

Introduction

This chapter will present various test cases and other points of interest that we have researched during our development and test phase. For the most part we got the expected results, but there were some surprises.

The initial problem presented by the tutorial we used is as follows: find the most optimal path from point A to point B on a matrix. The matrix contained obstacles that a unit would have to work around. These experiments are for showing search paths, and the path the unit chooses on different maps.

Legend

Green = Sweep path

Purple = Point A and Point B

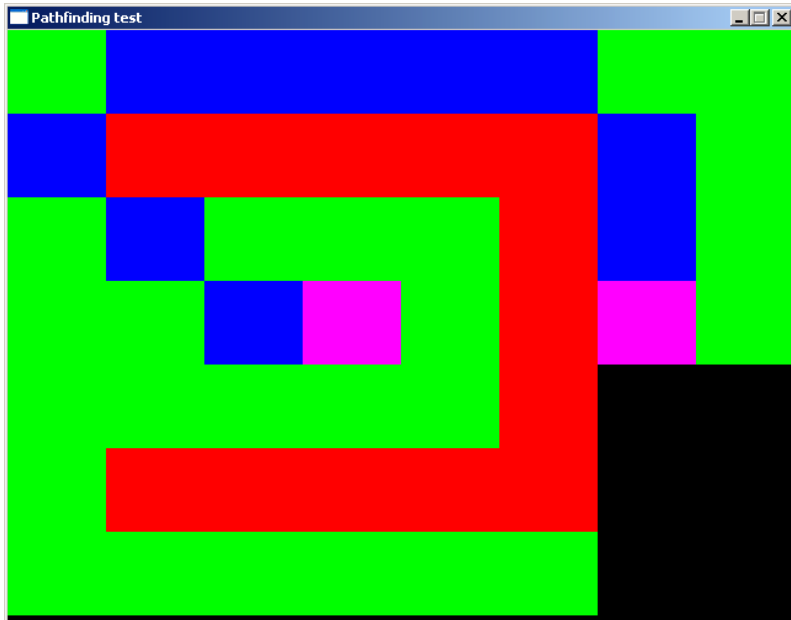
Red = Obstacles

Black = Walkable

Blue = Chosen Path

Solution

Here A* algorithm would first try the shortest way, when it hits the obstacle it will search the area behind until it finds an opening. Then it will search until it finds the point B. As the algorithm tries the shortest way first the search area is bigger than the initial problem.



There are two paths to point B which is exactly the same distance. In this case A* would just pick the path it finds first.

Computation speed

Various factors play a vital role in how fast the A* algorithm can calculate the path.

Map size

The size of the matrix greatly impacts the time it takes to calculate the path. The initial problems presented are very small in size and is calculated almost instantaneously. On a 7x5 matrix where the path from point A to point B is a straight path, it takes 1 millisecond to calculate. On a 160x120 matrix it takes 50 milliseconds to calculate a straight path. Thus the time for a straight path scales almost linearly to the distance from Point A to B.

Complexity

Obstacle complexity impacts the calculation speed more dramatically and is hard to predict. Optimizing code for this is outside our assignment as it might require specialized cases.

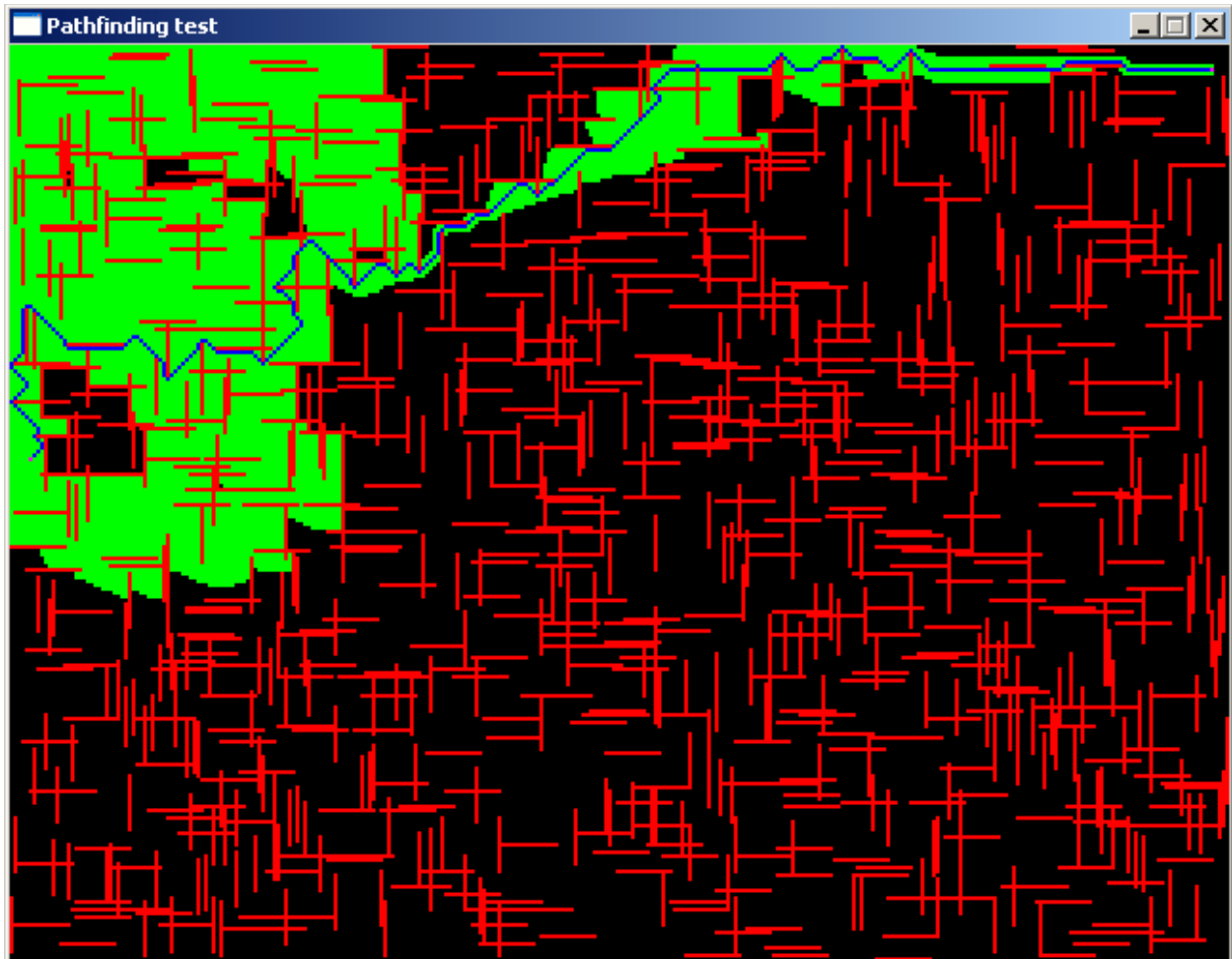
External factors

Location of the two points is highly vital for calculation speed. If the two points are close together size the map size hardly matters.

The A* algorithm takes heavy use of list operations. In our implementation we decided to use the standard C++ list library. This library is very complete and contains everything needed for a list class, but not might be optimal for our use. Accustomed link list implementation specifically written for our task would most likely perform much better.

Implementation of this class is outside our assignment, but should be considered in real uses of the algorithm.

Complex map



In this case a lot of the area had to be swept to find the most optimal path. This map combines several of the factors that impacts calculation speed: large map, many obstacles and a path that often changes direction. This path takes approximately 5 seconds to calculate. Note that there is possible to construct a problem that is much more complex than this, but these will most likely not be seen in a real scenario.

Memory usage

Total memory usage is dependent on the same factors as speed, but in all cases the memory requirements are minimal compared to computational requirements. Memory usage range from about 200 kbyte up to 1 mbyte, and this includes the full application as well as the path data. Any other considerations regarding memory are not necessary at this point.