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. Discord

---

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.

Introduction to UDT

This toolkit aims to make your life easier while debugging your game.
In this version we focused on the run time debugging

We are aiming for a load of improvements in the future versions of this toolkit. If you want to now more about the next features please check the What's Next section.

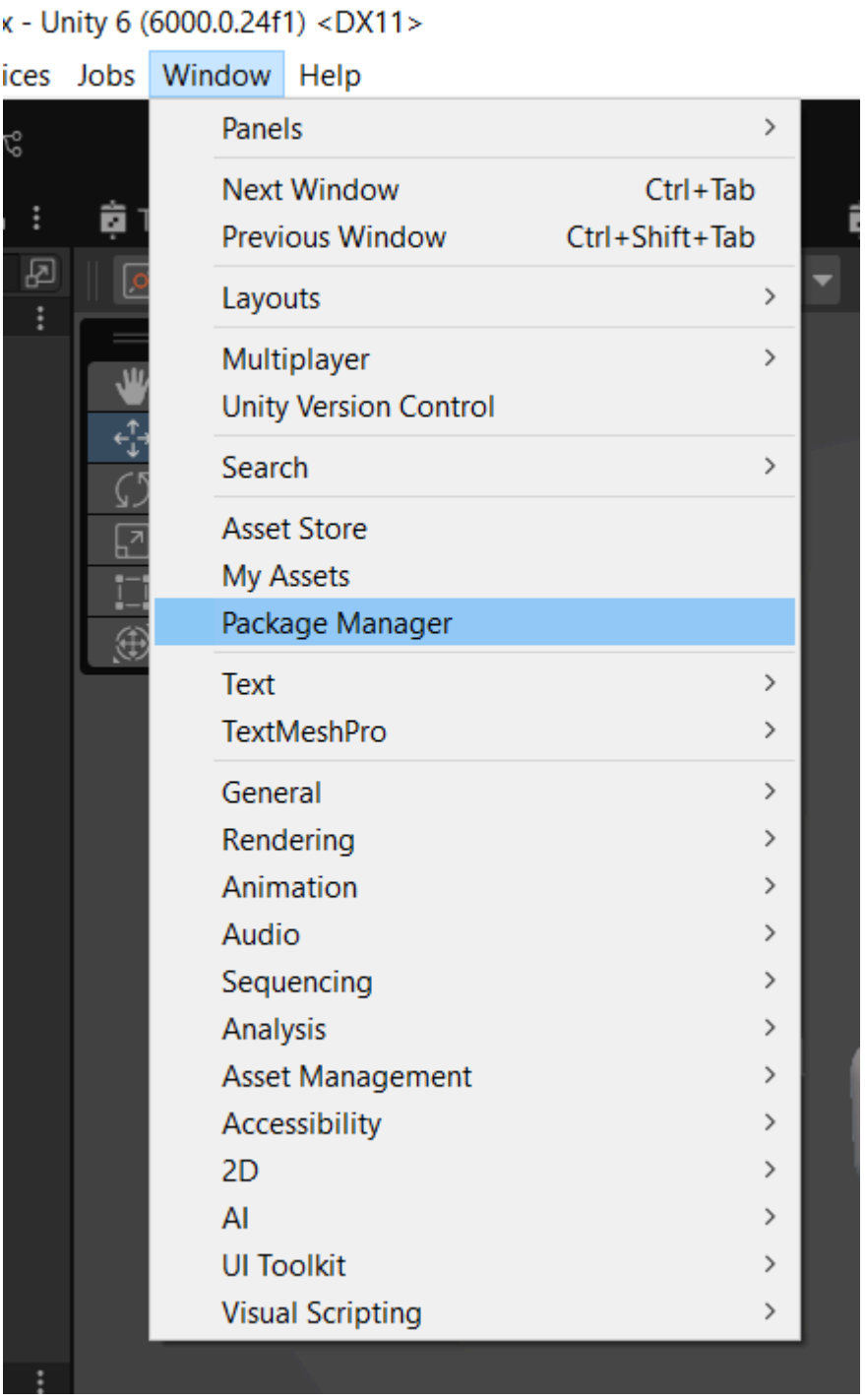
In the next few sections you are going to learn how to import UDT to your project and how to use it.

Install UDT

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

Package manager

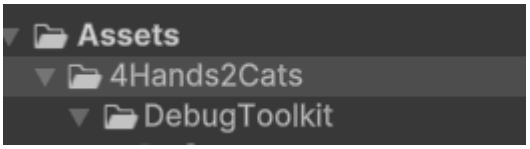
Get into the unity package manager at the top of the Unity Engine window and look for UDT in your assets



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

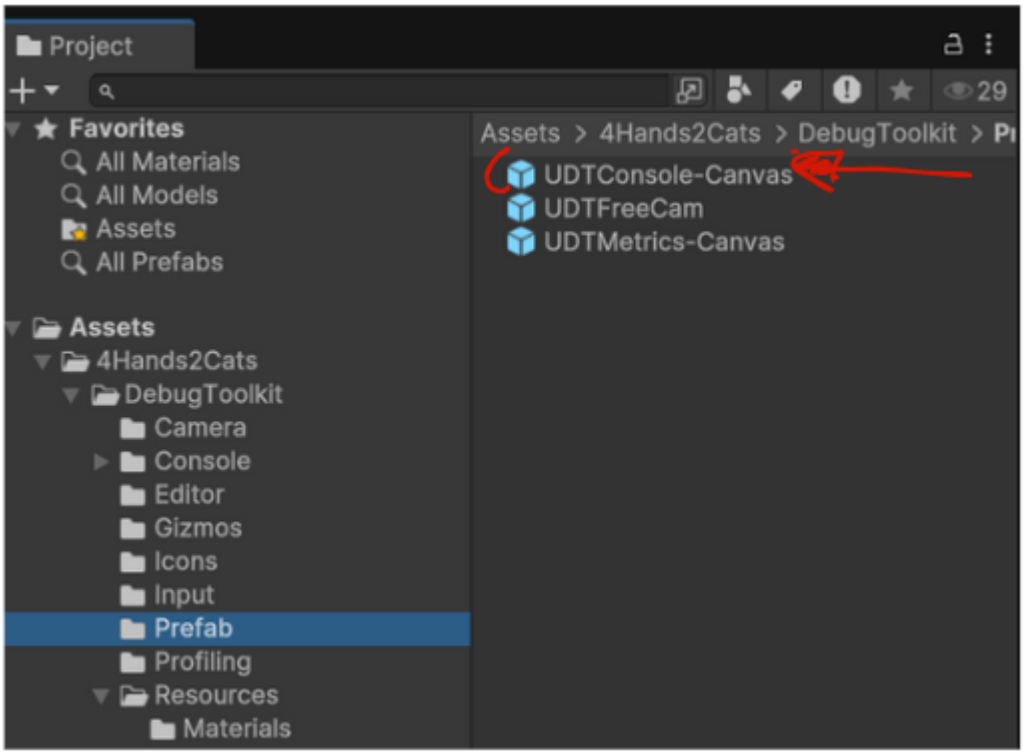
Once the files are downloaded, import them in your project by clicking on the **Import** button.

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 guide

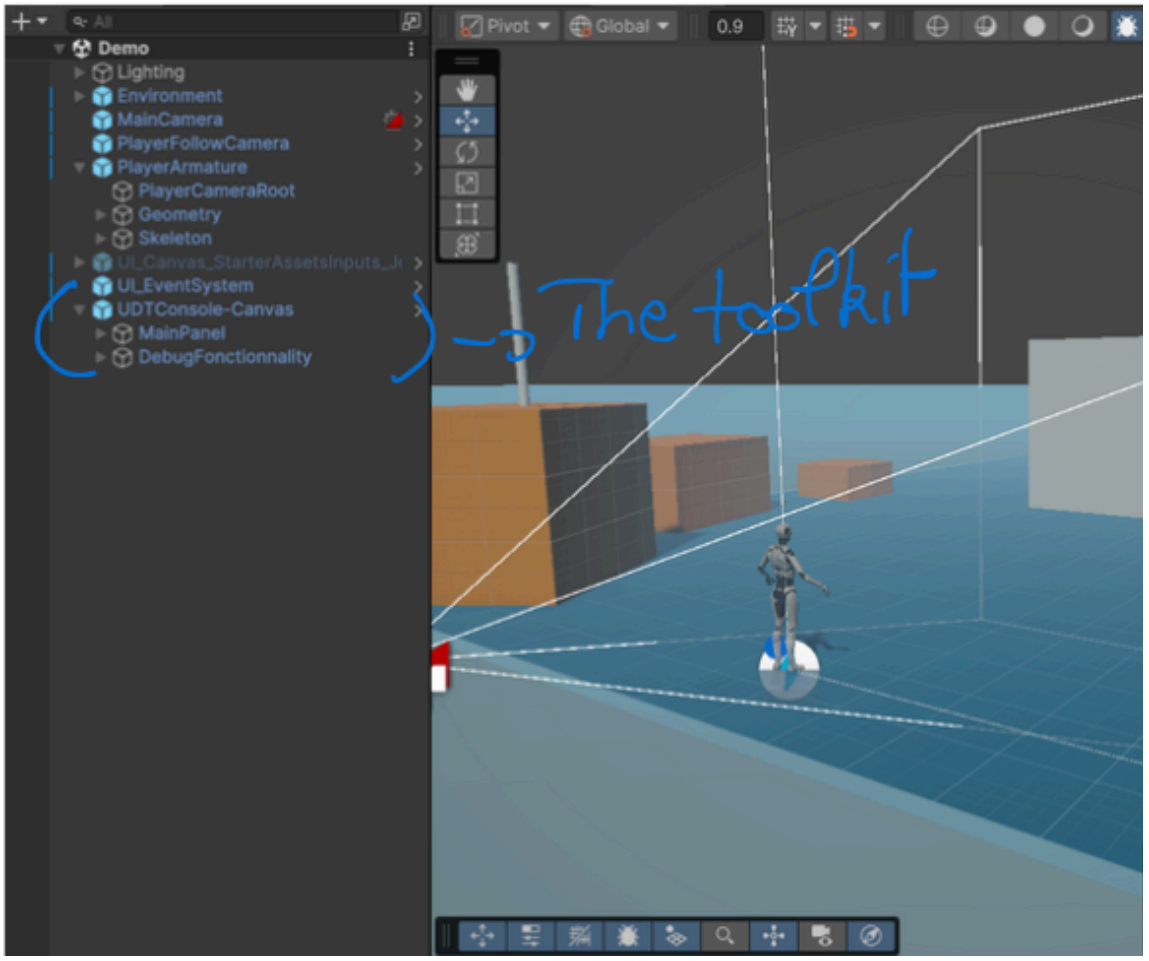
As you just seen in the demo scene, the toolkit is pretty simple to include in your project. You just have to drag and drop the **UDTConsole-Canvas** prefab from the prefab folder inside your scene.



Don't forget to remove it for production builds.

Demo Scene Tour

The demo scene is based on the third person template of Unity. We've added our debug toolkit in this scene.



To use this demo scene just press play. The Console and the all the other features are on the **F12** key.

To navigate in the scene use the **WSDQ** keys and the mouse to control the camera.

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.

Free Cam

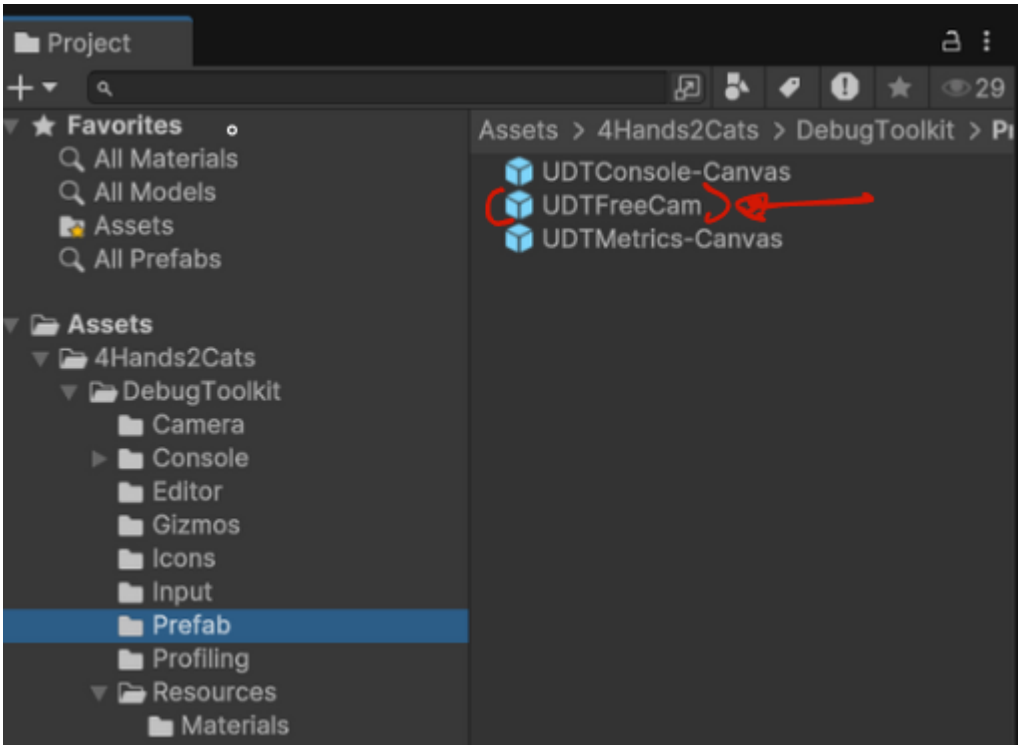
The free cam is here to give you a tool that's similar to the navigation in the scene panel of unity, but at runtime.

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

Though this embeded in the package, the freecam comes as stand alone feature. Feel free to use it for your gameplay if you want.

> *Future versions of the cam should be compatible with cinemachine.*

The prefab for this one is in the same folder as the console.



To use as a stand alone feature just drag and drop the prefab in your scene.

> *Note : If you use it as stand alone feature it'll not be control anymore by the console.*

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.

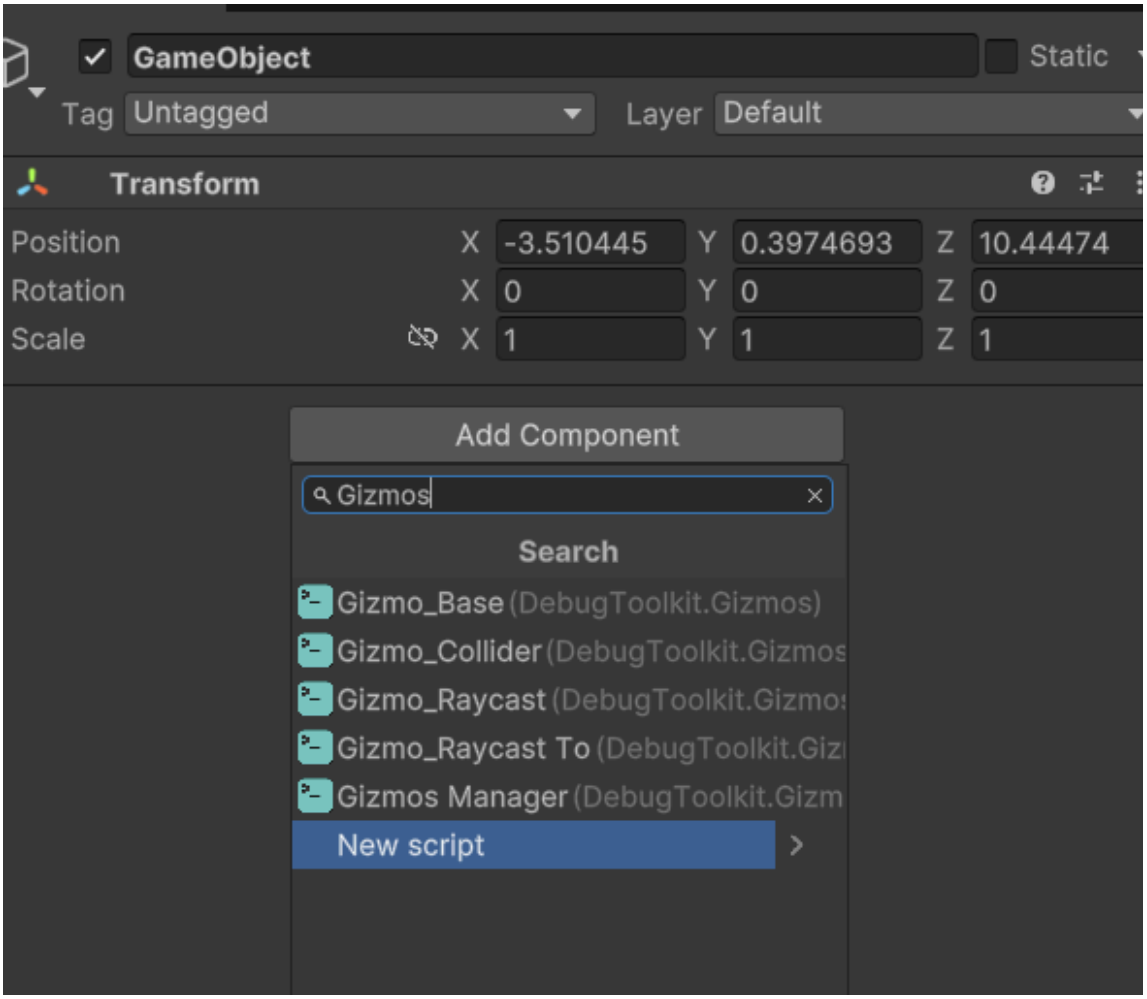
There are two way of using the gizmos. Manual and automatic using the console.

- 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.

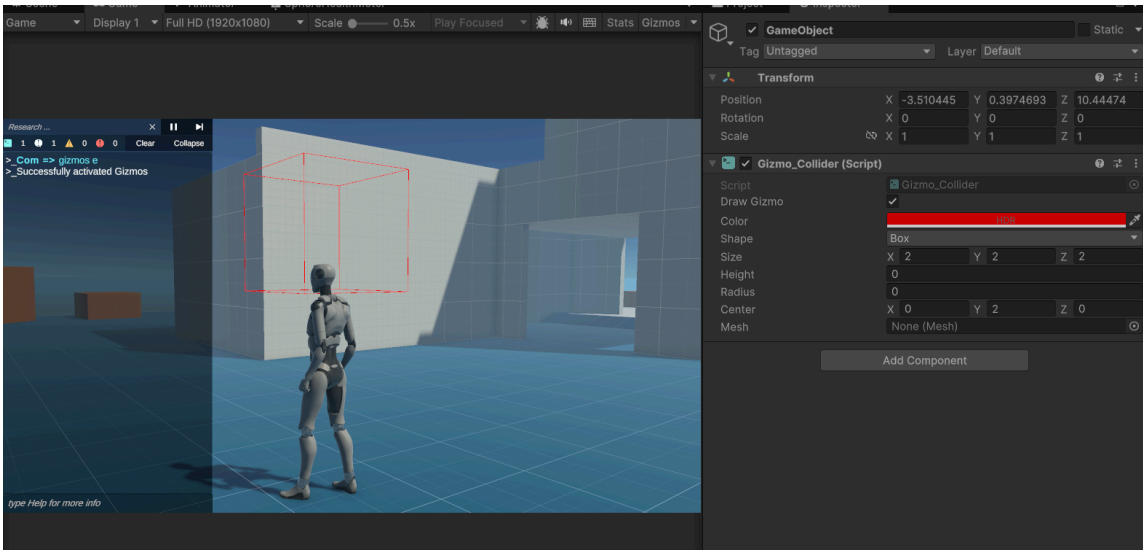
Gizmos that are manually setted up can are still managed by the ingame console. Simple type : *Gizmos/gizmos/-g/-G* followed by *enable/e* or *disable/d* to enable or disable the in game gizmos.

> *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.

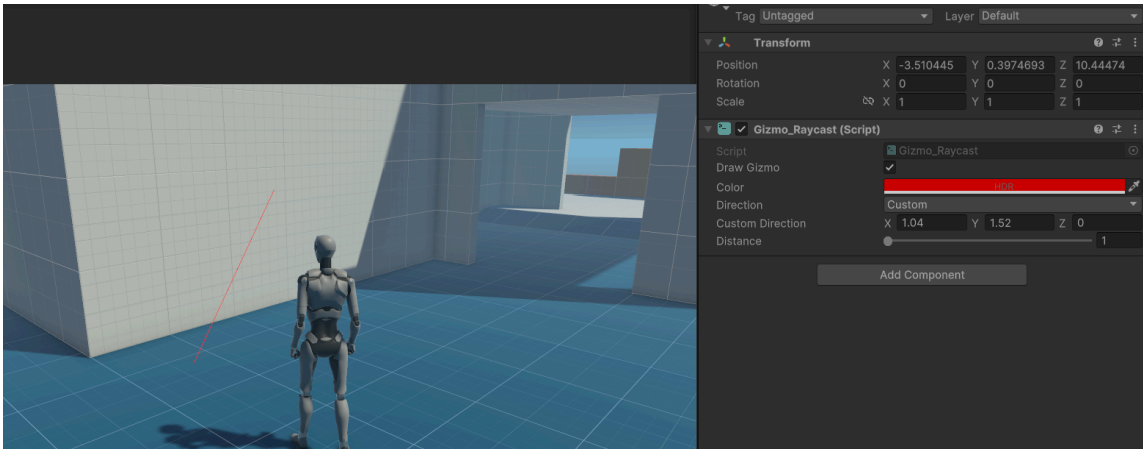


There are four options :

- 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].

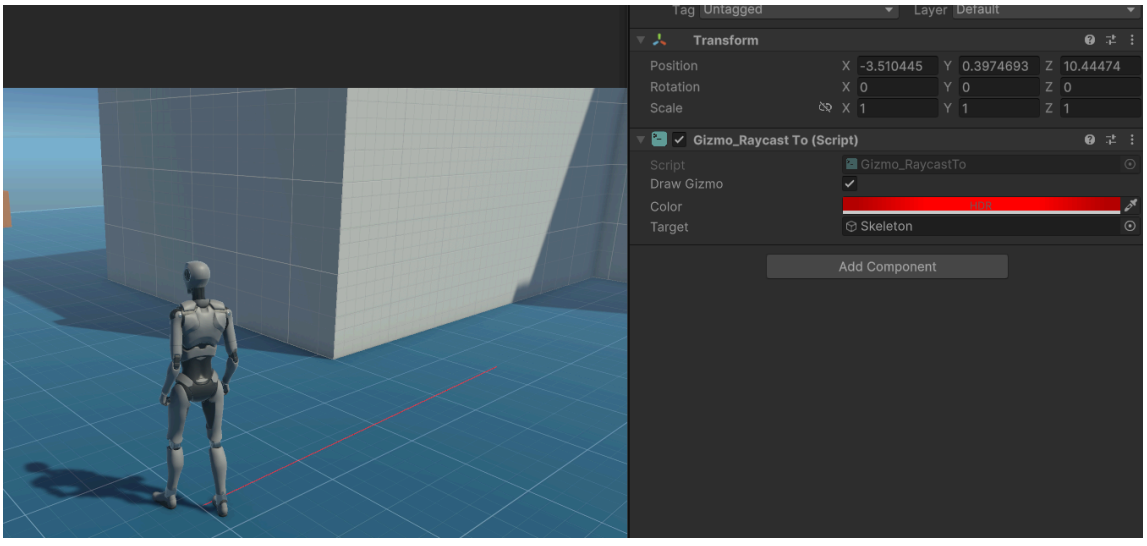
> *Note that those gizmos support bloom and they can be used as part of your project for other purposes than debugging.*

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.



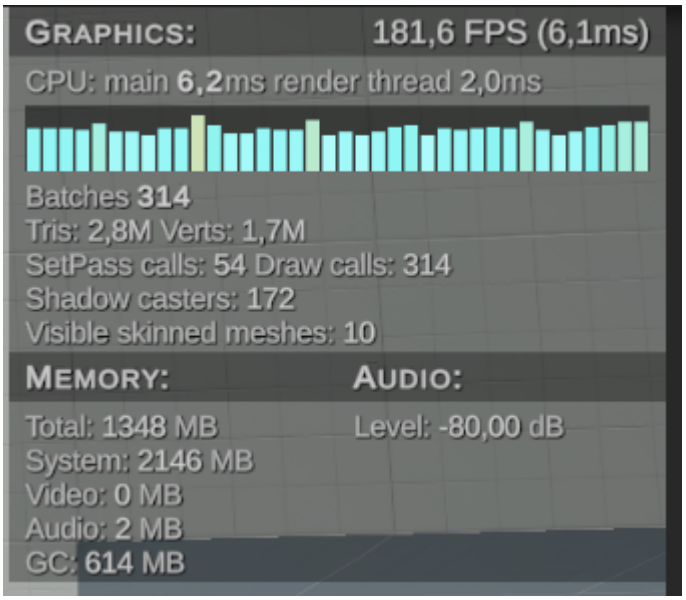
You just have to give it a ref to a gameobject and it'll draw a ray to it and update the ray at everyframes.

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:

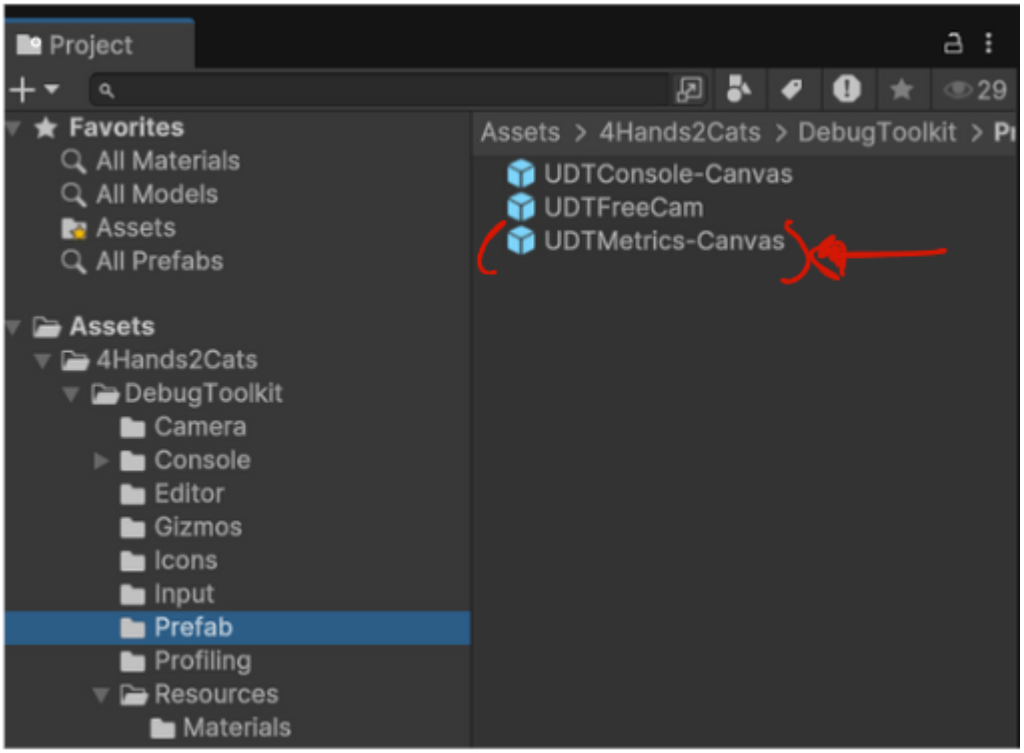
```
Engine
System
Video
Audio
Garbage collection
```

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.*

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
    }
```

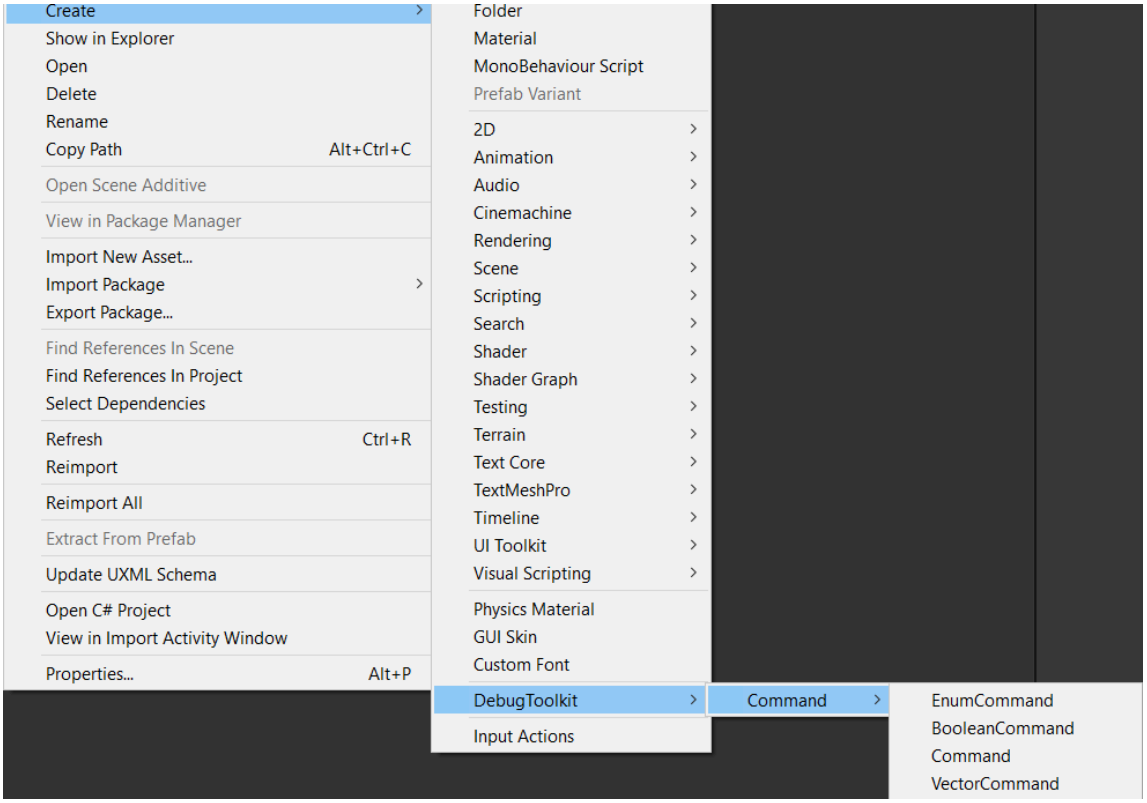
> *For know only **int**, **bool** and **string** are supported, but feel free to request more functionalities.*

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.

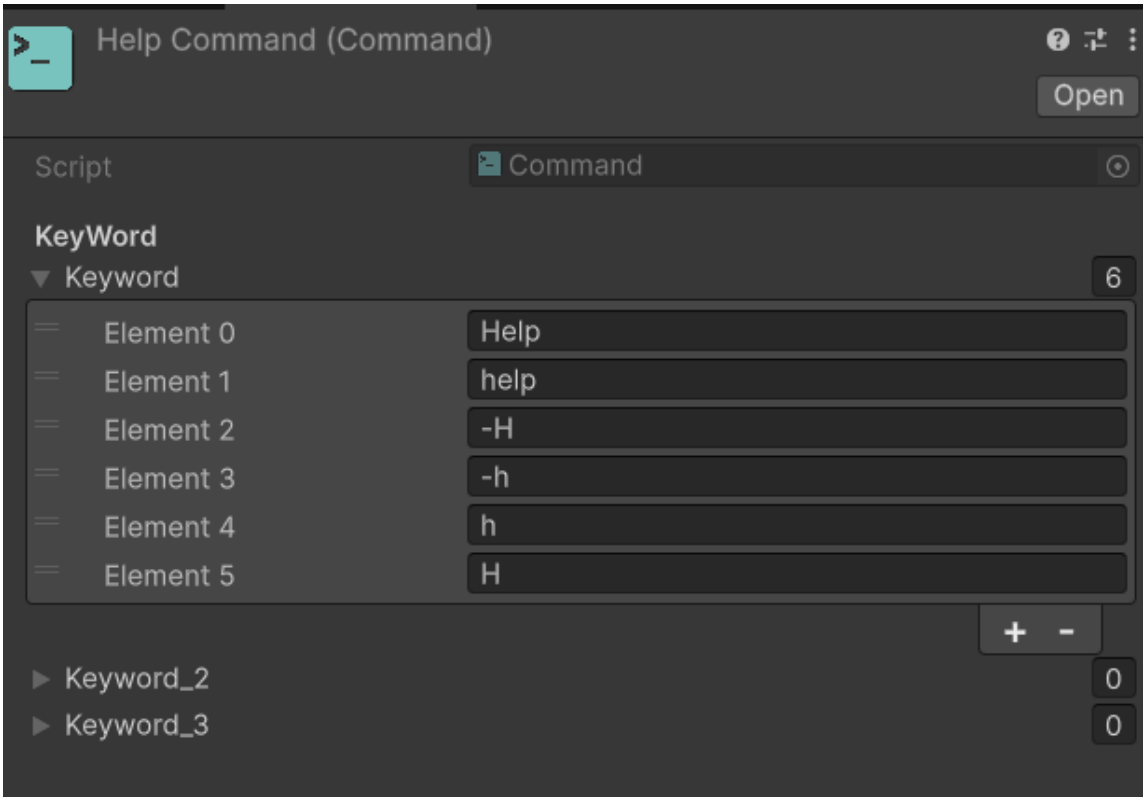
> *We are planning on making an API to create commands using C# attributs which'll give you two options to create your console commands.*

You can create like this

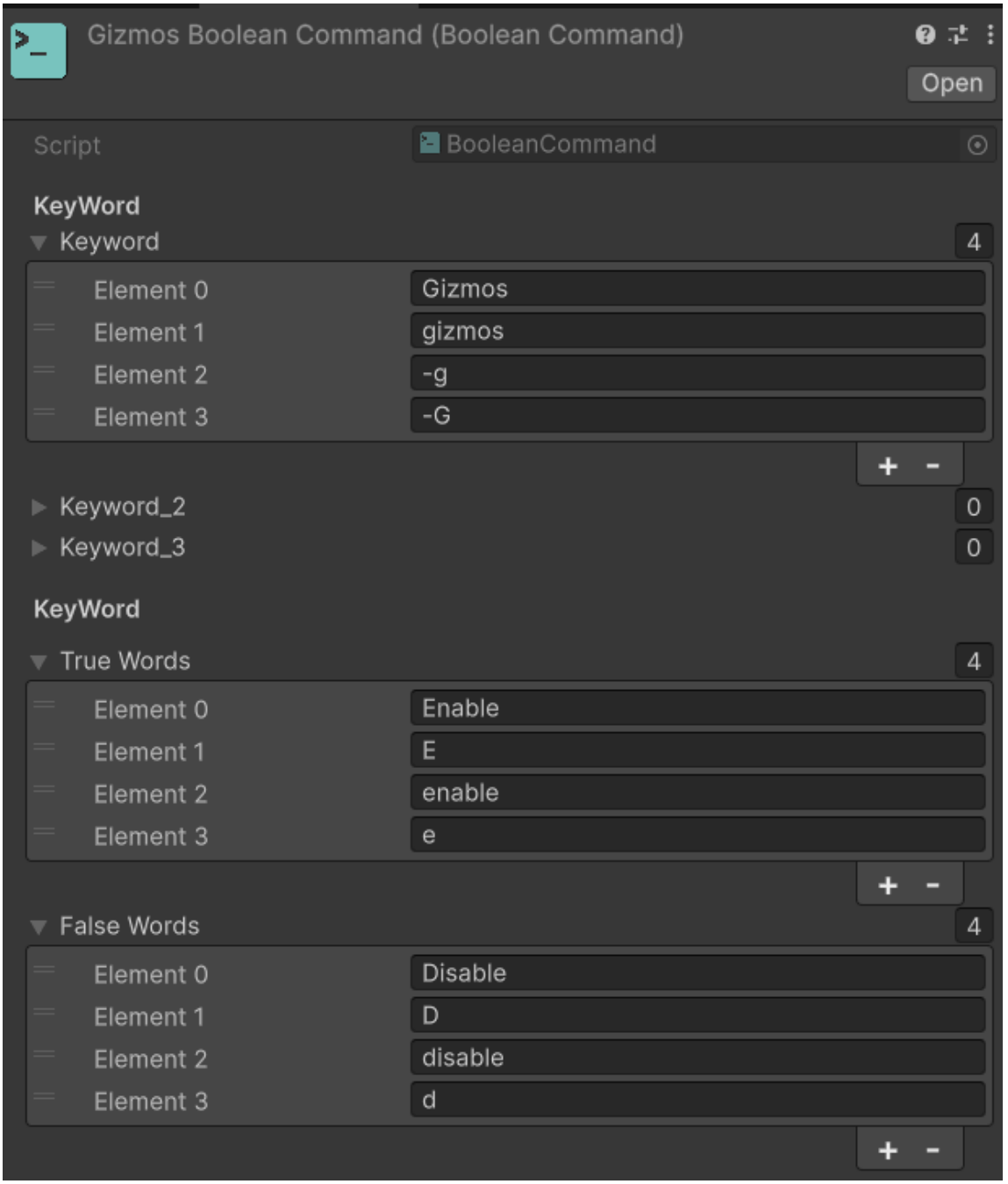


There are four type of commands :

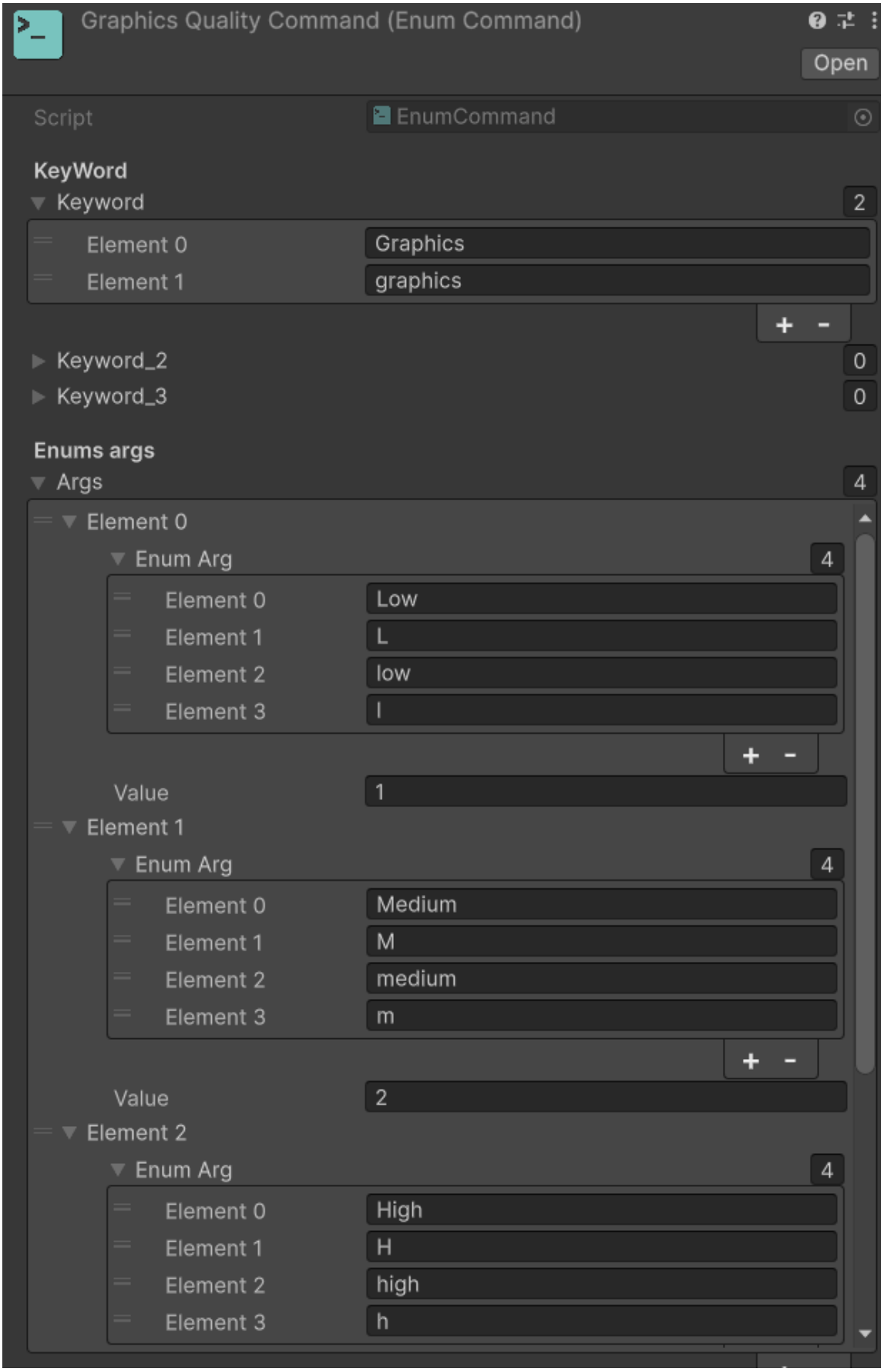
- The **Command** is the most basic one it is used to simply inputs a chain of up to three words that would send a signal of type => Command Got Activated. This is the one which is used for the help command per example.



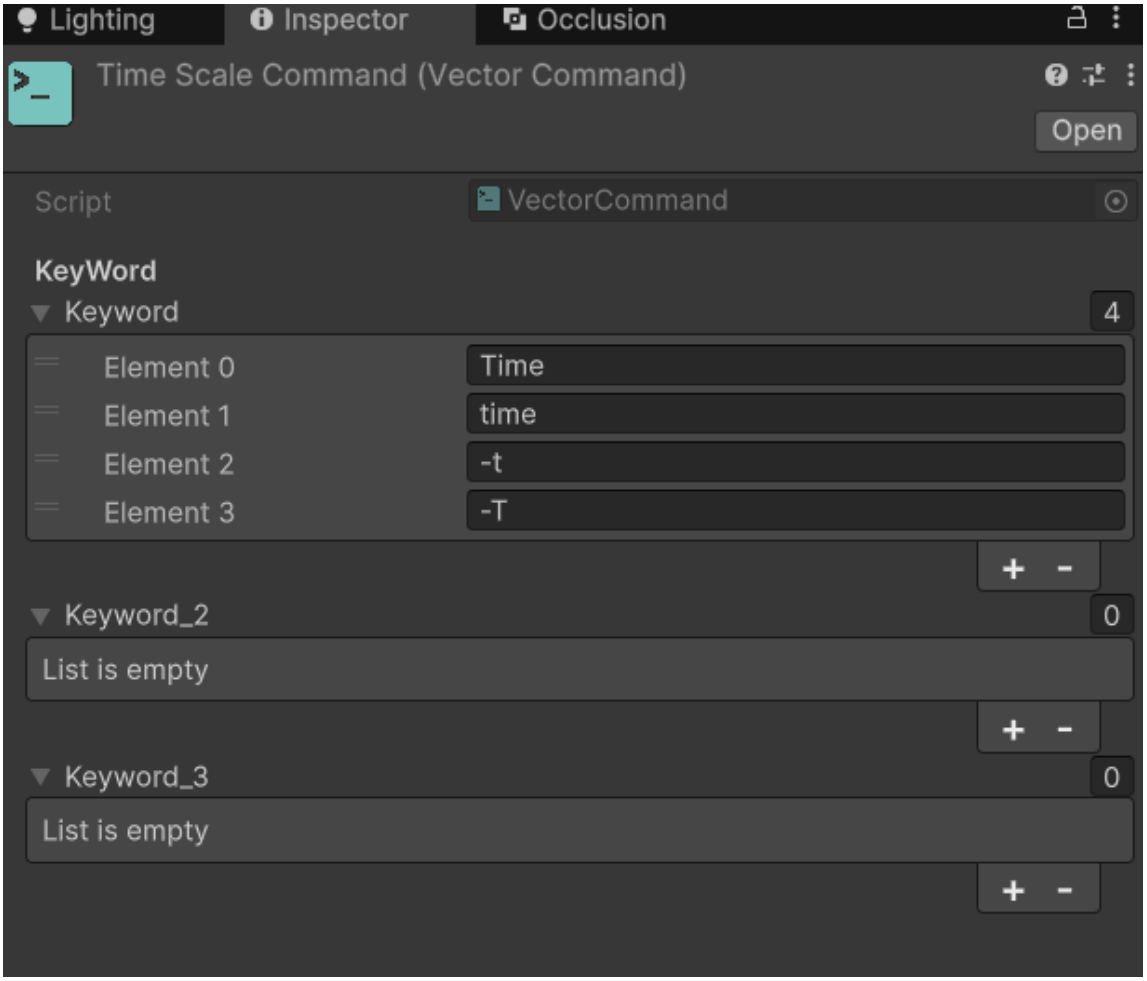
- The **Boolean Command** which takes two arguments to send either a signal activate or deactivated (true or false). This used for the Gizmos command per example. You've got the main keywords and then the keyword for true and the keyword for false.



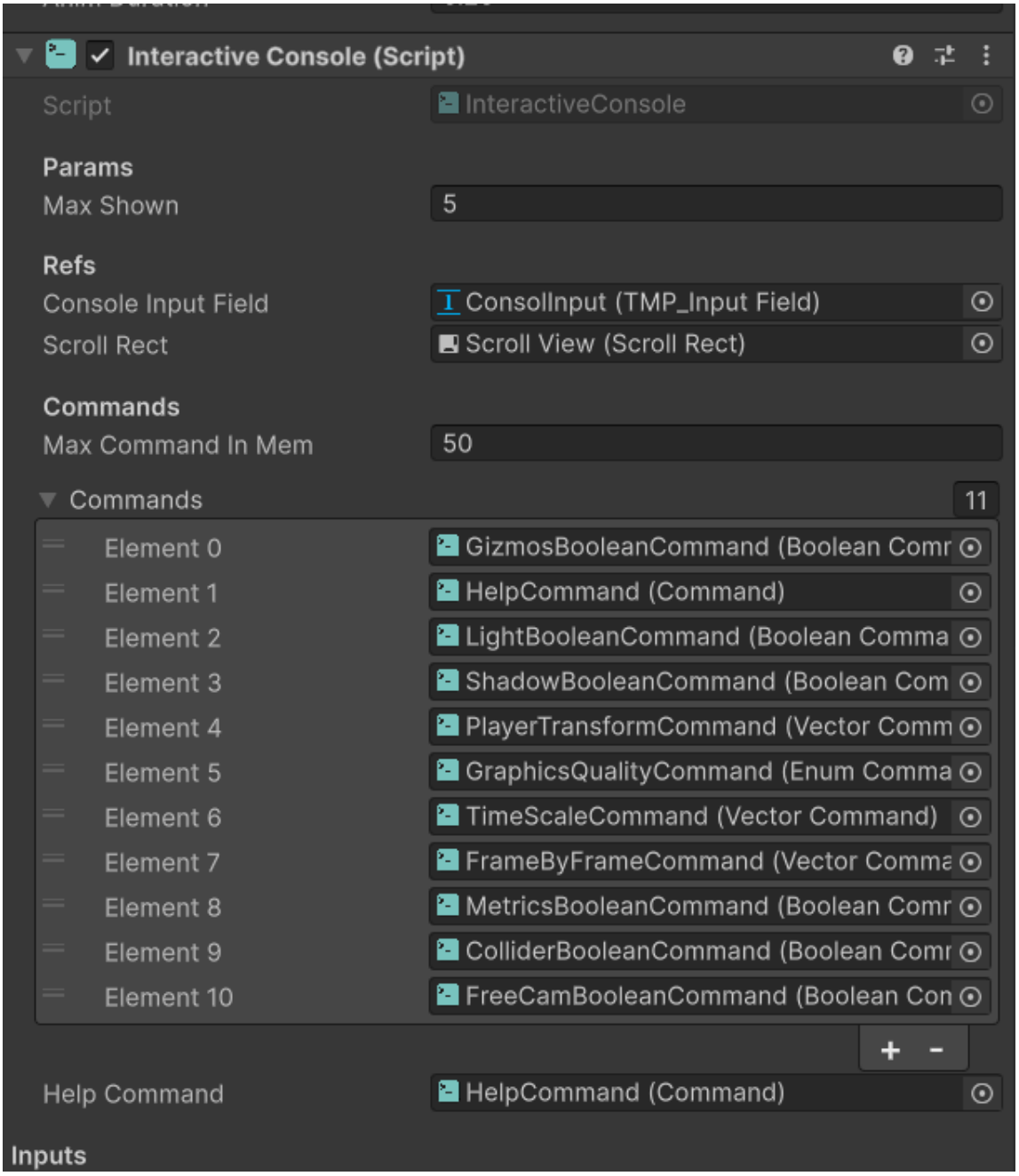
- The **Enum Command** which take n arguments to send a signal between 0 and n - 1. Its your job to handle what those index mean. You can bind those index with a keyword each so its easier to use in the in game console. This command is use in the Graphic Quality Command per example.


Graphics Quality Command (Enum Command)
Open
Script
EnumCommand
KeyWord
Keyword
2
Element 0
Graphics
Element 1
graphics
Keyword_2
0
Keyword_3
0
Enums args
Args
4
Element 0
Enum Arg
4
Element 0
Low
Element 1
L
Element 2
low
Element 3
l
Value
1
Element 1
Enum Arg
4
Element 0
Medium
Element 1
M
Element 2
medium
Element 3
m
Value
2
Element 2
Enum Arg
4
Element 0
High
Element 1
H
Element 2
high
Element 3
h

- 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("Command")]
    [SerializeField] private VectorCommand timeScaleCommand;
    [SerializeField] private VectorCommand frameCommand;

    [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 the 4Hands2Cats/DebugToolkit/Console/Interaction/CommandData folder.*

Static commands

> *Introduced in V1.2*

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*

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
- Completing the Gizmos API
- A system based on C# attributes to make your own commands directly in code while keeping the scriptable object work flow.
- Command auto-completion

Features

- A cinemachine freeCam
- Profiling features
- A way to create your own commands using a code API.
- A way to print the logs and the metrics to send them to an adress or post process the result using a spreadshit.
- 2D support for colliders
- Mobile device support for the console.
- VR/XR support for the console.

Known issues

Awaiting for your feedbacks

Is there a feature you need? Feel free to ask on the dedicated discord server and we'll see what we can do. Our goal is to improve this asset as much as possible so it fullfills all your needs in term of debuging in your Unity Engine journey.

Patch Note

This section is the log of all the changes and additions to UDT since release.

V1.2.0

Features

- Adds Commands for static functions as code attributes
- Adds Metrics graphic plot
- Adds metrics for audio and memory usage

Improvements

- More icons for the scripts of the plugin
- Performance improvements
- Console navigation improvement

V 1.1.0

Features

- Adds a runtime gizmos debug mode for the navmesh agents
- Adds a runtime worldspace info UI for the navmesh agents

Quality of life improvement

- More API for the Gizmos_RaycastTo component.
- More API for the Gizmos_Collider
- New icon for the scriptable objects of the commands
- Simplifies the way the commands have to be made

Fix

- Fixies an issue where an empty string would throw an error

- Fixies an issue where the console text area would sometime bug out when increasing too fast in size.
- Fixies an issue where having a space at the end of a command would fail its process.
- Fixies a bug where multi keyword commands would not always work.
- Improves performance of the interactive console.

V 1.0.0

UDT Release :

- In game Console
- In game gizmos
- In game FreeCam
- In game basic metrics