

Background

In 1995 Westwood Studios released Command & Conquer to universal acclaim. The game's success helped to popularize and define the RTS genre as a whole, and the franchise has been a commercial success ever since.

During those days the AI was fairly simple. The AI always chose the shortest route from point A to point B ignoring any enemy defenses, tiberium(infantry took damage if they walked on tiberium), and they wouldn't destroy sandbags as long as there's a way around it (you could guide the enemy straight to your base defenses).

The latest game in the Command & Conquer franchise Command & Conquer: Tiberium Wars had a very versatile AI. They would try to flank you, avoid your defenses, and attack your weak spot.

Requirements

First of all the pathfinder should be able to find the shortest way first on a simple map.

Additionally some units move faster on some terrain than other, and the pathfinder will be able to find the fastest route for the specific unit. For example a jeep would be fast over flat landscapes, but would be considerably slower in woods, while infantry wouldn't lose any speed in woods. The pathfinder should be able to distinguish between these, and jeeps would be given a route that goes around the woods, while infantry would be given a route straight through.

We would like to give the AI several options to choose from; shortest way, avoid enemies, etc.

Design Specifications

The algorithm we have chosen is called A*. A* uses a logic approach to find the goal, and by design reaches the best solution first. This fits in very well with the requirements for a game, where speed and good results are important.

We have chosen to implement the algorithm as a class. The class structure will be as follows:

Class cAstar public member functions:

- void Init(cPoint* start, cPoint* end)
- bool goNext()
- void checkBorders()
- void cleanup()

In addition to these public functions, the class has several private members it uses internally.

Using these public functions finding a path will work like this:

```
// Init positions
star.init(start, end);

// Search until found.
while(true)
{
    // Find the next optimal position and go there
    if(!star.goNext())
    {
        cout << "No path found!" << endl << endl;
        system("pause");
        exit(0);
    }

    // Are we at the goal? if so, stop searching
    if(star.found)
    {
        end->parent = star.pos->parent;
        break;
    }

    // Check squares next to this and add them to the open list
    star.checkBorders();
}
}
```

cPoint is a simple class that implements a point:

class cPoint

- cPoint* parent
- int x, y
- int cost
- int distance

Distance is the distance to the goal, calculated any way we find fit.

Cost is the amount of time needed to walk, or rather, an arbitrary number who's sum should be as small as possible. In our task the cost is related to movement; moving from one square to another, diagonally and so on. Different terrain will intice different cost, thus taking into account sand, grass, hills and other terrain types.