TARGET SENSES COMPONENT

TargetSenses is the simplest — and most important — bridge between the world and the Senses Component.

While the Senses Component is responsible for detecting things (seeing, hearing, smelling), **TargetSenses defines what can be detected.**

In short:

- **Senses** asks: "What can I perceive?"
- **TargetSenses** answers: "I am something that can be perceived."

Any GameObject that should be visible, audible, or otherwise detectable by an AI with a Senses Component must have a TargetSenses component attached. Without it, the object is completely ignored by the perception system — even if it has colliders, renders, or is standing right in front of the AI.

TargetSenses is intentionally lightweight and easy to use. It does not perform detection itself and does not contain complex logic. Its only job is to mark a GameObject as a valid perception target, define how it should be detected (single collider or rig-based colliders), and assign it to a Faction, which is later used by detection, stealth, and rule systems such as RedHanded.

Think of TargetSenses as a perception identity card:

If an object doesn't carry it, the Senses system treats it as if it doesn't exist.

Required Components

For TargetSenses to work, the GameObject must have some form of collider. This collider is what the Senses system actually interacts with when checking visibility, distance, and line-of-sight.

The good news is: almost any collider will work.

You can use:

- **Box Collider** – great for props, crates, doors, or simple objects
- **Capsule Collider** – ideal for characters and humanoid actors
- **Sphere Collider** – useful for small objects or abstract targets
- **Mesh Collider** – for complex or irregular shapes
- **Character Controller** – fully supported and commonly used for player characters

As long as the object has **at least one valid collider**, TargetSenses can make it detectable.

Important:
A GameObject without a collider cannot be detected, even if it has TargetSenses attached. The Senses system does not "see" meshes or renderers — it only works with colliders.

For characters or complex animated actors, you may also choose to use **RigCollider** instead of a single collider. This allows the Senses system to detect individual body parts for much more accurate perception. (This is optional and covered in a later section.)

In short:

No collider = invisible to Senses.

Add a collider, and the object instantly becomes part of the perception system.

Developer Note – Inspector Validation

The **TargetSenses Inspector** performs automatic validation to help prevent common setup mistakes.

When you select a **Target Type**, the inspector checks whether the required component is present on the GameObject:

- **Collider** → requires any standard Unity Collider
- **RigCollider** → requires a RigCollider component

If the required component is missing, the Inspector displays a **warning message** explaining what needs to be added.

This validation exists purely as a **design-time safety net**. It does not modify your scene, enforce components, or generate runtime errors. Its only purpose is to make incorrect or incomplete setups immediately visible, so designers can fix them before playtesting.

In short:

If something can't be detected, the Inspector will tell you why.

Factions – A Simple Way to Group Actors

The **Faction System** is an easy way to group characters and objects so the Senses system can understand **who is who** in the world.

Each object with **TargetSenses** belongs to a faction — for example BlueFaction, RedFaction, Guards, or Civilians. These groups help AI characters decide who to notice, **who to ignore, and who to react to**, without requiring any complex setup.

In the demo scene "**Demo 13 – Factions (Stress Test)**", all actors belong either to **BlueFaction** or **RedFaction**. This simple separation allows large numbers of characters to interact correctly while keeping performance high and behavior easy to understand.

Factions are meant to feel intuitive: they don't control behavior directly, but they give the system the information it needs to make smart decisions.

How Factions Are Used

Once factions are assigned, they become a powerful tool for shaping behavior in simple and predictable ways.

AI characters can use factions to decide **how to react** to what they detect. For example, a guard may ignore characters from the same faction, become suspicious of a different one, or react aggressively only to a specific group. A civilian might flee when detecting one faction, but feel safe around another.

Factions also help keep performance high. Instead of evaluating every detected object in the scene, the system can focus only on factions that matter for a given character. This makes large scenes with many actors run smoothly while still producing believable behavior.

Most importantly, factions let designers build complex interactions **without writing special logic**. By simply changing a faction value, an actor can instantly be treated as friendly, hostile, neutral, or forbidden — and all connected systems will respond accordingly.

In practice, factions turn raw detection into meaningful decisions, making AI behavior easier to design, easier to understand, and easier to scale.

SceneFactionManager – Where Factions Come From

The **SceneFactionManager** is a small helper object that connects your scene to the **Faction System**.

You don't need to add it manually.

As soon as you select or inspect any **TargetSenses** component in a scene, the **SceneFactionManager is automatically created** (if it doesn't already exist). This ensures that faction names and settings are always available when you need them, without extra setup or boilerplate steps.

The SceneFactionManager's only job is to provide access to **Faction Settings** — the place where you define how factions are named and presented across your project or scene.

Think of it as a quiet coordinator in the background:

- You rarely interact with it directly

- It keeps faction data consistent
- It makes sure the Inspector always shows correct faction names

How to Add Your Own Faction Settings

Faction names are defined using a **Faction Settings** asset. This asset controls how factions appear in the Inspector across your project or scene.

From a project perspective, it is usually best to have **one Faction Settings asset per project**, but the system also supports **different settings per scene** if your game requires it (for example, different factions in different game modes or levels).

Faction Settings are stored as a ScriptableObject and live in:
Senses / Resources / FactionSettings

Creating a New Faction Settings Asset

1. Open the **Project Window**
2. Right-click in any folder
3. Select:
   Create → CatBorg Studio → Senses → Faction Settings
4. Rename the asset to something meaningful (for example: *ProjectFactionSettings*)
5. Select the asset and edit the faction names to match your game
   (Guards, Prisoners, Civilians, Monsters, etc.)

The first entry (**None**) is reserved and should not be changed.

Assigning Faction Settings to the Scene

1. Select the **SceneFactionManager** in your scene
   (it will already exist if you've inspected a TargetSenses component)
2. In the Inspector, locate the **Faction Settings** field
3. Drag your **Faction Settings** asset into this slot

Once assigned, all **TargetSenses** components in the scene will automatically display the correct faction names in their Inspector dropdowns.

No additional setup is required.