Skinetic SDK

Unity Plugin

Documentation

Date	Ver.	Ref.	Description	Author	Approved by
14/12/2022	1.0	SSUPDO010	Initial version	de Tinguy	Begeot
11/01/2023	1.1	SSUPDO011	Package Manager	de Tinguy	Begeot
14/04/2023	1.2	SSUPDO012	Boost and Spatialization	de Tinguy	Begeot

Table of Content

1. Objective

The objective of this document is to provide a comprehensive overview as well as a quickstart for the **Skinetic SDK** plugin for *Unity 2019.4* and above.
The examples were tested on Windows:

- 2019.4
- 2020.3
- 2021.3
- 2022.2

As for Android, currently the version supported are:

- 2020.3.14 and above
- 2021.1.16 and above
- 2022

2. Import and prerequisite

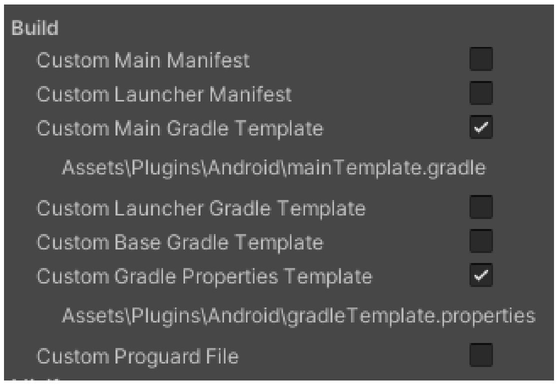
To import the **Skinetic SDK** package in any project, an archive is provided that you can decompress anywhere on your computer. Then you can manually import it using Unity's Package Manager using **Windows → Package Manager.** Then click on + → **Add package from disk...** and select the folder SkineticSDK.
You can also directly add this folder into the Packages folder of your project which will add it automatically to your project as an **embedded package** (More details on Unity's documentation).

Building for Windows:

In order to work properly on Windows **x64**, the **Microsoft Visual C++ Redistribuable** need to be installed on the computer. They are often already available as various softwares depend on it. If they are missing from the computer, they can be obtained at:
https://learn.microsoft.com/en-us/cpp/windows/latest-supported-vc-redist?view=msvc-170

Building for Android:

In order to work properly on Android devices, some options need to be set in the Unity project's settings:



Build
Custom Main Manifest ☐
Custom Launcher Manifest ☐
Custom Main Gradle Template ✓
Assets\Plugins\Android\mainTemplate.gradle
Custom Launcher Gradle Template ☐
Custom Base Gradle Template ☐
Custom Gradle Properties Template ✓
Assets\Plugins\Android\gradleTemplate.properties
Custom Proguard File ☐

- Under **Project settings** > **Player** > **Publishing Settings** > **Build**
  - Enable '**Custom Main Gradle Template**'
  - Enable '**Custom Gradle Properties Template**'

- In the Asset folder, two files are automatically added in the folder **Asset/Plugins/Android/**
  - Edit the **mainTemplate.gradle** file
  - Add in the following line in the **dependencies** section:
    implementation 'androidx.appcompat:appcompat:1.5.0'
  - Edit the **gradleTemplate.properties** file
  - Add in the following lines at the end:
    android.useAndroidX=true
    android.enableJetifier=true

If the settings are not correctly set, the bluetooth will always fail to find or connect to any devices.

3. Plugin General Overview

This plugin is designed to provide easy access to the **Skinetic SDK** inside *Unity* for both Android and Windows. When compiling your project, the corresponding platform is automatically enabled, no change is needed to handle the haptic layer.

The plugin relies on several key scripts:

- **SkineticDevice Component**
  A custom component to handle an instance of a Skinetic device.

- **PatternAsset ScriptableObject**
  A custom ScriptableObject containing a Skinetic haptic pattern (spn).

- **HapticEffect Component**
  A custom component representing an instance of the referenced pattern that can be configured to specify the haptic effect's rendering behavior.

These components are all using the **Skinetic** namespace.

4. SkineticDevice Component

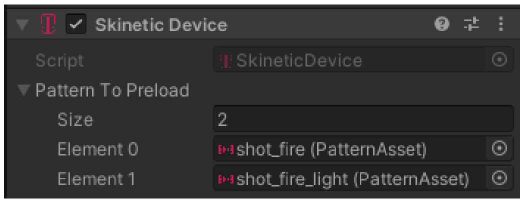
To enable haptic feedback using the **Skinetic SDK**, the first step is to set up a **Skinetic** device to connect to.
This can be done by creating a new *Data Asset* of class "**Skinetic Device**" (in *Miscellaneous/Data Asset*).

4.1. SkineticDevice component overview

The SkineticDevice component serves as the interface to the haptic rendering layer. This SkineticDevice component also abstracts the target platform of your project: developing an application for android, is the same as for windows.

The SkineticDevice component allows you to perform several kind of actions:

- Scan available devices
- Handle connection to a Skinetic device
- Get the current device information
- Control the device's global rendering state
- Handle the Global boost
- Handle haptic feedback



Script	SkineticDevice
Pattern To Preload	
Size	2
Element 0	shot_fire (PatternAsset)
Element 1	shot_fire_light (PatternAsset)

The last point will be detailed in section 5.

The SkineticDevice component can be added to a gameobject from the menu following **Skinetic → Skinetic Device.**

The connection to a Skinetic device can be established through usb, bluetooth or wifi. Once connected, the communication channel has no impact on the commands that can be given through this component.

4.2. Scanning procedure

A Skinetic device is uniquely identified by its serial number, independently of the connection protocol. To connect to a specific Skinetic device, the serial number is required. It can be obtained by scanning the available devices.

The SkineticDevice component provides 3 methods to handle the asynchronous scanning procedure:

- `ScanDevices` will start an asynchronous scanning procedure for all devices available with the connection type requested.

- `ScanStatus` returns the status of the scanning procedure or an error code.
- `GetScannedDevices` returns a list of devices to which it is possible to connect with, once the scanning procedure is complete.

The scanning procedure returns the list of all available devices while providing for each one: the supported output type (type of connection), the serial number, the device's type and the version of the embedded software. The same device can appear several times in the list if it is available through different connection protocols.

Note: The scanning procedure cannot be performed if the component is already connected.

4.3. Connection handling

The **SkineticDevice** component displays means to manage the connection to an actual Skinetic device. The connection procedure is also asynchronous. To connect to a specific device, its serial number is required. However, it is possible to also connect to any skinetic device available.

The **SkineticDevice** component provides 4 methods to handle the asynchronous scanning procedure:

- `Connect` will start the asynchronous connection procedure.
- `Disconnect` will start the asynchronous disconnection routine. The disconnection is effective once all resources are released.
- `ConnectionStatus` returns the current connection status to monitor the connection or disconnection procedures.
- `SetConnectionCallback` Set a callback function fired upon connection changes. The callback is triggered at the end of the connection or disconnection procedure.

Note: The connection cannot be performed if the scanning procedure is ongoing.

Note: Setting a callback is an alternative to monitoring the connection state, but is not mandatory. The callback is not executed by the main thread. Hence, the actions that can be performed by it are restricted by Unity.

4.4. Device Introspection

From a **SkineticDevice** component that is already connected to a Skinetic device, several methods are provided to check or display device information such as SDK version (`GetSDKVersion`), Skinetic version (`GetDeviceVersion`), Skinetic device type (`GetDeviceType`) or Skinetic serial number (`GetDeviceSerialNumber`).

4.5. Global Rendering State

The rendering of the device can be controlled with `PauseAll`, `ResumeAll` and `StopAll`. These calls affect all playing effects on the device, by pausing/resuming them or even stop and discharge all of them , freeing all resources.

4.6. Global Boost Setting

A global parameter is exposed and can be set through scripting. The boost increases the overall intensity of all haptic effects. However, the higher the boost activation is, the more the haptic effects are degraded.
We recommend exposing this parameter to the end-users so they can fix the boost level themselves. Some users might set high values while more sensitive ones might disable it.

More information about the boost strategy can be found in the next section.

5. Haptic Feedback

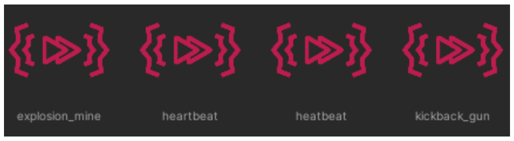
The haptic feedback is based on: 1. **Pattern Asset** ScriptableObjects that can handle *.spn* patterns. 2. **HapticEffect** components that can create and manage a parametric instance of the pattern.

5.1. PatternAsset

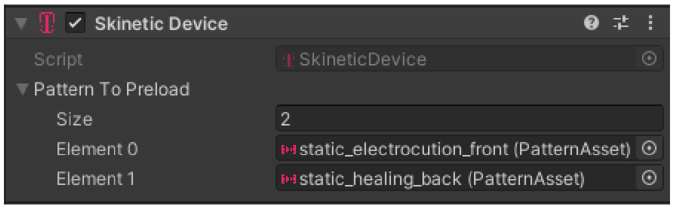
Patterns are descriptors of sequences of haptic samples that can be created using Unitouch Studio, while haptic effects are the actual feedback rendered using the sequence descriptor.

Every pattern created with **Unitouch Studio** can be used directly in *Unity*. Dragging the *.spn* file in the project window will automatically convert it to a **PatternAsset** ScriptableObject. These assets are then available for your project.

The PatternAsset has two fields: a Name that you can use to identify it and a json that the Skinetic SDK is able to process.



- **Load / Unload**
  All patterns to be played must be loaded beforehand into the connected device and unloaded once all interactions are completed (eg. while the game is stopping).
  A PatternAsset can be loaded and unloaded on the fly with the `LoadPattern` and `UnloadPattern` methods of the SkineticDevice component.
- **PreLoad**
  The SkineticDevice component also has a **PatternToPreload** public field which is a list of patterns that will be loaded during the **OnEnable** event.



Script: SkineticDevice
Pattern To Preload
Size: 2
Element 0: static_electrocution_front (PatternAsset)
Element 1: static_healing_back (PatternAsset)

- **LoadedPatterns**
  A read-only list of all loaded patterns is available on the SkineticDevice component.
- **SetAccumulationWindowToPattern**
  Enable the effect accumulation strategy: Whenever an effect is triggered on the primary pattern, the fallback one is used instead. More information can be found in the upcoming subsection.

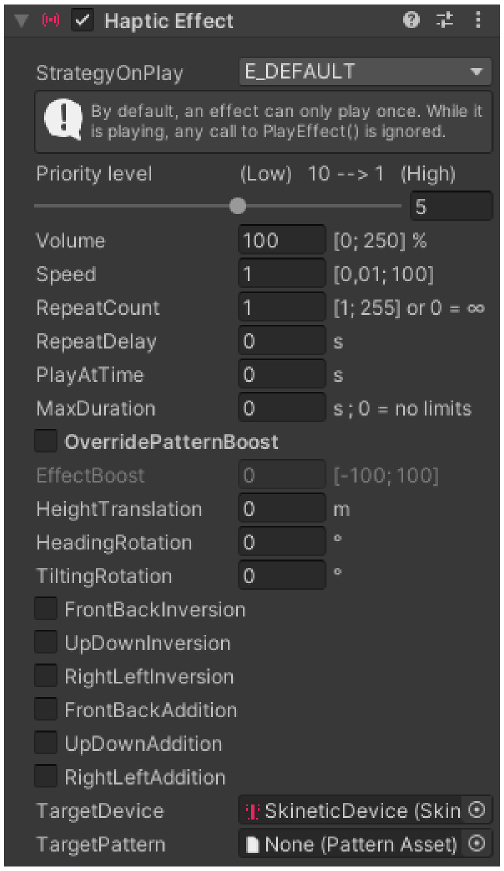
5.2. HapticEffect Component

HapticEffect components handle actual instances of PatternAssets. They provide several basic features:

- **Pattern Reference**
  In order to operate, a reference to a PatternAsset to play has to be set either in the inspector or through scripting under the field TargetPattern. An Haptic Effect is an instance of a Pattern.

- **StrategyOnPlay**
  When the effect is triggered, several strategies are available:
  - **Default:** By default, an effect can only play once. While it is playing, any call to PlayEffect() is ignored.
  - **Forced:** Immediately stop the current playing instance and start a new one.
  - **Pulled:** Each time the effect is triggered a new independent instance of the effect starts.

- **Play Parameters**
  Several parameters are available to modify the way the effect will be rendered when triggered. These parameters can be set in the inspector or through scripting.
  - **Priority**: Level of priority of the effect (from 1 to 10, 1 being the highest priority). The next section will go into details about the priority system.
  - **Volume**: percentage of the base volume between [0; 250]%.
  - **Speed**: time scale between [0.01; 100]
  - **Repeat Count**: number of repetition of the effect.
  - **Repeat Delay**: pause in seconds between two repetitions of the effect.
  - **Max Duration**: maximum duration in seconds of the effect (regardless of the repeat count).
  - **Effect Boost**: Boost intensity: [-100; 100] of the effect to use instead of the default pattern value if **overridePatternBoost** is set to true. By using a negative value, can



Haptic Effect

StrategyOnPlay: E_DEFAULT

By default, an effect can only play once. While it is playing, any call to PlayEffect() is ignored.

Priority level	(Low) 10 --> 1 (High)	
	5	
Volume	100	[0; 250] %
Speed	1	[0,01; 100]
RepeatCount	1	[1; 255] or 0 = ∞
RepeatDelay	0	s
PlayAtTime	0	s
MaxDuration	0	s ; 0 = no limits
OverridePatternBoost	☐	
EffectBoost	0	[-100; 100]
HeightTranslation	0	m
HeadingRotation	0	°
TiltingRotation	0	°
FrontBackInversion	☐	
UpDownInversion	☐	
RightLeftInversion	☐	
FrontBackAddition	☐	
UpDownAddition	☐	
RightLeftAddition	☐	
TargetDevice	SkineticDevice (Skin	
TargetPattern	None (Pattern Asset)	

decrease or even nullify the global intensity boost set by the user. More information can be found in the upcoming subsection.
    - **Override Pattern Boost**: By setting this boolean to true, the effect will use the effectBoost value instead of the default pattern value.
    - **Height Translation**: Height in meter to translate the pattern.
    - **Heading Rotation**: Heading angle in degree to rotate the pattern in the horizontal plan (y axis). A modulo operation is applied to the given angle.
    - **Tilting Rotation**: Tilting angle in degree to rotate the pattern in the sagittal plane (x axis). A modulo operation is applied to the given angle.
    - **frontBackInversion**: Invert the direction of the pattern on the front-back axis. Can be combined with other inversion or addition.
    - **upDownInversion**: Invert the direction of the pattern on the up-down axis. Can be combined with other inversion or addition.
    - **rightLeftInversion**: Invert the direction of the pattern on the right-left axis. Can be combined with other inversion or addition.
    - **frontBackAddition**: Perform a front-back addition of the pattern on the front-back axis. Overrides the frontBackInversion. Can be combined with other inversion or addition.
    - **upDownAddition**: Perform an up-down addition of the pattern on the front-back axis. Overrides the upDownInversion. Can be combined with other inversion or addition.
    - **rightLeftInversion**: Perform a right-left addition of the pattern on the front-back axis. Overrides the rightLeftInversion. Can be combined with other inversion or addition.

- **Stop Parameter**
  Once the effect is stopped, it will fade out from the current volume to 0 in the "**fadeout**" duration. The effect can still finish on its own earlier.

- **Play / Stop**
  HapticEffect can be triggered and interrupted with play and stop commands. It is also possible to check the state of an effect to determine if it is still playing or if it has been muted (see next section). To trigger a haptic effect, two workflow are available:
    - **From HapticEffect component:** If the **HapticEffect** component has a reference to a target **SkineticDevice** component, `PlayEffect`, `StopEffect`, and `GetEffectState` can be called from the HapticEffect component.
    - **From SkineticDevice component**: `PlayEffect`, `StopEffect`, and `GetEffectState` are called from a **SkineticDevice** component by passing a reference to an HapticEffect component that needs to be triggered.

5.3. Effect Priority Levels

HapticEffect have a priority level when they are triggered. The reason for this system is that there are a limited number of samples that can play simultaneously, i.e., only 10 independent haptic samples can be rendered simultaneously on the vest without regard to their spatialization. Extra samples are then “muted”, i.e., they are alive but not rendered.
However, it is a pattern instance (Haptic Effect) as a whole which is muted, i.e., all haptics samples composing it together are muted.

HapticEffect instances are then ordered by priority and only the effects with higher priority will be rendered, while the others will be muted.
The priority logic follows 3 rules to order two haptic effect:

- The priority effect is the one with the lowest priority level.
- The priority effect is the one using the smallest number of simultaneous samples (see Unitouch Studio).
- The priority effect is the most recent one, i.e., playing for the shortest amount of time.

Keep in mind that having too many effect playing simultaneously might:
- Make the actuators saturate and distort the rendered signals.
- Confuse the user as too many different sensations happen simultaneously.

5.4. Effect Transformation

When playing an effect, an instance of a pattern is created. If this pattern is **shape-based**, it can then be transformed using the three transformation parameters: **Height Translation, Heading Rotation** and **Tilting Rotation**.

- The three transformations are applied **in this order**: tilting, heading, height translation.
- The default position of a pattern is the one obtained when these three parameters are set to zero.
- There are no actual bounds to the angles as not to restrict the usage.
- The height is normalized between 1 and -1, when computing the translation, it has to be rescaled to fit your interaction, e.g. the actual size of the avatar.

The actual usage of these 3 parameters depends on the default position of the pattern and the targeted interaction. For example:

- For a piercing shot, a heading between [-180; 180]° can be combined with a tilting between [-90; 90] when using a shape-based pattern centered in the middle of the torso.
- For an environmental effect, a heading between [-90; 90]° (or [-180; 180]°) can be combined with a tilting between [-180; 180]° (resp. [-180; 0]°) when using a pattern with shapes centered on the top.

In addition, some mirrored operations are also available for shape-based patterns: with 3 inventions **frontBackInversion**, **upDownInversion**, **rightLeftInversion**, and 3 additions **frontBackAddition**, **upDownAddition**, **rightLeftAddition**.

5.5. Boost Guidelines

The boost feature allows to increase the perceived intensity of the haptic effect. There are two layers of boost which are simply added together for each haptic effect:

- The Global Boost which is applied to every effect being rendered as to increase them evenly. It can be set with setGlobalIntensityBoost() and is meant to be set by the user as an application setting.
- The Effect Boost which is set for a single effect instance only. A default value can be set in the Skinetic Studio and is saved in the .spn file as the reference value chosen during the design process. However, it can be overridden when triggering an effect with playEffect() by setting the boolean overridePatternBoost to true, in which case the effectBoost value passed is used instead of the default.

Since some effects are, by design, stronger than others, this two-layer feature is meant to allow the haptic effect designer to adjust the level intensity for each pattern of a set of patterns, while maintaining the design intent. As they all have a default boost value in the .spn that is added to the global boost intensity, it can be set to compensate for the discrepancy of intensity across a set of patterns. Weaker effects can have a high default boost value, while already strong effects can have a negative default boost value to reduce or even nullify the global intensity boost set by the user. Note that the overall boost can not decrease the base effect intensity.

The boost increases the overall perceived intensity of the haptic effects. However, the higher the boost activation, the more the haptic effects are degraded. Although the “texture” of the effect is less affected, high levels of boost result in a loss of precision in the spatialization of the effect. Hence, setting the value of the boost is a matter of compromise between perceived intensity/power and deterioration of the effect’s signal and spatial resolution.

Note that the boost is not impacted by the actuators’ activation levels but is modulated by the volume envelope defined in the pattern. This enforce the design process of:

- Defining the spatialization: where the effect is being rendered.

- Defining the volume envelope: how the intensity varies over the duration of the effect.
- Defining the default boost value: the reference value with respect to the whole set of patterns that will be used altogether.

5.6. Effects Accumulation

The effect accumulation strategy allows to trigger a secondary pattern as a fallback when an effect using the main pattern is already being rendered. The purpose of this feature is to enable the use of a lighter pattern if a specific pattern might be called too many times. It enables the main effect to be rendered fully
while having another effect triggered and still providing the additional event. This allows to alleviate several matters:

- When there are too many complex effects rendered at once, it might become even more confusing for the user as the vest is vibrating too much.
- When there are too many effects using too many samples simultaneously, the older effects might be muted by default, preventing any effect from being fully rendered.

Notice that, when too many effects are still being triggered simultaneously, this feature allows to reduce the amount of vibration and avoid immersion breaking behavior, but the user might still be confused.

The accumulation is done on the specified time window, starting with the first call. Each subsequent call within this time window will trigger the fallback pattern. It is possible to set a maximum amount of rendered effects within the time window to avoid an excessive amount of triggered effects, additional calls will then be ignored until the time window is finished. If the subsequent call has the same or higher priority, the actual priority is set to be just below the main effect, as to preserve the unity of the accumulation with respect to other playing effects. However, if the subsequent calls have lower priorities, then the standard priority system is used.

6. Errors & Logs

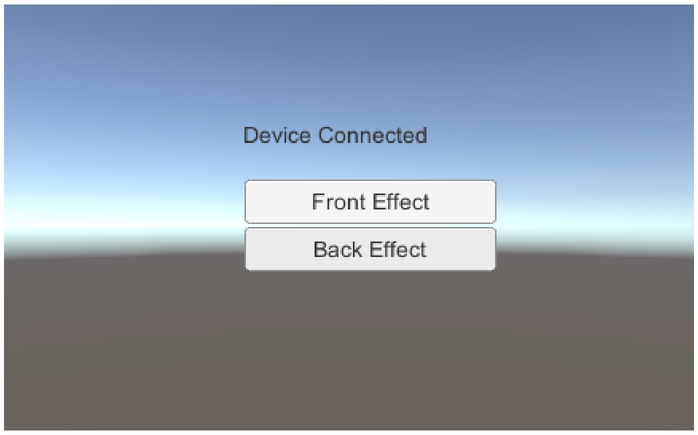
Most call the Skinetic SDK returns negative error codes when issues arise, you can check the corresponding error with `Skinetic.SkineticDevice.getSDKError()`.
In addition the Skinetic SDK also logs some information as it is running. The logs can be found in the SkineticSDK.log file which is overridden after each session. On Android, the logs are added to the **logcat** under SKINETICSDK.

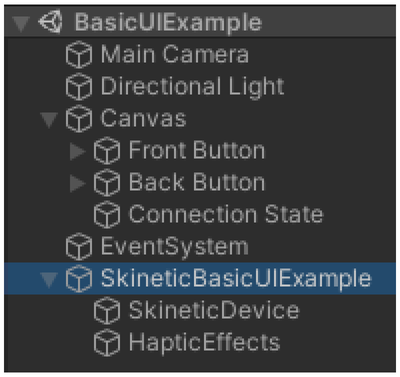
7. Examples Walkthrough

Several sample scenes are available in the plugin. To import them into your project open the package manager under **Windows → Package Manager**. In the dropdown menu, select either **In Project** or **Custom**, you will see the Skinetic SDK package appear in the list below. Within the package description, an import button allows you to import the sample scenes into your project.

7.1. Basic UI Example

This example is a basic "hello-world" example. In playmode, the device will automatically connect to any Skinetic device available, and update the connection status. Additionally two buttons are displayed in the middle of the screen, pressing one of them triggers a predefined haptic effect on the vest on the front (or back) of the haptic device.





The scene is composed of a canvas with two buttons and a text and an empty gameobject SkineticBasicUIExample containing two gameobjects: one to handle the **SkineticDevice** component and on to handle the **HapticEffects**.

- **Automatic connection**

The connection is automatically performed by the **SkiEx_AutoConnect** script on the SkineticDevice gameobject. This script has a reference set in the inspector to the SkineticDevice component and a Text on the canvas.
On the OnEnable event:

  - A connection callback is set:
    >> m_device.SetConnectionCallback(ConnectionCallback);
    This callback will be called whenever the connection to the device is made or broken, and update the state variable **m_isConnected.**

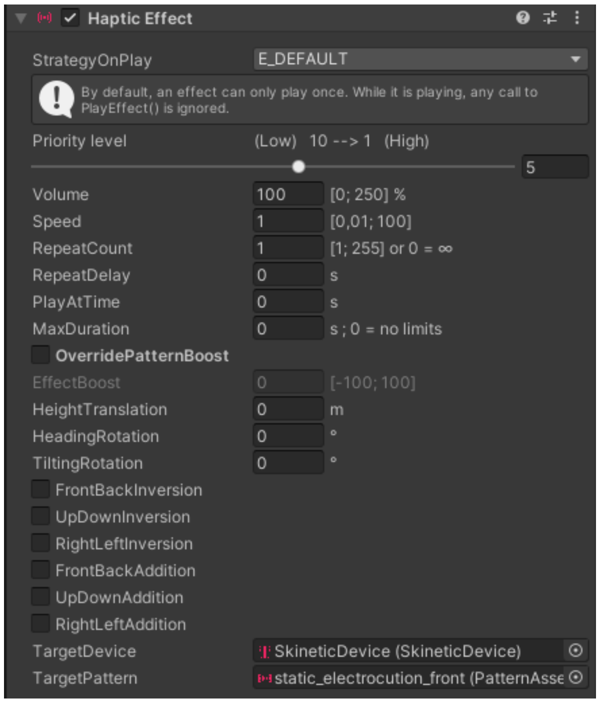
- A connection to any Skinetic device is requested by passing **Skinetic.SkineticDevice.OutputType.E_AUTODETECT** and **0** to allow any kind of connection and pick the first available device.
- Start a coroutine to update the connection status in the text field as **the callback being called from another thread cannot update the UI itself.**

During the OnDisable event, the disconnection routine is initiated and the function returns once the disconnection is completed.

- **Basic Effects**

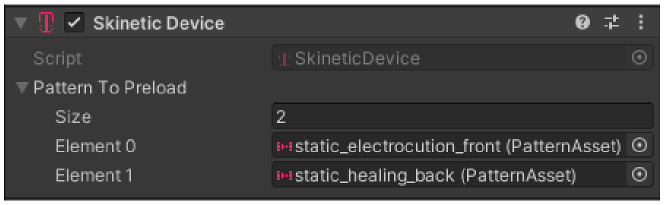
The **HapticEffects** gameobject contains 3 components:

- two preconfigured **HapticEffect** components whose parameters are predefined in the inspector and hold references to the **SkineticDevice** component in the scene and to the **PatternAsset** they have to use.
- a **SkiEx_BasicUIEffects** script which simply holds reference to the two **HapticEffect** and exposes 2 public functions registered in the inspector to each button's OnClick() event. These functions then call `PlayEffect` **from the HapticEffect** as the target device was set in the inspector.



```
StrategyOnPlay          E_DEFAULT
By default, an effect can only play once. While it is playing, any call to PlayEffect() is ignored.
Priority level          (Low)  10 --> 1  (High)
                                                5
Volume                  100   [0; 250] %
Speed                   1     [0,01; 100]
RepeatCount             1     [1; 255] or 0 = ∞
RepeatDelay             0     s
PlayAtTime              0     s
MaxDuration             0     s ; 0 = no limits
OverridePatternBoost
EffectBoost             0     [-100; 100]
HeightTranslation       0     m
HeadingRotation         0     °
TiltingRotation         0     °
FrontBackInversion
UpDownInversion
RightLeftInversion
FrontBackAddition
UpDownAddition
RightLeftAddition
TargetDevice            SkineticDevice (SkineticDevice)
TargetPattern           static_electrocution_front (PatternAsse
```

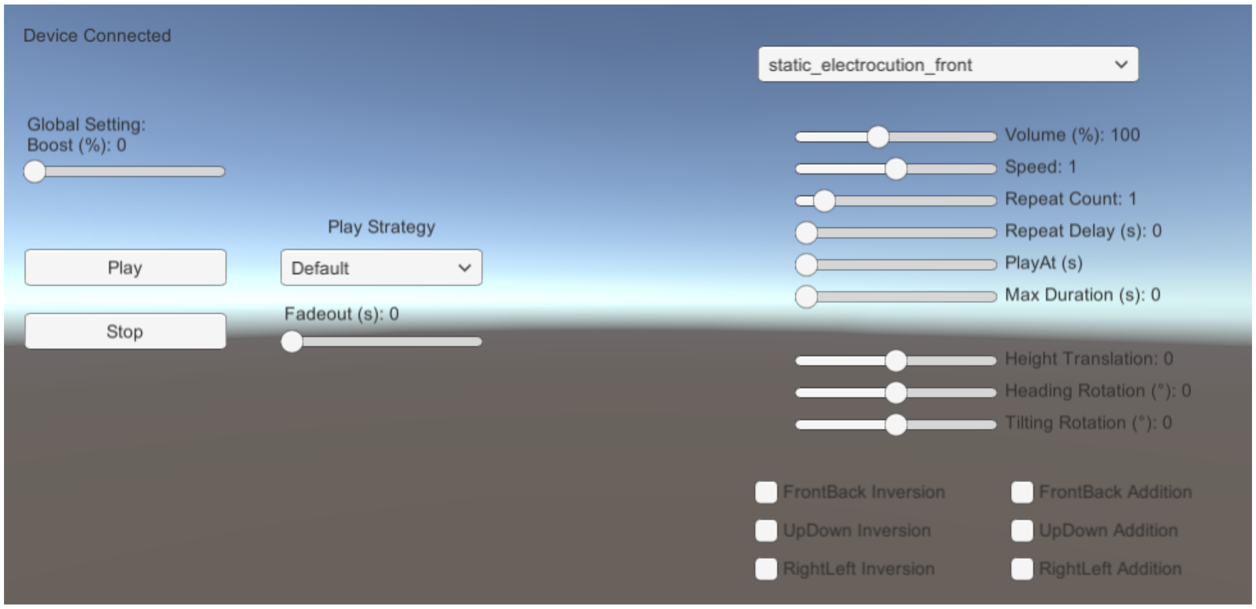
Additionally the two **PatternAssets** are added to the preload list of the **SkineticDevice** so there is no need to register them through scripting as they will be loaded right at the beginning.



```
Script                  SkineticDevice
Pattern To Preload
  Size                  2
  Element 0             static_electrocution_front (PatternAsset)
  Element 1             static_healing_back (PatternAsset)
```

7.2. Editable Parameters Example

In this example the haptic effect parameters can be edited from a UI. In playmode, the device will automatically connect to any Skinetic device available, and update the connection status. Below, the **Boost Global Setting** is exposed and allows to increase the overall intensity of every effect on the vest. Additionally the UI exposes 5 different **PatternAssets** that the user can select. Then, the user can change the parameters of the effect that are passed to the `PlayEffect` function. The transformation parameters only affect the 2 last patterns as the first 3 are static.

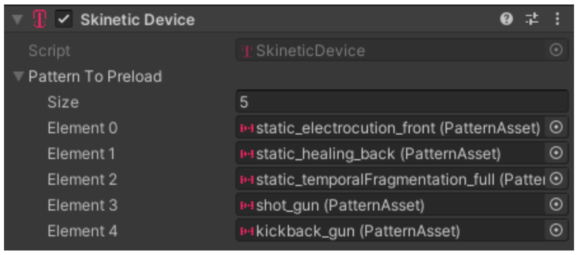


- **Automatic connection**

The automatic connection is the same as the Basic UI Example using the same **SkiEx_AutoConnect** script.

- **PatternAssets**

In this example, 5 **PatternAssets** are added to the preload list of the **SkineticDevice** component. After the OnEnable event, they will be loaded and accessible through **LoadedPatterns,** a read-only list of all loaded patterns available on the **SkineticDevice**


Skinetic Device

Script	SkineticDevice
Pattern To Preload	
Size	5
Element 0	static_electrocution_front (PatternAsset)
Element 1	static_healing_back (PatternAsset)
Element 2	static_temporalFragmentation_full (Patte
Element 3	shot_gun (PatternAsset)
Element 4	kickback_gun (PatternAsset)

component. The component then acts as a storage that another script will use.

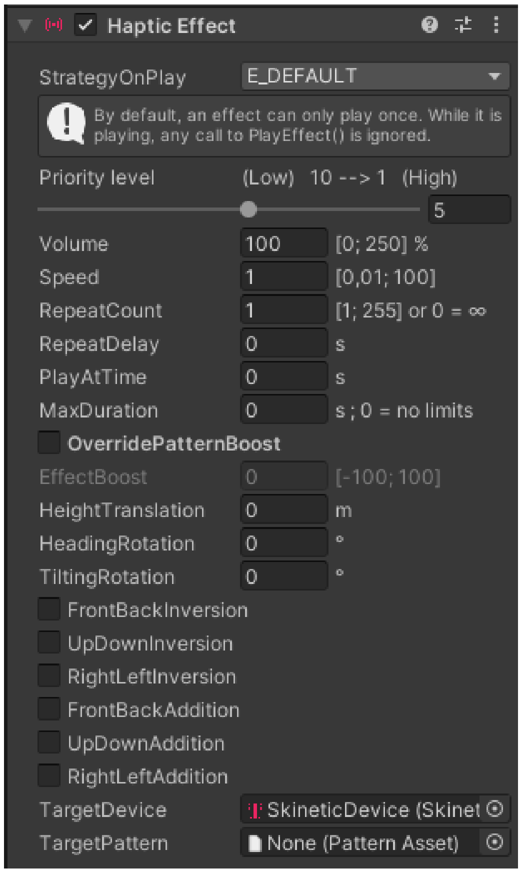
- **Parametric Effects**

The **HapticEffects** gameobject possesses a single **HapicEffect** component and a **SkiEx_ParametricPlay** script that handles the UI elements.

The **HapticEffect** component has a reference to the **SkineticDevice** as there is only one in the scene. However, the **TargetPattern** field is not set in the inspector as it will be set through scripting by the **SkiEx_ParametricPlay** script.

For each property of the haptic effect, the setter is registered to the **OnValueChanged** event of the corresponding slider. Except for the repeat count and the play strategy that are set through function of the **SkiEx_ParametricPlay** script as their type could not be directly handled by the UI's event.

The Height Translation, Heading Rotation and Tilting Rotation, as well as the 3 Inversion and Addition boolean are only affecting the transformable pattern.


Haptic Effect

StrategyOnPlay: E_DEFAULT

By default, an effect can only play once. While it is playing, any call to PlayEffect() is ignored.

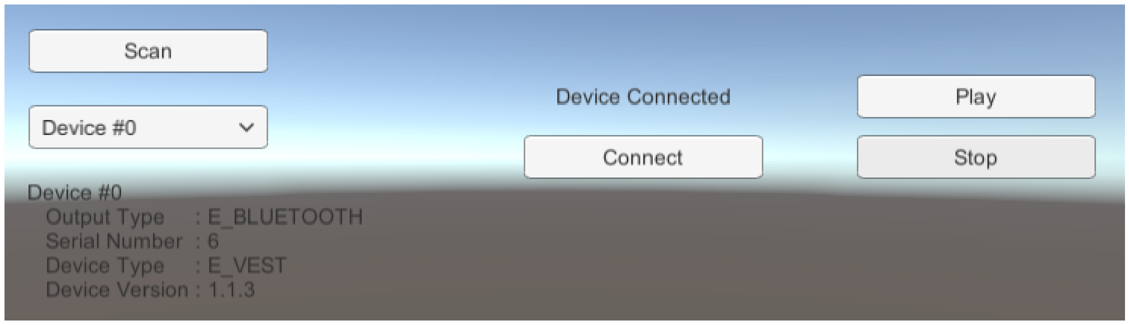
Priority level (Low) 10 --> 1 (High): 5

Volume: 100 [0; 250] %
Speed: 1 [0,01; 100]
RepeatCount: 1 [1; 255] or 0 = ∞
RepeatDelay: 0 s
PlayAtTime: 0 s
MaxDuration: 0 s ; 0 = no limits
OverridePatternBoost: ☐
EffectBoost: 0 [-100; 100]
HeightTranslation: 0 m
HeadingRotation: 0 °
TiltingRotation: 0 °
FrontBackInversion: ☐
UpDownInversion: ☐
RightLeftInversion: ☐
FrontBackAddition: ☐
UpDownAddition: ☐
RightLeftAddition: ☐
TargetDevice: SkineticDevice (Skinet
TargetPattern: None (Pattern Asset)

Any modification of the **HapticEffect** properties is then taken into account on the next `PlayEffect`. In addition to the play button, a stop button is also added to call `StopEffect`.

7.3. Scan Device Example

In this example the user can trigger a scan of all available Skinetic devices and display the result of the scan. He can then select one of the devices and connect to it. Play and stop buttons enable to trigger an effect in order to test the connection.



```
Device #0
   Output Type    : E_BLUETOOTH
   Serial Number  : 6
   Device Type    : E_VEST
   Device Version : 1.1.3
```

- **Basic Effects**

The **HapticEffects** gameobject contains only a basic predetermined effect as to test the connection. The effect is handled and triggers with buttons just like in the previous example. The only difference is that the **PatternAsset** is not in the **Preload** list of the **SkineticDevice** component and is registered through scripting in the **SkiEx_BasicSingleEffect** script that handles the effect and the triggers.

- **Scanning Procedure**

The Scanning (and connection) procedure is done in the **SkiEx_DeviceScan** script. The scan is triggered by pressing on the button Scan which triggers the ScanDevices coroutine.

  - This coroutine first disconnects any device that might still be connected, as the scan can only be done if no device is connected. Since no callback was set in this example, the connection state is monitored by calling `ConnectionStatus` every 0.2s until the state **E_DISCONNECTED** is obtained.
  - Then the scanning procedure is triggered to find Skinetic devices using any kind of output type. Then every 0.2s the scanning procedure status is checked with `ScanStatus` until it returns 0 for a successful scan.
  - Finally the result is obtained with `GetScannedDevices`, the dropdown menu is updated with the available devices and the first one in the list is displayed.

- **Displaying Device Info**

The Scanning procedure returns a list of **Skinetic.SkineticDevice.DeviceInfo**. A dropdown menu allows the user to select one of the devices. By doing so, the device information is displayed in a text field: OutputType, Serial Number, Device Type, Device Version.

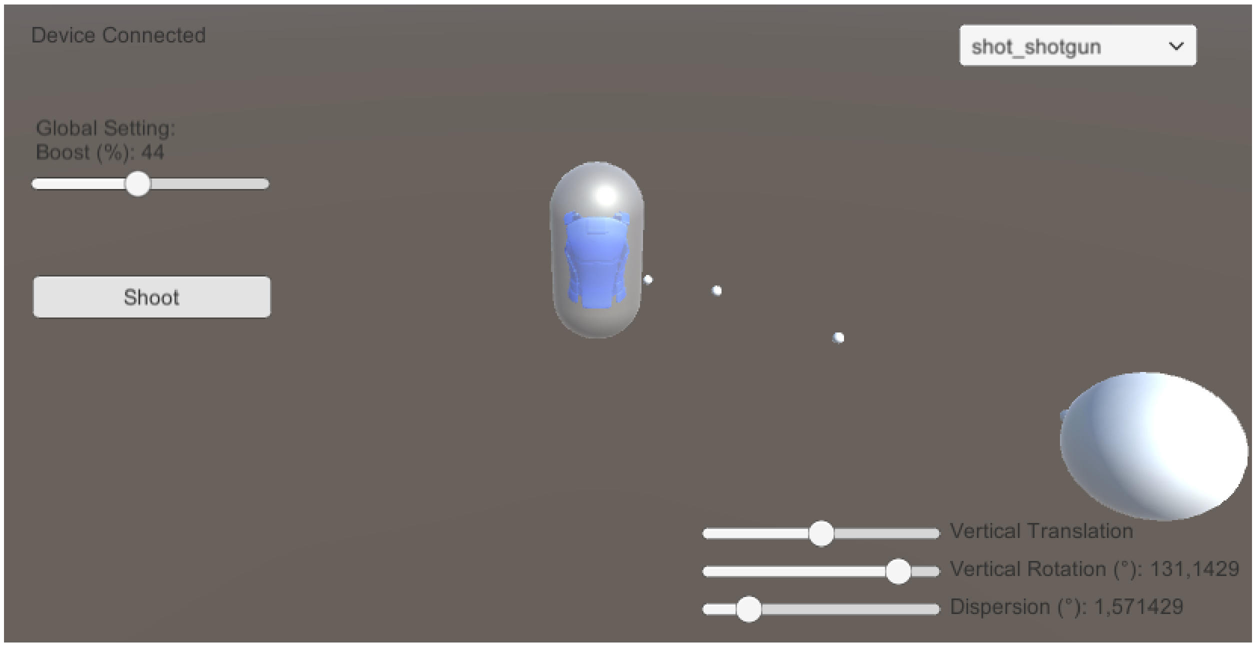
- **Connection Procedure**

As the user select a device from the scanned device list, he can connect to it by clicking on the Connect button which triggers the connection routine:

- This coroutine first calls `Connect` by passing as argument the **output type** and the **serial number** of the selected device in the scan list.
- As there is no connection callback set, the connection state is monitored by calling `ConnectionStatus` every 0.2s until the state **E_CONNECTED** or **E_DISCONNECTED** is obtained.

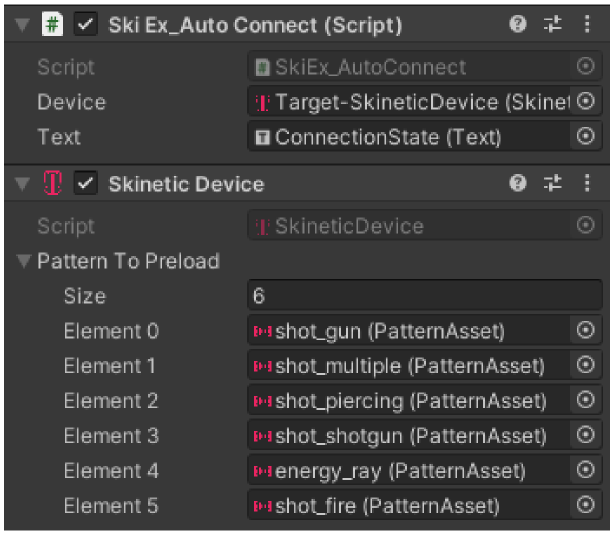
7.4 Spatialization Example

In this example the haptic effect transformation parameters can be tested as a sphere is shooting marbles onto a target capsule representing the vest. In playmode, the device will automatically connect to any Skinetic device available, and update the connection status. Below, the **Boost Global Setting** is exposed and allows to increase the overall intensity of every effect on the vest. Additionally the UI exposes 2 different **PatternAssets** that the user can select. Then, the user can change the position of the shooting sphere.



- **Automatic connection**
The automatic connection is the same as the Basic UI Example using the same **SkiEx_AutoConnect** script.

- **PatternAssets**
In this example, 6 **PatternAssets** are added to the preload list of the **SkineticDevice** component. After the OnEnable event, they will be loaded and accessible through **LoadedPatterns,** a read-only list of all loaded patterns available on the **SkineticDevice** component. The


Ski Ex_Auto Connect (Script)

Script	SkiEx_AutoConnect
Device	Target-SkineticDevice (Skinet
Text	ConnectionState (Text)

Skinetic Device

Script	SkineticDevice
Pattern To Preload	
Size	6
Element 0	shot_gun (PatternAsset)
Element 1	shot_multiple (PatternAsset)
Element 2	shot_piercing (PatternAsset)
Element 3	shot_shotgun (PatternAsset)
Element 4	energy_ray (PatternAsset)
Element 5	shot_fire (PatternAsset)

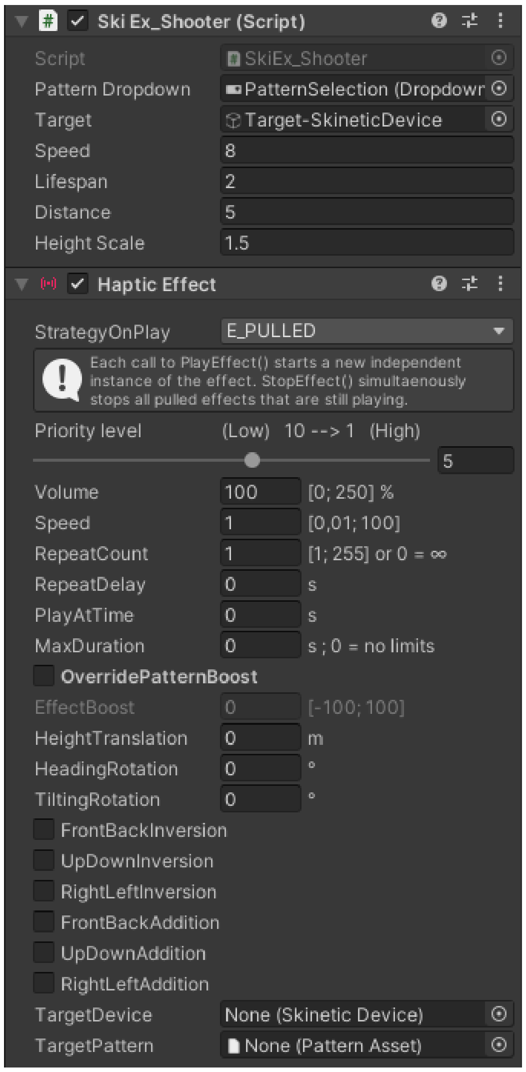
component then acts as a storage that another script will use.

- **Shooter**

Using the UI slider, the user can rotate the shooter around the target and perceive the spatialized haptic effect. The SkiEx_Shooter handles the HapticEffect Component and sets its TargetDevice and TargetPattern through scripting. It also instantiates the bullets and passes them the reference to the HapticEffect to use. In order to render each bullet impact individually, the HapticEffect component is set to **E_PULLED**.

- **Impact Handling**

The SkiEx_SpatializedCollision handles the computation of the transform parameters in order to apply the haptic effect on the right part of the Skinetic Vest and triggers the HapticEffect component of the Shooter.


Ski Ex_Shooter (Script)

Script	SkiEx_Shooter
Pattern Dropdown	PatternSelection (Dropdowr
Target	Target-SkineticDevice
Speed	8
Lifespan	2
Distance	5
Height Scale	1.5

Haptic Effect

StrategyOnPlay E_PULLED

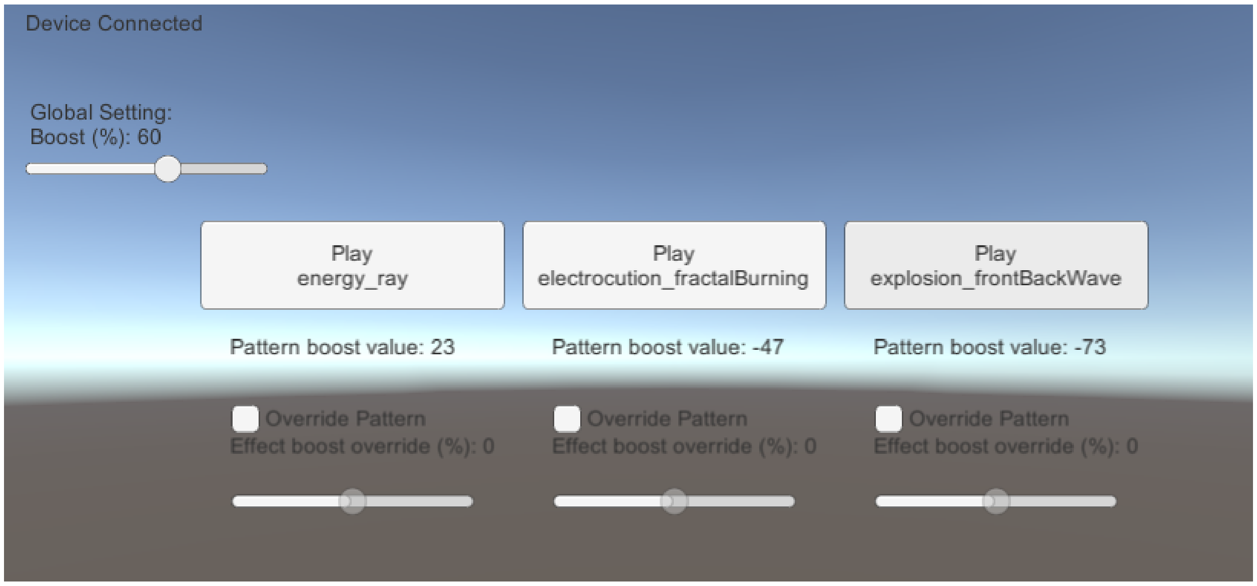
Each call to PlayEffect() starts a new independent instance of the effect. StopEffect() simultaenously stops all pulled effects that are still playing.

Priority level (Low) 10 --> 1 (High) 5

Volume	100	[0; 250] %
Speed	1	[0,01; 100]
RepeatCount	1	[1; 255] or 0 = ∞
RepeatDelay	0	s
PlayAtTime	0	s
MaxDuration	0	s ; 0 = no limits
OverridePatternBoost	☐	
EffectBoost	0	[-100; 100]
HeightTranslation	0	m
HeadingRotation	0	°
TiltingRotation	0	°
FrontBackInversion	☐	
UpDownInversion	☐	
RightLeftInversion	☐	
FrontBackAddition	☐	
UpDownAddition	☐	
RightLeftAddition	☐	
TargetDevice	None (Skinetic Device)	
TargetPattern	None (Pattern Asset)	

7.5. Boost Example

In this example the boost feature can be tested following the Boost Guidelines section. Three patterns with various boost pattern values can be overridden individually while the **Boost Global Setting** is exposed and allows to increase the overall intensity of every effect on the vest.



Global Setting:
Boost (%): 60

Pattern boost value: 23
Effect boost override (%): 0

Pattern boost value: -47
Effect boost override (%): 0

Pattern boost value: -73
Effect boost override (%): 0

- **Automatic connection**

The automatic connection is the same as the Basic UI Example using the same **SkiEx_AutoConnect** script.

- **PatternAssets**

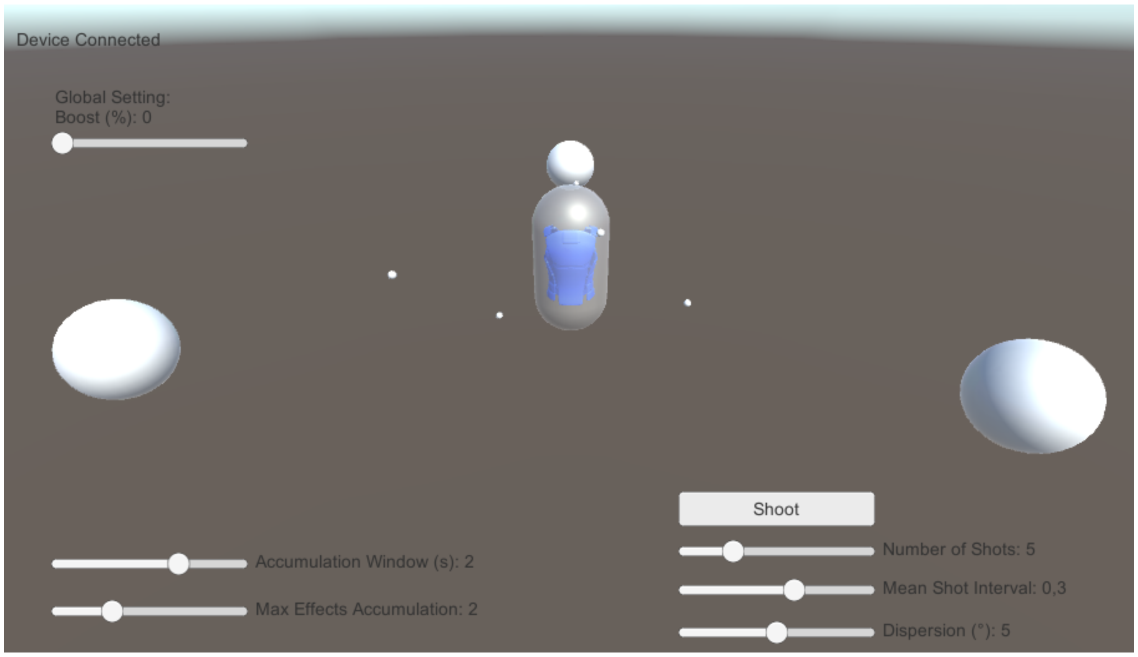
In this example, the 3 **PatternAssets** are not added to the preload list of the **SkineticDevice** component. Instead, the SkiEx_Boost component loads the **PatternAssets** that are set for each one of the three HapticEffect components.

- **Boost Handling**

Each pattern has a **patternBoost** property that is displayed by the **SkiEx_EffectBoostHandle**. This script also manages the effectBoost property of the associated HapticEffect component which can be used instead of the patternBoost by setting the overridePatternBoost setting.

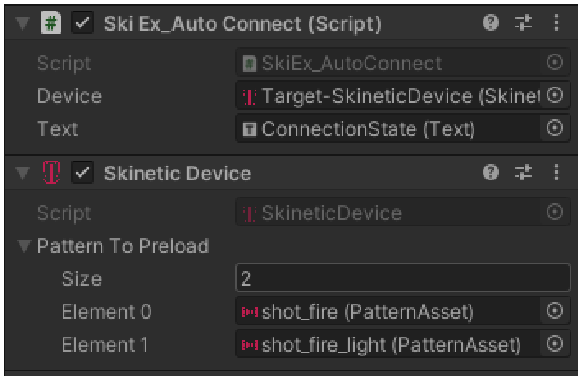
7.6. Effect Accumulation Example

In this example the accumulation strategy feature can be tested following the Effect Accumulation section. Two similar patterns used in this scene, shot_fire.spn and shot_fire_light.spn. The main one is made of an impact blast then a remaining burning effect on the impact area which contains several components (deep combustion, sparks and cracks). However this effect does not work too well when triggered too much. Hence, the lighter version is triggered instead as it is only made of the impact blast and the sparks.
The accumulation between these two patterns can be configured on the left side, while the right side allows to configure a salvo from the 3 shooting spheres.



- **Automatic connection**
The automatic connection is the same as the Basic UI Example using the same **SkiEx_AutoConnect** script.

- **PatternAssets**
In this example, the 2 **PatternAssets** are added to the preload list of the **SkineticDevice** component.



Ski Ex_Auto Connect (Script)

Script	SkiEx_AutoConnect
Device	Target-SkineticDevice (Skinet
Text	ConnectionState (Text)

Skinetic Device

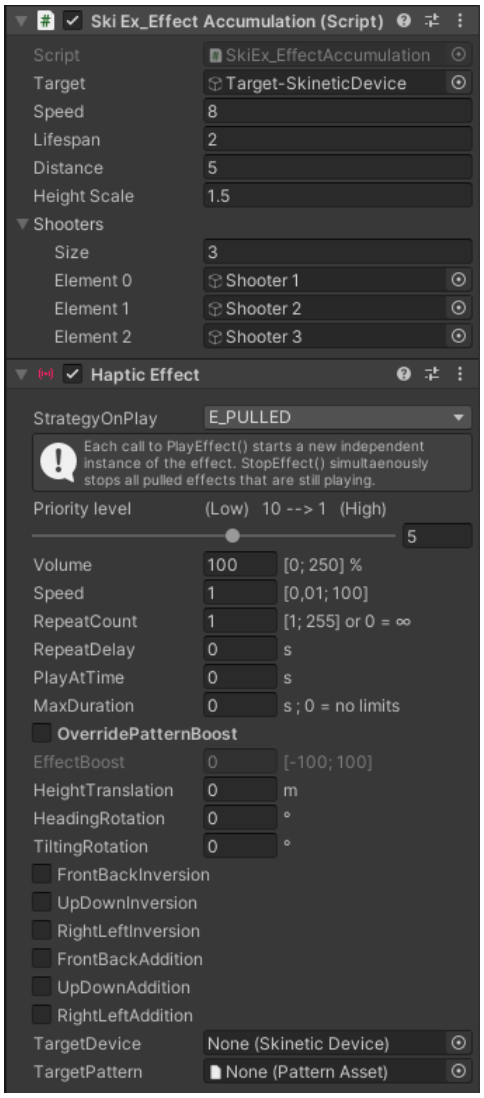
Script	SkineticDevice
Pattern To Preload	
Size	2
Element 0	shot_fire (PatternAsset)
Element 1	shot_fire_light (PatternAsset)

- **Effect Accumulation Handling**

Using the UI sliders, the user can set the duration of the accumulation window and the number of accumulated effects after the first one. The SkiEx_EffectAccumulation script then simply sets the accumulation using the reference to the haptic device.

- **Shooters**

The SkiEx_EffectAccumulation script handles the HapticEffect Component and sets its TargetDevice and TargetPattern through scripting. It also instantiates the bullets from the three shooters and passes them the reference to the HapticEffect to use. In order to render each bullet impact individually, the HapticEffect component is set to **E_PULLED**.



Ski Ex_Effect Accumulation (Script)

Script	SkiEx_EffectAccumulation
Target	Target-SkineticDevice
Speed	8
Lifespan	2
Distance	5
Height Scale	1.5
Shooters	
Size	3
Element 0	Shooter 1
Element 1	Shooter 2
Element 2	Shooter 3

Haptic Effect

StrategyOnPlay	E_PULLED	
Priority level	(Low) 10 --> 1 (High)	
	5	
Volume	100	[0; 250] %
Speed	1	[0,01; 100]
RepeatCount	1	[1; 255] or 0 = ∞
RepeatDelay	0	s
PlayAtTime	0	s
MaxDuration	0	s ; 0 = no limits
OverridePatternBoost	☐	
EffectBoost	0	[-100; 100]
HeightTranslation	0	m
HeadingRotation	0	°
TiltingRotation	0	°
FrontBackInversion	☐	
UpDownInversion	☐	
RightLeftInversion	☐	
FrontBackAddition	☐	
UpDownAddition	☐	
RightLeftAddition	☐	
TargetDevice	None (Skinetic Device)	
TargetPattern	None (Pattern Asset)	

Each call to PlayEffect() starts a new independent instance of the effect. StopEffect() simultaenously stops all pulled effects that are still playing.