

## AR -VR:

### Program1 :

Step 1: Create a Script

1. In Unity, go to Assets > Create > C# Script and name it TransformObjects.  
using UnityEngine;

```
public class TransformObjects : MonoBehaviour
{
    public GameObject cube, sphere, plane;

    void Update()
    {
        // **1. Position Transformation (Move in different directions at different speeds)**
        if (Input.GetKey(KeyCode.W)) cube.transform.position += new Vector3(0, 0, 0.05f); // Forward
        if (Input.GetKey(KeyCode.S)) cube.transform.position -= new Vector3(0, 0, 0.05f); // Backward

        if (Input.GetKey(KeyCode.A)) sphere.transform.position -= new Vector3(0.05f, 0, 0); // Left
        if (Input.GetKey(KeyCode.D)) sphere.transform.position += new Vector3(0.05f, 0, 0); // Right

        if (Input.GetKey(KeyCode.UpArrow)) plane.transform.position += new Vector3(0, 0.05f, 0); // Move Up
        if (Input.GetKey(KeyCode.DownArrow)) plane.transform.position -= new Vector3(0, 0.05f, 0); // Move Down

        // **2. Rotation Transformation (Rotate along different axes)**
        if (Input.GetKey(KeyCode.R)) cube.transform.Rotate(1, 0, 0); // Cube rotates along X-axis
        if (Input.GetKey(KeyCode.T)) sphere.transform.Rotate(0, 1, 0); // Sphere rotates along Y-axis
        if (Input.GetKey(KeyCode.Y)) plane.transform.Rotate(0, 0, 1); // Plane rotates along Z-axis

        // **3. Scaling Transformation (Scale in different directions)**
        if (Input.GetKey(KeyCode.Z)) cube.transform.localScale += new Vector3(0.05f, 0, 0); // Scale Cube in X direction
        if (Input.GetKey(KeyCode.X)) sphere.transform.localScale += new Vector3(0, 0.05f, 0); // Scale Sphere in Y direction
        if (Input.GetKey(KeyCode.C)) plane.transform.localScale += new Vector3(0, 0, 0.05f); // Scale Plane in Z direction
    }
}
```

```
}
```

}Steps to Use in Unity:

1. Create Cube, Sphere, and Plane (from GameObject > 3D Object).
2. Create an Empty GameObject and name it "SceneManager".
3. Attach the script to "SceneManager".
4. Assign the Cube, Sphere, and Plane in the Inspector under the script.
5. Press Play to see the transformations in action!

SO THAT YOU KNOW WHAT IS HAPPENING , NO NEED TO WRITE IN DS,BELOW THING

- Movement:
  - W/S → Move Cube forward/backward
  - A/D → Move Sphere left/right
  - Up/Down Arrow → Move Plane up/down
- Rotation:
  - R → Rotate Cube along X-axis
  - T → Rotate Sphere along Y-axis
  - Y → Rotate Plane along Z-axis
- Scaling:
  - Z → Scale Cube in X direction
  - X → Scale Sphere in Y direction
- **C → Scale Plane in Z direction**

## Program 2:

```
using UnityEngine;
```

```
using UnityEngine.UI;
```

```
public class ChangeAppearance : MonoBehaviour
```

```
{
```

```
    public GameObject cube, sphere, plane; // Assign objects in the Inspector
```

```
    public Material newCubeMaterial, newSphereMaterial, newPlaneMaterial; //
```

```
    New materials
```

```
    public Texture newCubeTexture, newSphereTexture, newPlaneTexture; // New
```

```
    textures
```

```
    // Function to change Cube appearance
```

```
    public void ChangeCubeAppearance()
```

```
    {
```

```
        Renderer cubeRenderer = cube.GetComponent<Renderer>();
```

```
        cubeRenderer.material = newCubeMaterial; // Change Material
```

```
        cubeRenderer.material.mainTexture = newCubeTexture; // Change Texture
```

```
        cubeRenderer.material.color = Color.red; // Change Color
```

```
    }
```

```
// Function to change Sphere appearance
public void ChangeSphereAppearance()
{
    Renderer sphereRenderer = sphere.GetComponent<Renderer>();
    sphereRenderer.material = newSphereMaterial;
    sphereRenderer.material.mainTexture = newSphereTexture;
    sphereRenderer.material.color = Color.green;
}

// Function to change Plane appearance
public void ChangePlaneAppearance()
{
    Renderer planeRenderer = plane.GetComponent<Renderer>();
    planeRenderer.material = newPlaneMaterial;
    planeRenderer.material.mainTexture = newPlaneTexture;
    planeRenderer.material.color = Color.blue;
}
}
```

## 1. Create a New Unity Project

1. **Open Unity Hub.**
2. Click **New Project.**
3. Select **3D** template.
4. Name the project (e.g., "ChangeAppearanceDemo").
5. Click **Create.**

## 2. Set Up the Scene

After Unity loads the project:

1. **Save the Scene:**
  - Go to **File > Save As.**
  - Name it "MainScene" and save it in the Assets/Scenes/ folder.
2. **Create the 3D Objects:**
  - **Cube:**
    - ◆ Go to **GameObject > 3D Object > Cube.**
    - ◆ Rename it "Cube" (Right-click > Rename).
  - **Sphere:**
    - ◆ Go to **GameObject > 3D Object > Sphere.**
    - ◆ Rename it "Sphere".
  - **Plane:**
    - ◆ Go to **GameObject > 3D Object > Plane.**
    - ◆ Rename it "Plane".
3. **Position the Objects:**
  - Select the **Cube:**
    - ◆ Set **Position** to (0, 1, 0).
  - Select the **Sphere:**

- ♦ Set **Position** to (2, 1, 0).
- Select the **Plane**:
  - ♦ Set **Position** to (0, 0, 0) (Default is fine).

### 3. Create Materials and Textures

#### 1. Create New Materials:

- In the **Project** window, go to Assets/.
- Right-click and choose **Create > Material**.
- Rename them:
  - ♦ CubeMaterial
  - ♦ SphereMaterial
  - ♦ PlaneMaterial

#### 2. Import Textures:

- Drag and drop texture image files (e.g., CubeTexture.png, SphereTexture.png, PlaneTexture.png) into the Assets/ folder.

### 4. Create UI Elements

#### 1. Create a Canvas:

- In the **Hierarchy**, go to **GameObject > UI > Canvas**.
- Select the **Canvas** and in the **Inspector**:
  - ♦ Set **Render Mode** to **Screen Space - Overlay**.

#### 2. Create Buttons:

- In the **Hierarchy**, go to **GameObject > UI > Button** (Repeat for three buttons).
- Rename them:
  - ♦ CubeButton
  - ♦ SphereButton
  - ♦ PlaneButton
- Select each **Button**, expand it in the **Hierarchy**, select **Text**, and change the text to:
  - ♦ "Change Cube"
  - ♦ "Change Sphere"
  - ♦ "Change Plane"

#### 3. Arrange Buttons in the UI:

- Select **CubeButton**, set **Position Y** to 100.
- Select **SphereButton**, set **Position Y** to 0.
- Select **PlaneButton**, set **Position Y** to -100.

### 5. Create and Attach the Script

#### 1. Create a New Script:

- In the **Project** window, go to Assets/.
- Right-click > **Create > C# Script**.
- Name it "**ChangeAppearance**".
- Double-click to open in **Visual Studio**.

- Copy and paste the following code:

```
csharp
CopyEdit
using UnityEngine;
using UnityEngine.UI;

public class ChangeAppearance : MonoBehaviour
{
    public GameObject cube, sphere, plane; // Assign objects in the Inspector
    public Material newCubeMaterial, newSphereMaterial, newPlaneMaterial; //
New materials
    public Texture newCubeTexture, newSphereTexture, newPlaneTexture; // New
textures

    // Function to change Cube appearance
    public void ChangeCubeAppearance()
    {
        Renderer cubeRenderer = cube.GetComponent<Renderer>();
        cubeRenderer.material = newCubeMaterial; // Change Material
        cubeRenderer.material.mainTexture = newCubeTexture; // Change Texture
        cubeRenderer.material.color = Color.red; // Change Color
    }

    // Function to change Sphere appearance
    public void ChangeSphereAppearance()
    {
        Renderer sphereRenderer = sphere.GetComponent<Renderer>();
        sphereRenderer.material = newSphereMaterial;
        sphereRenderer.material.mainTexture = newSphereTexture;
        sphereRenderer.material.color = Color.green;
    }

    // Function to change Plane appearance
    public void ChangePlaneAppearance()
    {
        Renderer planeRenderer = plane.GetComponent<Renderer>();
        planeRenderer.material = newPlaneMaterial;
        planeRenderer.material.mainTexture = newPlaneTexture;
        planeRenderer.material.color = Color.blue;
    }
}
```

## 2. Attach the Script to an Empty GameObject:

- In the **Hierarchy**, go to **GameObject > Create Empty**.
- Rename it "AppearanceManager".
- Drag the **ChangeAppearance** script onto "AppearanceManager".

## 6. Assign References in the Inspector

1. Select "AppearanceManager" in the **Hierarchy**.
2. In the **Inspector**, find the **ChangeAppearance** script.
3. Assign GameObjects:
  - Drag **Cube** into cube field.
  - Drag **Sphere** into sphere field.
  - Drag **Plane** into plane field.
4. Assign Materials:
  - Drag CubeMaterial into newCubeMaterial.
  - Drag SphereMaterial into newSphereMaterial.
  - Drag PlaneMaterial into newPlaneMaterial.
5. Assign Textures:
  - Drag CubeTexture into newCubeTexture.
  - Drag SphereTexture into newSphereTexture.
  - Drag PlaneTexture into newPlaneTexture.

## 7. Link Buttons to Script Functions

1. Select **CubeButton**:
  - In **Inspector**, under **Button** component, find **On Click ()**.
  - Click the **+** button.
  - Drag **AppearanceManager** into the empty field.
  - From the dropdown, choose **ChangeAppearance > ChangeCubeAppearance**.
2. Select **SphereButton**:
  - Drag **AppearanceManager** into **On Click ()** field.
  - Choose **ChangeAppearance > ChangeSphereAppearance**.
3. Select **PlaneButton**:
  - Drag **AppearanceManager** into **On Click ()** field.
  - Choose **ChangeAppearance > ChangePlaneAppearance**.

## 8. Test the Project

1. Click **Play** in the Unity Editor.
2. Click each button to see the **Cube, Sphere, and Plane** change appearance.

Program 3:

We need to use unity chan and Asobi chan

Program 4:

## Step 1: Create a New Unity Project

Open Unity Hub.

Click New Project.

Choose the 3D template (since this is an FPS controller for a 3D environment).

Name the project (e.g., "FPSControllerDemo").

Click Create.

## Step 2: Setup the Player Object

In the Hierarchy window, click Create → 3D Object → Capsule.

Rename it to "Player".

Set its Position to (0, 1, 0) (so it stands on the ground).

Click Add Component in the Inspector.

Search for and add CharacterController.

Adjust the CharacterController properties:

Center: (0, 1, 0) (Align with the capsule)

Radius: 0.3

Height: 2

Click Add Component again.

Click New Script → Name it FPSPlayerController.

Click Create and Add.

Double-click the script to open it in Visual Studio/VS Code.

Replace the auto-generated code with the provided FPSPlayerController.cs script.

Save the file (Ctrl + S or Cmd + S on macOS).

## Step 3: Setup the Camera

In the Hierarchy, select Main Camera.

Drag and drop the Main Camera onto the Player object to make it a child.

Set the Camera Position (Transform in the Inspector):

Position: (0, 1.6, 0) (Align the camera to eye level).

Rotation: (0, 0, 0).

The camera will now move and rotate with the player.

## Step 4: Setup the Ground (Environment)

In the Hierarchy, click Create → 3D Object → Plane.

Rename it to "Ground".

Set its Position to (0, 0, 0).

Scale the Ground (Transform → Scale):

X: 10

Z: 10

(This makes it large enough to walk on).

(Optional) Add a Material to the ground for better visuals:

Click Assets → Create → Material.

Name it "GroundMaterial".

Select the Ground.

Drag & drop GroundMaterial onto it.

#### Step 5: Configure Input Settings

Go to Edit → Project Settings.

Select Input Manager.

Ensure that Horizontal and Vertical axes exist (Default: WASD or Arrow Keys).

Ensure Mouse X exists for rotation control.

#### Step 6: Adjust Player Movement

Select Player in the Hierarchy.

In the Inspector, adjust the Speed variable (increase if needed).

#### Step 7: Run the Game

Click Play in the Unity Editor.

Use W, A, S, D to move.

Move the mouse left/right to rotate the camera.

Ensure smooth movement and rotation.

#### Step 8: Debugging (If Any Issues Occur)

Issue: Player Is Not Moving



Ensure the CharacterController component is added to the Player.



Check if FPSPlayerController.cs is correctly attached to Player.



Check the Inspector for any errors.

Issue: Rotation Not Working



Ensure Mouse X is mapped in Input Manager.



Try increasing Rotation Speed in the script.

#### Final Notes

This script allows basic FPS movement with mouse-controlled rotation.

For jumping and gravity, you need additional modifications.

You can replace the Capsule with a 3D model for a realistic FPS player.

Let me know if you need additional features! 🚀

You said:

#### Step 5: Configure Input Settings

Go to Edit → Project Settings.

Select Input Manager.

Ensure that Horizontal and Vertical axes exist (Default: WASD or Arrow Keys).

Ensure Mouse X exists for rotation control.



Give in detail how to do step 5

ChatGPT said:

### Step 5: Configure Input Settings in Unity

The FPSPlayerController script relies on Unity's Input Manager to detect player movement (WASD or Arrow Keys) and mouse rotation (Mouse X). To ensure everything is correctly set up, follow these detailed steps:

#### Step 5.1: Open Input Manager

Open Unity Editor.

Go to the top menu bar and click on Edit.

Select Project Settings from the dropdown.

In the Project Settings window, look for Input Manager in the left sidebar and click on it.

#### Step 5.2: Verify and Configure Movement Axes

Unity's Input Manager has predefined axes for movement and rotation. We need to ensure they exist and are correctly set.

##### 1. Check "Horizontal" (Left/Right movement)

Expand the Horizontal section.

Check the following properties:

Name: Horizontal

Type: Key / Mouse Button

Negative Button: a

Positive Button: d

Alt Negative Button: left

Alt Positive Button: right

Gravity: 3

Dead: 0.001

Sensitivity: 3

Snap: ☒ Checked

Invert: ☐ Unchecked

☒ Purpose: Allows movement left (A or ←) and right (D or →).

##### 2. Check "Vertical" (Forward/Backward movement)

Expand the Vertical section.

Verify the settings:

Name: Vertical

Type: Key / Mouse Button

Negative Button: s

Positive Button: w

Alt Negative Button: down

Alt Positive Button: up

Gravity: 3

Dead: 0.001

Sensitivity: 3

Snap: ☒ Checked

Invert: ☒ Unchecked

☒ Purpose: Allows movement forward (W or ↑) and backward (S or ↓).

### Step 5.3: Verify and Configure Mouse Rotation

The script uses Mouse X for rotation. Check the settings:

#### 1. Check "Mouse X" (Camera Rotation)

Expand the Mouse X section.

Ensure it has these settings:

Name: Mouse X

Type: Mouse Movement

Axis: X Axis

Gravity: 0

Dead: 0.001

Sensitivity: 0.1 (Increase if rotation is slow)

Snap: ☒ Unchecked

Invert: ☒ Unchecked

☒ Purpose: Moves the camera left/right when moving the mouse horizontally.

### Step 5.4: Save and Test

Close the Project Settings window.

Click Play in the Unity Editor.

Move the character using WASD or Arrow Keys.

Rotate the camera by moving the mouse left and right.

If movement or rotation doesn't work, check the Console (Window → General → Console) for errors.

### Step 5.5: Troubleshooting (If Something Doesn't Work)

using UnityEngine;

```
public class FPSPlayerController : MonoBehaviour{
    public float speed = 5f; // Movement speed
    public float rotationSpeed = 100f; // Rotation speed

    private CharacterController controller;

    void Start()
    {
        controller = GetComponent<CharacterController>(); // Get the Character
        Controller
    }

    void Update()
```

```

{
    // Get movement input (Forward/Backward & Left/Right)
    float moveZ = Input.GetAxis("Vertical"); // W & S for forward/backward
    float moveX = Input.GetAxis("Horizontal"); // A & D for left/right

    // Move the player based on input
    Vector3 move = transform.forward * moveZ + transform.right * moveX;
    controller.Move(move * speed * Time.deltaTime);

    // Rotate Left/Right using A & D
    float rotation = Input.GetAxis("Mouse X") * rotationSpeed *
Time.deltaTime;
    transform.Rotate(0, rotation, 0);
}
}

```

## Prog 5:

### Steps to Run Your Prog5 Script in Unity

Your script rotates a GameObject and shoots a 2D Raycast in the right direction. Follow these steps to test it in Unity:

#### Step 1: Open Unity & Create a 2D Project

Open Unity Hub and click New Project.

Select 2D Core as the template and click Create Project.

#### Step 2: Create a GameObject & Attach the Script

Create an Empty GameObject

In the Hierarchy window, click + (Create) → Create Empty

Rename it to RotatingObject

Attach the Script to the GameObject

Select RotatingObject

In the Inspector, click Add Component

Search for Prog5 and click it to attach the script

#### Step 3: Create a Visual Target for Raycast

To see raycasting in action, add an object for the ray to hit.

Create a 2D Sprite (Obstacle)

In the Hierarchy, click + (Create) → 2D Object → Sprite

Rename it to Obstacle

Position it to the right of RotatingObject (so the ray hits it)

Add a Collider:

In the Inspector, click Add Component

Search for BoxCollider2D and add it

Step 4: Adjust the Script's Settings

Click on RotatingObject.

In the Inspector, under Prog5:

Set Speed = 20 (controls rotation speed)

Set Distance = 10 (how far the ray shoots)

Step 5: Run the Scene

Click Play at the top of the Unity Editor.

Look at the Console Window:

If the ray hits the obstacle, it logs:

makefile

Copy

Edit

Hit: Obstacle

If the ray does not hit anything, it logs:

yaml

Copy

Edit

No hit

Step 6: View Debug Rays

Stop the game.

Go to Scene View (not Game View).

Play the game again and look for red/green lines:

Red Line = Ray hit an object

Green Line = Ray missed

```
using UnityEngine;
```

```
using UnityEngine.InputSystem;
```

```
public class Prog5 : MonoBehaviour
```

```
{
```

```
    [SerializeField] float speed = 20f;
```

```
    [SerializeField] float distance = 10f;
```

```
    RaycastHit2D hit;
```

```
    void Start()
```

```
    {
```

```
        Debug.Log("Press 'Space' to shoot a raycast");
```

```
    }
```

```
    void FixedUpdate()
```

```
    {
```

```
        transform.Rotate(Vector3.forward * speed * Time.deltaTime);
```

```
        hit = Physics2D.Raycast(transform.position, transform.right, distance);
```

```
        if (hit.collider != null)
```

```

    {
        Debug.DrawRay(transform.position, hit.point, Color.red);
        Debug.Log("Hit: " + hit.collider.name);
    }
    else
    {
        Debug.DrawRay(transform.position, transform.right * distance,
Color.green);
        Debug.Log("No hit");
    }
}
}
}

```

PROG 6 and 7:

Vuforia is used.

Wherein the database is created and managed .

steps:

Create a 3D project

Go to vuforia developer engine(in google)

Download : Add Vuforia Engine to a unity project(package in downloads)

Import this package in to the assets

Now in Vuforia, go to target manager :

Add database->create a database

Into the database now add an image

Now licenseManager->GetDevelopmentKey->copy the license key

Now in unity hierarchy window ->double click->vuforia engine->AR camera

Delete the main camera

Vuforia engine->AR camera

Arcamera->open vuforia engine configuration ->paste the license key

Download the database and import it to the project assets window

Imagetarget->image target behaviour ->type database->database - jo banana hai->image target ->the image uploaded

Place a 3d object on the image and run

Prog 8:

Step 1: Create a New Unity Project & Import 3D Models

Open Unity Hub → Click on New Project → Select 3D Template.

Name your project and click Create.

Open Asset Store (Window → Asset Store) and download a 3D model (e.g., a character, house, or tree).

Drag and drop the imported 3D model into the Hierarchy window.

Step 2: Place the Model in the Scene

Select the 3D model in the Hierarchy window.

Adjust its Position, Rotation, and Scale using the Inspector Panel to make it visible.

Example: If the object is underground, change its Y Position to bring it into view.

Step 3: Add an Audio Source Component

Download an MP3 file and import it into Assets (drag and drop).

Select the 3D model in the Hierarchy window.

Click Add Component → Audio Source.

Drag the imported MP3 file into the AudioClip field in the Inspector.

Uncheck Play on Awake so it only plays when triggered.

Step 4: Create an Animation Controller

Go to Assets → Create → Animator Controller.

Name it something like "CharacterAnimator".

Select the 3D model, then in the Inspector, click Add Component → Animator.

Assign the CharacterAnimator to the Animator component.

Open Window → Animation → Animator to see the animation state machine.

Step 5: Add Animation and Make Transitions

Download or create an animation (e.g., a running animation).

Drag the animation into the Animator window.

Create a transition between the default state (Idle) and the Run animation.

Step 6: Create a Trigger for Animation Activation

Open the Animator window.

Click the Parameters tab, then click "+" → Trigger and name it "run".

Select the transition from Idle → Run.

Set the condition for the transition to Trigger == run.

Step 7: Create a UI Button and Attach a Script

Go to Hierarchy → UI → Button to add a UI button.

Rename it to "RunButton".

Select the Button, go to the Inspector panel, and click Add Component → Script.

Name the script "Prog8.cs".

Attach the script to the Button's OnClick() event.

```
using UnityEngine;
```

```
public class Prog8 : MonoBehaviour
```

```
{
```

```
    Animator animator; // Reference to Animator component
```

```
    AudioSource audioSource; // Reference to AudioSource component
```

```
    void Start()
```

```
    {
```

```
        animator = GetComponent<Animator>(); // Get Animator component from  
the GameObject
```

```
        audioSource = GetComponent<AudioSource>(); // Get AudioSource  
component from the GameObject
```

```
    }
```

```
    public void Run()
```

```
    {
```

```
        animator.SetTrigger("run"); // Triggers the "run" animation
```

```
        audioSource.Play(); // Plays the attached audio clip
```

```
    }
```

```
}
```