# PATHFINDING BEHAVIOR

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

The **Pathfinding behavior** uses the A* pathfinding algorithm to efficiently find a short path around obstacles. It can either report the path as a list of nodes through expressions, or automatically move the object along the determined path.

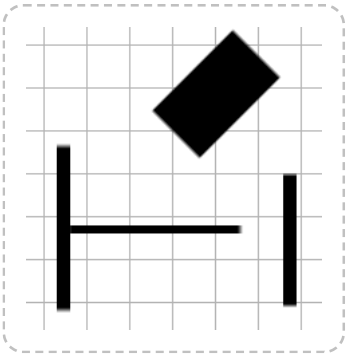
Click here to open an example of the Pathfinding behavior to see how it can be used. Search for *Pathfinding* in the Start Page to find an additional example.

### Scripting

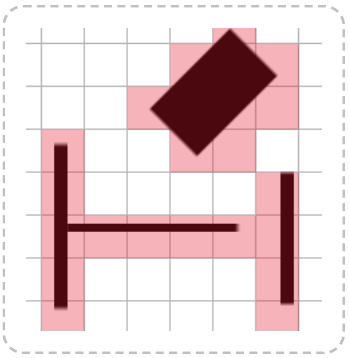
When using JavaScript or TypeScript coding, the features of this behavior can be accessed via the IPathfindingBehaviorInstance script interface.

## The pathfinding grid

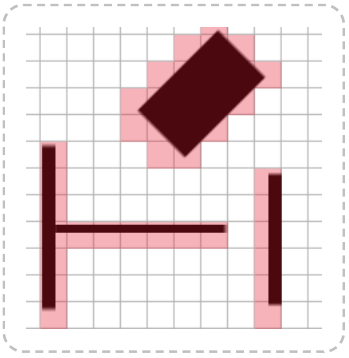
The pathfinding behavior works based on dividing the layout in to a grid. Since pixel-perfect pathfinding can be extremely slow to process, dividing the layout in to cells makes the pathfinding enormously more efficient. The cell size can be set in the behavior property, and the larger it is the more efficient pathfinding is. However setting a large cell size can cause problems: a cell can only be entirely obstacle or entirely free, and using large cells can close up small gaps. For example take the following arrangement of obstacles using a cell size of 32:



It appears that objects should be able to freely move around in between these objects. However if the cells that the pathfinding behavior marks as obstacle are highlighted in red, we see this:

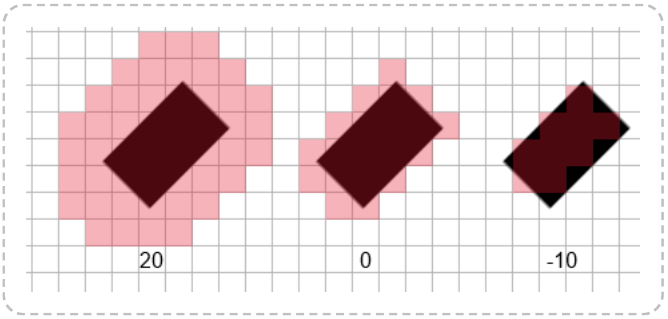


Some of the gaps have been closed off due to the cell size being relatively large compared to the size of the gap. This will make the pathfinding behavior route paths entirely around the obstacles, and never through them. We can help fix this by reducing the cell size to 20:



Now we can see that the Pathfinding behavior will be able to find routes between these obstacles. However, the smaller cell size will make the pathfinding more CPU intensive. Generally, try to use the largest cell size that does not cause problems navigating around obstacles.

The **Cell border** property can adjust how cells are marked as obstacle. If the border is larger than 0, then cells close to obstacles but not actually touching may also be marked as obstacles, effectively giving an extra "obstacle border". If the border is negative, cells only just touching an obstacle may *not* be marked as an obstacle, effectively shrinking the obstacle area inwards. The image below demonstrates the effect of different cell border values when using a cell size of 20.




For best efficiency, **use the same cell size and border** for all objects using the Pathfinding behavior in a layout. If different objects use different values, then the Pathfinding behavior must generate multiple obstacle grids in memory, and pathfind along them separately. You should also **avoid pathfinding every tick**, since this will cause extremely high CPU usage and also increase the amount of time it takes for other objects to determine their paths.

The grid of obstacles is only determined once on startup. If objects are moved in the layout, the pathfinding grid is **not** updated, and objects will continue to pathfind as if the objects were in their old positions. To update the entire obstacle grid use the *Regenerate obstacle map* action, but note this is a very CPU-intense operation and should only be done on one-off occasions. It is much more efficient to update only small parts of it (ideally only the area that has changed), which can be done with the *Regenerate region* and *Regenerate region around object* actions.

The obstacle map also applies to all instances using the Pathfinding behavior - different instances do not store separate obstacle maps, and so different per-instance obstacle settings

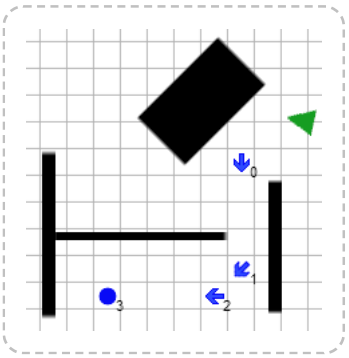
will not take effect.

Note all cells outside the layout area are always obstacles. Areas outside the layout area cannot be included in the pathfinding grid, since doing so would require an infinite amount of memory.

## Finding paths

Calculating a path can take a long time, especially if the cell size is small. To prevent this reducing the game's framerate, the paths are calculated in the background (using a Web Worker). This means after using the *Find path* action, the resulting path is **not** immediately available. You must wait for the *On path found* trigger to run. Only then can you move the object along the path, or access the list of nodes from the behavior's expressions. The game may continue to run for a fraction of a second in between *Find path* and *On path found*.

The result path is a sequence of nodes along the grid. The image below demonstrates a four-node path (nodes 0 to 3).



The nodes can be retrieved (only after *On path found*) using the *NodeCount* and *NodeXAt*/*NodeYAt* expressions. Alternatively, the *Move along path* action can be used to automatically move the object along the nodes, using the speed, acceleration and rotation rate set in the behavior's properties.

Note it may be impossible to find a path, such as trying to navigate to a destination inside a ring of obstacles. In this case, *On failed to find path* will be triggered instead of *On path found*.

If you ask the pathfinding behavior to pathfind to a destination inside an obstacle, it will simply find the nearest clear cell and pathfind to there instead.

## Pathfinding properties

**Cell size**

The cell size, in pixels, of the grid of obstacles. See above for more details about how this is used.

**Cell border**

The amount, in pixels, to expand the cell size by when testing for obstacles. See above for more details about how this is used.

### Obstacles

If *Solids*, the behavior will automatically mark cells touching objects with the Solid behavior as being obstacles. If *Custom*, you must define which objects are obstacles by using the *Add obstacle* action on startup.

> *Note this applies to all instances, since the obstacle map is shared. This setting cannot be used to affect individual instances differently.*

### Max speed

If the *Move along path* action is used, the maximum speed in pixels per second the object can move at.

### Acceleration

If the *Move along path* action is used, the acceleration rate in pixels per second per second.

### Deceleration

If the *Move along path* action is used, the deceleration rate in pixels per second per second, used when approaching the final node.

### Rotate speed

If the *Move along path* action is used, the rate at which the object can rotate in degrees per second. 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.

### Rotate object

Whether to automatically set the angle of the object with the behavior to the angle of motion.

### Diagonals

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.

### Direct movement

Specifies where path nodes may be removed if the box of cells enclosing nodes is completely clear. The options are:

- **None:** no path nodes are ever removed. This means even paths across entirely clear areas will still place nodes for movement along the pathfinding grid.

- **To destination:** if the area enclosing both the start and end positions of a path are completely clear, then this removes all path nodes except the last, allowing direct movement from the start to the destination. However if the area is not completely clear, it will still add path nodes to move along the pathfinding grid.

- **Anywhere along path:** checks for any groups of nodes along the whole path where the surrounding area is completely clear, and removes nodes so the path goes directly across the clear area. This can provide a smoother path with fewer nodes and fewer turns.

See the Pathfinding direct movement example for a visualization of how the setting changes paths.

---

**Enabled**

Whether the behavior is initially enabled or disabled. If disabled, it can be enabled at runtime using the *Set enabled* action.

## Pathfinding conditions

---

**Compare speed**

If moving along a path, compare the current speed of the object in pixels per second.

---

**Diagonals are enabled**

True if the *Diagonals* property allows moving diagonally along cells. This can also be changed with the *Set diagonals enabled* action.

---

**Is calculating path**

True if the object is currently calculating a path in the background. This is true between the *Find path* action and the *On path found* or *On failed to find path* triggers.

---

**Is cell obstacle**

Test 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.

---

**Is enabled**

Test if the behavior is currently enabled. When disabled it will have no effect on the object.

---

**Is moving along path**

True after using the *Move along path* action until *On arrived* triggers.

---

**On arrived**

Triggered after *Move along path* when the object finally arrives at its destination.

---

**On failed to find path**

Triggered after the *Find path* action if no path can be found to the destination, such as if it is surrounded by a ring of obstacles.

---

**On path found**

Triggered after the *Find path* action once a path has successfully been found to the destination. The nodes are now available via the *NodeCount*, *NodeXAt* and *NodeYAt* expressions, and the *Move along path* action can also be used.

## Pathfinding actions

---

**Add obstacle**

If the *Obstacles* property is *Custom*, add an object type to mark as an obstacle in the pathfinding grid. If this is done during the game (after *Start of layout*), you must also use *Regenerate obstacle map* for it to take effect.

> *Note this applies to all instances of both object types involved. This action does not affect individual instances.*

---

**Add path cost**

Add an object to increase the path cost in the pathfinding grid. This can be used to simulate rough terrain - the behavior will try to find paths around them if possible, unless the route is a major shortcut. See the *Pathfinding path cost* in the Start Page for a demo. If this is done during the game (after *Start of layout*), you must also use *Regenerate obstacle map* for it to take effect.

---

**Clear cost**

Remove all path cost objects added with *Add path cost*. You must also use *Regenerate obstacle map* for this to take effect.

---

**Clear obstacles**

Remove all obstacle objects added with *Add obstacle*. You must also use *Regenerate obstacle map* for this to take effect.

**Find path**

Start calculating a path to a destination in the layout. This is processed in the background and the results are not immediately ready after this action; you must wait until the *On path found* or *On failed to find path* triggers run before the result is known or the path can be moved along. If this action is used while *Is calculating path* is true, the old path is still calculated and the result triggered, but it then immediately begins calculating the new path and will also trigger for that result.

**Regenerate obstacle map**

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 one of the *Regenerate region* actions. Any changes made by using *Add obstacle*, *Clear obstacles*, *Add path cost* and *Clear cost* will take effect the next tick after this action. Note this means if you attempt to find a path immediately after this action, the obstacle map won't have been updated yet; add a *Wait* action with a short delay to make sure the updated map is used in that case.

**Regenerate region**

**Regenerate region around object**

As with *Regenerate obstacle map*, 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 next tick. *Regenerate region* takes a rectangle in layout co-ordinates to regenerate. *Regenerate region around object* similarly regenerates the rectangle in the layout given by an object's bounding box. Note if multiple instances have met the event's conditions, this will regenerate multiple rectangles (one for every picked object).

**Set enabled**

Set whether the behavior is enabled or disabled. If disabled, it will not calculate any paths or move the object.

**Set move cost**

Set the base path cost for moving a single cell. This affects the relative cost of additional costs added by other features such as the *Add path cost* action and path groups. 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.

**Start path group**

**End path group**

Start and end a *path group*, which can be used to spread out paths found while inside the group. When a path is found inside the group, it adds a cost to cells along the discovered path in order to discourage subsequent paths in the same group from following the same path. Once the path group is ended, all added costs from the group are removed and normal pathfinding is restored. The parameters that affect path groups are:

- **Base cost:** The cost to add for each cell along a found path. Higher costs will spread out paths more. Paths are found across multiple independent workers which don't include the costs added by paths found in other workers; in order to compensate for this, the base cost is multiplied by the number of workers.

- **Cell spread:** How many cells around the found path to add the cost to. A higher cell spread will cause paths to be spread out more. Prefer to use an odd number for a symmetrical spread around the path. For example a cell spread of 1 only adds costs to the cells directly along the path, and a cell spread of 3 will add costs to every 3x3 box of cells along the path.

- **Max workers:** The maximum number of Web Workers allowed to be used for finding paths inside this path group. As each worker is independent and doesn't see the costs added from paths found in other workers, using multiple workers can still result in some paths being found along the same route. Limiting to 1 worker avoids this happening, but also can significantly reduce performance, as it will not use all available CPU cores for pathfinding. Using a higher number improves performance, but can still result in some repeat paths. This number can only reduce the number of workers used (i.e. if it is higher than the number of CPU cores it will have no effect).

See the Pathfinding groups example for a visualization of the effect of spreading out paths with this feature.

---

### Move along path

Automatically start moving the object along the found path. This can only be used after *On path found* - the path is not immediately known after the *Find path* action.

> *You can also use the Move To behavior's Move along Pathfinding path action as an alternative, since the Move To behavior uses a different movement algorithm.*

---

### Set speed

Set 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 the maximum speed of the behavior.

---

### Stop

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

---

**Set acceleration**

**Set deceleration**

**Set diagonals enabled**

**Set max speed**

**Set rotate speed**

**Set direct movement**

Set the corresponding behavior properties. See the property definitions above for more information.

## Pathfinding expressions

---

**CurrentNode**

When moving along a path, the zero-based index of the node the object is currently moving towards. This may skip ahead just before the object actually reaches the next node, in order to help it round corners.

---

**MovingAngle**

The current angle of motion when moving along a path, in degrees.

---

**NodeCount**

The number of nodes in the path that was found. This is only updated after *On path found*.

---

**NodeXAt**

**NodeYAt**

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 *On path found*.

---

**Speed**

The current speed in pixels per second when moving along a path.

---

**Acceleration**

**CellSize**

**Deceleration**

**MaxSpeed**

**RotateSpeed**

Return the current values of the behavior properties. For more information, see the property definitions above.