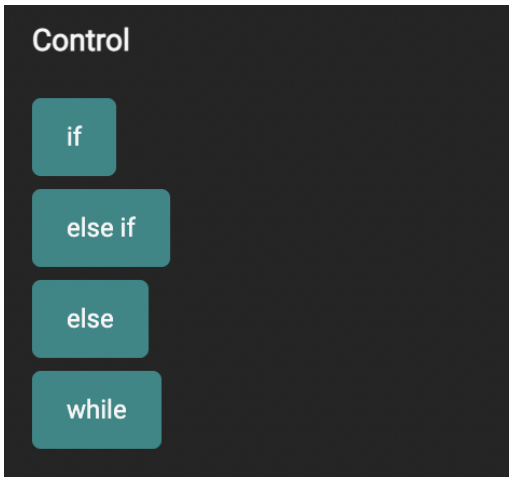
Events

Control

Events > Control



Conditional commands that control whether or not something can happen.

if

Events > Control > if

Execute the subsequent commands if a defined condition is met.

Appearance in Composition Pane

Description

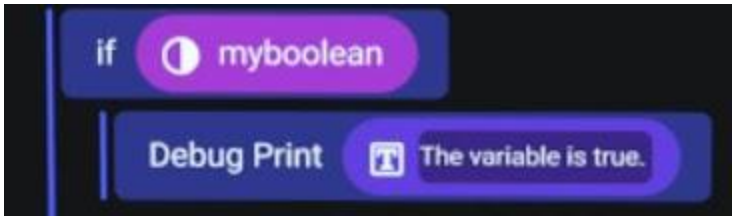
The commands indented below the **if** line will execute if the **condition** evaluates to true. Only booleans are accepted in an **if** block, however, because logic operators return a boolean, they are also accepted in an **if** block.

Parameters

boolean

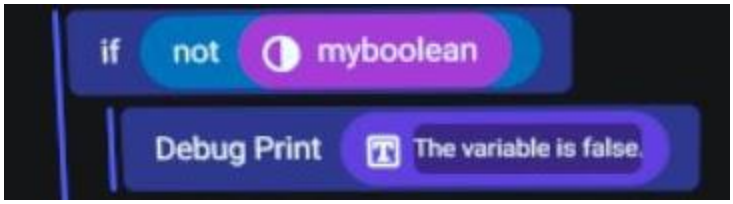
The if block executes the codeblocks indented below if the boolean is true.

Example 1: Execute a command if a boolean is true



myboolean here is a boolean variable created in the variables pane. It has previously been set to true, therefore the **if** statement evaluates to true and executes the **Debug Print** command.

Example 2: Execute a command if a boolean is false



myboolean here is a boolean variable created in the variables pane. It has previously been set to false. The **not** logic operator flips a false state to true, therefore the **if** statement executes when false and runs the **Debug Print** command.

Example 3: Toggle a boolean

set *toggle* **to not** (*toggle)* is an easy way to flip a boolean:

Please note, codeblocks within an event execute simultaneously, but in order. This means that a boolean is recommended to be toggled at the beginning or end of your event.

Example 4: Compare two integers

This statement uses the equal-to logic operator (**==**) to compare the values of two number inputs. The **if** statement evaluates to true and executes the **Debug Print** command.

See Also

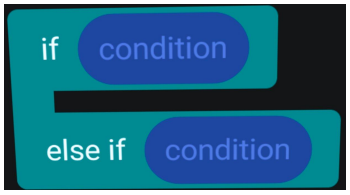
- Events > Event Actions > send event to object
- Events > Event Actions > send event with delay
- Events > Events > when event is received

else If

Events > Control > Else If

Will run codeblocks when the connected **if** is false and the **else if**'s condition is true.

Appearance in Composition Pane



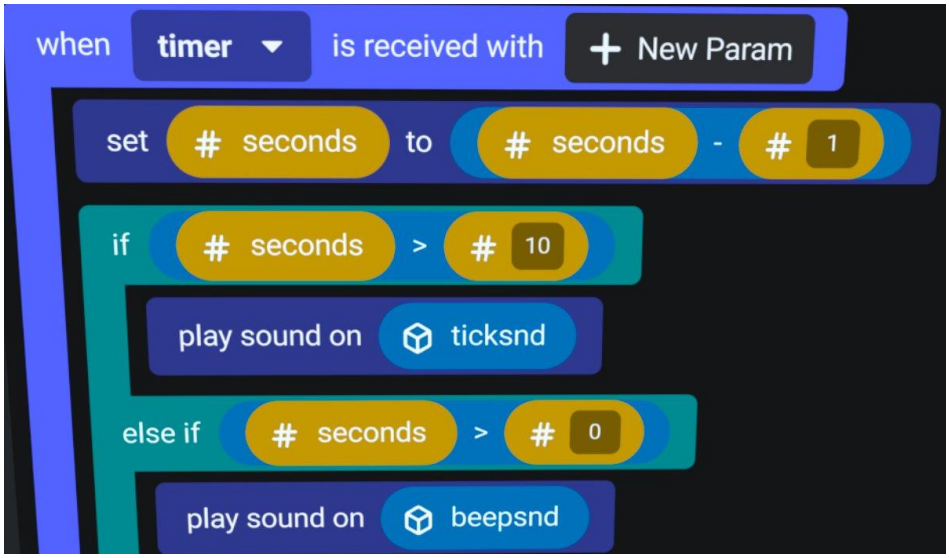
Description

Placed directly below an if, and will run indented codeblocks if the first **if** is false and the **else if**'s condition is true. Can also be placed under another **else if**.

Parameters

boolean

Example 1: Play different sounds based on the number of seconds left



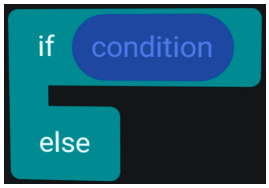
If there are more than 10 seconds remaining, play a tick sound. **Else If** there are greater than 0 seconds, play a beep sound.

else

Events > Control > Else

Will run codeblocks when the first **if** is false.

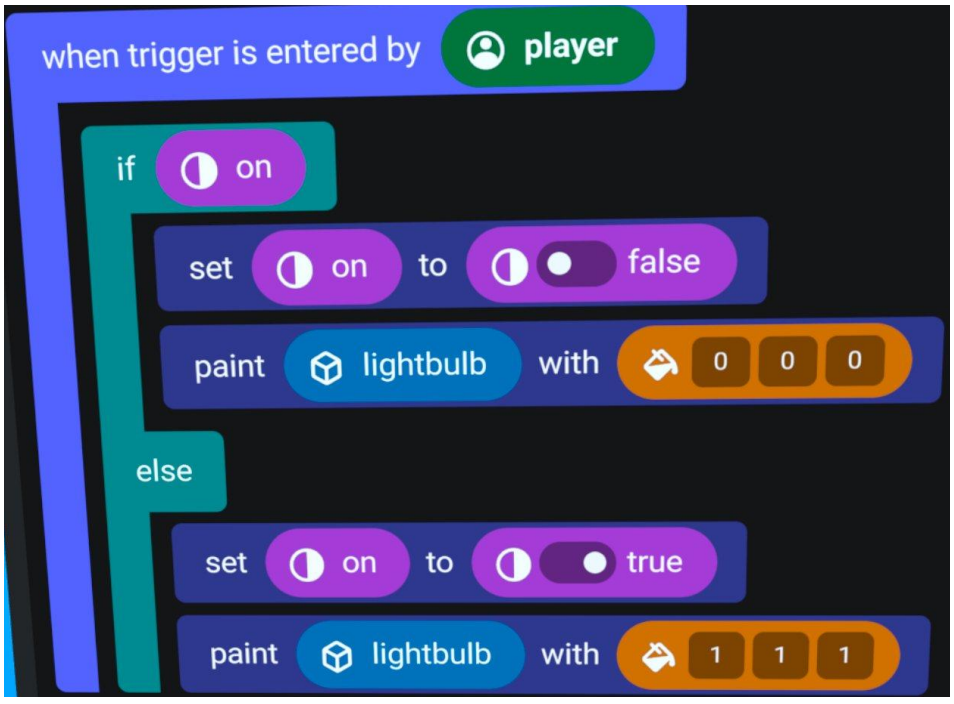
Appearance in Composition Pane



Description

Placed directly below an if, and will run indented codeblocks if the first **if** is false. Can also be placed under an **else if**.

Example 1: turn on and off a lightbulb



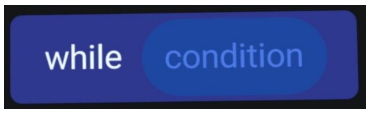
When a player enters a trigger, a light bulb is switched off and on.

while

Events > Control > while

Execute the subsequent codeblocks while a defined condition is met.

Appearance in Composition Pane



Description

The codeblocks indented below the **while** line will execute when the *condition* evaluates to true. This is useful when iterating through a list. This codeblock accepts a limited number of instructions to avoid expensive/infinite loops. If you encounter this limit, the script will stop executing and display an error message in the debug panel. Note that on local scripts, the limit is higher and allows for more executions. Consider using an event loop if you reach these limits.

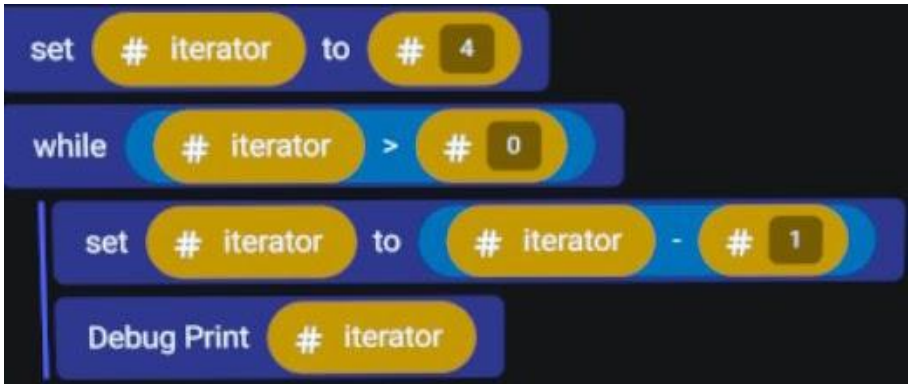
Only booleans are accepted in a **while** block, however, because logic operators return a boolean, they are also accepted in a **while** block.

Parameters

boolean

Runs the commands indented below the **while** block while the boolean is true.

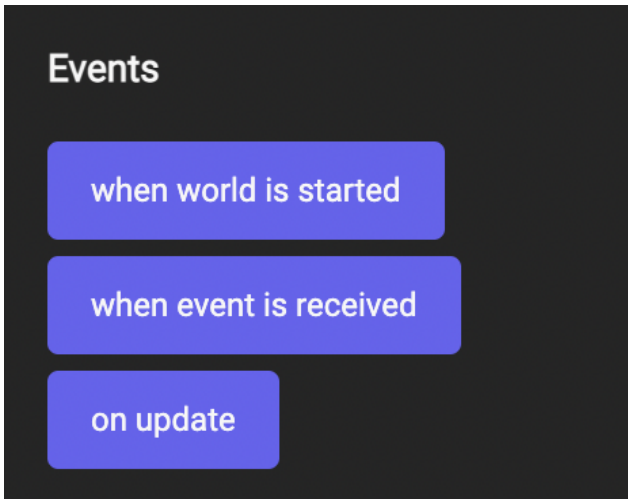
Example 1: Iterate through a list



This **while** loop will execute as long as its argument evaluates to true. In this example, it tests if the iterator is greater than 0. With each passage through the loop, the iterator decrements by 1.

Events

Events > Events



Codeblocks that execute when the game world is started or when a custom event is received.

when world is started

Events > Events > when world is started

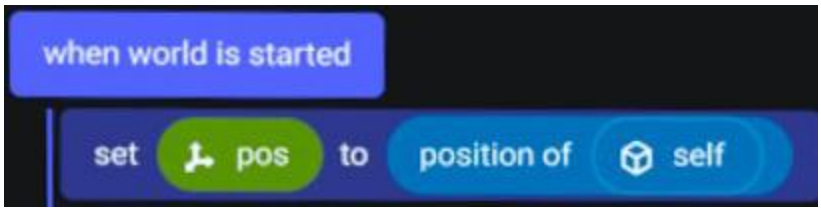
Event that runs when the world is started or reset. Also runs when a local script first executes. And can be executed using send event to object, with the event name start.

Appearance in Composition Pane

Description

The object the script is attached to runs **when world is started** at the start of the world, i.e., when the first person enters the world's instance or upon using the codeblock **reset world state**. It also runs on the start of a local script, such as when a player picks up an object that is locally scripted or the object is assigned to a new player.

Example 1: Record the position of an object



See Also

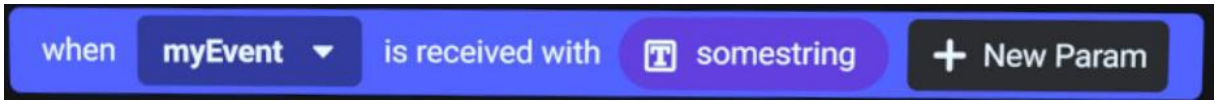
- Events > Event Actions > send event to object
- Actions > World > reset world state

when event is received

Events > Events > when event is received

When event is received will execute when a corresponding **send event** is fired.

Appearance in Composition Pane



Description

The **when event is received** codeblock is executed when it receives an event sent from either itself or another object. This event can be called using **send event to object** or **send event with delay** codeblock. It accepts up to three parameters.

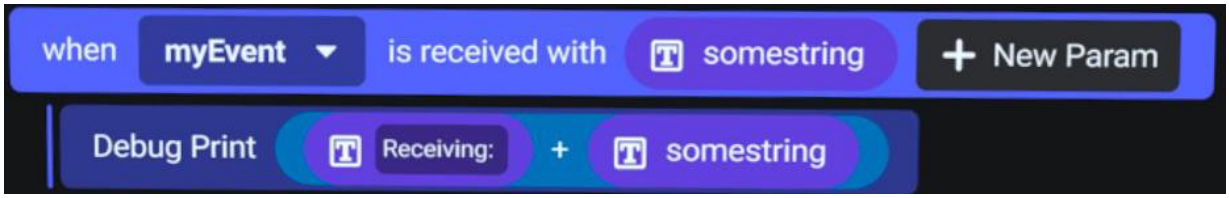
Parameters

+ New Param

Can be any of the value types, except a list. The names given to these input parameters are local to the event. If you need to reference the parameter elsewhere, use **set to** to assign its value to a variable, as those variable names are available to the entire script.

The names of parameters being received must be different from the names of your declared variables. The parameters in the receiving event must match and be in the same order as the parameters in the send event to object codeblock.

Example 1: Read an incoming string parameter



In this example, *myEvent* event is received with a string parameter, *somestring.* That parameter is then print to the debug panel.

See Also

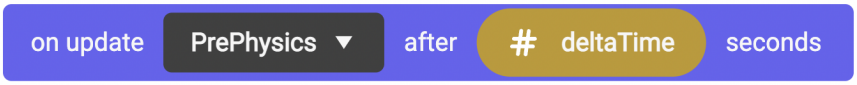
- Events > Event Actions > send event to object
- Events > Event Actions > send event with delay

on update

Events > Events > on update

Run code blocks every frame, specifying to run them before or after the physics engine updates.

Appearance in Composition Pane



Description

The “On Update” code block allows you to specify whether you want to receive the event before the physics system and scripted events run (“PrePhysics”) or after scripted events (“Default”).

Why does execution order matter? When building interactive experiences it's important to think about the execution order of scripting, in relation to the physics engine. For example, if you want a “halo” to follow a user's head then you want that script to happen *after* the physics engine runs (note that the physics engine includes updating the players) so that the halo follows the player. But if you are making a car you want to move the car *before* the player is moved so that the player follows the car.

PrePhysics Update

If you want players or objects to move in response to an object or player *X* in *the same frame* then you should update *X* inside of a PrePhysics update.

For example if you want to implement a vehicle, a moving platform, or anything that moves the player, then you want it to run before the player updates. Likewise if you wanted a cart to crash through a stack of barrels, you would move the cart in a pre-update so that car moves *before* the physics update runs.

Moving an object during PrePhysics update will allow scripts to read the position of that object on the same frame (the script and Default updates), if the scripts are on the same client.

Default Update

If you want a player or object *X* to move in response to other objects that moved in the *same frame* then you should update *X* inside of a Default update.

This allows you to write code that reacts to (or reads data from) player or object movement that occurred due to physics, locomotion, and script events (e.g. from “send event”, “when object is grabbed”, etc) previously in the frame. If you wanted a laser gun to follow a player in a “holster” then you would want to move the laser gun after the player is moved.

The Default update option updates after script events. If a script moves an object, Default update can find the position of the object in the same frame, if they're on the same client.

Parameters

PrePhysics / Default

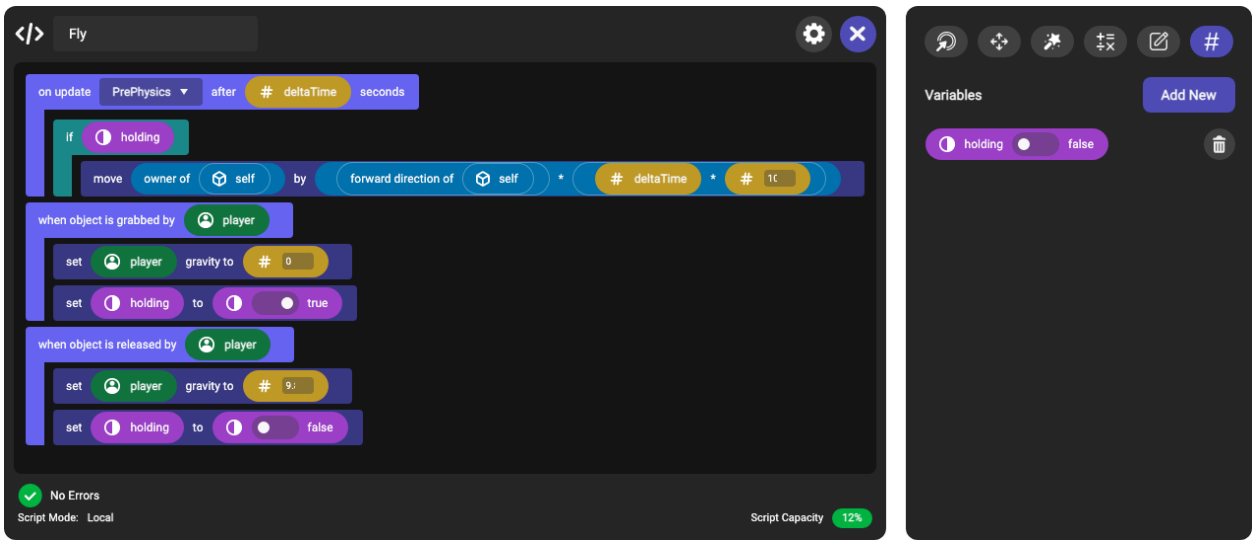
The PrePhysics setting specifies that this code block, and those nested within it, should run every frame and that it should run *before* the frame's physics update (where physical objects, players, and recorded animations update). The Default setting specifies that this code block, and those nested within it, should run every frame and that it should run *after* the frame's physics update (where physical objects, players, and recorded animations update).

deltaTime

The elapsed time in seconds since the last frame. The value is calculated as how much time passed between the start of this frame and the start of the previous frame. This value does not change during a frame (it is cached at frame-start).

This value is useful for moving objects at a consistent speed, no matter the framerate. To move an object 1 meter per second in the upwards direction, move it by (0, 1, 0) * deltaTime each frame, which you can now easily do with the “On Update” event. Likewise you can take the position of an object across two frames, get the difference, and divide by the deltaTime to get its instantaneous velocity in meters per second.

Example - Pull player forward with held object



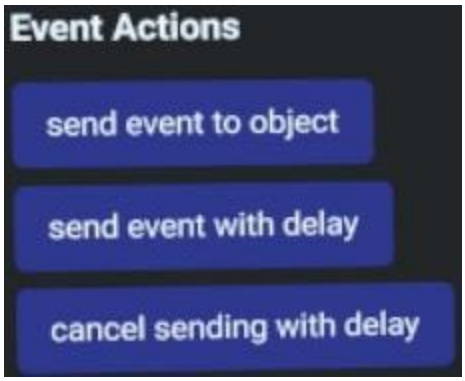
What it does: This example moves a player in the forward direction of an object that they are holding so that they can fly.

How it works: We want to look at where the held object is and then use that to move the player, so we want to run the script before the player update occurs. Thus the PrePhysics update is used to send the “move player” command.

If Default update was used instead, the grabbed object would lag behind. PrePhysics updates the player position before the object position so that the object can accurately move to the player's hand. We set the player's gravity to 0 while they are holding the object so that the script can take control of their movement (until they let go of the object).

Event Actions

Events > Event Actions



These commands allow you to send and execute predefined and custom events.

See Also

- Events > Events > when event is received

send event to object

Events > Event Actions > send event to object

Send an event to self or to another object.

Appearance in Composition Pane

Description

This sets off an event on self. Replacing self with an object variable, allows for the other object to receive the event. You can use this to create communication between multiple objects.

Please note that if you send parameters via the *New Param* pill slot, this allows for up to three values or variables to be sent. Also note that, the receiving event must have matching parameters filled out in the same order they were sent.

Parameters

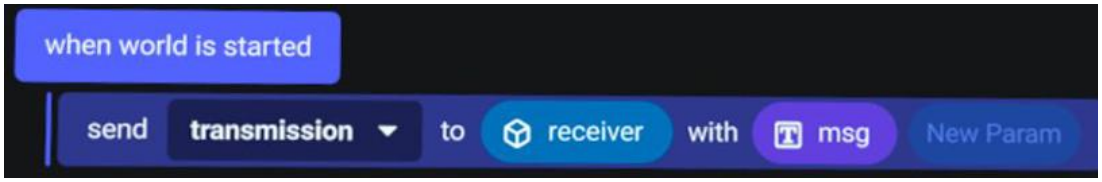
event, object, new parameters

Sends an event to the object with up to three parameters for the event to receive.

event, player, new parameters

Sends an event to a player with up to three parameters. Scripts that listen to events from the player will receive the event if the name and parameters match.

Example 1: Send an event with a parameter



Using **send event to object**, *transmission* is sent to the *receiver*, with a *msg*.

See Also

- Events > Events > when event is received
- Events > Event Actions > send event with delay

send event with delay

Events > Event Actions > send event with delay

Send a custom event after a delay.

Appearance in Composition Pane

Description

Send event with delay sends an event to itself, another object, or a player after a number of seconds. This is useful for delaying a set of commands. It can send up to 3 parameters. The **when event is received** codeblock is used to receive the event. Please note that the names must match and it must have the same parameters, in the same order that they were sent.

Important: Only one of each delayed event can be sent at a time.

Parameters

event, object, number, parameters

Sends an **event** to an **object** after a **number** of seconds with any **parameters** if given.

event, player, number, parameters

Sends an **event** to a **player** after a **number** of seconds with any **parameters** if given.

A script must listen to events on a player, to receive the event.

Example 1: Rotate object on a loop

The object will rotate 90 degrees for 1 second, and then again, continuously spinning.

See Also

- Events > Events > when event is received
- Events > Event Actions > send event to object

cancel sending event with delay

Events > Event Actions > cancel sending event with delay

Cancels an event that was sent with a delay.

Appearance in Composition Pane

Description

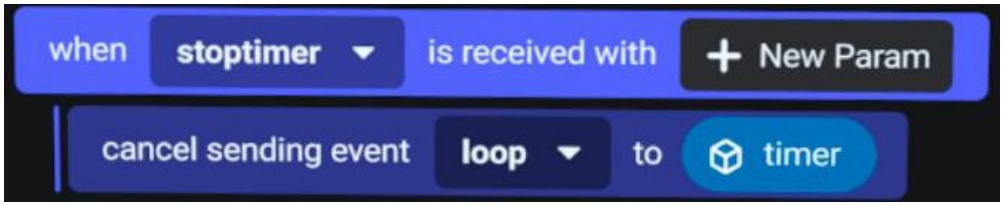
Cancel sending event with delay stops the transmission of a **send event with delay** event that is still in progress. This is useful for things like stopping a loop or for stopping a scheduled event from occurring.

Parameters

event, object

Cancels a delayed event on an object.

Example 1: Interrupt a looped event



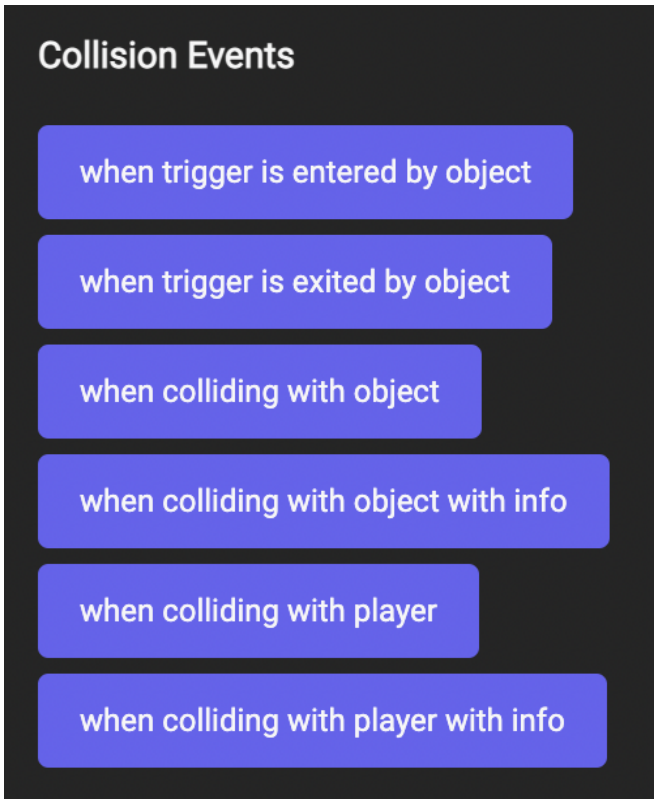
Loop is an event already in progress with **send event with delay**, and now that an event called *stoptimer* has been received. *Stoptimer* stops the *loop* on *timer*.

See Also

- Events > Event Actions > send event with delay

Collision Events

Events > Collision Events



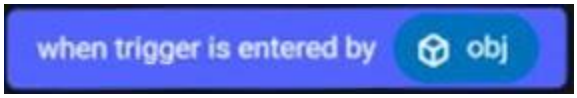
Events that execute with objects in a trigger or on a collision.

when trigger is entered by object

Events > Collision Events > when trigger is entered by object

Event that runs on a trigger, when an object enters the trigger's area.

Appearance in Composition Pane



Description

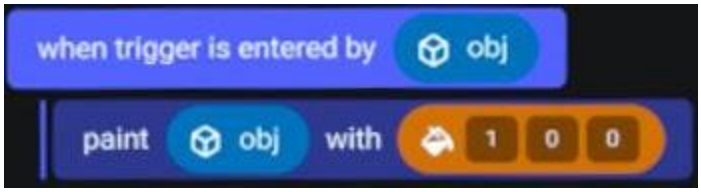
Commands below the **when trigger is entered by object** line will execute when an object enters the trigger. The trigger must be configured to detect objects with a specific tag, and the object must have that tag. The script should then be attached to the trigger, otherwise you can use connect to or listen to, to receive events from the trigger.

Parameters

object

A reference to the object that entered the trigger gizmo.

Example 1: Paint an object



This paints the object that entered the trigger red.

See Also

- Events > Collision Events > when trigger is exited by object

when trigger is exited by object

Events > Collision Events > when trigger is exited by object

Event that runs on a trigger, when an object exits the trigger gizmo's area.

Appearance in Composition Pane

Description

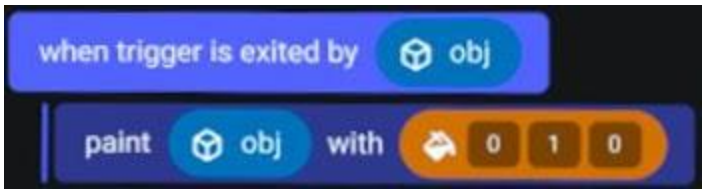
Commands below the **when trigger is exited by object** line will execute when an object exits the trigger. The trigger must be configured to detect objects with a specific tag, and the object must have that tag. The script should then be attached to the trigger, otherwise you can use connect to or listen to, to receive events from the trigger.

Parameters

object

A reference to the object that exited the trigger gizmo.

Example 1: Paint an object



This paints the object that exited the trigger green.

See Also

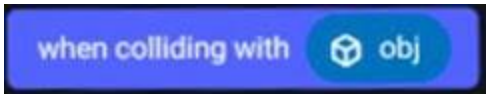
- Events > Collision Events > when trigger is entered by object

when colliding with object

Events > Collision Events > when colliding with object

Event that runs when an object collides with another object.

Appearance in Composition Pane



Description

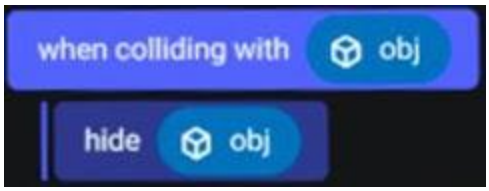
Codeblocks below **when colliding with object** will execute when another object collides with the object the script is attached to. The attached object must be configured in its properties panel to detect collisions: it must be collidable under Behavior > Collidable. Under More > Object Tag, enter a tag that matches the colliding object's tag. Which can be set within the colliding object's properties panel under Attributes > Tag.

Parameters

object

The object that collided with the object this script is attached to.

Example 1: Hide an object



When colliding with object causes the intercepted object *obj* to disappear. See notes above about properly configuring the attached and colliding objects.

See Also

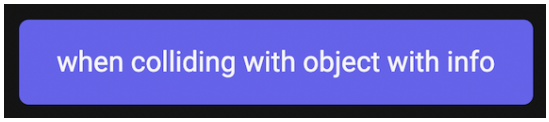
- Events > Collision Events > when colliding with player

when colliding with object with info

Events > Collision > when colliding with object with info

This code block returns the point, normal, and relative velocity when a player collides with an object.

Appearance in Library



Appearance in Composition Pane

Description

This code block returns four parameters when the player collides with an object: The object that collided with the target object, the point at which the object collided with the target object, the normal, and the relative velocity. Both the normal and the relative velocity are vectors. These vectors use world coordinates and the resulting relative velocity is measured by subtracting the world velocity of the target object from the world velocity of the object striking the target object.

Collision events must be turned on for an object in the property panel by going to the “More” tab and changing the “Collision Events From” to player, tagged objects, or both. If this is not enabled the target object will not register collision events. Additionally, an object will only register collision events with objects that are tagged with the target object’s object tag. Objects can be tagged under the “attributes” tab in the properties panel. The object tag can be set under the “more” tab in the properties panel.

All code blocks affecting player velocity or instant movement must have the “Custom Player Manipulation” enabled. This setting can be found by navigating to the player settings section under the “World” tab in the build menu.

Parameters

obj- the striking object that collided with the target object
point - the point at which the striking object collided with the target object

normal - a vector that is always 1 in magnitude and 1 in length. It is perpendicular to the surface of the target object.
relativeVelocity - the vector velocity of the striking object when it collided with the target object.

Example: test your strength: carnival hammer

What it does: When a player strikes an object with a hammer, a text gizmo displays the relative velocity at which the object was struck.

How it works: When a player strikes the object running the script with a hammer, the code block returns the point, normal, and relative velocity. To ensure that the text gizmo displays a positive number, we take the magnitude of the relative velocity and display it as a string. The harder a player swings the hammer, the higher the number will be!

See Also

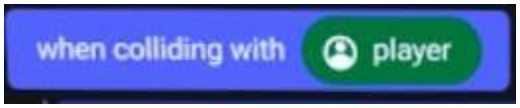
- Events > Collision > when colliding with player
- Events > Collision > when colliding with object
- Events > Collision > when colliding with player with info

when colliding with player

Events > Collision Events > when colliding with player

Event that runs when object collides with a player's head or torso.

Appearance in Composition Pane



Description

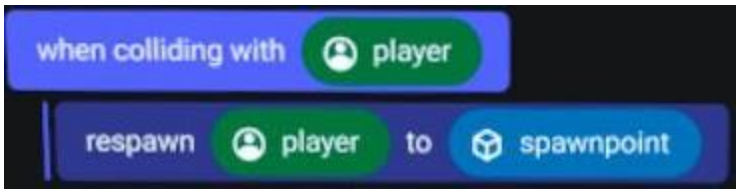
Commands below the **when colliding with player** line will execute when a player's head or torso collides with the object the script is attached to. The object must be configured to detect player collisions or both player and object collisions.

Parameters

player

The player who collided with the object this script is attached to.

Example 1: Respawn a player



Imagine this script is attached to an object that causes a player to respawn.

See Also

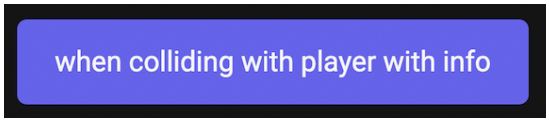
- Events > Collision Events > when colliding with object

when colliding with player with info

Events > Collision > when colliding with player with info

This code block returns the point, normal, and relative velocity when a player collides with an object.

Appearance in Library



Appearance in Composition Pane

Description

This code block returns four parameters when the player collides with an object. By default, only the player's head and torso have collision enabled. The player's hands can also have collision enabled by toggling "Can Hands Collide With Physical Objects" setting. This can be found in the build menu by going to the "Player Settings" option under the "World" tab. The parameters returned are: The player who collided with the object, the point at which the player collided with the object, the normal, and the relative velocity. Both the normal and the relative velocity are vectors. These vectors use world coordinates and the resulting relative velocity is measured by subtracting the world velocity of the target object from the world velocity of the player striking the object.

Collision events must be turned on for an object in the property panel by going to the "More" tab and changing the "Collision Events From" to player, tagged objects, or both. If this is not enabled the target object will not register collision events.

All code blocks affecting player velocity or instant movement must have the "Custom Player Manipulation" enabled. This setting can be found by navigating to the player settings section under the "World" tab in the build menu.

Parameters

player - the player who collided with the object
point - the point at which the player collided with the object

normal - a vector that is always 1 in magnitude and 1 in length. It is perpendicular to the surface of the object.
relativeVelocity - the vector velocity of the player when they collided with the object.

Example: test your strength: punching bag

What it does: When a player strikes an object with their hands, a text gizmo displays the relative velocity at which the object was struck.

How it works: When a player strikes the object running the script with their hands, the code block returns the point, normal, and relative velocity. To ensure that the text gizmo displays a positive number, we take the magnitude of the relative velocity and display it as a string. The harder a player strikes the object, the higher the number will be!

See Also

- Events > Collision > when colliding with player
- Events > Collision > when colliding with object
- Events > Collision > when colliding with object with info

Player Events

Events > Player Events

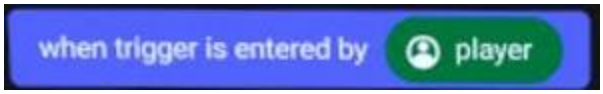
Events that execute when a player enters or exits a trigger or the world.

when trigger is entered by player

Events > Player Events > when trigger is entered by player

Event that runs when a player enters the trigger gizmo.

Appearance in Composition Pane



Description

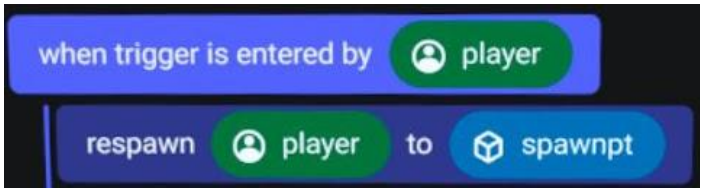
Commands below the **when trigger is entered by player** line will execute when a player enters the trigger the script is attached to. The trigger must be configured to detect players.

Parameters

player

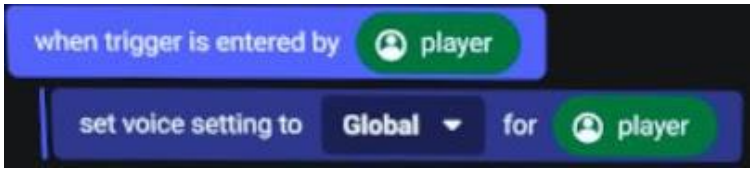
The player who entered the trigger.

Example 1: Respawn trigger



This script is critical for respawning and teleporting players. It's good practice to place a trigger like this below your world in case a player glitches out of a defined area.

Example 2: Set VOIP



This script enables a player to talk to everyone in the world by giving the player Global voice settings.

See Also

- Events > Player Events > when trigger is exited by player

when trigger is exited by player

Events > Player Events > when trigger is exited by player

Event runs when a player exits the trigger gizmo.

Appearance in Composition Pane

Description

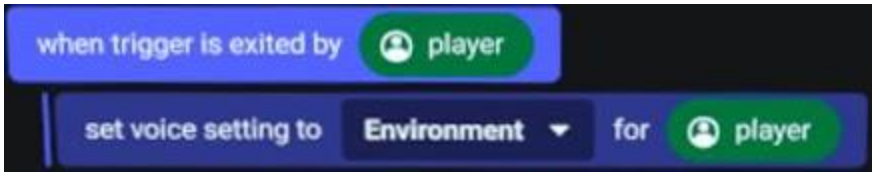
Commands below the **when trigger is exited by player** line will execute when a player exits the trigger the script is attached to. The trigger must be configured to detect players.

Parameters

player

The player who exited the trigger.

Example 1: Set VOIP



This script sets a player's voice settings back to the world default, which can be changed in the Environment Gizmo.

See Also

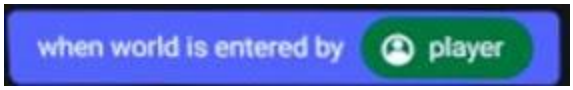
- Events > Player Events > when trigger is entered by player

when player enters the world

Events > Player Events > when player enters the world

Event that runs when a player enters the world, and again if the world is reset.

Appearance in Composition Pane



Description

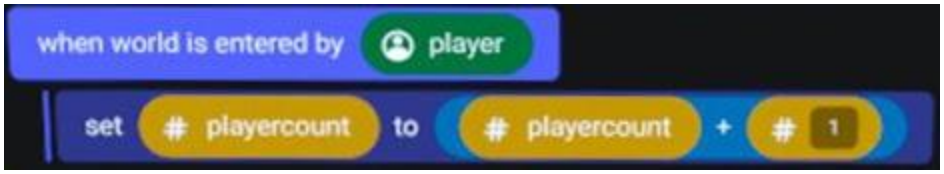
Commands below **when world is entered by player** will execute when a player enters the game world, and again if the world is reset. When editing your world, the **when player enters the world event** will execute when a builder enters preview mode.

Parameters

player

The player who entered the world.

Example 1: Track number of players



This script increments *playercount*.

See Also

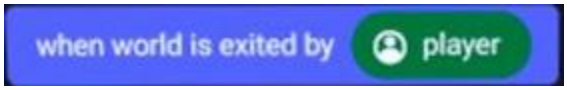
- Events > Player Events > when player exits the world

when player exits the world

Events > Player Events > when player exits the world

Event that runs when a player leaves the world.

Appearance in Composition Pane



Description

Commands below **when world is exited by player** will execute when a player leaves the world. When editing your world, the **when player exits the world event** will execute when a player enters build mode.

Parameters

player

The player who exits the world.

Example 1: Track number of players

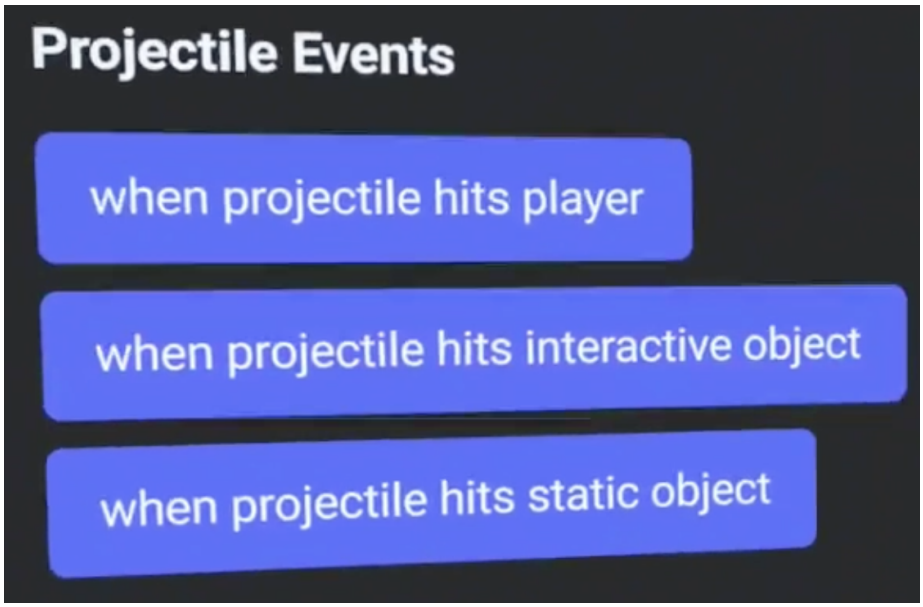
This script decrements *playercount*.

See Also

- Events > Player Events > when player enters the world

Projectile Events

Events > Projectile Events



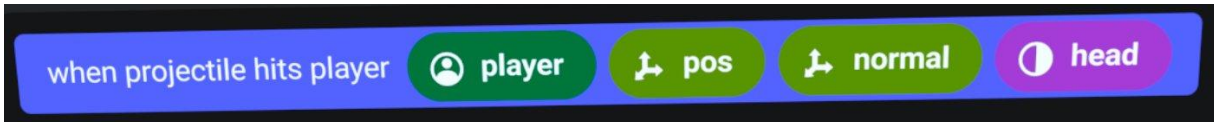
Events that execute on the projectile gizmo when a projectile hits a player or object.

When Projectile Hits Player

Events > Projectile Events > When Projectile Hits Player

Event executes when a projectile, from the launcher gizmo, hits a player.

Appearance in Composition Pane



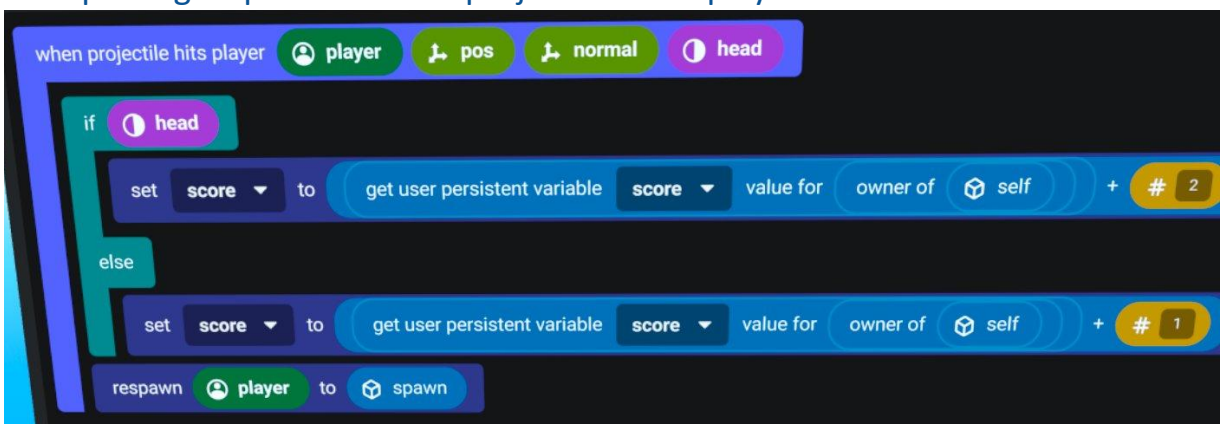
Description

Event executes when a projectile, from the **launcher gizmo**, hits a player. Event is received on the projectile launcher gizmo with four parameters. **Player**, indicates who was hit. **Pos**, is where the projectile hit. **Normal** is a direction vector, which indicates which side of the player was hit. And **head**, returns boolean true if the player's head was hit.

Parameters

player, vector, vector, boolean

Example 1: give points when a projectile hits a player



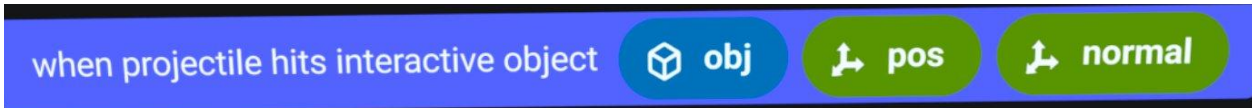
When the projectile hits a player, give the opponent 2 points if it hits their head and 1 point if it hits their torso. Then respawn the player that was hit.

When Projectile Hits Interactive Object

Events > Projectile Events > When Projectile Hits Interactive Object

Event executes when a projectile, from the launcher gizmo, hits an interactive object.

Appearance in Composition Pane



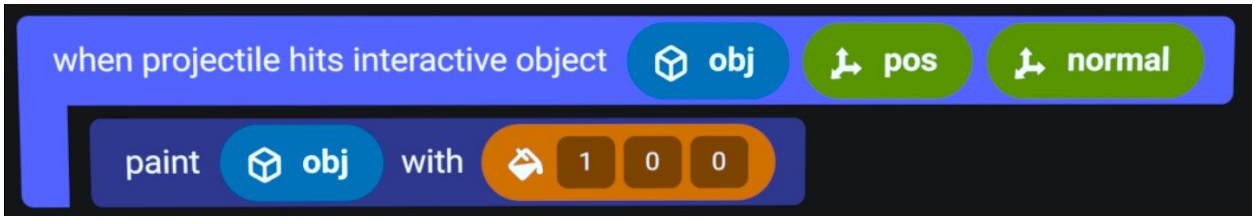
Description

Event executes when a projectile, from the **launcher gizmo**, hits an interactive object. Which is an *object*, with its *properties* set to *Animated* or *Interactive*. The event is received on the projectile launcher gizmo with three parameters. **Obj**, indicates what was hit. **Pos**, is where the projectile hit. And **normal** is a direction vector, which indicates which side of the object was hit.

Parameters

object, vector, vector

Example 1: paint an object when a projectile hits it.



When the projectile hits an interactive object, the object is painted red.

When Projectile Hits Static Object

Events > Projectile Events > When Projectile Hits Static Object

Event executes when a projectile, from the launcher gizmo, hits a static object.

Appearance in Composition Pane

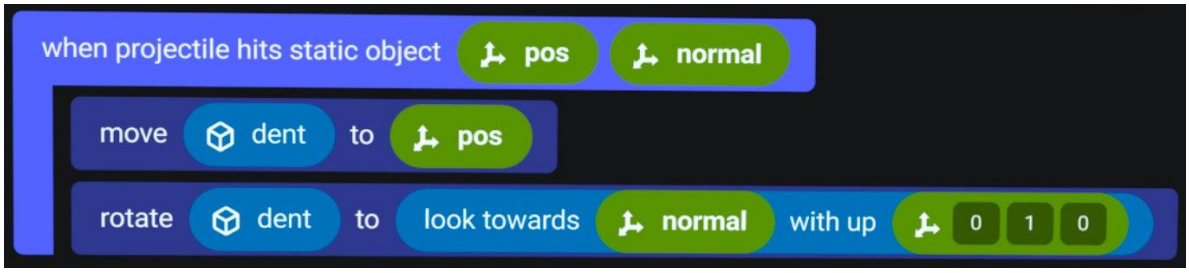
Description

Event executes when a projectile, from the **launcher gizmo**, hits a static object. The event is received on the projectile launcher gizmo with two parameters. **Pos** is where the projectile hit, and **normal** is a direction vector, which indicates which side of the object was hit.

Parameters

vector, vector

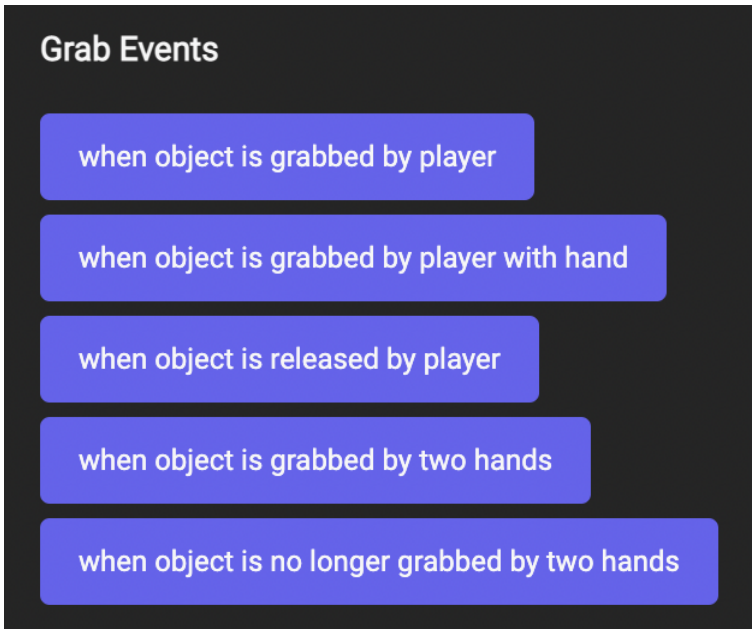
Example 1: create a dent in an object



When the projectile hits a static object, move a dark round disc, which represents a dent, to the hit position. Then, rotate it to face the same direction as the object.

Grab Events

Events > Grab Events



Events that execute when a player grabs or releases an object with one or two hands.

when object is grabbed by player

Events > Grab Events > when object is grabbed by player

Event that runs when the object is grabbed by a player.

Appearance in Composition Pane

Description

Commands below **when object is grabbed by player** will execute when a player grabs the object the script is attached to.

Parameters

player

The player who grabbed the object.

Example 1: Set a boolean

In this example, grabbing the object sets the *holding* boolean to true. This would be useful elsewhere in the script when testing *holding*'s state with the **if** codeblock.

See Also

- Events > Grab Events > when object is grabbed by player with hand
- Events > Grab Events > when object is released by player
- Events > Grab Events > when object is grabbed by two hands
- Events > Grab Events > when object is no longer grabbed by two hands

when object is grabbed by player with hand

Events > Grab Events > when object is grabbed by player with hand

Detect when an object is grabbed by a player and know which hand they grabbed it with.

Appearance in Library

Appearance in Composition Pane

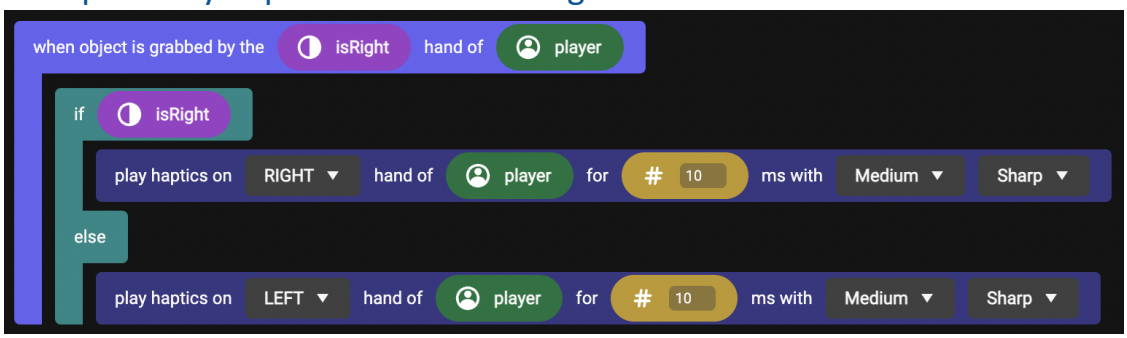
Description

This code block lets you respond to an object being grabbed and tells you which player grabbed it and whether it was with their left or right hand.

Parameters

player - the player grabbing the object
isRight - a boolean representing if the object was grabbed by the player's right hand

Example - Play haptic feedback when grabbed



In this example we play haptics on whichever hand the player grabbed the object with.

See Also

- Events > Grab Events > when object is grabbed by player
- Events > Grab Events > when object is released by player
- Events > Grab Events > when object is grabbed by two hands
- Events > Grab Events > when object is no longer grabbed by two hands

when object is released by player

Events > Grab Events > when object is released by player

Event that runs when the object is released by a player.

Appearance in Composition Pane

Description

Commands below **when object is released by player** will execute when a player holding the object the script is attached to releases it. The command will also execute when a player leaves the world while holding the object.

Parameters

player

The player who released the object.

Example 1: Set a boolean

In this example, releasing the object sets the *holding* boolean to false. This would be useful elsewhere in the script when testing *holding*'s state with the **if** codeblock.

See Also

- Events > Grab Events > when object is grabbed by player
- Events > Grab Events > when object is grabbed by player with hand
- Events > Grab Events > when object is grabbed by two hands
- Events > Grab Events > when object is no longer grabbed by two hands

when object is grabbed by 2 hands

Events > Grab Events > when object is grabbed by 2 hands

Event that runs when the object is grabbed by a player with two hands.

Appearance in Composition Pane

Description

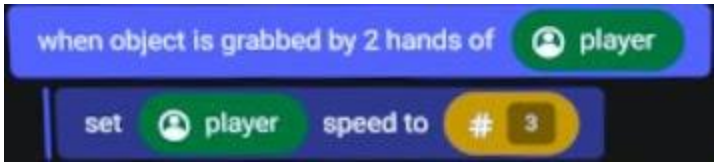
Commands below **when object is grabbed by 2 hands of player** will execute when a player grabs the object with two hands. “Two-Handed Grab,” is enabled by default on grabbable objects, and can be found on the “More” tab of the object’s property panel.

Parameters

player

The player who grabbed the object with two hands.

Example 1: Set player speed



In this example, grabbing the object with two hands changes the player's speed. Imagine if this script was attached to a shopping cart to simulate slower walking speeds. Or set higher for something like a motorcycle.

See Also

- Events > Grab Events > when object is grabbed by player
- Events > Grab Events > when object is grabbed by player with hand
- Events > Grab Events > when object is released by player
- Events > Grab Events > when object is no longer grabbed by two hands

when object is no longer grabbed by 2 hands

Events > Grab Events > when object is no longer grabbed by 2 hands

Event that runs when the object is no longer grabbed by 2 hands.

Appearance in Composition Pane

Description

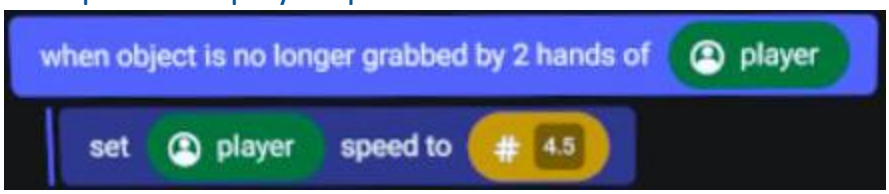
Commands below **when object is no longer grabbed by 2 hands of player** will execute when a player releases one of their two hands from the object the script is attached to.

Parameters

player

The player holding the object.

Example 1: Set player speed



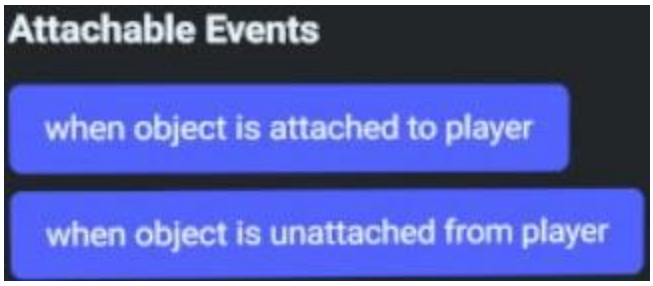
In the previous codeblock, “when object is grabbed by 2 hands,” we considered a shopping cart or a motorcycle. Now when letting go, we can return the player back to their original speed.

See Also

- Events > Grab Events > when object is grabbed by player
- Events > Grab Events > when object is grabbed by player with hand
- Events > Grab Events > when object is released by player
- Events > Grab Events > when object is grabbed by two hands

Attachable Events

Events > Attachable Events



Events that execute when an object is attached or unattached from a player's avatar.

when object is attached to player

Events > Attachable Events > when object is attached to player

Event that runs when the object is attached to a player.

Appearance in Composition Pane

Description

Commands below **when object is attached to player** will execute when a player attaches an object to their avatar. This command is commonly applied to clothing. The object being manipulated must be set to Interactive > Grabbable, and have Avatar Attachable turned on from the More tab of the object's property panel.

Parameters

player

The player the object is attached to.

Example 1: Set player gravity

Imagine if this script was attached to a low gravity broomstick.

See Also

- Events > Grab Events > when object is unattached from player

when object is unattached from player

Events > Attachable Events > when object is unattached to player

Event that runs when the object is unattached from a player.

Appearance in Composition Pane

Description

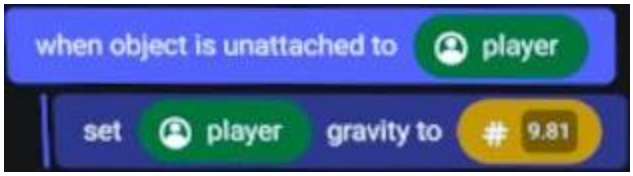
Commands below **when object is unattached to player** will execute when a player unattaches the object from their avatar. This event will also execute when a player leaves the world while wearing an attachable.

Parameters

player

The player the object is unattached from.

Example 1: Set player gravity



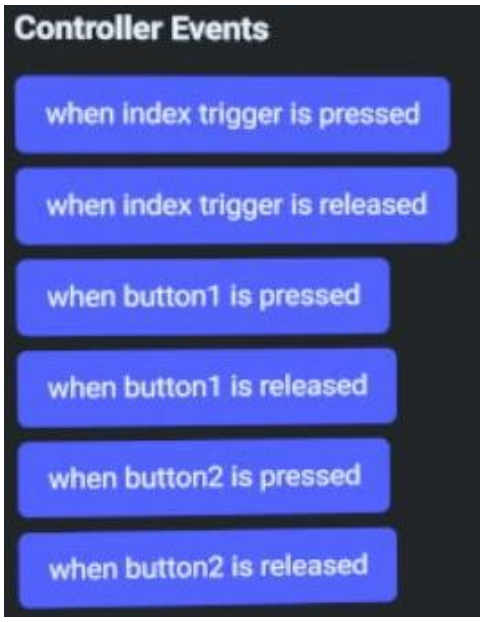
In the previous codeblock, “when object is attached to player,” we considered a low gravity broomstick, now we are returning the player back to regular gravity.

See Also

- Events > Grab Events > when object is attached to player

Controller Events

Events > Controller Events



Events that execute when an object is held and a player presses or releases the buttons and triggers on their Touch Controller.

when index trigger is pressed

Events > Controller Events > when index trigger is pressed

Event that runs when the index trigger on the controller is pressed.

Appearance in Composition Pane

Description

This event executes when a player presses the index trigger button on their controller while grabbing the object that the script is attached to.

Parameters

player

The player holding the object.

Example 1: Play music

Imagine if this script was attached to a remote control object, and when the index trigger is pressed, it plays MusicObjectVariable. Which could be a music sound object, placed in a speaker, to add local, directional music to your world.

See Also

- Events > Controller Events > when index trigger is released

when index trigger is released

Events > Controller Events > when index trigger is released

Event that runs when the index trigger on the controller is released.

Appearance in Composition Pane

Description

This event executes when a player releases the index trigger button on their controller when holding the object that the script is attached to.

Parameters

player

The player holding the object.

Example 1: Set a boolean

This script sets boolean *indexpressed* to false for use elsewhere in the script gizmo.

See Also

- Events > Controller Events > when index trigger is pressed

when button1 is pressed

Events > Controller Events > when button1 is pressed

Event that runs when button1 on the controller is pressed.

Appearance in Composition Pane

Description

Depending on which Touch Controller, the right or left, is holding the object, A or X buttons will represent button1.

Parameters

player

The player holding the object.

Example 1: Stop music

Consider the remote control example from “when index trigger is pressed,” this enables us to turn off the music.

See Also

- Events > Controller Events > when button1 is released
- Events > Controller Events > when index trigger is pressed

when button1 is released

Events > Controller Events > when button1 is released

Event that runs when button1 on the controller is released.

Appearance in Composition Pane

Description

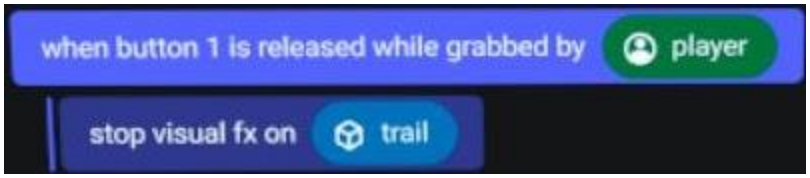
Depending on which Touch Controller, the right or left, is holding the object, A or X buttons will represent button1.

Parameters

player

The player holding the object.

Example 1: Stop Visual FX



Imagine if this script was attached to a wand with a visual effect grouped on the tip. You could even pair it with *when button 1 is pressed* to *play visual fx on* **trail** to create a drawing effect!

See Also

- Events > Controller Events > when button1 is pressed

when button2 is pressed

Events > Controller Events > when button2 is pressed

Event that runs when button2 on the controller is pressed.

Appearance in Composition Pane

Description

Depending on which Touch Controller, the right or left, is holding the object, B or Y buttons will represent button2.

Parameters

player

The player holding the object.

Example 1: Stop music

Consider the remote control example from “when index trigger is pressed,” this enables us to turn off the music.

See Also

- Events > Controller Events > when button2 is released

when button2 is released

Events > Controller Events > when button2 is released

Event that runs when button2 on the controller is released.

Appearance in Composition Pane

Description

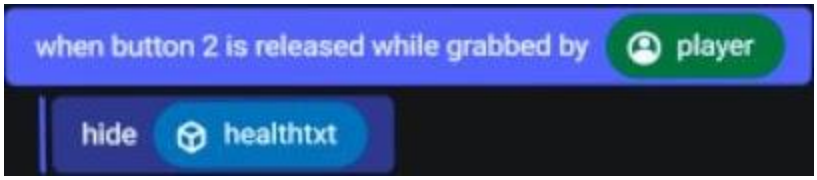
Depending on which Touch Controller, the right or left, is holding the object, B or Y buttons will represent button2.

Parameters

player

The player holding the object.

Example 1: Hide text



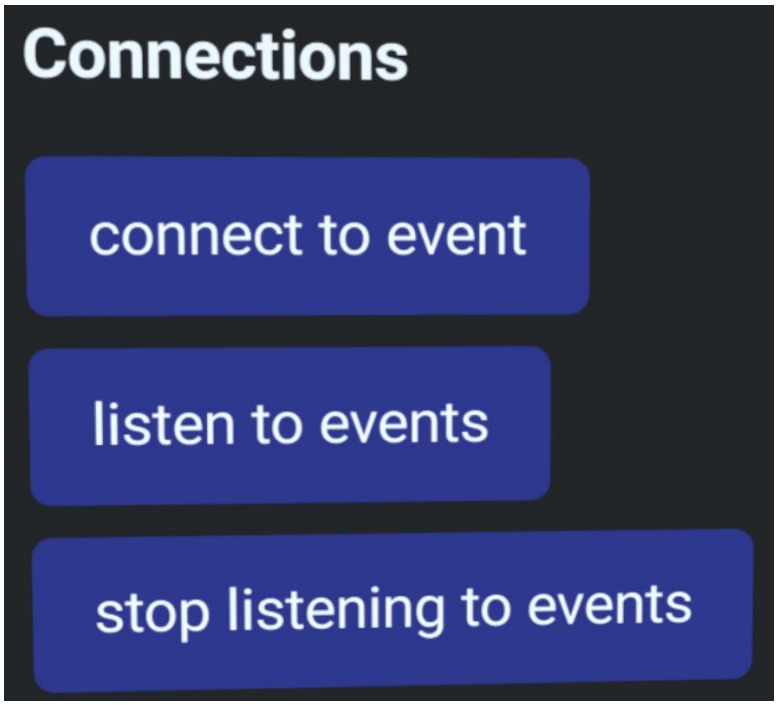
In this example we hide a text object when button 2 is released.

See Also

- Events > Controller Events > when button2 is pressed

Connections

Events > Connections



These codeblocks connect events from other objects to self. And can even be used to remap local events on self.

connect to event

Events > Connections > connect to event

Connects an event to another event in the script.

Appearance in Composition Pane

Description

Connect to event allows you to listen to specific events from other objects, and map them to custom events. Consider Example 1 below. When a player enters Trigger1 you could connect it to a custom event, causing actions to fire, without needing to attach a script to that trigger. Please note you will need to match parameters in the event.

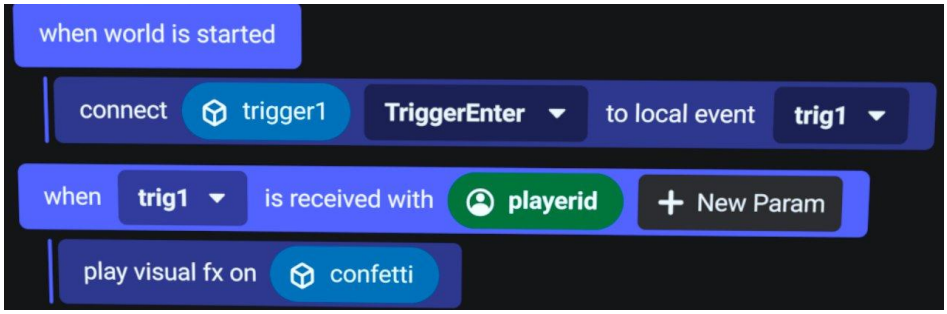
You can also use connect to event to remap events on self to other events. Consider button1 and button2 pressed events. If you wanted these buttons to run the same action, you could connect the button2 event to the button1 event, and just place codeblocks in button1.

Parameters

object, event, event

Connects the object's event to self's event.

Example 1: Connect to trigger



When the world is started we connect Trigger1's triggerEnter event to local event Trig1.

See Also

- Events > Event Actions > send event to object
- Events > Events > when event is received
- Events > Connections > listen to events

listen to events

Events > Connections > listen to events

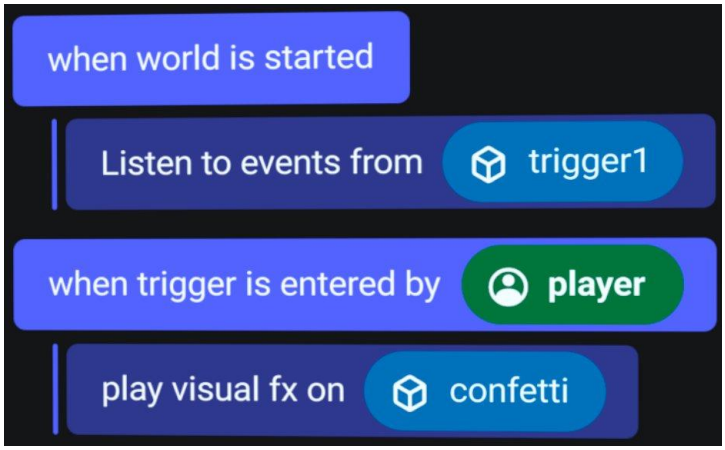
Connects all of the events on another object to self's events of the same name.

Appearance in Composition Pane

Description

Listen to allows you to receive all events from another object. Consider Example 1 below. When a player enters Trigger1 you could listen to it, causing actions to fire, without needing to attach a script to that trigger. Please note that if you are running a script on the trigger object, or if you need to differentiate between triggers, *connect to event* allows for custom event mapping.

Example 1: Listen to trigger



When the world is started we listen to all of the events on *Trigger1*. Allowing us to know when a player enters the trigger and then play *visual fx* on *confetti*.

See Also

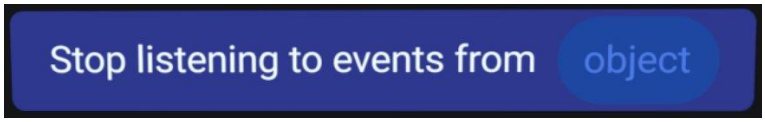
- Events > Connections > connect to event
- Events > Event Actions > send event to object
- Events > Events > when event is received

stop listening to events

Events > Connections > stop listening to events

Stop receiving events if you have previously used Listen To Events.

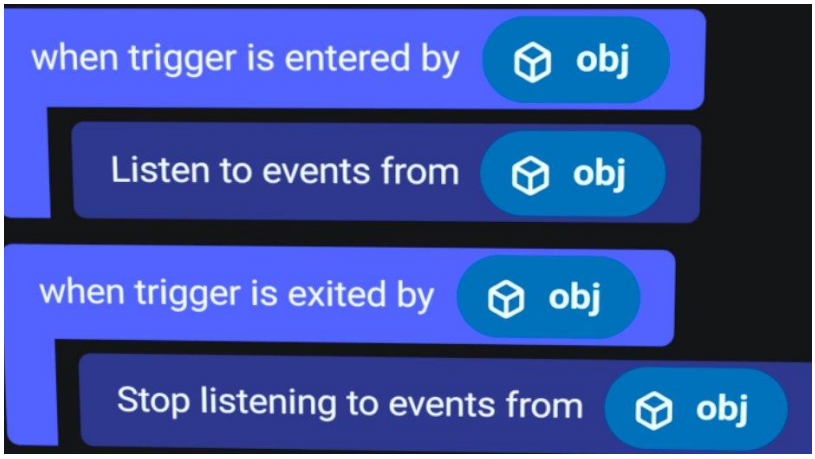
Appearance in Composition Pane



Description

Listen to allows you to receive all events from another object. Stop listening, prevents those events from continuing to be received.

Example 1: Listen To Object Inside Trigger



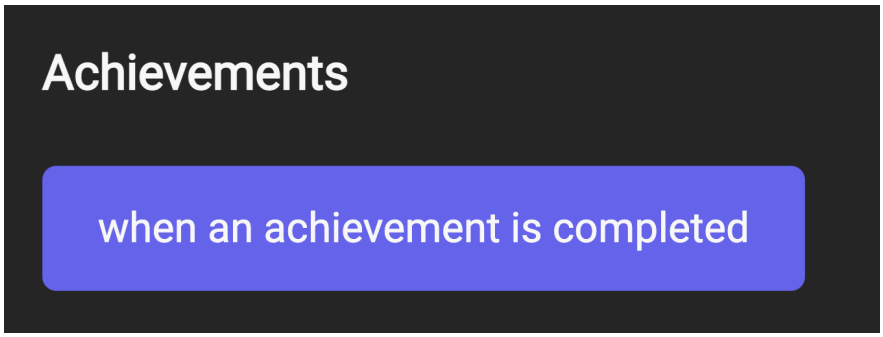
When trigger is entered by object, we begin to listen to that object, and when it exits we stop listening.

See Also

- Events > Connections > connect to event
- Events > Event Actions > listen to events

Achievements

Events > Achievements



These code blocks enable you to run code when achievement-related events are sent.

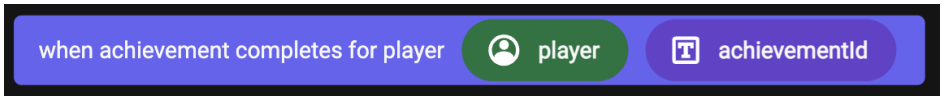
when an achievement is completed

Events > Achievements > when an achievement is completed

Run code when a player completes an achievement.

Appearance in Library

Appearance in Composition Pane



Description

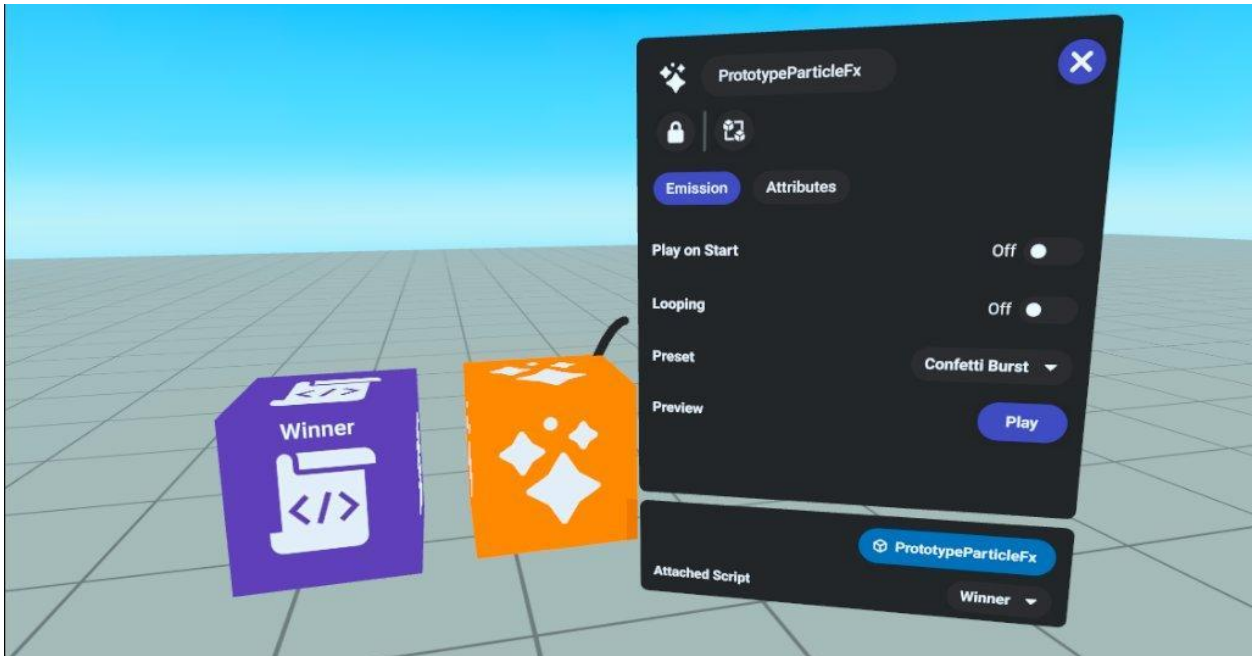
This event is delivered when any achievement is completed by a player. This event is useful when you want to take action of some sort when a player completes an achievement, such as modifying a PPV for a player, attaching objects to the player, teleporting the player, or making some other non player specific change to your world when any player completes certain achievements.

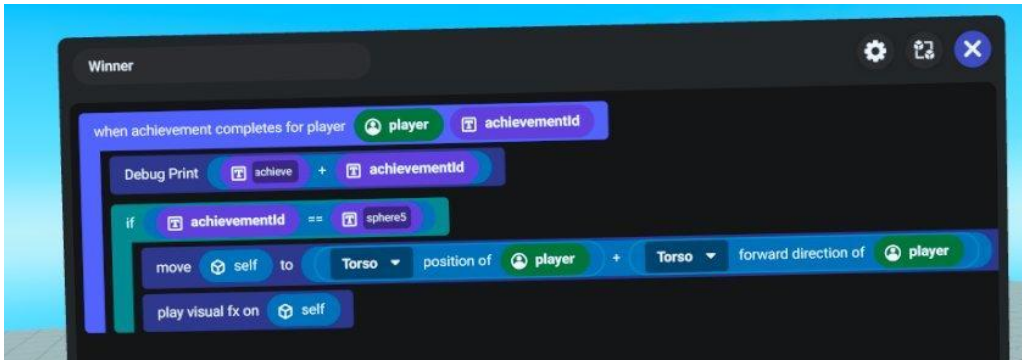
Parameters

Player: the player that completed the achievement.
achievementId: the script ID string of the achievement, which allows you to identify which achievement, exactly, was just completed.

Example: Achievement completion particle effect

This example shows a particle effect with a script attached that monitors for the completion of achievements.





Note that an event is received for every achievement completed by every player. If you want to detect a specific achievement, you need to compare the ‘achievementId’ with the Script ID (not the Name) of the achievement you care about. When found (in this case the ‘sphere5’ achievement), this script moves and plays the particle effect in front of the player completing the achievement.