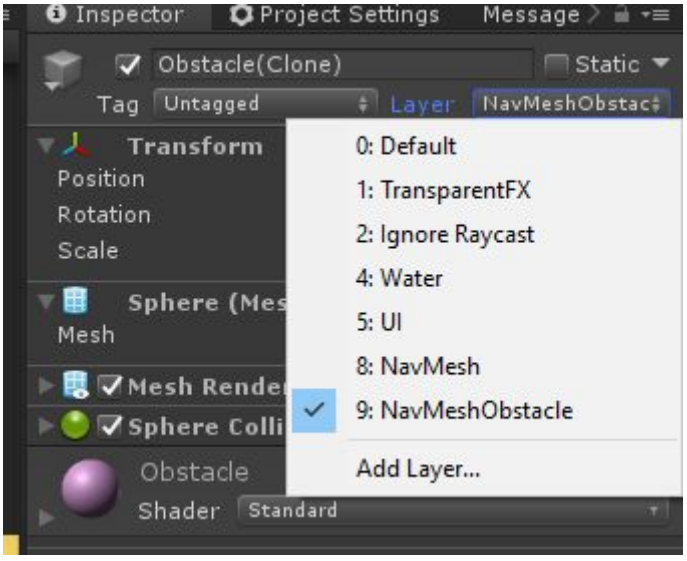
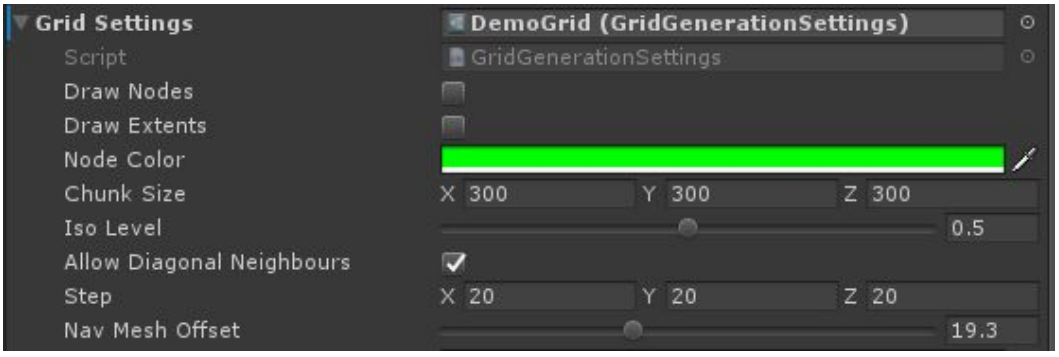
UNITY 3D PATHFINDING - HOW TO USE

How to use

- Add a NavMeshGenerator to your Scene
- Create a ‘NavMesh” Layer, it will be automatically assigned to the generated NavMesh
- Create a “NavMeshObstacle” Layer and assign it to any NavMesh relevant obstacles

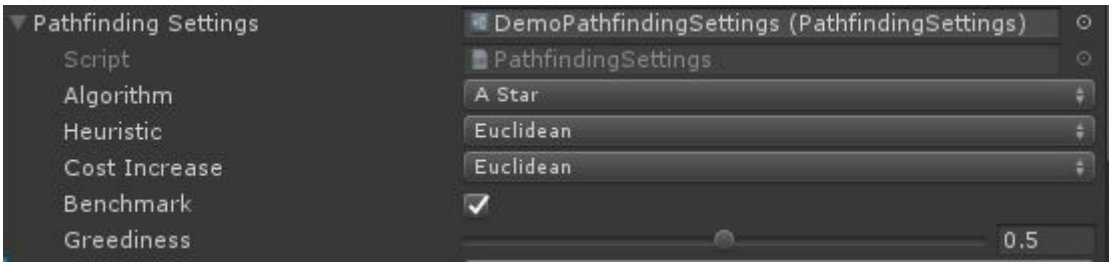


- Create GridGenerationSettings and PathfindingSettings via Assets > Create
- Assign these settings in the Generator

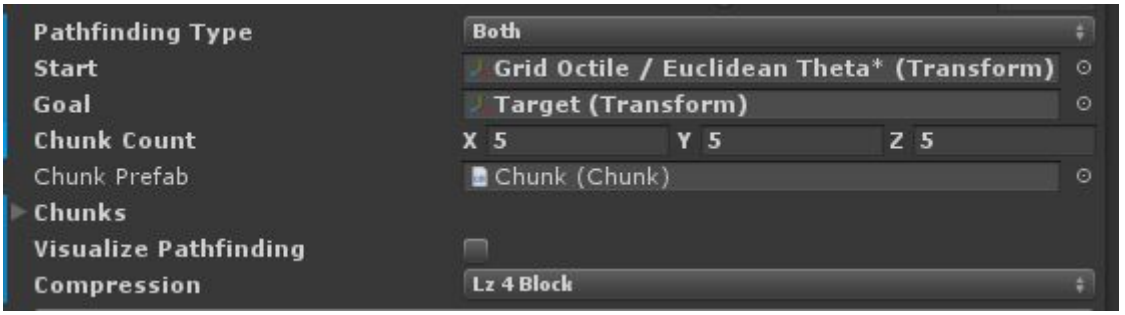


  - Draw Nodes: Whether to draw grid nodes as gizmos, can lag with big grids
  - Draw Extents: Whether to draw chunk bounds as wire cubes
  - Node Color: Color of walkable node gizmos
  - Chunk Size: Actual dimensions of chunks, try to keep them as large as possible (until the NavMesh vertex count exceeds 65535)
  - Iso Level: Threshold after which nodes are considered walkable (it usually isn’t necessary or profitable to change this, 0.5 is a good default)

- Allow Diagonal Neighbours: Whether to allow diagonal movement in the grid (has no impact on the NavMesh)
- Step: Distance between two grid nodes, try to keep this as big as possible to reduce storage and improve performance, too detailed grids require smaller chunks
- NavMesh Offset: Distance from the NavMesh to the actual obstacle, increase this for bigger Units

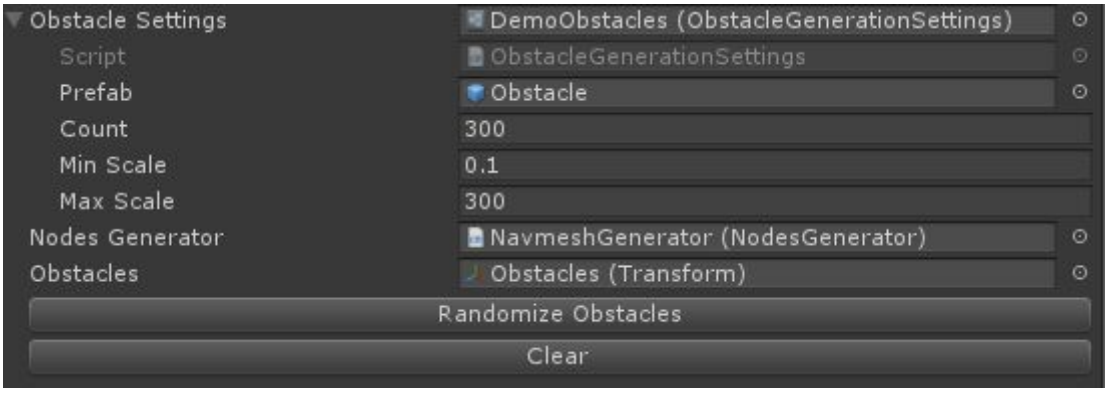


- Algorithms:
  - A Star*: Basic A* evaluating nodes using their cost and estimated distance to goal
  - Theta*: Variation of A* that cuts out nodes that can be skipped for shorter paths based on line of sight, results in shorter paths but is more expensive
- Heuristic / Cost Increase:
  - Functions that determine how cost and estimated goal are calculated
  - Euclidean: Standard line distance, a good default option that is best used for units that can move in any direction
  - Manhattan: Use this for grid based movement without diagonals
  - Octile: Use this for grid based movement with diagonals
  - Chebyshev: Useful for board games like chess figures that can move in several directions of inequal length but equal cost
- Benchmark: Whether to benchmark the algorithm and print stats like required milliseconds, relative path length, opened nodes etc.
- Greediness: Determines how much an algorithm explores its environment, this should usually be 0.7-0.8 and never lower than 0.5 on big grids, lower values produce slightly better paths at the cost of a huge performance drop

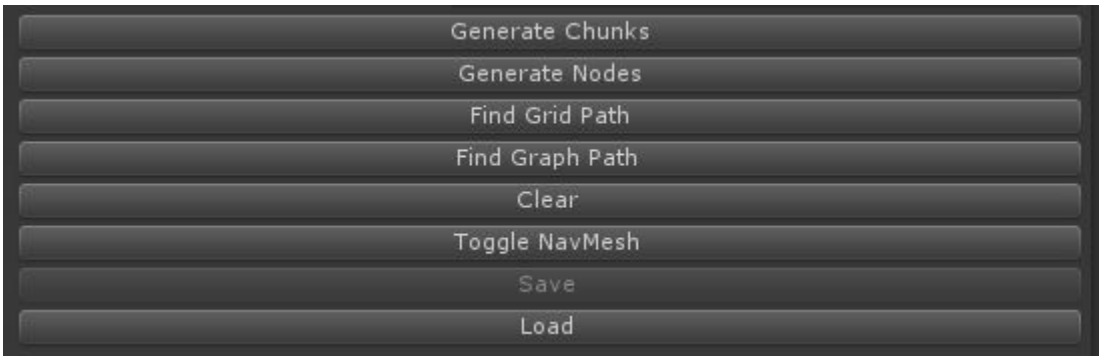


- Pathfinding Type: Whether to use grid and / or NavMesh for pathfinding, try to use NavMesh only unless you explicitly need a grid because it requires much more storage
- Start / Goal: Optional points to test pathfinding in editor
- Chunk Count: How many Chunks to generate in each dimension
- Visualize Pathfinding: Whether to visualize the opened / closed nodes and color them by their F values in relation to the highest / lowest F value
- Compression: Compression method for the MessagePack serialization, Lz 4 Block proved to produce the smallest files

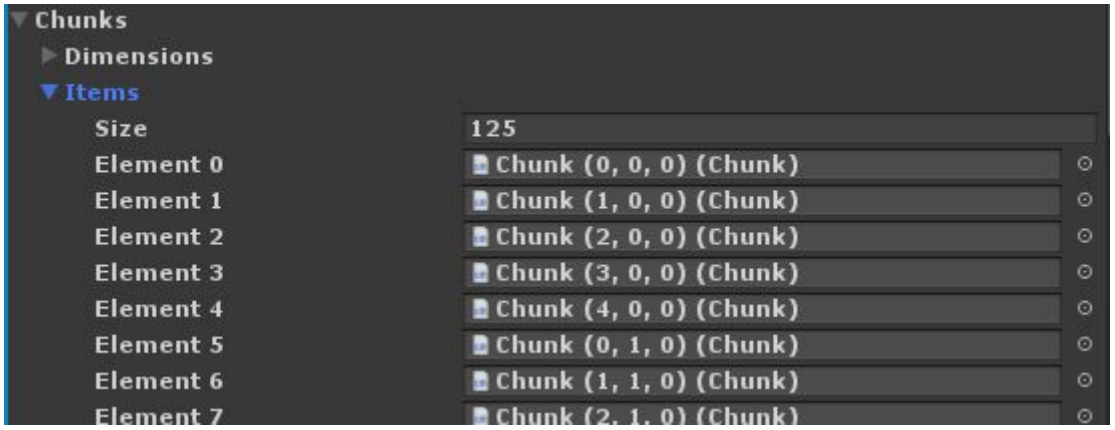
- For testing, you can add an ObstacleGenerator component to the generator and create ObstacleGenerationSettings for it



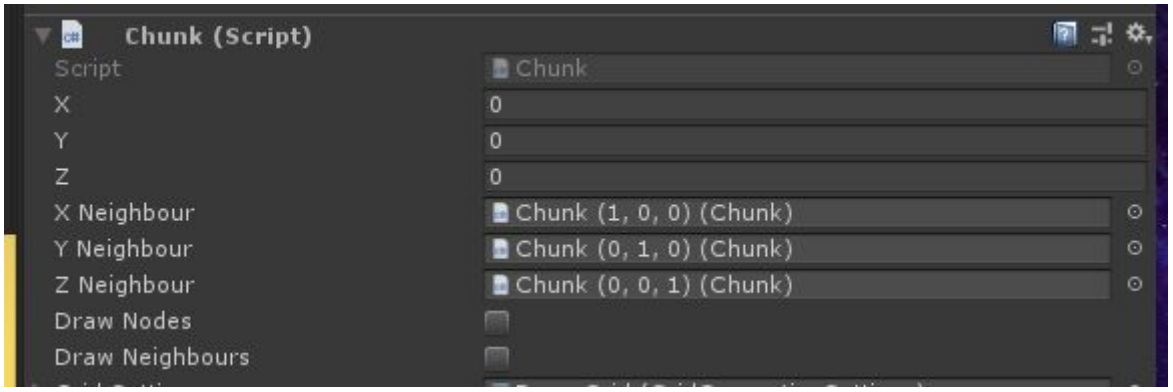
- Prefab: Obstacle prefab to generate, make sure it has the NavMeshObstacle layer and a collider for custom ones
- Count: Amount of obstacles to generate
- Min / Max scale: Range in which the obstacle range will be randomized
- Nodes Generator: Uses Generators chunk size and count to determine possible obstacle positions
- Obstacles: Parent object of instantiated obstacles, needs to have no other children
- Randomize Obstacles: Randomizes obstacle scale / position and instantiates or removes obstacles if Count was changed
- Clear: Removes all obstacles



- Once the settings are finished, you can start generating the nodes by going back to the Generator and pressing “Generate Chunks”, this should instantiate chunks as children of the generator and fill the serialized Chunks array:



- Now, you can press “Generating Nodes”. Depending on your grid and chunk size, this may take several seconds. After you see the console message that it finished, you should see a NavMesh appear, which you can hide / unhide using “Toggle NavMesh”. You can validate the generation by either enabling Draw Nodes in the Grid settings (if you are using a grid) or enabling Draw Nodes / Draw Neighbours on a chunk (if you are using a NavMesh):



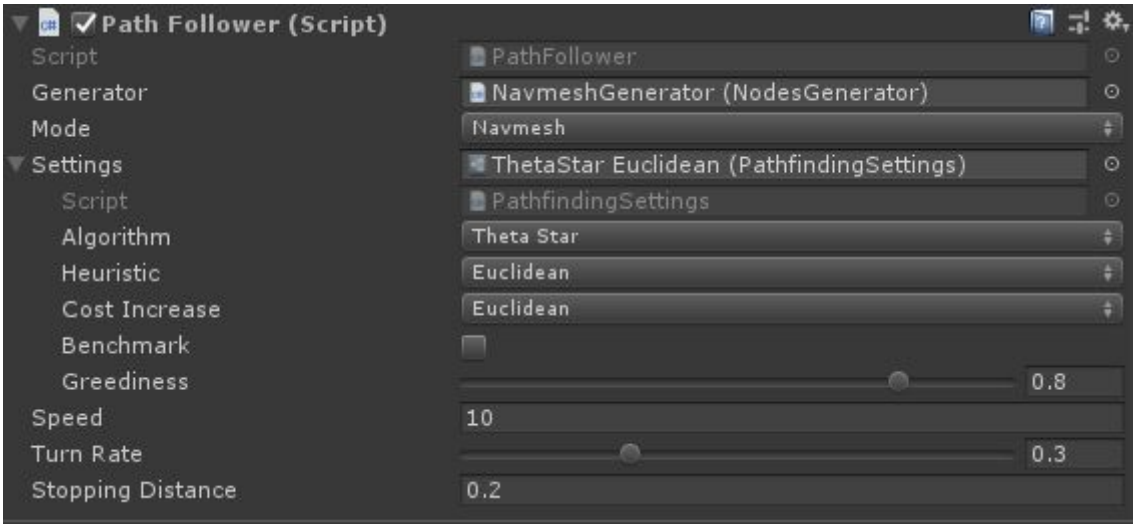
- Don’t forget to press “Save” if the generation was successful. You should now see a file in the StreamingAssets folder called

"SceneName_GeneratorName.nodes". This file will be loaded each start, which can also take some seconds.

- Note: Due to the loading time, path requests before the file is loaded will result in a fail, the NodesGenerator has an Action called OnInitialize which you must wait for

- Be sure to test the loading once by hitting "Clear" and then "Load"
  - While it may seem to load extremely slow, it will load much faster in a built game.
- If you have assigned a start and goal transform, you can now experiment with the pathfinding settings. After hitting "Find Grid Path" or "Find Graph Path", a LineRenderer will display the resulting path, and opened / closed nodes are visualized if Visualize Pathfinding is enabled.

- To make your units find and follow paths, add a PathFollower component to them:



- Generator: The generator to use to find a path (you usually only have one)
- Mode: Whether to move on the grid or NavMesh
- Settings: Pathfinding settings as explained above
- Speed: Speed at which the agent moves
- Turn rate: Rate at which its forward vector aligns to the move direction
- Stopping distance: Distance to a path point after which it is considered reached

PROBLEMS / KNOWN ISSUES

- for certain Greediness values, the pathfinding will sometimes fail, slightly changing it fixes this
- there is no safety connectivity check, if the goal is unreachable, all reachable nodes will be explored, resulting in a short freeze
- Theta* will sometimes path slightly through the NavMesh due to the line of sight Raycasts being unreliable directly on the surface of a mesh