



**M E R C U N A**

Mercuna Ground Navigation  
UE4 User Guide

v2.1



## Contents

[Installing](#)

[Configuring Agent Types for navigation](#)

[Agent Categories](#)

[Configuring the Navigation Grid](#)

[Setting up the navigable space](#)

[Nav Seeds](#)

[Exclusion Volumes](#)

[Building the Nav Grid](#)

[Memory usage and Performance](#)

[Runtime Grid Building](#)

[Runtime Grid Rebuild](#)

[Physical Materials](#)

[Physical Materials on Landscapes](#)

[Pathfinding](#)

[Nav Grid Testing Actor](#)

[Navigation Cost Multipliers](#)

[Creating a Mercuna navigated Agent](#)

[Navigation Component](#)

[Example for a wheeled vehicle](#)

[Example for a slowly turning animal \(e.g. a horse\)](#)

[Example for a humanoid Character](#)

[Obstacle Component](#)

[Nav Links](#)

[Nav Link Creation](#)

[Configuring the Nav Link](#)

[Cost](#)

[Agent Types and Nav Grids](#)

[Steering](#)

[Avoidance](#)

[Blueprint Functionality](#)

[EQS](#)

[Raycast - test whether there is a clear straight line path from the context to the positions.](#)

[Debugging Problems](#)

[Logging](#)

[Profiling](#)



[Debug Actor](#)

[Debug Draw](#)

[Navigation Grid](#)

[Debug Draw](#)

[Known Issues](#)

---

## Installing

The Mercuna middleware is integrated into Unreal Engine as a standard plugin compatible with **Unreal Engine 4.24** and later.

Binary versions of Mercuna (such as the evaluation) must be installed as an Engine plugin - simply copy the Mercuna directory into the **Plugins** directory within your Unreal Engine directory.

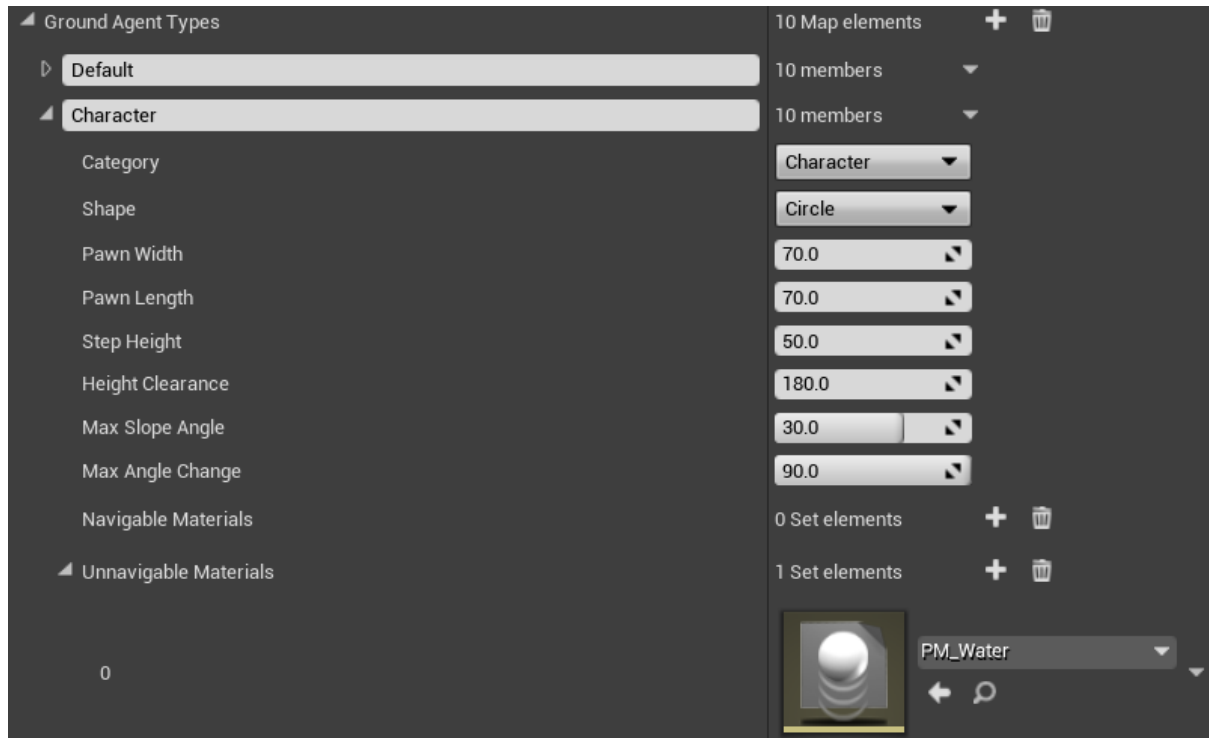
The full source version of Mercuna can be installed as a Game plugin and then will be rebuilt as part of your game, or, if you are building the engine from source and you prefer, it can be installed as an Engine plugin.

If you have also licensed Mercuna 3D Navigation, then the plugin will provide both ground and 3D navigation capability. See the Mercuna 3D Navigation User Guide for documentation of the 3D Navigation capabilities.

Once installed, the Mercuna components, actors and menu will automatically be available when you next start the editor.

## Configuring Agent Types for navigation

Mercuna Ground Navigation enables path finding and following for pre-configured agent types. There is no restriction on the number of agent types. The properties of these agent types must be set up on a per-project basis. In the UE4 editor, under **Edit | Project Settings... | Plugins | Mercuna** you will find **Ground Agent Types**. These specify the parameters for the different kinds of agents that will navigate using Mercuna Ground navigation.



Configurable Agent Type properties

The properties of these agent types change the generation of the **Nav Grid** (see later) which marks the areas of the geometry over which your agent may travel. These parameters are:

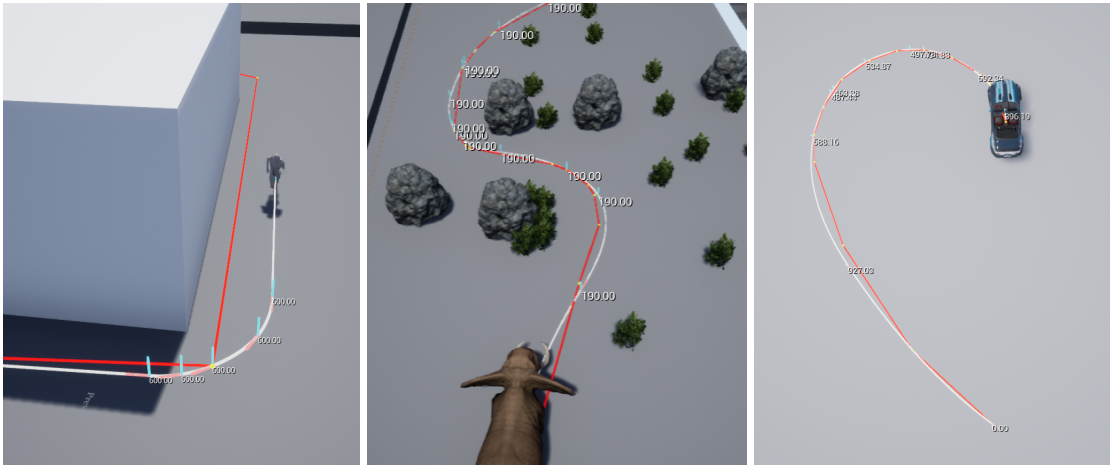
- **Category [Character/Animal/Vehicle]** - The category of the agent. This determines the kind of paths the agent can follow and how it can move, [more detail below](#).
- **Shape [Circle/Rectangle]** - The projected 2D shape of your agent, usually a circle for characters or a rectangle for animals and vehicles.
- **Pawn width** - How wide your pawn is (and hence how narrow a route you can path down). If the agent's shape is Circle, this gives the diameter of the agent.
- **Pawn length** - How long your pawn is (e.g. which destinations could your vehicle fit in lengthwise). If the agent's shape is Circle then this will automatically be set equal to the Pawn width.
- **Step height** - The height of the biggest step that an agent can step up or step down. This allows agents to go up stairs or make other steps between navigable surfaces that are at different heights.
- **Height clearance** - The height of your agent (i.e. the lowest tunnel your pawn can path through).
- **Max Slope Angle** - The maximum incline your pawn can ascend/descend (in degrees).
- **Max Angle Change** - The maximum slope angle between adjacent cells that should be considered navigable (reducing this prevents paths going over sharp peaks or troughs).
- **Navigable Materials** - Physical materials that the agent is allowed to navigate on. If any materials are specified here then the agent can only navigate on those materials



and no others. If no materials are specified here then the agent can navigate on any material except unnavigable materials.

- **Unnavigable Materials** - if no navigable materials are specified then the agent can navigate on any material except the physical materials specified here.

## Agent Categories



The agent category controls whether the agent is allowed to turn on the spot, and how edges of physical materials with different friction are considered.

**Characters** are allowed to turn on the spot, so a standard A\* pathfinder is used to find the shortest path to the character's destination regardless of how sharp the turns required are. Path smoothing is still applied so characters will avoid making sharp turns when there is space available to do so.

**Animals** prefer not to turn on the spot, so Mercuna's kinematic pathfinder is used. This means that animals will prefer to take corners in smooth arcs even if there is a shorter route that requires a sharp turn available. However, if there is no alternative the animal is able to turn on the spot.

**Vehicles** can't turn on the spot. Mercuna's kinematic pathfinder is used, but if the only route to a destination requires a turn sharper than the vehicle's minimum turning radius, then no path will be found.

Additionally, for vehicles the available surface friction is considered at the sides of the agent shape, to account for the positioning of the vehicle's tyres.

The agent category also determines the way in which the navigation component for navigating pawns can be configured, presenting options appropriate for that type of agent.



## Configuring the Navigation Grid

A **Mercuna Nav Grid** is used to find paths for characters, animals and vehicles.

The screenshot shows the configuration panel for the Mercuna Nav Grid. It is divided into two main sections: **Generation** and **Runtime**.

**Generation Section:**

- Supported Agent Types:** A list of agent types with checkboxes next to them:
  - Default: ☒
  - Character: ☒
  - Pickup: ☐
  - Elephant: ☒
  - Leopard: ☐
  - Horse: ☐
  - Roadster: ☒
  - Hatchback: ☐
  - Cargo Truck: ☐
- Cell Size:** A numeric input field set to 100.0.
- Physical Material Priorities:** A field showing 0 Array elements with a plus icon and a trash icon.
- Geometry Collision Channel:** A dropdown menu set to Pawn.

**Runtime Section:**

- Never Save:** A checkbox that is currently unchecked.

Example settings for Mercuna Nav Grid

A Nav Grid is generated in the level everywhere that is within a **Mercuna Nav Grid Volume**. A Nav Grid can simultaneously support multiple agent types, controlled by the **Supported Agent Types** setting. Create agent types in the project settings, as [detailed above](#).

**Cell Size** sets the resolution of the Nav Grid which Mercuna will use to find paths for agents. Using a smaller cell size will allow a more accurate representation of the navigable space, but more memory will be consumed and pathfinds will take longer. As a guide, we suggest setting cell size to half the width of your narrowest agent.

If you have agents of very different sizes, it can be more efficient to use a second nav grid with a larger cell size for the larger agents. We recommend that the largest agent using a nav grid should be no bigger than three times the size of the smallest agent.

The **Never Save** property indicates whether the nav grid will be built from procedurally generated data at runtime, so any navigation data generated in the editor should not be saved (see [Runtime Grid Building](#) below).

The **Geometry Collision Channel** specifies the collision channel used to query whether geometry should be considered for navigation by this nav grid. For a collider to be



considered by navigation it must have “Can Ever Affect Navigation” set to true, and give a Block response to this collision channel.

The **Physical Material Priorities** are [discussed below](#).

## Setting up the navigable space

To set up a nav volume, first add a **Mercuna Nav Grid Volume** actor into the level, and size it as required. Only boxes are supported, so to describe more complex boundaries of the navigable area, multiple Nav Grid Volumes can be configured.

If Precise Boundaries is switched on, the boundaries of the nav grid volume are considered hard edges, beyond which your agents can't move. However, if the Precise Boundaries option is turned off then navigable space will extend up to a high power of 2 boundary in the quad-tree. This allows a significant memory saving when there is open space around the navigable area and you don't need to precisely specify the edge of the area your agents will move within.



Example settings for Mercuna Nav Grid Volume

**LOD** and **Full Detail Build** settings are currently ignored by Mercuna Ground Navigation, they will be supported in a later version of the product.



The grid generation parameters to be used in this level are configured on the Mercuna Nav Grid actor, and on the Agent Types set in the project settings. Upon creation, the parameters are set to default values. These defaults can be modified in the Mercuna project settings.

## Nav Seeds

In order to identify which regions should be considered navigable, Mercuna requires you to place **Mercuna Nav Seed** actors into levels. This allows uninteresting regions, such as the small isolated areas inside hollow geometry or large areas outside of the level boundaries, to be excluded and avoids pawn positions getting clamped into disconnected regions.

A Mercuna Nav Seed needs to be placed on or above the ground in the area of the level where pawns will navigate. This seed is used during construction of the nav grid to find all connected reachable cells, by flood filling the region starting at the nav seed. Any area not connected to the seed will be considered unnavigable. If you have multiple disconnected areas in your level where you expect pawns to move, a seed must be placed in each separate area.

*Note that if you also have Mercuna 3D Navigation, the nav seed will also seed any nav octree.*

## Exclusion Volumes

A particular box volume can be completely excluded from navigable space by adding a **Mercuna Nav Exclusion Volume** actor to the level. By default the exclusion volume will be applied to all nav grids (and all nav octrees if you also have Mercuna 3D Navigation).

If you have multiple nav grids in a level, and only want the exclusion volume to apply to one of them, then you can explicitly specify which grid the exclusion volume is linked to via the volume's properties.

The exclusion volume may be rotated and scaled.

## Building the Nav Grid

Unless runtime generation is being used, the Nav Grid must be built after it is first configured or after the level geometry has changed. This can be done by selecting Build Grid from the Mercuna menu (accessed by clicking the Mercuna button in the Toolbar). An on screen notification displays the progress of the nav grid construction.

If full Mercuna logging is enabled (see [Logging](#) section below), then you will see the generation progress for each nav grid in the output log, and the total memory consumption of the grid is reported.





## Memory usage and Performance

The main influences on memory usage and performance are:

- **Cell size:** Smaller cell size nav grids use significantly more memory and take longer to generate and find paths through.
- **Density of geometry:** Large open navigable volumes are stored efficiently, volumes with dense geometry and narrow corridors use more memory and take longer to find paths through.

## Runtime Grid Building

It can sometimes be desirable to build the Nav Grid at runtime, particularly for procedurally generated levels. Since these grids will be generated at runtime it is recommended that they should have the **Never Save** property set on them. This prevents any data that might be generated in the Editor while testing from being unnecessarily saved.

The build can be triggered by making a request on the nav grid actor (via Blueprint or C++) using the **Build** function. Once nav grid generation has fully completed, the event **OnBuildComplete** is triggered.

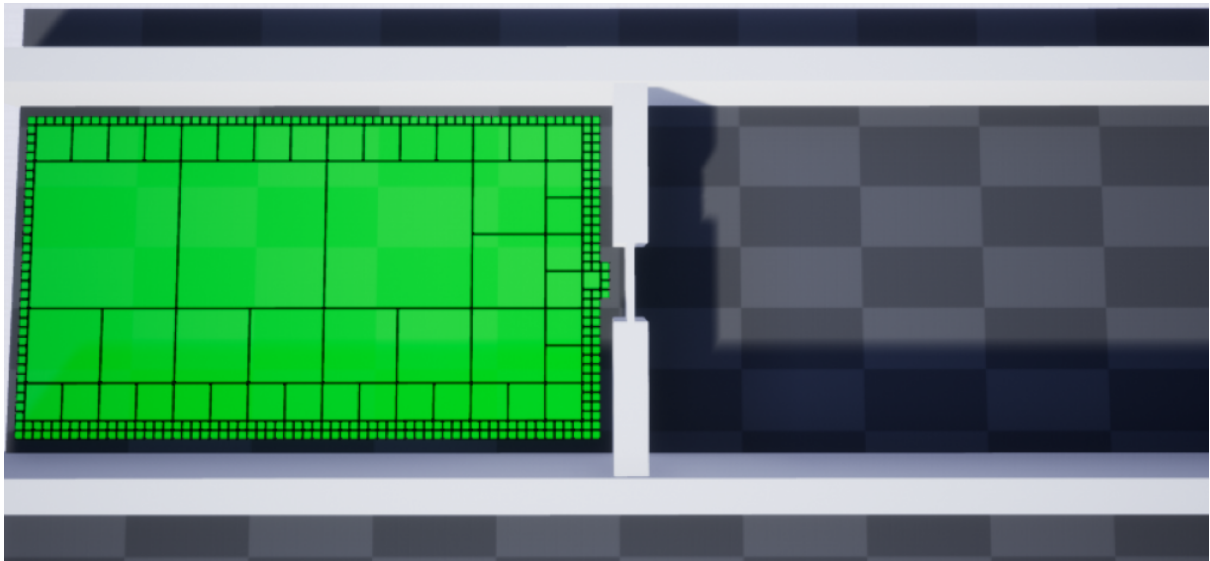
## Runtime Grid Rebuild

Specific volumes of the Nav Grid can also be rebuilt while the game is running using the **Rebuild Volume** or **Rebuild Volumes** functions on the Nav Grid actor. Rebuild Volume is normally used to pick up runtime changes when just a section of the level changes, such as a door opening.

Once the volume has been rebuilt, the event **OnRebuildComplete** is triggered. This event includes the volume that was regenerated, to aid with scripting.

Pathfinds and reachability tests that go through the regenerated region may fail while the regeneration is in progress. Once regeneration is complete, all active paths are recomputed if they go through or close to the regenerated region, causing actors to path around new obstacles or through newly available shortcuts. If a pathfind fails during regeneration, you may want to retry it once the OnRebuildComplete event has been triggered.

Regenerating a region will not cause newly connected areas outside the regenerated region to become seeded. If a runtime regeneration might connect a volume that is not connected to any other nav seed when the nav grid is built in the Editor, you must place a seed within that volume.



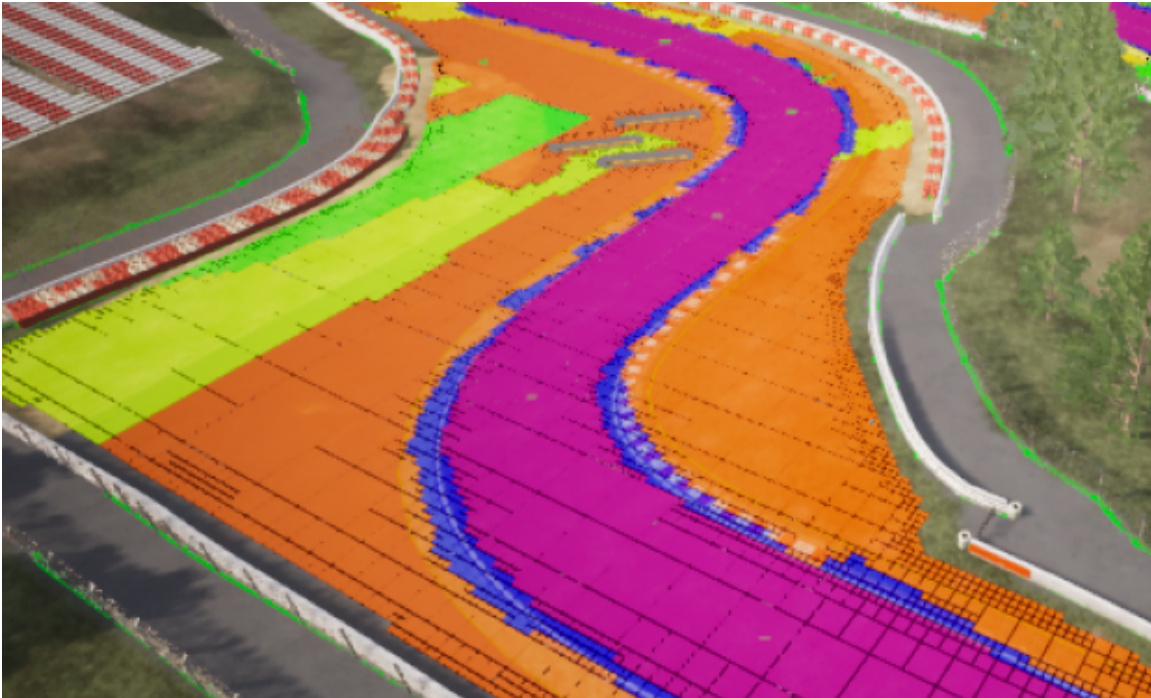
In the screenshot above, the green region on the left is navigable and seeded. The wall in the centre has a door in it, and when this opens Rebuild Volume is triggered through blueprint to regenerate the navigation data around the door.

However, navigation into the region on the right will still not be possible because that region was not seeded. This can be fixed by adding a Mercuna Nav Seed into the right hand region.

## Physical Materials

Physical Materials are used by Mercuna to determine the surface friction available for vehicles, as well as being one way of specifying regions of [higher path cost](#) (using a Mercuna Physical Material).

Therefore, it is important to check that the nav grid is picking up your physical materials correctly. You can do this with the **Cells - Surface Type** nav grid debug draw mode.



If meshes with different physical materials are overlapping, then the nav grid might pick the wrong one up during the build. To override which material is picked up, you can specify which physical materials have priority on the Mercuna Nav Grid actor.



Physical Material priority ordering

Any physical materials listed here will have priority in the order they are specified in the array, and all other physical materials will be given lower priority.

## Physical Materials on Landscapes

Mercuna picks up physical materials from landscapes if they are set up, but the Unreal Engine configuration can be confusing. If you are seeing unexpected behaviour then these notes may help:

- Any physical material set on the **Landscape Material** on the Landscape is ignored. You must set the **Default Phys Material** on the Landscape instead.
- For landscapes with multiple layers, the physical material is set on the LayerInfo.
- In UE4 versions prior to UE4.25, there was a bug that meant physical materials weren't applied to the Landscape after being set. Workaround this by toggling the



**Generate Overlap Events** setting in the **Collision** properties of the Landscape to cause UE4 to pick up the physical material change.

## Pathfinding

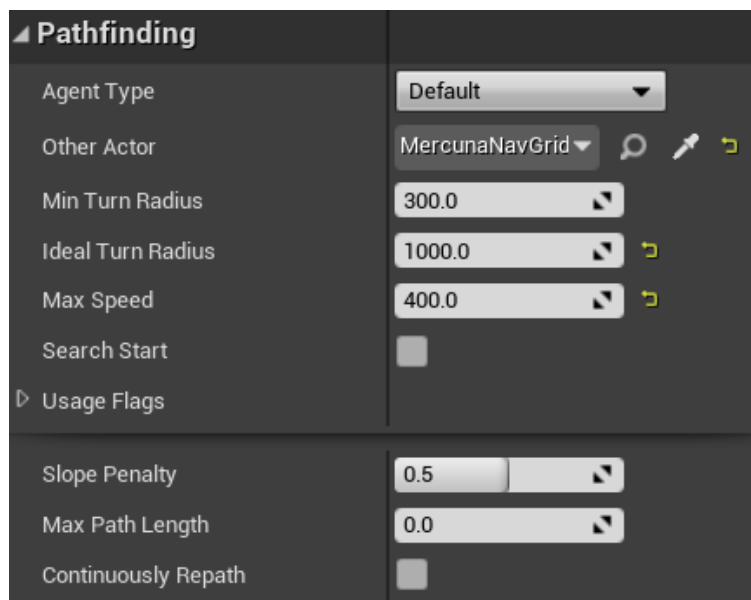
Finding paths through the Nav Grid can be done implicitly by making your pawn movement controlled by Mercuna, this method allows you to take advantage of the Mercuna steering system.

Mercuna also supports partial paths (disabled by default). A partial path is returned when a complete path can't be found to the specified destination, if it is disconnected from the start point, for example. Instead Mercuna returns a path to the closest point to the destination that is reachable.

## Nav Grid Testing Actor

In order to easily debug and understand pathfinding problems a pair of **Mercuna Nav Grid Testing Actors** can be used to generate test paths. Simply drag two testing actors into the level, and on one of the actors set the other one as the **Other Actor**, and set the **agent type**.

When you do this a test path will be drawn connecting the two actors. This path will update when either actor is moved. A white path means a complete path could be found, while an orange path means that only a partial path could be generated.

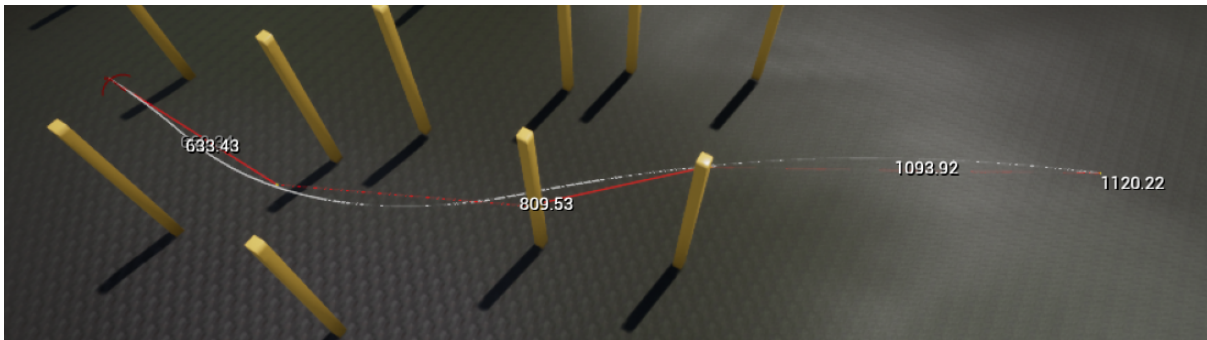


The **Min Turn Radius** property sets the radius of the smallest turning circle, and the **Ideal Turning Radius** sets the turning circle it ideally tries to turn at if space is available. The **Max Speed** allows for some basic speed integration.



# MERCUNA

**Slope Penalty** (0-1) indicates how much the agent disfavours slopes (note that this is relative to the maximum slope for the agent type). **Max Path Length** allows you to control how far the pathfinder will search.

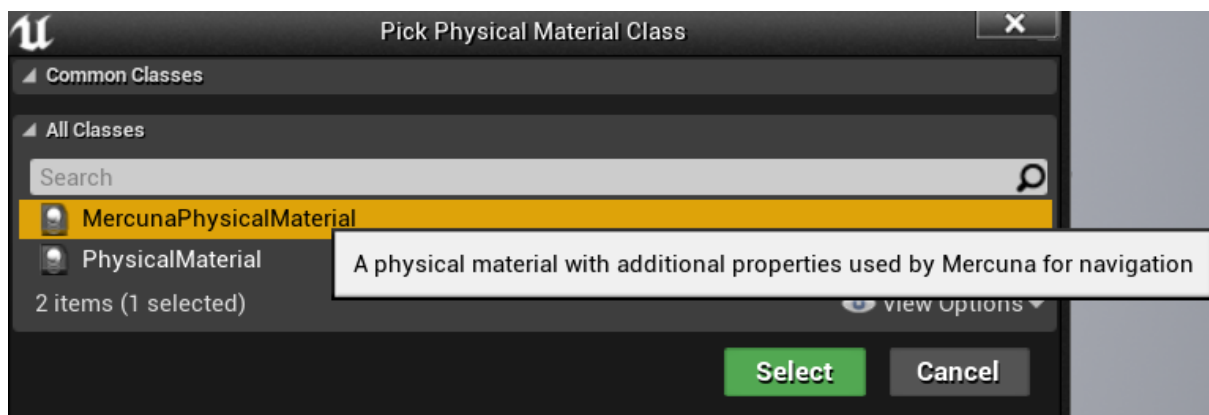


An example path between two Mercuna Nav Grid Testing Actors

## Navigation Cost Multipliers

There may be particular surfaces that agents should not path over. Mercuna supports a custom Physical Material to allow this to be specified.

To create a Physical Material with a navigation cost multiplier, create a new Physical Material blueprint and specify **MercunaPhysicalMaterial** as the class.



Creating a Mercuna Physical Material

The Mercuna Physical Material has an additional property, Cost Multiplier.



Mercuna will treat paths over any surfaces using this material as if they were **Cost Multiplier** times longer than their path length. This means if the multiplier is set above 1.0x then Mercuna will prefer paths that navigate around the surface with this material.

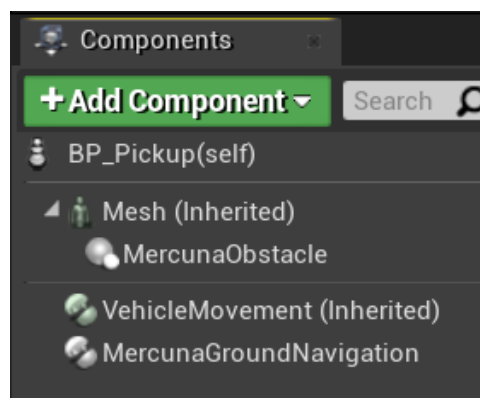


Due to the nature of the A\* algorithm used for pathfinding, costs can only be increased and not decreased below 1.0x. Using a very high cost multiplier will mean that the pathfinder will search a long way for alternatives, and thus can considerably increase the computational cost of the path find. For this reason, the maximum cost multiplier that can be set is limited to 15.0x.

## Creating a Mercuna navigated Agent

In order to allow pawns to use Mercuna to navigate they need to have the following components:

- **Mercuna Ground Navigation** - this component provides the pawn with navigation capabilities, accessible through Blueprint.
- **Mercuna Obstacle** - this marks the pawn as a dynamic obstacle for the purpose of ground navigation. The obstacle component must be a child of the root scene component.
- A suitable **movement component**, currently this must be either a **Character Movement** or **Wheeled Vehicle Movement** component



Vehicle blueprint setup for Mercuna navigation

## Navigation Component

The navigation component offers both a C++ and Blueprint interface for making movement requests to the pawn.

Mercuna will try to automatically find a Mercuna Nav Grid that is compatible with the **Agent Type** set on the Ground navigation component, and automatically fill in **steering parameters** based on the setup of the pawn. Alternatively, these can be explicitly overridden here.

For the purposes of navigation the size and shape of the agent will be taken from the Agent Type. When the navigation component is selected the agent's navigation shape will be drawn in the viewport so that you can check that it is a good fit for the agent.





Section	Parameter	Value
Navigation	Nav Grid	None
	Agent Type	Character
	Automatic Steering Parameters	<input type="checkbox"/>
	Steering Parameters	<input checked="" type="checkbox"/>
	Max Speed	600.0
	Max Acceleration	2000.0
Movement Style	Dynamic Avoidance	<input checked="" type="checkbox"/>
	Usage Flags	
Navigation	Nav Grid	None
	Agent Type	Roadster
	Automatic Steering Parameters	<input type="checkbox"/>
	Steering Parameters	<input checked="" type="checkbox"/>
	Max Speed	5000.0
	Max Throttle Acceleration	1000.0
Movement Style	Max Brake Deceleration	1500.0
	Dynamic Avoidance	<input checked="" type="checkbox"/>
	Usage Flags	
Movement Style	Stop at Destination	<input checked="" type="checkbox"/>
	Min Avoidance Time	1.0
	Slope Penalty	0.5
	Traction Estimate	1.0

Example ground navigation components where steering parameters have been explicitly set

## Example for a wheeled vehicle

For wheeled vehicles, check these settings in your agent type configuration:

- The **maximum slope angle** (in the project settings) - the vehicle may 'believe' that it can travel up slopes that it cannot, or cannot travel up slopes it is able to.
- The **step height** or **max angle change** is too high - The vehicle attempts to drive over terrain that is too rough

If you are manually setting the steering parameters that control how Mercuna expects the vehicle to behave, then

- **Max Speed** - The maximum speed of your vehicle
- **Max Throttle Acceleration** - The acceleration of your vehicle, limited by the power of your engine. Note that for high values you may be limited by the friction of the physical surface.
- **Max Brake Deceleration** - The deceleration provided by your brakes.

Specific to Vehicles, in the Movement Style there is the **Traction Estimate**, increasing this above 1 causes the vehicle to overestimate the available traction and make higher speed turns at the risk of sliding. Reducing the value has the opposite effect of underestimating the traction resulting in more cautious driving.

## Example for a slowly turning animal (e.g. a horse)

Mercuna ground navigation can be used to steer an agent that has a `CharacterMovementComponent`.



It is recommended to use velocity-based (rather than acceleration-based) path following (this is due to the UE character movement component rotating the character based on the acceleration vector input). To configure velocity-based path following make sure that **Use Acceleration for Paths** is disabled on the CharacterMovement component. It is also recommended that **Orient Rotation to Movement** is set.

If you are manually setting the steering parameters that control how Mercuna expects the animal to behave, then

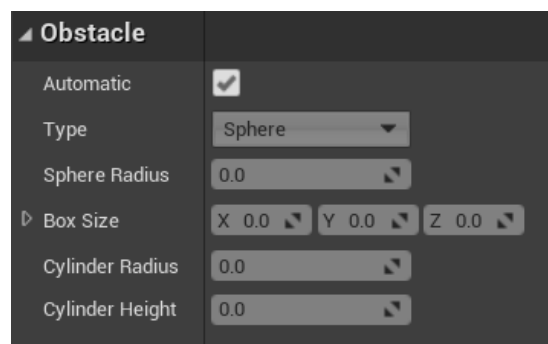
- **Max Speed** - The maximum speed of your animal
- **Max Acceleration** - Should be set to the maximum acceleration of your animal
- **Min Turning Radius** - Radius of the tightest turn your animal can make while moving

## Example for a humanoid Character

Settings should be similar to a slowly turning animal. By setting the agent category to Character the pathfinder will allow paths with sharp corners, taking advantage of the characters ability to change direction quickly.

## Obstacle Component

Any actor can be made a dynamic obstacle by adding a **Mercuna Obstacle** component. It is expected that Mercuna navigated pawns will usually have one, but other pawns such as the player, might also have an obstacle component so that the AIs avoid them.



Obstacle settings

By default Mercuna will automatically try to calculate the best shape of obstacle that encloses the pawn, for pawns with a **Mercuna Ground Navigation** component this will be set based on the agent type. Alternatively, the shape and size can be manually set. The available shapes are Sphere, Box, Cylinder. The shape only applies for avoidance.

Actors with an Obstacle component should have “Can Ever Affect Navigation” switched off on their Collision settings on all their components so they aren’t baked into the nav octree.

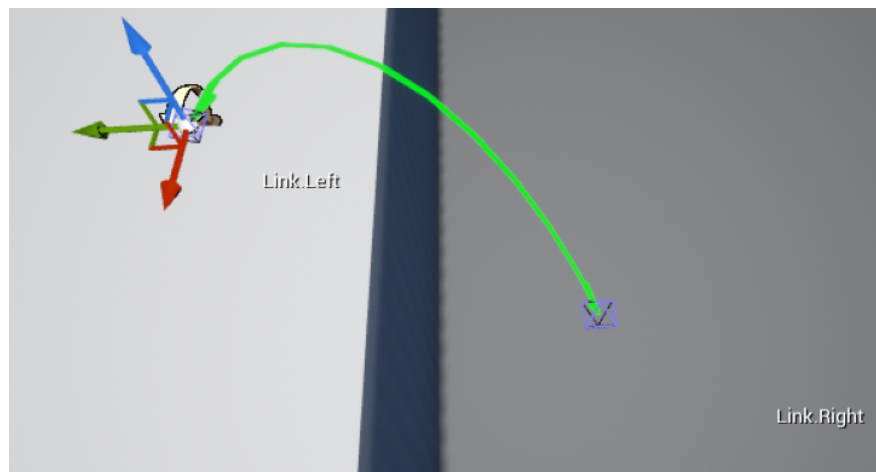




## Nav Links

Sometimes you will want agents to navigate in non-standard ways, for example climbing a ladder or making a jump. Mercuna Ground Navigation supports this through Nav Links.

A Nav Link has two ends, labelled Left and Right, and a connection is inserted into the navigation graph between the two ends when the nav grid is built. You can choose for the link to be navigable in one or both directions.



A simple navlink representing a jump

Your Blueprint or Game code is responsible for actually traversing the link, whether by teleportation, playing an animation or any other method desired. Once the agent reaches the end of the link, Mercuna takes control again and continues the navigation.

## Nav Link Creation

Normally, you will create a blueprint for each type of nav link supported in your game, and then place instances of these blueprints into your levels.

To create a Nav Link blueprint, add a new blueprint class, deriving from MercunaNavLink.

When an agent uses the nav link, the **Nav Link Start** event is triggered on the nav link. This supplies the Pawn navigating the link the direction the link is being navigated (left to right or right to left) and the position of the opposite end of the link.

When this event is fired your blueprint or game code should cause the agent to move to the destination, e.g. by triggering an animation.

To complete the nav link and resume the agent moving down the rest of the path, you can call the **Nav Link Complete** function on the **Mercuna Ground Navigation** component. Alternatively, you can set a non-zero auto completion distance for the destination end of the

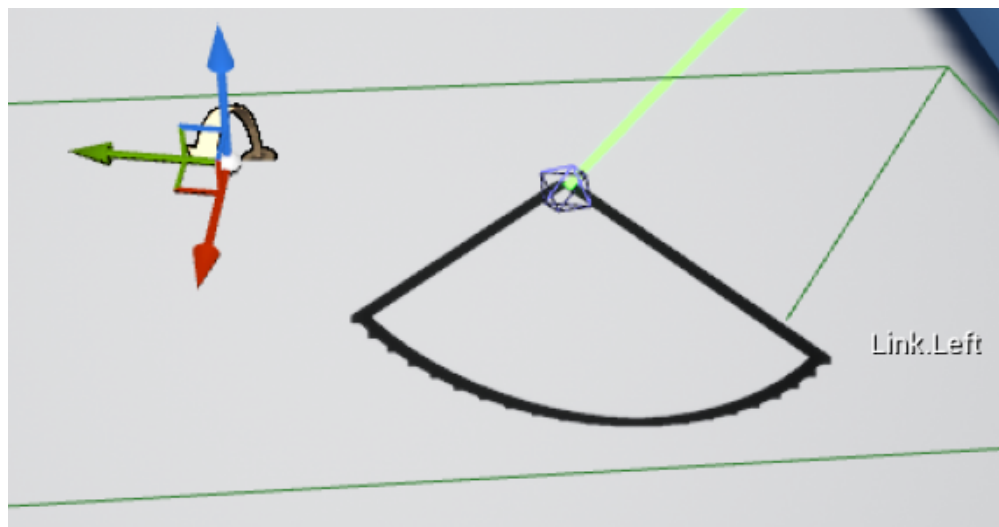


nav link, if this is set Mercuna will automatically trigger link completion once the agent is within that distance of the destination.

## Configuring the Nav Link

After selecting the nav link actor, you can position the ends of the nav link by clicking on them and then moving them in the normal way.

If you specify a limited **Entry Angle** range then you can also rotate the end to determine the acceptable range of entry angles, the black arc represents the valid entry angles.



## Cost

The path finder normally considers the cost of moving through the navigable space to be equal to the distance the agent needs to move. For nav links, the cost can be adjusted to make agents prefer to use the link, or prefer not to. By default, the cost is set to the distance between the endpoints.

Note that if the link has a lower cost than the straight line distance, then it will only be used preferentially if the path finder discovers the link during its search. This won't happen if a straightforward path is available, but the agent would have to go backwards to find the cheaper nav link, for example.

Nav Link	
Left	
Right	
Direction	Left to Right
Use Distance as Cost	<input checked="" type="checkbox"/>
Left to Right Cost	363.5931702
Right to Left Cost	363.5931702
Left Auto Completion D	0.0
Right Auto Completion	0.0
Left Entry Angle	45.0
Right Entry Angle	180.0
Supported Agent Type:	
Apply to All Nav Grids	<input checked="" type="checkbox"/>
Nav Grid	None
Enabled	<input checked="" type="checkbox"/>



## Agent Types and Nav Grids

You must specify which agent types the nav link supports. By default, the nav link will be associated with all nav grids that have navigable space at the endpoints of the link, if you want to restrict it to a particular nav grid, you can specify that explicitly.

## Steering

In order to give smooth, natural looking movement Mercuna uses polynomial curves to interpolate between path points to generate smooth curves. These splines are constructed taking into account the available space.

Whereas the pathfind is performed when an agent is given a destination, the smooth curve is generated on demand. If the agent is pushed, or has to avoid other actors causing it to move away from the curve, then the curve and, if necessary, the underlying path are automatically regenerated.

## Avoidance

Mercuna offers dynamic obstacle avoidance to ensure that pawns don't collide while moving. Any actor with a **Mercuna Obstacle Component** is automatically considered as an obstacle that needs to be steered around for the purpose of avoidance. For the purpose of avoidance agents will be treated as circles if the obstacle is a sphere or a cylinder, or as a rectangle for box shaped obstacles such as vehicles (if the rectangle is rotating rapidly it is treated as a circle).

The avoidance algorithm used by Mercuna is a modified version of ORCA velocity obstacle method that additionally takes into account the fixed level geometry stored in the nav grid. In the **Mercuna Ground Navigation Component** there is a **MinAvoidanceTime** which can be adjusted to change how early the agents attempt to avoid a collision (shorter times mean the agents move past each other more aggressively). A default avoidance time of around 0.3s is recommended for characters, or 1s for vehicles.

## Blueprint Functionality

The following functions are available on the **Mercuna Ground Navigation Component** and can be used to direct an agent to move between goals:

- **MoveToLocation** - move to a position, stopping within end distance of the goal.
- **MoveToLocations** - move through a series of positions, visiting them in order.
- **AddDestinationLocation** - add an additional destination to the end of the current path. The agent must currently be moving to a position or series of positions.



- **MoveToActor** - move to within a given end distance of a destination actor, if the destination moves while the pawn is moving, the path will be updated to track the destination.
- **CancelMovement** - immediately terminates the agent's current movement action
- **ConfigureMovement** - prompt the actor to reconfigure itself, for example if it has been moved between nav grids or the vehicle configuration has changed. Any current move actions are cancelled.
- **OnMoveCompleted** - a delegate that is triggered whenever a movement action is complete. Returns the result of the movement action as to whether it completed successfully, failed, was cancelled or was invalid (usually due to an invalid destination).
- **GetNavGrid** - Get the current nav grid this pawn is using.

## EQS

Mercuna currently offers three simple EQS tests. These tests are filters returning whether a point passes or fails, and do not score the points. The available **Mercuna Ground** tests are:

- **Navigable** - test whether a point is within the seeded, navigable area for a pawn. Be aware that the points might be disconnected from the querying pawn, however this test is much cheaper than a reachability test.
- **Reachable** - test whether a point is reachable by a pawn from its position within a specified path distance. If the path distance is set to 0.0, then a faster test is used that only filters out points that are not connected to the pawn at all. This test does not consider the turn rate limits of the pawn, so it only guarantees that a path can be found to the target for pawns that can turn in place.
- **Raycast** - test whether there is a clear straight line path from the context to the positions.

## Debugging Problems

If you find that your pawn is not moving as expected, or at all, there are several debugging mechanisms available within Mercuna to help quickly identify problems.

## Logging

Mercuna makes logs to the Unreal logging system to indicate progress and error conditions. By default, only Warning and Error logs are written, to enable progress information add the following to DefaultEngine.ini:

```
[Core.Log]  
LogMercuna=Info
```

If more information is needed then Editor Preferences > Mercuna > **Enable Extra Logging**, can be enabled. This option persists between editor restarts. Extra Logging causes Mercuna

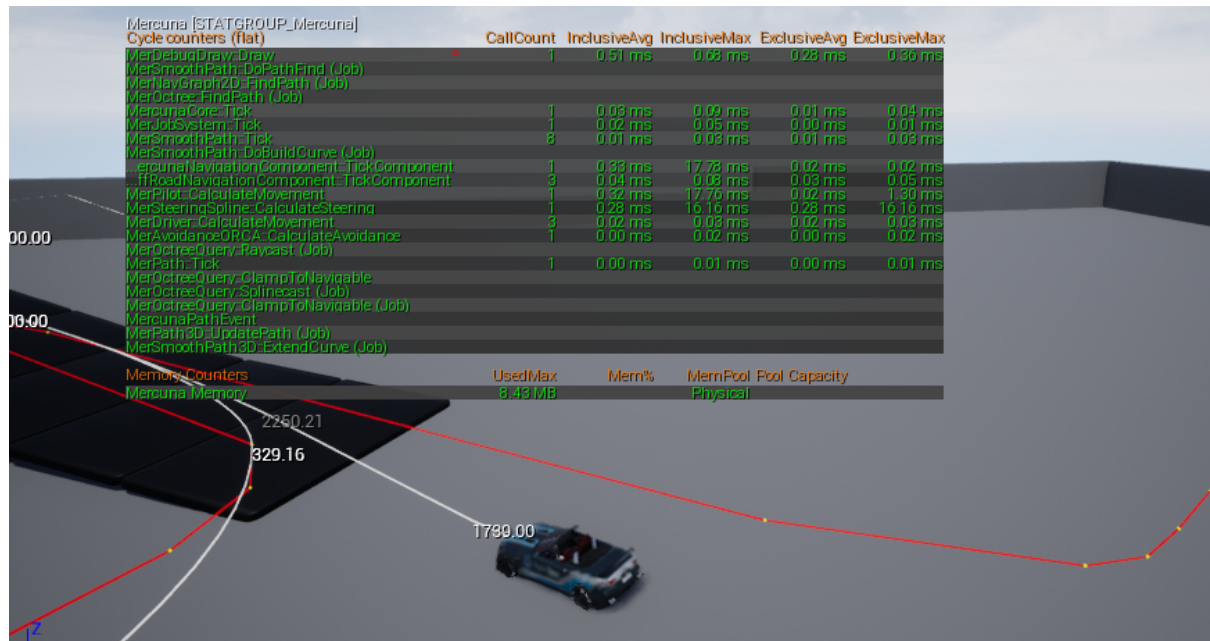


# MERCUNA

to output all log messages, including additional debug messages, to a dedicated **Mercuna.log** file, located in the same directory as the standard Unreal logs.

## Profiling

Mercuna is integrated with Unreal Engine's inbuilt profiler. The current amount of time Mercuna is taking to run each frame, as well as the current memory usage, can be seen using the **stat Mercuna** console command.



Example Mercuna in editor profile

If an entry in the profiling list has *(Job)* after it's name, then it is being run as an asynchronous job on a background thread and so will normally have no effect on the frame rate.

## Debug Actor

When trying to understand the actions of a particular agent, it can be useful to set it as the Mercuna debug actor. This can be done by selecting the agent and setting it as the Mercuna debug actor in the Mercuna toolbar menu. The same menu also allows the debug actor to be cleared.

On screen log messages will be displayed for the Mercuna debug actor and additional debug draw is available.

In Debug builds or if the define `MER_DEBUGGING` is set, then additional debug logs will also be recorded to `Mercuna.log` for the debug actor. If you report a movement problem to Mercuna support it is helpful to include these logs and a video capture of the problem.



## Debug Draw

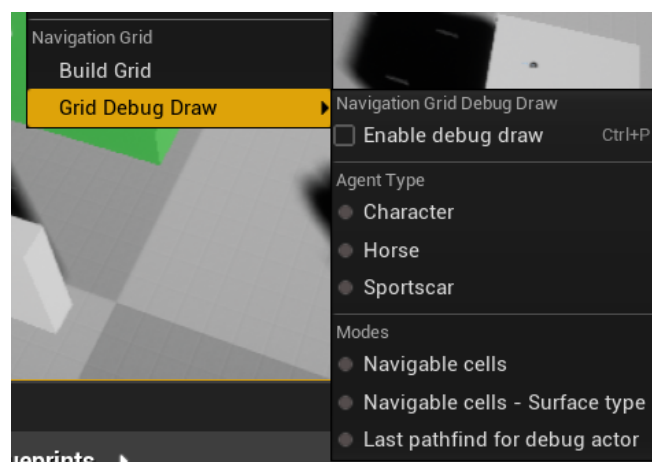
In order to help understand the current movement of Mercuna controlled actors the following debug draw is available from the Mercuna editor menu:

- **General:** Shows speed and velocity vector for all Mercuna actors
- **Obstacle bounds:** Shows blue spheres representing the dynamic obstacles registered with Mercuna
- **Paths:** Shows all paths currently being followed by Mercuna actors
- **Steering:** Shows the desired velocity vector for the current debug actor
- **Avoidance:** Shows various pieces of avoidance related debug draw for the current debug actor including the velocity obstacle cones and the ORCA planes.

## Navigation Grid

### Debug Draw

In order to help understand Mercuna's representation of the geometry for navigation, you can draw the navigation octree, there are the following modes:



If you only have a single **Agent type** defined in your project, the navigation properties for this will be automatically displayed, otherwise you will need to select the desired agent type.

- **Navigable cells** - Which cells an agent could traverse (in green).
- **Cells - Surface type** - Color the cells by surface type
- **Last pathfind for debug actor** - Grid representation of the last pathfind showing A\* costs

## Known Issues

Known issues in Mercuna Ground Navigation:

- Nav Links that cross into or out of the rebuilt area aren't handled correctly by runtime nav grid builds



- If a pawn is very close to the start of a nav link when it begins moving, it may not reach the correct orientation before entering the nav link.
- Hierarchical pathfinding is not yet available for vehicles and animals using the kinematic pathfinder, so the maximum path distance for these categories of agent is limited.
- When generating the navigable area, the agent step height is added to the agent's height clearance. This ensures that the agent can safely go up any steps, but is restrictive when the agent wants to navigate under low obstacles.
- The path distance limit on the EQS Is Reachable test doesn't work.
- Avoidance doesn't work correctly for agents navigating on Nav Grids that have up direction different to +Z.
- Nav Grid is generated above and below the specified bounds
- There is currently a limit of 254 vertically stacked layers while building the nav grid
- Avoidance can produce oscillating outputs in situations with more than 2 agents