TANC

Trailblazing, Autonomous, Navigating Car

A maze runner. A pathfinder engine, solving foreign, looping mazes in shortest number of moves.

The project began with the idea of building a "*maze runner*" robot, but I soon got carried away with simulating the algorithm to make debugging easier. Now, I want to build a robust pathfinding engine for anyone who wants to build their own maze solving robot and needs an engine to be the robot's brains. Right now, the algorithm works, but it hasn't undergone any rigorous testing and I haven't identified any bugs that I haven't solved. It was a great learning experience implementing my own foreign maze solving algorithm, doing some optimization, and writing a large project, 99.99% in pure ISO C (0.01% C++).

In my algorithm I see hints of A* and depth-first-search. I implemented a heuristic rating and sorting function for the paths. **This algorithm unique is that it solves for looping mazes, that is, mazes in which there are any number of circular paths.**

Stats

$1,469$ **Lines of code**

56 **Functions**

3 **Incorporated Algorithms**

$30+$ **Hours spent on algorithm design**

$MAZEWIDTH * MAZEHEIGHT * (1 + 3 + 4)$ **Bits used in data structure**

$0$$ **Spent on new parts**

1 **Algorithm**

Algorithm

The goal:

Design an algorithm that can explore foreign, perfect and imperfect mazes and find a path from a given starting point to a given endpoint in the maze.

The algorithm uses a heuristic rating function that takes into account both how far it robot has traveled as of yet, and how valuable each path around it is. The robot is trailblazing and using specially designed data structures to store a history of the maze as it explores. While designing the algorithm, care was taken to optimize both for time and for space(use littlest RAM as possible).

The algorithm can be defined as:

1. Capture measurements to walls in all 4 directions
2. Caste raw measurement data to processed data pertaining to passageways capable of being understood by the robot, and store in history.
   3.
      1. If there are both unexplored and explored passageways surrounding robot:
         - Rate all 4 directions as a function of:
            - Whether path is already explored
            - Distance of end of path to endpoint
            - Distance from robot to end of path

      2. Otherwise, if the robot is surrounded by a dead end or finds that it is surrounded by passageways, use recursive depth first search to find the nearest unexplored passageway.

3. Go to that next passageway
4. GOTO step 1

**Solved maze:*

- `#` = Wall or assumed wall
- ` ` = Explored node
- `:` = Unexplored, assumed floor node
- `+` = Starting location
- `*` = Robot

```
6|#####:|
5|#####:|
4|    #:|
3|###   |
2|#*#+# |
1|# ### |
```

```
0|#     |
  012345

FINISHED. MAZE SOLVED
```

Designing a robust, optimally efficient algorithm was one of the most important aspects of the project. Sometimes it was optimized for speed and sometimes for space. The algorithm has flavors of:

- **A* maze traversal**
- **Insertion sort**
- **Depth first search**

Discussion

The TANC algorithm was designed to be an **abstraction** and domain independant. It was built as a maze running engine into which you can plug in data from any source and get reliable suggestions as to where a maze runner should go. This modularizes the entire project and allows for maze solving applications to be used anywhere, from on a robot to a simulation. Designing a complex algorithm is difficult to debug, so a simulation was built for testing.

Of course solving mazes is not all this project can do. A more complex iteration could save lives in a disaster zone, but is far from what we have now. While this algorithm is powerful and can handle even the most tricky mazes, we were not building it to be mission critical. It traverses foreign graphs and can find the shortest route between two points, but cannot be used in its current form for anything else.