CMRoboBits (15-491) Fall 2007 – Homework 4

Due Wednesday, October 10th 2007 <u>at the beginning of lab</u> (12:30pm)

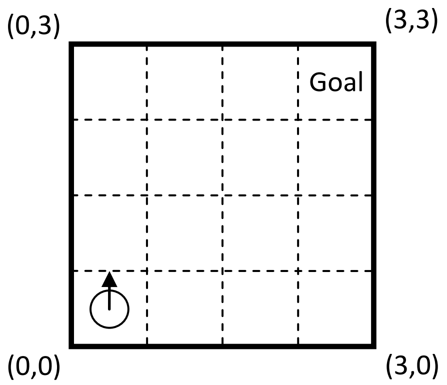
Path Planning

In this homework you will implement a grid-based path planning algorithm in C# to navigate a maze with a scribbler robot. The initial configuration of the maze will be unknown and your robot will have to use the distance sensor to detect walls and then re-plan its route as it goes along. **Once it reaches its goal, it should plan a path and drive back to the start using the map it has acquired along the way.**

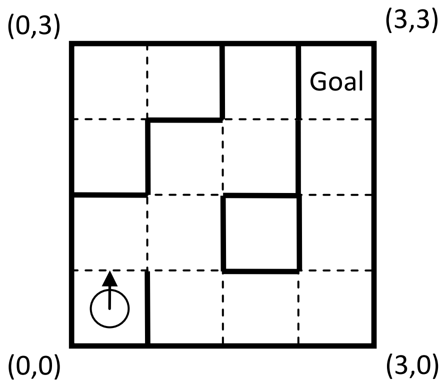
The following Domain Information is provided:

- Your robot will operate within a 4x4 grid domain. There will be outer walls enclosing the entire maze (see drawings further below).
- When starting, your robot will be placed at cell (0,0) facing North (that is, facing towards cell 0,1).
- The goal will be at cell (3,3).
- The size of one grid-cell is given by the cardboard-pieces in the REL.
- There exists a path from the Start to the Goal.
- Some cells in the grid might be unreachable.
- Your only possible actions in terms of planning are moving North, South, West, East (you should not move diagonally). In terms of the robot's actuation this will result in either turning in 90 degree increments or moving forward one grid-cell.
- There will be some walls in your way. Walls are always placed on the edge between two grid-cells. Your robot will need to detect these walls using its front distance sensor.

Note: During the demo you will be allowed to manually re-center the robot within the grid cell, and re-orient to the closest 90 degree angle after each move. This should take care of the dead-reckoning issue. You should probably use a dialog box to indicate program resumption to the computer.



Your initial domain knowledge.



One possible maze configuration.

Your basic algorithm outline should look somewhat like this (this is a suggested outline, feel free to modify to your needs):

1. In VPL: Check front distance sensor which will return either true or false
2. In your C# service: Check whether you have reached the goal: if your current robot's location is (3,3) then reverse the problem, setting field (0,0) to be the new goal.
3. In your C# service: update your internal map structure using the distance sensor result
4. In your C# service: run your grid-based path planning algorithm (preferably A-Star, or pure Breadth First search)
5. In your C# service: if your plan says to move in the direction you are currently oriented, return a "move forward one grid-cell" to VPL. If your plan says to move into a direction you are currently not oriented, return a "rotate left 90 degrees" or "rotate right 90 degrees" to VPL. Update your internal state in C# accordingly.
6. In VPL: execute the move (either turn or move forward)
7. In VPL: wait for "Ok" signal from user. You should use a dialog for this. Additionally, it is a good idea to also display your current internal state (which grid-cell and orientation) that your C# service returned. This allows you to manually re-align your robot to the center of the grid-cell and to re-orient it in case it went a bit sideways due to dead-reckoning. Once re-aligned, the user can hit "ok" and you go back to step 1.

Inside your C# code you should keep track of your domain. You should have some sort of data-structure/array that models the cells and walls in your world. You should furthermore keep track of your current cell and orientation (NSWE), as well as the goal.

Path Planning Algorithms

You can feel free to choose whichever grid-based path planning algorithm you prefer. We recommend however that you use A-Star because it's both fast and optimal (assuming you use a simple goal-heuristic, such as Euclidean distance). Alternatively, simple Breadth-First search is also a good choice because it's also guaranteed to be optimal. Make sure to discuss your choice of algorithm in the write-up.

C# Service and VPL

In order to create a C# service, it's best to start out with a simple VPL activity where you carefully define your function input and output and create a simple connection between the two (e.g. a calculate box connecting the input and output of your activity). Once you are absolutely sure that your function definition is complete you should hit "Build->Compile as a Service" from the Menu. This will generate C# code for your activity which you can then edit and compile to be used as a VPL service. **Important:** Note that when you hit Compile as a Service from the VPL menu, it will completely rewrite the C# code, so you should only do it once to create your code-skeleton. Once you start writing C# code you should no longer use this function, otherwise you will lose any changes you have done to the C# you generated previously.

Demo

As usual, your group will have to demo at lab-time.

Submission

Your entire homework should be submitted by copying it into the "**dropbox/lab04**" folder on your personal AFS space before the due time. **Make sure to submit your C# code as well!**

Write-Up

Please also submit a write-up for this lab (one write-up per group is fine). In it, describe what internal data-structures you used in your C# code and which path planning algorithm you implemented.