Alec Goebel
AI
11/20/07

A* Design:

The overall architecture of the A* code was based around using previously designed grids and nodes with a couple new helper functions and one primary function. Most helper functions have been broken out into either the lib or the ext folder, depending if they can be carried between projects (lib) or not (ext). An example of an ext function would be a case specific print-grid. The body of the code and test cases have been included in astar.scm, the only file outside of the subfolders.

The two primary algorithms were A* itself and its heuristic. The A* algorithm first calculated how far off from its goal it was. If it wasn't at the goal, it changed the current node to closed and it grabbed all new possible open nodes and added to a list. It then checked to see if any of the new nodes were a more efficient way to get to a closed/open node. If they were, it swapped them in and removed them from the new list. Finally the new nodes were added to open.

To estimate the remaining distance, I used the Manhattan distance squared of each cell. This made it so that squares which were close or in the right place were weighted more valuable. As a consequence, some of the larger squares had a much harder time sorting because it would not necessarily work towards moving a single square a prolonged distance. For larger squares to be more efficient the grid would have needed to be subdivided into either squares or rows.

The main new data type was a state which stored all important information from a previous loop through the function. This includes the direction moved, a copy of the grid, the estimate of distance left, and the distance moved. The state is used to retrace the steps of the program as well as compare whether or not a new node is cheaper than a previous. A state is essentially the byproduct of running the main loop once.

There are new important global variables except the test cases and the debug variable for debug printing.

CCV Design:

The overall architecture of the CCV code was based around using previously designed grids and nodes with a couple new helper functions and one primary function. The helper functions have been broken out the same way as A* used.

The first algorithm used was a simple placement algorithm, which added one CCV to each row of a completely empty grid. The other simple algorithm was the check-conflicts which checked if the CCV could see another CCV. It did this by taking a CCV's position (didn't actually need to have a CCV in it) and checked if it could see the edge of the board in every direction. It returned the number of directions it could not see the edge for.

The primary algorithm, check-ccv, cycled through the grid and tried to minimize the number of conflicts. First, it gets a list of all the CCV positions and how many conflicts each has. If there are no conflicts in any, it quits. Otherwise it picks one at random and finds the best possible place for it within its row according to the other positions and swaps itself into that space. The function then repeats.

Since the algorithm had a tendency to get caught (which is why it needed to be able to run for n cycles,) a bit of randomness was added. If the algorithm was caught for the square of the height of the grid, it would move a cell randomly in hopes to break out of the cycle. This actually worked, though the detection on whether or not the algorithm was caught could be better.

There were no new data types for this project, just the same grids and nodes used previously.

The program relied heavily on modifying global grids (test cases) instead of creating new local copies to solve the problem.