MICROBAR

Table of Contents

MicroBar

Animated health bar framework

MicroBar is a comprehensive framework designed to simplify the creation of various types of bars in Unity.

Contact

Discord
Website
X
contact@microlightgames.com

Showcase video

Video showcasing presets bundled with the MicroBar package.
Trailer video previewing MicroBar.
YouTube tutorial

About

MicroBar is a framework designed to streamline the creation of progress bars in Unity. It provides a wide array of animation commands, enabling developers to create visually appealing and dynamic bars with just a few simple steps.

- Manages display of entity's health through visual health bars
- Powerful yet simple animation creation
- Based on the DOTween library, offering minimal performance impact
- Multiple templates to get you started
- Well-documented with examples
- Video tutorial for easy following
- Supports Image/UI and World/Sprite bars
- Fast support via Discord or Email
- contact@microlightgames.com

Setup & Installation

Dependencies

MicroBar uses DOTween library as its animation engine. Thus, the DOTween library is required.
You can get DOTween for FREE on the Asset Store.

Asset Store

MicroBar can be found on the Unity Asset Store

Raw download

MicroBar can also be downloaded from the GitLab repository.
Download the '**MicroBar**' folder and place it inside the '**Assets**' folder of your project.
Be aware of the `IfSeeDeleteThis` file located in the `Microlight` folder, as it may cause errors if you're unfamiliar with assembly definitions.

Tutorial

A video tutorial is available on YouTube - MicroBar Tutorial.
And a new tutorial video for the Simple mode YouTube - Simple mode tutorial

This README will also explain all concepts of MicroBar further in the document.

API

The API is minimalistic, allowing users to focus on important tasks. Initialize the health bar and then just update its values.

Methods	Description
`Initialize`	Initializes the bar, making it useable
`SetNewMaxHP`	Sets a new maximum HP for the bar
`UpdateBar`	Updates the health value of the bar
`ChangeMaxHealthCalculation`	Changes the way max HP change calculation is done
`SnapshotDefaultValues`	Stores current image values as default value

Properties	Description
`CurrentValue`	Returns current HP value
`MaxValue`	Returns HP max value
`HPPercent`	Returns health in 0-1 range where 1 is full
`HPPercent`	Returns health in 0-1 range where 1 is full

Events	Description
`OnMaxValueChange`	When new max HP is set
`OnCurrentValueChange`	When current HP changes

Image vs Sprite

An Image bar consists of UI components that appear on a canvas, such as Images. This is ideal for displaying the player's health on the screen at all times, or for showing constant updates during boss battles, like a dragon's health bar.

In contrast, a Sprite bar is made of world sprites, commonly used in 2D games (but also functional in 3D games). This type of bar is useful when you want to display a small health bar that follows a character's movement in the game. You can achieve a similar effect with an Image bar placed on a world canvas.

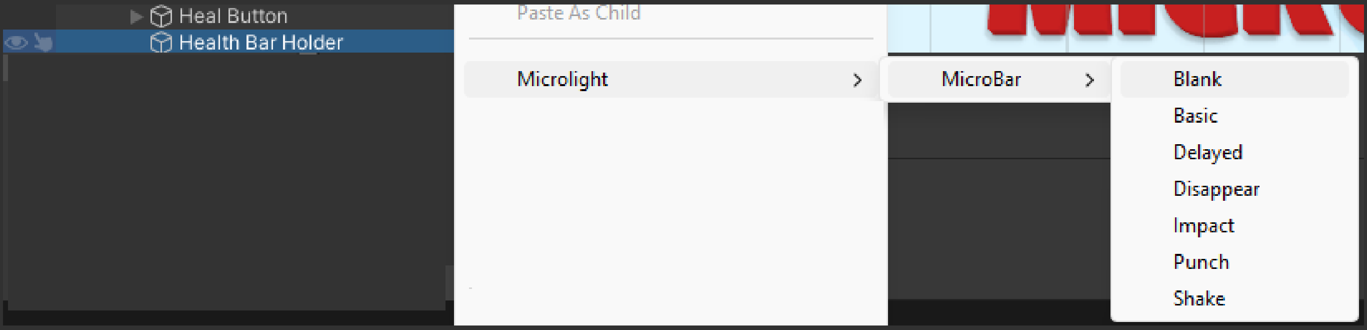
Max health calculation

The behavior of the health bar when max health changes is controlled by the public static property `MaxHealthCalculation`. This property affects all health bars, both friendly and enemy. You can modify its value using the `ChangeMaxHealthCalculation` method. By default, the setting is `FollowIncrease`. If you wish to change it, it's recommended to do so during the game's initialization process.
Available modes:

- Keep
- Follow
- FollowIncrease
- Proportional

`Keep` will retain the CurrentHP at its current value, regardless of changes to MaxHP. For example, if you increase the MaxHP of a 50/100 health bar to 150, the new values will be 50/150. The only rule is that CurrentHP cannot exceed MaxHP.
`Follow` adjusts CurrentHP in proportion to the change in MaxHP. For example, if you increase the MaxHP of a 50/100 health bar to 140, the new health bar will have values of 90/140. CurrentHP increases by 40 points because MaxHP has increased by 40 points.
`FollowIncrease` behaves like `Follow` when increasing MaxHP but acts like `Keep` when lowering it. For example, increasing the MaxHP of a 50/100 health bar to 160 will result in a 110/160 health bar. However, if you then lower the MaxHP to 130, the health bar will remain at 110/130.
`Proportional` maintains the ratio of CurrentHP to MaxHP. For example, if you increase the MaxHP of a 75/100 health bar to 200, the resulting health bar will be 150/200, since the original bar was 75% full, and the new one will be as well.

Spawning bars



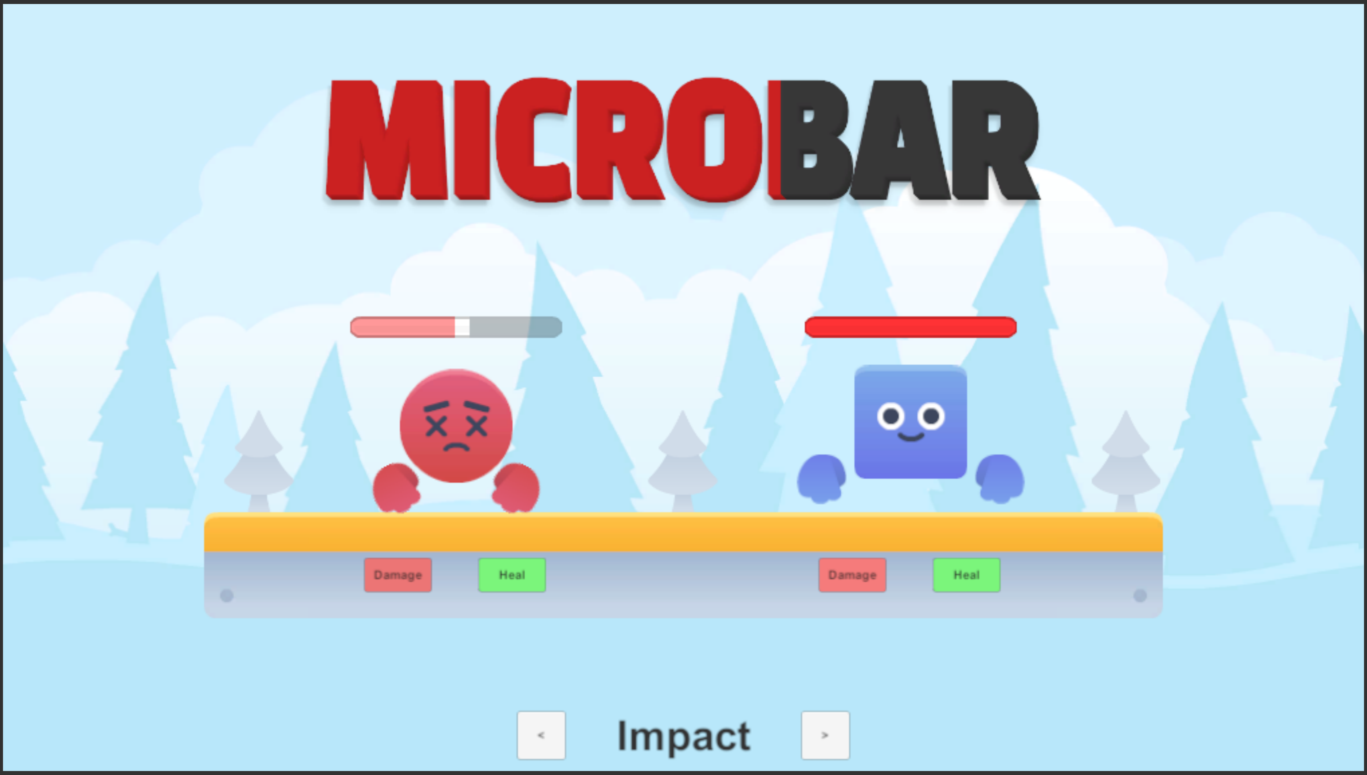
The easiest way to spawn a game-ready health bar is to `Right Click` > `Microlight` > `MicroBar` > and select one of the bar options. Experiment with the different choices to find the right health bar for your game.

One thing to note is that this menu is context-sensitive. If the menu is opened on a canvas, it offers Image/UI bars, but if it's opened on the world or a world object, it provides Sprite bars instead.

`MicroBar` also offers `Blank` canvas option. Spawning a `Blank` health bar leaves the animation empty, allowing you to create animations from scratch.

Alternatively, you can drag the prefabs from the `MicroBar` prefabs folder into the scene. It's recommended to unpack the prefab and create a new prefab for easier control of your health bars.

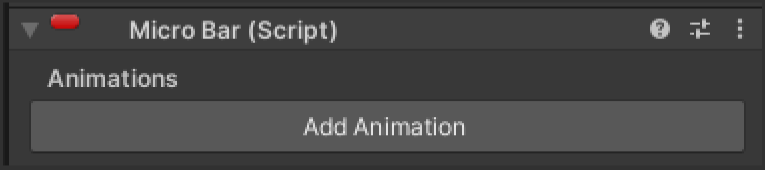
Demo



MicroBar includes a demo scene where you can test all the provided templates. Navigate to `Microlight` > `MicroBar` > `DemoScene` and open the `Demo` scene.

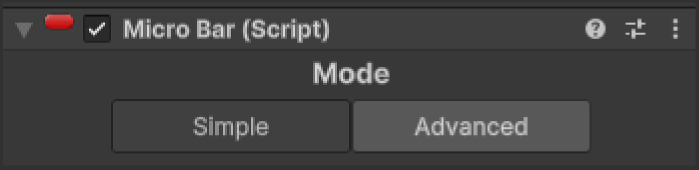
Components

MicroBar



While MicroBar simplifies many processes behind the scenes, understanding its system helps maximize its potential. The MicroBar component is the framework's core, holding all animations for the bar.

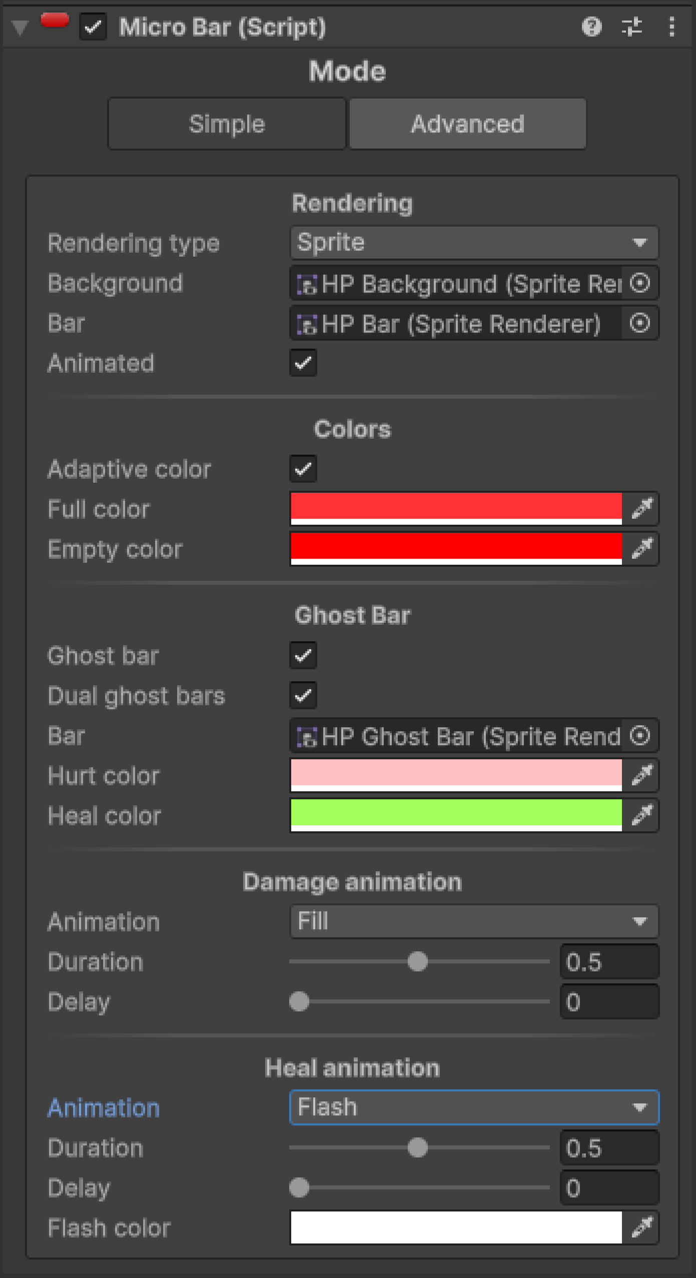
Editor mode



MicroBar provides two editor modes: `Simple` and `Advanced`. **Advanced** mode is the "normal" mode, available since the release of MicroBar v2. With version 2.5.0, the MicroBar v1 editor is back as **Simple** mode, featuring an improved implementation.

While **Advanced** mode offers greater flexibility and control over bar animations, it can sometimes feel overly complex for quick setups. When you need a straightforward, easy-to-configure bar, **Simple** mode is the ideal choice.

Simple Bar


Micro Bar (Script)

Mode

Simple | Advanced

Rendering

Rendering type	Sprite
Background	HP Background (Sprite Rei
Bar	HP Bar (Sprite Renderer)
Animated	✓

Colors

Adaptive color	✓
Full color	🟥
Empty color	🟥

Ghost Bar

Ghost bar	✓
Dual ghost bars	✓
Bar	HP Ghost Bar (Sprite Rend
Hurt color	▨
Heal color	🟩

Damage animation

Animation	Fill
Duration	0.5
Delay	0

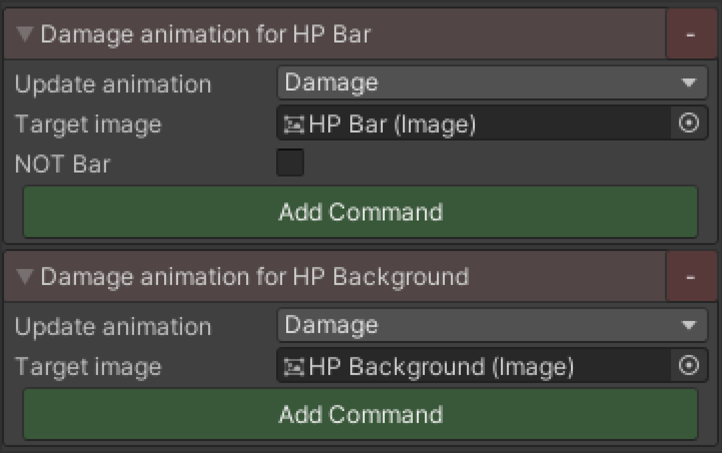
Heal animation

Animation	Flash
Duration	0.5
Delay	0
Flash color	⬜

- **Rendering type** defines whether the bar is a **UI image** or a **sprite**.
- **Background** and **Bar** are slots for the Image/SpriteRenderer components representing the respective parts of the bar.
- **Animated** specifies whether the bar will have animations; disabling this option will turn off various features like animations and ghost bars.
- **Adaptive color** allows the bar color to transition smoothly between **Full color** and **Empty color**. When disabled, the bar uses a single, static color set here.
- **Ghost bar** enables a secondary bar that stays behind when the main bar changes (e.g., when taking damage) and disappears or animates afterward.
- **Dual ghost bars** gives the ghost bar a different color based on whether the bar is decreasing (damage) or increasing (healing).
- **Animations** offer three primary options (Fill, Flash, and Shake), with an additional **None** option for no animation. These animations trigger on actions like damage or healing and have various adjustable parameters, such as **Duration**, **Delay**, and **Strength**, depending on the animation type.

For examples, explore the prebuilt bars in `MicroBar > Prefabs > SimpleBars`. If you're familiar with the **DOTween** library, you can also modify the `SimpleAnimBuilder.cs` script to add custom animations for your project.

MicroBar Animation


Damage animation for HP Bar
-
Add Command
Damage animation for HP Background
-
Add Command

Update animation: Damage
Target image: HP Bar (Image)
NOT Bar: ☐

Update animation: Damage
Target image: HP Background (Image)

MicroBar animation is the animated instance for an object. Each image needs its own animation, as shown above where `HP Bar` and `HP Background` each have their own animations.

Update animation

- Damage
- Heal
- CriticalDamage
- CriticalHeal
- Armor
- DOT
- HOT
- MaxHP
- Revive
- Death
- Custom

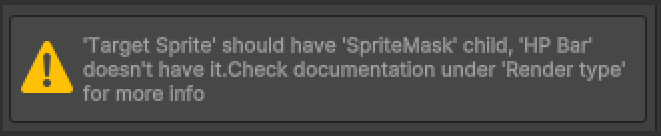
The `Update Animation` type defines when an animation will trigger. When updating the health bar value, you can specify the update type:

```
myMicroBar.UpdateBar(hp, UpdateAnim.Damage);
```

In this example, all animations of type `Damage` will trigger on `myMicroBar`. Additionally, the animation header in the editor changes color based on the animation type for better visibility.

Render type

The render type simply defines whether the animation is for a sprite or an image component, helping the system understand what it is working with.


'Target Sprite' should have 'SpriteMask' child, 'HP Bar' doesn't have it.Check documentation under 'Render type' for more info

In version 2.5.0, the rendering of SpriteRenderer bars has changed. The fill amount for sprite bars is no longer handled solely through the X scale of a bar. Instead, each bar that can be filled now has a child mask sprite, making sprite bars behave similarly to UI image bars. Bars without masks will continue to function due to **backwards compatibility**; however, they may exhibit unexpected behavior.

In the new structure:

- The bar should have a `SpriteRenderer` component with the **Mask Interaction** field set to **Visible Inside Mask**.
- The bar should also have a `Sorting Group` component, with the **Order in Layer** field set to the same value as the **Order in Layer** field of the `SpriteRenderer` component.

The bar's child GameObject will represent the **mask** for the bar:

- Add a `SpriteRenderer` component on this child object to serve as the mask. A **rectangle sprite** works best and should be slightly taller (in the Y axis) than the bar.
  - The **Order in Layer** should match that of the parent bar.
  - Set the **Color** to an alpha value of **0** (transparent).
- Add a `Sprite Mask` component to this child object, with the **Mask Source** field set to **Supported Renderers**.

For a template setup, refer to the `MicroBar > Prefabs > SpriteBars` folder and open any of the prefabs as a reference.

Note: These settings are provided for general use. Adjust them as needed for specific project requirements.

Target Image/Sprite

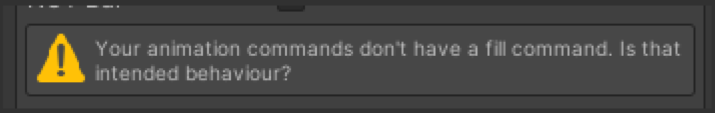
The Target Image or Target Sprite (depending on the render type) refers to the graphic that will be animated.

NOT Bar

If the render type is `Image`, the `NOT Bar` flag applies to images of type `Filled`. By default, every filled type image is treated as a bar. When skipping an animation, all images assumed to be bars will have their fill amount adjusted to the new health value. Enable the NOT Bar flag if you want to prevent an image from being affected when skipping animations.

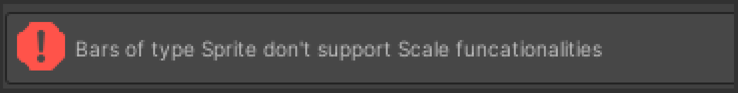
If the render type is `Sprite`, MicroBar will ask to include child object with `Sprite Mask` component. Please refer to the Render type section for more info on a new Sprite Renderer bar structure. Sprite graphics which are flagged as `NOT Bar` do not support `Fill` commands.

Fill warning


Your animation commands don't have a fill command. Is that intended behaviour?

When an image is considered a `bar` and your animation doesn't include a fill command, `MicroBar` will display a warning like this. This reminder suggests that you probably need a fill command. In rare cases, depending on your animation, this might not be true and this warning can be ignored.

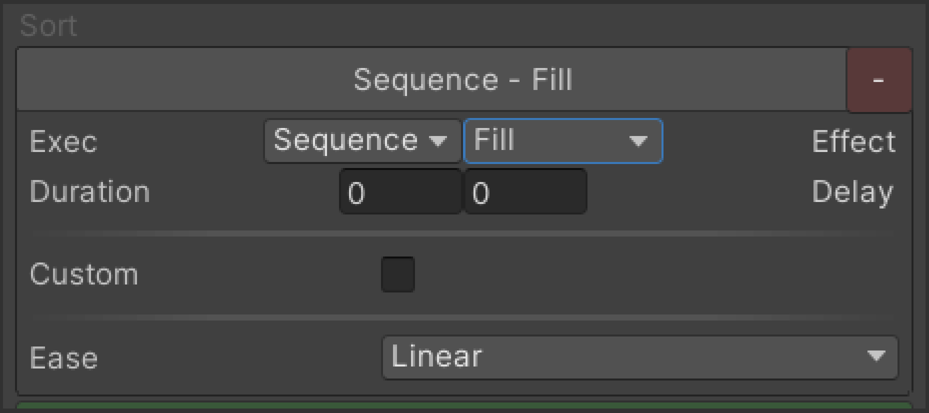
Unsupported commands


Bars of type Sprite don't support Scale funcationalities

When using `Sprite bars`, certain functionalities may not be supported, which could result in error messages. Although the game will not crash, it is recommended to remove any commands that trigger these errors to ensure stability and expected outcomes.

The commands most likely to cause such messages are those that attempt to alter the **anchored position** of the sprite, which is not applicable to sprites. Additionally, **sprite graphics** flagged as `NOT Bar` do not support `Fill` commands.

Commands



Exec: Sequence, Fill, Effect
Duration: 0, 0, Delay
Custom
Ease: Linear

Each animation consists of the list of commands. Commands define how will image behave over the course of the animation. There are many options while configuring but still very straightforward.

Execution

Execution (Exec) refers to the order of command execution based on the previous command.

- Sequence
- Parallel
- Wait

`Sequence` means the command will trigger when the previous command finishes.
A `Parallel` command will start when the previous `Sequence` command starts, excluding delay time.
`Wait` pauses for a specified time before proceeding to the next command.

Effects

An `Effect` is an instruction that dictates the image's behavior.

- `Color` changes the image's color
- `Fade` changes the alpha value of the image's color
- `Fill` changes the image's fill amount value
- `Move` changes the local position of the image's rect transform
- `Rotate` changes the z-axis value of the image's rect transform rotation
- `Scale` changes the local scale of the image's rect transform
- `Punch` vibrates one of the image's rect transform properties with decreasing strength
- `Shake` vibrates one of the image's rect transform properties with consistent strength
- `AnchorMove` changes the anchored position of the image's rect transform

* Scale is not supported in Sprite bars which are not flagged as NOT Bar * AnchorMove is not supported in Sprite bars

Duration and Delay

`Duration` defines how long the command lasts. `Delay` specifies the wait time before the command starts. In a `Parallel` command, timers start only when its parent `Sequence` command finishes its `Delay`.

Command values

Each command has various set of values that determine how will command be applied and the strength of the effect.

Value mode

- `Absolute` replaces the old value with the command value
- `Additive` adds the command value to the starting value
- `Multiplicative` multiplies the command value by the starting value
- `StartingValue` returns the property to its value at the start of the animation
- `DefaultValue` returns the property to its default value, stored when the object is created

The `StartingValue` can be volatile and may change, for instance, if you start an animation in the middle of another animation.
The `DefaultValue` always returns the image to its default values. This can be updated with the `SnapshotDefaultValue()` method, which stores the current values as the new default values.

Value types

Commands use different value types based on the context. For example, `Fade` in `Absolute` mode is a 0-1 slider, while in `Additive` mode, it's a float.

Axis

`Axis` is used to control image properties in two dimensions, like scale or position.

- `Uniform` modifies both axes equally
- `XY` allows separate control of each axis
- `X` controls only the X-axis
- `Y` controls only the Y-axis

Special Values

Fill

The `Fill` effect modifies the fill amount value of the image. By default, it adjusts to the current health value. Enabling the `Custom` flag allows manual control of the fill amount.

Punch and Shake

`Punch` and `Shake` effects use special values because of their increased complexity. Both effects can affect several image transform properties:

- `Position`
- `Rotation`
- `Scale`
- `AnchorPosition`

* AnchorPosition is not supported in Sprite bars

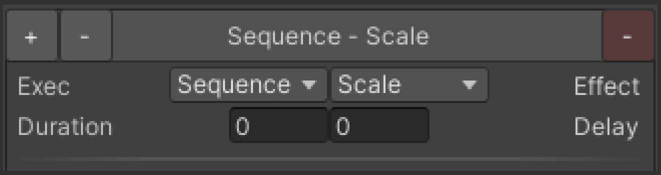
Both effects use `Frequency` (how erratic the movement is) and `Strength` (intensity of the effect). `Punch` also has an `Elasticity` value, allowing objects to exceed the strength value for a more dynamic effect.

Ease

`Ease` describes how the command behaves over time. The default is `Linear`, which is consistent throughout. Other options like `In Cubic` start weak and increase in strength. Experimenting with eases can enhance the visual appeal of your animations.

For visual representation of eases, visit this website.

Controls



Commands and animations have additional controls.

- The `Red button` with '-' icon deletes a command/animation
- The `Gray button` with '+' or '-' icon moves a command up/down in the list
- The `Header` of the animation allows for folding the animation for easier control

Much more

`MicroBar` isn't limited to health bars. You can use it for `mana` bars, `stamina` bars, `experience` bars, or any other type of bar such as `timers` or `goal` indicators. Your imagination is the only limit.

Have fun and showcase your creations on our Discord server where you can also ask for the help. You can also tag us (@microlightgames) in your posts on X, visit our X profile, or send us an email at contact@microlightgames.com. If you're also interested in our creations, beside Discord we also have a Website.