

4Players SCILL Unity SDK

Documentation

SCILL is a data driven platform providing various services for developers. Today, these services include:

- Challenges
- Battle Passes
- Leaderboards

Get started with SCILL

SCILL gives you the tools to activate, retain and grow your user base in your app or game by bringing you features well known in the gaming industry: Gamification. We take care of the services and technology involved so you can focus on your game and content.

The source of (user) action is motivation. Having an app that solves a daily need is enough to motivate users to launch the app and to use it. However, everyone is facing constant distractions and offerings on how to spend their time. So it's important to keep your users in the loop of being constantly motivated and wanting to launch your app or game. One way of doing that, and the gaming industry has done that for a long time very successfully, is by offering challenges and rewards.

However, while the concept of giving the users challenges and rewarding them for successful completion is easy, building and maintaining it is not so easy. You need to gather a lot of data and process it in real time. Next, you need to trigger your users at the right time. All of this is easier said than done.

This is where SCILL comes into play: If you follow this guide you will have integrated basic challenges into your app or application production ready in a couple of hours.

SCILL currently supports challenges, battle passes, and leaderboards, all of which are driven by events you send to us.

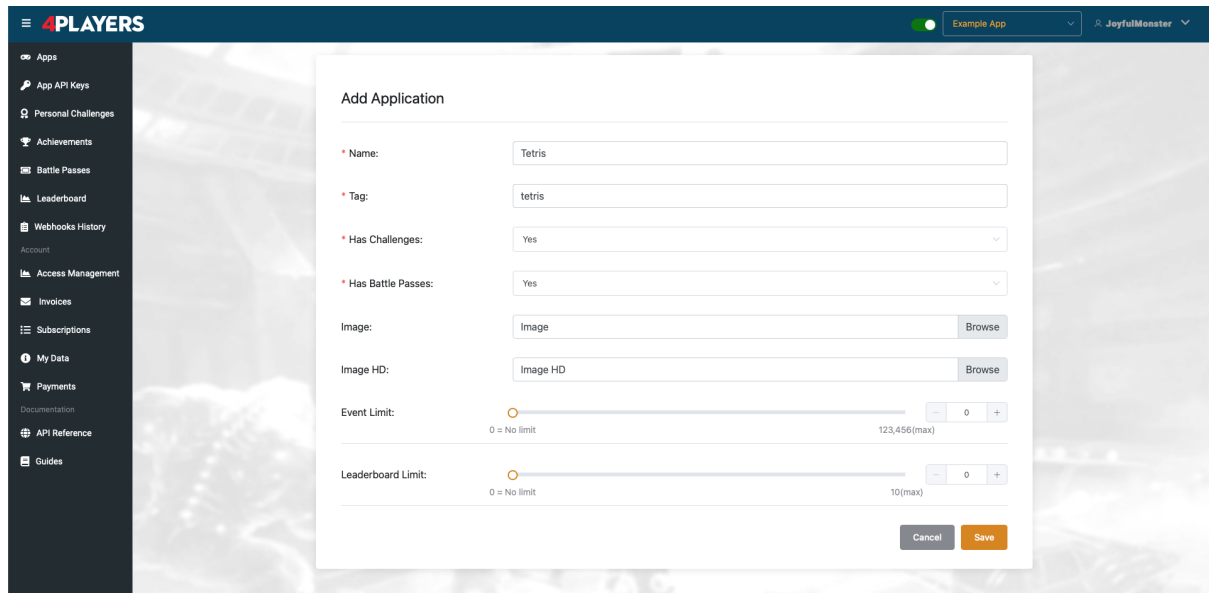
Tip

Testing SCILL does not require any coding. Using our Admin Panel you can build challenges and battle passes in minutes. Use our Playground application to simulate events, see your challenges and battle passes come to life. If you like it, add our SDK and in case of Unity you'll have a battle pass running after dropping two prefabs in your scene.

Create an app

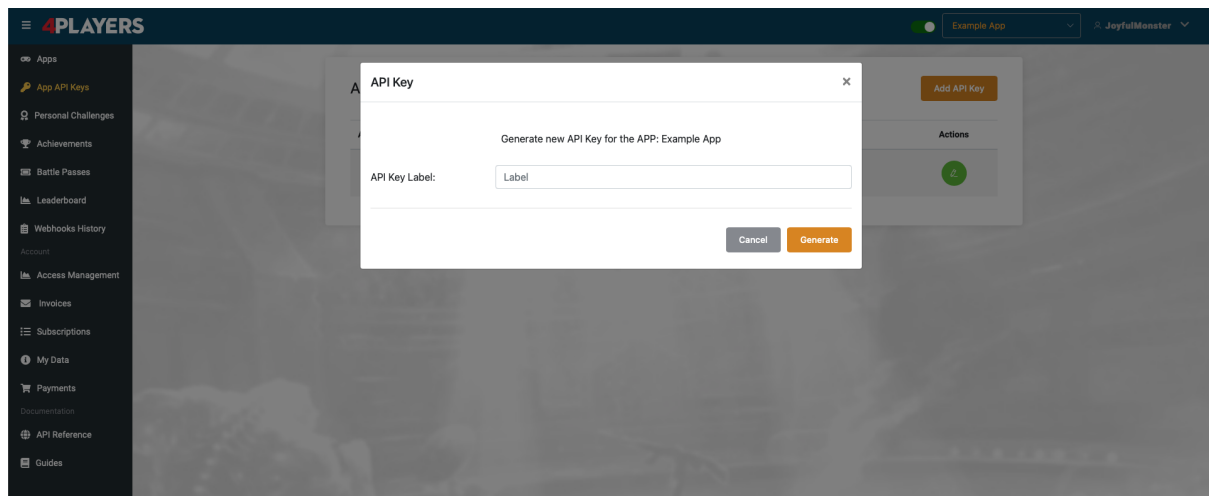
The core organizational unit in SCILL is an application.

Create an app in our [Admin Panel](#):



The screenshot shows the 4PLAYERS Admin Panel interface. On the left is a dark sidebar with a menu containing: Apps, App API Keys, Personal Challenges, Achievements, Battle Passes, Leaderboard, Webhooks History, Account, Access Management, Invoices, Subscriptions, My Data, Payments, Documentation, API Reference, and Guides. The top bar is dark blue with the 4PLAYERS logo, a green status indicator, a dropdown menu showing 'Example App', and a user profile 'JoyfulMonster'. The main content area displays a white modal titled 'Add Application'. This modal contains several form fields: 'Name' (text input with 'Tetris'), 'Tag' (text input with 'tetris'), 'Has Challenges' (dropdown menu with 'Yes'), and 'Has Battle Passes' (dropdown menu with 'Yes'). Below these are 'Image' and 'Image HD' fields, each with a 'Browse' button. At the bottom of the modal are two sliders: 'Event Limit' (ranging from 0 to 123,456) and 'Leaderboard Limit' (ranging from 0 to 10). The modal concludes with 'Cancel' and 'Save' buttons.

After that, select the new app from the app selection in the top bar and navigate to “API Keys” to add an API key for the new game:



This screenshot shows the same 4PLAYERS Admin Panel, but now the top bar dropdown menu is set to 'Example App'. The 'App API Keys' menu item in the sidebar is highlighted. A white modal titled 'API Key' is open in the center. The modal contains the text 'Generate new API Key for the APP: Example App' and an 'API Key Label' text input field with the placeholder 'Label'. At the bottom of the modal are 'Cancel' and 'Generate' buttons. In the background, a blurred view of the 'API Keys' page is visible, showing an 'Add API Key' button and an 'Actions' section with a green circular icon.

You need an API key to securely communicate with the SCILL backend. Keep your API key hidden and private; do not expose it in the browser.

Unity SDK

Our Unity SDK uses a special version of the [C# SDK](#). It uses [UnityWebRequest](#) instead of standard .NET requests and comes with some other customizations that are required for WebGL support. It mostly provides UI code and prefabs that you can use as a starting point for your SCILL integration directly into your game.

Our Unity SDK offers a quick “go to market” solution that you can just “drag & drop” into your game. The prefabs we provide are fully functional and handle all the heavy lifting with the SCILL cloud. They are completely open-source, and you are welcomed to adjust them to your own needs. Prefabs quickly can be adjusted to your games UI with prefab variants.

The main functionality is implemented by the classes, but virtual functions allow you to quickly make adjustments by overriding these classes and methods to adjust certain aspects of our implementation.

Tip

Please make sure you have the [Event System](#) script in your scene, as the SCILL Unity SDK uses UI code that requires the Unity Event System. To add a [Event System](#) to your scene in Unity, right-click into the [hierarchy](#) window and select the [GameObject](#) -> [UI](#) -> [EventSystem](#) option.

Video Tutorial

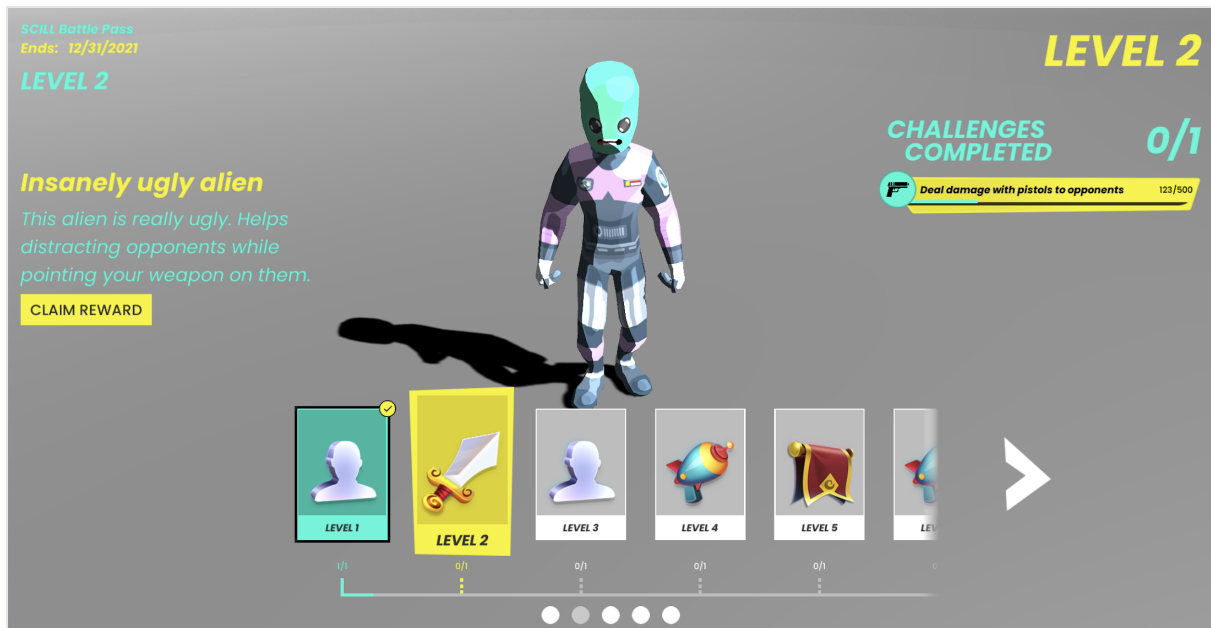
David Lieberman (CrystalMesh) has compiled an easy to follow and understandable video tutorial on how to add SCILL to your game and guides you to the first steps of leveraging our great tool kit to quickly integrate high-tech features like cloud based realtime challenges and leaderboards.

Youtube-Link: [▶ How to integrate SCILL Unity SDK to your game](#)

Production Ready Prefabs

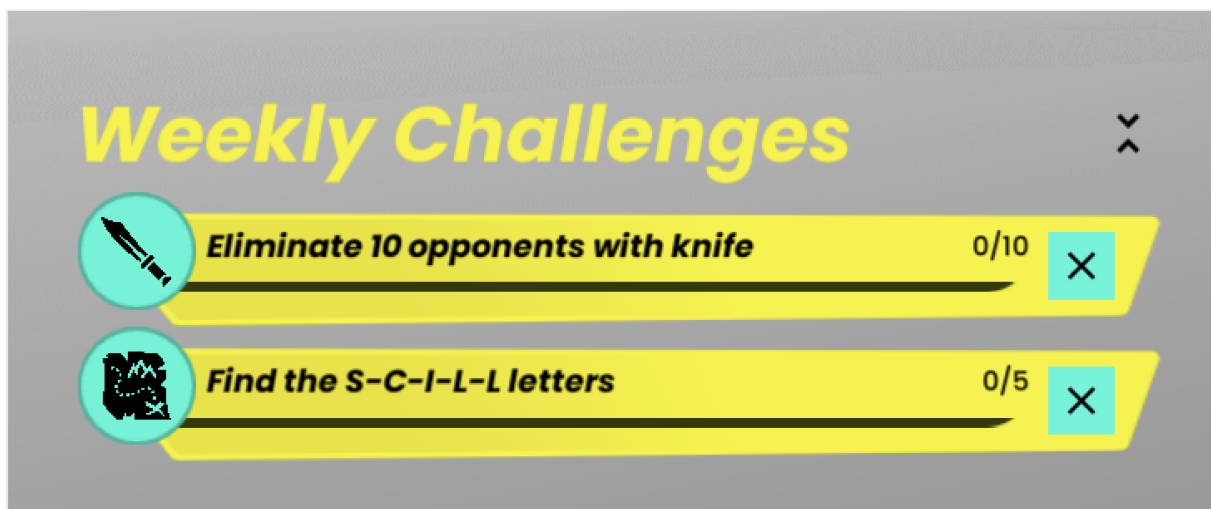
Our Unity SDK comes with fully working prefabs and samples with complete implementation of a battle pass and challenges. We have used Game Assets from [Kenney](#) for the samples, which are slim and awesome. Thank you very much Kenney for making those public domain.

This is a screenshot of the fully working Battle Pass UI Prefab with reward selection, realtime challenges and progress tracking.



Unity example of an example shooter battle pass

SCILL also offers simple challenges that can be utilized for many things like “Weekly Challenges”, giving your players enough to do each week and also allows users to collect rewards, coins and/or experience points.



Unity example of an personal challenges

Source Code

You can find the source code to our Unity SDK in our public Github repository: <https://github.com/scillgame/scill-unity>. If you encounter any bugs or issues, please let us know in the Issues tab of Github.

If you fix a bug in our code base, please commit your bugfix in a separate branch and create a merge request with a short description what you have fixed. Of course, we will credit you and your game in the changelog and at the start page of our Github repository in this case!

Installing the SDK

You have the various methods to install the Unity SDK.

- Using the Package Manager (recommended)
- Unity Package
- Download and “merge” with your Project.

Package Manager

Using the package manager will ensure that all dependencies are set up correctly and that you will have the most up to date version of our SDK. In most cases, using the Package Manager is the way to go.

Adding from git URL

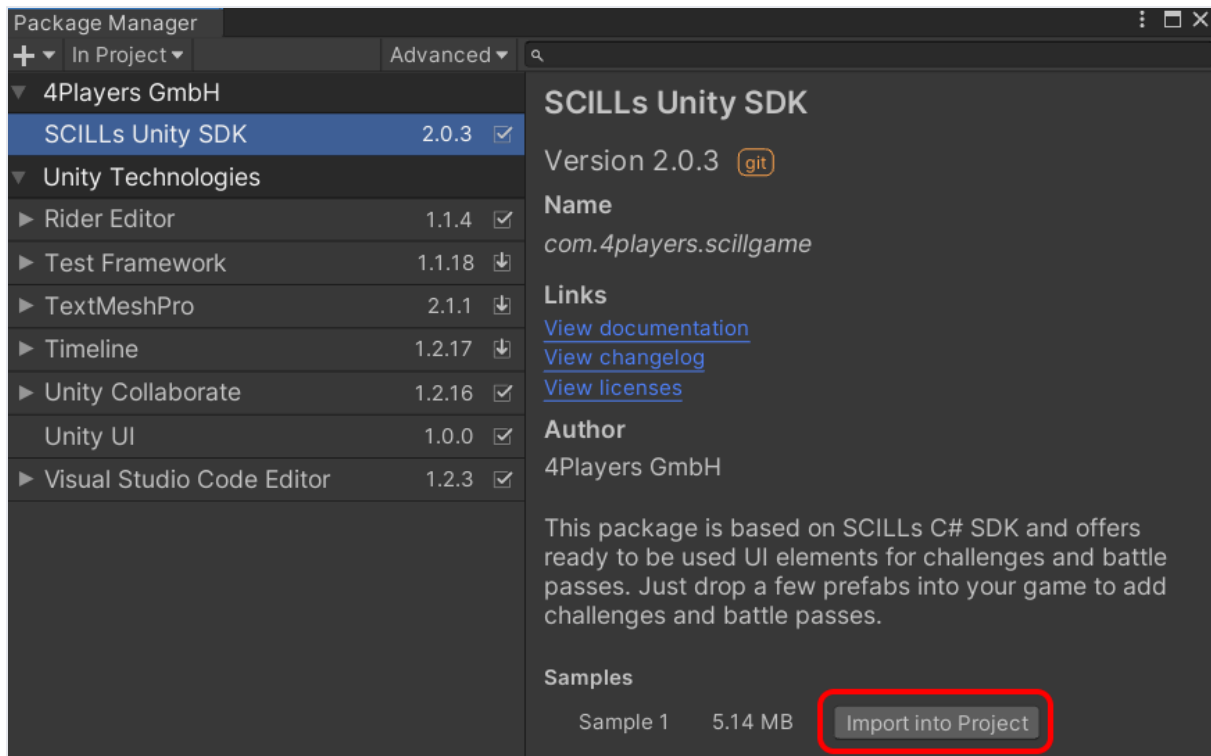
In Unity click on the menu **Window** -> **Package Manager** to open the Package Manager. In the Package Manager window, click on the + button and choose **Add package from git URL ...**, with this URL:

<https://github.com/scillgame/scill-unity.git?path=/unity-package>

Tip

To use the **Add package from git URL ...** option, please make sure the [Git client](#) (minimum version 2.14.0) is installed on your machine and that you have added the Git executable path to the PATH system environment variable. Take a look at the [Unity Documentation](#) for further information on using Git dependencies in the Unity Package Manager.

Don't forget to also import the **Sample 1** into your project if you'd like to test and change the sample scenes we created for you.

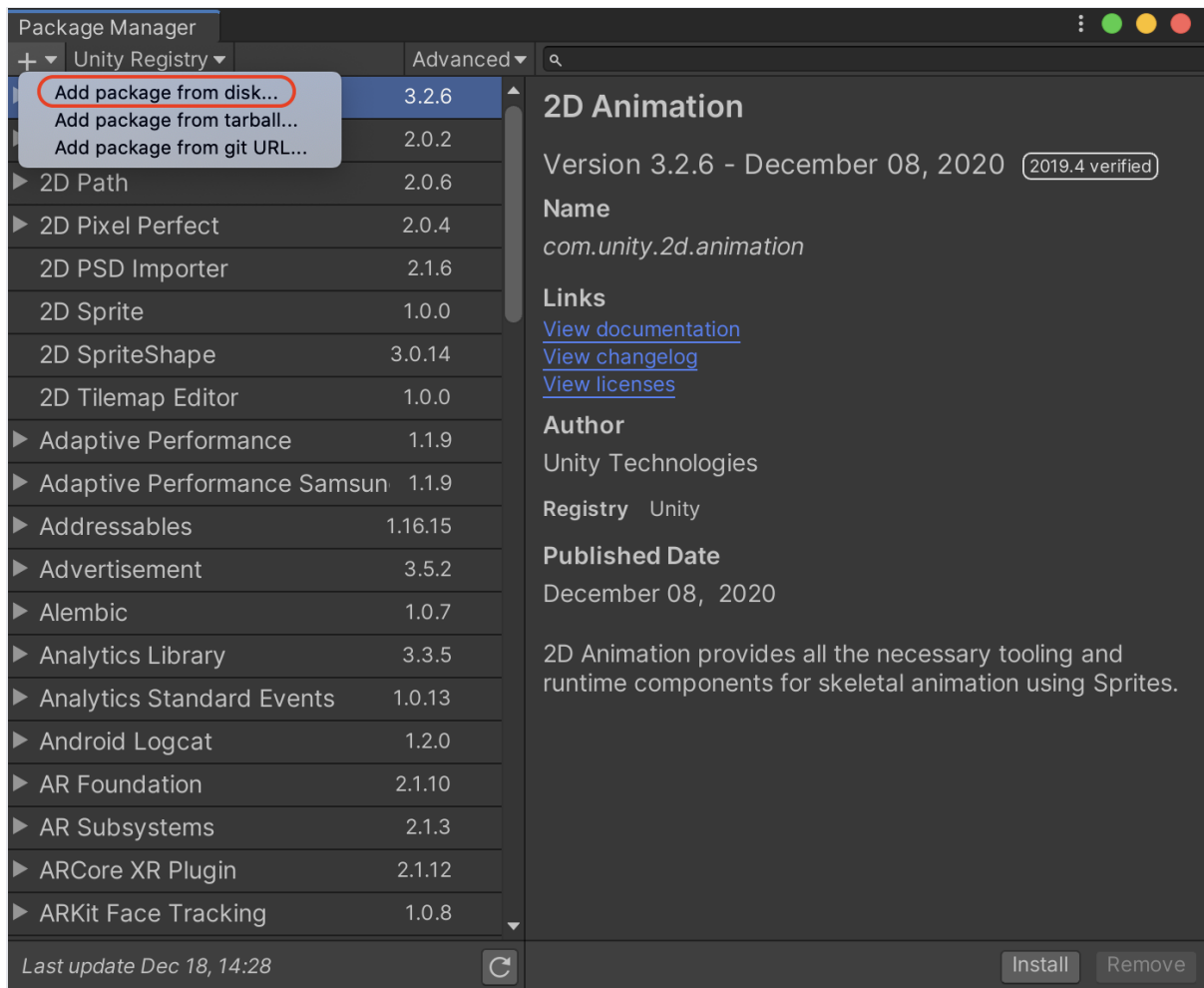


Importing the samples from the Git repository.

To update the SCILL Unity SDK, just add the package using the git URL method again - the Package Manager will automatically update the current local version to the newest available version in the repository.

Adding from disk

For this method you will need to first clone the SCILL Unity SDK from our [github repository](#). In the Package Manager window, click on the + button and choose **Add package from disk** Navigate to the `\unity-package` directory within the local repository on your device and select the `package.json` file. Unity will then import the package and resolve the dependencies.



Adding the SCILL Unity SDK with Package Manager

The advantage is, that you can pull new versions of our SDK and you can also directly work on the source code. So if you find and fix a bug, you can submit it to your fork and send a pull request to us, describing the changes you made. This way we can review your code changes and integrate them into our main branch. Of course, we will credit you and your game in the changelog and at the start page of our Github repository in this case!

Unity Package

Please download the SCILL Unity SDK as a `.unitypackage` from our Github releases page. Download: [SCILL-Unity-SDK.unitypackage](#)

Because Unity does not resolve dependencies automatically when importing Assets as a `.unitypackage`, you will need to ensure that Unity's Newtonsoft-JSON package has been added to your project. To add the Newtonsoft package to your

project, navigate to the Package Manager by using the **Window -> Package Manager** menu. In the Package Manager window, click on the **+** button, choose **Add package from git URL ...** and enter the package reference: **com.unity.nuget.newtonsoft-json**

Download and add source code

You can also download the ZIP file from Github and add the contents of the ZIP file into your Unity Asset folder. This way, our SDK will be “merged” with your project. If you commit your code to a Version Control System (like Git) then you will commit SCILL source files into your own repo.

Which way to choose?

It depends on how you want to leverage SCILL. If you want to use our SDK as is and don't plan to change much yourself, then it's best to use the Package Manager approach. You will profit from new features by just pulling the latest version in the cloned repo.

If you want to heavily change SCILLs code or prefabs, it's perhaps best to merge the code into your codebase. This way it will be perhaps a bit harder to get new features, but if you plan to build on top of our code, you will most likely have built them either yourself or you don't care.

Using the Samples

In the **Sample 1/Scenes** folder we have prepared a couple of scenes you can try out. These samples are linked to an example App. You may see the battle pass or challenges not in a fresh state, as the sample scene uses a hard coded user id for testing. In this case please select the **SCILL Manager** GameObject in the hierarchy of the sample scene and change the **User Id** setting to something random. When leaving the **User Id** field empty, the **SCILLManager** Script will attempt to use your device unique identifier.

After that, you'll see the battle pass or challenges in a fresh state (of course if nobody before chose that user id before).

Replace the API-Key and App-ID in the **SCILL Manager** settings with your own values et voilà: You have a fully working battle pass!

Of course, there are various other ways to add our SDK to Unity:

Personal Challenges

Challenges are a nice way of adding user interaction into your game by letting users pick challenges and pickup rewards once they have achieved those challenges. Personal Challenges are defined like this:

- Need to be unlocked and activated (not necessarily by users)
- Have a duration (in minutes)
- Can be repeatable - users can try over and over again, or they are a one-shot

Follow this Step-by-Step guide to add Personal Challenges to your game.

- 1

Create Challenges

In the [Admin Panel](#) create an app, and API key for the app, create categories and challenges. More info on this topic can be found in our [Guides section](#).
- 2

SCILL Manager

Add the [SCILL Manager](#) prefab into your scene and set your app settings like the **API key** and the **App Id**. Also make sure a **User Id** is set. For testing purposes you can choose anything you like, but in further testing you need to override the [SCILLManager](#) class and implement your own [GetUserId](#) method.
- 3

Add UI

Drag & Drop either [Personal Challenges List](#) or [Personal Challenges Grid](#) into your canvas or use the [Personal Challenges Canvas](#) prefab which comes with a Canvas and implements both versions. Delete the one you don't like and proceed.
- 4

Press Play

Run your application and depending on the user interface you choose you will now see a list of challenges that you have created earlier in the Admin Panel. Play around with unlocking and activating challenges. Everything is already working for you.

- 5 Open Playground
- Select the **SCILL Manager** prefab in your scene. Click on the three dots in the Inspector of the **SCILLManager** script to reveal the component menu. There you'll find the **Open Playground** menu item. A new browser window will be opened with the Playground application setup already with the same values you used in Unity. You should see the same challenges in the same state as you see in Unity. Unlocking or activating a challenge will be immediately reflected in the Unity UI (in play mode of course).
- 6 Test events
- Use the Playground application to setup an event that will trigger one of your challenges you have created and press "Send Event". Make sure the challenge is activated and unlocked. Whenever you send an event, it will progress in Playground and Unity (in play mode) and will show a "Claim" button once the challenge has been achieved.
- 7 Add events
- Below the "Send Event" button you'll see code generated for you that you just need to Copy & Paste in your applications source code. Just select the "Unity" tab which will generate code suited to be used together with the **SCILLManager** prefab.
- 8 Unlock rewards
- Depending on your settings, Challenges have rewards that users can claim. There are various different methods of unlocking rewards to the user, however, the simplest approach is to derive a new class from **SCILLPersonalChallenges** and override the **OnPersonalChallengeRewardClaimed** function.
- 9 Customize
- Create prefab variants of our prefabs and customize colors, font and layout to fit your games UI.

That's it. You have added SCILL Personal Challenges to your game.

Battle Passes

Our Battle Passes are built on top of the challenge system, but they can both co-exist in a game. Fortnite for example combines both elements with weekly challenges to earn coins and battle passes as a monetization tool, delivering great and unique rewards only available in battle passes that users must purchase.

Our battle pass system is very flexible yet easy to implement:

- Battle Passes have a start and end date, which allows you to sell multiple battle passes per user with a seasoning approach.
- Battle Passes are compiled of levels that can have a reward
- Levels are locked by default (the first level is unlocked once the battle pass is unlocked for the user)
- Levels contain one or more challenges which must all be achieved before the level is complete
- Completed levels automatically unlock the next level and users can claim the reward.

It depends on your game how deep you want to integrate our battle pass system. You can either add it as a monetization feature but you can also control the flow of your game using our battle passes. For example you can connect your maps or levels to the battle pass levels, that is, users first need to unlock a certain level in the battle pass before they can play a new map.

Follow this Step-by-Step guide to add Battle Passes to your game. **Please note:** That's a very detailed description of what you should do. You'll have a basic but fully functional battle pass integrated into your game within 1-2 hours!

1 Create Challenges

In the [Admin Panel](#) create an app, and API key for the app, create categories and challenges. More info on this topic can be found in our [Guides section](#). Once you have created the challenges, you can copy them to the Battle Pass system. Of course you can also directly can create battle pass challenges, but it's easier to test personal challenges than battle pass challenges.

2 SCILL Manager

Add the [SCILL Manager](#) prefab into your scene and set your app settings like the [API key](#) and the [App Id](#). Also make sure a [User Id](#) is set. For testing purposes you can choose anything you like, but in further testing you need to override the [SCILLManager](#) class and implement your own [GetUserId](#) method.

- 2 SCILL Battle Pass Manager

Add the [SCILL Battle Pass Manager](#) script onto your manager GameObject. The battle pass manager will load available battle passes and will select the first one. If you want to customize selection, override the class and implement your own selection mechanism.
- 3 Add UI

Drag & Drop the [Battle Pass Canvas](#) into your scene. It will have everything included that is required. Battle Pass levels, unlock button to unlock/purchase the battle pass, active challenges and UI to navigate the levels and to preview rewards.
- 4 Press Play

Run your application and depending on the user interface you choose you will now see the battle pass available in the Admin Panel. If you created multiple battle passes make sure the [SCILL Battle Pass Manager](#) selected the correct one. You can navigate the levels and you can unlock the battle pass. for you.
- 5 Open Playground

Select the [SCILL Manager](#) prefab in your scene. Click on the three dots in the Inspector of the [SCILLManager](#) script to reveal the component menu. There you'll find the [Open Playground](#) menu item. A new browser window will be opened with the Playground application setup already with the same values you used in Unity. You should see the same state as you see in Unity.
- 6 Test events

Use the Playground application to setup an event that will trigger one of your challenges you have created and press "Send Event". Make sure the challenge is activated and unlocked. Whenever you send an event, it will progress in Playground and Unity (in play mode) and will show a "Claim" button once the challenge has been achieved. Start sending events in Playground and notice how levels get unlocked in real time in Unity (in play mode). More info on Playground can be found [here](#).
- 7 Add events

Below the "Send Event" button you'll see code generated for you that you just need to Copy & Paste in your applications source

code. Just select the “Unity” tab which will generate code suited to be used together with the [SCILLManager](#) prefab.

- 8 Unlock rewards Implement a new class in Unity and add a listener on the [OnBattlePassLevelRewardClaimed](#) event available in the [SCILLBattlePassManager](#) class. Example code can be found in the [SCILL Battle Pass Manager](#) reference.
- 9 Customize Create prefab variants of our prefabs and customize colors, font and layout to fit your games UI.

That's it. You have added SCILL Battle Passes to your game.

Leaderboards

Leaderboards are a must have for each and every game. It's the simplest and yet one of the most powerful tools for user retention. Core features of our leaderboards:

- Driven by the events you already send to SCILL
- Define the sort order, events types to process - that's it

Follow this Step-by-Step guide to add Personal Challenges to your game.

- 1 Create
 Leaderboard In the [Admin Panel](#) create an app, and API key for the app and create a leaderboard. More info on this topic can be found in our [Guides section](#).
- 2 SCILL Manager Add the [SCILL Manager](#) prefab into your scene and set your app settings like the [API key](#) and the [App Id](#). Also make sure a [User Id](#) is set. For testing purposes you can choose anything you like, but in further testing you need to override the [SCILLManager](#) class and implement your own [GetUserId](#) method.

- 3 Add UI

Drag & Drop either [Leaderboard](#) into your canvas or use the [Leaderboard Canvas](#) prefab which comes with a predefined canvas.
- 3 Set your
 leaderboard id

In the inspector of leaderboard instance in the [SCILLLeaderboard](#) script, set the [Leaderboard ID](#) to the id of the leaderboard. See [SCILLLeaderboard](#) for more info on that.
- 4 Press Play

That's it. You have a fully functional leaderboard. If it's empty, make sure you send some events that trigger the leaderboard (i.e. the event names and the event name defined when creating the leaderboard as well as any meta data must match).
- 5 Open
 Playground

Select the [SCILL Manager](#) prefab in your scene. Click on the three dots in the Inspector of the [SCILLManager](#) script to reveal the component menu. There you'll find the [Open Playground](#) menu item. A new browser window will be opened with the Playground application setup already with the same values you used in Unity. You should see the same leaderboards in the same state as you see in Unity.
- 6 Test events

Use the Playground application to define an event that will trigger your leaderboard you have created and press "Send Event".
- 7 Add events

Below the "Send Event" button you'll see code generated for you that you just need to Copy & Paste in your applications source code. Just select the "Unity" tab which will generate code suited to be used together with the [SCILLManager](#) prefab.
- 8 Customize

Create prefab variants of our prefabs and customize colors, font and layout to fit your games UI.

That's it. You have added a SCILL leaderboard to your game.

Prefabs

The SCILL Unity SDK comes with many prefabs that you can use to implement a Personal Challenges and/or Battle Pass UI very quickly.

All classes handle the complete data transfer with the SCILL cloud and will trigger delegation events that other parts of the SCILL Unity SDK listen to. Therefore, you don't need to connect all the pieces together in the Unity inspector. Just drop the prefabs into your scene, and they will work immediately.

Tip

We provide two different styles as a starting point. Style 1 is a flat, modern style with a layout that can be easily adjusted to your liking. Just replace a couple of "border images" to adapt to your type of game.

The best approach for using Style 1 is to increment the camera depth to 1 or above and to load the battle pass scene with `SceneManager.LoadScene("Battle Pass", LoadSceneMode.Additive)`. This way, the battle pass scene "overlays" your game scene but as the camera has a higher depth than your game camera (make sure it's value is larger) only the battle pass scene is visible. If user want to go back, just delete the Battle Pass Scenes root node to remove the battle pass UI from the screen.

Style 2 uses `RenderTarget` for showing the rewards and can be shown as a canvas in your game. No need for loading the scene there.

SCILL Manager

Drop this item into your scene and set your App settings in the inspector to setup SCILL within your Unity project. This GameObject will be set to be persistent using Unitys `DontDestroyOnLoad` function.

This prefab implements the following scripts:

[SCILLManager](#)

Handles generating the Access Token and User ID. You need to override this class to provide your secure access token implementation and user id generation. However, for testing purposes you can just use the prefab as it is.

[SCILLBattlePassManager](#)

Loads available battle passes from the SCILL cloud and selects the first battle pass it finds. This `SelectedBattlePass` will be used in the other scripts as the battle pass to show. Override this class with your own class if you have multiple battle passes or want to implement your own logic of selecting a battle pass.

Battle Pass Canvas

This prefab is complete Canvas with Battle Pass User Interface that will render the `SelectedBattlePass` of the [SCILLBattlePassManager](#).

This prefab compiles all battle pass related prefabs into one so that you can easily see how to use them.

- [Battle Pass Challenges](#)
- [Battle Pass](#)
- [Battle Pass Header](#)

This prefab uses other prefabs as building blocks for the UI

[Battle Pass Level](#)

Renders a single level of the battle pass. It's available in three different variants (current, completed and locked) and are set as prefabs in the [SCILLBattlePassLevels](#) component.

This prefab implements the following scripts:

[SCILLBattlePass](#)

Renders UI to show info about the battle pass and provides an unlock button that will offer users to unlock/purchase the battle pass.

[SCILLBattlePassLevels](#)

Shows `itemsPerPage` levels available in the battle pass and will offer UI to paginate through the challenges. It will instantiate [Battle Pass Level](#) prefabs which will render UI to select a level and show the available rewards using [Reward Icon](#) prefabs.

[SCILLBattlePassLevelChallenges](#)

Renders the current active list of challenges linked to the currently active level. Typically you will show this list somewhere else in your UI, for example in your games HUD. You can just Drag & Drop this item in your games HUD as this works independently.

Battle Pass Header

Renders the current level, pagination buttons and the name of the battle pass. It's used as a building block in the [Battle Pass](#) prefab.

Battle Pass Challenges

Drop this prefab into your own Canvas to render the active list of challenges. You don't need to connect anything, but you must make sure that the [SCILL Manager](#) prefab is also dropped and correctly setup in your scene.

Some scripts in this prefab require setting other prefabs as building blocks:

[Challenge](#)

A single battle pass challenge with name, and progress slider. It's instantiated for each active challenge.

This prefab implements the following scripts:

[SCILLBattlePassLevelChallenges](#)

Renders the current active list of challenges linked to the currently active level. Typically you will show this list somewhere else in your UI, for example in your games HUD. You can just Drag & Drop this item in your games HUD as this works independently.

Battle Pass Level

Two additional prefab variants are available: [Battle Pass Level Current](#) and [Battle Pass Level Locked](#) of these prefabs. They implement a single battle pass level and will render the levels number and instantiate [Reward Icon](#) prefabs for each reward available in the battle pass level.

Some scripts in this prefab require setting other prefabs as building blocks:

[Reward Icon](#)

Renders an icon of the reward and shows if the reward is locked or claimed.

This prefab implements the following scripts:

[SCILLBattlePassLevel](#)

Shows the progress of the level and handles clicks to trigger previews of the reward using the [Reward Preview](#) component.

Challenge

Renders a single battle pass challenge and shows a progress bar and or counter/goal indicator and the name of the challenge. It should be attached to the `challengePrefab` property of the [SCILLBattlePassLevelChallenges](#) script.

[SCILLBattlePassChallengeItem](#)

Shows a battle pass challenge and automatically updates the progress in realtime.

Reward Preview

This is used to render [SCILLReward](#) assets whenever a level is clicked and a reward with the name set in the Admin Panel is available in a Unity `Resources` folder.

This prefab implements the following scripts:

[SCILLRewardPreview](#)

Loads [SCILLReward](#) assets and shows a UI displaying name, description and a 3D model preview of the reward if available.

Warning

The reward preview will instantiate the 3D model preview prefab in a “photo box”. In Style 2 we show you how to render this model into a render texture to be used in the UI: The photo box has a camera as a child that renders into a Unity `RenderTexture` that is used as the source for a `RawImage` item in the reward preview prefab. As we don’t want these models in the photo box to show up in the game, you need to do the following:

1. Create a new Unity layer and name it `3D UI` (or whatever you like)
2. In the hierarchy of the Reward Preview prefab find the `Reward Camera` GameObject.
3. Set the `Culling Mask` setting of that camera to only render `3D UI` layers.
4. In your main games camera, exclude the `3D UI` setting from the `Culling Mask`
5. Make sure all your prefabs you set in the [SCILLReward](#) asset are on the `3D UI` layer.

Reward Icon

Is only used as a building block in the [Battle Pass Level](#) prefab and shows an icon of the reward and indicates with small icons if the reward is locked or has been already claimed.

This prefab implements the following scripts:

[SCILLBattlePassRewardIcon](#)

Loads [SCILLReward](#) assets and shows the icon of the reward.

Personal Challenges Canvas

This prefab is complete Canvas with Personal Challenge List and Personal Challenge Grid.

This prefab compiles all personal challenges related prefabs into one so that you can easily see how to use them.

- [Personal Challenges List](#)
- [Personal Challenges Grid](#)

This prefab uses other prefabs as building blocks for the UI:

[Challenge](#)

Renders a single challenge with user interface elements to unlock and activate a challenge. It also shows progress in a progress bar and handles all events. It's use interface is a list, so it should be used as the `challengePrefab` in the [SCILLPersonalChallenges](#) component.

[Category](#)

Renders a category header element that supports expansion and collapsing challenges within that category. Use as the `categoryPrefab` in the [SCILLPersonalChallenges](#) component. This prefab is designed to be used in lists.

[Challenge Grid](#)

Renders a single challenge with user interface elements to unlock and activate a challenge. It also shows progress in a progress bar and handles all events. It's use interface is square item. It should be used as the `challengePrefab` in the [SCILLPersonalChallenges](#) component.

[Category Grid](#)

Renders a category header element that supports expansion and collapsing challenges within that category. Use as the `categoryPrefab` in the

[SCILLPersonalChallenges](#) component. This prefab is designed to be used in grids.

This prefab implements the following scripts:

[SCILLPersonalChallenges](#)

Loads personal challenges for the user defined in the [SCILL Manager](#) and instantiates the `categoryPrefab` and adds that as children. The categories will then add the challenges as instances of the `challengePrefab`.

Personal Challenges Grid

This prefab renders a grid of [SCILLChallengeItem](#) prefabs as children. Use the [Category Grid](#) prefabs as an example as a building block for the categories.

This prefab uses other prefabs as building blocks for the UI:

[Category Grid](#)

Renders a category header element that supports expansion and collapsing challenges within that category. Use as the `categoryPrefab` in the [SCILLPersonalChallenges](#) component. This prefab is designed to be used in grids.

This prefab implements the following scripts:

[SCILLPersonalChallenges](#)

Loads personal challenges for the user defined in the [SCILL Manager](#) and instantiates the `categoryPrefab` and adds that as children. The categories will then add the challenges as instances of the `challengePrefab`.

Personal Challenges List

This prefab renders a list of [SCILLChallengeItem](#) prefabs as children. Use the [Category](#) prefabs as an example as a building block for the categories.

This prefab uses other prefabs as building blocks for the UI:

[Category](#)

Renders a category header element that supports expansion and collapsing challenges within that category. Use as the `categoryPrefab` in the [SCILLPersonalChallenges](#) component. This prefab is designed to be used in lists.

This prefab implements the following scripts:

[SCILLPersonalChallenges](#)

Loads personal challenges for the user defined in the [SCILL Manager](#) and instantiates the `categoryPrefab` and adds that as children. The categories will then add the challenges as instances of the `challengePrefab`.

Category

This is the building block used as a prefab in the [Personal Challenges List](#). It renders the name of the category and UI to expand and collapse the challenges linked to this category.

This prefab uses other prefabs as building blocks for the UI:

[Challenge](#)

Renders a single challenge with user interface elements to unlock and activate a challenge. It also shows progress in a progress bar and handles all events.

This prefab implements the following scripts:

[SCILLCategoryItem](#)

Renders a category and adds or deletes prefabs set in the `challengePrefab` property and adds them as children. This prefab has a `VerticalLayout` attached that will add [Challenge](#) prefabs.

Category Grid

This is the building block used as a prefab in the [Personal Challenges Grid](#). It renders the name of the category and UI to expand and collapse the challenges linked to this category.

This prefab uses other prefabs as building blocks for the UI:

[Challenge Grid](#)

Renders a single challenge with user interface elements to unlock and activate a challenge. It also shows progress in a progress bar and handles all events.

This prefab implements the following scripts:

[SCILLCategoryItem](#)

Renders a category and adds or deletes prefabs set in the `challengePrefab` property and adds them as children. This prefab has a `VerticalLayout` attached that will add [Challenge](#) prefabs.

Challenge

A building block used to render a challenge item as a list. It should be used as the `challengePrefab` setting in the [SCILLCategoryItem](#).

This prefab implements the following scripts:

[SCILLChallengeItem](#)

Renders a challenge and provides user interface to unlock, activate and claim challenges. It's user interface is designed to be added as child of a `VerticalLayout` group so that they form a list.

Challenge Grid

A building block used to render a challenge item as a grid. It should be used as the `challengePrefab` setting in the [SCILLCategoryItem](#).

This prefab implements the following scripts:

[SCILLChallengeItem](#)

Renders a challenge and provides user interface to unlock, activate and claim challenges. It's user interface is designed to be added as child of a `GridLayout` group so that they form a grid.

Leaderboard Canvas

This prefab is complete Canvas with a fully functional leaderboard. This is basically just a Canvas that contains the [Leaderboard](#) prefab.

Leaderboard

Creates an infinite scrolling leaderboard UI with regular updates. In uses building blocks for the ranking items and offers various options for customization.

This prefab implements the following scripts:

[SCILLLeaderboard](#)

Loads leaderboard rankings instantiates the prefabs for the rankings and adds them as children.

The prefab already contains the hierarchy required for the leaderboard and also uses a [ScrollRect](#) for the infinite scrolling functionality.

Just drop it inside a Canvas and set the [Leaderboard Id](#) which you copy in the Admin Panel for the leaderboard you created there.

This prefab uses other prefabs as building blocks for the UI:

[Ranking](#)

This is the prefab used for the rankings in the leaderboard. It contains username, avatar image, rank and score items.

[Ranking User](#)

A prefab variant of [Ranking](#) that has different color to highlight the users ranking.

[Ranking Basic](#)

A prefab variant of [Ranking](#) that has different color and is used for all rankings that are not related to the user and are not in the top ranks.

Ranking

A building block used to render a ranking item in a leaderboard.

This prefab implements the following scripts:

[SCILLLeaderboardRankingItem](#)

Renders a ranking item.

Ranking User

A building block used to render a ranking item in a leaderboard for the user. It has highlighted color to help the user figuring out his position in the leaderboard quickly.

This prefab implements the following scripts:

[SCILLLeaderboardRankingItem](#)

Renders a ranking item.

Ranking Basic

A building block used to render a ranking item in a leaderboard for the user. It has basic color and can be used for all default rankings.

This prefab implements the following scripts:

[SCILLLeaderboardRankingItem](#)

Renders a ranking item.

Classes

The Unity SDK consists of various classes handling data flow and UI.

SCILLManager

This class is designed as a “Singleton” and uses `DontDestroyOnLoad` to make sure, that the class instance persists even after scene changes. It provides access to the SCILL SDK APIs and provides some convenience functions to make it easier to work with SCILL from your own code.

Create an empty `GameObject` in your Scene and add this script. Then set your API key, AppId and the language. If your app supports multiple languages, you can use the `SetLanguage` method to set the language via Script.

You can also choose an environment: You should leave that in `Production`. Sometimes, when working closely with our development team we might ask you to choose a different value but usually `Production` is the correct setting.

Production:

In production, you should not use this class by providing your API key! Instead you should override this class and override these functions:

- `GenerateAccessToken`
- `GetUserId`

More info on this topic can be found here: [User IDs and Access Tokens](#). To summarize this topic: API keys are very powerful and therefore should be kept secret. Exposing things on client side aren't secret, as everyone with a Debugger will be able to extract the API key very quickly from your code. The only way to keep the API key secret is to use it on server side - which can be a cloud function (AWS Lambda, Google Cloud Function, ...) or your own HTTP server. Recommended procedure is to create that cloud function for example in NodeJS and to use the SCILL JavaScript SDK to generate the access token for the user.

Your own Subclass of `SCILLManager` will then override the functions listed above to load the access token from your backend, instead of generating it on client side with the API key as we do in the default implementation of this class.

SCILLBattlePassManager

This class is designed as a "Singleton" and should be attached to the same `GameObject` that you have `SCILLManager` attached. It will completely manage battle passes. It selects the first available active battle pass and selects that battle pass.

This class is derived from the `SCILLThreadSafety` class.

Add this component to the same `GameObject` that has the `SCILLManager` attached. If you want to customize Battle Pass selection, derive a new class from this class and override `SelectBattlePass` function which receives the array of available battle passes from the backend to return the selected battle pass.

This class handles all data required for displaying and acting on claimed battle pass level rewards. Use the various delegates to connect your own classes to the manager. In this documentation we provide some example code. Our code and all prefabs connect to the manager with these delegates. There is no need to connect prefabs or other things together in Unity - just drop in the prefabs, and they will work automatically as they just listen on the delegates and update their UI whenever data changes and respective delegates are called.

SCILLBattlePass

Add this component to a Unity `GameObject`. It will create a UI for the `SelectedBattlePass` of the `SCILLBattlePassManager`. `SCILLBattlePass` connects to

these delegates of `SCILLBattlePassManager` to get notifications whenever the battle pass changes and updates UI accordingly:

- `OnBattlePassUpdatedFromServer`
- `OnBattlePassLevelsUpdatedFromServer`

The `SCILLBattlePass` will only handle battle pass related UI, like unlocking/purchase buttons.. `SCILLBattlePassLevels` is responsible to render the levels of the battle pass and do the proper levels pagination.

The best way to get started is to drop the `BattlePass prefab` into a Canvas. This prefab already has prepared the connections and hierarchy for a battle pass.

SCILLBattlePassLevels

This component will instantiate level UI prefabs and will add them as children. Use some sort of auto layout like the vertical, horizontal or grid layout components to build the user interface.

This component will also handle pagination and exposes functions to be connected to buttons click event and properties that you can use to connect buttons that will be hidden and shown depending on the current pagination state.

There are three different types of levels in a battle pass:

- Locked
- Unlocked but not complete (current level)
- Completed

You need to connect prefabs from your Asset browser for each of these states. It's easier to build three different prefabs rather than having one very complex prefab for these states. Best way is to build a level prefab and then creating prefab variants for the other states.

Please note: The level prefab must have a `UnityEngine.UI.Button` element somewhere. After instantiating the level prefab it will search for a `Button` component in the level prefab and will attach a listener to that button. Whenever the level is clicked, the delegate `OnSelectedBattlePassLevelChanged` will be triggered.

The `SCILLRewardPreview` will listen on that event and will render a preview of the reward (if available) of the selected level.

SCILLBattlePassLevel

Implements a level component in the Battle Pass User Interface. It will embed `SCILLBattlePassRewardIcon` prefabs to show available rewards for this level. Create a prefab with this component attached and set as the `levelPrefab`, `currentLevelPrefab` and `lockedLevelPrefab` in the `SCILLBattlePassLevels` component.

SCILLBattlePassLevelChallenges

Battle Passes have a number of levels and each level has challenges attached that need to be achieved before the level is completed and the next level is unlocked.

Use this component to render a list of active challenges, i.e. the challenges from the current level. Set the `challengePrefab` to a prefab that has the `SCILLBattlePassChallengeItem` item attached.

Only active challenges will be rendered, i.e. those challenges that have a `in-progress` type. It is intended to be shown in the games HUD and to be always visible, so gamers can quickly see what they should do next. However, you can override this class and implement basic logic to your liking, too.

SCILLBattlePassChallengeItem

This component will handle UI for a challenge of a battle pass level. It will update progress whenever the challenge progress changes.

It listens on the `OnBattlePassChallengeUpdate` delegate of the `SCILLBattlePassManager` and will update the progress accordingly.

Please note: This class is intended to be used with the `SCILLBattlePassLevels` component (set as prefab for a challenge). As `SCILLBattlePassLevels` does hide all challenges that are not in progress, this class does not handle challenges that are not `in-progress` correctly, i.e. the progress bar is always visible, as the goal. If

you want to implement special behavior, derive this class and override the `UpdateUI` function to adjust the UI accordingly.

SCILLBattlePassRewardIcon

This component will handle a reward available for a battle pass level. Every level can have a reward and it typically is represented with an icon and some sort of state information. Often rewards can be clicked to show a nice preview of the reward.

Create a prefab with this component attached to the root Game Object and build a nice UI below it to render an icon with lock and claimed state icons. Connect that prefab to the `rewardIconPrefab` setting of `SCILLBattlePassLevel`. The level component will then instantiate this prefab automatically.

You need to create a `SCILLReward` asset and set the reward to the name of this reward asset in the Admin Panel. This class will load this resource (make sure its in a `Resources` folder in Unity) and will take the name and image from this reward asset to set the UI elements connected.

SCILLPagination

This class handles pagination for the `SCILLBattlePassLevels` class. Not all levels fit into the screen, so they are paginated. Use this class to manage that.

This class offers two types of pagination. Via forward and backward buttons. And via pagination dots, where each dot stands for one page. This allows the user to quickly navigate to pages further away.

SCILLReward

This class is part of the SCILL reward system. Use it to create reward assets that describe the reward. You can also attach a prefab (3D model) that will be used to render a 3D model in the reward preview model.

Warning

Make sure you place your reward assets in a `Resources` folder as they will be loaded at runtime! Also make sure that you place all reward assets like the 3D model preview prefab and the sprite in a `Resources` folder.

SCILLRewardPreview

This component will listen to the

`SCILLBattlePassLevels.OnSelectedBattlePassLevelChanged` event to get notified whenever the user clicks on a `SCILLBattlePassLevel` component.

When a reward is set for this level (`reward_amount` of the `BattlePass` object) it will try to load `SCILLReward` asset with that name. If the reward is available it will show its UI and will set image, name and description to connected UI elements.

The reward preview will instantiate the prefab set in the `SCILLReward` asset as child into this `GameObject`. The default preview has this hierarchy:

- Reward photo box
 - Reward camera
 - Photo Box

`Photo Box` is connected to the `photoBox` property and models will be instantiated into this transform. The reward camera renders the model into a Render Texture that is used in the Reward Preview UI as a `RawImage` field.

Important: You need to set the layer that the reward camera renders to the same layer that you set in the prefab. This layer should be included from the main games camera.

As every game has a different layer structure we cannot set a specific layer for SCILL. You need to organize that yourself.

SCILLPersonalChallenges

This component is derived from the `SCILLThreadSafety` class. It handles all communication with the SCILL backend to load and update personal challenges in real time. It also implements user interfaces.

This class does three things:

- Load personal challenges with the `SCILLClient.GetPersonalChallenges` method
- Instantiate the prefab set in `categoryPrefab` property for each `ChallengeCategory` object contained in the `response` and add as child to the transform.
- Listen on server-side changes and update the challenges state in the UI

The `categoryPrefab` must have a `SCILLCategoryItem` attached that will create instances of the `challengePrefab` and add them as childs to category game object instance or if provided to the `SCILLCategoryItem` transform provided in the challenge category prefab.

In the end, you'll have this hierarchy:

- Container
 - Challenge Category 1
 - Header
 - Challenge 1
 - Challenge 2
 - ...
 - Challenge Category 2
 - Header
 - Challenges Container
 - Challenge 1
 - Challenge 2
 - ..
 - ...

SCILLCategoryItem

Challenges are grouped into categories. Attach this script to a prefab that you set as the `categoryPrefab` in the `SCILLPersonalChallenges` component. Categories instantiate `challengePrefab` objects that are either set directly in this class or it's taken from the parent `SCILLPersonalChallenges` component.

Category UI support expanding and collapsing the UI. Use the `expanded` setting to set the expansion state at start.

SCILLChallengeItem

This class implements user interface for a personal challenge. Attach it to a game object and connect properties with user interface elements. You need to create a prefab and connect to the `challengePrefab` in the `SCILLPersonalChallenges` component.

SCILLLeaderboard

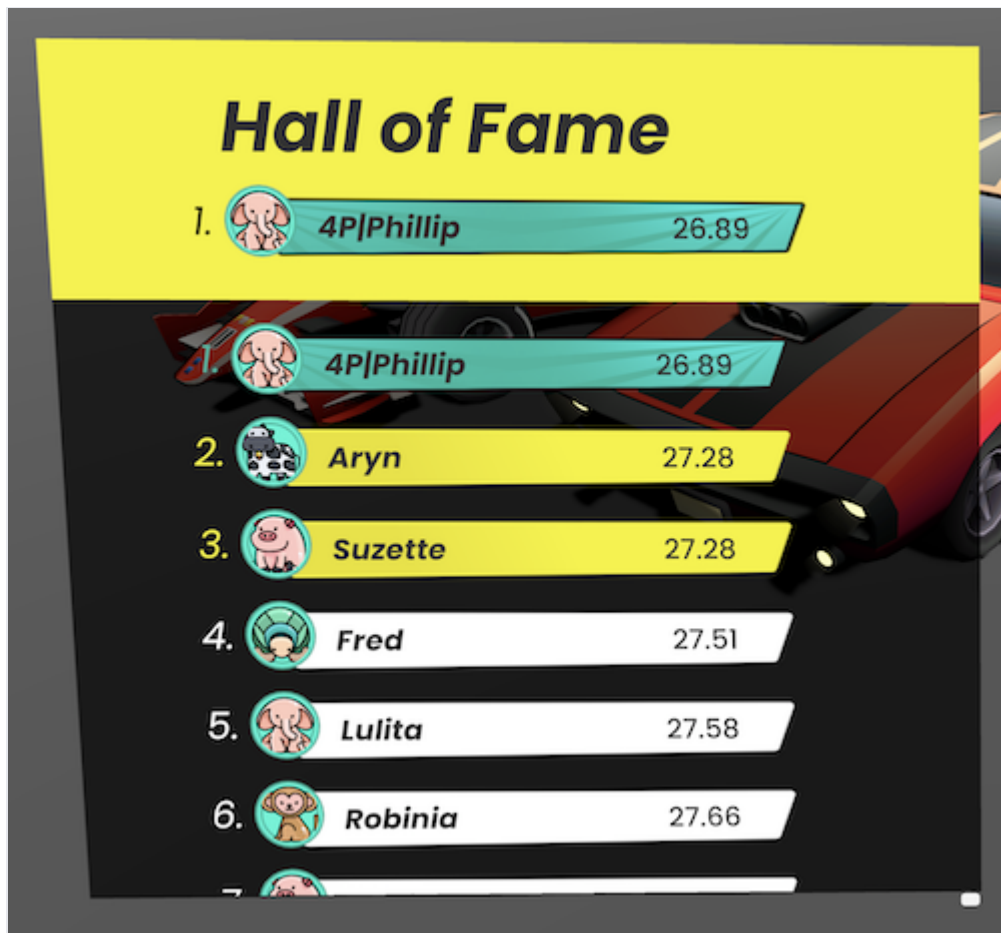
This component is derived from the `SCILLThreadSafety` class. It handles all communication with the SCILL backend to load and update personal challenges in real time. It also implements user interfaces.

This class does two things:

- Load Leaderboard provided by the leaderboard id and update it every 10 seconds.
- Instantiate one of the ranking item prefabs (i.e. `userRankingPrefab`) property for each `LeaderboardRanking` object contained in the `response` and add as child to the `rankingsContainer` transform.

The prefabs like `userRankingPrefab` must have a `SCILLLeaderboardRankingItem` attached that handles UI for each ranking item.

Sometimes, it's not easy for the player to find himself in the leaderboard. To solve that, we added support for the users position to be shown in the header of the leaderboard UI (or somewhere else that makes sense for your own game).



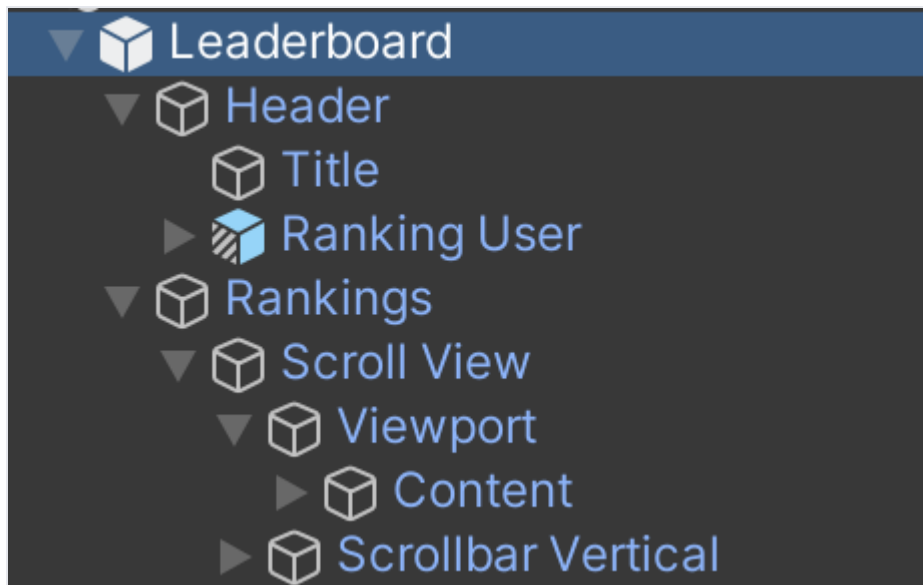
An example leaderboard UI

Use the [userRanking](#) inspector property to connect an instance of the [userRankingPrefab](#). The leaderboard will update the values regularly for the individual user and the whole leaderboard.

Usage

This leaderboard implements infinite scrolling via a [ScrollRect](#) which should be below the hierarchy of this component. Build this hierarchy as seen in the image at the right side.

Attach the [SCILLLeaderboard](#) script to the [Leaderboard](#) game object and adjust inspector settings as seen below.



The recommend hierarchy for leaderboards

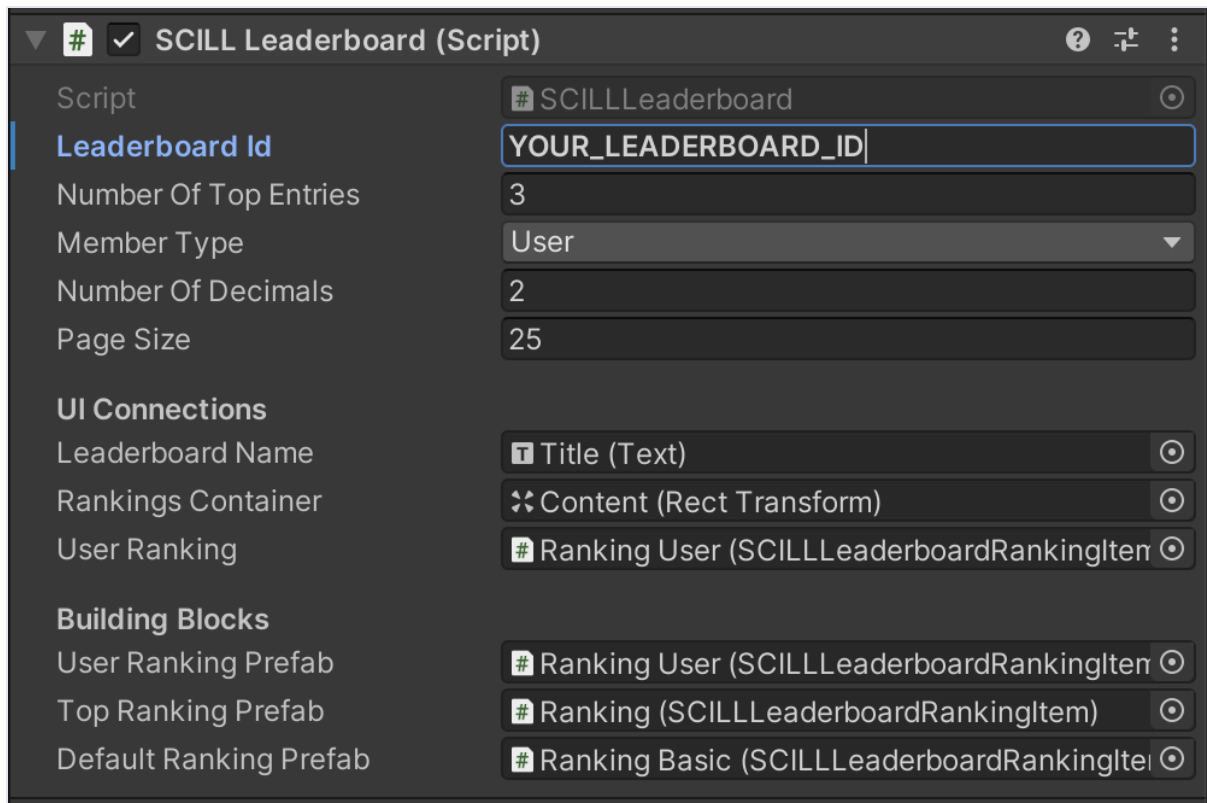
First, you need to enter your leaderboard id (you can also set this via script). This leaderboard id can be found in the [Admin Panel](#). Then adjust the settings for page size and number of top entries as described below.

Next, you need to connect a [Unity.UI.Text](#) component to [Leaderboard Name](#) so that the name of the leaderboard can be set dynamically from the values you entered when creating the leaderboard in the Admin Panel.

The [Rankings Container](#) is a transform that prefabs for rankings will be added to. In this case we chose that [Content](#) item from the Scroll View which has a [VerticalLayoutGroup](#) to align the items vertically.

[User Ranking](#) is connected to the user ranking prefab which has been put in the leaderboard header and which will be updated regularly with the users position in the leaderboard - even if the user is not within the currently loaded rankings.

Last but not least, connect prefabs for the ranking items. These prefabs need to have a [SCILLLeaderboardRankingItem](#) attached which will handle updating the UI.



The recommend hierarchy for leaderboards

Available Prefabs

In the SCILL SDK we already provide ready to be used prefabs of ranking items and a fully functional leaderboard with infinite scrolling feature.

SCILLLeaderboardManager

This is a utility script for subscribing to realtime updates when a specific leaderboard was first loaded from the SCILL backend and for subscribing to realtime changes to the leaderboard.

When being enabled, this script will request a full leaderboard reload and call the `OnLeaderboardRankingLoaded` unity event on response. Changes to the leaderboard will be broadcasted to the `OnLeaderboardRankingChanged` unity event. Use this script if you'd like to have access to those events over the inspector or via code.

Please note that `SCILLLeaderboardManager` is designed to work with leaderboards of type `user` only, not for type `teams`.

SCILLLeaderboardRankingItem

This class implements user interface for a leaderboard ranking (i.e. rank position in a leaderboard table). It gets a **LeaderboardRanking** via its **ranking** property and updates text and images provided in properties.