MicroBar

Animated health bar framework

MicroBar is a comprehensive framework designed to simplify the creation of health bars in Unity games.

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.

Setup & Installation

Asset Store

MicroBar can be found on the Unity Asset Store

Raw download

MicroBar can also be downloaded from the GitLab repository.
Download the '**Microlight**' folder and place it inside the '**Assets**' folder of your project.

Tutorial

A video tutorial is available on YouTube - MicroBar Tutorial.

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

About

MicroBar is a framework designed to simplify the creation of the health bars in Unity games. It offers a wide range of animation commands to create beautiful animated health bars with just a few clicks.

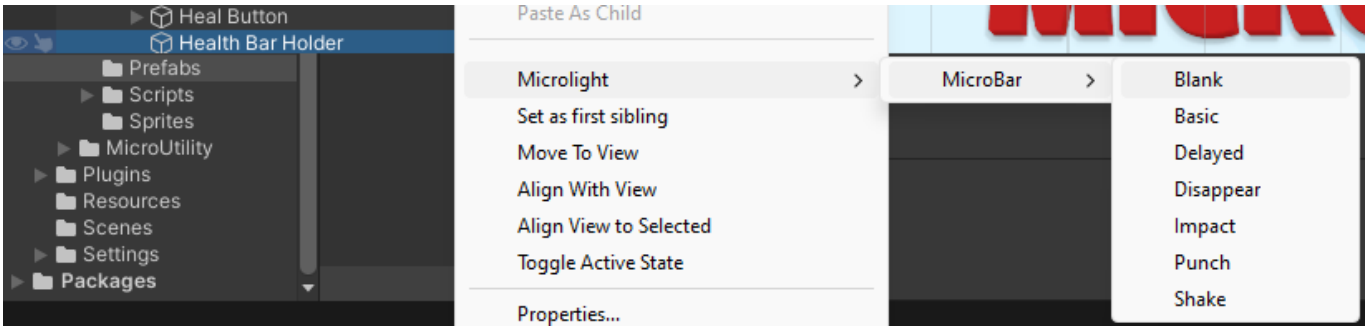
- Manages display of entity's health through visual health bars
- Powerful yet simple animation creation
- Based on the DOTween library, offering minimal performance impact
- Multiple templates to get you started
- Well-documented with examples
- Video tutorial for easy following
- Fast support via Discord or Email

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
`SnapshotDefaultValues`	Stores current image values as default value

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.

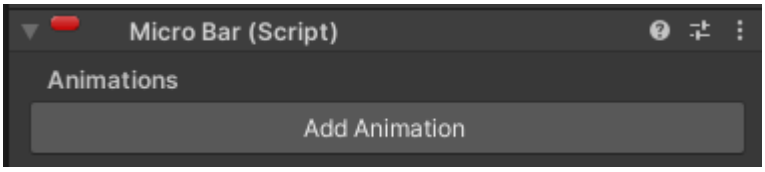
`MicroBar` also offers `Blank` canvas option. Spawning a `Blank` health bar leaves the animation empty, allowing you to create animations from scratch.

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

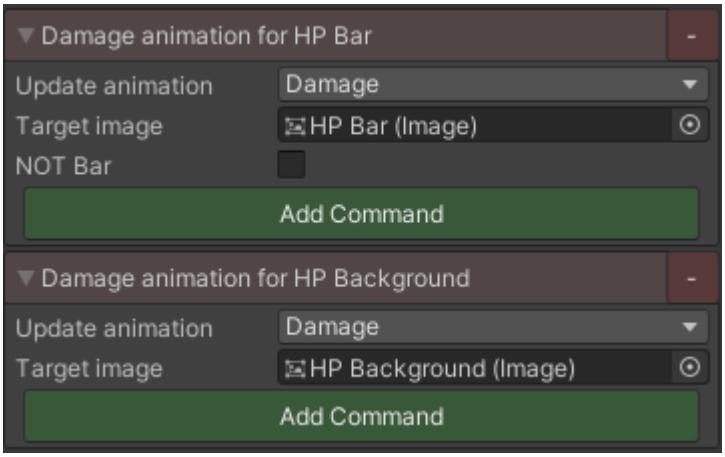
MicroBar includes a demo scene where you can test all the provided templates. Navigate to `Microlight` > `MicroBar` > `DemoScene` and open the `Demo` scene.

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.

MicroBar Animation



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.

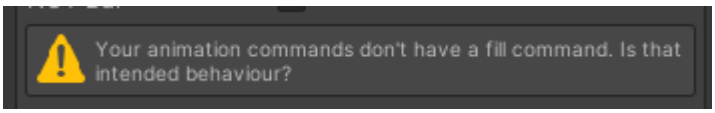
Target Image

Target image is the reference to the image which will be animated.

NOT Bar

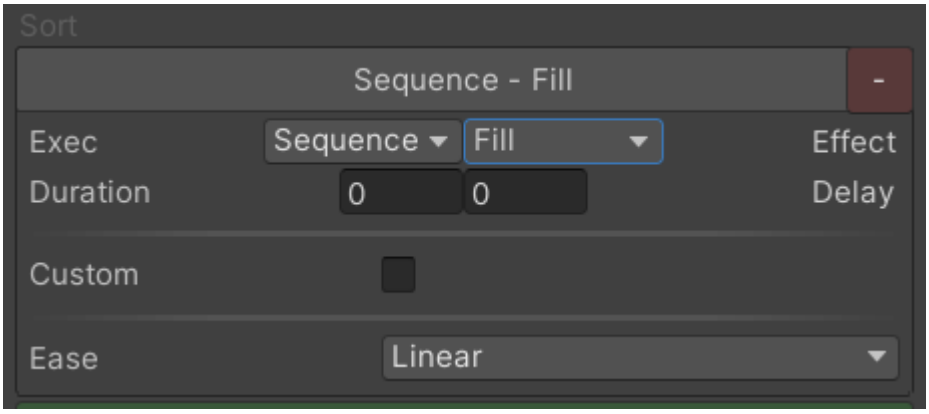
The `NOT Bar` flag is for images of type `Filled`. Every `Filled` type image is assumed to be a bar. When skipping an animation, all images assumed to be bars will have their fill amount adjusted to the new health value. Enable the `NOT Bar` flag if you don't want an image affected when skipping animations.

Fill warning



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.

Commands



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

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 existing value
- `Multiplicative` multiplies the command value by the existing 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`

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 microlightgames@gmail.com.