Path Planning

CP468 Artificial Intelligence Term Project

Jake Buller: 090376280

Michael Dougall: 090327760

Travis Miehm: 090338930

04/04/13

0. Project Description

Consider a Museum room that is patrolled by N robots at night. At a pre-determined time, the robots are supposed to rendezvous at a given point R in the room. The robots move inside the room, and the room contains obstacles, such as chairs and benches for the visitors, paintings, sculptures etc. The robots are supposed to know the location of the obstacles in the room.
Implement an A*-based algorithm to compute the path of each robot, from its initial position to the given rendezvous point R.

1. Design Decisions

In order to tackle the problem presented the first decision we needed to make was which programming language to use. For this project we decided to use C#. We chose C# because it is a modern language and it is used in the industry for producing release-quality software. Early on it was also decided that we would use GitHub for version control during the project. It helped in the case that a problem arose as we could go back to previous versions of code and determine the changes that led to the problem. It also ensured that everyone was using the same version of the code when they went to work on it.

Next we needed to decide how the various elements of the problem would be represented programmatically. Since A* is designed as a graph/tree search we needed to implement a representation of a graph to perform the search on. A graph is composed of both edges and vertices. We created a point structure to represent the vertices in the graph. The resulting data structure, the class ExtendableMap in our program, contains a list of points, and a list of edges (as Strings). In order to do searching a structure is needed for the frontier of A*. For this we have used a List of Points. This list is sorted by the weight of each Point in the frontier where a nodes weight is the sum of the heuristic function at that point, and the actual path cost from the current Point to the Point in the frontier.

A* is a heuristic based algorithm so choosing a heuristic function for our implementation was the next major design consideration. We chose to use a direct path heuristic. This heuristic is a good choice for two main reasons. First of all, the direct path heuristic is admissible. For any Point in the graph, in order to get from that point to another, the absolute best case is a straight line path, in which case the heuristic will be equal to the actual path cost. In the case where the path is not a straight line the heuristic will always underestimate the cost to get from the current node to the goal node. Since the heuristic never over estimates the actual cost, it is admissible. The second reason that this heuristic is a good choice is because of its computational simplicity. We can compute the heuristic in linear time (a few small operations) so this will not have a negative effect on our run time.

2. Optimization Techniques

In order to create an efficient implementation we made two decisions to improve the programs run time. The first major optimization was to build the map with only points that the robot could go to. This means that as the map was built, if a point in the map was an obstacle it would not have a point in the graph. The robot was able to “see” the available paths at each point by its neighbouring points that had an edge to the current node, but was also able to detect an obstacle by seeing that a node did not have an edge to that particular obstacle point. In the case of a graph that has no obstacles this would have no effect but for the average graph which have a number of obstacles this technique reduced the search space by eliminating all of the obstacles from the graph.

The second optimization we used was parallelizing our search. The robots in the map work independently and each only cares about its own path to the goal state. We utilized this fact to parallelize our program. The program first builds the map; this is the only piece we run serially because each robot uses the same map. This initial map does not include the robots start states. We run a parallel loop that creates an instance of the problem, now including the robots initial position, and the

goal state. On this instance of the problem we run our A* search to find the path from the robots initial state to the specified goal state.

3. Compiling and Execution Instructions

Before our code can be compiled and run, Microsoft .NET Framework v4 must be installed. To check if it is already installed, navigate to "C:\Windows\Microsoft.NET\Framework\" and look for the folder "v4.0.30319". If it is not installed, please first download it from here "http://www.microsoft.com/en-ca/download/details.aspx?id=17851" and install it. Next, our code can be downloaded from our github repository, "https://github.com/jakebuller/PathFindingProject". If you have a github account, you can clone the repository, but downloading it as a zip file may be easiest. Once that is complete, extract the zip file to some location, which will be called "%EXTRACTION%". Open a command prompt, and navigate to %EXTRACTION%. Now run the build.bat file, like so:

%EXTRACTION% > .\build.bat

The output of this operation is copied to %EXTRACTION%\PathFindingProject\bin\Release\ - navigate there. You can now run our program as follows:

%EXTRACTION%\PathFindingProject\bin\Release > .\PathFindingProject.exe %FILEPATH%

where %FILEPATH% is an absolute or relative path to the file you wish to use as the input file. Some sample files have been provided, located at %EXTRACTION%\PathFindingProject\Maps.

The output of our program is put into a file, called output.txt. This file is located in the same directory as the executable, namely %EXTRACTION%\PathFindingProject\bin\Release\.

4. Test Results

To test our program's performance we ran a series of tests to determine how long the program took to solve different instances of the problem. We ran three different maps with 1, 2, 3 and 4 robots positioned in them, each configuration was run 100 times to obtain an average time to find the solution to the instance of the problem.

Map: 20x20 with obstacles

# Robots	Average time to find solution
1	6.49ms
2	13.97ms
3	16.50ms
4	20.68ms
1 No solution	2.53ms

Map: 25x 25 with obstacles

# Robots	Average time to find solution
1	11.15ms
2	13.51ms
3	17.49ms
4	21.01ms
1 No solution	7.13ms

Map: 30x 30 with obstacles

# Robots	Average time to find solution
1	17.74ms
2	24.99ms
3	27.05ms
4	32.81ms
1 No solution	16.28ms

We wanted to do test runs to see how our program handled a large map so we ran our program on a 1000 x 1000 map with no obstacles. Our program solved the map in 46 seconds.