12.4.3 Game Engine - Logic - Controllers

Controllers

Introduction

The controllers are the bricks that collect data sent by the sensors, and also specify the state for which they operate. After performing the specified logic operations, they send out pusle signals to drive the actuators to which they are connected.

When a sensor is activated, it sends out a positive pulse, and when it is deactivated, it sends out a negative pulse. The controllers’ job is to check and combine these pulses to trigger the proper response.

The logic blocks for all types of controller may be constructed and changed using the *Logic Editor*; details of this process are given in the *Controller Editing* page.

Controller Types

There are eight types of controller logic brick to carry out the logic process on the input signal(s): these are described in the separate pages shown below:

- *AND*
- *OR*
- *XOR*
- *NAND*
- *NOR*
- *XNOR*
- *Expression*
- *Python*

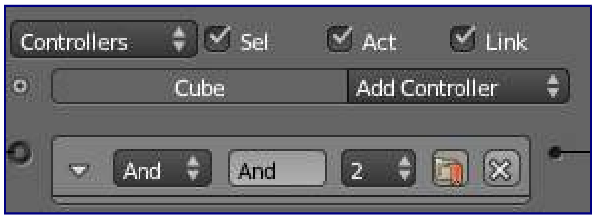
This table gives a quick overview of the logic operations performed by the logical controller types. The first column, input, represents the number of positive pulses sent from the connected sensors. The following columns represent each controller’s response to those pulses. True means the conditions of the controller are fulfilled, and the actuators it is connected to will be activated; false means the controller’s conditions are not met and nothing will happen. Please consult the individual controller pages for a more detailed description of each controller.

Note

It is assumed that more than one sensor is connected to the controller. For only one sensor, consult the “All” line.

Positive sensors	**Controllers**					
	AND	*OR*	*XOR*	*NAND*	*NOR*	*XNOR*
None	False	False	False	True	True	True
One	False	True	True	True	False	False
Multiple, not all	False	True	False	True	False	True
All	True	True	False	False	False	True

Controller Editing

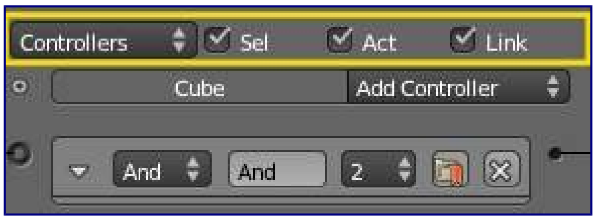


Controller Column with Typical Sensor

Blender controllers can be set up and edited in the central column of the Logic Panel. This page describes the general column controls, those parameters which are common to all individual controller types, and how different states for the objects in the logic system can be set up and edited.

The image shows a typical controller column with a single controller. At the top of this column, and for sensors and actuators, the column heading includes menus and buttons to control which of all the controllers in the current Game Logic are displayed.

Column Heading



Controller Column Headings

The column headings contain controls to set which controllers appear, and the level of detail given, in the controller column. This is very useful for hiding unecessary controllers so that the necessary ones are visible and easier to reach. Both these can be controlled individually.

Controllers:

Show Objects
: Expands all objects.

Hide Objects
: Collapses all objects to just a bar with their name.

Show Controllers
: Expands all Controllers.

Hide Controllers
: Collapses all Controllers to bars with their names.

It is also possible to filter which controllers are viewed using the three heading buttons:

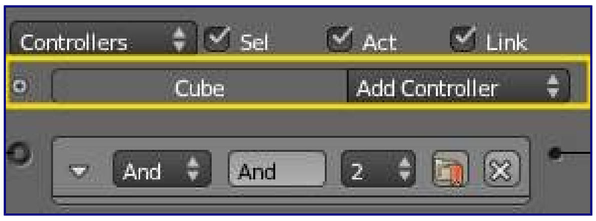
Sel
: Shows all controllers for selected objects.

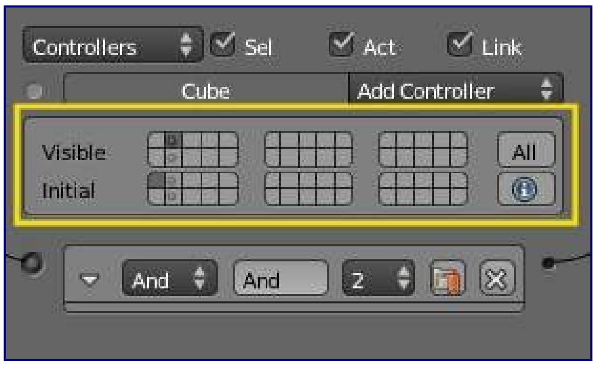
Act
: Shows only controllers belonging to the active object.

Link

Shows controllers which have a link to actuators/sensors.

Object Heading





In the column list, controllers are grouped by object. By default, controllers for every selected object appear in the list, but this may be modified by the column heading filters.

At the head of each displayed object controller list, three entries appear:

(Used States Button) Shows which states are in use for the object. Detailed description of the marked panel is given in *States*.

Name

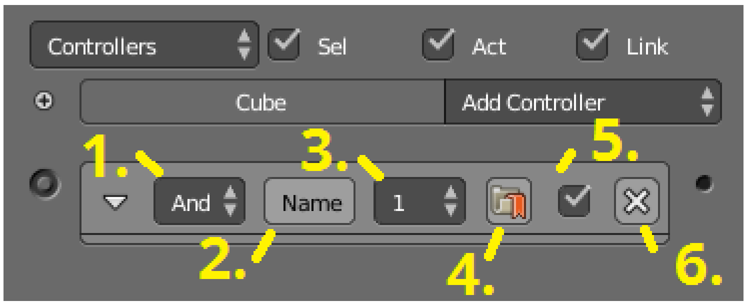
The name of the object.

Add Controller

When clicked, a menu appears with the available controller types. Selecting an entry adds a new controller to the object. See *Controllers* for a list of available controller types.

Standard Controller Parts

The controller heading is standard to every controller.



1. **Controller Type menu**

   Specifies the type of the controller.

2. **Controller Name**

The name of the controller. This can be selected by the user. It is used to access controllers with python; it needs to be unique among the selected objects.

3. **State Index**

   Sets the designated state for which this controller will operate.

4. **Preference Button**

   If on, this controller will operate before all other non-preference controllers (useful for start-up scripts).

5. **Active Checkbox**

   When unchecked the controller is deactivated, no pluses will be sent to the connect actuators.

6. **X Button**

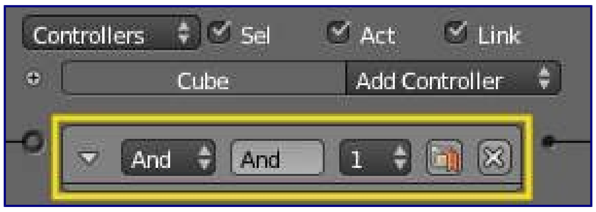
   Deletes the sensor.

Controller Types

AND Controller

This controller gives a positive (TRUE) output when All its inputs are TRUE, and The object is in the designated State. For all other conditions the controller gives a negative (FALSE) output.

Options:



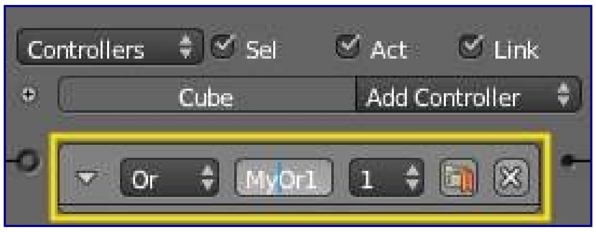
AND Controller

See standard controller parts for descriptions of the remaining options.

OR Controller

This controller gives a positive (TRUE) output when Any one or more of its inputs are TRUE, and The object is in the designated State. For all other conditions the controller gives a negative (FALSE) output.

Options:



OR Controller

See standard controller parts for descriptions of the remaining options.

NAND Controller

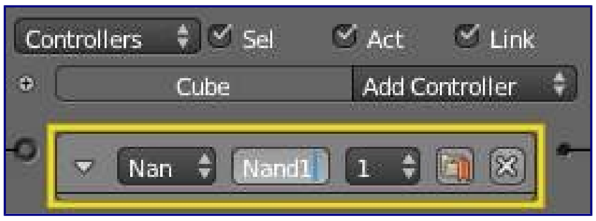
This controller **activates** all connected actuators if

- the game object is in the designated state
- at least one connected sensor triggers the controller
- at least one connected sensor evaluated False

This controller **deactivates** all connected actuators if

- the game object is in the designated state
- at least one connected sensor triggers the controller
- ALL connected sensor evaluated True

Options:



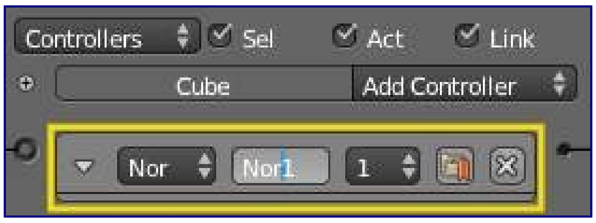
NAND Controller

See standard controller parts for descriptions of the remaining options.

NOR Controller

This controller gives a positive (TRUE) output when None of its inputs are TRUE, and The object is in the designated State. For all other conditions the controller gives a negative (FALSE) output.

Options:



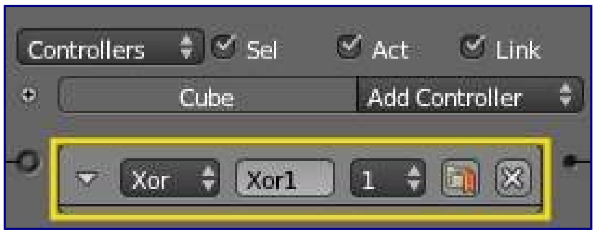
NOR Controller

See standard controller parts for descriptions of the remaining options.

XOR Controller

This controller gives a positive (TRUE) output when One (and only one) of its inputs are TRUE, and The object is in the designated State. For all other conditions the controller gives a negative (FALSE) output.

Options:



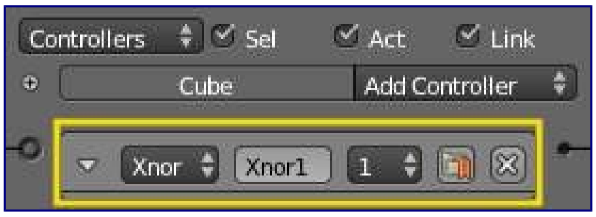
XOR Controller

See standard controller parts for descriptions of the remaining options.

XNOR Controller

This controller gives a positive (TRUE) output when One (and only one) of its inputs are FALSE, and The object is in the designated State. For all other conditions the controller gives a negative (FALSE) output.

Options:

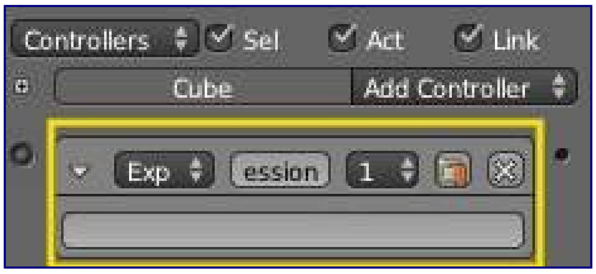


XNOR Controller

See standard controller parts for descriptions of the remaining options.

Expression Controller

This controller evaluates a user written expression, and gives a positive (TRUE) output when The result of the expression is TRUE, and The object is in the designated State. For all other conditions the controller gives a negative (FALSE) output.



Expression Controller

Expression The expression, which is written in the box, can consist of variables, constants and operators. These must follow the rules laid out below.

Variables

You can use:

- **sensors names**,
- **properties** : assign a game property to an object and use it in a controller expression.

These cannot contain blank spaces.

Operations

Mathematical operations

Operators: `*`, `/`, `+`, `-`

Returns: a number

Examples: `3 + 2`, `35 / 5`

Logical operations

- Comparison operators: `<`, `>`, `>=`, `<=`, `==`, `!=`
- Booleans operators: `AND`, `OR`, `NOT`

Returns: `True` or `False`.

Examples: `3 > 2 (True)`, `1 AND 0 (False)`

Conditional statement (if)

Use:

```
if( expression, pulse_if_expression_is_true, pulse_if_expression_is_false )
```

If the controller evaluates `expression` to True:

- if `pulse_if_expression_is_true` is `True`, the controller sends a positive pulse to the connected actuators.
- if `pulse_if_expression_is_true` is `False`, the controller sends a negative pulse to the connected actuators.

If the controller evaluates `expression` to False:

- if `pulse_if_expression_is_false` is `True`, the controller sends a positive pulse to the connected actuators.
- if `pulse_if_expression_is_false` is `False`, the controller sends a negative pulse to the connected actuators.

Examples

Given the object has a property `coins` equal to 30:

```
coins > 20
```

returns True (the controller sends a positive pulse to the connected actuators).

Given the object has:

- a sensor called `Key_Inserted` equal to True,
- a property named `Fuel` equal to False,

```
Key_Inserted AND Fuel
```

returns False (the controller sends a negative pulse to the connected actuators).

This is the same as doing:

```
if (Key_Inserted AND Fuel, True, False)
```

Instead, you could do:

```
if (Key_Inserted AND Fuel, False, True)
```

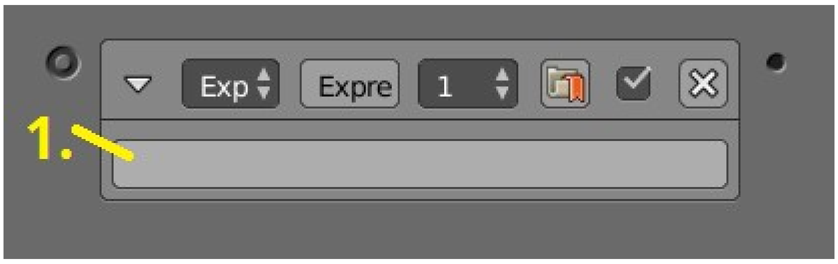
to return a positive pulse when `Key_Inserted AND Fuel` returns False.

You can also do:

```
if ((Key_Inserted AND Fuel) OR (coins > 20), True, False)
```

This expression returns True, hence in this case the controller sends a positive pulse to the connected actuators.

Parts of the Expression Controller



1. Expression

   The Expression to calculate.

See standard controller parts for descriptions of the remaining options.

Python Controller

The python controller runs a python script when a sensor triggers the controller. This python script can interact with the scene or logic bricks through the *Blender game engine API*.

A python script can either run as an entire file or a single module. A file must be added in the text editor, and is identified simply by its name, not its path. Names are case sensitive. Modules are identified by the file name *without* the extension followed by a `.` and then the name of the module. For example:

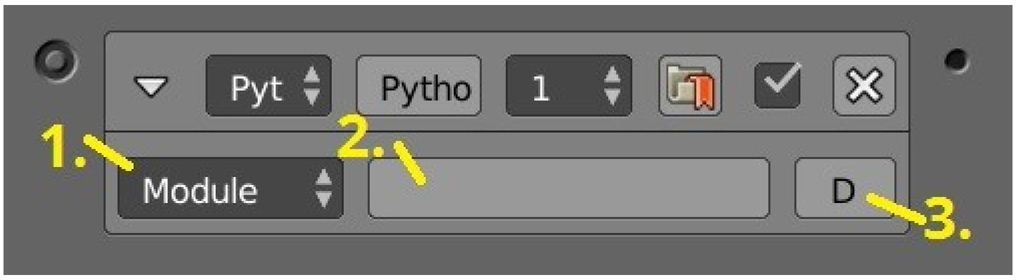
A file `myscript.py` contains:

```
def myModule ():
   print("Go Open Source!");
```

The function can be accessed as `myscript.myModule`, which will run `print("Go Open Source!");` every time the controller is triggered.

The entire file can be run by setting the type to *Script* and setting the name to myscript.py.

Parts of the Python Controller



1. **Type**

   Specifies whether it is a module or entire file.

2. **Name**

   The name of the file to be loaded.

3. **D (Use Debug)**

   Continuously reload the file.

See standard controller parts for descriptions of the remaining options.

More information on the python API can be found *here*.