GAME ENGINE

Copyright (c) Pampel Games e.K. All Rights Reserved.

Support **Contact** **More Tools**

UNLOCKED DISCOUNTS

	
Spawn Machine	**AI Engine**

SUMMARY

GAME ENGINE
SUMMARY
INSTALLATION
    Requirements
    Installation
    Integrations
QUICK START
    General
    Example 1 - Volume Animation
    Example 2 - Cinemachine Shake
TIPS & TRICKS
    Play Settings
    Actions
    Modifiers
    Events
    Engine Systems
SAVE SYSTEM
BUILD
    Managed Code Stripping
API
    GameEngine.cs
    GameEngineSystem.cs
    GameEngineAPI.cs

INSTALLATION

Requirements

- Minimum Unity Version 2022.3 +

Dependencies

- unity.mathematics

Installation

If you have other tools from Pampel Games installed in your project, it is recommended that you update them to the latest version before importing the asset.

After importing, the 'PampelGames' folder can be moved to a different location within the 'Assets/' folder if desired.

Integrations

You can find all available integrations in the 'PampelGames/GameEngine/Integrations/' folder. Before installing an integration, ensure that you meet the required specifications, otherwise the Unity Editor will display errors.

Cinemachine

- Requires the “Cinemachine” package. You can find it under Window → Package Manager → Packages: Unity Registry → Cinemachine.

Built-in Volume

- Your project needs to be set up for the Built-In Render Pipeline and you have to install the Post Processing package. Please refer to the official Unity documentation for more information.

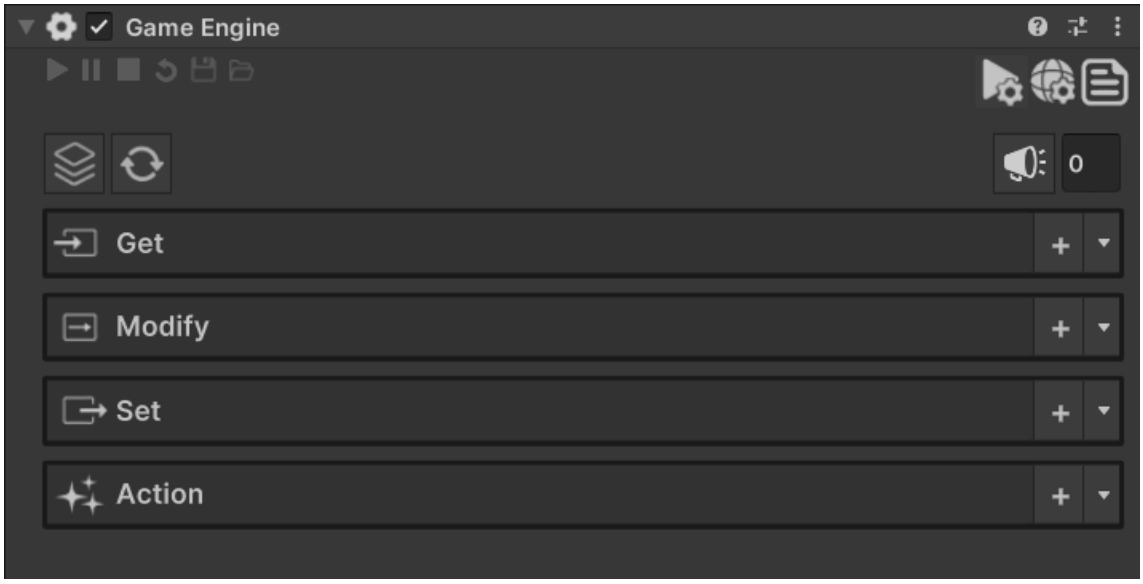
SRP Volume

- Your project needs to be set up for either the Universal Render Pipeline (URP) or the High Definition Render Pipeline (HDRP). Please refer to the official Unity documentation for more information.

QUICK START

General

To get started, simply add a GameEngine component to any GameObject you want.



The tool features a fully modular design, allowing you to choose from various modules using the dedicated buttons. It is built to create simple setups as quickly as possible, while also offering a wide range of possibilities.

The general workflow using Game Engine follows a top-down approach:

- In the 'Get' module, you retrieve a property or field, either manually or from a component.
- Optionally you modify the value. For example to add some randomness.
- You set the value into one or multiple components. That being said, you don’t have to set a value. Another use case would be to only read a value and send a Unity Event under certain conditions or spawn an effect.
- In the 'Actions' section, you determine how the value will be set. Without Actions, no values will be set.

All buttons and modules have detailed tooltips that explain their functionalities. If you are unsure about what a module does, you can simply add it first and then hover over it (or its variables) to get more information.

The setup may seem intimidating at first, but don’t worry. In just a few minutes, you'll be able to create your own components quickly and easily.

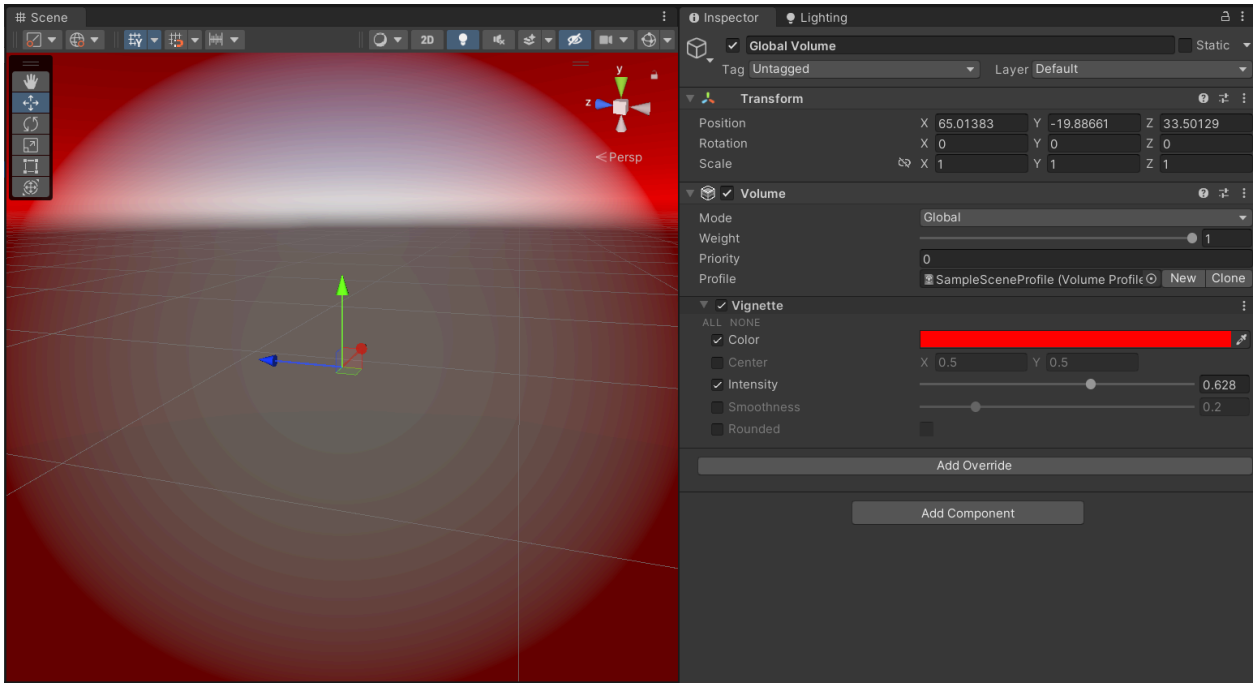
Example 1 - Volume Animation

In this example, we will animate a post-processing volume.

If you are using the Universal Render Pipeline or High Definition Render Pipeline in Unity, make sure to install the SRP Volume integration package found in the GameEngine/Integrations folder. For the Built-In render pipeline, install the Built-In Volume package.

- ➢ Add a volume to your scene (Right-click in the hierarchy - Volume - Global Volume).
- ➢ Create a new volume profile and add the vignette override to it. Configure the settings accordingly, ensuring the effect is visible in both the scene view and camera view by adjusting the intensity slider.

  If the effect is not visible in your camera view, you may need to activate the Post Processing option on the camera itself.



- ➢ Set the vignette intensity back to 0 while keeping it activated. This is the value we want to animate.
- ➢ Create a new GameObject in the hierarchy and name it ‘Vignette Animation’.
- ➢ Add a Game Engine component to it.
- ➢ Click on the small '+' sign on the right side of the Get section, then add the Manual module. Set the float value to 1.



- ➢ Click on the small '+' sign on the right side of the Set section, then add the Volume module. Tip: To open a second inspector, use the hotkey Ctrl + P.
- ➢ Specify the values according to this image (intensity is of type Float):



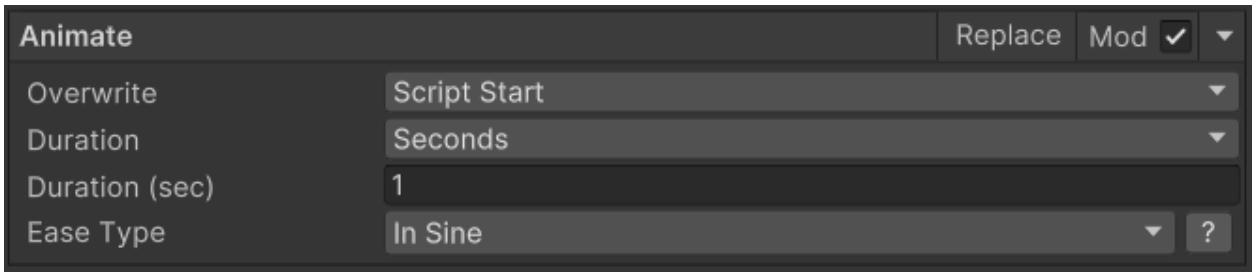
Now that we have specified the value and where we want to set it, we are almost done. However, if we were to execute the engine now, nothing would happen because it wouldn't know what to do with the value. This is where actions come into play.

- ➢ Within the Action section, add the Animate module.
- ➢ Set the Ease Type to Out Sine. Tip: If you click on the small '?' on the right side you will get an overview of the different easing functions.

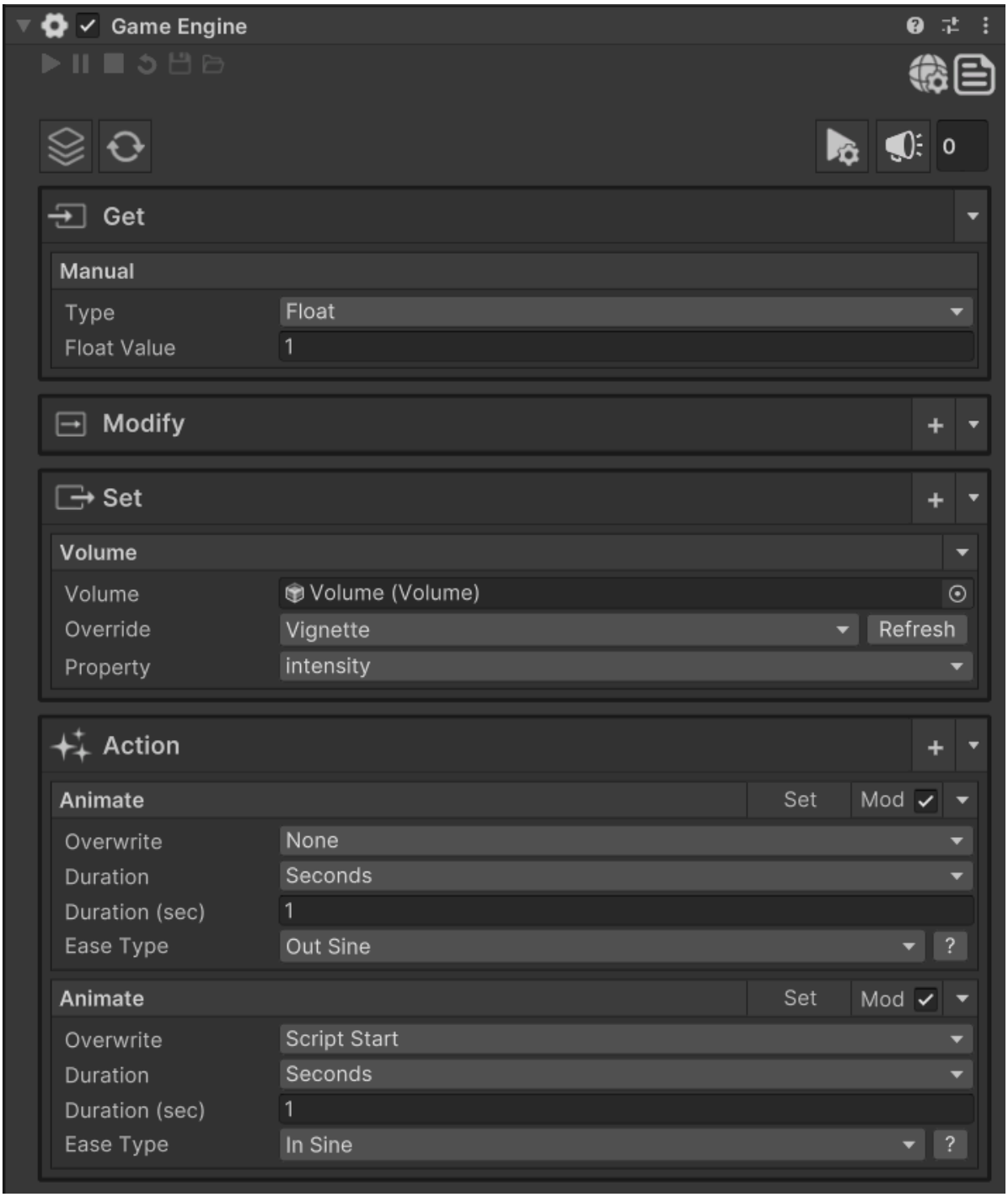


This action alone would animate the vignette intensity to 1 and leave it there. Instead, we want it to animate back on each execution.

- ➢ Within the Animate module, on the very right side (next to Mod), click on the small dropdown menu and choose Duplicate. Specify the values according to this image:



The full component should now look something like this:

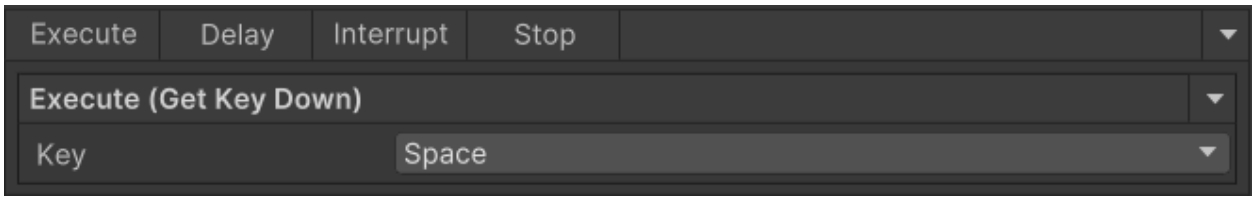


There is one final optional step:

Click the ‘Play Settings’ button at the top right. This will reveal a new toolbar at the top.

Now, when you hover over the Execute button, a tooltip will appear explaining the different ways you can execute the Game Engine.

- ➢ Click on the Execute button and choose Get Key Down.



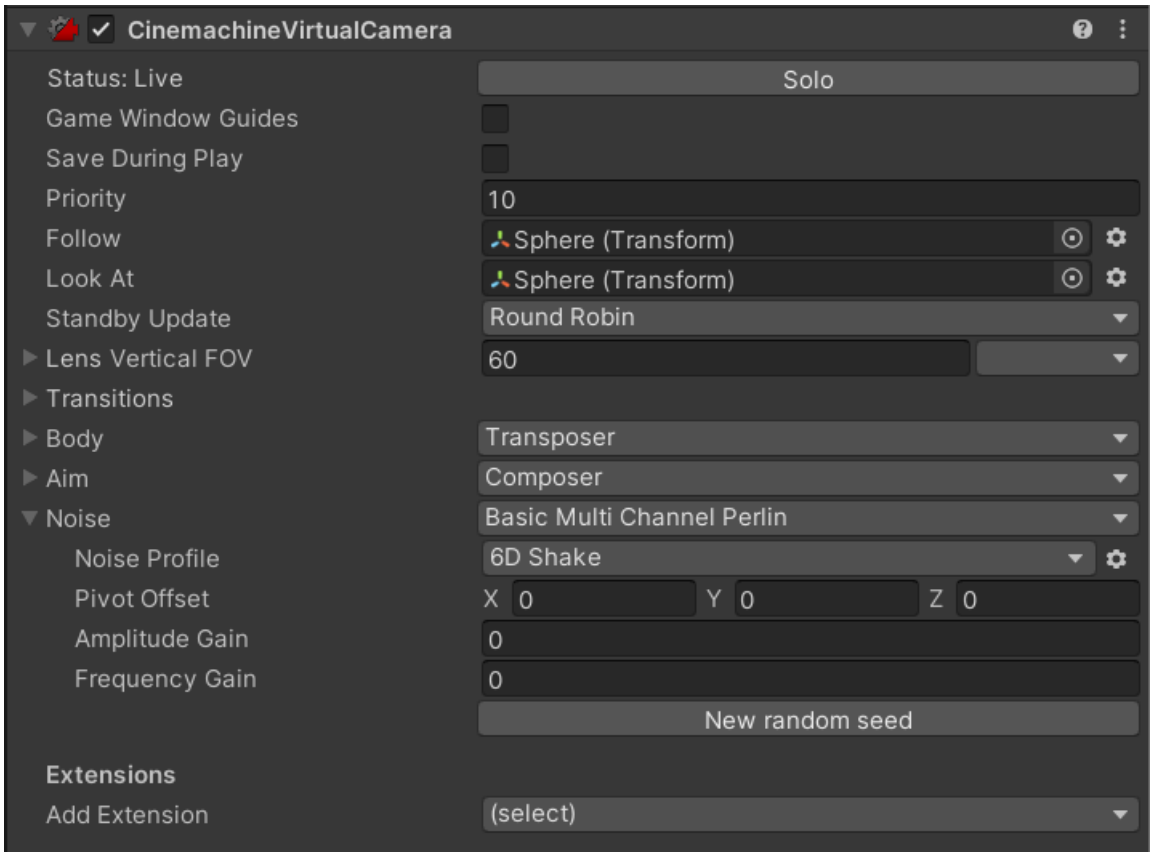
That's it! Enter play mode and press the spacebar to see the vignette intensity animate.

Example 2 - Cinemachine Shake

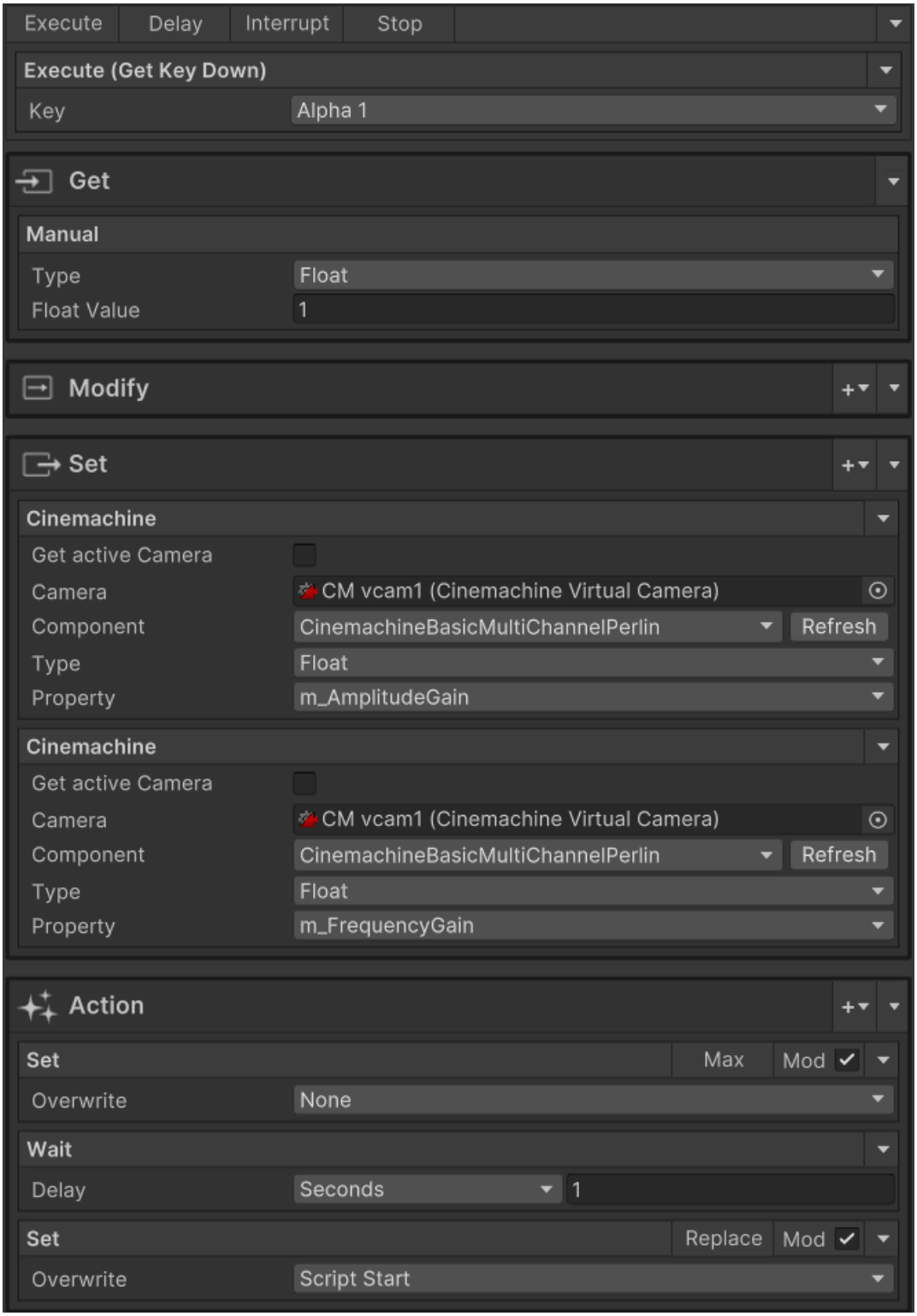
Note that this example is for Cinemachine version 2. In version 3 and later, it's even easier - you can simply use the Object module and assign the camera or Cinemachine components directly.

For the second example, please ensure you have the Unity Cinemachine package installed via the package manager. Then, import the Cinemachine integration package from the GameEngine/Integrations folder.

- ➢ Create a sphere in the hierarchy (Right-Click - 3D Object - Sphere). This will be the camera target.
- ➢ Create a cinemachine virtual camera via the top menu bar (GameObject - Cinemachine - Virtual Camera).
- ➢ Set the sphere as Follow and Look At target.
- ➢ Add the Basic Multi Channel Perlin noise and choose the 6D Shake profile.



- Create a new GameObject in the hierarchy and call it ‘Cinemachine Shake’.
- Add an Effect Engine component to it and set it up according to the following image. Remember, you can use the hotkey Ctrl + P to open a second inspector, which makes it easier to drag in a specific component.



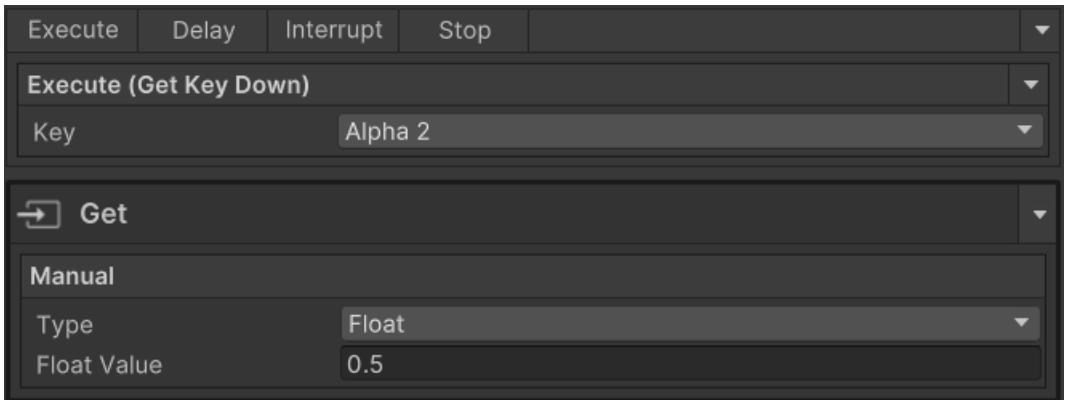
Please Note:

- Make sure to set the Alpha 1 key to start execution at the top.
- Change the value blending mode in the first Set Action from Replace to Max. You will soon see why.

Also note that we are using the Set action instead of the Shake action. This is because we are utilizing the integrated shake from Cinemachine. The Shake action could be used, for example, to shake the position of a transform component.

➢ Copy the ‘Cinemachine Shake’ GameObject in the hierarchy.

➢ Change the manual Get value to 0.5 and the key for execution to Alpha 2.



All right, that's it for the setup. When you hit play and press the 1 and/or 2 keys, you will see the camera shaking. Now you can see why the blending mode on Max could be helpful.

When the camera shakes with 0.5, the other component with 1 will overwrite it but not the other way around. This may be useful, for example, if you have small camera shakes on jumps but always want to be able to override those when a grenade is thrown - but not the other way around.

TIPS & TRICKS

Game Engine was designed first and foremost to help you move your projects forward quickly and efficiently.

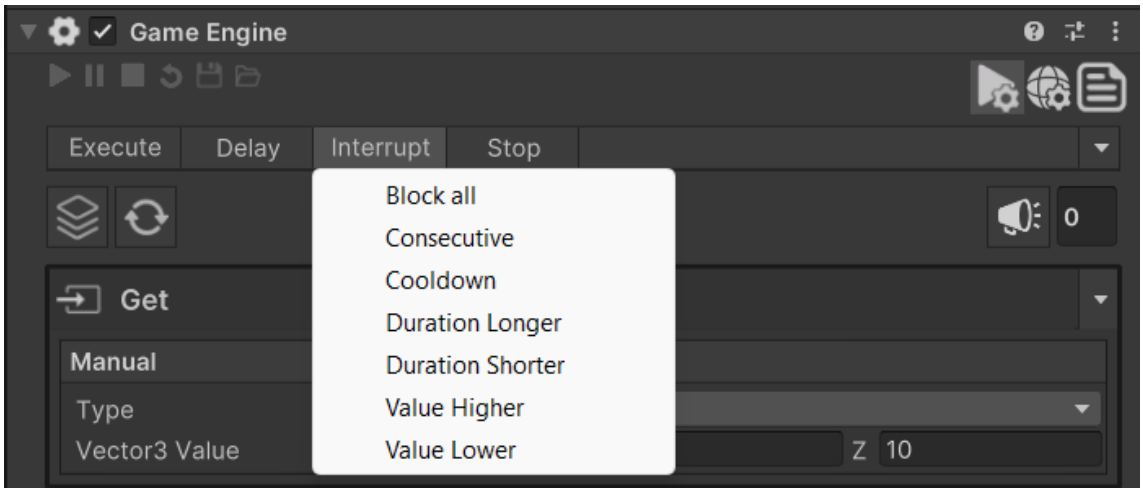
Here are a few features worth highlighting:

Play Settings

To execute (pause, stop, resume, etc.) GameEngine via script, you need a reference to the GameEngine component. You can then call gameEngine.Execute().

However, there may be situations where you want to prevent execution using interruption settings, or where it's more convenient to use the built-in Execute and Stop modules.

These modules can be found in the Play Settings by clicking the small arrow icon at the top right of the component.



Also, check out Demo02_Explosion, which demonstrates a practical example of using execute and interruption settings.

Actions

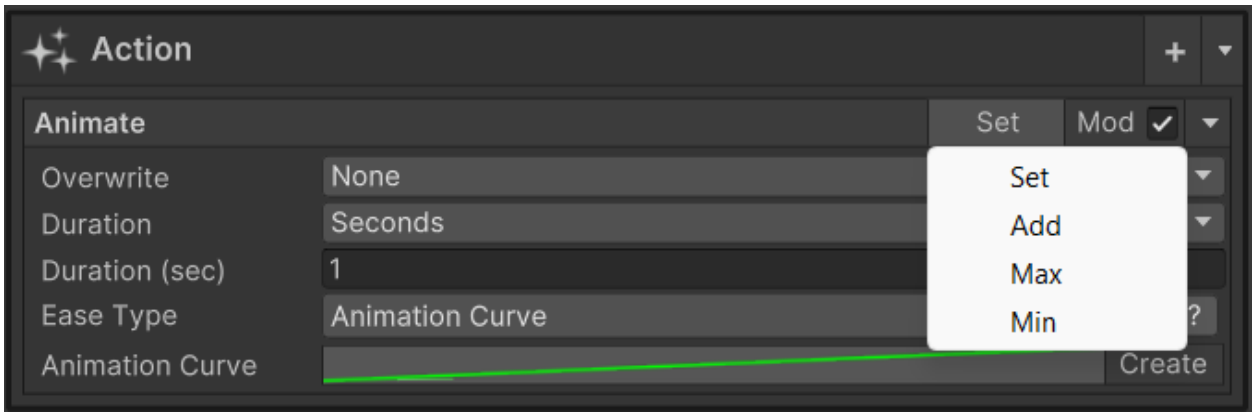
Actions can be added using the Action Toolbar located at the bottom of the GameEngine component:



Without any actions, GameEngine won't get or set any values. The simplest action is the Set action, which assigns the current get value to the specified Set objects in a single frame.

However, there are multiple actions available that you can combine to create a variety of effects quickly and easily.

Note that besides setting the value directly into an object, you have options to blend values differently, omit modifiers, or overwrite the value.



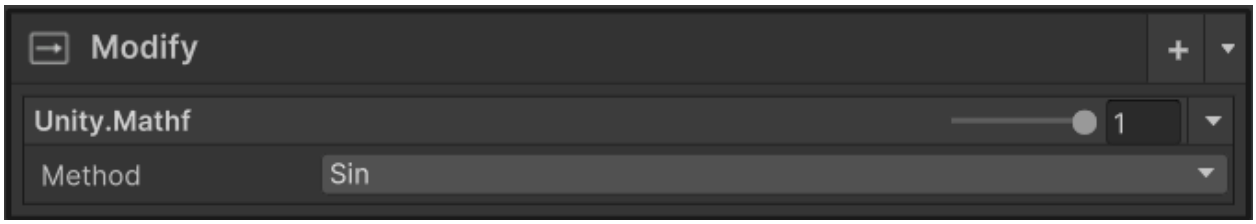
A common use case for overwriting the value is to reset an animated effect on a permanent scene component—for example, a camera shake (see the example in Demo02_Explosion).

Modifiers

Modifiers are optional and provide ways to alter incoming values before they are set (unless the action has the modifier option disabled). Like actions, modifiers can be stacked in any combination you choose.

Most use cases are covered by the included modifiers, which are designed to be generic and make use of namespaces.

For example, the UnityEngine.Mathf modifier offers all the single-parameter methods available in that namespace.



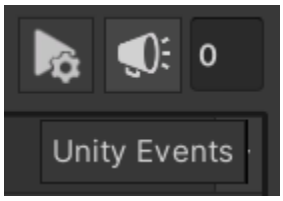
You can create your own modifiers by simply inheriting from ModifyBase.cs. Those are automatically added to the dropdown menu after clicking the Reload button at the top:



Or, if you feel something is missing, please don’t hesitate to reach out!

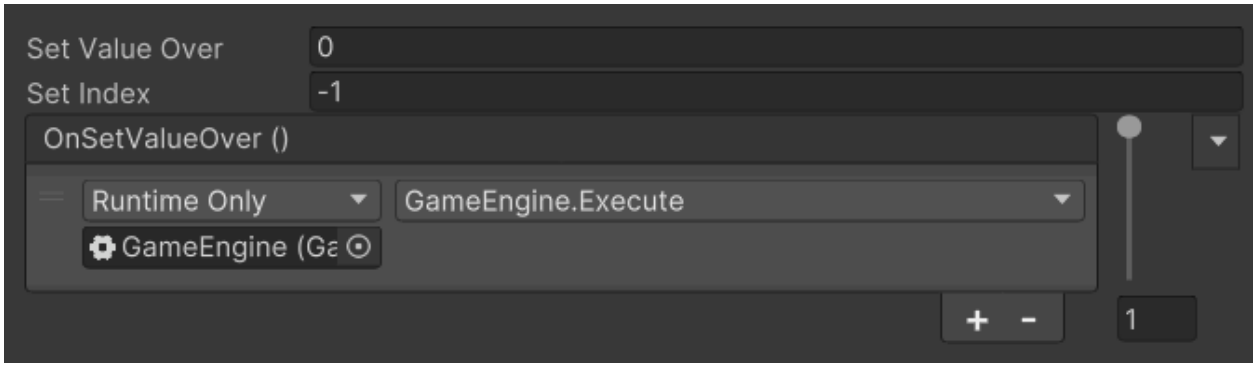
Events

Events are a great way to automate processes. You can add them by clicking the Event icon located at the top right:



After adding an event from the dropdown, it appears at the bottom of the component.

In addition to default events like OnExecutionStart, there are several other useful events that help you create more advanced logic. For example, the OnSetValueOver event is triggered only when the set value exceeds a specified threshold (Check out the Demo01_HealthBar for an example).



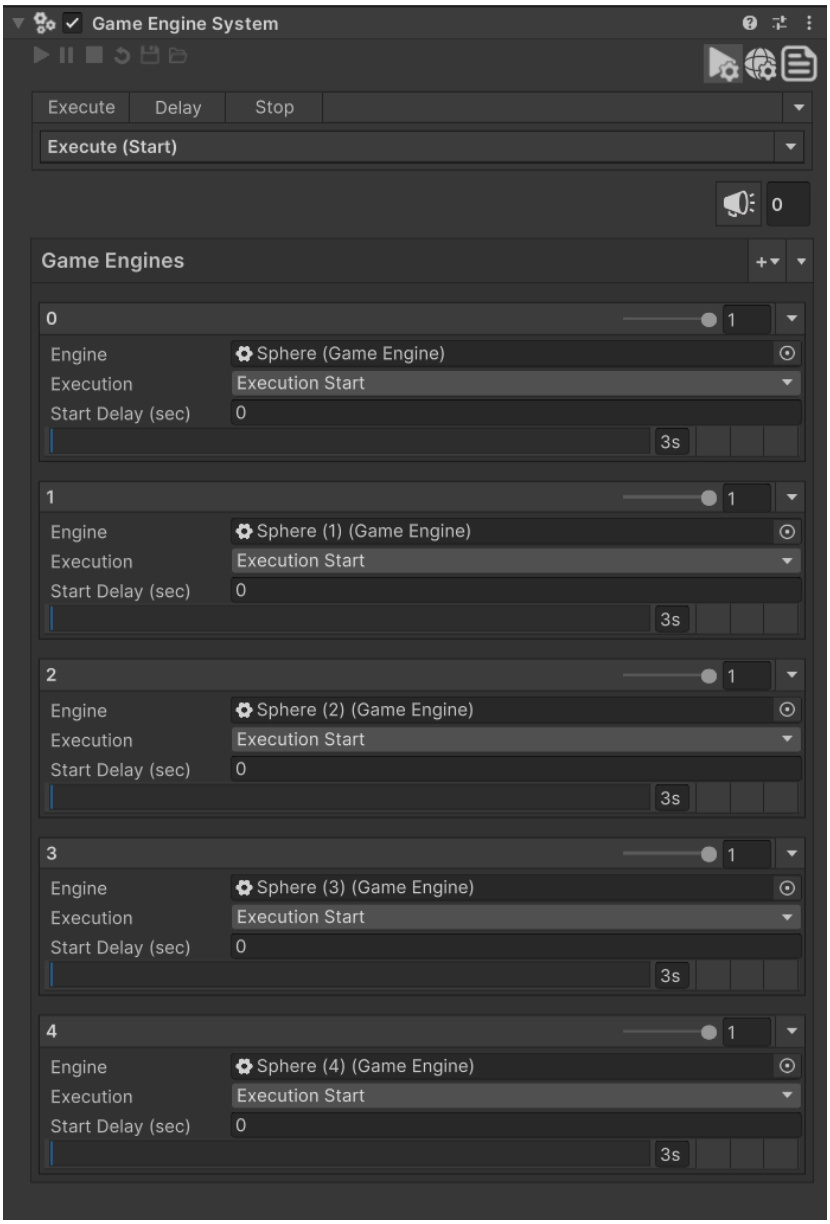
If you're unsure about what an event does, simply add it from the dropdown and hover over it to see a helpful tooltip.

Engine Systems

The GameEngineSystem component can be used to conveniently control multiple Game Engines from one component.

The recommended way to use it would be to add it to a GameObject and then have all the included Game Engines as children parented to it.

In the GameEngine/Demo/Demo03_System scene, you can find a ready-made system as an example.



SAVE SYSTEM

The Game Engine provides a convenient way to serialize and load the current state of actions, including the current get and set values.

All you need to do is call the following methods (see the API for details):

```
public override void Save(int slotIndex, string encryptionKey)

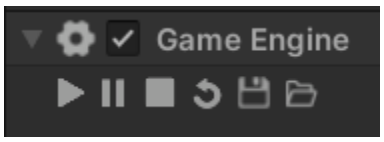
public override bool Load(int slotIndex, string encryptionKey)
```

The slotIndex must be unique for each GameEngine you save. The encryption key is optional and can be shared across multiple components.

Tip: You can use the instance ID of the GameObject to which the GameEngine is attached as the slotIndex:

```
gameObject.GetInstanceID()
```

At the top right of the GameEngine component, you'll find a small disk and folder icon. These can be used to test saving and loading directly from the Inspector (those are greyed out if not in play-mode).



BUILD

Game Engine supports all of the scripting backends Unity offers: Mono and IL2CPP.

Managed Code Stripping

If you enable managed code stripping in your Player Settings (disabled in Mono by default, can't be disabled in IL2CPP) you may have to exclude some of the assemblies you are using within your Game Engines.

Unfortunately it is not always clear if you have to exclude them. In fact, most of the time it is not necessary at all. Also IL2CPP is getting smarter with every Unity release. In the end, only a build will show since the Unity Editor play-mode does not work with C++ code stripping.

If you build your project with managed code stripping enabled and you run into errors that point to Game Engine - no worries, there is an easy fix.

Under "PampelGames/GameEngine/Scripts/ManagedCodeStripping" you find a link.xml file which already excludes the most used UnityEngine assemblies. Here you can remove the assemblies you don't need or add your own if needed.

Please also take a look in the Unity documentation for more information about the link.xml file.

Focussing on whole assemblies and not bothering with single methods or properties is absolutely fine in the majority of cases. Excluding all the heavy Unity assemblies in the example link.xml mentioned above increases the IL2CPP build size by around 10-15 Mb (stripped Mono even less) and has a very negligible impact on performance.

API

GameEngine.cs

Public Actions

```
public event Action OnExecutionCalled;
public event Action OnExecutionStart;
public event Action<EffectEngine> OnExecutionStartCustomEngine;
public event Action OnExecutionStop;
public event Action OnExecutionPause;
public event Action OnExecutionResume;
public event Action OnExecutionInterrupt;
public event Action OnExecutionPrevented;
public event Action<int> OnActionStart;
public event Action<int> OnActionStop;
public event Action<float> OnGetValueOver;
public event Action<float> OnGetValueUnder;
public event Action<float> OnSetValueOver;
public event Action<float> OnSetValueUnder;
public event Action<float> OnTimePassed;
public event Action<int> OnFramesPassed;
public event Action OnDestroyAction;
```

Public Methods

```
/// <summary>
///     Execute this <see cref="GameEngine" />.
/// </summary>
public void Execute()


/// <summary>
///     Execute this <see cref="GameEngine" /> with custom actions.
/// </summary>
/// <param name="customGameEngine">A Game Engine component that
contains a List of Actions to execute.</param>


/// <summary>
///     Execute this <see cref="GameEngine" /> and ignore any delay
set.
/// </summary>
public void ExecuteIgnoreDelay()


/// <summary>
///     Execute this <see cref="GameEngine" /> with custom actions and
ignore any delay set.
/// </summary>
/// <param name="customGameEngine">A Game Engine component that
contains a List of Actions to execute.</param>
public void ExecuteIgnoreDelay(GameEngine customGameEngine)


/// <summary>
///     Stops current execution loop immediately.
/// </summary>
public void Stop()


/// <summary>
///     Pauses current execution loop and stores the values for
resume.
/// </summary>
public void Pause()
```

```
/// <summary>
///     Resumes paused execution loop.
/// </summary>
public void Resume()


/// <summary>
///     Executes the next action from the list ignoring delay and
interruption mode.
/// </summary>
public void NextAction()


/// <summary>
///     Executes the previous action from the list ignoring delay and
interruption mode.
/// </summary>
public void PreviousAction()


/// <summary>
///     Executes the specified action from the list ignoring delay and
interruption mode.
/// </summary>
public void ExecuteActionIndex(int index, float startTime)


/// <summary>
///     Re-Initializes <see cref="PropertyGet" /> and <see
cref="PropertySet" />.
/// </summary>
public void ReInitialize()


/// <summary>
///     Destroy the GameObject this Game Engine is attached to.
/// </summary>
public void DestroyGameObject()


/// <summary>
///     Despawns all objects created by the <see cref="ActionSpawn" />
module within this component.
/// </summary>
public void DespawnObjects()
```

```
/// <summary>
///     Sets the value for the property getter in this <see
cref="GameEngine" />.
/// </summary>
/// <param name="value">The value to set to the property
getter.</param>
public void SetGetValue(object value)


/// <summary>
///     Assigns the specified value to the property set at the given
index.
/// </summary>
/// <param name="index">The index of the property within the <see
cref="propertySets" /> list to set.</param>
/// <param name="value">The value to assign.</param>
public void SetSetValue(int index, object value)


/// <summary>
///     Saves the current state of the <see cref="GameEngine" /> to
the specified save slot.
/// </summary>
/// <param name="slotIndex">
///     The index of the save slot where the state will be stored.
/// </param>
/// <param name="encryptionKey">
///     The encryption key used to secure the data being saved.
/// </param>
public override void Save(int slotIndex, string encryptionKey)


/// <summary>
///     Loads game data from the specified slot and executes the
loaded action index.
/// </summary>
/// <param name="slotIndex">The index of the save slot from which to
load data.</param>
/// <param name="encryptionKey">The encryption key used to decrypt the
saved data.</param>
/// <returns>True if the data was successfully loaded and applied;
otherwise, false.</returns>
public override bool Load(int slotIndex, string encryptionKey)
```

```
/// <summary>
///     Loads game data from the specified slot without executing.
/// </summary>
/// <param name="slotIndex">The index of the save slot from which to
load data.</param>
/// <param name="encryptionKey">The encryption key used to decrypt the
saved data.</param>
/// <returns>True if the data was successfully loaded and applied;
otherwise, false.</returns>
public bool LoadDirty(int slotIndex, string encryptionKey)


Public - Info

/// <summary>
///     Returns true if an execution is currently running.
/// </summary>
public bool IsExecuting()


/// <summary>
///     Returns true if an execution is currently in Pause mode.
/// </summary>
public bool IsPaused()


/// <summary>
///     Returns the index of the action that is currently playing.
Returns -1 if no action is playing.
/// </summary>
public int CurrentActionIndex()


/// <summary>
///     Returns the current value of the Property Get module as object
value.
/// </summary>
public object GetCurrentGetValue()


/// <summary>
///     Returns the current value of the Setter module with the
specified index as object value.
/// </summary>
public object GetCurrentSetValue(int index)
```

```
/// <summary>
///     Retrieves the current set values from the <see
cref="propertySets" /> list.
/// </summary>
/// <returns>A list of objects representing the current set
values.</returns>
public List<object> GetCurrentSetValues()


/// <summary>
///     Total time it takes for one execution.
/// </summary>
public float TotalExecutionTime()


/// <summary>
///     Time in seconds since the last execution started.
/// </summary>
public float TimeSinceExecutionStart(bool includePauseTime = false)


/// <summary>
///     Time in seconds since the last action started.
/// </summary>
public float TimeSinceActionStart(bool includePauseTime = false)


/// <summary>
///     Time in seconds since the first action started.
///     This is similar to TimeSinceExecutionStart but resets if the
Repeat action is used.
/// </summary>
public float TimeSinceFirstActionStart(bool includePauseTime = false)


/// <summary>
///     Amount of frames since the last execution started.
/// </summary>
public float FramesSinceExecutionStart(bool includePauseFrames =
false)
```

GameEngineSystem.cs

Public Actions

```
public event Action OnExecutionCalled = delegate { };
public event Action OnExecutionStart = delegate { };
public event Action OnExecutionStop = delegate { };
public event Action OnExecutionPause = delegate { };
public event Action OnExecutionResume = delegate { };
public event Action OnDestroyAction = delegate { };
```

Public Methods

```
/// <summary>
///     Execute this <see cref="EffectEngineSystem"/>.
/// </summary>
public void Execute()


public void Pause()


public void Resume()


public void Stop()


public void ReInitialize()


public bool IsExecuting()


public bool IsPaused()
```

GameEngineAPI.cs

```
/// <summary>
///     Executes all <see cref="GameEngine" />s.
/// </summary>
public static void ExecuteAll()


/// <summary>
///     Executes all <see cref="GameEngine" />s that share the
specified group identifier.
/// </summary>
public static void ExecuteGroup(int groupIdentifier)


/// <summary>
///     Stops all <see cref="GameEngine" />s.
/// </summary>
public static void StopAll()


/// <summary>
///     Stops all <see cref="GameEngine" />s that share the specified
group identifier.
/// </summary>
public static void StopGroup(int groupIdentifier)


/// <summary>
///     Pauses all <see cref="GameEngine" />s.
/// </summary>
public static void PauseAll()


/// <summary>
///     Pauses all <see cref="GameEngine" />s that share the specified
group identifier.
/// </summary>
public static void PauseGroup(int groupIdentifier)
```

```
/// <summary>
///     Resumes all <see cref="GameEngine" />s.
/// </summary>
public static void ResumeAll()


/// <summary>
///     Resumes all <see cref="GameEngine" />s that share the
specified group identifier.
/// </summary>
public static void ResumeGroup(int groupIdentifier)


/// <summary>
///     Destroys all <see cref="GameEngine" /> GameObjects.
/// </summary>
public static void DestroyAll()


/// <summary>
///     Destroys all <see cref="GameEngine" /> GameObjects that share
the specified group identifier.
/// </summary>
public static void DestroyGroup(int groupIdentifier)


/// <summary>
///     Returns the total number of active Game Engines in the
hierarchy.
/// </summary>
public static int ActiveGameEngines()


/// <summary>
///     Returns the total number of Game Engines that are currently
being executed.
/// </summary>
public static int ExecutingGameEngines()
```

```
/// <summary>
///     Returns the total number of paused Game Engines in the
hierarchy.
/// </summary>
public static int PausedGameEngines()


/// <summary>
///     Despawns all objects created by the <see cref="ActionSpawn"/>
module.
/// </summary>
public static void DespawnAllObjects()


// Game Engine System

/// <summary>
///     Executes all <see cref="GameEngineSystem" />s.
/// </summary>
public static void ExecuteAllSystems()


/// <summary>
///     Executes all <see cref="GameEngineSystem" />s that share the
specified group identifier.
/// </summary>
public static void ExecuteAllSystemGroups(int groupIdentifier)


/// <summary>
///     Stops all <see cref="GameEngineSystem" />s.
/// </summary>
public static void StopAllSystems()


/// <summary>
///     Stops all <see cref="GameEngineSystem" />s that share the
specified group identifier.
/// </summary>
public static void StopAllSystemGroups(int groupIdentifier)
```

```
/// <summary>
///     Pauses all <see cref="GameEngineSystem" />s.
/// </summary>
public static void PauseAllSystems()


/// <summary>
///     Pauses all <see cref="GameEngineSystem" />s that share the
specified group identifier.
/// </summary>
public static void PauseAllSystemGroups(int groupIdentifier)


/// <summary>
///     Resumes all <see cref="GameEngineSystem" />s.
/// </summary>
public static void ResumeAllSystems()


/// <summary>
///     Resumes all <see cref="GameEngineSystem" />s that share the
specified group identifier.
/// </summary>
public static void ResumeAllSystemGroups(int groupIdentifier)


/// <summary>
///     Destroys all <see cref="GameEngineSystem" /> GameObjects.
/// </summary>
public static void DestroyAllSystems()


/// <summary>
///     Destroys all <see cref="GameEngineSystem" /> GameObjects that
share the specified group identifier.
/// </summary>
public static void DestroyAllSystemGroups(int groupIdentifier)


/// <summary>
///     Returns the total number of active Game Engine Systems in the
hierarchy.
/// </summary>
public static int ActiveGameEnginesSystems()
```