Flexible Interaction System in *Archive: DZ* by Brandon Ludford

Note: I use blueprintUE later on in the document for the sake of allowing you to follow along with the relevant blueprints I am referring to, please not that blueprintUE is not perfect and will show blueprint paths as not straight with some overlapping nodes. This is **not** how it appears in the editor, so please take the layouts/ organisation of the example blueprints with a pinch of salt, and maybe some paprika.

Overview

The interaction system that I'm using in *Archive: DZ* was built for specific needs in the game. I needed a simple way for the player to be able to interact with their environment by walking up to interaction points, and either holding or tapping the interaction key to commence an action of some-sort.

Key Objects

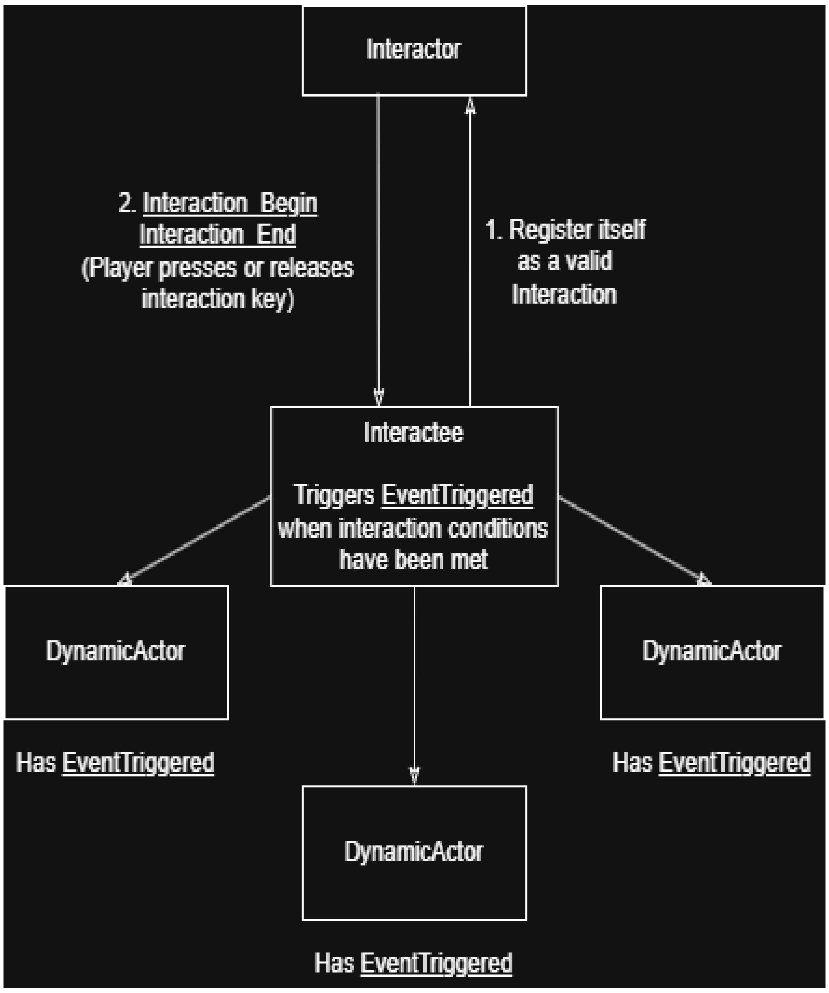
The system utilises a few actor types in order to function correctly, an **Interactor**, **Interactee**, and **Dynamic Actor.**

The definition of these actor types are as follows:

- Interactor
  - The actor commencing the interaction. Defined by implementing **AC_Interactor**.
- Interactee
  - The actor that is being interacted with. Defined by implementing **AC_Interactee**.
- Dynamic Actor
  - The actor that does something after being messaged by an **Interactee**.

Key Objects – Flow

The main flow between these three actor types is illustrated below.



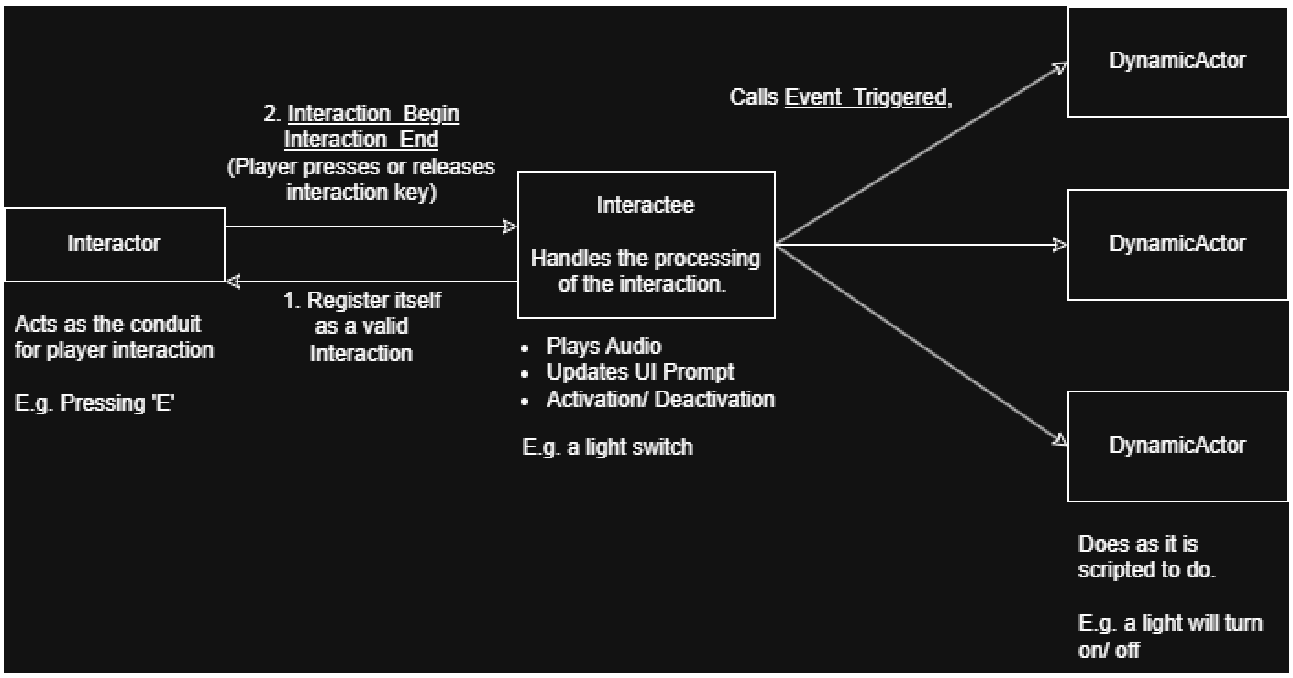
Interactor
2. Interaction_Begin
Interaction_End
(Player presses or releases interaction key)
1. Register itself as a valid Interaction
Interactee
Triggers EventTriggered when interaction conditions have been met
DynamicActor
Has EventTriggered
DynamicActor
Has EventTriggered
DynamicActor
Has EventTriggered

The flow is controlled within each of these actor types. The **Interactor** is the player, as such they lay at the first step of the interaction system, if the player doesn’t want to interact with the **Interactee**, then a majority of the scripts will not trigger.

When the **Interactor** begins their interaction, they tell the **Interactee**, then the Interactee is responsible for processing whether or not the player has completed the interaction. Due to interaction being unique depending on the context of the **Interactee**, the data of the required interaction, hold-time, repeatability etc, is held within the **Interactee**, or more accurately, in the Interactee’s **AC_Interactee**.

Data Flow

While we're on the topic of data, I will illustrate the data that is being shared between the different actor types.


Interactor
Acts as the conduit for player interaction
E.g. Pressing 'E'
2. Interaction_Begin
Interaction_End
(Player presses or releases interaction key)
1. Register itself as a valid Interaction
Interactee
Handles the processing of the interaction.
• Plays Audio
• Updates UI Prompt
• Activation/ Deactivation
E.g. a light switch
Calls Event_Triggered,
DynamicActor
DynamicActor
DynamicActor
Does as it is scripted to do.
E.g. a light will turn on/ off

So when an Interactor is near an **Interactee**, that **Interactee** will provide it's **AC_Interactee** to the **Interactor**, that way the Interactor can compare each **Interactee** and determine the most relevant one and select it as the Interaction Target. The reason why this works well is because outside of the Interactor searching an array of potential interaction targets, nothing runs autonomously, so I can have 1000s of **Interactees** in a level, and performance does not drop, (unless I attach an extremely dense Niagara Emitter to each of them, but that's a totally different issue).

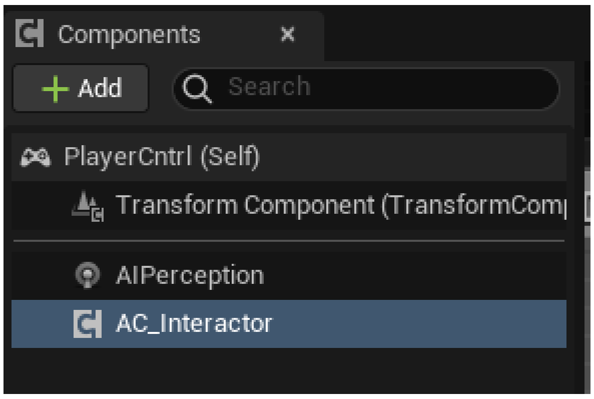
Blueprint implementation

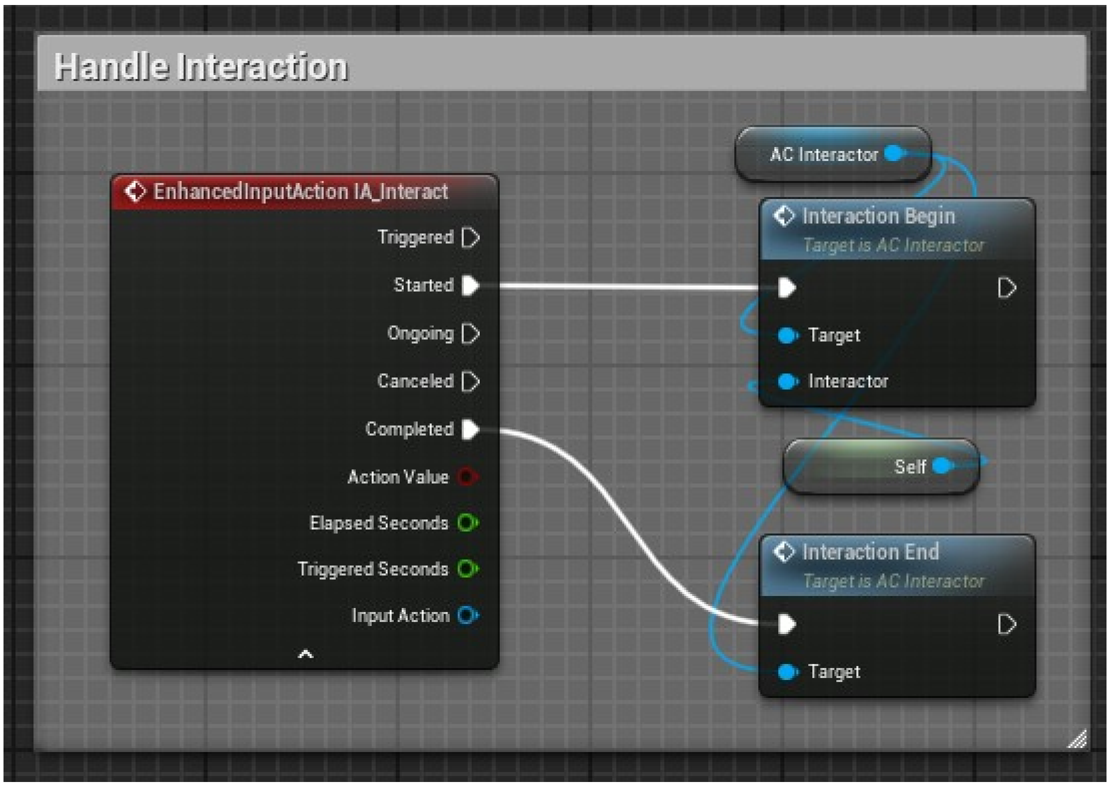
Right, so we've covered the theoretical side of the Interaction System, now I can talk about the fun stuff. The…

So I've covered the different actor types, but I've only briefly touched on the way to define an actor as one of these types. If you've got better short-term memory than me, you'll remember that **AC_Interactor**, and **AC_Interactee** are the two ActorComponents that you will use to define what is an **Interactor** and **Interactee**.

Now the reason for this, is because it makes it incredibly easy to convert an actor into something capable of interaction.

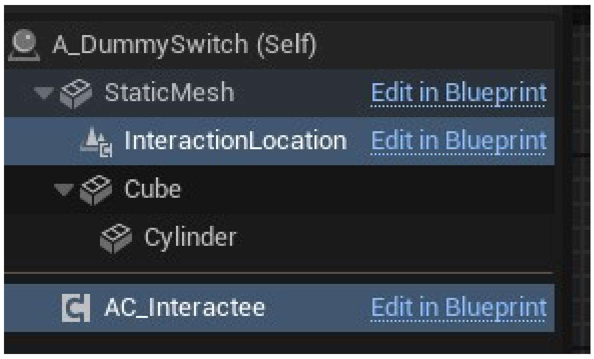
For an **Interactor**, this is the setup required (outside of parameters):

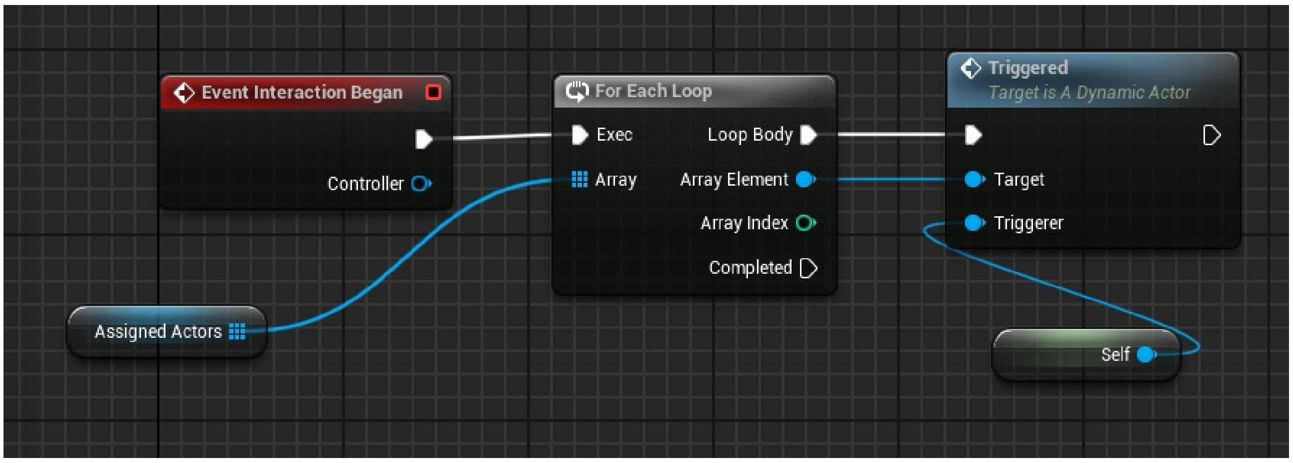




Interaction Begin and Interaction End, trigger these two events within **AC_Interactor**: [blueprintUE]

For an **Interactee** this is the setup required (outside of parameters, again):


A_DummySwitch (Self)
StaticMesh Edit in Blueprint
InteractionLocation Edit in Blueprint
Cube
Cylinder
AC_Interactee Edit in Blueprint



InteractionLocation is used to spawn the collisions that are used by the Interactor, to recognize that it is near a potential interaction target (more on than later).

Interactor Setup

So firstly I want to cover how **AC_Interactor** sets itself up.

And about that, is doesn't require any setup of itself. From the get go, it starts searching the array of TargetInteractionTargets (Array of **AC_Interactee**) and uses the one that the Interactor is most facing towards (closest DotProduct to 0) to determine what **AC_Interactee** to use. [blueprintUE for reference]

Interactor Receives (and loses) a Potential Interaction Target

So what happens when the Interactor receives a potential interaction target?

[blueprintUE for reference]

Well firstly the **Interactor** checks to see if the **Interactee** is already a potential target, if not, then it adds it to the array of potential Interactions for determine the best interaction target (see previous section)

For the sake of ease, I'm not going to bother putting in a separate section for what happens when an **Interactee** is no longer relevant, it's more or the less the inverse of what happens when a potential target it found. [blueprintUE]

For an overview of the code that determines the best **Interactee** target, here is the overview: [blueprintUE]

Interactee Setup

The **Interactee** is responsible for registering itself as a valid interaction for the player, deciding whether or not the player has 'Interacted enough" to execute the 'InteractionFinished' script.

BeginPlay – Collisions

[blueprintUE]

When the **Interactee** blueprint first executes, it goes through a process of finding a spawn location for it's collisions. It determines this either by searching for a SceneComponent named "InteractionLocation", or if that cannot be found, then it just utilises the RootComponent of the Interactee

Listening for Overlaps (Begin/End)

After the creation of the InnerZone collision, the script is bound to the Overlap and EndOverlap events. It uses these to recognise whether or not a valid Interactor is nearby.

[blueprintUE]

The Linetrace is used to check to see if there is a line of sight to the potential Interactor, if the trace is blocked by the Interactor, then we have line of sight to them, if not, then we do not care to get the AC_Interactor from that actor, if so, then we want to register ourselves as a potential Interactee Target.

If the OverlappedActor leaves the InnerZone, we want to remove it as a potential Interactor, and deregister it from said Interactor.

[blueprintUE]

Interaction Flow – Tap, Hold, Skill Check

[blueprintUE]

The last section that I feel is relevant to cover in the Interactee, is how it handles the Interaction_Begin and End calls that it receives from AC_Interactor

1a) Tap interaction

Tap interaction is very straight forward, when the Interactee recieves an interaction it will automatically trigger the InteractionBegin event, for the owning actor, if it is repeatable, then start the cooldown before it can be interacted with again.

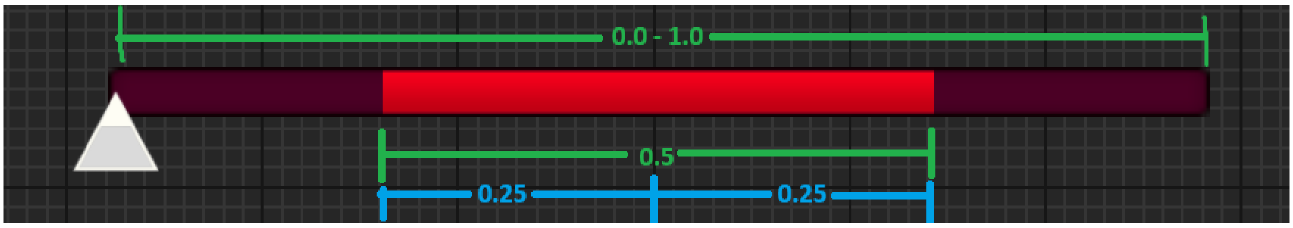
1b) Hold interaction

Hold interaction sets a timer, the duration of the timer is set to the amount of time that is needed before the interaction is considered complete. So when the player releases the interaction key, if the timer is still active, then the interaction is cancelled, otherwise, the timer fires off an event that clears the timer, and stops the interaction, and marks it as complete.

1c) Skill check

Skill check presents the user with a UI mini-game, in which they have to press the interact key when an indicator is pointing towards a coloured section, if the user successfully does it, it progresses the interaction, until it is complete.

I am going to be brief in regards to my explanation on how the Interactee determines whether or not the user has pressed interact at the correct time.



There is a progress bar within the UI that scales out from the centre.

There are two parameters that are being used by this widget to determine the correct interaction zone.

1. Range - Float
2. Difficulty – Int

Range determines the total size of the correct input zone

Difficulty is the speed of the arrow indicator

The arrow is purely a visual representation of a float called “TargetFloat”, and oscillates between 0.0 and 1.0, and when the user presses the interact key, this float is checked, and if it falls after the MinRange ((Abs(0.5 - (Range / 2)))) and before the MaxRange (0.5 + (Range / 2)), then the interaction is correct. Note: We use 0.5 because we are scaling the zone from the centre of the progress bar (which will always be a float of 0.5)

Let’s use an example.

Range is 0.25

MinRange is 0.325

MaxRange is 0.675

Right, now that we’ve covered how we get to the correct arrow position for the UI element, I’ll skim through the rest.

AC_Interactee will cycle through to the next skill check (if added), and if we are already on the final skill check, will fire off InteractBegin that then passes the interaction to the completion logic. It’s a little messy, but that’s because this is a remanent of the prototyping phase of Archive:DZ. It’s been kept in for archival reasons, and in case I have a need for it later in the project’s development.

Closing Thoughts

After an interaction has been completed, you can trigger events in actors that are assigned to the Interactee the event used depends on the interaction method used. [blueprintUE]

Overall this interaction system while it is some-what basic, was built out of a need for a generic system that allows the player to interact with their environment.