Theta* - Any Angle Smooth Pathfinding Documentation

1. Introduction

Theta* algorithm is an improvement on the more common A* algorithm. The only difference comes in smoothing the path and giving nodes with optimal angles between them. It requires only two inputs: start and end positions in world space. The output is a list of type Vector3, which correspond to world space positions for the shortest path.

2. Usage

2.1 Prefabs

Put the prefab found under “Theta Star Pathfinding/Prefabs/Theta Star Pathfinding” into the scene. It can be used instantly combined with the wall prefabs that come with the asset. If custom objects will be used, their layer must match with the layermask configuration of the ThetaStar component on the Theta Star Pathfinding prefab. The layermask can be changed in the inspector. As long as the layers match, it can be used with any object that has a 3D Collider component (Box Collider, Sphere Collider etc).

The “Smooth Path” boolean can be set to false if path smoothing is wanted. Smoothing is on by default.

The prefab also has the MapFiller component attached. MapFiller is used to generate the grid map and place the obstacles on it accordingly. For the MapFiller to work properly, the ground and obstacle tags must be set. The values in the prefab are currently set to work with the Demo Scene. The MapFiller will work automatically with the Ground and Obstacle prefabs. If custom ground and obstacle objects will be used, the correct tags must be set in the inspector.

The “Test Node Walkable States” boolean can be set to true before hitting “Play”. When set to true, it will instantiate objects for every node in the map. Nodes that are walkable (not blocked by obstacles) and nodes that are not will have different colors. This can be used to debug the map that will be used. It is set to false by default.

2.2 Calling The Method In Script

The ThetaStar class is configured as a singleton. When put to the scene it will initialize itself and generate the map. It can then be accessed using “ThetaStar.instance”. Call the FindPath method found in this class to generate the shortest path. This is the method signature:

public List<Vector3> FindPath(Vector3 _startPoint, Vector3 _endPoint)

_startPoint: This is the starting point in world space coordinates.
_endPoint: This is the end point in world space coordinates.
Returns a list of type Vector3. Let’s suppose we have 2 Transform references, named “Transform1” and “Transform2”; and we wish to find the shortest path between them. An example usage would be like this:

List<Vector3> path = ThetaStar.instance.FindPath(Transform1.position, Transform2.position);

After this, the positions found in the list can be visited sequentially. Example:

1. Move the object directly towards path[0].
2. After arriving at path[0], move the object directly towards path[1].
3. And so on.

3. Demo Scene

The demo scene has a level set up with multiple objects. When testing on the Unity Editor or a Standalone build, left clicking on the ground will change the start point. Likewise, right clicking on the ground will change the end point. The shortest path will be automatically calculated and it will be indicated by instantiating spheres and lines. When testing on a mobile device, touch input will alternate between setting the start and end points. First touch will set the start point, second touch will set end point, third touch will set the start point again and so on.

If you have any questions or suggestions, please feel free to write a review on the Asset Store, contact me by e-mail or join our discord server. You can e-mail me at the address shown on the Asset Store page, or the e-mail address given below:

burak.bayboga@gmail.com