游戏地图表示方法

一个好的表示方法可以提高寻路效率和质量。一般来说，表示的结点越少，效率越高。

Grids

Common grids in use are square, triangular, and hexagonal.

If your movement costs are uniform across large areas of space, then using grids can be quite wasteful. There's no need to have A* move one step at a time when it can skip across the large area to the other side.

- Squares easy to use

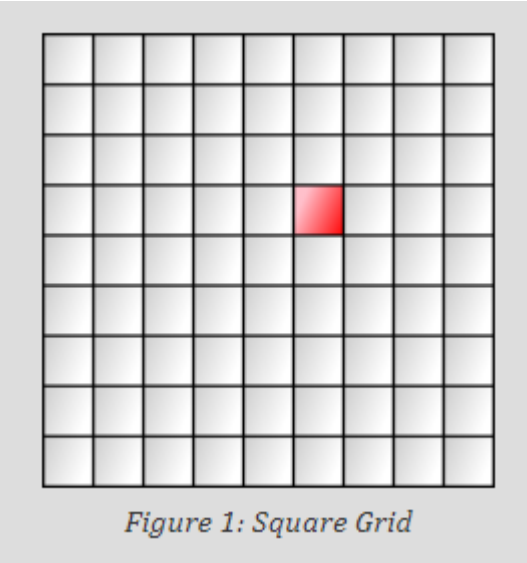


Figure 1: Square Grid

- Hexagons offer less distortion of distance.

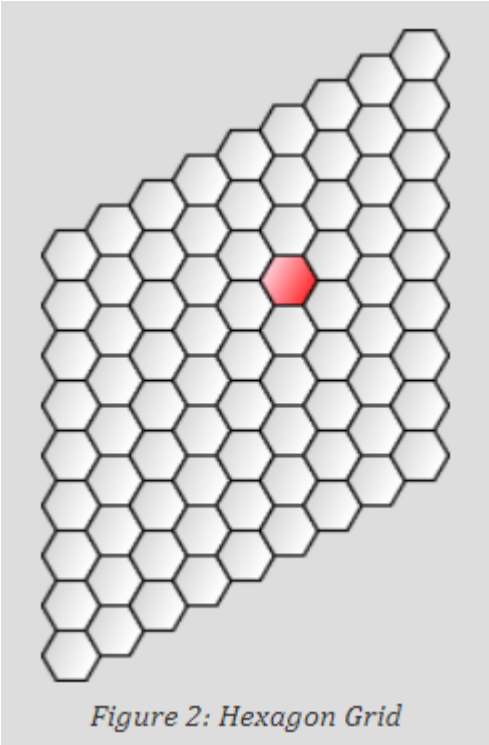


Figure 2: Hexagon Grid

- Triangles rare to use

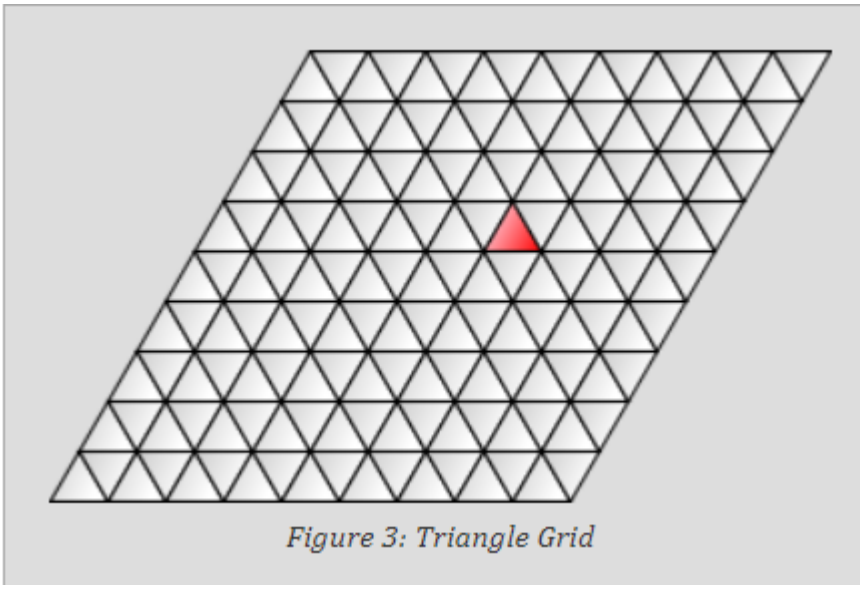
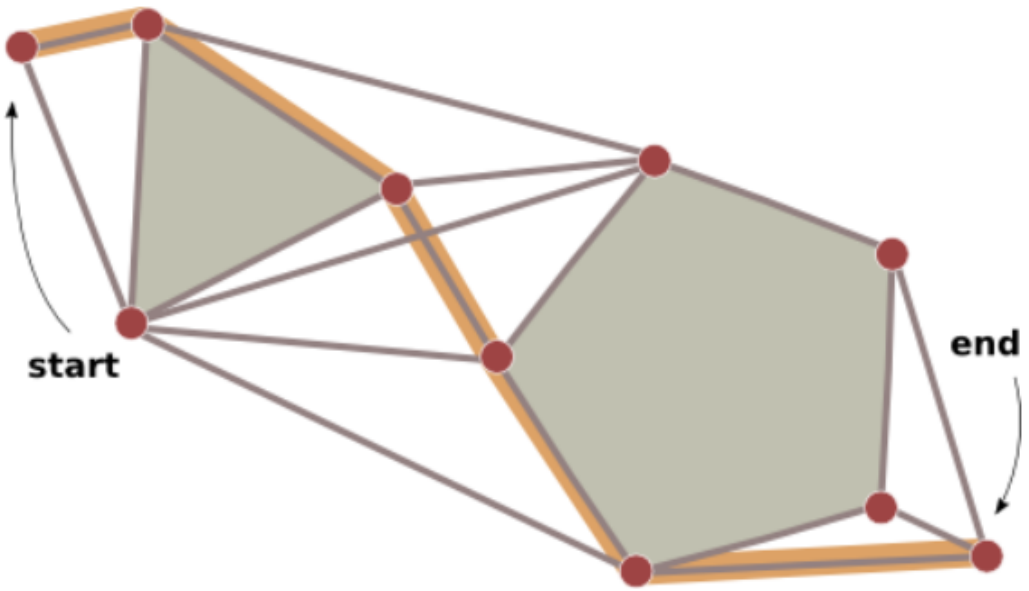


Figure 3: Triangle Grid

Polugonal maps



If the movement cost across large areas is uniform, and if your units can move in straight lines instead of following a grid, you may want to use a non-grid representation.

Three conditions:

1. Using navigation points to tell A* about map data.
2. Visibility graph to build adjance matrix.
3. Travel time between points.

Hierarchical

Having fewer nodes in the map representation is better for pathfinding speed. One way to reduce the problem is to have multiple levels of searching.

For example, to get from your home to a location in another city, you would find a path from your chair to your car, from the car to the street, from the street to a freeway, from the freeway to the edge of the city, from there to the other city, then to a street, to a parking lot, and finally to the door of the destination building. There are several levels of searching here:

- At the *street* level, you are concerned with walking from one location to a nearby location, but you do not go out on the street.
- At the *city* level, you go from one street to another until you find the freeway. You do not worry about going into buildings or parking lots, nor do you worry about going on freeways.
- At the *state* level, you go from one city to another on the freeway. You do not worry about streets within cities until you get to your destination city.

Dividing the problem into levels allows you to ignore most of your choices. When moving from city to city, it is quite tedious to consider every street in every city along the way. Instead, you ignore them all, and only consider freeways. The problem becomes small and manageable, and solving it becomes fast.

We can use different graph-searching algorithm to solve different level map.

For small levels, we can precompute the shortest path between all pairs of nodes(Floyd-Warshall).

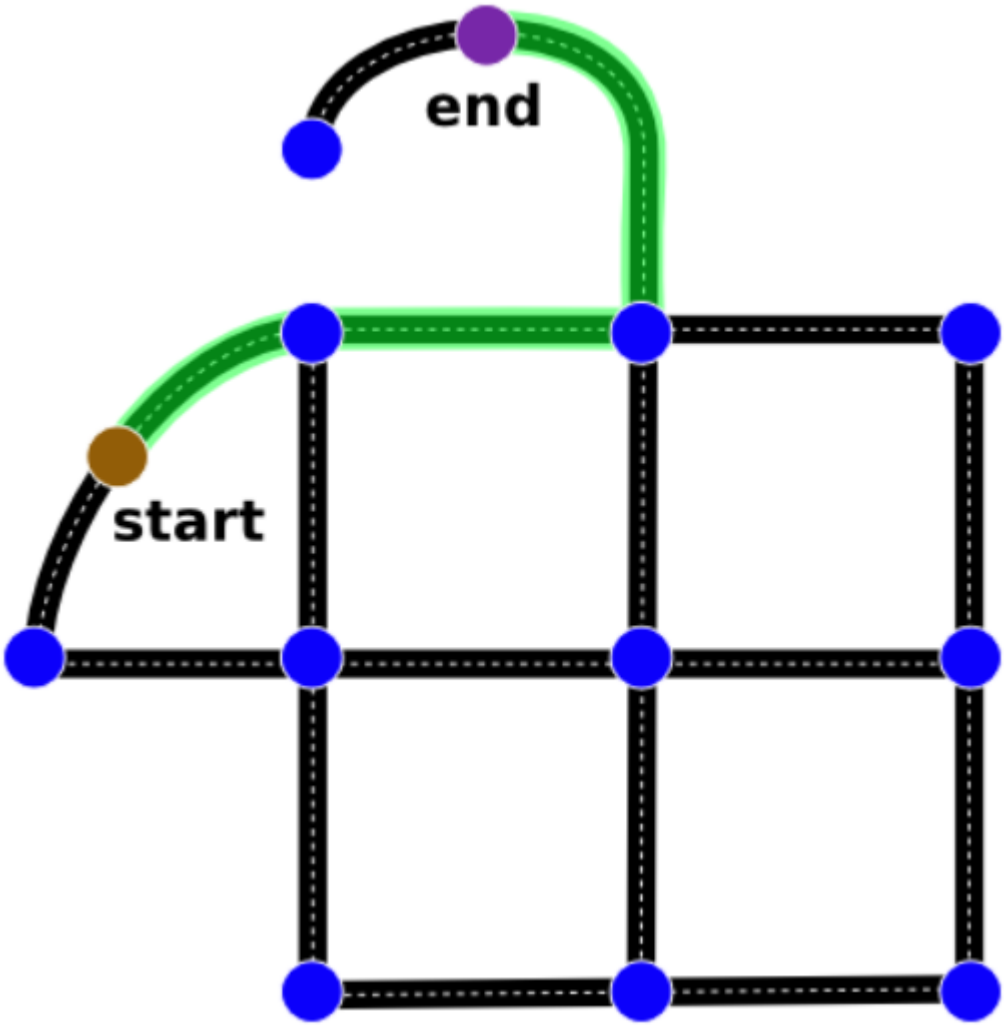
Connected Components

In some game maps, there's no path between the source and destination. If you ask A* to find a path, it will end up exploring a large subset of the graph before it determines that there's no path. If the map can be analyzed beforehand, mark each of the connected subgraphs with a different marker. Then, before looking for a path, check if the source and destination are both in the same subgraph. If not, then you know there's no path between them. Hierarchical pathfinding can also be useful here, especially if there are one way edges between subgraphs.

预处理地图，确定起点和终点是否连通。

即确定两个点是否可达，可以使用连通子图的方式确定。

Road maps



A* can find paths in road graphs environment fairly quickly, because there are few choices to make at each graph node, and there are relatively few nodes in the map.

Skip links

A skip link is a shortcut link.

What should the movement cost be for a skip link? There are two approaches:

- Make the cost match the movement cost of the best path. This preserves nice properties of A*, like finding optimal paths. To give A* a nudge in the right direction, break the tie between the skip link and the regular links by reducing the skip link's cost by 1% or so.
- Make the cost match the heuristic cost. This makes a much stronger impact on performance but you give up optimal paths.

Adding skip links is an approximation of a hierarchical map. It takes less effort but can often give you many of the same performance benefits.

Waypoints

A *waypoint* is a point along a path.

Waypoints can specific to each path or be part of the game map.

Waypoints can be entered manually or computed automatically.

When automatically computed along a path, waypoints can be used to compress the path representation.

Since the goal of skip links is to make pathfinding faster when those links are used, skip links should be placed between designer-placed waypoints. This will maximize their benefit.

If there are not too many waypoints, the shortest paths between each pair of waypoints can be precomputed beforehand (using an all-pairs shortest path algorithm, not A*).

The common case will then be a unit following its own path until it reaches a waypoint, and then it will follow the precomputed shortest path between waypoints, and finally it will get off the waypoint highway and follow its own path to the goal.

Graph Format Recommendations

Using visibility graph representation produces the best paths, but use lots of memory for edges.

Using navigation meshes work work properly.

Hierarchical maps use multiple levels of representation to handle both corase paths over long distances and detailed paths over short distances.

- keeping walkable polygons to keeping only navigation points
- moving along vertices to moving along polygon centers.

Using vertices is better for obstacle avoidance, and if you're using path smoothing it won't negatively affect path quality. Without path smoothing, edges might perform better, so consider either edges or edges+vertices.