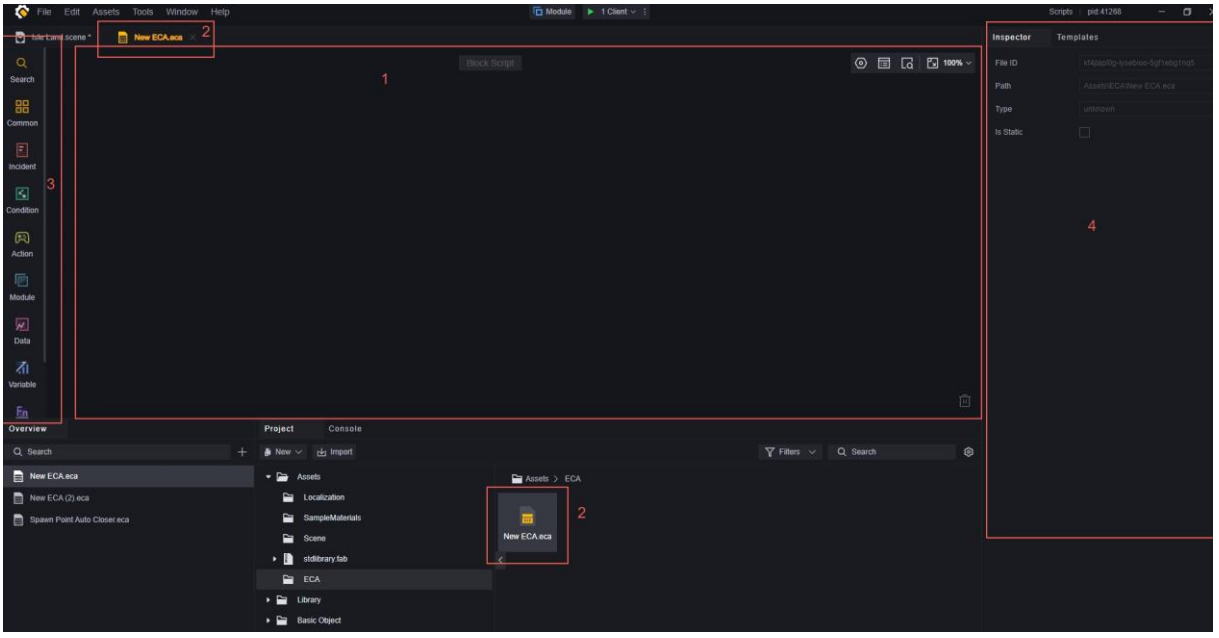
Block Scripts - User's Manuals

This article describes the method of editing the Block script.

Description of the interface



1. Canvas, main editing area
2. Currently editing a file
3. Node classification, clicking on any node will expand the node selection screen.
4. Property Panel

Description of node classification

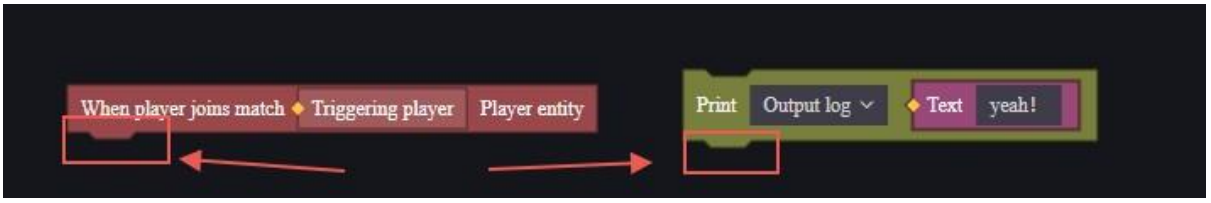
3. The node classifications corresponding to the regions are:

1. Search: Enter keywords to search out nodes.
2. Common: Nodes set as common nodes will be here.
3. Event: Triggered when conditions are met, it is the beginning of logic.
4. Conditional: Used for flow control of scripts.
5. Behaviour: the actual manipulation of the entity\s data.
6. Modules: some module-specific behaviours.
7. Data: processing of data.
8. Variables: Use existing variables or add variables.
9. Functions: Use or create new custom functions.
10. External calls: call custom functions from other scripts.

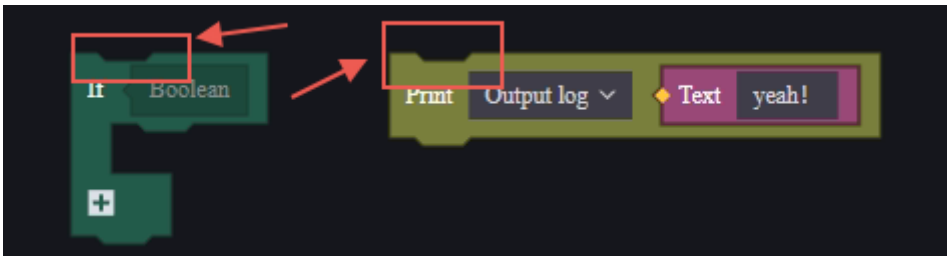
Use of nodes

Node Description

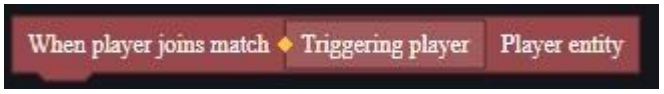
The bump below the node represents the ability to link other nodes after that node.



The indentation above the node represents the fact that other nodes can be linked before the node and must be run by the predecessor node before it will run.



If there is no depression above but a bump below, it represents the node as the starting point of a piece of logic, typically an event node.



If there are no bumps or depressions above or below, it means that the node is a piece of data that can be applied to other nodes.

The node will come with input and output parameters:

Input parameters:

The input parameter is blank by default, and its internally labelled type is the type of the desired parameter.

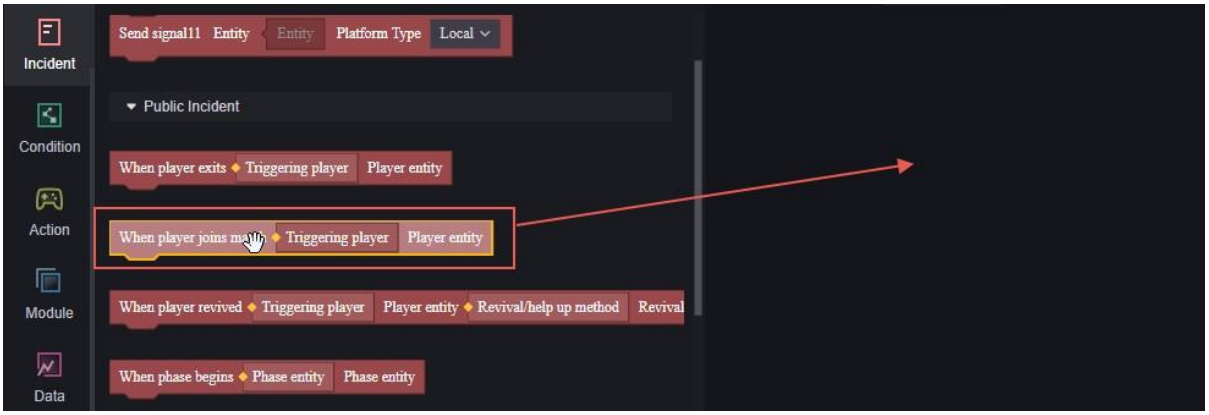
> The script name is also a required parameter, but uses a resource selector that allows you to directly select resources within the project without having to consider type matching.

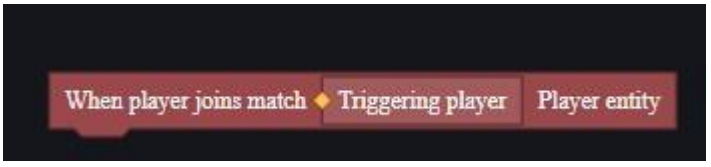
Output parameters:

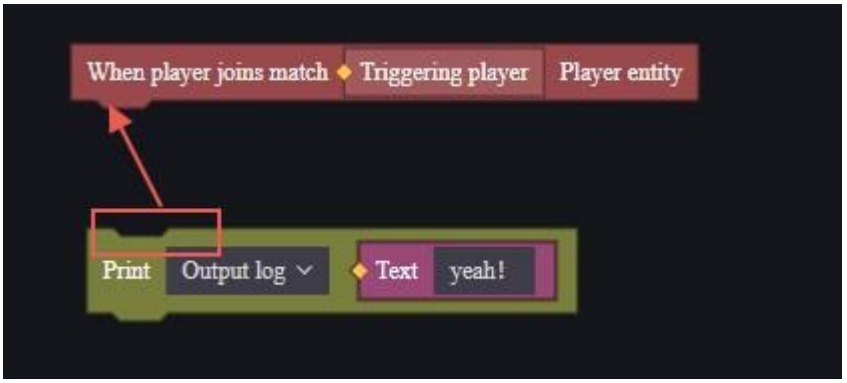
The output parameter defaults to a coloured square, which can be dragged to fill in the desired input parameter box:

Node Usage

After selecting the node you want to use in the category, click or drag it to the canvas to use the corresponding node.

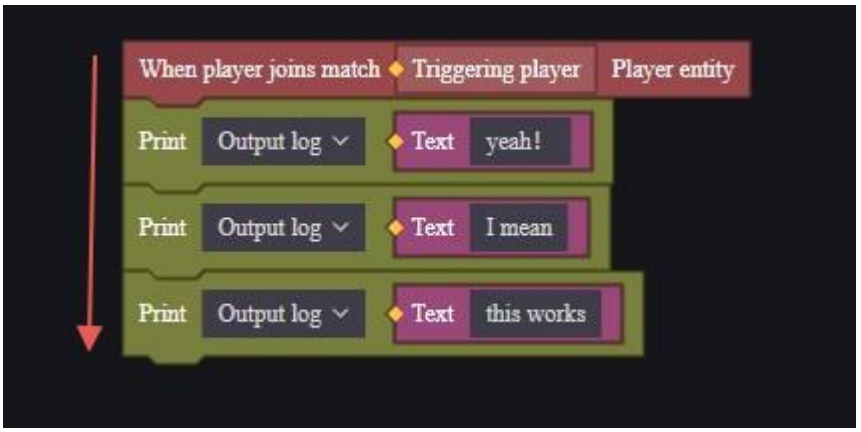




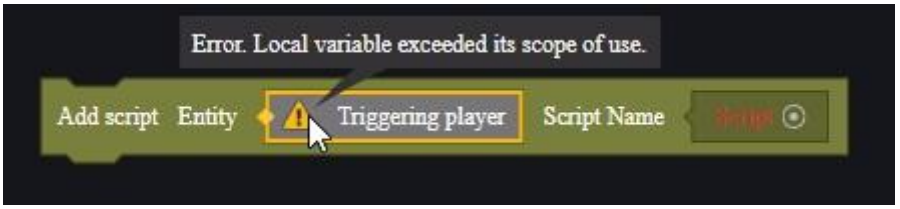


Two nodes can be spliced together using the node with the notch above it:

The logic of a set of nodes is always from top to bottom:



For a group of nodes connected in series, the necessary inputs need to be filled with appropriate data to ensure that the script works properly. Inappropriate variables will report errors, and you can see the error message by clicking on the error:



process control

The default execution order of metascripts is top-to-bottom, but from time to time you may need to control the flow of your code to implement complex logic. Please refer to Meta Scripts - Flow Control in the link below for a detailed description:

Scripts-Additional Instructions-User Manual.md

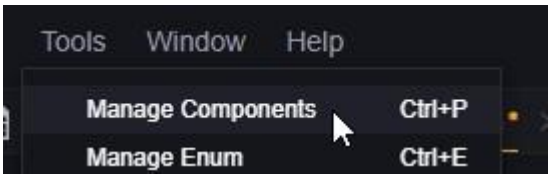
variant

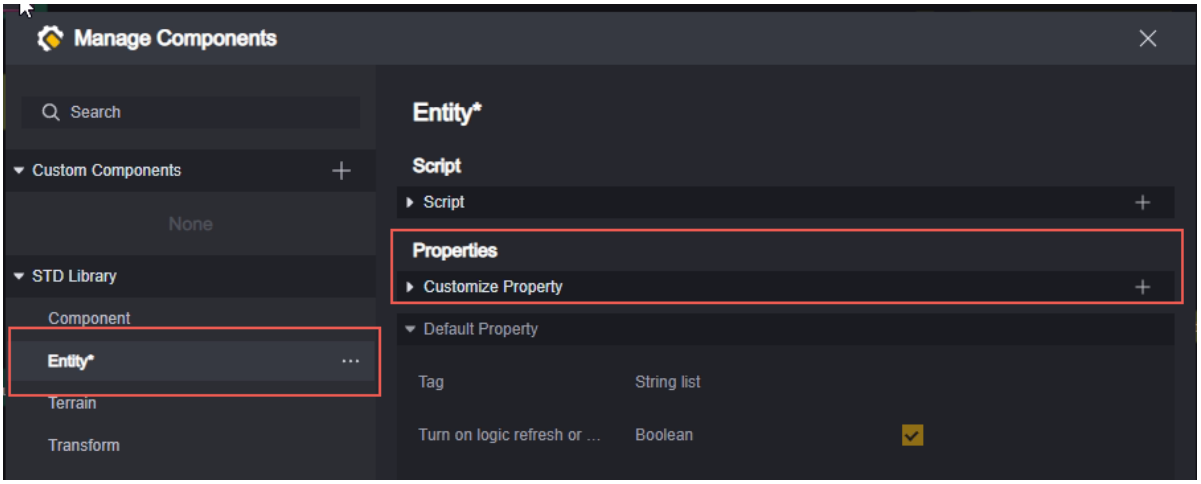
There are three types of variables that can be defined or modified at Variables:

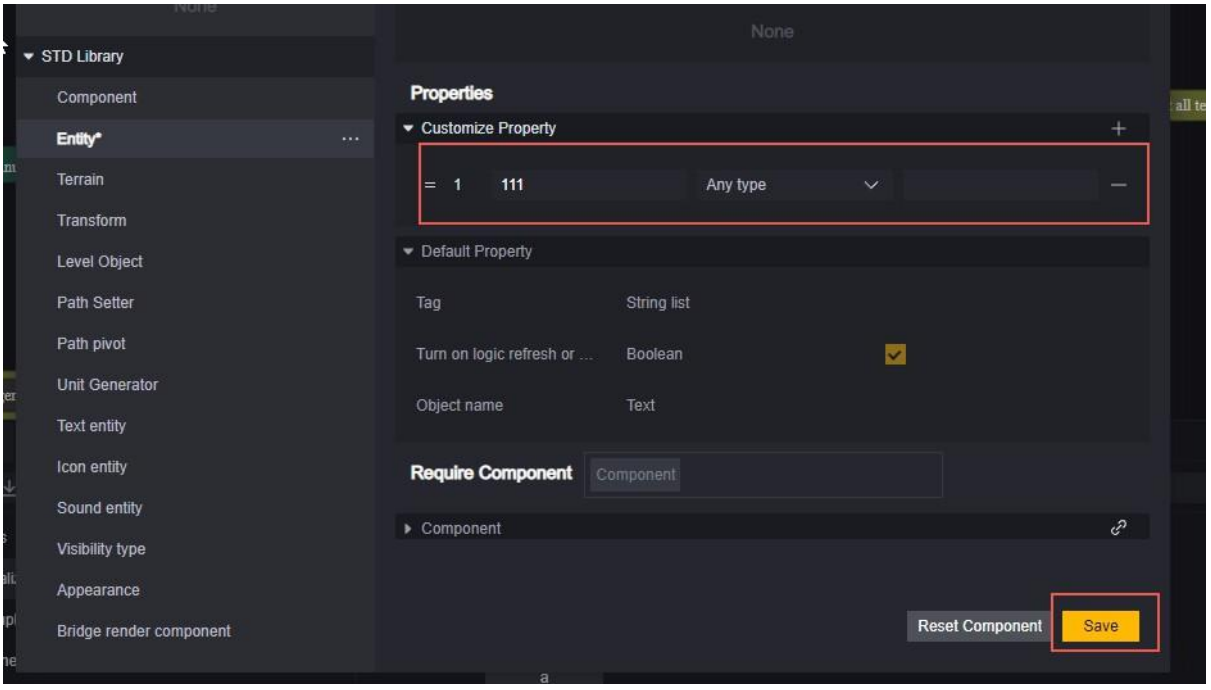
1. Global Entity Properties: Properties of global components that support customisation. Custom global entity properties can be fetched or modified in any script.
2. Script Variables: Variables used only in the current script, which can be obtained and modified externally by other scripts.
3. Local Variables: Variables used only in the current code block, not valid outside the code block.

Global Entity Properties

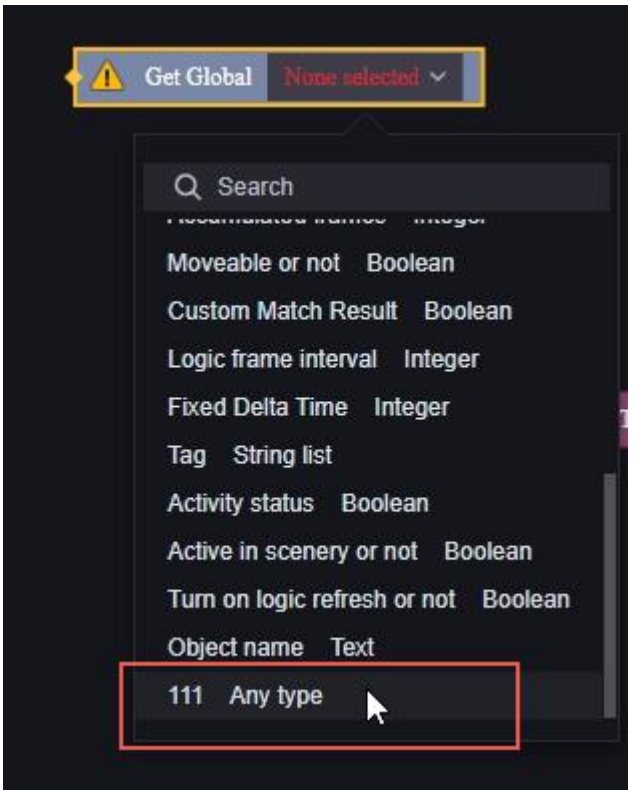
Global entity properties can be added through the entity component properties in the component settings.



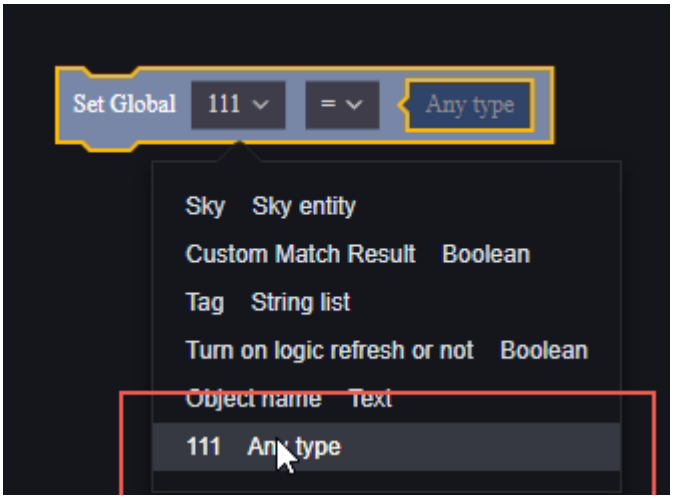




Once you have created a global entity property, you can use the Get Global node to obtain that property:

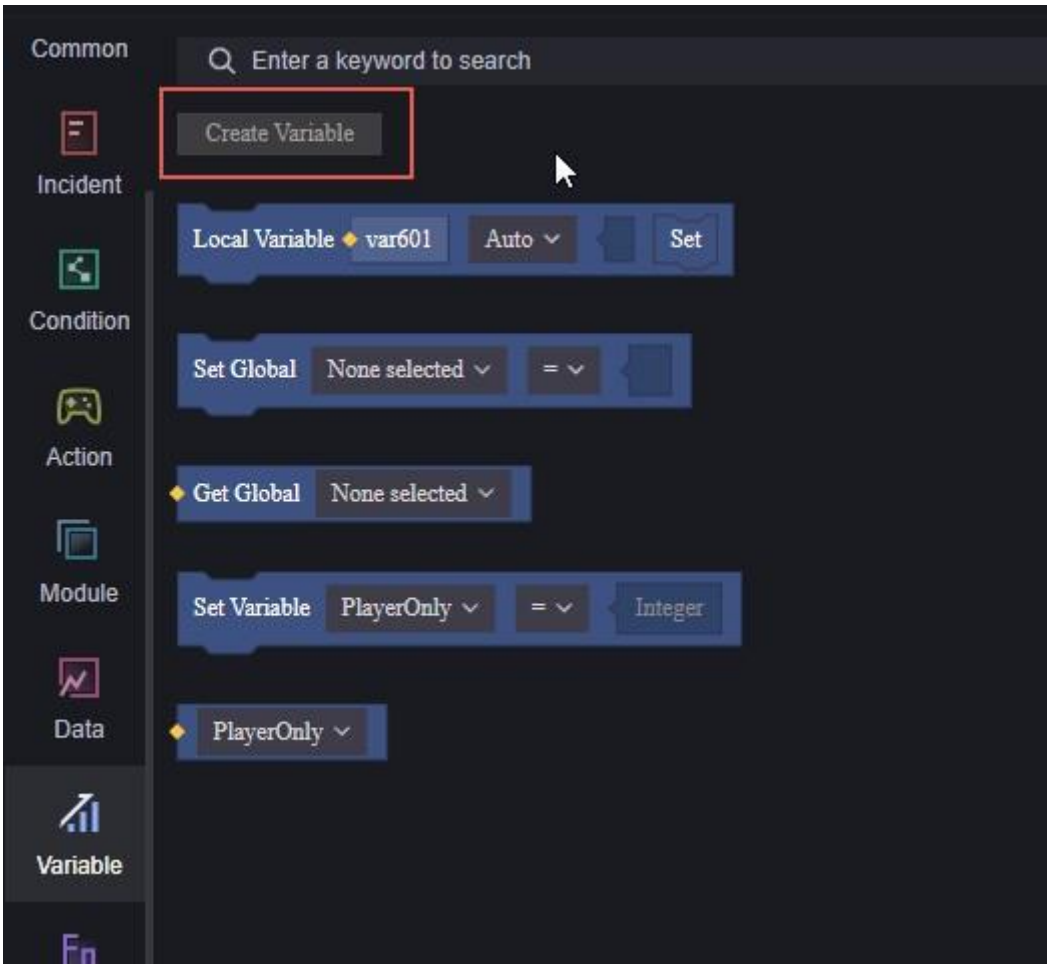


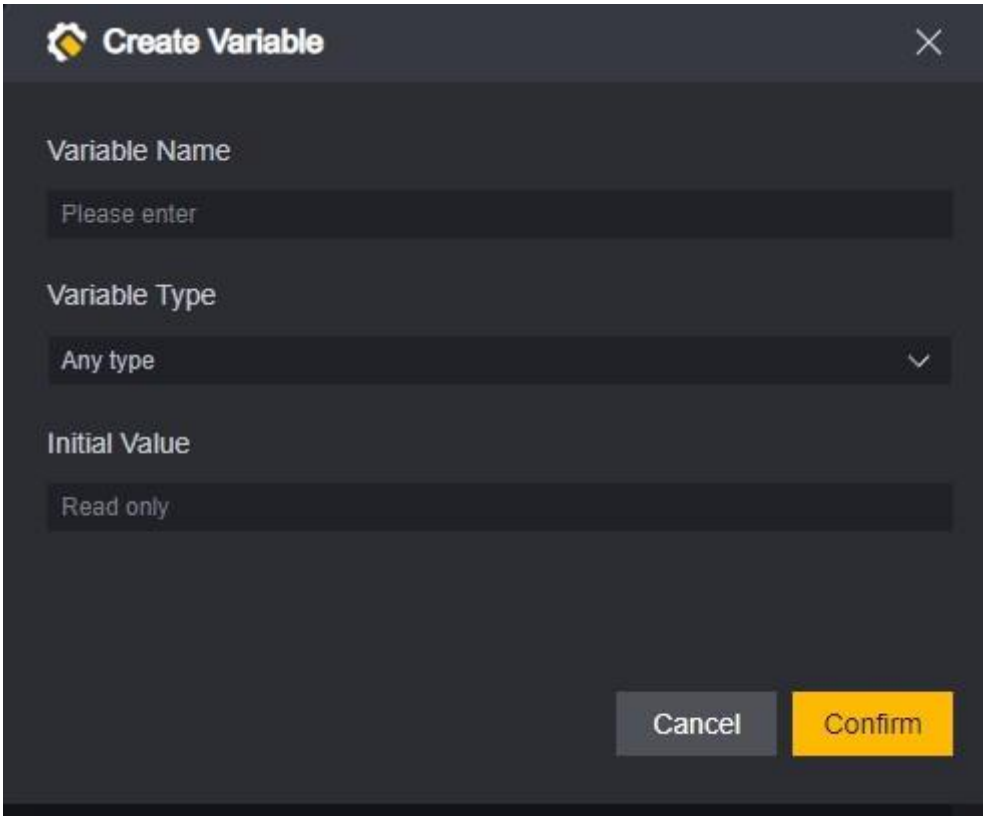
Use the Set Global node to set this property:



script variable

A Create Variable button exists in the Variable category:





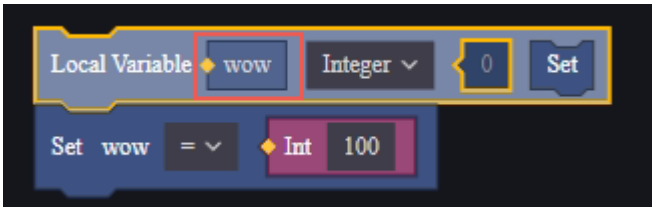
Variables created in this way are script variables and are used for data processing in the current script. The variable can be obtained and set in other scripts by external reference:

> Note when referencing outside of other scripts: you need to specify the corresponding entity of the referencing script mount when getting and setting it.

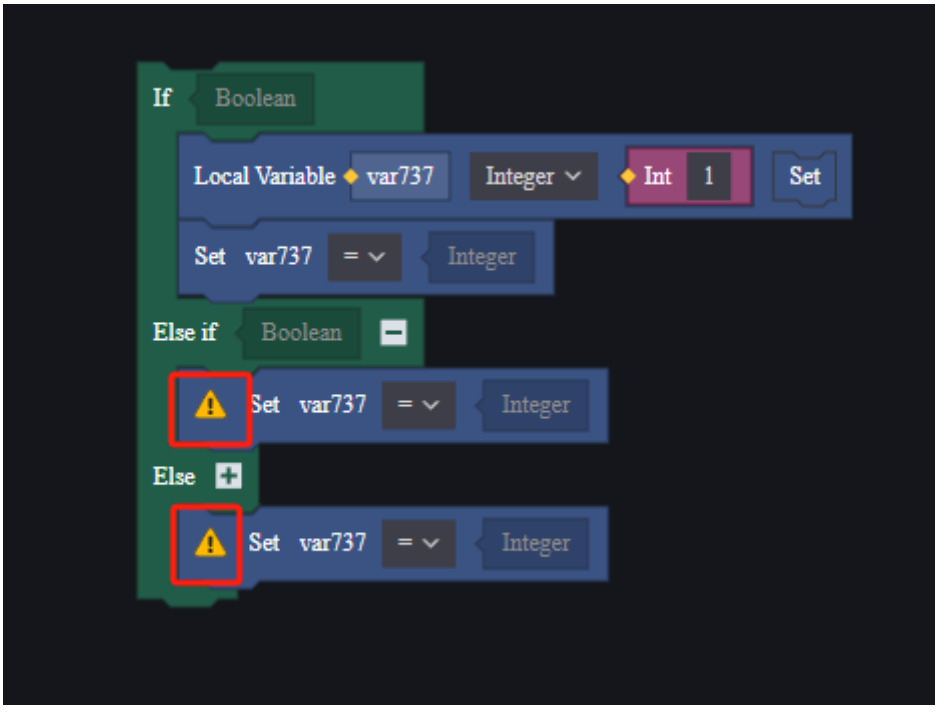
local variable

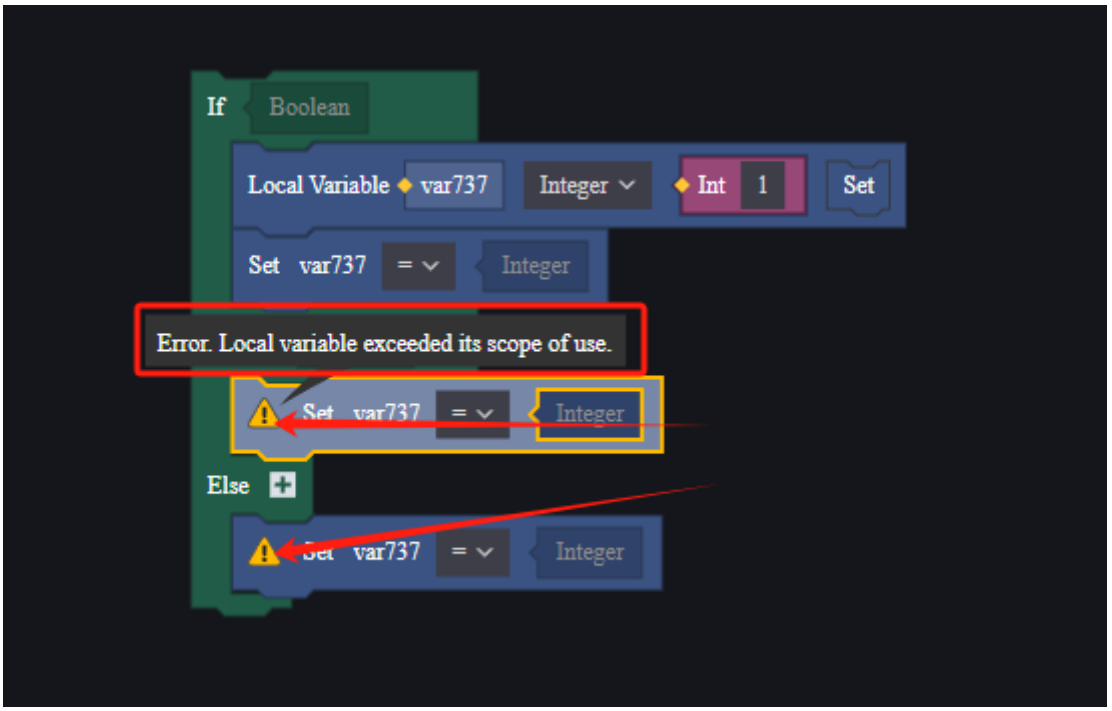
Local variable tuples allow you to create variables that are only available to the current block of code:

The variable can be renamed by double-clicking on the variable name:



A code block is a contiguous segment of tuples, and it is worth noting in particular that the "if-or-else" tuples in the conditional classification are two code blocks by default. Each if, else, or else is a separate block.

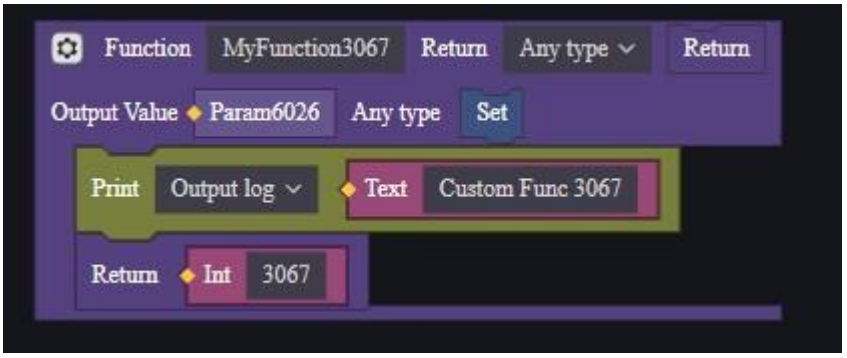


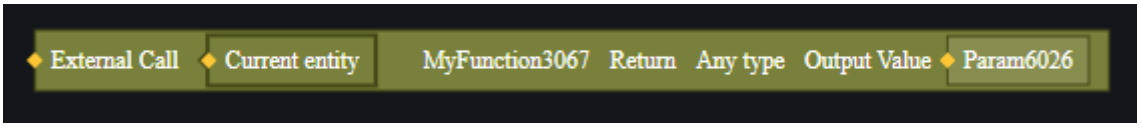


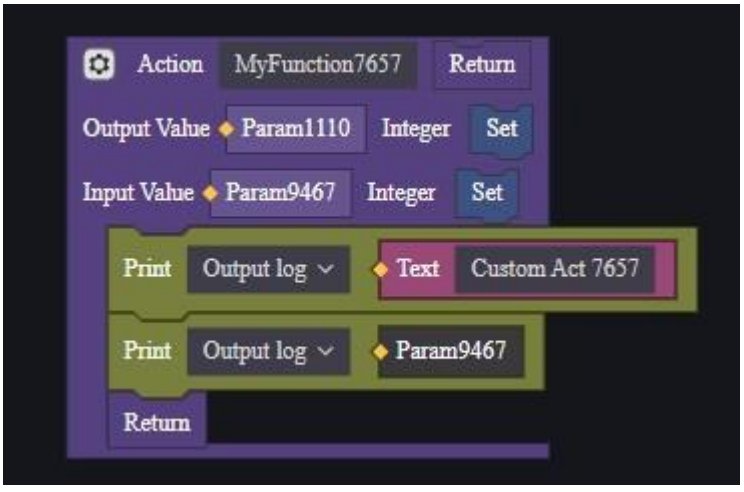
Custom Functions and Usage

Functions with return values and functions without return values are supported in the tuple script.

Functions with return values may not be linked to other nodes when called, and the calling tuple is the return value.





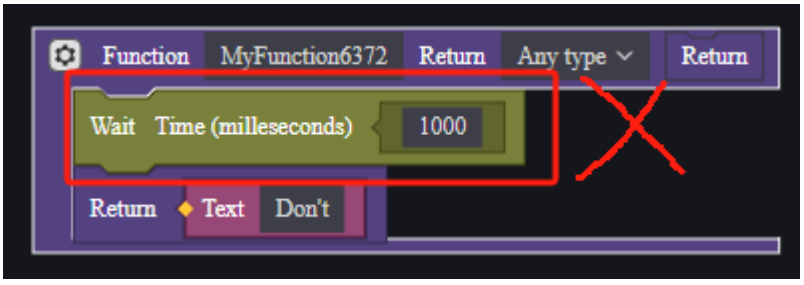


Functions with no return value can link other nodes when called.

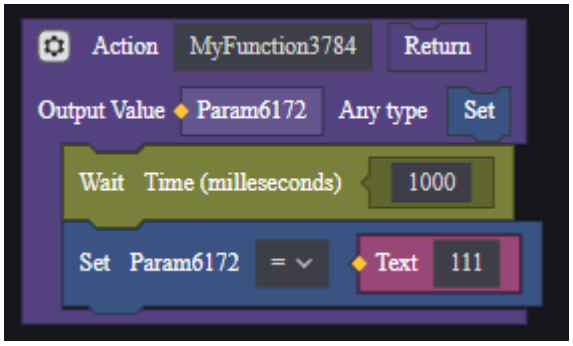
Wait: block when the node is an asynchronous function, and wait for the asynchronous function to finish executing before executing the following node.
Execute: Do not block when the node is an asynchronous function, continue to execute the following node.

Asynchronous functions are function implementations that use nodes that do asynchronous processing, such as "wait".

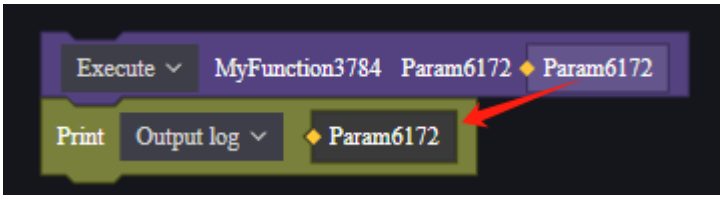
Note: The use of asynchronous tuples in functions with return values is not supported.

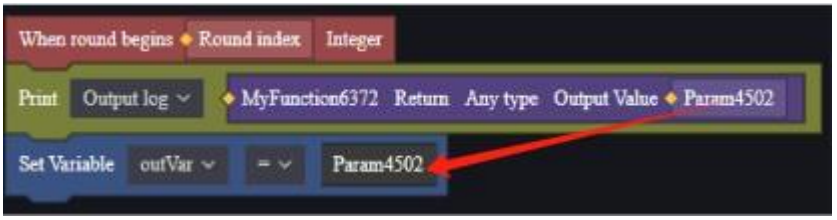


If you need an asynchronous function to return a value, use a no-return-value function and use an output variable.



Whether or not there is a return value, a custom function can use the function's output variable below the calling tuple to get the output.

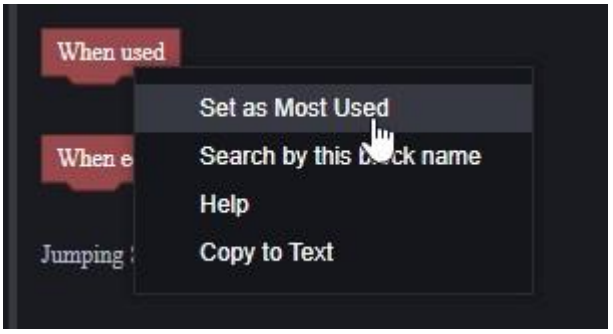


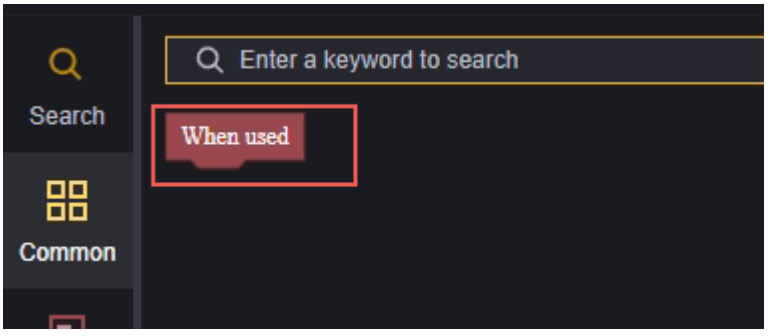


Node search and common use

Enter a keyword to search for the corresponding node

You can right-click a node in the node selection screen to set the node as a frequently used node, and you can quickly use the node in the frequently used node category.





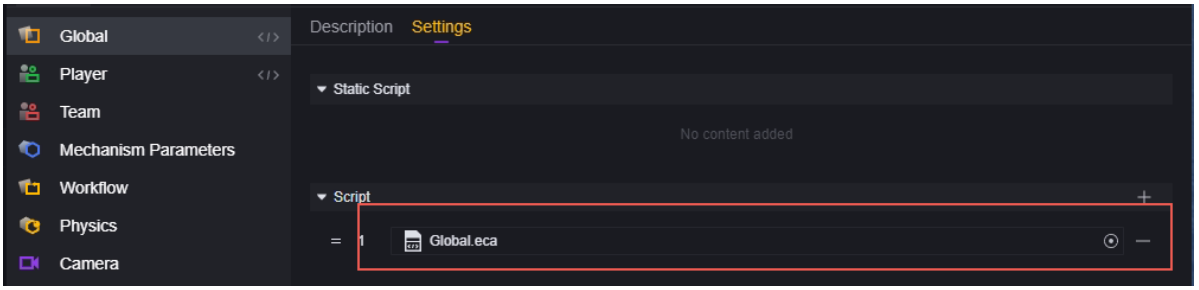
typical example

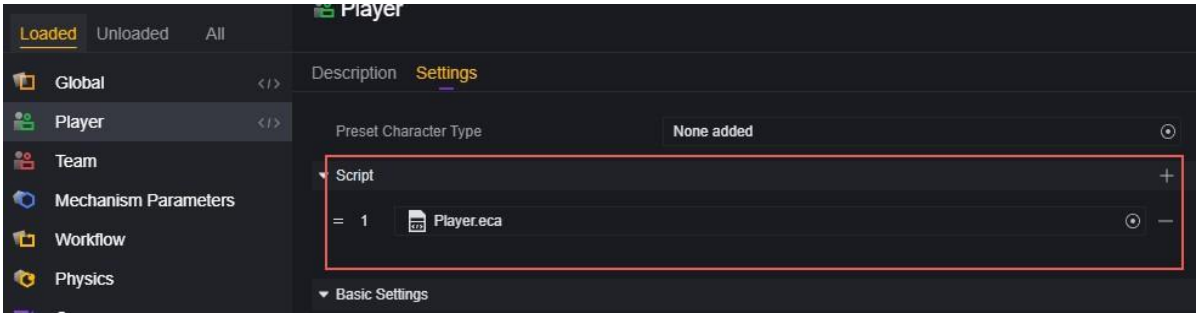
The use of the script is demonstrated using a simple example: the design is as follows:

1. Each player is issued a M4A1 for each player when they join the tournament.
2. When a player fires, they deduct 1 life point from themselves per shot.

Create a script:

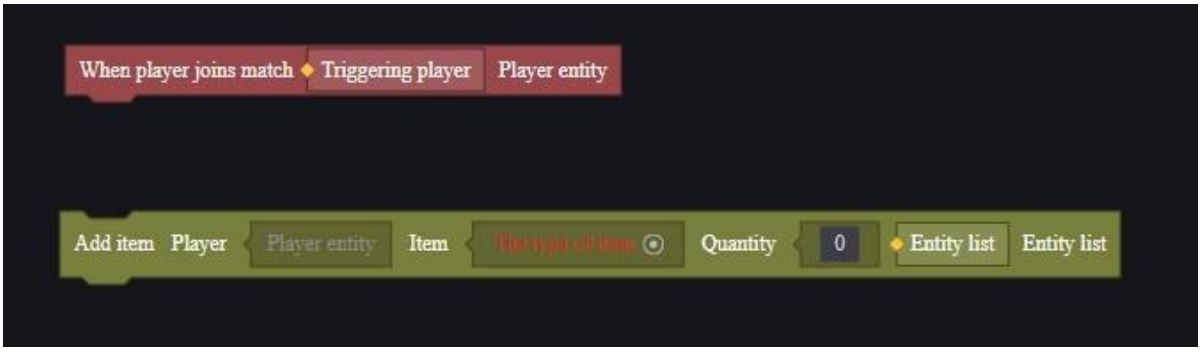
The need for 1. is global and 2. is for each player. So one script needs to be mounted on the global and one on the player. Both giving out props and deducting lives need to be known to the server, so both are created as server scripts.





Edit Script:

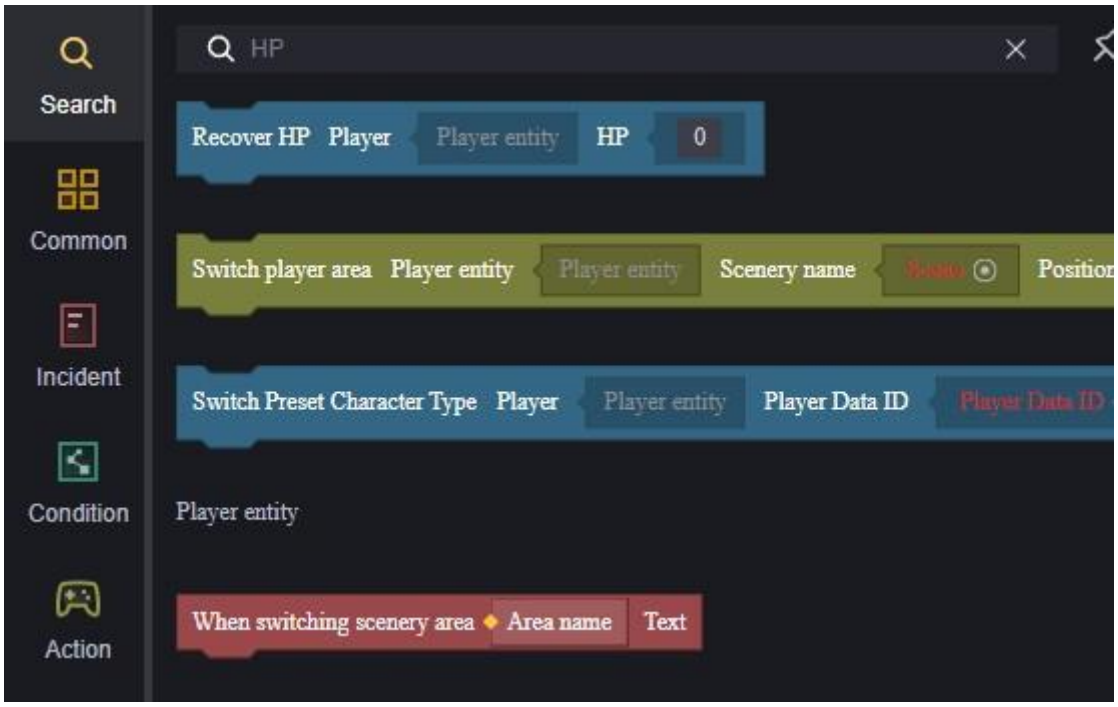
For 1, it is required that each player joining the tournament performs a single issuance of props:



For the Add Props node, three parameters are required: the target of adding props, the props to be added, and the number of props to be added.

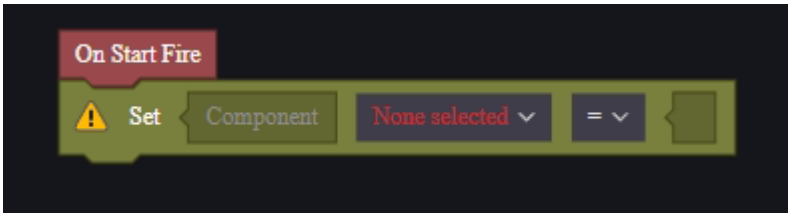
The target for adding props is the player who triggered the When Player Joins Match event, and the props are selected via Explorer for the M4A1 with a quantity of 1.

For 2, a deduction of life value needs to be performed each time it fires: we find that there are actually no nodes that deduct life value:



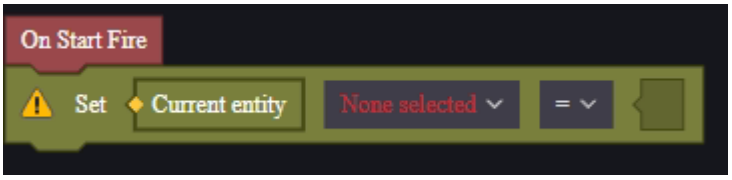
> Negative numbers in the parameters of the Restore Life node do not deduct life from it.

However, life value is used as an attribute of the player and can be adjusted by setting the attribute:



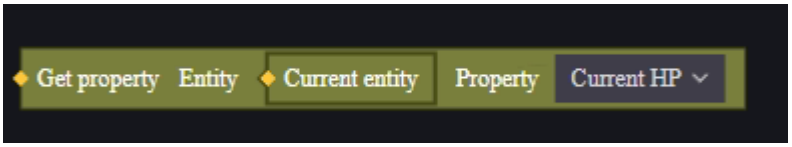
Setting an attribute node requires several parameters: the entity whose attribute is being set, the attribute and value being set, and the operation.

One of the entities whose properties are set is the current player, so you can use this entity and double-click on the parameter location to quickly populate this entity.

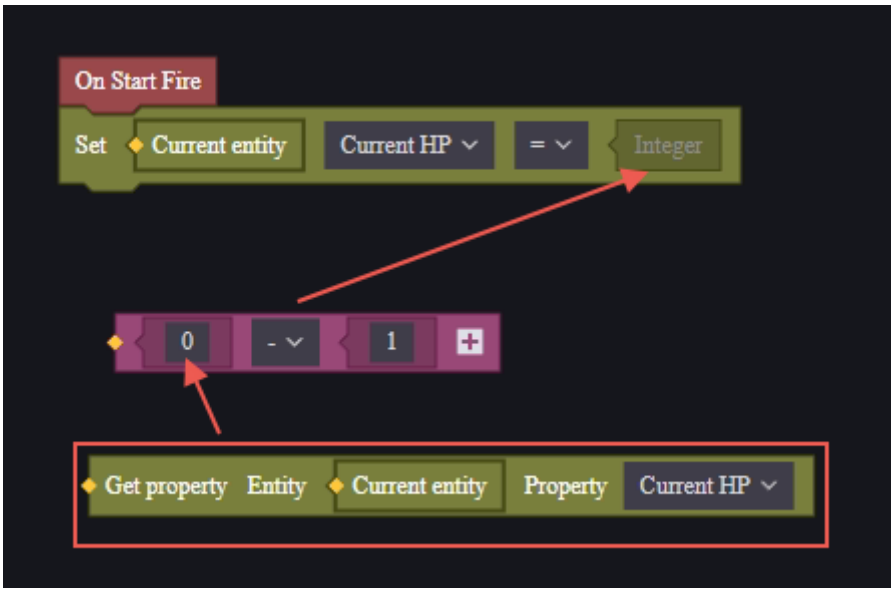


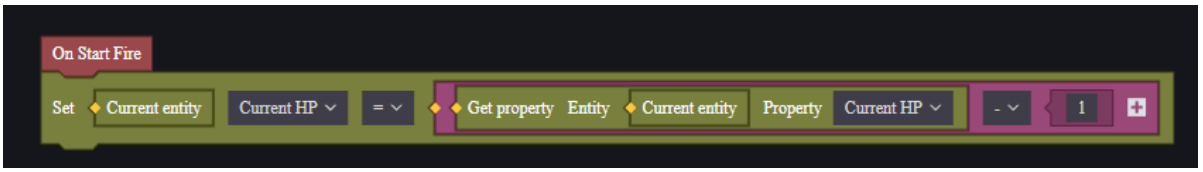
The attribute value that needs to be changed is Current Life Value, select Current Life Value.

And the value set is the current life minus 1, so you need to get the current life.

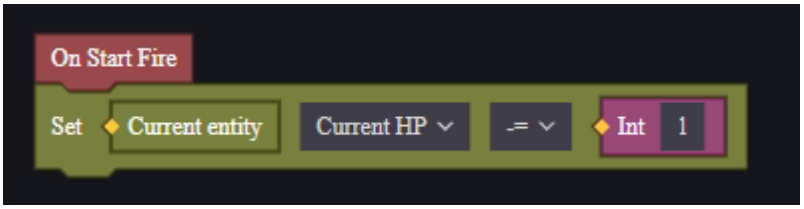


Use subtraction in data classification to subtract one from it and add in the parameters:





It is also possible to implement this logic directly using the operations that set the attribute nodes:



Run debugging to see the results:

Everyone was issued an M4A1.

Fire 8 times dropping 8 life points.

Trigger When Firing is triggered each time the fire command is executed; shots fired by a repeating weapon before the weapon is lowered are considered to be a single fire, and can only trigger this event once.