Interactor Documentation

Please refer to Online Documentation
for latest changes.

You can access Online version of this document from here. Online Version is suggested!

Introduction

Thank you for your interest in Interactor!

Interactor designed to cover all aspects of any kind of interaction, from the designing stage in the editor to handling complex interactions in runtime. It contains editor tools to simplify the process of preparing a fully interactable environment and boosts creativity in terms of interaction freedom. And in runtime, it handles which player parts are going to interact with which parts of the interacted object, while using its own IK (or Final IK) to animate player bones.

If you have any questions that are beyond the scope of this documentation, please feel free to contact me.

Installation

Interactor installation is easy. Just import into your project (import Final IK first if you wish to use integration with that).

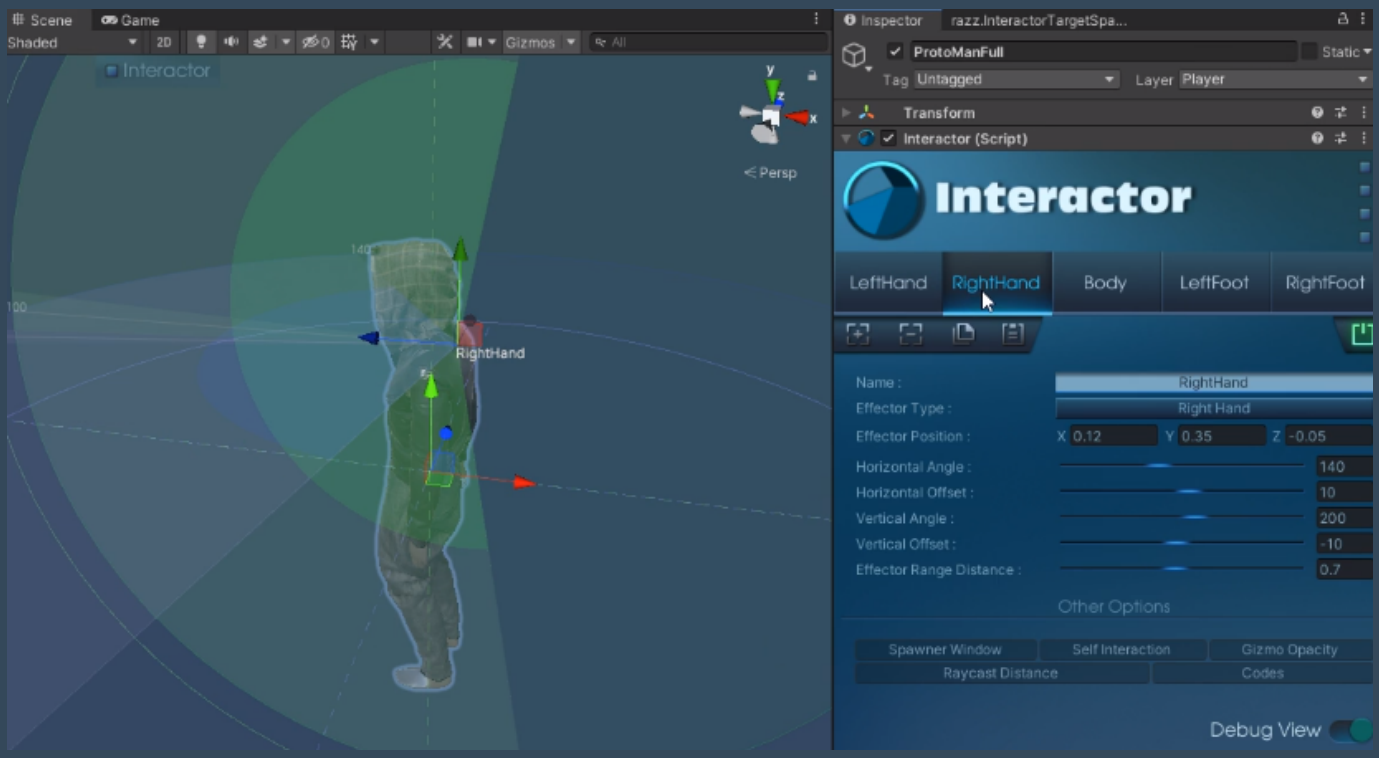
It needs a “Player” layer for raycast operations. Create a new one if you don’t have Player layer and assign your player to that layer. This will prevent conflicts while spawning new targets on SceneView and Runtime camera rays for distance objects to interact. Otherwise, player triggers & colliders will block those raycasts.

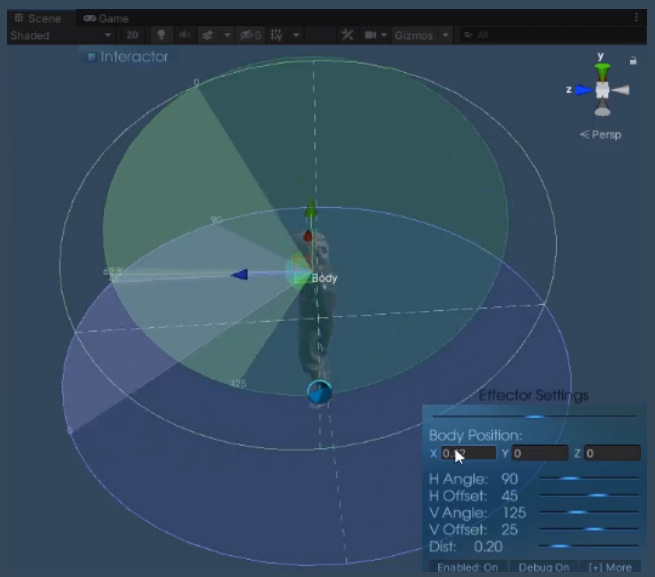
Change your Color Space to Linear on "Project Settings > Player > Color Space". This is for getting some materials to look correct in example scenes. Otherwise, it is not necessary.

Done!

How Does It Work

After Interactor being added to the player character, you need Effectors for each interactable player part. You can add as many effectors as you need onto your character. Effectors need to be positioned as they are the root of interactable parts so this way you can properly adjust their range and angles.





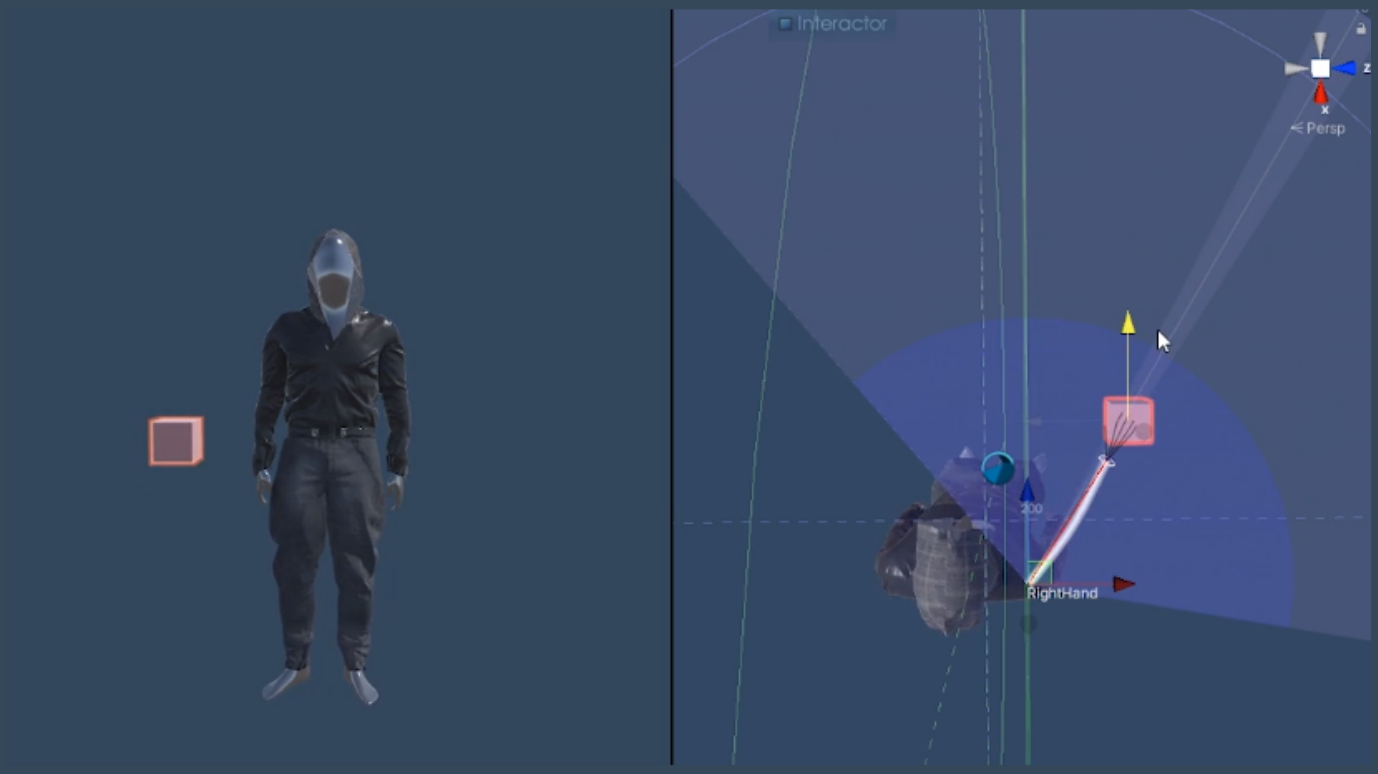
These are actually short videos, if you click on them it will direct to Online Documentation page. You can watch videos from there.

It's quite easy to set positions for effectors and you can place them exactly where you want them to be, with custom made gizmos.

One Trigger to Rule Them All

When an interaction object enters Interactor's trigger (player's interaction trigger from now on), effectors start to check that object's targets which are children of it. Target has another component (InteractorTarget (InteractorIK) or InteractionTarget (Final IK's class)) for interaction and it has effector type property and some override settings for range & angles. Effectors check those types to get a match and then check their positions to see if the interaction is possible.

If the interaction is possible, the effector goes into a possible state and activates that interaction object. From that moment it is possible to interact with object automatically or with player input (depending on interaction type) until the object's target position stays in range & angles. On the animation side, it's InteractorIK's or Final IK's turn. Interactor sends an interaction start call for that specific player part and its target.



Interactor Workflow

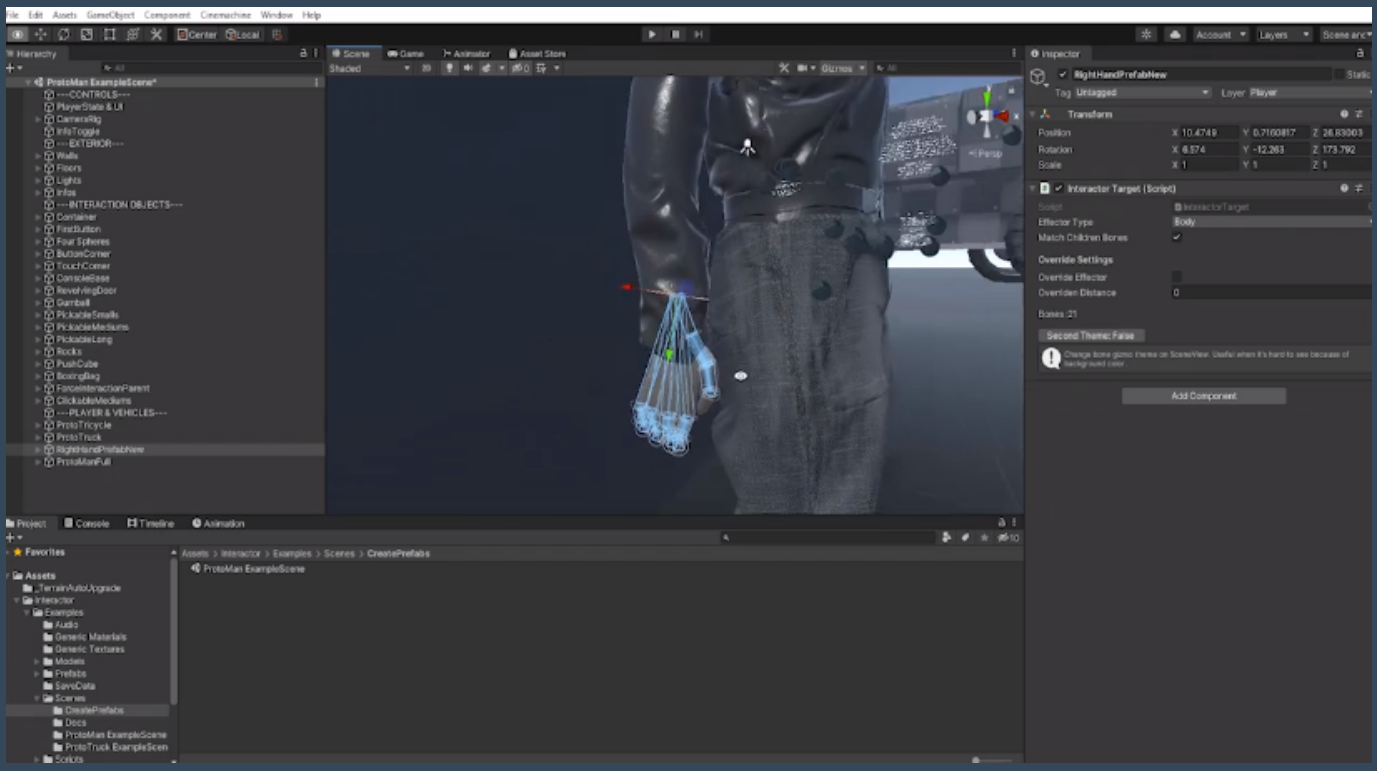
Interactor has three main components:

Itself which operates its effectors and interactions,

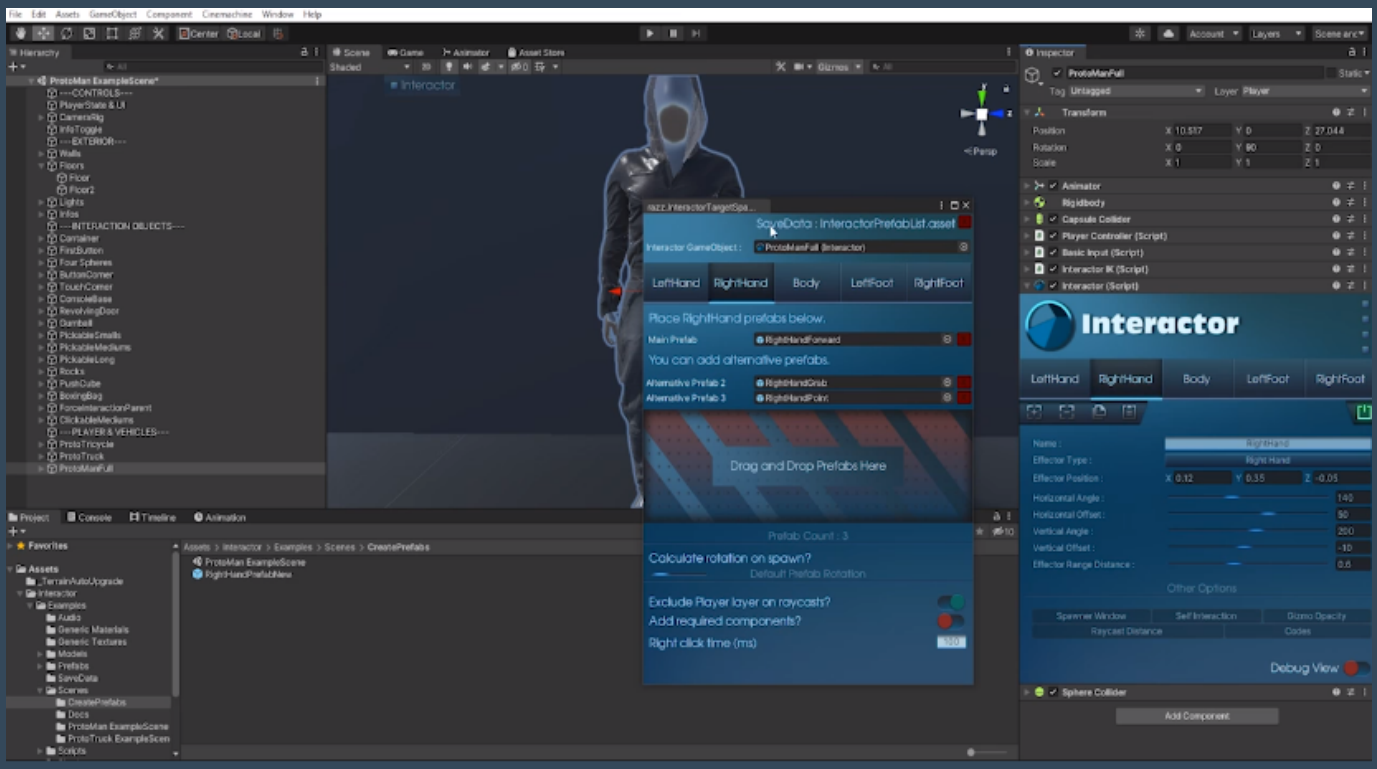
InteractorObject which handles the interaction object's side, caches its values & targets for being used in Interactor and helps with interactions,

InteractorTargetSpawner which plays a big role in the designing stage as a target prefab holder and spawning them in the scene.

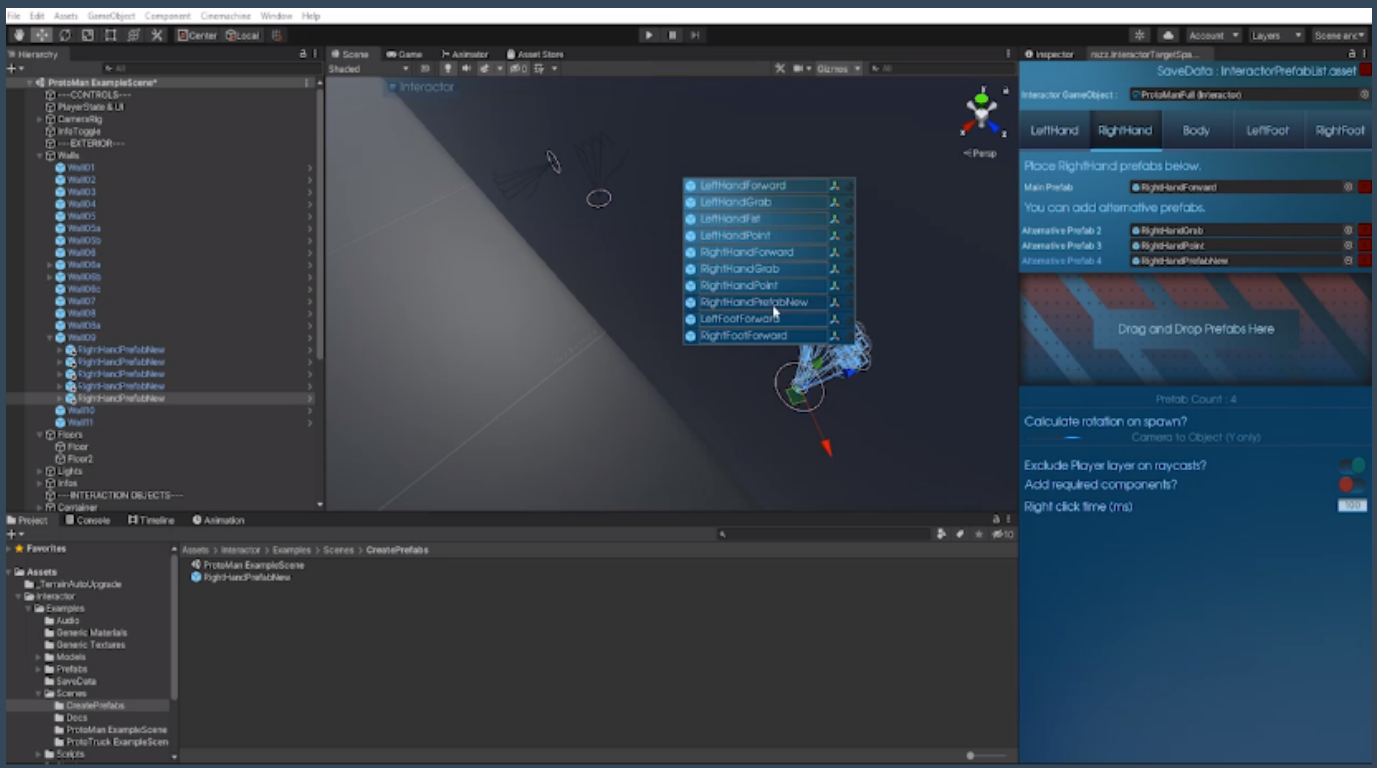
First, you need to prepare some prefabs to work with. These are regular player parts with specifically rotated to fit desired interactions. For example a grabbing hand, finger pointing, etc. It's really easy to prepare them and it's just a one-time setup.



Select your body part and duplicate it in the scene. Assign InteractorTarget (InteractionTarget for Final IK) script on it. Select the effector type and other settings on the target script. Set its position to (0, 0, 0) and bone rotations as you wish. Drag it in the project folder to make it prefab and done, you have your first prefab. Make a list of selection for each effector type & body parts. Assign them on InteractorTargetSpawner to add them on SceneView right click menu. Now they can be spawned anywhere and anytime on any object you want. Only InteractorTargetSpawner window needs to stay opened (It can stay as deactivate tab on any place in the editor) for activating the right click menu in SceneView.



Besides of right click menu, also Effector Settings menu on SceneView also needs InteractorTargetSpawner window to be opened. Once you created effectors on the Interactor component, you can edit them on there or with SceneView Effector Settings window. Effector Settings menu can be used in fullscreen SceneView (Shift + Space for default shortcut while cursor on SceneView) for maximum effectiveness.

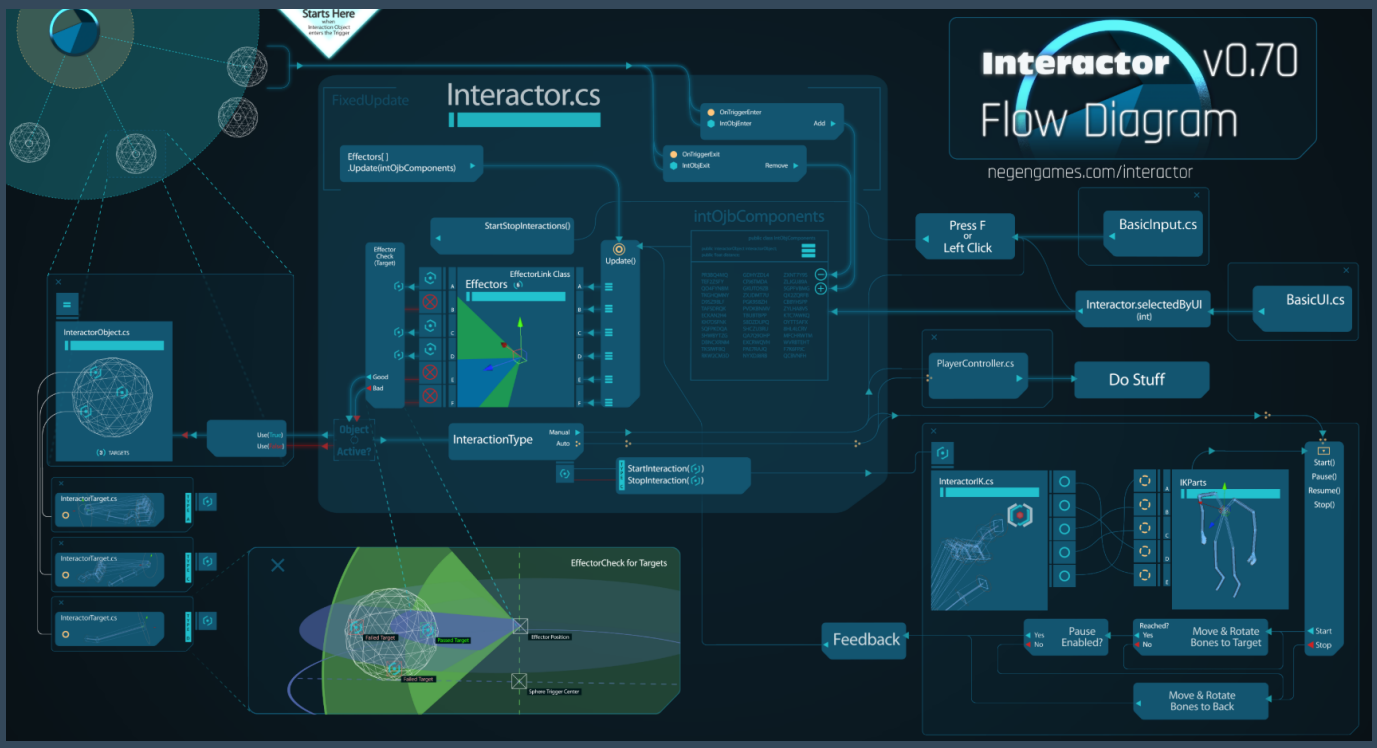


So you have effectors and targets on the scene. While spawner targets in the scene, there is an option on InteractorTargetSpawner to add components. It will add InteractorObject on the spawned target's parent. InteractorObject operates interaction object's settings (On Final IK you also need InteractionObject on objects). You can select interaction types and have a lot of options to make tiny adjustments on interactions. More information can be found at InteractorObject component page.

This is the basic workflow of the Interactor. You can find more information for each component at Interactor Components section and some more tips at the Tips section. Also, don't forget to watch tutorial videos, it's a lot easier to follow steps while watching.

Flow Diagram

This is how Interactor and other scripts communicate. It will be improved and some unnecessary scripts will be removed with upcoming updates.



Click on image to open fullscreen link.

User Interface

This section explains the UI workflow, not the settings one by one. For that, you can read the Interactor Components for each one of them.

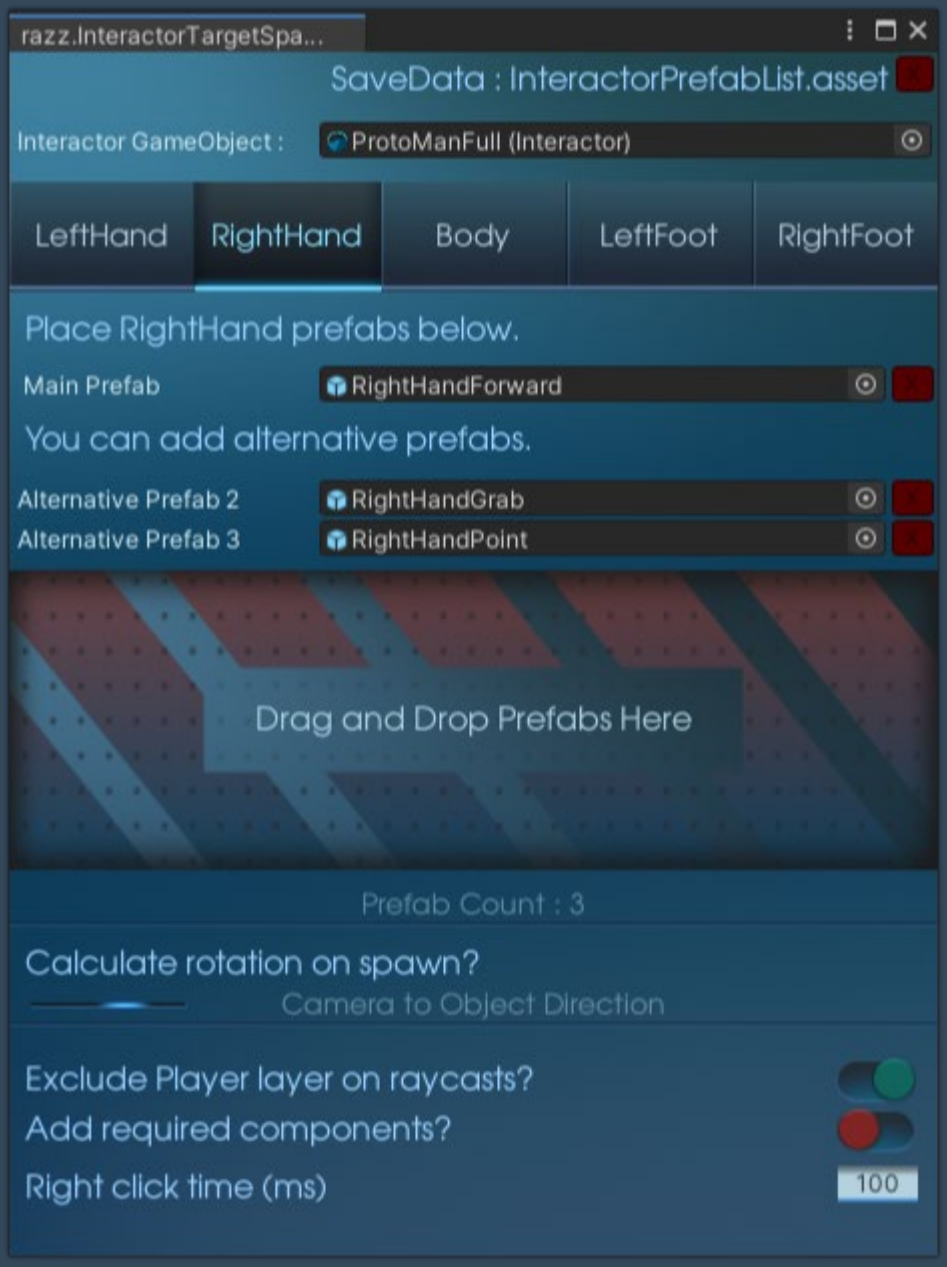
Interactor has the heaviest editor script in all of the components that come with the package. It handles SceneView custom gizmos as well as the tab selection and drawing a good looking UI. It almost works as fast as regular Unity Inspector components (On Unity 2020 regular Inspector component takes 0.15-0.20ms to complete it's editor loop, while the main Interactor UI is taking 0.25ms average). And all windows have zero effect when the mouse cursor is not on them. But still, Interactor designed to work without the need of its main component to be selected. You don't even need to see Interactor UI to work with. You can use the SceneView menu or SceneView handles to edit effectors. Interactor is needed just for adding new effectors and some one-time setups (Assigning Self Interaction object etc). And also if you're not working with a really really tight performance budget while profiling on Editor (It doesn't even make sense), you don't need to worry about anything.

Interactor, InteractorTargetSpawner and SceneView Effector Settings UIs are working in sync. If you make any changes, other UIs will show results at the same time. Also, almost every change can be undone with Alt + Z or Undo selection on editor.

Interactor

For information about settings, see Interactor component section.

Interactor Target Spawner



For information about settings, see InteractorTargetSpawner Window section.

Interactor SceneView Menu

For information about settings, see InteractorTargetSpawner Window section. SceneView right click menu and Effector Settings windows are connected with InteractorTargetSpawner.

Go to Interaction Types page

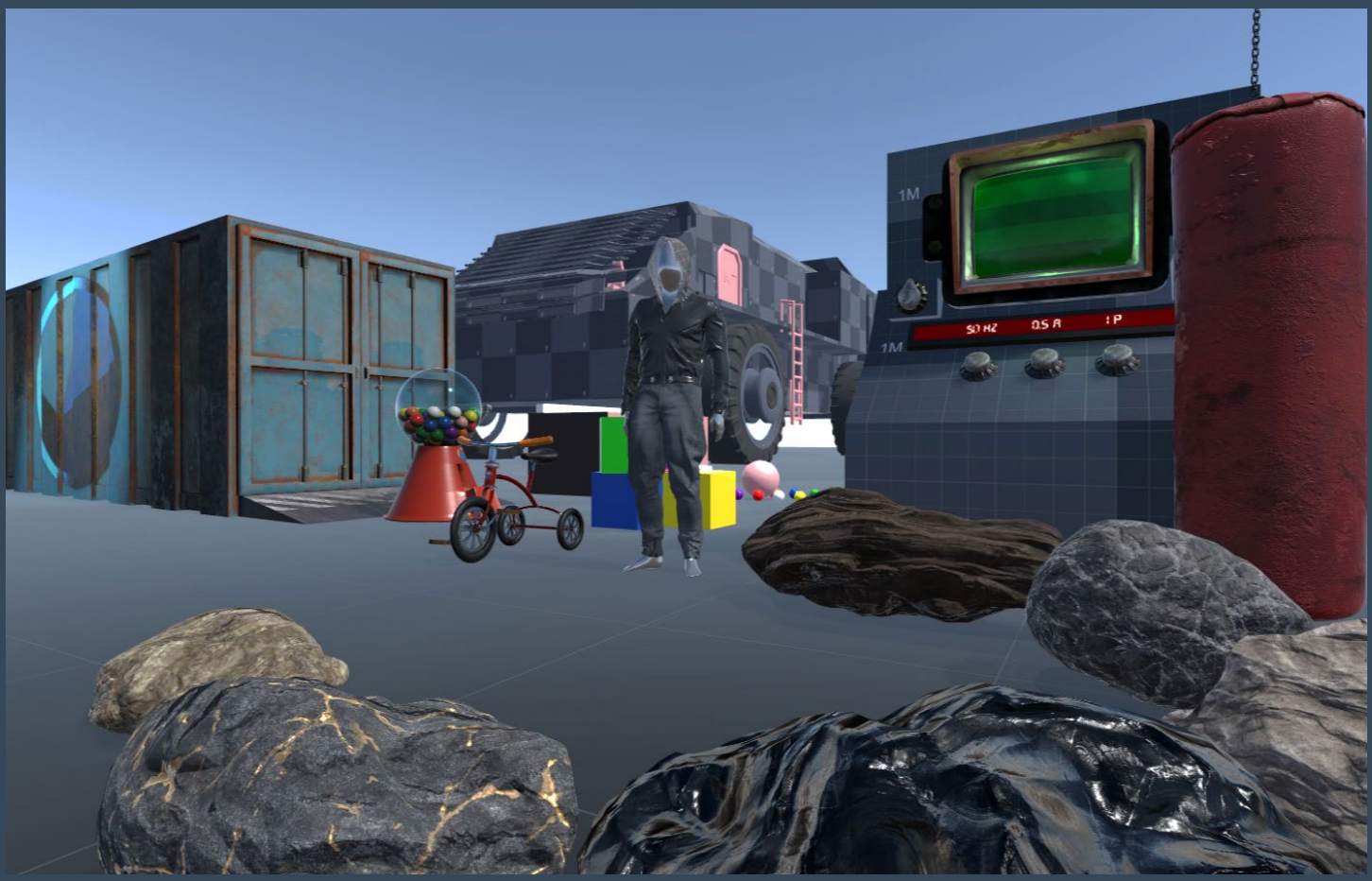
Go to Interactor Components page

Example Scenes

You will find examples for every interaction type in example scenes. There will be more examples in those scenes when new interaction types added.

It's a good playground the see how everything works. Some interactions can interrupt each other and that can cause some glitches (trying to climb ladders while using the bike etc) but it's not a polished game in the end. But mostly I set controls/conditions for such glitches and it will be better with time since I'm also using Interactor in my game.

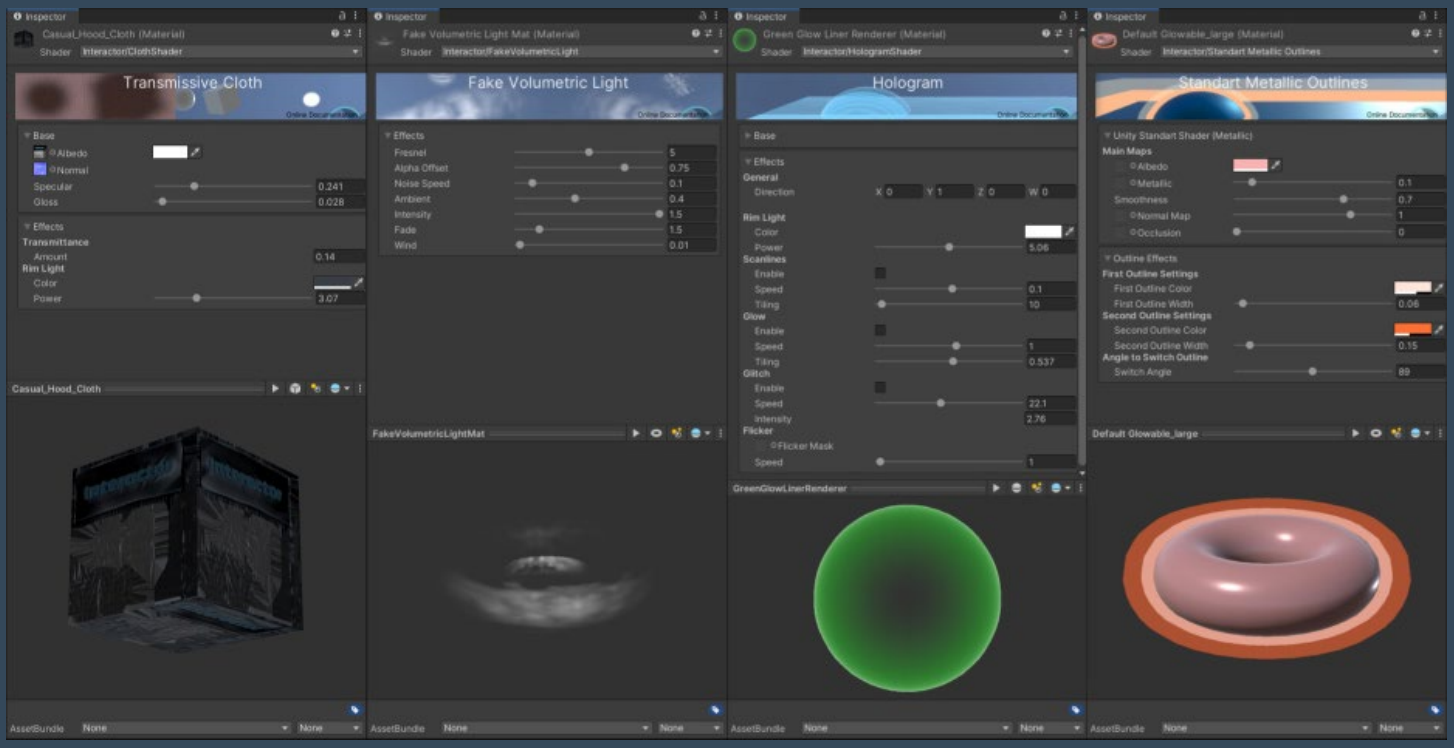
Models & Textures



Most of the 3D models are modeled & painted by me. While adding new interactions, I will also add new models to play in new ways. All of the textures are PBR and available in 4K but Interactor comes with 2K textures to keep the size as small as possible (so updates would be easier). You can download the 4K texture pack from here. Once you imported the package, all textures will change with 4K versions, no extra procedure needed, just import and done.

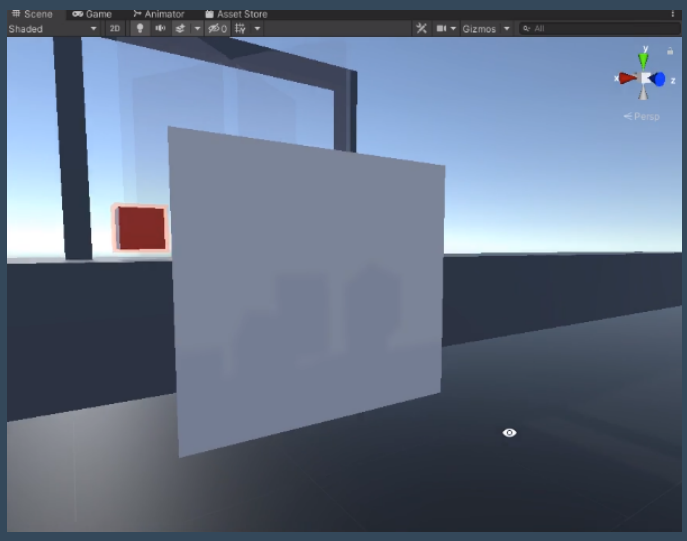
Shaders

I included some bonus shaders. Outlines Shader is directly used in examples because it controls the interacted object's material but others are mostly cosmetic. Shaders are written just for Built-in Render Pipeline so probably they won't work in HDRP or URP. But they are not necessary for Interactor, as I said they're just bonuses. Besides the shaders, Interactor will just work the same as Built-in RP for Scriptable Render Pipelines (HDRP and URP). You will find these shaders under the Interactor menu in shader selection for materials.

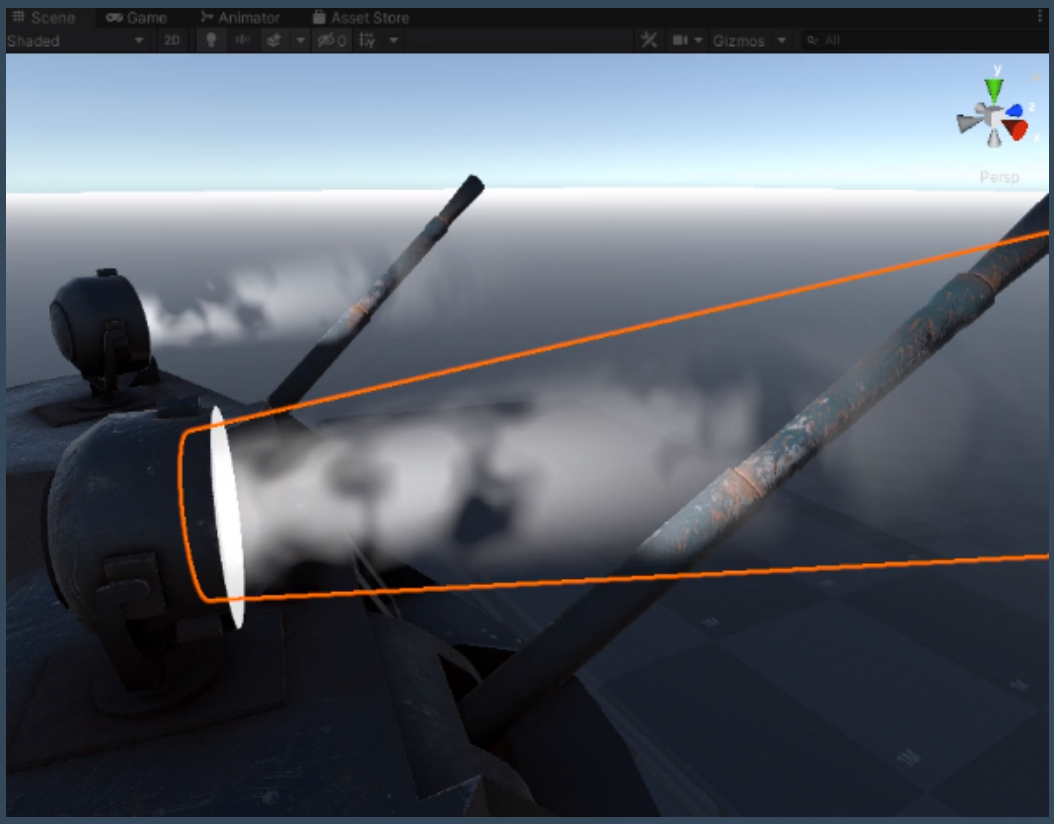


Cloth Shader

Cloth shader has a transmissive feature and lets light pass through. So the object shadows can be seen from both sides of the cloth. It has Albedo and Normal map slots and Specular & Gloss sliders to adjust the material. The transmittance amount can be adjusted and there is a rim effect that glows the whole object with the selected color.



Fake Volumetric Light Shader

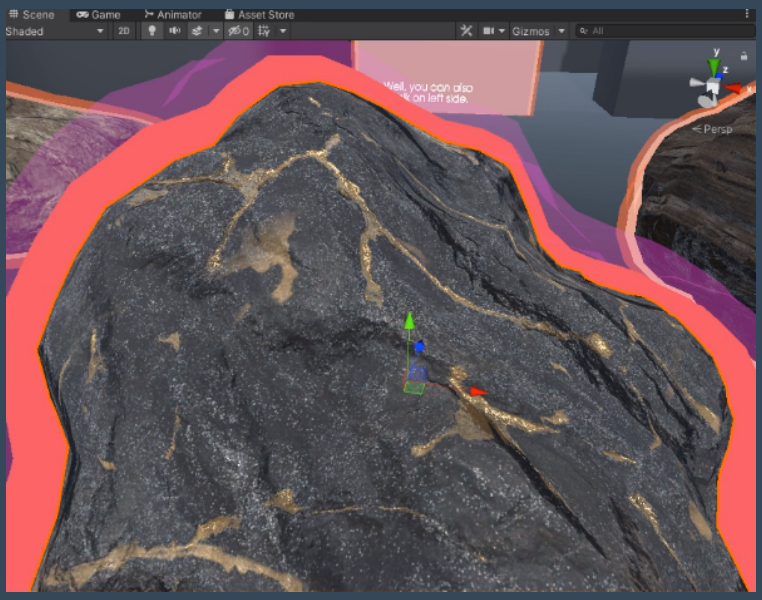


FakeVolumetricLights used in the ProtoTruck example scene and they create a basic fake volumetric light effect for turret lights. They used on a cone object to look like a light beam. You can play with settings to get the desired look that suits your projects.

Hologram Shader

Hologram shader is used for creating an old terminal screen effect. It has properties for setting the old tv scanlines, glitches, flickering and more. It can be used for any kind of project.

Standart Metallic Outlines Shader



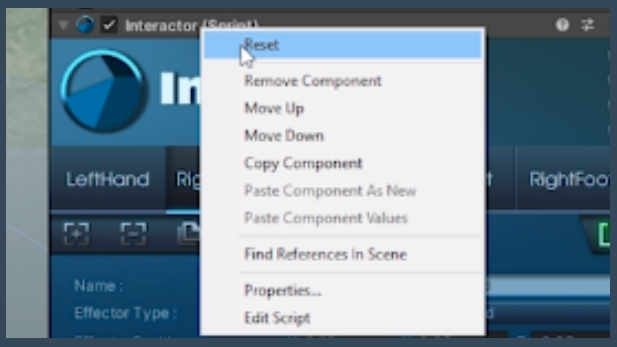
Outlines shader is the main material for interaction objects in examples. When objects get activated their properties adjust its glow to notify the user. It has two different glowing options and they can be set with different colors as well as with different widths. I scripted it to look exactly the same with Unity Standart Metallic shader so when not glowing, it will just look exactly the same with Standart materials.

3D Text Shader

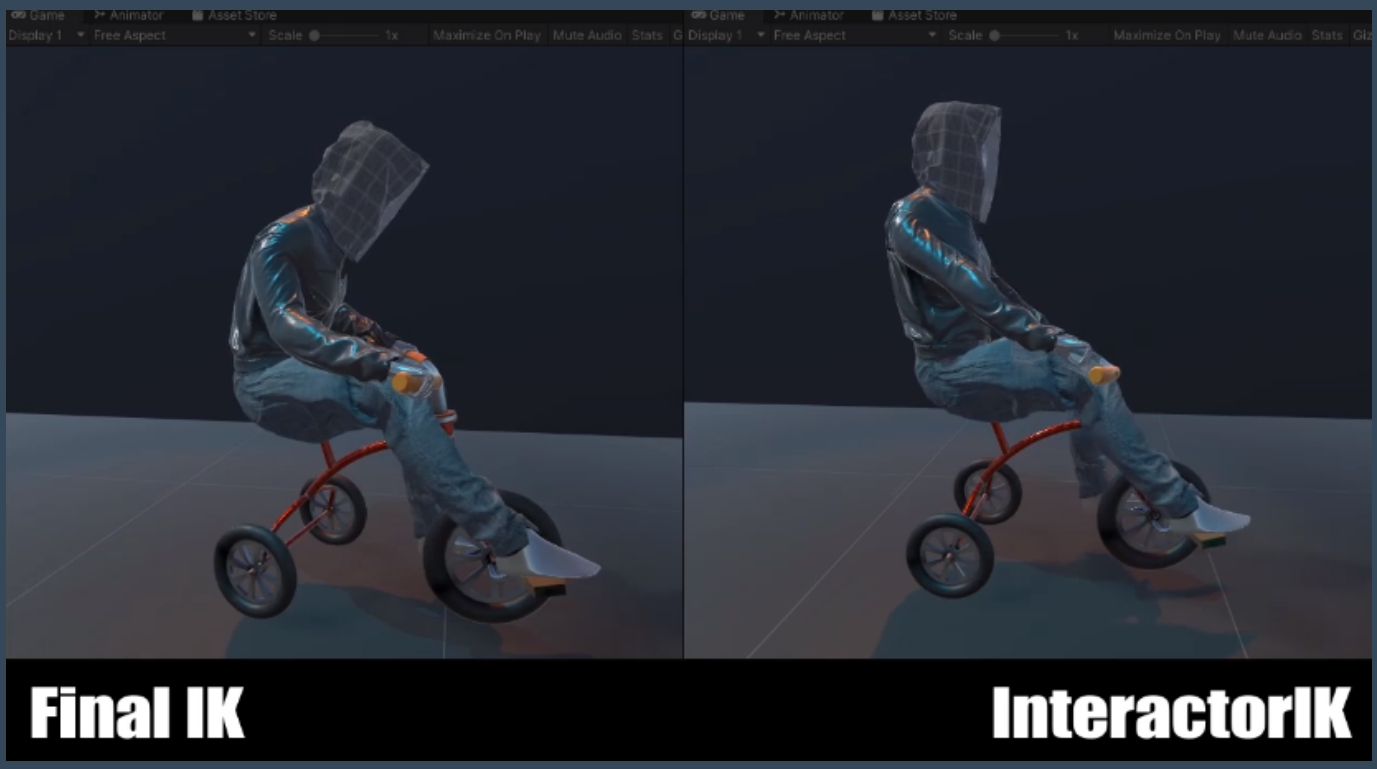
3DText shader and SceneText shaders are used for some text objects in scenes. 3DText is for the console's display frame to show values.

Integrations

Currently, there are three versions of Interactor. When you import, it comes with DefaultPack (InteractorIK) and you can convert between DefaultPack and FinalIKPack (Final IK integration) on the Interactor component at any time with just a button click. To do so, add the Interactor component on any hierarchy gameobject and you'll see the integrations section on newly added Interactor. If you already have Interactor with effectors added, just reset it to see its front page (Integrations section only appear on the front page without any effectors added).



All of the interactions work mostly the same between InteractorIK and Final IK. But since Final IK is a whole different package with more capabilities, it's a better solution for interactions. For example, some of the interactions need body parts to go beyond their boundaries, in that case, Final IK can pull or push the whole body hierarchy to adjust that. But InteractorIK is Two Bone IK system depending on Unity IK. It can only affect hands & feet and their two parents (Forearms & arms or Uplegs & legs). So for example picking objects from the ground is not possible with just those bones without affecting whole body. When InteractorIK needs to out of it's IK part boundaries, it will just stay at the boundary at most where it's two parent bones would allow.



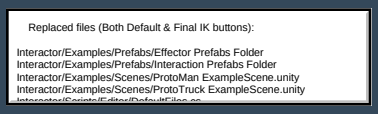
But both IK solutions mostly do their jobs and works pretty same. Maybe in the future updates, I would replace InteractorIK with a Full Body IK solution.

InteractorIK & Final IK

What actually integration change buttons do?

Default and Final IK integration buttons unpack and imports necessary scripts. Some of them are replacements and some of them are exclusive to themselves.

Those files/folders get replaced by their respective package.



Those files are added from their respective package while deleting other package's files.


```
    Replaced files (Both Default & Final IK buttons):

Interactor/Examples/Prefabs/Effector Prefabs Folder
Interactor/Examples/Prefabs/Interaction Prefabs Folder
Interactor/Examples/Scenes/ProtoMan ExampleScene.unity
Interactor/Examples/Scenes/ProtoTruck ExampleScene.unity
Interactor/Scripts/Editor/DefaultFiles.cs
Interactor/Scripts/Interactor.cs
Interactor/Scripts/InteractorObject.cs
Interactor/Scripts/Editor/Links.cs

```

Also Final IK button checks for Final IK's two interaction scripts and adds two overloaded methods (if not added before). Those methods do and change nothing for Final IK itself, so no need to worry about it.

```
    Default (InteractorIK) button:

Interactor/Scripts/Editor/InteractorTargetEditor.cs
Interactor/Scripts/InteractorTarget.cs
Interactor/Scripts/InteractorIK.cs

    Final IK button:

Interactor/Scripts/InteractorOverride.cs

```

CorePack & Examples Removal

Also, there is CorePack which has only core features without any example scripts. It's for users who want only core Interactor features (Interactor, Spawner, SceneView menus, etc) without any interaction examples. So you'll need to write your own interaction types by yourself. This will also remove the whole examples folder for you. Do not delete the Examples folder by yourself because it has example scripts tied with Interactor. If you don't need examples, use Interactor's Delete Examples button to convert to the Core version. You can delete everything in the example folder except the example scripts folder. Just move them inside the main scripts folder before deleting the examples folder if you wish to use interactions. Example scripts are not modular right now.

As a last reminder: DO NOT run these packs by yourself, use Interactor (add Interactor to any scene object and select at Integrations section). Because some files need to be deleted and Interactor handles that for you.

Unity Animation Rigging

Unity Animation Rigging package is also another IK solution for users. It has been in development for some time and still in development. It’s a great companion for Interactor. It has tools to animate bones and Interactor has tools to tell which bones needed for what kind of interactions. They complement each other, Animation Rigging package is a great replacement for InteractorIK. But since it’s still in the preview stage and in early development (which means it’s not ready for production), it will be wise to wait a bit longer. It also uses the Burst package (another preview package in development) which makes it not suitable for most users.

Go to Tutorial Videos page

Interaction Types

There are more than 16 different interaction types as examples. All interaction objects should have InteractorObject component and most of them have child objects called targets. Targets are prefabs that you prepared earlier on Interactor Workflow. They are basically hand, feet or spline bones with InteractorTarget (InteractionTarget on Final IK) component attached to them. And they have an effector type property to set them for an effector (Left Hand target only belongs to Left Hand effector etc).

You can modify interactions or improve them to create new ones by scripting. The list will expand and some of them will improve with upcoming updates.

On the scripting part, you'll find lots of comments on Interactor codes. Interactor > Codes > Interactor Main Loop button will automatically open that method for you.

Default Animated

Default Animated interaction will be activated when gets in close with the Body type effector by it's "Default Animated Distance" value. No angles or range will be checked. Only Default Animated Distance and Obstacle Raycast (if it is selected) will be checked.

If this interaction object is a child of Vehicle (with VehiclePartControls) and Vehicle has a parameter in its animation state with the same name, it will get true to change Vehicle's animator's state.
If it's not a vehicle part, you can add an event to it's InteractorObject to change it's own animator's state or set something else as you wish.
More info for Unity Events.

Manual Button

Manual Button type interaction object's all children targets will be checked by their effector types for angles & range. If it is positive, the object will be activated. When it's respective effector checks fail it will go back to deactivated state.

While being activated, IK animation and interaction will start if player input comes for that object. And on the half of the IK animation (which means the IK part is on the target, like pressing the button moment) it's events would start. If there is a pivot object between InteractorObject and target and assigned into InteractorObject's pivot slot, it will rotate itself to the player when interacted. See FirstButton in the example scene to see how pivot works.

Then IK animation would continue its second half of animation which means it would return to its initial position.

Manual Switch

Similar to Manual Button interaction but more customizable. An interaction object can have InteractiveSwitch component to animate itself with its own events. You can add multiple events to InteractiveSwitch and every time you use this object, it will loop these events one by one. These events also have position and rotation values for creating different button/switch animations easily. For example, one event for a button-up position and one event for the down position will make you a quick button animation. You can create more than two states to animate the object. In the example scene, the console has both examples on "OnOffButton" and "WaveTypeButton".

Manual Rotator

Like Manual Switch, Manual Rotators have a component to animate itself. IteractiveRotator component can be added on the interaction object to rotate it constantly with mouse Y input. It also has it's own events to start when interacted but unlike Manual Switch, it can have only one event (But it's a list so you can add more stuff to start all together). Rotator has a direction setting and when

interacted it will rotate on that axis. Since it will rotate constantly until the interaction end, the player can control this object precisely as he/she wishes with Mouse input. Gumball crank and console rotators use this component. Manual Rotator also locks the camera Y-axis when used because it is using that input.

Manual Rotators and Switches can be used with AutoMover class (on interacted IK parts). So while they are animating the object itself, AutoMover can animate the target accordingly. All examples have this feature, you can inspect their child targets to see AutoMover settings.

Manual Hit

Manual Hit is in the experimental stage and will be improved to cover all IK parts. Right now it's just an example to show different kinds of procedural animations & interactions. And sometimes it glitches.

It is a bit more complex than other interactions because it has more stages. It works with the HitHandler component which works as a pivot object to turn itself to the player and handles hit methods.

When InteractorObject enters the player's trigger it will start to rotate it's HitHandler pivot object to relocate its child target (LeftHandFist in this case). It will constantly rotate itself and its child will orbit around to get the right position relative to the player.

On interaction input, it will create a new position for target based on hit settings on the HitHandler component. This new position is for pullback animation for hand and IK will move left hand bone to the new target. When reached (Animation paused), the target will move towards to its old position to create hit animation. And once it reaches to its old position, HitForce (on HitHandler) method kicks in to apply a force to InteractorObject based on the distance between target and effector positions. And also IK animation resumes at the same time to return its start position (where left hand stays before interaction).

Once updated, it will have more customization and easier use for all IK parts, so you will able to create better hit & kick interactions on objects.

Manual Force

Manual Force has two types of use; one for the player (ProtoMan example scene) and one for the truck (ProtoTruck example scene). It uses unused effector types (thighs and shoulders).

On the player, it will just create a force for its effector checked targets. So targets those who are in a good position to interact, will get the force.

On the truck, the back door won't be able to open when InteractorObject's children are in its effector

area. So it will be blocked by obstacles. And turrets will shoot child targets when they are in their area to protect the truck. And missed ones will start the windshield's animation to close it for more protection. It's just an example to show Interactor's use cases on non-human players. It could work as sensors to create awareness of surrounding objects. Of course, it's up to you, you can even use it for different needs.

To create new interactions by yourself, see the Tutorial Videos section for more information.

Touch Vertical

When the Touch Vertical type interaction object enters the trigger, it will be checked by a raycast in every 10 frames to check if the object or the wall is in a good position to interact. Raycast settings are on InteractorObject. Once raycast hit is in a good position for that effector, the interaction will start after the cooldown timer ends.

And it will continue to raycasting in every fixed update frame to move target with raycast hit. If raycast hits an out of range position or not that interaction object, the interaction will end.

Touch HorizontalUp

Touch Horizontal Up works pretty similar to Vertical one except it will raycast to upward direction with a forward offset to detect wall edge earlier.

Touch Still

Touch Still is quite simple, if InteractorObject's child target is in a good position for its effector (selected effector type), then it will start interaction as long as it stays in a good position. Once it's out, then the interaction will end itself.

Touch interactions are fully automatic and need no player input.

Distance Crosshair

Distance interaction objects will enter Interactor's object list (sphere trigger) when the crosshair is on them. There is a camera raycast to detect them and that raycast's distance can be set on Interactor > Raycast Distance.

There is no effector check for distance or angles. And they will be useable once selected on UI and used with a mouse click. They use different input keys because this way players can use them without conflicting themselves. For example, you can use truck doors while holding pick up items or driving the tricycle.

Climbable Ladder

Climbable is one of the complex interactions. It uses four effectors at once and they constantly change their targets when closer one exists. PlayerController controls the player's climbing state and changes the input to move the player up instead of forward while locking side movements.

Controller gets start and end positions from Interactor. Interactor calculates them from the lowest and the highest target positions for feet type effector targets. Also, it helps to relocate the player when climbing starts. You can see gizmo lines on the top and the bottom of the ladder when interacted to start climbing. Those positions are transferred to PlayerController.

See the Tutorial Videos section for a more detailed explanation. Climbing is a good example to show all hands & feet effectors in use. And it can be used for rock climbing, moving between cracks, etc. It will be improved with updates to create these kinds of interactions easier and with more freedom.

Multiple Cockpit

Multiple Cockpit uses all five of effectors at once. In its roots it's a basic enter vehicle interaction and interaction starts with body effector. Once the body target is in a good position for body effector, the object will enter the interactable state. And player input starts all effector's interactions at once. Also, player transform gets a sit parent to move with it. Player inputs are disabled and vehicle inputs get activated with this interaction. All these handled by PlayerController. There are two examples (truck and tricycle) at the ProtoMan example scene.

Self Interactions

To create self interactions, you'll need an empty gameobject attached to the player (spline bones would be better since they move with player idle animation accordingly). On that gameobject, add InteractorObject and as many as child targets to create self interactions. There should be a PathMover component on targets to create its animation (tween). Self interaction will get PathMover's odd's and select randomly to start that interaction according to its possibility (Only when the player is idle, checked on PlayerState singleton on the scene). Selected and started interaction will start its PathMover points. See examples on the player at example scene and Tutorial Videos section.

Pickable OneHand

Pickable objects have regular effector checks as like other interactions except for Y-axis. On InteractorObject, Z Only should be selected because these objects could stand on the ground while waiting for picked by. Z Only eliminates vertical checks in effector.

When it is in a good position for horizontal checks, it will be activated like others. Player input will start IK animation and if it has a pivot object (like Manual Button), it will be rotated with children targets to effectors for a smooth pick up animation. Otherwise, targets will cause deformed pick up positions for IK parts, especially if it is a sphere and rolling on the ground while picking up. So once the animation starts, it will constantly rotate itself to effectors until the IK part reaches to target. Then it will be a child of that IK part's bone. See examples and Tutorial Videos for more.

Pickable TwoHands

Pickable TwoHands also works similarly with OneHand, except picking up the object handled by itself on InteractorObject's update (LateUpdate). And also the main difference is Hold Point. Hold Point is an empty gameobject located as a child to the player and its position will be taken as a holding point for the picked object's center. The object will be picked up from its ground to that Hold Point. It was hand bones for OneHand objects but TwoHands objects need a different location to create illusion of holding them up. This Hold Point should be assigned into InteractorObject's Hold Point slot. See examples and Tutorial Videos for more.

Push Pull

Pushing (Pull is not ready yet) is similar to pick ups. It waits for both hand effectors to get in position to activate and when used with input, it will start both hand IK animations. Hand animations pause on the object and the object will be a child of the player to move with it. Also, BasicInput component gets the pushing state from PlayerState singleton and decreases the forward movement amount by half. That makes player movement slower to create the impression of pushing or pulling an object.

Once this interaction done with upcoming updates, it will also affect body (and maybe feet) to create a more realistic pushing pose for player.

Interactor Components

Interactor

Interactor's main script and used on the player's itself. For now, there should be only one Interactor in the same scene. It runs an update loop for all of its effector in FixedUpdate to check their states for interaction object's children targets. And sends interaction start calls for InteractorObject & IK components. Has some main options for Interactor and all effector settings.

Logo - If you click on the logo, it will minimize and give you more space on Inspector.

Right Top Buttons - These are shortcuts for documentation, forum thread, sending a message to me and the store page. All of them have descriptions of them as a tip. These buttons disappear with the minimized logo.

Effector Tabs - Every effector has it's own settings. You'll see selected effector's gizmo on SceneView. You can add, delete, copy these settings and paste them onto another effector. Also, you can enable (Green) and disable (Red) effector on the power button. A disabled effector won't work as it never exists.

Name - You can give any names for your effectors. If you leave blanks between the words you gave, they will get proper line positions on effector tab buttons. Long words will stay center aligned.

Effector Type - Interactor has player parts for a biped humanoid by default. If you want to change these types, you can add your own effector types by coding. See scripting API for more information.

Effector Position - Effector's position offset for added sphere trigger's center. (0, 0, 0) position means it's on the center of the sphere. Position values can't go out of the sphere radius because Interactor checks only the targets inside this trigger. Also, effector can't get close edges within its own range distance. It will automatically arrange itself around the trigger edges if you push it more. See Effector Range Distance.

Horizontal Angle - Effector's horizontal angle value. Interaction object's targets should be in these angles to be positive. You can see angle values on SceneView gizmo with white small texts.

Horizontal Offset - Set horizontal angles in any horizontal direction. It shows the middle with a thick white line on SceneView.

Vertical Angle - Same for the vertical axis.

Vertical Offset - Same for the vertical axis.

Effector Range Distance - Effector's range to start an interaction. It can't be bigger than the sphere trigger because it makes no sense. If the effector's range touches the sphere trigger's edges, it will be pushed to center to fit inside automatically. This value as another sphere inside the sphere trigger. If the effector position won't move closer to the edges, it means it's edge touches the edge on somewhere. And it will slide over the edge if you try to move.

These settings work only for this effector. Every effector has it's own rules. You can copy these settings with Copy Button and paste them onto another effector to set its values easier.

Other Options

>**Spawner Window** - Opens InteractorTargetSpawner window. Also on Window>Interactor>Interactor Spawner Window opens too.

Self Interaction - If you wish to use self interactions, you should assign the parent object here. More info on Interaction Types > Self Interaction.

Gizmo Opacity - Set SceneView gizmo's opacity value.

Raycast Distance - Distance interaction range. Distance type objects can be interacted by crosshair within this range. Raycast will shoot only this long.

Codes - Shortcuts for opening Interactor effector loop in the main script or to expose more properties on the inspector in the editor script. So you can easily open main and editor scripts from here.
Debug View - Main switch for turning on and off SceneView gizmos. Interactor gizmos stay on SceneView as long as the debug switch turned on, even Interactor object won't be selected. So you can work easier.

InteractorTargetSpawner Window

InteractorTargetSpawner is the prefab holder window and saves those prefab lists in specified project folder as ScriptableObject. Saves are called as SaveData and can be used for different setups. For example, you can create a SaveData for a specific need and change save file temporarily with some other SaveData to switch back later. Active SaveData showed at the top of the window and can be replaced by any time. But you always need a SaveData to use Spawner window & SceneView right click menu. When you add your first effectors to the Interactor, you will be asked for creating one. Don't forget, when you close a SaveData by clicking red X button, data will be saved as it is before closing it down.

Also you'll see all the effectors you have on Interactor. If you want to add more than one prefab for each effector tab, you need to attach main prefab first. Then you can add other prefabs all together by dragging them into the drag area (No need to worry about repetitive ones, it will automatically check for duplicates). You can remove any of them by clicking its red X button. If you want to remove all prefabs, just remove the main prefab, it will wipe out the list.

Spawner window has some options for the designing stage. Once you have prefabs on it, SceneView right click menu will be activated. Just select any prefab you want and it will be spawned as a child of that gameobject, on that clicked spot. You have four rotation modes for the spawning. Default prefab rotation, surface normal rotation, SceneView camera direction and SceneView camera direction for Y-axis only.

Exclude Player layer on raycasts - This will ignore player's triggers and colliders when raycasting in SceneView. Otherwise, it could become quite annoying while working around the player gameobject because it can spawn target prefabs on Interactor's trigger.

Add required components - If enabled, InteractorObject component will be added to selected object(If it has not yet) when targets spawned. And interaction type will be set as Manual Button for a quick test run.

Right click time (ms) - The maximum time needs to pass before not moving the mouse cursor when right clicked in milliseconds. If you move mouse cursor after this time passed, right click menu won't show up. To prevent conflicts for default SceneView right click behaviour since it's used for the camera rotation. If you're slow clicker, set higher values and vice versa.

Besides of the right click menu, also Interactor Effector window on SceneView gets enabled by opening the InteractorTargetSpawner window. You'll see an Interactor toggle button on top left corner of the SceneView when Spawner window opened. That toggle activates the Effector Settings menu on SceneView. Spawner window doesn't need to be seen all the time, just put it on any editor tab to keep the Effector Settings menu enabled. Once you close Spawner window, Effector Settings menu will also close down, except the full screen mode.

InteractorIK

InteractorIK is based on Unity's IK and right now works just for hands & feet. It works as Two Bone IK system, which means if you move the hand, it will rotate it's forearm & arm accordingly. But it won't move or rotate further into the body. Final IK is Full Body IK system so it can pull or push other bones too.

IK Parts should be a maximum of 5 elements because it has only 5 part types right now. Set each element's part type.

Weight - Shows this IK part's current weight. It's just for debugging.

Rotation - Bone's rotation should be set as well as the position? If disabled, only the position will be set to target's.

Target Transform - Shows this ik part's current target. Debugging purposes.

Duration - Duration for each interaction in total. 2 seconds means 1 second for reaching the target, 1 second for returning to the default position.

Multiplier For Second Half - Multiplier for returning phase. If you set this to 0.5f then instead of 1 second, it will move twice in 0.5 seconds. So total interaction duration will become 1.5 seconds. Only the second part will be affected.

Match Children Bones - If you want to match all bone rotations activate this. A hand needs to match it's children bone rotations to get the same exact pose of its target.

InteractorObject

InteractorObject is the main component for interaction objects (besides Final IK's InteractionObject for Final IK version). It will evolve in upcoming updates but for now, it handles all interaction types. Some settings may not exist in both versions (InteractorIK & Final IK).

Interaction - Interaction type, if you don't choose anything this field will be highlighted with red color. For more information about the interactions see InteractionTypes section.

Pause On Interaction - Select if you want body parts to stay on target when interacted (like touching

or holding an object). Otherwise, IK parts will return to their initial position when reached to target.

Interruptible - Select if interaction can be interrupted by another interaction. Same body part can't leave the uninterruptible interaction and start a new one.

Ease Type - Rate of change of velocity/time for IK animation. This will be changed with animation curves for more freedom in upcoming updates. For now, you can see a great cheat sheet for used curve types here.

Z Only - Deactivates effector check for Y-axis. So the interaction object's target height won't matter. Useful for ground objects for picking them up.

Default Animated Distance - Activation range for Default Distance type interaction. Automatically opening truck doors for example, they won't work if this condition has not met.

Object Raycast - Interaction object's target will be checked once with raycast before starting interaction to be sure there is no obstacle between the effector & the object. Useful for doors at downstairs or upstairs.

Other Targets Root - If some of the target children are not parented by this object, you can select the other parent that has them all.

Pivot - Buffer object between the interaction object and its targets to rotate them towards to effector when interacted. Pivot objects are important, especially for buttons. Pivot calculation activates automatically when any object assigned to this property.

Pivot X, Y, Z - Deselect if you don't want pivot to rotate any of these axes.

Outline Mat - If Outline material (with Outline shader) is this object's material, leave empty. If it is another object, assign here.

Touch Settings
Touch Height - Height setting for Touch Vertical interaction (Touching walls, etc).

Touch Forward - Forward offset for Touch Vertical interaction.

Touch Rotation - Rotation offset for Touch Vertical target.

Touch Ray Length - Touch Vertical works with raycast to detect its wall. This is useful to adjust ray's length to prevent it from hitting too far with low angles.

Touch Lerp - Lerping target position between wall and effector to make fine adjustments in some cases.

Touch V Cooldown - Time needs to be passed for Touch Vertical interaction to start. Instant interactions can look bad because the IK part will try to interact constantly.

Touch Horizontal Forward - Forward offset for Touch Horizontal Up target.

Touch Horizontal Right - Left/Right offset for Touch Horizontal Up target.

Touch Horizontal Ray Length - Touch Horizontal raycast distance.

Touch H Cooldown - Time needs to be passed for Touch Horizontal Up interaction to start.

Two Hands Closer - Lerp between Pickable TwoHands targets between object center and player position (since pick up objects mostly stay on the ground while the player position is on the ground too).

Hold Point - Assign a hold point which should located under the player gameobject as a child. Needed for Pickable TwoHands interaction.

Info - Just for example scenes but could be useful. Text mesh object slot for interaction object for activating/deactivating while being interactable and not. Also instead of texts, any type of gameobject works the same (activates/deactivates) if assigned here.

Unity Event - You can create Unity Events for almost every interaction type. When interacted, these events will be called.

InteractorTarget

InteractorTarget is a target script for responsible of showing gizmos of bones and registering themselves to parent InteractorObject as a target. It has Effector Type settings to tell Interactor which effector's target when interacted with its parent object. Also has override settings to override the effector's own rules. It's just for the range now but angles and other offsets will be included to be overridden in upcoming updates.

The second theme is for changing color themes for target bone gizmos to see better in some cases when background colors are similar.

InteractorTarget is exclusive for InteractorIK. To override effector rules, you can use InteractorOverride component for Final IK.

InteractorOverride

InteractorOverride is exclusive for Final IK version to override some effector settings for that specific target. It's useful when you set the effector's range for 0.5 but want some objects to be interactable with more or less distance. Other effector settings will be added in upcoming updates.

BasicUI

BasicUI written just for example scenes to show Interaction selection with UI and inputs. Also enables a crosshair on GUI. When added in the scene, all interactions in the trigger will be showed and will be selectable to interact except Self Interactions. If selected & used interaction is not useable at that position, list selection will be reset. BasicUI can be added to any scene object.

BasicInput & BasicInputInteract

BasicInputs are modified Unity Standart Asset input classes and should be on player gameobject. It will handle inputs and send them to Interactor when pressed. Interactor will decide if that call will start an interaction or not depending on effector checks.

All properties are for just debugging purposes.

PlayerController

PlayerController taken from Standart Assets and modified more than mildly. It handles entering/exiting vehicles, extra move abilities and climbing. It has a default setting on it.

PlayerState

PlayerState will be removed in newer versions of the Interactor. Right now some information about the player passed through here between BasicInput, Interactor, Vehicles and PlayerController because it's a singleton and easily accessible from any class. It holds states of the player.

FreeLookCam

This is a regular camera script to control third-person camera. It's a bit modified to lock the cursor with right click easily. And also has a tiny method for locking Y-axis when interacting with rotators.

VehicleBasicInput & BikeBasicInput

Basic Standart Asset scripts for vehicle input. The vehicle has some extra methods for controlling its windshield & back door animations.

VehicleController & BikeController

Mildly modified Standart Asset vehicle scripts. The vehicle has accelAmount and steerAmount for Pedals and SteeringWheel to tell their rotating amounts. The bike has a method for pedals to look always up when rotating with front wheel on FixedUpdate().

VehiclePartControls

VehiclePartControls is for vehicle animation examples and gets every interaction object on the vehicle and caches their animation strings as ids then store them on their InteractorObject classes to use later when interacted. If that part has an animation state on the vehicle's animator, then it will activate their animations when interacted. Some animations like Elevator, Back door and windshield have their animation events on their animation files to automatically deactivate themselves.

TurretAim

ProtoTruck example scene has turrets to show a non-human player example. Turrets and their lights aim to their targets with different speeds and fire in this script. Also shot targets get their push force from here.

Pedals

Pedal rotation class gets accelAmount from VehicleController for rotation amount. Both pedals have it.

SteeringWheel

Steering wheel rotation class gets steerAmount from VehicleController for rotation amount. So when rotating the vehicle, the steering wheel rotates accordingly with set limits.

HitHandler

Handler class for Manual Hit interaction. It rotates itself when enters the player sphere, repositions its

target for a hit position(pulling hand back), moves the target back to original position on the object and deals force when hit happens. It works like pivots and positioned between target and object as a buffer object in the hierarchy.

Hit Obj - InteractorObject for hit interaction.

Y Offset - Height of the target for pullback reposition.

Distance Percentage - Lerp between the initial target position and pull back position.

Angle - Angle between the initial target's position and effector's position.

Hit Force - The force to be applied when hit by body part to target's parent (InteractorObject).

InstantiateRandomAreaPool

This is a really useful class for spawning prefabs from a determined place randomly in the scene on runtime. Also pools the prefabs on start for maximum performance. Set its sizes with transform scale and pooled prefabs will be randomly spawned in that boundaries. They can be spawned by pressing the Enter key or calling SetPooledPrefabActive(int count) method in script or events with count to spawn per call. Gizmo will show the spawn area and prefab array is for prefabs to spawn. Max prefab count is pool count and that amount of object will be pooled on start.

BasicOnOff

Very basic on off script for setting assigned gameobjects active or inactive. It has an array for gameobjects and Toggle() script for calling from code or events.

DigitalDisplayTextFloat

Sets Text Mesh values with a prefix, multiplication and determined decimals. It is used for the console terminal's digital frame to show graph values.

InteractiveRotator

Rotator class is for rotating interacted objects with mouse Y input. It has settings for adjusting the rotation amount and style. Console rotators and Gumball crank uses this to showcase more interactable ways in example scene. Set the direction and it will rotate constantly with given settings during interaction time with mouse Y input. Will be improved on upcoming updates.

InteractiveSwitch

Switch is also rotating objects when interacted but also can change positions too. It works one time when interacted, unlike rotators. It has its own events and you can add more than two events for each use and it will cycle these events. So this way you can have more complex animations on the object. It is used for OnOffButton and WaveTypeButton on the console in the example scene.

WaveGraphLineRenderer

Uses LineRenderer to draw mathematical functions like Sine, Square and Triangle wave types.

AutoMover

Used for changing rotation & position values with determined duration and ease type. The ability for tweening targets in runtime creates a new set of possibilities while interacting objects. This class will be improved on upcoming updates to create more easy and customizable movements.

PathMover

A bit different version of AutoMover with multiple points to loop for the position. It is used for Self Interaction targets and has an odd property to set it’s happening chance.

Support

You can reach me with several options. All of them will be responded as soon as possible (mostly on the same day).

You can follow the official forum thread on Unity Forum and reply a post so others can read too.

You can start a private conversation with me on Unity Forum.

And you can send an email to me.