Maze Simulation

Riccardo Caprile (4370774)

4370774@studenti.unige.it

Heading 1: Introduction

Heading 1.1: Netlogo

NetLogo is multi-agent programmable modelling environment for simulating complex systems. Programmers can give instructions to hundreds of independent agents all operating in parallel.

The NetLogo world is made up of agents and the can follow instructions. Each agent can carry out its own activity , all simultaneously. In addition to agents called Turtles , there are Patches which are a square piece of ground over which turtles can move and Observer which does not have a location but can be imagined as an entity looking out over the world of turtles and patches.

With NetLogo is possible to explore the connection between the micro-level behavior of individuals and the macro-level patterns that emerge from the interaction of many individuals.

Easy to understand and to develop new environment and agents.

NetLogo provides a community place where it is possible to upload your work and ideas. Looking through it , there were no application like the one I am going to explain here. There were for example : a random maze generator (not useful for my experiment) or games where you have the possibility to solve the maze using your abilities.

NetLogo Version 6.3 was used.

Heading 1.2: Maze Simulation

The principal goal is to create an autonomous agent which is able to find a way out of the maze.

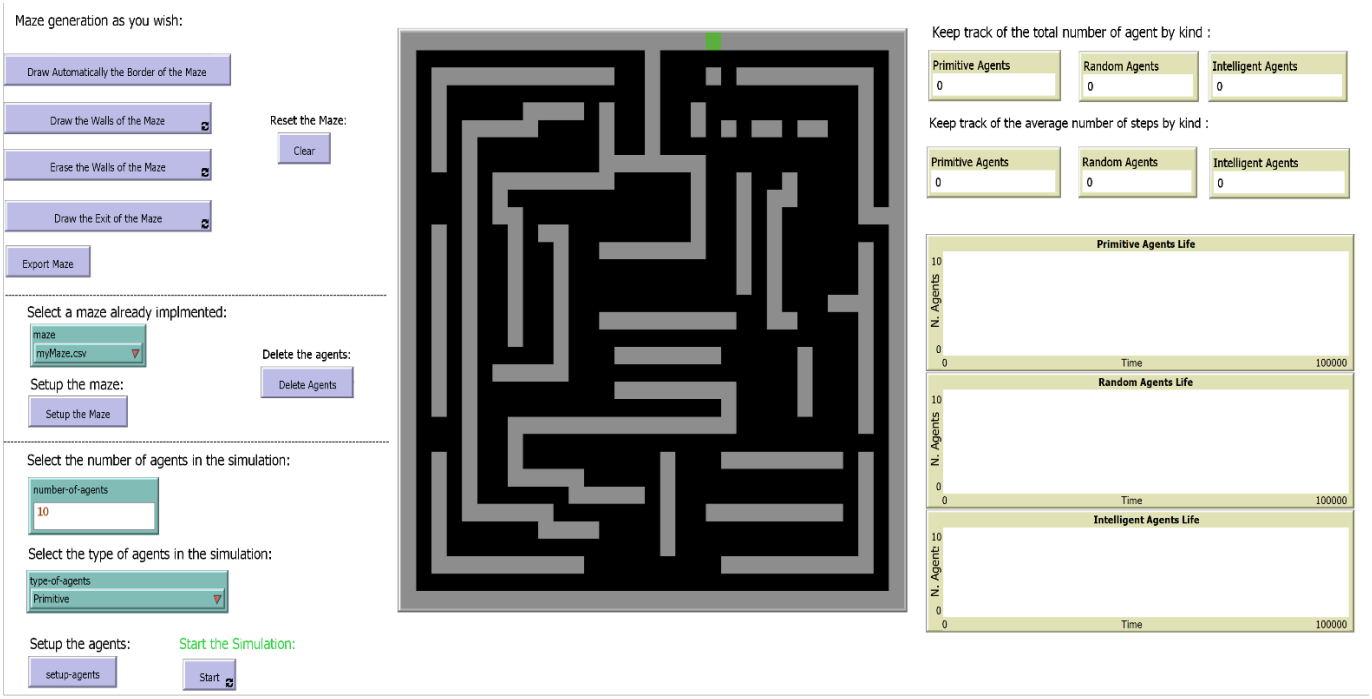
The agent must , also , be able to find the solution in a finite time without getting stuck in endless loops.

In this simulation , the environment we are considering is maze. The agents can be meant as “people” that are trying to escape from the maze.

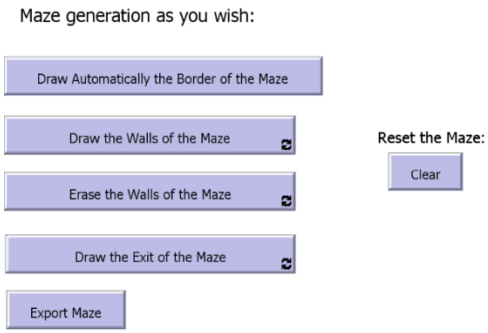
There are three type of agents , that follow three different algorithms. In the NetLogo application they are called : Primitive Agents , Random Agents and Intelligent Agents. We will see later in the report how they are implemented and how their strategies work. There are three types of patches , too : the first one represents the walls of the maze and its color is grey , the second one is the ground where agents can walk and its color is black and the last one is the exit of the maze coloured in green.

Heading 2: Implementation

This the overview of how the NetLogo application looks like. Now we go deep into the implementation of all of the different parts that compose the application: Maze Generation , Maze import and agents set up and the monitors tracking.



Heading 2.1: Maze Generation



I gave the possibility to the users to create their own maze as they wish. In this way , there is the possibility to simulate the application using different mazes with different characteristics , varying from very complex mazes to simpler ones.

The first button is **“Draw Automatically the Border of the Maze”** that set the color of all the patches, that represent the border with grey. With this button the generation of the maze is going to be quicker.

Then , the second Button present in the application is **“Draw the Walls of the Maze”**. It lets you draw the walls of the maze in the environment.

This was possible attaching to the button the procedure draw-gate. It asks to the patches clicked by the user to set their color to grey. In this way we can draw all the walls of the maze.

In addition to this , the third Button is **“Erase the Walls of the Maze”** . It lets you erase the walls created with the button explained before.

It works with the procedure erase-wall that works in the exact same way as before, but it colors the patch with black.

The next button in the application is “**Draw the exit of the Maze”.** It lets you draw the exit of the maze the agents are looking for.

The procedure draw-gate do the job, it let the user draw a green patch where he clicked. The user can only draw only one exit. If he tries to draw 2 of them , everything will remain as it was with only one green patch as exit.

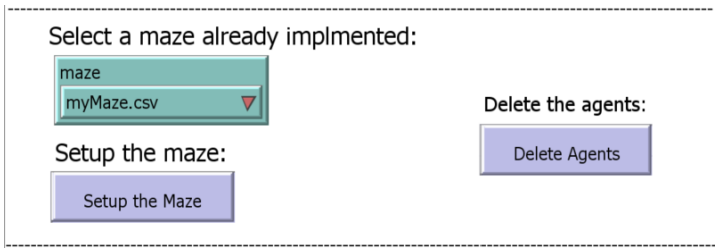
The second-last is the "**Clear"** button that will delate anything inside the environment.

The last button for the Maze Generation is "**Export Maze".**

It gives the possibility to export the maze as a myMaze.csv. In this way , the maze will not be lost and you can use it for future simulations.

It works using the export-maze procedure. Inside of it there is another procedure , which is export-world provided by the NetLogo library and the local variable filepath that represent the path where the .csv file is saved.

Heading 2.2: Maze Import and Setup

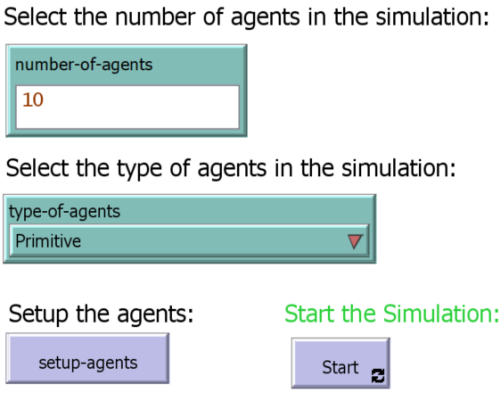


Basically , when we clicked the "**Setup the Maze"** Button , the maze selected in the chooser "maze" (this is a global variable) will appear and be ready to welcome the agents.

myMaze.csv is the maze exported in the step before.

There are more mazes created by myself for making the simulation more interesting called myMaze2 , myMaze3 , myMaze4 , myMaze4. We will see later in this report how they looks like and how agents behave inside of them.

Heading 2.3: Agents Setup

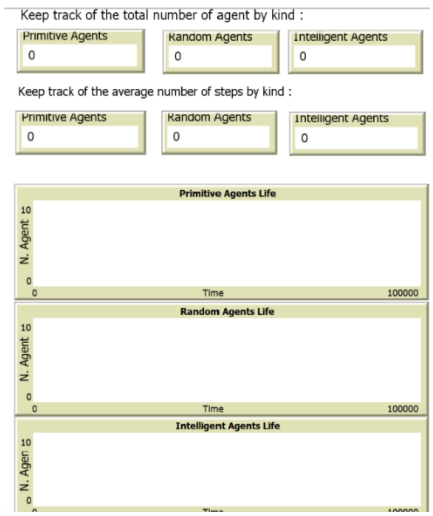


In this part of the application , we can select how many agents we want inside the simulation and we can , also , decide which kind we want to try to solve the maze. There is the possibility to select a huge number of agents , but in this case the application will be inevitably slower. In the type-of-agents we have many options of deciding which agent we want in : Primitive , Intelligent , Random , Primitive Random Intelligent , Random Primitive , Primitive Intelligent.

After these decisions we have to clicked the "**setup-agents"** button. It uses three different procedure. They create the agents basing on the global variable type-of-agents and number-of-agents. All the three procedure will position the turtles at the bottom left part of the screen with three different shapes and three different colors.

After that it is possible to start the simulation by pressing "**Start"** Button

Heading 2.4: Monitors



In the end , we have three different monitors.

The first tells us how many agents by type are still inside the maze trying to solve.

The second monitors tells us the average steps that agents needed to solve the maze. (For every step taken a variable is increased)

The last one is a graphical monitors that tells us the number of robots remaining in the maze during the time elapsed. On the x-axis we have the time and on the y-axis the number of robots.

Heading 3: Agents

Now , we can understand the three different algorithms the three kind of agents are using.

Furthermore , in the “Start” procedure there are the three different algorithm that make the agents walk : Primitive-step , Random-Step and Intelligent-step.

Then there is another procedure called step-solved that will let the agents go inside the exit and solve the maze.

The last procedure inside of it is maze-solved that will kill the agents if the patch where it is , is green. This means that he was able to get out of the maze .

Heading 3.1: Primitive Agents

This is the most simple agent because the behaviour algorithm they have is very basic.

Basically they move straight if they do not encounter any obstacle (any grey patches) otherwise if they encounter an obstacle they will turn left. This kind of algorithm will obviously get the agents stuck in endless loops and they will not be able to solve the majority of mazes. Their behaviour is implemented with Primitive-step procedure.

Heading 3.2: Random Agents

The second type of agent is the Random Agent. He works similarly to the Primitive agent, but I wanted to add some randomness. Basically it will go straight if he does not encounter any obstacles

, otherwise if he encounters one it will randomly turn left or right. We will see that this solution is better than the previous one.

Heading 3.3: Intelligent Agents

The last type of agents implemented is the Intelligent Agent.

It uses the Left Hand Rule Algorithm that tells : always turn left if you can. If you cannot turn left , go straight. If you cannot turn left , or go straight , turn right. Left Hand Rule is good for all the mazes that do not contain endless loops. This robot is meant to be able to solve mazes that contains endless loops. For this reason , to the algorithm has been added a pinch of randomness. In one percent of cases the robot will determine its rotation using a random number generator. In this way , it is able to solve all the mazes.

Heading 4: Results

Now I would like to show all the results obtained with all the mazes and all the agents.

Simulation done using 100 agents by type.

	Primitive Robots		Random Robots		Intelligent Robots	
Maze	Robots Left	Avg Steps	Robots Left	Avg Steps	Robots Left	Avg Steps
myMaze	100	53183	0	1036.34	0	864.82
myMaze2	100	82722	53	45189.86	0	704.25
myMaze3	100	48118	100	48118	0	10170.34
myMaze4	100	33102	100	33102	0	180.18
myMaze5	0	138	0	67.4	0	94.7

Heading 5: Conclusion

The results table shows us clearly how the algorithms works in every maze. The primitive one was not able to solve all the mazes except the last one that was the easiest maze. The random algorithm was able to solve some of them but if the agent get stuck in an endless loops he was not able to get out of the maze. The intelligent algorithms was pretty effective completing all the mazes.

I think that the application worked quite nicely and showed which algorithm you should use when you are trapped inside a maze.

In addition to this , given the possibility to create the maze as we like we can experiment thousands of different situation where we can see how the agents behave. It is , also , very easy and interesting implement more type of agents that use different algorithms (eg. Right Hand Rule , or an agent which make a step completely randomly)