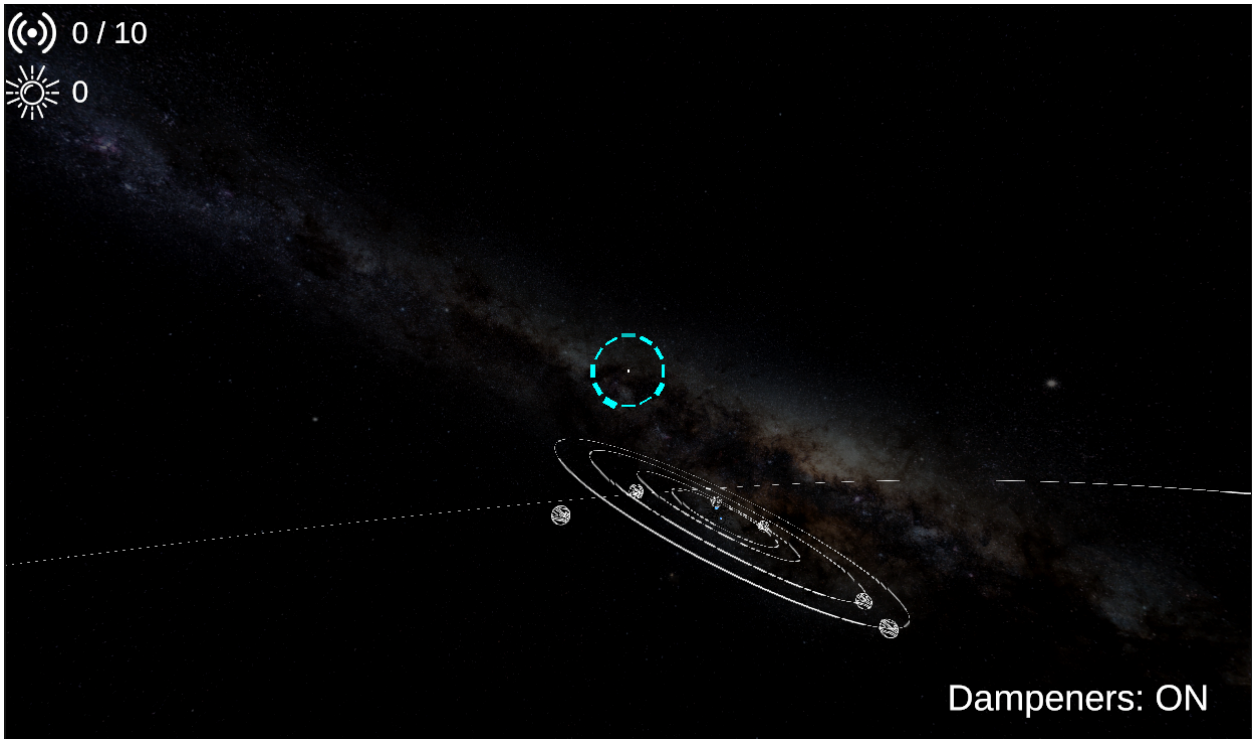
Hazel Litwiller

Sound Beacons in a Procedural Solar System

Sounds of Space

Sounds of Space is a procedurally generated space exploration game where the player searches for beacons that emit sound. The player spawns in their ship in a solar system where they are asked to collect all nearby beacons. Upon collecting these beacons, the player warps to a new procedurally generated system.



A procedurally generated system in *Sounds of Space.*

Beacon Design Goals

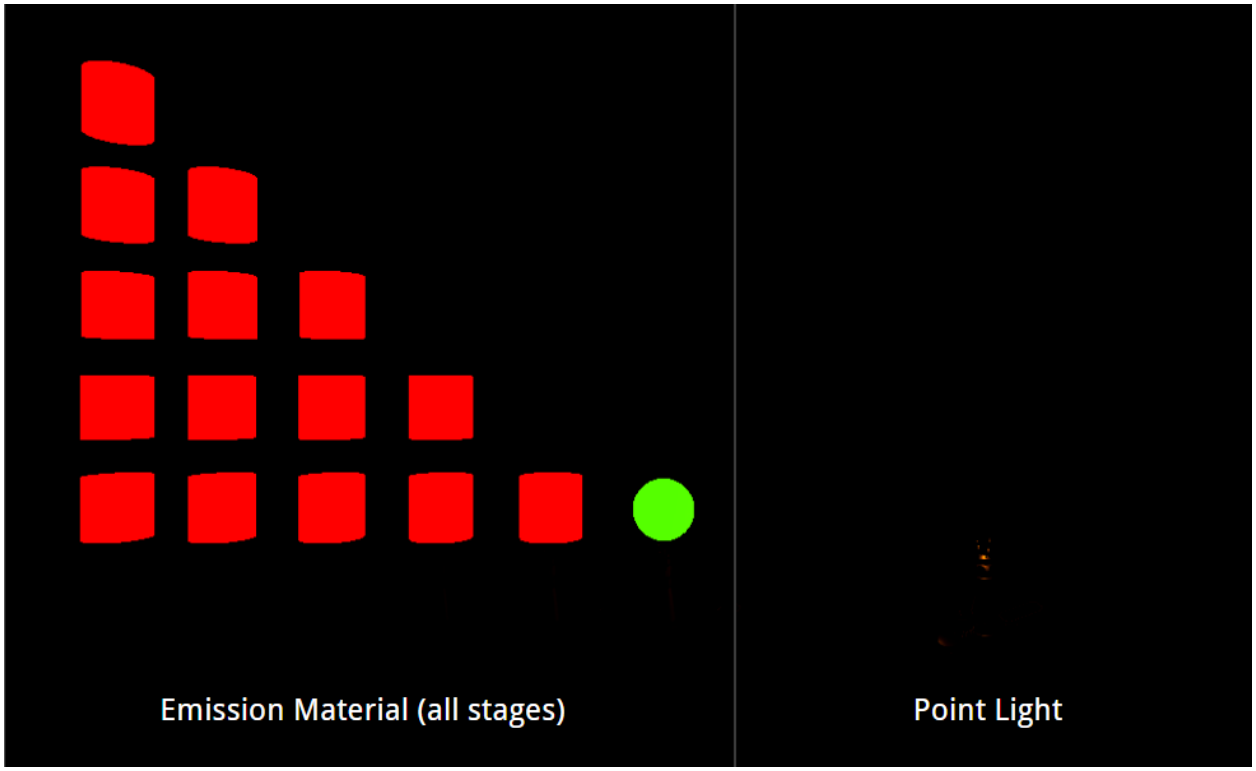
When designing the sound-emitting beacons that spawn in these systems the following principles were kept in mind:

1. A beacon must be visible from far away and in dark conditions
2. Beacon spawns must be dispersed throughout the entire system
3. The audio of beacons must not interfere with the audio of the ship
4. The player must be able to collect each beacon and receive clear audio/visual feedback when doing so

The beacon model used in *Sounds of Space.*

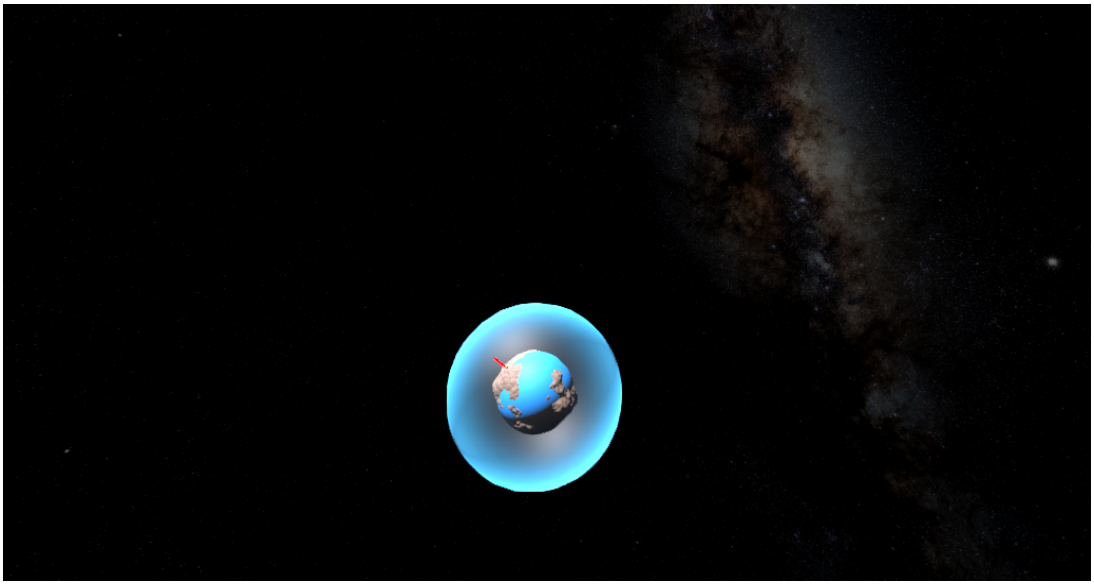
Beacon Illumination

Sounds of Space uses Unity's HDR pipeline for lighting and materials. This means there are two possible techniques to light up the beacon: a light source or an emissive material. Below is a beacon using a point light compared to one using an emissive material.



The emissive beacon's stages represent the keyframes of its animation.

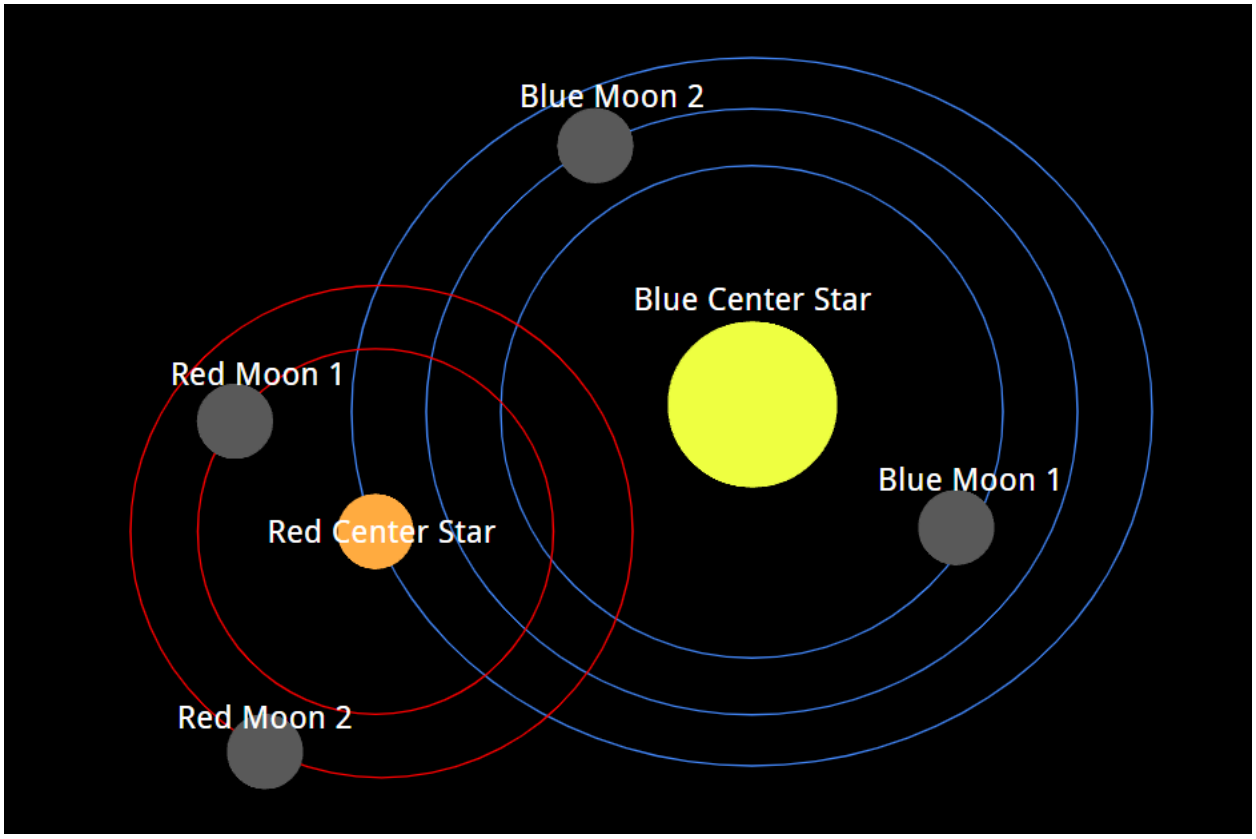
The emissive beacon is clearly more visible in a dark environment, at the slight cost of illuminating the beacon less. However, the beacon model will be lit by the light of the stars and the ship in the game, so this is not a problem we need to fix. As an added benefit, an emissive material will not become invisible at range as quickly as a point light will. To further improve the visibility, the number of red cylinders visible is animated to create a pulsing effect. When a beacon is collected, the cylinders are replaced by a green sphere.



A water planet with a beacon. The light blue outer sphere is its atmosphere.

Procedural Beacon Placement

As the planetary systems in *Sounds of Space* are procedurally generated, the beacons must also be procedurally placed. The procedural algorithm for system generation creates clusters of objects, typically a center star, or planet, and its moons, that may recursively have sub-clusters. To spread beacon spawns evenly across all generated bodies, these clusters are each considered individually. In *Sounds of Space*, a cluster of bodies can contain up to two beacons, with only one beacon allowed per body; and non-solid bodies, such as stars, cannot have beacons. This keeps the player engaged no matter where they explore, but does not overwhelm them with too many objects to interact with.



A simplified procedural system. This system contains two clusters (red and blue) that each have a center star surrounded by moons. The blue cluster is the parent of the red cluster, which means that both clusters will contain up to two beacons, generated independently of each other.

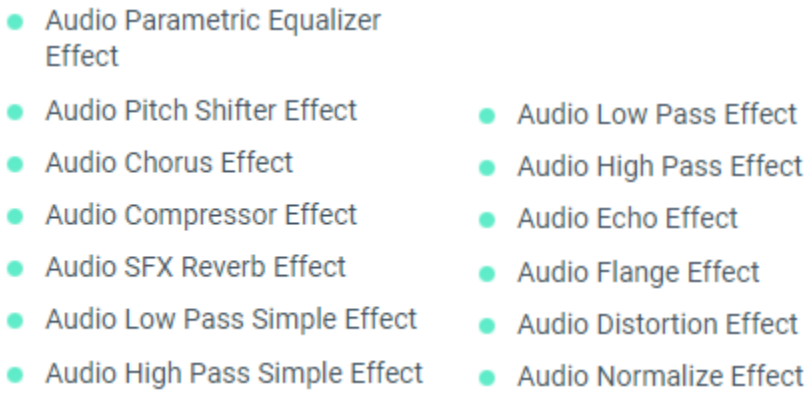
After a location for a beacon has been selected, we can use a raycast and some vector math to align the beacon's position and rotation with that of the planet. We also need to record the offset between the beacon's position and that of its planet to maintain that distance once the orbit simulation starts.

```
Vector3 spawnPoint = (UnityEngine.Random.onUnitSphere * body.Radius);
GameObject beacon = Instantiate(BeaconPrefab,spawnPoint,Quaternion.identity);
float beaconScale = body.Radius / 100;
beacon.transform.localScale = new Vector3(beaconScale,beaconScale,beaconScale);

Vector3 rayDirection = beacon.transform.position - body.transform.position;
RaycastHit hit;
if(Physics.Raycast(beacon.transform.position, rayDirection, out hit)){
    beacon.transform.position = hit.point;
}
beacon.transform.LookAt(body.transform.position);
beacon.transform.rotation *= Quaternion.Euler(180, 0, 0);

beacon.GetComponent<Beacon>().parent = body;
beacon.GetComponent<Beacon>().parentOffset = beacon.transform.position - body.transform.position;

if(uiManager != null) {
    beacon.GetComponent<Beacon>().marker = uiManager.CreateMarker(uiManager.beacon, this, target: beacon.transform);
}

body.hasBeacon = true;
currSpawnCount++;
```

This code places a beacon at *spawnPoint*, then snaps its position down to the surface of the planet and rotates it facing away from the center of the planet. After this, other beacon fields are initialized and the UI icon for the beacon is created.

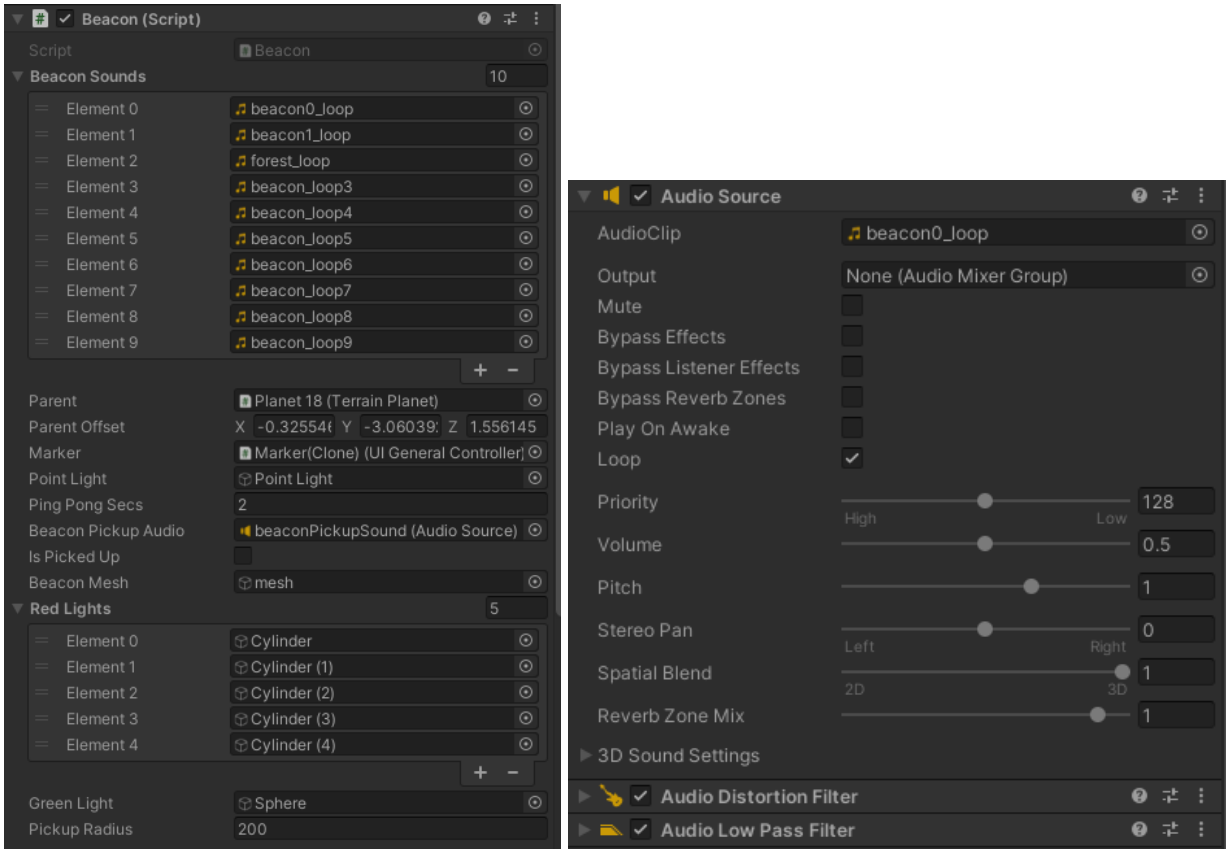
Beacon Audio

Sounds of Space uses FMOD for Unity to handle dynamic audio effects for the player's ship, so our first instinct was to use FMOD for the sound emitted by beacons as well. However, there were a few things to consider before making this decision. First, FMOD's audio pool in this project was already overloaded from the large number of ship audio events running at any given time. If FMOD's audio buffer becomes overloaded, it can cause popping and clicking sounds to occur, and this already happened on rare occasions with just the ship audio. Additionally, the beacon audio doesn't need to be as dynamic as the ship audio; in fact Unity's built in audio already provides all of the sound filters we need to implement beacon audio.



Unity's built in Audio Filter Components.

Before we add an audio file to our beacon, we need to find an audio clip and prepare it. All of the audio clips for this project were obtained from FreeSound.org and ProSoundEffects.com. After obtaining an audio file, it would be imported in Adobe Audition where the sound would be cut into a loop and normalized. Lastly the audio loop would be imported into Unity and added to a list of potential beacon audio files. When instantiated, each beacon selects one of these loops at random and sets it as the current clip in its Audio Source component.

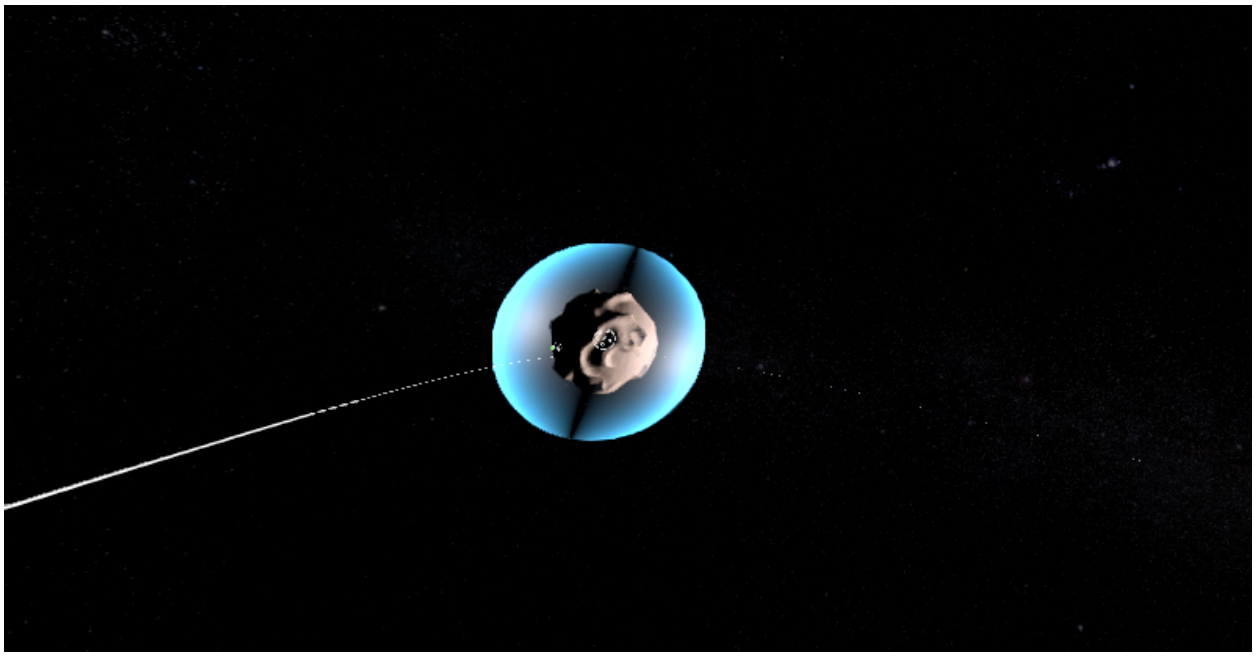


The components of a Beacon prefab.

There is still one important sound-related aspect of a Beacon to implement: atmospheric dampening. In *Sounds of Space*, non-gaseous bodies are surrounded by an atmosphere, which means a beacon will always spawn within its planet's atmosphere, and the audio should be modified appropriately. To simulate a sound being dampened by an atmosphere, each beacon has a low pass filter, using a logarithmic curve at 1118 Hz with Q=1, and a slight distortion effect. If the player flies inside an atmosphere, any beacons also inside will have their effects removed.

Beacon UX

When in proximity to a beacon, the player can click on it to pick it up, which triggers a sound effect and increments a counter on the player's UI. Then the red animating emission cylinder's of the beacon are replaced by a green sphere and the beacon's sound fades out. As with other objects in *Sounds of Space*, beacons also have their own UI markers on screen. If the player scans a planet by pressing Tab, that planet's beacon will be marked, ensuring unfavorable lightning conditions do not conceal the base of a beacon.



A moon and a beacon that have been marked by the player. The long white line denotes the orbit of the moon.

Challenges and Experience

The main challenge of implementing this feature was integrating it with the existing procedural generation of the bodies and their atmosphere. The unpredictable nature of procedural generation made clear the importance of gaining a solid understanding of any features that may interact with a new feature you plan to create. This feature also provided ample experience balancing the use of two audio engines in a single project, and avoiding creating conflicting audio interactions.