University of Tromsø

You are likely to be eaten by a Grue

INF-2101 Algorithms

Mazes can be solved by a number of different methods. When tasked to do so using python I was able to create a decent labyrinth crawler that is able to find the goal.

Jonas
5/10/2012

1 Introduction

1.1 Overview

The goal of this assignment was to implement a maze-solving algorithm. This algorithm should be able to solve a given maze, within the time limit given.

1.2 Technical background

For this assignment we were to use Python, a language we have used numerous times before. Several files of precode were given, including the maze server as well as the basic functions for controlling the character on the server.

2 Design

Several design solutions were considered for this assignment[1], but all the best algorithms require knowledge of the whole maze. My choice landed on Trémaux's algorithm as it seemed like a simple solution with a moderate success rate, it can be described in brief:

> Walk straight forward until you hit a wall, turn to either side and continue to walk forward until you once again hit a wall. Repeat until you hit a dead end, and then backtrack until you reach a junction you have not yet fully explored and repeat the abovementioned.

Doing this ensures that you never visit any cell twice, you will however backtrack over it when you return from a dead end.

[1] http://www.astrolog.org/labyrnth/algrithm.htm

3 Implementation

This section will explain my implementations coarse feautures

3.1 LabyrinthWalker class

The main class used to solve the labyrinth, containing:

- A dictionary with information on all tiles visited and their adjacent tiles
- A list with information on all the last moves made
- Current position and which way you are facing

3.1.1 decideNextMove method

The “brain” of the implementation, it will make a decision on where to go next if there are any valid moves, if not it will backtrack. All the desitions made here are according to Trémaux's Algorithm.

3.1.2 move method

Will try to move in the direction decided by method 3.1.1. It will then read the output from the server and validate the move, if it was unsuccessful it will check the surrounding so the next move will be based on better understanding of the area.

3.1.3 backtrack method

When the walker reaches a dead end it needs to backtrack, this is done by popping the last movement from the path list and return the way we came.

3.2 Test

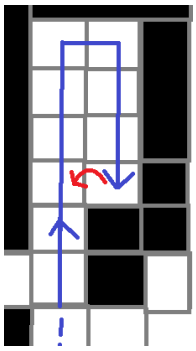
Tested in Linux Mint 12 (32-bit) using python (2.7.2+)

To run the program I point at the src folder and type

```
$ python labyrinth.py
```

4 Discussion

The solution is working as intended according to my anticipations. The code however is not ideal, as it can end up using a fairly hefty amount of moves exploring dead ends. This could have been solved by scanning the surroundings more and finding shorter backtracking paths, for example when we have a loop it could simply walk one tile instead of backtracking the whole loop, see illustration.



5 Conclusion

The algorithm works as intended according to the goal I set for my solution.

6 Sources

http://www.astrolog.org/labyrnth/algrithm.htm