Path Finding Algorithms

In any game there are a certain mandatory things that need to be incorporated. There are many situations where game play requires movement of NPC towards a certain location and/or towards the player. There's usually more than one possible path available to NPC and finding the optimum one is crucial for an intelligent AI.

Concepts:

Path finding techniques can be broadly classified into two types.

1. Decision Based and
2. Recursive

1. Decision Based Algorithms

While dealing with simple games and/or early levels of a game where the AI doesn't have to be very intense, these algorithms come in very handy. There are two commonly used algorithms, namely, a) Non-rotational and 2) Rotational.

a) Non-Rotational Path Finding Algorithm

One of the benefits of this algorithm is that it requires very little processing power. It is not very realistic with main idea being to move in the direction of the enemy. It's as simple as moving right or moving forward. It aligns firstly in the x-direction and later in the y-direction. Randomizer could also be added to decide the order.

b) Rotational Path Finding Algorithm

If you have created a simple game in which your characters rotate when moving around i.e. they look where they go then this is the desired algorithm. This algorithm, however, has disadvantages like trigonometry requirement and it is computationally more expensive.

2. Recursive Path Finding Algorithms

Recursive path finding algorithms are the most powerful algorithms and hence are widely popular. They require heavy computer power as they run in the exponential time and a with small increase in map size, run time increases a lot.

Algorithms falling under this category are, i) Depth-First Search, ii) Breadth-First Search, iii) A* path finding and iv) Dijkstra's.

i) Depth First

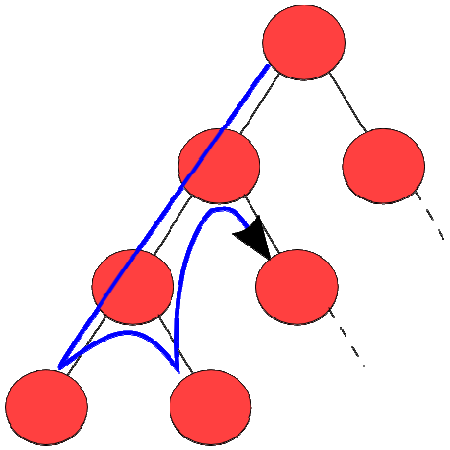

Fig. 1. Depth-First path finding

ii) Breadth First

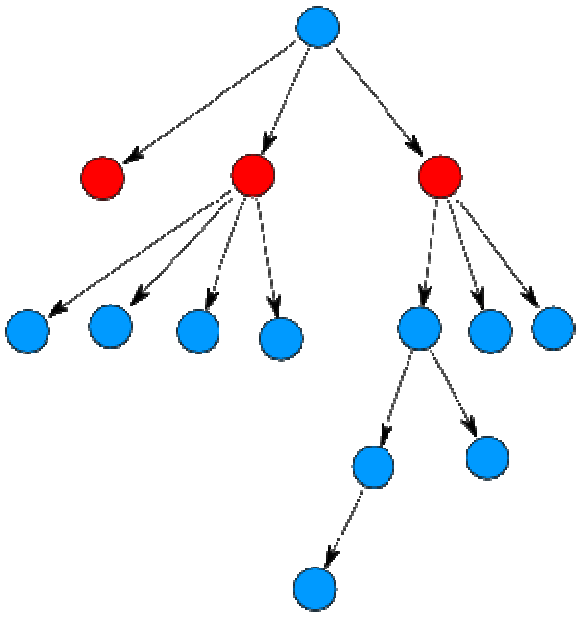

Fig. 2. Breadth-First path finding

iii) A* path finding

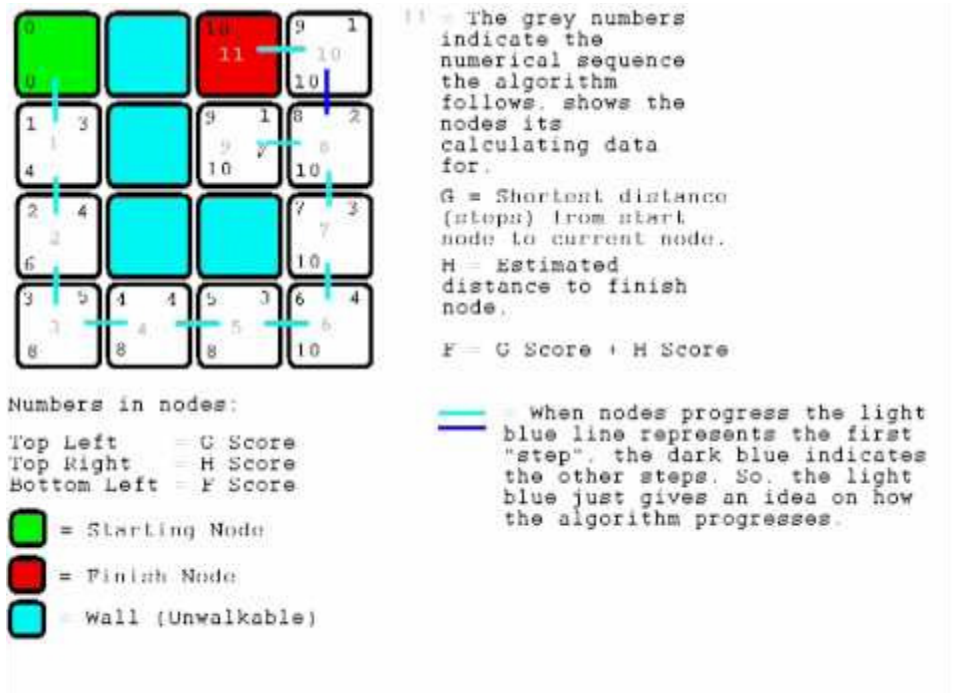

Fig. 3. A* path finding

Search

We begin the search by doing the following: 1) Begin at the starting point A and add it to an "open list" of squares to be considered. 2) Look at all the reachable squares adjacent to the starting point, ignoring non-reachable squares Add them to the open list, too. For each of these squares, save point A as its "parent square". 3) Drop the starting square A from your open list, and add it to a "closed list" of squares that you don't need to look at again for now.

Score

Path selection is done using the below mentioned equation

F = G + H, where

G = the movement cost to move from the starting point A to a given square on the grid.

H = the estimated movement cost to move from that given square on the grid to the final destination, point B. This is often referred to as the heuristic.

iv) Dijkstra's

While A* is generally considered to be the best path finding algorithm there is at least one other algorithm that has its uses – Dijkstra's algorithm. Dijkstra's is essentially the same as A*, except there is no heuristic (H is always 0). Because it has no heuristic, it searches by expanding out equally in every direction. As you might imagine, because of this Dijkstra's usually ends up exploring a much larger area before the target is found. This generally makes it slower than A*.

Tasks:

1. Create classes for path finding algorithms.
2. Create a sample map and implement the above mentioned algorithms.
3. Try and find out which algorithm works better than the other in different conditions.