> **NOTE**
>
> Join the Astra RPG Discord server!
> There is now a dedicated **Discord server** for Astra RPG Framework and its extensions. Join to **receive notifications** about new extension releases and important updates, **ask for new features**, **report bugs**, **share ideas**, and **showcase your Astra creations** with other developers.
>
> **Join the Discord Server: https://discord.gg/nJVRMkGrZg**

Introduction

Astra RPG Health extends the base framework, [](Astra RPG Framework), by adding functionality for managing health and calculating damage for entities. The package is designed with the same design philosophy as the base asset: a Scriptable Object-based architecture to encourage flexibility, modularity, and testability. If you're already familiar with the base package, you'll feel right at home with its features.

You can define your own damage types, designate defensive statistics used to reduce each damage type, configure the damage calculation pipeline with the desired steps, configure lifesteal for certain damage types, define strategies to execute upon entity death, and much more.

Astra RPG Health Vocabulary

Damage Type

Damage types represent the different categories of damage that can be inflicted on entities. Common examples include "Physical", "Magical", "Bleeding", "Drowning", etc. You can create custom damage types to suit your game's needs.

Damage Source

Damage sources represent the origin of the damage inflicted. This is highly specific to your game's context. For example, one game might have damage sources such as "Entity", "Environment", "Potion", etc., while another might want to define more specific damage sources like "Attack", "Spell", "Equipment", "Trap", "Environment", "Damage Over Time", etc. The main difference between damage source and damage type is that damage types categorize damage based on its nature, while damage sources identify where the damage originates. The distinction can be subtle, but it's important for game logic and mechanics. We'll return to these concepts later, where we'll see the practical differences between the two.

Heal Source

Heal sources represent the origin of healing. Similar to damage sources, heal sources are specific to your game's context. Common examples include "Potion", "Spell", "Lifesteal", "Ability", "Environment", etc. In

some games, a classic Heal Source definition might include "Self" and "Ally", as these enable mechanics for increasing healing provided/received based on the caster and target.

Barrier

The concept of temporary HP can take various names across different games, though the underlying mechanic remains the same: provide an amount of extra and ephemeral hit points that are deducted instead of health when damage is taken. In Astra, these temporary hit points are called "Barrier".

Raw and Net Damage

Raw Damage refers to the damage that a certain attack or skill intends to inflict. This damage does not account for resistances, critical hits, modifiers, etc. Net Damage is the result of processing Raw Damage while accounting for damage modifiers, resistances, barriers, critical hits, etc.

Damage Modifiers

Damage modifiers are components that can alter calculated damage in various ways. They can be used to implement mechanics such as damage reduction, damage increase, resistances, vulnerabilities, and more. Damage modifiers are generally utilized by the damage calculation pipeline (which we'll see shortly).

How is Astra RPG Health organized and how does it work?

Astra RPG Health Config

The `AstraRPGHealthConfig` is a `ScriptableObject` that serves as the central configuration point for the Astra RPG Health package. It has several properties that allow you to define how the health and damage systems should behave in your game.

With Astra RPG Framework, no configuration was needed. However, Astra RPG Health needs to be configured to work around the actual instances of the base framework's components defined for your game. For example, if you defined a certain statistic for general damage reduction in your game with Astra RPG Framework, you need to inform Astra RPG Health about it so that it can use it when calculating damage. The needed configuration is kept minimal, and convention-over-configuration is applied where possible to reduce the amount of setup required.
Configuration will be deeply discussed later in Package Configuration.

EntityHealth

`EntityHealth` is a brand new `MonoBehaviour` that you can add to your entities to provide them with health management capabilities. Worth mentioning, it exposes the most important method of the package: `TakeDamage()`, which allows you to inflict damage on the entity.

Damage Type

`DamageType` is a `ScriptableObject` that represents a specific type of damage. Each damage type can be optionally configured with a defensive `Stat` that will be used to reduce incoming damage of that type. If a defensive stat is assigned, also a piercing stat can be assigned to ignore a portion of the defense when calculating damage.
Both for the defensive and piercing stat, you can select a `DamageReductionFormula` and a `DefenseReductionFormula` respectively, to define how the stats will affect damage reduction and defense piercing.

Damage Source

`DamageSource`, derived from `ScriptableObject`, represents the origin of the damage inflicted. They don't have any specific properties, but they can be assigned to other objects of the package to create specific behaviors based on the damage source. We will see for what and how in the Workflows.

Damage Reduction Functions

The package comes with three built-in `DamageReductionFunction`s that you can use to define how defensive stats reduce incoming damage:

- `FlatDamageReductionFn`: Reduces damage by a flat amount based on the defense stat value.
- `PercentageDamageReductionFn`: Reduces damage by a percentage based on the defense stat value.
- `LogarithmicDamageReductionFn`: Reduces damage using a logarithmic scale based on the defense stat value.

In case of need, custom damage reduction functions can be created by extending the `DamageReductionFn` class.

Defense Piercing Functions

As for damage reduction functions, the package comes with three built-in `DefensePiercingFunction`s that you can use to define how piercing stats ignore a portion of the defense stat:

- `FlatDefensePiercingFn`: Ignores a flat amount of the defense stat based on the piercing stat value.
- `PercentageDefensePiercingFn`: Ignores a percentage of the defense stat based on the piercing stat value.
- `LogarithmicDefensePiercingFn`: Ignores a portion of the defense stat using a logarithmic scale based on the piercing stat value.

Also in this case, custom defense piercing functions can be created by extending the `DefensePiercingFn` class.

Heal Source

`HealSource`, derived from `ScriptableObject`, represents the origin of healing. Similar to `DamageSource`, they don't have any specific properties, but they can be assigned to other objects of the package to create specific behaviors based on the heal source. We will see for what and how in the Workflows.

Damage Calculation Strategy

The damage calculation pipeline is the component of the framework responsible for processing raw damage and producing net damage. The pipeline will use a given `DamageCalculationStrategy` to determine the sequence of steps to apply when calculating damage.
Therefore, a `DamageCalculationStrategy` is a `ScriptableObject` that defines a sequence of `DamageStep`s to be executed in order when calculating damage.

 On-death & on-resurrection `GameAction`s

Astra RPG Framework v1.4.0 introduced the concept of GameAction: a modular and reusable unit of game logic that can be assigned via the inspector to various components to define custom behaviors.

Astra RPG Health leverages Game Actions to define custom behaviors when an entity dies or is resurrected. The framework allows you to assign a default on-death Game Action that will be executed when any entity dies, unless the entity has its own custom on-death Game Action assigned. Similarly, a default on-resurrection Game Action can be assigned to be executed when an entity is resurrected.

Lifesteal Configuration

`LifestealConfig`, deriving from ScriptableObject, allows you to define how lifesteal mechanics work for specific damage types. This allows to bind a statistic to each damage type you want to have lifesteal for. That statistic will be used to calculate the amount of health to restore to the attacker when they deal damage of that type.
The configuration allows also to configure the damage pipeline timing of the lifesteal effect. For example, you might want lifesteal to occur before or after damage reduction is applied. We will see this in detail later in the Workflows.

Health Scaling Component

Astra RPG Health provides a brand new `HealthScalingComponent` that you can use in your `ScalingFormula`s to have skills or abilities scale based on either the attacker or the target's health. You can choose to scale upon one or more among Maximum HP, Current HP, and Missing HP.

 Experience Collector

An `ExpCollector` is a `MonoBehaviour` that you can add to your entities to allow them to collect experience points from entities that die and have an  `ExpSource` component attached. The `ExpCollector` can be configured with different strategies ( `ExpCollectionStrategy`) to define under which conditions experience is collected from the dead entity.

 Experience Collection Strategy

An `ExpCollectionStrategy` is a `ScriptableObject` that defines the logic for determining if an entity collects experience upon the death of another entity.

More Game Events

Astra RPG Health comes with many new Game Events that you can use to react to health&damage-related events in your game. Some of the most important ones are:

- `PreDamageGameEvent`: Triggered before damage is applied to an entity. Useful for modifying or canceling damage. Use this for implementing custom passives or effects that need to react before damage is taken.
- `DamageResolutionGameEvent`: Triggered when an entity takes damage. Can be used to react to damage being applied.
- `EntityDiedGameEvent`: Triggered when an entity dies.
- `EntityHealedGameEvent`: Triggered when an entity is healed.
- `EntityResurrectedGameEvent`: Triggered when an entity is resurrected.

And many more events. We will discuss them in detail later in the Workflows.

Package Configuration

Astra RPG Framework needed no configuration. The health package, however, needs to be configured to have the system work around the specific instances of your game. For example, if you defined a "Super Duper All-damage resistance" `Stat` in your game, you need to tell Astra RPG Health to use it when calculating damage.

The package uses a flexible configuration system that balances convenience with explicit control. This page explains how to set up and configure the health system for your game.

Configuration Overview

The Astra RPG Health system uses a **two-tier configuration architecture**:

1. **Global Settings** (`AstraRpgHealthGlobalSettings`) - A lightweight pointer stored in `Resources` that references your active configuration
2. **Gameplay Configuration** (`AstraRpgHealthConfig`) - The actual configuration containing all gameplay parameters

This separation allows you to:

- Switch between different configuration profiles easily (e.g., for testing, different game modes)
- Keep configuration data separate from the loading mechanism
- Support convention-based fallbacks for quick prototyping

Global Settings

Automatic Setup

The package automatically creates the Global Settings asset on first import or when the editor loads. You don't need to do anything manually.

What happens automatically:

1. The `AstraRpgHealthGlobalSettings.asset` is created in `Assets/Resources/`
2. If a default configuration exists (e.g., from imported samples), it's automatically assigned
3. The system is immediately ready to use

> 📁 **Default Location:** `Assets/Resources/AstraRpgHealthGlobalSettings.asset`

Project Settings

You can manage the health system configuration through Unity's Project Settings window:

1. Open **Edit → Project Settings**
2. Navigate to **Astra RPG Health**
3. Assign your desired **Active Config Profile**

Status Indicators:

- ✅ **Gray "Using Explicit Configuration"** - A configuration is explicitly assigned
- ⚠️ **Yellow "Using Fallback"** - No explicit configuration; using convention-based fallback
- 🛑 **Red "No Configuration Found"** - Critical: No configuration available

Quick Actions:

- **Create New Config Asset** - Opens a save dialog to create a new configuration

Convention Over Configuration

The package follows a **convention-over-configuration** philosophy to reduce setup friction:

Fallback Resolution

If no explicit configuration is assigned in Project Settings, the system automatically searches for a configuration named:

> `Astra Rpg Health Config`

located in any `Resources` folder in your project.

Search Order

The configuration provider uses a **three-step loading strategy**:

1. **Explicit Configuration** (Project Settings)
    - Loads `AstraRpgHealthGlobalSettings` from `Resources/AstraRpgHealthGlobalSettings`
    - If it has an `ActiveConfig` assigned, use it
2. **Convention-Based Fallback**
    - Searches for `Astra Rpg Health Config` in any `Resources` folder
    - Logs a warning indicating fallback usage
3. **Error State**
    - If neither is found, logs an error with instructions
    - System will not function until a configuration is provided

> 💡 **Tip:** For production projects, always use **explicit configuration** via Project Settings for clarity and control.

Configuration Loading Strategy

The health configuration is loaded lazily on first access and cached for performance:

```
// Automatically loads configuration on first access
var config = AstraRpgHealthConfigProvider.Instance;

// Pre-load during initialization to avoid runtime overhead
AstraRpgHealthConfigProvider.WarmUp();

// Force reload (useful for testing)
AstraRpgHealthConfigProvider.Reset();
```

When is the configuration loaded?

- Automatically before the first scene loads (via `RuntimeInitializeOnLoadMethod`)
- Lazily when first accessed via `AstraRpgHealthConfigProvider.Instance`
- Explicitly when calling `WarmUp()`

Creating Configuration Assets

If for any reason you need to create a new `AstraRpgHealthConfig` asset, you can do so via two methods:

Via Project Settings

1. Open **Edit** → **Project Settings** → **Astra RPG Health**
2. Unassign any existing configuration, if any
3. Click **Create New Config Asset**
4. Choose a save location
5. The new configuration is automatically assigned

Via Asset Menu

1. Right-click in the **Project Window**
2. Select **Create** → **Astra RPG Health** → **Configuration**
3. Name your configuration
4. Assign it in Project Settings or in the Global Settings asset

---

Health Configuration Reference

> [!INFO] In the `AstraRpgHealthConfig` asset, you can hover over each field to see a tooltip with a brief description.

The `AstraRpgHealthConfig` asset contains all gameplay parameters for the health system. Below is a detailed explanation of each field.

Health

Health Attributes Scaling

Type: `AttributesScalingComponent`
Required: No
Description: Defines how entities' maximum health scales based on character attributes (e.g., Vitality, Endurance).

> **See Also:** Astra RPG Framework Scaling documentation

Generic Heal Amount Modifier Stat

Type: `Stat`
Required: No
Description: The stat that modifies **all** healing received by an entity.

Stacking Behavior:

- Combines **additively** with Source modifications

How it works:

- The stat value represents a **percentage modifier**
- Positive values increase healing, negative values decrease it
- Example: A value of `25` means +25% healing received

Example:

- Base Heal: 100 HP
- Generic Heal Amount Modifier: 25 (means +25%)
- Final Heal: 100 * 1.25 = **125 HP**

Heal Source Modifications

Type: `Dictionary<HealSource, Stat>`
Required: No
Description: Maps each healing source (e.g., Skill, Trap, Environment) to a modifier stat.

Use cases:

- Create specialized healing boosts or reductions (e.g., "Resurrection healing is 50% more effective", "Healing from potions is reduced by 20%")

Stacking Behavior:

- Stats stack additively with Generic modifications

Example:

- Incoming Heal: 100 from resurrection
- Generic Heal Amount Modifier: +30% (means +30% healing received)
- Resurrection Heal Amount Modifier: +20 (means +20% healing received)
- **Total Heal Modification:** +30% +20% = +50% → Final Heal: **150**

Damage

Default Damage Calculation Strategy

Type: `DamageCalculationStrategy`
Required: No
Description: The default strategy used to calculate net damage when an entity doesn't specify its own strategy.

Purpose:

- Provides a default damage calculation strategy to use for "regular" entities. In most cases, this is sufficient
- Can be overridden per-entity for custom damage pipelines in their `EntityHealth` component

> **See Also:** Damage Calculation Pipeline documentation

Generic Damage Modification Stat

Type: `Stat`
Required: No
Description: A universal damage modifier that applies to **all damage received**, regardless of type or source.

Usage:

- Applied in the damage calculation pipeline if `ApplyDmgModifiersStep` is included in the used strategy

Stacking Behavior:

- Combines **additively** with Type and Source modifications

Example:

- Incoming Damage: 100
- Generic Damage Modification: -20 (means -20% damage taken)
- **Damage Reduction:** -20% → Final Damage: **80**

Damage Type Modifications

Type: `Dictionary<DamageType, Stat>`
Required: No
Description: Maps each damage type (e.g., Fire, Ice, Physical) to a modifier stat.

Usage:

- Applied in the damage calculation pipeline if `ApplyDmgModifiersStep` is included in the used strategy

Use cases:

- Configure different resistances/vulnerabilities per damage type **Stacking Behavior:**
- Stats stack additively with Generic and Source modifications

Example:

- Incoming Damage: 100 Fire damage
- Generic Damage Modification: +10 (means +10% damage taken)
- Fire Damage Modification: -30 (means -30% damage taken)
- **Total Damage Modification:** +10% - 20% = -10% → Final Damage: **90**

Damage Source Modifications

Type: `Dictionary<DamageSource, Stat>`
Required: No
Description: Maps each damage source (e.g., Skill, Trap, Environment) to a modifier stat.

Usage:

- Applied in the damage calculation pipeline if `ApplyDmgModifiersStep` is included in the used strategy

Use cases:

- Create specialized defenses (e.g., "Traps deal 50% less damage")
- For keeping track of damage origins for a combat log or analytics

Stacking Behavior:

- Stats stack additively with Generic and Type modifications

Example:

- Incoming Damage: 100 Physical damage from a Trap
- Generic Damage Modification: +30% (means +30% damage taken)
- Physical Damage Modification: -10 (means -10% damage taken)
- Trap Damage Modification: -50 (means -50% damage taken)
- **Total Damage Modification:** +30% -10% -50% = -30% → Final Damage: **70**

Health Regeneration

Health Regeneration Source

Type: `HealSource`
Required: No
Description: The heal source used for passive regeneration effects.

Use cases:

- Allows tracking and modifying regeneration separately from active healing
- Can be used for effects like "Increase Regeneration by 50%"
- Tracking passive healing in analytics or combat logs

Passive Health Regeneration Stat (HP/10s)

Type: `Stat`
Required: No
Description: Determines the amount of health regenerated passively.

> **WARNING**
> The stat value represents health regenerated **per 10 seconds**.

Calculation:

```
Health Per Tick = (Stat Value / 10) * Interval In Seconds
```

Example:

- Stat Value: 50 HP/10s
- Interval: 1 second
- Health Per Tick: (50 / 10) * 1 = **5 HP per second**

Passive Health Regeneration Interval

Type: `float`
Default: 1.0 seconds
Required: Yes (must be > 0)
Description: The time (in seconds) between passive regeneration ticks.

Configuration Examples:

Interval	Stat Value	Result
1.0s	50 HP/10s	5 HP every 1 second
0.5s	50 HP/10s	2.5 HP every 0.5 seconds
2.0s	50 HP/10s	10 HP every 2 seconds

> **WARNING**
> Smaller intervals increase CPU overhead. Recommended range: 0.5s - 2.0s.

Suppress Passive Regeneration Events

Type: `bool`
Required: No
Description: If enabled, prevents triggering any heal events during passive regeneration ticks. Keep it disabled if you need to track passive regeneration in a combat log or if you have effects that trigger on heal events and should also apply to passive regeneration. If your game doesn't require any of the above and you want to minimize overhead, you can enable this option and skip all heal-related logic during regeneration ticks. Useful if your game has a lot of entities with passive regeneration and you want to optimize performance. If you need both to keep sending regeneration events and to minimize overhead, I advise to increase the regeneration interval and to keep the "Suppress Passive Regeneration Events" option disabled. This way you can have less regeneration ticks and still trigger events for each tick.

Manual Health Regeneration Stat

Type: `Stat`
Required: No
Description: Determines health regenerated when triggering manual regeneration via API. The amount of health regenerated is equal to the value of this stat.

Use Cases:

- **Turn-based systems:** Regenerate health at the end of each turn
- **Rest mechanics:** Trigger regeneration when resting at campfires
- **Time-skip systems:** Apply regeneration for elapsed time

API Usage:

```
// Trigger manual regeneration
entityHealth.ManualHealthRegenerationTick();
```

Lifesteal

Lifesteal Config

Type: `LifestealConfig`
Required: No
Description: Configuration for lifesteal mechanics (healing based on damage dealt).

Typical Settings:

- Lifesteal percentage stat
- Heal source for lifesteal effects
- Restrictions (e.g., only on critical hits, only physical damage)

> **See Also:** Lifesteal documentation

Death

Default On Death Game Action

Type: `GameAction`
Required: Yes
Description: The game action executed when an entity dies (if the entity doesn't have its own on-death game action). Use a composite game action to chain multiple effects.

Common on-death Game Action Ideas:

- **Destroy GameObject** - Removes the entity from the scene
- **Ragdoll** - Enables ragdoll physics
- **Respawn** - Respawns the entity after a delay
- **Loot Drop** - Spawns loot and destroys the entity

Example:

```
Death → Execute composite on-death Game Action → [Spawn Death VFX → Drop Loot →
Destroy GameObject]
```

Default On Resurrection Game Action

Type: `GameAction`
Required: No
Description: The game action executed when an entity is resurrected (if the entity doesn't have its own on-resurrection game action). Use a composite game action to chain multiple effects.

Common on-resurrection Game Action Ideas:

- **Simple Resurrection** - Restores health and enables the entity
- **Resurrection VFX** - Plays visual effects during resurrection
- **Stat Penalties** - Applies temporary debuffs after resurrection

Default Resurrection Source

Type: `HealSource`
Required: Yes
Description: The heal source used when an entity is resurrected.

Use cases:

- Categorizes resurrection healing separately from normal healing
- Allows effects like "Increase Resurrection Healing by 50%"
- Used for analytics and gameplay feedback

Troubleshooting

⚠️ "No Configuration found!" error

Cause: No configuration is assigned and no fallback exists.

Solution:

1. Check **Project Settings** → **Astra RPG Health**
2. Assign a configuration or create a new one
3. Alternatively, create a config named `Astra Rpg Health Config` in a `Resources` folder

⚠️ "Using Fallback" warning

Cause: No explicit configuration assigned in Project Settings.

Solution:

1. Open **Project Settings → Astra RPG Health**
2. Assign the fallback configuration explicitly
3. This warning is just informational and won't break functionality

⚠️ Configuration not updating in Play Mode

Cause: Configuration is cached on first access.

Solution:

```
// Force reload
AstraRpgHealthConfigProvider.Reset();
```

⚠️ Missing Resources folder

Cause: The `Assets/Resources/` folder doesn't exist.

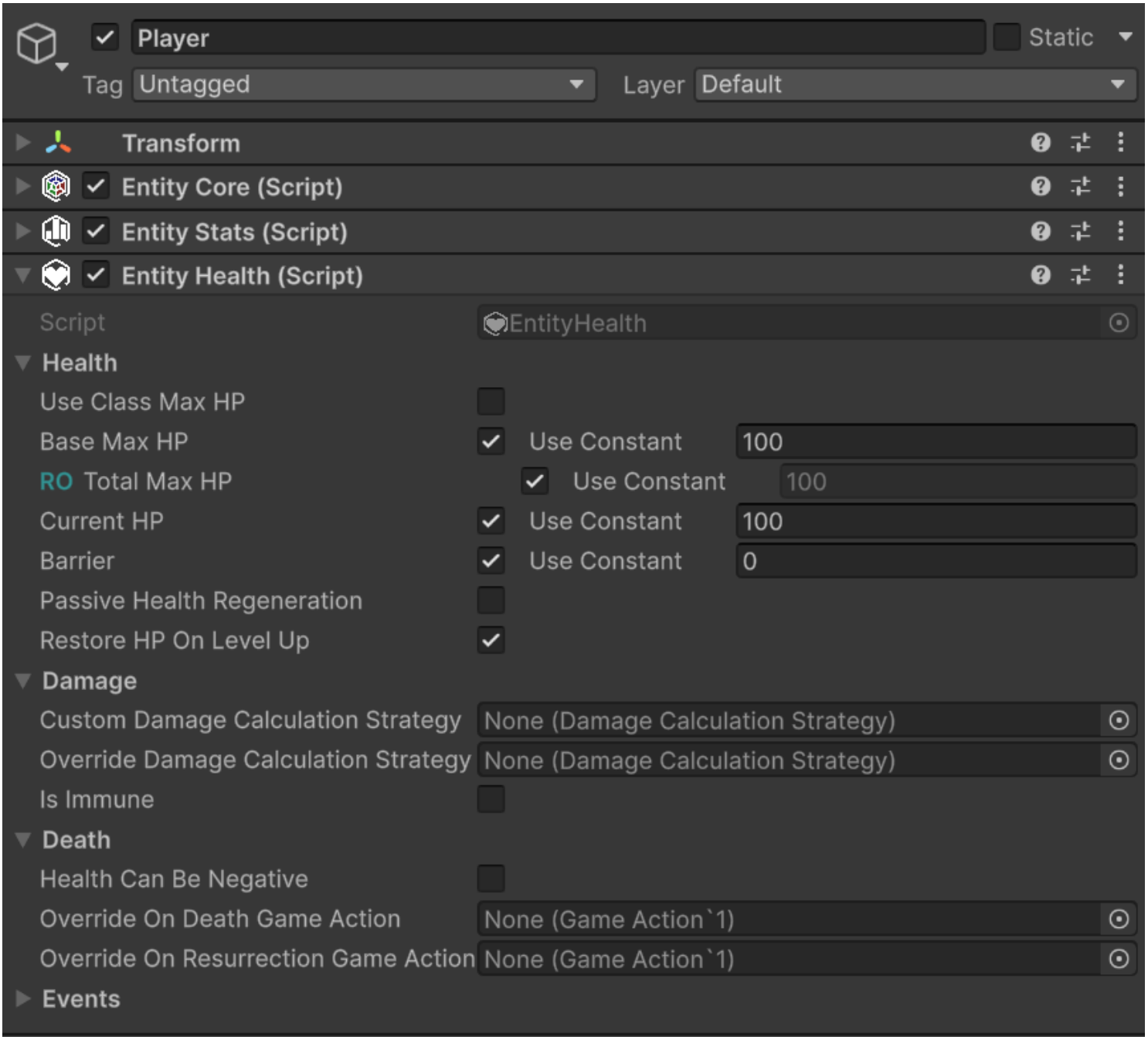
Solution: The package creates it automatically. If it's missing, create it manually:

1. Right-click `Assets`
2. Create → Folder → Name it `Resources`
3. Restart Unity to trigger the bootstrapper

`EntityHealth` Component

With Astra RPG Health, the new `EntityHealth` MonoBehaviour allows you to add health points to an entity. In this way, the entity can take damage and, consequently, die.

Here is an example of the `EntityHealth` component:



The events section is collapsed by default since there are many events, and it is more practical to expand it only when necessary.

Let's proceed in order and analyze every property of the `EntityHealth` component.

Health

- **Use Class Max HP**: Boolean indicating whether to use the max health points defined in the entity's class as the base max health. If disabled, you can manually specify the base max health for the entity.
- **Base Max HP**: LongRef representing the **base** max health of the entity. If "Use Class Max HP" is enabled, this value is marked with teal `RO` (Read Only). Read Only, for LongRef fields, makes `const` values non-editable, and suggests not manually modifying values contained by the associated LongVar variable, if a `const` value is not used.
- **Total Max HP**: LongRef representing the **total** max health of the entity, calculated based on the base max health and any modifiers. This field is always read-only (RO) and is automatically updated when base max health or modifiers change.
- **Current HP**: LongRef representing the entity's current health points. Editable field that is automatically updated when the entity takes damage or is healed. This field is also updated if `Base Max HP` is modified from the inspector.

> **NOTE**
>
> If `Current HP` drops to 0, modifying `Base Max HP` will not update `Current HP`. This is intended behavior, as if an entity is dead, it makes no sense for its health points to be modified by a change in max health. If instead the entity is alive, modifying `Base Max HP` will update `Current HP` accordingly even if `Current HP` is currently 0. This can happen if, for example, you change from 10 base max HP to 25 by deleting the text box with backspace: 10 (backspace) -> 1 (backspace) -> 0 (2 pressed) -> 2 (5 pressed) -> 25. Note that by deleting all numbers and leaving the box empty, `Base Max HP` becomes 0, and consequently `Current HP` also becomes 0. At this point, the entity is considered dead.

- **Barrier**: LongRef representing any barrier points (or temporary HP) of the entity. The barrier absorbs damage before it affects the entity's health points.
- **Passive Health Regeneration**: Boolean that decrees whether the entity passively regenerates HP over time or not. The regeneration frequency, as well as the statistic to consider for the amount of passively regenerated HP, are defined in the Astra RPG Health Config configuration.
- **Restore HP On Level Up**: Boolean indicating whether the entity is fully healed when leveling up.

Damage

For a better understanding of the first two properties of this section, I recommend taking a look at the Damage Calculation Strategy documentation. Simply put, a Damage Calculation Strategy defines how the damage an entity is about to take is calculated (e.g., applying damage reduction for the defensive stat first, or damage absorption by the barrier first, when to apply the critical multiplier, etc.). Also recall that a default strategy can be assigned via configuration. See Default Damage Calculation Strategy.

- **Custom Damage Calculation Strategy**: Field of type `DamageCalculationStrategy`. If the entity should use a custom damage calculation strategy, you can specify it here. This, if defined, takes precedence over the default one defined via configuration. A common use case could be, for example, a boss that cannot take more than 10% of its max health in damage at a time.
- **Override Damage Calculation Strategy**: Field of type `DamageCalculationStrategy`. If defined, it takes precedence over all other damage calculation strategies. Designed to be assigned at runtime to implement special effects or for testing/debug. For example, an entity is affected by a debuff that turns all physical damage into guaranteed critical hits. This debuff could therefore be implemented through a custom strategy assigned to the entity in this field.
- **Is Immune**: Boolean indicating whether the entity is immune to all damage or not. If enabled, the entity will take no damage, regardless of the damage calculation strategy used.

Death

- **Health Can Be Negative**: Boolean indicating whether the entity's health points can drop below 0 before dying. If disabled, the entity's health points will never drop below 0, and the entity will die as soon as its health points reach 0. If enabled, the entity's health points can drop below 0, and the entity will die only when its health points reach a specified negative value (Death Threshold).
- **Death Threshold**: LongRef representing the entity's death threshold. Visible only if "Health Can Be Negative" is enabled. If the entity's health points reach this threshold, the entity dies.
- **Override On Death Game Action**: Game Action that is automatically executed when the entity dies. If defined, this takes precedence over the one defined in the configuration. Useful for implementing special behaviors upon an entity's death (for example, entities that explode upon death dealing area damage).
- **Override On Resurrection Game Action**: Game Action that is automatically executed when the entity is resurrected. If defined, this takes precedence over the one defined in the configuration. Useful for implementing special behaviors upon an entity's resurrection.

Damage vs RemoveHealth and Heal vs AddHealth

`EntityHealth` provides a couple of public methods to increase current HP and two to decrease them:

- `Heal` and `AddHealth` to increase current HP.
- `TakeDamage` and `RemoveHealth` to decrease current HP.

`TakeDamage` and `Heal` are extensively documented in the Dealing Damage to an Entity and Healing an Entity sections respectively. However, this sounds like a good moment to introduce the difference between these two pairs of methods, and when to use one or the other.

It is important to clarify why two different methods exist to increase and decrease current HP, and when to use one or the other.

`AddHealth` and `RemoveHealth` operate at a lower level of abstraction than `Heal` and `Damage`, as they directly modify the entity's current HP by a specified `long` value, without passing through the damage calculation pipeline or without taking into account any healing or damage modifiers. They are very predictable and direct methods, and are mainly used internally by the framework. These methods can be useful in specific situations where the gain of HP is not due to healing, or the loss of HP is not due to damage, but rather to particular mechanics that require a direct modification of current HP. For example, suppose that following a cutscene we want to force the player's HP to 1. In this case, we could use `RemoveHealth` to remove all HP except 1, without having to worry about any damage modifiers or the damage calculation strategy that would alter the damage taken based on the player's stats and equipment, as well as raising damage events that could trigger game logic we don't want to activate in this specific case.

`Heal` and `TakeDamage`, on the other hand, are more complex methods that take into account various factors such as the damage calculation strategy, damage immunity, healing modifiers, etc. These methods are intended to be used primarily by game developers to apply damage and healing to entities, as they guarantee that all mechanics and rules of the health system are respected. For example, if you want to inflict damage on an entity, it is advisable to use the `TakeDamage` method, so that the damage is calculated correctly based on the configured damage calculation strategy, and that any immunities or modifiers are taken into consideration. Unsurprisingly, `Heal` and `TakeDamage` internally use `AddHealth` and `RemoveHealth` to effectively modify the entity's current HP after calculating the net HP gain or loss to apply.

In 99% of cases, you will use `Heal` and `TakeDamage` to apply healing and damage to entities. `AddHealth` and `RemoveHealth` are available to cover rarer and more particular use cases.

Events

Events are, by default, collapsed as there are many of them and they would expand excessively in the inspector. By opening them, we should see something like this:

In the image, you see already assigned events, but when you first add the component, all events will be unassigned and empty.

Let's start by introducing the difference between *Global Events* and *Extra Events*.

Global Events

Global Events are fundamental events that transmit important information to the whole framework. Events assigned to these slots must have a global scope, i.e., they must be able to be listened to by any entity or system in the game. These events are used by the framework to handle essential features, such as lifesteal and other mechanics that require centralized communication between the various parts of the system.

> **WARNING**
> Reserving these slots for events of a global nature is fundamental to ensure the correct functioning of the framework. Assigning Game Event instances that are specific to one or a few entities would involve operating problems at the framework level.

Extra Events

Extra Events, instead, are designed to transmit information to specific components or a restricted circle of entities. A practical example is the communication between the player's EntityHealth and the HUD displaying their HP: only the player possesses a dedicated HUD, so it is useful to assign an exclusive event for this interaction. In this way, the GameEvent associated with the player's EntityHealth is listened to only by the HUD, guaranteeing clear compartmentation and greater efficiency. Thus, the HUD does not receive events from all entities, but only those relevant to the player.

Events Breakdown

Here is a detailed description of each event. Each type of event has both the global event and a list of extra events, but it is sufficient to describe each event once.

Damage Related Events

To better understand the first two events of this section, I recommend taking a look at the Damage documentation to get a better understanding of damage in Astra RPG Health.

- **Pre Damage Info Event**: Event raised before the entity takes damage, and before the damage calculation pipeline calculates the net damage. This event transmits information about the damage the entity is about to take, such as the damage type, the damage source (an entity, `null` if not applicable), the damage source type (e.g., environmental, skill, trap, etc.), the raw damage you intend to inflict, and other relevant information. For more details regarding the context parameter and its fields, refer to the API documentation PreDamageContext. This event can be useful for implementing passive abilities or, in general, advanced mechanics that trigger effects in response to specific conditions related to the damage an entity is about to take. For example, a debuff or status modifier that amplifies critical damage taken by 50% when the damage type is "Fire", or a passive

ability that negates all instances of damage currently being taken if they had raw damage less than 50.

- **Damage Resolution Event**: Event raised after the entity has taken or ignored damage. Similarly to the context parameter of the previous event, this event transmits detailed information about the damage just taken or ignored. For more details regarding the context parameter and its fields, refer to the API documentation DamageResolutionContext.

In general, the thumb rule for deciding whether to use the Pre Damage Info Event or the Damage Resolution Event is the following: if you want to manipulate the damage an entity is about to take, it is better to subscribe to Pre Damage Info Event, and modify the context as desired; if instead you want to react to the damage an entity has just taken, and trigger effects in response to it, it is better to subscribe to Damage Resolution Event, and react based on the information transmitted by its context.

Health Related Events

- **Gained Health Event**: Event raised when the entity gains HP, whether through healing or other mechanisms (e.g., Max HP modifiers that caused a gain of health). Refer to the EntityHealthChanged Context API for more details on the context parameter.
- **Lost Health Event**: Event raised when the entity loses HP, whether through taking damage or other mechanisms (e.g., Max HP modifiers that caused a loss of health). The context parameter of this event is the same as the previous one, so refer to the EntityHealthChangedContext API for more details.

> **NOTE**
>
> Because `TakeDamage` and `Heal` internally invoke `RemoveHealth` and `AddHealth`, calling these methods will also raise the `EntityGainedHealth` and `EntityLostHealth` events respectively.

- **Max Health Changed Event**: Event raised when an entity's total max health points change. This event is raised both when total max hp increase and when they decrease. See EntityMaxHealth ChangedContext API for more details on the context parameter.
- **Entity Died Event**: Event raised when the entity dies. See EntityDiedContext API for more details on the context parameter.

Healing Related Events

- **Pre Heal Event**: Event raised before the entity is healed. This event transmits information about the healing the entity is about to receive, such as the amount of HP you intend to heal, the entity that provided the healing (`null` if not applicable), the heal source (e.g., skill, item, potion, etc.) and other relevant information. For more details regarding the context parameter and its fields, refer to the API documentation PreHealContext. The amount of health points intended to heal, transmitted by this event, is the value before applying any healing modifiers. This event can be useful for

implementing passive abilities or, in general, advanced mechanics that trigger effects in response to specific conditions related to the healing an entity is about to receive. For example, a buff that amplifies healing derived from potions (heal source) by 30%.

- **Entity Healed Event**: Event raised when the entity has been healed. This event transmits information about the healing just received, such as the amount of HP the entity actually gained after the application of any healing modifiers. For more on the context parameter see the Received HealContext API.

Here too, as for damage events, the thumb rule for deciding whether to use the Pre Heal Event or the Entity Healed Event is the following: if you want to manipulate the healing an entity is about to receive, it is better to subscribe to Pre Heal Event, and modify the context as desired; if instead you want to react to the healing an entity has just received, and trigger effects in response to it, it is better to subscribe to Entity Healed Event, and react based on the information transmitted by its context.

Resurrection Related Events

- **Entity Resurrected Event**: Event raised when the entity is resurrected. See ResurrectionContext API for more details on the context parameter.

Damage

Damage Sources

Damage Types

Defensive Stats

Defense Penetration

Damage Calculation Pipeline

PreDamageContext and DamageResolutionContext

Damage Step

Damage Calculation Strategy

Dealing Damage to an Entity

The API method you will use the most with this package is certainly `TakeDamage`, whose responsibility is to apply damage to the entity, taking into account modifiers, immunity, the damage calculation strategy, and other relevant mechanics. This method takes a `PreDamageContext` as input.

The recommended way via code to inflict damage on an entity is as follows:

1. Construct an instance of `PreDamageContext` with all relevant information about the damage you intend to inflict through its fluent builder.
2. Call `TakeDamage` passing the newly constructed context.

Suppose we are implementing a skill that deals 50 fire damage to the target. The code to apply damage to the target could be the following:

```
// Assuming that:
// - dmgType is a DamageType representing fire damage
// - dmgSource is a DamageSource representing the damage coming from a skill
// - target is the EntityCore that we want to damage
// - skillCaster is the EntityCore that casts the skill

// first we ensure that the target has an EntityHealth component
if (target.TryGetComponent(out EntityHealth targetHealth)) {
    // then we build the PreDamageContext with all the relevant information
    var preDamageContext = PreDamageContext.Builder
            .WithAmount(50)
            .WithType(dmgType)
            .WithSource(dmgSource)
```

```
            .WithTarget(target)
            .WithDealer(skillCaster)
            .Build();

    // finally, we call TakeDamage to apply the damage to the target
    targetHealth.TakeDamage(preDamageContext);
}
```

Thanks to the `PreDamageContext` fluent builder, the IDE will automatically suggest the fields to fill in one at a time. As long as it presents them one at a time, it means they are required fields. If instead it presents more than one at a time, it means those fields are optional, and you can decide whether to fill them in or build the context without them. Optional fields are, for example, the critical hit flag and the critical multiplier. In the example, for simplicity, I did not fill in these fields.

Now, we know that hardcoding the damage value directly in the code is not a good practice. Let's see how to use a `ScalingFormula` to dynamically calculate the amount of damage to inflict. The step builder creation would become the following:

```
// Assuming that scalingFormula is a ScalingFormula that calculates the damage amount based
on the skill caster's stats...

// ...we calculate the damage amount by evaluating the scaling formula
long damageAmount = scalingFormula.CalculateValue(skillCaster);
var preDamageContext = PreDamageContext.Builder
        .WithAmount(damageAmount)
        .WithType(dmgType)
        .WithSource(dmgSource)
        .WithTarget(target)
        .WithDealer(skillCaster)
        .Build();
```

Healing

An entity can be healed in 4 different ways:

- Direct healing through the `Heal` method.
- Health regeneration. This can be passive, as defined in the Health Regeneration configuration, so automatically triggered by the framework every so often, or it can be manually activated via the `ManualHealthRegenerationTick` method. Also for the latter, you must configure the statistic to consider for the calculation of healing through the configuration.
- Through lifesteal, that is healing the entity based on the damage dealt. However, lifesteal functioning is detailed in Lifesteal as it deserves a dedicated section.
- Via resurrection with the two `Resurrect` methods. One to resurrect the entity with a percentage of HP, and one to resurrect it with a fixed amount of health points. Applicable only if the entity is dead. Also resurrection is detailed in a dedicated section, Resurrection.

> **NOTE**
>
> Regeneration (both passive and manual), lifesteal, and resurrection all use, behind the scenes, the `Heal` method to effectively heal the entity.

We will first introduce some concepts that are common to all the four healing methods, and then we will analyze direct healing and health regeneration. As aforesaid, lifesteal and resurrection are analyzed in their respective sections, Lifesteal and Resurrection.

Heal Source

Una `HealSource` rappresenta la fonte di guarigione. Alcuni esempi di `HealSource` potrebbero essere: pozioni curative, abilita', rigenerazione passiva, lifesteal, sistema (ad esempio, il giocatore viene curato alla vita massima durante il tutorial da un macchinario di supporto), e cosi' via. La `HealSource` è un concetto importante in quanto permette di distinguere tra le diverse fonti di guarigione, e quindi di applicare modificatori di guarigione specifici per ogni fonte di guarigione, oltre ai modificatori di guarigione generici che si applicano a tutte le fonti di guarigione. Inoltre, le `HealSource` sono importanti anche perche' vengono comunicate nei contesti di guarigione, e quindi nei listener degli eventi di guarigione, permettendo a questi ultimi di reagire in modo specifico in base alla fonte di guarigione che ha scatenato l'evento. Grazie a questo si potrebbe, ad esempio, implementare un buff che aumenta la guarigione ricevuta dalle pozioni curative. Infine, le `HealSource` aprono la strada all'implementazione di un combat log che traccia non solo la quantità di guarigione ricevuta, ma anche la fonte di guarigione, permettendo al giocatore di avere una comprensione più profonda di cosa lo ha curato piu' efficacemente in una situazione di gioco specifica.

Heal Modifiers

Direct Healing

The `Heal` method is the most straightforward way to heal an entity. It takes a `PreHealContext` as input, which contains all relevant information about the healing you intend to apply and operates as follows:

1. It checks if the entity is dead. If the entity is dead, will rais an exception, as you cannot heal a dead entity. If you want to resurrect a dead entity, check the Resurrection section.
2. It checks if the healing to be applied is marked as a critical hit and, if so it applies the critical multiplier to the healing amount.
3. It retrieves the generic heal modifiers for the entity.
4. It retrieves the heal modifiers specific to the heal source.
5. It sums the retrieved modifiers and applies them to the healing amount.
6. It increases the entity's current HP by the final healing amount.
7. It raises the *Entity Healed Event* with a `HealResolutionContext` containing all relevant information about the healing that was just applied.
8. It returns the `HealResolutionContext` created in the previous step so that the caller can use it for further processing if needed.

The `Heal` method, differently from the `TakeDamage` method, cannot be configured to reorder the steps of the healing pipeline. This choice was made as healing is generally more straightforward than damage, and there are usually no mechanics that require reordering the steps of the healing pipeline. Therefore, I evaluated simplicity and ease of use more important than flexibility for this method.

`Heal` Usage Example

Suppose we want to heal an entity for 20% of its total max HP. The code could be the following:

```
// Assuming that:
// - healSource is a HealSource representing the healing coming from a skill
// - skillCaster is the EntityCore that casts the healing skill
// - target is the EntityCore that we want to heal

if (target.TryGetComponent(out EntityHealth targetHealth)) {
    // in case the target has a EntityHealth component...
    long healAmount = target.GetMaxHpPortion(0.2d);

    PreHealContext preHealContext = PreHealContext.Builder
        .WithAmount(healAmount)
        .WithSource(healSource)
        .WithHealer(skillCaster)
        .WithTarget(target.EntityCore)
        .Build();
```

```
    targetHealth.Heal(preHealContext);
}
```

It is not a good practice to hardcode the healing amount directly in code (20% of max hp in the example). Hardcoded values make tuning and balancing difficult, prevent reuse across different skills or items, and force code changes and recompilation for values that designers or content creators should be able to tweak. Prefer one of these approaches instead:

- **Serialize the amount** as an inspector-editable field so designers can tweak values without touching code.
- **Use a `ScalingFormula`** (or equivalent) to compute the heal amount dynamically from the caster's or target's stats, allowing the effect to scale with progression and remain data-driven.

Example using a `ScalingFormula` to calculate the heal amount at runtime:

```
// Assuming that scalingFormula is a ScalingFormula that calculates the heal amount
// based on the skill caster's stats...
long healAmount = scalingFormula.CalculateValue(skillCaster);

PreHealContext preHealContext = PreHealContext.Builder
    .WithAmount(healAmount)
    .WithSource(healSource)
    .WithHealer(skillCaster)
    .WithTarget(target.EntityCore)
    .Build();

targetHealth.Heal(preHealContext);
```

Recommended pattern when healing programmatically:

1. Construct a `PreHealContext` with all relevant information using the fluent builder.
2. Call `Heal` passing the constructed context.

The `PreHealContext` fluent builder is helpful because it guides you through required inputs first — the IDE will typically present required builder steps one at a time. When the builder presents multiple fields together, those fields are optional and can be omitted.

Health Regeneration

La rigenerazione della vita è un meccanismo che permette a un'entità di rigenerare la propria vita passivamente nel tempo. Nei giochi in generale, questa meccanica e' implementata attraverso tick discreti di rigenerazione. I tick, in base al gioco, possono essere scanditi dal tempo che passa, o da certi

eventi, come il finire di un turno in un gioco a turni.
Questo package supporta sia la rigenerazione passiva che quella manuale.

Passive Health Regeneration

La rigenerazione passiva, per funzionare, necessita di essere configurata attraverso la sezione Health Regeneration della configurazione del package. Li' vanno configurati i seguenti parametri:

- **Health Regeneration Source**: la `HealSource` da associare alla rigenerazione (passiva e manuale). Il package userà questa `HealSource` per creare i contesti di guarigione da passare al metodo `Heal` quando è il momento di rigenerare la vita.
- **Passive Health Regeneration Stat (HP/10s)**: la quantità di HP che l'entità rigenera ogni 10 secondi. Il package si occuperà di scalare questa quantità in base al tempo che passa tra un tick e l'altro, in modo da garantire una rigenerazione fluida e coerente.
- **Passive Health Regeneration Interval**: l'intervallo di tempo, in secondi, tra un tick di rigenerazione e l'altro.

Inoltre, e' importante ricordarsi di attivare nell'`EntityHealth` dell'entita' la rigenerazione passiva.

Per fare un esempio, se vogliamo che un'entità rigeneri 5HP ogni 0.5 secondi, dobbiamo configurare il parametro "Passive Health Regeneration Interval" a 0.5 e l'entita' deve avere la statistica associata a "Passive Health Regeneration Stat (HP/10s)" con un valore di 100 (5 HP ogni 0.5 secondi corrispondono a 10HP al secondo, che corrispondono a **100 HP ogni 10 secondi**).

Performance Considerations

Prima di fare alcuna considerazione sulle performance, ci tengo a ricordare che l'ottimizzazione prematura è la radice di tutti i mali. Prima di intraprendere azioni per ottimizzare le performance, andrebbe identificato un reale problema di performance attraverso il profiling. Detto questo, dal momento che di default la rigenerazione passiva innalza gli Entity Healed Events (global ed extra), è importante considerare l'impatto sulle performance qualora si abbiano un gran numero di entità che rigenerano vita contemporaneamente con un intervallo di rigenerazione molto breve. Se ci dovessero essere anche svariati listener sottoscritti al global Entity Healed Event, si otterrebbero svariate chiamate di funzione ad ogni tick di rigenerazione X ogni entita'. Se questo dovesse diventare un problema per le performance del vostro gioco, avete a disposizione diverse leve per ottimizzare le performance:

- **Aumentare l'intervallo di rigenerazione**: aumentando l'intervallo di rigenerazione, si riduce il numero di tick di rigenerazione per unità di tempo, e quindi il numero di Entity Healed Events innalzati.
- **Disabilitare l'innalzamento di Entity Healed Events per la rigenerazione passiva**: se non avete bisogno di reagire alla rigenerazione passiva attraverso i listener degli Entity Healed Events, potete disabilitare l'innalzamento di questi eventi per la rigenerazione passiva attraverso la configurazione. In questo modo, anche se avrete un gran numero di entità che rigenerano vita

contemporaneamente con un intervallo di rigenerazione molto breve, non avrete un impatto significativo sulle performance dovuto all'innalzamento di eventi e alle chiamate di funzione associate.
- **Utilizzare gli Extra Entity Healed Event**: se avete bisogno di reagiere solo alla rigenerazione passiva di una o alcune specifiche entita', potete configurare queste entità per innalzare un Extra Entity Healed Event specifico per la rigenerazione passiva, e sottoscrivere i vostri listener a questo evento invece che al global Entity Healed Event. In questo modo, le uniche chiamate di funzione che avrete ad ogni tick, saranno quelle dei listener sottoscritti all'Extra Entity Healed Event specifico delle entita' configurate per innalzare questo evento, riducendo cosi' l'impatto sulle performance.

Ci tengo a ribadire che, a meno che non abbiate identificato un reale problema di performance attraverso il profiling, non è necessario intraprendere nessuna di queste azioni per ottimizzare le performance. Probabilmente non avrete alcun problema di performance anche con un migliaio di entita' e decine di listener sottoscritti al global Entity Healed Event.

Manual Health Regeneration

La rigenerazione manuale è invece triggerata manualmente attraverso il metodo `ManualHealthRegenerationTick()` delle API.

Questo metodo, quando chiamato, fa scattare un tick di rigenerazione per l'entità. Attenzione che la statistica considerata per il calcolo della vita NON coincide con quella configurata per la rigenerazione passiva, ma è una statistica a parte, sempre configurata attraverso la configurazione del package, per la precisione: Manual Health Regeneration Stat. Il valore di questa statistica determina la quantità di HP rigenerati ad ogni tick di rigenerazione manuale. Ovviamente, i modificatori di guarigione, sia generici che specifici per la heal source configurata per la rigenerazione, influenzeranno anche la rigenerazione manuale, proprio come per la rigenerazione passiva.

Passive vs Manual Health Regeneration: Which One Should I Use?

Se avete dubbi su quale tipo di rigenerazione utilizzare, ecco alcune linee guida che potrebbero aiutarvi nella scelta:

- **Real Time**: se il vostro gioco è in tempo reale, la rigenerazione passiva è generalmente più adatta, in quanto si integra meglio con il flusso di gioco continuo.
- **Turn-Based**: se il vostro gioco è a turni, la rigenerazione manuale è generalmente più adatta, in quanto potete farla scattare alla fine di ogni turno, garantendo un controllo più preciso sul flusso di gioco.
- **Regeneration in the base**: se il vostro gioco prevede una meccanica di rigenerazione che avviene solo quando il personaggio è in un luogo specifico (es. una base), potete utilizzare la rigenerazione passiva, ma attivarla solo quando il personaggio si trova in quel luogo specifico, e disattivarla quando esce da quel luogo. Se invece vi interessa averla attiva anche quando il personaggio è fuori

dalla base, ma volete comunque che sia più forte quando è in base, potete conferire un Heal Modifier specifico per la Heal Source della rigenerazione quando il personaggio è in base, in modo da aumentare la quantità di HP rigenerati quando il personaggio si trova in base. All'uscita dalla base, potete rimuovere questo Heal Modifier, in modo da ridurre la quantità di HP rigenerati quando il personaggio è fuori dalla base.
- **Regeneration out of combat**: se il vostro gioco prevede una meccanica di rigenerazione che avviene solo quando il personaggio è fuori dal combattimento, potete utilizzare la rigenerazione passiva, ma attivarla solo quando il personaggio è fuori dal combattimento, e disattivarla quando entra in combattimento. Se invece vi interessa averla attiva anche durante il combattimento, ma volete comunque che sia più forte quando il personaggio è fuori dal combattimento, potete, come per la rigenerazione in base, conferire un Heal Modifier specifico per la Heal Source della rigenerazione quando il personaggio è fuori dal combattimento.
- **Regeneration as a buff**: se l'entita' riceve un buff che fornisce rigenerazione HP secondo un timing particolare, potete utilizzare la rigenerazione manuale, in modo da far scattare i tick di rigenerazione in corrispondenza del timing particolare previsto dal buff.

Chiaramente queste sono solo delle linee guida e non delle regole rigide. Per ogni problema ci sono sempre svariate soluzioni possibili, e la scelta della soluzione migliore dipende sempre dal contesto specifico del vostro gioco e dalle meccaniche che volete implementare. Quindi, se ritenete che una soluzione diversa da quelle suggerite sia più adatta al vostro gioco, sentitevi liberi di utilizzarla.

Lifesteal

Health Scaling Component

Resurrection

Advanced topics

Limitations

Requirements

- Astra RPG Framework
- Unity 6 or later

Package Contents

Samples

Installation instructions

Importing Astra RPG Health and its samples

1. From the package manager, import Astra RPG HEalth. You can find the package in the "My Assets" section.
2. After importing, head to the "In Project" section, always in the Package Manager, and click on "AstraRPGHealth".
3. Click on the "Samples" tab and import the samples you desire. Find out more about the samples in the Samples documentation.
4. If you imported the "Example scene and instances" samples you need to import also TextMeshPro Essentials. Click on "Window > TextMeshPro > Import TMP Essential Resources".

Changelog

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog.

[1.0.0] - 2025-10-30

Added

- Initial release of Astra RPG Health.

Migration Guide