PATHFINDING BEHAVIOR SCRIPT INTERFACE

View online: https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/behavior-interfaces/pathfinding

---

The `IPathfindingBehaviorInstance` interface derives from IBehaviorInstance to add APIs specific to the Pathfinding behavior.

An additional `IPathfindingMap` interface is also used to represent the pathfinding map, i.e. grid of obstacles, which is shared between all Pathfinding behavior instances using the same cell size and cell border settings.

Examples

See the Scripting pathfinding example for a demonstration of using these APIs to find and display a path around obstacles.

Pathfinding behavior events

See behavior instance event for standard behavior instance event object properties.

"arrived"

Fired when a moving object comes to a stop at its destination.

Pathfinding behavior APIs

---

map

The `IPathfindingMap` interface representing this behavior instance's pathfinding map, such as where obstacles are. See the documentation on `IPathfindingMap` below.

---

async findPath(x, y)

Starts calculating a path to the given position in layout co-ordinates, and returns a promise that resolves when the path has been calculated. The promise resolves with a boolean indicating whether a path was found.

---

async calculatePath(fromX, fromY, toX, toY)

Calculates a path between any two positions, and returns a promise that resolves when the path has been calculated. The promise resolves with a boolean indicating whether a path was found. This can be used for pure pathfinding calculations without taking in to account

the state of the instance: `findPath()` is designed for use with the instance movement and so will only allow one path starting at the object position to be calculated at a time, but `calculatePath` allows any number of paths to be calculated for any positions at any time.

> *Note that when the promise resolves the path nodes should be read immediately, as the path will be overwritten when a subsequent call to `findPath()` or `calculatePath()` resolves.*

---

startMoving()

Automatically start moving the object along the found path. This can only be used after a path has been successfully found.

---

stop()

If the object is moving along its path, causes it to stop.

---

maxSpeed

Set or get the maximum speed in pixels per second the object can move at, for use with *startMoving()*.

---

speed

Set or get the current speed of the object if it is currently moving along its path, in pixels per second. This cannot be negative or greater than *maxSpeed*.

---

acceleration

deceleration

Set or get the acceleration and deceleration rates in pixels per second per second, for use with *startMoving()*.

---

rotateSpeed

Set or get the rate at which the object can rotate in radians per second, for use with *startMoving()*. Note this can affect the speed of the object: if the rotation speed is low, the object will have to slow down on tight corners.

---

isCalculatingPath

A read-only boolean indicating if a path is being calculated, e.g. via *findPath()*.

---

isMoving

A readonly boolean indicating if the object is currently moving along its path after calling *startMoving()*. It is set back to false after the object arrives at its destination.

---

currentNode

A read-only number indicating the zero-based index of the node the object is currently moving towards, while *isMoving* is true. This may skip ahead just before the object actually reaches the next node, in order to help it round corners.

---

getNodeCount()

Returns the number of nodes in the path that was found, after a path has been successfully found.

---

getNodeXAt(i)

getNodeYAt(i)

getNodeAt(i)

Return the position of a node in the path that was found, in layout co-ordinates, using the zero-based index of the node. This is only available after a path has been successfully found. The `getNodeAt()` variant returns `[x, y]`.

***nodes()**

Iterates all nodes in the path that was found. This returns the same information as `getNodeAt()` but as a generator, yielding values of the form `[x, y]`.

---

directMovementMode

Set or get a string of one of `"none"`, `"to-destination"` or `"anywhere-along-path"` reflecting the direct movement mode. For more information about the effect of each mode, see the *Direct movement* property in the Pathfinding behavior manual entry.

---

isEnabled

A boolean indicating if the behavior is enabled. If disabled, the behavior no longer has any effect on the object.

IPathfindingMap interface

This interface is accessed via the `map` property of the Pathfinding behavior script interface, providing access to details such as the grid of obstacles.

---

cellSize

cellBorder

Read-only numbers with the corresponding Pathfinding behavior properties for this map.

---

widthInCells

heightInCells

Read-only numbers with the current size of the pathfinding map in cells.

---

isCellObstacle(x, y)

Returns a boolean indicating if a cell in the obstacle grid is marked as an obstacle. This is useful for debugging or displaying the obstacle grid. Note the position is taken in cell co-ordinates rather than layout co-ordinates.

---

isDiagonalsEnabled

Set or get a boolean indicating whether paths moving along diagonals are allowed. If disabled, the result nodes along paths will only ever change at 90-degree angles (up, right, down and left). If enabled nodes can move along diagonals as well.

---

moveCost

Set or get an integer of the base path cost for moving a single cell. This affects the relative cost of additional costs added by other features. The default is 10. The move cost is rounded to an integer, and it is multiplied by the square root of 2 for the diagonal move cost if diagonals are enabled.

---

async regenerateMap()

Determine whether each cell in the obstacles grid is an obstacle again. This is a very CPU intensive action and should not be used regularly. If only part of the obstacle map has changed, prefer to use *regenerateRegion()* or *regenerateObjectRegion()*. Returns a promise indicating when the regeneration has finished. Note finding paths before the promise has resolved will not use the updated map.

---

async regenerateRegion(startX, startY, endX, endY)

async regenerateObjectRegion(objectClass)

As with *regenerateMap()*, but only the specified area is updated. This is usually considerably faster than regenerating the entire map. However as with regenerating the entire obstacle map, changes only take effect after the returned promise resolves. *regenerateRegion()* takes a rectangle in layout co-ordinates to regenerate. *regenerateObjectRegion()* similarly regenerates the rectangle in the layout given by the bounding boxes of the instances of an IObjectClass. Note this can cover multiple rectangles if there are multiple instances.

---

startPathGroup(baseCost = 1, cellSpread = 1, maxWorkers = 1)

endPathGroup()

Start and end a *path group*, which can be used to spread out the paths found inside the group. For more information refer to the corresponding *Start path group* and *End path group* actions in the Pathfinding behavior manual entry.