Node Graph Tutorial

How it Works

Without a node graph, the bot cannot navigate a level. The graph consists of a set of points with 3D coordinates. By comparing its ingame position to the nodes' positions, the bot knows where it is in the map and how to go elsewhere. Each node can store up to 10 links to other nodes. It is only along these links that the bot can walk; from node to node.

Links are one-way by themselves, but two nodes can certainly be linked to each other to make a two-way link. There are 5 types of links: 1, standard (purple); 2, teleport (green); 3, jump (pink); 4, crouch (orange); and 5, look-ahead (red). Teleport links tell the path finding algorithm the effective distance is 0, meaning the bot may prefer the teleporter to other paths. Jump, crouch, and look-ahead force the bot to perform those actions as it follows the link.

There's nothing stopping you from making a link between two nodes on opposite sides of a wall. Make sure you don't place links where obstacles in the map are, like scenery objects or vehicle spawns. If the bot is moving too slowly on its path, it will assume it has hit an obstacle and will try to jump over it.

The Basics

It takes about an hour of work to cover a medium-to-large map. Once you know the shortcut commands, it's pretty easy. If you type *help* into the Node Graph tab's command box, you'll get a list of all the possible commands. I'll go over the graph construction ones here in greater detail:

add
: Place a new node at your current ingame position. Nodes are placed at head height. You can also create new nodes ingame by pressing the INSERT key. Each node is automatically assigned a number which is used to refer to it in commands.

delete <#>
: Delete the specified node. Not only is the node removed, but so are any links to it from other nodes. Note that the next node you add will take a deleted node's number. By pressing the DELETE key ingame, the node closest to you is deleted.

move <#>
: Move the specified node to your current ingame position. Links are preserved.

link <source #> <destination #> <type #>
: Create a one-way link from the source node to the destination node with the given type (1-5). This is the only way to create non-standard links. Only use this when you need special or one-way links.

clink <#> <#> …
: Link the specified nodes (at least two) in series, creating a chain. This command creates two-way links and is handy for linking nodes in hallways, on catwalks, or along room borders.

qlink <#> <#> …
: Link every specified node to every other specified node. If you're familiar with graph theory, this is a transitive and symmetric relation on the given nodes. This is useful when you want to fill in large areas of terrain with links.

isolate <#>

Remove any links to and from the given node. I tend to use this when I'm adding nodes to an area which has already been linked—first remove existing links in the area, then re-link everything using the new node for better integration.

unlink <#> <#>

Remove any links between the two given nodes. I mostly use this when I make a typo and link the wrong nodes.

deleteall

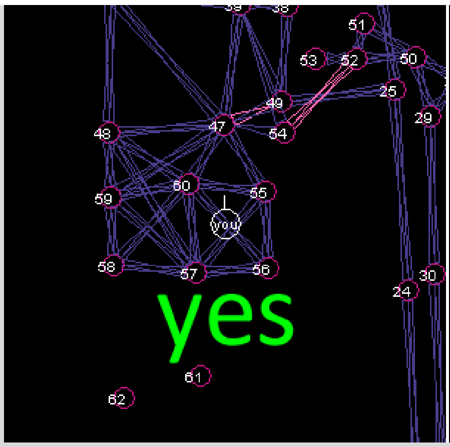
Delete everything in the graph. Remember, there is no undo! If the graph isn't already saved, you can't get your work back.

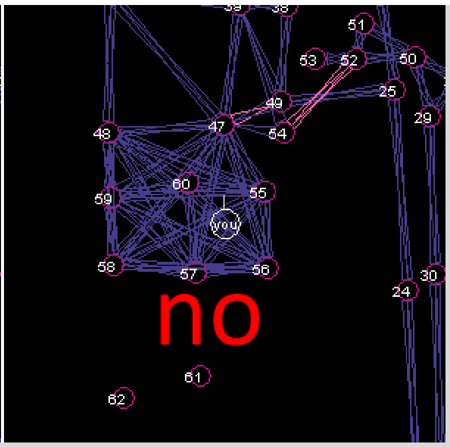
deleteiso

Delete all isolated nodes. Isolated nodes may be hard to spot if they're placed in the same location as another node.

Two more hotkeys you can use are HOME and END. HOME enables auto-link mode, while END disables it. When auto-link mode is enabled, any nodes you place using the INSERT key will automatically be linked in sequence as if you were using *clink*.

To start making a new graph, make sure there's nothing already loaded. If the title bar doesn't say Untitled*, use *deleteall*. For your first graph, I suggest doing a map like Chiron TL-34. The rooms are small and simple and you'll get to use teleporter links. Just load up a map in Halo and start placing nodes. Don't forget to use auto-link mode to save yourself time. It's not sufficient to just run around the map pressing INSERT. Nodes should be linked in certain ways to be effective, so take a look at the node graphs I've already made for you.





The qlink *command can make a mess of things if you aren't careful. Try using the command on just 4 nodes at a time. Otherwise, it can add redundant links such as the one from 58 to 56 (whereas you could just visit 57 instead). Use* qlink *to fill within a circle of nodes.*

Test your node graph by using the *goto* <#> or *altgoto* <#> command.

Teleporters

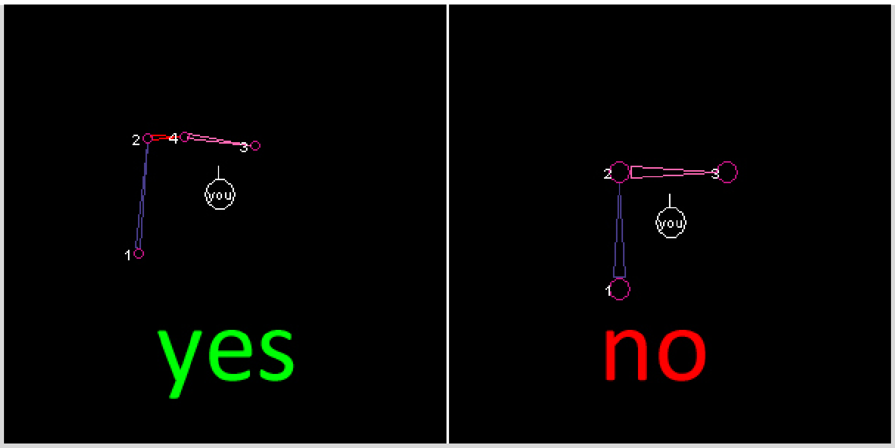
With teleporters, one node is placed at the entrance and one node at the exit. Link the two together using link type 2 (green). The trick to doing teleporter links well is finding the right place for the entrance node.

If the entrance node is too far away from the teleporter, the bot may reach it and start walking in the opposite direction if the teleporter exit does not continue along the same line of movement. This would result in the bot trying to walk to the exit without actually taking the teleporter. If the entrance node is too close to the teleporter, then the bot may teleport away before GS even knows it has reached the entrance node. In this case, the bot would try walking back to the entrance even though it successfully teleported.

A problem you'll encounter is actually placing the entrance node without teleporting away first. You can block the exit, come through the teleporter in the opposite direction, or use dev cam. When using dev cam, check "Use Camera" to place nodes at the camera's location instead of the player's. Don't forget to place them at head height.

Jumps

They key to a successful jump is to be facing in the right direction and have a running start. GS can walk along paths even when it's not facing forward. It does this by pressing a combination of the W, A, S, and D keys. Because this spells disaster for a jump, I forced the bot to aim forwards when traversing a jump link so it only has to press W. However, this isn't enough. If the bot is walking one way, then suddenly needs to jump in a totally different direction, the jump will fail because its velocity before the jump was in the wrong direction. To solve this, place short look-ahead links before the jump link. Take a look at *damnation.wmap* as an example.

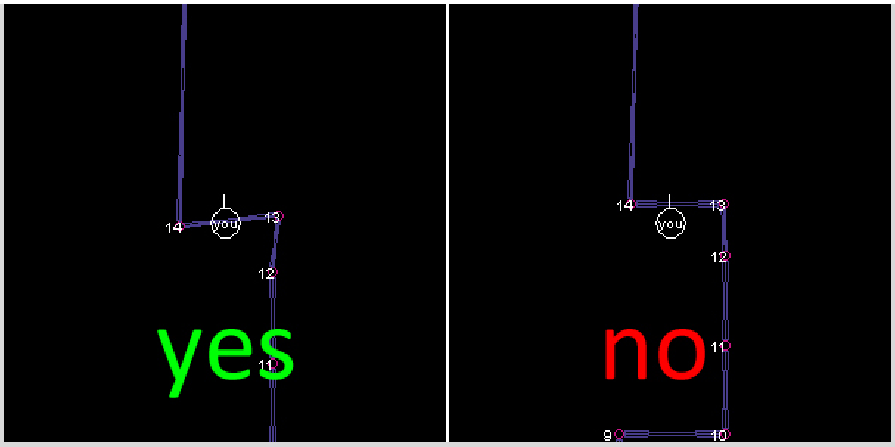


Give the bot a chance to start moving in the right direction before it jumps.

To get extra height from your jump, you can add a crouch link in mid air using dev cam.

Corridors and Thin Walkways

It's important to remember the bot uses the node radius on the Node Graph tab. This means that it may not actually reach the node's center before walking in the direction of the next node. This will cause it to cut corners. To see what I mean, try setting increasing the node radius and telling the bot to walk through the base in Blood Gulch using the *goto* command. To prevent the bot from getting hung up on the inside corners of bend in a hallway, take a wide turn.

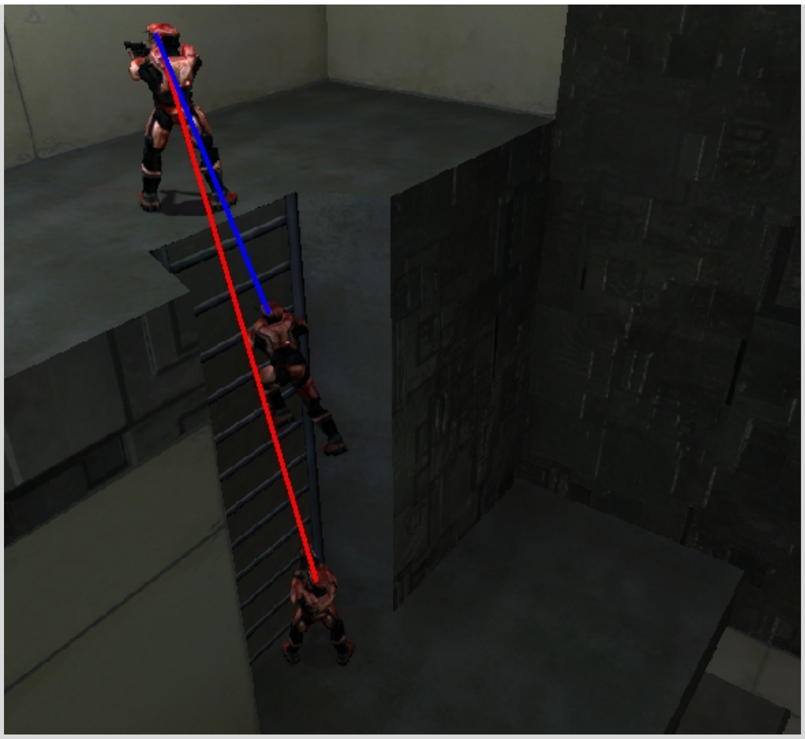


This is the shotgun hallway from Hang 'Em High. Keep the nodes at the bends in the corridor (13 and 14) to the outside corner.

For thin walkways, the same rule applies when it comes to corners. You don't want the bot cutting corners and falling off the walkway. Something else you can do is make the walkway links look-ahead. This will make the bot walk in a perfectly straight line rather than weaving back and forth as it presses a combination of movement keys. The walkways in *damnation.wmap* use look-ahead links.

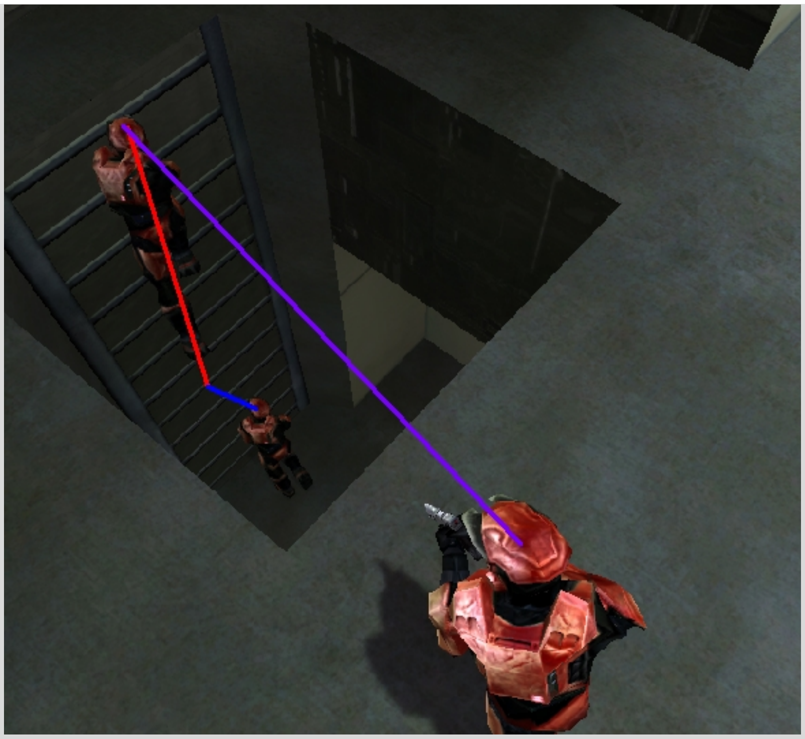
Ladders

Climbing, descending, and leaving ladders is facilitated by the look-ahead link type. Ascending and descending should be handled as different paths rather than a single bidirectional one. The goal is to have the bot look upwards when ascending, and down when descending. A look-ahead link can let us do this, but we can't just put one node at the bottom and one at the top. The bot always needs to face the ladder surface so it doesn't fall. To do this, place the destination node behind the ladder's surface but close enough so the player can still enter the node radius.

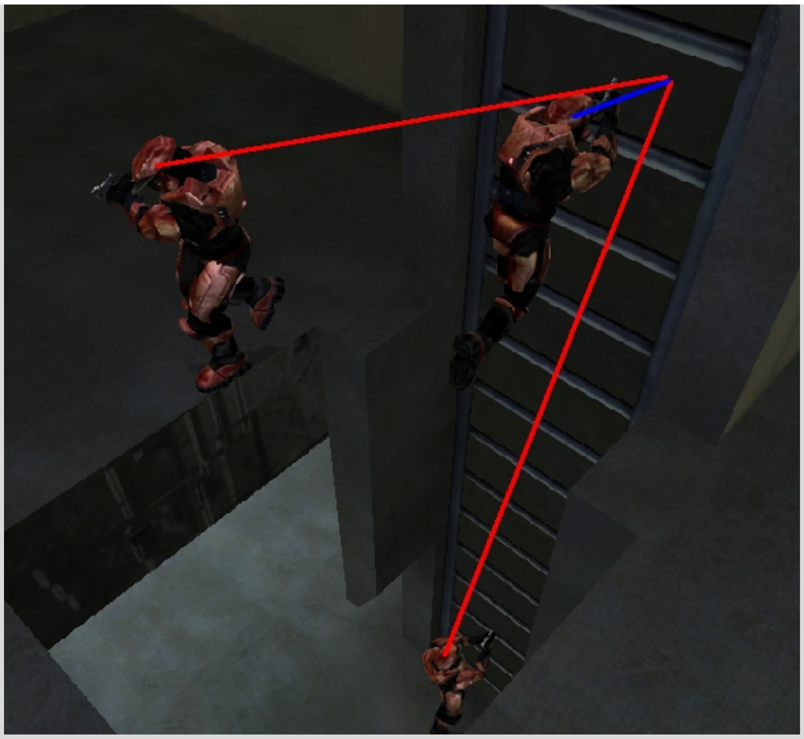


This is the simplest case for ascending a ladder. The red line represents the look-ahead link which starts at the bottom and ends at the top. The blue line shows how no matter where the bot is on the ladder, it looks upwards and forwards.

Unfortunately, this is also the hardest case for descending the ladder. At heights like this, I suggest just having the bot walk off the edge. For taller ladders, look for an alternate way down or have the bot crouch as it walks off the top edge, then turn around in mid air and move back towards the ladder as it falls by using a look-ahead link.



To descend this ladder, simply walk the player onto the ladder surface then use a look-ahead link as shown. The bottom node of the look-ahead link is behind the ladder surface so that the bot always faces down and forwards when descending. Upon reaching the bottom, the bot should be close enough to the node to continue along its path.



If your ladder ends like this, climb to the top and then dismount by turning around and moving away. Another look-ahead link can accomplish this. Make sure the top ladder node is behind the ladder surface and high enough so the bot doesn't fall back down the hole.

Vehicle Paths

The first thing you'll want to do is set the node radius to at least 2. Vehicles move faster and are harder to control, so they need more leeway. Since vehicles require you to aim to steer, you can either use look-ahead links or force look-ahead mode with the checkbox on the Node Graph tab. For banshee paths, use dev cam to place the nodes. Make sure you uncheck "Use Camera" before using *goto* or a script because a 3rd person view is not true to your player's position. If the vehicle isn't moving, it could be that the bot is trying to jump over an obstacle by holding SPACE, which is the brake key for vehicles. If this happens, stop the bot and place the vehicle directly at the first node before trying again.

Additional Tips

- You can set the "Opens With" of .wmap files to GuiltySpark.
- If you have a wall separating two rooms, make sure the nodes along each side of the wall are spaced closer than the thickness of the wall. The path finding algorithm starts at the closest node, so don't let it pick a node that's not even in the same room.
- The graph doesn't need to be as dense in less-traveled areas. Prioritize the important directions of travel.
- Put nodes on weapon spawns and objective locations so your AI scripts can use them.
- Share your node graphs so other people can use them too.
- Changing the "Link Size" or hiding parts of the graph can make it easier to view nodes