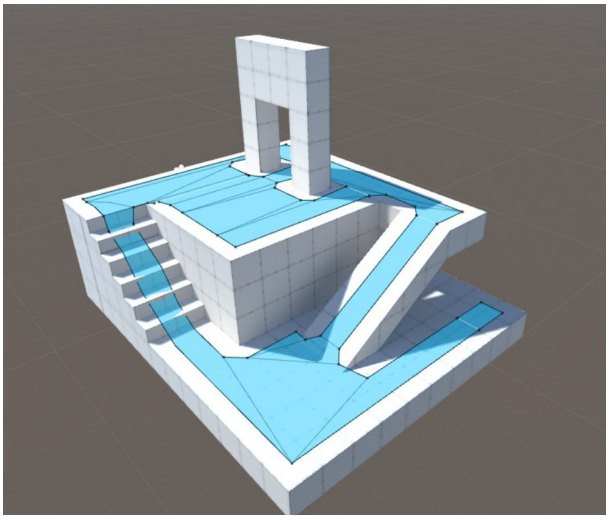
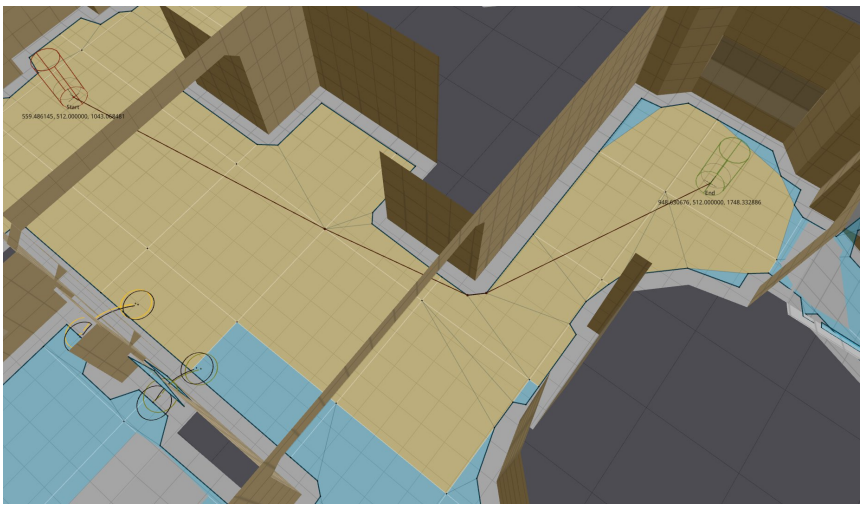
Evobot Nav Editor Documentation

NavMesh Overview

Evobot uses a navigation mesh (“navmesh”) to find paths throughout the maps. Whereas waypoints represent “breadcrumbs” or a chain of dots which form paths throughout a map, a navmesh instead defines a walkable area that the bot (“agent”) can freely move through. An example of a navmesh is below:



In this case, the agent knows it can move anywhere within the blue. Pathfinding is done by connecting each triangle (outlined in the dark colour above) and finding a chain of triangles that take it to the target destination. If a chain is found, then a set of points can be generated to produce a nice, straight and minimalist path without wobbly detours:



Evobot specifically uses the Recast and Detour open-source frameworks created by Mikko Mononen and maintained by many dedicated users.

https://github.com/recastnavigation/recastnavigation

Using the Editor

Nav Editor Tool Overview

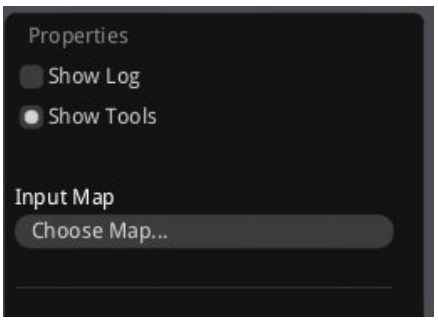
The nav editor tool that comes with Evobot is an adaptation of a demo application produced by the team maintaining the Recast and Detour libraries. It has been adapted to open Goldsrc BSP files and to correctly save and export the results for use in-game.

The editor loads and saves .nav files, which contain all the navigation data needed by the bot to find paths around the map.

Basic Usage

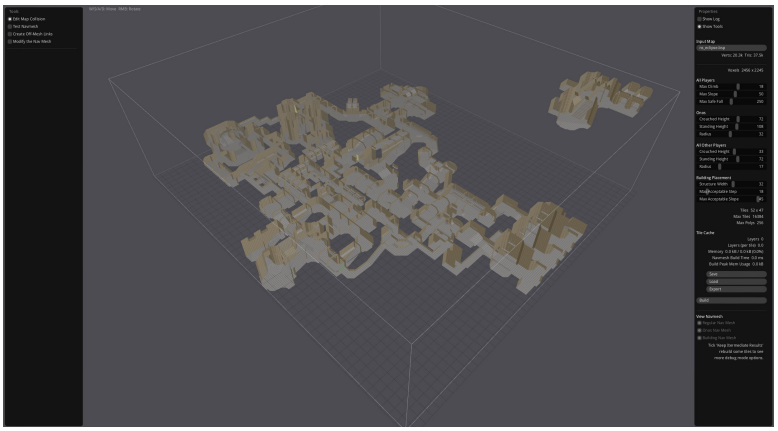
Once you have run the editor, you will see a screen like this:

To get started, click *Choose Map* in the top right.



You should be presented with a list of available BSP files. If you do not see anything, then you need to go back in to *navtool.cfg* and ensure the MapsDirectory is set correctly.

Choose a map from the list and the map will be opened and visible as below:

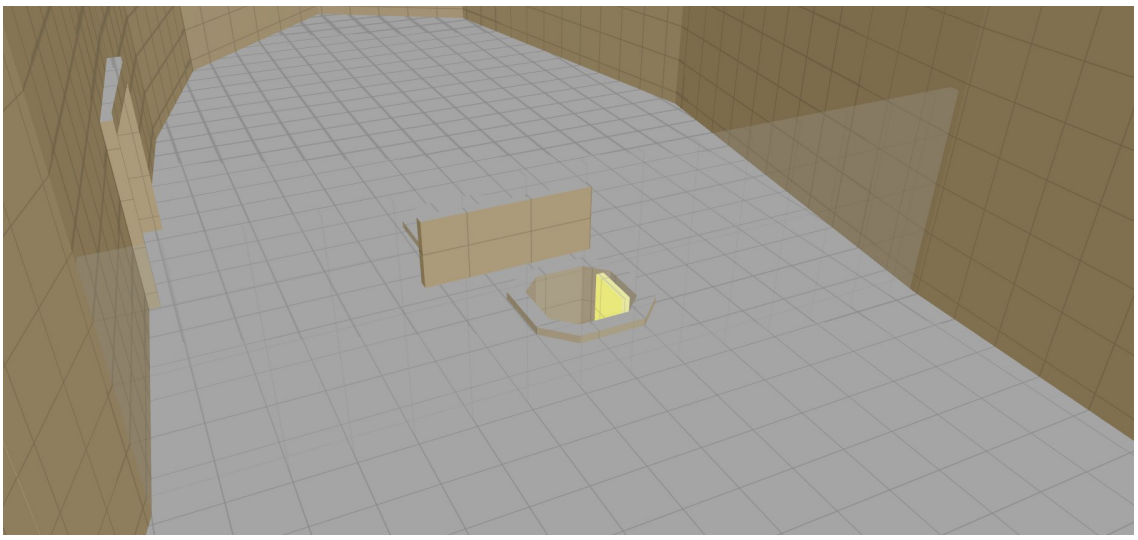


The editor uses WASD controls to fly around the map. To use mouselook, just click and hold the right mouse button. Hold shift to “sprint”, and scroll the mouse wheel up or down to increase or decrease the camera flight speed.

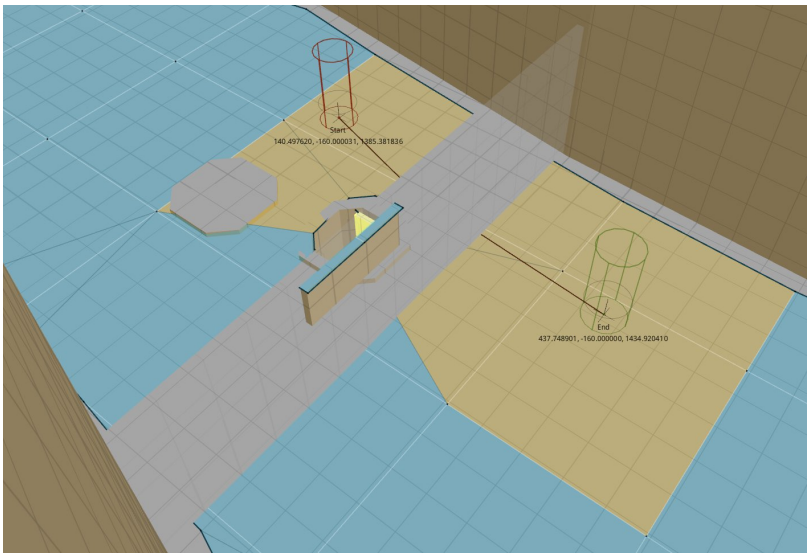
Preparing the Map For Mesh Generation

Goldsrc is a weird beast at times. There are a few ways that mappers can create things like see-through fences and railings which sometimes confuse the nav editor tool as to whether they should be solid or not. Consider this fence in the main courtyard in cs_militia:

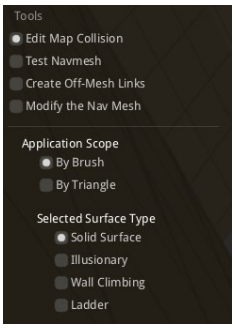
This is a func_illusionary and difficult to distinguish between something solid like the fence, and a non-solid illusionary like some decorative vines or the curtains in cs_747. By default, the nav editor treats func_illusionary as non-solid, displayed as a transparent brush:



This is a problem, as the nav mesh will be generated as if the fence isn't there, and the agent will think it can walk through it:



To fix this, you can use the *Edit Map Collision* tool by selecting it in the top-left:



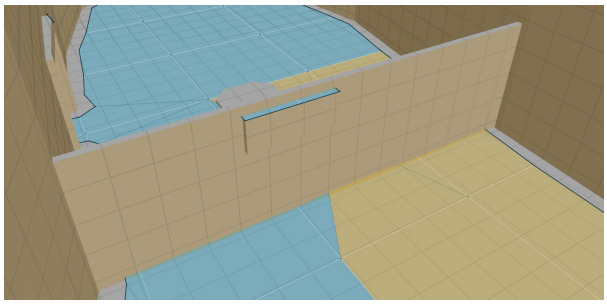
Once selected, you can change the scope to *By Brush* (will apply the setting to the entire func_illusionary entity) or *By Triangle* (will apply the setting to a single triangle). Sometimes mappers like to group multiple brushes up into a single entity, so if you find changing the collision by brush also changes some other brushes you don't want changed, then use the by triangle.

The surface type will determine what you mark that brush/triangle as being.

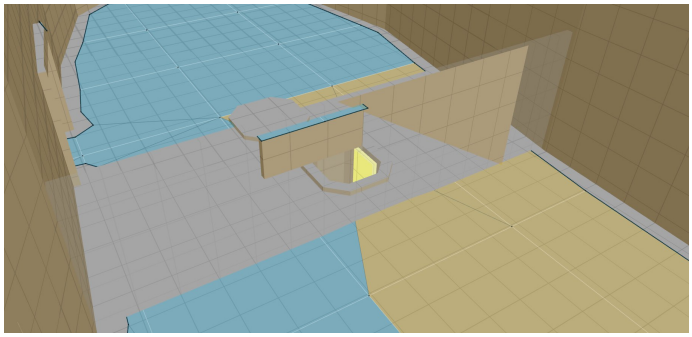
- **Solid surface** – Wall/floor
- **Illusionary** – Not solid
- **Wall climbing** – Skulks, fades and lerks can traverse it (useful for steep slopes that are normally marked as too steep to traverse!)
- **Ladder** – All agents can climb up it (**NB:** this is only for certain sloped ladder surfaces, for standard vertical ladders you still need off-mesh connections)

Simply choose a scope and a surface type, then left-click the brush or section you want to change.

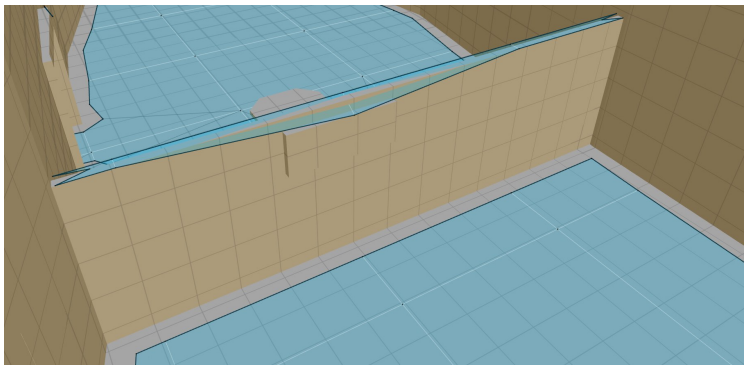
By brush will set all surfaces associated with that entity. Note that if the mapper has grouped multiple brushes together into one entity, you may find that it changes more than you expected:



By triangle will only modify the individual triangle. Useful if the map has grouped multiple brushes together including some you don't want to modify:



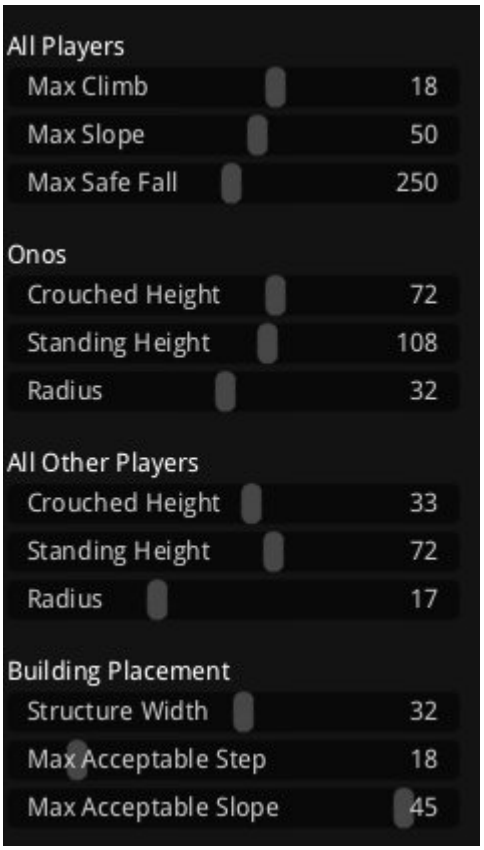
Once you have finished modifying the map to your satisfaction, click *Build* again to regenerate the nav mesh with the new collision settings:



Notice how the navmesh now stops at the fence as it should. It also generates some walkable space on top of the fence, since it now understands that players can walk along the top of the fence as well.

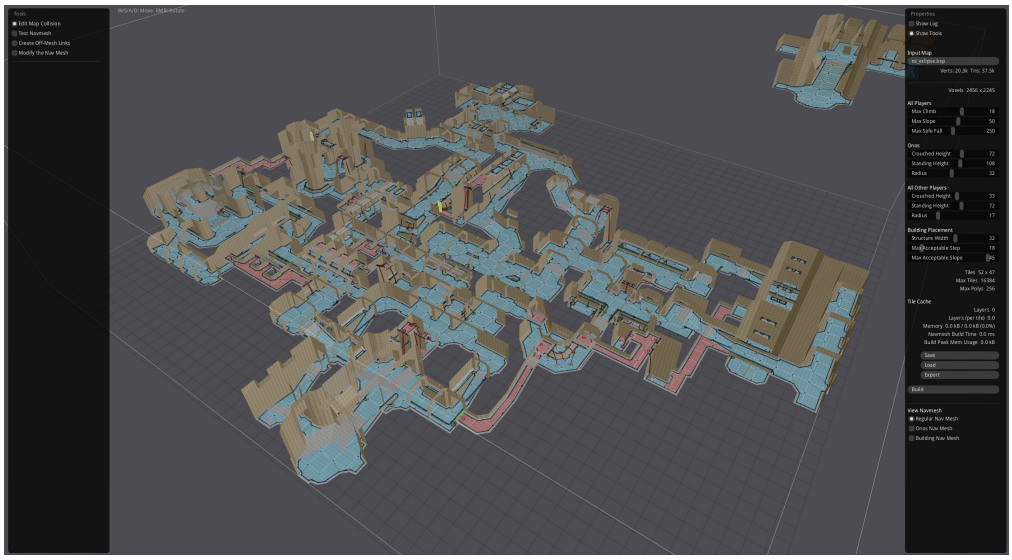
Generating the NavMesh

Once you've prepared the map, there will be a series of settings down the right-hand side of the tool. These allow you to adjust how the navmesh is generated. All settings are in GoldSrc units:



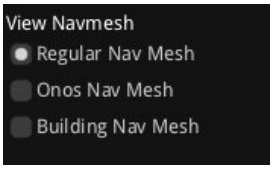
- **Max Climb** – This is the maximum height your AI is allowed to "step up" without having to jump. Think of it as max stair height. The navmesh will flow up stairs but stop at places where the height difference is greater than max climb.
- **Max Slope** – The maximum slope the player can comfortably climb up (in degrees). If a surface has a slope greater than this value, it will be considered non-traversable.
- **Max Safe Fall** – This comes in to play later with off-mesh connections (see the section on this for more info). How far can a player safely fall without taking unacceptable damage?
- **Crouched Height** – How tall is the player when crouched? Any areas smaller than this are considered non-traversable.
- **Standing Height** – How tall is the player when standing? Any areas smaller than this but equal or larger than crouched height will be considered crouch areas.
- **Radius** – How wide the player is (half-width). If a gap is narrower than the radius * 2 then the area is considered non-traversable.
- **Structure Width (Natural Selection only)** – How big structures should be considered. Impacts how the navmesh used for building placement is generated.
- **Max Acceptable Step (Natural Selection only)** – Similar to max climb, but for structure placement specifically.
- **Max Acceptable Slope (Natural Selection only)** – Similar to max slope, but for structure placement specifically.

Once you are happy with the settings (the defaults should work for 99% of use cases in GoldSrc, unless you're generating a mesh for a mod with different player dimensions), then click *build* to generate the mesh:



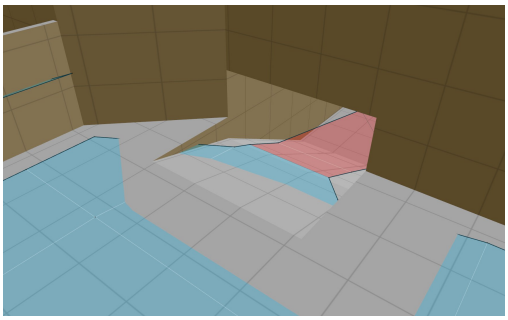
For Natural Selection, 3 meshes will be generated: one for all players except Onos, one for Onos specifically (as it has significantly different dimensions to the other players and needs its own special navmesh), and one for building placement (again, this represents areas where gorges and the marine are allowed to place structures, and needs its own mesh).

To switch between the three, use the pips at the bottom right:



This will allow you to visualise the navmesh and make adjustments specifically for each agent type as needed (Onos, normal, building) if you're not happy with the results.

Blue mesh areas denote regular walking, red denotes crouch areas (bot will crouch when traversing these). Note that small agents (skulks, lerks and gorges) will treat crouch areas as regular walking since they can't crouch.



At this point, you can click *save* to produce a NAV file ready to use in-game if you don't need to do any further work on the navmesh.

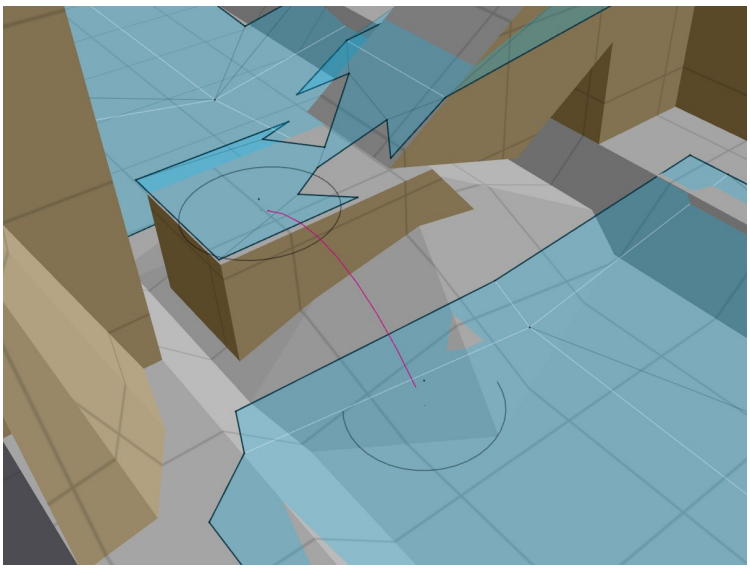
You can click *Build* again to regenerate the meshes as much as you like.

Off-Mesh Connections

Overview

While the navmesh by itself defines a clear walkable area for an agent, players usually have far more movement capabilities than just walking and crouching along the floor. We need a way to mark areas where a player can jump, drop down, climb ladders and other advanced movement techniques.

This method is known as “off-mesh connections” and consist of a start and end point that represents a point where the agent can cross a gap in the navmesh:

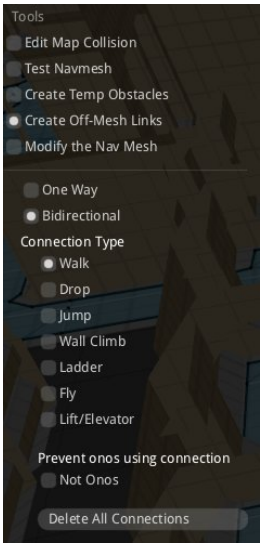


Off-mesh connections act in many ways like traditional waypoints: they can be one-way or bidirectional (unless specified below), and come in several flavours which tell the agent how they should traverse between the two points.

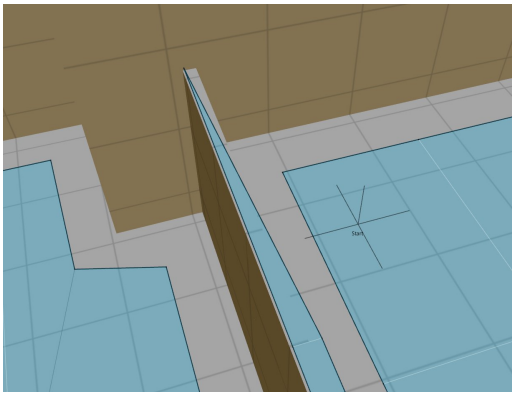
The bots have defined behaviour for approaching link types, so you may need to experiment a little to get a feel for exactly how best to place the links to get the most reliable results. In general however, you can be fairly relaxed with placement and the bot does not require high precision to figure out how to move properly.

Placing and Deleting Off-Mesh Connections

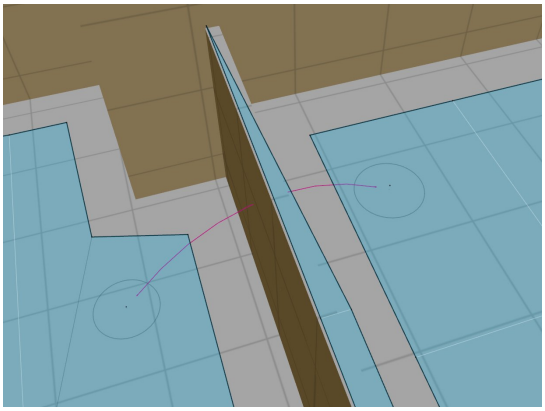
To start placing off-mesh connections, switch to the *Create Off-Mesh Links* tool on the left:



Select a connection type, set any special flags on it (one-way, bidirectional, or other flags), then left click once anywhere on the map to place a start point. It will be marked as “start”:



Then left-click again somewhere else to set the end point and complete the connection.



If a connection’s start or end points are too far from the nav mesh, then they will be invalid and the bot won’t use them, so ensure that the start and end points are always on the nav mesh.

To delete a connection, simply hold shift and left-click inside either the start or end circle of a connection.

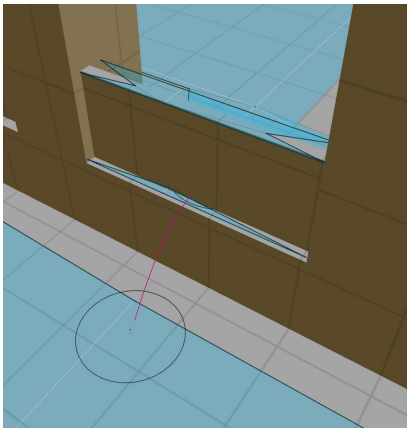
Connection Types

There are a range of different connection types which are colour-coded, summarised below:

- **Walk (White)** – The bot will attempt to walk directly between the two points (it will automatically crouch if it needs to).
- **Drop (Red)** – The bot will attempt to drop off a ledge. You could use a walk point for this as well, but drop has special behaviour to detect how far the fall is. Based on the *Max Safe Fall* setting (see *Generating the Navmesh* above), this will either be a "fall" or "high fall" connection. Some agent types (marines, gorges, onos) will only use high falls if they have no other choice, while skulks, lerks and fades will use them freely. Note that drops are one-way only (which makes sense since you can't drop upwards...). If you want the bot to drop down into water, then use a one-way walk point so the connection doesn't consider it a high fall.
- **Jump (Yellow)** – The bot will attempt to jump directly between the start and end points. This can be placed either side of a gap the bot should jump, or on either side of a barrier (e.g. railing) the bot can jump over. The bot will keep jumping until it gets to the other side. Select the *Requires Duck Jump* special setting to mark if this jump requires a duck-jump. Gorges and skulks will not use this connection if it is set since they cannot perform that move.
- **Wall Climb (Green)** – The bot will attempt to directly climb up the wall (if a skulk), fly up it (if lerk) or blink up it (if fade). Like drops, these are one-way only.
- **Ladder (Blue)** – The bot will use the nearest ladder to reach the target point. Skulks will climb ladders too, so no need to use both a ladder and wall climb in the same place.
- **Fly (Green)** – Lerks know they can fly from point A to B. Useful for reaching high places no other player type can.
- **Lift/Elevator (Purple)** – Bots will look for a nearby func_door, func_plat or func_train to operate. More on this later.
- **Not Onos** – Bot will not use this connection if Onos. You don't need to worry about this if you're placing wall climbing connections since Onos can't use them anyway, or if the connection is inaccessible from the Onos navmesh. This is for when you have something like a jump that an Onos can't manage because of their bigger size.

Jump Connections

While traversing a jump connection, the bot will attempt to move directly towards the end point while repeatedly duck-jumping (if it can) as needed until it gets there. This means if the bot needs to jump up onto a ledge and then over something else, you can put just one jump connection as the bot will automatically keep jumping. For example:

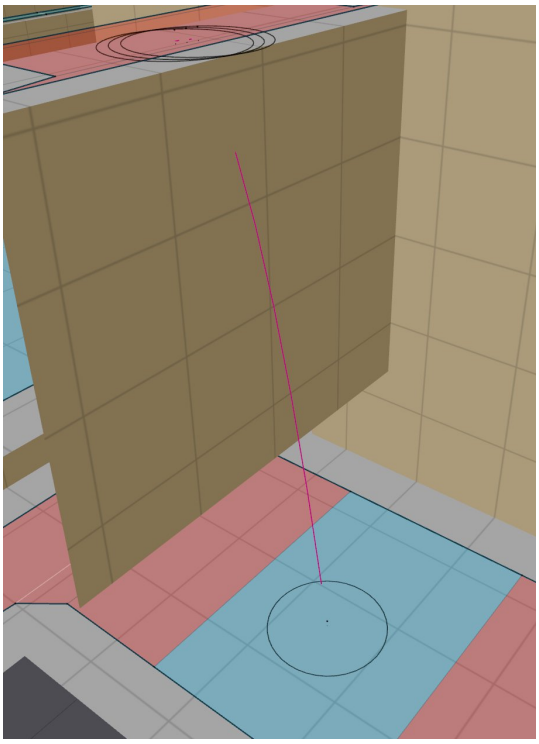


In this example, there is no need to add a jump link to the small ledge and then over the railing, just one connection is needed because the bot will automatically jump again once it hits the ledge.

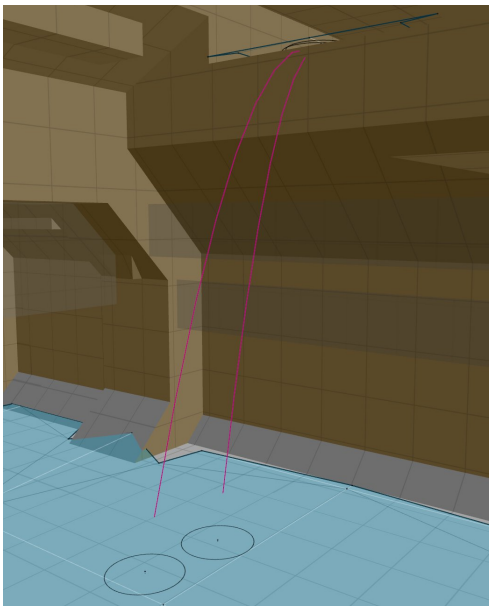
If a jump connection is marked “duck jump only” then skulks and gorges will not use it. For skulks, you can use a wall-climb connection instead, while gorges will need to find a different way to get to that location (if they can...).

Wall Climb Connections

When traversing a wall climb link, the bot will start at the beginning point and attempt to move in the direction of the connection while looking up or blinking. This means that a wall climb connection like this **will not work**:



Because the bot will end up going under the ledge and get confused. However this connection **will work**:



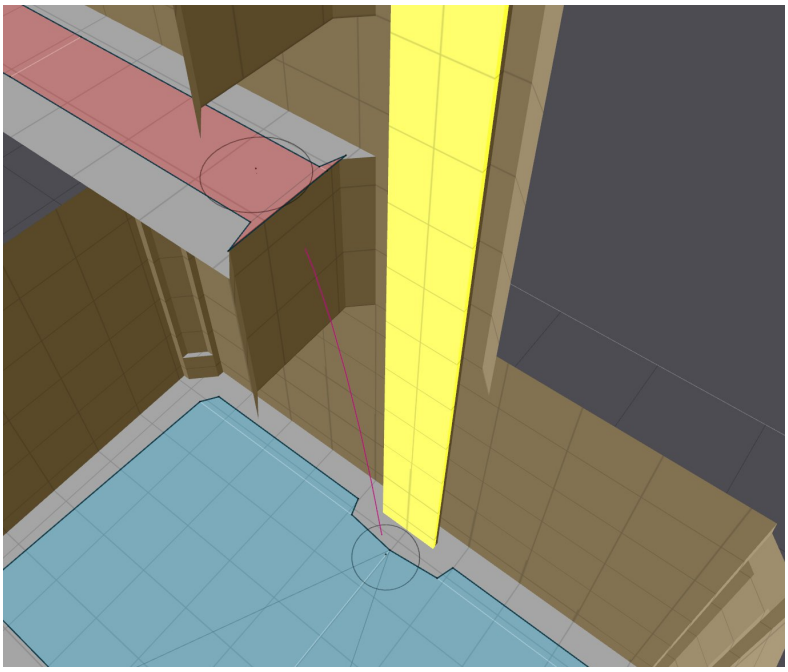
Because you can actually climb under and up this sloped surface by just looking upwards and running forwards.

This limitation will be dealt with in a future version of the tool to allow multi-point connections to draw out climbing paths rather than a blind “run forward and look up” approach.

Ladder Connections

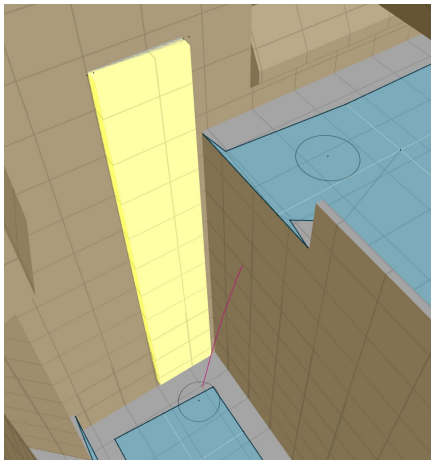
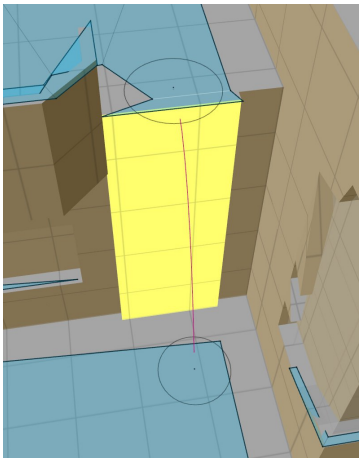
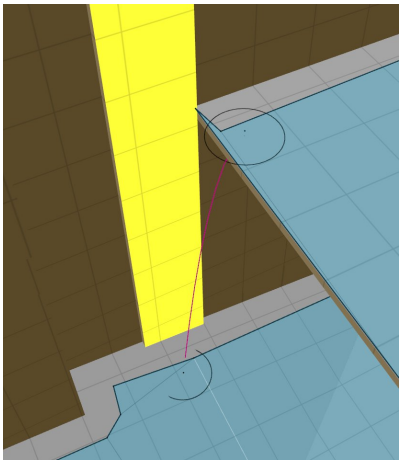
Ladder connections are a lot more sophisticated than wall climbing. Upon reaching the start point at the top or bottom of the ladder, the bot will detect the closest ladder, work out the climbing surface and then move to stick to the ladder. It will then move either into or away from the ladder to climb up/down, and detect when it can safely disembark.

Because of this detection and respect for the ladder itself, you only need to place one point where the bot should get on the ladder, and one point where it should get off. You do not need to precisely place points at the top of the ladder. For example:



The bot here can use a ladder to climb into a vent part-way up the ladder. All you have to do is place one point at the base of the ladder, and one in the vent where you want the bot to get off. The bot will do the rest.

Some more examples here:

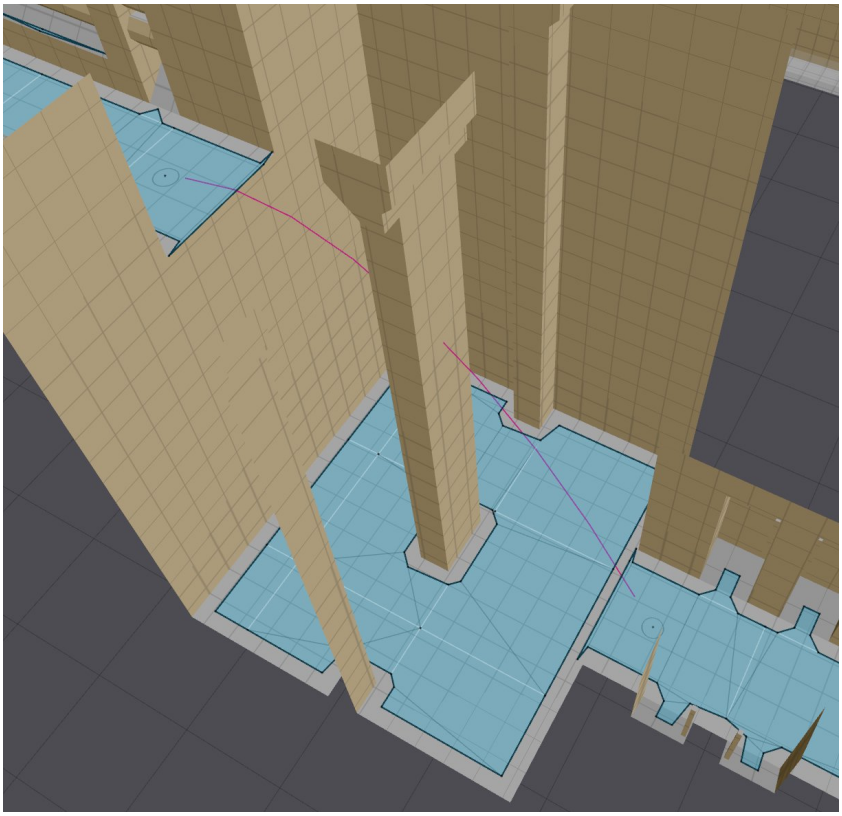




Lift/Elevator Connections

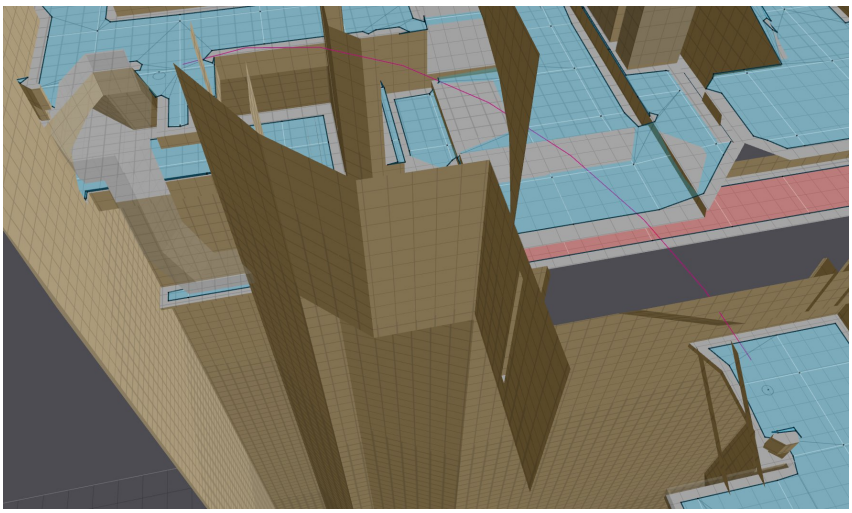
Lift and elevator connections are very simple to operate, and work in a similar way to ladder points. Simply place the start and end points where you want the bot to wait for the lift to arrive (if it's in the wrong position). Make sure these points are clear of the lift so they don't accidentally block it while waiting. **Make sure the platform itself is not solid so it doesn't generate walkable areas on it. The nav mesh is not dynamic, so it will confuse the bot if the lift isn't where the nav mesh was generated.**

The bot will automatically handle everything else such as how to activate the lift and getting on and off it. Examples below.

The big lift in ns_nothing:



The moving platform in ns_machina:



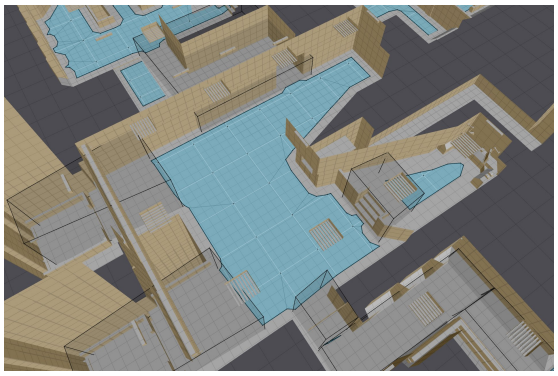
Modifying the Navmesh

Sometimes you need to fine-tune the generated navmesh, without modifying the generation settings. Perhaps there is just one small area where the mesh has incorrectly generated it as walkable when the agent needs to crouch, or it has incorrectly calculated that an agent can squeeze through a gap you know isn't possible due to GoldSrc collision detection trickery.

In this scenario, rather than modify the agent radius and height and risk breaking the mesh in other areas, a better option is to just manually prune or modify the bit that isn't quite right. This is where the navmesh modification tool comes in. To use it, select the *Modify the Navmesh* tool on the left:

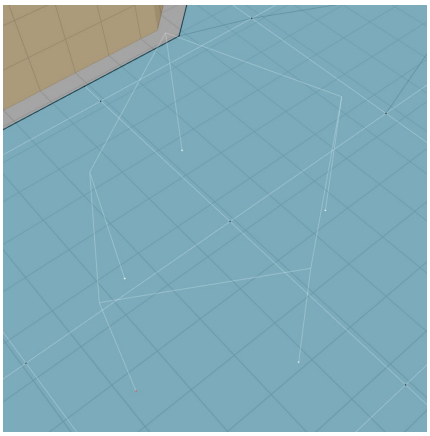
This allows you to draw out a shape, and any part of the navmesh which touches this shape will be modified accordingly:

- **Null** – Cuts a hole in the navmesh, any are touched will not be traversable. Also affects the building nav mesh.
- **Crouch** – This area will be marked as crouch.
- **Wall Climb** – Only agents capable of climbing walls will use this. Useful for very long, steep slopes which can be climbed up, but the start and end points are too far away for an off-mesh connection (remember: tiles have to be adjacent!)
- **Jump** – Bot will keep jumping while traversing this area. Useful for something like a damaging floor that you want the bot to avoid touching as much as possible.
- **No Build (Only available and visible when selecting the building nav mesh)** – Will block the structure nav mesh generating in that area, but does not affect the regular or onos nav mesh. Useful for blocking off areas you don't want a gorge or the commander to place structures. For example, below in ns_nothing, I've prevented the commander placing structures in the stairs or hallways:

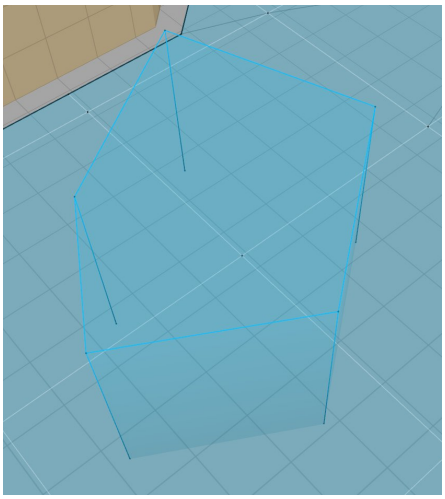


To create the shape, first set the shape height and descent. The descent will determine how sunk into the floor the shape is, while height determines how tall the shape is. You can also modify these settings while drawing out the shape to adjust as needed.

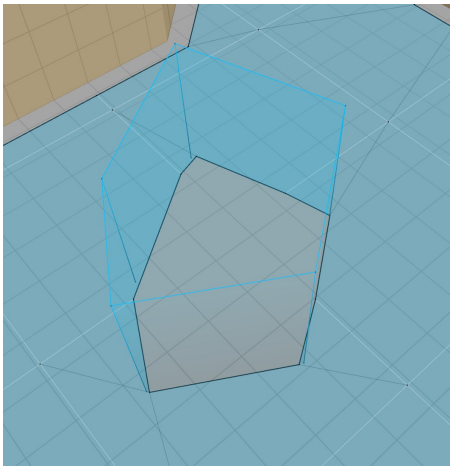
Reduce the height if it's encroaching onto areas above it you don't want modified, or adjust the descent if it's not quite reaching areas under it you also want affected. Now start clicking points on the floor where you want the shape created, you will see a wireframe of the shape taking form:



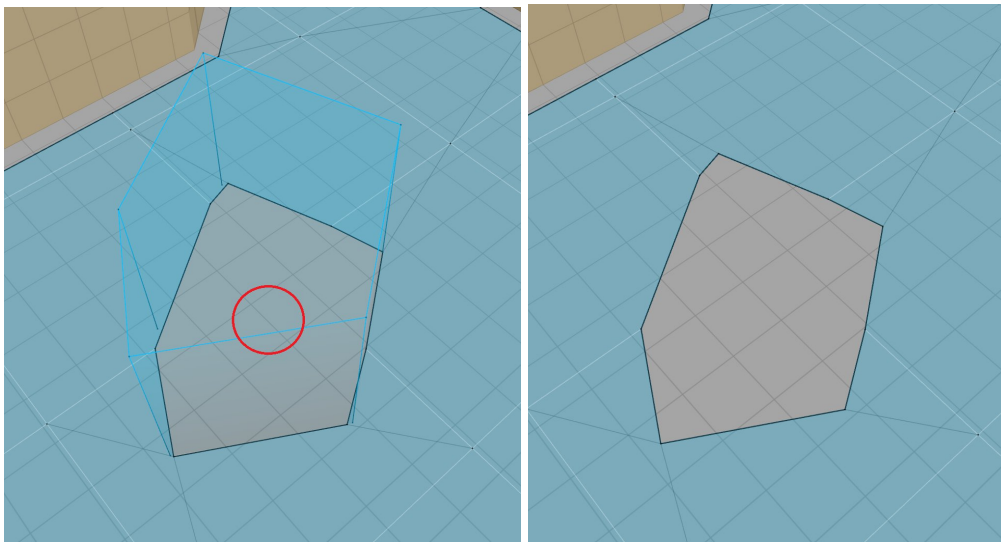
Once you are happy with the shape, click again on the red vertex point (or double click) to finish the shape:



Now the next time you click *Build*, the mesh will be updated accordingly:



To delete a shape, simply shift+click somewhere inside the shape (e.g. Where the red circle is below):



Click *Build* again to refresh the mesh.

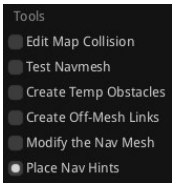
Navigation Hints

Navigation Hints are points placed on the map that can hint to the bot that this location is useful for something e.g. Placing an infantry portal if commander, or an ambush point for skulks. Currently, the bot only supports nav hints for placing structures, but will use hints for more in the future. This tool is useful if you're finding that the commander is placing structures in bad places.

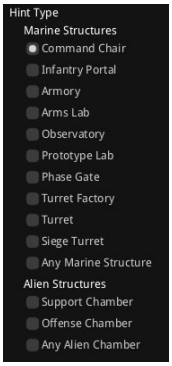
NB: You do not need to place hints for all structures: the bot will fall back to finding places itself if a hint does not exist for a building. The bot is also smart enough not to use invalid hints (e.g. Infantry portal hint placed too far from the comm chair). If you find a bot isn't using a hint at all, it might be because it's in an invalid build location, so you may need to adjust it.

Placing and Removing Hints

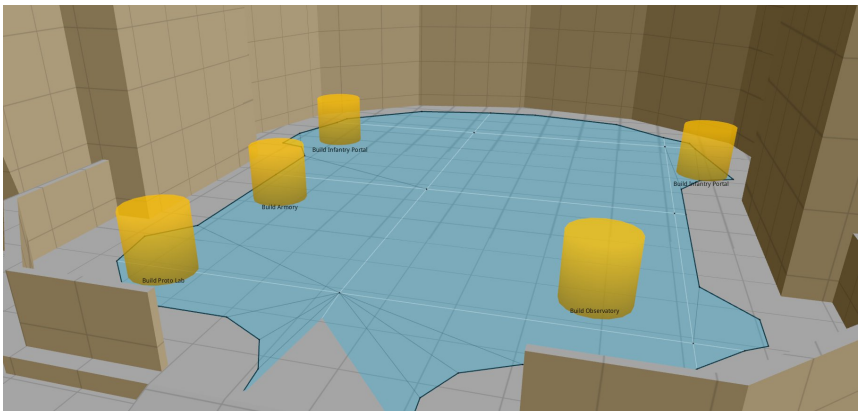
To place a hint, simply select the *Place Nav Hints* tool in the menu:



Once selected, use the options on the left to define the type of hint to be placed. You can select multiple types to allow the bot to use the hint for more than one purpose:



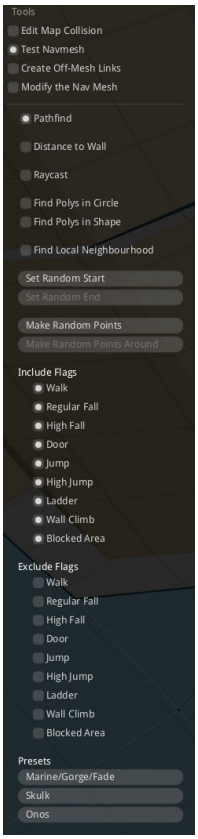
Once selected, click anywhere on the map to place a hint, or shift-click to delete a hint. If placing hints for building placement, it's recommended you have the building nav mesh selected to ensure you're placing them somewhere in a buildable location:



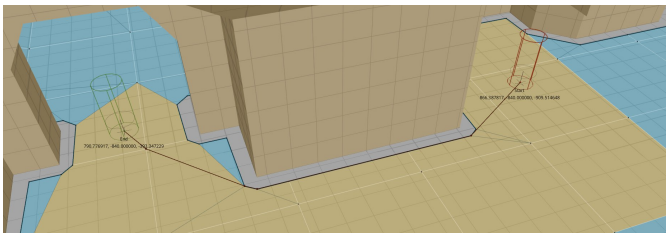
Testing the Navmesh

In The Editor

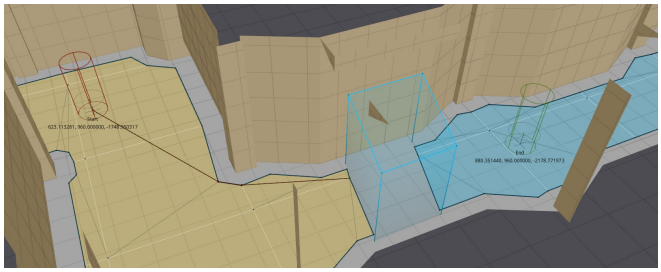
You can test the navmesh in the editor by clicking the *Test Navmesh* tool on the left:



You can test path finding by using the include and exclude flags (or the presets) to define what kind of agent type you want to test path finding for (e.g. Skulk) and then use left and right clicks to set a start and end point. If successful, a black line indicating the route will be drawn:



If the path is not successful, the line will end at the closest point it managed to get to:



Note that this only indicates the bot's ability to find a path, not necessarily perform the move. For that, you will need to test in-game (see below).

Testing In The Game

You can also test the navmesh in-game. Once you have made some changes to a navmesh and exported it with the tool, you can reload it in game (even during a match) with the following command:

sv_reloadnavmesh

The bots will continue playing as if nothing had changed, but any modifications you have made to the navigation files will be updated accordingly. This makes it possible to alt-tab between the editor and game and instantly get feedback on how your changes are performing.

The bot comes with two debug modes: drone and test nav. These are set using the *mp_botdebugmode* cvar:

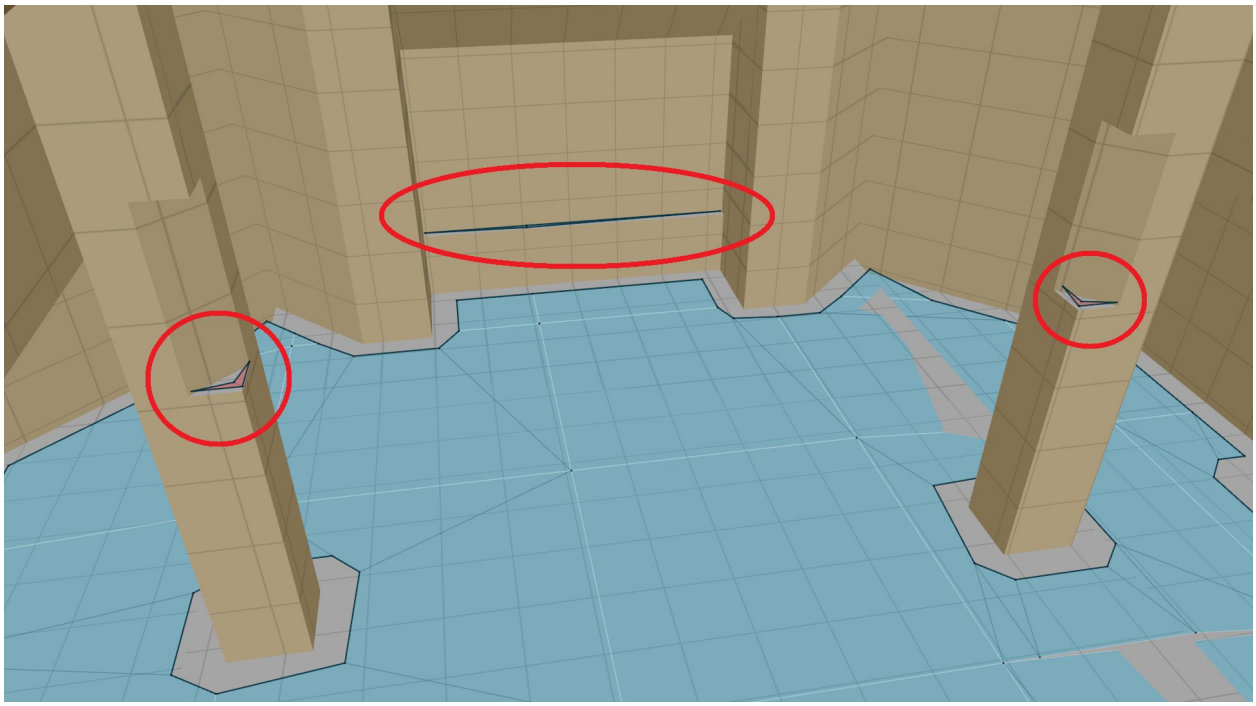
mp_botdebugmode 0	Disables debug mode, bot plays normally.
mp_botdebugmode 1	Drone mode, bots do nothing unless commanded to.
mp_botdebugmode 2	Test nav mode, bots will randomly patrol the map. They will not fight or pursue objectives. Alien bots will evolve into different life forms to ensure they have at least one skulk, gorge, lerk, fade and onos (if they have enough players).

If using drone mode, use the console command *bot_cometome* to order every bot in game to come to you. Alternatively, marine bots can be individually ordered by the commander as usual.

Troubleshooting Common Problems

The Bot Tries to Get Through Solid Walls

If you find the bot repeatedly gets stuck in very specific areas of the map trying to get inside a wall, it may be because there are rogue bits of nav mesh that the bot thinks are accessible from where it is. Example below from *ns_nancy*:



These little bits of nav mesh along edges might be harmless, but if you find bots getting stuck trying to merge with the pillars, or humping the wall at the back, you could try using the collision editor tool to eliminate those ledges so the nav mesh doesn't generate there. In some cases, islands of nav mesh can actually generate *inside* solid walls or pillars. Again, eliminate them if they're giving you trouble. The magic of GoldSrc.

You don't have to do this for every instance of this, only where bots seem to get confused!

Commander Tries to Build in Bad Locations

Likewise, if you find that the commander tries to place structures in bad locations, ensure the building nav mesh has been appropriately cut with the mesh editor tool in those locations so the commander doesn't try to place anything there.

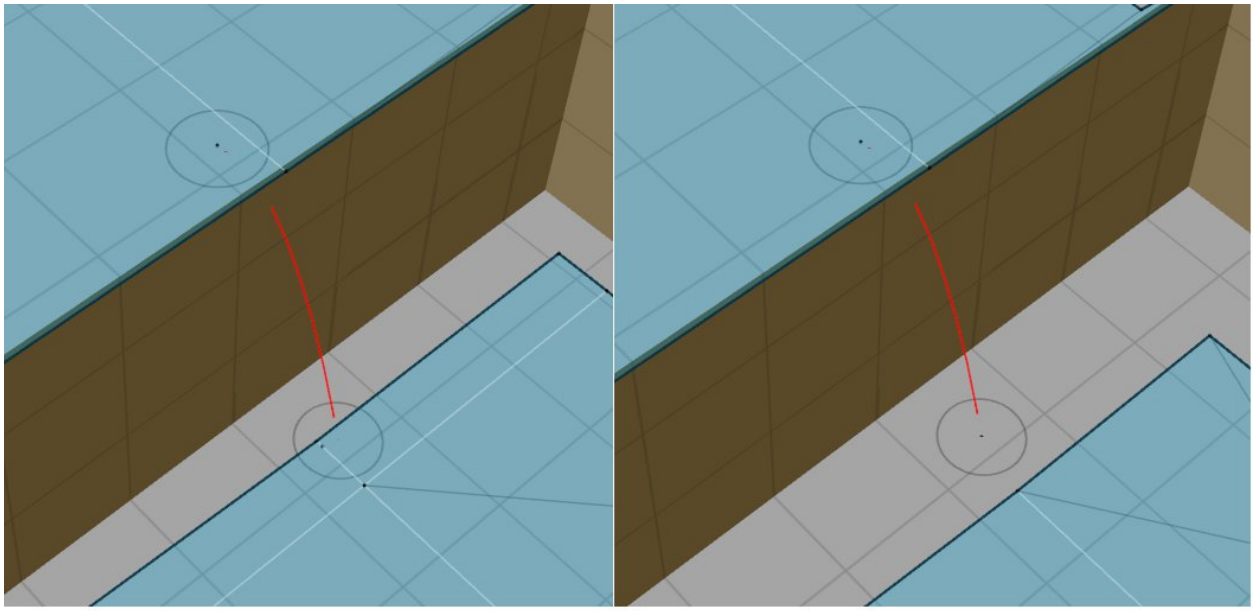
The Commander Ignores my Nav Hints

If you find the commander ignores some or all of your nav hints when placing structures, check a couple of things:

- Make sure nav hints aren't too close together – If they are, then the commander won't be able to place a structure in both locations since the first structure blocks placement of the second.
- Make sure the nav hints are in a valid location – Obviously, ensure they're not too close to some map geometry, or on a *func_nobuild* area that would block placement.
- Ensure the nav hints aren't too far away from the desired location – The commander will try to establish a relocation base or outpost as close to the hive location as possible. Ensure your nav hints for relocating are within 20 metres or so (2D distance) from where the hive appears in game. Likewise for siege bases, ensure the nav hints are more than 10m from the hive, but still within siege range.

The Onos Can't Reach Areas It Should

Make sure when placing connections you want the onos to use, they are properly connected to the Onos nav mesh. An easy mistake to make is place the connection on the edge of the regular mesh, and not realising it will be off the edge of the Onos mesh:



Regular Mesh **Onos Mesh**

Notice how this drop connection is linked correctly on the regular mesh, but is actually off the Onos mesh. It's recommended for any all-purpose links like this, you place them with the Onos mesh visible so you can easily see they're all connected.