RED HANDED SYSTEM

The RedHanded System is a general-purpose framework for handling *rule-based behavior changes* in your game. Whenever an actor enters a restricted area, interacts with a special object, or performs an action that should have consequences, the RedHanded System helps you detect it and respond automatically—without writing custom logic for every case.

The system is built from four modular parts:

- **RedHanded Component** – the core logic placed on any actor that can trigger rule checks. It continuously monitors the world for zones, objects, and actions that might affect the actor's state (usually their Faction Name).
- **RedHanded Zones** – volumes in the world that define conditional behavior rules. Crossing into, staying inside, being seen inside, or exiting a zone can all trigger state changes.
- **RedHanded Objects** – individual items or interactables that can apply RedHanded logic when approached, touched, or interacted with.
- **RedHanded Actions** – designer-created interactions (via ScriptableObjects) that define how an action behaves, whether it is instant or lasting, abortable or not, and under which conditions consequences should be applied.

None of these features are tied to a specific gameplay theme. You can use them for stealth mechanics, simulation rules, social systems, faction-based logic, security checks, puzzle conditions, or any scenario where an actor's state should change in response to what they do or where they go.

The benefit is consistency: once configured, the RedHanded System manages detection, rules, and state changes across your whole project—letting you build complex behaviors without rewriting detection logic for each new interaction.

RED HANDED COMPONNET

The RedHanded Component gives your actors the ability to react to rule-based "forbidden behaviors" in your game world—whether that means entering restricted areas, interacting with sensitive objects, or performing actions that shouldn't go unnoticed. As the actor moves and plays, the component constantly evaluates what RedHanded elements are around them and determines whether their current behavior should trigger a consequence, such as switching their **Faction Name** or escalating their status.

The system supports two levels of use. Designers can rely entirely on **RedHanded Zones and RedHanded Objects**, which require no coding at all: simply place them in the world, configure their rules, and the RedHanded Component automatically handles entry, exit, detection checks, and faction

changes. For deeper gameplay, developers can use **RedHanded Actions**—the advanced option—allowing any interaction or gameplay mechanic to become "RedHanded-aware" with just a small amount of code.

Everything runs behind the scenes: detection radius, collider scanning, zone logic, and action handling are all processed internally. Designers set the rules; the component enforces them, ensuring every RedHanded behavior in your project is consistent, reliable, and easy to expand.

Before diving into Zones, Objects, or Actions, it's important to understand what happens when the RedHanded System decides that an actor has broken a rule. These outcomes are called Repercussions, and they form the foundation of how the system changes an actor's state or behavior once a rule is violated.

Repercussions

Repercussions are the consequences applied to an actor when the RedHanded System determines they have crossed a rule boundary—such as entering a restricted area, touching a protected object, or completing an action that shouldn't go unnoticed. In most cases, this consequence is a change to the actor's **Faction Name**, signaling that the character's status has shifted in response to their behavior.

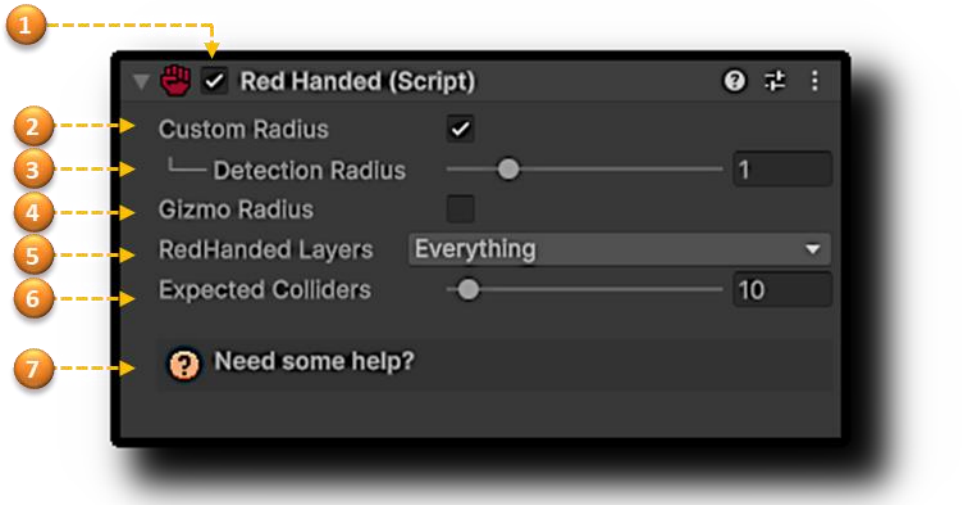
Each RedHanded feature—Zones, Objects, and Actions—defines *when* a repercussion should occur, and the RedHanded Component ensures that the rules are applied consistently. A repercussion is only triggered if the actor's current faction appears in the element's **Required Factions** list. Once triggered, the system assigns the **Result Faction**, effectively marking the actor as having been "caught," "exposed," or otherwise reassigned according to the designer's logic.

Depending on how the feature is configured, repercussions can be:

- **Immediate**, applying the moment the actor interacts
- **Conditional**, triggered only if the actor is detected
- **Completion-based**, applied when an action successfully finishes
- **Avoidable**, if the actor escapes or cancels in time
- **Unavoidable**, if the rules enforce it once started
- **Global**, affecting multiple actors across the scene

Repercussions are the glue that ties the RedHanded System together—they define what happens when the rules are broken, while Zones, Objects, and Actions define *how* those rules are encountered.

RedHanded Component Inspector Tab



1	Enabled / Disabled	When disabled, the RedHanded Component stops interacting with RedHanded Zones, Objects and Actions.
2	Custom Radius	When enabled, allows custom radius of detection for RedHanded zones and objects.
3	Detection Radius	Radius for checking around colliders with RedHanded zones and objects.
4	RedHanded Layers	Which layers contain RedHanded Zones and Objects. Only colliders on these layers can affect RedHanded component.
5	Gizmo Radius	When enabled, draws gizmo radius for visual debugging
6	Expected Colliders	Expected number of colliders in detection radius.
7	'Need some help?'	Displays Help Information.