CSE462

HW4

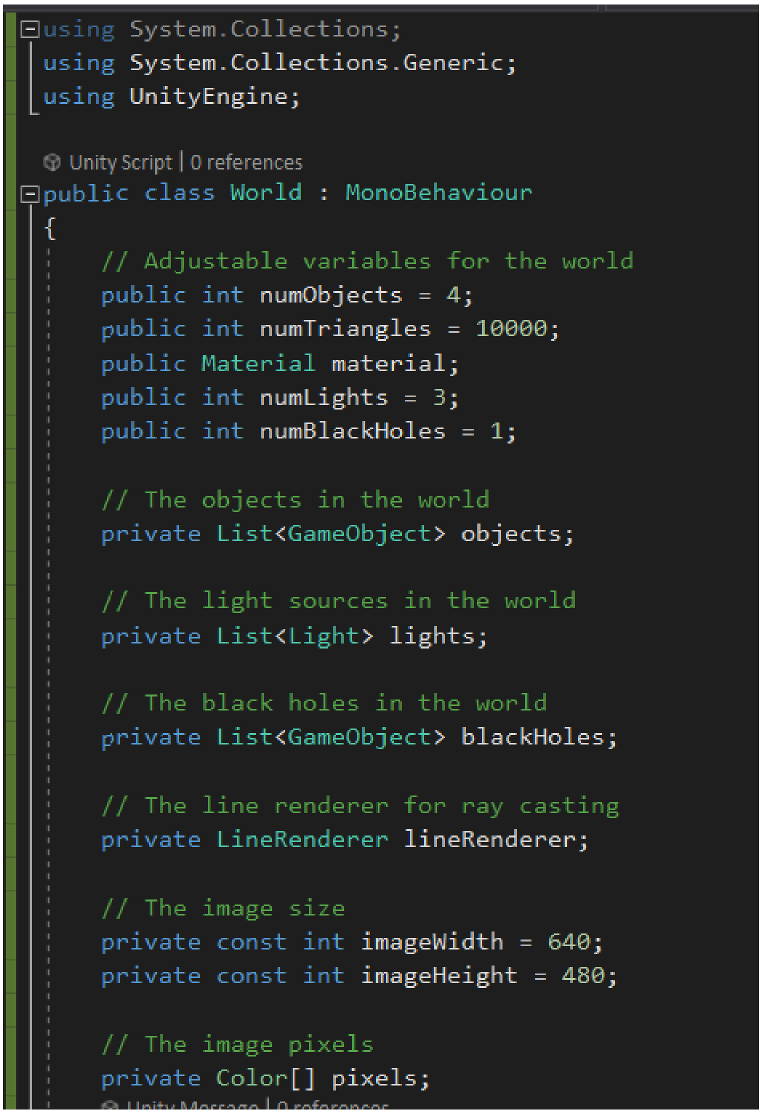
REPORT

1801042631

In this assignment, all operations are done in a script file named 'World.cs'.

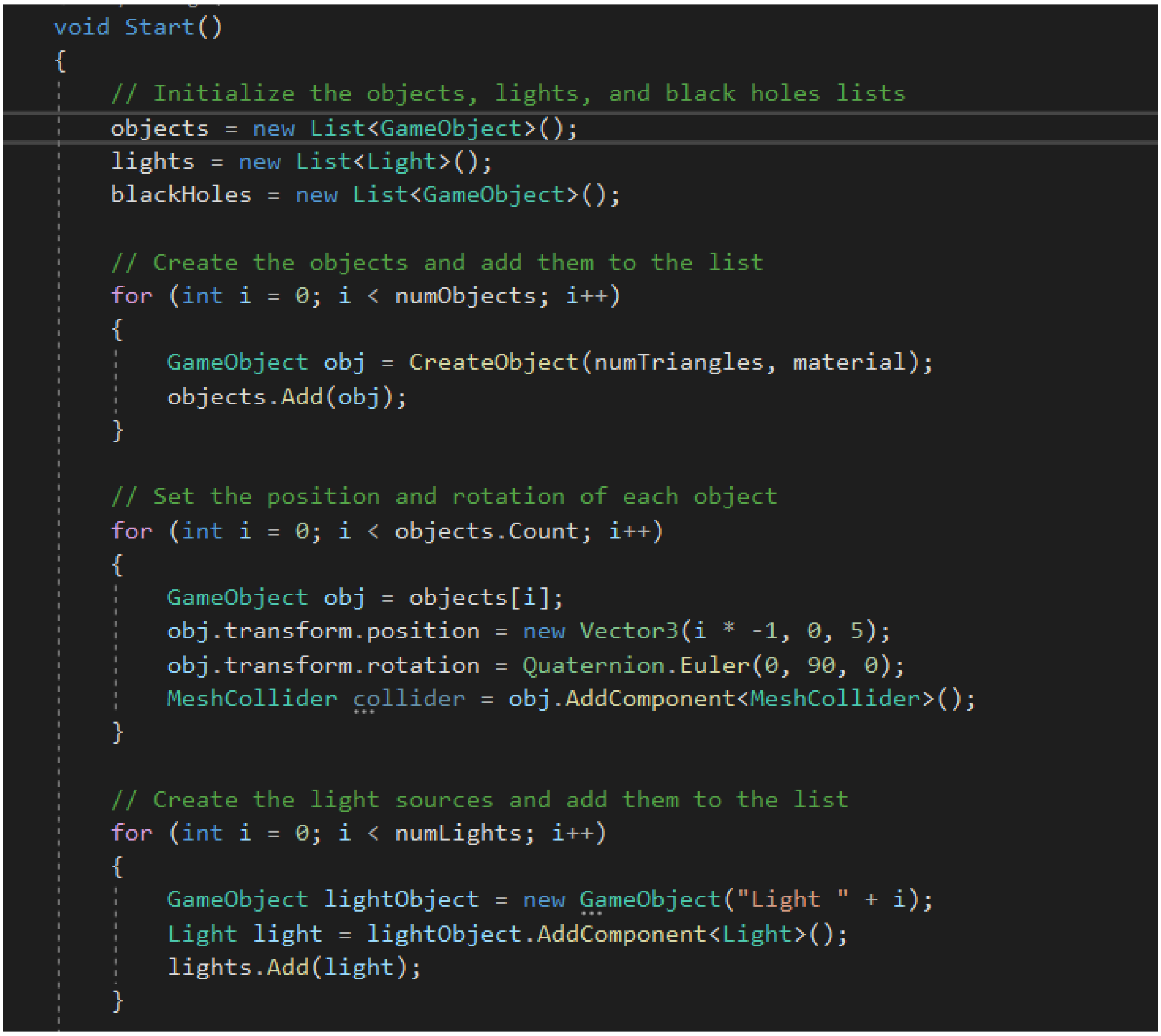
First, a unity 3D project is created. Then an empty object is created. Then put 'World.cs' in the empty object and run it.

World.cs content:



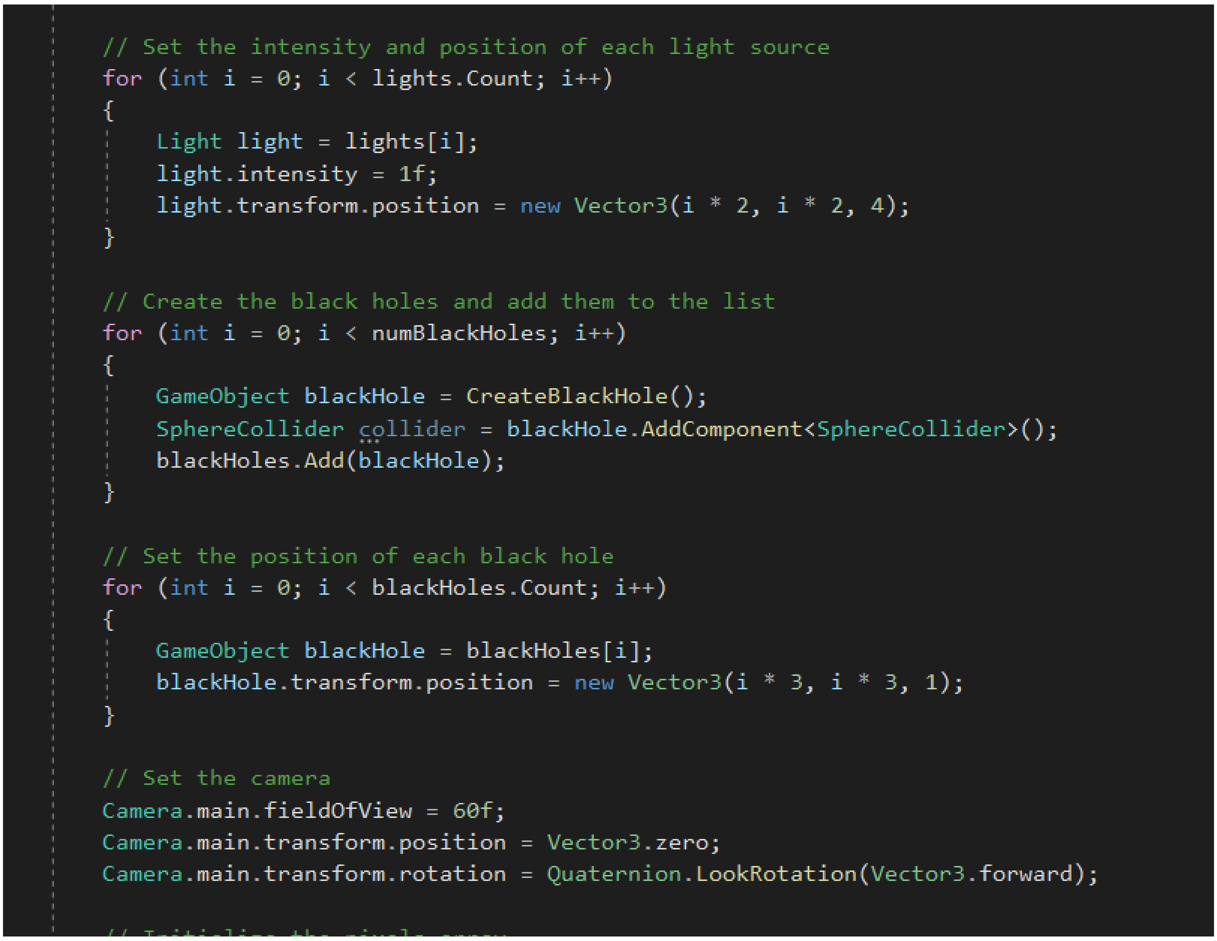
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Unity Script | 0 references
public class World : MonoBehaviour
{
    // Adjustable variables for the world
    public int numObjects = 4;
    public int numTriangles = 10000;
    public Material material;
    public int numLights = 3;
    public int numBlackHoles = 1;

    // The objects in the world
    private List<GameObject> objects;

    // The light sources in the world
    private List<Light> lights;

    // The black holes in the world
    private List<GameObject> blackHoles;

    // The line renderer for ray casting
    private LineRenderer lineRenderer;

    // The image size
    private const int imageWidth = 640;
    private const int imageHeight = 480;

    // The image pixels
    private Color[] pixels;
    Unity Message | 0 references
```

In the first part of the project, definitions were made. The list of objects, the list of lights, the list of black holes, and the necessary materials are defined. 4 objects, 3 lights and 1 black hole were used. Preparations were made for the ray casting operation.



```
    void Start()
    {
        // Initialize the objects, lights, and black holes lists
        objects = new List<GameObject>();
        lights = new List<Light>();
        blackHoles = new List<GameObject>();

        // Create the objects and add them to the list
        for (int i = 0; i < numObjects; i++)
        {
            GameObject obj = CreateObject(numTriangles, material);
            objects.Add(obj);
        }

        // Set the position and rotation of each object
        for (int i = 0; i < objects.Count; i++)
        {
            GameObject obj = objects[i];
            obj.transform.position = new Vector3(i * -1, 0, 5);
            obj.transform.rotation = Quaternion.Euler(0, 90, 0);
            MeshCollider collider = obj.AddComponent<MeshCollider>();
        }

        // Create the light sources and add them to the list
        for (int i = 0; i < numLights; i++)
        {
            GameObject lightObject = new GameObject("Light " + i);
            Light light = lightObject.AddComponent<Light>();
            lights.Add(light);
        }
```

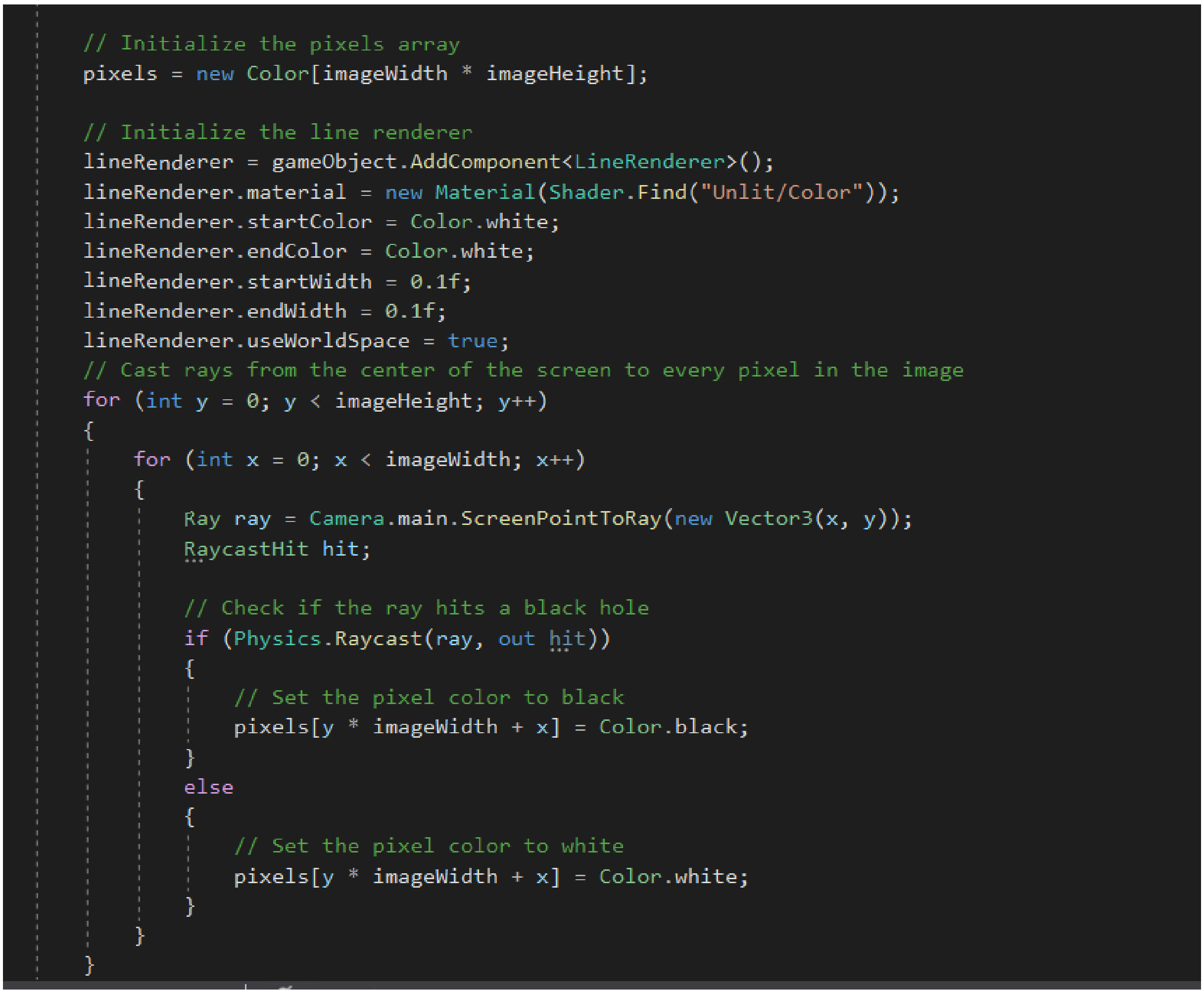
In the start function, objects and lights are created in a loop. The **'MeshCollider'** component is used so that the rays can hit the objects.



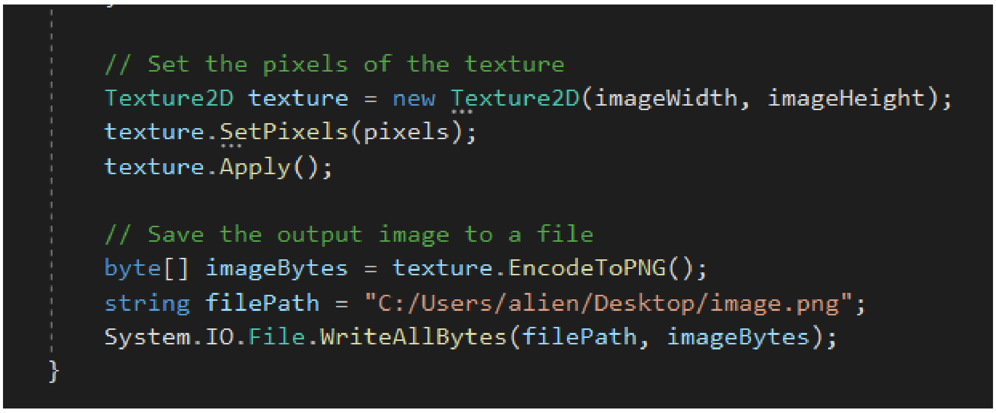
```
// Set the intensity and position of each light source
for (int i = 0; i < lights.Count; i++)
{
    Light light = lights[i];
    light.intensity = 1f;
    light.transform.position = new Vector3(i * 2, i * 2, 4);
}

// Create the black holes and add them to the list
for (int i = 0; i < numBlackHoles; i++)
{
    GameObject blackHole = CreateBlackHole();
    SphereCollider collider = blackHole.AddComponent<SphereCollider>();
    blackHoles.Add(blackHole);
}

// Set the position of each black hole
for (int i = 0; i < blackHoles.Count; i++)
{
    GameObject blackHole = blackHoles[i];
    blackHole.transform.position = new Vector3(i * 3, i * 3, 1);
}

// Set the camera
Camera.main.fieldOfView = 60f;
Camera.main.transform.position = Vector3.zero;
Camera.main.transform.rotation = Quaternion.LookRotation(Vector3.forward);

// Initialize the pixels array
```

Again, a black hole was created within the start function and the **'SphereCollider'** component was used to affect the light from the black hole.

Camera settings have been made.

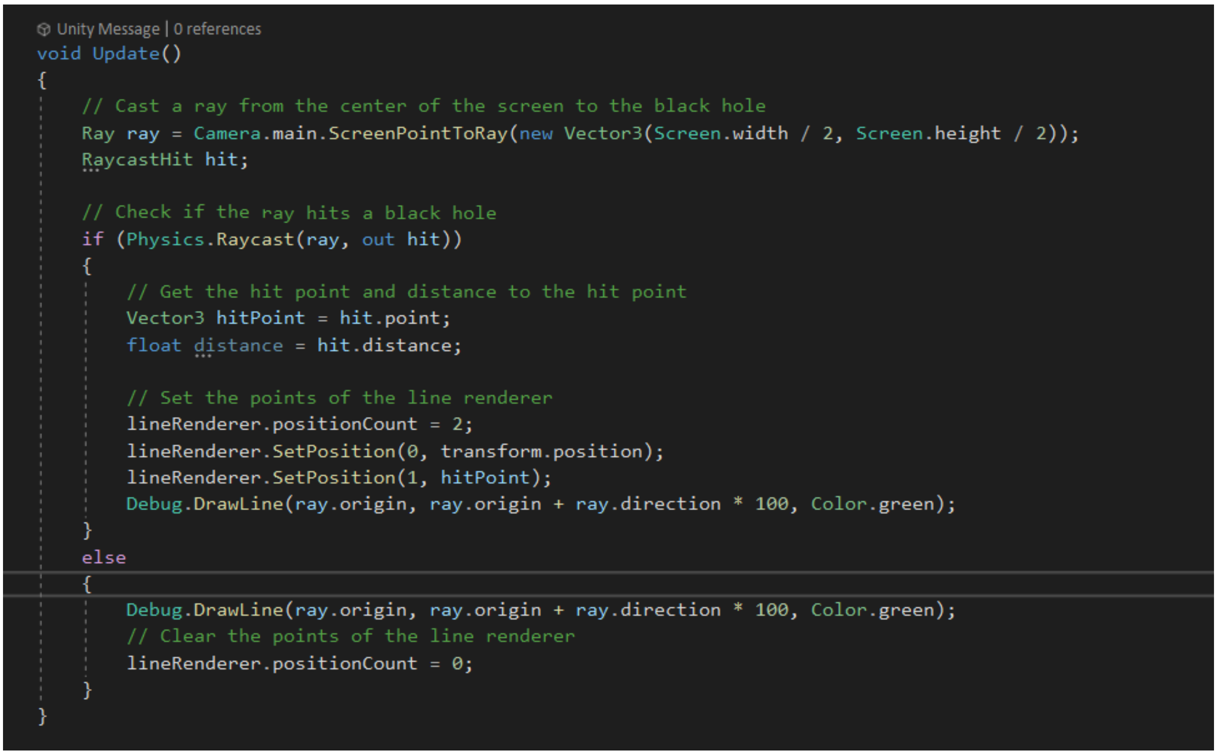


```
// Initialize the pixels array
pixels = new Color[imageWidth * imageHeight];

// Initialize the line renderer
lineRenderer = gameObject.AddComponent<LineRenderer>();
lineRenderer.material = new Material(Shader.Find("Unlit/Color"));
lineRenderer.startColor = Color.white;
lineRenderer.endColor = Color.white;
lineRenderer.startWidth = 0.1f;
lineRenderer.endWidth = 0.1f;
lineRenderer.useWorldSpace = true;
// Cast rays from the center of the screen to every pixel in the image
for (int y = 0; y < imageHeight; y++)
{
    for (int x = 0; x < imageWidth; x++)
    {
        Ray ray = Camera.main.ScreenPointToRay(new Vector3(x, y));
        RaycastHit hit;

        // Check if the ray hits a black hole
        if (Physics.Raycast(ray, out hit))
        {
            // Set the pixel color to black
            pixels[y * imageWidth + x] = Color.black;
        }
        else
        {
            // Set the pixel color to white
            pixels[y * imageWidth + x] = Color.white;
        }
    }
}
```



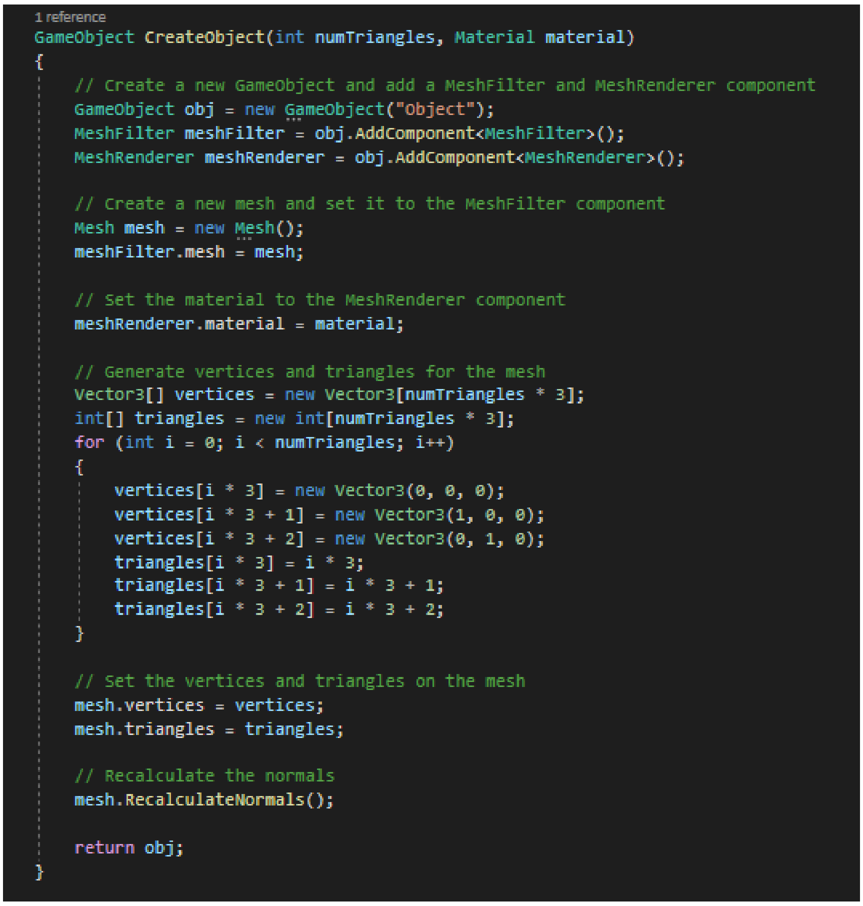
```
    // Set the pixels of the texture
    Texture2D texture = new Texture2D(imageWidth, imageHeight);
    texture.SetPixels(pixels);
    texture.Apply();

    // Save the output image to a file
    byte[] imageBytes = texture.EncodeToPNG();
    string filePath = "C:/Users/alien/Desktop/image.png";
    System.IO.File.WriteAllBytes(filePath, imageBytes);
}
```

Again, within the start function, 'LineRenderer' adjustments have been made to be used in Ray Casting operation. In addition, adjustments have been made to save the result as a png file.



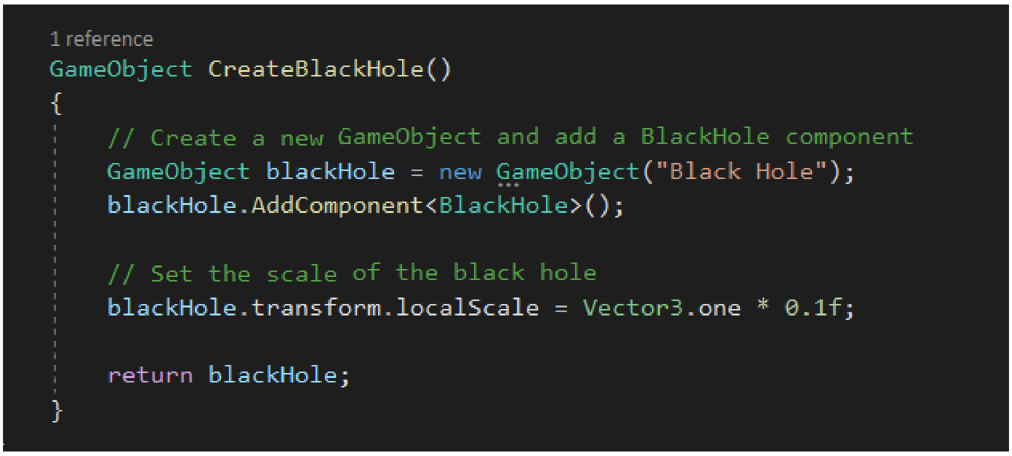
```
Unity Message | 0 references
void Update()
{
    // Cast a ray from the center of the screen to the black hole
    Ray ray = Camera.main.ScreenPointToRay(new Vector3(Screen.width / 2, Screen.height / 2));
    RaycastHit hit;

    // Check if the ray hits a black hole
    if (Physics.Raycast(ray, out hit))
    {
        // Get the hit point and distance to the hit point
        Vector3 hitPoint = hit.point;
        float distance = hit.distance;

        // Set the points of the line renderer
        lineRenderer.positionCount = 2;
        lineRenderer.SetPosition(0, transform.position);
        lineRenderer.SetPosition(1, hitPoint);
        Debug.DrawLine(ray.origin, ray.origin + ray.direction * 100, Color.green);
    }
    else
    {
        Debug.DrawLine(ray.origin, ray.origin + ray.direction * 100, Color.green);
        // Clear the points of the line renderer
        lineRenderer.positionCount = 0;
    }
}
```

Ray Casting operation is applied within the Update function. Two different cases of whether the rays hit an object or not were examined. By using the "Debug.drawline" method, the direction of the rays is drawn with a green line.



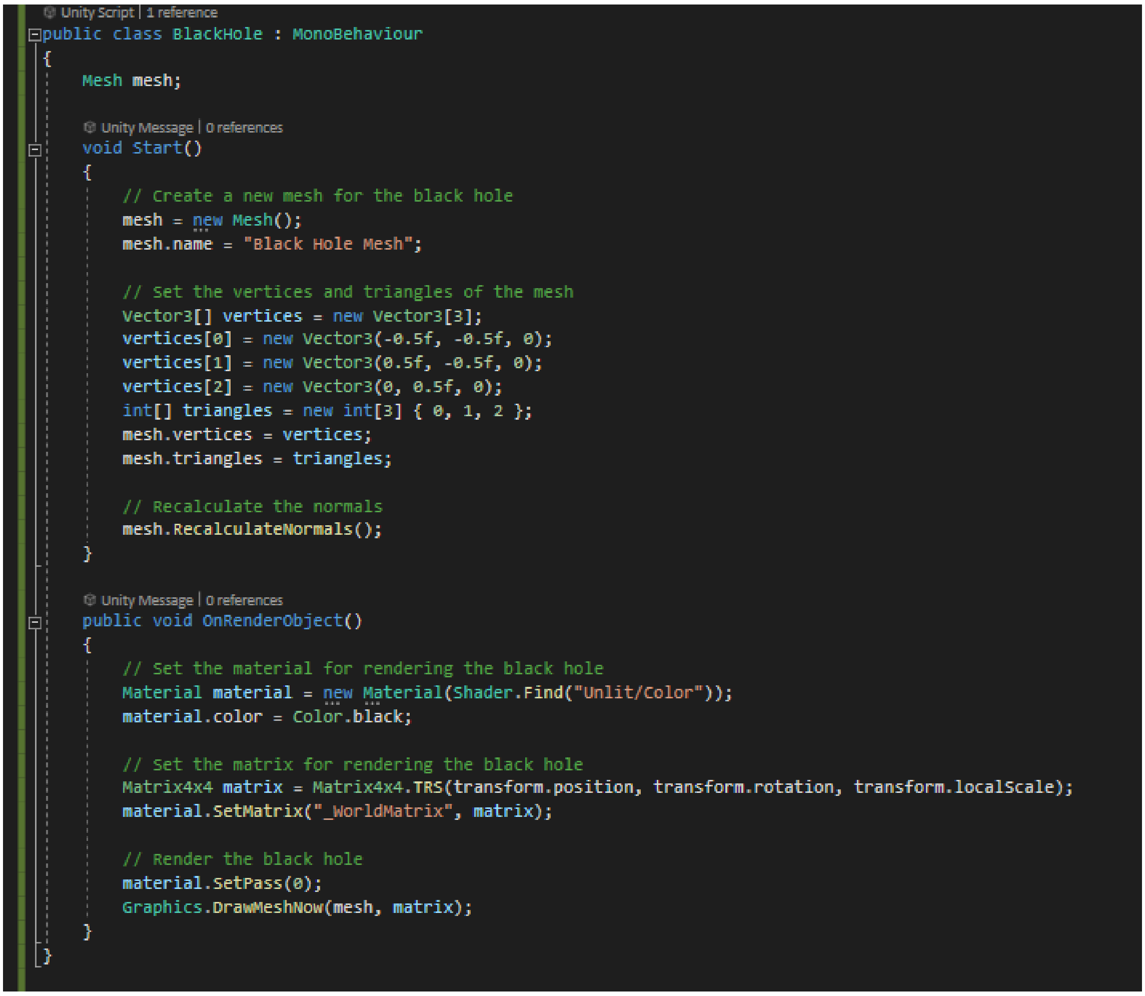
```
1 reference
GameObject CreateObject(int numTriangles, Material material)
{
    // Create a new GameObject and add a MeshFilter and MeshRenderer component
    GameObject obj = new GameObject("Object");
    MeshFilter meshFilter = obj.AddComponent<MeshFilter>();
    MeshRenderer meshRenderer = obj.AddComponent<MeshRenderer>();

    // Create a new mesh and set it to the MeshFilter component
    Mesh mesh = new Mesh();
    meshFilter.mesh = mesh;

    // Set the material to the MeshRenderer component
    meshRenderer.material = material;

    // Generate vertices and triangles for the mesh
    Vector3[] vertices = new Vector3[numTriangles * 3];
    int[] triangles = new int[numTriangles * 3];
    for (int i = 0; i < numTriangles; i++)
    {
        vertices[i * 3] = new Vector3(0, 0, 0);
        vertices[i * 3 + 1] = new Vector3(1, 0, 0);
        vertices[i * 3 + 2] = new Vector3(0, 1, 0);
        triangles[i * 3] = i * 3;
        triangles[i * 3 + 1] = i * 3 + 1;
        triangles[i * 3 + 2] = i * 3 + 2;
    }

    // Set the vertices and triangles on the mesh
    mesh.vertices = vertices;
    mesh.triangles = triangles;

    // Recalculate the normals
    mesh.RecalculateNormals();

    return obj;
}
```

CreateObject method. Thanks to this method, objects were created and materials were added.



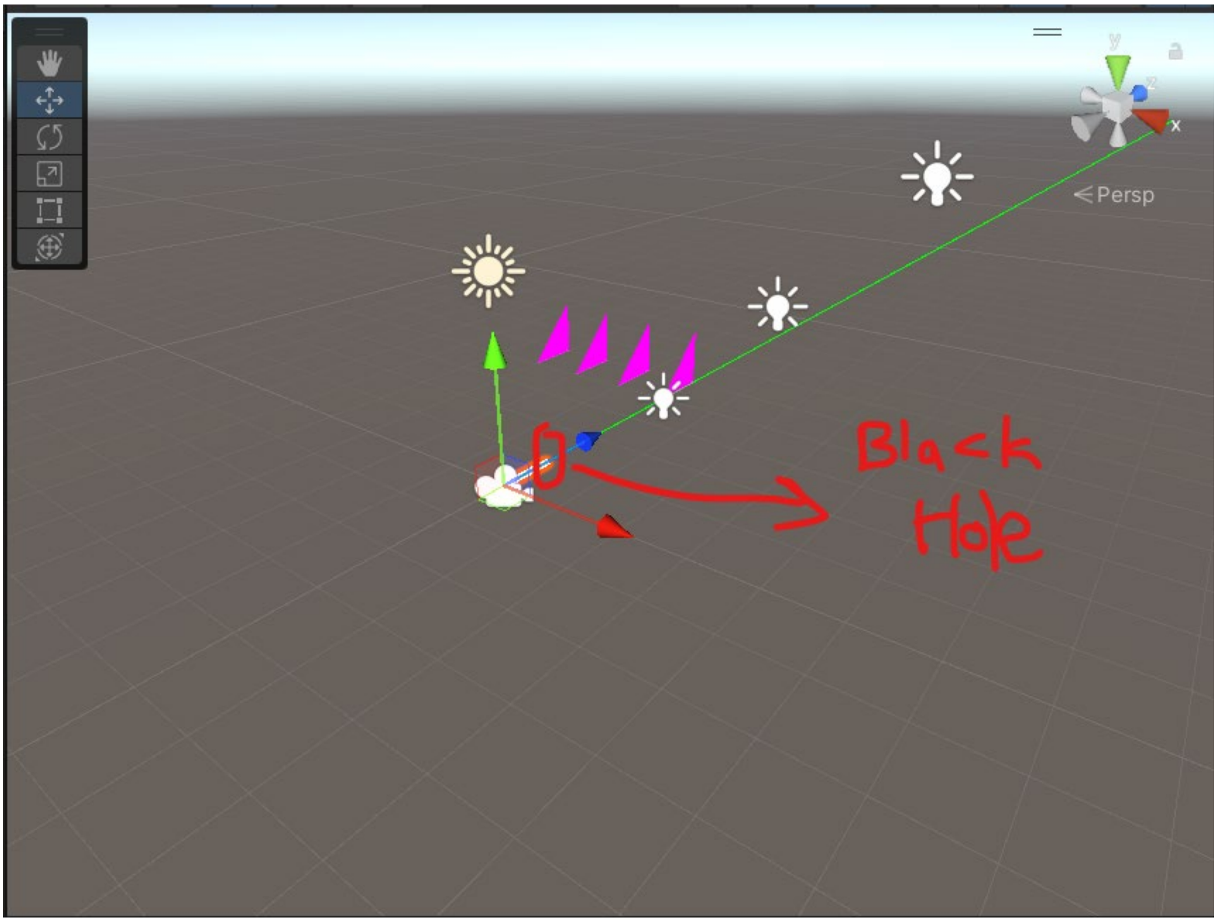
```
1 reference
GameObject CreateBlackHole()
{
    // Create a new GameObject and add a BlackHole component
    GameObject blackHole = new GameObject("Black Hole");
    blackHole.AddComponent<BlackHole>();

    // Set the scale of the black hole
    blackHole.transform.localScale = Vector3.one * 0.1f;

    return blackHole;
}
```

CreateBlackHole method. This method create blackhole

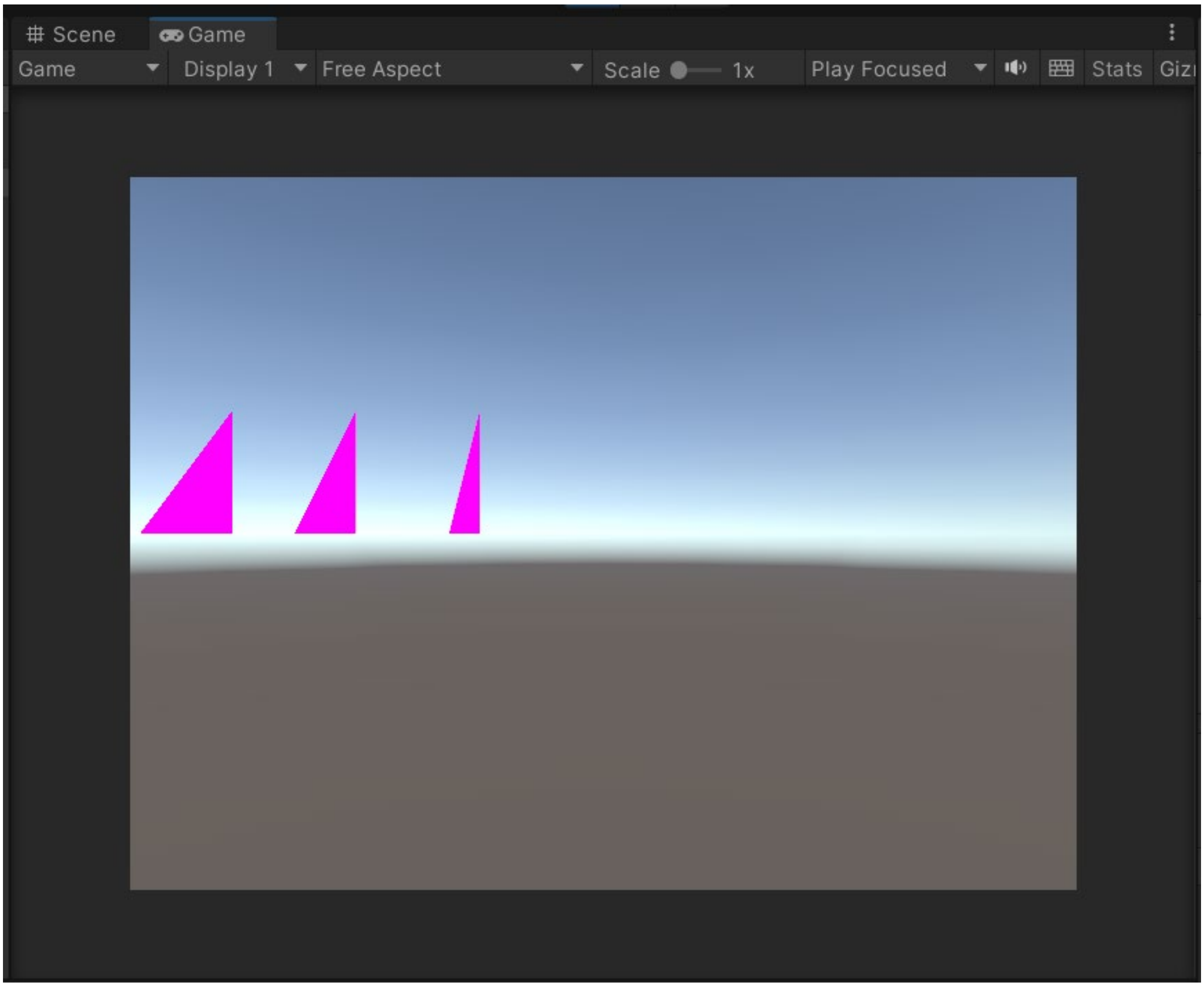


```
Unity Script | 1 reference
public class BlackHole : MonoBehaviour
{
    Mesh mesh;

    Unity Message | 0 references
    void Start()
    {
        // Create a new mesh for the black hole
        mesh = new Mesh();
        mesh.name = "Black Hole Mesh";

        // Set the vertices and triangles of the mesh
        Vector3[] vertices = new Vector3[3];
        vertices[0] = new Vector3(-0.5f, -0.5f, 0);
        vertices[1] = new Vector3(0.5f, -0.5f, 0);
        vertices[2] = new Vector3(0, 0.5f, 0);
        int[] triangles = new int[3] { 0, 1, 2 };
        mesh.vertices = vertices;
        mesh.triangles = triangles;

        // Recalculate the normals
        mesh.RecalculateNormals();
    }

    Unity Message | 0 references
    public void OnRenderObject()
    {
        // Set the material for rendering the black hole
        Material material = new Material(Shader.Find("Unlit/Color"));
        material.color = Color.black;

        // Set the matrix for rendering the black hole
        Matrix4x4 matrix = Matrix4x4.TRS(transform.position, transform.rotation, transform.localScale);
        material.SetMatrix("_WorldMatrix", matrix);

        // Render the black hole
        material.SetPass(0);
        Graphics.DrawMeshNow(mesh, matrix);
    }
}
```

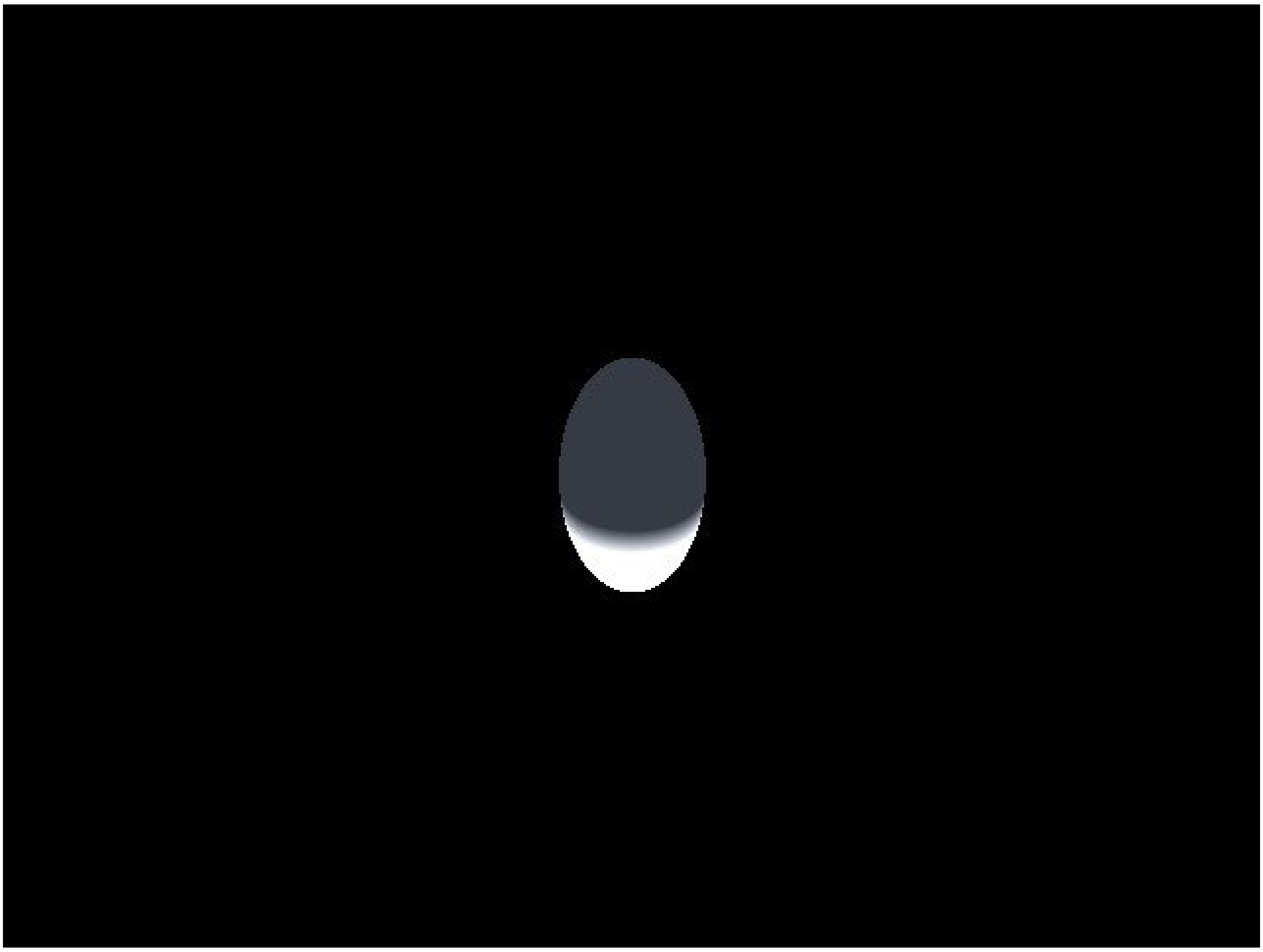
BlackHole class. Thanks to this class, black holes have been identified. The CreateBlackHole method made use of this class.



This is the world. There are 4 pink colored objects in the world. There is a black hole and 3 lights. The beam sent by Ray Casting is shown in green. The beam was cut by the black hole.



This is the camera screen.



Here is an example output png file. The image of the object is shadowed because of the black hole.

This is the Github link of the project:

https://github.com/SALTU-19/CSE462-Augmented-Reality-Homeworks-/tree/main/HW%204