LINE-OF-SIGHT BEHAVIOR

View online: https://www.construct.net/en/make-games/manuals/construct-3/behavior-reference/line-of-sight

The **Line-of-sight (LOS) behavior** allows the ability to check if two objects can "see" each other. More precisely, it will check if there are any obstacles blocking a line between the two objects. Click here to open an example of Line-of-sight.

Line-of-sight can also perform *Raycasting*. Normal line-of-sight checks there are no obstacles in a straight line between two objects. With raycasting, if there is an obstacle in the way, you can find the exact position of the obstacle in the way, as well as the surface normal and angle of reflection. The Instant hit laser example provides a demonstration of how to use raycasting.

Scripting

When using JavaScript or TypeScript coding, the features of this behavior can be accessed via the ILOSBehaviorInstance script interface.

Line-of-sight properties

Obstacles

Whether to use *Solids* as blocking line-of-sight, or *Custom*, where the objects blocking line-of-sight must be added using the *Add obstacle* action.

Range

The maximum distance in pixels that line-of-sight can reach. If an object is further away than this distance, the object will never have line-of-sight to it, even if the intervening space is clear.

Cone of view

The angle of the cone of view in which the object can have line-of-sight to other objects, relative to the current angle of the object. For example if this is 180, then the object can have line-of-sight to any objects anywhere in front of it, but never behind it. If 360, the object can have line-of-sight to objects at any angle.

Use collision cells

Whether to use the collision cells optimisation when testing line of sight. Usually this is faster, but in some cases over extremely long distances it can be slower.

Line-of-sight conditions

Has LOS to object

Check if the object currently has line-of-sight to another object. For the condition to be true, the object must be within range, within the cone of view, and with no obstacles in the way of a straight line between the two objects. This condition also picks the instances of the chosen object that are in the line of sight. By default this checks if there is line-of-sight to the object origin, but you can optionally specify an image point instead.

Has LOS to position

Check if the object currently has line-of-sight to a position in the layout. For the condition to be true, the object must be within range, within the cone of view, and with no obstacles in the way of a straight line between the two objects.

Has LOS between positions

Check if there is line-of-sight between any two positions in the layout, instead of using the object's own position.

Ray intersected

Use after a *Cast ray* action to determine if the ray intersected with any obstacle.

Line-of-sight actions

Add obstacle

If the *Obstacles* property is *Custom*, adds an object type to count as an obstruction to line-of-sight.

Clear obstacles

If the *Obstacles* property is *Custom*, clears any object types added as obstacles with the *Add obstacle* action.

Set cone of view

Set range

Sets the corresponding behavior properties. For more information, see *Line-of-sight properties*.

Cast ray

Check for obstacle intersection between any two positions in the layout. This sets the *Ray intersected* condition true or false depending on if the ray intersected an obstacle; typically this can be checked in a sub-event after the action. If the ray did intersect an obstacle, the hit, normal and reflection expressions are set according to the obstacle that was hit.

This action ignores the range and cone of view properties, to allow raycasting anywhere in the layout.

Line-of-sight expressions

ConeOfView

Range

Retrieve the corresponding behavior properties. For more information, see *Line-of-sight properties*.

HitX

HitY

If *Ray intersected* is true, the position of the first obstacle the ray intersected, in layout co-ordinates.

HitDistance

If *Ray intersected* is true, the distance between the ray start point and the hit position.

HitUID

If *Ray intersected* is true, the UID of the instance that was the first obstacle the ray intersected.

NormalAngle

If *Ray intersected* is true, the angle of the surface normal at the point of intersection, in degrees.

NormalX(length)

NormalY(length)

If *Ray intersected* is true, return a position at a given distance along the surface normal vector.

ReflectionAngle

If *Ray intersected* is true, the angle of the reflection at the point of intersection, in degrees.

ReflectionX(length)

ReflectionY(length)

Reflection X (length)

If *Ray intersected* is true, return a position at a given distance along the reflection vector.