Get Started Guide

This is an excerpt from the full documentation. You can view the full documentation here (https://arongranberg.com/astar/documentation/stable). Most links on this page will just take you to the full documentation.

Get Started with the A* Pathfinding Project.

Pathfinding is all about finding the best path between point A and B. This is what the A* Pathfinding Project does, in this tutorial you will learn how to set up the project in a new scene and get a simple AI moving while avoiding obstacles.

This AI you will write will not be very advanced, it is just the minimal amount of code needed to get moving and following a path. If you want a more advanced AI, you can either extend the script you will write in this tutorial or use (or extend) the AIPath or RichAI (https://arongranberg.com/astar/documentation/stable/richai.html) scripts included in the package (see part 2 for basic usage of the RichAI component).

Installation

The first thing you need to do, if you haven't done so already, is to download the A* Pathfinding Project.

Please read the installation guide here: Installation Guide (https://arongranberg.com/astar/documentation/stable/installation.html).

The project can be downloaded from here (https://www.arongranberg.com/astar/download). You can either download the free version with some limited features (but still very powerful) or buy the pro version which has more cool stuff included.

If you want, you can explore the different example scenes in the project before you start with the next section. Depending on how you installed the package, you may need to import these separately in the Unity Package Manager.

> See
> Example Scenes (https://arongranberg.com/astar/documentation/stable/examplescenes.html)

Overview

There are several different parts of the package. Broadly, they can be grouped into:

- Movement scripts, which tell the agent how to move and where it should move (see Movement scripts (https://arongranberg.com/astar/documentation/stable/movementscripts.html)).
- Graphs, which describe where an agent can move (see Graph Types (https://arongranberg.com/astar/documentation/stable/graphtypes.html)).
- Temporary obstacles, which cut holes in the navmesh or update it in other ways (see Graph Updates (https://arongranberg.com/astar/documentation/stable/tilemaps.html#graph-updates)).
- Off-mesh links, which allow an agent to move or jump between otherwise disconnected parts of the navmesh (see NodeLink2 (https://arongranberg.com/astar/documentation/stable/nodelink2.html)).
- Path modifiers, which post-process paths to for example smooth them (see Using Modifiers (https://arongranberg.com/astar/documentation/stable/modifiers2.html)).

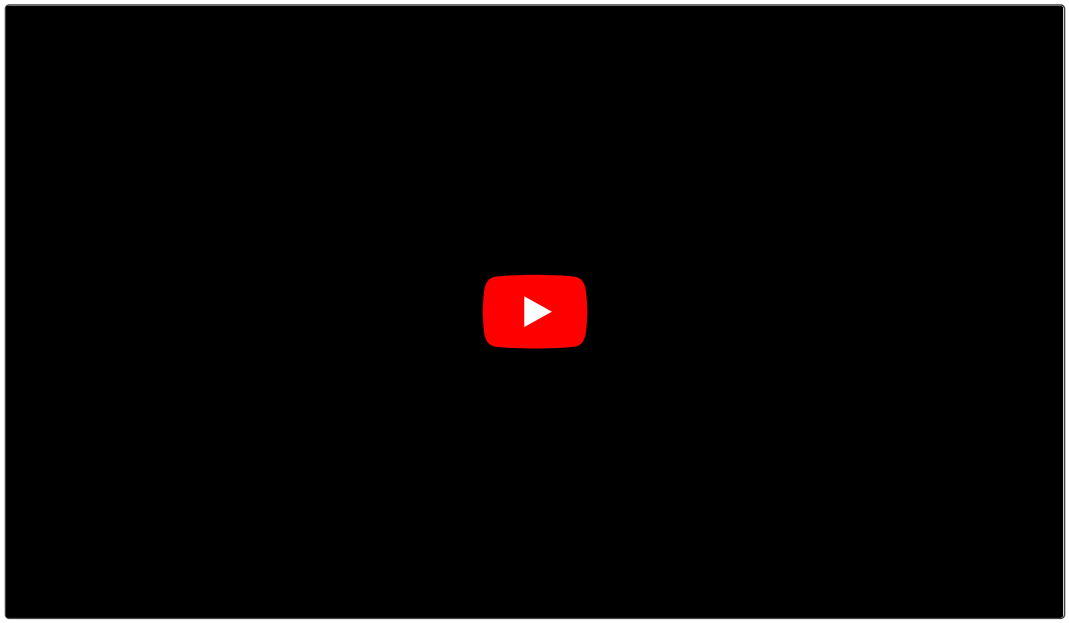
What you will primarily interact with is the movement script and the Seeker (https://arongranberg.com/astar/documentation/stable/seeker.html) component. Both need to be attached to an agent that needs to move. The movement script controls how the agent should move, its velocity, rotation, and so on, as well as what the current destination of the agent is and when it should recalculate its path. The Seeker (https://arongranberg.com/astar/documentation/stable/seeker.html) component is controlled by the movement script. The movement script tells it to calculate a path, and the Seeker will chug along and later (possibly in a later frame) return the result to the movement script.

The AstarPath (https://arongranberg.com/astar/documentation/stable/astarpath.html) component holds all the graph data in a scene. It follows the singleton pattern (https://en.wikipedia.org/wiki/Singleton_pattern), so there should only be one such component in a scene. It may contain one or many graphs, of the same or different types. Each graph, in turn, contains and manages all its nodes (of which there may be many, sometimes up to millions).

There are a number of included movement scripts in the package (e.g AIPath (https://arongranberg.com/astar/documentation/stable/aipath.html), RichAI (https://arongranberg.com/astar/documentation/stable/richai.html), AILerp (https://arongranberg.com/astar/documentation/stable/ailerp.html), FollowerEntity (https://arongranberg.com/astar/documentation/stable/followerentity.html)). You may use one of the included ones, or you can write your own (see Writing a movement script (https://arongranberg.com/astar/documentation/stable/custom_movement_script.html)). You can find a comparison of the built-in movement scripts here: Movement scripts (https://arongranberg.com/astar/documentation/stable/movementscripts.html).

Video Tutorial

If you prefer a video tutorial instead of a text tutorial. Here is a video for you. The video tutorial takes a more high-level approach, and you will learn how to use the built-in movement scripts instead of writing a custom one. Since the video and text tutorials cover slightly different ground, it is not a bad idea to take a look at both.



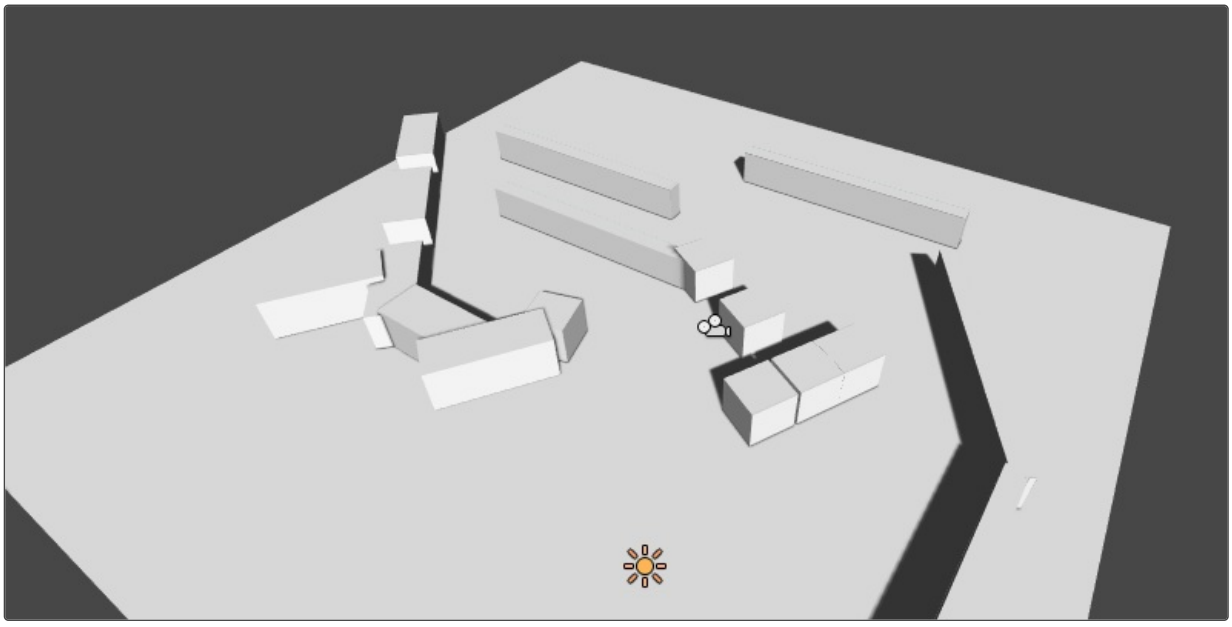
You can also take a look at the excellent tutorial by Gabriel Williams (Unity Cookie) in part 8 of the series on making a Tower Defence game: https://www.youtube.com/watch?feature=player_embedded&v=PUJSvd53v4k (https://www.youtube.com/watch?feature=player_embedded&v=PUJSvd53v4k) The video covers most things which will be discussed in the text tutorial.

New Scene

Create a new scene, name it "PathfindingTest". Now let's create something which an AI could walk on and something for it to avoid: add a plane to the scene, place it in the scene origin (0,0,0) and scale it to 10,10,10.

Create a new layer (Edit → Project Settings → Tags) named "Ground" and place the plane in that layer. Now create some cubes of different scales and place them on the plane, these will be obstacles which the AI should avoid. Place them in a new layer named "Obstacles".

Your scene should now look something like this:



Adding A*

Now we have ground for an AI to stand on and obstacles for it to avoid. So now we are going to add the A* Pathfinding System to the scene to enable Pathfinding.

Create a new GameObject, name it "A*", then add the "AstarPath" component to it (Menu bar → Components → Pathfinding → AstarPath (https://arongranberg.com/astar/documentation/stable/astarpath.html)).

The AstarPath (https://arongranberg.com/astar/documentation/stable/astarpath.html) inspector is divided into several parts. The two most important are the Graphs area and the Scan button at the bottom. The Graphs area holds all the graphs in your scene, you may have up to 256, but usually 1 or 2 will be sufficient. A single graph is usually preferred for simplicity.

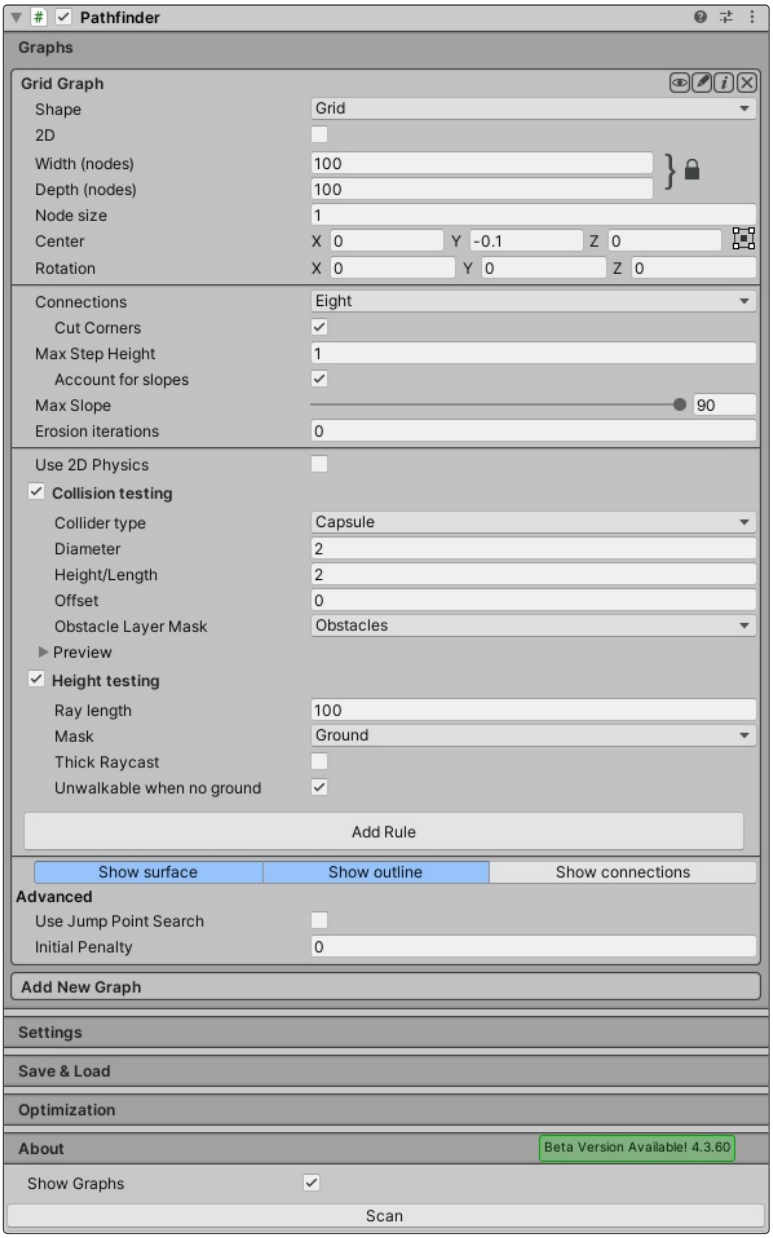
If you open the Graphs area by clicking on it, you will see a list of graphs which you can add. For this tutorial, we will create a GridGraph (https://arongranberg.com/astar/documentation/stable/gridgraph.html), which generates nodes in a grid pattern.

> See
>
> You can read more about the different graph types in Graph Types (https://arongranberg.com/astar/documentation/stable/graphtypes.html).

After you have added the grid graph, click its label to bring up the graph settings.

At the bottom of the inspector, you will find a button called "Scan". This is used to calculate the graph based on its settings and the world. After you change any settings, you will have to scan the graph to see the changes. There is a handy shortcut (https://arongranberg.com/astar/documentation/stable/shortcuts.html) for this: Cmd+Alt+S (mac) or Ctrl+Alt+S (windows).

All graphs are scanned by default when the game starts (unless the startup is cached, more about that in another part (https://arongranberg.com/astar/documentation/stable/saveloadgraphs.html)).



As the name implies, the GridGraph will generate a grid of nodes with the size width*depth. A grid can be positioned anywhere in the scene, and you can rotate it any way you want.

The Node Size variable determines how large a square/node in the grid is, for this tutorial you can leave it at 1, so the nodes will be spaced 1 unit apart.

The position needs to be changed, though. Switch to bottom-left in the small selector to the right of the position field (currently named "Center"), then enter (-50,-0.1,-50). The -0.1 is to avoid floating point errors, in our scene the ground is at Y=0, if the graph was to have position Y=0 too, we might get annoying floating point errors when casting rays against it for example (like the height check does).

To make the grid fit our scene, we need to change the width and depth variables, set both to 100 in this case. You can see that the grid is correctly positioned by the white bounding rectangle in the scene view, which should now be enclosing the plane exactly.

Height Testing

In order to place the nodes at their correct height, the A* system fires off a bunch of rays against the scene to see where they hit. That's the Height Testing settings. A ray is fired from [Ray Length] units above the grid downwards, a node is placed where it hits. If it doesn't hit anything, it is either made unwalkable if the Unwalkable When No Ground variable is toggled or the node is placed at Y=0 relative to the grid if it is set to false.

To make sure our height testing hits the correct things, we need to change the mask used. Currently it includes everything, but that would include our obstacles as well, and we don't want that. So set the Mask to only include the "Ground" layer which we created earlier.

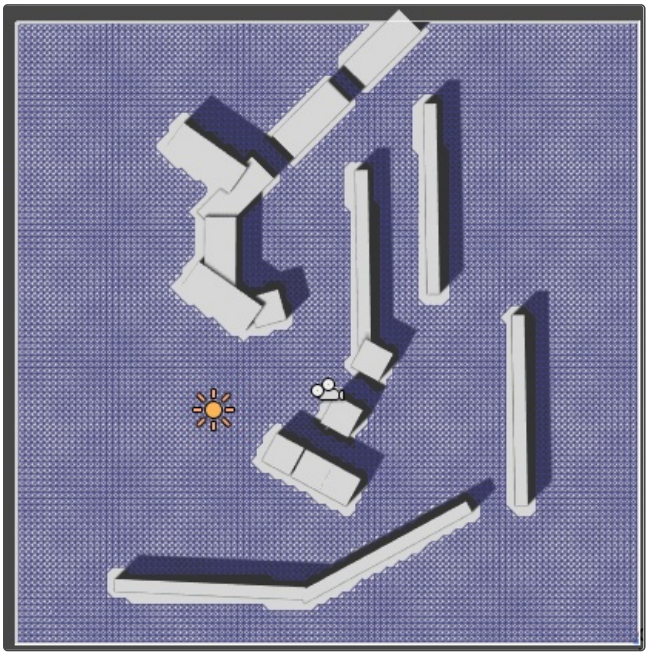
Collision Testing

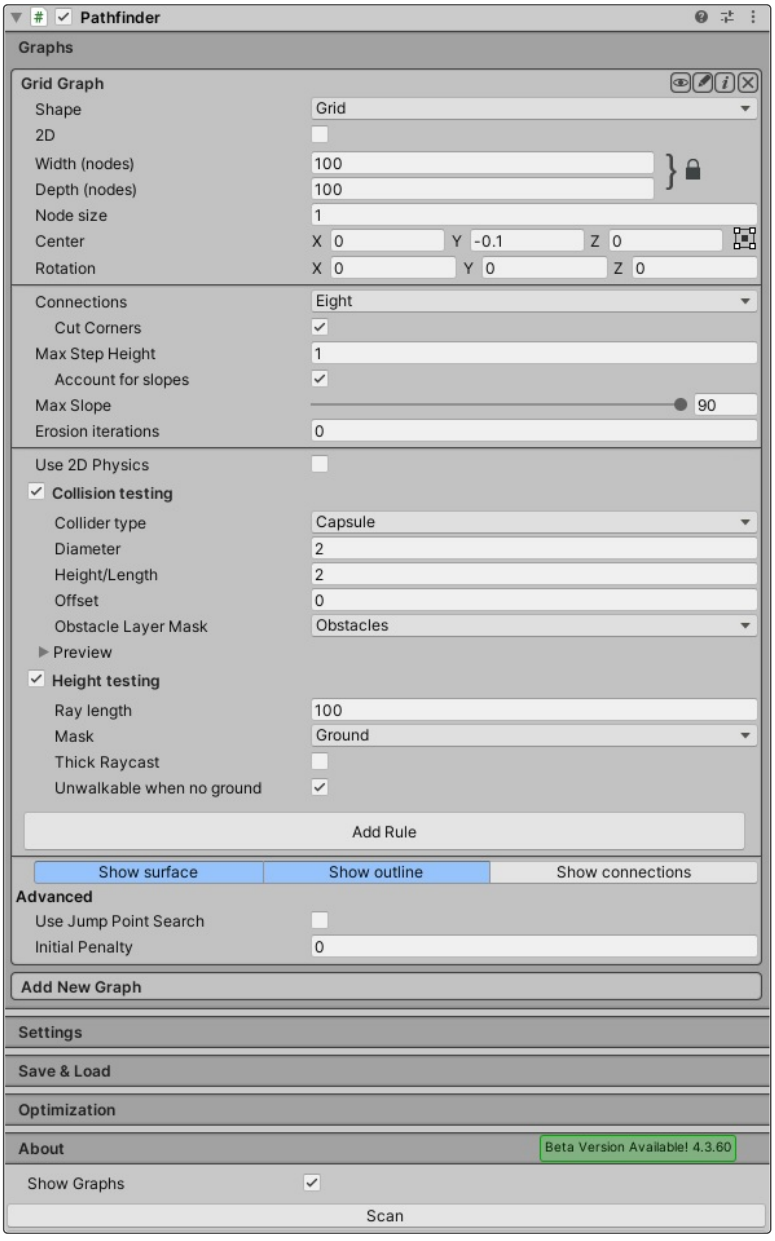
When a node has been placed, it is checked for walkability, this can be done with a Sphere, Capsule or a Ray. Usually a capsule is used with the same diameter and height as the AI character which is going to be walking around in the world, preferably with some margin though.

Our AI will have the standard diameter and height of 1 and 2 world units respectively, but we will set the diameter and height for the collision testing to 2 and 2 to get some margin.

Next, to make the system aware of the obstacles we placed, we need to change the mask for the Collision Testing, this time set it to contain only the "Obstacles" layer as we wouldn't want our ground to be treated as an obstacle.

Now that everything is set up correctly you can press the Scan button. Wait a fraction of a second and you've got a generated grid! (if you have done everything correctly, that is. Compare your settings to the image below and make sure that Show Graphs is true).





Adding the AI

What is a pathfinding test without some moving stuff? Not fun at all, so let's add an AI to play around with.

Create a capsule and add the Character Controller component to it, also place it somewhere visible on the plane.

Add the Seeker component to the AI, this script is a helper script for calling requesting paths from other scripts, it can also handle path modifiers which can e.g. smooth the path or simplify it using raycasts.

There are two alternatives now. You can either write your own movement script or you can use one of the built-in movement scripts. The included scripts are much more advanced than what you write in the tutorial linked below, so for most cases I recommend using them. However, I still recommend following the tutorial for writing a custom movement script even if you end up using one of the built-in ones in your game because it makes it easier to understand how the system works under the hood.

Check out this subpage for the tutorial: Writing a movement script (https://arongranberg.com/astar/documentation/stable/custom_movement_script.html)

The included scripts are called AIPath (https://arongranberg.com/astar/documentation/stable/aipath.html), RichAI (https://arongranberg.com/astar/documentation/stable/richai.html) and AILerp (https://arongranberg.com/astar/documentation/stable/ailerp.html). The AIPath and AILerp scripts can be used on any graph, while RichAI is primarily for navmesh based graphs. While the AIPath and RichAI scripts follow the path loosely, the AILerp script uses interpolation to move along the path very precisely, but perhaps not in the most realistic way. Which one you use depends on your game.

> See
>
> For more information about the included movement scripts, take a look at Movement scripts (https://arongranberg.com/astar/documentation/stable/movementscripts.html). You can also see how they are used in the included example scenes.

For this tutorial, you can attach the AIPath (https://arongranberg.com/astar/documentation/stable/aipath.html) component to the AI. Also create a new GameObject named "Target" and position it where you want the AI to move. Then attach the AIDestinationSetter (https://arongranberg.com/astar/documentation/stable/aidestinationsetter.html) component to the AI. This component is just a very simple helper script which will tell the AIPath script to move to a particular location. You will likely replace this script with your own game-specific script in the future. The AIDestinationSetter component has a single field called "target", assign the "Target" GameObject that you created earlier to this field.

If you press play now, the AI should move to the target. How the movement scripts work and how to configure them is explained in more detail in the video tutorial linked above. Take a look at that if something doesn't seem to work.

Smoothing

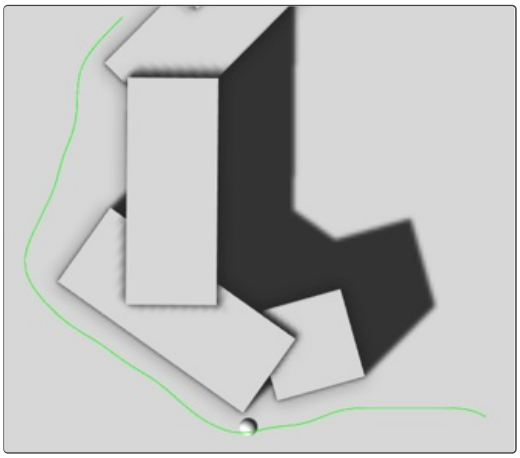
Now you have learned how to set up a simple grid graph and how to calculate paths Pathfinding, but surely there must be a way to get those paths to look a bit smoother?
Sure it is. Path smoothing and simplification scripts are called Path Modifiers and are scripts which can be added to the same GameObject as the Seeker.

The most straight forward one is the Simple Smooth modifier, which can be found at Menu bar → Components → Pathfinding → Modifiers → Simple Smooth. Add that to our AI.

What this modifier is going to do, is to subdivide the path a number of times until each segment becomes smaller than the Max Segment Length variable. Then it will smooth the path by moving the points closer to each other. The modifier has a number of settings, I won't go through all of them here. See the SimpleSmoothModifier (https://arongranberg.com/astar/documentation/stable/simplesmoothmodifier.html) documentation for more info about each variable. For this tutorial you can set Max Segment Length to, say 1. Iterations to 5 and Strength to 0.25. Experiment with it to get good values.

Now press play again, the path should look much smoother, just as we wanted.

> Warning
>
> Smoothers don't usually take world geometry or the graph into account, so be careful with applying too much smoothing since that could cause paths to pass through unwalkable areas.



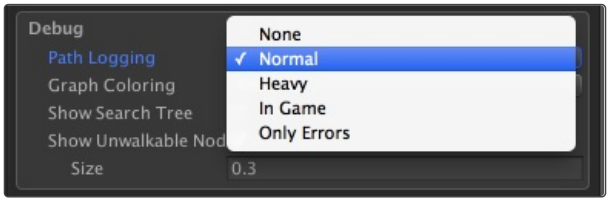
Another good modifier to use is the FunnelModifier (https://arongranberg.com/astar/documentation/stable/funnelmodifier.html) which will simplify the path a great deal. This modifier is almost always used when using navmesh/recast graphs.

Read more about modifiers on the page Using Modifiers (https://arongranberg.com/astar/documentation/stable/modifiers2.html).

Logging settings

Every time a path is calculated by the system, it can optionally be logged to the console. This can be a big help in understanding what the system is doing and also to spot performance issues. Logging is not free, however, so for release builds it is recommended that you disable it.

You can change the logging settings under the A* Inspector → Settings → Debug tab.



Use less debugging to improve performance (a bit) or just to get rid of the console spam. Use more debugging (heavy) if you want more information about what the pathfinding scripts are doing. The InGame option will display the latest path log using the in-game GUI.

Conclusion

That was the end of the Get Started tutorial part 1. I hope you learned something from it. From here on, you can explore the rest of the documentation or dig straight in to the project. If you want a little better AI, you can use the AIPath script, which is included in the project.

You can continue with the next part of the get started tutorial, where we will use navmesh graphs: Using navmeshes (https://arongranberg.com/astar/documentation/stable/getstarted2.html)

You can also take a look in the sidebar, there you will find a number of tutorials for how to use the package.

Good Luck!