SIMPLE INTERACTION SYSTEM

A simple way to implement interaction, but with great potential. Produce interactive objects faster, with versatility and scalability.

HOW TO USE

→ Installation and Dependencies

Assets need to support our code:

- NaughtyAttributes - Asset Store

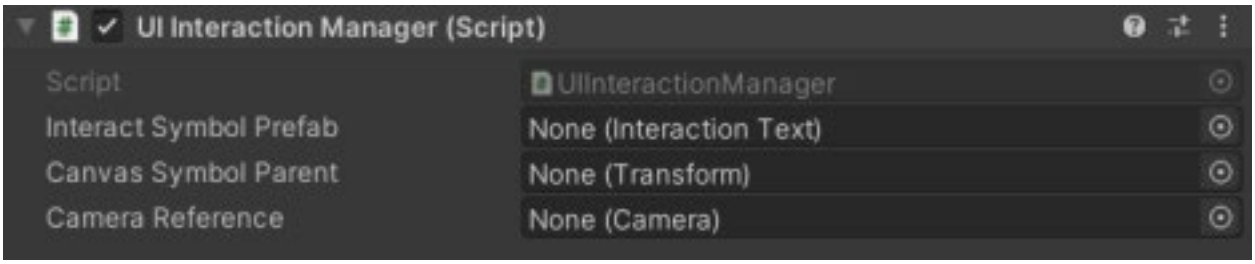
Packages need to use sample assets:

- Universal RP - Install via the Package manager.
- TextMeshPro - Install via the Package manager.

→ UIInteractionManager class

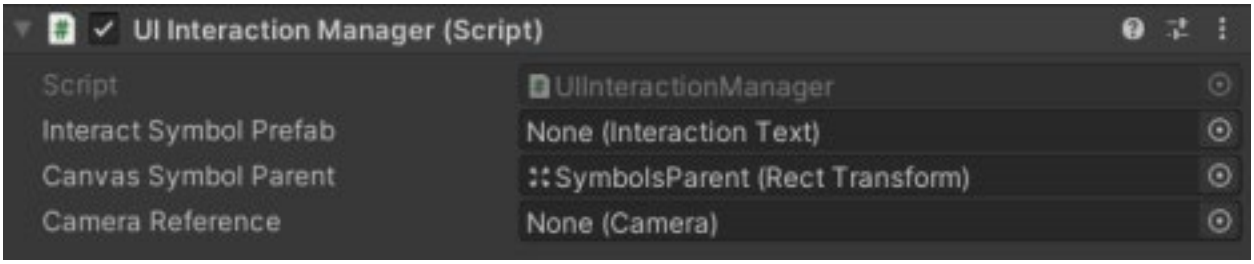
In this step, we will create the UI manager. If you want to skip this part, add the prefab "UIInteractionManager" to your scene.

Add the interaction management UI component to an empty game object. It will be responsible for managing the UI of interactive objects when they are added, removed, or updated by detectors.



Now, create your Canvas and set it up as you like. Add an empty object as a child. This will be added to the "Canvas Symbols Parent" field, and the icons will be instantiated below.

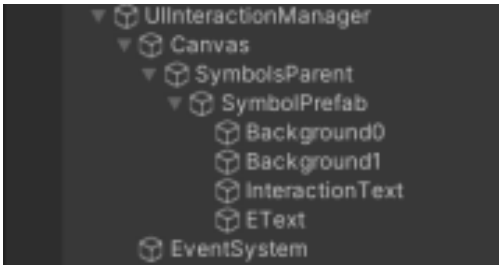
The main camera calculates the position and rotation of the icons, so if you destroy the camera, don't forget to replace it with the “SetNewCamera (Camera newCamera)” method.



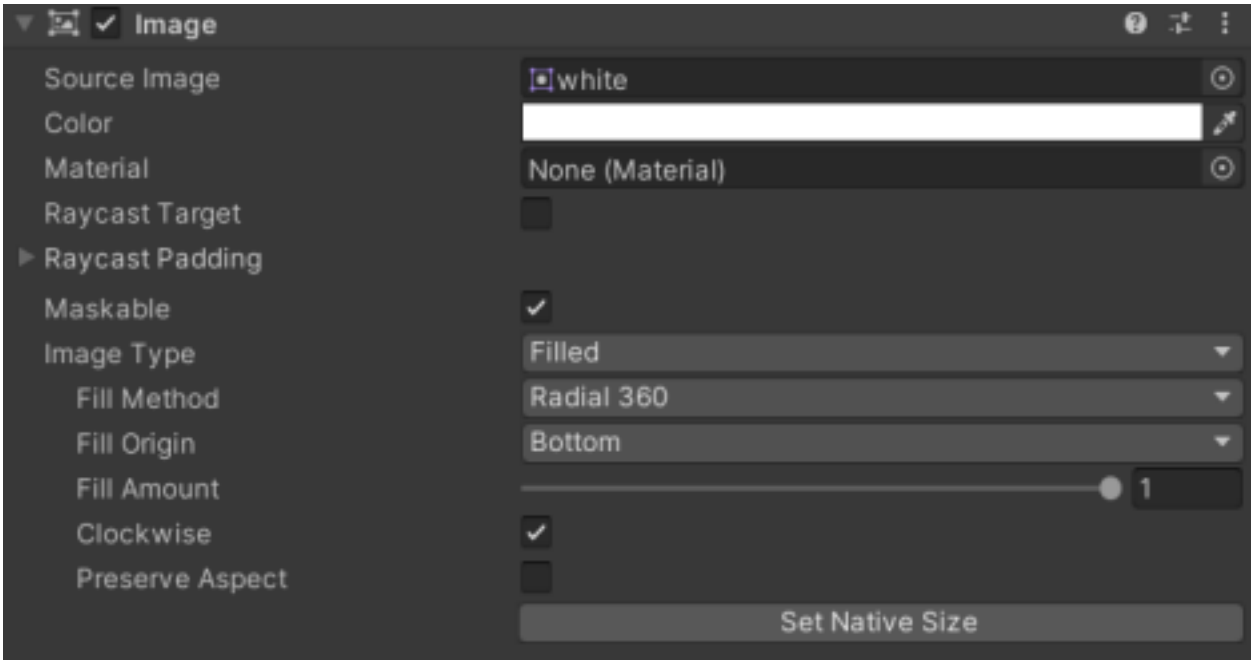
InteractionText

In this step, we will create the prefab of the component that the manager will instantiate. If you used the prefab mentioned in the first item, you could skip it.

Create an empty object with a size of 75. Inside it, we will put two image components, the first one with a size of 90 and the second with a size of 75. We will set the color to black, then add two texts. In the end, it should look like the image below.

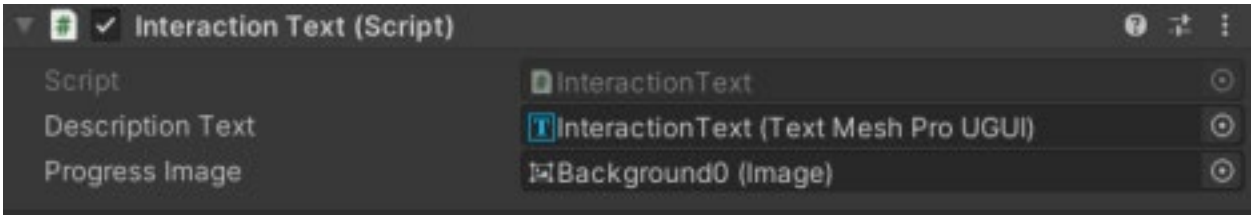


The “background0” image will be used to show the progress of the interaction, so we should set it as shown in the image below:

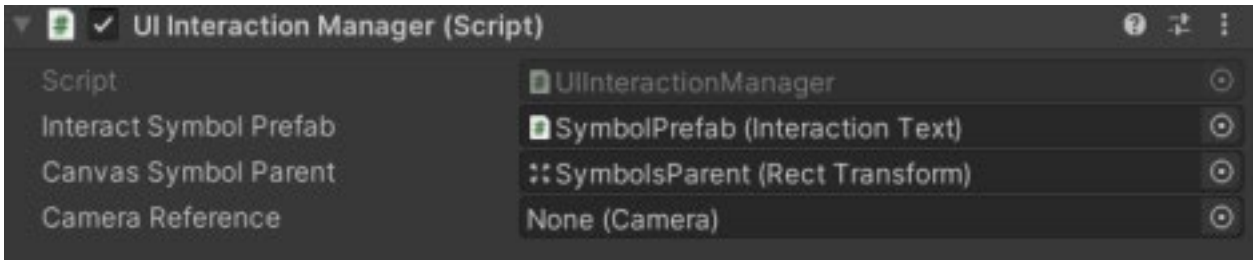


If you can't find it, know that one sprite must be added in the "Source Image" field to enable the Filled type.

In the "SymbolPrefab" object, add the TextInteraction component and configure it as follows:



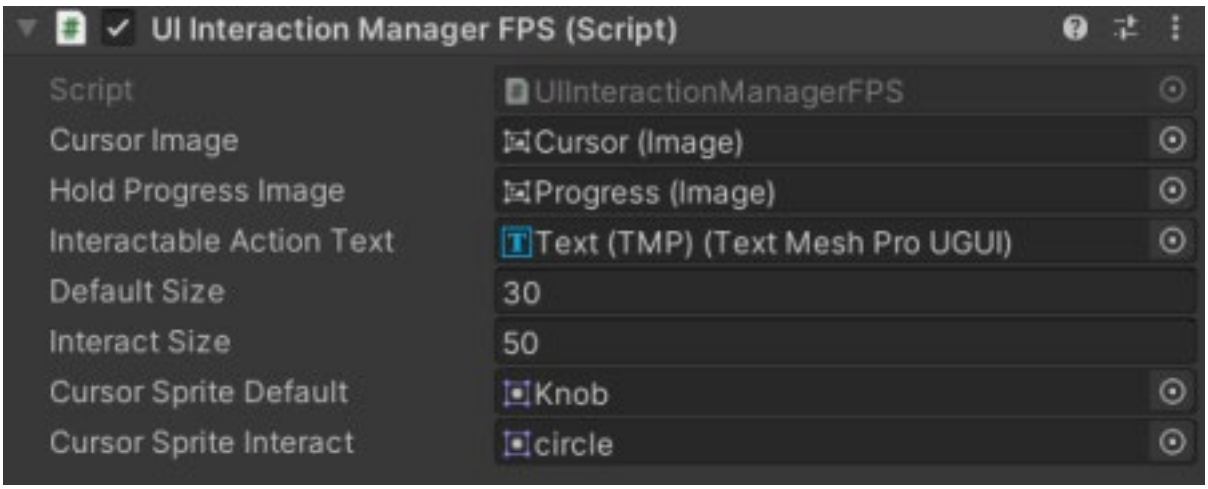
Create a prefab of the "SymbolPrefab" object and delete it from the scene. Then add the prefab to the UI interaction manager in the "Interact Symbol Prefab" field. Final setup:



→ UIInteractionManagerFPS class

In this step, we will create the UI manager for First-person games. If you want to skip this part, add the prefab "UIInteractionManagerFPS" to your scene.

You can configure your UI with the "UIInteractionManagerFPS" component with raycast detection mode for FPS games.



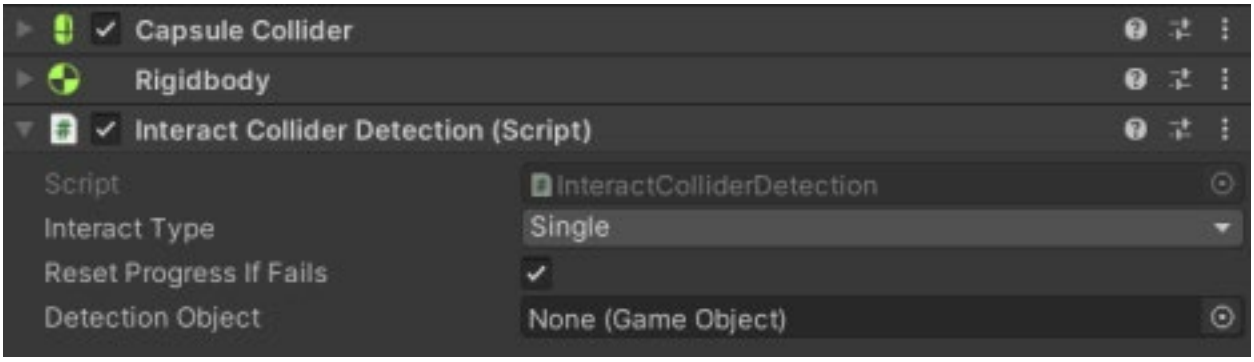
→ Detection Component in Player

The detectors are responsible for storing and interacting with the interactive objects found. By default, there are collision and raycast detectors.

Setting up collider detection component

The collision detector uses the collider to know when an interactive object collides with the player.

Add the "Interact Collider Detection" component to the same object as your player's rigidbody and collider.



In Single mode, even if more objects are in contact with the detector, it only interacts with one. In Multiple modes, it interacts with all the interactable that the collider has detected.

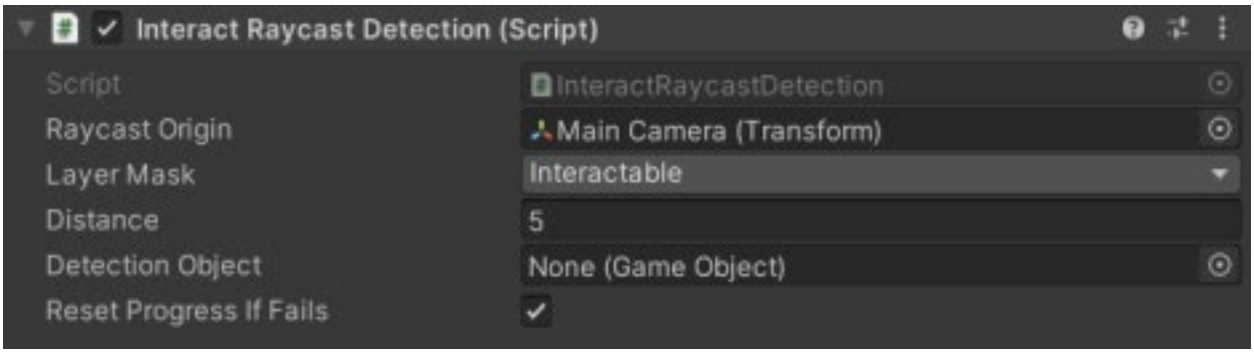
The "Detection Object" field is the detector object. Interactive objects will have access to it via an interface. If the field is null, the script will use the gameObject itself.

The "Reset progress If fails" field resets the progress of the hold interaction when the interaction fails. You can check this behavior in the demo scenes.

Setting up the Raycast detection component

The raycast detector uses the reference point to originate the ray and the forward for direction. Using the distance and the layer mask, it checks if the ray is hitting an interactive object and stores it.

Create a new layer in your project to use a filter on the raycast and set the maximum distance the ray should reach.

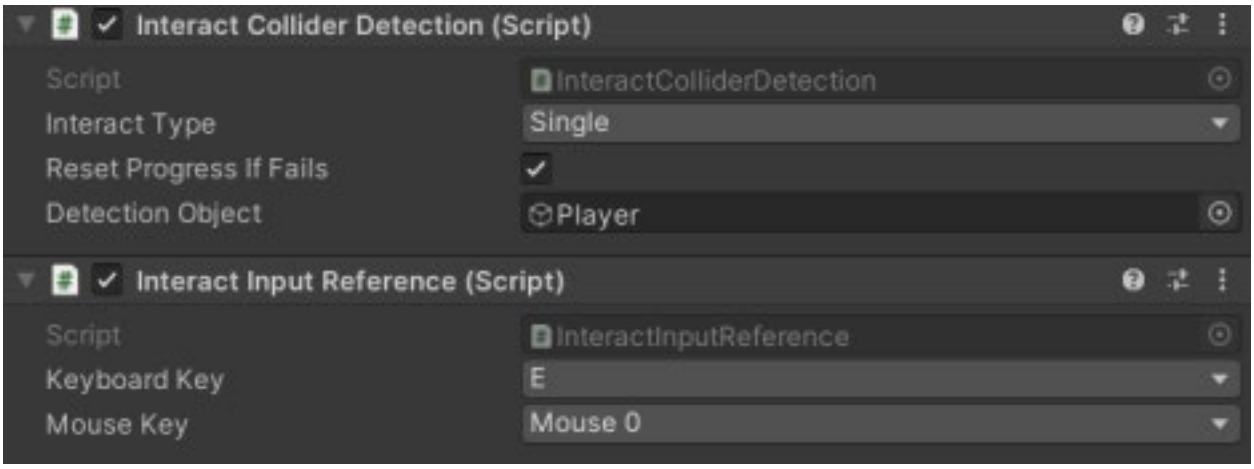


InteractInputReference class

The "InteractInputReference" class is responsible for storing the inputs for the detection systems. Both components mentioned before using this class to know when the player presses some button.

The class uses the old input, but you can inherit from the "InteractInputReferenceBase" class to create using the new input or a custom one.

The component should be on the same object as the detector, for example:



→ Add the custom interactable script

This step will demonstrate how to create an interactive object. It's straightforward.

In your folder, create a new script with the name you want. In this case, we will use "SimpleInteractableExample". Once created, it inherits from the "InteractableBase" class. We will remove the unit start and update methods for better visualization. At the end of the process, it should look like this:

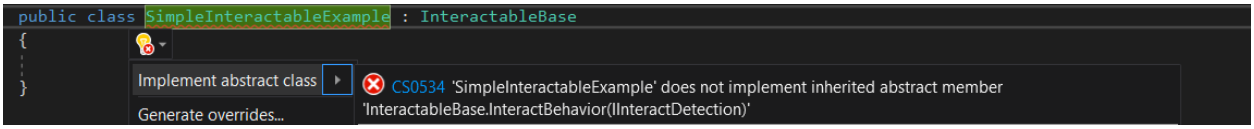
```
public class SimpleInteractableExample : InteractableBase
{

}
```

Now we must implement the interaction method. You can do this manually by typing the method or using your editor to help you.

Rider:

Visual Studio:



Your editor will generate the method "InteractBehaviour", then you can implement what will happen when the detector interacts with the interactive object. You must return the result of this interaction as a Boolean. If it returns true, the action was successful. If it returns false, the action failed. We will leave only returning true to continue.

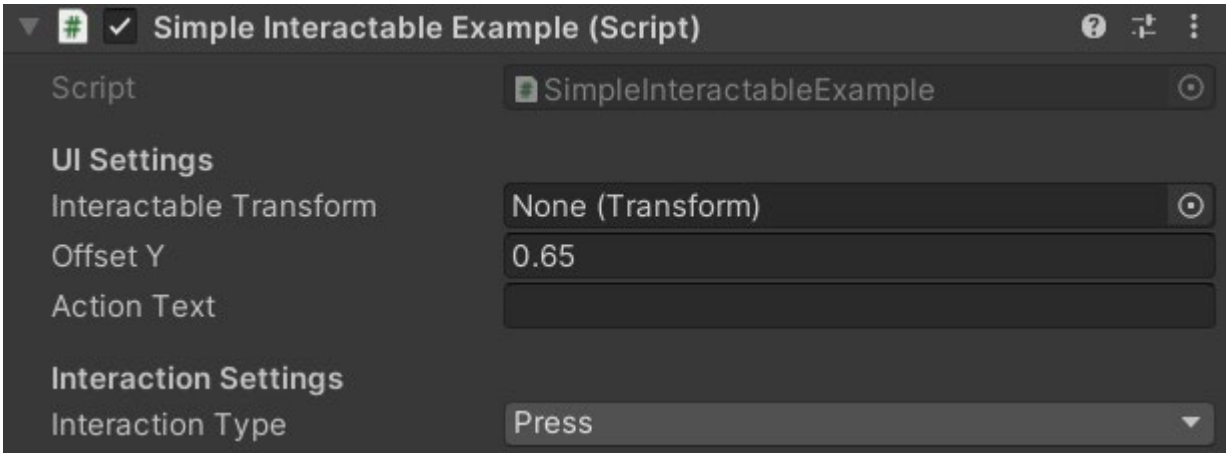
```
public class SimpleInteractableExample : InteractableBase
{
    Frequently called  0+1 usages
    protected override bool InteractBehavior(IInteractDetection detectionReceiver)
    {
        return true;
    }
}
```

Setting up the interactable

Create an empty object in the scene and place a collider. It will be the area of the interactive object. If your player uses the collision detection component, it must be of type trigger. If you use the raycast detection component, remember to place the correct layer on the object and then place the script you made.

Transform and offset are used for the top-down sample but not for the First-person demo.

Every interactive object must have a collider. A warning will be shown if you try to put it before the collider.



In the UI settings, you can set the center of the object's UI, the offset, and a description that will appear on the UI. If the center is null, the script will take the transform itself.

The interaction type can be "press" and "hold". In hold mode, you can choose how long it will take to perform this action. It's Done!

toodoo studio

MAKE YOUR CUSTOM CODE

Some system scripts you can customize to better suit your project, like UI and detection systems through interfaces or inheriting classes. Inside the classes, you will also find more documentation to help you in this process.

In the demo, several scripts were created to speed up the creation process. These will be discussed in topic 4.

→ UIInteractionManagerBase class

You can inherit from this class to make your UI system. It contains the methods for adding, removing, and updating text and the progress bar.

UpdateText(IInteractableUI interactableUI)
Update UI text.

AddInteractable(IInteractableUI interactableUI)
Add interactable to UI list.

RemoveInteractable(IInteractableUI interactableUI)
Remove interactable elements from the UI list.

UpdateHoldProgress(IInteractableUI interactableUI, float percent)
It is used to inform the user interface about the progress of the hold.

→ InteractionText class

You can also integrate with your localization system in the text interaction component.

SetText(string newText)
Set new text.

UpdateHoldProgress(float percent)
It is used to inform the user interface about the progress of the hold.

→ InteractableBase class

The InteractableBase class is the system's main component that provides various events as callbacks, enabling the modular and scalable implementation of any behavior. You can also access essential methods for new implementations.

All these events already have an abstract class for you to inherit from. These classes will be covered in the next topic.

Events:

OnInteractStart
Called at the beginning of the interaction, it returns the object of interactivity and whether it was successful.

OnInteractStatusChange
Called when activating or deactivating the interaction, it returns the new state.

OnInteractableAdded
Called when the interactive object is added to a detector, it returns the object of interactivity.

OnInteractableRemoved
Called when the interactive object is removed from a detector, it returns the object of interactivity.

OnHoldUpdateStart
Called when the holding progress begins, it returns the object of interactivity.

OnHoldUpdateProgress
Called when the holding progress is updated, it returns the object of the interactivity and current progress in percent.

OnHoldUpdateStop
Called when the holding progress stops, it returns the object of the interactivity.

Public methods:

Interact(IInteractDetection detectionReceiver)
Call the interactive object action.

UpdateHoldProgress(IInteractDetection detectionReceiver, float percent)
Tells the user interface about the progress of the hold.

ResetHoldProgress(IInteractDetection detectionReceiver)
Reset progress on UI and trigger stop event.

EnableInteract()
Enable interactable collider and UI.

DisableInteract()
Disable interactable collider and UI.

UpdateInteractionText(string newText)
Change the action description and tell UI that the text should change.

AddDetection()
It tells you that a detector has added the interactive object;

RemoveDetection()
Tells the interactive object to be removed by a detector.

→ **Interactable base events**

Within the "EventBase" namespace, the abstract classes are already ready to use these events. You can inherit them to implement your code.

AddOrRemoveInteractableEventBase
Inherit from it to implement commands when the interactable is added to or removed from a detector.

InteractionHoldProgressEventBase
Inherit from this class to implement behaviors related to interactive objects of type hold.

InteractStatusEventBase
Inherit from it to implement commands when the interactable state is changed, activated, or deactivated.

PostInteractionEventBase
Inherit from it to implement commands when the detector interacts with the interactable, returning the success or failure of the interaction.

→ Detection interface

The Interactable base class uses the interface "IInteractDetection" to detect an object that can interact with it. Implement it to create new detection types or customize existing detectors. To ease the process, you can use already programmed collider and raycast detection systems as examples.

→ InputReferenceBase class

The "InputReferenceBase" class is responsible for storing the inputs for the detection systems. You can inherit from it to create using the new input or a custom one.

toodoo studio

HOW TO SPEED UP YOUR FLOW WITH SAMPLES SCRIPTS

Several scripts have been created to exemplify the use of the event base that the "InteractableBase" class provides to us. They are used to make the code more modular and speed up the creation and learning process.

→ Sample scripts

ActionTextOnAddOrRemoveInteract
Responsible for updating the interaction description when the interactable is added to or removed from the detector.

ActionTextOnHoldProgress
Responsible for updating the action text when the progress of the hold changes.

ActionTextOnInteract
Responsible for updating the interaction description when the detector interacts with the interactable.

AnimateOnInteract
Responsible for triggering animations when the detector interacts with the interactable.

HandleObjectsOnInteract
Responsible for activating or deactivating objects when the detector interacts with the interactable.

MaterialOnInteract
Responsible for updating materials when the detector interacts with the interactable.

toodoo studio

DEMO SCENE

The package has two demos, one for first-person and one for third-person or top-down games. The previous topic used all the classes mentioned in the demo scenes. All classes are documented in the code. Simple systems of doors, keys, and stairs interact with the player, simulating an inventory.