1. Create a new Unity 3D project.
2. Right click in the hierarchy and follow 3D Object > Sphere to create a spherical object within the scene. Change the scale of the sphere to be 30 in the X, Y, and Z directions.
3. From the NASA Deep Star Map database, download a jpeg image of the celestial coordinate grid (or other desired map).
4. Right click in the project manager and click Import New Asset. Import the image and change the texture shape to “Cube.” Doing so will create a cubemap, which translates a flat image into a form that can be smoothly wrapped around a three-dimensional object.
5. Right click in the project manager to create a new folder named “Shaders.” Right click again within this folder and follow Shader > Standard Surface Shader. Name this new shader “CubemapShader,” and the script for this should read as follows:

```
Shader "Custom/NEWShader 1"
{
    Properties
    {
        _CubeMap( "Cube Map", Cube ) = "white" {}
    }

    SubShader
    {
        Pass
        {
            Tags { "DisableBatching" = "True" }

            Cull Front

            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag
            #include "UnityCG.cginc"

            samplerCUBE _CubeMap;

            struct v2f
            {
            float4 pos : SV_Position;
            half3 uv : TEXCOORD0;
            };

            v2f vert( appdata_img v )
            {
                v2f o;
                o.pos = UnityObjectToClipPos( v.vertex );
                o.uv = v.vertex.xyz * half3(1,1,1);
                return o;
            }

            fixed4 frag( v2f i ) : SV_Target
            {
                return texCUBE( _CubeMap, i.uv );
            }
            ENDCG
        }
    }
}
```

When applied to a material, this shader will allow the previously imported cubemap image to be wrapped around the selected object.

6. Right click in the project manager, follow Create > Folder, and name the folder “Materials.”

7. Within the Materials folder, right click and follow Create > Material. Click on the dropdown menu to the right of “Shader” and follow Custom > “CubemapShader” created above.
8. Click the “Select” button on the upper right hand corner of the material information, and choose the image that was previously imported as a cubemap.
9. Open the sphere created earlier. Under the Material section, click the circle icon to the right of “Element 0” and select the material created above. This will apply your imported cubemap image as a “wrap” to the sphere, which should now be visible in the scene view.
10. Right click in the project manager, follow Create > Folder, and title this folder “Scripts.” Within this folder, right click, follow Create > C# Script, and title this new script “RotateCamera.”
11. Create a new empty object in the hierarchy named “Focal Point” and drag the Main Camera onto it to designate it as a child object. Drag the RotateCamera script onto the focal point.
12. Change the position of the Main Camera to 0 in both the x and z directions, and 3 in the y direction.
13. The “RotateCamera” script should contain the following:

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

public class RotateCamera : MonoBehaviour
{

    private float x;
    private float y;
    private Vector3 rotateValue;

    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        y = Input.GetAxis("Mouse X");
        x = Input.GetAxis("Mouse Y");
        Debug.Log(x + ":" + y);
        rotateValue = new Vector3(x * -1, y * -1, 0);
        transform.eulerAngles -= rotateValue;
    }
}

```

This script, when applied to the Focal Point, will allow users to move their mouse (on computer) or slide to move (on touch screen device) in order to change the direction the camera is facing. This lets users “look around” the scene and choose the direction and angle of viewing.

14. Within the Scripts folder, create a new script titled “RotateSphere.” Drag this script onto the sphere in the hierarchy to attach it to the object. The RotateSphere script should contain the following:

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

public class RotateSphere : MonoBehaviour
{
    public GameObject sphere;
    public float speed;

    // Start is called before the first frame update
    void Start()
    {
        sphere.transform.position = new Vector3(0.0f, 0.0f, 0.0f);
    }

    // Update is called once per frame
    void Update()
    {
        sphere.transform.Rotate(0.0f, speed * Time.deltaTime, 0.0f);
    }
}

```

15. Select the sphere from the hierarchy and under the “Rotate Sphere (Script)” section, input the desired speed variable in the box next to “Speed” (in this case, 1). This speed variable will dictate how quickly the celestial sphere rotates around its axis, which runs through the North and South celestial poles.
16. Within the celestial sphere object, change the rotation in the X direction to 56 degrees. Doing so sets the initial position of the North celestial pole to an angle of 34 degrees (as it would be positioned in Athens, GA).
17. Create a plane by right clicking in the hierarchy and following 3D Object > Plane. This will act as the “ground” of the simulation and designate the horizon and visible night sky in any particular location.
18. Change the scale of the plane to 3 in the x, y, and z directions. This will allow it to extend beyond the edges of the sphere.

Note: This original simulation is a simplified approximation of the night sky throughout a calendar year in Athens, GA. The North celestial pole sits at an angle of 34 degrees above the horizon when the observer is facing North. The orientation of the celestial coordinate grid slowly rotates around this point at a rate of about 1 degree per day.