MERCUNA

UE4 User Guide

v1.2

MERCUNA

Contents

Installing

The Mercuna middleware is integrated into Unreal Engine as a standard plugin compatible with Unreal Engine 4.17+.

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.

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

Configuring the Navigation Octree

A **Mercuna Nav Octree** is used to find paths for pawns through 3D space, much like a 2D Nav Mesh is used to navigate pawns over terrain.

Nav Octree is generated in the level everywhere that is within a **Mercuna Nav Volume**. A Nav Octree can simultaneously support multiple actor sizes (within limits), so in most cases a single octree should be sufficient. However, more complex setups with multiple Nav Octrees in a single level are possible if support for very different actor sizes is required.

Multiple Nav Volumes can be present in one level, and overlapping Nav Volumes that are linked to the same Nav Octree allow pawns to navigate seamlessly between them. All space outside of the Nav Volumes is treated as unnavigable.

Setting up the navigable space

To set up a nav volume, first add a Mercuna Nav Volume actor into the level, and size it using the Brush Settings. Only axis aligned boxes are supported, so you must not rotate or scale the volume, and you must leave the brush shape set to Box. To describe more complex boundaries to the navigable volume, multiple Nav Volumes can be configured.

Example settings for Mercuna Nav Volume

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

Example settings for Mercuna Nav Octree

The parameters determine how detailed the representation of the navigable space is in the octree, and the sizes of pawns that can accurately navigate through it.

The **cell size** determines the side length of the cubes that make up the lowest level of the octree. Cells are considered unnavigable if there is any level geometry within them, so the larger the cell size the greater the error margin in the representation of the geometry.

The **minimum** and **maximum pawn radius** determine what navigation data is stored in the octree. Paths will never go closer to geometry than the minimum pawn radius, and the octree doesn't store data to allow paths to be found with more clearance from geometry than the maximum radius.

The radius is expressed as multiples of cell size, so for a cell size of 30, a minimum radius of 2 and a maximum radius of 5, entities of radius between 60 and 150 can be accurately navigated through the level. Entities that are smaller than the minimum radius will navigate successfully, but might not take paths through small gaps they could fit through. Entities that are larger than the maximum radius aren't supported, as their paths might make them collide with geometry.

The **runtime regeneration** property indicates whether the navigation data is built from procedurally generated data at runtime. If this flag is set, then any navigation data generated in the editor isn't saved, instead the octree must be built after the level is loaded - see Runtime Octree Rebuild below.

Level of Detail

It is possible to specify that part of the navigation octree is built at a different level of detail by setting the LOD on a nav volume. This would normally be used when you want high precision navigation in particular parts of the level without the expense of using a high resolution octree across the entire level.

The normal set up is to create a large nav volume with ½ or ¼ level of detail, and then specify smaller overlapping nav volumes that generate particular areas at full detail. It is also possible to set the large volume to full detail and add smaller volumes at lower detail.

When generating the octree, Mercuna assumes that smaller nav volumes override the level of detail of overlapping larger volumes.

When an area of the octree is generated at lower level of detail, it is as if the voxel size in that area has been increased, so for example a ½ LOD area in an octree with a voxel size of 30 will have an effective voxel size of 60.

In low level of detail areas, agents won't be able to navigate as close to walls or through as small gaps as they would if full LOD was used. However, the generation time and memory usage for low LOD volumes is substantially less than for full LOD volumes.

Multiple Octrees

Mercuna supports multiple Nav Octrees within a single level, which is normally used to allow navigation for agents of very different sizes. In this case, you must select which Nav Octree each Nav Volume is associated with - each Nav Volume can only be linked to a single Nav Octree.

By default, Mercuna is configured to automatically link Nav Volumes to Octrees, which means that no manual configuration is required when there is only a single octree in the level. If no Octree is present in a level, Mercuna will automatically create a Nav Octree actor when the first Nav Volume is added to the level, and link the Nav Volume to it.

If automatic linking is disabled (in Project Settings -> Plugins -> Mercuna) then you must manually create Mercuna Nav Octrees and link the Nav Volumes to them. This can help avoid errors when using multiple octrees, as the Octree must be set explicitly, rather than Nav Volumes potentially being automatically linked to a different Octree to the one that was intended.

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 to the wrong side of polygons.

A Mercuna Nav Seed needs to be placed in the main part of the level where pawns will move. This seed is used during construction of the octree to find all connected reachable cells, by flood filling the region starting at the nav seed. If you have multiple disconnected areas in your level where you expect pawns to move, a seed must be placed in each separate area.

Building the Octree

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

When multiple Octrees are present in the level, the menu changes to show Build All Octrees and Build Selected Octree. This allows you to build all the octrees at once, or select a specific Octree and just build that one.

If full Mercuna logging is enabled (see below), then you will see the generation progress for each navigation volume in the output log, and the total memory consumption of the octree is reported.

Multiple levels

Mercuna supports both level streaming and world composition by saving the octree that is relevant to each streamed or composed level (sub-level) within that level.

When using level streaming or world composition, Mercuna Nav Volumes and Mercuna Nav Seeds should be placed in the sub-level so that they are loaded and unloaded at the correct times. Nav Volumes and Nav Seeds should only be placed when editing the sub-level.

When editing the Persistent level you will see one Mercuna Nav Octree in each sub-level, this is automatically associated with the Nav Volumes and Nav Seeds that are in that sub-level.

Mercuna does not currently support seamless navigation between Octrees from different levels. However, it is possible to use the Set Nav Octree function on the Mercuna Navigation Component to switch an agent from navigating on an Octree loaded from one level to an Octree loaded from another level.

The best way to generate the octrees is from the Persistent level. Load all your sub-levels using the levels window and then select Build All Octrees from the Mercuna menu. This ensures that geometry that overlaps between levels is correctly represented in each level's octree. Once generation is complete, save all the sub-levels.

If it is not possible to load all levels due to memory constraints, load and generate groups of levels at a time, saving only the levels for which all overlapping geometry is included after each generation.

Memory usage and Performance

The main influences on memory usage and performance are:

- **Cell size**: Smaller cell size octrees use significantly more memory and take longer to generate.
- **Density of geometry**: Large open navigable volumes are stored efficiently, and are quick to pathfind through, volumes with dense geometry and narrow corridors use more memory and take longer to find paths through.
- **Maximum pawn radius**: Larger maximum pawn radiuses take longer to generate and slightly increase memory usage.

Runtime Octree Rebuild

The octree can be built or rebuilt while the game is running. The build is triggered by making a request on the octree actor (via Blueprint or C++). Two functions are available: **Rebuild All** to request a complete rebuild of the whole of an octree, or **Rebuild Volume** to rebuild just a portion of the octree. Normally, the Rebuild All option is used after a level has been procedurally generated, whereas Rebuild Volume is used to pick up runtime changes, such as a door opening.

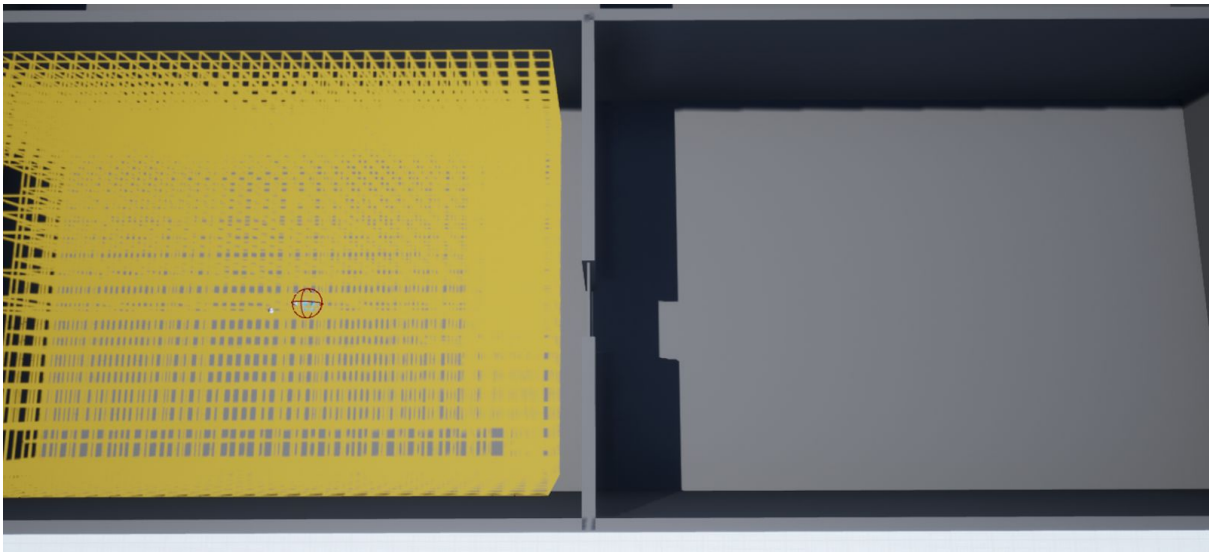
When the whole octree is rebuilt with Rebuild All, the generation happens in two phases - first a low level of detail version of the octree is built. This build completes quickly and allows agents to start navigating within open spaces (but not close to geometry). The event **OnRebuildLowResReady** is triggered once this low resolution octree is available and indicates that agents can start navigating.

The two phase build is optional when only part of the octree is rebuilt - use the Staged Build flag on Rebuild Volume to specify whether or not to use it. When rebuilding only small volumes it is more efficient to do a single phase build.

Once octree generation has fully completed, the event **OnRebuildComplete** is triggered. This event indicates the volume that was regenerated (if any), 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 navigation octree is built in the Editor, you must place a seed within that volume.



In the screenshot above, the yellow 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.

Pathfinding

Finding paths through the Nav Octree can be done implicitly by making your pawn movement controlled by Mercuna, this method allows you to take advantage of the Mercuna steering and avoidance systems. Alternatively, path finding can be requested explicitly by making a request directly to the octree (via Blueprint or C++) and receiving a MercunaPath (or MercunaSpline) object which can then be used as required.

As pathfinding is performed asynchronously, the returned MercunaPath/MercunaSpline object is not immediately valid, but can take one or two frames to complete. You must either check each frame to see if it is ready yet or subscribe to its PathUpdated/SplineUpdated delegate.

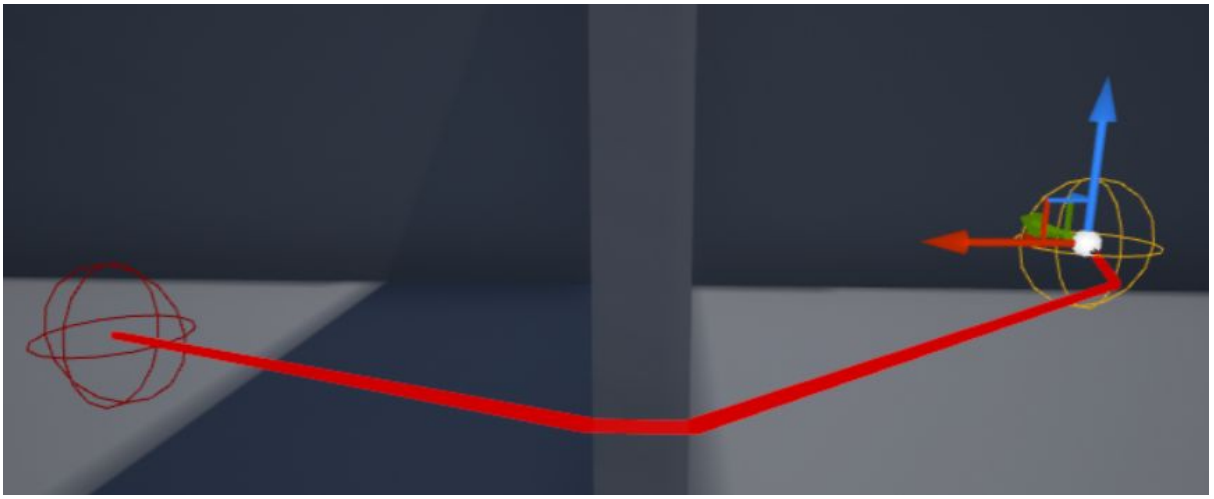
For longer paths Mercuna uses hierarchical pathfinding. An approximate path is found through a simplified representation of the level and then a detailed path find is performed guided by the approximate path. This allows much longer paths to be found than would be possible with simple A* pathfinding alone.

Mercuna also supports partial paths (enabled 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.

Path Testing Actor

In order to easily debug and understand pathfinding problems a pair of **Mercuna Nav 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' property. When you do this a test path will be drawn connecting the two actors. This path will update when either actor is moved. A red path means a complete path could be found, while an orange path means that only a partial path could be generated.

The Radius property specifies how much clearance there should be around the test path - this allows you to check what path larger and smaller actors would take.

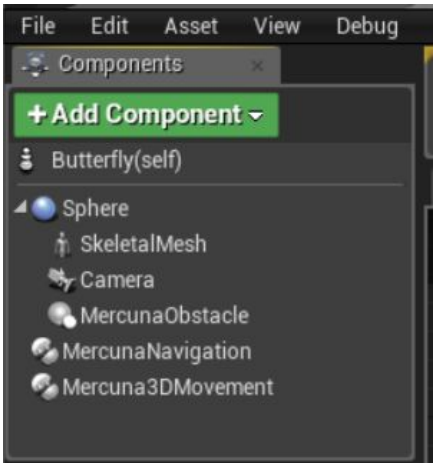


A test path between two Mercuna Nav Testing Actors

Creating a Mercuna navigated Pawn

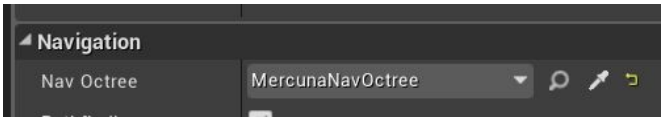
In order to allow pawns to use Mercuna to navigate they need to have the following components:

- **Mercuna Obstacle** - this marks the pawn as a dynamic obstacle for the purpose of 3D navigation. The obstacle component must be a child of the root scene component.
- **Mercuna Navigation** - this component provides the pawn with navigation capabilities, accessible through Blueprint.
- A suitable **movement component**, e.g. the **Mercuna 3D Movement** component



Pawn blueprint setup for Mercuna navigation

When there are multiple octrees in a level, Mercuna will automatically try and choose the best one to use based on which nav volume the pawn is currently in and which octree best fits the pawn's size. The octree that a pawn uses can be overridden by explicitly setting the Nav Octree parameter on the pawn's navigation component.



Navigation octree to use can be overridden on the navigation component

Setting a Flight Style

How a pawn moves depends on how its flight style is configured. The flight style options are set on the **Mercuna Navigation** component:

Flight style options

The options include:

- **Max Pitch** - The default maximum angle that the pawn can pitch up or down. This is ignored if Move Only In Look Direction is set.
- **Max Roll** - The maximum roll when turning, in degrees. Causes the pawn to bank when turning.
- **Move Only in Look Direction** - Whether to restrict movement to facing direction. This stops pawns from flying backwards when turning around.
- **Stop at Destination** - Whether should slow down in order to stop at its destination.
- **Smooth Paths** - Whether to use spline based paths, or simple straight line segment paths. See Steering section for more detail.

Steering

In order to give smooth, natural looking movement Mercuna uses polynomial splines to interpolate between path points to generate smooth curves. These splines are constructed taking into account available space and define a continuous velocity curve that can be followed precisely. Mercuna navigates the pawn down the spline path taking into account the maximum speed and acceleration configured on the pawn.

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

Spline based steering can be disabled to fall back to previous simple steering by turning off **Smooth Paths** in the flight style options. However, the simple steering method suffers from the problem that pawns frequently overshoot corners and fall off the path, resulting in collisions with level geometry and it is not recommended to be used unless necessary for backwards compatibility.

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. 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 octree.

It is possible to specify particular actors to be excluded from avoidance on a per pawn basis (using the SetAvoidanceAgainst function on Mercuna Navigation Component). A common use case is temporarily turn avoidance off against the player when executing an attack to avoid the attacking pawn veering off as it gets close to the player.

Movement

In order for the Mercuna navigation component to drive the moment of a pawn, the pawn needs to have a suitable movement component. A simple default movement component is

provided - the **Mercuna 3D Movement Component**. This is suitable for a variety of 3D flight styles.

Custom movement components can easily be implemented, but in order to be used by Mercuna they must provide the **IMercuna3DMovement** interface. The Mercuna Navigation component automatically detects and uses the first movement component it finds on the pawn that provides that interface.

Mercuna 3D Movement Component

The Mercuna 3D Movement component provides a Newtonian flight model for a pawn moving freely in space. It allows you to configure:

- **Max Speed** - The maximum speed the pawn may move at
- **Max Accel** - The maximum acceleration of the pawn in each axis direction (in the pawn's local coordinates).
- **Max Ang Speed** - The maximum angular speed the pawn may rotate at, in radians/sec
- **Max Ang Accel** - The maximum angular acceleration allowed on each axis, in radians/sec

Blueprint Functionality

The following functions are available on the Mercuna navigation component and can be used to direct a pawn to move between goals:

- **MoveToLocation** - move to a position, stopping within end distance of the goal.
- **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.
- **TrackActor** - get to and stay within a given distance of a target actor
- **Stop** - bring the pawn to a complete stop as quickly as possible.
- **CancelMovement** - immediately terminates the pawn's current movement action
- **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).

- **LookAt** - specify a target actor that the pawn will try to face as it moves. If no look at target is set then by default the pawn faces in the direction of movement.
- **CancelLookAt** - clear the look at actor.

- **CheckReachable** (Latent action) - Test whether the pawn would be able to move to a given destination position from its current position.

The MercunaNavOctree actor offers the following Blueprint functions:

- **IsNavigable** - Does a point fall within navigable space
- **ClampToNavigable** - Clamp a position to the nearest point in navigable space
- **Raycast** - Perform a raycast through the navigable octree. If it fails, return the first point it hit
- **CheckReachable** - Check whether there is a path from Start to End

- **FindPathToLocation** - Start an asynchronous path find from Start to End positions
- **FindPathToActor** - Start an asynchronous path find from Start position to End actor. Path will update as the destination actor moves
- **FindSplineToLocation** - Start an asynchronous path find from Start to End positions, and build a spline giving a smooth curve through navigable space along the resulting path.

- **RebuildVolume** - Rebuild the navigation octree within the bounds of the given actor.
- **RebuildAll** - Rebuild all of a navigation octree. Normally used after procedural generation of a level is complete.
- **OnRebuildComplete** - a delegate that is triggered once a RebuildVolume or RebuildAll has completed.
- **OnRebuildLowResReady** - a delegate that is triggered when RebuildAll or a staged RebuildVolume has finished building the low resolution data, to indicate that navigation may now be possible.

EQS

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

- **Navigable** - test whether a point is within the seeded, navigable volume for a pawn of a given radius. 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 of a given radius 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 with extremely long paths.
- **Raycast** - test whether there is a clear straight line path from the context to the positions.

Additionally, Mercuna offers one EQS test that modifies the positions of the test points:

- **Project** - test whether a point is in or close to navigable space. If the point is outside then clamp it to back to the closest point within navigable space that is within a given search radius. The test fails and the point is not moved if the point is further than the search radius from navigable space.

Mercuna offers two EQS generators, they simply generate points without considering whether the resulting points are in navigable space or not. Add the Mercuna Navigable or

Reachable EQS tests to filter out points in navigable regions or inside objects. The two available generators are:

- **Sphere** - generates points in concentric shells either uniformly or randomly distributed.
- **3D Ring** - generates points in rings in multiple vertical layers.

BT Nodes

Mercuna offers the following Unreal BT nodes:

- **Reachable** - Decorator - Test whether a given point is reachable by a particular actor
- **MoveTo** - Task - Move a pawn with a Mercuna navigation component to a specified location or actor read from the AI's blackboard. If the blackboard value is a location and the value changes while the node is running the path will attempt to update.

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

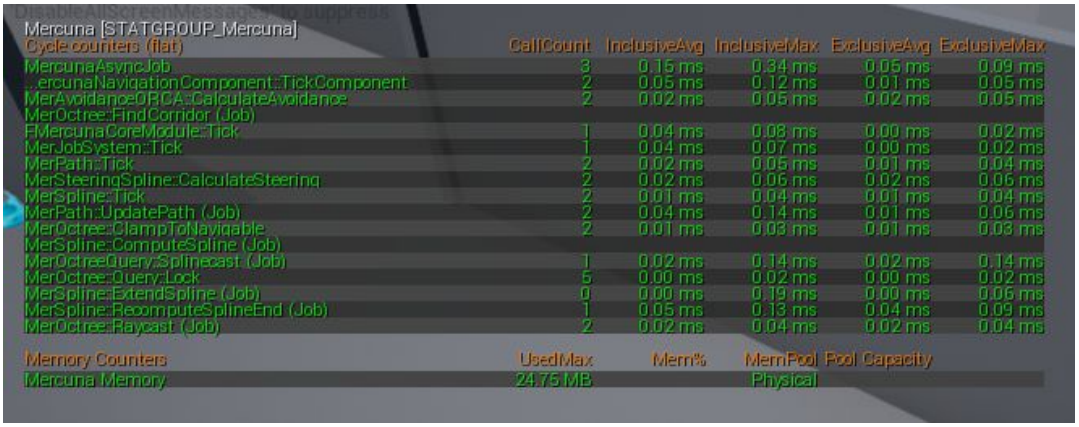
Mercuna makes logs to the Unreal logging system to indicate progress and error conditions. By default, only Warning and Error logs are written, to enable progress information add the following to DefaultEngine.ini:

```
[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 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.



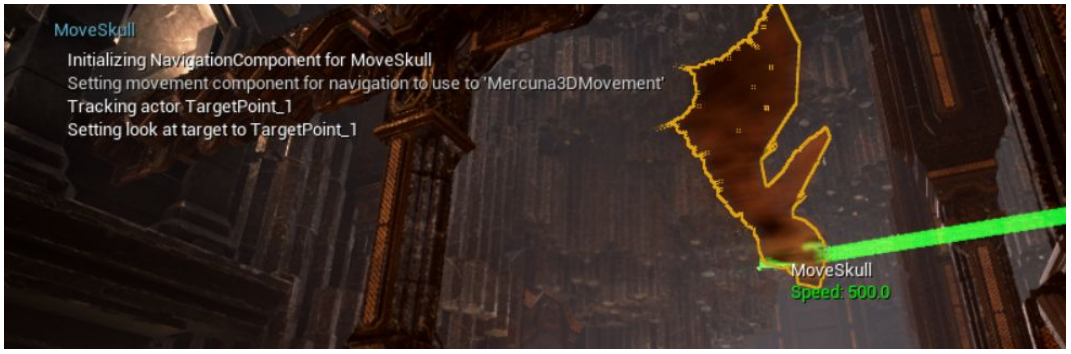
Example Mercuna in editor profile

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 pawn, it can be useful to set it as the Mercuna debug actor. This can be done by selecting the pawn and setting is as the Mercuna debug actor in the Mercuna toolbar menu. The same menu also allows the debug actor to cleared.

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



Onscreen logging and information available about the debug actor

Debug Draw

In order to help understand the current movement of Mercuna controlled pawns 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 Octree

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:

- **Unnavigable**: Draw the unnavigable part of the octree (in red).
- **Navigable**: Draw the navigable part of the octree (in green).
- **Navigable and unnavigable**: Draw both.

If the Mercuna debug actor is set, then the navigation radius of that actor is used to determine which cells are treated as unnavigable.

There are also options to particular cells in the octree:

- **Pathfind**: This will give a visualization of the last pathfind through the octree made by the Mercuna debug actor, or by a Mercuna Nav Testing actor. Explored cells are shown in green, cells that are on the path are shown in cyan.
- **Reachability**: This shows the cells that were included in the last reachability query. Like the pathfind, it gives a visualization of how the test flooded through the octree from the test point.

Troubleshooting

Once the octree has built, the octree debug draw can be used to check the geometry has been built into the octree correctly. Switch on **Unnavigable** debug draw through the Mercuna menu, and you should see red boxes, representing unnavigable regions, around your geometry:

Known Issues

- Runtime octree regeneration is cancelled by a world origin shift
- Agents can fall off splines when making tight corners
- Asking agents to go to a destination behind them when using spline steering can result in a very slow U turn.