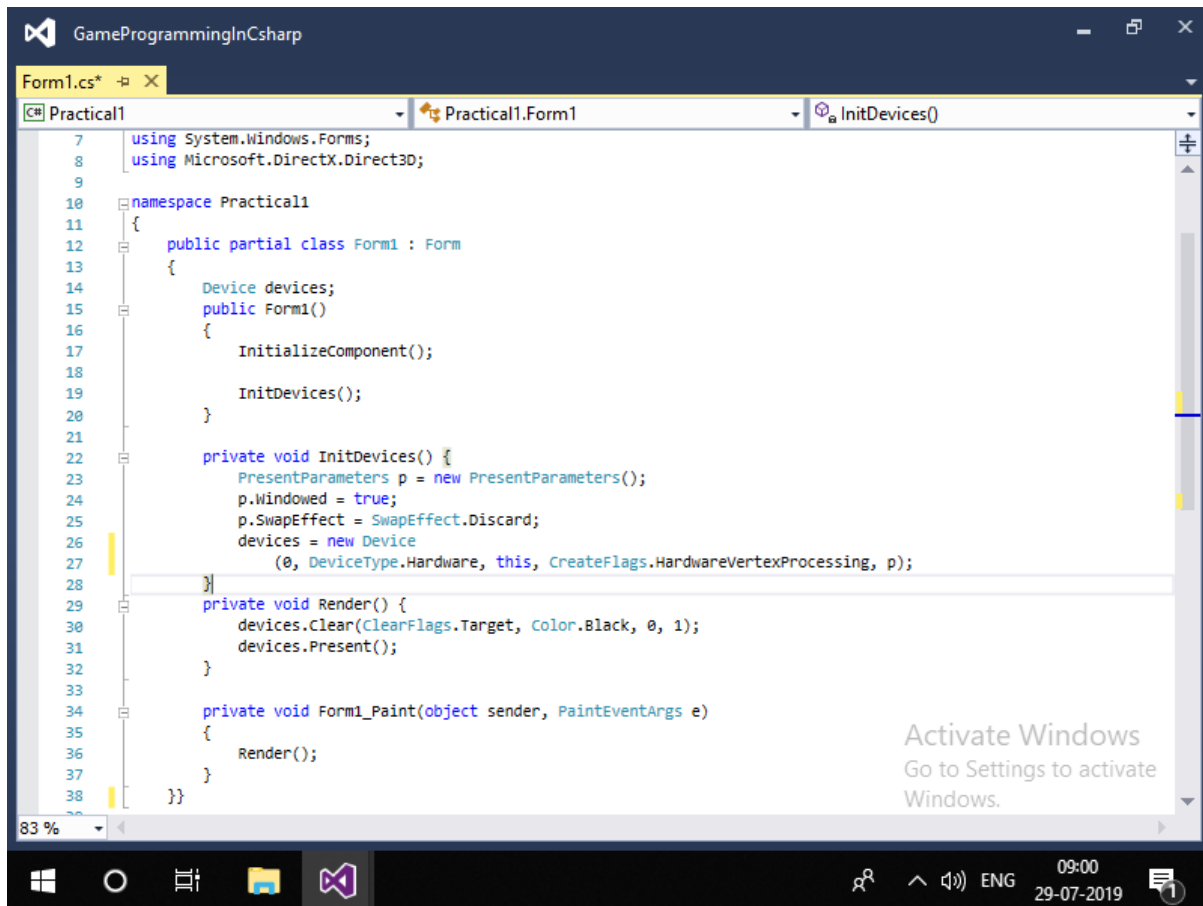
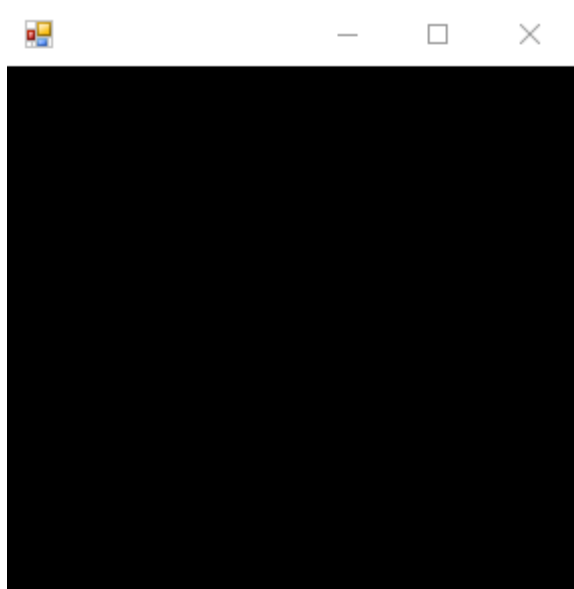


Practical 1



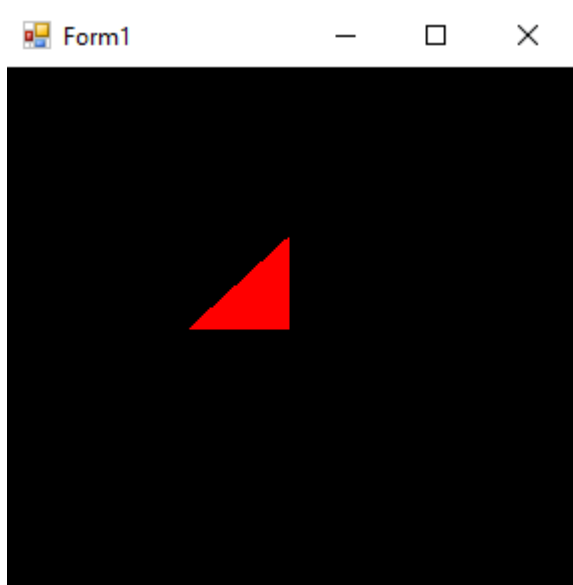
```
7 using System.Windows.Forms;
8 using Microsoft.DirectX.Direct3D;
9
10 namespace Practical1
11 {
12     public partial class Form1 : Form
13     {
14         Device devices;
15         public Form1()
16         {
17             InitializeComponent();
18
19             InitDevices();
20         }
21
22         private void InitDevices() {
23             PresentParameters p = new PresentParameters();
24             p.Windowed = true;
25             p.SwapEffect = SwapEffect.Discard;
26             devices = new Device
27                 (0, DeviceType.Hardware, this, CreateFlags.HardwareVertexProcessing, p);
28         }
29
30         private void Render() {
31             devices.Clear(ClearFlags.Target, Color.Black, 0, 1);
32             devices.Present();
33         }
34
35         private void Form1_Paint(object sender, PaintEventArgs e)
36         {
37             Render();
38         }
39     }
40 }
```

Activate Windows
Go to Settings to activate Windows.



Practical 2

```
using Microsoft.DirectX;
using Microsoft.DirectX.Direct3D;
namespace Practical2
{
    public partial class Form1 : Form
    {
        private Device device;
        private CustomVertex.PositionColored[] vertex
            = new CustomVertex.PositionColored[3];
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            device.Clear(ClearFlags.Target, Color.Black, 1, 0);
            device.BeginScene();
            device.VertexFormat = CustomVertex.PositionColored.Format;
            device.DrawUserPrimitives(PrimitiveType.TriangleList, vertex.Length / 3, vertex);
            device.EndScene();
            device.Present();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            PresentParameters p = new PresentParameters();
            p.Windowed = true;
            p.SwapEffect = SwapEffect.Discard;
            device = new Device(0, DeviceType.Hardware, this, CreateFlags.HardwareVertexProcessing, p);
            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.PositionColored(new Vector3(), Color.Red.ToArgb());
            vertex[1] = new CustomVertex.PositionColored(new Vector3(3, 0, 0), Color.Red.ToArgb());
            vertex[2] = new CustomVertex.PositionColored(new Vector3(0, 3, 0), Color.Red.ToArgb());
        }
    }
}
```



Practical:-3

```
using Microsoft.DirectX;
using Microsoft.DirectX.Direct3D;
namespace Practical3
{
    public partial class Form1 : Form
    {
        private Device device;
        private CustomVertex.PositionTextured[] vertex = new CustomVertex.PositionTextured[3];
        private Texture texture;
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            PresentParameters pp = 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));

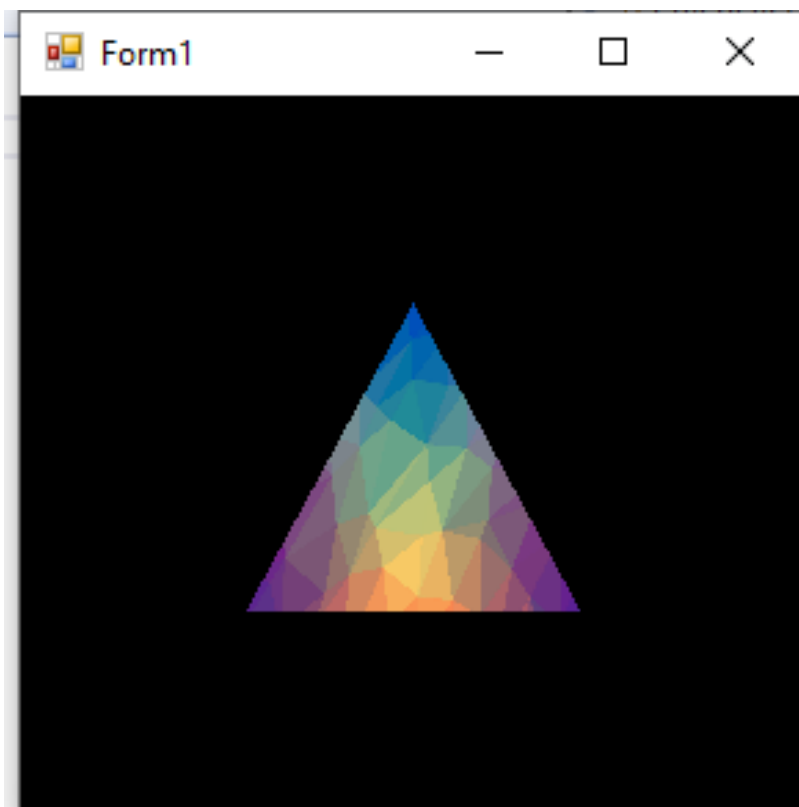
            device.RenderState.Lighting = false;
            vertex[0] = new CustomVertex.PositionTextured(new Vector3(0,3,3),0,0);
            vertex[1] = new CustomVertex.PositionTextured(new Vector3(-3,-3,3),-1,0);
            vertex[2] = new CustomVertex.PositionTextured(new Vector3(3,-3,3),0,-1);
            texture = new Texture(device,
                new Bitmap(@"D:\Game Practicals\GameProgrammingInCsharp\GameProgrammingInCsharp\Practical3\triangles.png"),
                0, Pool.Managed);
        }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            device.Clear(ClearFlags.Target, Color.Black, 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();
        }
    }
}
```

Texture to be applied



OUTPUT:-



Practical:-4

```
using Microsoft.DirectX;
using Microsoft.DirectX.Direct3D;
namespace Practical4
{
    public partial class Form1 : Form
    {
        private Device device;
        private CustomVertex.PositionNormalColored[] vertex =
            new CustomVertex.PositionNormalColored[3];

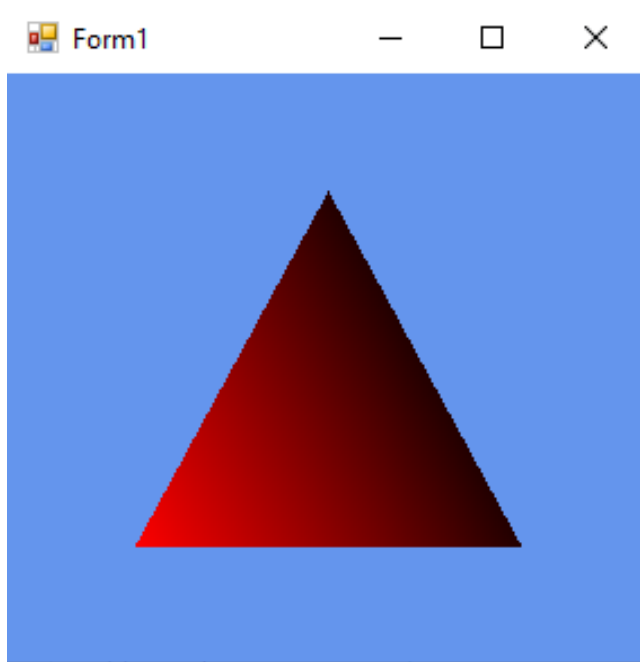
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            PresentParameters pp = 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, 2, 2), new Vector3(1, 0, 1), Color.Red.ToArgb());
            vertex[1] = new CustomVertex.PositionNormalColored(new Vector3(-2, -2, 2), new Vector3(1, 0, 1), Color.Red.ToArgb());
            vertex[2] = new CustomVertex.PositionNormalColored(new Vector3(2, -2, 2), 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;
        }

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

OUTPUT:-



Practical:-5

```
using Microsoft.DirectX;
using Microsoft.DirectX.Direct3D;

namespace Practical5
{
    public partial class Form1 : Form
    {
        Device device;
        Texture texture;
        Microsoft.DirectX.Direct3D.Font font;
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            PresentParameters pp = new PresentParameters();
            pp.Windowed = true;
            pp.SwapEffect = SwapEffect.Discard;
            device = new Device(0, DeviceType.Hardware, this,
                CreateFlags.HardwareVertexProcessing, pp);
            System.Drawing.Font f = new System.Drawing.Font("Arial", 16f, FontStyle.Regular);

            font = new Microsoft.DirectX.Direct3D.Font(device, f);
            texture = TextureLoader.FromFile(device,
                @"D:\Game Practicals\GameProgrammingInCsharp\GameProgrammingInCsharp\Practical5\multicolor-triangle.jpg",
                400, 400, 1, 0, Format.A8B8G8R8, Pool.Managed, Filter.Point, Filter.Point, Color.Transparent.ToArgb());
        }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            device.Clear(ClearFlags.Target, Color.CornflowerBlue, 0, 1);
            device.BeginScene();
            using (Sprite s = new Sprite(device)) {
                s.Begin(SpriteFlags.AlphaBlend);
                s.Draw2D(texture, new Rectangle(0, 0, 0, 0),
                    new Rectangle(0, 0, device.Viewport.Width, device.Viewport.Height),
                    new Point(0, 0), 0f, new Point(0, 0), Color.White);
                // font.DrawText(s, "Paul ", new Point(0, 0), Color.White);
                s.End();
            }
            device.EndScene();
            device.Present();
        }
    }
}
```

OUTPUT:-



Practical-6

Creating a 2D UFO game.

1. We start by creating a 2d new project and make sure to import assets from 2D UFO tutorial.
2. Next we drag and drop sprites from the sprite folder in our scene view, first the game background and next the ufo object.
3. Then you add a Rigidbody2D component to ufo object. This has to be done to apply any type of physics logic to your component.
4. Now we want our object to move so add a new script and edit it as follows:
5. Next you may observe that your ufo goes down when you start to play, so make gravity to zero from Rigidbody2d property.
6. Now your object should move properly. But still the object isn't bound in space it moves everywhere around. So here we require more physics component, we would add colliders to our ufo and the walls of our game background.
7. Add a Circle Collider 2d to ufo object, adjust the radius as appropriate.
8. Add a Box Collider 2d to background, now we need to adjust it in such a way that one side of the background is covered, it should look like a rectangle when we adjust. Do the same for all the sides and we should have our boundaries ready. If you play now you should see that our ufo collides with each of the wall and does not escape. Now to complete our game we need some asteroid like structure, so when the ufo collides with it then the asteroid disappears. What we need is now to add the asteroid sprite named pickup from sprites folder to our scene view.
9. Since we need some collision to occur, we will add Circle Collider 2D Component to pickup object.
10. Now we need this object everywhere in our background, so simply right click and duplicate it, change the x and y position and place it all around the ufo.

Now if play it you may observe that our ufo collides with each one of the pickups or asteroids in surrounding, all we need is them to disappear when they are hit.

11. First thing we need to do is make our pickup object a prefab. So for this just drag the pickup from hierarchy and drop it the project assets folder. Now because of this you could adjust every other pickup behaviour using the prefab.
12. Select the prefab and in the inspector(properties) window add tag (Pick Up) if not set, this will allow us to use it in the script part for our player(ufo). Also in Circle Collider 2d part check the on trigger event as we would be using for collision check.
13. Now we edit our script create a new OnTriggerEnter2D (Collision), and edit as given. Now our game should work fine and the pickups must disappear once you collide with them. One final thing we would like to add, is some text to display our count of collecting pickups and to display win as we pick our last pickup.
15. So from hierarchy add text element and adjust it in top left of screen. Add another text and place it above the ufo. Give them some appropriate tags as we will be using them in player (ufo) script. Make the appropriate changes as given in the script and then our game is ready.

```

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

public class PlayerController : MonoBehaviour {

    // Use this for initialization

    public Text countT, winT;
    private int count;
    public float speed;
    private Rigidbody rb;

    void Start () {
        rb=GetComponent<Rigidbody>();
        count=0;
        winT.text="";
        setCount();
    }

    // Update is called once per frame
    void FixedUpdate () {
        float mvh=Input.GetAxis("Horizontal");
        float mvv=Input.GetAxis("Vertical");
        Vector3 mv=new Vector3(mvh,0.0f,mvv);

        rb.AddForce(mv*speed);
    }

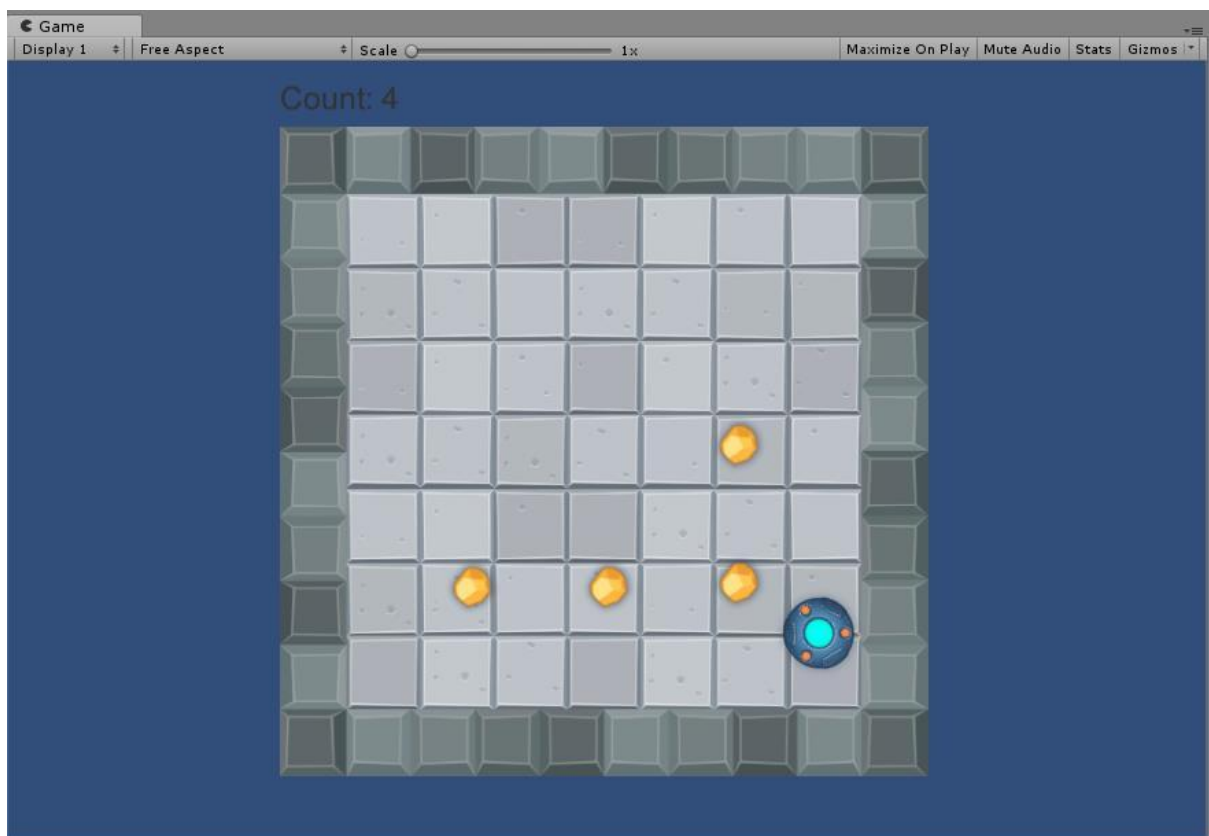
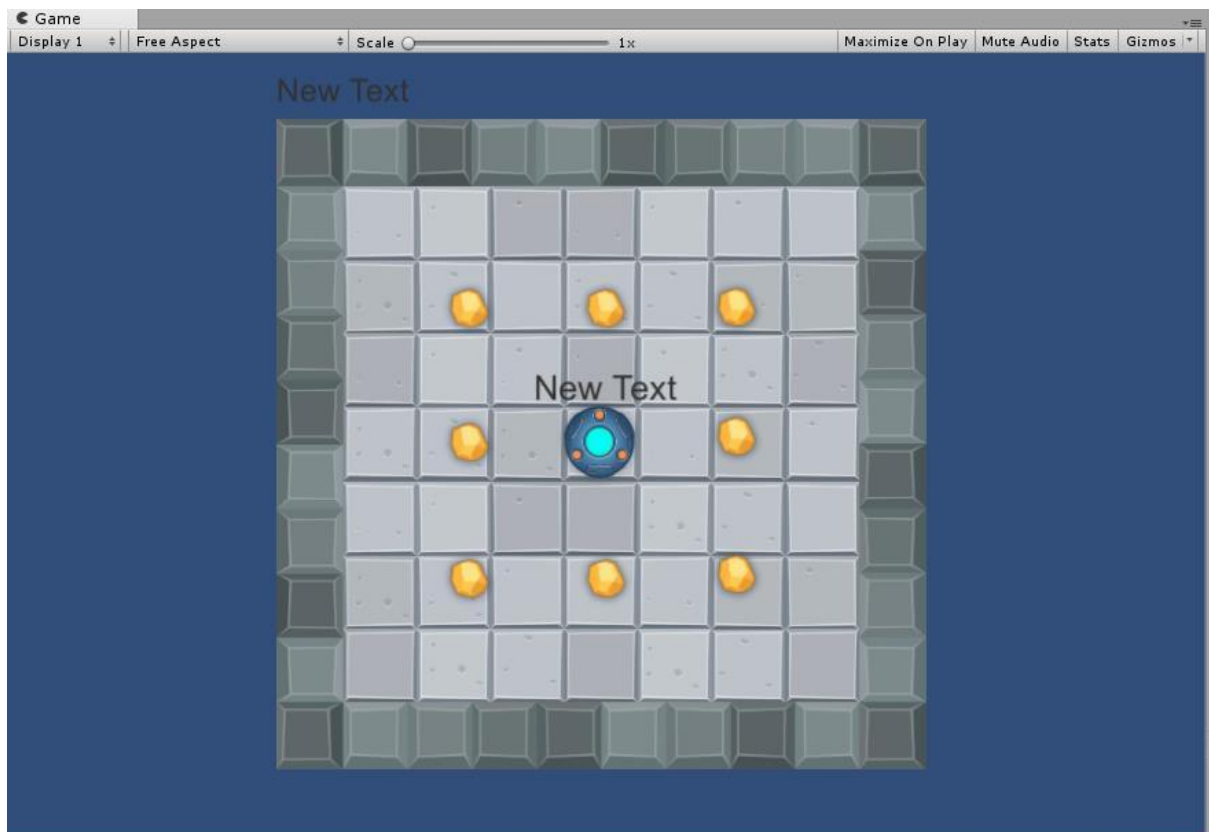
    void OnTriggerEnter(Collider other){

        if(other.gameObject.CompareTag("Pick Up")){

            other.gameObject.SetActive(false);
            count++;
            setCount();
        }
    }

    void setCount(){
        countT.text="Count: "+count.ToString();
        if(count>=7){
            winT.text="You win";
        }
    }
}

```





Practical-7

Creating a 3D game to Roll a ball.

This is very similar to the previous 2D game we created, only the components now we will use and physics around them would be of 3d type.

1. Create a new project, make sure it is of 3d type.
2. Next we will require a some surface in our scene, where we could place our 3d objects , as we are not going to use he sprites. So create a new plane from hierarchy view. You would require to adjust the camera , so reset it to default and rotate it about 90 degree on x axis and then move it towards the positive end of y axis.
3. Next create a sphere (our ball), that will be moving on our surface. So follow the same steps and now create a sphere. By default it is placed at origin so just move it up on y axis(0.5) and it should be perfect.
4. Now let's set boundaries for the ball otherwise it will move out of screen, so add cube the very same way from hierarchy view. Adjust each of them in rectangular component that will represent our walls on all the 4 sides.
5. As you should observe , unlike the previous 2d game we don't really need to add any kind of collider to our components in 3d as they come by default with colliders suited to their shapes. Though for our ball to move around we would need a Rigidbody attach to it, remember 2d and 3d physics system works differently and we should not use Rigidbody2D here.
6. After that attach a new script to sphere from add component in inspector (properties) window. Then edit to take speed as public variable adjusted by the user and use Rigidbody for giving it appropriate force in horizontal and vertical direction. This is very similar the way we moved our component las time around.
7. Now as everything looks white we would require to add some color to our components. So create a material now from you projects view, then in inspector window use he albedo(Color adjuster) to provide the color to material. Now all we would need is drag he material and place it on the component we want to see this color on , and we could do the same for surface and sphere.
8. Until now sphere must move smoothly all around the surface and the colliders should let the components collide. But like previously we would need pickups for our sphere to collect.
9. Add a cube rotate it 45 degrees on each of the axis and material to color it.
10. Now drag it from the hierarchy view and drop it in the projects (Asset part) view,so it could be a prefab. Now simply duplicate it all around the scene to for our ball to collide with it.
11. Once done we need to update our script as well as check the is trigger component for our pick ups, do this by selecting the prefab and then from inspector window select Is Trigger. Also remember to add a tag named Pick Up that we call in our program(use prefab for this too).
12. Next update the script as show here we use OnTriggerEnter() , you may remember we used same method just 2D at end in previous example and the rest remains same.
13. Finally add some text elements from hierarchy view (UI part) and use it in the program for the very similar logic we countered previously.

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
public class PlayerController : MonoBehaviour {

    public Text winT, countT;
    private int count;
    private Rigidbody2D rb2d;
    public float speed;

    void Start(){

        rb2d=GetComponent<Rigidbody2D>();
        count=0;
        winT.text="";
        setCount();
    }

    // Update is called once per frame
    void FixedUpdate () {

        float mvh=Input.GetAxis("Horizontal");
        float mvv=Input.GetAxis("Vertical");
        Vector2 mov =new Vector2(mvh,mvv);

        rb2d.AddForce(mov*speed);
    }

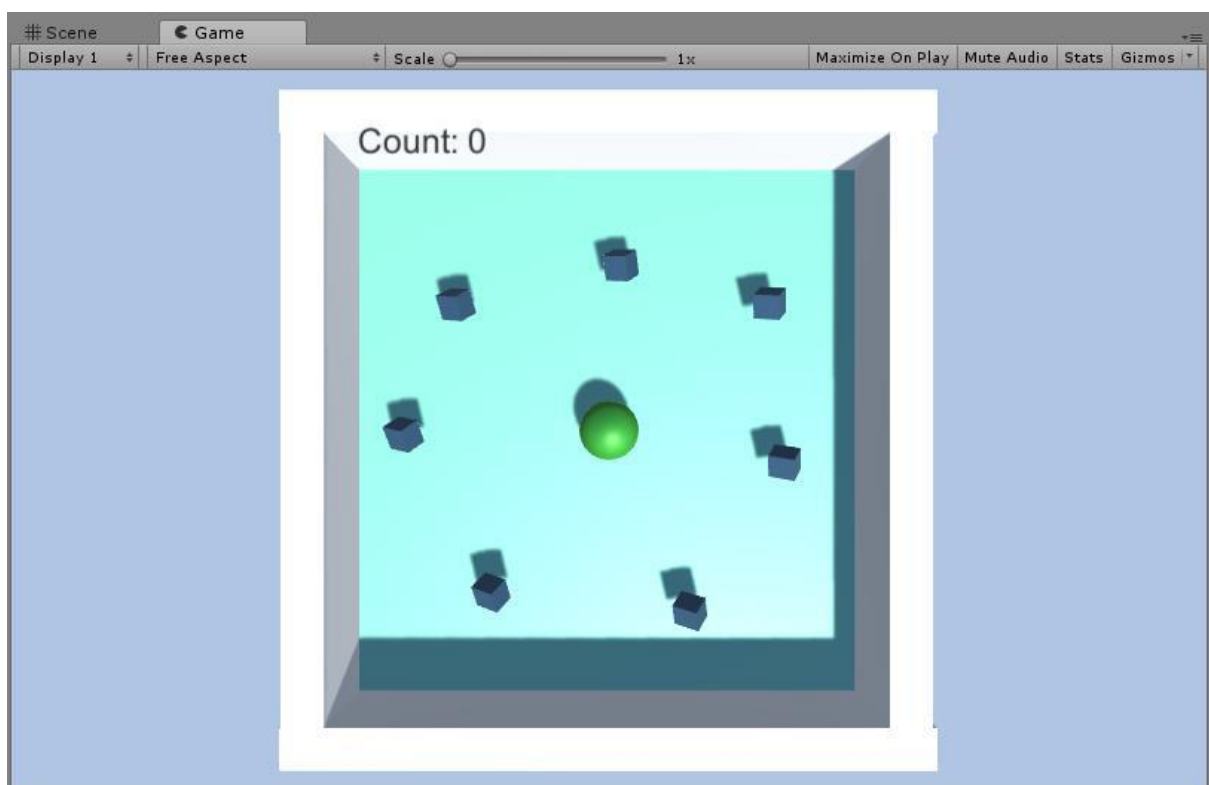
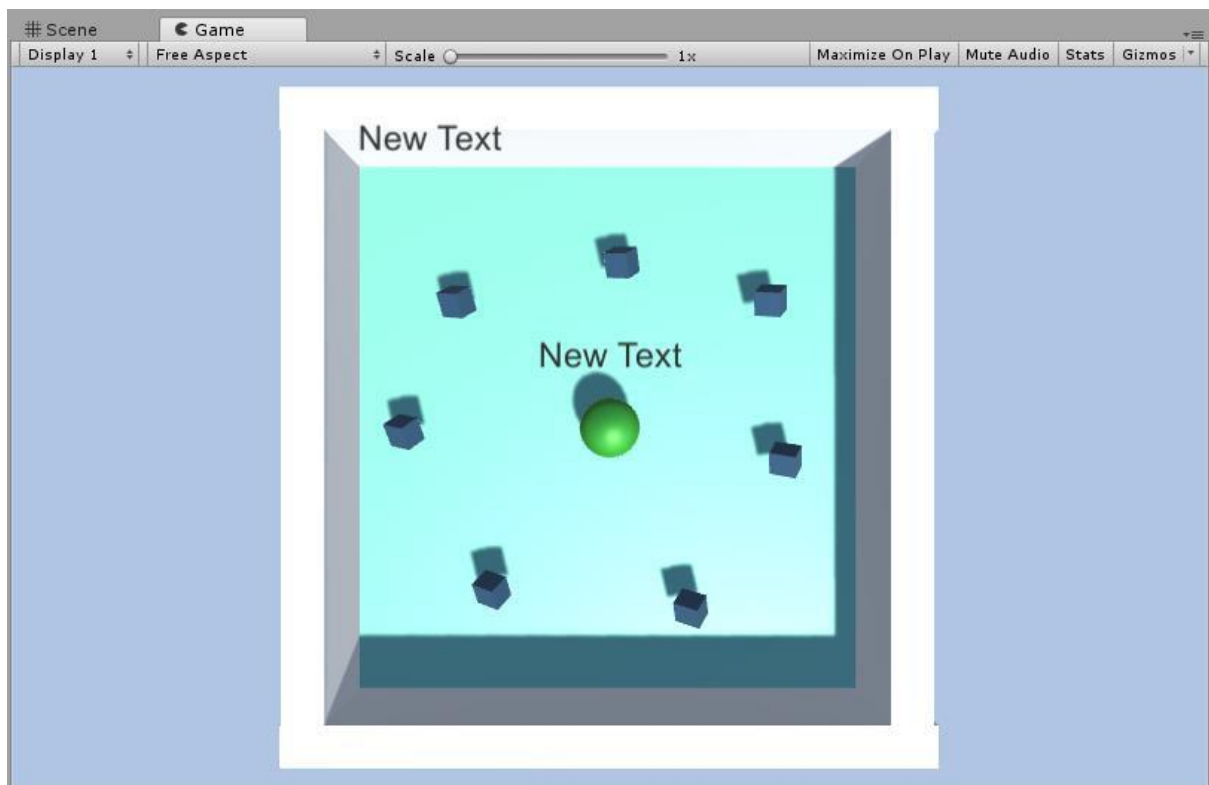
    void OnTriggerEnter2D(Collider2D other){
        if(other.gameObject.CompareTag("PickUp")){

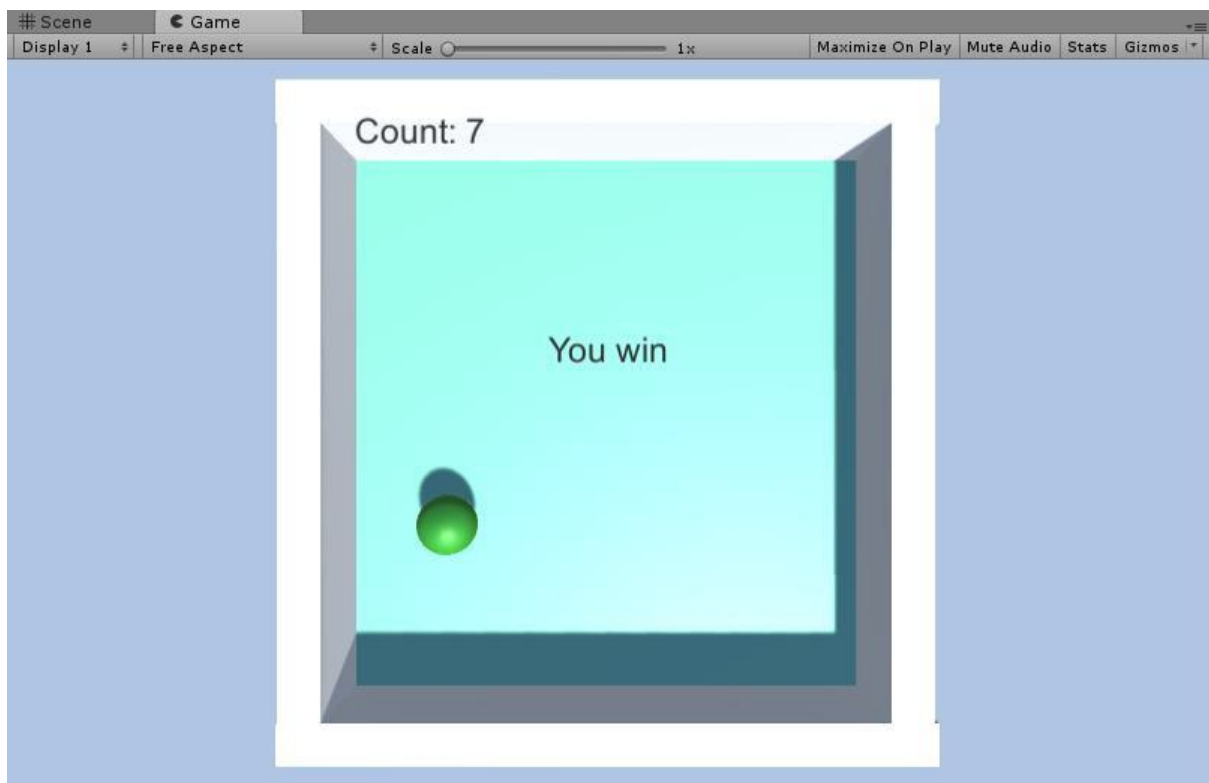
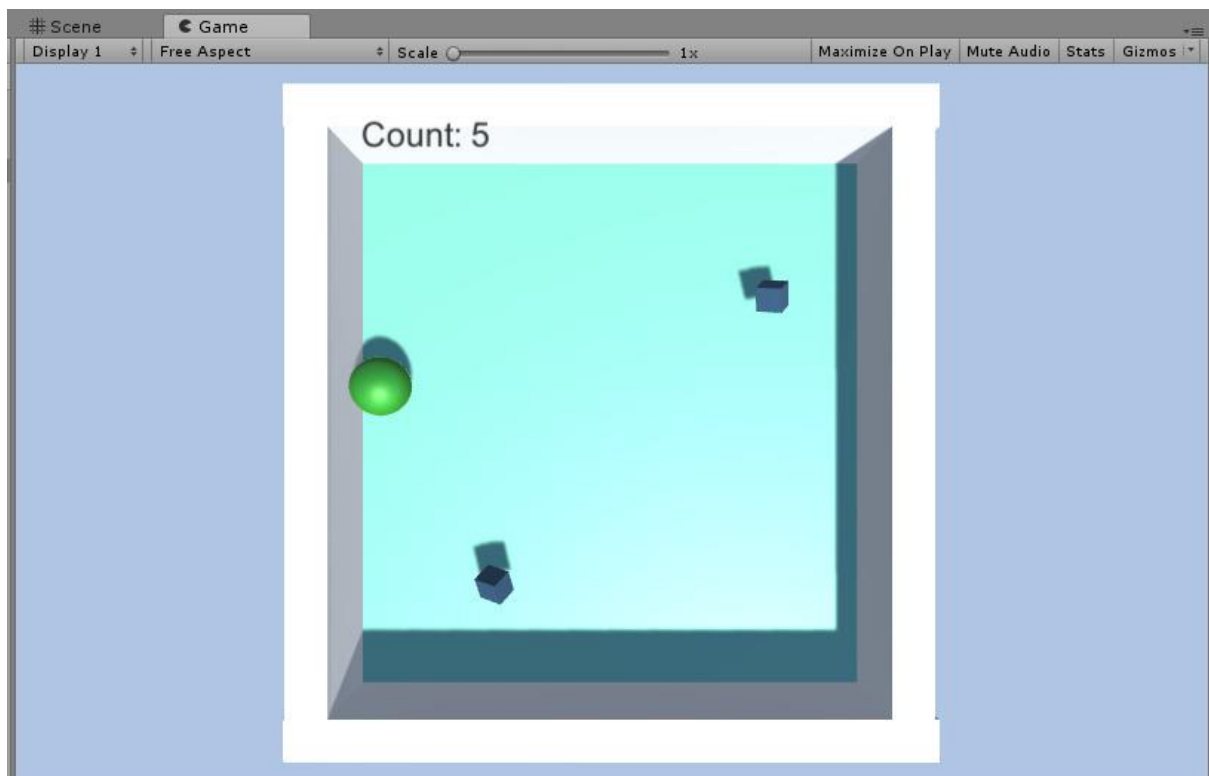
            other.gameObject.SetActive(false);
            count=count+1;
            setCount();
        }
    }

    void setCount(){

        countT.text="Count: "+count.ToString();
        if(count>=8){
            winT.text="You win";
        }
    }
}

```

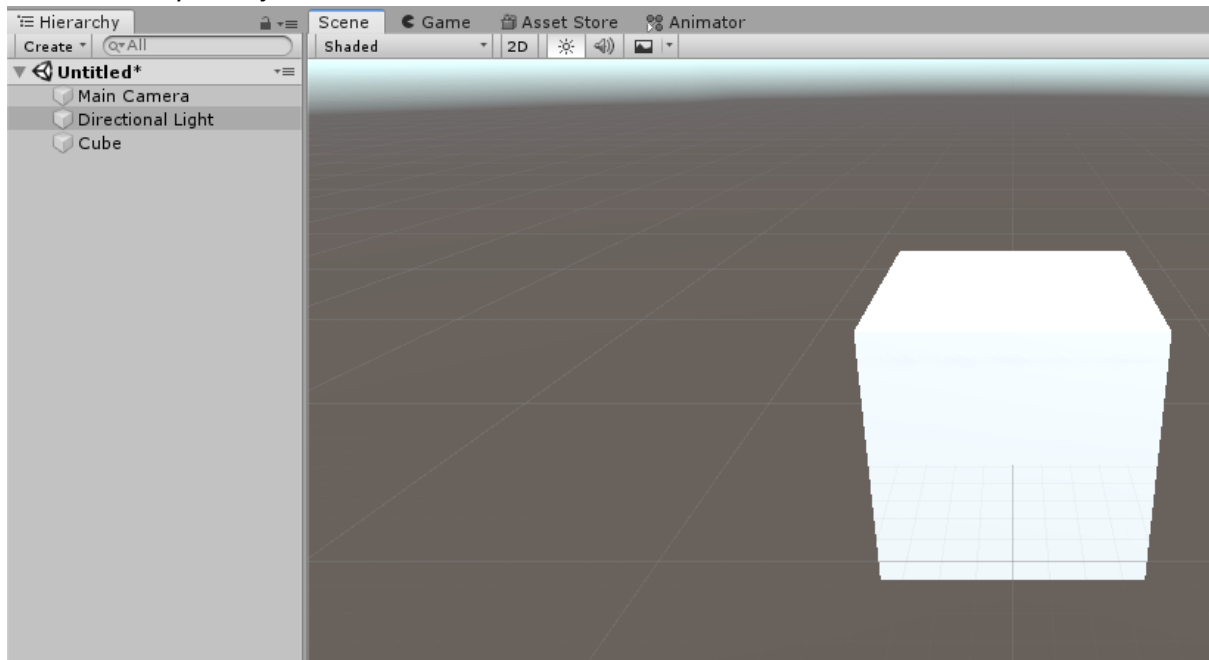




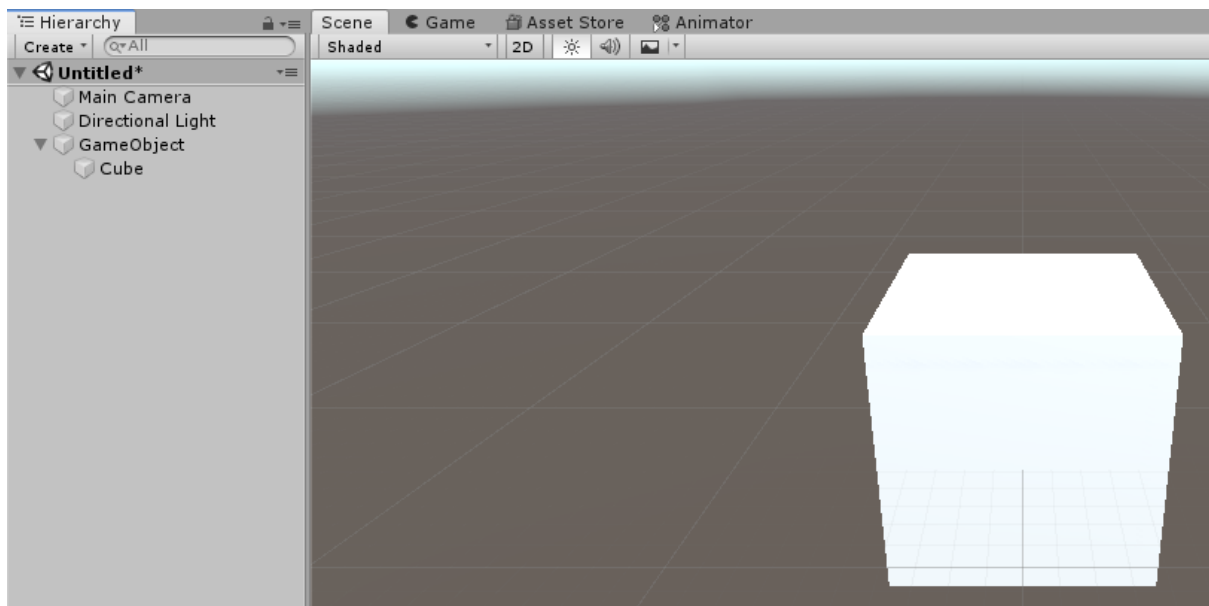
Practical-8

Animating and create prefabs.

Create and any 3d object in our case it's a cube.

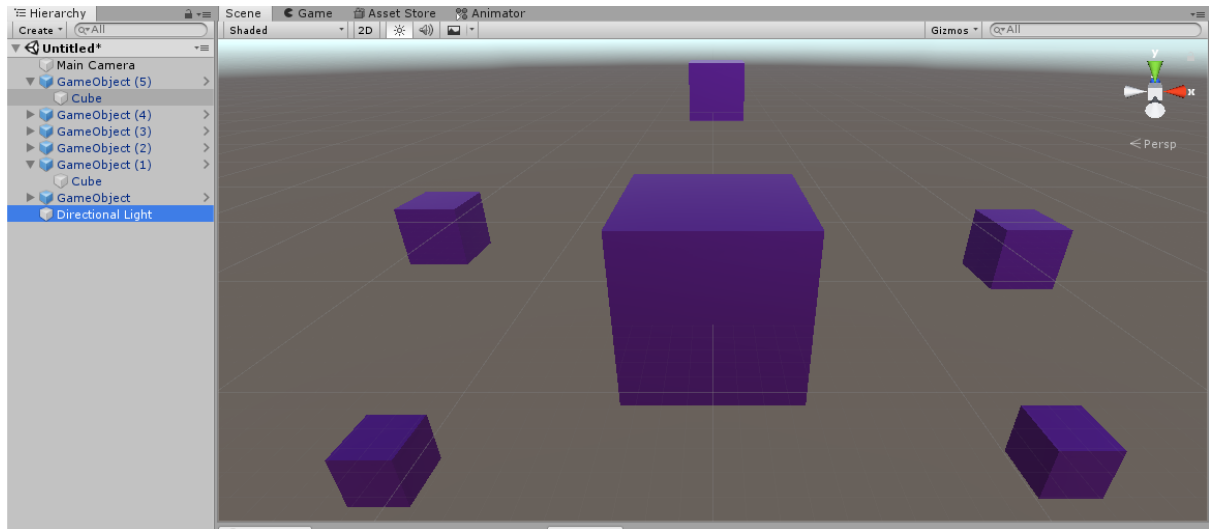


Since we want animation for each object with respect to their position, create an empty game object and make our 3d object a child of it.

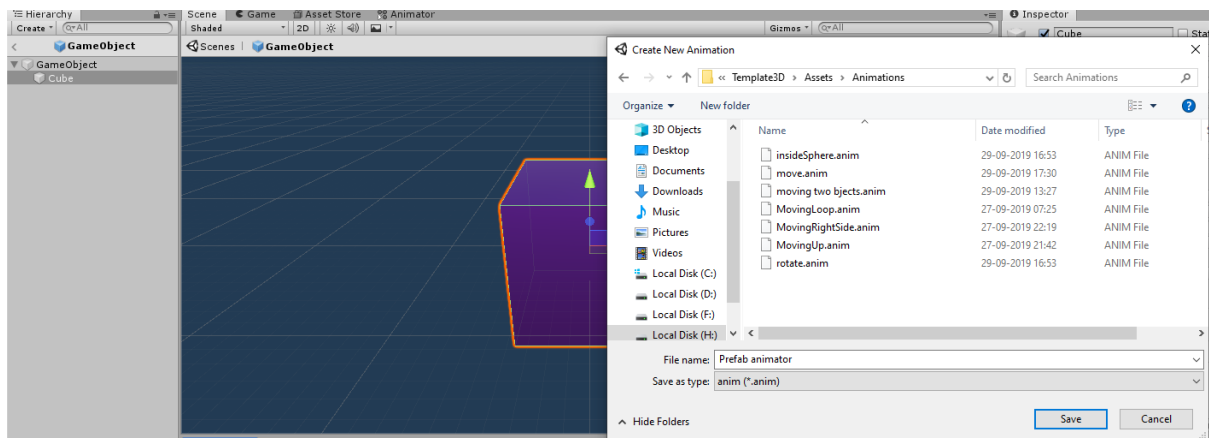


Next we make it a prefab by simply dragging the game object from hierarchy window to project window in assets part the blue color on object specify that it's an prefab asset.

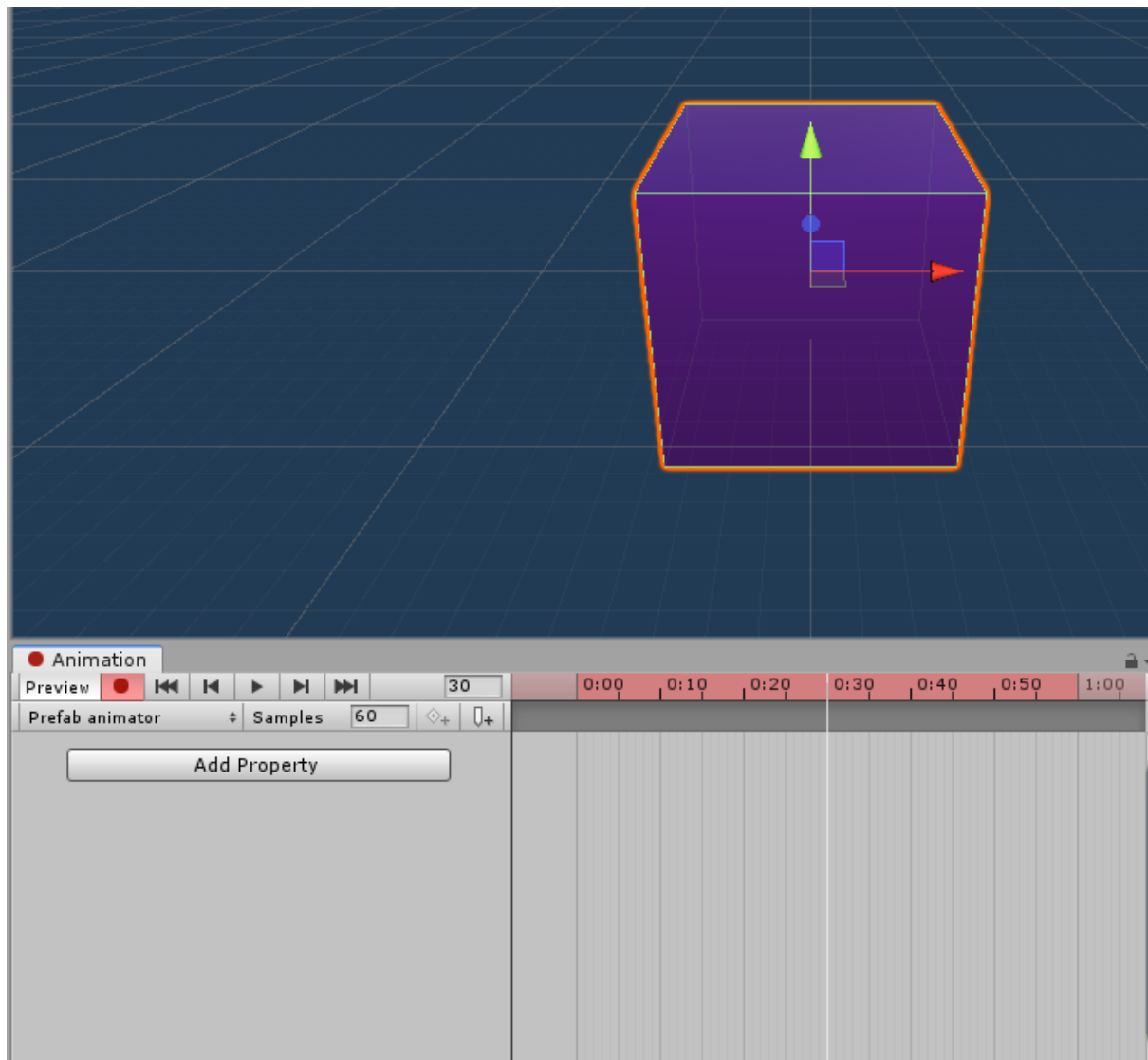
Add few more of them on screen and add color to one of them in prefab mode so that all gets the same color(don't forget to create a material for doing so).



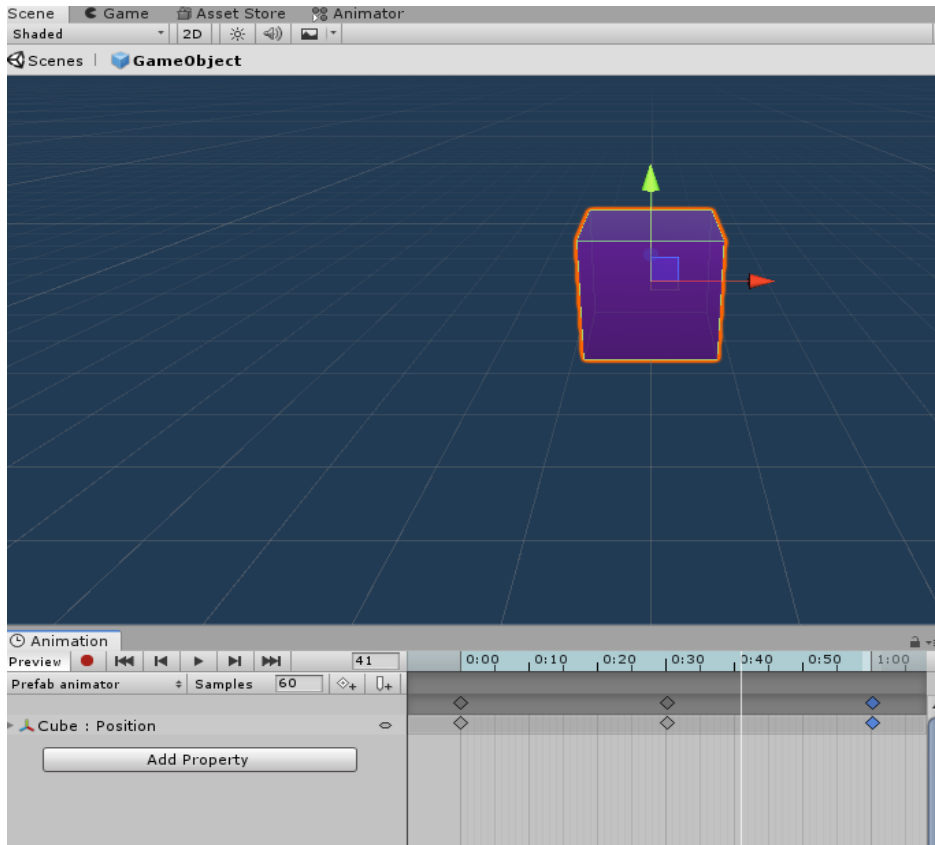
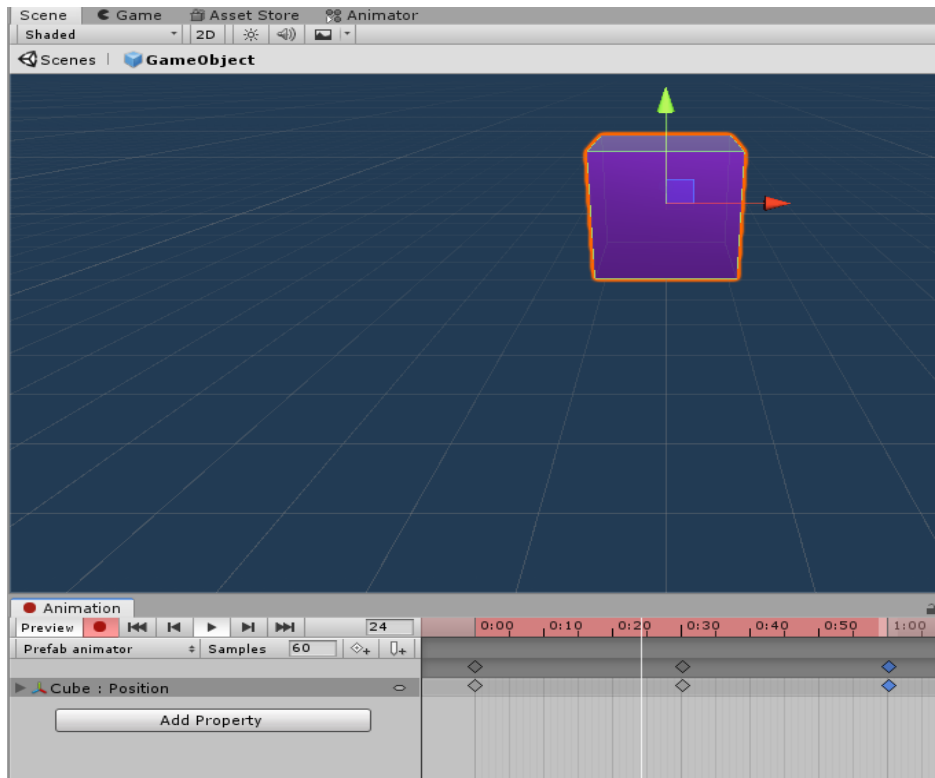
Next we animate them as they are prefabs simply add an animation from prefab window to the object



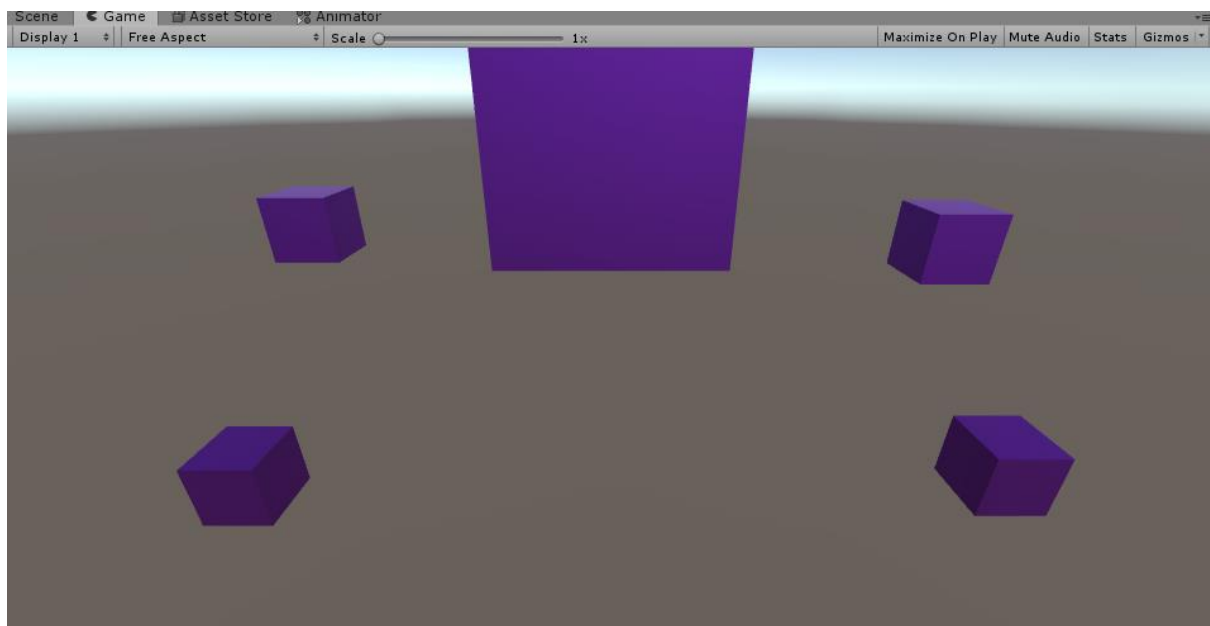
Press the record button in the animation window and set the white marker line to a certain a time.



Next start moving it around in our case we it 3 units in Y direction, after you have finished initial movement copy the start position and place it when you want to end animation and click on red button to stop recording.



Now if you play the game you could see that all the cubes are moving relative to their game object.

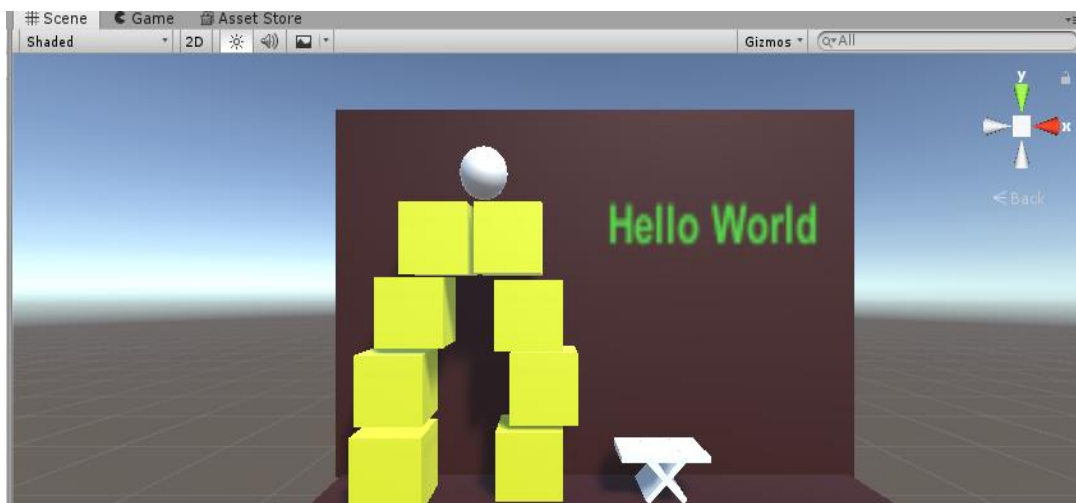


Practical-9

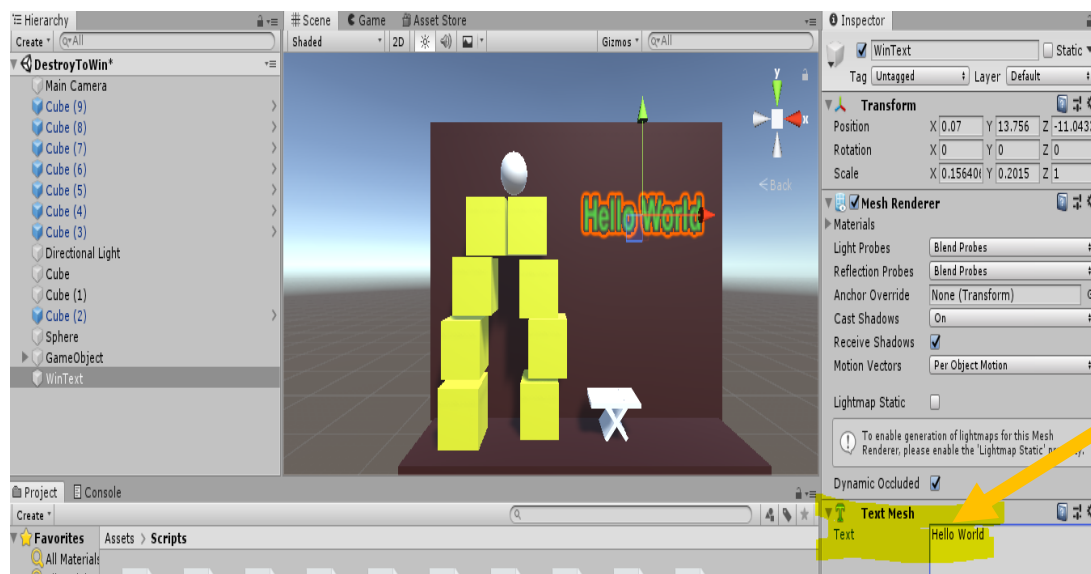
Creating a game where prefabs are destroyed on click

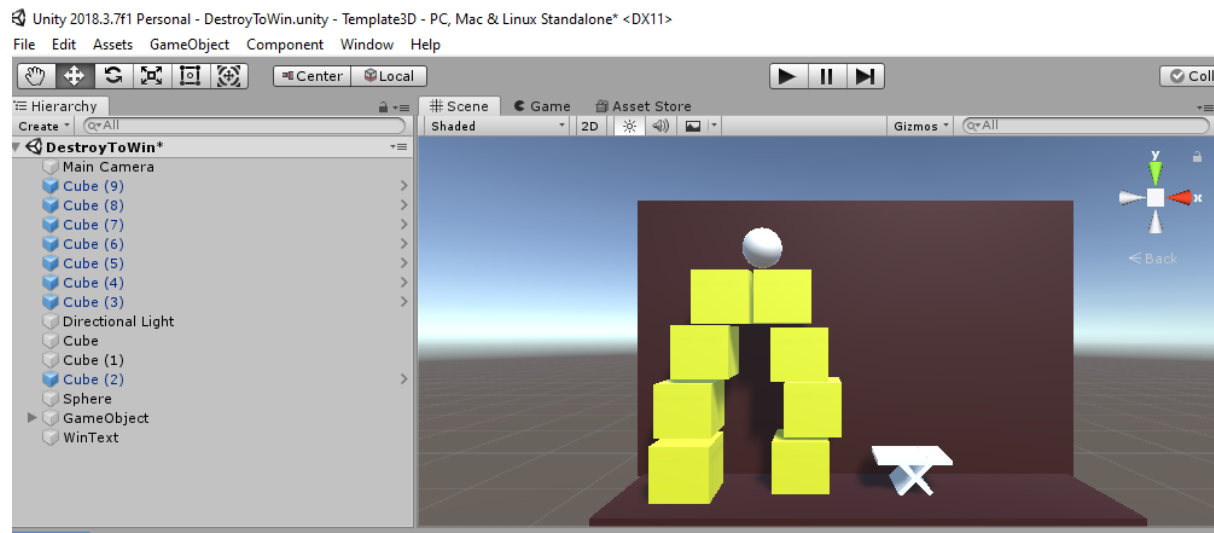
1. Create a cube make it prefab as we have done previously.
2. Arrange them in some order and also create an target(which on collision triggers the win).
3. Add a sphere place it on top of the cubes.
4. Add rigidbody to those components that you want to move in our case that is prefab asset cube and the sphere object.
5. Now add a 3d text by going at hierarchy window -> create 3d object -> 3Dtext.

Place the 3d text accordingly in empty space as shown below.



As we want the text to be displayed only on when the sphere collides with the target, you should set the text property from text mesh to empty as shown(from inspector window)





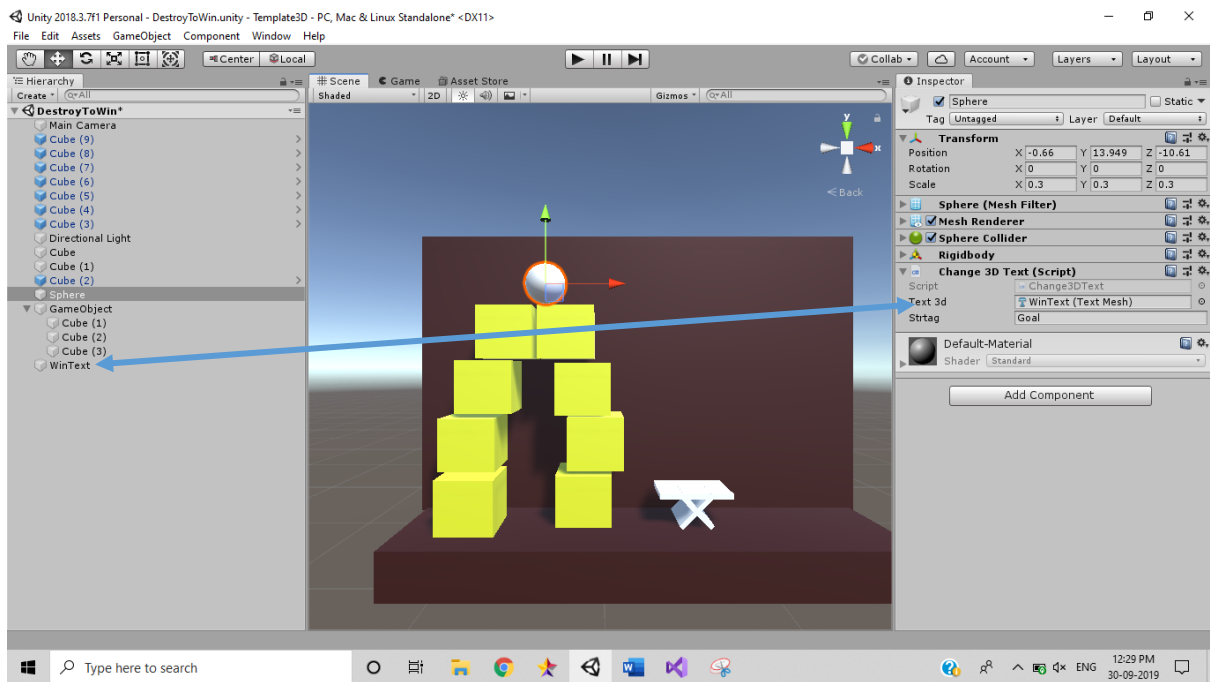
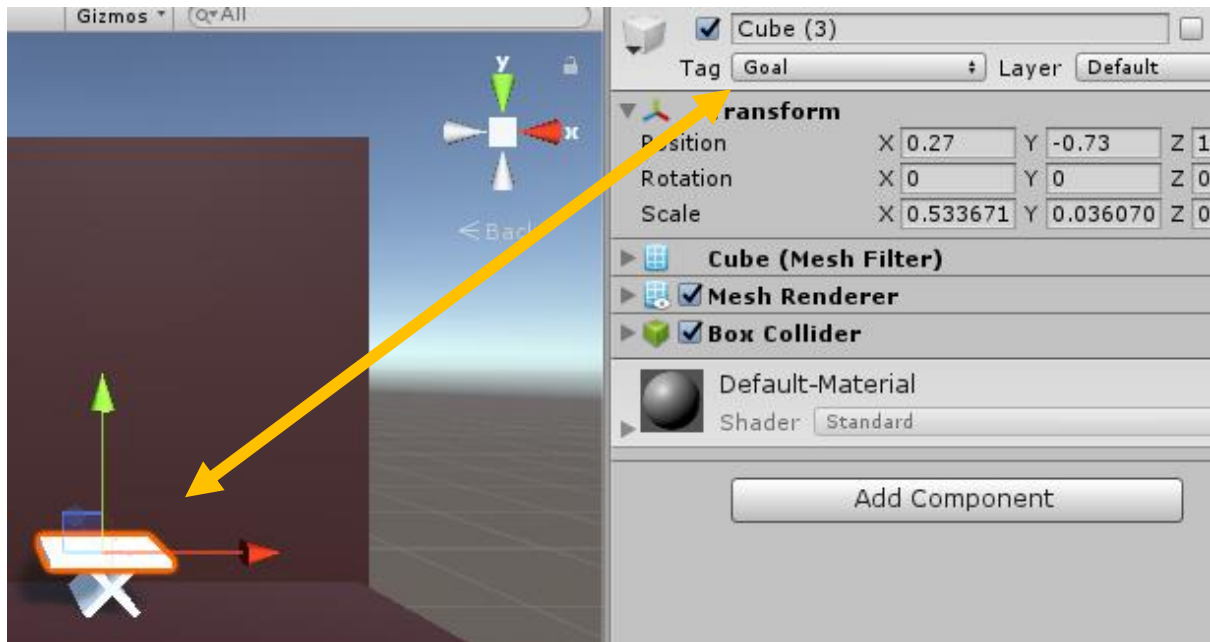
Now attach the following script to it sphere object

```

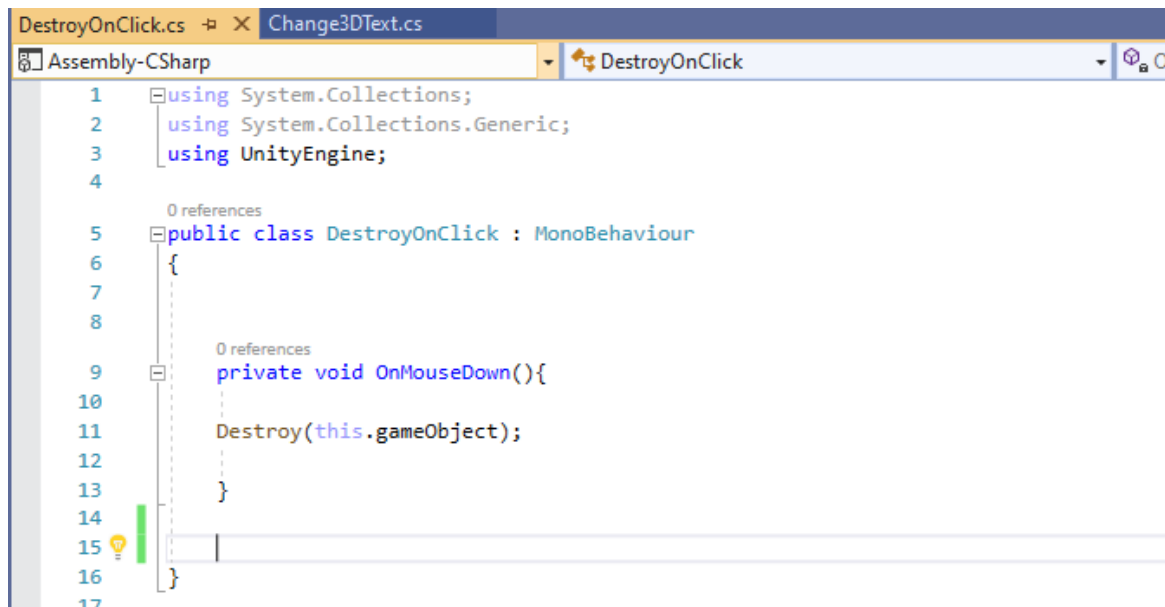
Change3DText.cs
Assembly-CSharp
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5
6  0 references
7  public class Change3DText : MonoBehaviour
8  {
9      public TextMesh text3d;
10     public string strtag;
11     0 references
12     private void OnCollisionEnter(Collision collision)
13     {
14         if (collision.collider.tag == strtag) {
15             text3d.text = "YOU WIN";
16         }
17     }
18 }

```

Also add the references of the text and the tag of the target from inspector window.

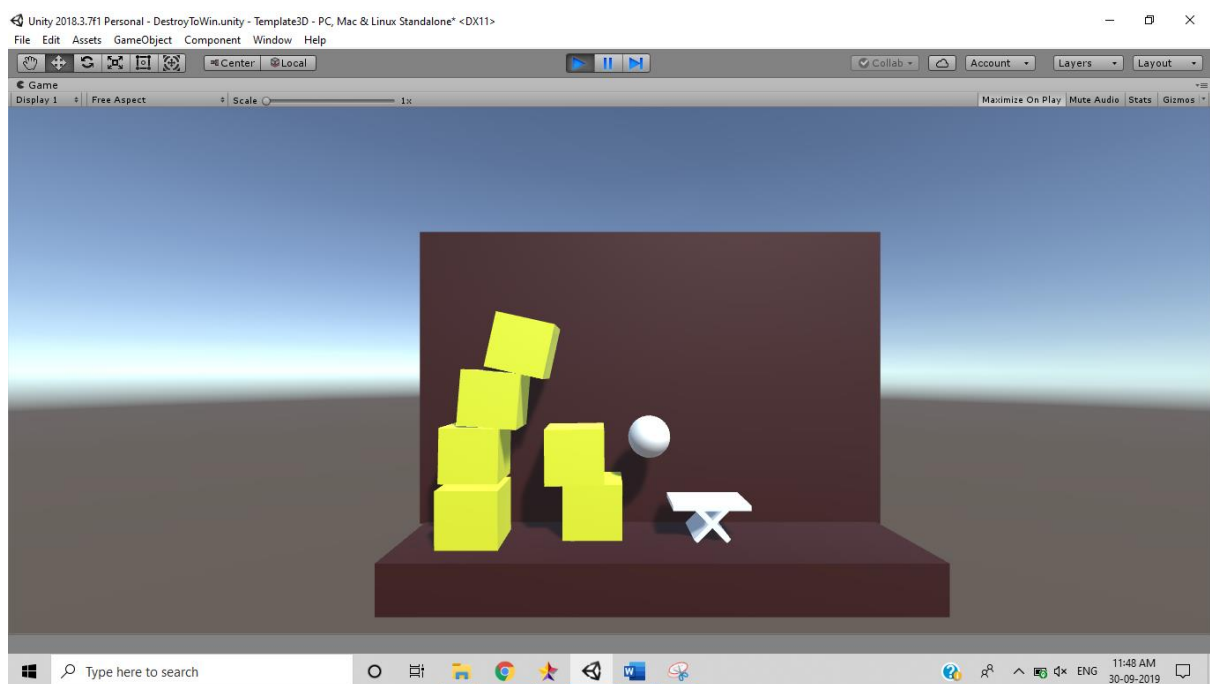


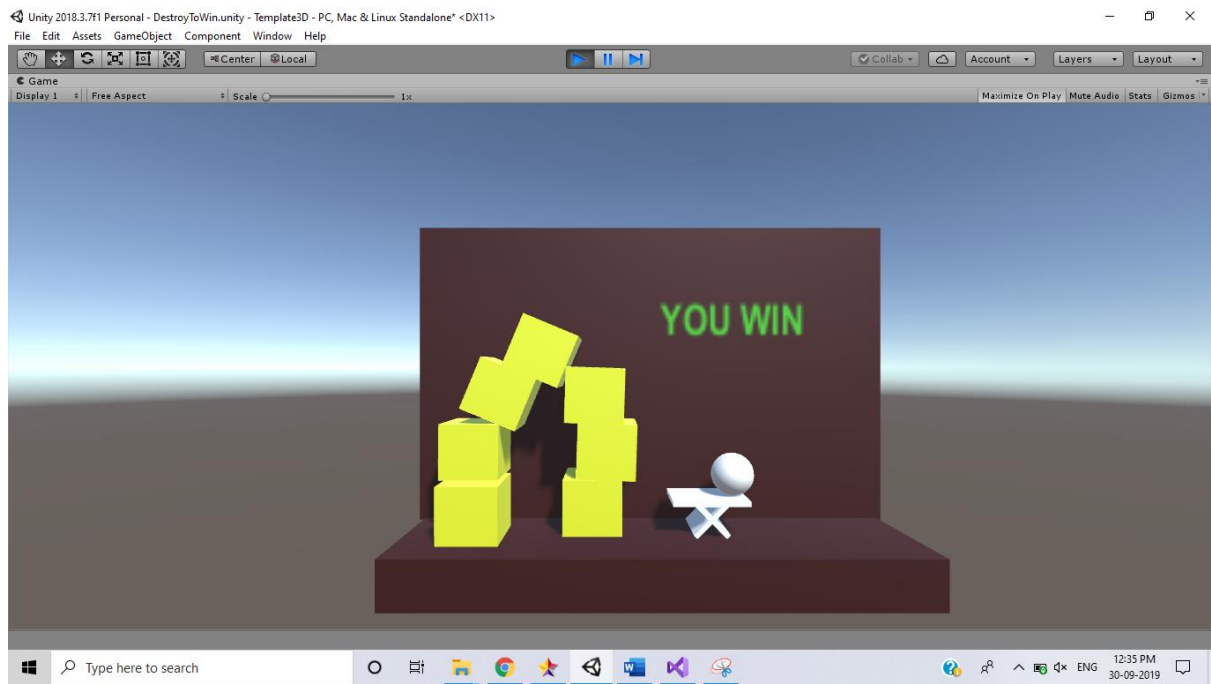
At last add script to prefab in prefab mode to destroy it.



The screenshot shows the Unity Inspector window with the 'DestroyOnClick.cs' script selected. The script is a C# class that inherits from 'MonoBehaviour'. It contains a private method 'OnMouseDown()' which calls 'Destroy(this.gameObject);' to destroy the object when it is clicked. The script is attached to a prefab, as indicated by the 'DestroyOnClick' icon in the top right corner of the Inspector.

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class DestroyOnClick : MonoBehaviour
6 {
7
8
9     private void OnMouseDown(){
10
11         Destroy(this.gameObject);
12
13     }
14
15
16 }
17
```



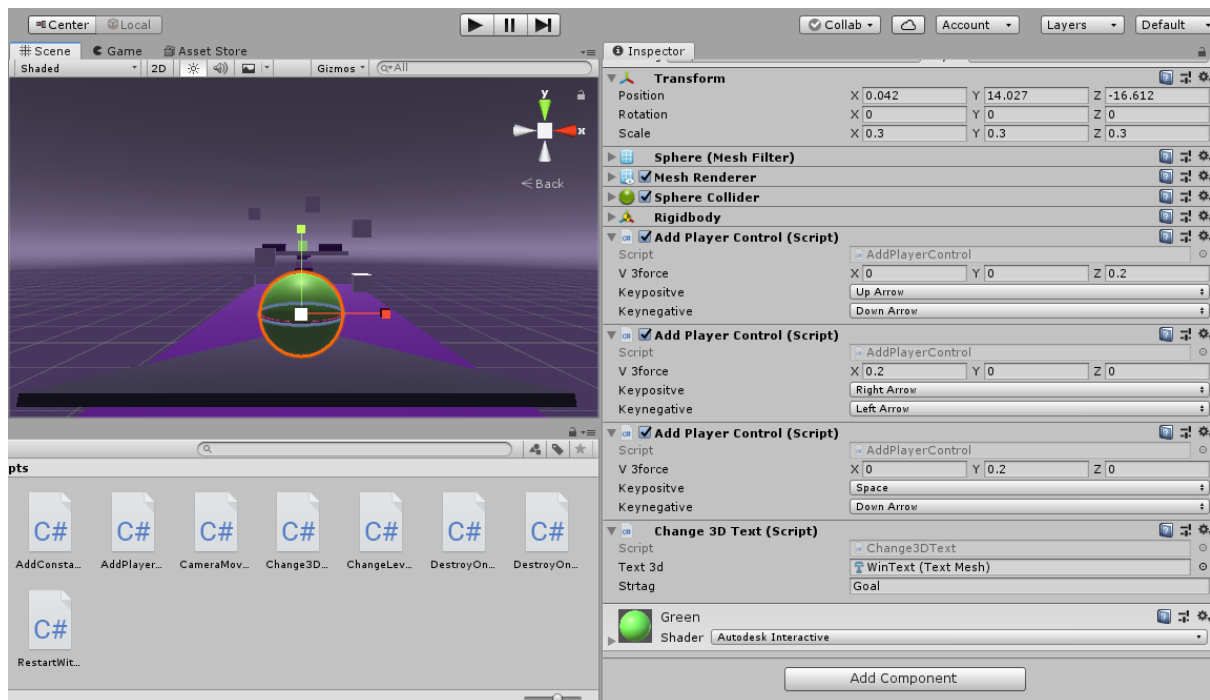


Practical-10

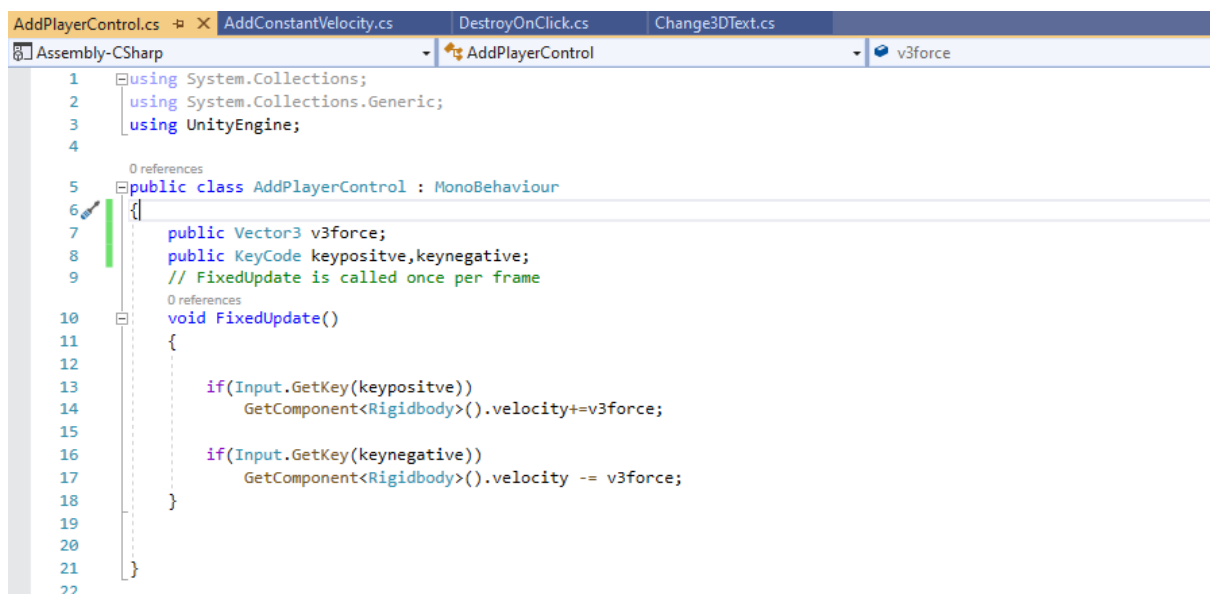
Creating a multiple and animation, handling multiple key events and restart scenes

It involves all the previously mentioned concept and some new scripts which are to be attached

Scripts attached to player(sphere) object are as follows



1. Add this script multiple (3 times) and give key values (keyboard values).

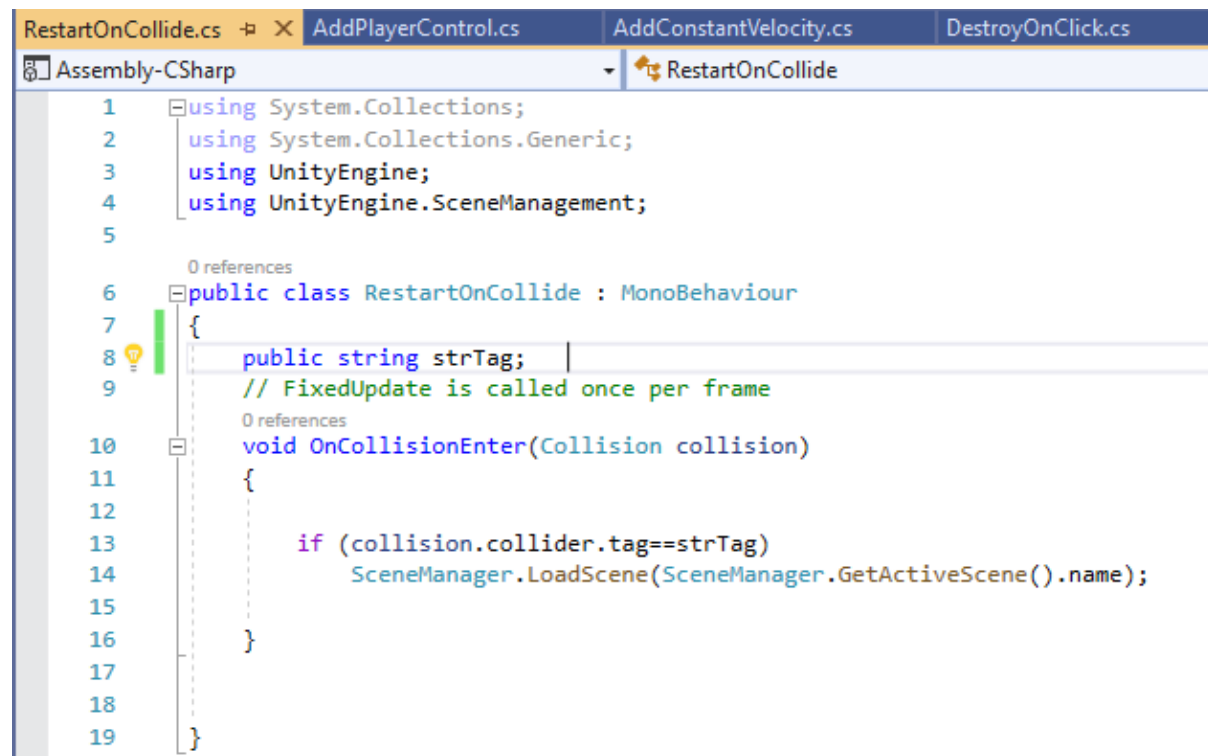


2. For finishing the game same as previous condition are executed.

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

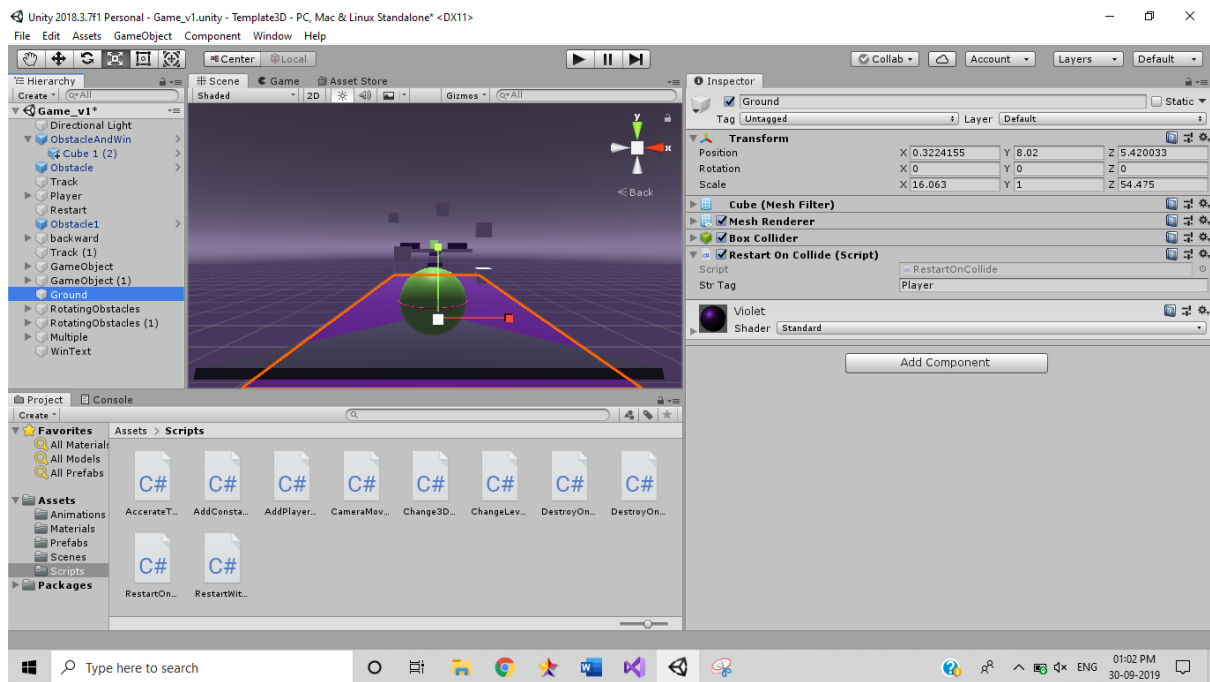
0 references
public class Change3DText : MonoBehaviour
{
    public TextMesh text3d;
    public string strtag;
    0 references
    private void OnCollisionEnter(Collision collision)
    {
        if (collision.collider.tag == strtag) {
            text3d.text = "YOU WIN";
        }
    }
}
```

Next we are attaching a script to ground so that when player hits it the game restarts. Remember the Player(or Sphere) needs a Player tag to achieve so.



```
RestartOnCollide.cs  AddPlayerControl.cs  AddConstantVelocity.cs  DestroyOnClick.cs
Assembly-CSharp  RestartOnCollide

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.SceneManagement;
5
0 references
6  public class RestartOnCollide : MonoBehaviour
7  {
8      public string strTag;
9      // FixedUpdate is called once per frame
0 references
10 void OnCollisionEnter(Collision collision)
11 {
12
13     if (collision.collider.tag==strTag)
14         SceneManager.LoadScene(SceneManager.GetActiveScene().name);
15
16 }
17
18
19 }
```



Next only animation along multiple game objects are doing their job for the game environment.



