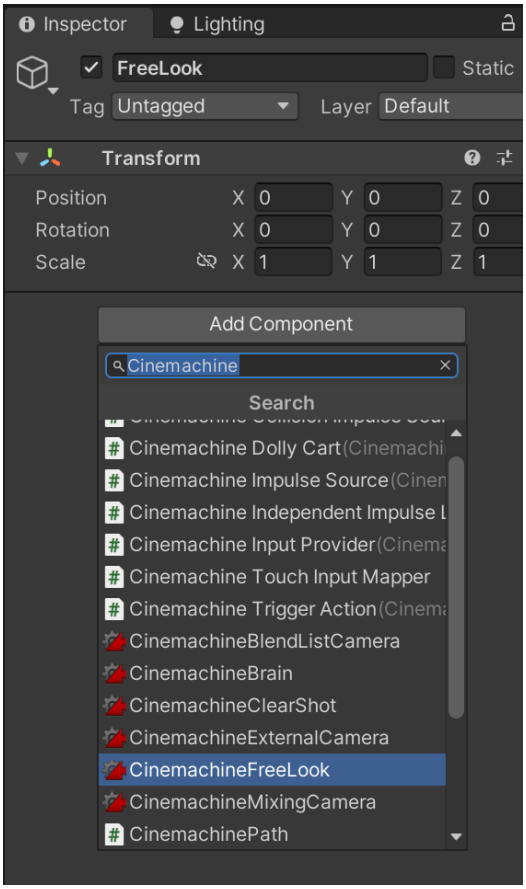
- **Install Cinamachine**
    1. `Window -> Package Manager -> (Unity Registry)` Find `Cinemachine` and install. The Editor may restart.



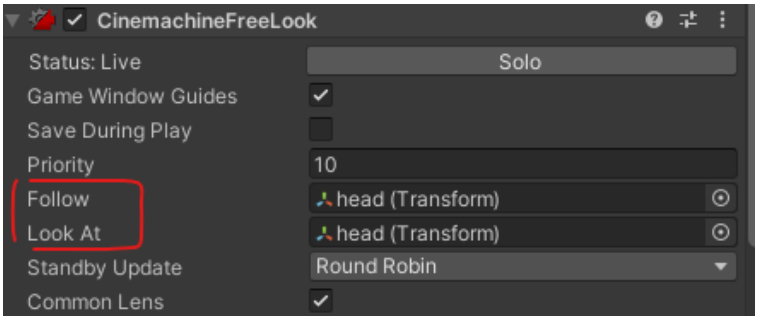
- **Create FreeLook**
  1. Create an **empty GameObject** in the scene



  2. Add a component `CinemachineFreeLook` for the GameObject



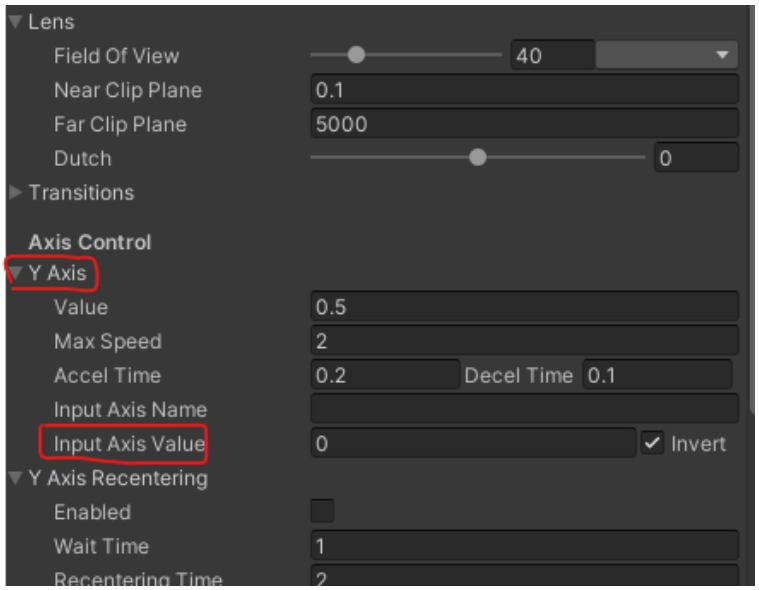
3. Refer to Reference or Video in Youtube to learn how to set `follow` and `look at` properties



4. Control the rotation of FreeLook

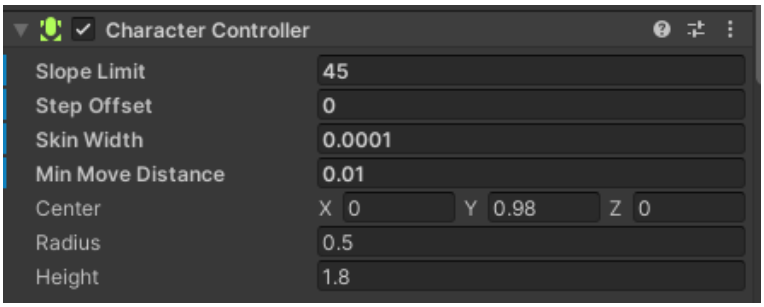
```
// In FreeLookController.cs
private void HandleRotate()
{
    // Get the mouse input from your Input System
    Vector2 cameraRotateVec = playerInput.GetCameraRotateNormalized();

    // Control the properties of FreeLook
    freeLook.m_XAxis.m_InputAxisValue = cameraRotateVec.x;
    freeLook.m_YAxis.m_InputAxisValue = cameraRotateVec.y;
}
```

You may notice that `m_YAxis` and `m_InputAxisValue` looks like the properties in the Editor. Therefore, you could control the properties of FreeLook by adding `m_` before the name of property.



Character Controller

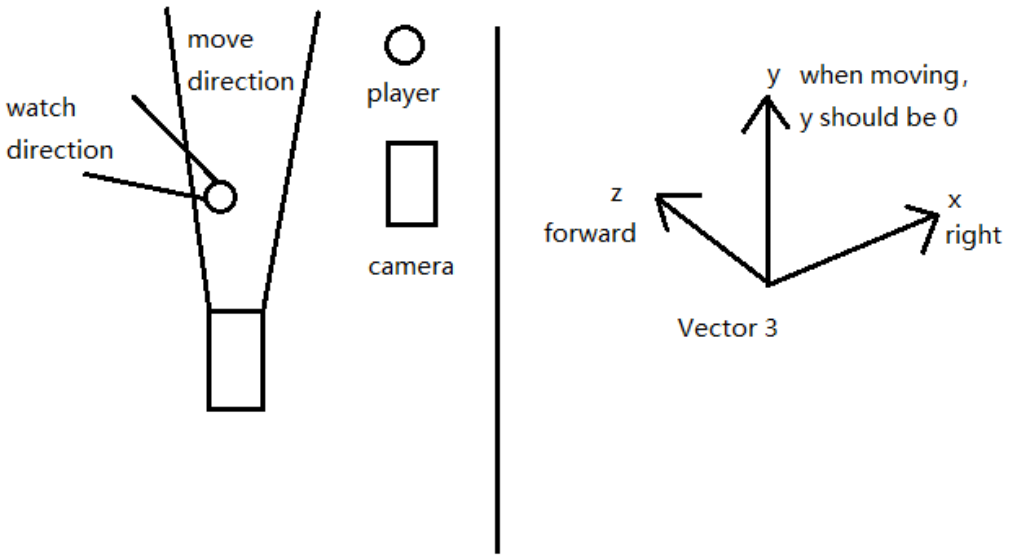
- **Character Controller Component**



- **Control using Scripts**

```
// In Player.cs
private void Awake()
{
    // Audio Controller
    Audio = GetComponent<AudioSource>();
    // Character Controller
    controller = GetComponent<CharacterController>();
    // Animator, control the animation
    animator = GetComponent<Animator>();
}

private void Update()
{
    HandleMovement();
}
```

```
private void HandleMovement()
{
    // Get Input from your Input System
    Vector2 inputVector = playerInput.GetMovementVectorNormalized();

    Transform mainCamera = Camera.main.transform;
    Vector3 forward = mainCamera.forward;
    Vector3 right = mainCamera.right;

    // Lock the Move Direction in the same y-axis(height)
    // so that Player can move On the x-z plane
    forward.y = 0;
    right.y = 0;

    Vector3 moveDirection = forward * inputVector.y + right *
inputVector.x;
    moveDirection = moveDirection.normalized;

    // Rotate the Player. The forward direction of Player should be the
same as Camera
    transform.forward = Vector3.Slerp(transform.forward,
                                      moveDirection,
                                      rotateSpeed * Time.deltaTime);
    controller.Move(moveDirection * moveSpeed * Time.deltaTime);

    float speed = inputVector.sqrMagnitude;
    // The walk animation will auto-adjust according to Player's speed
    animator.SetFloat("Blend", speed, StartAnimTime, Time.deltaTime);
}
```



6. GUI

You could find more at https://unity3d.com/learn/tutorials/topics/user-interface-ui.

Canvas

The three modes in **Render Mode** determine the way the Canvas is treated and rendered in the game. Let's explore these options further to understand them better.

1. Unity 3D Canvas: Screen Space - Overlay

   This render mode places UI elements at the top of the screen of the scene rendered. It's what you'll be using most of the time when you're making an HUD or any similar, static UI element. It's quite useful because it automatically scales UI elements in that Canvas to the required size as per the screen's size.

2. Unity 3D Canvas: Screen Space

   This is quite similar to how the **Overlay** mode works, but in this mode, we specify a camera which renders our UI. As such, changing properties in the camera like its shape, size, area of coverage or resolution will change the way the UI looks on the screen. Developers don't usually use this option, as they find Overlay to be of better use because of its ability to auto-scale UI elements.

3. Unity 3D Canvas: World Space

   This mode is quite different from the other two. In World Space, UI elements are treated simply as another gameObject in the scene instead of having priority rendering on top. World Space is very useful when dealing with UI elements that are part of the game itself, and not simply for the player.

Anchor and Pivot

- **Anchor**

  UI Anchors position is based on the parent game object. So, if your UI game object is a child of Canvas then the Anchor position will change based on the Canvas scale.

  In Unity there are two kinds anchor mode, Center Anchor and Corner Anchor. In the first case, where the Anchors are set to the center the UI element will not scale based on the canvas size but remain at a fixed distance from the center.

- **Pivot**

  Pivot is the center point for all scaling and rotation of the UI element. You can set the pivot anywhere you want but remember all the values in the rect transform that you have set while configuring the anchor refers to the pivot.

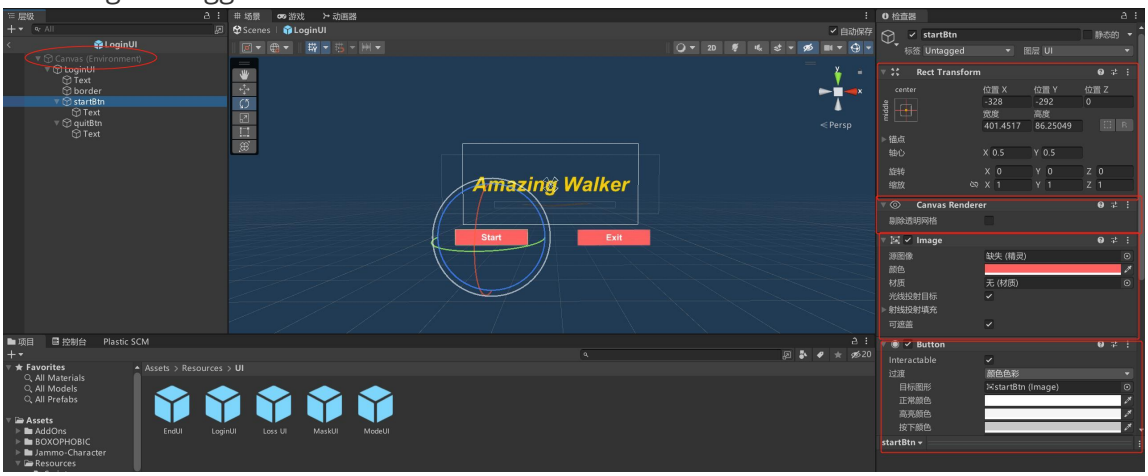
Tips: In Rect Transform bar, click the icon, then push "Shift" to select pivot, push "alt" to set position as well.

Demo

To create a UI interface with buttons and input fields in Unity, you can follow these steps:

1. **Create Canvas and UI Elements**:
   - Right-click in the Hierarchy panel, select "UI," and choose "Canvas" to create a Canvas.
   - Create various UI elements under the Canvas, such as buttons and input fields. Right-click the Canvas again, select "UI," and then choose the desired UI elements like "Button" and "Input Field."
2. **Adjust the Position and Size of UI Elements**:
   - Select each UI element and use the RectTransform component in the Inspector panel to set the element's position, size, and alignment. You can use Anchors, Pivot, and Position to adjust the layout.
3. **Add Text and Images**:
   - For buttons, you can add text within the button to give it a descriptive label.
   - Customize the button's appearance by setting images in the button's Image component.
4. **Add Button Event Handlers**:
   - Select the button, then in the Inspector panel, find the "OnClick()" event of the Button component.
   - Click the "+" button to add a target by dragging it into the event list or selecting a function.
   - Create a function to handle the button's click event and assign it to the button's click event.
5. **Add Input Field Event Handlers**:
   - Select the input field, then in the Inspector panel, find the "End Edit" event of the Input Field component (triggered when the user finishes input and presses Enter).
   - Similar to the button, add a target by dragging it into the event list or selecting a function.
   - Create a function to handle the input field's input completion event and assign it to the "End Edit" event.

6. **Write Corresponding Scripts**:
    - Create a C# script to handle the logic of button clicks and input field completion events.
    - Define the necessary functions in the script, such as handling button clicks and input field completion.
7. **Associate Scripts with UI Elements**:
    - Attach the script to an appropriate GameObject (such as the Canvas).
    - In the Inspector panel, drag the script onto the corresponding event handlers of UI elements.
8. **Write Logic**:
    - Write the logic for button click events and input field completion events in the script. For example, you can perform certain actions in button click events or retrieve user input text in input field completion events.
9. **Test the Interface**:
    - Run the game to see the UI interface in action.
    - Click the buttons and type text into the input fields to ensure that the corresponding event logic is triggered.



By following these steps, you can create a UI interface with buttons and input fields in Unity and implement interactive logic by writing scripts. Be sure to adjust the layout, style, and event handlers of UI elements according to your specific requirements.

Scripts:

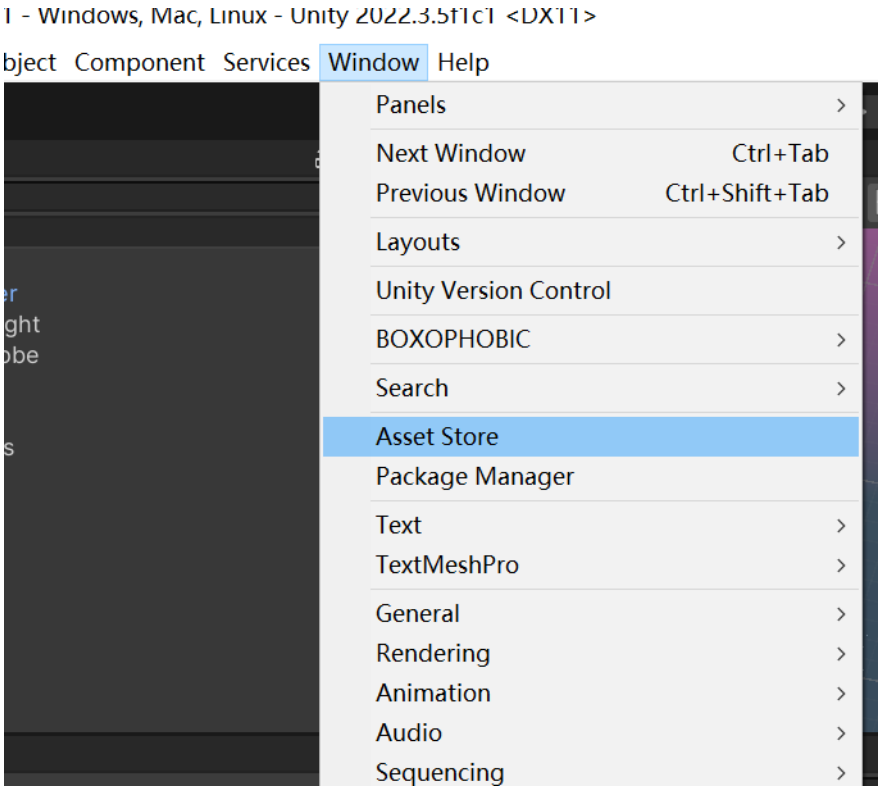
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class LoginUI : MonoBehaviour
{
    void Start()
    {
        transform.Find("startBtn").GetComponent<Button>
().onClick.AddListener(onStartBtn);
        transform.Find("quitBtn").GetComponent<Button>
().onClick.AddListener(onQUITBtn);
```

```
    }

    private void OnEnable()
    {
    }
    private void OnDisable()
    {
    }
    public void onStartBtn()
    {
        GameManager.uimanger.CloseAllUI();
        GameManager.uimanger.ShowUI<ModeUI>("ModeUI");
    }
    public void onQUITBtn()
    {
        Application.Quit();
    }
}
```

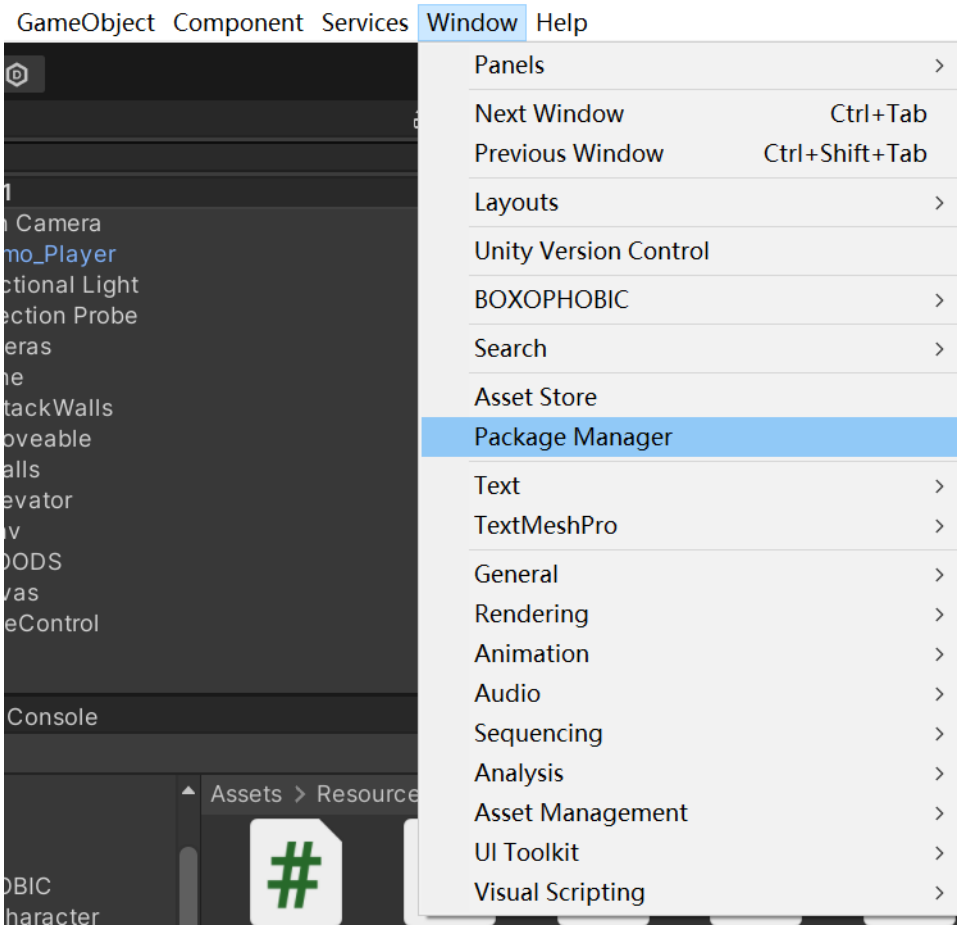
7. Asset store

https://assetstore.unity.com/ is the unity asset score link.

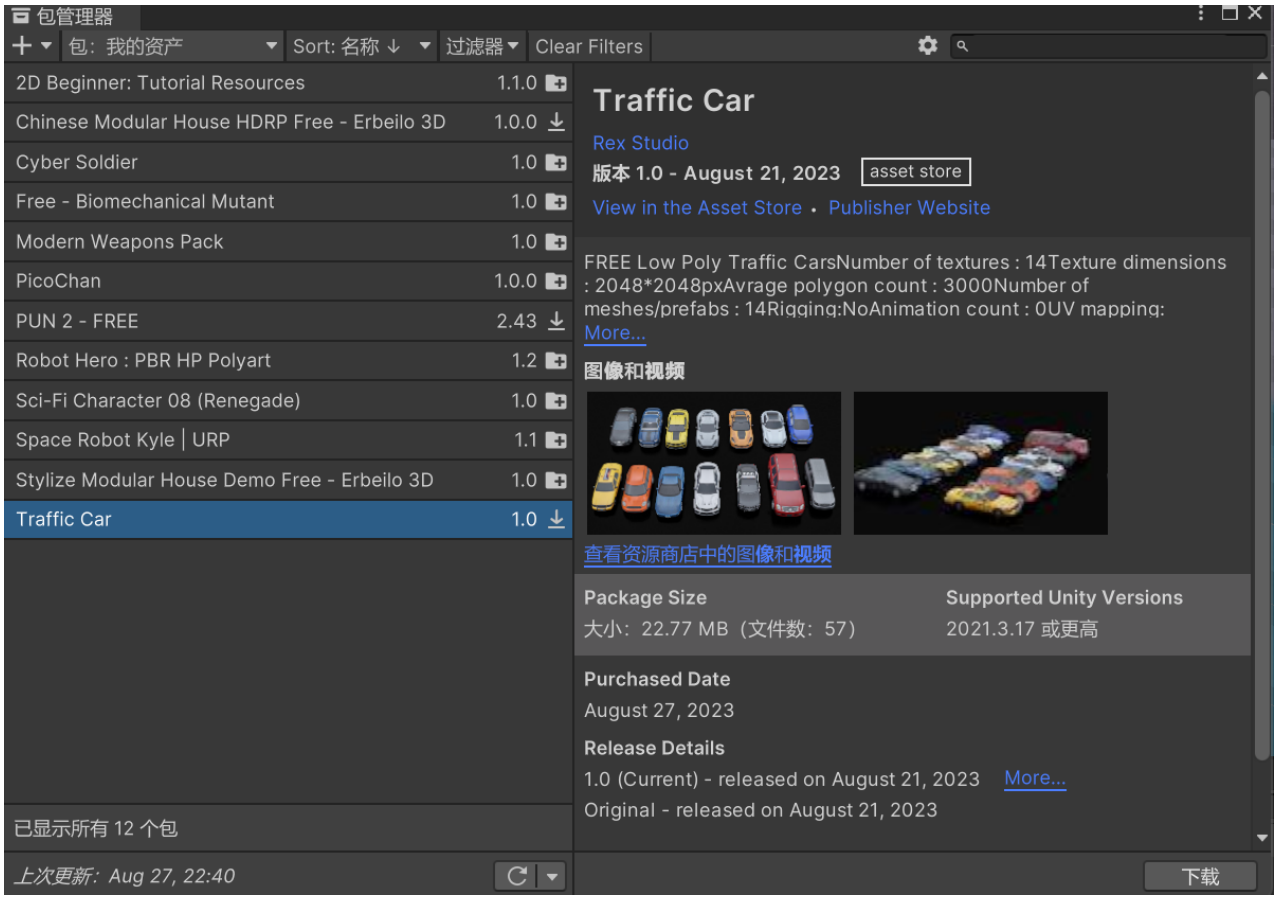
After choosing an asset, click the blue button, and this asset will be added to your package.



And then in your unity, click the "包管理器" (package manager) in the "窗口" (window)



And then you can see the asset in the `package: My Assets`

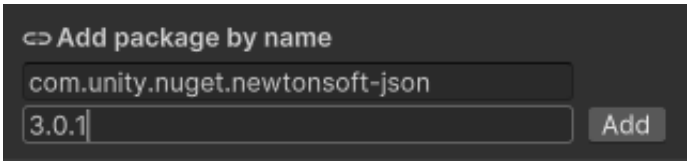


Use URL updating newer package in UPM

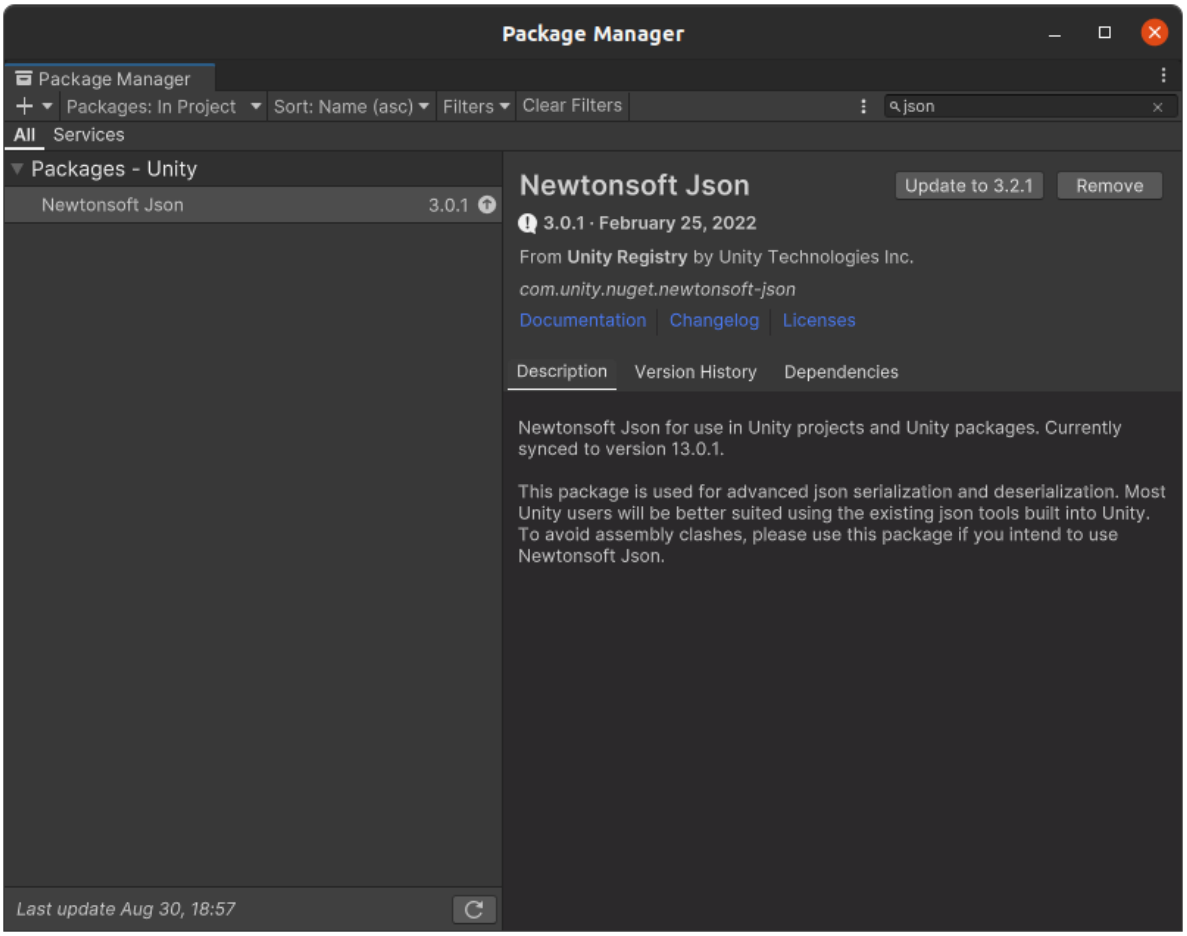
Here we suppoet to use UPM (Unity Package Manager), or you could build from source code, make your customized package as well.

1. Open UPM window.
2. Click "add package"" --> click "Add package by name".

   If you are using an older Unity version, you may not have the "Add package by name" option here. In this case, instead follow the guide installing via `manifest.json` below.
3. Enter `URL` as package name, and `version number`.)



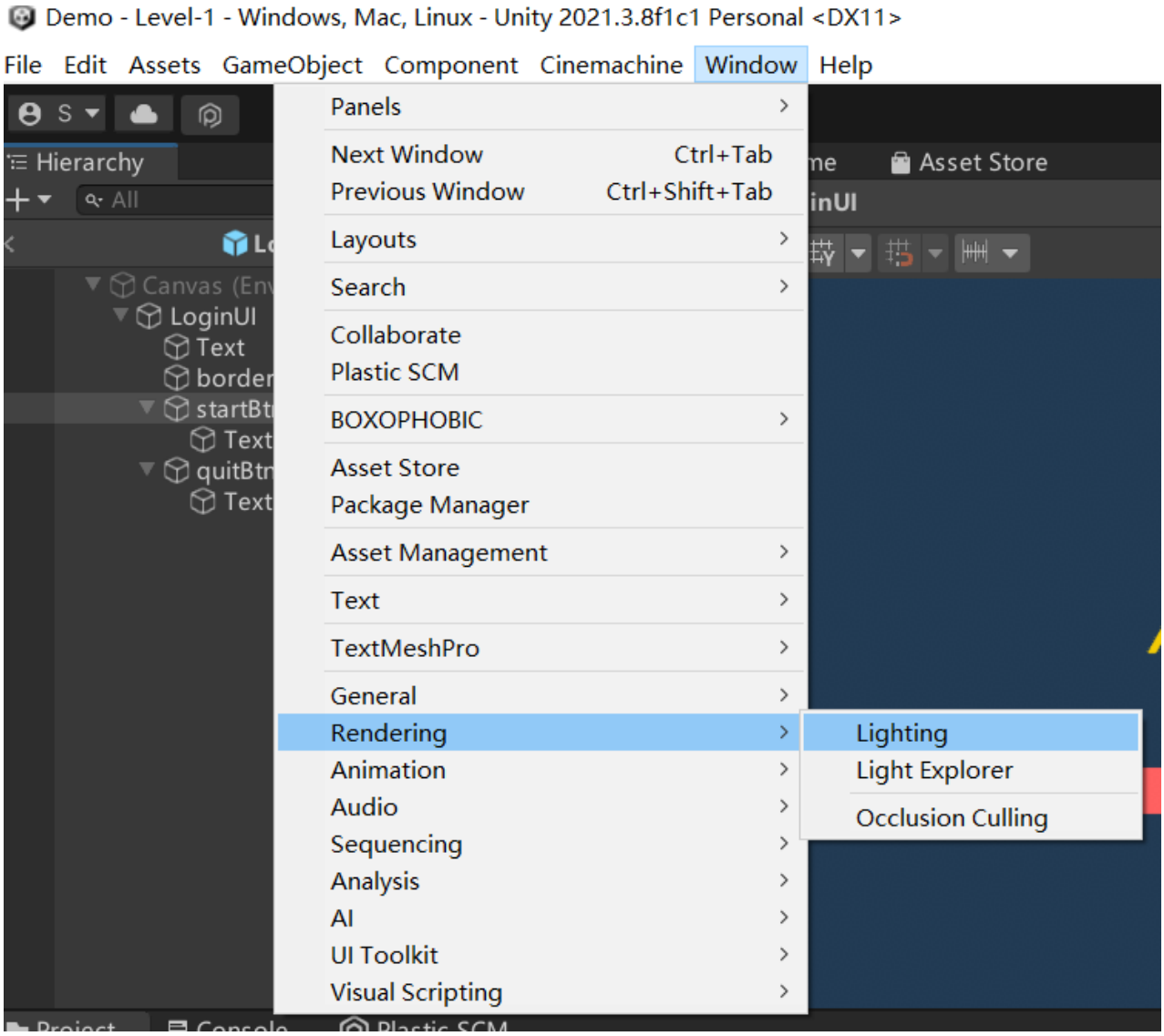
4. Then you could find the newer package now.

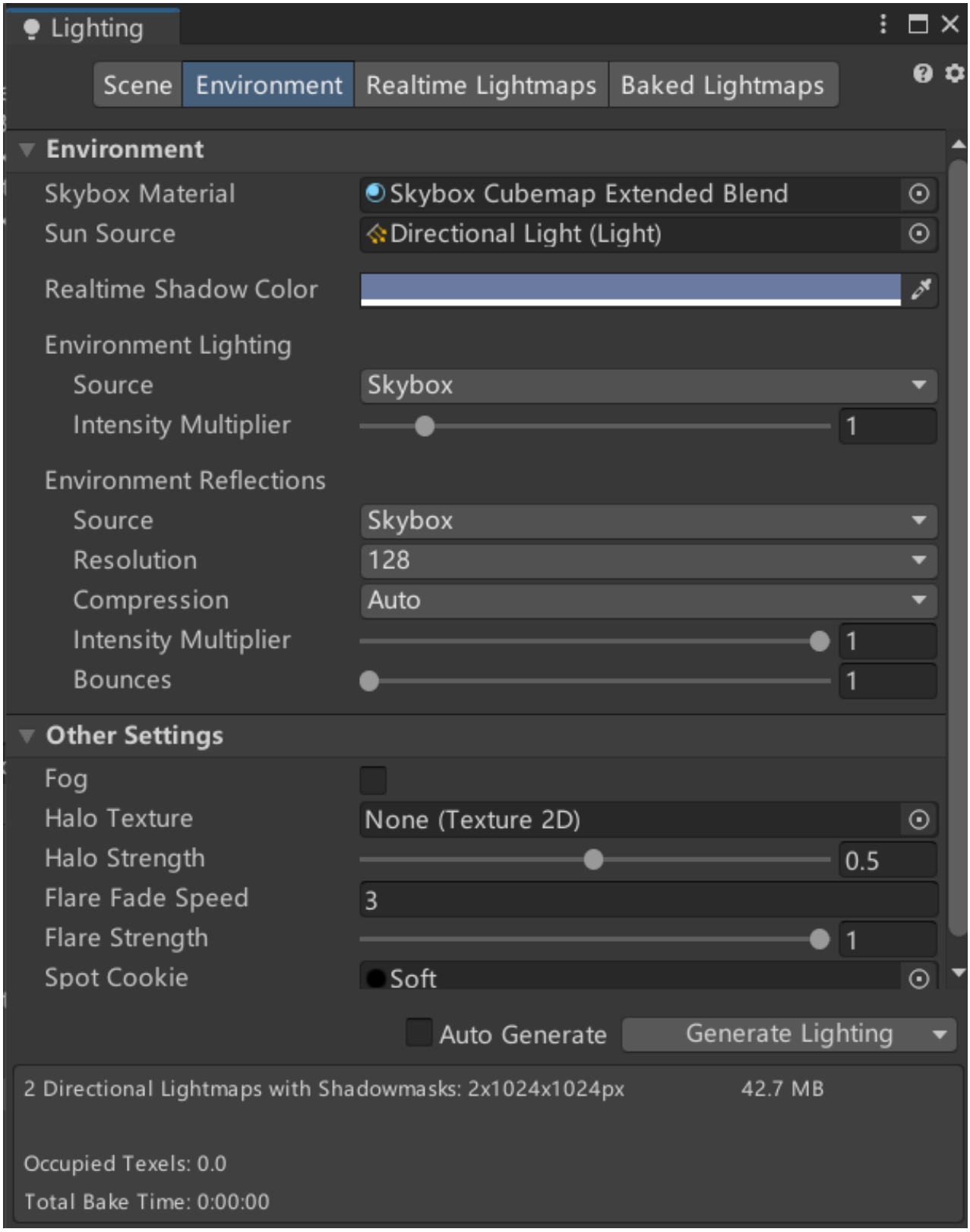


8. Skybox

A Skybox in Unity is a 360-degree background that surrounds your game world, creating the illusion of a vast and immersive environment. It's a large cube with images on its inner faces, simulating distant scenery such as sky, mountains, and landscapes. The Skybox provides a backdrop to your scenes, enhancing the visual experience and adding depth to your game's ambiance.

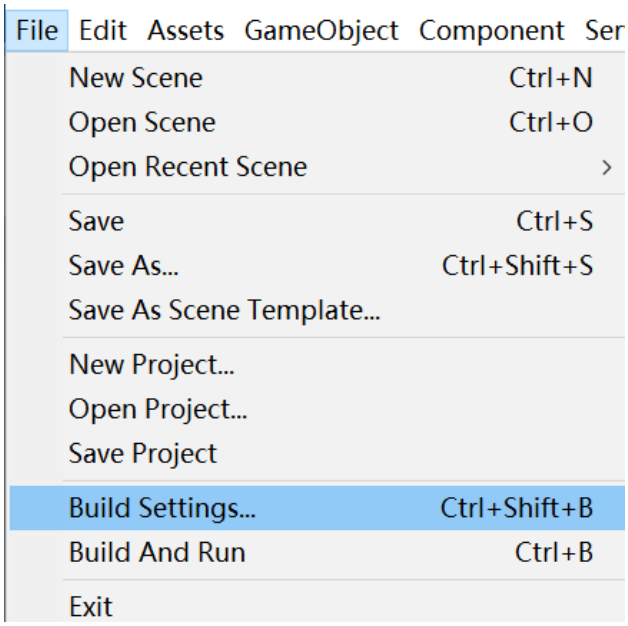
You can set your sky box by following steps:



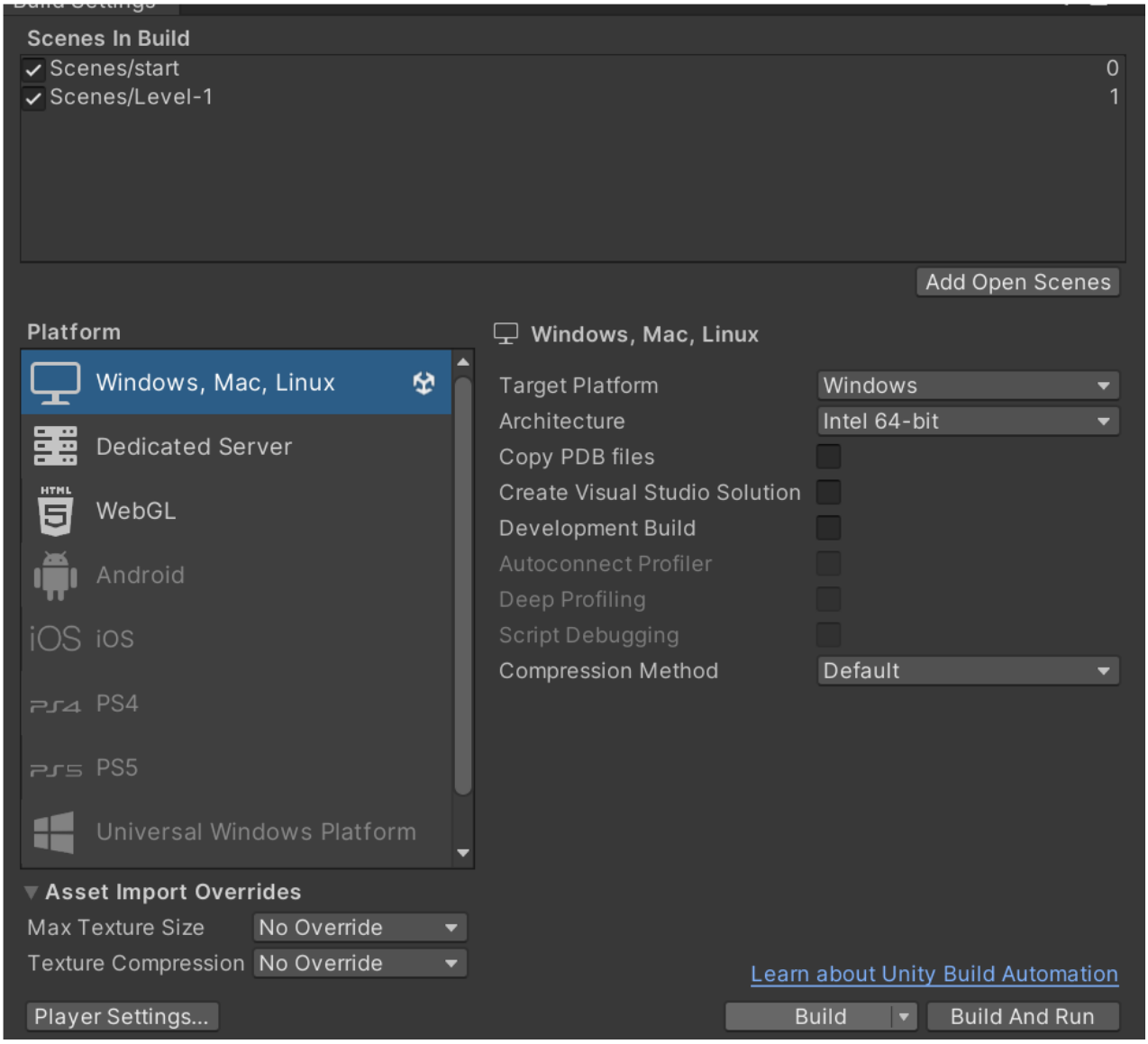


Generate the executable file

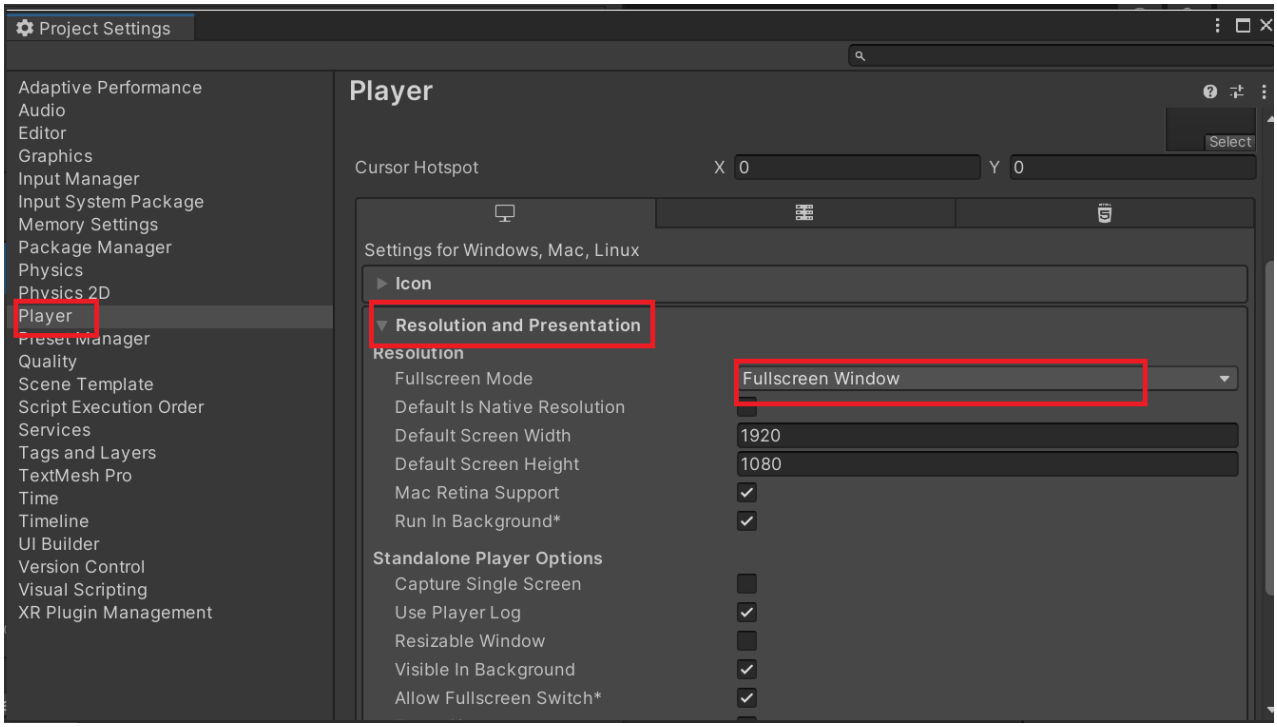
First go to "文件", find "生成设置" (Build Setting)



And click "生成", Then an exe file will be in the folder you selected.



If you want to set "Full Screen", click "玩家设置" on the "生成设置", and you can see the setting.



In linux platform, the file type should be `file.x86_64`, the output include a dirctionary of resources, a executable file and a library file named `UnityPlayer.so`.

```
output_dir
├── output_Data
├── output.x86_64
└── UnityPlayer.so
```