Computer Graphics 1

Tutorial

Assignment 1

Summer Semester 2024

Ludwig-Maximilians-Universität München

Contact

If you have any questions:

cg1ss24@medien.ifi.lmu.de

Organization

- **Theoretical part:**
  - Prepares you for the exam
  - Solve the tasks at home
  - We present the solutions during the tutorial sessions
- **Practical part:**
  - We will indicate which parts you need to do at home
  - Else this will be done & explained during the tutorial session
  - Ask questions!

Task 1A: Dot Product

Step 1: Definition of Dot Product

The dot product (also known as the scalar product) of two vectors $\mathbf{a}$ and $\mathbf{b}$ is calculated by multiplying corresponding components of the vectors and then summing up these products.

Step 2: Identify Components

Given the vectors:

$$\mathbf{a}=(2,3,5,1)$$

$$\mathbf{b}=(6,7,9,8)$$

Step 3: Multiply Corresponding Components

Multiply each component of vector $\mathbf{a}$ with the corresponding component of vector $\mathbf{b}$:

- $2 \times 6 = 12$
- $3 \times 7 = 21$
- $5 \times 9 = 45$
- $1 \times 8 = 8$

Step 4: Sum the Products

Add all the individual products together: $12 + 21 + 45 + 8 = 86$

The dot product of vectors $\mathbf{a}$ and $\mathbf{b}$ is 86.

Task 1B: Cross Product

Step 1: Definition of Cross Product

The cross product (also known as the vector product) of two vectors $\mathbf{c}$ and $\mathbf{d}$ results in a vector that is perpendicular to both $\mathbf{c}$ and $\mathbf{d}$. The formula for the cross product in three-dimensional space is given by:

- Formula: $c \times d = (c_y d_z - c_z d_y, c_z d_x - c_x d_z, c_x d_y - c_y d_x)$

Step 2: Identify Components

Given the vectors: $\mathbf{c}=(2,3,5)$ $\mathbf{d}=(6,1,2)$

Task 1B: Cross Product

Step 3: Calculate Each Component of the Cross Product

- First Component: $c_y d_z - c_z d_y$ $\quad 3\times2-5\times1 = 6-5 = 1$
- Second Component: $c_z d_x - c_x d_z$ $\quad 5\times6-2\times2 = 30-4 = 26$
- Third Component: $c_x d_y - c_y d_x$ $\quad 2\times1-3\times6 = 2-18 = -16$

The cross product of vectors $\mathbf{c}$ and $\mathbf{d}$ is $(1,26,-16)$.

Task 2A: Homogeneous Coordinates in 3D Graphics

- **Introduction to Homogeneous Coordinates**
  - Homogeneous coordinates are a system of coordinates used in projective geometry, which allows for the convenient representation of points at infinity and simplifies mathematical formulas when applying transformations such as translations, rotations, and scaling.

- **Representation of 3D Points**
  - In standard Cartesian coordinates, a point in 3D space is represented as (X, Y, Z).
  - In homogeneous coordinates, an extra dimension (W) is added, and the point is represented as (X, Y, Z, W).
  - For typical 3D operations, W is set to 1. Thus, a 3D point (X, Y, Z) becomes (X, Y, Z, 1) in homogeneous coordinates.

Task 2A: Homogeneous Coordinates in 3D Graphics

- **Advantages of Using Homogeneous Coordinates**
  - Simplifies matrix operations: All affine transformations (translation, scaling, rotation) can be performed with matrix multiplication without needing special handling for translations.
  - Handles points at infinity: By setting W to 0, the coordinates represent directions or points at infinity, useful in perspective projections and rendering.
- **Example**
  - Standard 3D point: (2, 3, 4)
  - Homogeneous representation: (2, 3, 4, 1)
  - If this point is at infinity in the direction of vector (2, 3, 4), it would be represented as (2, 3, 4, 0).

Task 2B: Translating a Vector Along the X-axis

1. **Understanding Vector Translation**
   - Translation is a geometric transformation that moves every point of a shape or object by the same distance in a given direction.
   - Translation involves shifting a vector along the x-axis by adding a random value to its x-coordinate. This transformation does not affect the y-coordinate.
2. **Given Vectors**
   - Vector A: (2, 2)
   - Vector B: (12, 8)
3. **Translation Process**
   - Choose a random value to shift each vector along the x-axis.
   - For example, let's randomly select to translate Vector A by 3 units and Vector B by 5 units.
4. **Calculation of New Coordinates**
   - New Coordinates for Vector A:
   - $(2+3, 2) = (5, 2)$
   - New Coordinates for Vector B:
   - $(12+5, 8) = (17, 8)$

Task 2C: Rotating Vectors 90° Around the Y-axis

1. **Understanding Rotation Around an Axis**
   - Rotation around the y-axis means spinning the vector around the y-axis, which affects the x and z coordinates
2. **Given Vectors in 3D for Clarity**
   - For the purpose of rotation around the y-axis, let's consider the vectors in 3D.
   - Vector A: (3, 0, 0) — Originally along the x-axis
   - Vector B: (0, 0, 3) — Originally along the z-axis

Task 2C: Rotating Vectors 90° Around the Y-axis

3. **90° Rotation Matrix Around the Y-axis**

- The rotation matrix for a 90-degree rotation around the y-axis in a right-handed coordinate system is:

$$\begin{bmatrix} \cos 90^\circ & 0 & \sin 90^\circ \\ 0 & 1 & 0 \\ -\sin 90^\circ & 0 & \cos 90^\circ \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix}$$

4. **Applying the Rotation**

$$\text{New Coordinates for Vector A: } \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 3 \end{bmatrix}$$

$$\text{New Coordinates for Vector B: } \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 3 \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \\ 0 \end{bmatrix}$$

Task 2D: Scaling Vectors

1. **Understanding Scaling**
   - Scaling is a transformation that changes the size of an object by enlarging or reducing all of its coordinates by a consistent factor. It does not alter the shape's proportions.
   - Scaling factor of 2 means each dimension of the vector is multiplied by 2, effectively doubling its size.
2. **Given Vectors**
   - Vector A: (4, 0, 9)
   - Vector B: (5, 0, 8)
3. **Scaling Matrix for a Factor of 2**
   - This matrix doubles each coordinate of the vector.
   - The scaling transformation matrix for a 3D vector scaled by a factor of 2 on all axes is: $$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$
4. **Applying the Scaling**
   - New Coordinates for Vector A: $$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} 4 \\ 0 \\ 9 \end{bmatrix} = \begin{bmatrix} 8 \\ 0 \\ 18 \end{bmatrix}$$
   - New Coordinates for Vector B: $$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} 5 \\ 0 \\ 8 \end{bmatrix} = \begin{bmatrix} 10 \\ 0 \\ 16 \end{bmatrix}$$

Task 2E: Resulting Matrix

1. **Recap of Individual Transformations:**

   - Scaling Matrix (S) - Scales all coordinates by 2:
   - Rotation Matrix (R_y) - 90-degree rotation around the y-axis:
   - Translation Matrix (T) - if no translation, effectively identity matrix(here: assumed translation along x-axis):

$$S = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad R_y = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T = \begin{bmatrix} 1 & 0 & 0 & 5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Task 2E: Resulting Matrix

Step 2: Multiply Rotation Matrix by Scaling Matrix

The rotation matrix R_y multiplies the scaling matrix S. This step applies scaling first, then rotates the scaled coordinates:

$$R_y \times S = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 2 & 0 \\ 0 & 2 & 0 & 0 \\ -2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Task 2E: Resulting Matrix

Step 3: Multiply Translation Matrix by Result from Step 2

Now, we apply the translation matrix T to the result from Step 2. This incorporates the translation into the scaled and rotated object:

$$T \times (R_y \times S) = \begin{bmatrix} 1 & 0 & 0 & 5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 2 & 0 \\ 0 & 2 & 0 & 0 \\ -2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 2 & 5 \\ 0 & 2 & 0 & 0 \\ -2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

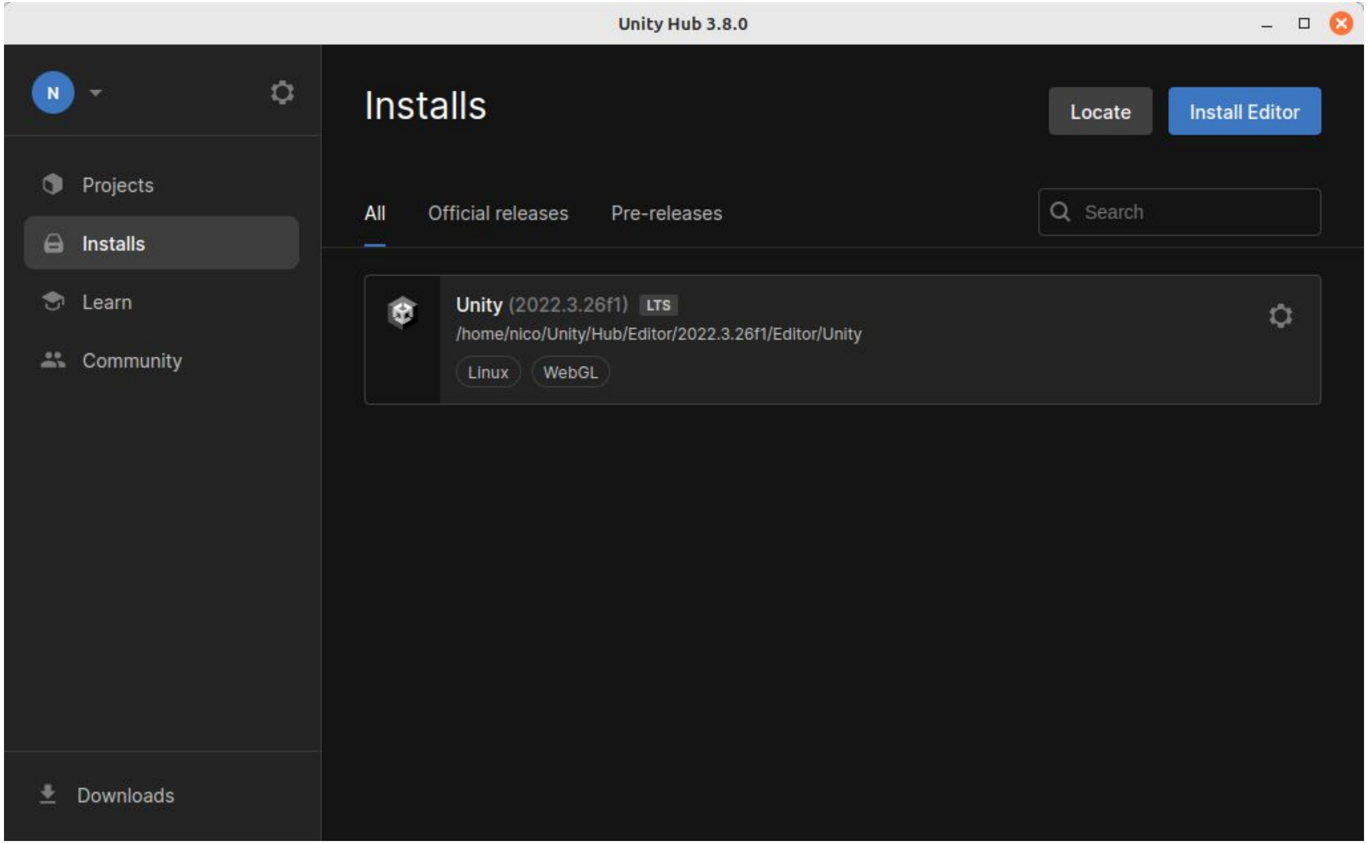
The final matrix multiplication results in a matrix that:

- Order matters: in computer graphics rotation and scaling are usually done with respect to the origin of the coordinate system
- Applies a scaling factor of 2 to all coordinates.
- Rotates the result 90 degrees around the y-axis, swapping the scaled x and z values and reversing the new x.
- Translates the final coordinates by adding 5 to the new x values (which are produced by the z values due to rotation).

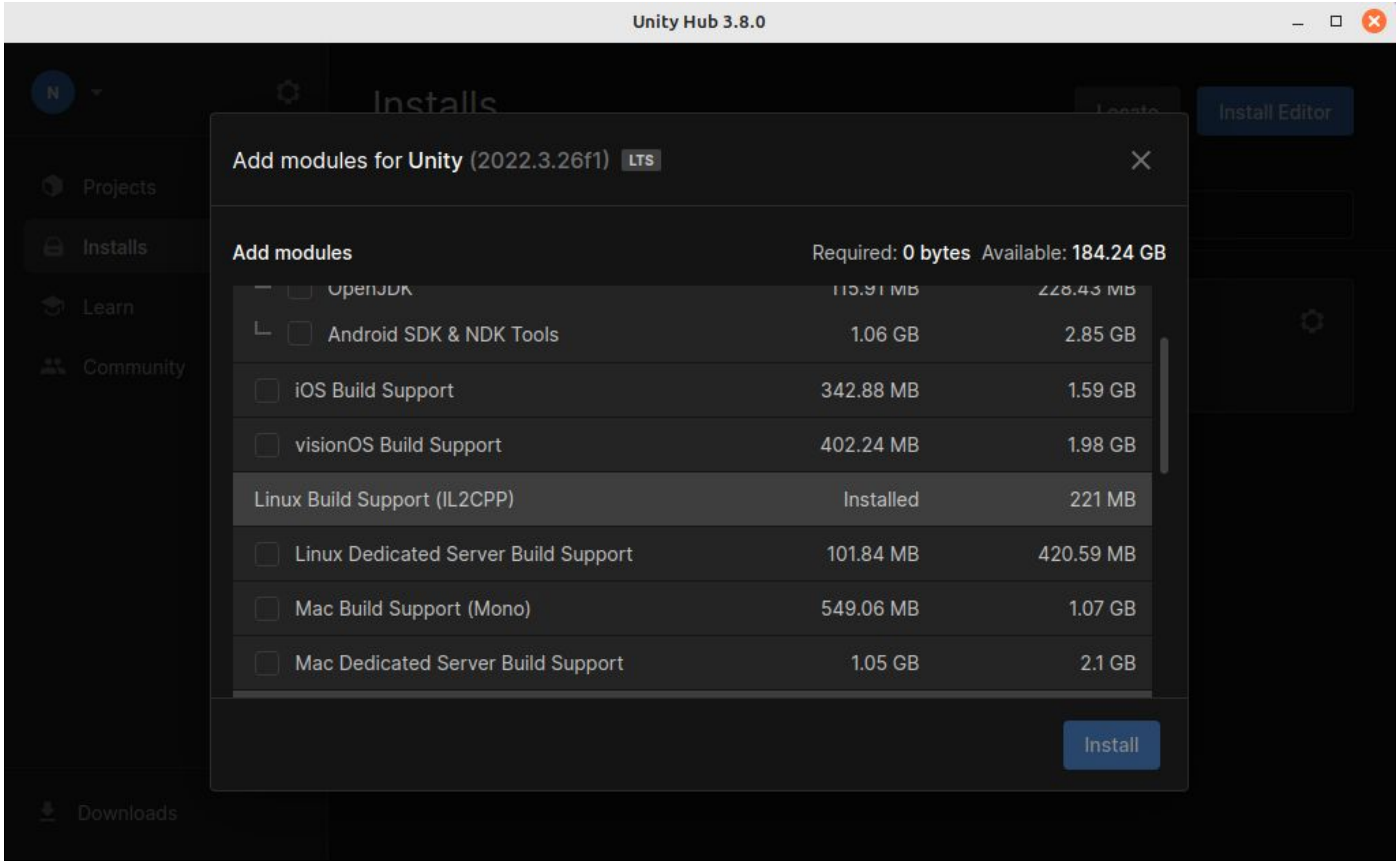
Unity Installation

- Download Unity Hub
- Start Unity Hub
  - Installs
    - Install Editor
      - 2022.3.26f1 (LTS)
        - Add module for build support
          - Install
            - Wait and start editor

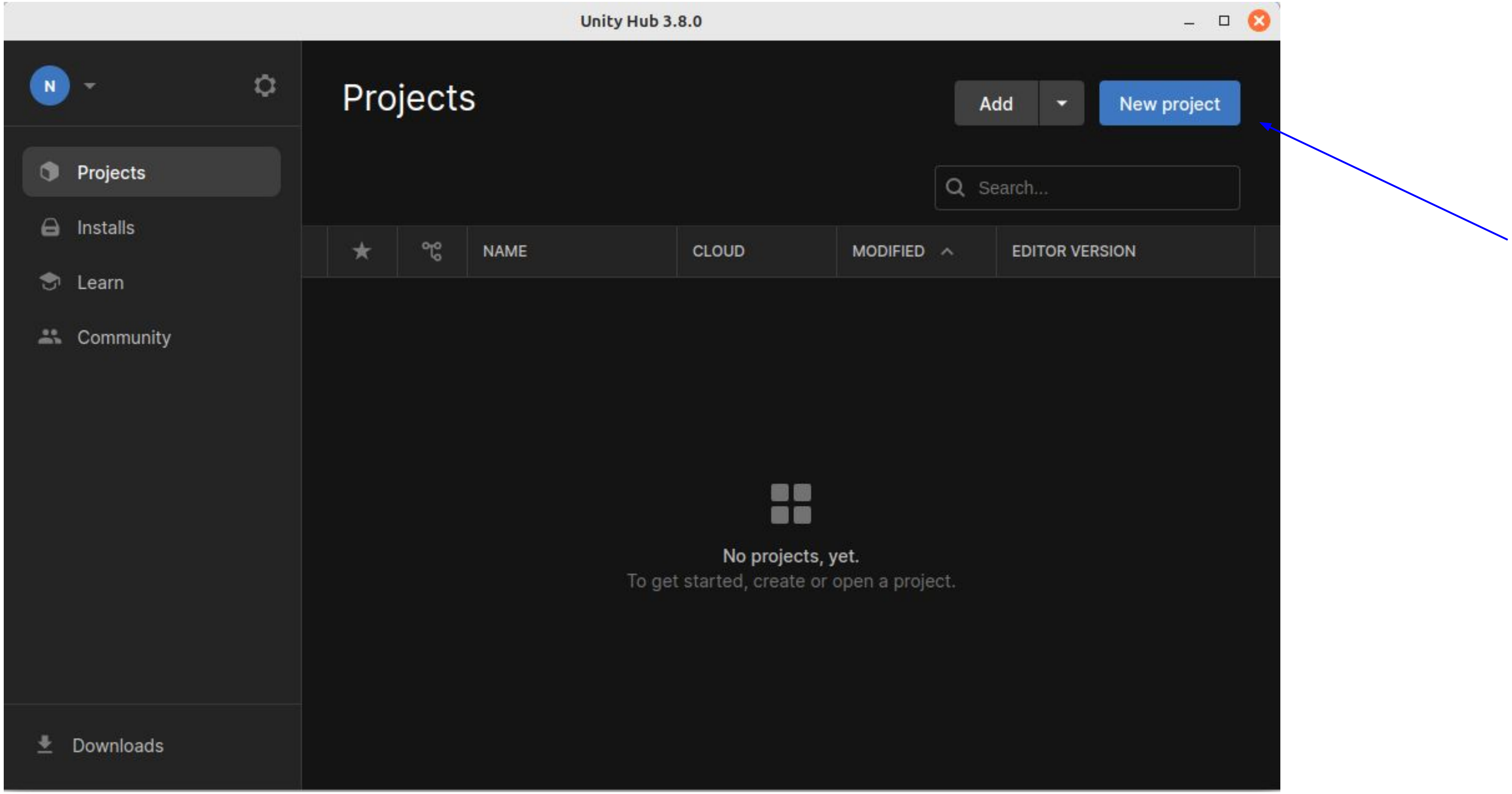
Unity Installation: what it should look like



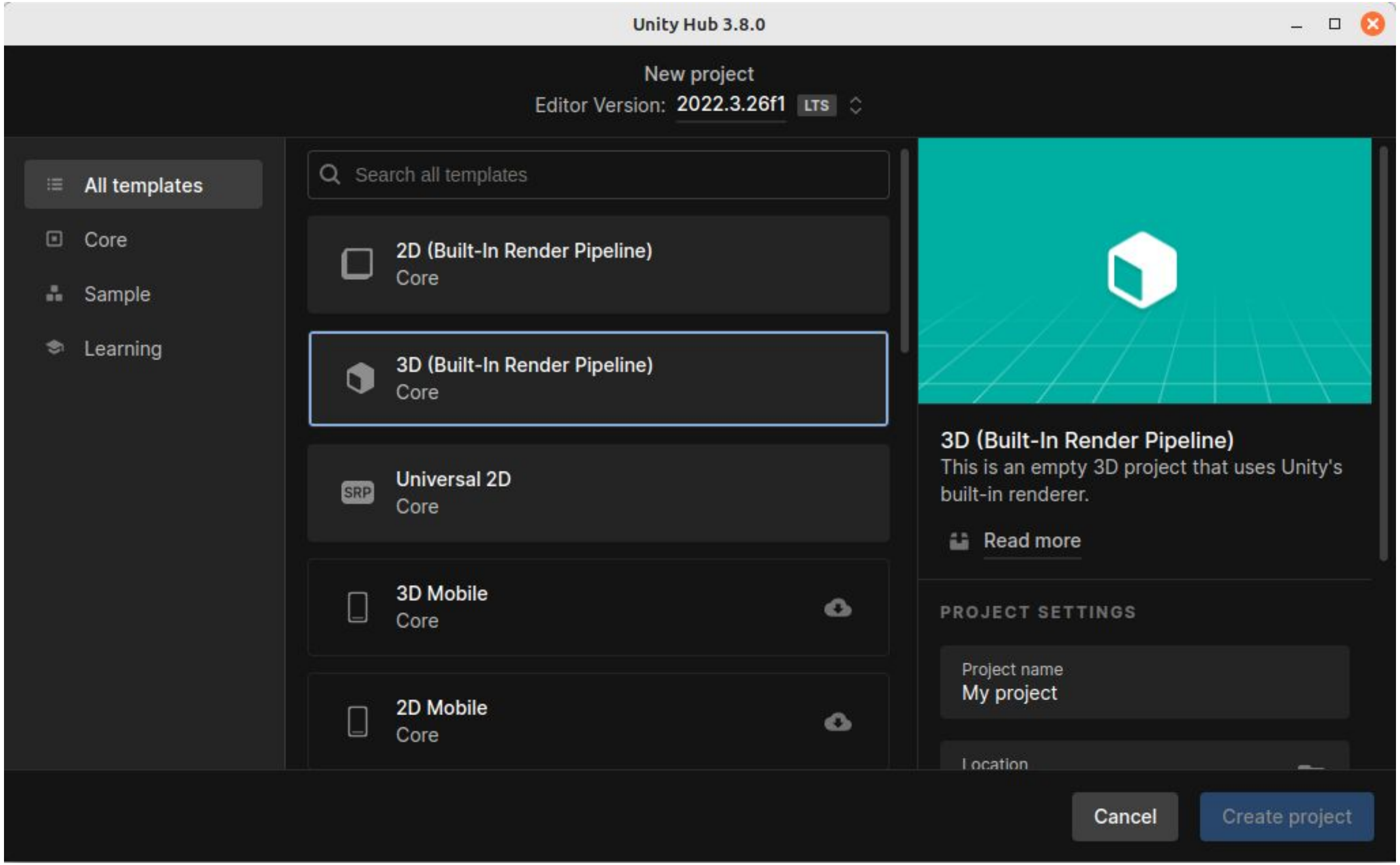
Unity Installation: what it should look like



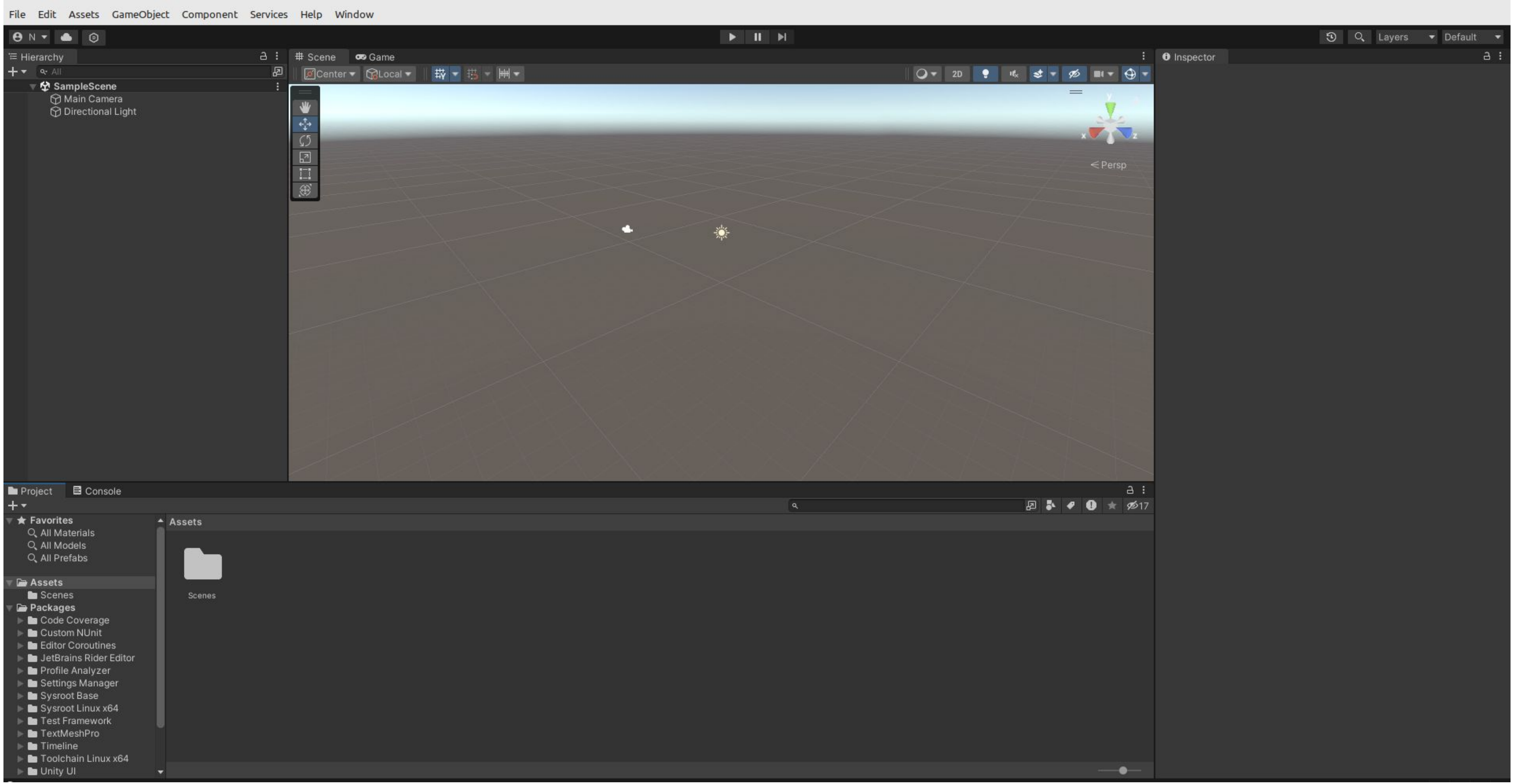
Unity: New Project



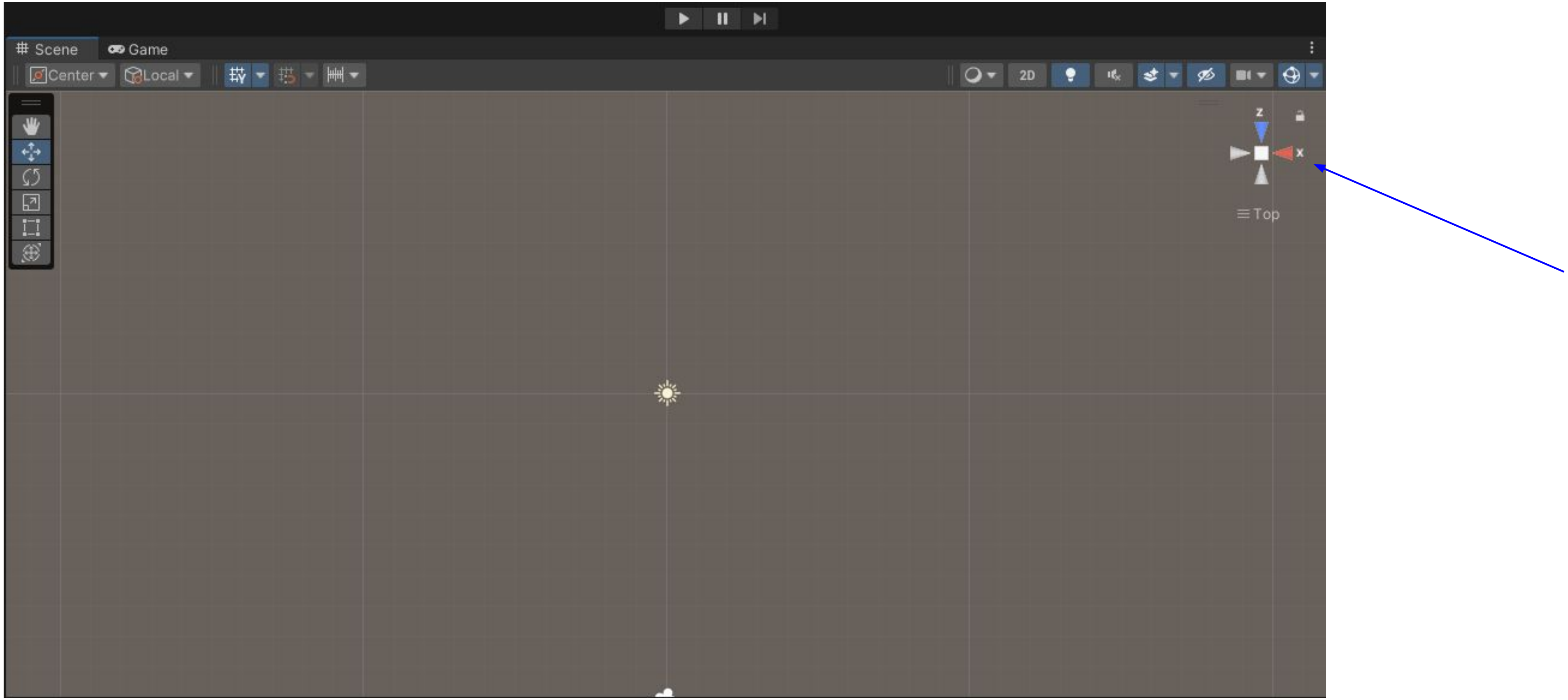
Unity: New Project



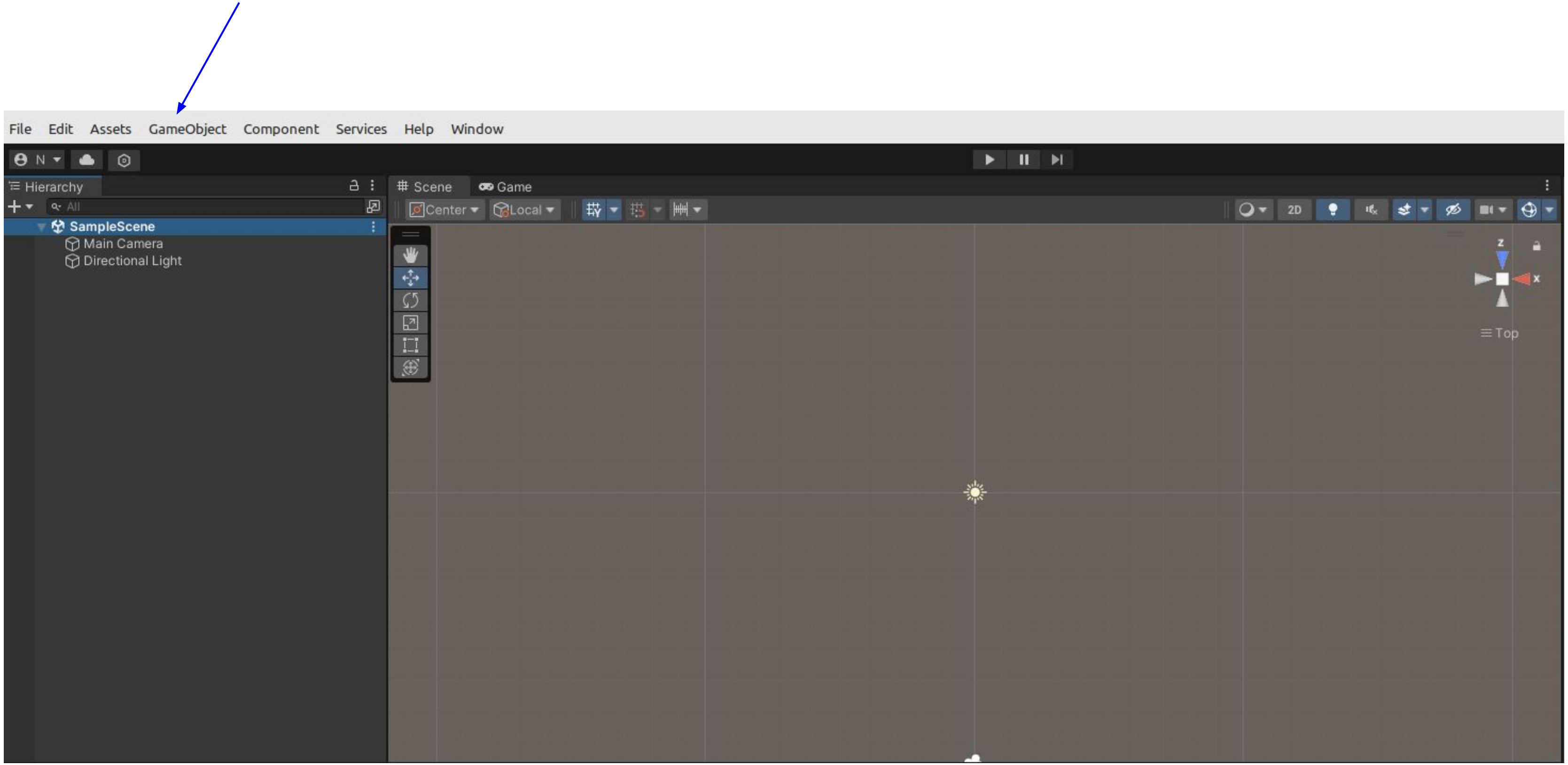
Unity: New Project



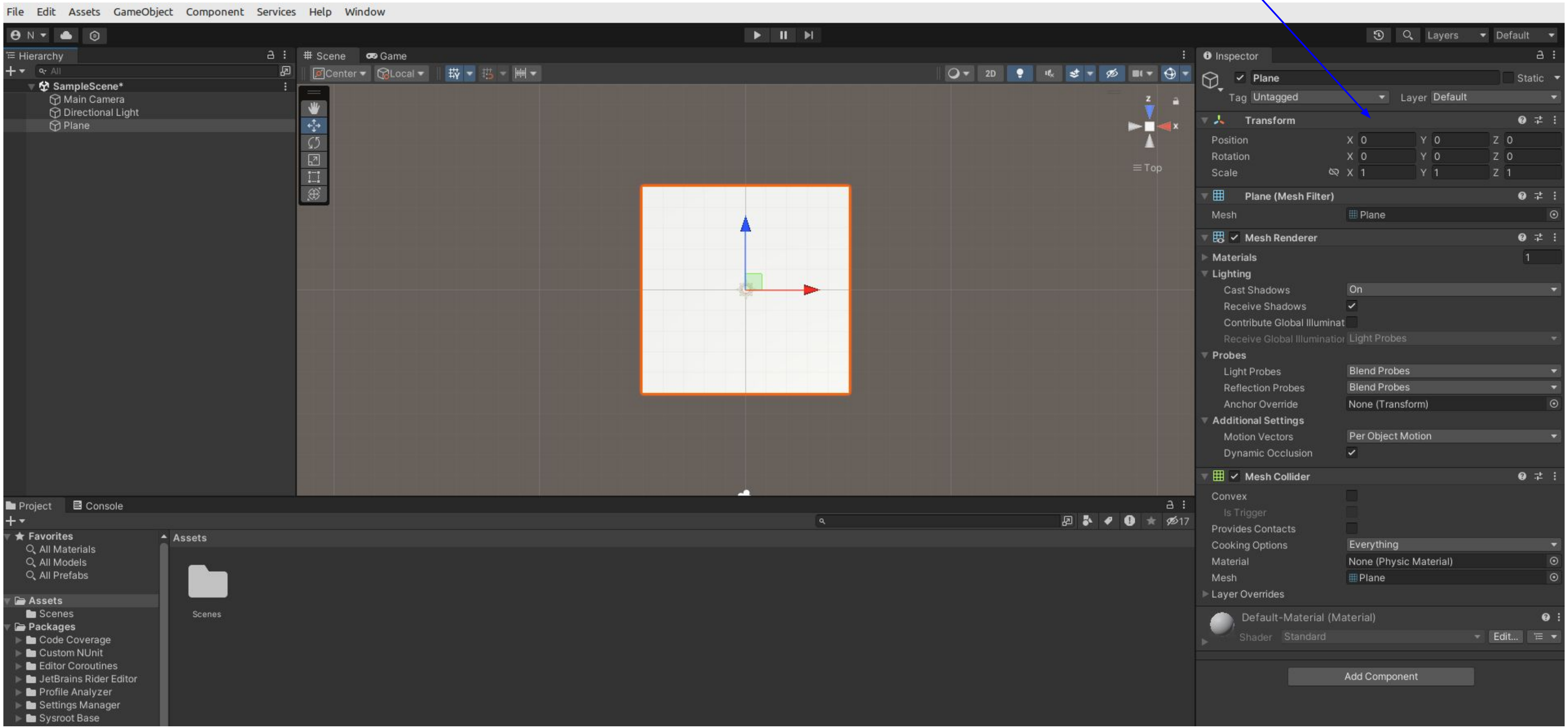
Task 3B: Top View



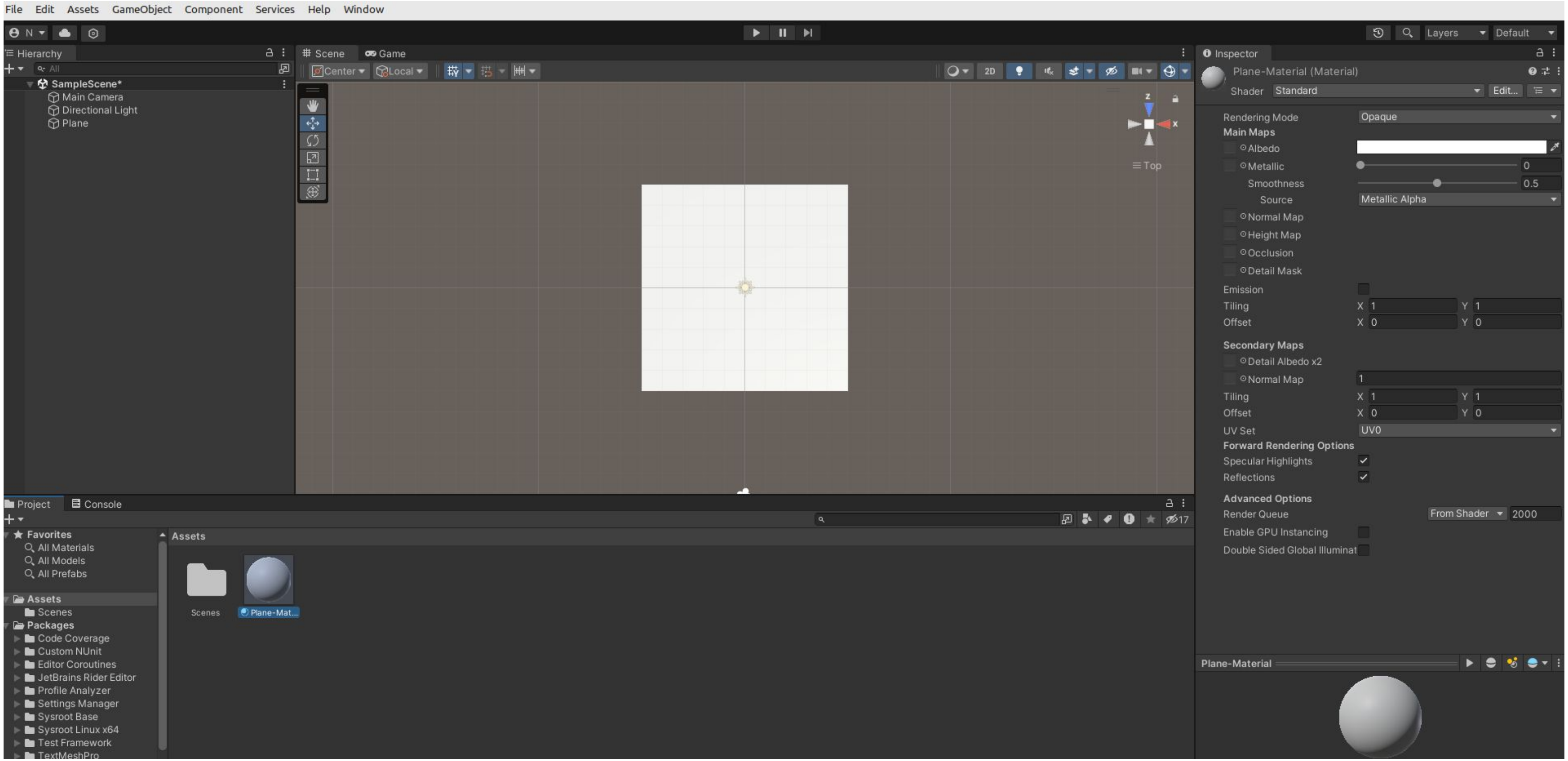
Add Plane: Game Object -> 3D Object -> Plane



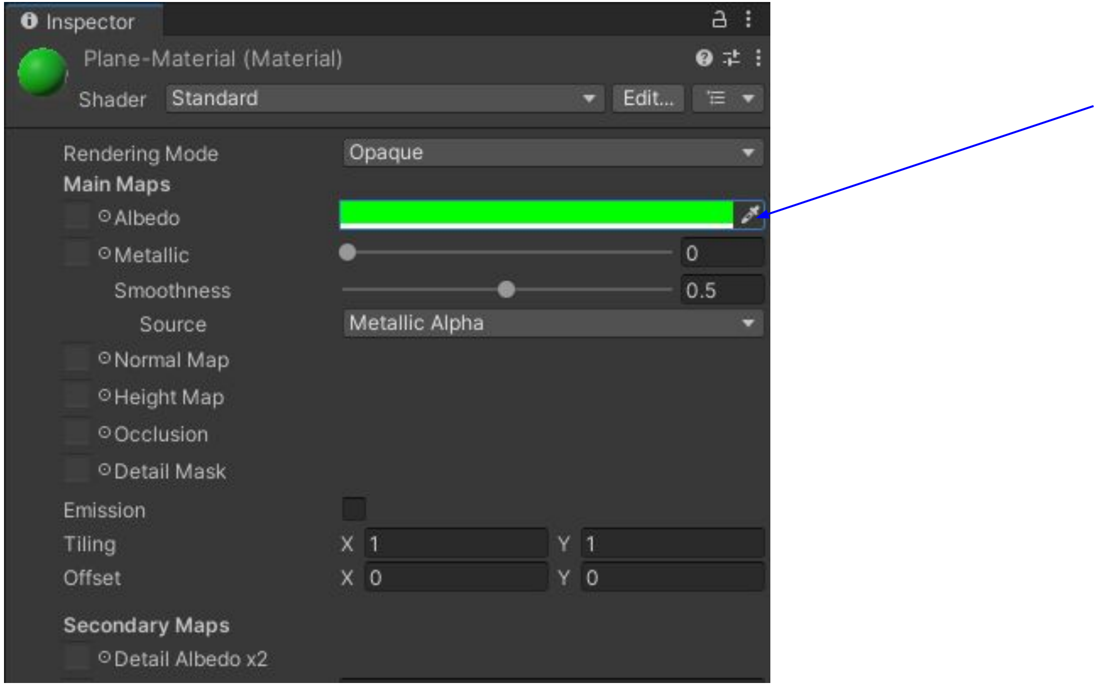
Check object properties in the Inspector



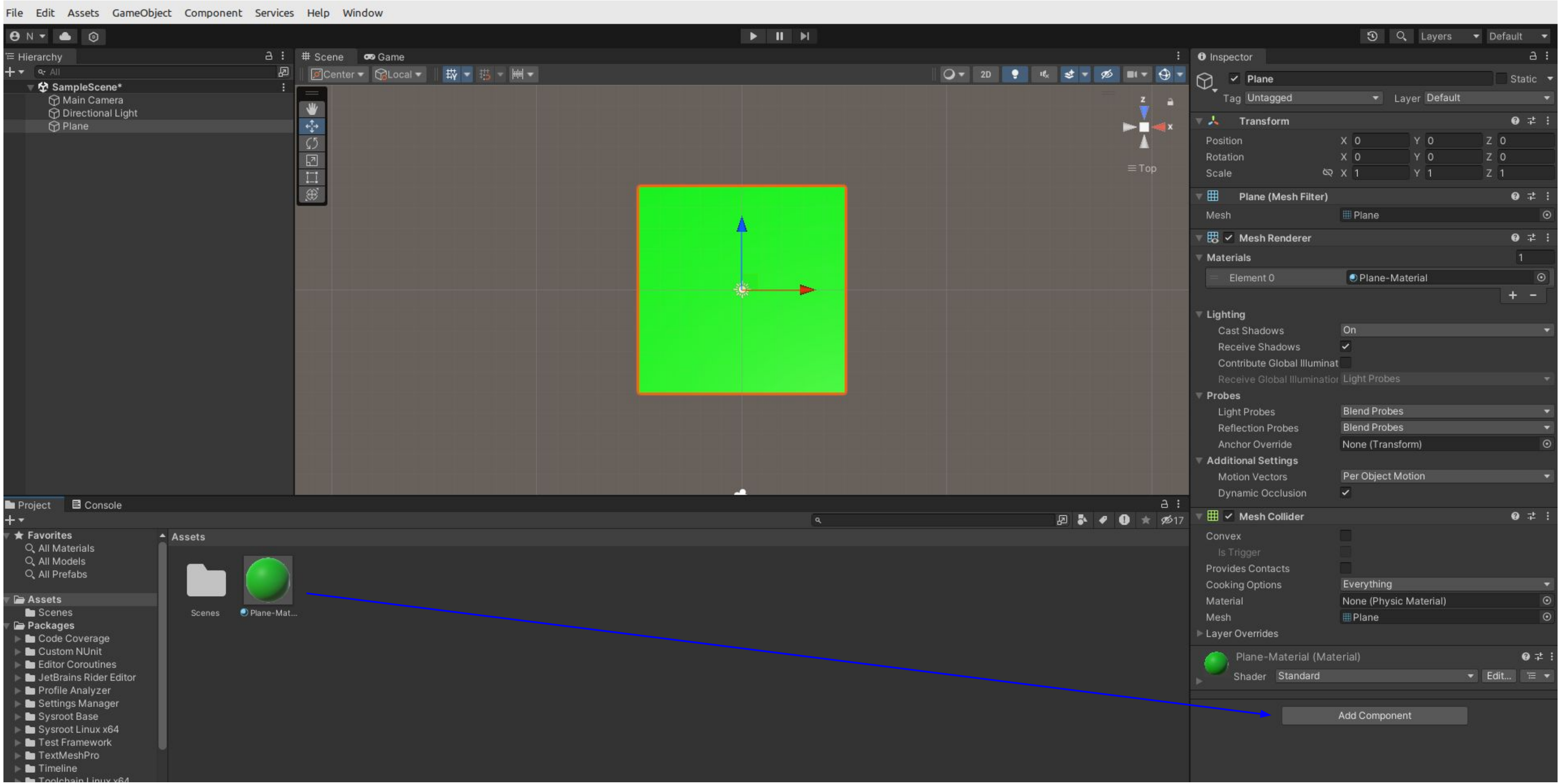
Assets -> Create -> Material



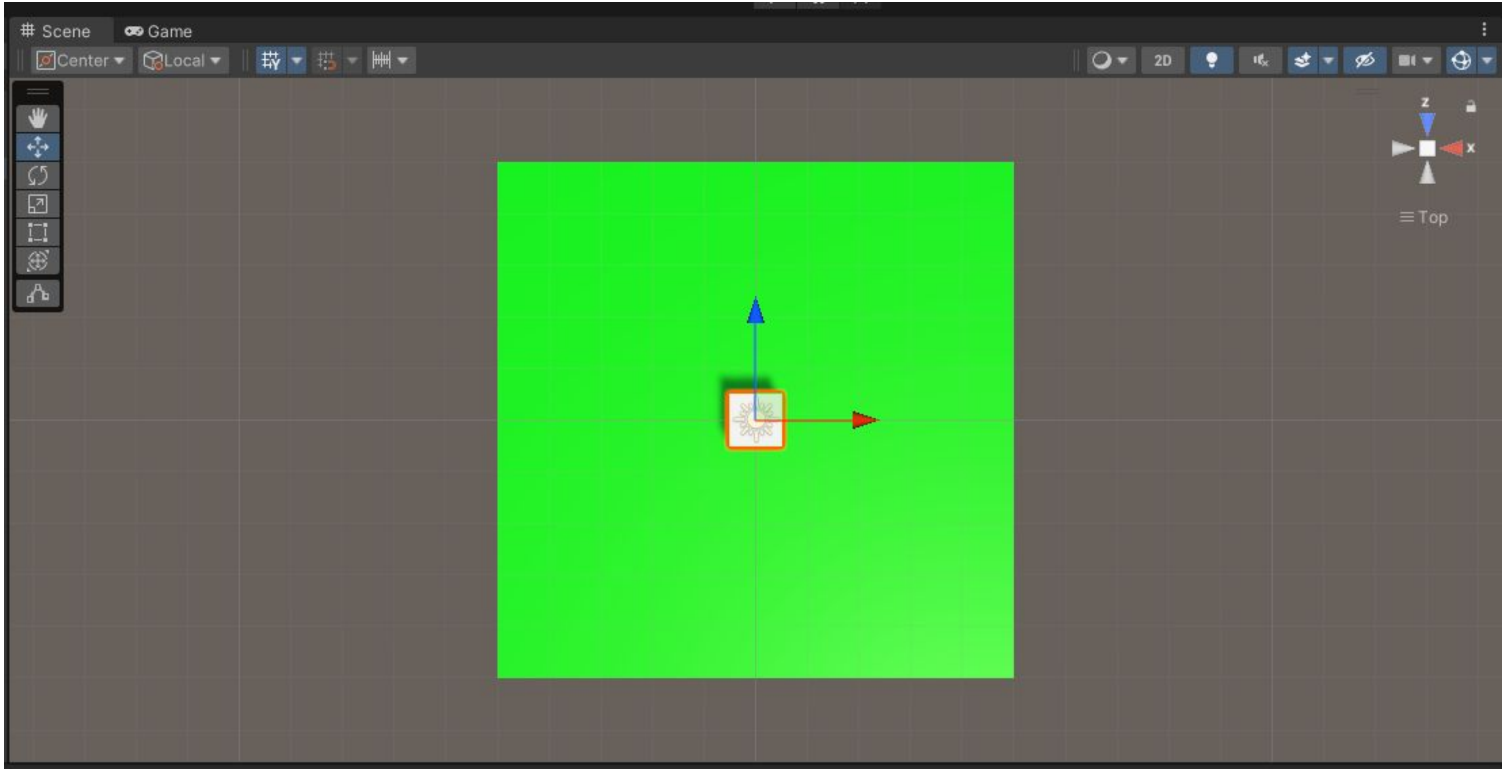
Change Material Colour in Inspector



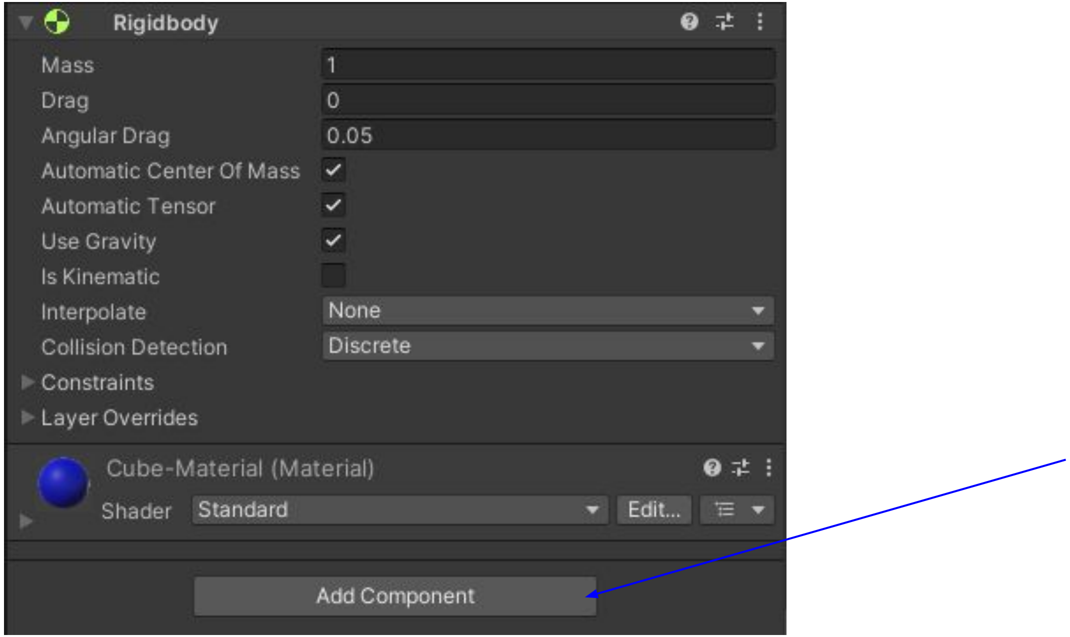
Inspector: Drag and drop material asset



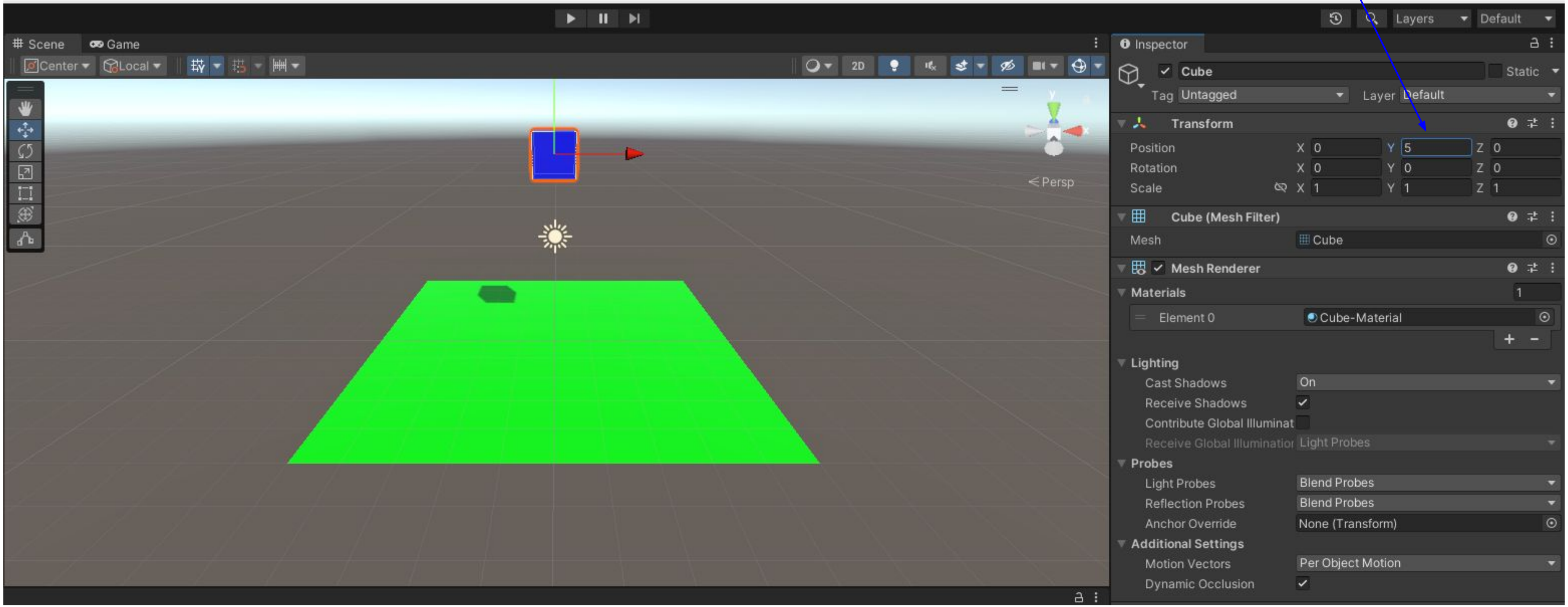
Same as with the Plane: create cube object and add a blue-coloured material



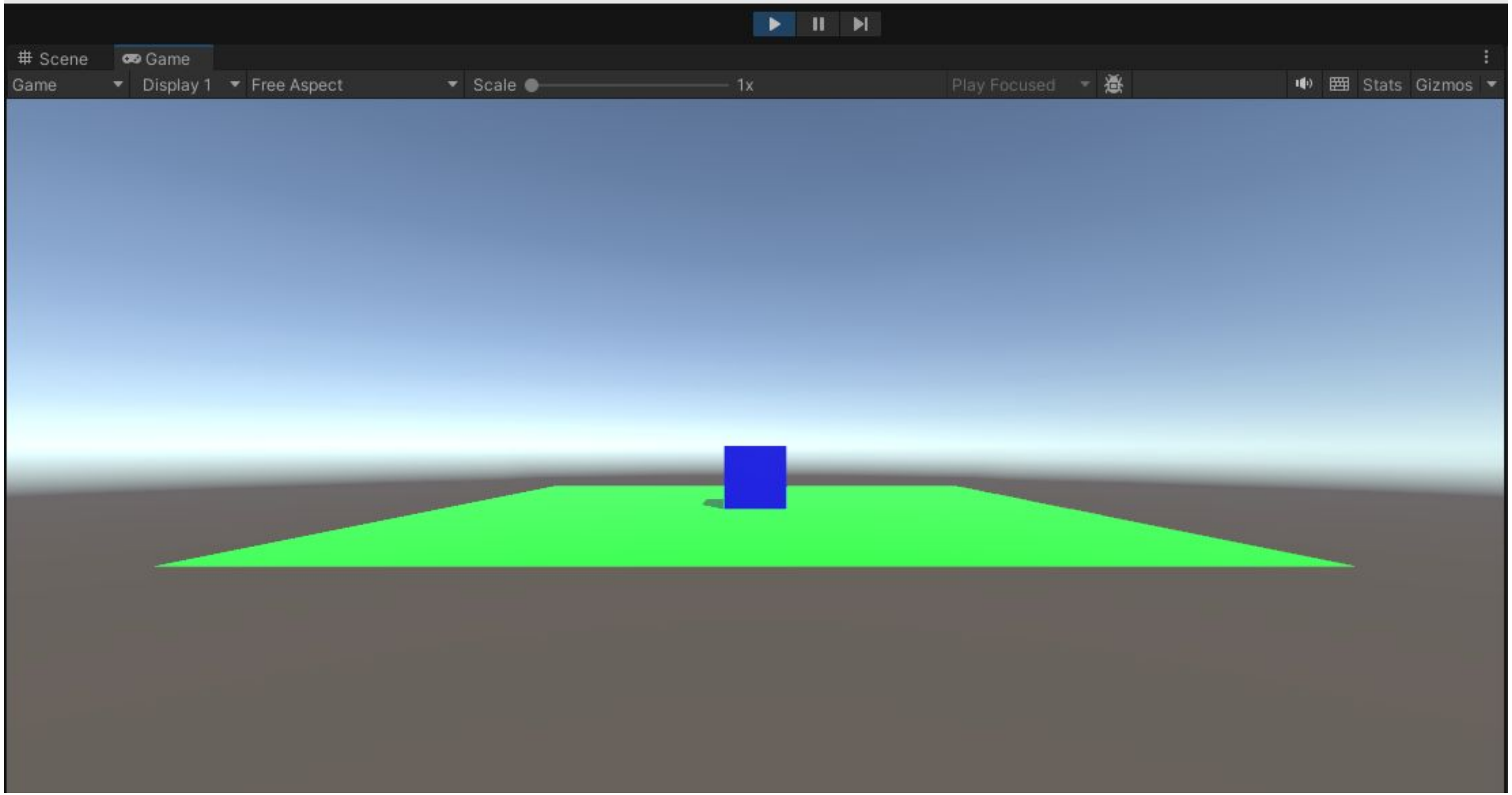
Rigidbody: Add Component -> Physics -> Rigidbody



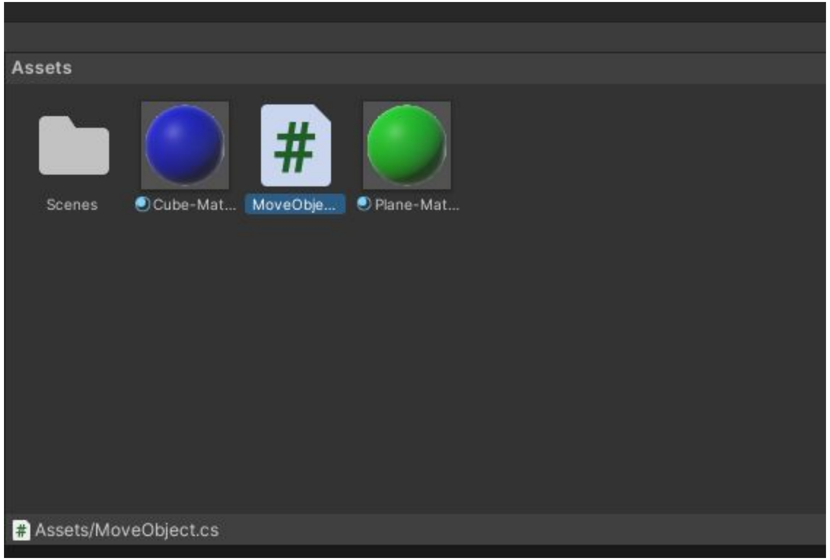
Change the y-coordinate of the cube



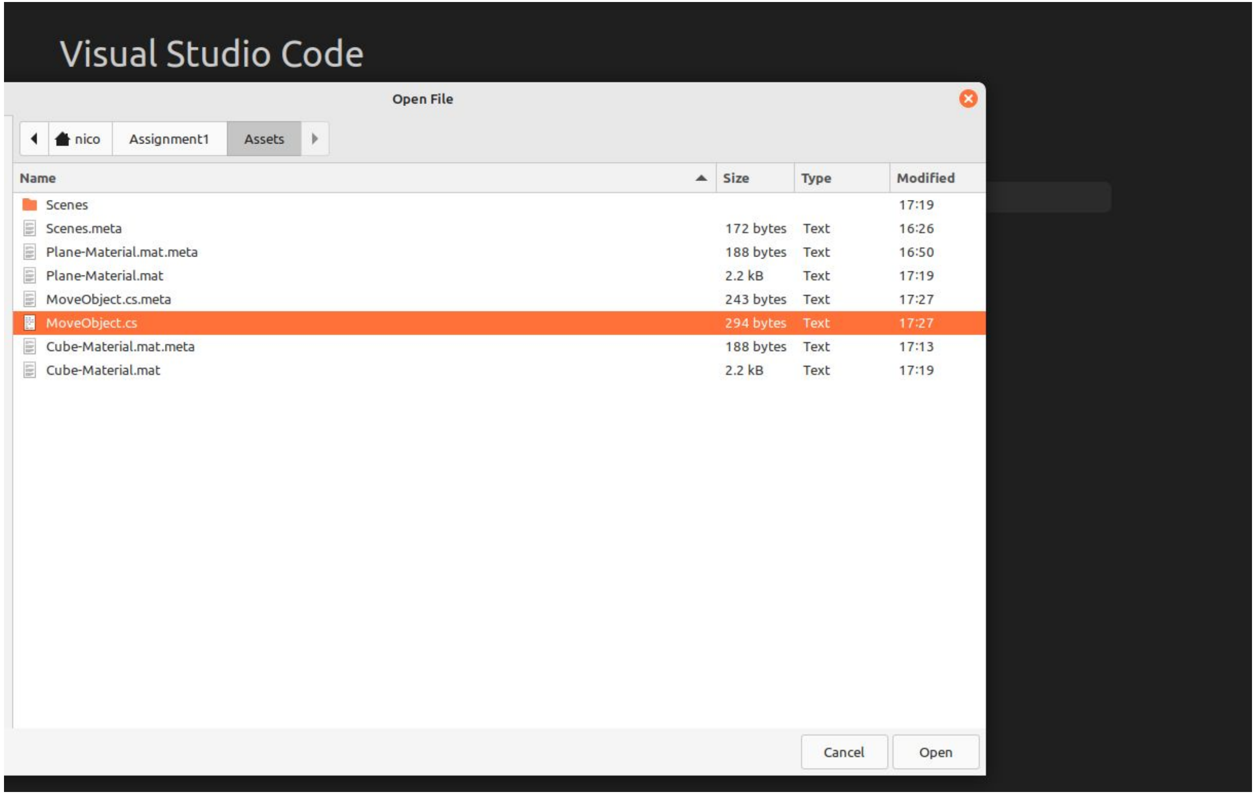
Hit the play button and see what happens



Task 3C: Select Cube -> Inspector -> Add Component -> New Script -> Call it “MoveObject” -> Create and Add



Open the script in the editor of your choice



Use Input.GetAxis to compute the desired behaviour

MoveObject.cs

home > nico > Assignment1 > Assets > MoveObject.cs

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

public class MoveObject : MonoBehaviour
{
    public float speed = 5.0f; // Speed of the cube movement

    void Update()
    {
        float moveHorizontal = Input.GetAxis("Horizontal") * speed * Time.deltaTime; // Calculate horizontal movement
        float moveVertical = Input.GetAxis("Vertical") * speed * Time.deltaTime; // Calculate vertical movement

        // Update the position of the cube
        transform.position = new Vector3(transform.position.x + moveHorizontal, transform.position.y, transform.position.z + moveVertical);
    }
}

```

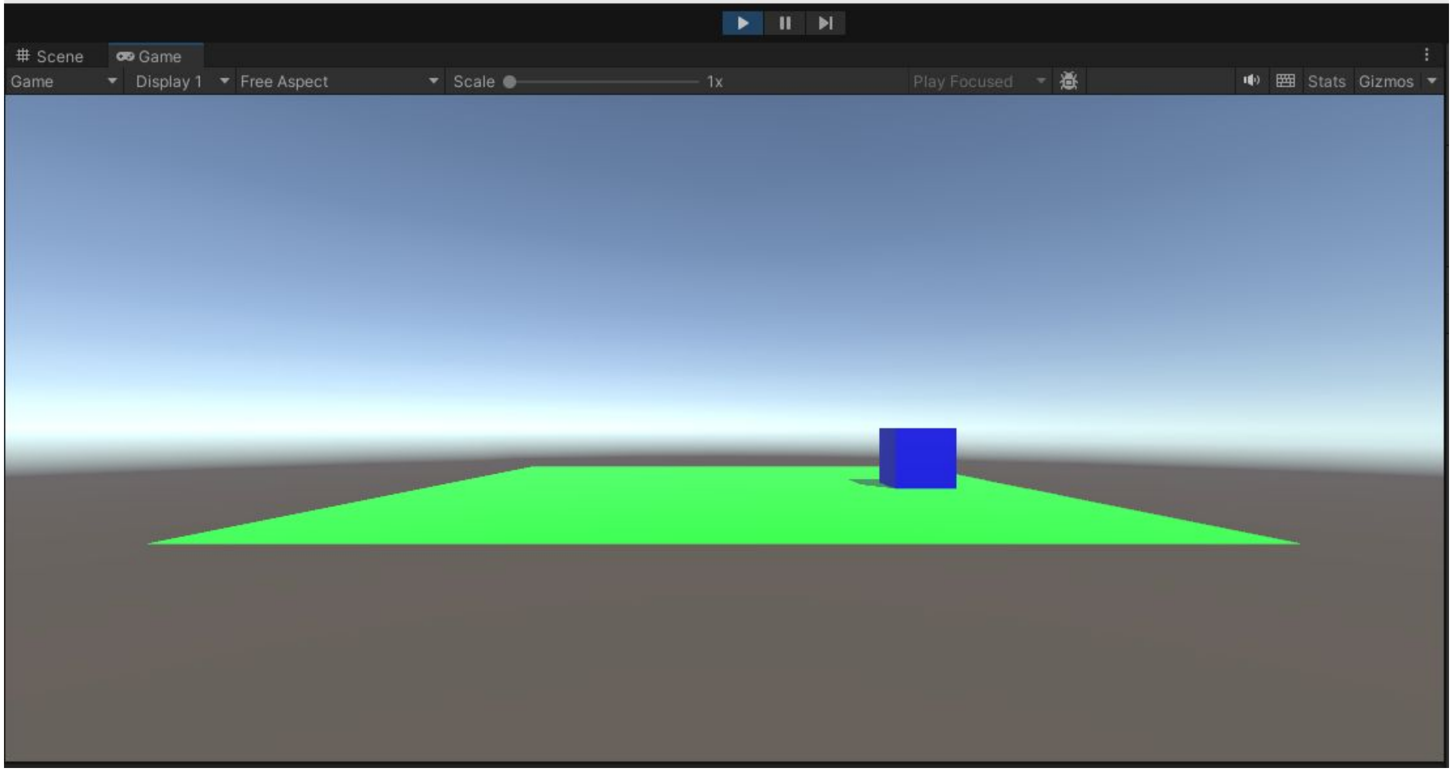
Alternative Version using FixedUpdate for rolling the cube

MoveObject.cs

home > nico > Assignment1 > Assets > MoveObject.cs

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

public class MoveObject : MonoBehaviour
{
    public float speed = 10f; // Speed of the cube movement
    private Rigidbody rb;

    void Start()
    {
        rb = GetComponent<Rigidbody>(); // Get the Rigidbody component
    }

    //use FixedUpdate to modify physics properties
    void FixedUpdate()
    {
        float moveHorizontal = Input.GetAxis("Horizontal"); // Get horizontal input
        float moveVertical = Input.GetAxis("Vertical"); // Get vertical input

        Vector3 movement = new Vector3(moveHorizontal, 0.0f, moveVertical);

        rb.AddForce(movement * speed); // Apply the force to the Rigidbody
    }
}
```

Task 3C: Test it out in game mode



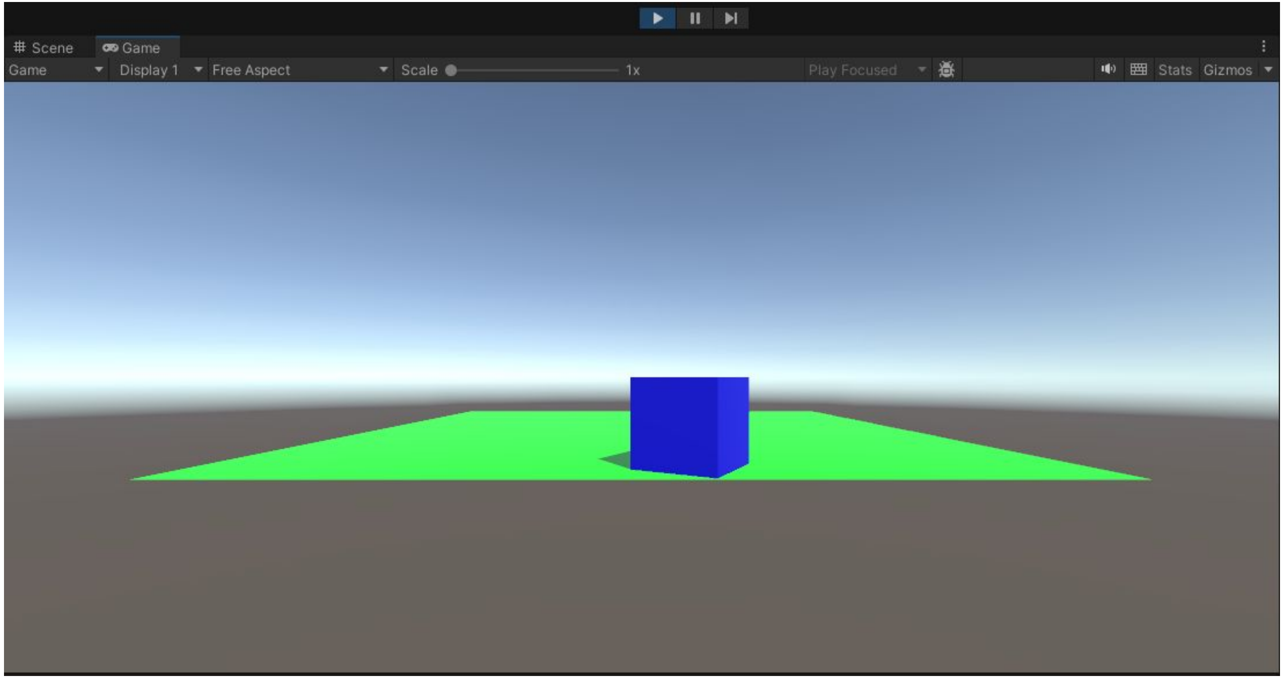
Task 3D: Rotate Object; Workflow same as Task 3C

RotateObject.cs

home > nico > Assignment1 > Assets > RotateObject.cs

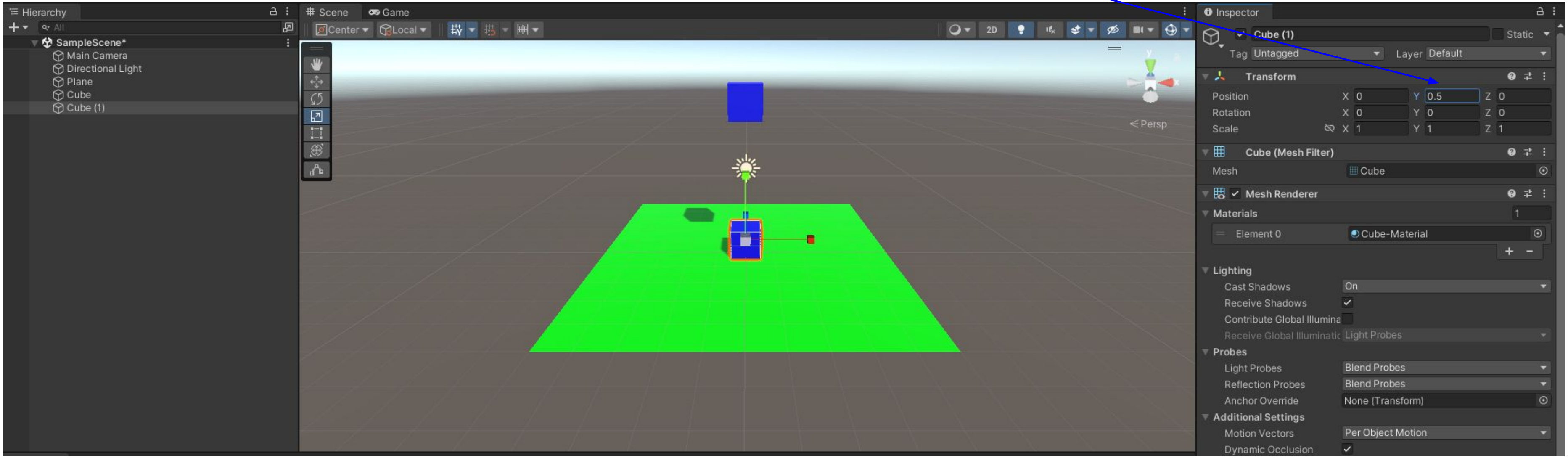
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class RotateObject : MonoBehaviour
{
    public float rotationSpeed = 100f; // Rotation speed in degrees per second

    void Update()
    {
        if (Input.GetKey("r"))
        {
            // Rotate clockwise
            transform.Rotate(0, rotationSpeed * Time.deltaTime, 0, Space.World);
        }
        if (Input.GetKey("l"))
        {
            // Rotate counterclockwise
            transform.Rotate(0, -rotationSpeed * Time.deltaTime, 0, Space.World);
        }
    }
}
```

Task 3D: Test it out in game mode



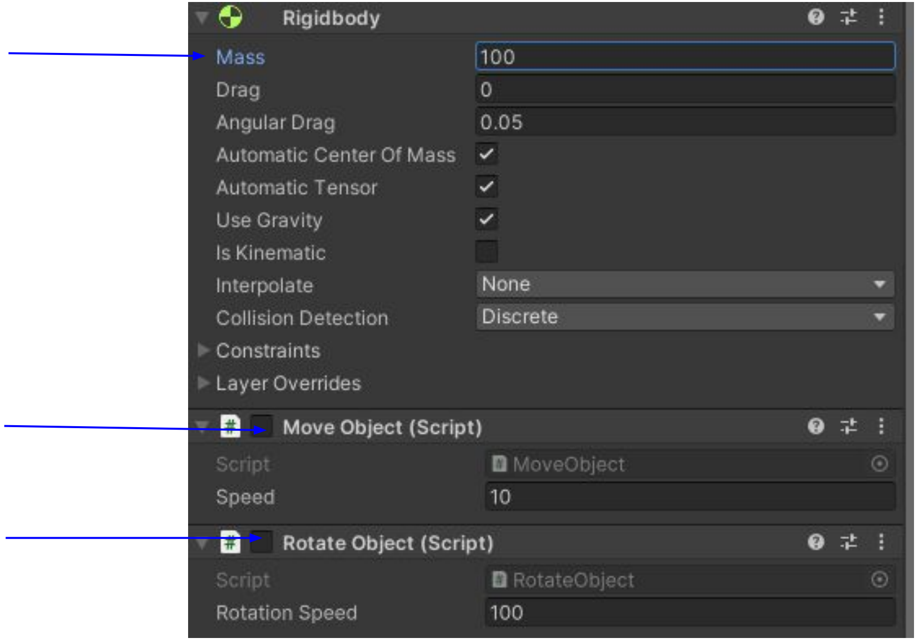
Task 3E: Build your own minigame

- You can duplicate the cube object in the Hierarchy (right click)
- Place the cube where you want it using the Inspector or Scene View

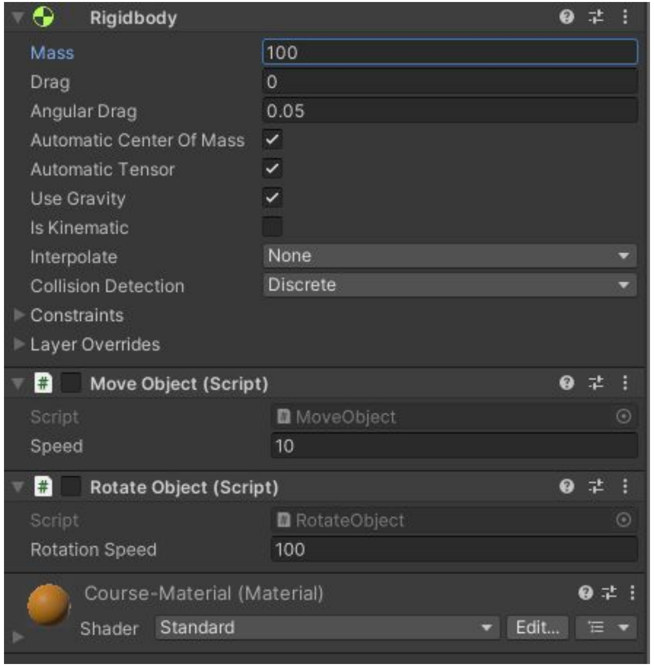


Task 3E: Build your own minigame

- Increase the mass of the cubes to stop them from moving
- Add as many cubes as you want in your course
- Uncheck the scripts (we only need them for the player cube)



Task 3E: Change the Material for better distinction



Task 3E: Set up your course and add as many objects as you want