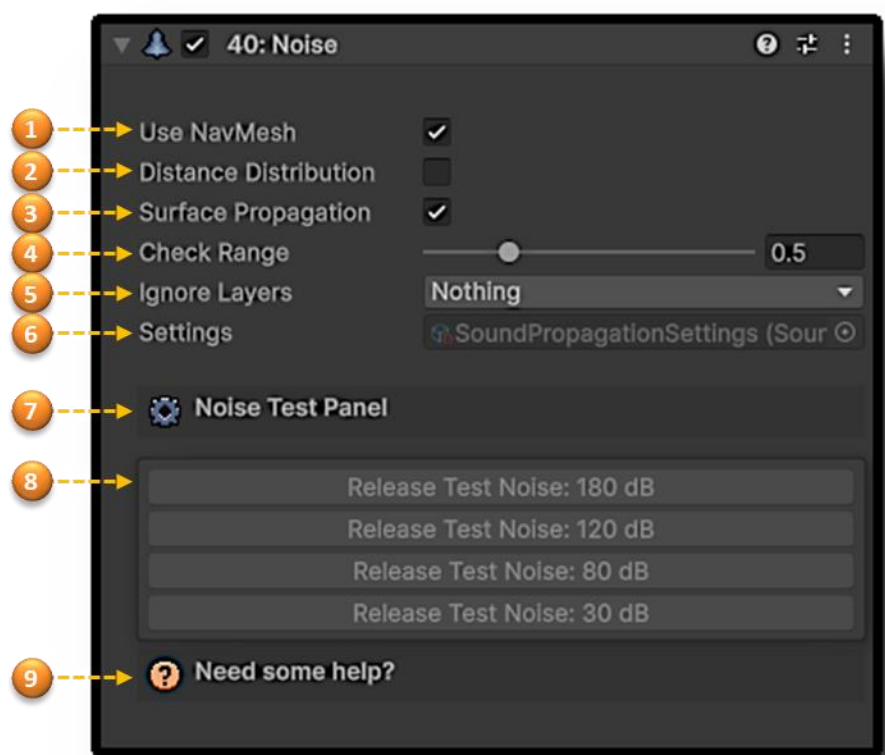


NOISE COMPONENT

The Noise Component allows an actor to emit gameplay noise signals that can be detected by any Senses Component equipped with the Hear Sense. Whenever an actor performs an action that should produce audible feedback—such as footsteps, weapon fire, object impacts, or scripted interactions—the Noise Component converts that action into a measurable noise value and distributes it across the environment, making it available to all eligible listeners. Noise values are influenced by distance, surface type, and optional NavMesh-based pathing, creating a propagation system that behaves naturally and consistently with gameplay expectations. Whether triggered through animation events or from custom scripts, the Noise Component gives designers full control over how audible actions affect AI behavior.

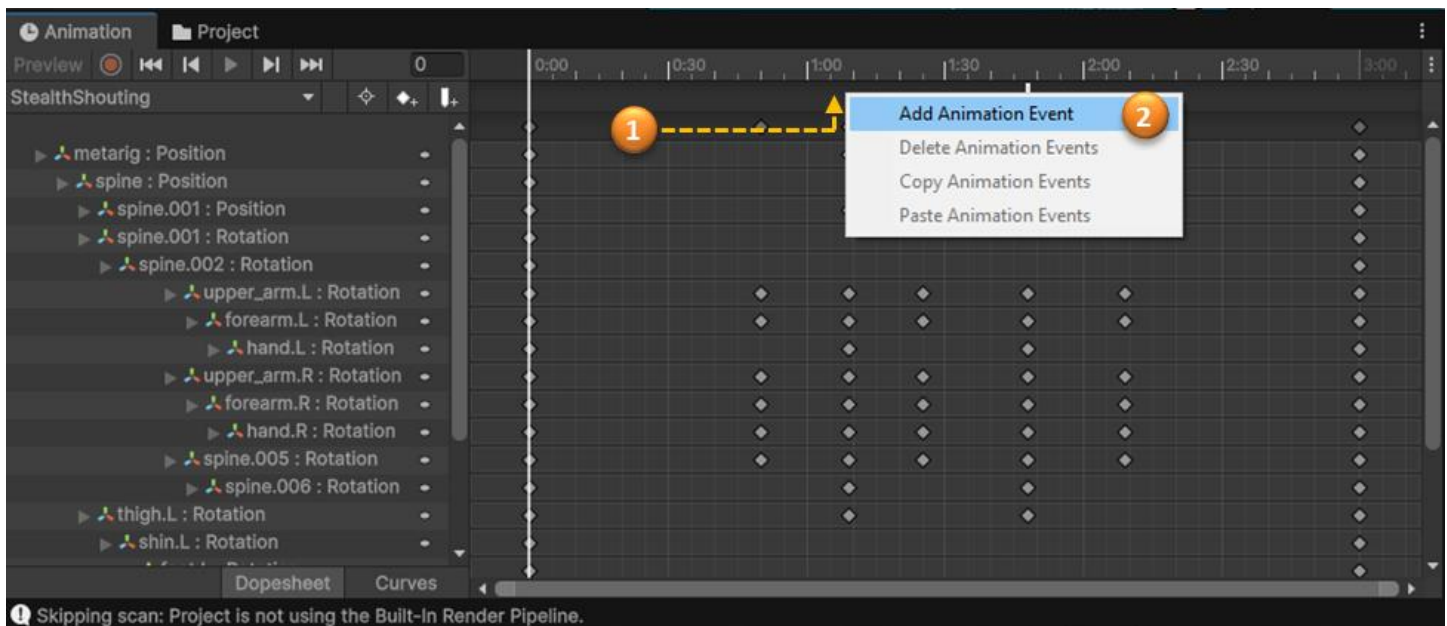
Noise Component Inspector Tab



1	Use NavMesh	When enabled, the distance to a Senses Component with a Hear Sensor will be calculated using the NavMesh. If no valid path is found, the distance will be calculated in a straight line.
2	Distance Distribution	When enabled, all Senses Components will receive emitted noise in order of distance (physically accurate solution). When disabled, all Senses Components will receive emitted noise in sphere cast order (performance-efficient solution).

3	Surface Propagation	When enabled, the noise value will be multiplied based on the type of surface below the agent.
4	Check Range	Allows configuring the collider check range that affects sound propagation. If multiple elements are detected, sound propagation will be influenced by the element with the highest Y position.
5	Ignore Layers	Allows setting up ignored layers.
6	Settings	Allows setting up sound propagation settings, TAGs and Factors.
7	Noise Test Panel	Shows or hides the Noise Test Panel used for quick debugging.
8	Release Noise Button	Emit predefined noise values for easy testing against Hear Sense-enabled actors.
9	'Need some help?'	Displays Help Information.

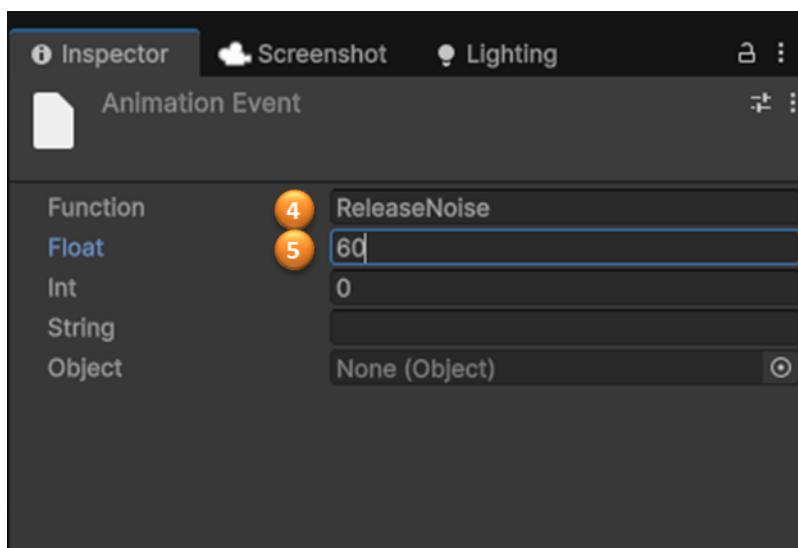
Integration - Releasing Noise with Animation Events



Animation Events provide a simple and intuitive way to trigger noise directly from an animation clip. This is especially useful for actions such as footsteps, body impacts, shouts, or any movement where the timing of the noise naturally matches the animation. To add a noise event, open the animation in Unity's Animation window and place an event on the exact frame where the action occurs—for example, when a foot makes contact with the ground.



To create your first noise event, right-click on the top bar just below the time ruler (Step 1) and choose Add Animation Event (Step 2). A new event marker will appear on the timeline. Select this marker (Step 3) to show its details in the Inspector. In the Inspector, set the Function field to ReleaseNoise (Step 4), then enter the desired Noise Value into the Float field (Step 5).



Once assigned, the animation will automatically call `ReleaseNoise()` every time it reaches that frame, generating a noise signal with the exact intensity you specified. This gives designers precise control over when noise is produced and how strong it should be, making it ideal for footsteps, attack swings, landing impacts, and other animation-driven actions that need reliable timing.

Integration - Releasing Noise with Scripts

Not every noisy action comes from an animation. Weapons firing, doors slamming, objects falling, or scripted interactions often happen entirely through gameplay logic. In these cases, your script can generate a noise signal by calling:

```
ReleaseNoise(float noiseValue)
```

This does not play any actual audio. Instead, it informs AI equipped with the Hear Sense that “a noise event occurred here”, and they react based on their hearing settings, distance to the source, obstacles, NavMesh pathing, and any surface-based modifiers.

To use it, your script simply stores a reference to the Noise Component—usually in `Awake()`—and calls `ReleaseNoise()` at the right moment.

```
public class Gun : MonoBehaviour
{
    private Noise m_noise;

    void Awake()
    {
        m_noise = GetComponent<Noise>();
    }

    public void Shoot()
    {
        // Fire weapon / spawn projectile / play audio separately...

        // Send a loud gameplay noise signal
        m_noise.ReleaseNoise(120f);
    }
}
```

Noise Value

Noise Value defines how strong a noise signal is within the Senses system. Every noise event generated by an actor—whether from footsteps, weapon fire, impacts, or scripted interactions—uses a single numeric value that represents its intensity. This value behaves like a simplified decibel (dB) scale: higher numbers travel farther and are detected more easily by Hear Sensors, while lower numbers remain subtle and limited in range.

The Noise Component does not simulate real-world audio physics; instead, it uses a designer-friendly scale that is intuitive, predictable, and flexible for gameplay. A Noise Value determines how far the

signal spreads before becoming too weak for AI to detect, with additional adjustments applied based on distance, NavMesh path length, and optional surface-based propagation.

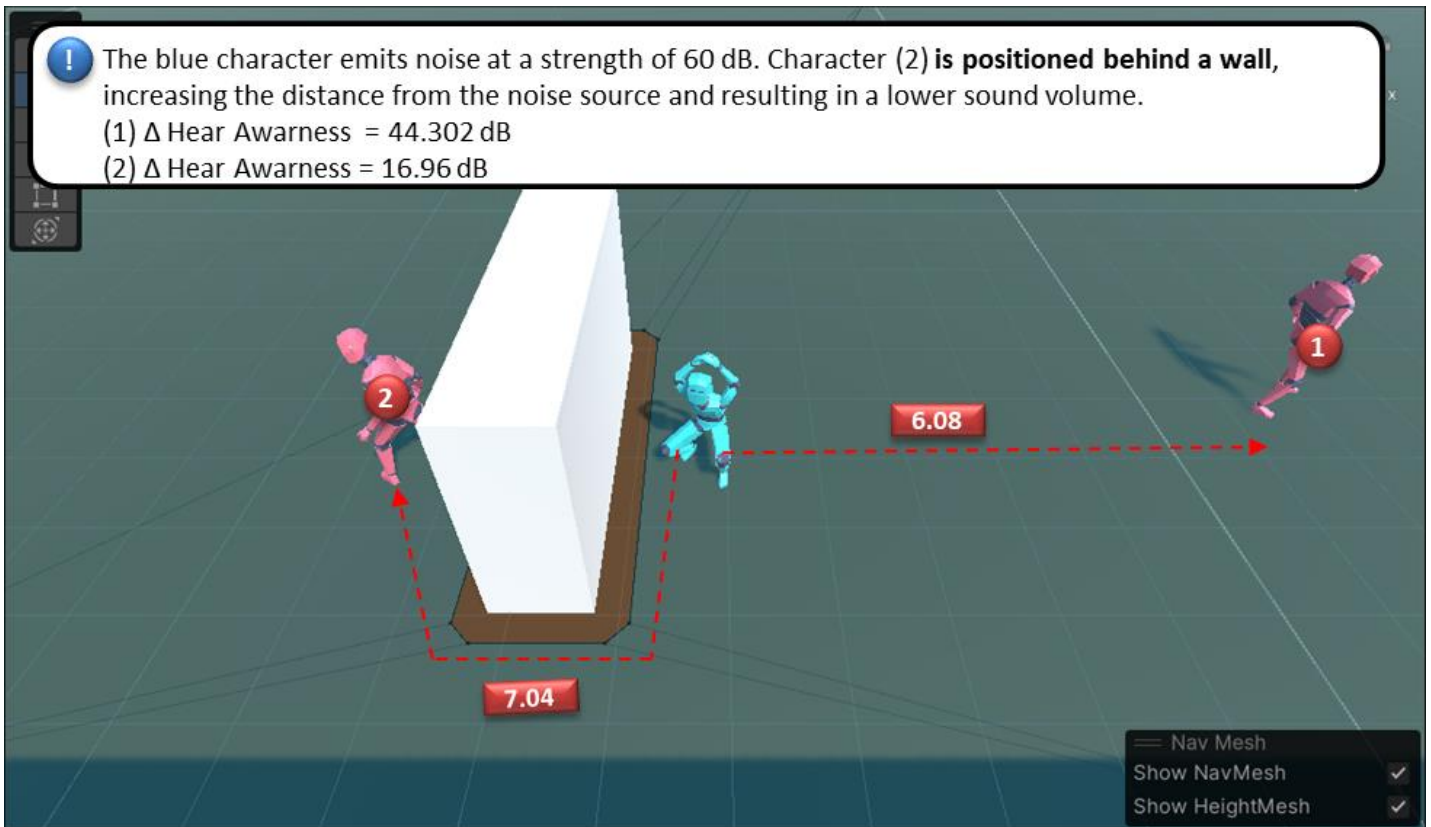
Recommended Noise Value Examples

Below is a curated list of practical noise ranges for typical gameplay actions. These are not strict rules, but well-balanced defaults that align with the internal propagation logic of your system.

ACTION		SUGGESTED NOISE VALUE
Footsteps & Movement		
Sneaking / soft footsteps		10–25 dB
Normal walking		25–40 dB
Running / sprinting		40–55 dB
Heavy armor footsteps		50–65 dB
Landing after a jump		50–70 dB
Weapons & Combat		
Silenced pistol		60–75 dB
Regular pistol		90–110 dB
Rifle / shotgun		110–130 dB
Heavy weapons		130–150 dB
Explosions		150–180 dB
Environmental Interactions		
Quiet object pickup		5–20 dB
Door opening (normal)		25–50 dB
Door slam		70–100 dB
Breaking objects (crates, glass)		80–120 dB
Machinery, engines, generators		90–140 dB
Vocalizations & Creature Sounds		
Whisper		10–20 dB
Talking		35–55 dB
Shouting		70–90 dB
Animal calls / roars		90–130 dB

Use NavMesh

When enabled, the Noise Component uses the NavMesh path distance instead of straight-line distance to determine how strongly a Hear Sense perceives a noise signal. This is especially important in environments where sound cannot travel freely in a direct line—such as when walls, corners, or dense level geometry force noise to move through longer routes. In these cases, the NavMesh provides a more realistic representation of how far the noise must “travel,” resulting in weaker perceived intensity for listeners positioned behind obstacles or outside of direct paths.



As shown in the image above, the blue character emits a 60 dB noise. Even though both listeners are placed at similar straight-line distances, the listener standing behind a wall has a much longer NavMesh path to the source. This increased travel distance results in a significantly lower perceived noise value:

- **Listener (1)** – clear path: ~44.3 dB
- **Listener (2)** – path blocked by wall: ~16.9 dB

By relying on pathfinding rather than raw geometry, the system creates believable and intuitive noise behavior. Noise no longer “passes through walls,” but instead behaves according to the actual structure and layout of the level.

Distance Based Distribution

Distance-Based Distribution controls the order in which a noise signal is delivered to all Senses Components that use the Hear Sense. When this option is enabled, the Noise Component processes listeners from the closest to the farthest, which reflects how sound naturally reaches nearby characters before those farther away. When disabled, the system relies on a more performance-oriented sphere-cast order—still effective, but not as physically accurate.

In most projects, the difference is subtle because all AI agents usually receive the noise during the same frame. However, this feature becomes valuable in more advanced or custom mechanics, such as alerting only the nearest guard first, triggering reactions based on distance, or implementing custom noise falloff logic that depends on strict listener order. In these scenarios, Distance-Based Distribution ensures that noise spreads through the environment in a predictable and realistic sequence.

Surface Based Propagation

Surface Based Propagation lets the Noise Component change how loud an action feels depending on what surface the character is standing on. Not all surfaces carry sound the same way: some materials absorb noise, while others amplify it. With this feature enabled, footsteps on a soft carpet will be much quieter than the same footsteps on a wooden floor or metal grate.

This creates more believable and immersive sound behavior for AI hearing. Instead of every sound having the same volume everywhere, noise naturally reacts to the world around it.

The Sound Propagation Settings asset allows you to define how different surfaces affect the loudness of noise. This is where you tell the system which materials should make actions quieter (like carpet) and which should make them louder (like wood or metal).

In the example above, two entries are added:

- **Carpet** — reduces noise (Factor: **0.5**)
- **Wood** — increases noise (Factor: **1.5**)

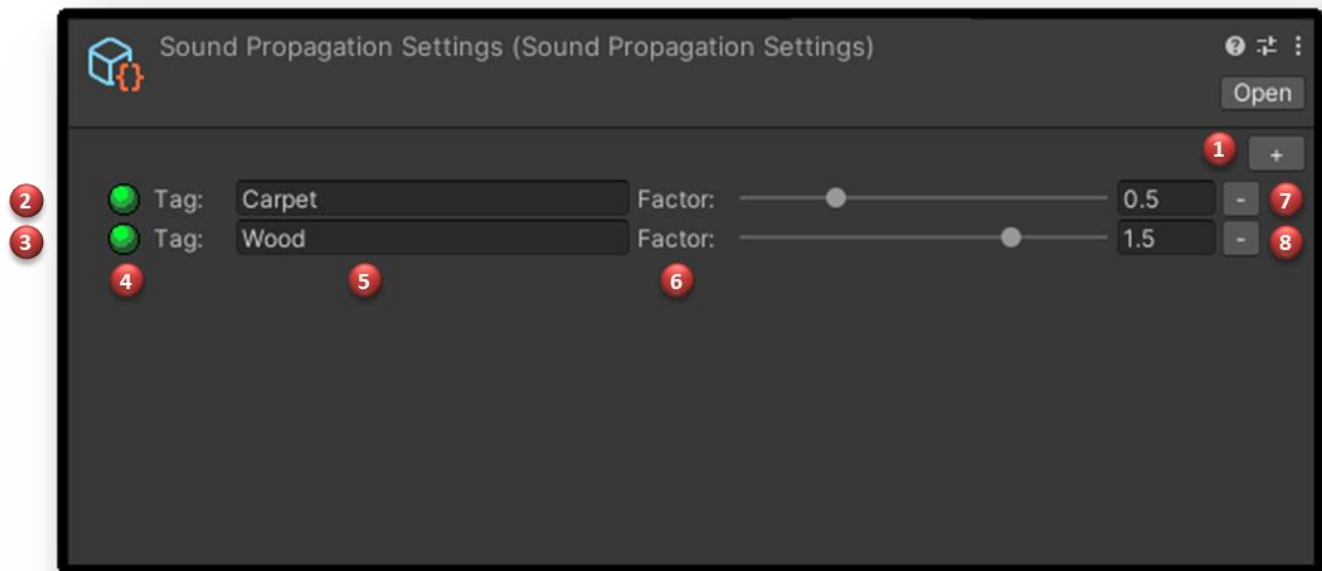
This means that footsteps on a soft carpet will be much quieter, while movement on a wooden floor will sound sharper and travel farther — just like in real life.

Where Sound Propagation Settings Are Stored

All Surface Propagation rules are stored inside a single **SoundPropagationSettings** ScriptableObject located in the project's **Resources** folder. This asset acts as a global project setting—every Noise Component uses the same file to ensure consistent behavior across the entire game. You can open and edit this asset directly from the Resources folder, but if you have trouble finding it, the Noise

Component Inspector provides a shortcut: the currently assigned SoundPropagationSettings file is shown right in the inspector, allowing you to click and highlight the asset instantly. No matter which character or object emits noise, they all reference this same configuration file, making it easy to maintain a unified sound propagation system.

Editing Sound Propagation Settings



1	Add Next Sound Propagation Data	Allows adding next Sound Propagation Data
2	Sound Propagation Data	Sound Propagation Data, for GameObject with TAG: "Carpet" with factor 0.5f;
3	Sound Propagation Data	Sound Propagation Data, for GameObject with TAG: "Wood" with factor 1.5f;
4	TAG check	Icon color inform about TAG status. Green mean TAG is correct, red mean TAG do not exist.
5	Text field	Allows setting up TAG string.
6	Factor	Allows setting up float value of factor which will multiply "_noiseValue".
7	Remove Sound Propagation Data	Allows removing Sound Propagation Data.