Implementation of the RRT Algorithm

Scurtu Jasmin
Firtala Maria
Codarcea Alexandru

Content

1. Introduction
2. The Algorithm
3. Improvements
4. Interactive Menu

Introduction

The RRT (Rapidly-exploring Random Tree) algorithm is a path planning method used in robotics and artificial intelligence to find accessible routes in an obstacle-filled space.

RRT builds a random tree starting from the initial position, extending to randomly chosen points, with a constant distance. The goal is to connect the starting position with a target position, avoiding collisions with obstacles in the environment.Acesta este utilizat in aplicatii precum:

- Video Games
- Mobile robot navigation
- Road planning for autonomous vehicles

The Algorithm - main steps

1. We initialize the tree: we create the start node, which we set as the root, and the target node

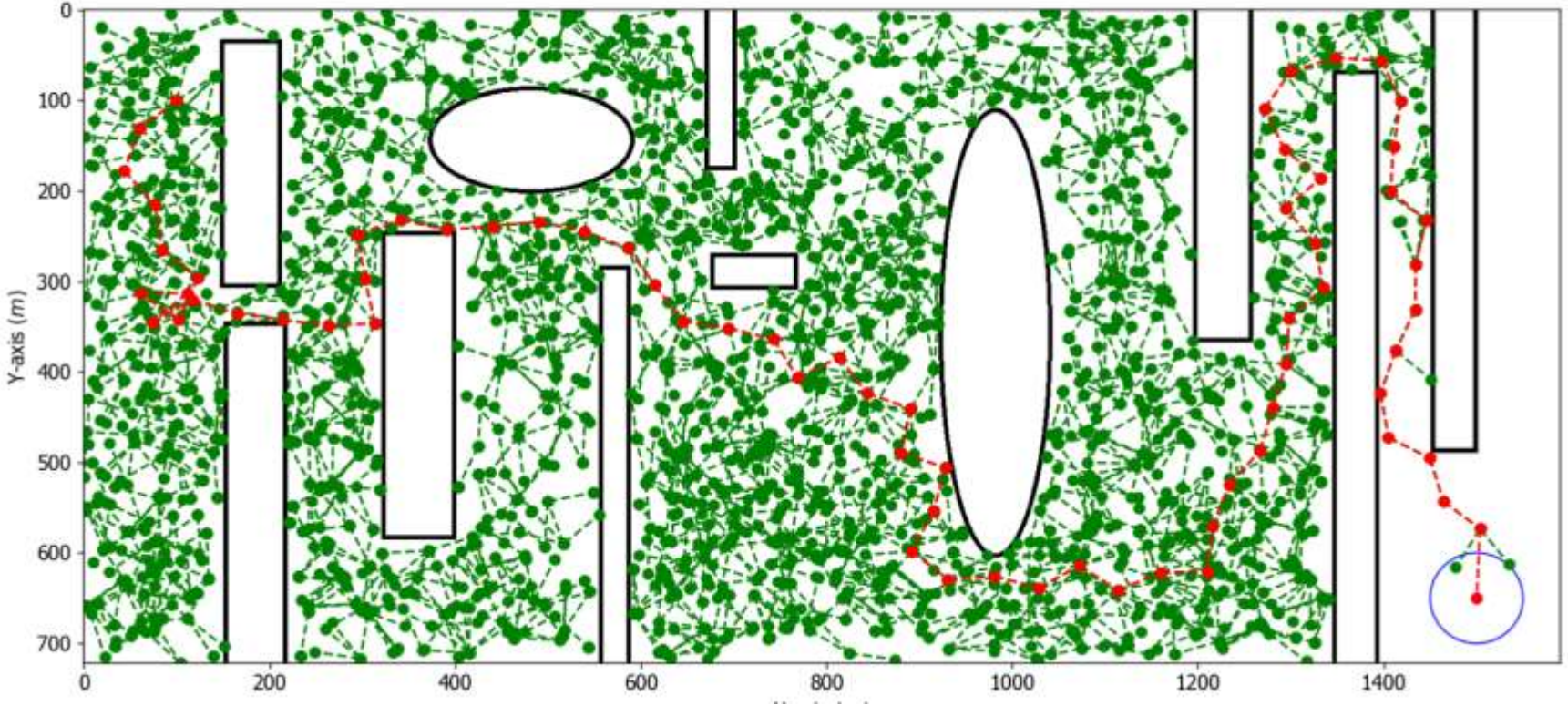
2. For each iteration:

   a. Generate a random point on the map
   b. We are looking for the closest node already existing in the tree to the randomly generated point
   c. We extend the tree towards that point by a predefined distance
   d. We add the node to the tree as a child of the nearest node
   e. If the node is in the target region, it connects to the goal and stops searching.

3. We reconstruct the final path, walking the tree in reverse, from the goal to the start

4. We display the number of waypoints and the total distance of the route

Classic implementation



I. Improvements made

Classic implementation

- If on the route between the nearest node and a point oriented towards the random point, at a distance stepSize, there is an obstacle, the algorithm **does not add any points** and **an iteration is lost.**

The version implemented by us

- In this case, the algorithm **places a node at the edge of the obstacle**, on the calculated trajectory, thus avoiding **wasting an iteration** and allowing the tree to expand continuously.

II. Improvements made

Context: When the distance between the random point and the nearest node is **less than the preset distance (stepSize)**, the algorithm will still try to **go exactly stepSize** in that direction.

Classic implementation

- If the movement with **stepSize** would lead the point **off the map**, the algorithm **does not force exit** and wastes an iteration.

The version implemented by us

- In this case, it **calculates the maximum possible position on the edge of the map**, in that direction, and **places the node there**.Thus, the tree continues to expand continuously.

III. Improvements made

Classic implementation:

- The random points are completely random, and the algorithm can "wander" through space too much.

To speed up the approach to the target, we introduced a **strategy**:

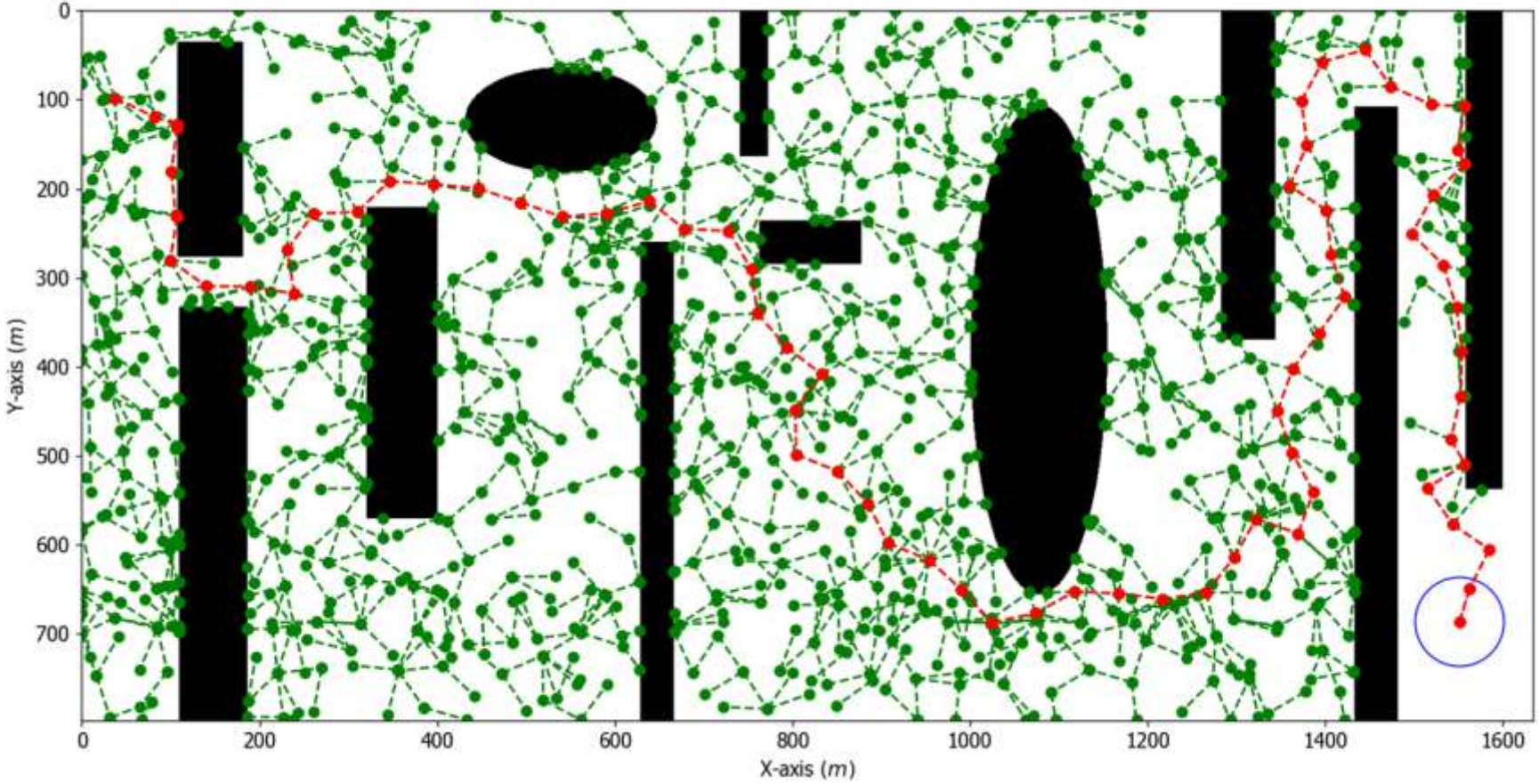
Once every 10 iterations, the randomly generated point has :

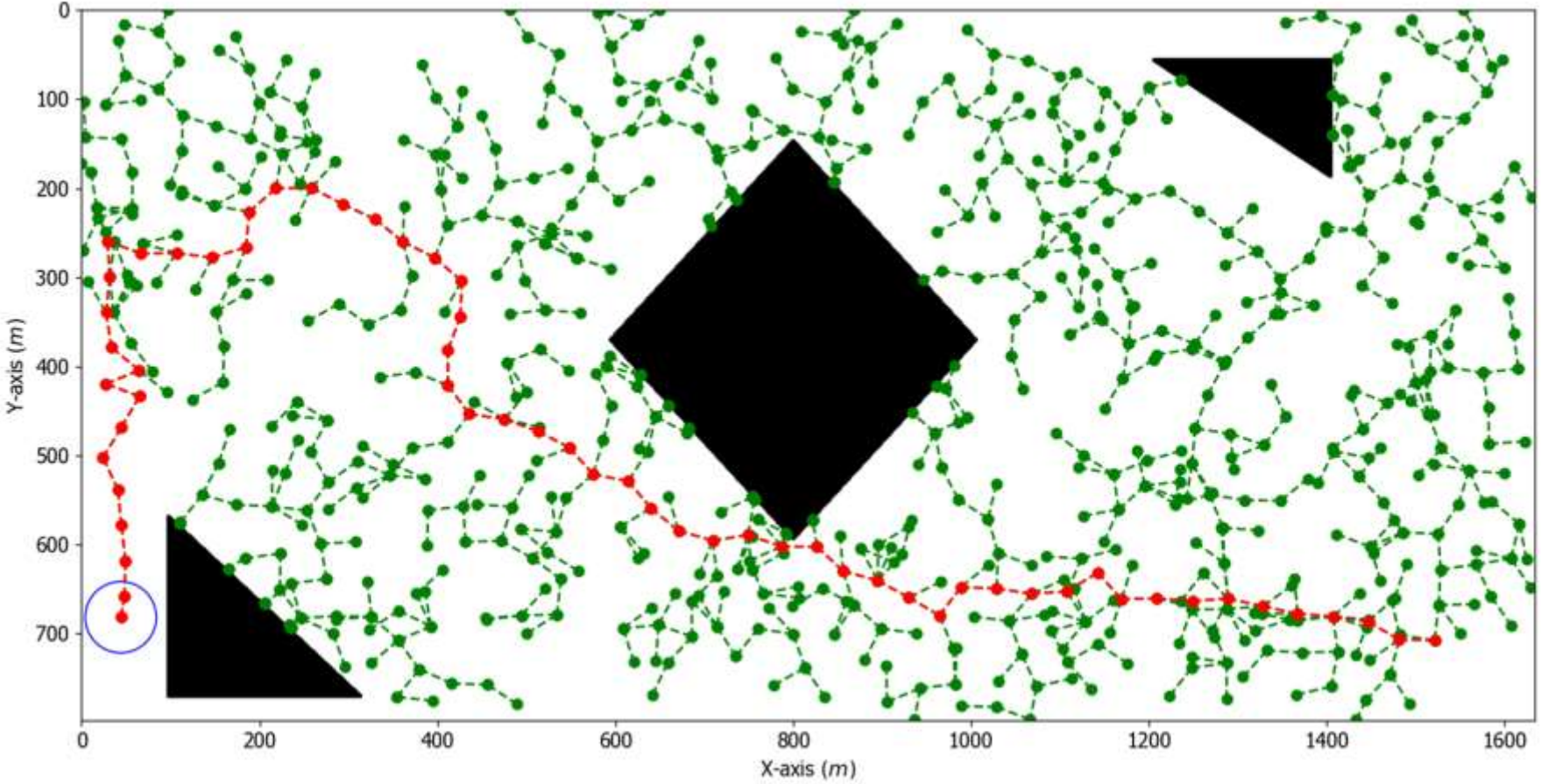
- **coordinate y of the target (goal)** and a random x, or **coordinate x of the target** and a random y

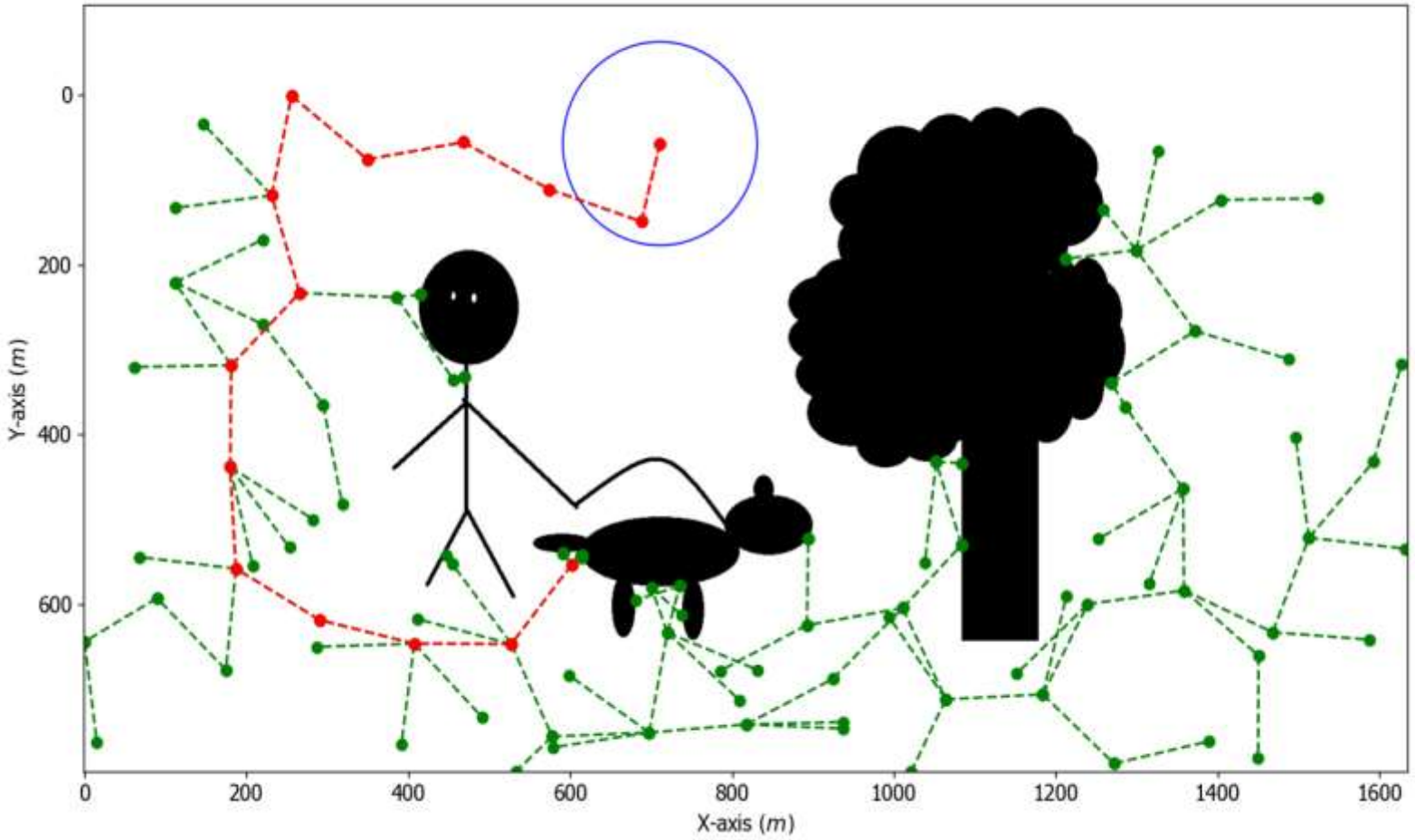
The purpose of this technique:

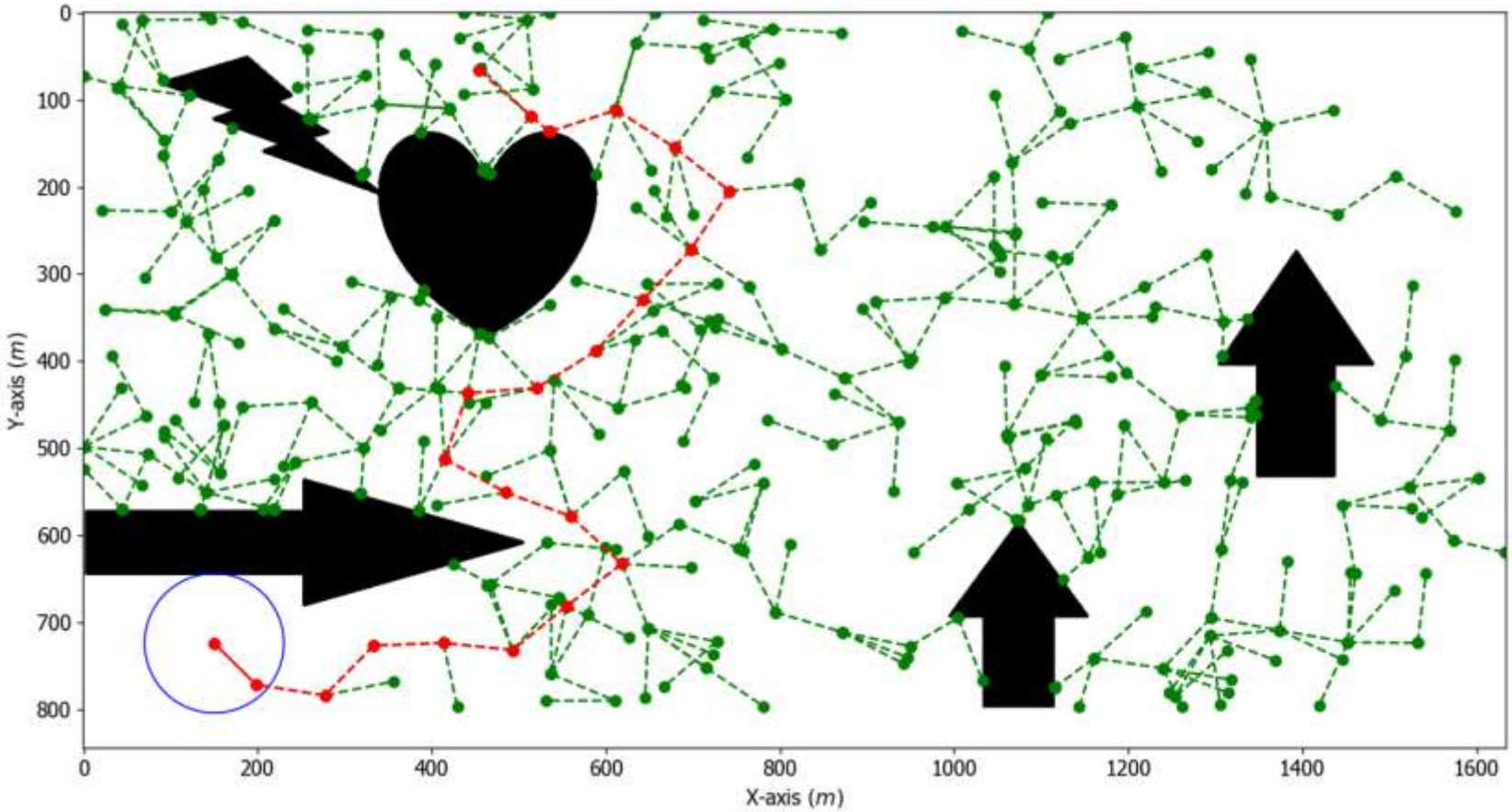
- **Increases the chance** that the tree will head towards the target area
- **Reduces the number of useless steps i** in other directions
- Helps the algorithm to **find a way faster**

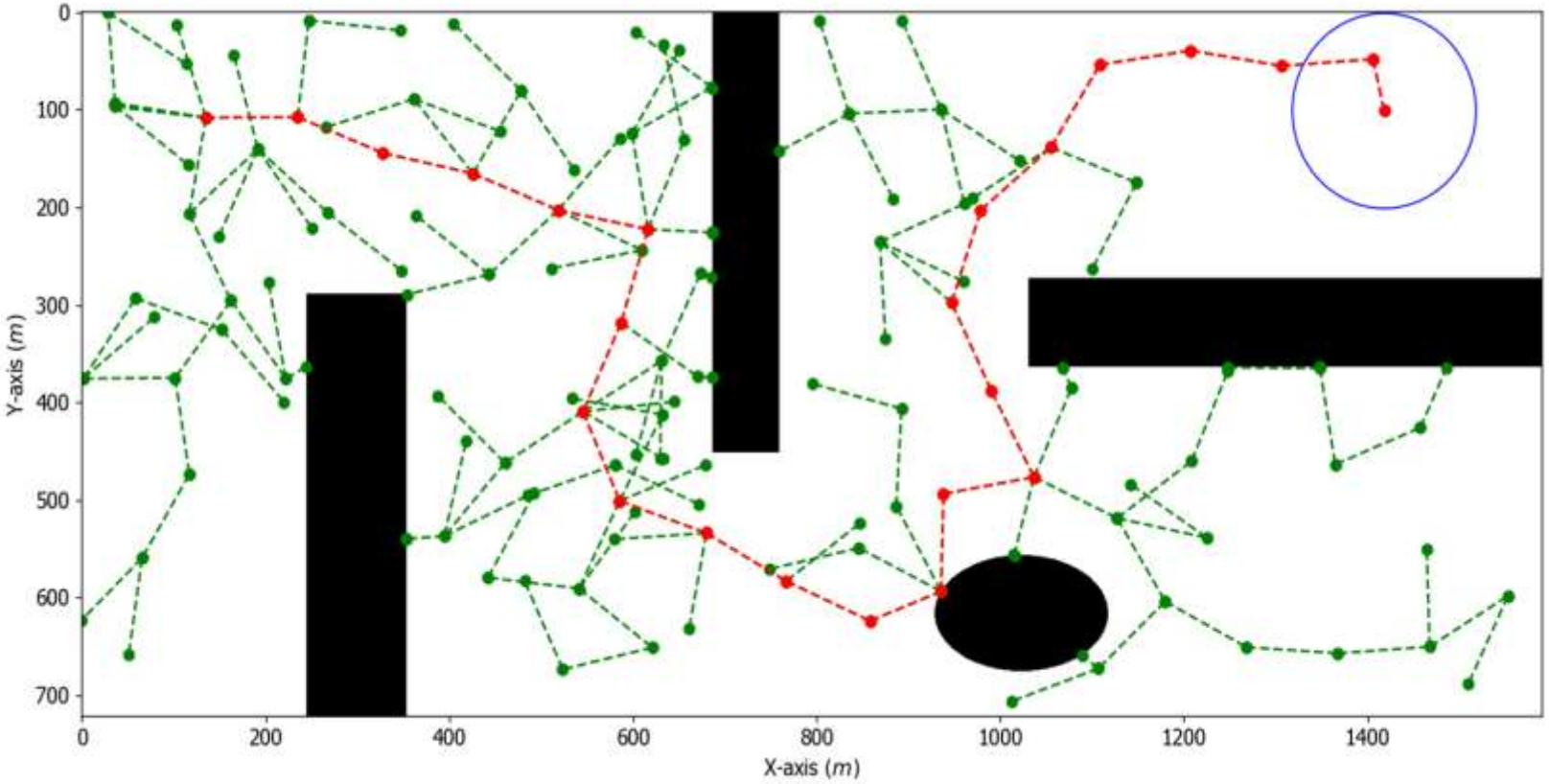
The version implemented by us




0
100
200
300
400
500
600
700
Y-axis (m)
0
200
400
600
800
1000
1200
1400
1600
X-axis (m)


0
200
400
600
Y-axis (m)
0
200
400
600
800
1000
1200
1400
1600
X-axis (m)


0
100
200
300
400
500
600
700
800
Y-axis (m)
0
200
400
600
800
1000
1200
1400
1600
X-axis (m)


0
100
200
300
400
500
600
700
Y-axis (m)
0
200
400
600
800
1000
1200
1400
X-axis (m)

Interactive Menu

We added a user friendly menu through which we can select a different grid to test the performance of the algorithm.

```
Select a number between 1 and 11 to load a grid.
Choose 0 for a random grid.
Enter your number:
5
```

Thank you for your attention!