Practical No. 1:

Aim: Setup DirectX 11, Window Framework and Initialize Direct3D Device

DirectX is an application program interface (API) for creating and managing graphic images and multimedia effects in applications such as games or active Web pages that will run in Microsoft's Windows operating systems.

In this practical we are just learning the window framework and initializing a Direct3D device.

Step 1:

Create new project, and select "Windows Forms Application", select .NET Framework as 2.0 in Visuals C#.
Right Click on properties Click on open click on build Select Platform Target and Select x86.

Step 2: Click on View Code of Form 1.

Step 3:

Go to Solution Explorer, right click on project name, and select Add Reference. Click on Browse and select the given .dll files which are "Microsoft.DirectX", "Microsoft.DirectX.Direct3D", and "Microsoft.DirectX.DirectX3DX".

Step 4:

Go to the Properties Section of Form, select Paint in the Event List and enter as Form1_Paint.

Step 5:

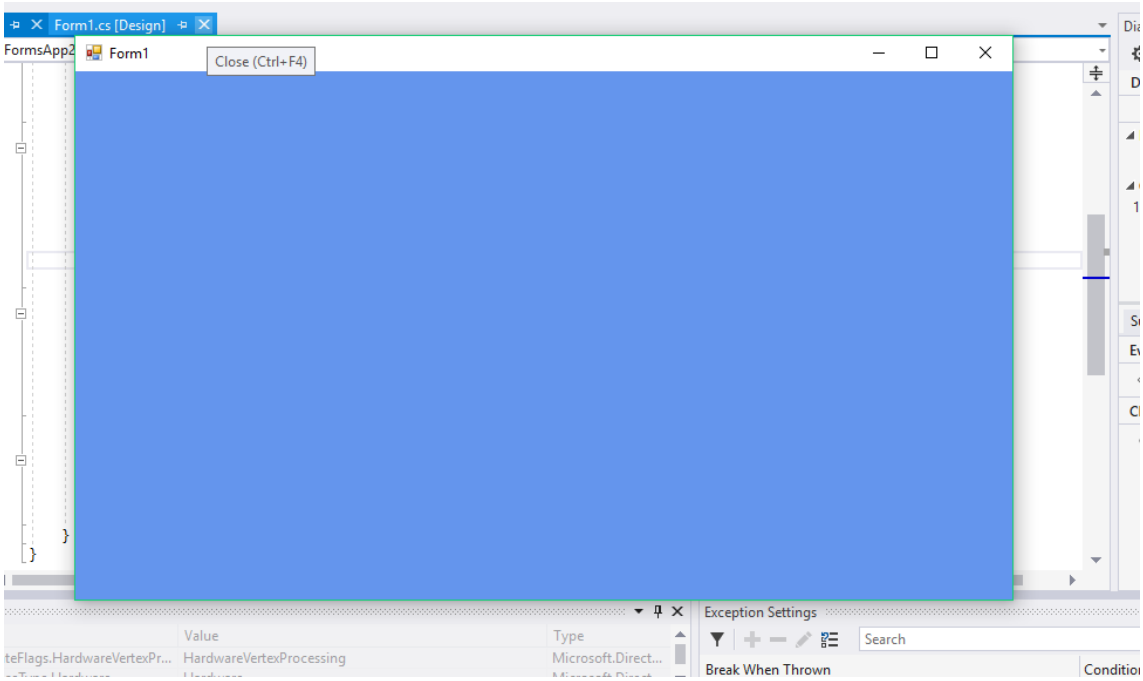
Edit the Form's C# code file. Namespace must be the same as your project name.

```
using System;
usingSystem.Collections.Generic;
usingSystem.ComponentModel;
usingSystem.Data;
usingSystem.Drawing;
usingSystem.Text;
usingSystem.Windows.Forms;
usingMicrosoft.DirectX;
using Microsoft.DirectX.Direct3D;
namespace GP_P1
{
public partial class Form1 : Form
   {
      Microsoft.DirectX.Direct3D.Device device;
public Form1()
      {
InitializeComponent();
InitDevice();
```

```
        }
public void InitDevice()
        {
PresentParameterspp = new PresentParameters();
pp.Windowed = true;
pp.SwapEffect = SwapEffect.Discard;
device = new Device(0, DeviceType.Hardware, this,
                                CreateFlags.HardwareVertexProcessing, pp);
        }
private void Render()
        {
device.Clear(ClearFlags.Target, Color.Orange, 0, 1);
device.Present();
        }
private void Form1_Paint(object sender, PaintEventArgs e)
        {
Render();
        }
    }
}
```

Step 6: Click on Start. And here is the output. We have initialized 3D Device.

Output:



Practical No. 2:

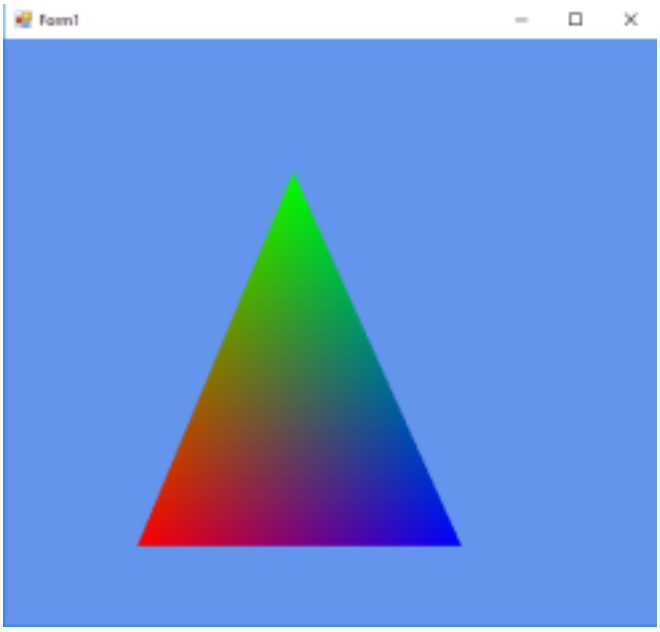
Aim: Buffers, Shaders and HLSL (Draw a triangle using Direct3D 11)

Solution:

```
using System;
usingSystem.Collections.Generic;
usingSystem.ComponentModel;
usingSystem.Data;
usingSystem.Drawing;
usingSystem.Text;
usingSystem.Windows.Forms;
usingMicrosoft.DirectX;
using Microsoft.DirectX.Direct3D;
namespace GP_P2
 {
public partial class Form1 : Form
    {
       Microsoft.DirectX.Direct3D.Device device;
public Form1()
        {
InitializeComponent();
InitDevice();
        }
private void InitDevice()
        {
PresentParameterspp = new PresentParameters();
pp.Windowed = true;
pp.SwapEffect = SwapEffect.Discard;
device = new Device(0, DeviceType.Hardware, this,
CreateFlags.HardwareVertexProcessing, pp);
        }
private void Render()
        {
CustomVertex.TransformedColored[] vertexes = new
CustomVertex.TransformedColored[3];

vertexes[0].Position = new Vector4(240, 110, 0, 1.0f);//first point
vertexes[0].Color = System.Drawing.Color.FromArgb(0, 255, 0).ToArgb();

vertexes[1].Position = new Vector4(380, 420, 0, 1.0f);//second point
vertexes[1].Color = System.Drawing.Color.FromArgb(0, 0, 255).ToArgb();
vertexes[2].Position = new Vector4(110, 420, 0, 1.0f);//third point
vertexes[2].Color = System.Drawing.Color.FromArgb(255, 0, 0).ToArgb();
```

```
device.Clear(ClearFlags.Target, Color.CornflowerBlue, 1.0f, 0);
device.BeginScene();
device.VertexFormat = CustomVertex.TransformedColored.Format;
device.DrawUserPrimitives(PrimitiveType.TriangleList, 1, vertexes);
device.EndScene();
device.Present();
        }
private void Form1_Load(object sender, EventArgs e) { }

        private void Form1_Paint(object sender, PaintEventArgs
                e) {
Render();
        }
    }
}
```

Output:



Practical No. 3:

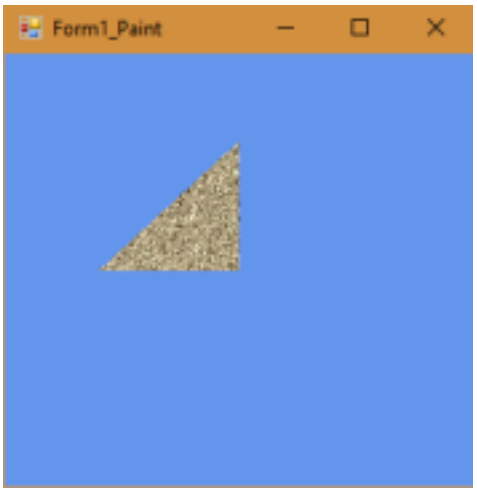
Aim: Texture the triangle using Direct3D 11

Solution:

```
using System;
usingSystem.Collections.Generic;
usingSystem.ComponentModel;
usingSystem.Data;
usingSystem.Drawing;
usingSystem.Text;
usingSystem.Windows.Forms;
usingMicrosoft.DirectX;
using Microsoft.DirectX.Direct3D;

namespace Gp_prac3
 {
public partial class Form1 : Form
    {
private Microsoft.DirectX.Direct3D.Device device;
privateCustomVertex.PositionTextured[] vertex = new CustomVertex.PositionTextured[3];
private Texture texture;
public Form1()
        {
InitializeComponent();
InitDevice();
        }
private void InitDevice()
        {
PresentParameterspp = new PresentParameters();
pp.Windowed = true;
pp.SwapEffect = SwapEffect.Discard;
            device = new Device(0,DeviceType .Hardware ,this,
            CreateFlags.HardwareVertexProcessing, pp);

            device.Transform.Projection = Matrix.PerspectiveFovLH(3.14f / 4,
            device.Viewport.Width / device.Viewport.Height, 1f, 1000f);

device.Transform.View = Matrix.LookAtLH(new Vector3(0, 0, 20), new Vector3(), new
Vector3(0, 1, 0));

device.RenderState.Lighting = false;

vertex[0] = new CustomVertex.PositionTextured(new Vector3(0, 0, 0), 0, 0);
vertex[1] = new CustomVertex.PositionTextured(new Vector3(5, 0, 0), 0, 1);
```

```
vertex[2] = new CustomVertex.PositionTextured(new Vector3(0, 5, 0),-1, 1);
texture=new Texture (device,new Bitmap ("E:\\TYCS\\images\\img1.jpg"), 0,
Pool.Managed );
        }

private void Form1_Load(Object sender, EventArgs e)
        { }

private void Form1_Paint(Object sender, PaintEventArgs e)
        {
device.Clear(ClearFlags.Target, Color.CornflowerBlue, 1, 0);
device.BeginScene();
device.SetTexture(0,texture);
device.VertexFormat = CustomVertex.PositionTextured.Format;
device.DrawUserPrimitives(PrimitiveType.TriangleList, vertex.Length / 3, vertex);
device.EndScene();
device.Present();
        }
    }
}
```

Output:

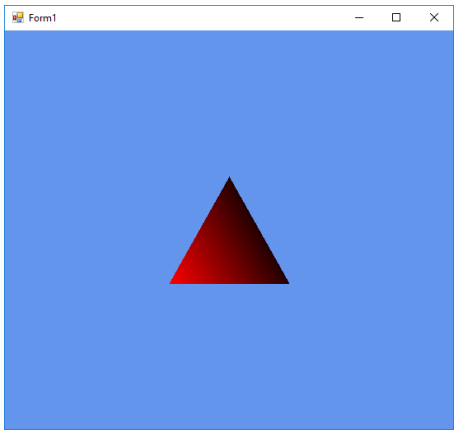


Practical No. 4:
Programmable Diffuse Lightning using Direct3D 11

```
using System;
usingSystem.Collections.Generic;
usingSystem.ComponentModel;
usingSystem.Data;
usingSystem.Drawing;
usingSystem.Text;
usingSystem.Windows.Forms;
usingMicrosoft.DirectX;
using Microsoft.DirectX.Direct3D;
namespace GP_P2
{
public partial class Form1 : Form
   {
private Microsoft.DirectX.Direct3D.Device device;
privateCustomVertex.PositionNormalColored[] vertex = new
CustomVertex.PositionNormalColored[3];
public Form1()
      {
InitializeComponent();
InitDevice();
      }
public void InitDevice()
      {
PresentParameterspp = new PresentParameters();
pp.Windowed = true;
pp.SwapEffect = SwapEffect.Discard;
device = new Device(0, DeviceType.Hardware, this, CreateFlags.HardwareVertexProcessing,
pp);
device.Transform.Projection = Matrix.PerspectiveFovLH(3.14f / 4, device.Viewport.Width /
device.Viewport.Height, 1f, 1000f);
device.Transform.View = Matrix.LookAtLH(new Vector3(0, 0, 10), new Vector3(), new
Vector3(0, 1, 0));
device.RenderState.Lighting = false;
vertex[0] = new CustomVertex.PositionNormalColored(new Vector3(0, 1, 1), new Vector3(1, 0,
1), Color.Red.ToArgb());
vertex[1] = new CustomVertex.PositionNormalColored(new Vector3(-1, -1, 1), new Vector3(1, 0,
1), Color.Red.ToArgb());
vertex[2] = new CustomVertex.PositionNormalColored(new Vector3(1, -1, 1), new Vector3(-1, 0,
1), Color.Red.ToArgb());
device.RenderState.Lighting = true;
device.Lights[0].Type = LightType.Directional;
device.Lights[0].Diffuse = Color.Plum;
device.Lights[0].Direction = new Vector3(0.8f, 0, -1);
device.Lights[0].Enabled = true;
```

```
        }
public void Render()
        {
            device.Clear(ClearFlags.Target, Color.CornflowerBlue, 1, 0);
            device.BeginScene();
            device.VertexFormat = CustomVertex.PositionNormalColored.Format;
            device.DrawUserPrimitives(PrimitiveType.TriangleList, vertex.Length / 3, vertex);
            device.EndScene();
            device.Present();
        }
private void Form1_Load(object sender, EventArgs e)
        {
        }
private void Form1_Paint(object sender, PaintEventArgs e)
        {
Render();
        }
    }
}
```

Output:



Practical no 5

AIM: Specular Lightning (Programmable Spot Lightning using Direct3D 11)

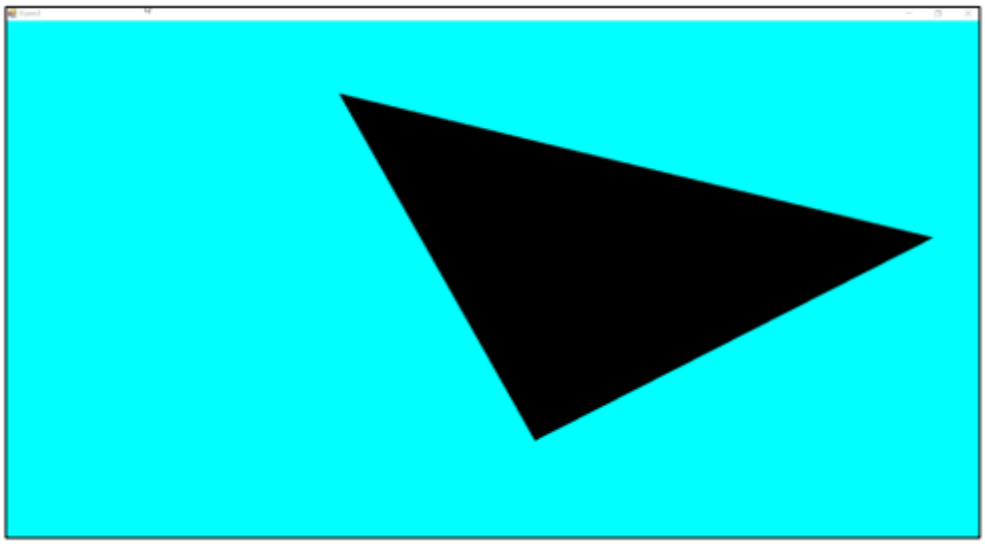
1) After completing the steps from the initialization file, now open
“Form1.cs” file in your project, and code the part where it is commented as //OUR CODE

-----Form1.cs------

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using Microsoft.DirectX.Direct3D; //OUR CODE
using Microsoft.DirectX; //OUR CODE
namespace p11
{
public partial class Form1 : Form
{
private Device device; //OUR CODE
private float angle = 0f; //OUR CODE
public Form1()
{
InitializeComponent();
InitDevice(); //OUR CODE
this.SetStyle(ControlStyles.AllPaintingInWmPaint | ControlStyles.Opaque,true); //OUR CODE
}
private void InitDevice() //OUR CODE
{
PresentParameters pp = new PresentParameters();
pp.Windowed = true;
pp.SwapEffect = SwapEffect.Discard;
device = new Device(0, DeviceType.Hardware, this,CreateFlags.SoftwareVertexProcessing,pp);
device.RenderState.CullMode = Cull.None;
device.RenderState.Lighting = true;
device.Lights[0].Type = LightType.Spot;
device.Lights[0].Range = 4;
device.Lights[0].Position = new Vector3(0, -1, 0f);
device.Lights[0].Enabled = true;
}
private void Render() //OUR CODE
{
```

```
device.Transform.Projection = Matrix.PerspectiveFovLH((float)Math.PI / 4,this.Width /
this.Height, 1f, 50f);
device.Transform.View = Matrix.LookAtLH(new Vector3(0, 0, 30),
new Vector3(1, 0, 0), new Vector3(0, 5, 0));
CustomVertex.PositionNormalColored[] vertices = new
 CustomVertex.PositionNormalColored[6];
vertices[0].Position = new Vector3(10f, 12f, 0f);
vertices[0].Normal = new Vector3(0, 2, 0.5f);
vertices[0].Color = Color.Yellow.ToArgb();
vertices[1].Position = new Vector3(-5f, 5f, 0f);
vertices[1].Normal = new Vector3(0, 2, 0.5f);
vertices[1].Color = Color.Blue.ToArgb();
vertices[2].Position = new Vector3(5f, 5f, -1f);
vertices[2].Normal = new Vector3(0, 0, 0.5f);
vertices[2].Color = Color.Pink.ToArgb();
vertices[3].Position = new Vector3(5f, -5f, -1f);
vertices[3].Normal = new Vector3(0, 0, 0.5f);
vertices[3].Color = Color.Green.ToArgb();
vertices[4].Position = new Vector3(10f, 12f, 0f);
 vertices[4].Normal = new Vector3(0, 0, 0.5f);
vertices[4].Color = Color.Green.ToArgb();
device.Clear(ClearFlags.Target, Color.Cyan, 1.0f, 0);
device.BeginScene();
Vector3 v;
device.VertexFormat = CustomVertex.PositionNormalColored.Format;
device.Transform.World = Matrix.Translation(-5, -10 * 1 / 3, 0) *
Matrix.RotationAxis(new Vector3(), 0);
Console.WriteLine(device.Transform.World.ToString());
device.DrawUserPrimitives(PrimitiveType.TriangleStrip, 3, vertices);
device.EndScene();
device.Present();
this.Invalidate();
}
private void Form1_Paint(object sender, PaintEventArgs e)
{
Render(); //OUR CODE
}
}
}
```

2) Click the Start button to run file > (you'll see the following OUTPUT of the window with



Practical no 6

AIM: Loading models into DirectX 11 and rendering.

1) After completing the steps from the initialization file(“base setup.pdf”), now open “Form1.cs” file in your project, and code the part where it is commented as //OUR CODE

-----Form1.cs------

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using Microsoft.DirectX; //OUR CODE
using Microsoft.DirectX.Direct3D; //OUR CODE
namespace p9
{
public partial class Form1 : Form
{
Microsoft.DirectX.Direct3D.Device device; //OUR CODE
Microsoft.DirectX.Direct3D.Texture texture; //OUR CODE
Microsoft.DirectX.Direct3D.Font font; //OUR CODE
public Form1()
{
InitializeComponent();
InitDevice(); //OUR CODE
InitFont(); //OUR CODE
LoadTexture(); //OUR CODE
}
private void InitFont() //OUR CODE
{
System.Drawing.Font f = new System.Drawing.Font("Arial", 16f,FontStyle.Regular);
font = new Microsoft.DirectX.Direct3D.Font(device, f);
}
private void LoadTexture() //OUR CODE
{
texture = TextureLoader.FromFile(device, "D:\\beach.jpg",  400, 400, 1, 0,
Format.A8B8G8R8,Pool.Managed, Filter.Point, Filter.Point, Color.Transparent.ToArgb());
}
private void InitDevice() //OUR CODE
{
PresentParameters pp = new PresentParameters();
pp.Windowed = true;
pp.SwapEffect = SwapEffect.Discard;
```

```
device = new Device(0, DeviceType.Hardware, this,
CreateFlags.SoftwareVertexProcessing, pp);
}
private void Render() //OUR CODE
{
device.Clear(ClearFlags.Target, Color.Cyan, 0, 1);
device.BeginScene();
using (Sprite s = new Sprite(device))
{
s.Begin(SpriteFlags.AlphaBlend);
s.Draw2D(texture,, new Rectangle(0, 0,device.Viewport.Width, device.Viewport.Height),new
sizeF(), new Point(0, 0), 0f, new Point(0, 0),Color.White);
font.DrawText(s, "GAME PROGRAMMING", new Point(0, 0), Color.Black);
s.End();
}
device.EndScene();
device.Present();
}
private void Form1_Paint(object sender, PaintEventArgs e)
{  Render(); //OUR CODE
} } }
```

2) Click the Start button to run file > (you'll see the following OUTPUT of the window with the color you specified).

practical no 7
Roll-a-ball tutorial

1. Introduction to Roll-a-Ball

Environment and Player

1. Setting up the Game

Create Game Object -> 3D Object -> Plane.

Rename it as “Ground”

Reset all transform coordinates.

Scale the Ground object appropriately.

(Y axis scale doesn’t work as it doesn’t have any volume. Scaling will be done on x-2 and z-2)
Create Game Object -> 3D Object -> Sphere. Rename it as “Player”

Translate the sphere on Y axis to move a little bit above the origin.(Y-0.5)

Create a new folder “Materials” inside the asset and create a new material.

Rename it “Background”

Change the Albedo property to Blue(0-32-64)

Select the material and drag it to the background view.

Select Directional Light and Transform rotation on Y-axis to 60 to get better lighting

2. Moving the Player

Select Player object and add RigidBody component.

Create a folder to store scripts and attach it to Player object to control player movements.

Validate the game by toggling with the Speed component

Camera and Play Area

3. Moving the Camera

Transform(Y-10) and Rotate(X-45) the “Main Camera” on top so we can view the scene.

Now create a script under Main Camera

Once done associate the Player object in the script area of the camera and test the game.

Camera should move with Player.

4. Setting up the Play Area

Create an empty game object and name it “Walls”(This should be in the end when it comes to hierarchy)

Create a 3D game object Cube and name it “Wall1”.

Create 4 such walls by duplicating to create a boundary around the background.

Test the game and the ball shouldn’t fall off the edges

Collecting, Scoring and Building the game

5. Creating Collectable Objects

Disable the Player game object.

Create a Cube Pickup object and size it to 0.5(in each axes) and rotate to 45 (in each axes)

Attach a script to perform rotation

Once done, validate the rotation of single Pick Up.

Now create a folder of PreFab and drag this created prefab.

Create an empty object and add Pickup inside this.

Duplicate those to the count you want.

Finally create a material and attach it.

6. Collecting Pickup Objects

Edit the script of Player object

Create tags for Prefab and name it “Pick Up”.

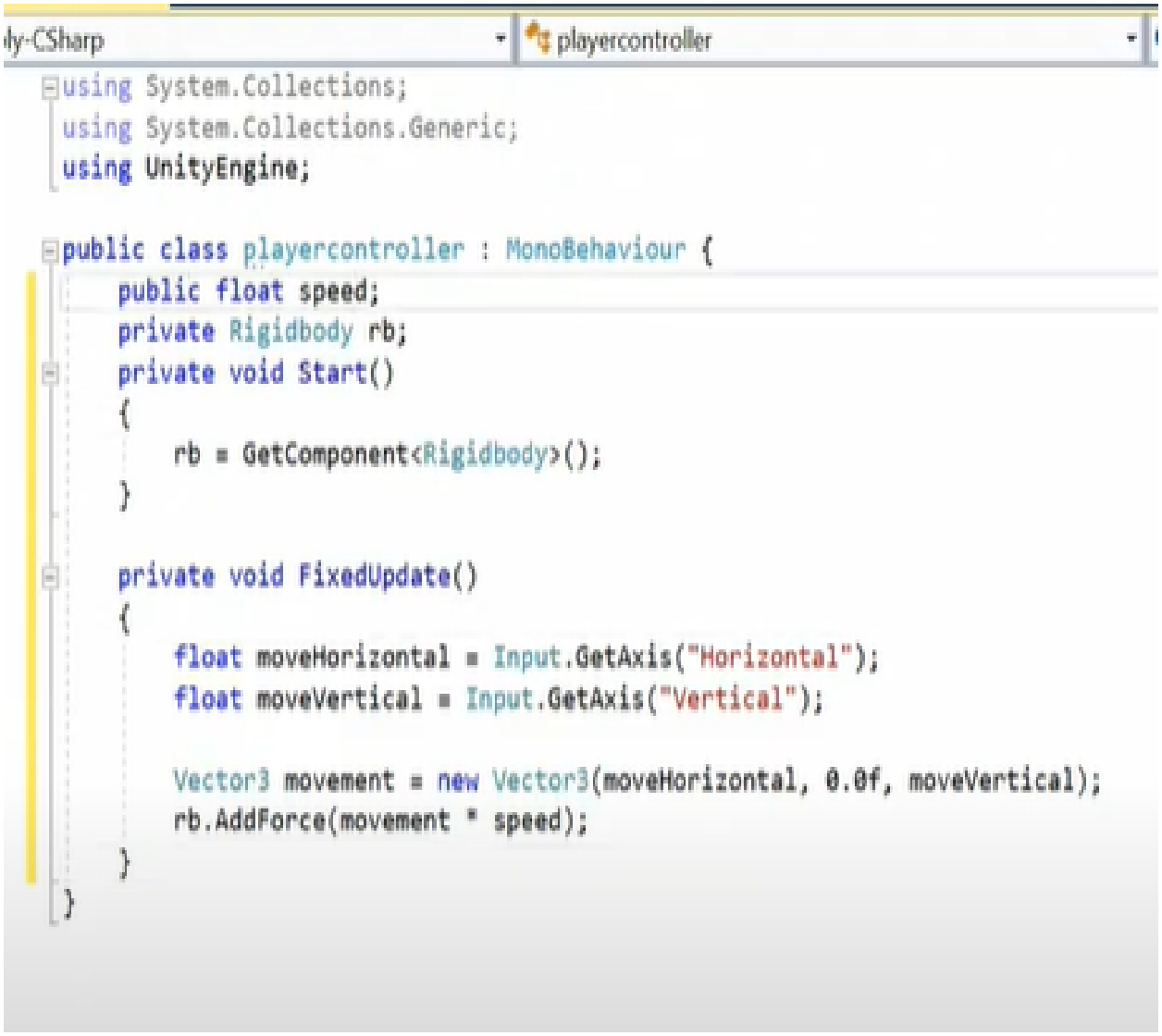
The name in the script should match the Tag name

Add Rigid Body component to PickUp.

Check IsTrigger for Box Collider and IsKinematic also should be checked

7. Displaying Score and Text

1, To rotate a ball (playercontroller)



2. To move the Camera (cameracontroller)

```
bly-CSharp                                  cameracontroller
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class cameracontroller : MonoBehaviour {

    public GameObject player;
    private Vector3 offset;
    private void Start()
    {
        offset = transform.position - player.transform.position;

    }


    // Update is called once per frame
    void Update()
    {
        transform.position = player.transform.position + offset;

    }
}
```

3. To rotate pickups (rotator)

```
bly-CSharp                                  rortaor
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class rortaor : MonoBehaviour {
    void Update()
    {

        transform.Rotate(new Vector3(15, 30, 45) * Time.deltaTime);
    }


}
```

4. To tigger the pickups (write the code in playercontroller)

```
private void OnTriggerEnter(Collider other)
{
    if (other.gameObject.CompareTag("pickup"))
    {
        other.gameObject.SetActive(false);

    }
}
```

5. To count the pickups (Write the code in playercontroller)
Add the namespaces " UnityEngine UI "

```
==========================
public Text countText;
public Text winText;
private int count;

void strat{
count = 0;
setCountText();
winText.text = "";
}
```

```
private void Start()
{
    rb = GetComponent<Rigidbody>();
    count = 0;
    setCountText();
    winText.text = "";
}
```

```
private void OnTriggerEnter(Collider other)
{
    if (other.gameObject.CompareTag("pickup"))
    {
        other.gameObject.SetActive(false);
        count = count + 1;
        setCountText();

    }
```

```
void setCountText()
{
    countText.text = "Count : " + count.ToString();
    if (count >= 6)
    {
        winText.text = "You Win";
    }
}
```

Vector3

describes a vector in 3D space, typically usually used as a point in 3D space or the dimensions of a rectangular prism.

An **offset** is the number of hours or minutes a certain time zone is ahead of or behind

deltaTime is the completion time in seconds since the last frame. This helps us to make the game frame-independent.

the difference between time time and time deltaTime?

time is the time that has passed since the beginning of the cycle, this can be used to get the time since the game started. Time. deltatime is the time that has passed to complete the last frame,

the use of tag in Unity

Tags help you identify GameObjects for scripting purposes. They ensure you don't need to manually add GameObjects to a script's exposed properties using drag and drop, thereby saving time when you are using the same script code in multiple GameObjects.

Practical No. 8:

PlayerController

```
private Rigidbody2D rb2d;
public float speed;
void Start()
{
rb2d = GetComponent<Rigidbody2D> ();
}
void FixedUpdate()
{
float moveHorizontal = Input.GetAxis (“Horizontal”;);
float moveVertical = Input.GetAxis (“Vertical”);
Vector2 movement = new Vector2 (moveHorizontal, moveVertical);
rb2d.AddForce (movement * speed);
}
```

CameraController

```
using UnityEngine;
using System.Collections;
public class CompleteCameraController : MonoBehaviour
 {
public GameObject player;
private Vector3 offset;
void Start ()
{
position. offset = transform.position - player.transform.position;
}
void LateUpdate ()
{
transform.position = player.transform.position + offset;
}
}
```

Rotator

```
Void Update ()
{
transform.Rotate (new Vector3 (0, 0, 45) * Time.deltaTime);
}
```

select the PlayerController script and open it for editing by double clicking

```
using UnityEngine.UI;
public Text countText;
public Text winText;
void SetCountText()
{
countText.text = “Count:” + count.ToString ();
```

```
if (count >= 12)
winText.text =”You win!”
}
void OnTriggerEnter2D(Collider2D other)
{
if (other.gameObject.CompareTag(“PickUp”))
{ other.gameObject.SetActive(false);
} }
```

//Inside Start

```
{
count = 0;
winText.text = “”;;
SetCountText ();
}
```

// Inside OnTriggerEnter2D(Collider2D other)

```
{
if (other.gameObject.CompareTag(“PickUp”))
{
count = count + 1;
SetCountText ();
}
}
```