PathNavigator3D for Unity manual

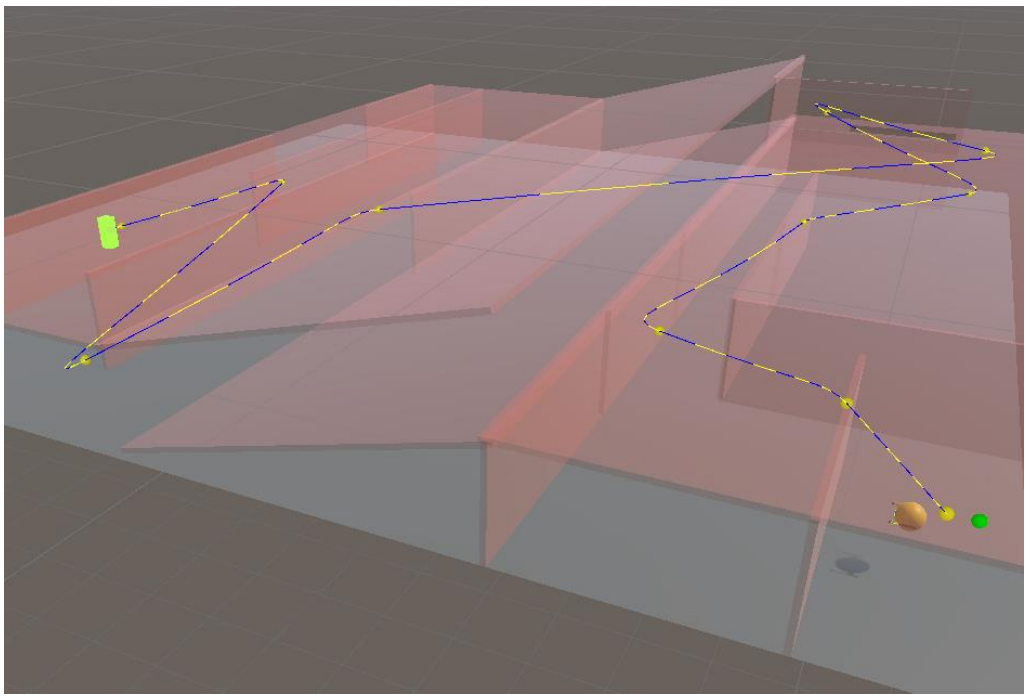
Contents

General description

PathNavigator3D © is geared towards creators that need a robust but still configurable path navigation capability in 3D space. These components rely only to Unity's core components and do not require any extra libraries (other than delivered with PathNavigator3D). You have multiple options to optimize the quality or performance of the path resolving. Path solving calculation is spread over multiple frames to prevent over usage of the resources.

Few highlights:

- Path LoS (Line of Sight) optimization.
- Bi-layered navigation grid for low and high LOD (Level of Detail).
- Hybrid grid path finding utilizing different LOD grids to find the path efficiently.
- Path calculation automatic global load balancing based on the game frame rate (CPU frame time) to prevent throttling.
- Centralized control API to manage all the path solver instances across scene.
- Path smoothing to get Bezier squared path points.
- Surface aligned navigation grid for near ground navigation.
- Closest point path solving when target is unreachable.



Requirements

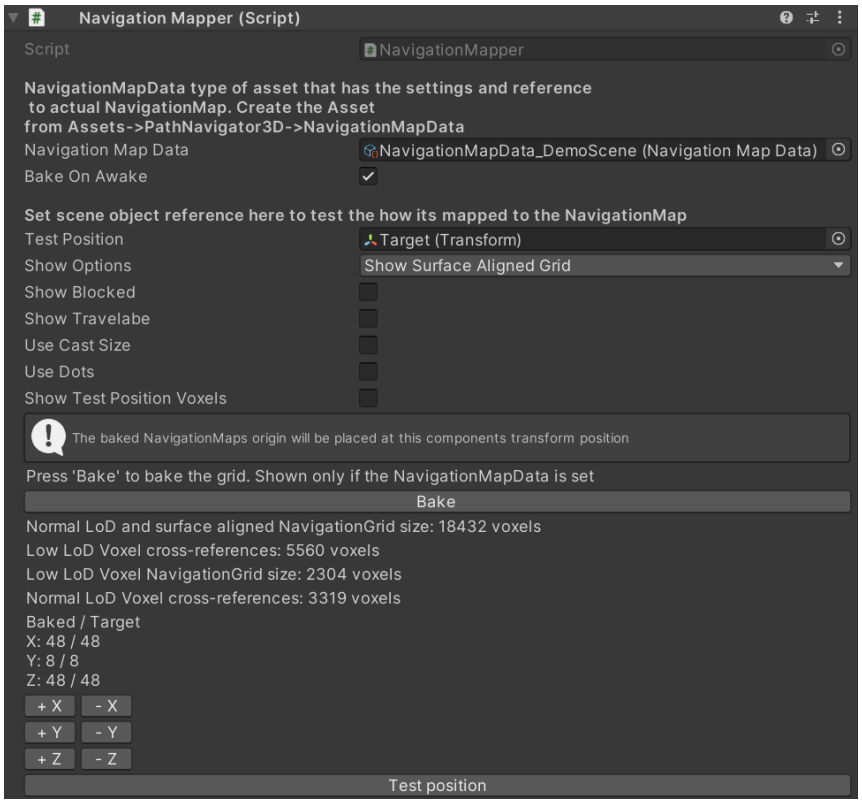
Unity versions starting from 2021 LTS are supported. API compatibility level .NET Standard 2.1

Tested with Unity LTS 2022.3.3f1

MonoBehaviour Components

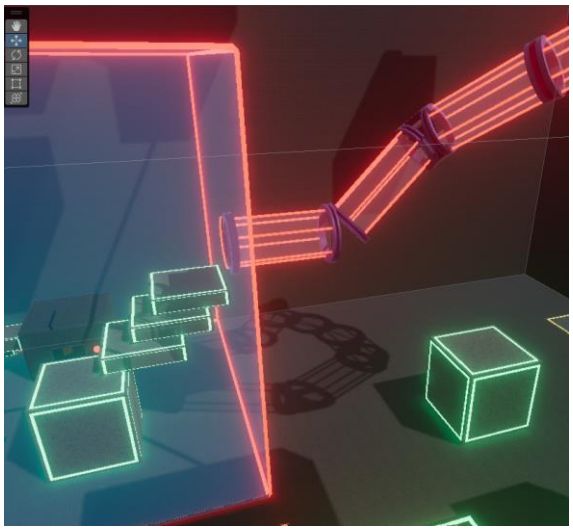
NavigationMapper

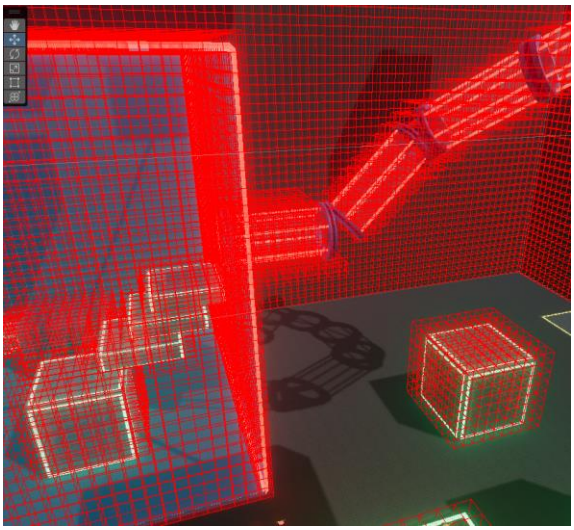
This component takes the NavigationMapData asset as a parameter and bakes the underlying NavigationMap to the scene based on the component's location in the scene. The location is static so if you move NavigationMapper the actual NavigationMap remains where it was baked and only re-baking will move the map.

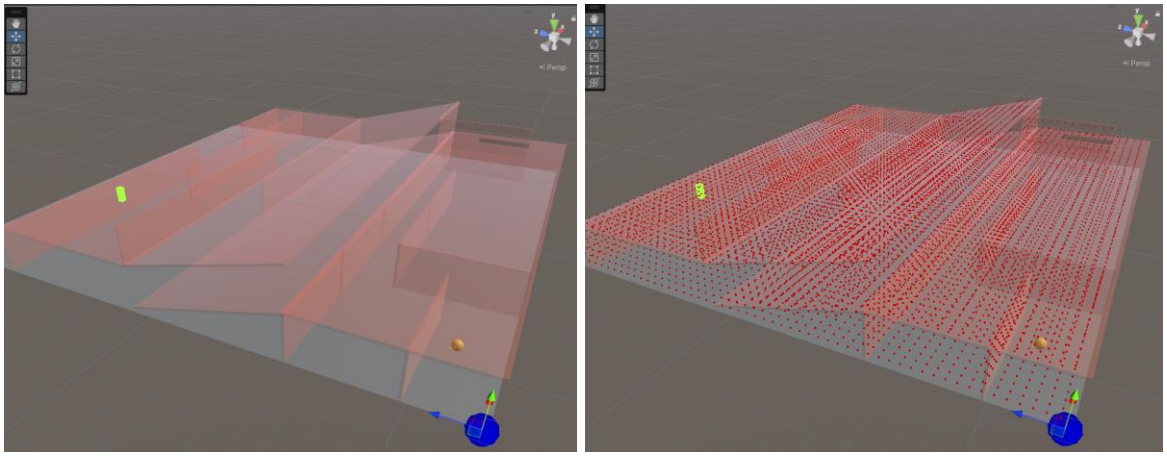


+- X, +-Y and +-Z buttons help to adjust the map to the needed size without going to the NavigationMapData asset to do it. After resizing press bake.

You can adjust how the NavigationMap is shown in the Scene view via gizmo setting.







*Picture: Normal scene view and **Show blocked voxels** checked view.*

NavigationAgentRB

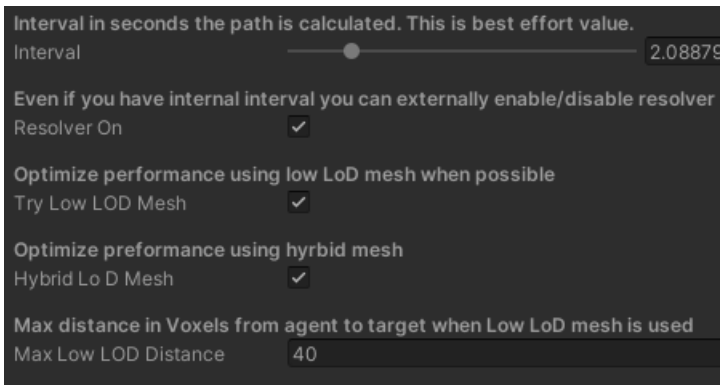
Example Agent for Rigidbody objects that utilizes directly the PathSolver to hunt down the set target.

It gives examples how to utilize the PathSolver and few different strategies how to follow the calculated path.

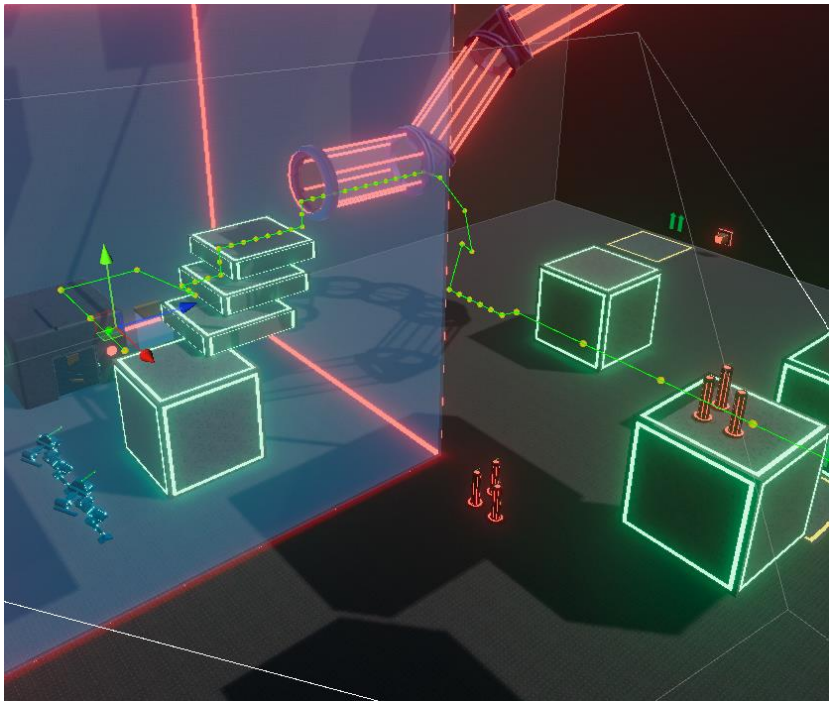
Navigator

Navigator is ready made component that can take any two points in the scene, assuming the points reside inside the baked NavigationMap, and calculate a path, using PathSolver, between the points.

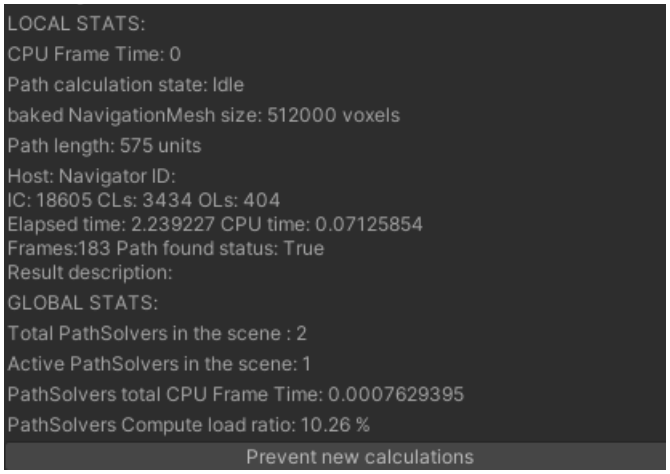
In addition to the PathSolver SettingsForEditor the Navigator has some own properties.



If PathSolver Debug is set to true, then Navigator shows the found path and the nodes in the Scene view and some of statistics are printed to Console.



Navigator shows PathSovler local and global statistics in the Inspector:



Notice that Global statistics show the **aggregated information about all the PathSolver instances** in the scene. See PathSolver Global Settings for further information.

Use of Navigator in your own implementation

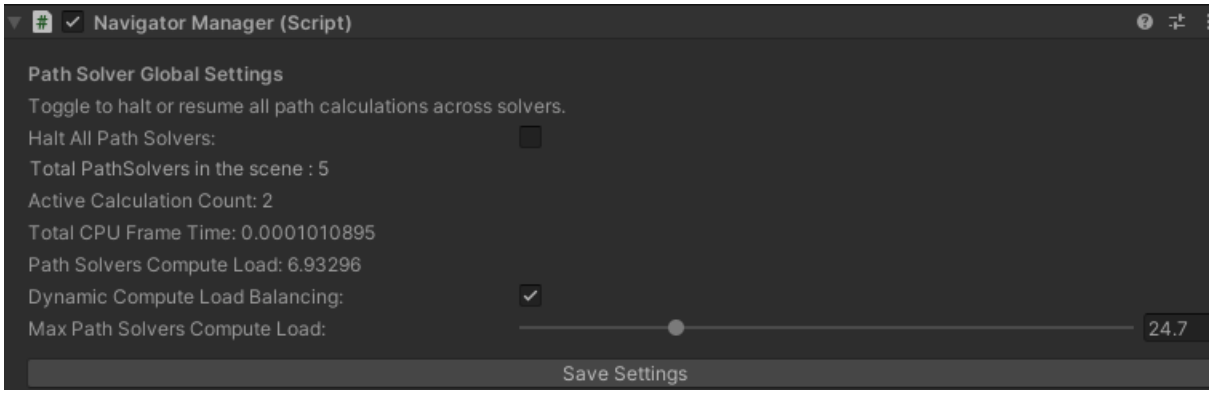
You can add a callback function to the Navigator via `AddPathReadyCallback(PathSolver.OnPathProcessed callback)` to get the calculated path and other statistics. Callback function example:

```
private void PathReadyCallback(List<Node> path, Result result)
{
    //… do what you want with the path.
}
```

You can also create variants of Navigator component to suit your own needs.

NavigatorManager

Singleton component where you can see and control the PathSolver's global setting.



NavigationObstacle

This component is used to make a GameObject that has collider as an obstacle.

When it is attached then all the dimensions of the colliders are calculated, and a volume (rectangle) is created where all the NavigationGrid's Voxel Blocking values are set to the *blockingValue* set in the inspector. If *Leave Trail* is set to False the old values are restored automatically when object is moving away from old position. Otherwise, the object will leave a trail of changed blocking values behind. The only way to reset the values is to re-bake the whole NavigationMap.

This is evaluated all the time the object is moving or rotating.

Notice that you can also carve a non-blocking area with this by setting the blockingValue to 0.

ScriptableObjects

NavigationMapData

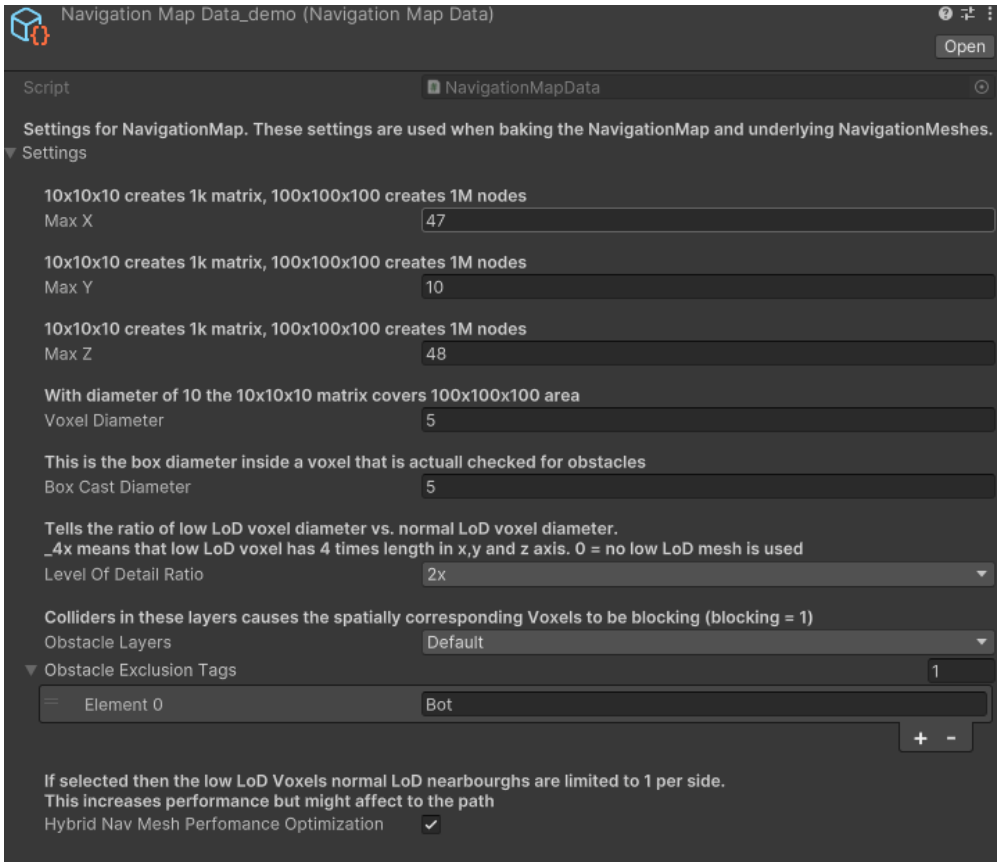
This component is a wrapper asset for and holds the basic settings of a NavigationMap like its dimensions, voxel size etc. When the NavigationMap is baked using the NavigationMapper then the component holds the reference to the baked NavigationMap

You can use NavigationMapData as an Inspector parameter to your own components. It is a mandatory parameter for NavigationMapper, Navigator and NavigationAgentRB.

PathSolver itself doesn't use NavigationMapData but underlying NavigationMap directly. However, the PathSolver's SettingsForEditor takes the NavigationMapData as a parameter and then .GetSettings() copies the NavigationMap reference from it to the PathSolver.Settings.

Create NavigationMapData asset selecting:

Asset →Create →PathNavigator3D → NavigationMapData



Core classes

PathSolver

PathSolver is the core component for path finding. PathSolver uses a variant of popular A* algorithm to find path between two points in the NavigationMap. There are multiple options to optimize the algorithm to fit your use cases.

PathSolver's path finding method, SolvePath and its sub-methods are always run internally as a Coroutine. That's why you need to provide a reference to a Monobehaviour instance so it can be used as a host to the coroutine process. PathSolver has an internal mechanism to prevent overlapping calculations in the same class instance. You can also check its idle state via `ResolverRunning` property.

SettingsForEditor

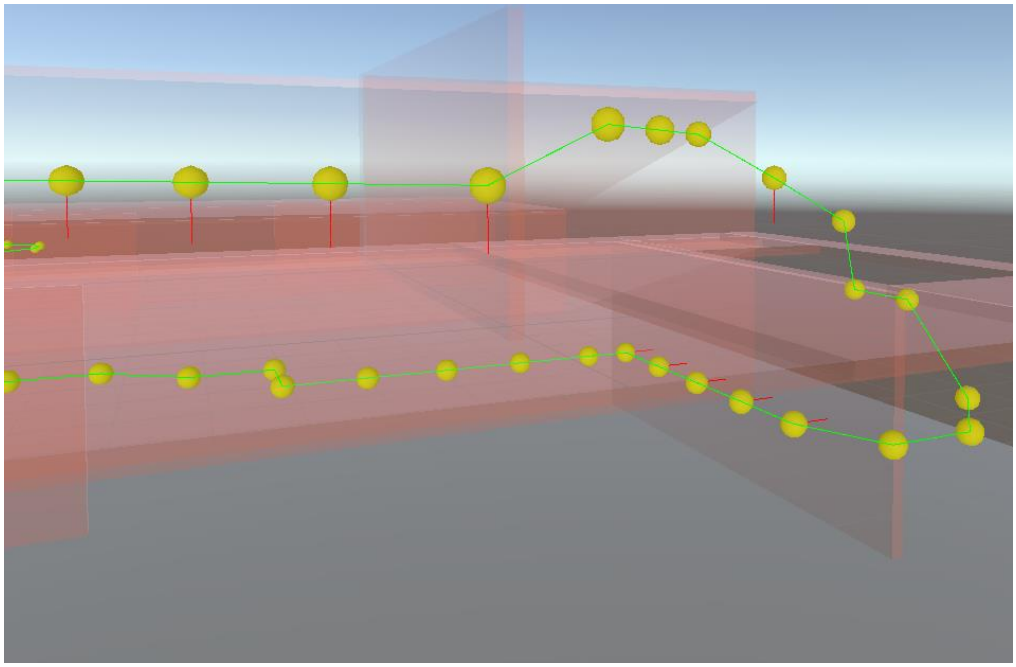
PathSolver has a serialized version of the *PathSolver.Settings* class called *PathSolver.SettingsForEditor*. This gives easy way to implement the PathSolver to your own components. Just add the SettingsForEditor to your MonoBehaviour component.

To convert the SettingsForEditor instance to actual Settings type object use *Settings settings = new Settings(settingsForEditor);*

Further notes on some of the setting

UseObstacleAvoidanceInPathPositions

If checked then the path positions will be offset a **ObstacleAvoidanceDistance** away from the obstacles. Obstacles are checked to 6 directions. If debug is on, then you can see the offset in the inspector in red lines connected to the recalculated path points. Other end of the line indicates the original path position before the offset. Notice that the offset is done to the actual NavigationGrid so you will only see the offset after the first path calculation. After that the positions are offset only if some other solver uses the same positions with different offset value (or no offset value at all) or the grid is re-baked.



Example usage

In your own component add:

```
public SettingsForEditor settingsForEditor;
private PathSolver pathSolver;

private void Start()
{
    //create a new instance of the solver
    pathSolver = new PathSolver();
    //get the actual Settings from Inspector data
    Settings settings = new Settings(settingsForEditor);
    //Calculate the path. Normally you might want this to be called periodically.
    pathSolver.SolvePath(this, PathReadyCallback, settings));
}
// Callback function that you pass to the pathSolver.SolvePath to get the path.
private void PathReadyCallback(List<Node> path, Result result)
{
    //… do what you want with the path.
}

//Always dispose the PathSolver instance when it's no longer needed.

private void OnDestroy()
{
    pathSolver.Dispose();
}
```

It is recommended to create only one instance of PathSolver per component. Anyway remember to dispose all the PathSolver instances in the OnDestroy() Unity method with `pathSovlerInstance.Dispose()`.

PathSolver Global Settings

You can get information and control centrally all the PathSovler instances. Following properties are global and affect all the PathSolver instances in the scene.

Call them for example `PathSovler.HaltSolvers=true;`

HaltSolvers

Type: bool

Default Value: false

Readonly: No

Description: A single point of control for all PathSolvers in the scene. Setting this to true will halt all the path calculations.

ActiveCalculationCount

Type: int

Readonly: Yes (computed property)

Description: Counts the number of Solvers where there is an active calculation running.

PahtSolversTotalCPUFrameTime

Type: float

Readonly: Yes (computed property)

Description: Represents the current total CPU frame time taken by all the PathSolvers.

LatestCPUFrameTime

Type: float

Readonly: Yes (computed property)

Description: Represents the latest CPU frame time.

PathSolversComputeLoad

Type: float

Readonly: Yes (computed property)

Description: Indicates the percentage of CPU frame time all the PathSolvers are currently taking of the total CPU frame time.

DynamicComputeLoadBalancing

Type: bool

Default Value: true

Readonly: No

Description: Determines if the system should automatically keep the PathSolvers total compute load under the MaxPathSolversComputeLoad.

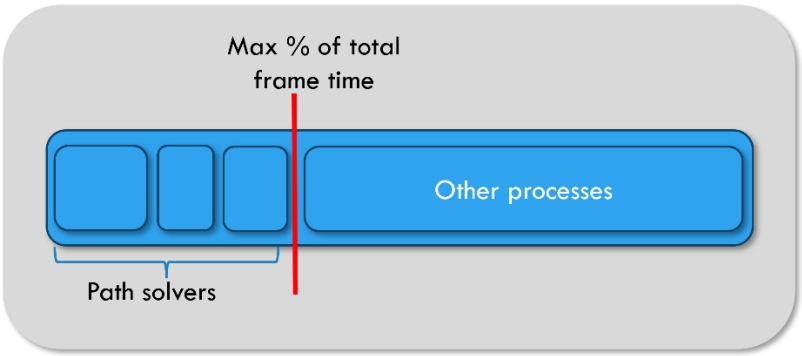
MaxPathSolversComputeLoad

Type: float

Default Value: 30f

Readonly: No

Description: Specifies the maximum percentage value the PathSolvers should take of the total CPU frame time. The default is set to 30%. The tolerance of this limiter value is +-5%. What is limited is the instance specific batch size of the calculation. You set the maximum batch size in the PathSolver Settings. If the total load of the PathSolver instances takes over the MaxPathSolversComputeLoad value of the CPU frame time, then the batch size is reduced on the fly by the needed amount. The reduced amount is relative to BatchSize value. So, for example if we have 100 and 1000 batch size instances and we go over the limiter and limiter decides to reduce 10% from both then the resulting batch sizes will be 90 and 900. If we then go under the threshold, then the limiter starts to increase the batch size until the instance specific limit is reached or we hit the MaxPathSolversComputeLoad value.



AvgPathSolversComputeLoad

Type: float

Readonly: Yes (private setter)

Description: Average compute load of 120 PathSolversComputeLoad requests. Zero loads are ignored.

Solvers

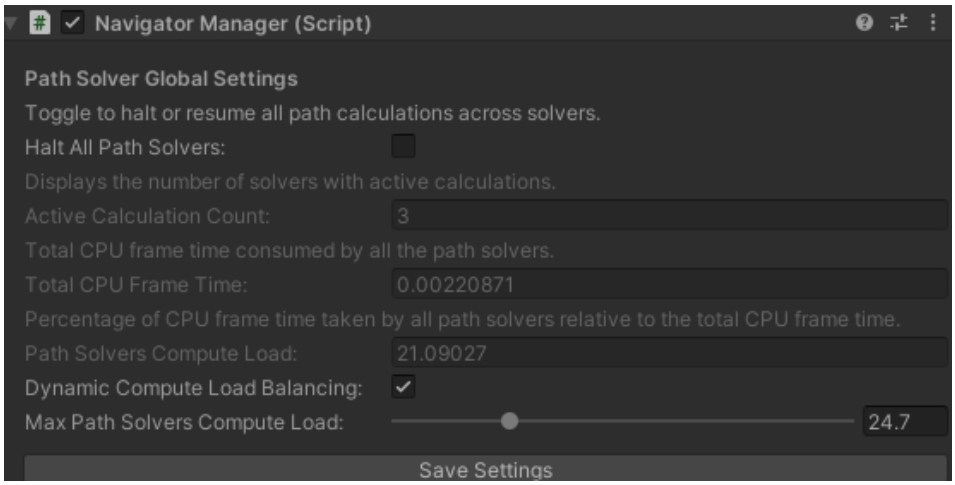
Type: List<PathSolver>

Default Value: new List<PathSolver>()

Readonly: Yes (private setter)

Description: Keeps track of the PathSolver instances.

There is also a ready-made component, NavigatorManager, where you can edit and see the global settings.



PathSolver.Result

Result is returned via `pathSolver.SolvePath` methods callback parameter.

IterationCount

Type: int

Readonly: Yes (public getter, internal setter)

Description: Represents the total number of path node calculation iterations performed.

ExecutionTimeTotal

Type: float

Readonly: Yes (public getter, internal setter)

Description: Represents the total time elapsed from the start to the end of the path calculation.

ExecutionTimeReal

Type: float

Readonly: Yes (public getter, internal setter)

Description: Represents the actual CPU time used for calculations across frames.

ClosedListSize

Type: int

Readonly: Yes (public getter, internal setter)

Description: Represents the size of the closed list at the end of the calculation.

OpenListSize

Type: int

Readonly: Yes (public getter, internal setter)

Description: Represents the size of the open list at the end of the calculation.

ResultCode

Type: Enum, ResultCode

Readonly: Yes

Description: Result code of the path calculation

PathFound

Type: bool

Readonly: Yes (public getter, internal setter)

Description: Indicates whether a path was found. true if a path was found, false otherwise.

ResultDescription

Type: string

Readonly: Yes (public getter, internal setter)

Description: Contains extra information or description regarding the result.

Id

Type: string

Readonly: Yes (public getter, internal setter)

Description: Represents a user-defined ID assigned to the calculation.

Frames

Type: int

Readonly: Yes (public getter, internal setter)

Description: Represents the number of frames that participated in the path calculation.

CalculatedGridType

Type: CalculatedGridType

Readonly: Yes (public getter, internal setter)

Description: Indicates the type of AgentGrid that was used in the path calculation.

LoSOptimizedNode

NOT IN USE

SmoothedPath

Type: List<Vector3>

Readonly: Yes (public getter, internal setter)

Description: Vector3 Bezeier smoothed vesion of the original Path. This is set if it was requested in the Settings. FloorPath setting affect this.

Path

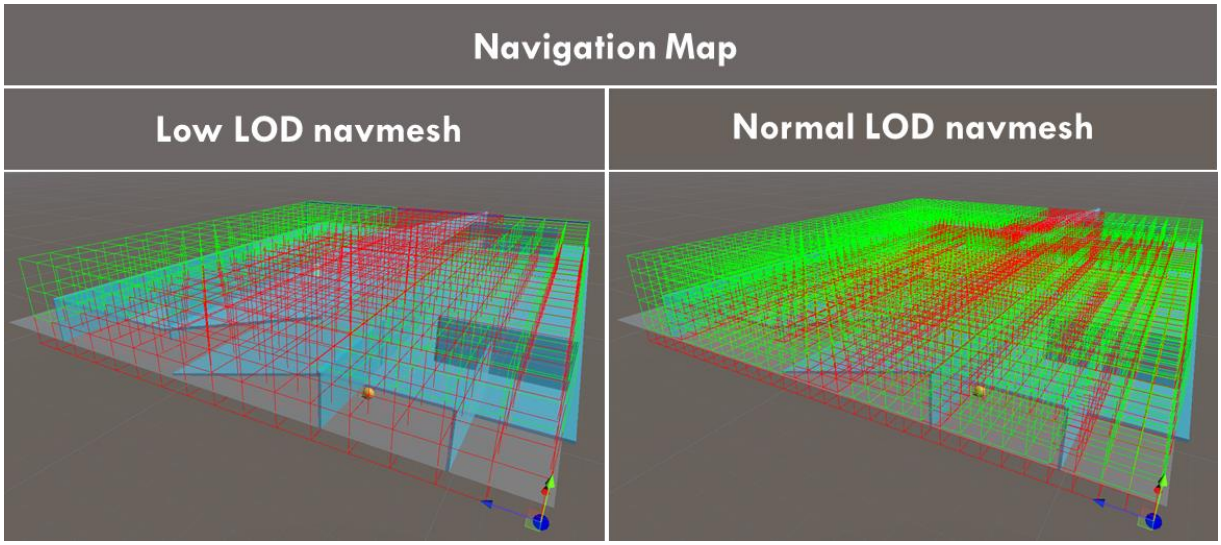
Type: List<Vector3>

Readonly: Yes

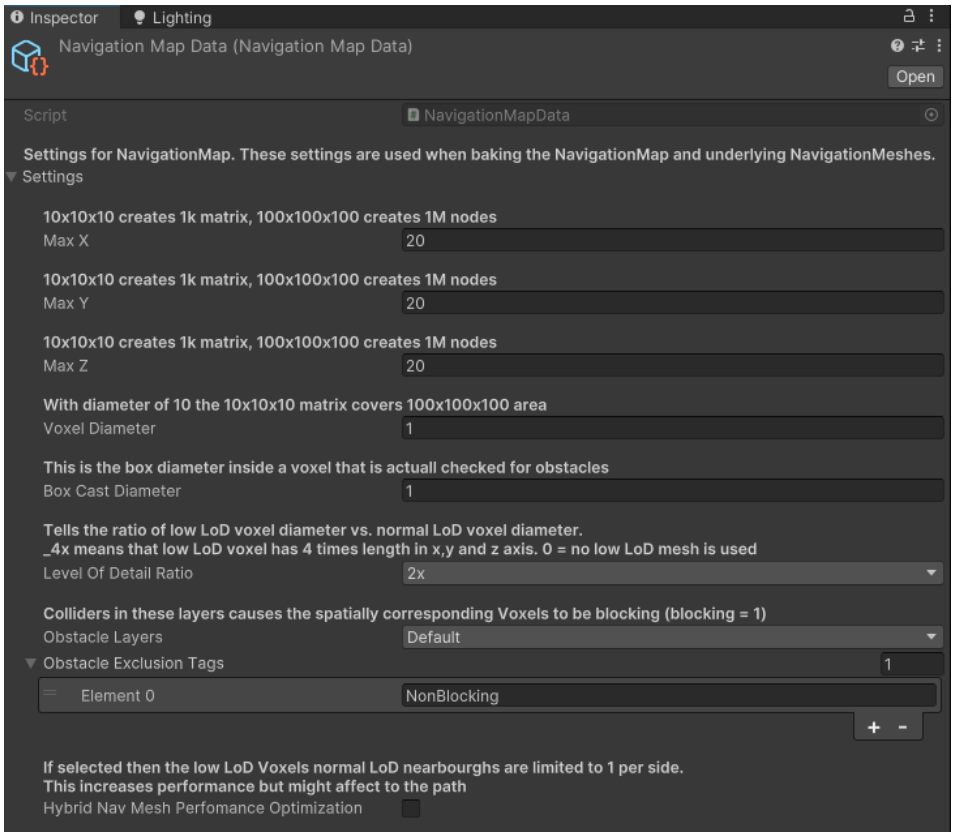
Description: Vector3 vesion of the original Path. FloorPath setting affect this.

NavigationMap

Class that holds all the information needed for PathSolver to process the path finding. NavigationMap has three NavigationGrids for low and normal level of detail matrices and also a surface aligned navgrid for close to ground navigation.


Navigation Map
Low LOD navmesh
Normal LOD navmesh

NavigationMap has a serialized settings (NavigationMap.Settings) for Inspector use:



NavigationMap is fully created only when it is baked.

Example code:

```
public NavigationMap.Settings settings;
public NavigationMap map;

private void Start()
{
	map = new NavigationMap();
	map.BakeGrid(this.settings, this.transform.position);
}
```

You can have multiple NavigationMaps in the scene, but PathSolver can only search a path within one map at a time.

If you don't need to code it yourself you can store the NavigationMap setting to an asset in NavigationMapData and then use NavigationMapper to bake and visualize the NavigationMap.

NavigationMap properties

These are readonly properties. Use the NavigationMap.Settings for setup.

LodRatio

Type: Enum

Readonly: Yes

Description: Enumeration representing the Level of Detail (LoD) ratio. Values such as _10x represent a multiplier (e.g., _10x means 0.1) used to determine the voxel diameter ratio between low and normal LoD.

NavGridLoDNormal

Type: NavigationGrid

Readonly: Yes

Description: Represents the navigation grid at normal resolution, based on the maxX, maxY, and maxZ settings.

NavGridLoDLow

Type: NavigationGrid

Readonly: Yes

Description: Represents the navigation grid at a lower level of detail, based on the normal grid resolution and LevelOfDetailRatio.

NavGridSurfaceAligned

Type: NavigationGrid

Readonly: Yes

Description: Variant of 3D navgrid that only have non-blocking voxels near the blocking surfaces. This is used to search a path when near ground path points are needed. You can also set the Settings.FloorPath=true to get the path points exactly attached to the ground surface.

MaxX, MaxY, MaxZ

Type: int

Readonly: Yes

Description: Represents the number of voxels in the X, Y, and Z dimensions respectively. Default value is 20 for each dimension.

WorldSize

Type: Vector3

Readonly: Yes

Description: Represents the world size of the map. Default is set to Vector3.zero.

MidPoint

Type: Vector3

Readonly: Yes

Description: Represents the midpoint of the map. After baking, it points to the actual world position of the midpoint.

VoxelDiameter

Type: float

Readonly: Yes

Description: Represents the diameter of one voxel. Default value is 2.

BoxCastDiameter

Type: float

Readonly: Yes

Description: Represents the physics box cast diameter inside a voxel. Default value is 2.

LevelOfDetailRatio

Type: LodRatio

Readonly: No

Description: Represents the LoD vs normal LoD voxel diameter value. Default is set to _4x. This means that in one Low LoD Voxel 4^3 = 43 Normal LoD Voxels fit in. _2x means 2^3 = 8 and so on..

ObstacleLayers

Type: LayerMask

Readonly: Yes

Description: Layers that contain colliders causing corresponding voxels to be blocking.

ObstacleExclusionTags

Type: List<string>

Readonly: Yes

Description: List of GameObject tags that are not baked as blocked, even if the collider is in the blocking physical layer.

UseObstacleExclusionTags

Type: bool

Readonly: Yes

Description: A flag indicating whether to use the ObstacleExclusionTags.

HybridNavGridPerfomanceOptimization

Type: bool

Readonly: Yes

Description: If true, the low LoD voxels' normal LoD neighbors are limited to 1 per side, increasing performance but possibly affecting the path.

LowLodGridMultiplier

Type: float

Readonly: Yes (getter only)

Description: Returns the actual Low LoD multiplier based on the LevelOfDetailRatio. Calculated based on the case values in the LevelOfDetailRatio enum.

NavigationGrid

Class that consists of a single map entity made of Voxels (cubes) that have specific dimensions and relative position. NavigationGrids are residing in the NavigationMap and when it is baked then the NavigationGrids will have the final scene position.

Useful method in NavigationGrid is `GetVoxelByWorldPosition` that returns a Voxel where the given world position is in.

You should use NavigationGrid trough the NavigationMap.

Voxel

Voxel is a position and volume unit inside a NavigationGrid. When the grid is baked the Voxels will also have information about if it's blocking or not. PathSolver calculates the optimal path using the Voxels.

Voxel Properties:

WorldPos

Type: Vector3

Readonly: No

Description: Represents the Unity 3D world position of the Voxel (midpoint). When baked, it also includes the reference position.

Blocking

Type: float

Readonly: No

Description: A value indicating the blocking status of the voxel. A value of 1 means it's blocking. Values between 0 and 0.999 are used to set movement cost. The default is 0 for unblocked objects.

VoxelPos

Type: Vector3Int

Readonly: No

Description: Represents the Voxel's position in the NavigationGrid's 3D voxel matrix. The voxel's position Voxel[x][y][z] corresponds to the Unity coordinate system directions.

LowLODRef

Type: Voxel

Readonly: No

Description: Reference to an unblocked low Level of Detail (LoD) grid Voxel, if any. This means that if this Voxel is in the normal LoD Grid, then it's inside this Low LoD Voxel.

NormalLODRef

Type: List<Voxel>

Readonly: No

Description: List of references to unblocked normal grid Voxels, if any. This means that if this Voxel is in the low LoD Grid, then it has these normal LoD Voxels as neighbors.

ParentNavigationGrid

Type: NavigationGrid

Readonly: No

Description: Provides access to the grid that this Voxel is part of.

Node

Node is an evaluated (potential) Path position. PathSolver processes Voxels and make Nodes out of them to add path finding algorithm specific parameters. PathSolver returns the found path in a list of Nodes. A Node has always a reference to the corresponding Voxel and via this link the actual scene position can be found.

For example if we have `path` that is type of List<Node> then to get the position of the first (that's the starting point of a path) node we can write

```
Vector3 firstPosition = path[0].Voxel.worldPos;
```

Node properties:

Cost

Type: float

Readonly: Yes (internal setter)

Description: Represents the total cost associated with the node.

G

Type: float

Readonly: Yes (internal setter)

Description: Represents the G cost component, which is the cost from the start to the current node.

H

Type: float

Readonly: Yes (internal setter)

Description: Represents the H cost component, which is the estimated cost from the current node to the target.

Parent

Type: Node

Readonly: Yes (internal setter)

Description: Reference to the parent node in the pathfinding algorithm. Notice that the parent is always the previous point in the path so if you iterate the path using parent relationship you need to start from the destination Node.

Voxel

Type: Voxel

Readonly: Yes (private setter)

Description: Reference to a navgrid voxel that this path node represents.

Notes on the performance optimization

Performance comparison

Below are some results from a performance test in the Demo scene.

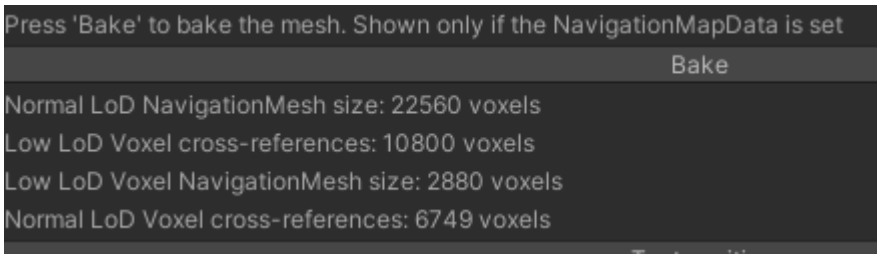
Performance in relation to base A* search algorithm (Normal grid + no options)	
Grid used and performance related settings	**X times more performant**
Hybrid + all off	1,85
Normal + reduce directions	1,98
Normal + performance over optimal path	4,51
Hybrid + Hybrid grid perf. opt.	6,75
Normal + performance over optimal path + reduce directions	19,56
Hybrid + performance over optimal path	34,17
Hybrid + Hybrid grid perf. opt. + performance over optimal path	79,29

Performance difference is calculated from the total CPU time the path calculation takes using the same computer, the same NavigationMap and the same start and destination positions.

Notice that these values are taken just from a specific path and can be different in other scenarios.

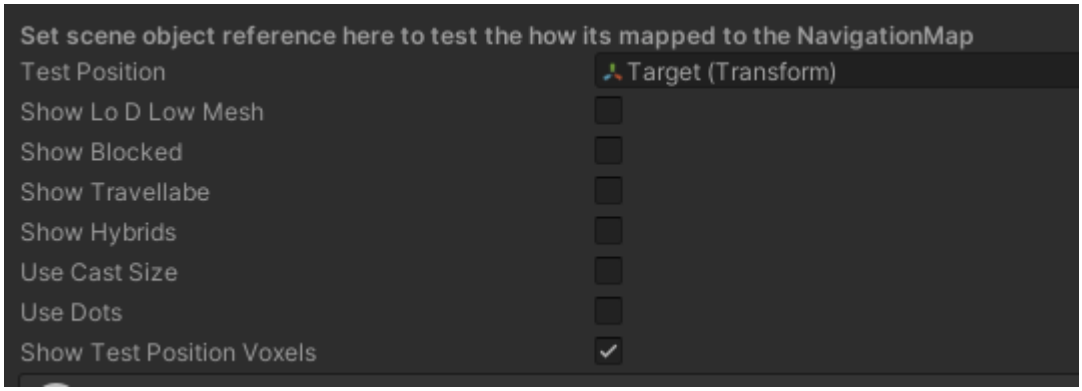
About Hybrid grid performance

The performance of Hybrid grid is strongly related to the LOD Ratio that was 2x in the tests. The larger LOD ratio (that means larger low LOD Voxels) the faster the Hybrid performance is. However, if LOD ratio is too large it might cause too many Low LOD voxels to be blocking that negates the performance advantage of the Hybrid grid. So, it's important to check the map baking results case by case and test different values.

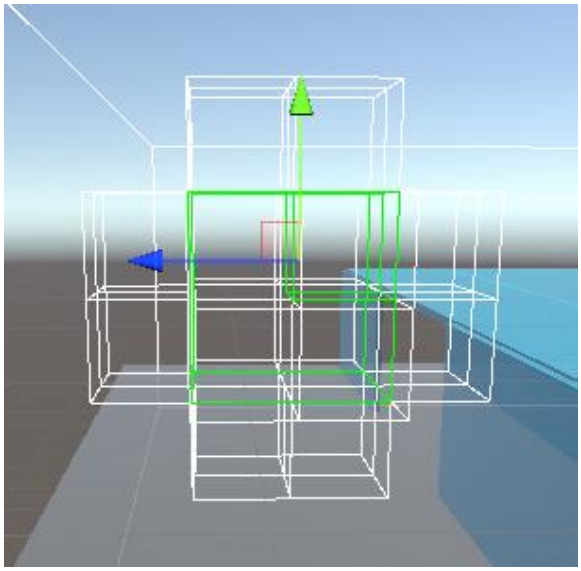


Above picture shows both grids Voxels amounts and the cross-reference amounts. These references are important when using Hybrid grid path solving. The PathSolver uses these references to jump between then grids. References are always made to the non-blocking voxels.

In the NavigationMapper you set a test object reference and place it in the NavigationMap in scene and select "*Show Test Position Voxels*" to see the corresponding Voxel and its references.



In the picture below you can see the big green box is the Low LOD Voxel and smaller green is a Normal LOD Voxel. The white boxes surrounding the Low LOD Voxel are the Normal LOD Voxel references it has. In the other way the Normal LOD Voxel has a reference to the Low LOD Voxel it is in.

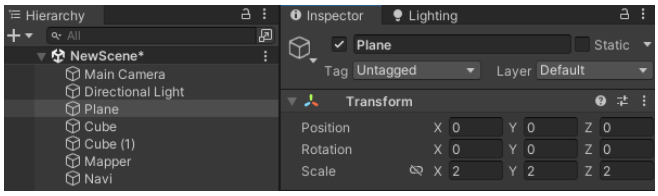


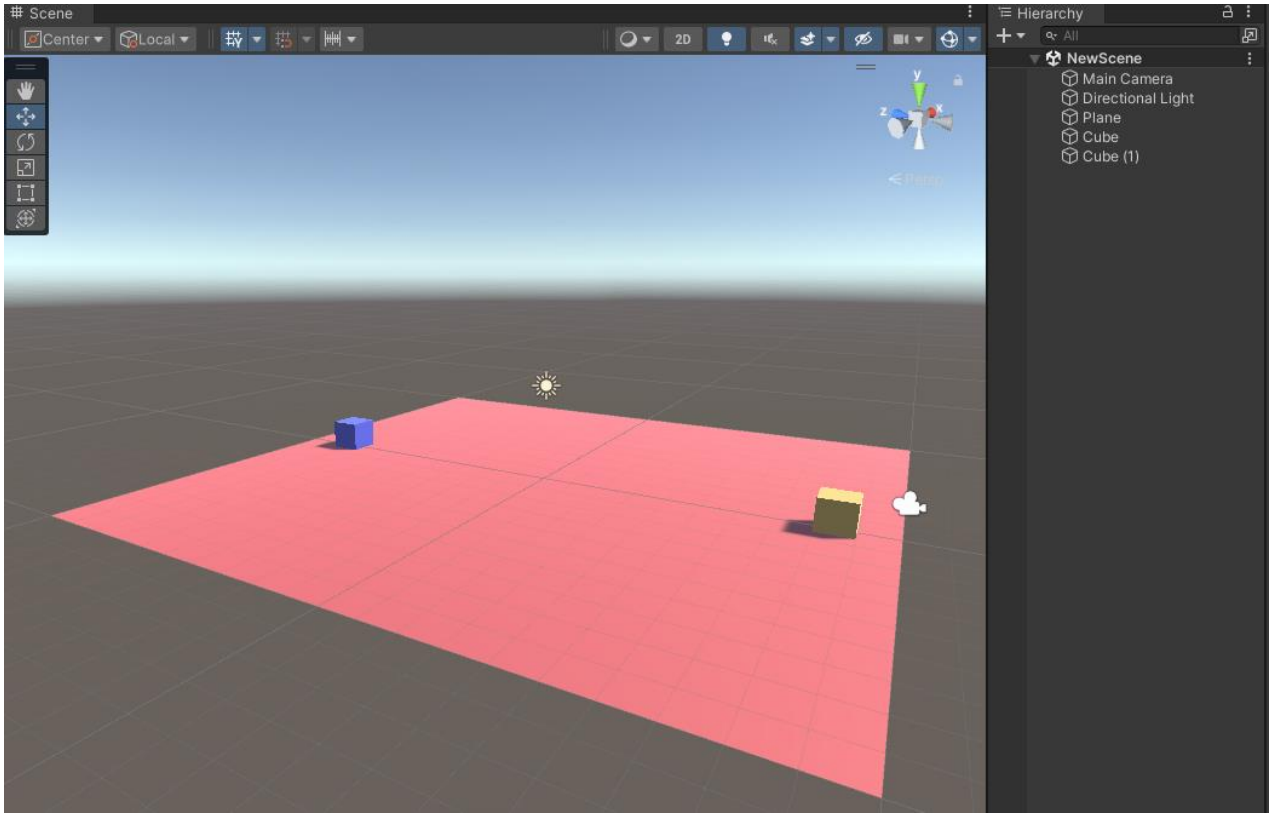
Example workflow

In a scenario you start with no PathNavigator Components in the scene. Target is to have two GameObjects in the scene and find a path from one to other.

Add objects to the scene

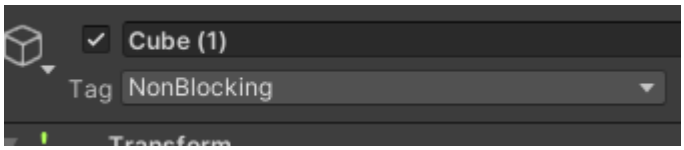
Let's create a plane and two cubes and color them (create material for them). Scale them to get some real distance between the cubes. Scale the Plane to 2,2,2.





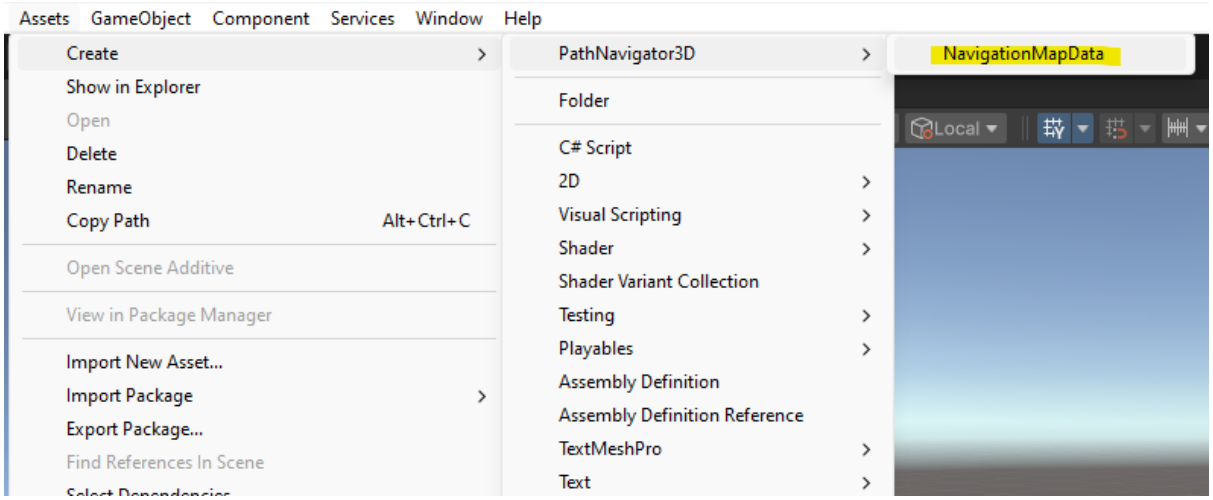
Notice all of them are in the Default physical layer.

Add a new Tag for the both cubes for example "NonBlocking":

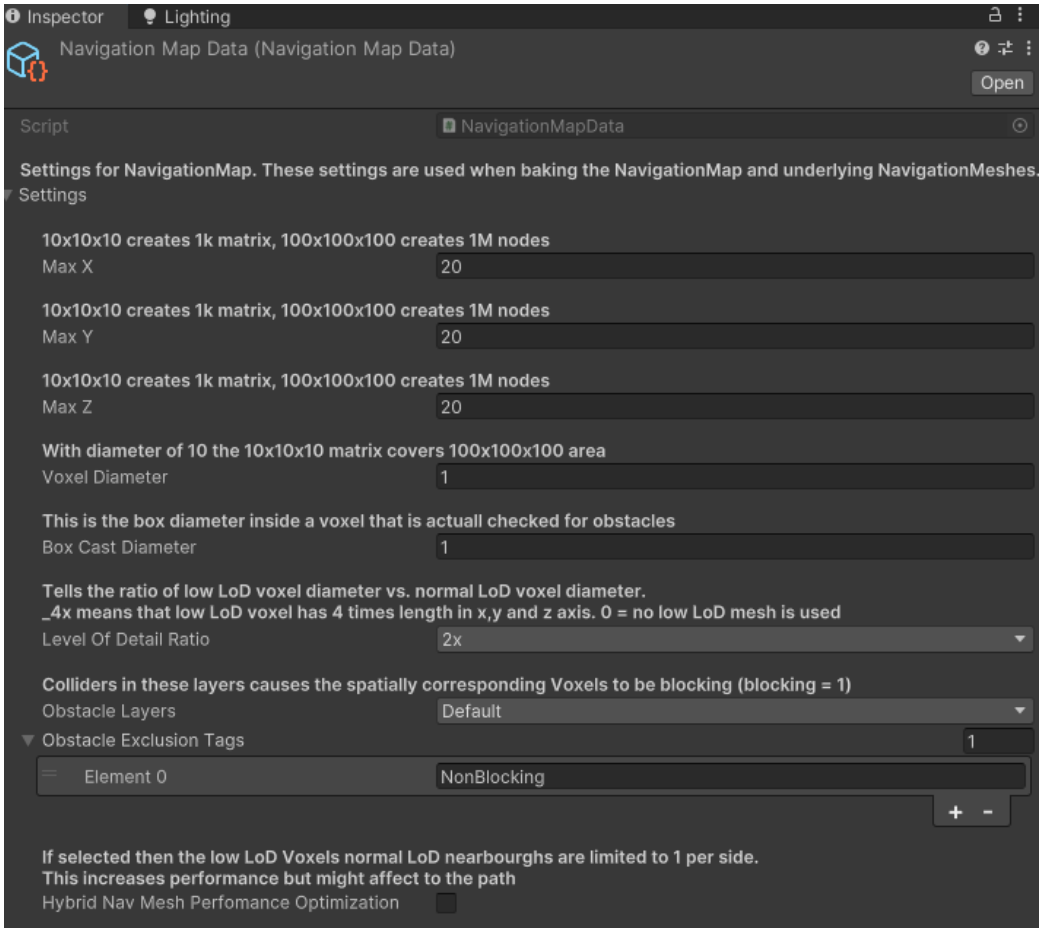


Make a NavigationMap

Add the new NavigationMapData asset. *Asset→Create→PathNavigator3D→ NavigationMapData*



Set asset's parameters as shown below:

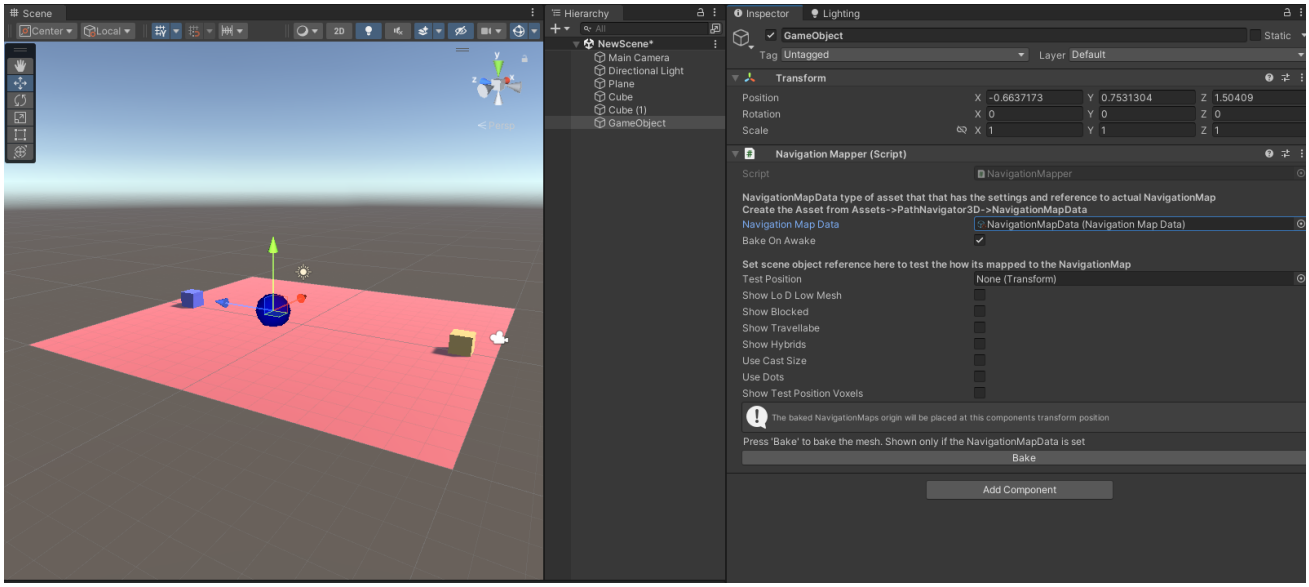


Notice! Make sure that the *Obstacle Exclusion Tag* is the same as the tags in the cubes.

Add NavigationMapper

Create an empty GameObject and add NavigationMapper component to it.

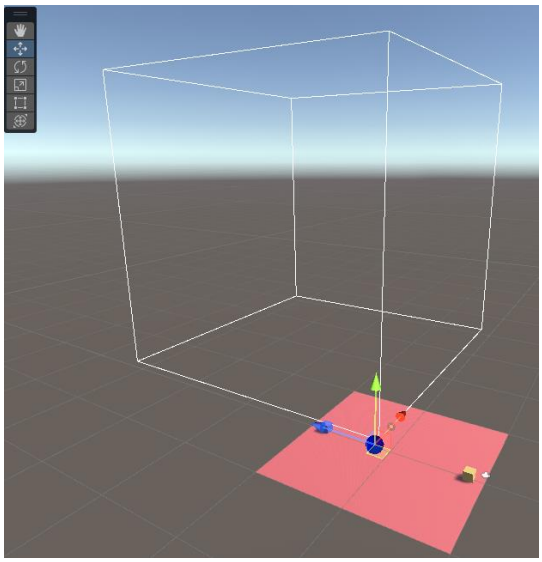
Add newly created NavigationMapData asset to the corresponding place in the component.



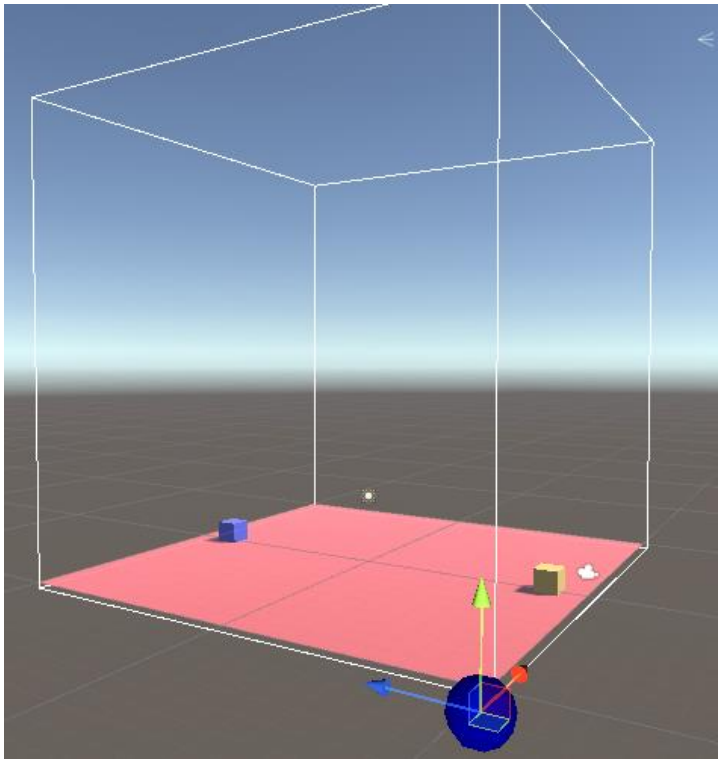
(Notice. The NavigationMapper layout has been updated and might differ slightly from the picture)

Press *Bake*

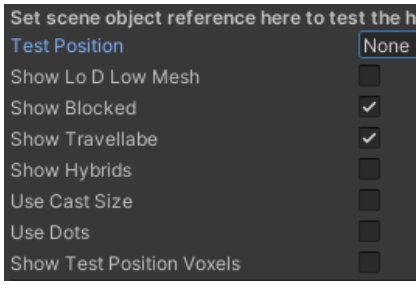
You can see in Scene view (if the Gizmos are enabled) a white wire box that represents the baked NavigationMap area.

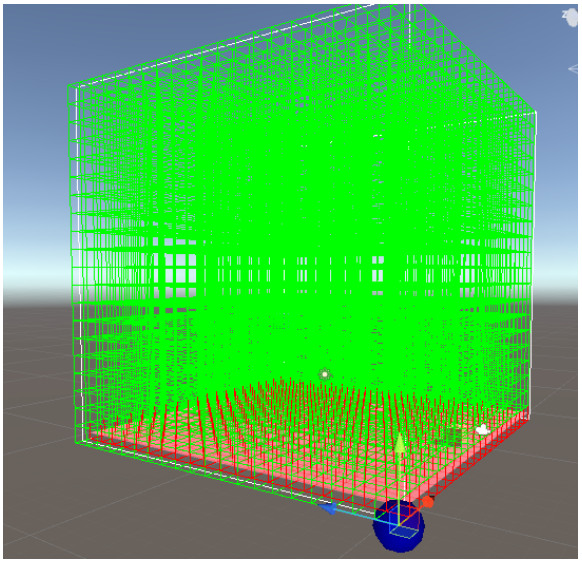


Now move NavigationMapper game object (the blue circle) to the corner of the plane and press *Bake* again: Now you should see situation like this:



Check the *Show Blocked* and Show *Travelable* to see the actual baked grid:



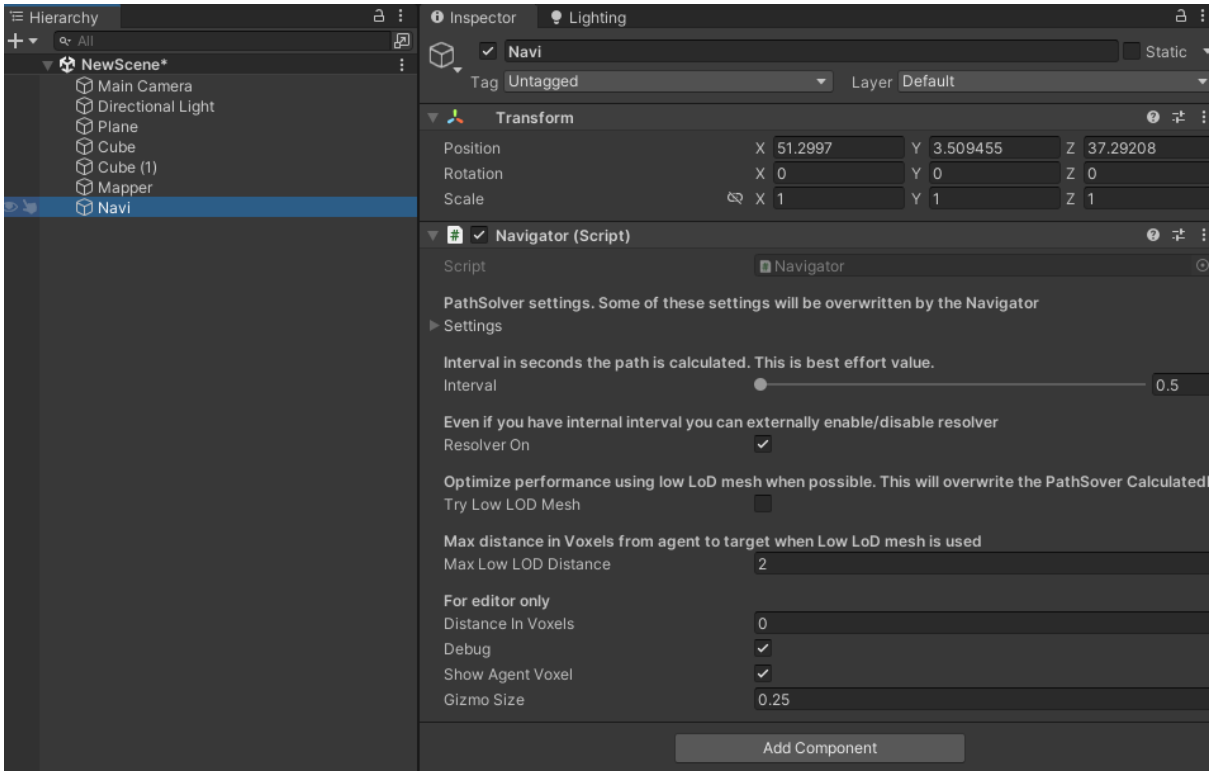


Green boxes are non-blocking Voxels and the red ones are the blocking.

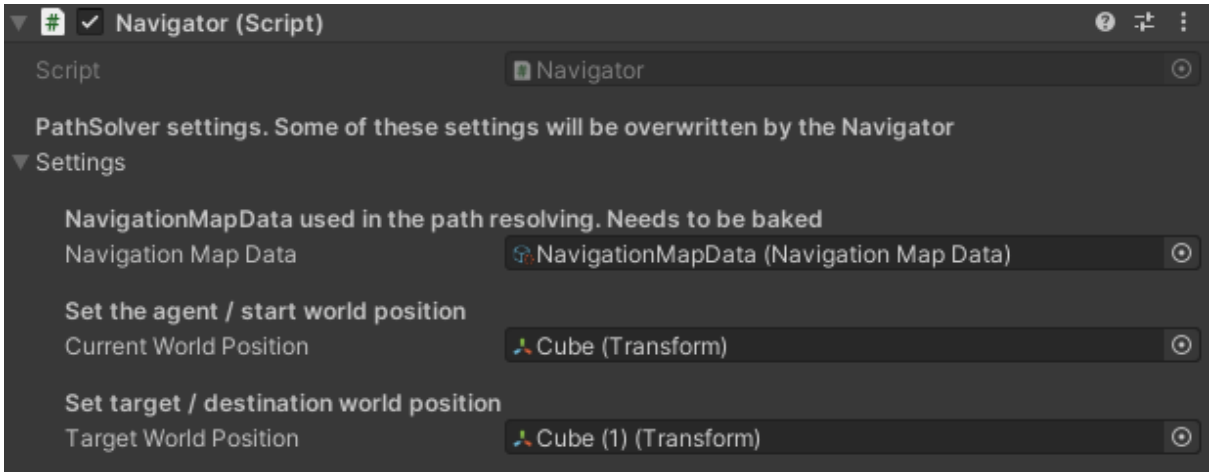
Let's also rename the GameObject the NavigationMapper is in as Mapper. This is just for clarity's sake.

Add Navigator Component

Make a new GameObject and add Navigator component to it. Rename the GameObject as Navi (or as you like). Set the Navigator setting as shown below.



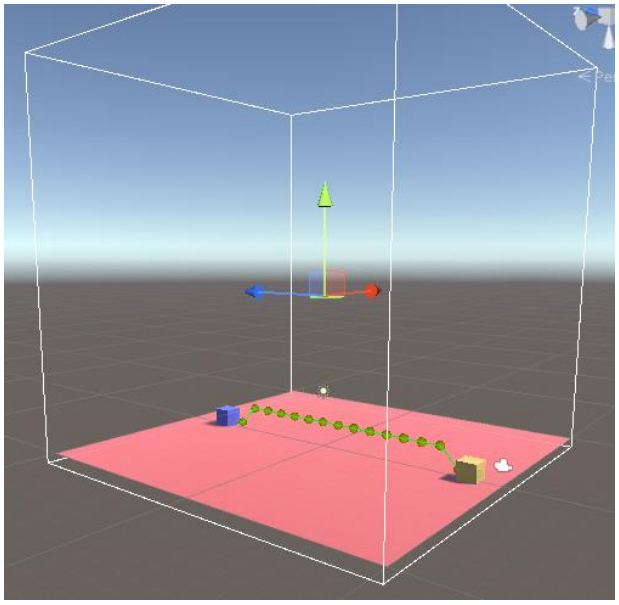
Open the Settings in the Navigator component. This is actually the PathSolver.SettingsForEditor. Set the settings as shown below.



Leave everything else as they are in default settings (you can try different configuration later to get to know what the best config for your use case is).

Find a path

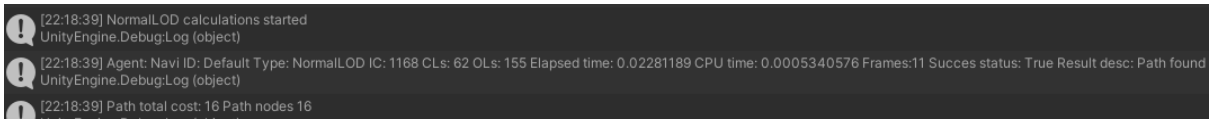
Press Play and check from the Scene view the path that is found (shown only if you have Debug checked in the Navigator and Gizmos enabled in the Scene view):



You can move around both the cubes to see how the path is changing.

You can also add obstacle between the objects to see how the path is found around them.

In the Console you can see extra information from the Path calculation. This is actually passed via PathSolever.Result data object to the Navigator and Navigator just prints them out (when Debug is on).



You can check how to get the actual path from the Navigator from here.