Debug Toolkit for Unity (UDT)

> Thanks for using the UDT. Unity 6+ is supported for this asset.

This documentation aims to get you ready to use and modify UDT for your awesome projects.

The current version of the toolkit 1.0.

If you are looking for the last patch note please check the Patch Notes. If you are looking for features in development take a look at the What's Next section.

Join us on Discord to directly talk to the developpement team.

Features

UDT is composed of many elements :

- A modular console, for which can create custom commands but also log anything you want at runtime. Maybe you wish to have a simple way to kill your player, teleport to some place, show all the colliders that are in the scene or maybe you want to go frame by frame to know what is happening. Now its a simple command line.
- A system of optimized runtime Gizmos with an easy to use API. It makes your debugging way easier. Control them with the console, or as a standalone feature.
- A FreeCam to inspect your scene at runtime. With a simple command in the console activate or deactivate the FreeCam and navigate through your scene.
- A metrics system to now your fps, the number of batches, and your tris/vert usage.
- A runtime navmesh debugger.

Table of Contents

Introduction to UDT

This toolkit focuses on runtime debugging and offers a variety of features to improve your workflow and help you resolve bugs efficiently.

The asset can also be used seamlessly within the Unity editor, and many features are reusable directly in your games.

We have many exciting improvements planned for future releases. To learn more about upcoming features, please visit the What's Next section.

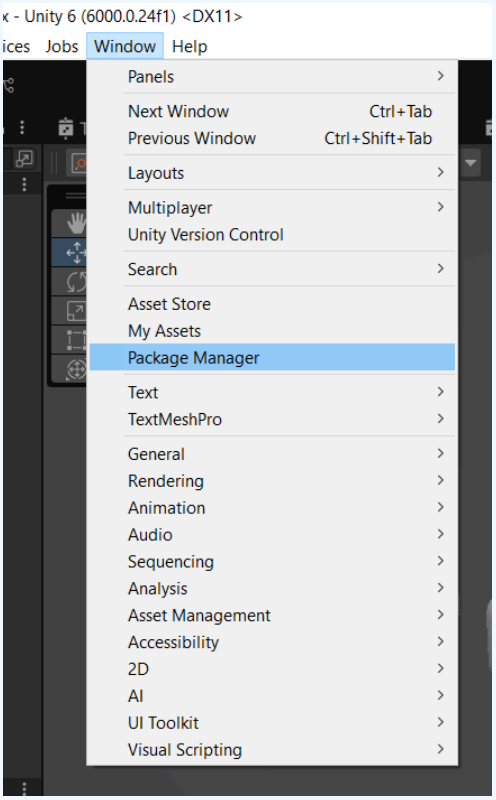
In the following sections, you will discover how to import UDT into your project and get started using its powerful features.

Install UDT

> Make sure that you are using Unity 6 or above.

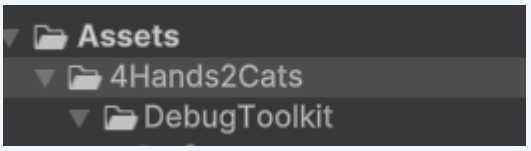
Package manager

Get into the unity package manager at the top of the Unity Engine window and look for UDT in your assets



Click on the **download** button at the bottom right of the menu.

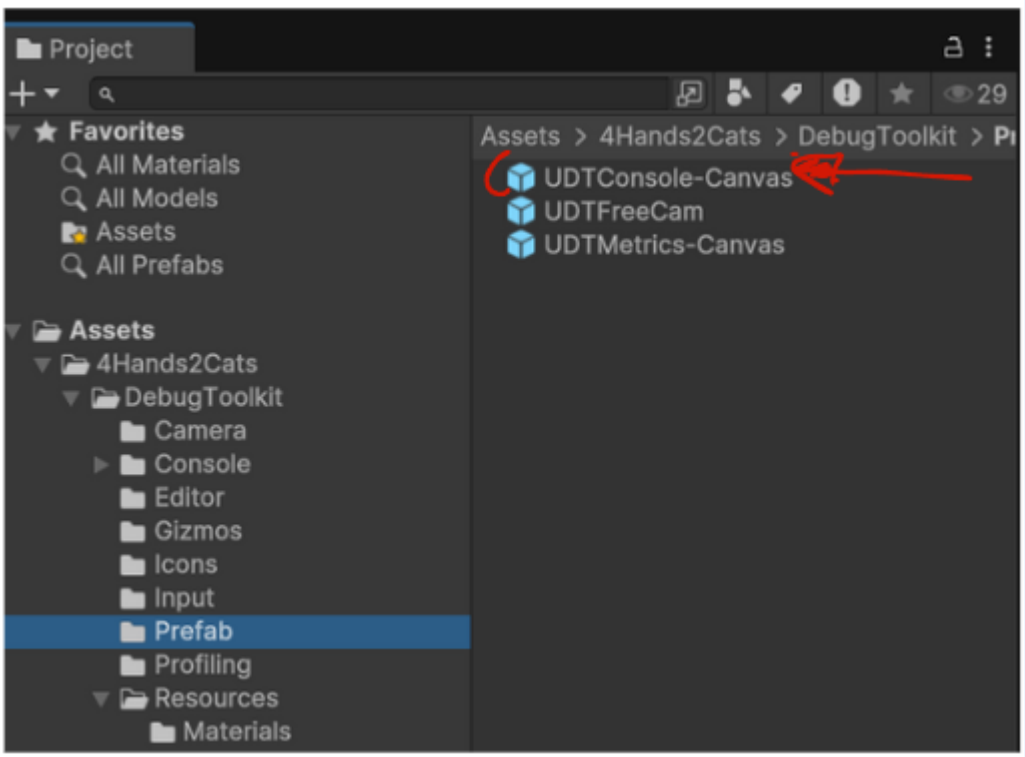
Once the files are downloaded, import them in your project by clicking on the **Import** button.



The toolkit is in the 4Hands2Cats folder. In the future if we do more assets they'll install in this folder as well.

Quick Start

As you just seen in the demo scene, the toolkit is pretty simple to include in your project. You just have to drag and drop the **UDTConsole-Standalone** prefab from the prefab folder inside your scene. Alternatively, use the **UDTConsole-Mobile** for your mobile project (Android/IOS).



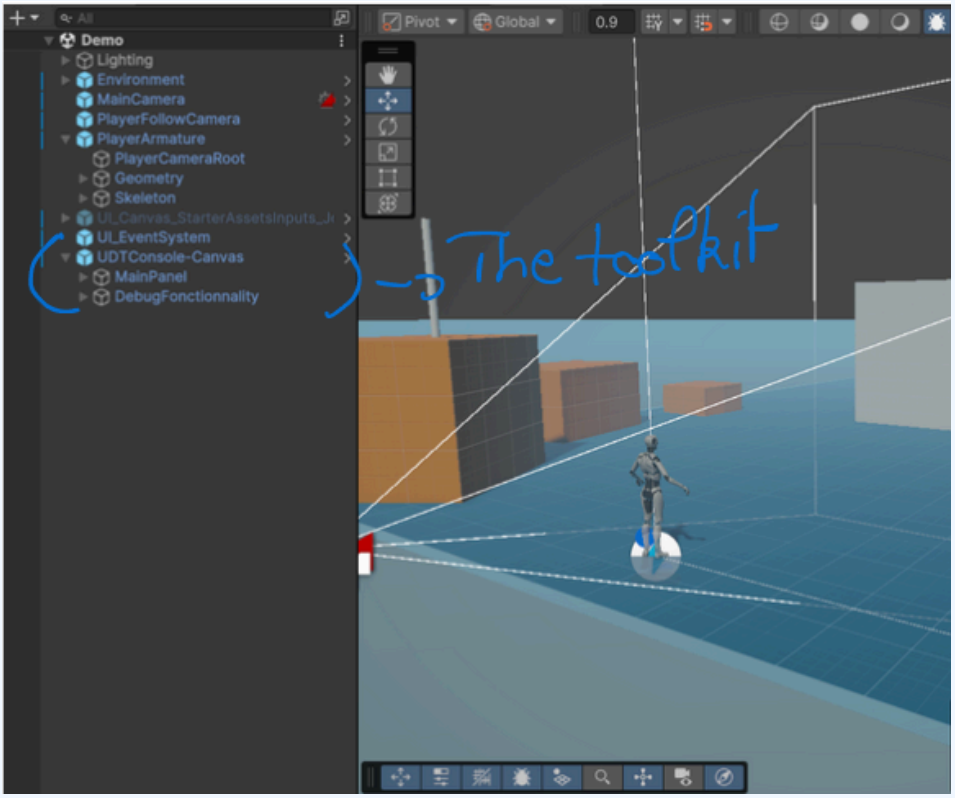
Don't forget to remove it for production builds.

Demo Scene Tour

We provide three demo scenes to help you quickly explore and test the features of the toolkit in different contexts:

1. Third-Person 3D Demo

Based on Unity's third-person template, this scene showcases our debug toolkit in a 3D environment. It includes character movement, camera follow, and in-world interaction. To use this demo scene just press play. The Console and the all the other features are on the **F12** key. Controls: Use **WSDQ** to move the character, the mouse to control the camera, and press F12 to toggle the debug console and other tools.



2. 2D Demo Scene

This lightweight scene is designed to demonstrate the toolkit in a 2D context. It’s ideal for side-scrollers, top-down games, or any 2D project. To use this demo scene just press play. The Console and the all the other features are on the **F12** key.

3. Mobile Demo Scene

Tailored for touchscreen interaction, this scene demonstrates how the toolkit behaves on mobile devices. It includes on-screen controls and supports both portrait and landscape modes. To use this demo scene just press play. The Console and the all the other features are on the **F12** key. Controls: Use the on screen controller to move the character and control the camera.

Each scene is plug-and-play ready—just open the one you need and press Play in the Unity Editor. You can easily customize them or use them as a starting point for your own testing environments.

Console

The In-game console is the central piece of UDT. It controls every features of the toolkit and more.

> This section is the documentation of the usage of the console as is. In the customization section you'll find explanations on how to make your own commands for the console.

Refer to the Quick Start section to learn how to enable the in game console for your project.

Built in commands

There is a set of built in commands that you can use to control the toolkit and enable/disable features.

Like in any console you can use the top and bottom arrow of your keyboard to navigate through the command you've already written.

Each feature has its family of commands.

Help

```
help : To show how to use everycommpands in the toolkit
```

Metrics

```
metrics enable/disable : To show or hide the metrics
```

Freecam

```
freecam enable/disable : To use the frecam. (It instantiate the cam and deletes it)
```

Lighting

```
    light enable/disable : to toggle the directional light.
```

Shadows

```
    shadows enable/disable : to toggle the shadows.
```

Collider

```
     collider enable/disable : to show all the collider or hide them all (This cost a lot on big scenes)
```

Gizmos

```
    gizmos enable/disable : to show all gizmos or not
```

Navmesh debug

```
    navA gizmos enable/disable : To draw the gizmos of the navAgent pathing

      navA info enable/disable : To show the info about the pathing of the navAgent

    navA all enable/disable : To activate all commands
```

Time

```
    time 0-100 : to set the time scale. Note that the value can be between 0 and 100.

    frame 0-100 : to jump to a frame. Do you wish to pause in exactly 42 frames? Now you can.
```

Graphism

```
graphics low/medium/high/ultra : to change the quality of the graphisme.
```

> [Note : The scriptable object of this command will have to be changed if your quality settings are not following the standard ones of an URP project (*verylow/low/medium/veryhigh/ultra*)].

Log in the console only

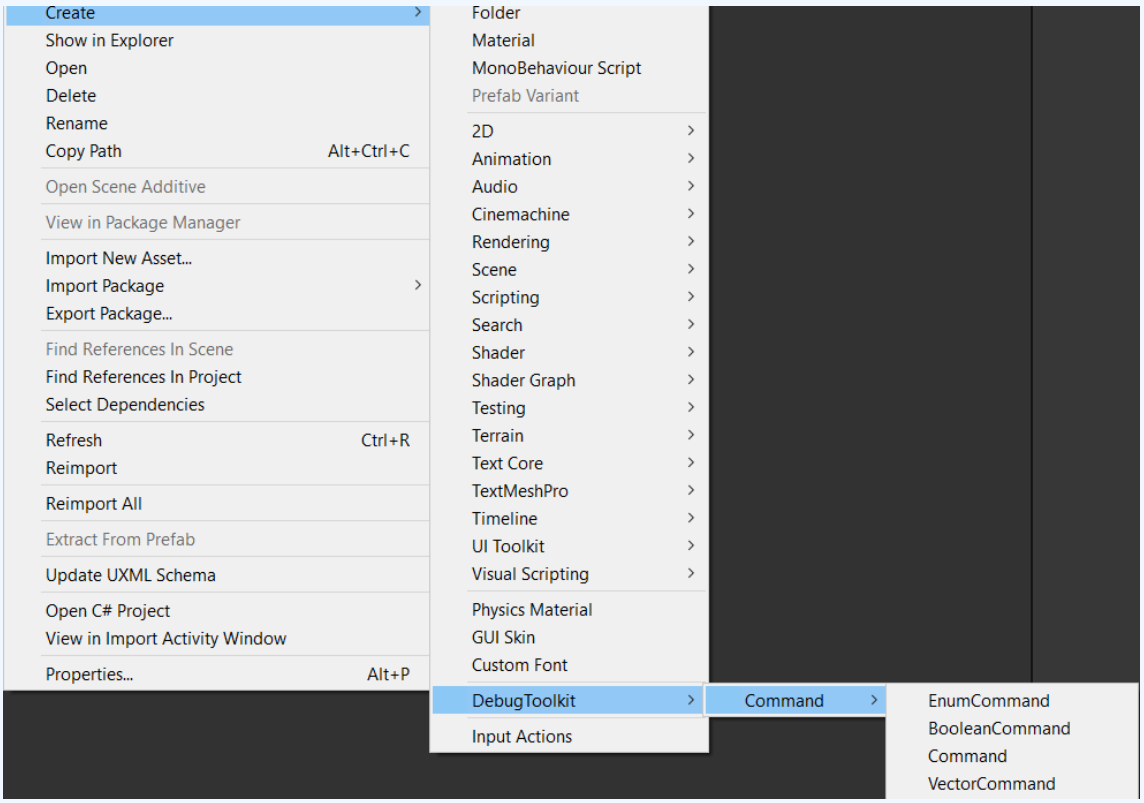
There is an API to log in the in-game console only. Go in the section Console to discover how to use it. Don't forget that the logs from Unity are retargeted to the console anyway.

Make your own commands

The command system is based on scriptable objects. Scriptables object are really practical to use because they are Gameobject and scene agnostic.

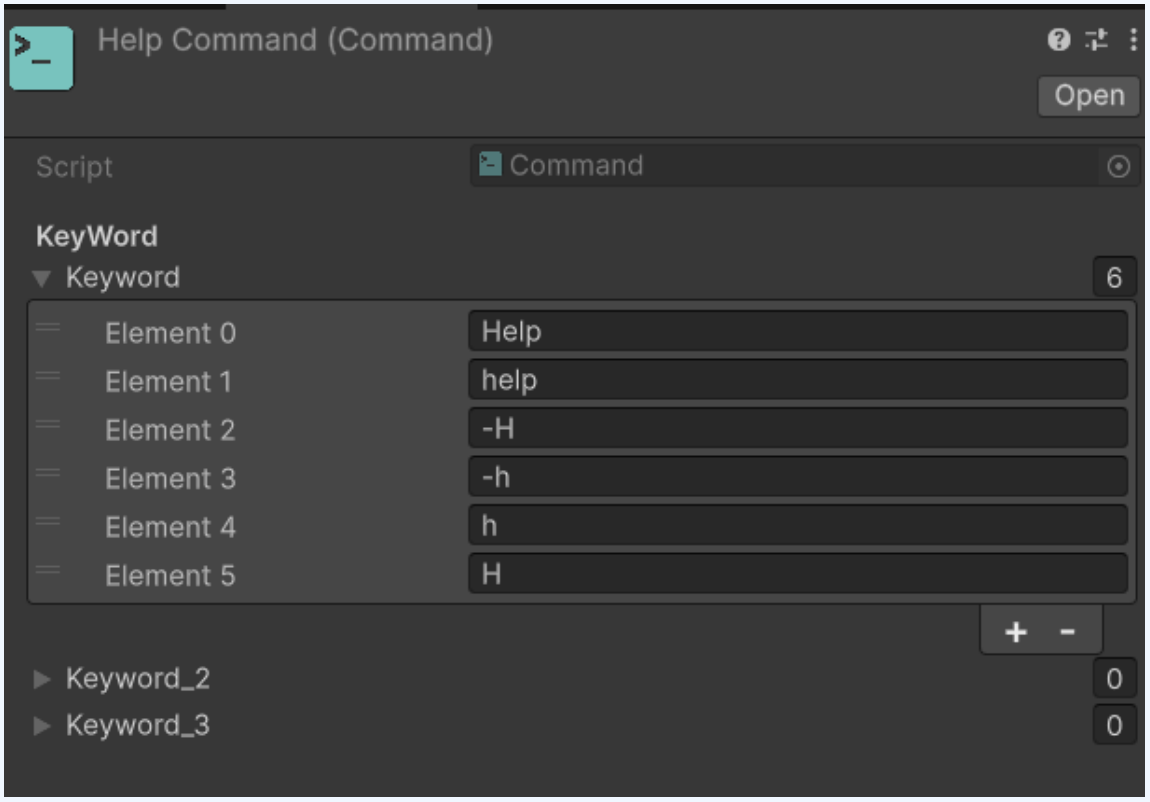
> We are planning on making an API to create commands using C# attributs which'll give you two options to create your console commands.

You can create like this

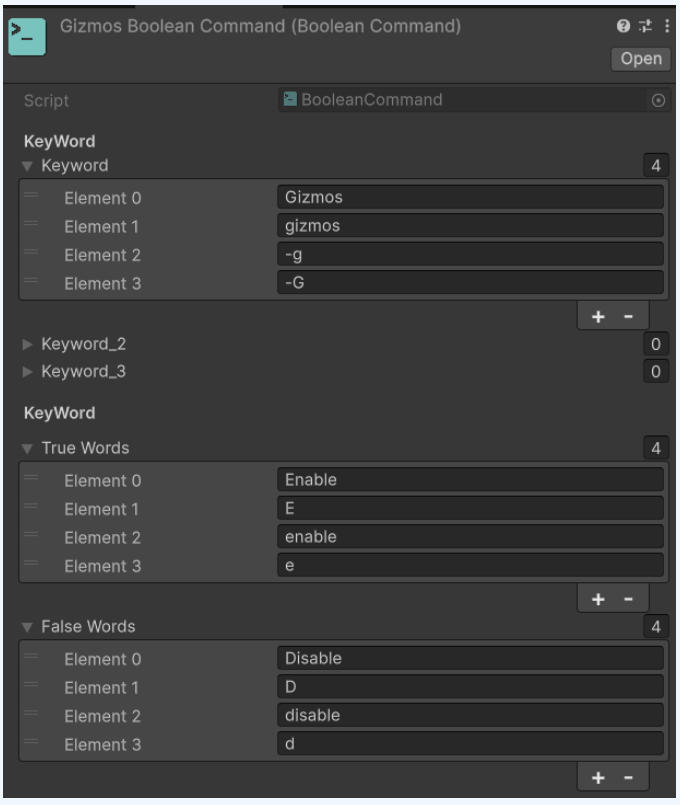


There are four type of commands :

- The **Command** is the most basic one it is used to simply inputs a chain of up to three words that would send a signal of type => Command Got Activated. This is the one which is used for the help command per example.

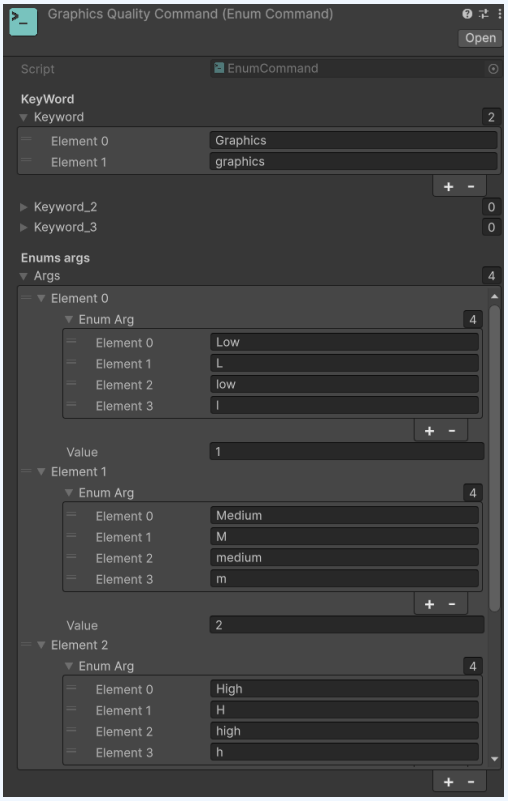


- The **Boolean Command** which takes two arguments to send either a signal activate or deactivated (true or false). This used for the Gizmos command per example. You've got the main keywords and then the keyword for true and the keyword for false.

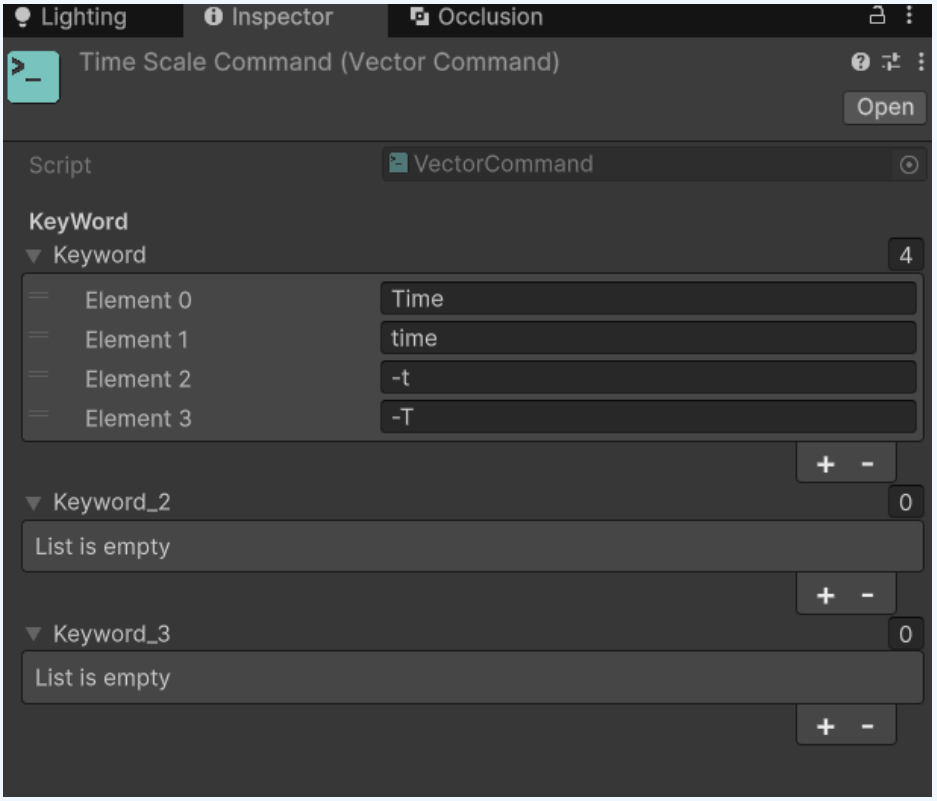


- The **Enum Command** which take n arguments to send a signal between 0 and n - 1. Its your job to handle what those index mean. You can bind those index with a keyword each

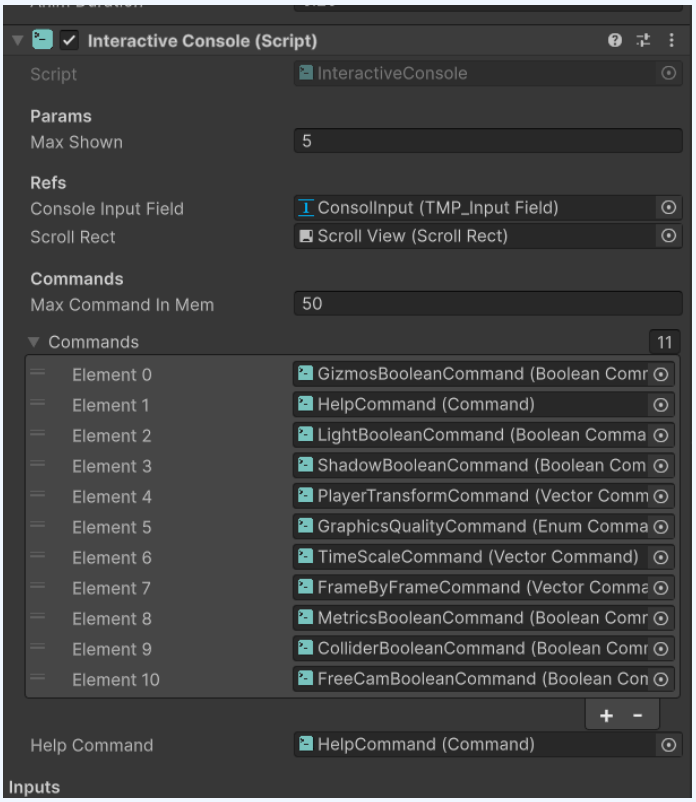
so its easier to use in the in game console. This command is use in the Graphic Quality Command per example.



- The **Vector Command** which takes as an argument of size 1 to 4, so basically a float, a Vector2, a Vector3, or a quaternion. It has four signals, one four each vector size. Its used for the timescale command per example.



So now you now how to make a command but there is a need for a bit more set up. First you need to add this command to the interactive console component in the UDTConsole-Canvas prefab.



Now there is only one thing left for you to do : connect this command to a script of your choice. Once again this is pretty straight forward. To show you how to do it we are going to take a look at the **DebugTimeScale** script which is in the sample folder.

To be able to bind a command to one of your script you need to add the **DebugToolkit.Console** assembly to the assembly of your script.

Then in the script you need to :

- get a reference to the command you've create (you can use the resources folder or the addressable package to avoid some clicks in the inspector, here we used a serialized field).
- subscribe to the event of your choice (depending on the type of command).
- not forgetting to unsubscribe onDestroy to avoid memory leaks (pro tip : some times unity dosen't calls the OnDestroy correctly when you stop the play mod so don't forget to add a *if(this==null)return*)).


```
public class DebugTimeScale : MonoBehaviour
{
    [Header(header: "Command")]
    [SerializeField] private VectorCommand timeScaleCommand;
    [SerializeField] private VectorCommand frameCommand;

    [Header(header: "Control")]
    [SerializeField] private TimeControlButton pauseButton;
    [SerializeField] private TimeControlButton frameButton;


    private void Awake()
    {
        timeScaleCommand.OnVector1Inputed += HandleTimeScaleChanged;
        frameCommand.OnVector1Inputed += HandleFrameCommand;

        pauseButton.OnActivation.AddListener(delegate {
            pauseButton.Active = !pauseButton.Active;
            if(pauseButton.Active)
            {
                HandleTimeScaleChanged(val: 0);
            }
            else
            {
                HandleTimeScaleChanged(val: 1);
            }
        });
        frameButton.OnActivation.AddListener(delegate {
            frameButton.Active = !frameButton.Active;
            if(frameButton.Active)
            {
                HandleFrameCommand(amountOfFrames: 1);
                pauseButton.Active = true;
                pauseButton.UpdateColor();
                frameButton.UpdateColorAsync();
            }
        });
    }

    private void OnDestroy()
    {
        timeScaleCommand.OnVector1Inputed -= HandleTimeScaleChanged;
        frameCommand.OnVector1Inputed -= HandleFrameCommand;
    }
```

> **Note** You can find the existing commands used in the toolkit here => 4Hands2Cats/DebugToolkit/Console/Interaction/CommandData

Static commands

> 🆕 **Introduced in version 1.2**

There is now a way to directly create debug command for the console in your code base using attributes.

There are three different attributes available :

- [SimpleCommand] : a simple command that will be executed when the command is called
- [BooleanCommand] : a command that will return a Boolean value
- [VectorCommand] : a command that will return a Vector4 or vector3 or vector2 or vector1

Please Notice that the Enum command is not supported. If you fee like you need it request it on our discord :D.

Better then any long explanation here is a code sample you can find in the project. The main idea of this sample is to propose an integration of the debug toolkit console commands in a class that uses the Singleton anti-pattern.

You can find this script in the sample folder.

```
public class CommandSample : MonoBehaviour
{
    /// <summary>
    /// Since the commands attributs are only working on static method
    /// a singleton pattern can be a sollution for a debbuging class.
    ///
          ///  Even if your methods got to be static, they can be
internal/private/public
    /// </summary>
    private static CommandSample Instance { get; set; }

    [SerializeField] private Transform playerTransform;

    private void Awake()
    {
        Instance = this;
    }

    [SimpleCommand("Hi")]
    public static void SayHi()
    {
        Debug.Log("HelloWorld");
    }

    [BooleanCommand("navA", "move", "stop")]
    public static void SetAgentOnOff(bool isMoving)
    {
                    NavMeshAgent[] agents = FindObjectsByType<NavMeshAgent>
(FindObjectsInactive.Include, FindObjectsSortMode.None);
        for (int i = 0; i < agents.Length; i++)
        {
            agents[i].isStopped = !isMoving;
        }
    }

    /// <summary>
     /// Note this command would work for a method with a param as a float, a
Vector2, a Vector3 or either a Vector 4
    /// </summary>
    /// <param name="position"></param>
    [VectorCommand("player-tp")]
    public async static void TeleportPlayer(Vector3 position)
    {
         Instance.playerTransform.GetComponent<ThirdPersonController>().enabled
= false;
         Instance.playerTransform.GetComponent<CharacterController>().enabled =
false;
        Instance.playerTransform.position = position;
```

```
            await Awaitable.NextFrameAsync();

             Instance.playerTransform.GetComponent<ThirdPersonController>().enabled
= true;
             Instance.playerTransform.GetComponent<CharacterController>().enabled =
true;
        }

        [SimpleCommand("scene-reload")]
        public static void ReloadScene()
        {
            SceneManager.LoadScene(SceneManager.GetActiveScene().name);
        }
    }
```

> There is no support yet for non-static methods but its doable. If you feel like you are needing those, just contact us on the discord.
>
> If you are wondering how it works behind the scene don't hesitate to contact us :D

Command Prediction

> 🆕 **Introduced in version 1.3**

This feature makes navigation in the console easier.

To use it just open the console, and start typing. The console will show you any possibilities.

The complexity is related to the length of the commands and not to the total amount in the project. Any command you make are going to be automatically supported by this feature.

- You can click on the commands to select it.
- You can navigate using the UP and DOWN arrows. When navigating using the arrows you can use the RIGHT arrow to select the command.

When a command is selected it appears in the console, you can either complete it or just execute it pressing enter.

Free Cam

The Free Cam provides a runtime navigation tool similar to Unity's Scene view controls.

> 🚧 **Future versions** of the Free Cam are planned to support Cinemachine.

Where to Find It

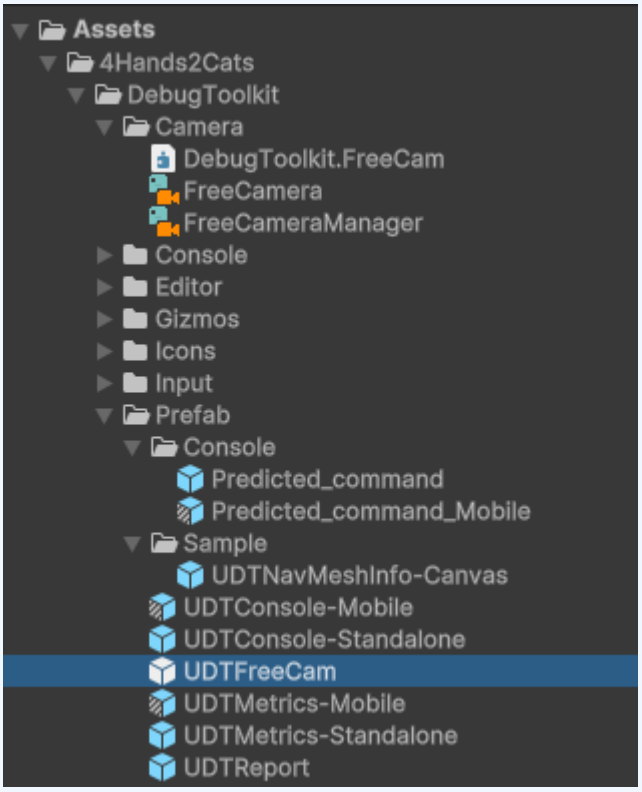
Instantiation via UD Console

To use it, type `freecam` or `Freecam` followed by `enable` or `disable` in the console after opening it using **F12**.

Using as a Prefab

Although embedded in the package, the Free Cam is also available as a standalone feature.
Feel free to integrate it into your gameplay directly.

The Free Cam prefab is located in the same folder as the console prefab.



To use it as a standalone component, just drag and drop the prefab into your scene.

> **Note:** If you use it as a standalone feature, it will no longer be controlled by the console.

How to Use It

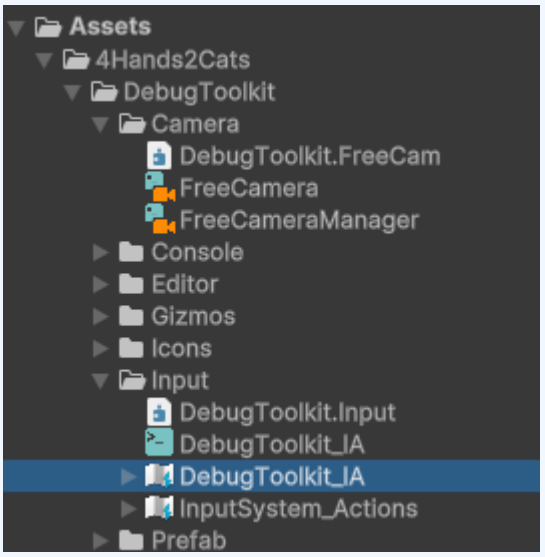
The controls are similar to those in the Unity Scene view:

- Hold **Right Mouse Button** to enter fly mode.
- Use **W/S** to move forward/backward (Z axis).
- Use **A/D** to move left/right (X axis).
- Use **Q/E** to move down/up (Y axis).
- Move the mouse to rotate the camera.
- Hold **Shift** to sprint.
- Use the **Mouse Scroll Wheel** to adjust speed.

You can also incrementally move forward using the scroll wheel **without** holding the right mouse button.

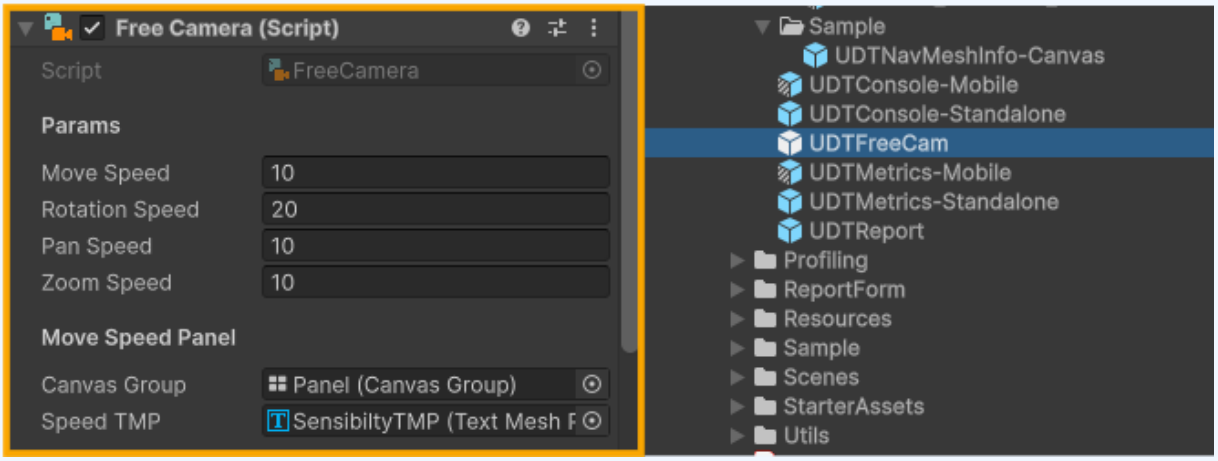
Customizing Controls

You can easily modify the controls using the `DebugToolkit_IA` (Input Action):



Modifying Parameters

Adjust Free Cam parameters directly on the prefab:


Free Camera (Script)
Script
FreeCamera
Params
Move Speed
10
Rotation Speed
20
Pan Speed
10
Zoom Speed
10
Move Speed Panel
Canvas Group
Panel (Canvas Group)
Speed TMP
SensibiltyTMP (Text Mesh F
Sample
UDTNavMeshInfo-Canvas
UDTConsole-Mobile
UDTConsole-Standalone
UDTFreeCam
UDTMetrics-Mobile
UDTMetrics-Standalone
UDTReport
Profiling
ReportForm
Resources
Sample
Scenes
StarterAssets
Utils

Gizmos

As you now there is already an API in unity to draw gizmos for debugging purposes. But you cannot draw gizmos for run time. Those gizmos are harvesting the power of the **GL** API to show performance friendly gizmos at runtime for quite anything.

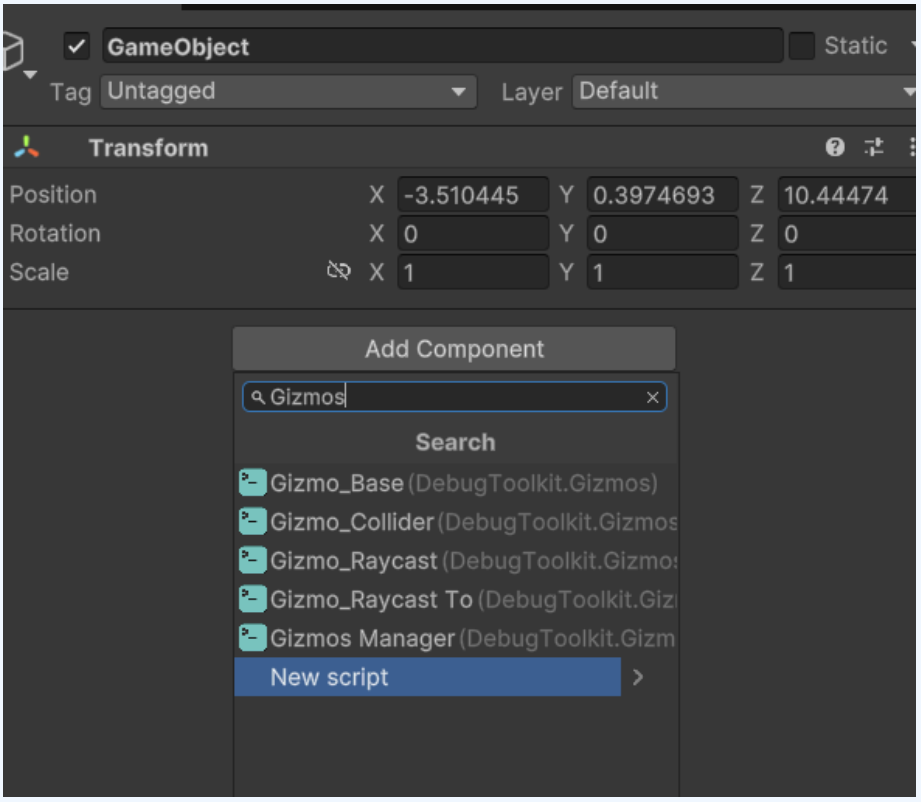
There are two way of using the gizmos. Manual and automatic using the console.

- The manual way consist of adding the gizmos components to your gameobjects
- The automatic way is used to show all colliders in the scene. To show all colliders simple type the command : *Collider/collider/-c/-C* followed by *enable/e* or *disable/d* to enable or disable the in game gizmos rendering for the collider (beware on large scene there might be a small freeze when enabling). This command also activates all the other in game gizmos [going to change in next version] int the ingame console.

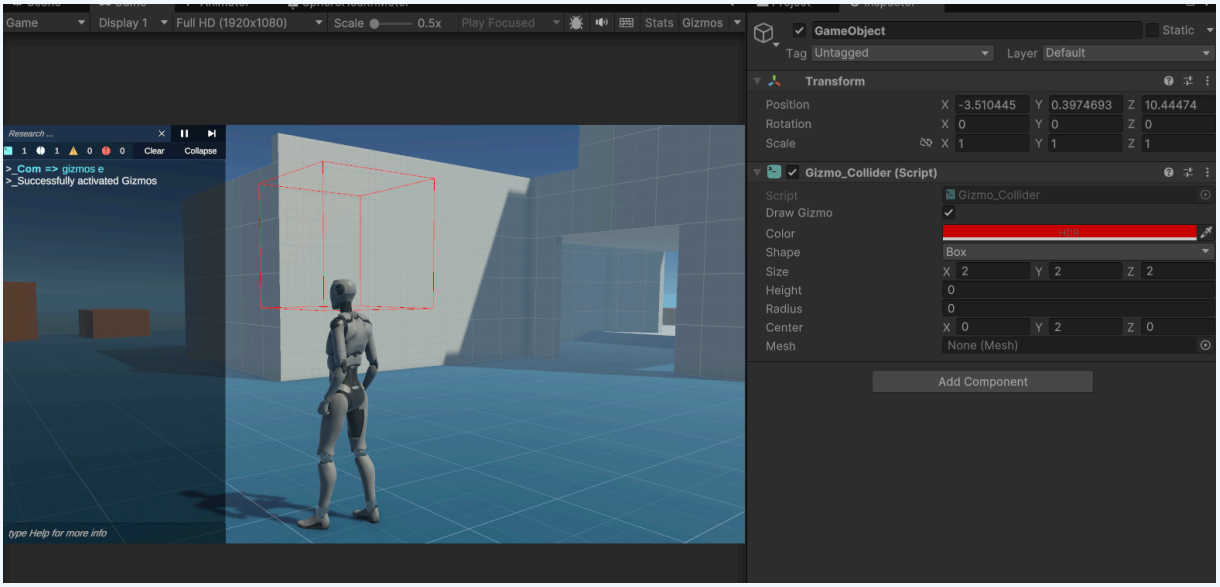
Gizmos that are manually setted up can are still managed by the ingame console. Simple type : *Gizmos/gizmos/-g/-G* followed by *enable/e* or *disable/d* to enable or disable the in game gizmos.

> Note : this commands would also hide the collider gizmos. We are awaiting for you feedback to improve this feature.

To set up a Gizmos as component simple add the choosen Gizmos to your gameobject.



Use the gizmos Collider to choose show a collider like shape.

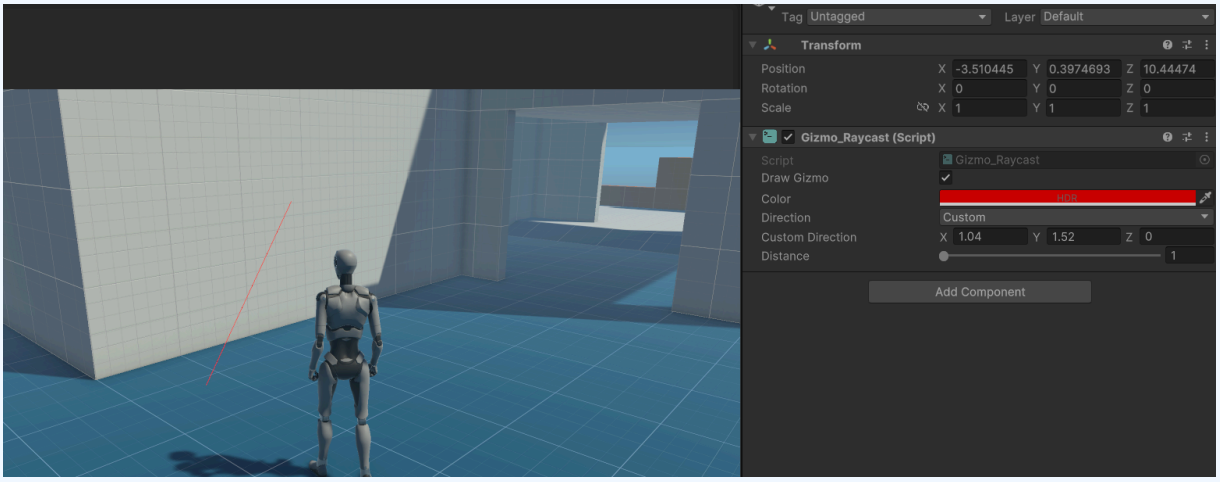


There are four options :

- The **Box**, that you can control using the **Size** and the **Center** parameters
- The **Sphere**, that you can control using the **Radius** and the **Center** parameters
- The **Capsule**, that you can use using the **Radius**, the **Center** and the **Height** parameters
- The **Mesh**, that you can use by drag and dropping a mesh in the mash container of the gameObject. [To enable this feature you need to enable Read/Write on your meshes in the import settings].

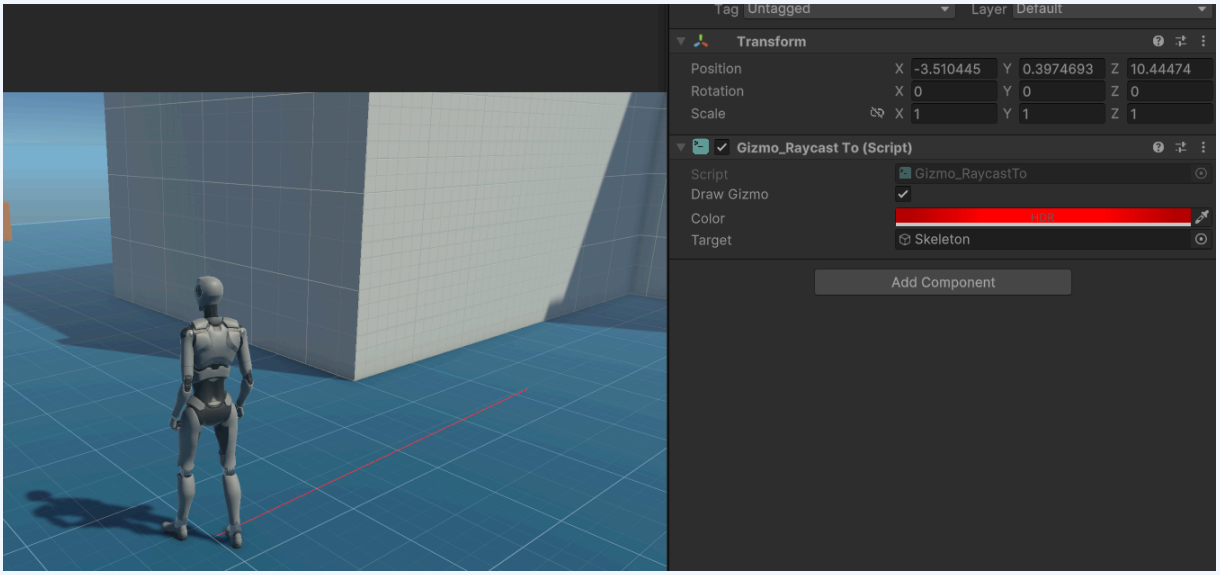
  Note that those gizmos support bloom and they can be used as part of your project for other purposes than debugging.

Use the Gizmos Raycast component to draw a Ray cast.



There are some direction preset for all the cartesian direction (up, left, etc...) but you can use custom to choose your own direction.

Use the Gizmos Raycast To component to draw a gizmos between two targets.



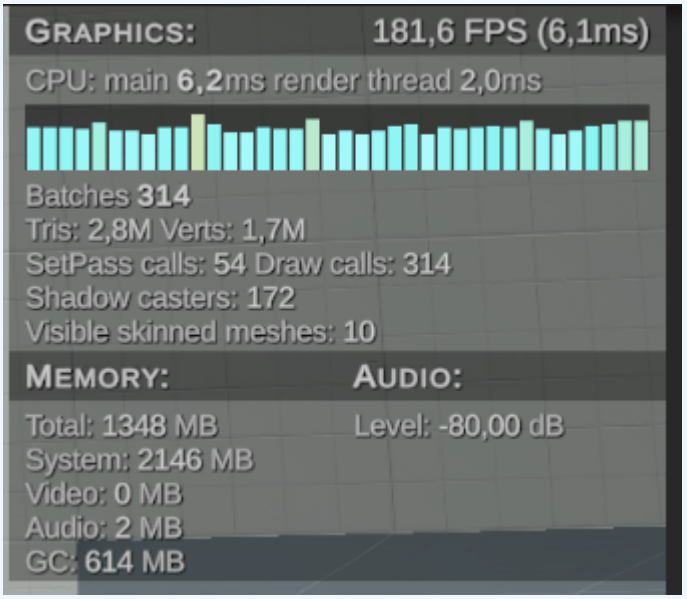
You just have to give it a ref to a gameobject and it'll draw a ray to it and update the ray at everyframes.

We are planning on adding custom editors in the future to simplify the usage of those features. Please take a look at the section Gizmos API to learn about the code API for the gizmos.

Metrics

The metrics are enabled using the interactive console with the command :

> *metrics/Metrics/-m/-M* followed by *enable/e* or *disable/d* to enable or disable the metrics.



The Graphics section displays the FPS and the time elapsed since the last frame. It also shows the time spent by the Main thread and a breakdown of the time taken by the Render thread. The graph represents the time spent by the main thread, targeting 120 FPS. Additionally, this section includes:

```
The number of batches
The number of triangles
The number of vertices
The number of SetPass calls
The number of draw calls
The number of shadow casters
The number of visible skinned meshes
```

The Memory section displays the total memory usage across:

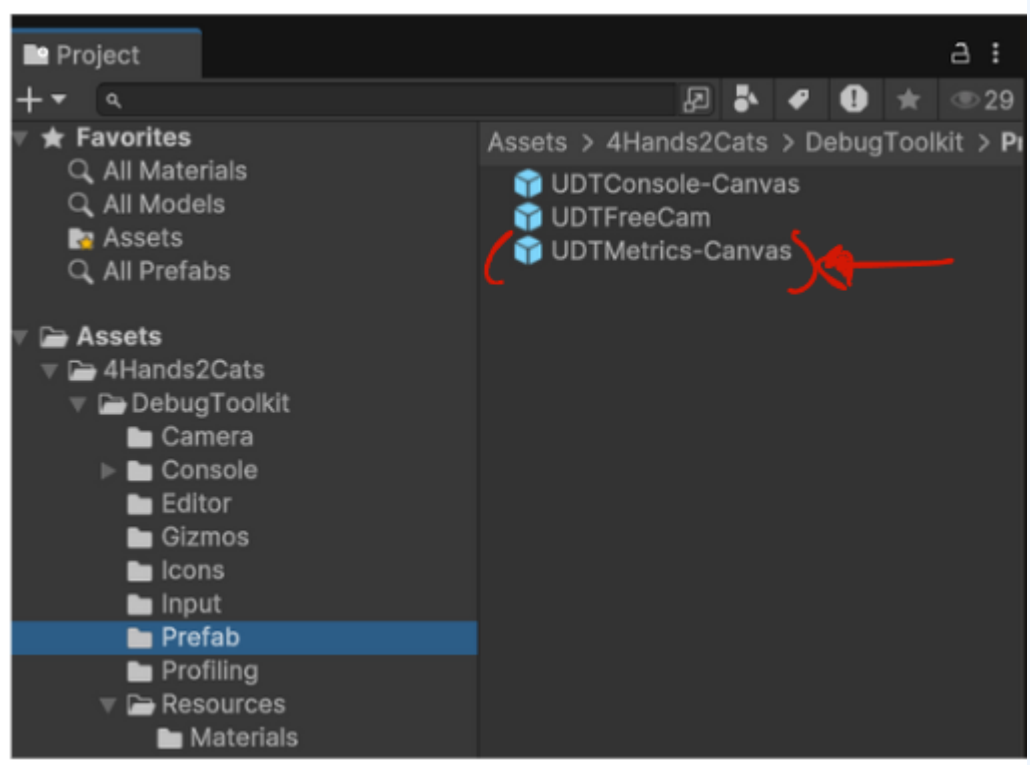
```
Engine
System
Video
Audio
Garbage collection
```

The Audio section shows the current audio level.

To adjust the update rate of the metrics, modify the Update Delta Time parameter in the metrics prefab:

```
Increase the value to reduce the update frequency.
Decrease the value to increase the update frequency.
```

You can use the metrics as a stand alone feature by drag and dropping it in your scene.



> **Note** If you use the metrics as a standalone feature, it'll not be managed by the console anymore.

NavMesh Debugging

This functionality allows to debug the navmesh at runtime.

To use : there are 3 commands available in the interactive console.

```
    navA gizmos enable/disable : To draw the gizmos of the navAgent pathing

      navA info enable/disable : To show the info about the pathing of the
navAgent

    navA all enable/disable : To activate all commands
```

This features cannot be used as a standalone yet.

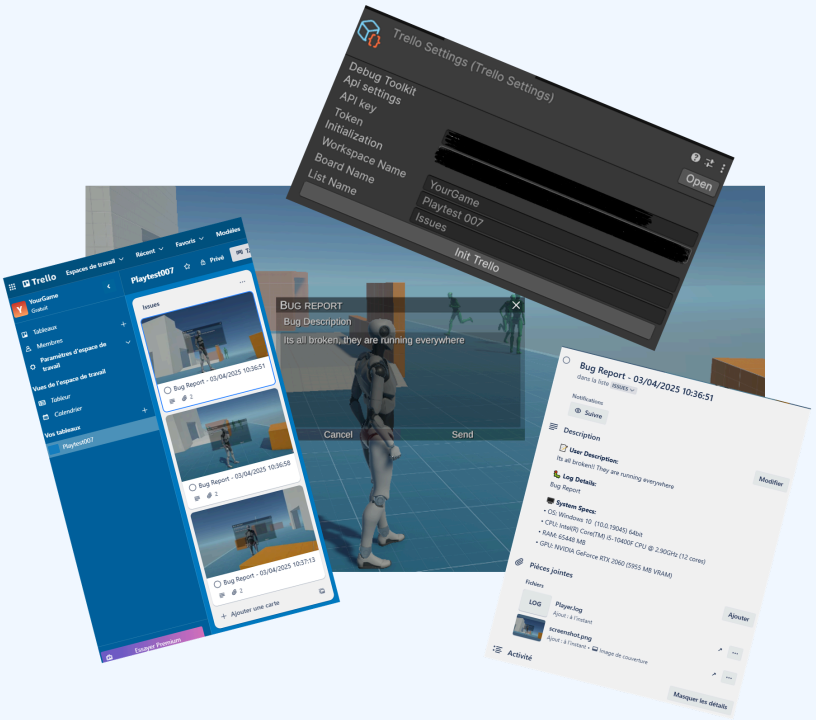
> **Note** Do not hesitate to request more features for the navmesh debbuging.

Bug Report

> 🆕 **Introduced in version 1.3**

Thanks to this feature you can send tickets to a board. The system supports both Trello and Discord. We are planning to extend to Jira, Notion, CardDeck and Azure. Just tell us and we'll add it for you !!

This section is all about guiding you through the process of setting up your project to enable the Trello or/and Discord



> As always we listen carefully to your feedbacks and we'd love to improve the feature based on those !!

Trello Setup

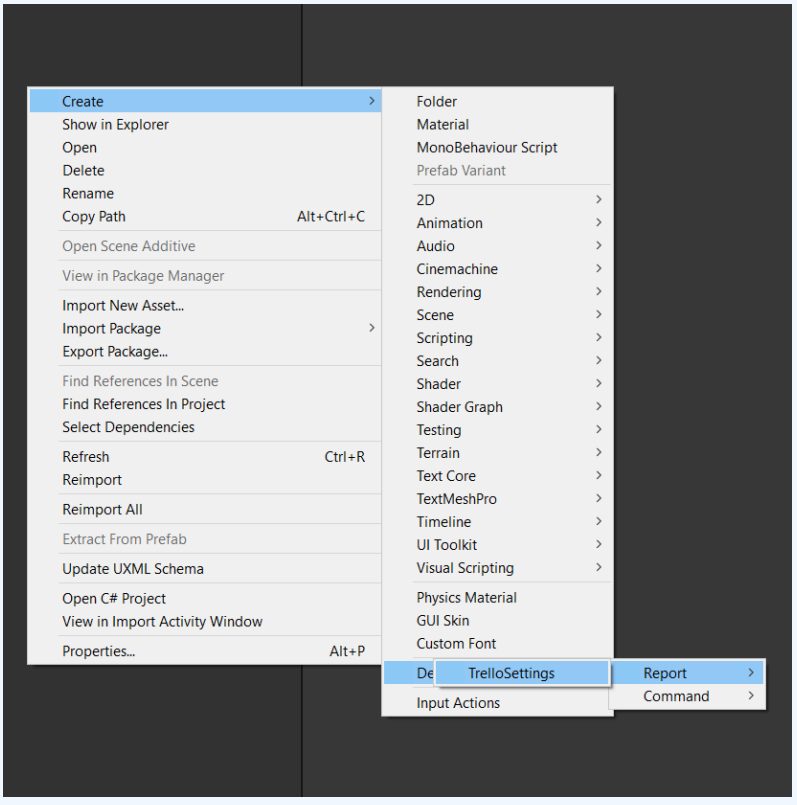
> 🆕 **Introduced in version 1.3**

The set up is really guided thanks to a custom inspector. The main idea is to store your board information in a scriptable object. Thanks to this process you can have multiple boards for your project. Maybe you want each playtest to have its own board.

For this part you are going to need a Trello account. Since your API key and Token are going to be present in your build we most advise to make a new account for the debug toolkit.

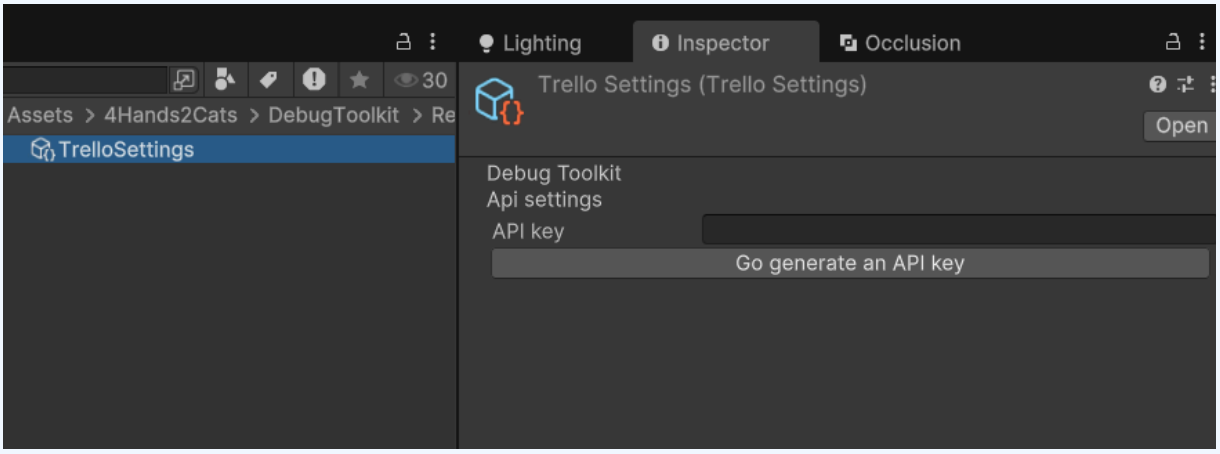
Step 1 : Create the scriptable object

Right click and create the SO



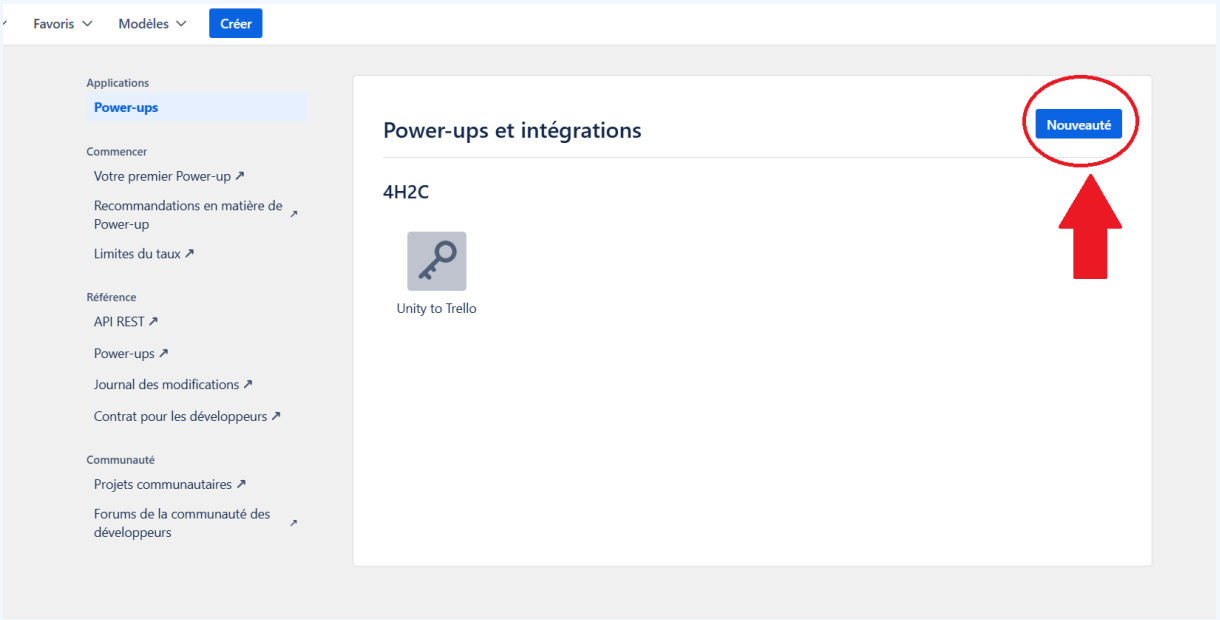
Step 2 : Set up you API key

Click on the Go generate an API keybutton. It'll open the Trello website. If you are not connected or don't have a trello account you are going to have do both.

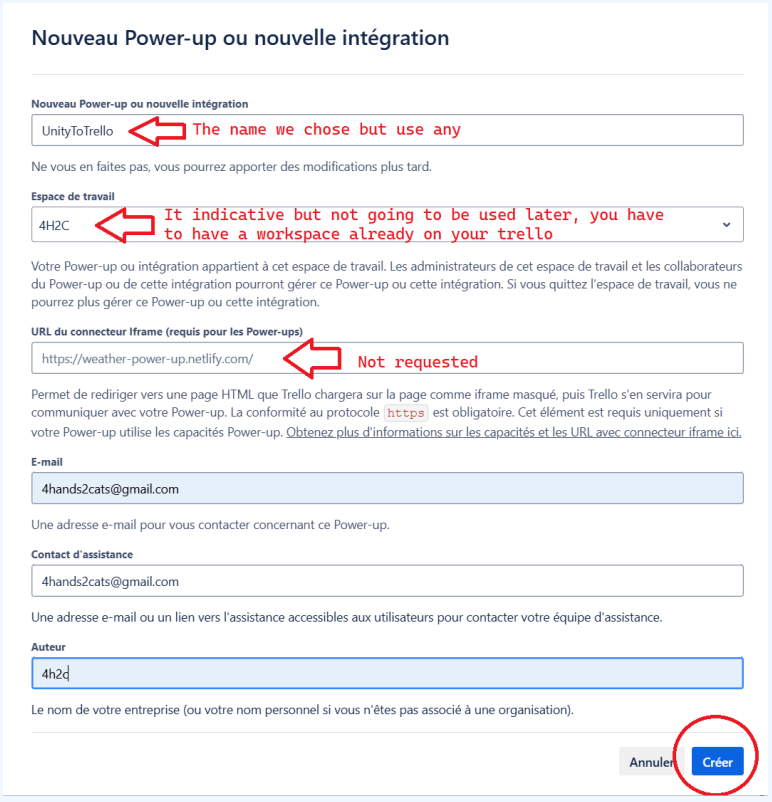


Here click on the blue hightlighted text.

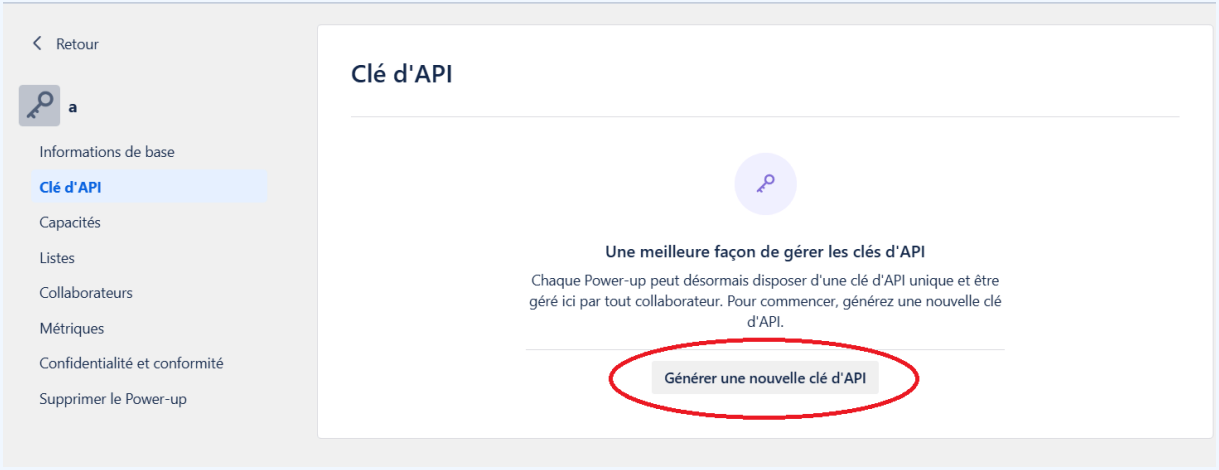
You are going to need to click on the top right button, to create a new power up. On this screenshot we already created ours. You need yours.



Now just fill the form and click on Create



Now generate your API key

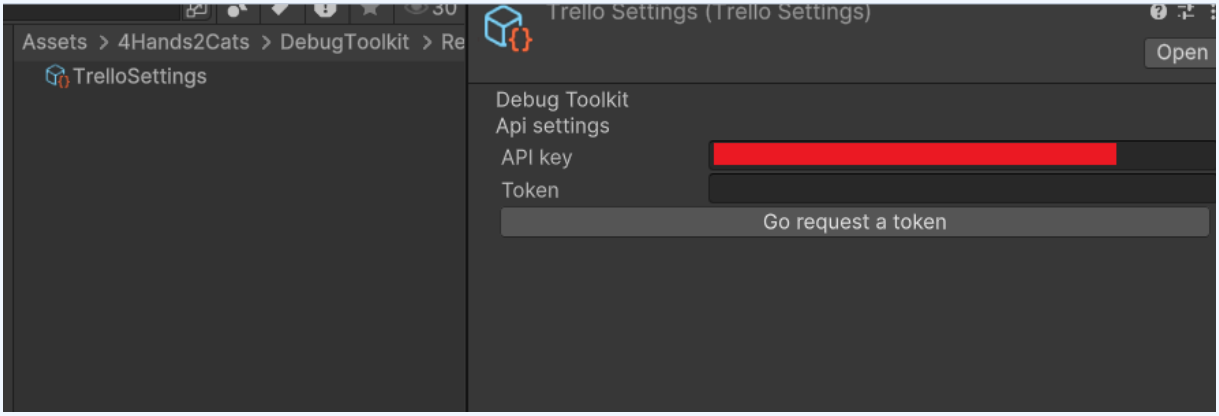


Copy the API key you just go on the scriptable object in unity.

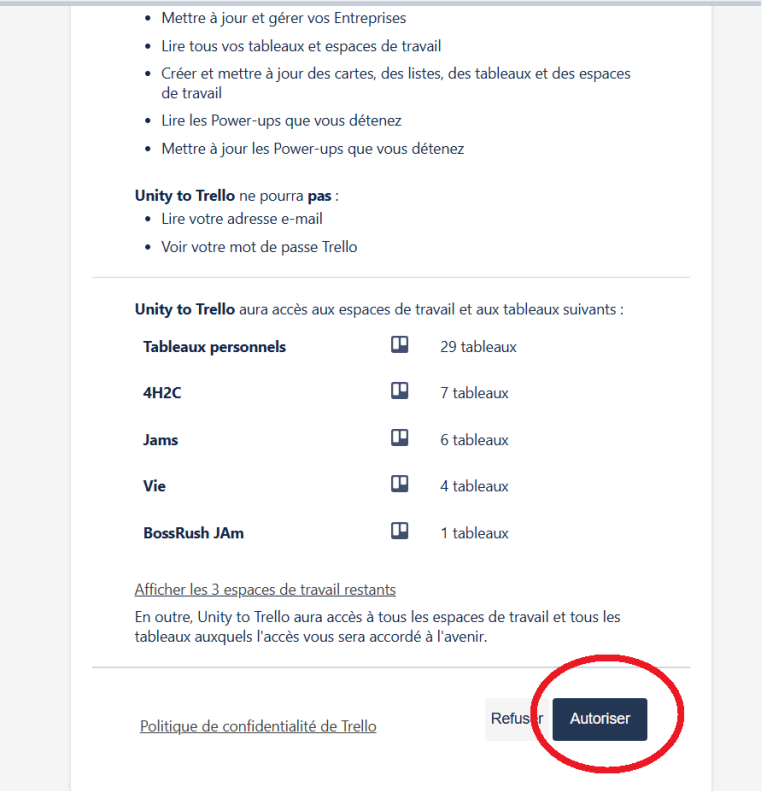
Step 3 : Get your token

Don't worry its way faster than getting the API key.

Now that you've entered your API key, the scriptable object changed and gives you a button to request the Token. Click on the button Go Request Token.



You'll get to a trello page where they ask you if you allow your power up to communicate with the boards related to your account. Just allow it !!

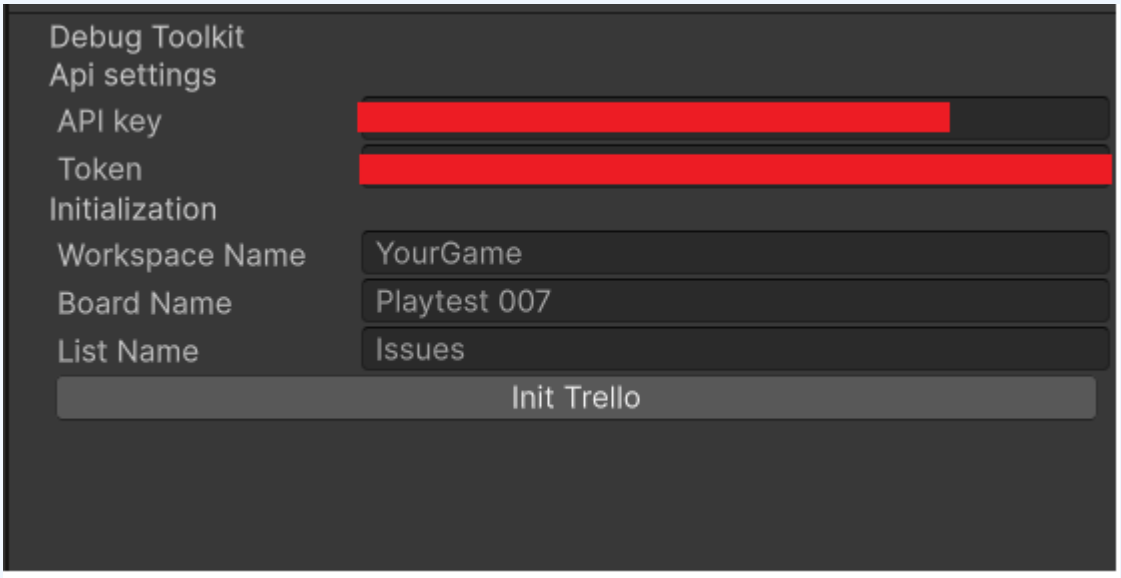


You are going to be redirected to a page for your token. Just copy past it to your scriptable object in unity.

Step 4 : Init the board

This is the last step !!

Name your Workspace, board and the list where the bugs are going to be reported. And press Init trello. You should see some messages in the Unity consol. If you get any error please contact support on the discord.



Discord Setup

> 🆕 **Introduced in version 1.4**

> **Disclaimer:**
> Setting up Discord is a bit harder then Trello. The main reason is that Discord's Token are too powerful. One need to protect the token such as your user cannot abuse it.
>
> As a consequence the request have to go through a relay server. For the sake of simplicity we chose to use the Unity's Cloud Code Service. In term of pricing it goes for 1 000 000 free request per month and 0.5$ per supplementary million. If you feel like your project is going to need more, reach out to us so we could make a custom AWS implementation just for you.
>
> Also discord is limited to 50 messages per second per bot.

Let's do it step by step.

Step 0

You need a discord server, that you administrate. This server needs to be a community sever, where you can create forums.

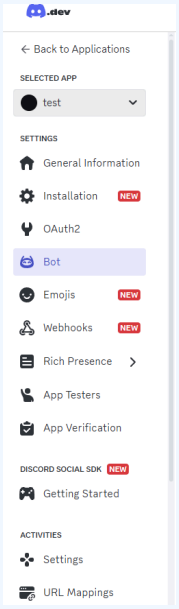
Get a discord bot token

This step is pretty easy. But beware to set the settings right. Go to this adresse : https://discord.com/developers/applications and click on the New application button at the top right of the page.

Give it a name that makes sens to you, agree to the term of services, click submit and confirm that you are not a robot

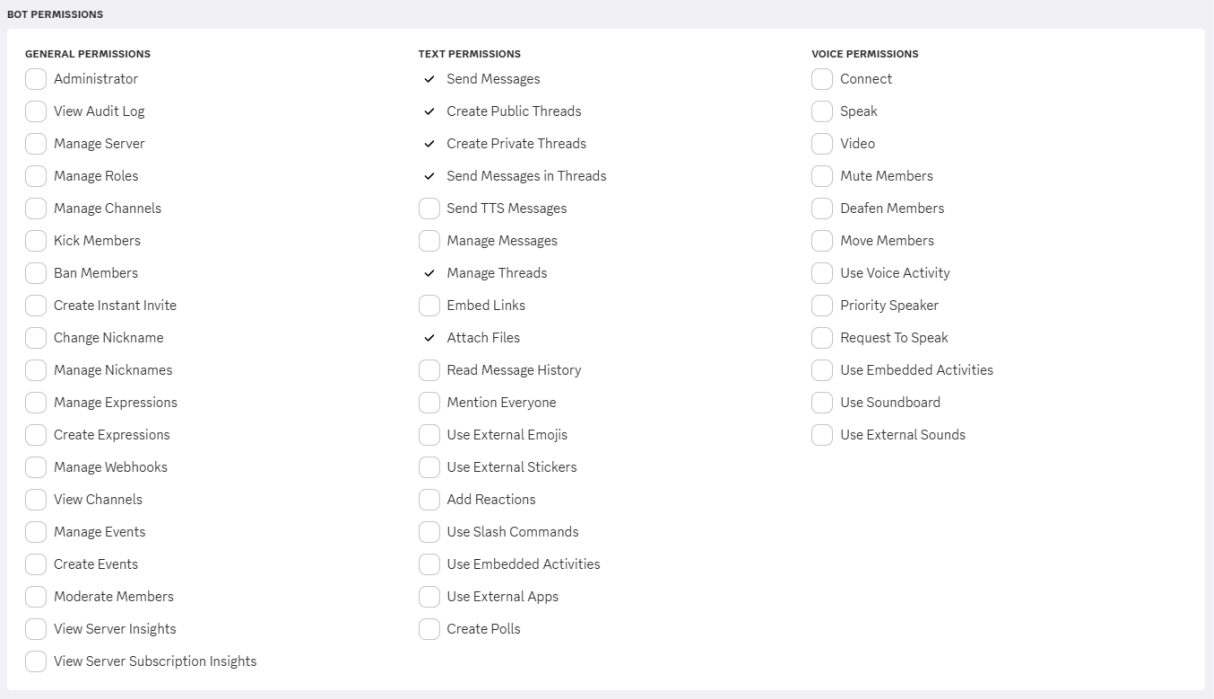
Please remember that you can get your application ID and your public key on the page of the discord bot that has just been created.

> In the bot token panel you can get your token. Click on reset token to have your first token. Copy this token in a bloc note, and never share it with anyone, never ever, ever !!
>
> Now get in the OAuth2 Section (In the left navbar) and get down to the OAuth2 URL Generator.

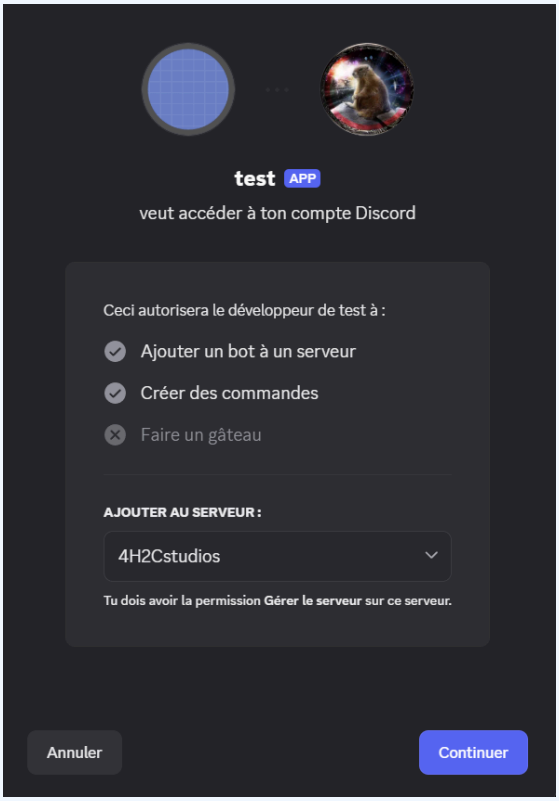


SCOPES

☐ identify	☐ email	☐ connections
☐ guilds	☐ guilds.join	☐ guilds.members.read
☐ guilds.channels.read	☐ gdm.join	☑ bot
☐ rpc	☐ rpc.notifications.read	☐ rpc.voice.read
☐ rpc.voice.write	☐ rpc.video.read	☐ rpc.video.write
☐ rpc.screenshare.read	☐ rpc.screenshare.write	☐ rpc.activities.write
☐ webhook.incoming	☐ messages.read	☐ applications.builds.upload
☐ applications.builds.read	☐ applications.commands	☐ applications.store.update
☐ applications.entitlements	☐ activities.read	☐ activities.write
☐ activities.invites.write	☐ relationships.read	☐ relationships.write
☐ voice	☐ dm_channels.read	☐ role_connections.write
☐ presences.read	☐ presences.write	☐ openid
☐ dm_channels.messages.read	☐ dm_channels.messages.write	☐ gateway.connect
☐ account.global_name.update	☐ payment_sources.country_code	☐ sdk.social_layer_presence
☐ sdk.social_layer	☐ lobbies.write	
☐ applications.commands.permissions.update		

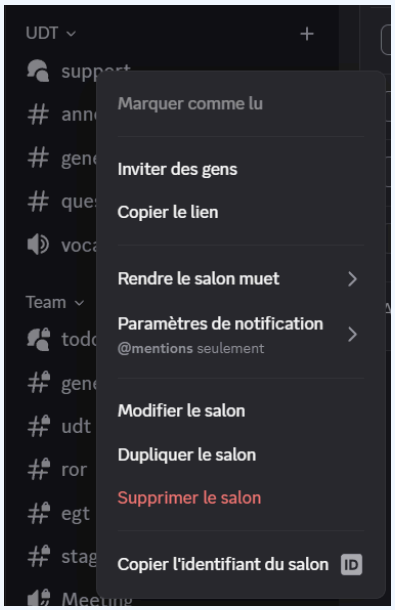


It should have generated an URL for you. Copy and paste this URL in your browser. This screen should open in your discord APP.



Select the server you want to add the bot to. Just Accept.

Get the forum ID



Its really quick first put your discord account in dev mode.

Then go to the forum, right click and copy the ID.

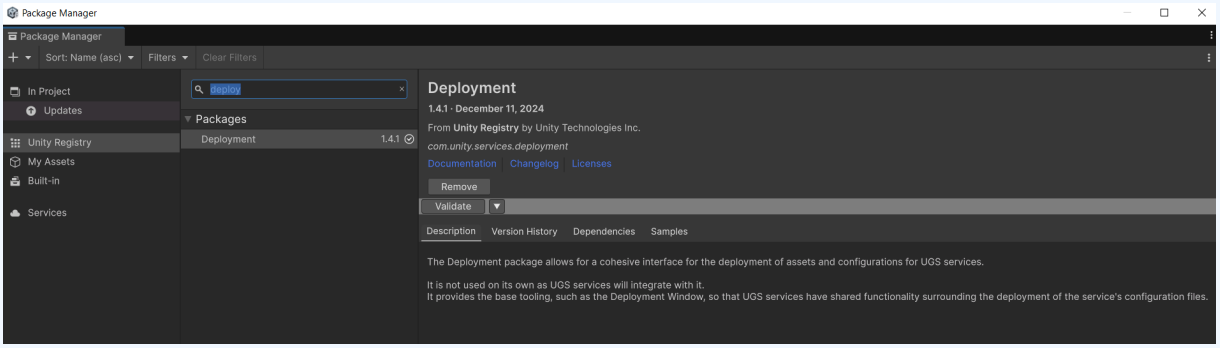
Configure the C# cloud module

Ok now this is the tuff part. You are going to do some cloud code and use the unity services.

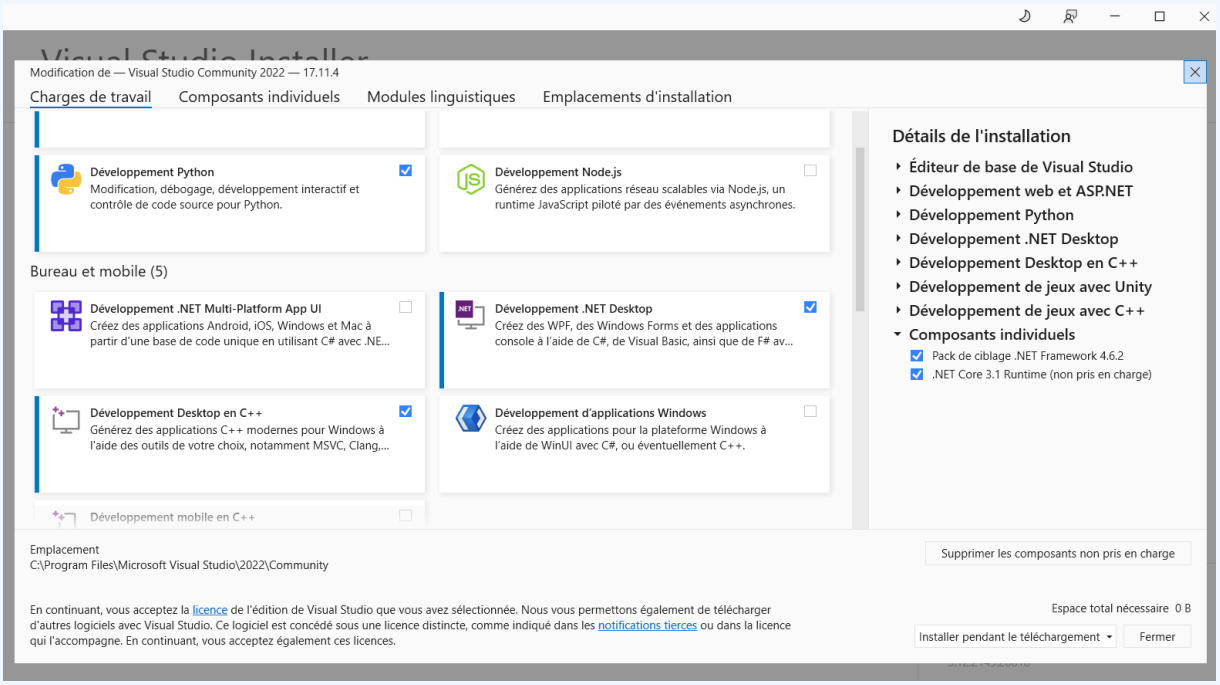
> If you already use the Authentication service in your project you might want to change a bit the implementation. (Just remove the authentication from the Discord API script)

Normally both cloud code and authentication got installed as dependencies of the debug toolkit. If they didn't you might already know it (error messages). Please install them.

Then you have to install Unity's Deployement package, from the package manager.



Now you've got to install .Net on your computer. Go in your visual studio installer and click on modify the install and add the .Net dev.



Make sure your projet is connected to Unity services. ProjectSettings > Services. If you don't have a unity project ID. Please get one.

In the folder 4Hands2Cats/DebugToolkit/ReportForm/CloudModules/DiscordModule right click and create a new cloud code module. Then click on the DiscordThread Asset and in the inspector click on generate solution. It should compile for a few minutes (basically it create yet another C# project in visual studio).

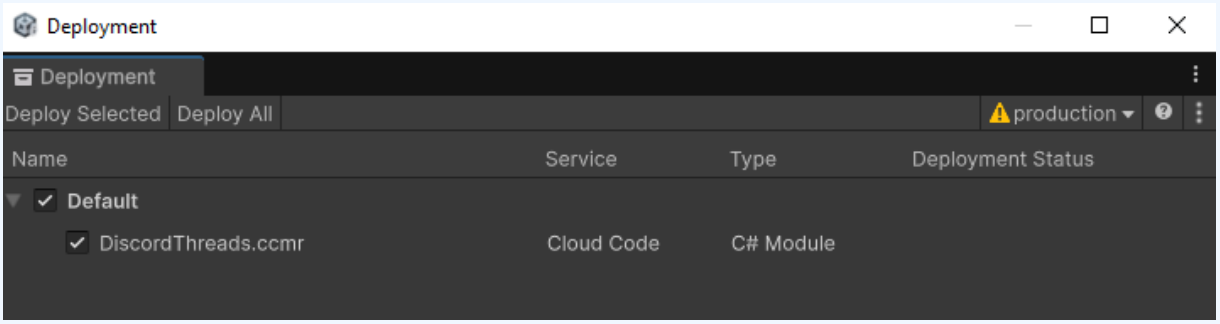
Now go to the project and replace the sample script by this one and replace the Token by the token of your bot and forum channel id by the one of your forum. You should rename the sample script like : DiscordThreadCreator.cs.

```
using System.Net.Http;
using System.Text.Json;
using System.Text;
using System.Threading.Tasks;
using Unity.Services.CloudCode.Core;
using System;

namespace DiscordThread;

public class DiscordThreadCreator
{
    private readonly HttpClient client = new HttpClient();

    [CloudCodeFunction("PostToDiscordForum")]
         public  async  Task<object>  PostToDiscordForum(string  title,  string
description,
        string imageFileName, string imageMimeType, string imageBase64,
        string textFileName, string textFileMimeType, string textBase64)
    {
          // Please make sur you never push to a public repository with your
Discord Bot Token hardcoded.
        var discordBotToken = "YOUR_TOKEN";
        var forumChannelId = "YOUR_FORUM_ID";

                                                          var       url       =
$"https://discord.com/api/v10/channels/{forumChannelId}/threads";

        var payload = new
        {
            name = title,
            auto_archive_duration = 1440,
            message = new
            {
                content = description
            }
        };

        var json = JsonSerializer.Serialize(payload);

        var content = new MultipartFormDataContent();

        var imageBytes = Convert.FromBase64String(imageBase64);
        var imageContent = new ByteArrayContent(imageBytes);
                                    imageContent.Headers.ContentType    =    new
System.Net.Http.Headers.MediaTypeHeaderValue(imageMimeType);
        content.Add(imageContent, "files[0]", imageFileName);

        var textFileBytes = Convert.FromBase64String(textBase64);
        var textFileContent = new ByteArrayContent(textFileBytes);
```

```
                                    textFileContent.Headers.ContentType     =     new
System.Net.Http.Headers.MediaTypeHeaderValue(textFileMimeType);
        content.Add(textFileContent, "files[1]", textFileName);

         content.Add(new StringContent(json, Encoding.UTF8, "application/json"),
"payload_json");

        var message = new HttpRequestMessage(HttpMethod.Post, url);
        message.Headers.Add("Authorization", $"Bot {discordBotToken}");
        message.Content = content;

        var response = await client.SendAsync(message);
        var resultJson = await response.Content.ReadAsStringAsync();

        if (!response.IsSuccessStatusCode)
        {
            return new { success = false, error = resultJson };
        }

        using var doc = JsonDocument.Parse(resultJson);
        var threadId = doc.RootElement.GetProperty("id").GetString();
        var guildId = doc.RootElement.GetProperty("guild_id").GetString();

        return new
        {
            success = true,
            threadId,
            url = $"https://discord.com/channels/{guildId}/{threadId}"
        };
    }
}
```

Ok now get back to Unity and open the deployment window : Services > Deployment (it's in the top bar of Unity).

Fill it like on the image, and click on Deploy Selected.



Test it out

Go in the Demo scene. Select the UDTReport in the hierarchy and set the report manager's Report Target to Discord.

Now launch the scene and test it out. >Reminder : F11 to open the bug report form.

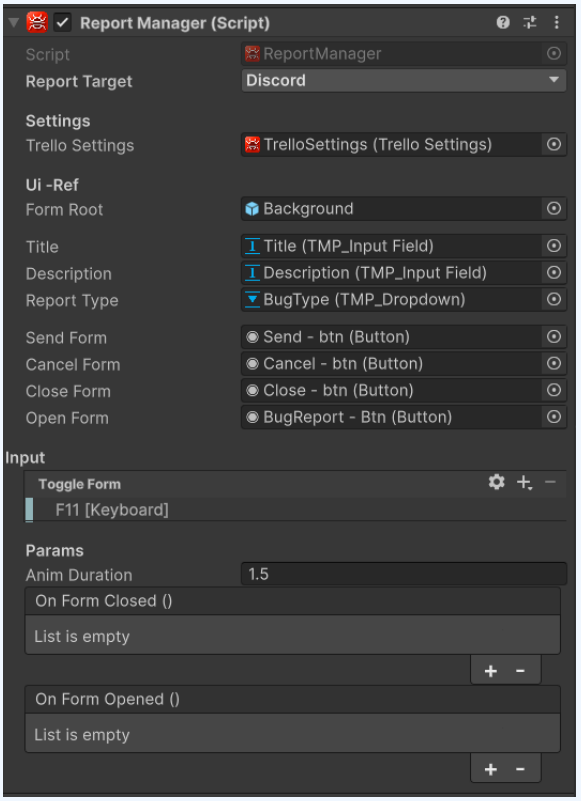
> **Note:**
> If you encounter any issue please contact support in the discord server.

Enable feature

> 🆕 **Introduced in version 1.3**

To enable the feature you need to add the UDTReport-Canvas prefab in your scene. Then you need to reference your TrelloSettings in the Report manager of this prefab.

There is a dropdown menu of the report manager select the Report Target you've set up.



> **Note** You can change the Trello settings to interact with multiple boards.

Report a bug

> 🆕 **Introduced in version 1.3**

To report a bug just open the game. Press F11 or click on the button right icon to open the bug report form.

Write your description end press the send button.

That's it.

If you get errors, or if the card is not created on your Trello, feel free to contact support on the discord.

Code APIs

In this section you'll learn how to use the different components of the toolkit in your scripts.

> Many functionalities are going to be added to the APIs of the different component over time. Stay tuned.

Gizmos

You can create and delete gizmos at runtime using the Gizmos API.

> **Note:**
> This APIs might change depending on your feedback. Make sure to come back to this section if your logic breaks.

For this version there only is API for the Collider Gizmos and the RaycastTo Gizmos.

> Feel free to request API features. If you make your own please show them to us, maybe they could be integrated in the toolkit.

Console

You can use the **DebugLog** API of the in-game console to print logs in the in-game console directly.

Collider Gizmos

The collider Gizmos is composed of four static method. You can use them anywhere to make your own instances the gizmos, and draw your own shape based on a collider.

- The **targetObject** is the gameObject on which your gizmos drawer is going to be instantiated.
- The second parameter is a **collider**.
- The third is the **color** you want to draw with.

```
        public static Gizmo_Collider DrawBoxGizmos(GameObject targetObject, BoxCollider boxCollider, Color colliderColor)
    {
        // Logic
    }

        public static Gizmo_Collider DrawSphereGizmos(GameObject gameObject, SphereCollider sphereCollider, Color gizmosColor)
    {
        // Logic
    }

       public static Gizmo_Collider DrawCapsuleGizmos(GameObject gameObject, CapsuleCollider capsuleCollider, Color gizmosColor)
    {
        // Logic
    }

        public static Gizmo_Collider DrawMeshGizmos(GameObject gameObject, MeshCollider meshCollider, Color gizmosColor)
    {
        // Logic
    }
```

In addition to this API you can use the **DrawBox** method to repurpose the drawer to draw any box, anywhere.

- The first parameter is the **size** corresponds to the size of your box.
- The second parameter is the **center** corresponds to the position of the box relative to the position of the drawer.
- The third is the **color** you want to draw with.

```
public void DrawBox(Vector3 size, Vector3 center, Color color)
{
    // Logic
}
```

We plan on completing this API with way more functionalities. We also plan to add support for 2d collider support.

RaycastTo Gizmos

This component has a simple API for now. Notice that for now it always draw from the position of the drawer to another position. If you need a feature to draw from a selected position to another one, please ask on the discord.

Those methods are pretty simple to come around. Either pass a GameObject or a position you want to draw to an then choose the color.

```
public void DrawTo(GameObject target, Color color)
{
    // Logic
}

public void DrawTo(Vector3 targetPosition, Color color)
{
    // Logic
}
```

Console API

The console has an API that passes through the **DebugLog** static class.

```
public static void Log(int i, LogColor logColor = LogColor.Default, LogType logType = LogType.Log)
{
    //Logic
}

public static void Log(bool b, LogColor logColor = LogColor.Default, LogType logType = LogType.Log)
{
    //Logic
}

public static void Log(string message, LogColor logColor = LogColor.Default, LogType logType = LogType.Log)
{
    //Logic
}
```

- The first parameter is always what you wish to print.
- The second parameter is the color that you can select from a pre selected set of colors :

```
    public enum LogColor
    {
        Default, /// color of the log type
        Red,
        Green,
        Blue,
        Yellow,
        White
    }
```

- The third parameter is the log type. Each log type has a predefined color. This color can be overridden by selecting another color then default :

```
public enum LogType
{
    Log, //white
    Warning, // orange
    Error, // red
    Command // blue
}
```

For know only **int**, **bool** and **string** are supported, but feel free to request more functionalities.

About us

At 4Hands2cats, we create debugging tools to streamline development. As two devs (and two cats), we focus on robust, user-friendly solutions for Unity. Our first asset, a complete debug toolkit, works in build and runtime for full control. We're committed to improving and expanding our tools.

FAQ

Where can I contact you to send you feedback or request some features ?

```
- Just come on the discord server and ask your question on the dedicated channel.
```

🚀 What's Next?

Usability

- Custom inspector for all components.
- Completion of the Gizmos API.
- System based on C# attributes to create your own commands directly in code while keeping the ScriptableObject workflow.
- Command auto-completion.

Features

- Cinemachine-based FreeCam.
- Profiling features.
- Export logs and metrics to a file or remote endpoint for post-processing (e.g. spreadsheet analysis).
- VR/XR support for the console.

Known Issues

Awaiting your feedback!

Is there a feature you need? Feel free to ask on our dedicated Discord server and we'll see what we can do.
Our goal is to improve this asset as much as possible so it fulfills all your debugging needs on your Unity Engine journey.

📝 Patch Notes

This section logs all updates and additions to **UDT** since release.

🔷 Version 1.3

✨ Features

- Trello ticketing to simplify bug collection during playtests and post-launch.
- 2D Gizmos support.

🔧 Improvements

- Custom inspectors for gizmos to simplify usage.
- Command prediction in the console.

🔹 Version 1.2.1

🐞 Fixes

- Fixed a major issue in static command validation.
- Removed leftover development logs.

🔷 Version 1.2.0

✨ Features

- Added support for static function commands via code attributes.
- Added graphical plotting for metrics.
- Added metrics for audio and memory usage.

🔧 Improvements

- More icons for plugin scripts.
- General performance improvements.

- Improved navigation inside the console.

🔷 Version 1.1.0

✨ Features

- Added runtime gizmo debug mode for NavMesh agents.
- Added world-space UI for NavMesh agents.

🛠️ Quality of Life

- Expanded API for `Gizmos_RaycastTo` and `Gizmos_Collider` components.
- New icon for command scriptable objects.
- Simplified command creation process.

🐞 Fixes

- Fixed a crash when using an empty string as a command.
- Fixed console text area layout issues on rapid size increase.
- Fixed issue with trailing spaces breaking command execution.
- Fixed multi-keyword command parsing inconsistencies.
- Improved performance of the interactive console.

🎉 Version 1.0.0 – Initial Release

Included

- In-game console.
- In-game gizmos.
- In-game FreeCam.
- Basic metrics system.