TDDBUDDY.COM

Maze Walker

Rules

1. Strictly practice TDD
    a. Follow the: Test, Implement, Reflect, Refactor cycle.
    b. Be sure to run your test after each reflect-refactor cycle.
2. Practice wishful coding
    a. Do not write any production code until you have a failing test.
    b. Or compilation failures to drive the need to implement the method.
3. Use your TDD Cube and Reference card if you have, be-sure to cycle the cube with each step

The Kata [1]

You are given a program which is capable of traversing 2D mazes. However, the solution isn't perfect and fails to find the exit in certain types of mazes. Moreover, the current solution is messy, doesn't follow any sensible coding principles (e.g. SOLID) and is completely untested.

The program takes input from a text file representing the maze.
There are two exmaple mazes provided in the repository https://github.com/T-rav/CodeKatas.git

- *maze1.txt* is an example of a maze where the current solution succeeds.
- *maze2.txt* is an example of a maze where the existing solutions fails to find the exit.

Currently the program outputs the x,y position of the dumb maze walker at each step.

Refactoring Work[2]

Your goal is to modify this program to work with all the example mazes in two steps:

1. Refactor the code.
    a. Ensure code is covered by unit test before refactoring
    b. It should be easy to read, follow the SOLID design principles
2. Branch by abstraction and extend the existing maze walker to find the exit in both example mazes.
    a. Ensure strict TDD is followed
    b. Enable toggling between the two implementations at runtime

Constraints

- The only valid movements in the mazes are up, down, left and right. You can't move diagonally.
- Both maze walkers, new and old, should be demonstrably correct through unit testing.
- You can edit the DumbMazeWalker class, but you must not change the way in which it ultimately traverses the points in the maze. You can change the output function if you want, though.
- You must not change the technique the existing DumbMazeWalker uses to traverse the maze, i.e. the existing maze walker must traverse the maze in the same sequence of steps after your refactorings.

Bonus

Make it easy to see what's going on with your new maze walker. Example output in *output1.txt*. Currently the program only outputs the x,y position of the dumb entity at each step.

---

[1] Kata sourced from Redgate's Blog : https://www.red-gate.com/blog/building/code-kata-2-amaze
[2] The kata has been modified to include branching by abstraction