



Quick Trigger Interaction

Official Tool Manual

Version 1.0

Introduction	4
First Steps	5
1. Creating a Trigger	5
2. Adding Interactions	6
3. Interaction Chaining	7
4. Trigger Replacing and the Interactor	10
5. Triggering interactions with the player	11
6. Understanding the Input Trigger	12
Triggers	15
1. Unity Lifecycle Triggers	17
1.1. OnAwakeTrigger	17
1.2. OnEnableTrigger	17
1.3. OnStartTrigger	17
1.4. OnUpdateTrigger	17
1.5. OnFixedUpdateTrigger	17
1.6. OnLateUpdateTrigger	18
1.7. OnDisableTrigger	18
1.8. OnDestroyTrigger	18
2. Physics Triggers	19
2.1. Unity 3D Physics Triggers	20
2.1.1. CollisionEnterTrigger	20
2.1.2. CollisionExitTrigger	20
2.1.3. CollisionStayTrigger	20
2.1.4. StepOnTrigger	20
2.1.5. StepOffTrigger	20
2.1.6. StayOnTrigger	21
2.2. Unity 2D Physics Triggers	22
2.2.1. CollisionEnter2DTrigger	22
2.2.2. CollisionExit2DTrigger	22
2.2.3. CollisionStay2DTrigger	22
2.2.4. StepOn2DTrigger	22
2.2.5. StepOff2DTrigger	23
2.2.6. StayOn2DTrigger	23
3. Mouse Triggers	24

3.1. MouseDownTrigger	24
3.2. MouseDragTrigger	24
3.3. MouseUpTrigger	24
3.4. MouseUpAsButtonTrigger	24
3.5. MouseEnterTrigger	25
3.6. MouseExitTrigger	25
3.7. MouseOverTrigger	25
3.8. PointerDownTrigger	25
3.9. PointerUpTrigger	26
3.10. PointerClickTrigger	26
3.11. PointerEnterTrigger	26
3.12. PointerExitTrigger	26
3.13. PointerMoveTrigger	27
3.14. PointerDragTrigger	27
3.15. PointerDropTrigger	27
4. Other	28
4.1. OnDistanceTrigger	28
4.2. OnDistance2DTrigger	28
4.3. InputTrigger	29
4.4. Input2DTrigger	30
4.4.1. Interaction Visual	30
4.5. KeyTrigger	31
5. Inspector Shortcuts	32
Interactions	33
1. Built-in Interactions	34
1.1. ActivationInteraction	34
1.2. AnimationInteraction	34
1.3. AudioPlayOneShotInteraction	35
1.4. AudioSourceInteraction	35
1.5. DelayInteraction	36
1.6. DestroyComponentInteraction	36
1.7. DestroyInteraction	36
1.8. InstantiateInteraction	36
1.9. UnityEventInteraction	38
1.10. ConditionInteraction	38
1.11. AddForceInteraction	38
1.12. SetPositionInteraction	40
1.13. SetRotationInteraction	40
1.14. SetScaleInteraction	41
1.15. DialogueInteraction	41
1.16. DebugLogInteraction	42
2. Inspector Shortcuts	43
Interactors	44
1. IInteractor	44
2. Interactor	44

3. IInputInteractor	44
4. IInput2DInteractor	44
Custom Triggers	45
Custom Editors	45
Custom Interactions	46
Custom Editors	46
Graph View	47
1. Opening Graph View	47
2. Window Overview	47
3. Node Overview	48
Settings	49
1. InteractionsSettings	49
2. PrioritiesEnum	50
3. Graph View Settings	51
DemoScenes	52
1. Showcase Demo	52
2. First Person Puzzle Demo	52
3. Third Person Demo	53
4. 2D Platformer Demo	54
Credits	55

Introduction

The purpose of this tool is to create a way for the developer to quickly implement an ecosystem of interactions in the game world without ever having to open a C# script but providing an intuitive way to do it if so desired.

Interactions are comprised of multiple common in-game actions like:

- Activating/Deactivating an Object.
- Instantiating/Destroying an Object.
- Playing an Animation using Unity Animator or changing its parameters.
- Playing Audio. (optional integration with FMOD).

There's also the `UnityEventInteraction` for full customizable actions.

The other part of the tool is Triggers, this is how you spark the desired interactions.

- *e.g.*: The player can trigger an Interaction by stepping on a collider with a `StepOnTrigger`.

You can also expand the system for custom use with your own Interactions by expanding our classes which makes them automatically integrate with the rest of the system by working with all Triggers and other Interactions.

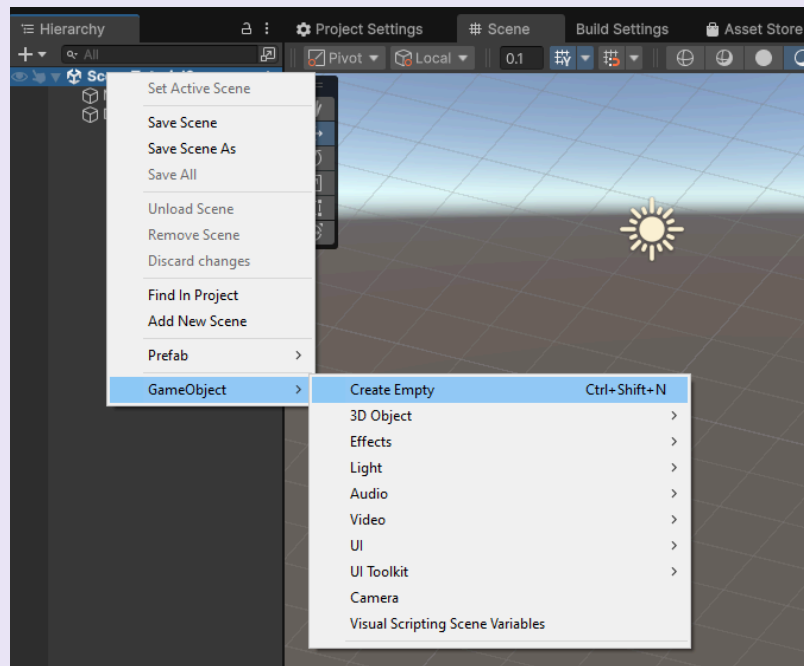
There is no further setup needed, you can just start using the new components and setting up your Triggers and Interactions, but for an easy start tutorial you can follow the steps in the following chapter:

First Steps

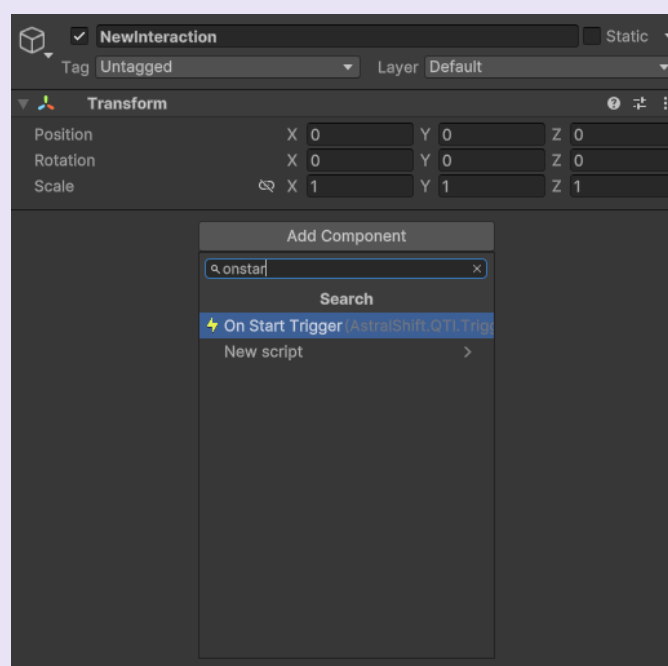
The first step is to download and import the package. After that, simply follow these straightforward steps for a quick setup tutorial.

1. Creating a Trigger

Start by creating an empty GameObject and naming it “NewInteraction”.

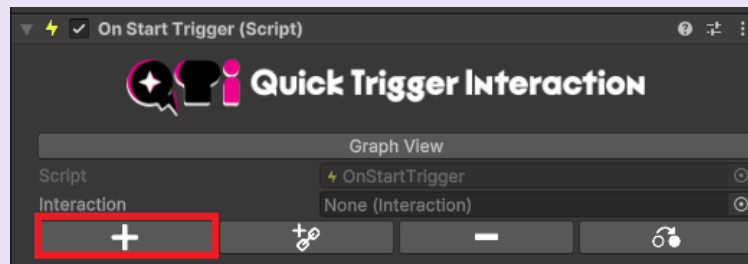


Select the “NewInteraction” GameObject and add an [OnStartTrigger](#) component in the inspector:

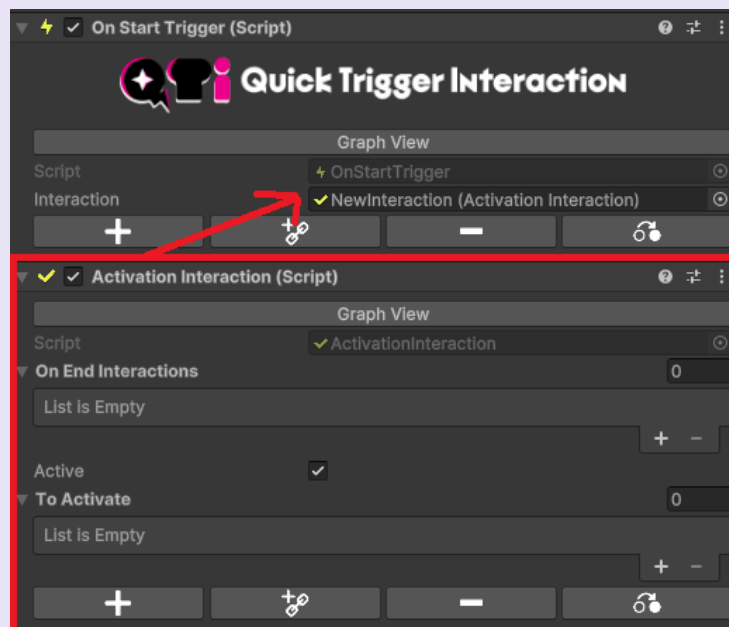


2. Adding Interactions

Click the [Add Interaction](#) button.



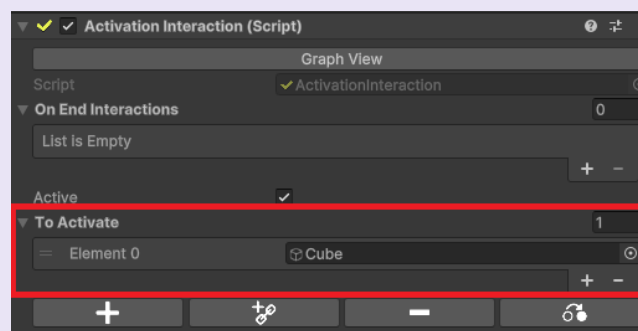
Select the [ActivationInteraction](#). You'll notice that the new Interaction is automatically assigned to the Interaction field of the Trigger.



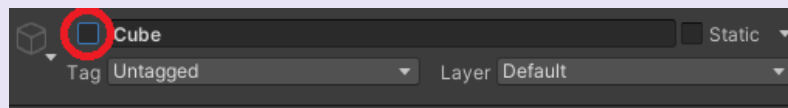
Now with the [ActivationInteraction](#) added we should add a GameObject to activate in the **ToActivate** List.

To do that, let's create a **Cube** by going to **GameObject -> 3D Object -> Cube**.

Drag the **Cube** to the newly created [ActivationInteraction](#) in the **ToActivate** Section.



Next, disable the **Cube**:

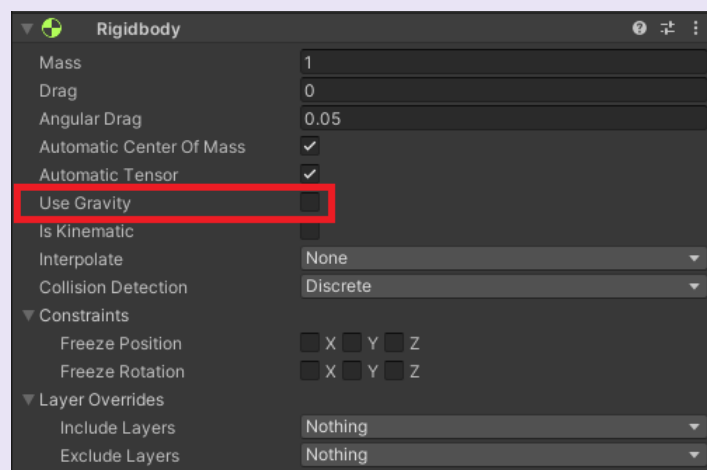


And now we can play the game and see it getting activated via [OnStartTrigger!](#)

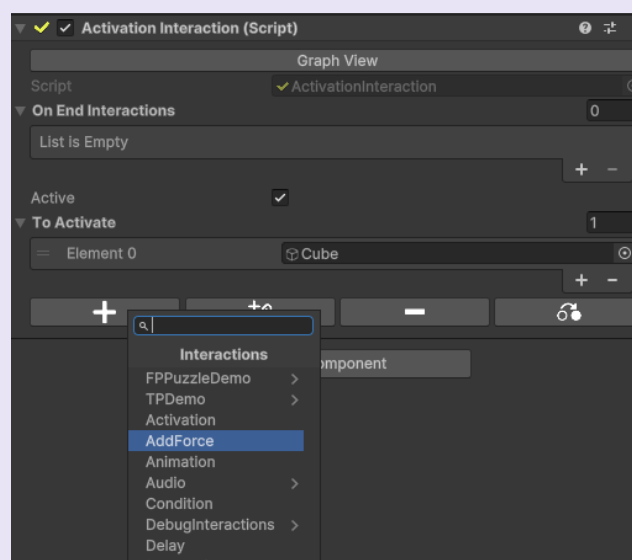
3. Interaction Chaining

Now let's add an [AddForceInteraction](#) to our interactions chain, so the **Cube** can be launched after being activated.

First off we wanna make sure that our **Cube** can react to physics, for that let's select our **Cube** and **AddComponent -> Rigidbody** and disable **UseGravity**.

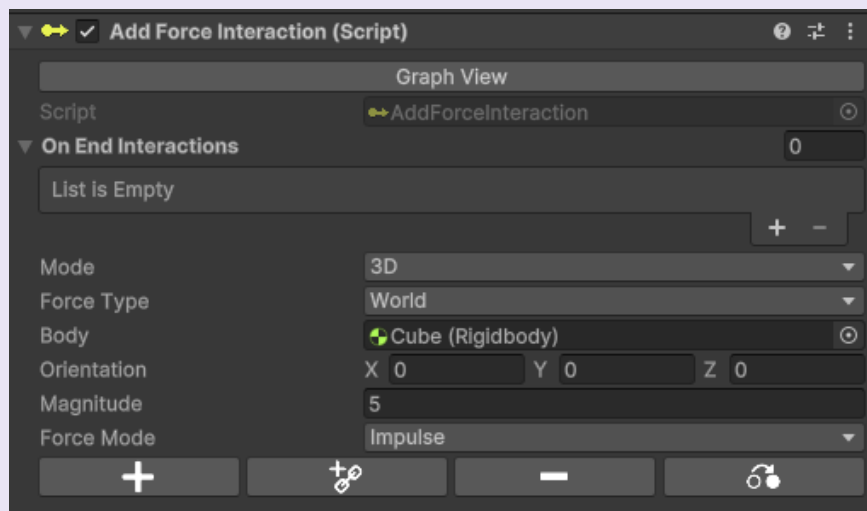


Now add an [AddForceInteraction](#) using the '+' shortcut in the [ActivationInteraction](#), this will make it so the [AddForceInteraction](#) will run after the [ActivationInteraction](#), getting automatically linked to its onEndInteractions list.



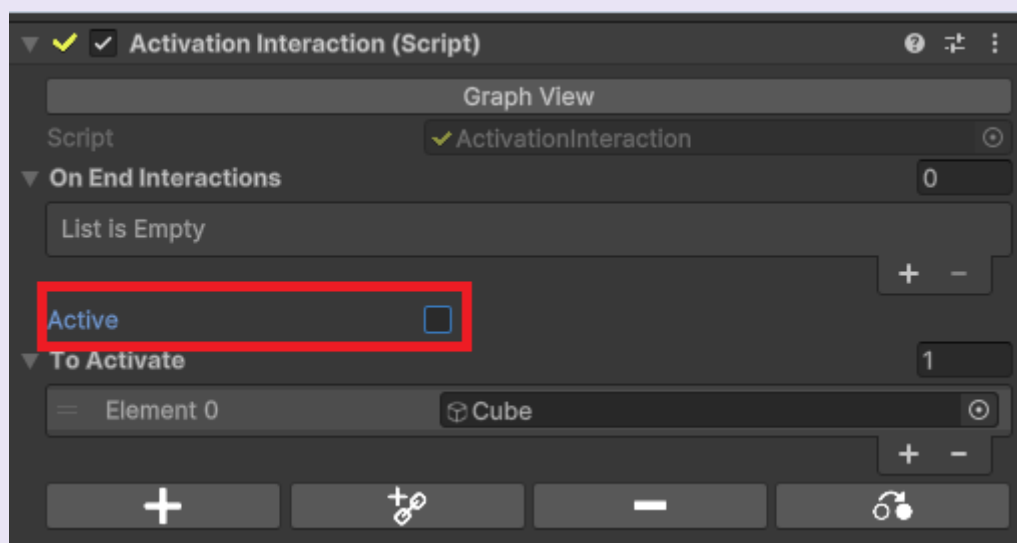
Then you must assign the **Cube** to the **Body** field, and change **Magnitude** to 5 and **ForceMode** to **Impulse**.

Of course these values may change depending on your configuration.



Enter Play Mode and you can see the **Cube** being pushed!

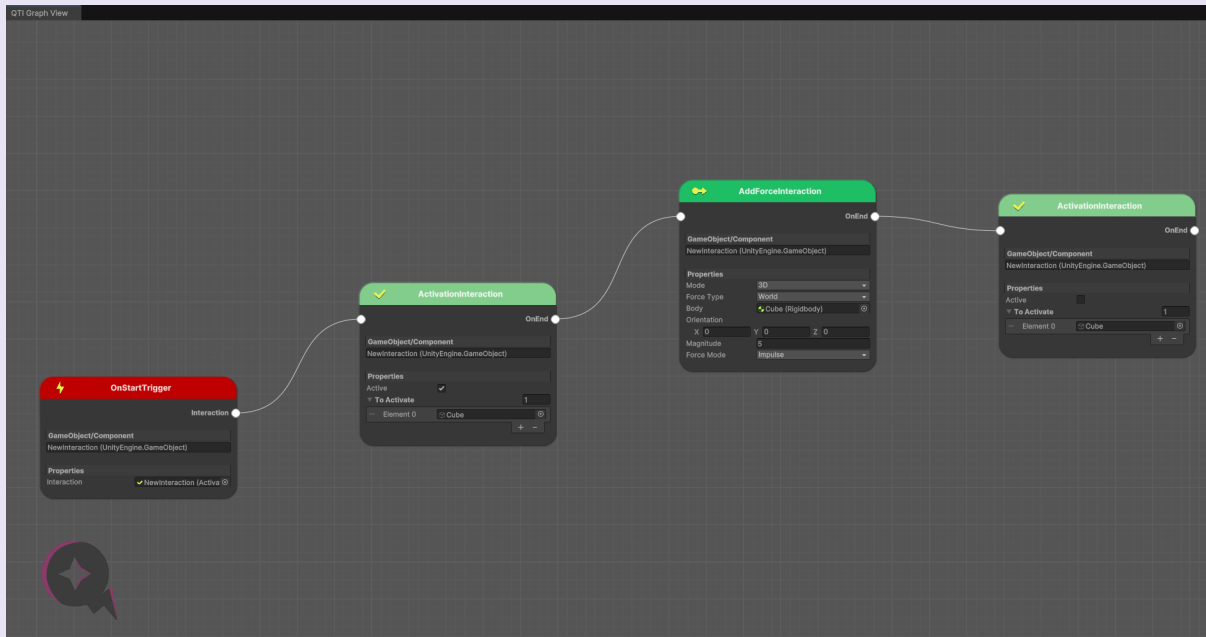
Now let's add another [ActivationInteraction](#) with the purpose of deactivating the **Cube** after it gets pushed. Remember how to do that?



That's right! You use the '+' button on the [AddForceInteraction](#) to make the new interaction happen right after that one. Remember to set the [ActivationInteraction](#)'s **Active** flag to false so the **Cube** gets deactivated instead of activated.

If you hit play now you won't see anything happen! Actually the **Cube** now doesn't even activate... Well, it seems that way but in reality what's happening is the whole interaction chain is occurring in one frame, which means activating, adding force and deactivating it in the same frame resulting in only the deactivation of the **Cube** at runtime.

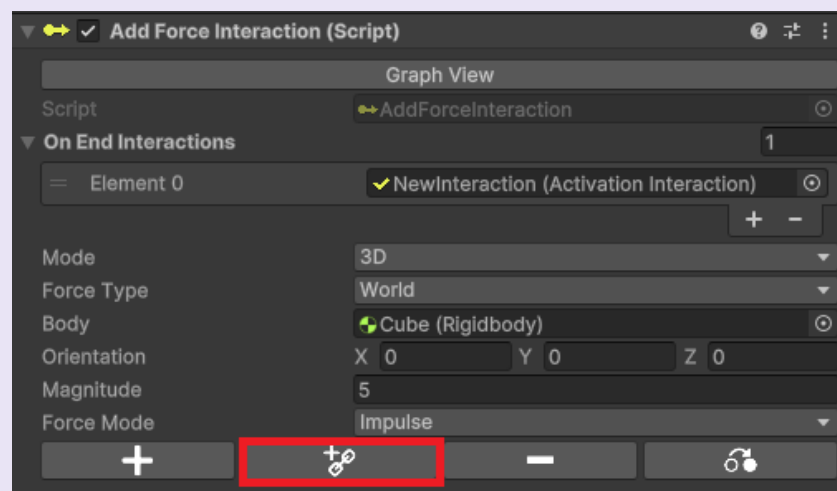
The better visualize this you can use the '[Graph View](#)' button which will show you something akin to this:



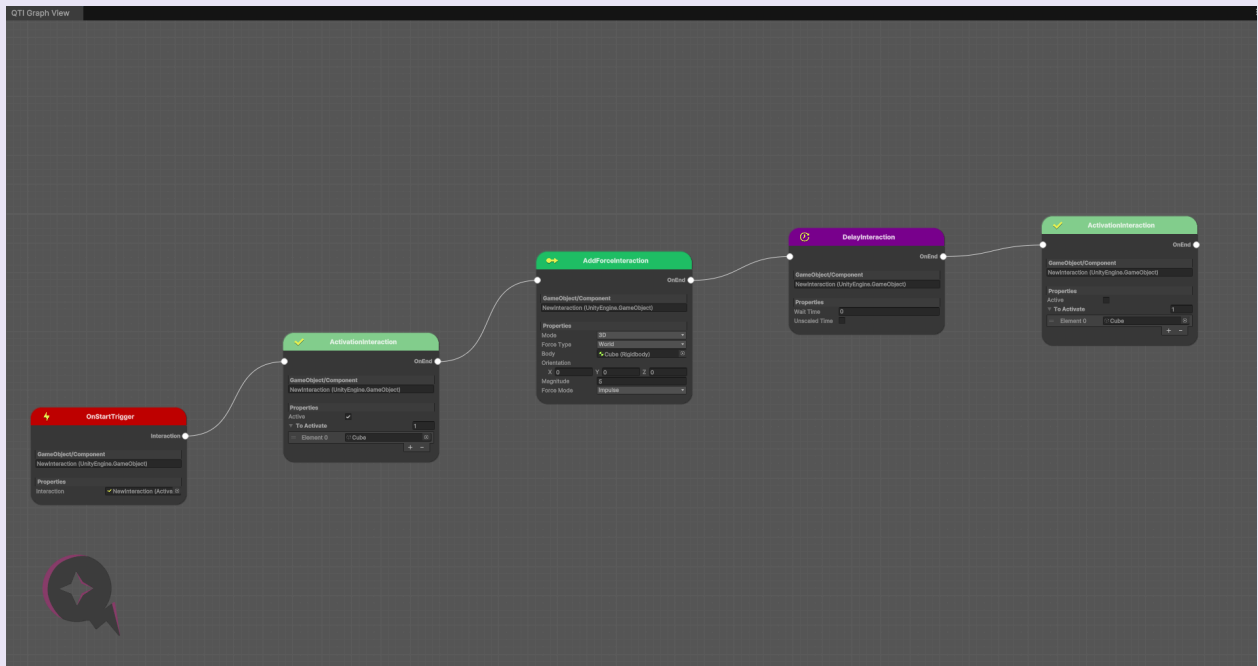
You can see the [OnStartTrigger](#) starting the **interaction chain**, ending with an [ActivationInteraction](#) which **deactivates the Cube**.

In order to achieve our original goal we'll want to allow the **Cube** some time to be pushed and get deactivated afterwards.

For that we'll want to insert a [DelayInteraction](#) between both Interactions. We can achieve that by pressing this button on the [AddForceInteraction](#):



If we go into the [Graph View](#) again, you'll notice that the [DelayInteraction](#) now fits neatly between those two interactions!



Just set the **WaitTime** in the [DelayInteraction](#) to 5 so you can give the **Cube** a good five seconds before it gets deactivated.

And voilà, if you hit play you can see our intention come to fruition!

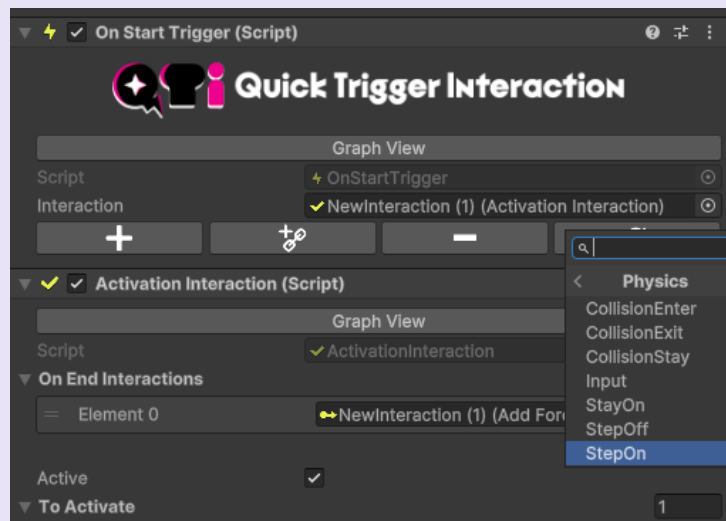
4. Trigger Replacing and the Interactor

Now let's try the same interaction but using a different trigger.

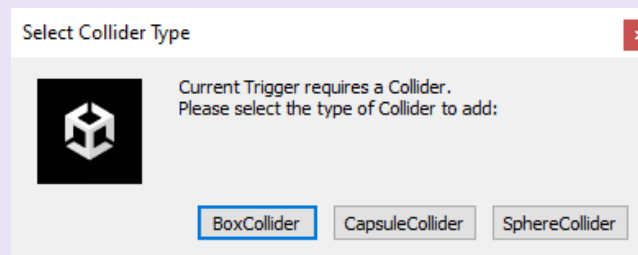
Start by duplicating the "NewInteraction" GameObject. Create a new **3D Object**, this time a **Sphere**, and assign the **Sphere** to each interaction in the cloned "NewInteraction." Make sure to add a **Rigidbody** to the **Sphere** so it can be affected by the [AddForceInteraction](#). Position the **Sphere** away from the **Cube**, but keep it within the viewport to easily observe the results.

In this case, the interaction chain should not start immediately, as it did before. Instead, it should activate when the **Cube** reaches a specific area.

To achieve this, we should replace the Trigger with a [StepOnTrigger](#) by hitting the [Replace Button](#).



A [StepOnTrigger](#) will react to an area being entered as the name indicates. Once you replace it you'll be asked to add a **Collider** to the object.



For now pick a **BoxCollider**.

Now you should have a [StepOnTrigger](#) and a **BoxCollider** added in the new GameObject. You'll note that the **IsTrigger** option of the **BoxCollider** is turned on and if you turn it off and press play it will turn on again, that's because a [StepOnTrigger](#) forces this option on the attached collider.

You can enlarge that collider and place the interaction in front of the **Cube**'s path. We want the **Cube** to activate the **Sphere** upon contact..

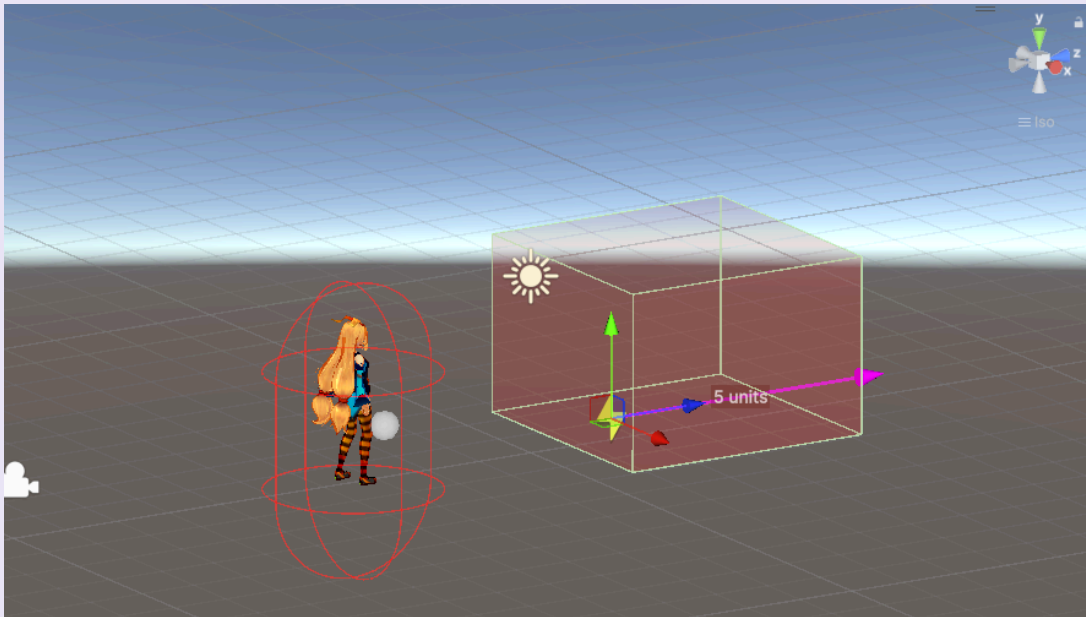
To achieve that we'll need to add an Interactor to the **Cube**, via **AddComponent -> QTI -> [Interactor](#)**. This allows the object to be recognized by [PhysicsTriggers](#) (eg. [StepOnTrigger](#)).

Now, **when the game runs**, the **Cube** will collide with the cloned "NewInteraction," triggering the **Sphere**'s activation and movement!

5. Triggering interactions with the player

Now we can disable the original "NewInteraction" and just interact with the clone using our player prefab. The player prefab is called **QTI_ThirdPersonDemo_Player**, grab it from the project folder and add it to the scene, also turn off the **UseGravity** option from the **Rigidbody**.

Now the scene should look akin to this:



The player prefab comes with its own player controller, which already interacts with triggers by itself. Since it implements [IInteractor](#), there won't be a need of adding an Interactor like we did for the **Cube**.

Now you can already hit play and move the player with WASD or whatever key configuration you're using and run into the interaction multiple times to keep triggering it.

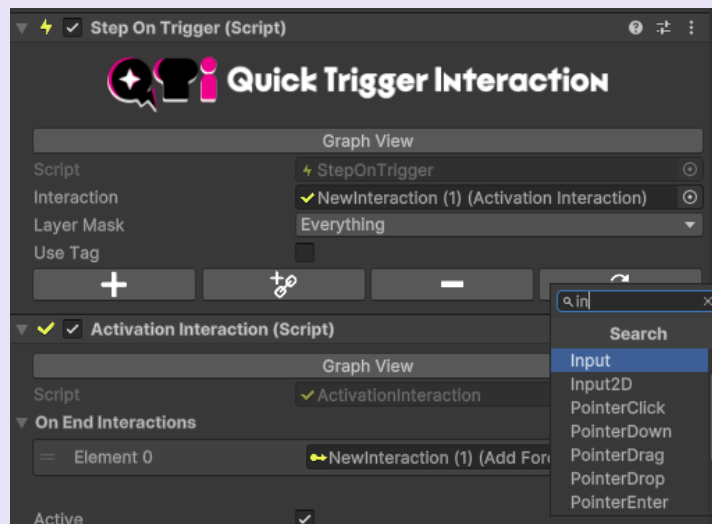
You can **turn on Gizmos** in the Game window to **better visualize the Trigger**!

If you play now you'll see that running into the interaction triggers the **Sphere's** Interaction chain!

6. Understanding the Input Trigger

Now we're gonna want to press a key to trigger the Interactions, instead of just stepping on it, since that's how lots of interactions are started in games.

For that we should replace the Trigger in the "NewInteraction" into an [InputTrigger](#).



Beware that the [InputTrigger](#) comes with a predefined **Angle** value, this limits the directions from which the player can interact with it. For now you can **change it to 360** so the player can interact with it from anywhere.

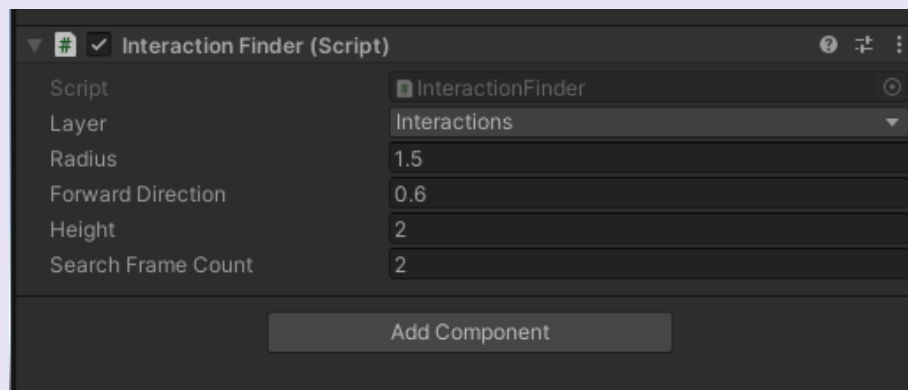
Now we need to come up with a way for the player to detect the nearby [InputTriggers](#). Fortunately, the **Player Prefab** already includes an **Interaction Finder**, which implements [IInputInteractor](#). The player controller manages the **Interact** method when the 'Jump' input is used (we use the **Spacebar**).

For your game you might use this **Interaction Finder** or try making your own. You can take a look into the [First Person Puzzle Demo Scene](#) for an example of a First person [IInputInteractor](#) implementation.

You can also take a look into the [Interactors](#) section for more information on how to implement your own.



The **capsule** around your player character **turns green** when there's an **Interaction nearby**. You can adjust the **radius** of the capsule and other relevant values in the **Interaction Finder** inspector:



Now, **hit play**, run up to the "NewInteraction," and **press the Spacebar to interact** with it. You'll see the **Cube enlarge and disappear** each time!

That wraps up the initial steps with the tool! Next, you'll discover a variety of **Trigger and Interaction types included**, along with **script documentation** to help you **create your own** and meet your game's needs!

Triggers

Triggers are **MonoBehaviours** that activate an **Interaction**. There are **several built-in types**, including those based on **Unity's lifecycle and physics events**. Because **Triggers** and **Interactions** are separate, any **Trigger** can call any **Interaction**, allowing for a wide range of ways to connect them.

For example, a **DestroyInteraction** can be used to remove an item either when a collision occurs or based on player input, using either a **CollisionEnter** or an **InputTrigger**.

The same **Interaction** behaviors can be applied in various situations without the need for additional coding. Even if a new **Trigger** is required, it will automatically function with all existing **Interactions** in the game, and vice versa. This allows for less time spent on edge cases and more reuse of the behaviors already developed.

A **list of all included Triggers** is provided below. For **instructions on implementing Custom Triggers**, refer to the [guide](#) in the relevant section of this documentation.

Lifecycle	
OnAwakeTrigger	OnStartTrigger
OnEnableTrigger	OnUpdateTrigger
OnDisableTrigger	OnLateUpdateTrigger
OnDestroyTrigger	OnFixedUpdateTrigger

Physics	
CollisionEnterTrigger	StepOn2DTrigger
CollisionExitTrigger	StepOff2DTrigger
CollisionStayTrigger	StayOn2DTrigger

Physics2D	
CollisionEnter2DTrigger	StepOn2DTrigger
CollisionExit2DTrigger	StepOff2DTrigger
CollisionStay2DTrigger	StayOn2DTrigger

Mouse	
MouseDownTrigger	PointerDownTrigger
MouseDownDragTrigger	PointerUpTrigger
MouseUpTrigger	PointerClickTrigger
MouseUpAsButtonTrigger	PointerEnterTrigger
MouseEnterTrigger	PointerExitTrigger
MouseExitTrigger	PointerMoveTrigger
MouseOverTrigger	PointerDragTrigger
	PointerDropTrigger

Input	
InputTrigger	Input2DTrigger
KeyTrigger	

Other	
OnDistanceTrigger	OnDistance2DTrigger

1. Unity Lifecycle Triggers

This type of **Trigger** operates using Unity's Lifecycle messages. For further information, please refer to the official Unity [documentation](#).

1.1. **OnAwakeTrigger**

Triggers its interaction on Unity's [Awake](#) Lifecycle message.

Interaction Interaction	The Interaction component to be triggered.
-----------------------------------	--

1.2. **OnEnableTrigger**

Triggers its interaction on Unity's [OnEnable](#) Lifecycle message.

Interaction Interaction	The Interaction component to be triggered.
-----------------------------------	--

1.3. **OnStartTrigger**

Triggers its interaction on Unity's [Start](#) Lifecycle message.

Interaction Interaction	The Interaction component to be triggered.
-----------------------------------	--

1.4. **OnUpdateTrigger**

Triggers its interaction on Unity's [Update](#) Lifecycle message.

Interaction Interaction	The Interaction component to be triggered.
-----------------------------------	--

1.5. **OnFixedUpdateTrigger**

Triggers its interaction on Unity's [FixedUpdate](#) Lifecycle message.

Interaction Interaction	The Interaction component to be triggered.
-----------------------------------	--

1.6. OnLateUpdateTrigger


Triggers its interaction on Unity's [LateUpdate](#) Lifecycle message.

Interaction Interaction	The Interaction component to be triggered.
-----------------------------------	--

1.7. OnDisableTrigger

Triggers its interaction on Unity's [OnDisable](#) Lifecycle message.

Interaction Interaction	The Interaction component to be triggered.
-----------------------------------	--

 **Warning:** As this can trigger when the *Trigger's* GameObject is Destroyed, interactions that run Coroutines (eg: DelayInteraction) won't be able to run.

1.8. OnDestroyTrigger

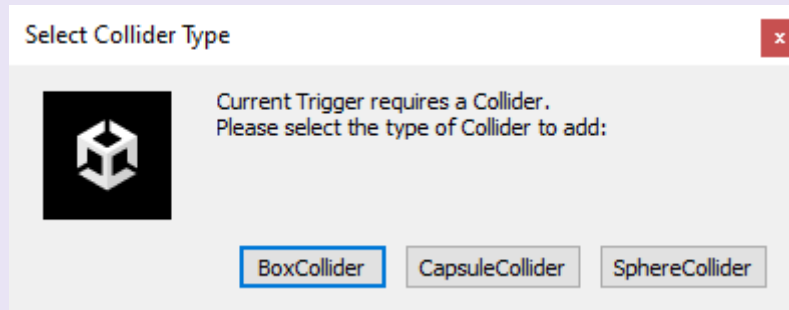
Triggers its interaction on Unity's [OnDestroy](#) Lifecycle message.

Interaction Interaction	The Interaction component to be triggered.
-----------------------------------	--

 **Warning:** This may trigger when the Trigger's GameObject is destroyed, which means that interactions running Coroutines (e.g., DelayInteraction) will not be able to execute.

2. Physics Triggers

All Physics Triggers are based on Unity's Physics System messages and therefore will require a **Collider**. If you do not have a collider attached to the GameObject a prompt will appear for you to select one of the basic Collider types.



If you want to use another type of collider, for example a **Mesh Collider**, please attach the Trigger component after having a Collider already present in the GameObject.

Physics Triggers also have the following base properties:

Interaction Interaction	The Interaction component to be triggered.	
Layer Mask LayerMask	The Physics Layer Mask to use as a filter in collisions, allowing control over which layers can interact with this Trigger .	
Use Tag bool	Filters the collisions by a tag if enabled.	
	Select Tag string	<p>The tag of the GameObjects to check, used as a filter to determine which objects can interact with the Trigger. This filtering mechanism operates in conjunction with the Layer Mask option.</p> <p>For instance, if only the player has the “Player” tag, this will identify and utilize only the player for calculations.</p>

2.1. Unity 3D Physics Triggers

These Triggers only function with Unity's Physics System messages for 3D collisions.

2.1.1. CollisionEnterTrigger

Triggers its interaction on Unity's Physics [OnCollisionEnter](#) message.

NOTE: This component will automatically set the required Collider's **isTrigger** property to **FALSE**.

2.1.2. CollisionExitTrigger


Triggers its interaction on Unity's Physics [OnCollisionExit](#) message.

NOTE: This component will automatically set the required Collider's **isTrigger** property to **FALSE**.

2.1.3. CollisionStayTrigger

Triggers its interaction on Unity Physics [OnCollisionStay](#) message with the following properties:

Has Cooldown bool	Determines if the <i>Interaction</i> triggers every time or waits every Cooldown Timer Seconds to trigger.
Cooldown Timer float	The time in Seconds to wait every time before triggering the <i>Interaction</i>

 **NOTE:** This component will automatically set the required Collider's **isTrigger** property to **FALSE**.

2.1.4. StepOnTrigger

Triggers its interaction on Unity Physics [OnTriggerEnter](#) message.

NOTE: This component will automatically set the required Collider's **isTrigger** property to **TRUE**.

2.1.5. StepOffTrigger


Triggers its interaction on Unity Physics [OnTriggerExit](#) message.

NOTE: This component will automatically set the required Collider's **isTrigger** property to **TRUE**.

2.1.6. StayOnTrigger

Triggers its interaction on Unity Physics [OnTriggerStay](#) message with the following properties:

Has Cooldown bool	Determines if the <i>Interaction</i> triggers every time or waits every Cooldown Timer Seconds to trigger.
Cooldown Timer float	The time in Seconds to wait every time before triggering the <i>Interaction</i>

 **NOTE:** This component will automatically set the required Collider's **isTrigger** property to **TRUE**.

2.2. Unity 2D Physics Triggers

These Triggers only function with Unity's Physics System messages for 2D collisions.

2.2.1. CollisionEnter2DTrigger

Triggers its interaction on Unity Physics [OnCollisionEnter2D](#) message.

NOTE: This component will automatically set the required Collider's **isTrigger** property to **FALSE**.

2.2.2. CollisionExit2DTrigger


Triggers its interaction on Unity Physics [OnCollisionExit2D](#) message.

NOTE: This component will automatically set the required Collider's **isTrigger** property to **FALSE**.

2.2.3. CollisionStay2DTrigger

Triggers its interaction on Unity Physics [OnCollisionStay2D](#) message with the following properties:

Has Cooldown bool	Determines if the <i>Interaction</i> triggers every time or waits every Cooldown Timer Seconds to trigger.
Cooldown Timer float	The time in Seconds to wait every time before triggering the <i>Interaction</i>

 **NOTE:** This component will automatically set the required Collider's **isTrigger** property to **FALSE**.

2.2.4. StepOn2DTrigger

Triggers its interaction on Unity Physics [OnTriggerEnter2D](#) message.

NOTE: This component will automatically set the required Collider's **isTrigger** property to **TRUE**.

2.2.5. StepOff2DTrigger


Triggers its interaction on Unity Physics [OnTriggerExit2D](#) message.

NOTE: This component will automatically set the required Collider's **isTrigger** property to **TRUE**.

2.2.6. StayOn2DTrigger

Triggers its interaction on Unity Physics [OnTriggerStay2D](#) message with the following properties:

Has Cooldown bool	Determines if the <i>Interaction</i> triggers every time or waits every Cooldown Timer Seconds to trigger.
Cooldown Timer float	The time in Seconds to wait every time before triggering the <i>Interaction</i>

 **NOTE:** This component will automatically set the required Collider's **isTrigger** property to **TRUE**.

3. Mouse Triggers

These Triggers function with Unity's Mouse and Pointer messages.

3.1. **MouseDownTrigger**

Triggers its interaction on Unity's [OnMouseDown](#) message.

Use for gameObject's containing a collider or collider2D.

For UI mouse events use PointerDownTrigger.

Interaction Interaction	The Interaction component to be triggered.
-----------------------------------	--

3.2. **MouseDownTrigger**

Triggers its interaction on Unity's [OnMouseDown](#) message.

Use for gameObject's containing a collider or collider2D.

Interaction Interaction	The Interaction component to be triggered.
-----------------------------------	--

3.3. **MouseUpTrigger**

Triggers its interaction on Unity's [OnMouseUp](#) message.

Use for gameObject's containing a collider or collider2D.

For UI mouse events use PointerUpTrigger.

Interaction Interaction	The Interaction component to be triggered.
-----------------------------------	--

3.4. **MouseUpAsButtonTrigger**

Triggers its interaction on Unity's [OnMouseUpAsButton](#) message.

Use for gameObject's containing a collider or collider2D.

For UI mouse events use PointerUpTrigger.

Interaction	The Interaction component to be triggered.
--------------------	--

Interaction	
-------------	--

3.5. **MouseEnterTrigger**

Triggers its interaction on Unity's [OnMouseEnter](#) message.
 Use for gameObject's containing a collider or collider2D.
 For UI mouse events use [PointerEnterTrigger](#).

Interaction Interaction	The Interaction component to be triggered.
-----------------------------------	--

3.6. **MouseExitTrigger**

Triggers its interaction on Unity's [OnMouseExit](#) message.
 Use for gameObject's containing a collider or collider2D.
 For UI mouse events use [PointerExitTrigger](#).

Interaction Interaction	The Interaction component to be triggered.
-----------------------------------	--

3.7. **MouseOverTrigger**

Triggers its interaction on Unity's [OnMouseOver](#) message.
 Use for gameObject's containing a collider or collider2D.

Interaction Interaction	The Interaction component to be triggered.
-----------------------------------	--

3.8. **PointerDownTrigger**

Triggers its interaction on Unity's [OnPointerDown](#) message.
 Use for UI object's that are raycast targets.
 For non-UI mouse events use [MouseDownTrigger](#).

Interaction Interaction	The Interaction component to be triggered.
-----------------------------------	--

3.9. PointerUpTrigger

Triggers its interaction on Unity's [OnPointerUp](#) message.

Use for UI object's that are raycast targets.

For non-UI mouse events use [MouseUpTrigger](#).

Interaction Interaction	The Interaction component to be triggered.
-----------------------------------	--

3.10. PointerClickTrigger

Triggers its interaction on Unity's [OnPointerClick](#) message.

Use for UI object's that are raycast targets.

For non-UI mouse events use [MouseDownTrigger](#) or [MouseUpTrigger](#).

Interaction Interaction	The Interaction component to be triggered.
-----------------------------------	--

3.11. PointerEnterTrigger

Triggers its interaction on Unity's [OnPointerEnter](#) message.

Use for UI object's that are raycast targets.

For non-UI mouse events use [MouseEnterTrigger](#).

Interaction Interaction	The Interaction component to be triggered.
-----------------------------------	--

3.12. PointerExitTrigger

Triggers its interaction on Unity's [OnPointerExit](#) message.

Use for UI object's that are raycast targets.
For non-UI mouse events use [MouseExitTrigger](#).

Interaction Interaction	The Interaction component to be triggered.
-----------------------------------	--

3.13. PointerMoveTrigger

Triggers its interaction on Unity's [OnPointerMove](#) message.
Use for UI object's that are raycast targets.

Interaction Interaction	The Interaction component to be triggered.
-----------------------------------	--

3.14. PointerDragTrigger

Triggers its interaction on Unity's [OnBeginDrag](#), [OnDrag](#) or [OnEndDrag](#) message, depending on the value of the **TriggerOn** enum.
Use for UI object's that are raycast targets.

Interaction Interaction	The Interaction component to be triggered.
TriggerOn enum	Determines if the interaction is called OnBeginDrag , OnDrag or OnEndDrag .

3.15. PointerDropTrigger

Triggers its interaction on Unity's [OnDrop](#) message.
Use for UI object's that are raycast targets.

Interaction Interaction	The Interaction component to be triggered.
-----------------------------------	--

4. Other

4.1. OnDistanceTrigger

Triggers its interaction based on the Unity unit distance to itself, Useful when you want to save resources and not use physics for a distance/area check.

Only works for specific objects tagged with a specific tag unlike the physics which take Layers into consideration.

For a 2D project use [OnDistance2DTrigger](#) instead.

Select Tag string	The Tag of the GameObjects to check, for example, if only the player has the “Player” Tag then it will find and use only the player for its calculations.
Distance float	The distance radius in Unity units to calculate
Mode enum	When to fire the Trigger, options can be: OnEnter, OnExit, OnStay. Just like a normal Unity trigger/collision would work.
➡ OnEnter	Fires when the GameObject enters the distance radius.
➡ OnStay	Fires when the GameObject stays at the distance radius.
➡ OnExit	Fires when the GameObject exits the distance radius.

4.2. OnDistance2DTrigger

Triggers its interaction based on the Unity unit distance to itself, Useful when you want to save resources and not use physics for a distance/area check.

Only works for specific objects tagged with a specific tag unlike the physics which take Layers into consideration.

For a 3D project use [OnDistanceTrigger](#) instead.

Select Tag string	The Tag of the GameObjects to check, for example, if only the player has the “Player” Tag then it will find and use only the player for its calculations.
Distance float	The distance in Unity units to calculate
Mode enum	When to fire the Trigger, options can be: OnEnter, OnExit, OnStay.

	Just like a normal Unity trigger/collision would work.
➡ OnEnter	Fires when the GameObject enters the distance radius.
➡ OnStay	Fires when the GameObject stays at the distance radius.
➡ OnExit	Fires when the GameObject exits the distance radius.

4.3. InputTrigger

The Input Trigger is a special use case that doesn't actually trigger an **Interaction** on its own, but instead provides relevant information for other components to interact with it and fire the trigger themselves.

It is specially tailored to be used by a [Player Interactor](#) and provides the following parameters:

Interaction Interaction	The Interaction component to be triggered.	
Priority enum	Importance of this Trigger. Useful for filtering and creating hierarchy between Triggers/Interactions that are close together. Can be adjusted in the PrioritiesEnum list.	
Fixed Angle bool	If the Trigger is interactable only from a specific Direction/Angle	
	Direction float	Direction in degrees that the Player should be in to interact with this Trigger. Only appears if Fixed Angle is TRUE .
Angle float	Angle range that the Player can interact with this Trigger, the more narrow the more aligned with the object the player needs to be and vice versa.	
Interaction Visual InteractionVisual	An optional visual component for this Trigger/Interaction. This Trigger automatically manages certain calls to the visual component and supports additional functions triggered by an Interactor. For more detailed information, refer to the section on Interaction Visuals .	

NOTE: This Trigger Requires a **Collider** Component much like the [Physics Triggers](#), it needs the collider to be recognized by an interactor. You can read more and see an example of our supplied **Player Interactor** [here](#).

4.4. Input2DTrigger

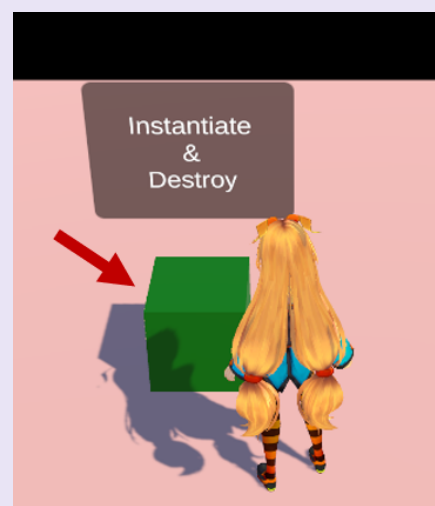
The 2D Variant of the normal [InputTrigger](#), Has the same properties but requires a 2D Collider instead.

4.4.1. Interaction Visual

An optional visual component for the [InputTrigger](#). Calls **Animator** parameters to different trigger states, when assigned to.

It implements three states:

Highlight Bool string	Animator boolean parameter for the Highlight State. Executes when the InputTrigger can be interacted (in-range).
Disable Bool string	Animator boolean parameter for the Disable State Executes when the InputTrigger is disabled.
Interact Trigger string	Animator trigger parameter for the Interact state. Executes when the InputTrigger is interacted with.
Animator Animator	The Animator to affect.



 **NOTE:** This component requires an **Animator** to work.

4.5. KeyTrigger

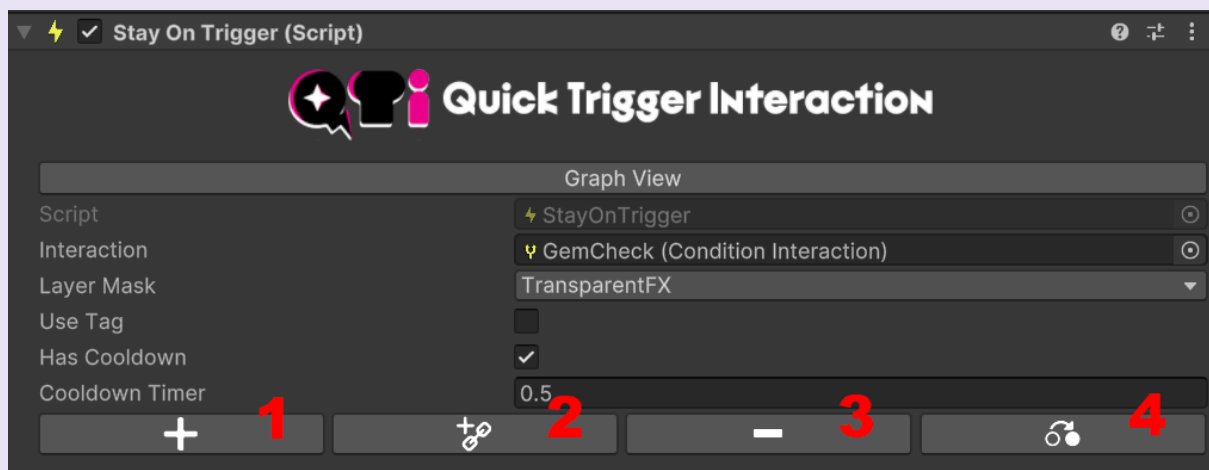
This trigger activates upon a specified **key press** and **does not require an [Interactor](#)**. It operates using [Unity's Old Input System](#).

Input Type enum		
➤ Key	Sets the input to use a KeyCode .	
	KeyCode enum	The chosen Keycode . (e.g. Tab)
➤ Axis	Set the input to use an Axis .	
	Button enum	The chosen Axis . (e.g. Horizontal)
Press Type enum		
➤ Press	Uses Input.GetKeyDown() if set to Key or Input.GetButtonDown() if set to Axis .	
➤ Hold	Uses Input.GetKey() if set to Key or Input.GetButton() if set to Axis .	
➤ Release	Uses Input.GetKeyUp() if set to Key or Input.GetButtonUp() if set to Axis .	

5. Inspector Shortcuts

The buttons under each trigger:

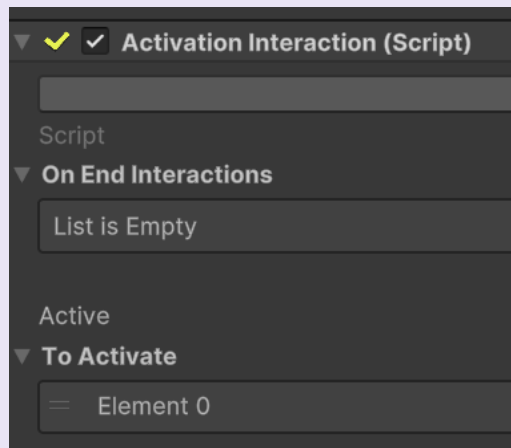
- 1) **Adds an interaction** to this game object. Sets itself as this trigger's interaction if the interaction field is empty.
- 2) **Adds an interaction in between** this trigger and the interaction that's currently assigned, adding that one to the new interaction's onEndInteractions list.
- 3) **Removes the interaction** assigned to this trigger. If no interaction is assigned, an existing interaction from this game object will be removed.
- 4) **Replaces this trigger** with a different one, keeping its interaction field and any other relevant fields they may have in common.



Interactions

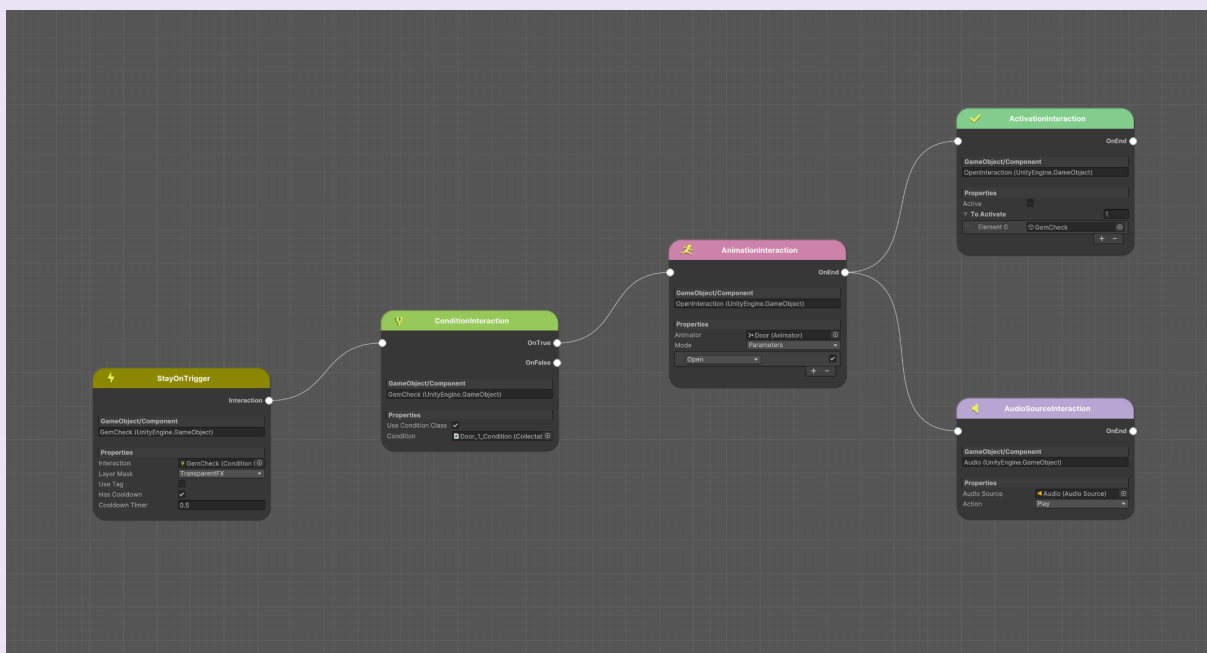
Interactions are components that do specific actions which are triggered by *Triggers*. When triggered, the *Interaction* receives information about the [Interactor](#) sent by the *Trigger*.

It can then use that information if needed along with its own parameters to perform its task. Interactions are **Chainable** so once its code is done, it will check to see if any **On End Interactions** are present and call them.



By using this powerful chaining method, you can combine **simple** Interactions to perform **complex** processes without writing more code.

It can get overwhelming to navigate Unity's Inspector View when you have a big enough chain going so we provide a visualization method you can access by hitting the "[Graph View](#)" button on any Interaction.



By combining the built in *Interactions* with your own [Custom Interactions](#), you can reuse code more effectively as by combining Interactions with different triggers you can call the same code in different ways **AND** you can combine and chain Interactions to make new more complex behavior without the need to write new code.

You can view a List of the already built in interactions below, or you can check out a guide on how to create your own custom interactions for your game.

1. Built-in Interactions

1.1. ActivationInteraction

Activates the selected GameObjects

To Activate List<GameObject>	The list of GameObjects to activate.
--	--------------------------------------

1.2. AnimationInteraction

Performs actions on a supplied Animator

Animator Animator	The Animator to affect.	
Mode enum		
➡ Play	Calls the Play method for a specific clip	
	Animation	The animation to play.
	Wait for animation to End	Will only call OnEnd Interactions after animation ends.
➡ Parameters	The parameters to send to the animator.	

1.3. AudioPlayOneShotInteraction

Plays an Audio Clip using the AudioSource.PlayClipAtPoint() method

Audio Clip AudioClip	The Audio clip to play	
Volume float	The volume at which to play the Audio Clip	
Mode enum		
➤ Audio Source	Uses an audio source to play the clip.	
	Audio Source AudioSource	Audio Source to use to play the clip, if no Audio Source is present, one will be created at the interaction position.
➤ Position2D	Creates a one time use Audio Source to play the audio clip at AudioListener position.	
➤ Position3D	Creates a one time use Audio Source to play the audio clip at clip position.	
	Transform Transform	Fetches a Transform world space position to play the clip, only used if no Audio Source is provided, otherwise will use Audio Source Position

1.4. AudioSourceInteraction

Performs actions on an Audio Source

Audio Source AudioSource	AudioSource to affect.
Action enum	Action to be performed: Play, Pause, Resume, Stop.
➤ Play	Play the AudioSource.
➤ Pause	Pauses the AudioSource.

➡ Resume	Resume the AudioSource.
➡ Stop	Stop the AudioSource.

1.5. DelayInteraction

Waits for the specified amount of time then will call other Interactions.

Wait Time float	The amount of time in Seconds to wait
Unscaled Time bool	If the time should be unscaled, useful for when the time scale has been set to zero.

1.6. DestroyComponentInteraction

Destroys a list of Components

To Destroy List<Component>	The list of Components to destroy.
--------------------------------------	------------------------------------

1.7. DestroyInteraction

Destroys a list of GameObjects. Can also destroy the interactor.

To Destroy List<GameObject>	The list of GameObjects to destroy.
AlsoDestroyInteractor bool	If on, will also destroy the interactor that interacts with the interaction.

1.8. InstantiateInteraction

Instantiates a collection of GameObjects.

To Instantiate List<SpawnOptions>	The list of GameObjects and their configurations to Spawn.
---	--

You can add different settings per object to spawn:

Prefab Prefab (GameObject)	The GameObject to spawn.	
Transform Mode enum	Specifies the Spawn Position.	
➡ Original	Uses Prefab's Transform values .	
➡ Transform	Uses a supplied Transform , can add offsets for Position, Rotation and Scale .	
	Transform Transform	Supplies a Transform to fetch position, rotation and scale
	Position Offset Vector3	Position offset to be added to the Transform position
	Rotation Offset Vector3	Rotation offset to be added to the Transform rotation
	Scale Offset Vector3	Scale offset to multiply by the Transform scale
➡ Manual	Uses supplied Position, Rotation and Scale values.	
	Position Vector3	Position in world space.
	Rotation Vector3	Rotation in world space.
	Scale Vector3	Scale in world space.
Parent Mode enum	Specifies if the Object will spawn at the Root of the Scene or at a supplied Object Parent .	
➡ Root	Spawns at the Scene Root.	
➡ Parent	Will spawn as a child of a supplied parent. The parent doesn't affect the spawn position.	
	Transform Transform	The parent Transform

1.9. UnityEventInteraction

Calls the specified list of Unity Events, can be used to call any function of any component you want, including unity components.

Unity Event UnityEvent	The list of Events to call.
----------------------------------	-----------------------------

Use examples can be found in the Demos.

⚠ Warning: This is the most powerful Interaction as it can call almost any code, however it is also the most generic, therefore, illegible. If you want your code to be readable and organized, use this Interaction with caution.

1.10. ConditionInteraction

This is a very powerful interaction that allows two different outputs base on the value of a flag.

You can also setup your own custom Condition class to be evaluated to return different output automatically.

UseConditionClass bool	Determines if the ConditionInteraction should evaluate the Condition or use the builtIn IsTrue flag instead. Default is false.
Condition Condition	The Condition to evaluate in case UseConditionClass is on. When the condition is interacted with, runs OnTrueInteractions if true, and OnFalseInteractions if false.
IsTrue bool	The flag to evaluate in case useConditionClass is off. When the condition is interacted with, runs onTrueInteractions if true, and onFalseInteractions if false.
OnTrueInteractions List<Interaction>	The list of interactions to run if the supplied flag is true.
OnFalseInteractions List<Interaction>	The list of interactions to run if the supplied flag is false.

In the Graph View the ConditionInteraction will show the interactions it runs if the supplied flag is true or false.

1.11. AddForceInteraction

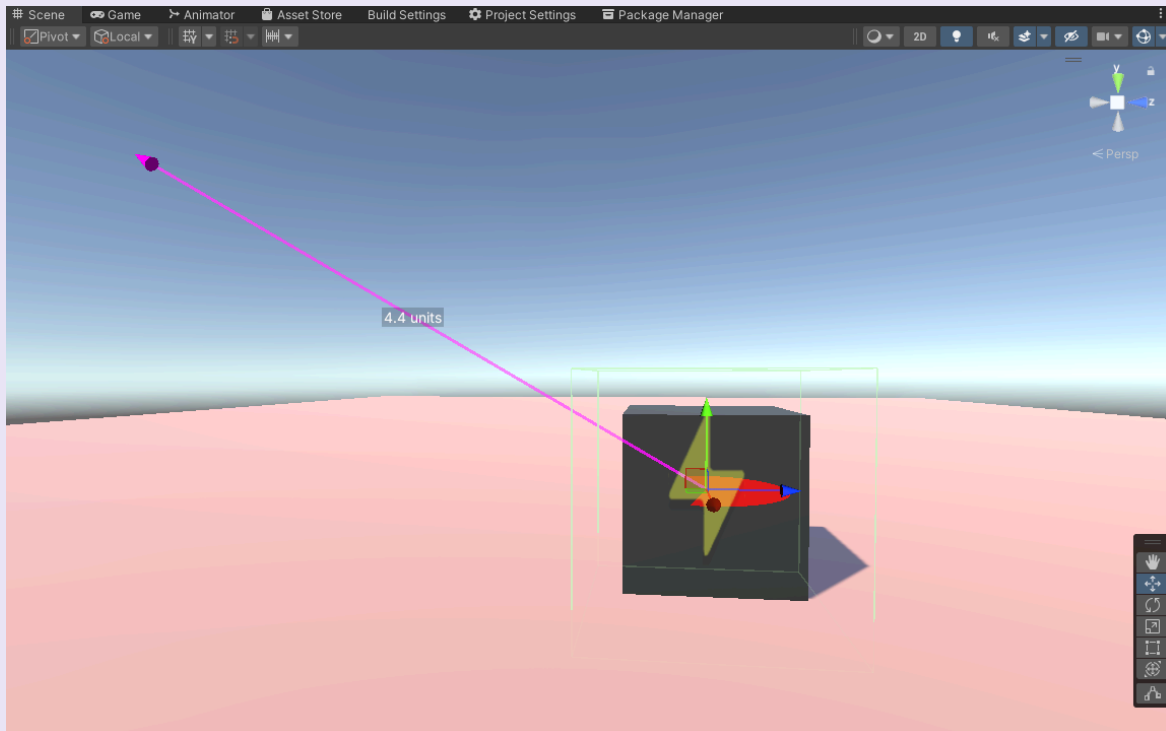
Adds a force to a rigidbody. You can select multiple force modes and use it for 2D and 3D games.

You can attach it to a FixedUpdate trigger for example, so make an object be pushed by a constant force every fixed timestep.

Be sure to turn on Gizmos in the Scene View so you can see the **Rotation/Magnitude** of the force properly!

Mode enum	2D for 2D games, 3D for 3D games.	
➡ 3D	Sets Interaction properties to 3D mode. (XYZ space)	
	ForceType enum	Can be World , Local or Relative .
	➡ World	Orientation and Magnitude are applied in World Space.
	➡ Local	Orientation and Magnitude are applied in Local Space.
	➡ Relative	The force direction is set from the GameObject position to the Rigidbody position.
	Body Rigidbody	The Rigidbody component to apply the force to.
	Orientation Vector3	The rotation of the force vector.
	Magnitude float	The magnitude of the force vector.
	ForceMode enum	Select the ForceMode you wish to use on your force. Uses Unity's ForceMode enum.
➡ 2D	Sets Interaction properties to 2D mode. (XY plane)	
	Body2D Rigidbody2D	The Rigidbody2D component to apply the force to.
	Angle float	The angle of the force vector.
	Magnitude float	The magnitude of the force vector.
	ForceMode2D enum	Select the ForceMode2D you wish to use on your force. Uses Unity's ForceMode2D enum.

With gizmos on you can easily verify the orientation and magnitude of the force.



1.12. SetPositionInteraction

Sets a new position for the assigned **GameObject**.

TargetObject Transform	The GameObject Transform to set the new position to.	
Mode enum	Determines if the targetObject position is set to the newPosition vector or to the position of the transform assigned to newPositionTransform .	
➤ Transform	The TargetObject position is set to a Transform position	
	NewPositionTransform Transform	The Transform to be used as the new position.
➤ Position	The TargetObject position is set to a supplied Vector3	
	NewPosition Vector3	The Vector3 to be used as the new position.

1.13. SetRotationInteraction

Sets a new rotation for the assigned **GameObject**.

TargetObject Transform	The GameObject Transform to set the new rotation to.	
Mode enum	Determines if the TargetObject rotation is set to a new Vector3 rotation or rotates by a given angle around a given axis.	
➔ Euler	The TargetObject rotation is set to the given Vector3 euler angles.	
	NewRotation Vector3	The Vector3 to be set as the new rotation's euler angles.
➔ AngleAxis	The TargetObject is rotated by a given angle around a given axis.	
	Angle float	The angle value to rotate by.
	Axis enum	The axis to rotate around, can be Up , Right or Front .

1.14. SetScaleInteraction

Sets a new position for the assigned **GameObject**.

TargetObject Transform	The GameObject Transform to set the new localScale to.
NewScale Vector3	The NewScale to be set as the new TargetObject 's localScale.

1.15. DialogueInteraction

Writes text on a desired TextMeshPro text. Handy for showing instructions and even dialogues in your game.

TextField TextMeshProUGUI	The field where to display the text .
Text string	The text to display on the textField .

1.16. DebugLogInteraction

This is a development focused Interaction to aid in testing out the Interaction system. It can be placed anywhere in the Interaction Chain to log a test message.

Debug String string	The message to display in the console.
-------------------------------	--

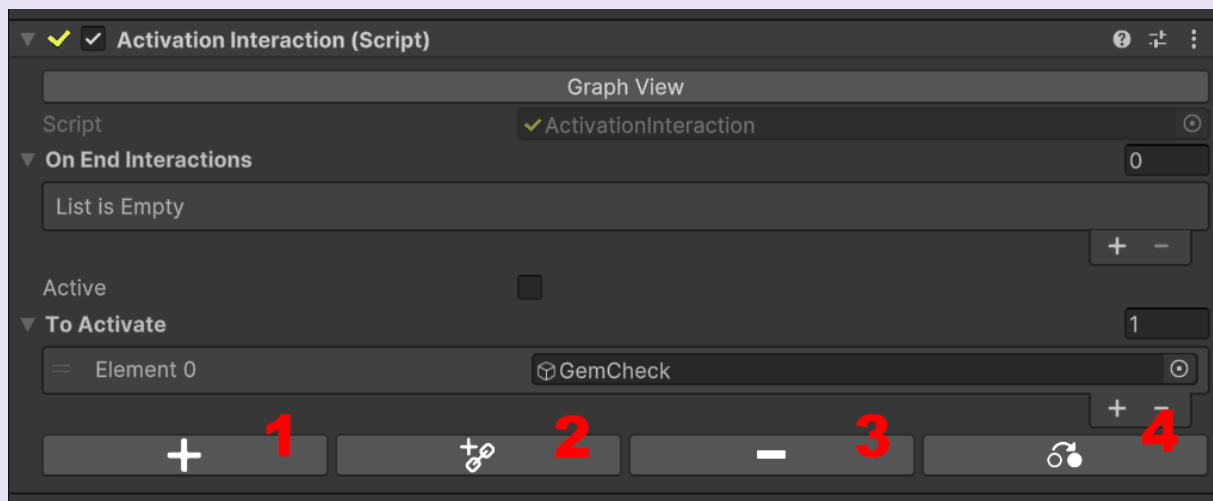
You can always customize your message by using [Rich Text Markup](#) to improve visibility, for example by using the **color** tag.

For more information visit Unity's official [documentation](#).

2. Inspector Shortcuts

The buttons under each interaction:

- 1) **Adds an interaction** that runs after the current one, adding it to the OnEndInteractions list.
- 2) **Adds an interaction in between** this one and the next ones in the chain, removing the OnEndInteractions list from the current one and assigning it to the new interaction.
- 3) **Removes one interaction** from the OnEndInteractions list.
- 4) **Replaces this interaction** with a different one, keeping its OnEndInteractions list, and reassigning every reference to the previous interaction to the new one.



Interactors

1. IInteractor

An interface that must be implemented by any object that interacts with **Triggers**.

2. Interactor

A **MonoBehaviour** class that implements **IInteractor**. This component can be added to any object requiring interaction with **Triggers** without further customization.

3. IInputInteractor

An interface that must be implemented by a component to identify the most suitable **InputTrigger** for interaction, tailored to the specific needs of the game when using these triggers.

How to Configure:

The **IInputInteractor** includes the following methods:

GetNearestInteraction(): Returns the **Interaction** closest to the player, indicating which one can be interacted with upon pressing the interact button. This method should be implemented accordingly.

TryInteract(): To be called when the player presses the interact button. This should be implemented to interact with the closest **Interaction**, obtained via **GetNearestInteraction()**.

You can use the **InteractionFinder** prefab which already implements this interface and call its methods directly for an easy to use solution.

For a 2D game scenario, there is an **Interaction2DFinder**, which works similarly.

4. IInput2DInteractor

A variation of **IInputInteractor** designed for use in a 2D game context, providing convenience for interactions specific to that environment.

Custom Triggers

Custom **Triggers** can be implemented by extending the **InteractionTrigger.cs** abstract class if the built-in options do not meet specific needs. All provided **Triggers** derive from this class or its subclasses.

This class includes several helpful methods, with the **Interact** method being the most significant. It automatically invokes the assigned **Interaction** when the custom logic for the **Trigger** is implemented.

Note: The **Interact()** method accepts one parameter, an **IInteractor**, which should be the **GameObject** (e.g., the Player) interacting with the **Trigger** for further processing by the **Interaction**. However, this argument can be **NULL**; for example, the **OnAwakeTrigger** does not involve an **IInteractor**, which is acceptable as long as it is not paired with **Interactions** that require this for additional processing. **Custom Triggers** can be developed with or without the **IInteractor** object.

Custom Editors

To create a **Custom Editor** for a custom **InteractionTrigger**, the editor class should inherit from the **InteractionTriggerEditor** class and override the **DrawProperties()** method to implement a custom inspector. Below is an example of such an implementation:

Custom InteractionTrigger class:

```
public class CustomInteractionTrigger: InteractionTrigger
{
    // Your custom code here.
    // To trigger the interaction chain run base.Interact().
}
```

Custom InteractionTriggerEditor class:

```
[CustomEditor(typeof(CustomInteractionTrigger))]
public class CustomInteractionEditor : InteractionEditor
{
    public override VisualElement CreateInspectorGUI()
    {
        // Your custom initialization code here

        return base.CreateInspectorGUI();
    }

    public override void DrawProperties()
    {
        // Your custom OnInspectorGUI() code here
    }
}
```

Custom Interactions

In addition to the provided **Interactions**, other examples of **Interaction** implementations can be found in the demo scenes. These examples demonstrate more advanced use cases and can assist in understanding how to create custom implementations tailored to specific needs.

Custom Editors

To create a **Custom Editor** for a custom **Interaction**, the editor class should inherit from the **InteractionEditor** class and override the **DrawProperties()** method to implement a custom inspector. Below is an example of such an implementation:

Custom Interaction class:

```
public class CustomInteraction: Interaction
{
    // Your custom code here.
    // Override Interact() method and run base.Interact()
}
```

Custom InteractionEditor class:

```
[CustomEditor(typeof(CustomInteraction))]
public class CustomInteractionEditor : InteractionEditor
{
    public override VisualElement CreateInspectorGUI()
    {
        // Your custom initialization code here

        return base.CreateInspectorGUI();
    }

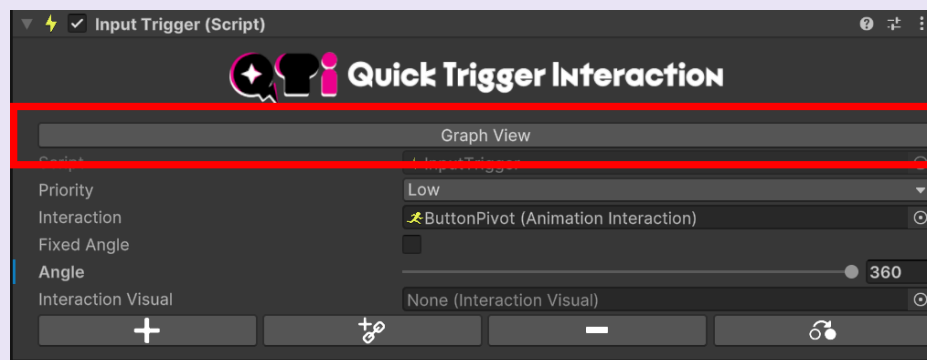
    public override void DrawProperties()
    {
        // Your custom OnInspectorGUI() code here
    }
}
```

Graph View

The **Graph View** is a custom Unity window designed to visualize and edit the properties of the Interaction and Trigger system. It provides a node-based interface where each **Interaction** and **Trigger** is represented as a node. These nodes are connected to illustrate the flow and structure of the interaction chain, allowing for an intuitive and organized way to manage complex behavior sequences.

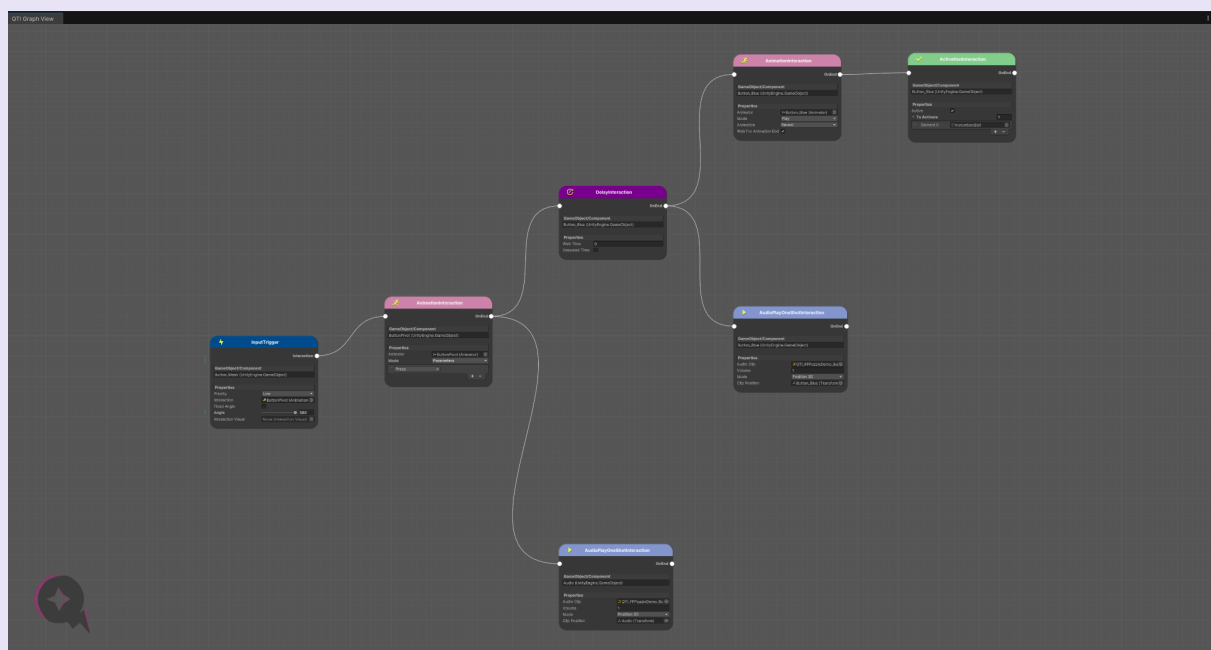
1. Opening Graph View

The Graph View **can be accessed by clicking** the **Graph View button** located on any **Interaction** or **Trigger** component.



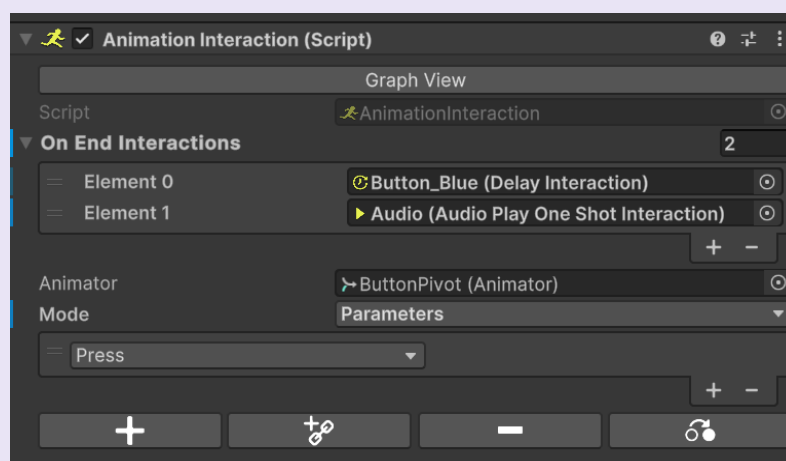
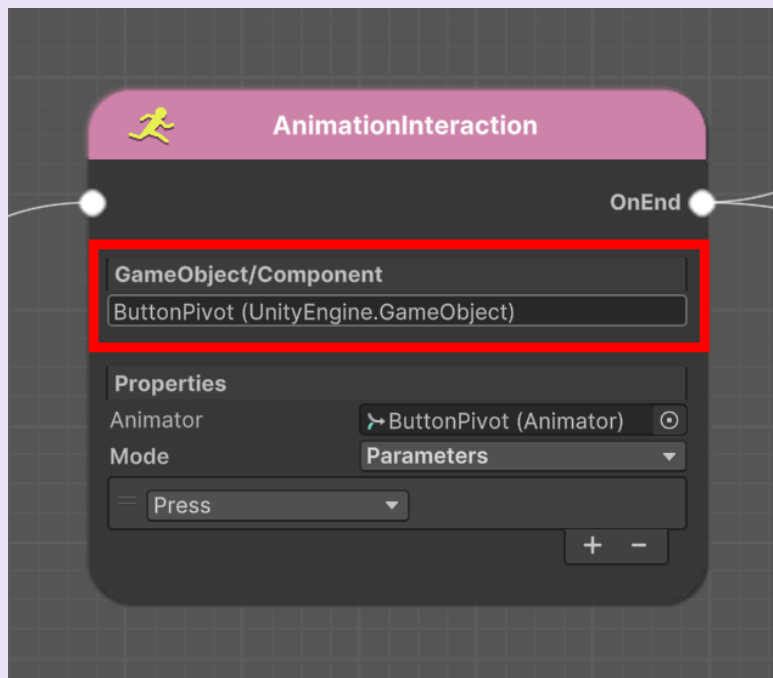
2. Window Overview

When the window is opened, the full tree structure containing the selected **Interaction** or **Trigger** is displayed. The view can be panned by holding the right mouse button and zoomed in/out using the scroll wheel, allowing closer examination of individual nodes.



3. Node Overview


Each node includes a header section with the component name, an **On End Interactions** field that links to other Interaction nodes **if the node is an Interaction**, or an **Interaction** field **if the node is a Trigger**. Additionally, there's a **GameObject reference field** displaying the associated GameObject's name. **Clicking this field highlights** the corresponding **component in the Inspector** and the **GameObject in Unity's hierarchy**. Below the header, a properties section **mirrors the component properties** from Unity's Inspector, **enabling direct editing**.



Settings

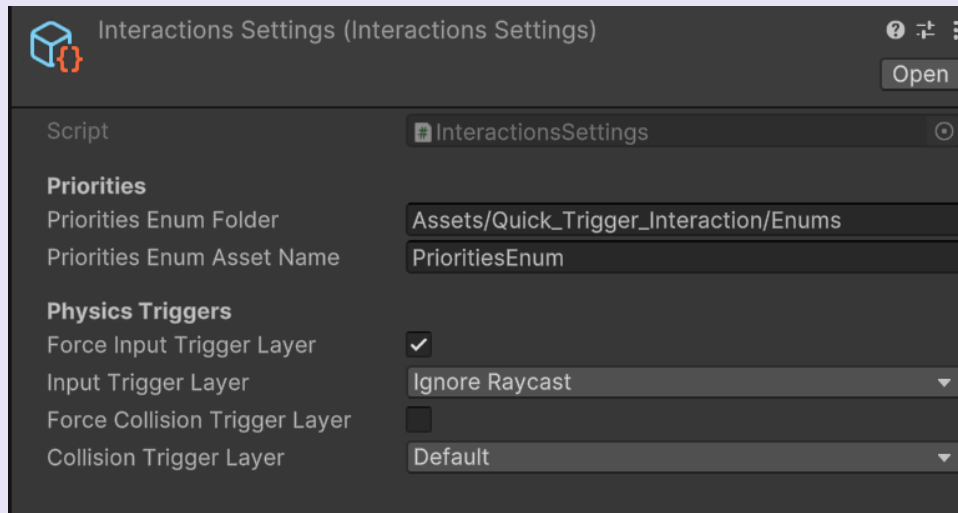
1. InteractionsSettings

The **InteractionSettings** asset is a **ScriptableObject** found at “Assets/Quick_Trigger_Interaction/Settings” that contains global settings for **Triggers** and **Interactions**.

 **Avoid changing the asset’s directory, as doing so will prevent it from functioning properly!**

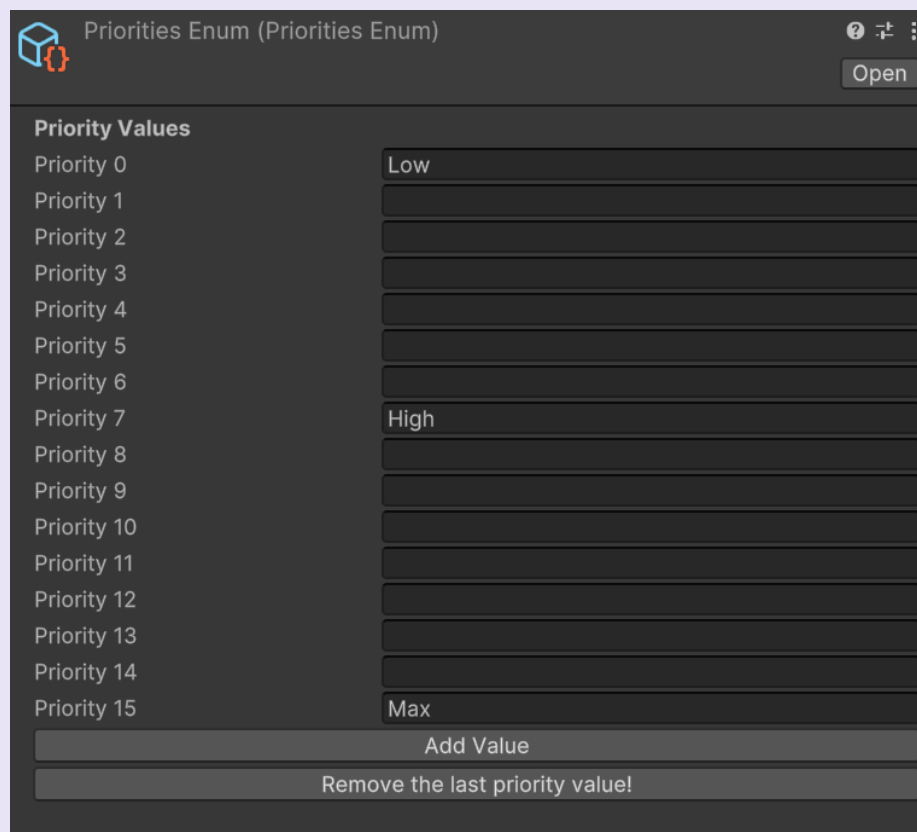
PrioritiesEnumFolder string	The asset path of the PrioritiesEnum ScriptableObject. Default is “Assets/Quick_Trigger_Interaction/Enums”
PrioritiesEnumAssetName string	The name of the PrioritiesEnum ScriptableObject. Default is “PrioritiesEnum”
ForceInputTriggerLayer bool	Forces a layer for all InputTriggers
InputTriggerLayer int	The layer index that is assigned to all InputTriggers if ForceInputTriggerLayer is enabled.
ForceCollisionTriggerLayer bool	Forces a layer for all PhysicsTriggers (excluding InputTriggers).
CollisionTriggerLayer int	The layer index that is assigned to all PhysicsTrigger (excluding InputTriggers) if ForceCollisionTriggerLayer is enabled.

Enforcing layers on objects is beneficial, particularly for **InputTriggers**, which must be on the same layer to enable detection by **InteractionFinders** and simplify the process. This practice remains optional to accommodate developers’ preferences. Similarly, while layers for **PhysicsTriggers** are not mandatory, the option to use this functionality is available.



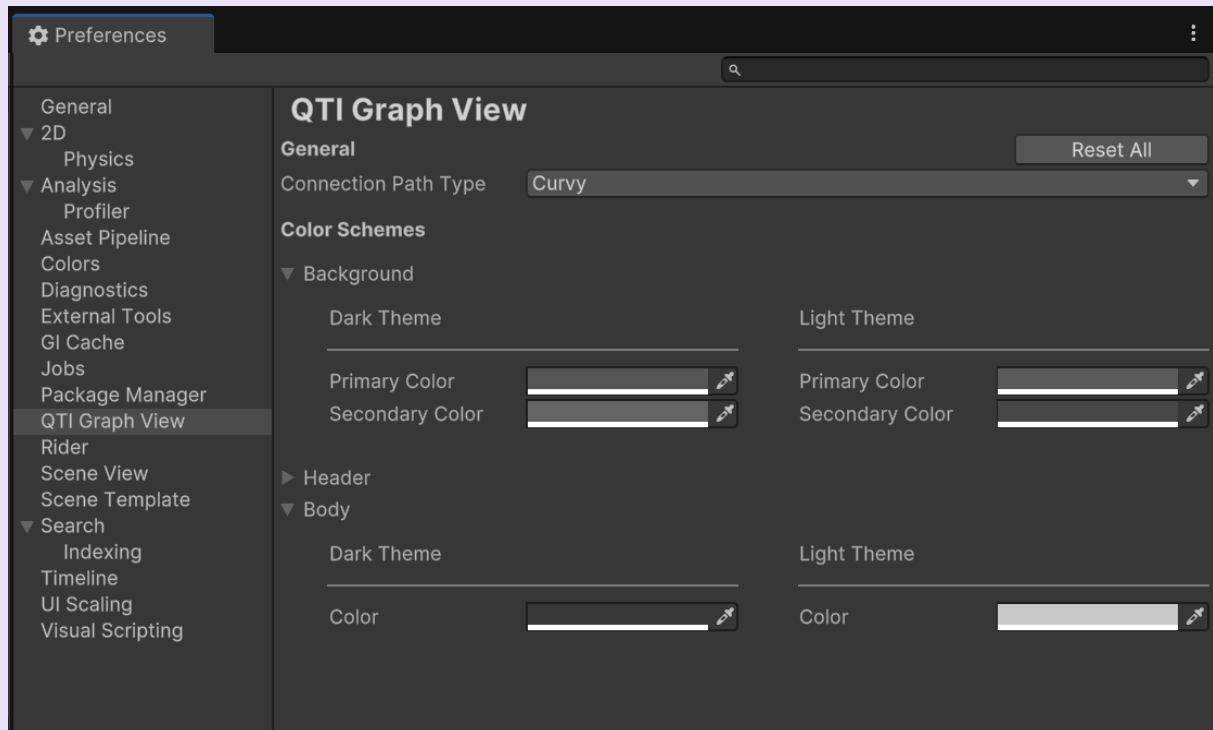
2. PrioritiesEnum

The **PrioritiesEnum Asset** is a Scriptable Object that allows users to customize the number and names of each priority level. Priorities are utilized in the game by **InputTriggers** to determine which InputTrigger is activated when the activation input is pressed. This asset should be properly loaded by the **InteractionSettings** object by providing the correct path and name.



3. Graph View Settings

The Graph View can be customized by navigating to the QTI Graph View Settings menu, found under “Edit/Preferences/QTI Graph View.”

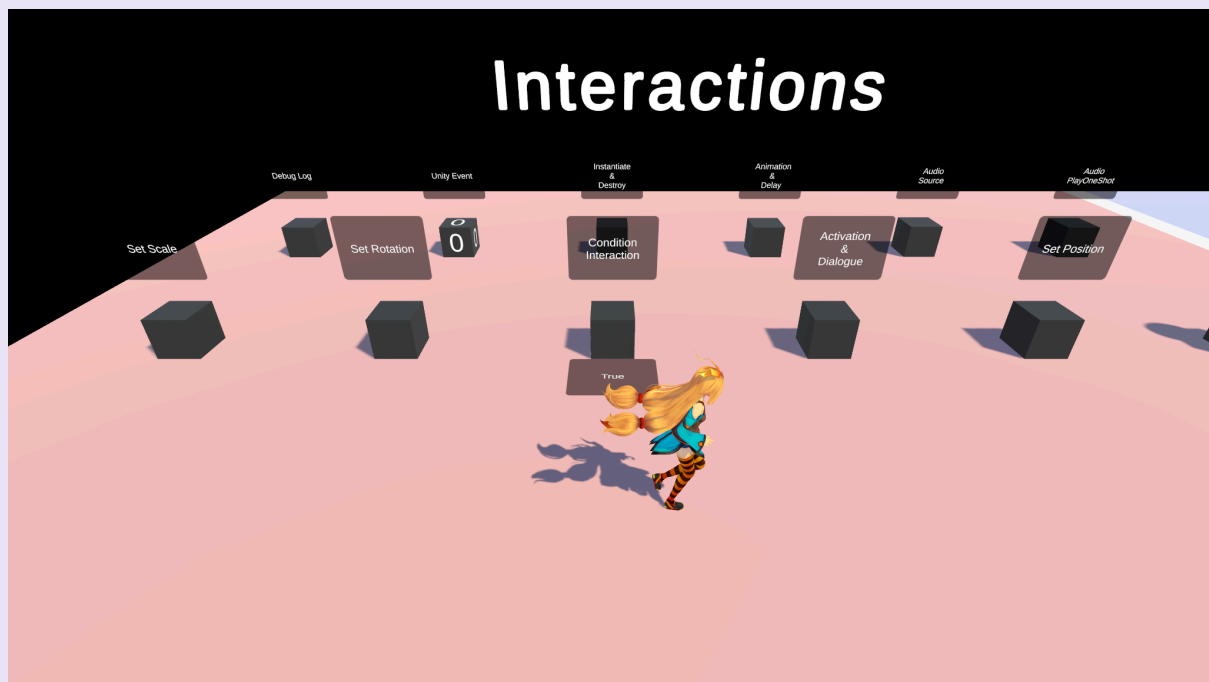


DemoScenes

This package contains a collection of demo scenes intended to showcase the system's versatility and functionality, as well as ways to extend it in different contexts. These scenes offer practical examples of how the system can be applied across various game settings.

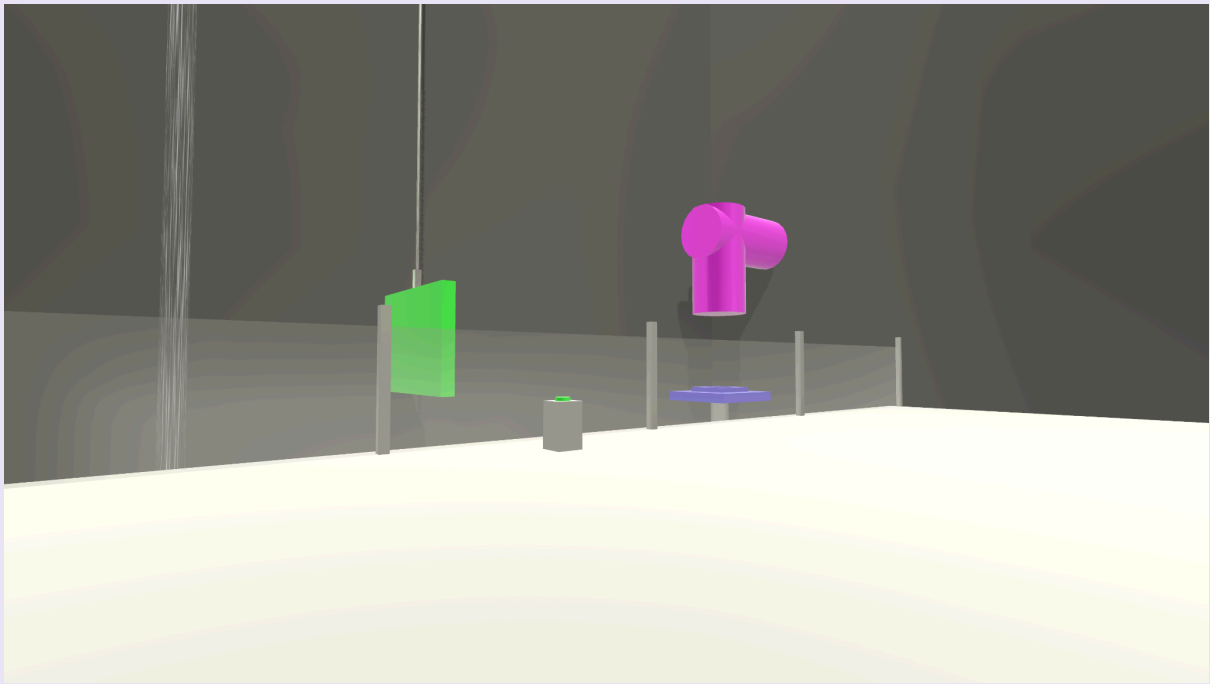
1. Showcase Demo

The **Showcase Demo** scene offers an extensive demonstration of all the triggers and interactions included. It delivers a clear overview of each one's functionality, making it an essential reference.



2. First Person Puzzle Demo

This demonstration scene showcases an in-depth use of various interactions within a first-person puzzle game. It utilizes some of the built-in interactions, along with custom implementations such as **TranslateInteraction**, designed to address specific gameplay scenarios. Additionally, it incorporates the **IInputInteractor** interface in the **DemoFPController** class, as well as the **IInteractor** interface in the **PlatformInteractor** and **PhysicsObject** classes.



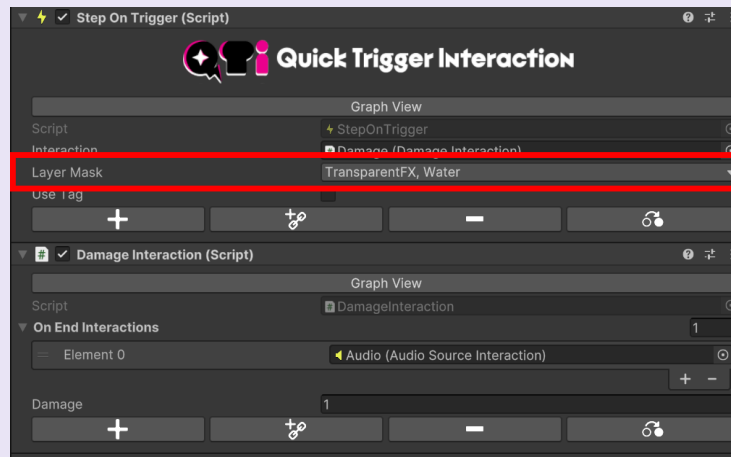
3. Third Person Demo

In this scene, the player can be controlled within a 3D environment, interacting with enemies, traps, doors, and collectibles. Both enemies and traps utilize the custom **DamageInteraction**, which can be beneficial for various game types. This serves as an example of implementation, though other methods may be more suitable for different projects.



The built-in **ConditionInteraction** is used to open doors throughout the level when the player meets specific criteria. It also demonstrates how triggers can engage multiple actors in the scene, enhancing the overall gameplay experience.

To ensure compatibility in different environments, we utilized Unity's default layers: the player is assigned to the **TransparentFX** layer, while the enemies are placed in the **Water** layer.

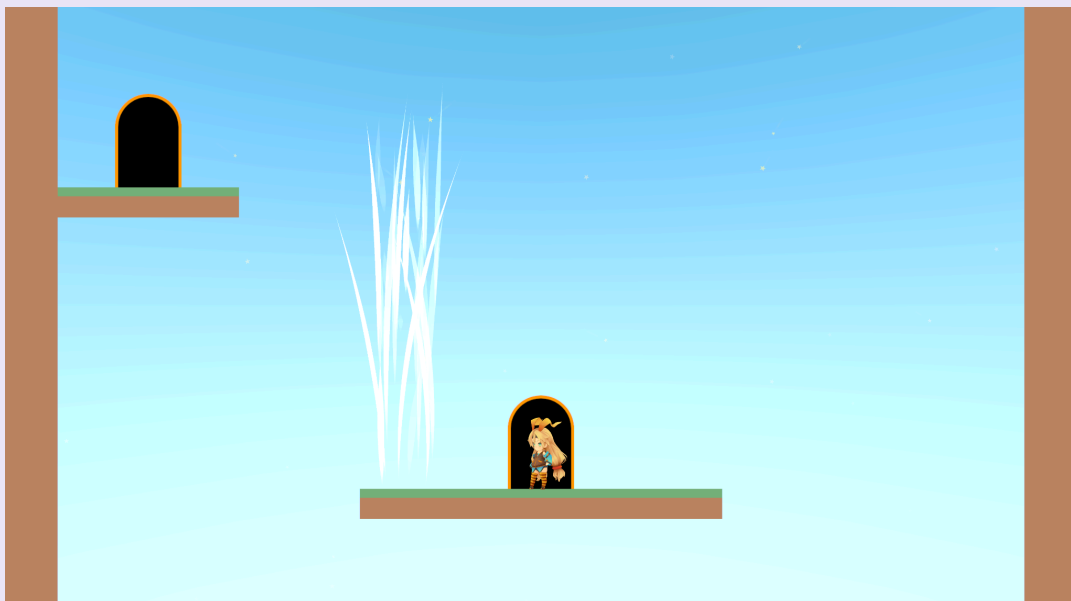


4. 2D Platformer Demo

This scene demonstrates the Interaction system in a 2D platformer environment, highlighting two key interactions: **AddForceInteraction** and **SetPositionInteraction**.

AddForceInteraction is used throughout the level to simulate wind streams that boost the player to higher platforms. These streams are activated by **StayOn2DTrigger**.

SetPositionInteraction enables the player to teleport between doors, with **Input2DTrigger** making the doors interactable through user input.



Credits

Copyright (c) AstralShift. All rights reserved.

Third party assets and their respective licenses can be consulted in the **Third Party Notices** file located at: Quick_Trigger_Interaction\Third Party Notices.md.

In compliance with the UNITY-CHAN License:

