Implementarea Algoritmului RRT

Scurtu Jasmin
Firtala Maria
Codarcea Alexandru

Cuprins

Introducere

Algoritmul RRT (Rapidly-exploring Random Tree) este o metoda de planificare a traiectoriilor folosita in robotica si inteligenta artificiala pentru a gasi rute accesibile intr-un spatiu cu obstacole.

RRT contruieste un arbore aleatoriu pornind de la pozitia initiala, extinzandu-se catre puncte alese aleatoriu, cu o distanta constanta. Scopul este de a conecta pozitia de start cu o pozitie tinta, evitand coloziunile cu obstacolele din mediu.

Acesta este utilizat in aplicatii precum:

- Jocuri video
- Navigatia robotilor mobili
- Planificarea drumului pentru vehicule autonome

Algoritm - pasi principali

1.Initializam arborele: cream nodul de start, pe care il setam ca radacina, si nodul tinta

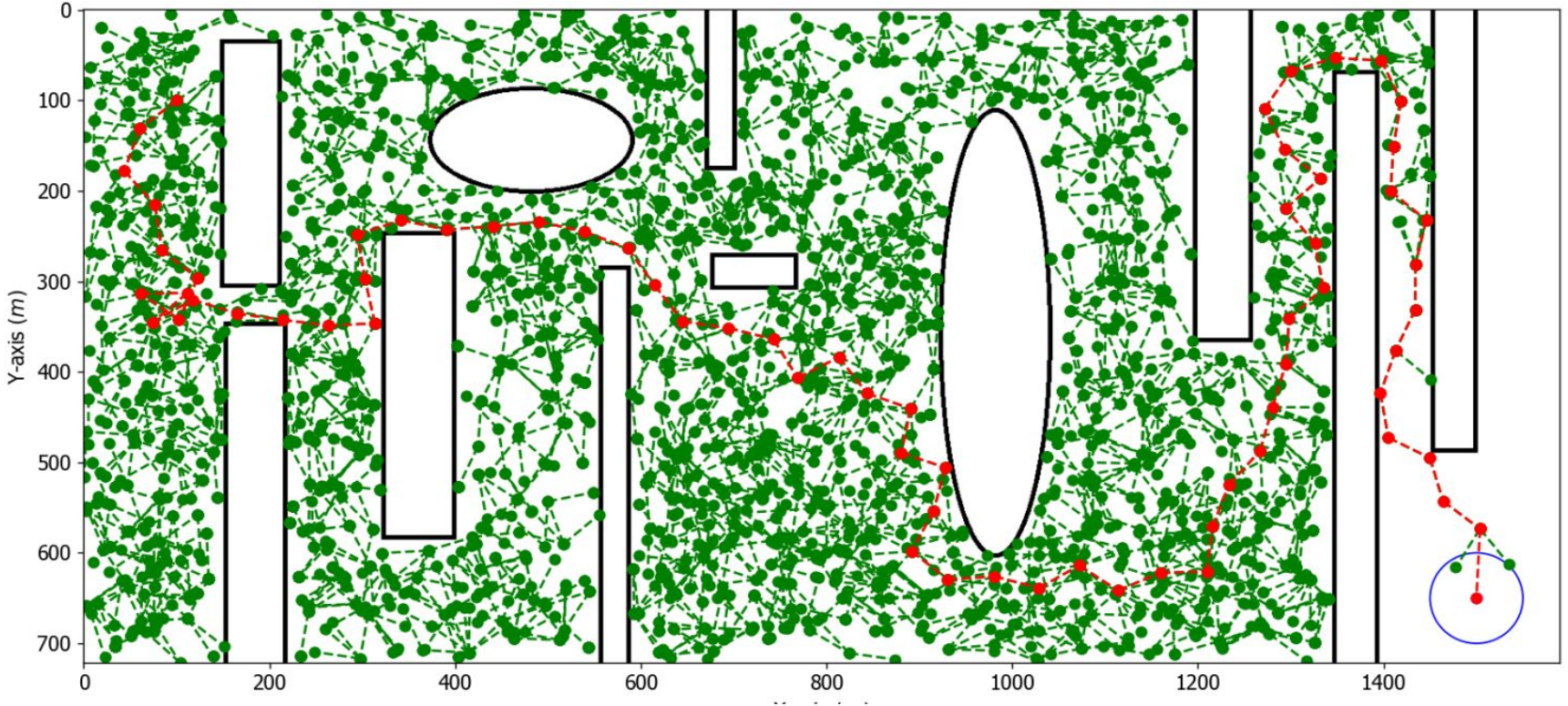
2. Pentru fiecare iteratie:

a. Generam un punct aleatoriu in harta
b. Cautam cel mai apropiat nod deja existent in arbore fata de punctul random generat
c. Extindem arborele spre acel punct cu o distanta predefinita
d. Adaugam nodul in arbore ca fiu al celui mai apropiat nod
e. Daca nodul este in regiunea tinta, se conecteaza la goal si opreste cautarea

3. Reconstruim traseul final, parcurgand arborele invers, de la goal la start

4. Afisam numarul de waypoints si distanta totala a traseului

Implementare clasica



I. Imbunatatiri aduse

Implementare clasica

- Daca pe traseul dintre nearest node si un punct orientat spre random point, la o distanta stepSize, exista un obstacol, algoritmul **nu adauga niciun punct** si **se pierde o iteratie**.

Varianta implementata de noi

- In acest caz, algoritmul **plaseaza un nod la marginea obstacolului**, pe traiectoria calculata, evitand astfel **irosirea unei iteratii** si permitand extinderea arborelui in mod continuu.

II. Imbunatatiri aduse

Context: Atunci cand distanta dintre punctul aleatoriu si nodul cel mai apropiat este **mai mica decat distanta prestabilita (stepSize)**, algoritmul tot va încerca sa **parcurga exact stepSize** în acea directie.

Implementare clasica

- Daca deplasarea cu **stepSize** ar duce punctul **in afara hartii**, algoritmul **nu forteaza iesirea** si iroseste o iteratie.

Varianta implementata de noi

- In acest caz, se **calculeaza pozitia maxima posibila pe marginea hartii**, in acea directie, si se **plaseaza nodul acolo**.Astfel, arborele continua sa se extinda continuu.

III. Imbunatatiri aduse

Implementare clasica:

- Punctele aleatorii sunt complet random, iar algoritmul poate „rataci” prea mult prin spatiu.

Pentru a accelera apropierea de tinta, am introdus o **strategie**:

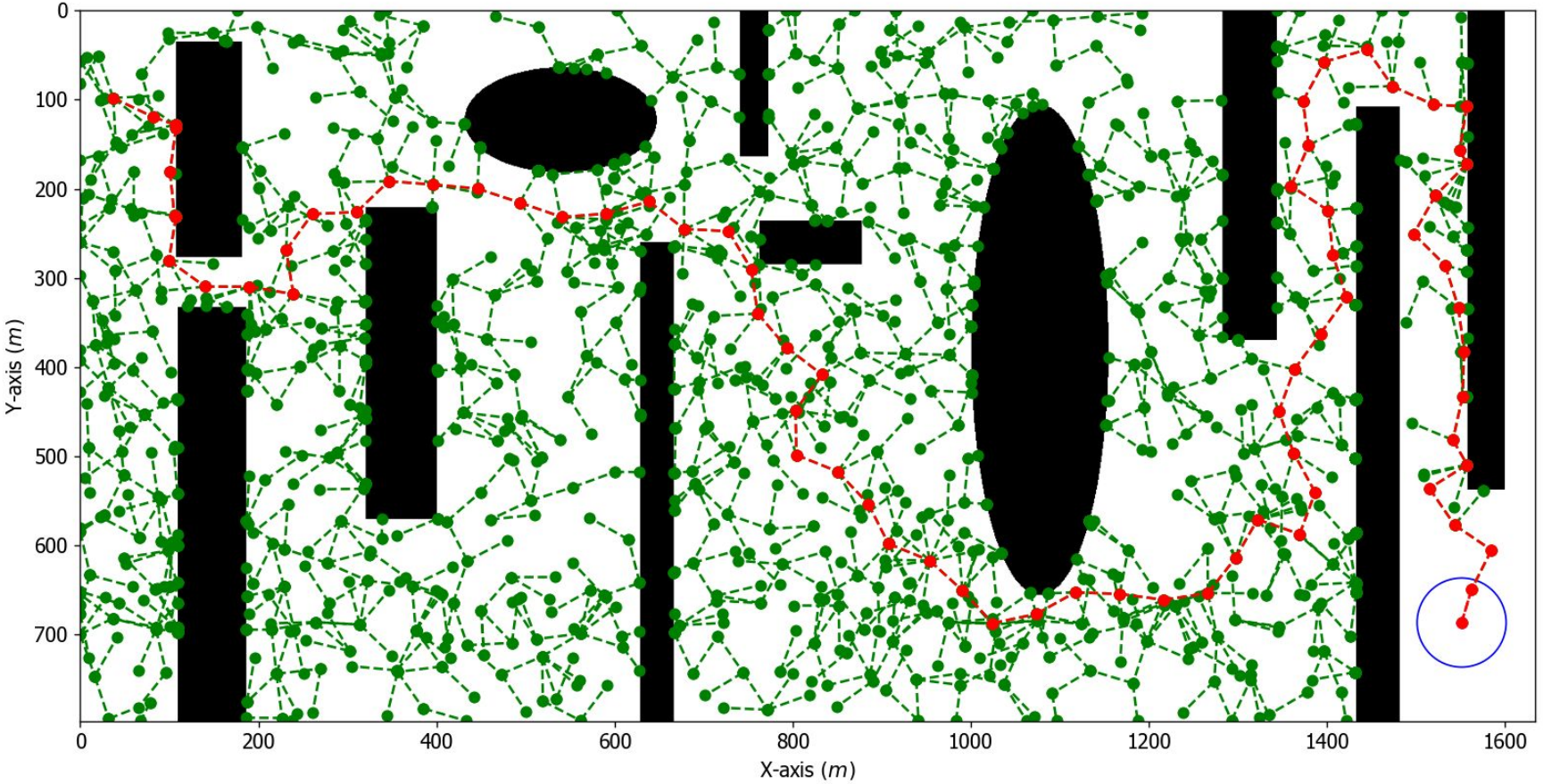
O data la 10 iteratii, punctul aleatoriu generat are:

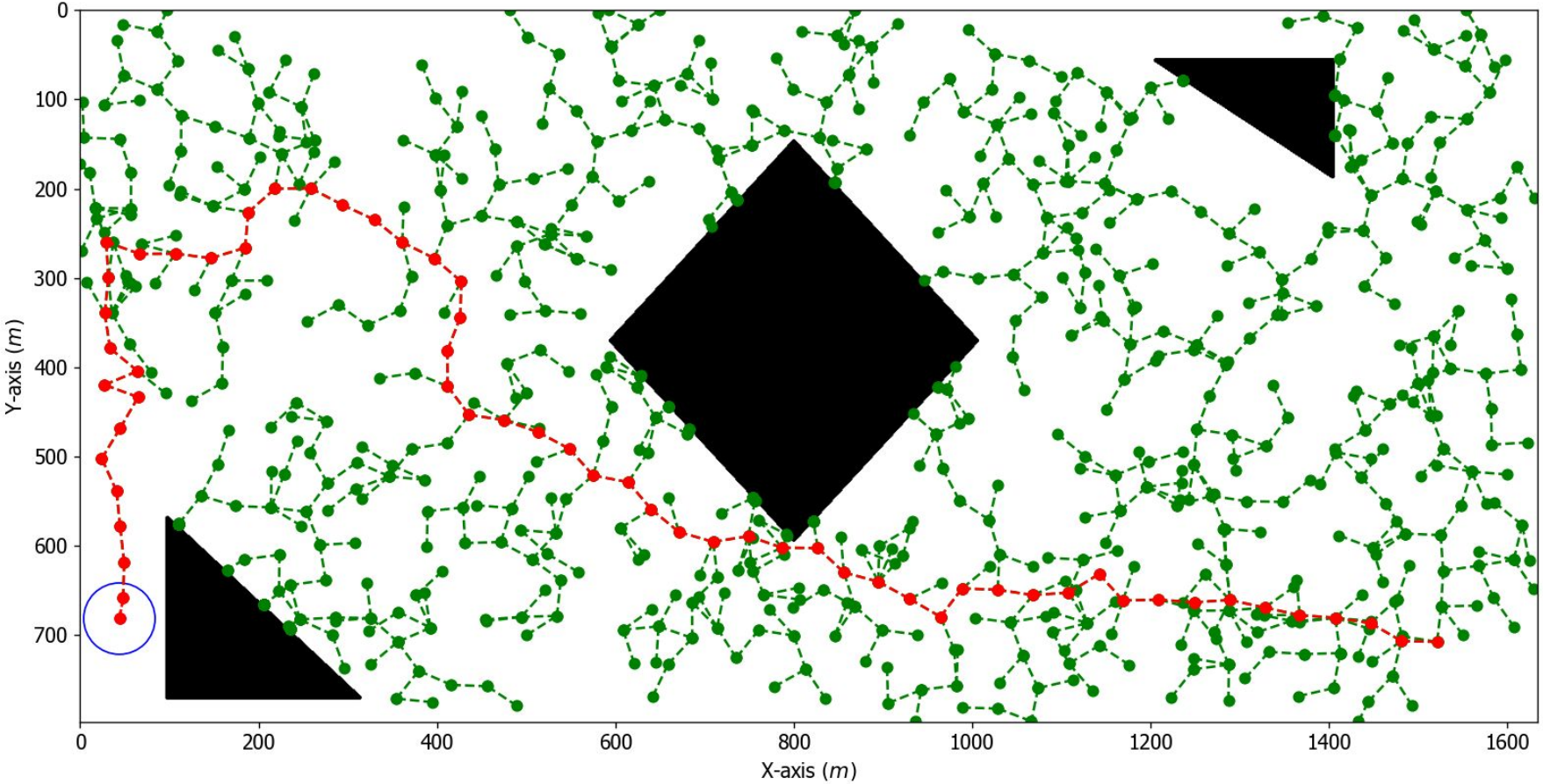
- fie **coordonata y a tintei (goal)** și x aleator, fie **coordonata x a tintei** și y aleator

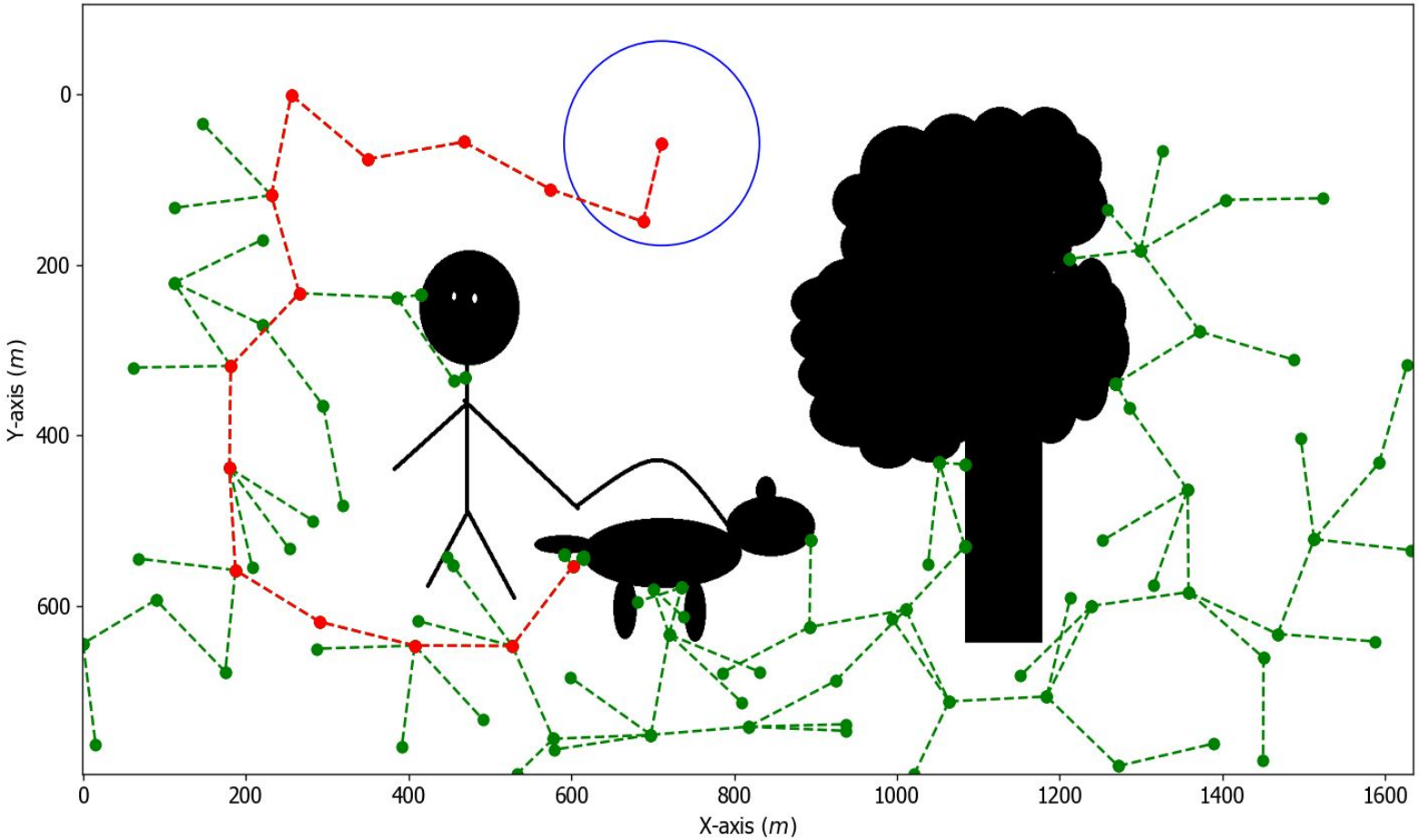
Scopul acestei tehnici:

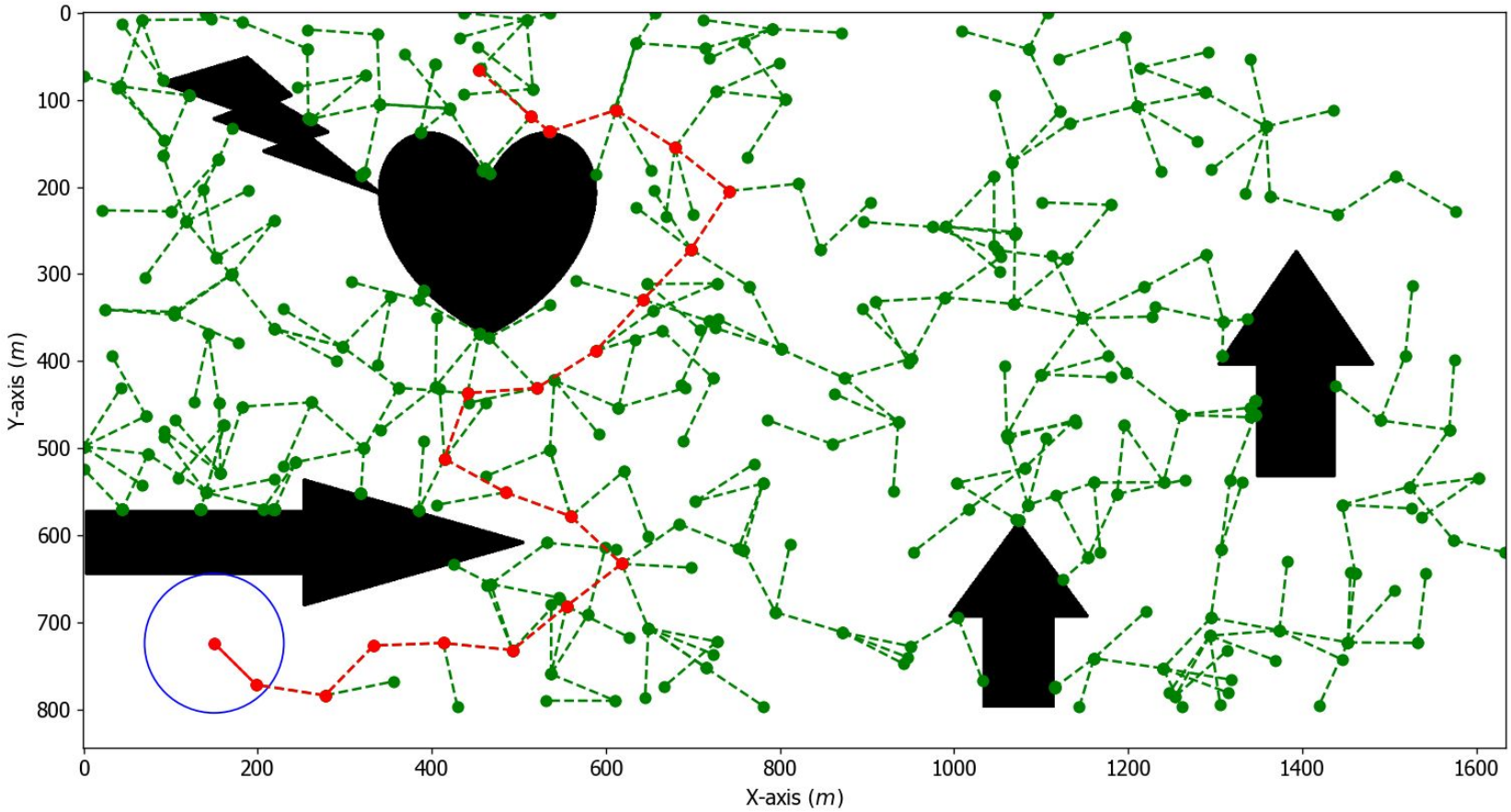
- **Creste sansa** ca arborele sa se indrepte catre zona tintei
- **Reduce numarul de pasi nefolositori** in alte directii
- Ajuta algoritmul sa **gaseasca mai rapid un drum**

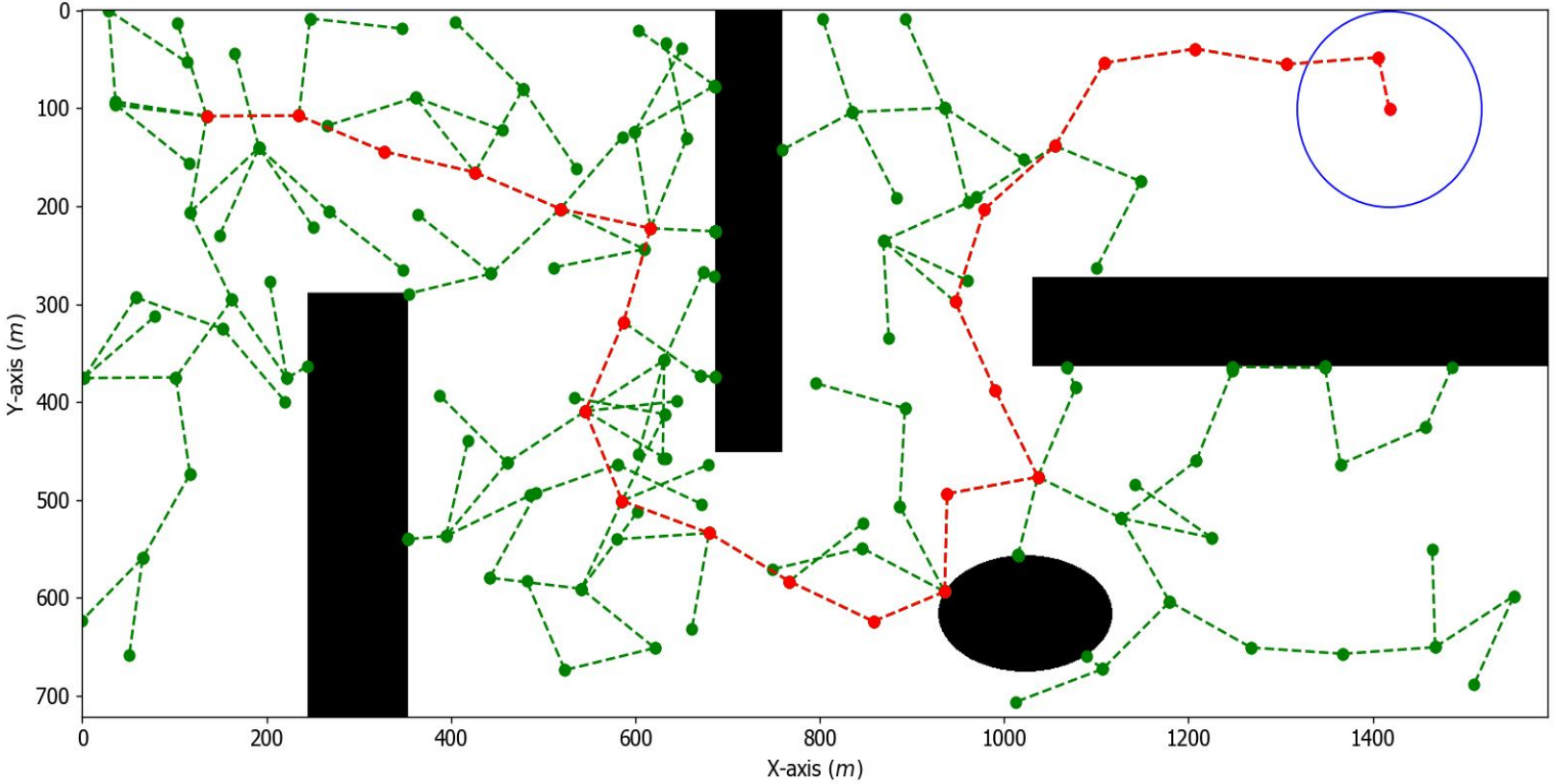
Varianta implementata de noi




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)

Meniu

Am adaugat un meniu user friendly prin care putem selecta un grid diferit pentru a testa performanta algoritmului.

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

Va multumim pentru atentia acordata!