12.4.2 Game Engine - Logic - Sensors

Sensors

Introduction

Sensors are the logic bricks that cause the logic to do anything. Sensors give an output when something happens, e.g. a trigger event such as a collision between two objects, a key pressed on the keyboard, or a timer for a timed event going off. When a sensor is triggered, a positive pulse is sent to all controllers that are linked to it.

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

The following types of sensor are currently available:

Actuator
: Detects when a particular actuator receives an activation pulse.

Always
: Gives a continuous output signal at regular intervals.

Collision
: Detects collisions between objects or materials.

Delay
: Delays output by a specified number of logic ticks.

Joystick
: Detects movement of specified joystick controls.

Keyboard
: Detects keyboard input.

Message
: Detects either text messages or property values

Mouse
: Detects mouse events.

Near
: Detects objects that move to within a specific distance of themselves.

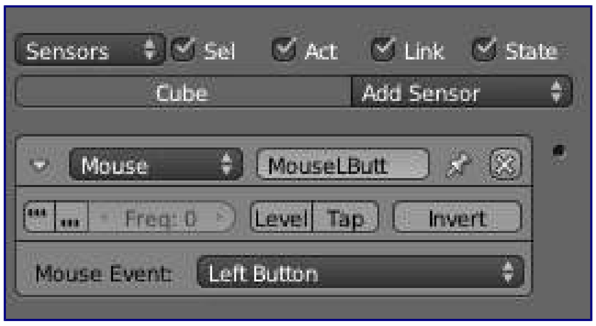
Property
: Detects changes in the properties of its owner object.

Radar
: Detects objects that move to within a specific distance of themselves, within an angle from an axis.

Random
: Generates random pulses.

Ray
: Shoots a ray in the direction of an axis and detects hits.

Sensor Editing

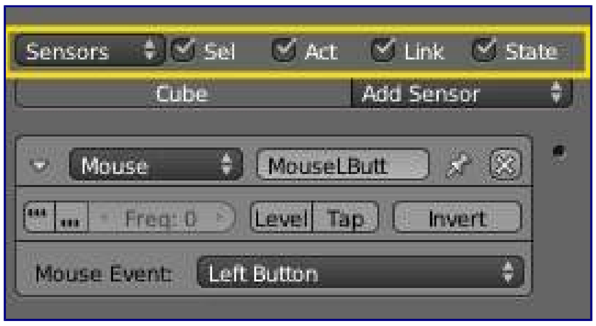


Sensor Column with Typical Sensor

Blender sensors can be set up and edited in the left-hand column of the Logic Panel. This page describes the general column controls, and also those parameters which are common to all individual sensor types.

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

Column Heading



Sensor Column Heading

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

Sensors:

Show Objects
    Expands all objects.
Hide Objects
    Collapses all objects to just a bar with their name.
Show Sensors
    Expands all sensors.
Hide Sensors
    Collapses all sensors to bars with their names.

It is also possible to filter which sensors are viewed using the four heading buttons:

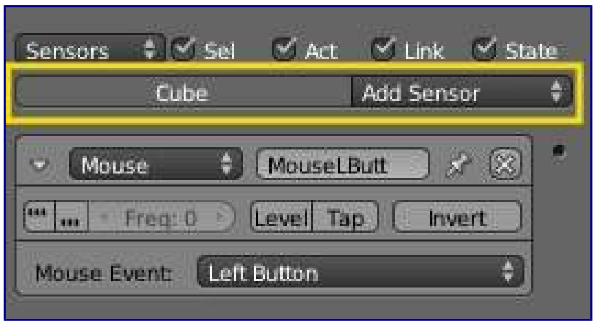
Sel
    Shows all sensors for selected objects.
Act
    Shows only sensors belonging to the active object.
Link

Shows sensors which have a link to a controller.

State

Only sensors connected to a controller with active states are shown.

Object Heading



Sensor Object Heading

In the column list, sensors are grouped by object. By default, sensors 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 sensor list, two entries appear:

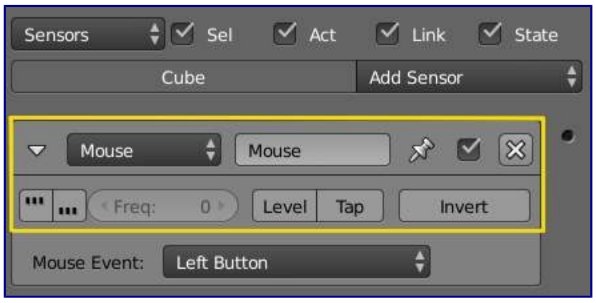
Name

The name of the object.

Add Sensor

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

Sensor Common Options



Common Sensor Options

All sensors have a set of common buttons, fields and menus. They are organized as follows:

Triangle button

Collapses the sensor information to a single line (toggle).

Sensor type menu

Specifies the type of the sensor.

Sensor name

The name of the sensor. This can be selected by the user. It is used to access sensors with Python; it needs to be unique among the selected objects.

Pin button
: Display the sensor even when it is not linked to a visible states controller.

Checkbox button
: Sets active state of the sensor

X button
: Deletes the sensor.

Note

Note about triggers

If a controller does not get trigger by any connected sensor (regardless of the sensors’ state) it will not be activated at all.

A sensor triggers the connected controllers on state change. When the sensor changes its state from negative to positive or positive to negative, the sensor triggers the connected controllers. A sensor triggers a connected controller as well when the sensor changes from deactivation to activation.

The following parameters specifies how the sensor triggers connected controllers:

True level triggering
: If this is set, the connected controllers will be triggered as long as the sensor’s state is positive. The sensor will trigger with the delay (see parameter: frequency) of the sensor.

False level triggering
: If this is set, the connected controllers will be triggered as long as the sensor’s state is negative. The sensor will trigger with the delay (see parameter: frequency) of the sensor.

Freq

Despite it’s name “Frequency”, this parameter sets the delay between repeated triggers, measured in frames (also known as logic ticks). The default value is 0 and it means no delay. It is only used at least one of the level triggering parameters are enabled.

Raising the value of *freq* is a good way for saving performance costs by avoiding to execute controllers or activate actuators more often than necessary.

Examples: (Assuming the default frame rate with a frequency of 60 Hz (60 frames per second)).

freq	meaning	frames with trigger	frames without trigger	period in frames	frequency in frames/sec
0	The sensor triggers the next frame.	1	0	1	60
1	The sensor	1	1	2	30

freq	meaning	frames with trigger	frames without trigger	period in frames	frequency in frames/sec
	triggers at one frame and waits another one until it triggers again. It results in half speed.				
29	The sensor triggers one frame and waits 29 frames until it triggers again.	1	29	30	2
59	The sensor triggers one frame and waits 59 frames until it triggers again.	1	59	30	1

***Level* Button**

Triggers connected controllers when state (of the build-in state machine) changes. (For more information see *States*).

The following parameters specifies how the sensor’s status gets evaluated:

***Tap* Button**

Changes the sensor’s state to to negative one frame after changing to positive even if the sensor evaluation remains positive. As this is a state change it triggers the connected controllers as well. Only one of *Tap* or *Level* can be activated. If the *TRUE level triggering* is set, the sensor state will consecutive change from True to False until the sensor evaluates False. The *FALSE level triggering* will be ignored when the *Tap* parameter is set.

***Invert* Button**

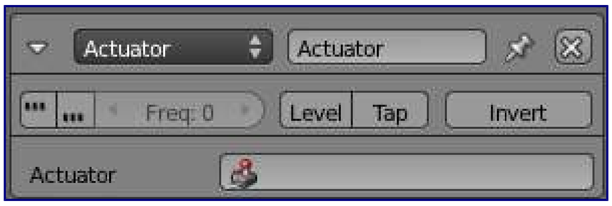
This inverts the sensor output. If this is set, the sensor’s state will be inverted. This means the sensors’s state changes to positive when evaluating False and changes to False when evaluating True. If the *Tap* parameter is set, the sensor triggers the controller based on the inverted sensor state.

Sensor Types

- Actuator Sensor
- Always Sensor
- Collision Sensor
- Delay Sensor
- Joystick Sensor
- Keyboard Sensor

- Message Sensor
- Mouse Sensor
- Near Sensor
- Property Sensor
- Radar Sensor
- Random Sensor
- Ray Sensor

Actuator Sensor



Actuator sensor

The Actuator sensor detects when a particular actuator receives an activation pulse.

The *Actuator* sensor sends a TRUE pulse when the specified actuator is activated.

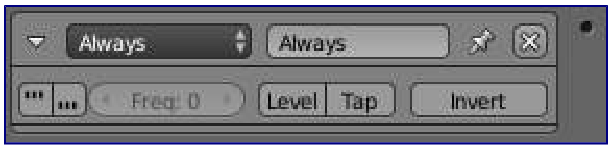
The sensor also sends a FALSE pulse when the specified actuator is deactivated.

See *Sensor Common Options* for common options.

Special Options:

Actuator
Name of actuator (NB This must be owned by the same object).

Always Sensor



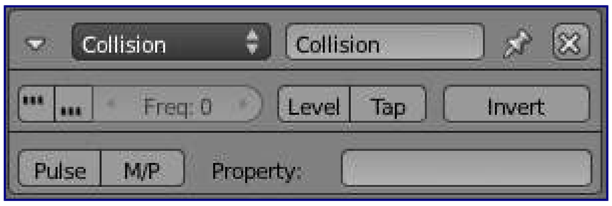
Always sensor

The *Always* sensor is used for things that need to be done every logic tick, or at every *x* logic tick (with non-null *f*), or at start-up (with *Tap*).

See *Sensor Common Options* for common options.

This sensor doesn’t have any special options.

Collision Sensor



Collision sensor

A *Collision* sensor works like a *Touch* sensor but can also filter by property or material. Only objects with the property/material with that name will generate a positive pulse upon collision. Leave blank for collision with any object.

See *Sensor Common Options* for common options.

Special Options:

Pulse button
: Makes it sensible to other collisions even if it is still in touch with the object that triggered the last positive pulse.

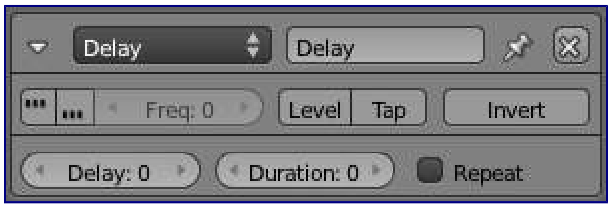
M/P button
: Toggles between material and property filtering.

Note

Note about soft bodies

The *Collision* sensor can not detect collisions with soft bodies. This is a limitation in Bullet, the physics library used by the Game Engine.

Delay Sensor



Delay sensor

The *Delay* sensor is designed for delaying reactions a number of logic ticks. This is useful if an other action has to be done first or to time events.

See *Sensor Common Options* for common options. Special Options:

Delay

The number of logic ticks the sensor waits before sending a positive pulse.

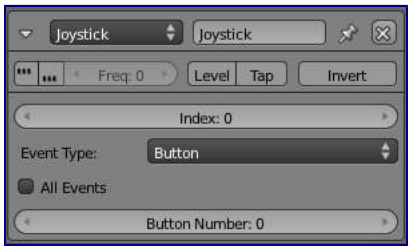
Duration

The number of logic ticks the sensor waits before sending the negative pulse.

Repeat Button

Makes the sensor restart after the delay and duration time is up.

Joystick Sensor



Joystick sensor

The *Joystick* sensor triggers whenever the joystick moves. It also detects events on a range of ancilliary controls on the joystick device (hat, buttons, etc.). More than one joystick may be used (see "Index"). The exact layout of the joystick controls will depend on the make and model of joystick used.

See *Sensor Common Options* for common options.

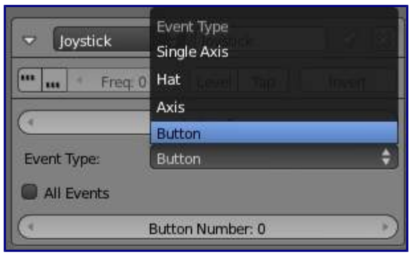
Special Options:

Index

Specifies which joystick to use.

All Events

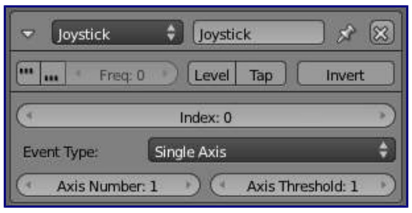
Sensor triggers for all events on this joystick's current type



Joystick Events

Event Type

A menu to select which joystick event to use



Index: 0
Event Type: Single Axis
Axis Number: 1
Axis Threshold: 1

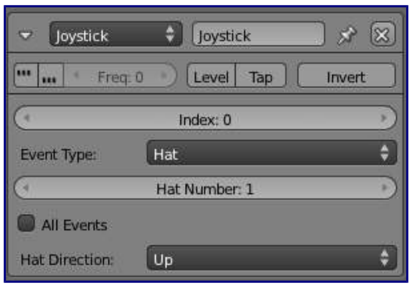
Joystick Single Axis

Single Axis

Detect movement in a single joystick Axis.

Axis Number
: 1 = Horizontal axis (left/right) 2 = Vertical axis (forward/back) 3 = Paddle axis up/down 4 = Joystick axis twist left/right

Axis Threshold
: Threshold at which joystick fires (Range 0 - 32768)



Index: 0
Event Type: Hat
Hat Number: 1
All Events
Hat Direction: Up

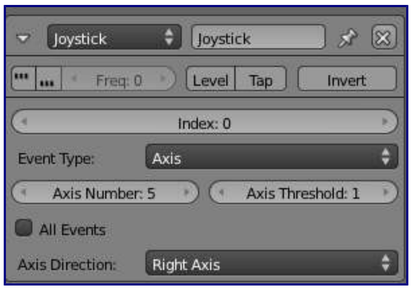
Joystick Hat

Hat

Detect movement of a specific hat control on the joystick.

Hat number
: Specifies which hat to use (max. 2)

Hat Direction
: Specifies the direction to use: up, down, left, right, up/right, up/left, down/right, down/left.



Index: 0
Event Type: Axis
Axis Number: 5
Axis Threshold: 1
All Events
Axis Direction: Right Axis

Joystick Axis

Axis

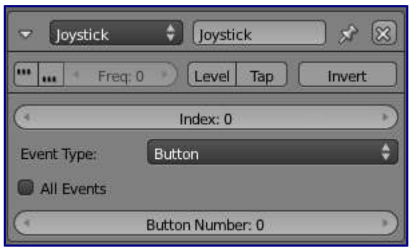
Axis Number
: Specifies the axis (1 or 2)

Axis Threshold
Threshold at which joystick fires (Range 0 - 32768)

Axis Direction

Specifies the direction to use:

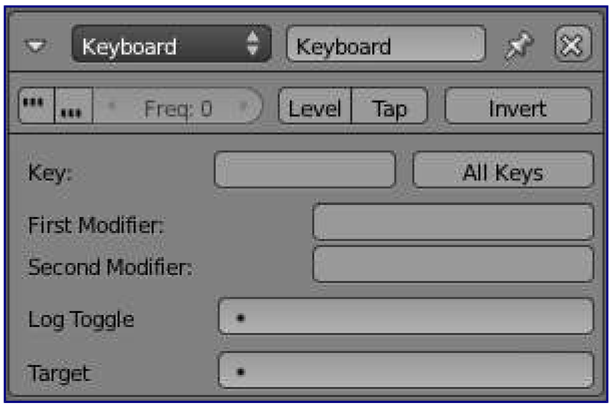
(Axis Number = 1) Joystick Left, Right, Up, Down (Axis Number = 2) Paddle upper (Left); paddle Lower (Right); Joystick twist left (Up) Joystick twist right (Down)



Joystick Button

Button
Specify the *button number* to use.

Keyboard Sensor



Keyboard sensor

The *Keyboard* sensor is for detecting keyboard input. It can also save keyboard input to a String property.

See *Sensor Common Options* for common options.

Special Options:

Key
This field detects presses on a named key. Press the button with no label and a key to assign that key to the sensor. This is the active key, which will trigger the TRUE pulse. Click the button and then click outside of the button to deassign the key.

A FALSE pulse is given when the key is released.

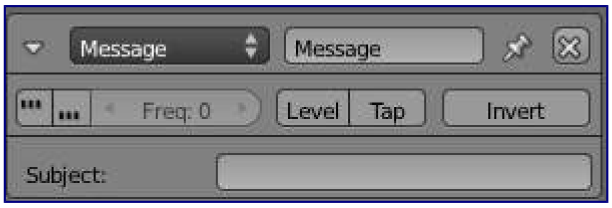
All keys button
Sends a TRUE pulse when any key is pressed. This is useful for custom key maps with a *Python controller.*

First Modifier, Second Modifier
Specifies additional key(s), all of which must be held down while the active key is pressed in order for the sensor to give a TRUE pulse. These are selected in the same way as Key. This is useful if you wish to use key combinations, for example `Ctrl-R` or `Shift-Alt-Esc` to do a specific action.

LogToggle
Assigns a *Bool* property which determines if the keystroke will or will not be logged in the target *String*. This property needs to be TRUE if you wish to log your keystrokes.

Target
The name of property to which the keystrokes are saved. This property must be of type *String*. Together with a *Property* sensor this can be used for example to enter passwords.

Message Sensor



Message Sensor

The *Message* sensor can be used to detect either text messages or property values. The sensor sends a positive pulse once an appropriate message is sent from anywhere in the engine. It can be set up to only send a pulse upon a message with a specific subject.

See *Sensor Common Options* for common options.

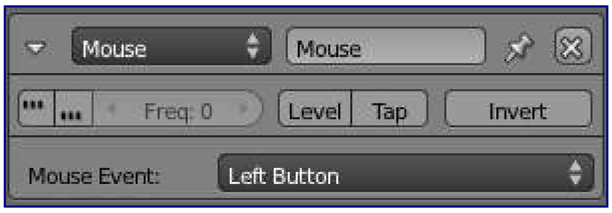
Special Options:

Subject
Specifies the message that must be received to trigger the sensor (this can be left blank).

Note

See *Message Actuator* for how to send messages.

Mouse Sensor



Mouse sensor

The *Mouse* sensor is for detecting mouse events.

See *Sensor Common Options* for common options.

Special Options: The controller consist only of a list of types of mouse events. These are:

Mouse over any
: gives a TRUE pulse if the mouse moves over any game object.

Mouse over
: gives a TRUE pulse if the mouse moves over the owner object.

Movement
: any movement with the mouse causes a stream of TRUE pulses.

Wheel Down
: causes a stream of TRUE pulses as the scroll wheel of the mouse moves down.

Wheel Up
: causes a stream of TRUE pulses as the scroll wheel of the mouse moves up.

Right button
: gives a TRUE pulse.

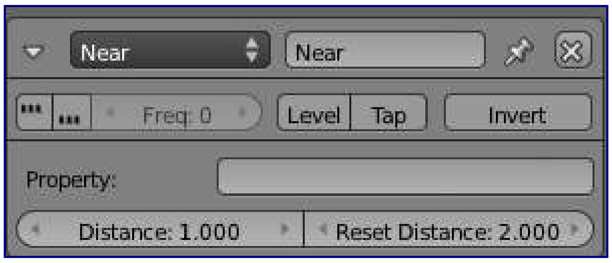
Middle button
: gives a TRUE pulse.

Left button
: gives a TRUE pulse.

A FALSE pulse is given when any of the above conditions ends.

There is no logic brick for specific mouse movement and reactions (such as first person camera), these have to be coded in python.

Near Sensor



Near sensor

A *Near* sensor detects objects that move to within a specific distance of themselves. It can filter objects with properties, like the *Collision* sensor.

See *Sensor Common Options* for common options.

Special Options:

Property
: This field can be used to limit the sensor to look for only those objects with this property.

Distance
: The number of blender units it will detect objects within.

Reset
: The distance the object needs to be to reset the sensor (send a FALSE pulse).

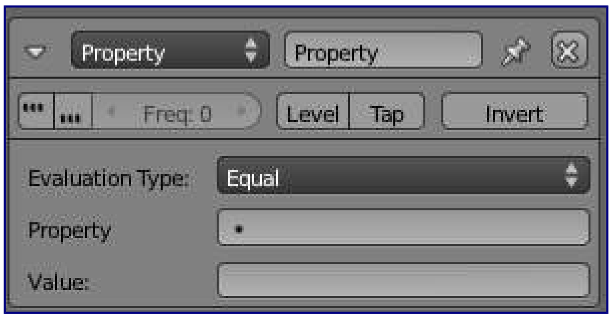
Notes: 1) The Near sensor can detect objects “through” other objects (walls etc). 2) Objects must have “Actor” enabled to be detected.

Note

Note about soft bodies

The *Near* sensor can not detect soft bodies. This is a limitation in Bullet, the physics library used by the Game Engine.

Property Sensor



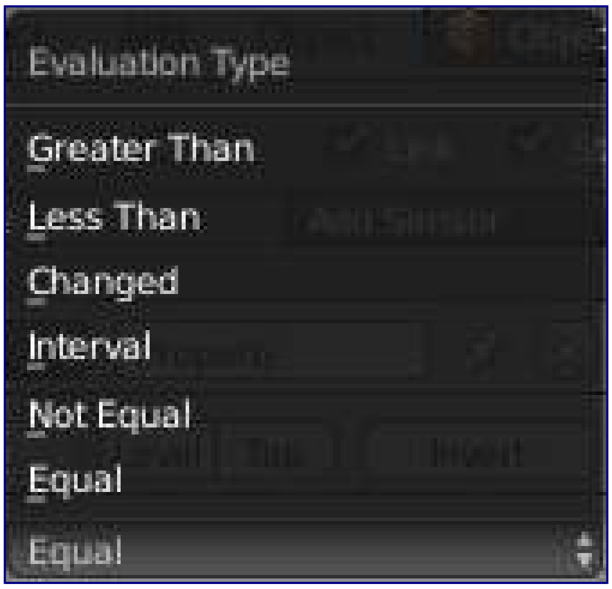
Evaluation Type: Equal
Property
Value:

Property sensor

The *Property* sensor detects changes in the properties of its owner object.

See *Sensor Common Options* for common options.

Special Options:



Property Evaluation

Evaluation Type
: Specifies how the property will be evaluated against the value(s).

Greater Than
: Sends a TRUE pulse when the property value is greater than the *Value* in the sensor.

Less Than
: Sends a TRUE pulse when the property value is less than the *Value* in the sensor.

Changed
: Sends a TRUE pulse as soon as the property value changes.

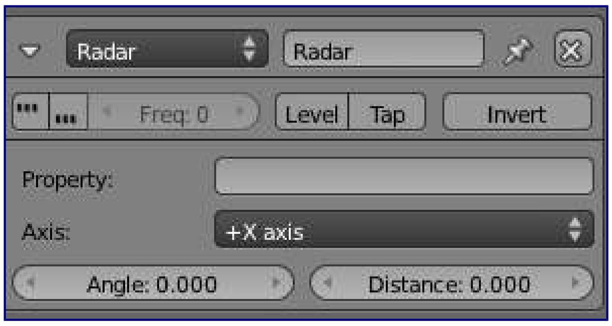
Interval
: Sends a TRUE pulse when the *Value* of the property is between the *Min* and *Max* values of the sensor.

Not Equal
: Sends a TRUE pulse when the property value differs from the *Value* in the sensor.

Equal
: Sends a TRUE pulse when the property value matches the *Value* in the sensor.

Note the names of other properties can also be entered to compare properties.

Radar Sensor



Radar sensor

The *Radar* sensor works much like a *Near* sensor, but only within an angle from an axis, forming an invisible cone with the top in the objects’ center and base at a distance on an axis.

See *Sensor Common Options* for common options.

Special Options:

Property
This field can be used to limit the sensor to look for only those objects with this property.

Notes: 1) The Radar sensor can detect objects “through” other objects (walls etc). 2) Objects must have “Actor” enabled to be detected.

Axis
This menu determines the direction of the radar cone. The ± signs is whether it is on the axis direction (+), or the opposite (-).

Angle
Determines the angle of the cone. (Range: 0.00 to 179.9 degrees).

Distance
Determines the length of the cone. (Blender units).

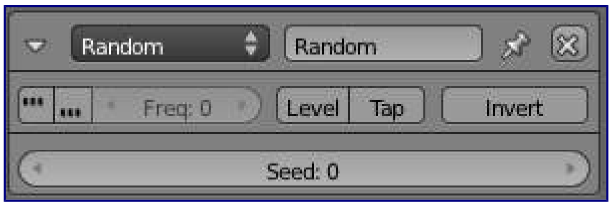
This sensor is useful for giving bots sight only in front of them, for example.

Note

Note about soft bodies

The *Radar* sensor can not detect soft bodies. This is a limitation in Bullet, the physics library used by the Game Engine.

Random Sensor



Random sensor

The *Random* sensor generates random pulses.

See *Sensor Common Options* for common options.

Special Options:

Seed This field to enter the initial seed for the random number algorithm. (Range 0-1000).

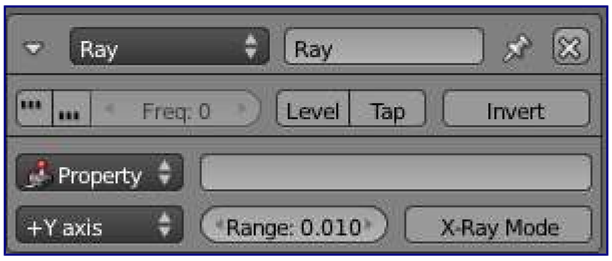
Note

0 is not random, but is useful for testing and debugging purposes.

Note

If you run several times with the same Seed, the sequence of intervals you get will be the same in each run, although the intervals will be randomly distibuted.

Ray Sensor



Ray sensor

The *Ray* sensor shoots a ray in the direction of an axis and sends a positive pulse once it hits something. It can be filtered to only detect objects with a given material or property.

See *Sensor Common Options* for common options.

Special Options: It shares a lot of buttons and fields with *Radar* sensor.

Property
This field can be used to limit the sensor to look for only those objects with this property.

Note

1. Unless the Property field is set, the Ray sensor can detect objects “through” other objects (walls etc).
2. Objects must have “Actor” enabled to be detected.

Axis
This menu determines the direction of the ray. The ± signs is whether it is on the axis direction (+), or the opposite (-).

Range
Determines the length of the ray. (Blender units).

X-Ray Mode button
Makes it x-ray, so that it sees through objects that don’t have the property or material specified in the filter field.