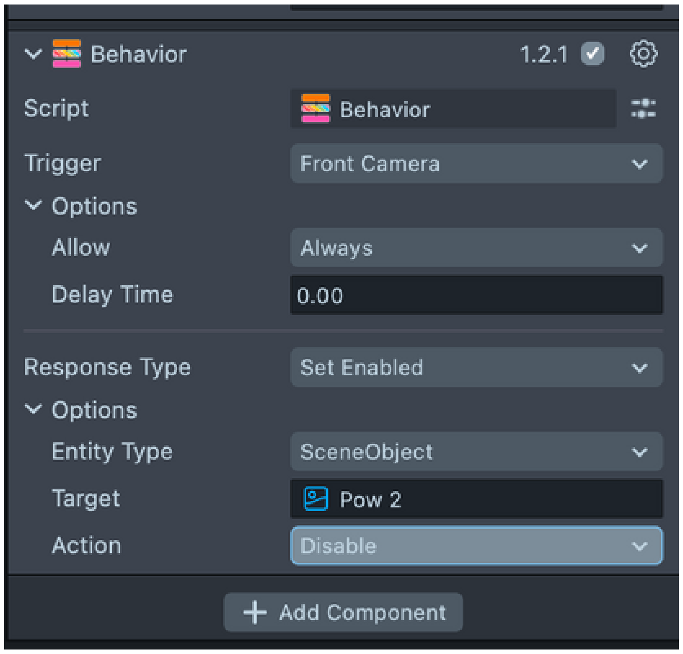
Make it YOUR Experience!

AR lenses can feel like magic, but that magic is actually a series of smart decisions made by you, the developer. Take this time to customize what you want the experience to be for the user when they activate the Front or Rear camera.

Front Camera
When a user is looking at the Front Camera (Selfie Mode), we want the focus to be on them. An example of customized settings can be as shown below:

- *Trigger*: Front Camera
- *Allow*: Always
- *Delay Time:* 0.00
- *Response Type:* Set Enabled
- *EntityType*: SceneObject
- *Target*: Pow 2
- *Action*: Disable

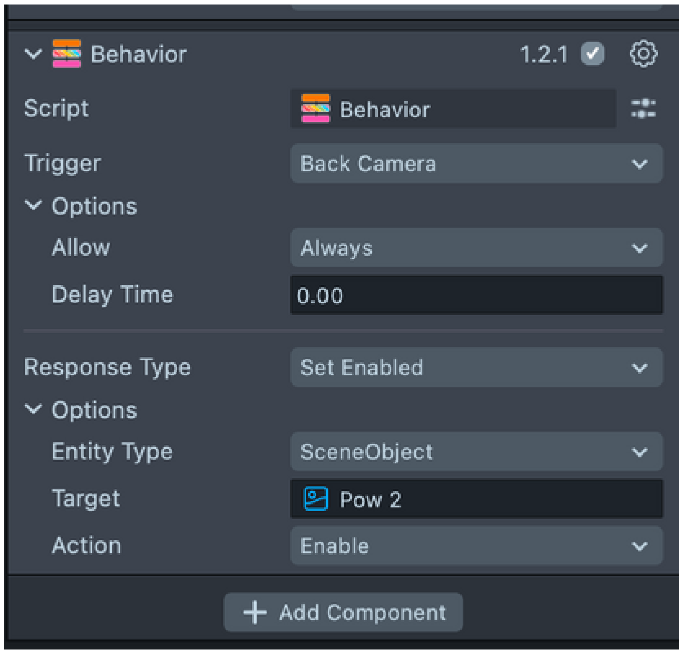


Behavior 1.2.1

Script	Behavior
Trigger	Front Camera
Options	
Allow	Always
Delay Time	0.00
Response Type	Set Enabled
Options	
Entity Type	SceneObject
Target	Pow 2
Action	Disable

Rear Camera
When the user switches to the back camera, we want the "Pow 2" sticker to reappear so they can place it on a desk, a wall, or even their pet! We are going to write a rule that says: "When the back camera is on, wake up the art."

- *Trigger*: Back Camera
- *Allow*: Always
- *Delay Time:* 0.00
- *Response Type:* Set Enabled
- *EntityType*: SceneObject
- *Target*: Pow 2
- *Action*: Enable



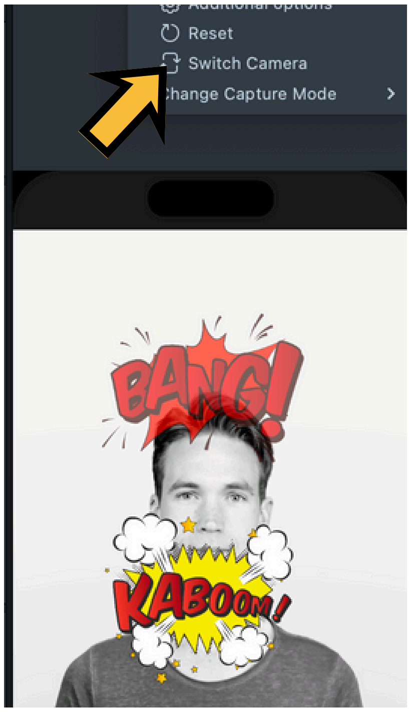
Behavior 1.2.1

Script	Behavior
Trigger	Back Camera
Options	
Allow	Always
Delay Time	0.00
Response Type	Set Enabled
Options	
Entity Type	SceneObject
Target	Pow 2
Action	Enable

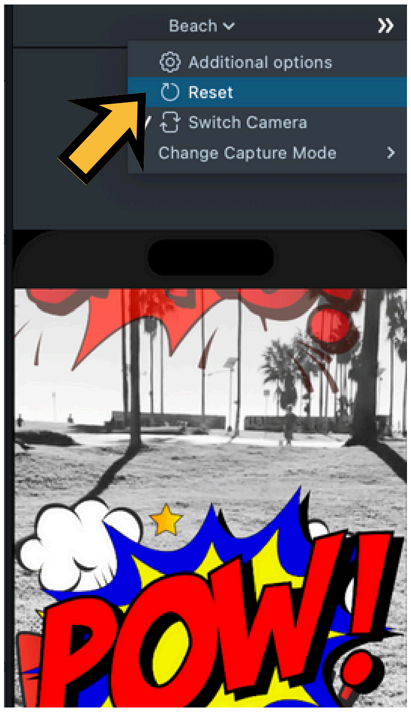
Test, Test, Test!

The Preview Panel allows us to see if the animation is applied.

Switch Camera
Change the camera so we can see the Back Camera preview.



Reset
Reset your filter if the Pow image is not showing.



Continue to customize the experience by setting images to be enabled (appear) or disabled (disappear) for each camera setting.

Knowledge Check

Custom Scripts & Camera Logic

Instructions: Answer the following questions to demonstrate your understanding of how to integrate your own code and how to make your Lens react to the user's camera.

1. Bringing Your Code to Life

You have written a custom JavaScript file in your Asset Browser. What is the specific process to make that code act as the "brain" for a 3D object in your scene?

A. Double-click the script in the Asset Browser and hope for the best.
B. Select the object in the Objects Panel, add a Script Component in the Inspector, and then link your .js file to that component.
C. Drag the script file directly onto your computer's webcam.
D. Type the entire code into the name of the object.

2. The Hardware Handshake

Why is it a "best practice" for an AR developer to test their Lens using both the Front and Rear camera triggers?

A. Because the computer forgets the code if you only use one camera.
B. Because users interact with their world (Rear) differently than they interact with themselves (Front/Selfie), and your Lens should feel natural in both modes.
C. To make the Lens file size larger for the App Store.
D. Because the Rear camera only sees in black and white.

3. Logical Switching

Imagine you want a "Robot" character to appear when the user looks at the world (Rear Camera) but disappear when they take a selfie. Which Action would you set in your Behavior Script when the Front Camera is the trigger?

A. Enable
B. Delete
C. Disable
D. Export

4. Short Answer: The Power of State

We learned that using the Set Enabled response allows us to hide objects without deleting them. Why is this better for the user's experience than trying to create or destroy objects every time they flip the camera?

Answer: ____

(Hint: Think about speed. Does it take longer to wake someone up from a nap or to build a whole new person?)

Learning More About JavaScript

Introduction to JavaScript in AR
In the world of STEAM (Science, Technology, Engineering, Arts, and Math), JavaScript (JS) is the language that bridges the gap between a static piece of art and a living, breathing digital experience. While tools like the Behavior Script provide pre-built "blocks" of logic, JavaScript allows you to write the actual rules of the world from scratch.

What is JavaScript?
JavaScript is a high-level, text-based programming language used to make web pages and applications interactive. In Augmented Reality, it acts as the "nervous system" of your Lens.

- It listens for inputs (like a screen tap or the camera flipping).
- It processes data (like calculating the distance between your phone and a 3D object).
- It triggers reactions (like playing a sound or starting a "POW" animation).

The Relationship with Lens Studio
Lens Studio is designed to be accessible for creators of all levels. It uses a Component-Based System, where you can attach "brains" to your 3D objects.

- *The Interface*: For beginners, Lens Studio offers checkboxes and dropdowns to manage simple behaviors.
- *The Script:* For advanced creators, it provides a Script Editor where you can paste code like powAnimated.js. This allows the software to execute complex, custom instructions that aren't available in the standard menus.

While Lens Studio utilizes standard JavaScript (ES6), it operates within a specialized environment. It isn't just "general" code; it is an API-driven language designed to talk directly to the hardware of your phone and the physics of a 3D engine.

The Flavor of JS: Scripting vs. Programming

In Lens Studio, you are technically scripting. This means your code is waiting for "Events" (like a face being found or a screen tap) and then executing a reaction.

1. The Scripting API

Lens Studio uses a set of built-in "names" called an API (Application Programming Interface). When you see words like script.getSceneObject() or Component.Animation, you are using the specific vocabulary Lens Studio understands.

- Relationship: If JavaScript is the "grammar," the Lens Studio API is the "dictionary".

2. Component-Based Syntax

Instead of writing a program from scratch, you write scripts that "attach" to components.

- Example: To move a comic sticker, your JS doesn't just say "move." It says: script.getSceneObject().getTransform().setLocalPosition(...).
- STEAM Connection: This is Systems Engineering. You are learning how to navigate a hierarchy of parts to reach a specific goal.

Common Syntax Learners Will See

The "Dot" Notation

This is how we "drill down" into an object to find its properties.

- Code: image.alpha = 0.5;
- Analogy: This is like an address: Street . House . Room. You are telling the computer to look at the image, find the alpha (transparency) setting, and change it.

Event Listeners

Scripts in AR spend most of their time "listening".

- Syntax: script.createEvent("TapEvent");
- STEAM Connection: This is Event-Driven Programming. It teaches students that technology is reactive and depends on human interaction.

Deconstructing powAnimated.js Syntax

In the code we are using, you'll notice specific Babylon.js syntax, which is a specialized library for 3D web graphics:

1. async function: This tells the computer that some things (like loading a large "POW" image from the internet) might take a second, so it should wait before moving to the next step.
2. billboardMode: This is a specific property name that tells the math engine to keep the image facing the user.
3. onPointerDown: This is the "Listener." It’s a specialized version of a "Tap" that records exactly where in 3D space your finger landed.

Understanding the Script

To help you master this new level of custom scripting, let’s look at the "DNA" of your code. You’ll notice that while the language looks different, the logic follows the exact same patterns we used with our checkboxes and folders.
Here are the four big parallels between your previous workbook exercises and this JavaScript file:

Parallel 1: The "Digital Inventory" (Arrays)
In our earlier lessons, we selected one image at a time for a sticker. In this script, we use an Array to hold a whole collection at once.

- Workbook Concept: Selecting a "Texture" in the Inspector.
- Script Parallel: The imageURLs list.

Parallel 2: The "Smart Toggle" (Modulo Logic)
Remember how we used a Toggle to switch an effect on and off? This script uses a mathematical trick called Modulo (%) to toggle through your images.

- Workbook Concept: The "Toggle" checkbox in the Behavior Script.
- Script Parallel: (currentImageIndex + 1) % textures.length.
- The Logic: This keeps the "Comic Pop" looping. It says: "Go to the next image, but if you hit the end of the list, jump back to 0." It’s a never-ending loop that keeps your Lens from crashing.

Parallel 3: The "Always-On" Behavior (The Render Loop)
Earlier, we looked at "On Awake" triggers. This script uses a Render Loop, which is like an "On Awake" trigger that never stops firing.

- Workbook Concept: Continuous rotation or "On Awake" triggers.
- Script Parallel: scene.registerBeforeRender(() => { ... }).
- The Logic: This is the "Heartbeat" of your code. It checks 60 times every second: "Is the user tapping? Should I be fading out right now?" This constant checking is what makes the "POW" animation look so smooth.

Parallel 4: Local vs. World (Spatial Mapping)
We spent a lot of time talking about Local (relative to a parent) and World (the 3D grid). This script uses "Picking" to bridge that gap.

- Workbook Concept: The "Local" vs. "World" checkbox.
- Script Parallel: comicPlane.position.copyFrom(pickResult.pickedPoint).
- The Logic: This is like the GPS coordinates we discussed. When you tap your screen, the script finds the "World" coordinate of your finger and snaps the "Local" image directly to that spot.