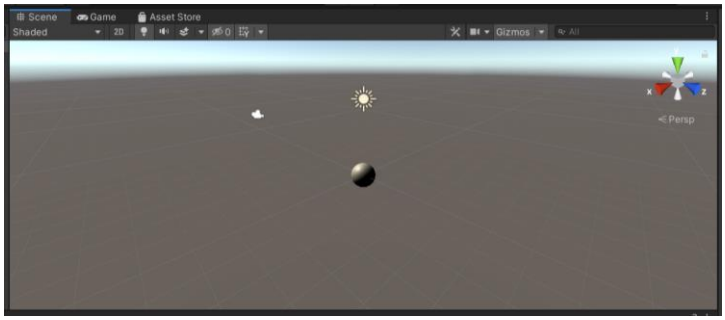
Advanced perception in Unity

Today you will create multiple raycasts at different angles from a single point, this will increase your capacity to enhance the sensory capabilities of gameobjects within the environment.

Step 1 – Create a new 3d file giving it a relevant name.

Step 2 – Create a sphere in the hierarchy.



Step 3 – Create a script in the asset manager called MultiPointSensor

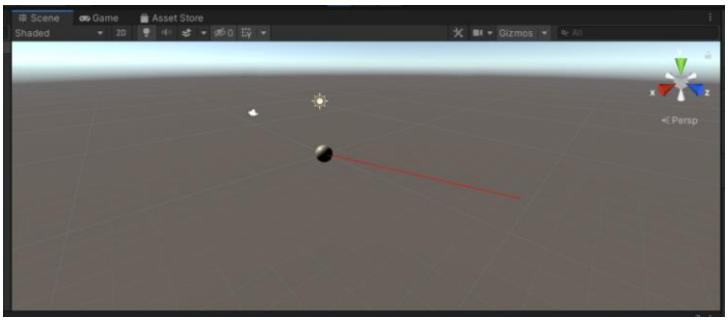
Step 4 – Add the script to the sphere you just created.

Step 5 – Enter the code below into the MultiPointSensor.cs file

```
void FixedUpdate()
{
    Vector3 noAngle = transform.TransformDirection(Vector3.forward);
    Quaternion spreadAngle = Quaternion.AngleAxis(-15, new Vector3(0, 1, 0));
    Vector3 newVector = spreadAngle * noAngle;

    Debug.DrawRay(transform.position, newVector * 10, Color.red);
}
```

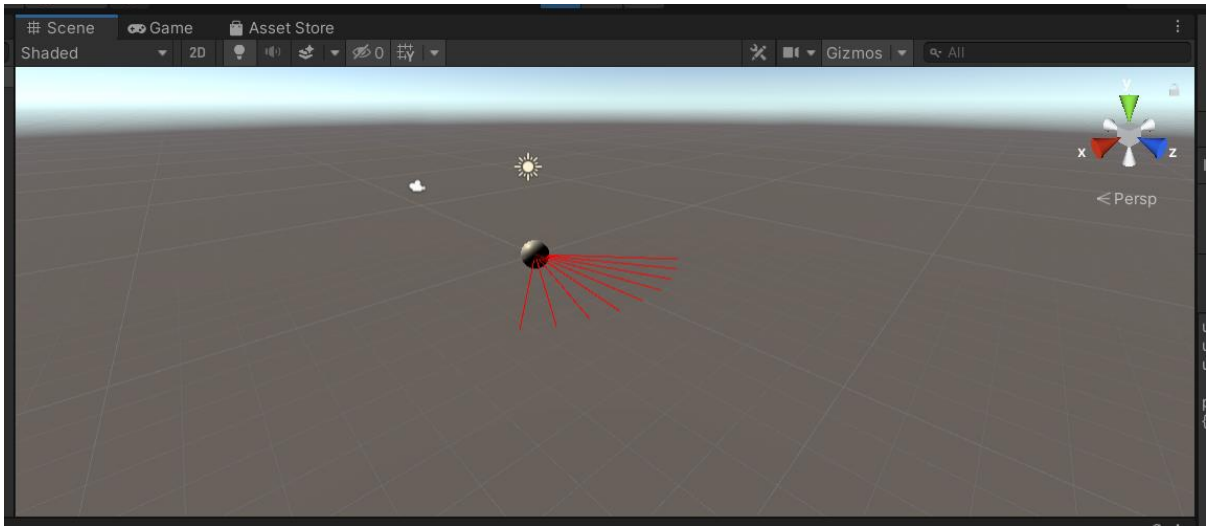
Step 6 – save and run the unity file, you should see a single red raycaset at an angle of 15 degrees.



Step 7 – Your main challenge for this session is to now create a scalable version of this by writing a new function and calling if from a for loop in the fixed update function. A hint is provided below along with the expected output.

```
void FixedUpdate()
{
    for (int i = 0; i < 5; i++)
    {
        CastRay(10 * i, 5f);
    }
}

1 reference
void CastRay(float angle, float distance)
{
    Vector3 noAngle = transform.TransformDirection(Vector3.forward);
    Quaternion RightSpreadAngle = Quaternion.AngleAxis(angle, new Vector3(0, 1, 0));
    Quaternion LeftSpreadAngle = Quaternion.AngleAxis(-angle, new Vector3(0, 1, 0));
    Vector3 newRightVector = RightSpreadAngle * noAngle;
    Vector3 newLeftVector = LeftSpreadAngle * noAngle;
    Debug.DrawRay(transform.position, newRightVector * distance, Color.red);
    Debug.DrawRay(transform.position, newLeftVector * distance, Color.red);
}
```



You can also use this strategy in your assignment to improve sensory capabilities.