Multi-agent system for solving a maze

Students: Maria Noaptes, Mohammed Skouti

Requirement

Implement a maze-solving algorithm using a multi-agent system. The maze is an n x m grid of cells, some of which can be passed-through (the "floor"), and some of which cannot (the "walls"). Multiple agents try to map out the maze and find an exit. During the search for the exit, each agent behaves as follows:

- Each agent keeps a map of the maze where for each cell the agent assigns 4 weight values, one for each direction: up, down, left, right. Initially, the directions leading into “floor” cells have a weight of 1 and the directions leading into “wall” cells will always have a weight of 0. - The weights are in the range [0, 1]. A weight of 1 means that the agent may freely move in
the corresponding direction, while a weight of 0 means the agent will always avoid moving in that direction. The lower the weight of a direction, the less likely that the agent will choose to move in that direction.

- An agent chooses the direction with the highest weight that isn’t occupied by another agent. - If there exist multiple directions with the same weight, the agent will choose a direction not travelled by it (i.e. it will choose not to turn back). - Whenever an agent moves in a certain direction, they will broadcast this direction to the other agents. The others will slightly decrease their own weights for that direction. This means that an agent will be less likely to travel to cells that have already been explored by others.

- If an agent encounters a dead end, they will turn back until they can once again move in multiple directions. They will set the weights of the directions leading to the dead end to 0 and they will inform the other agents accordingly.

- At some point an agent will discover a cell marked as “exit” and will broadcast its position to the others. The other agents will move to that position so that each agent exits the maze. Run the application for multiple numbers of agents. Compare the time it takes for the maze to be solved by a single agent, or multiple collaborating agents, and present your findings in the documentation.

You may use a predefined maze or generate it procedurally. For procedual generation, the easiest method is Prim’s algorithm for generating perfect mazes (perfect maze = the is a unique path between any two positions).

A simple visual representation of the maze and the moving agents would be helpful in assessing the execution of the application.

Interface

The interface is a maze generated from a matrix with zeroes and ones, 1 being a wall cell and 0 being a floor cell. There is only one entry and one exit.

The explorer agents are represented by circles with randomly generated colors that start from the entry and try to find the exit.

The planet agent is responsible with rendering the nodes and synchronizing the explorers (every agent should know the exact position of the others so the agents move one at a time).

Messages

Explorer to explorer

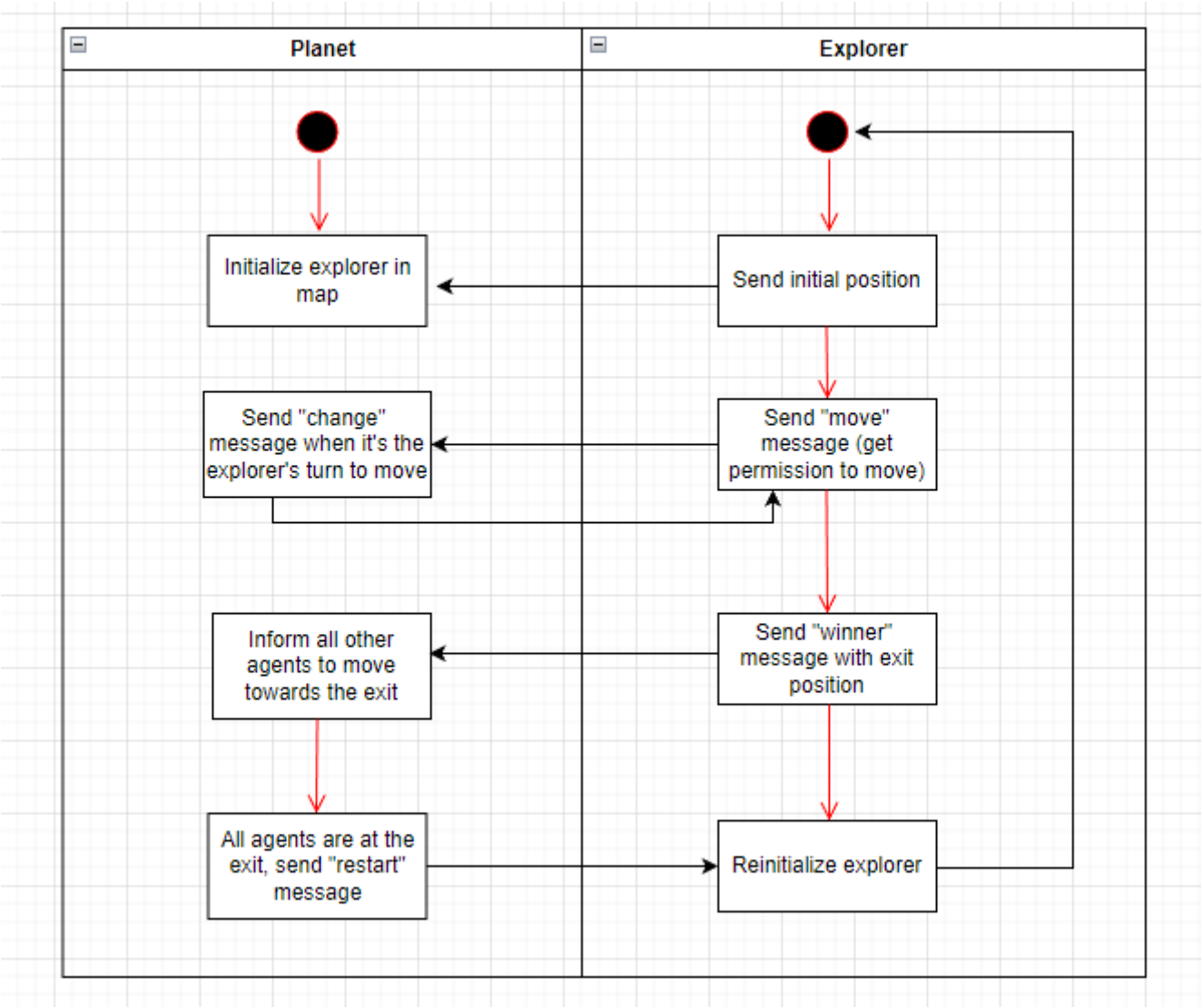
- "change x y w": used to notify other agents of explored positions and weights (x and y are coordinates, w is the weight)

Explorer to planet

- "move x y": to change the coordinates x and y stored by the planet for an agent (to finally render it in other position)
- "winner x y": first agent to get to the exit sends a message winner with the coordinate of the exit

Planet to explorer

- "position x y": initial message with the position of an agent
- "restart": it is used to make another round to return all the variables to the original state and to calculate the times and steps again
- "winner x y": in the event of reaching the end, the winner's message is used to remove the agent from the map, to send the coordinates of the end and to inform others of the route to be taken



Activity diagram

Movement and synchronization

Every time an agent wants to move, it informs the planet by a “change” message. The planet puts it in a queue if other agent is moving and sends it a message “move” when it’s his turn to move. Along the “move” message it also sends the exact location of all the other agents
(an explorer should know the obstructed paths before choosing its next position).

When it’s his turn to move, the agent has to choose between the directions that are not obstructed by a wall or by another agent. To choose the best path, it keeps a dictionary with cells that were visited and their weight. The weight starts from 0 (not visited) and are increased with 1 every time the cells are visited. The explorers broadcast their positions to all the other explorers when they move so they increase their weights too. By choosing the minimum from the weights of the possible directions, each explorer will preferably go through less explored paths.

Dead ends

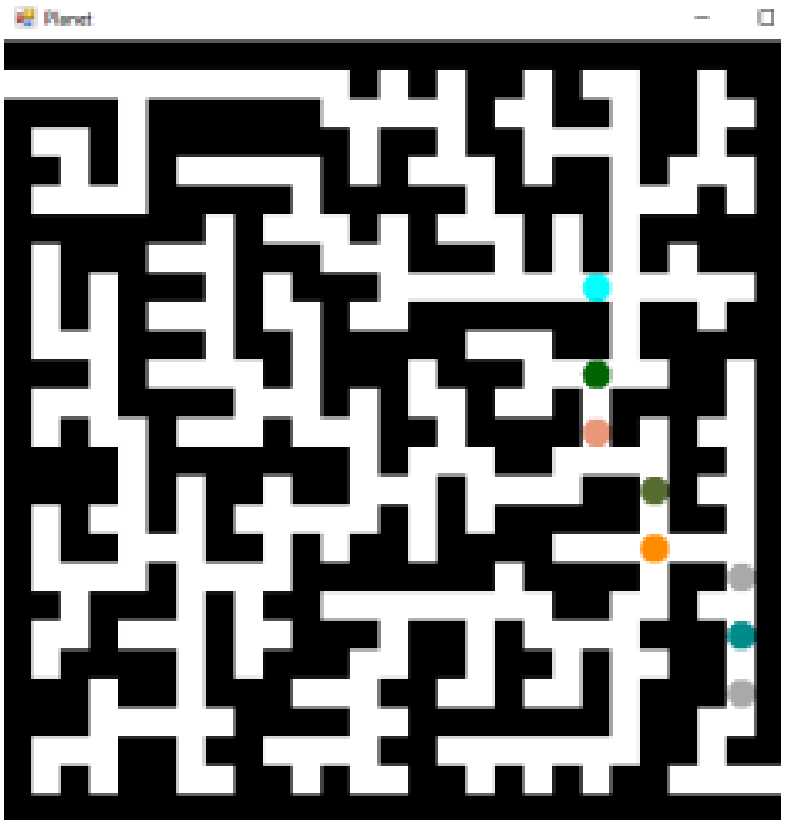
The cells that have 3 neighbouring walls will be considered dead ends. This means that the path that got the agent there should be avoided in the future. The way we ensure this is by increasing the weights of the cells preceding the exit from the dead end with 5 instead of 1. The exit from the dead end is a cell with 3 neighbouring floors.

Examples of dead ends

Encountering an exit

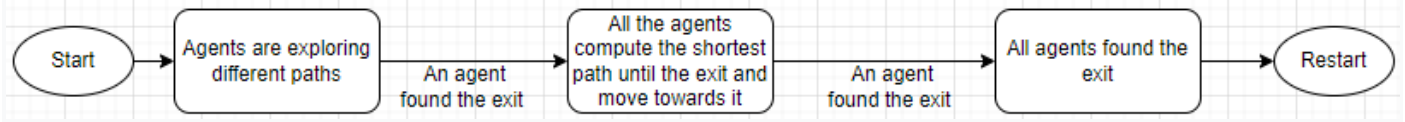
When an explorer finds the exit it broadcasts its position to all other explorers by a "winner" message. They will then compute the shortest path until the exit using a modified version of BFS (https://www.geeksforgeeks.org/shortest-path-unweighted-graph/).

The graph with nodes and edges will be obtained from the keys in the dictionary with explored positions (weights dictionary). After computing the shortest path, all agents will move to the exit.



All agent move towards the now known exit

The basic flow of the program is shown in the figure below:



Findings

We run multiple iterations of the program to perform the average of moves and times but we saw that for the same maze and number of agents it does not change (movement is based on rules, it's not random).

Nr. Explorers	Maze size	Elapsed time (ms)		Nr. of moves	
		First agent exits	Last agent Exits	First agent exits	Last agent exits
1	27*27	26635	26636	650	650
5	27*27	26905	33407	185	232
10	27*27	44829	56683	184	239
20	27*27	80946	144450	138	198

By comparing time, which is dependent on the hardware the program is run on, we can conclude that the time for solving the maze is bigger for a greater number of agents. This is because of the synchronization step for avoiding collisions (one agent moves at a time and informs all the others of his position, no other agent is moving until it's his turn).

Knowing this, we also counted the number of moves every agent does until the exit and displayed it for the first and the last agent.

Conclusion: A bigger number of agents in the maze can explore more paths than a smaller number, thus the program performs better with more agents (the exit is found with fewer steps for the first agent that encounters it).

Running the program

No prior configuration is needed to run the program. The executable is at Reactive/Reactive/bin/Debug/Reactive.exe.

Responsibilities of each member of the team

Mohammed Skouti: Interface, movement of the agents (avoiding walls and occupied positions, staying in the maze), movement synchronization of agents with a queue, dead ends, writing times and number of moves in a file for first and last agent

Maria Noaptes: keeping a dictionary with weights determined by current explorer's moves and messages received from other explorers, sending and receiving exit position, different behavior when all agents know the exit (calculating the shortest path, following it), reiteration of the program when all the agents found the exit

Both: Documentation