Introduction to Artificial Intelligence

Assignment 1 report, 26.10.2022

Author: Marat Gubaidullin
Group: BS21-02

Code

Source code

Algorithms

A*:

- For this assignment, I have implemented a basic A* algorithm, which represents the map as an NxN node map. Each node has the following parameters: a - actual cost, p - potential cost, sum = a + p, x coordinate, y coordinate, cell - type of cell (empty, Jack, Chest etc.), list of parent nodes. I have implemented a node as an A_star_node class, and Cell as an enumeration. When a map is set and the algorithm starts running, it initializes the nodeMap, potential cost of every node and the actual cost of the very first node. Then, it tries to find the shortest path in two different ways (that is where my first improvement of the algorithm starts).
- First way is to try to reach the chest without visiting Tortuga and killing the Kraken. For each node from the area of the current node, the algorithm counts its actual cost via incrementing the actual cost of its parent (current node or other "better node" from which we have visited this node earlier and which gives it better actual cost => better sum). Then the algorithm adds the neighboring node in a list sorted by variable sum, the next node will be the least expensive node from that list. Not passable nodes get a > 1000, which helps the algorithms to understand if the map is not solvable (if the first element of the sorted list has a > 1000, the map is not solvable, at least via the first way).
- The second way implies solving the map via visiting Tortuga and (potentially) killing the Kraken. It runs twice, from the Start node to Tortuga, and then from Tortuga to Chest. If the algorithm finds the Chest before visiting Tortuga, it breaks, because it means that the path, found by the first way, will always be more efficient than the second.

Backtracking:

- The backtracking algorithm utilizes a similar idea. The algorithm recursively checks all possible paths from the start to the destination, which in the first path is Chest, and in the other is Tortuga, then Chest. I had to limit the maximum number of steps to the destination and the number of recursion iterations, because, otherwise, it causes infinite loops. This made backtracking much less efficient than the A*.
- As an improvement, I have added a dynamically reducing maxSteps variable: when the algorithm finds a way to the destination, the maximum number of steps for all other iterations will become the number of steps needed to reach the destination.
- As another improvement, I have added the method "AreaOptimization". The coordinates of the current and destination cells are passed to this method, and the method returns a list sorted by "distance" from a neighborhood cell to the destination cell, that contains the coordinates x and y. The algorithm chooses the cell for the next recursive iteration from that list.

Statistical Comparison

Stats:

A*:

- Wins: 918
- Loses: 82
- Win rate: 91.8%
- Mean: 0.5615652 ms
- Mode-Median: 0.5304 ms
- Standard deviation: 0.522362 ms

Backtracking:

- Wins: 918
- Loses: 82
- Win rate: 91.8%
- Mean: 45.7508428 ms
- Mode-Median: 1.73595 ms
- Standard deviation: 287.593142 ms

Comments:

- W/L results of both algorithms are the same.
- A* algorithm performs approximately 9 times faster than Backtracking. Also, the Standard deviation of the Backtracking algorithm is approximately 6 times larger than the mean, which means that on some maps Backtracking shows a very poor performance.

PEAS

- **P**erformance measure - Time that the Algorithm takes to find the path.
- **E**nvironment - Map.
- **A**ctuator - Ability to move, loot the Tortuga and the Chest, kill the Kraken.
- **S**ensor - Ability of the actor to see the danger/loot before interacting with it.

Impossible maps

- Maps, where Davy and Kraken both prevent Jack from getting the Chest or Tortuga. example
- Maps, where Davy and Rock prevent Jack from getting the Chest. example
- Maps, where Davy or Kraken spawn near Jack and he dies immediately. example