Moving problem solving robot(s)
Multiagent A*

Imara van Dinten

University of Southern Denmark, Campusvej 55, 5230 Odense M, Denmark
imvan13@student.sdu.dk

Abstract. In this paper I explain how I implemented an A* algorithm to reach an object with multiple agents. Collision detection is used so the agents won't collide. This is all done with the minimum amount of external libraries. The initial concept saw use of lpzrobots for this implementation. However, due to time restrictions the project was mainly focussed on the implementation of a multiagent A* algorithm.

1 Introduction

For Artificial Intelligence 4, I was given the opportunity to do a project of my choice in the area of Artificial Intelligence. I decided to take a look at implementing a way to let multiple robots work together in order to move an object. See project description.[1]

2 Materials and Methods

The original planning was to use lpzrobots[2] as this facilitated a 3D environment I could use. There is a lot of research going on using this software. This means that there is a lot of functionality I could have used.

During the research stage I concluded that this framework could be of good use for my project. To get everything properly working would take some time. See Figure 1 for my set-up of the environment and robots for this project.

Before continuing my project using lpzrobots I decided to go back to the main problem of the project, which is being able to get multiple robots to move to an object without obstructing each other. After this they need to push the object to a different location.

I decided to change the path my project was going. Ideally, I wanted to get the algorithms working while receiving "simple" feedback in the form of a terminal ASCII image. The whole project was done using C++11, as this had a few nice utilities in comparison to C++95.

[1] Imara van Dinten. "Moving problem solving robot(s)". In: 2014.
[2] Universitat Leipzig. *lpzrobots*. URL: `http://robot.informatik.uni-leipzig.de/` (visited on 05/31/2015).

Other methods used were an A* algorithm. Compared to Dijkstra, A* will be faster. This is due to the fact that the environment is already known. Even if the program is implemented in a real world situation, the environment will still be already known. In a real world environment, the problem still deals with moving an object from point A to point B. So similar abstractions can be used. This makes it so that using this algorithm won't cause any problems later on when switching to a real environment. Combined with the A* algorithm, the chosen distance policy is to use the Euclidian distance. The Manhattan and Chebyshev distance policies are also implemented and you are able to easily switch between these if needed. Though the Euclidian distance gave faster results compared to the others.

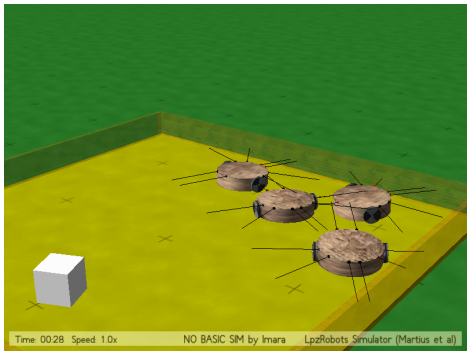

Fig. 1: lpzrobots

3 Experiments and Results

My first experiment was to be able to implement an A* algorithm while using a single Agent. See Figure 2 for the end result. A* was chosen over other pathfinding algorithms because of its performance characteristics while still being relativity easy to implement.

After being able to use an A* algorithm for a single Agent, my next experiment was to be able to use two or more Agents. One of the necessities for this goal was to implement collision detection. This was done by running multiple passes of the pathfinding algorithm while keeping pre-calculating routes in mind. The implementation of this collision detector is vital if the agents are to work together without hindrance.

A future goal of this project is to try and solve the problem with real robots, by use of my pathfinding algorithm. Figure 3 shows the end result of this algorithm. In this case I let two agents start on the same coordinates (0:0). In real life this wouldn't be possible, but this is the easiest way to see that the algorithm works. They are able to cross each other's path as long as this is in a different

time frame. The number of steps away from its starting position is used as a measure of time. Each step is one unit of time.

Finally, the algorithm was adapted to work dynamically with more agents. Figure 4 shows the final result. Four agents are planning to go to the same object without hindering each other. After reaching this point, they will move together to a new point, see the second image in Figure 4. This is to test how they will work together if the object needs to move to a new location.

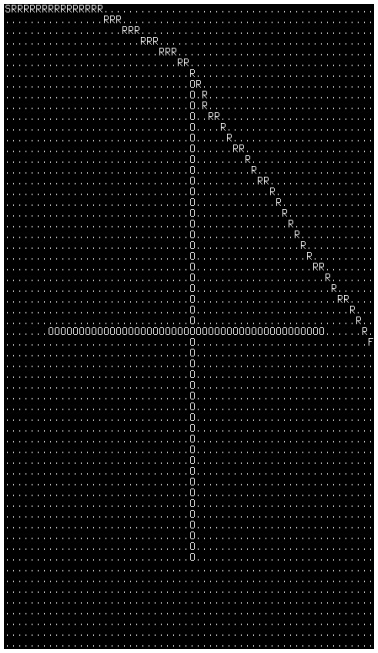

Fig. 2: Single agent

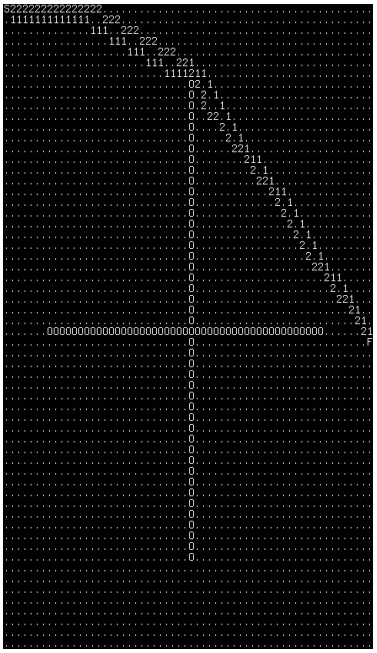

Fig. 3: Collision detection

4 Discussion and Conclusion

In the end there was no more time to implement the algorithm within the lpzrobots environment. But I'm confident this is possible given some time. Lpzrobots gives the project an extra layer of complexity, which was interesting, but unfortunately took more time than estimated. The last part in which the robots work together to move the object, needs some more fine tuning. One thing which would be nice is to implement patterns that the robots will use to gather around the designated target in order to move it. Right now, they all focus on getting to the same X and Y target coordinates. It is possible to implement an algorithm that takes the desired moving direction of the object into account. This means that the agents would have to adjust their position and direction accordingly.

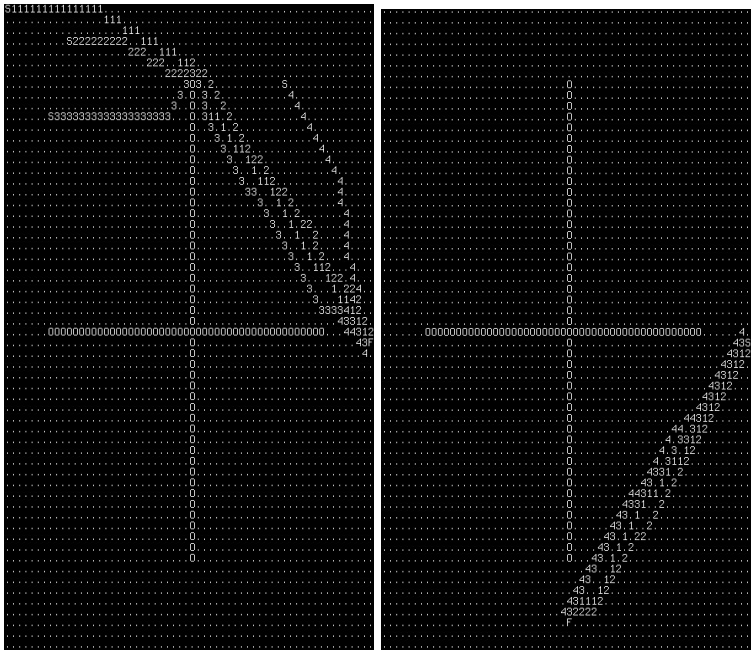

Fig. 4: Multiple agents. Going from point A to B and from B to C.

Using lpzrobots will be better to show the implications of implementing said algorithm. The current simulation limits the possible directions to 8, which is vertical and horizontal displacement combined with diagonals. In my original project description, I also designated some features as "could" and "would like" according to the MoSCoW[3] priority system. These features included the addition of several unique characteristics for each robot. This would definitely be nice but will definitely take some time to implement. This would be better suited for inclusion in an lpzrobots simulation.

The code can be found in my Git repository,[4] a copy of this document can also be found there.

5 Acknowledgements

I would like to thank Xander G. Bos for the brainstorm sessions on how to better abstract various data as separate classes. I would like to thank the ActiveState[5]

[3] DSDM. *MoSCoW*. URL: `http://www.dsdm.org/content/10-moscow-prioritisation` (visited on 05/31/2015).

[4] Imara van Dinten. *Git Repository*. URL: `https://github.com/DarkRiddle/MultiAgentPathfinding` (visited on 05/31/2015).

[5] FB36 et al. *A-star Shortest Path Algorithm*. URL: `http://code.activestate.com/` (visited on 05/31/2015).

community for their help. They helped me troubleshoot several bugs in the core of the A* algorithm.

A book that was invaluable to understand some core mechanics in the development of my algorithm was "Artificial Intelligence for Games"[6] by Ian Millington and John Funge. The authors have written a great deal about Pathfinding and other AI constructs.

[6] John Funge Ian Millington. "Artificial Intelligence for Games". In: Morgan Kaufmann, 2009, second edition.

References

Dinten, Imara van. *Git Repository*. URL: https://github.com/DarkRiddle/MultiAgentPathfinding (visited on 05/31/2015).

— "Moving problem solving robot(s)". In: 2014.

DSDM. *MoSCoW*. URL: http://www.dsdm.org/content/10-moscow-prioritisation (visited on 05/31/2015).

FB36 et al. *A-star Shortest Path Algorithm*. URL: http://code.activestate.com/ (visited on 05/31/2015).

Ian Millington, John Funge. "Artificial Intelligence for Games". In: Morgan Kaufmann, 2009, second edition.

Leipzig, Universitat. *lpzrobots*. URL: http://robot.informatik.uni-leipzig.de/ (visited on 05/31/2015).