C++ Project

Hui Pang
940218-T086

Solution to Requirements:

1. Understand how the provided code works

Since I am not familiar with the debug environment under Linux, I use a lot of "cout"s as breakpoints to check which part runs first, and write a simple subclass SingleRectangleWorld by following the logic in the given example SingleCircleWorld to better understand how it works.

- For the parameter input of obstacles, In SingleCircleWorld, the parameters of the single circle obstacle are explicitly initialized in the constructor, even before the pointer to read the files was created.

- For collision queries, the PRM algorithm calls the member function collidesWith repeatedly to write points in a list and write them in the Matlab file later.

- For Matlab file for display, the proper grammar to generate the code needs to be tested under Matlab first.

2. Implement a base class for an obstacle and provide it with an appropriate interface

A base class Obstacle is implemented to describe both obstacles.

Since it is pointed at in another base class World, they have to share some of their member functions. Obstacle keeps collidesWith and writeMatlabDisplayCode for interface with obstacles, and leaves the readObstacles for it actually needs to call both member functions under Obstacle.

3. Implement sub-classes of your base class from above that match the obstacle types in this project

Base on Obstacle, subclasses Circle and Rectangle are implemented, with detailed implementation of collidesWith and writeMatlabDisplayCode, Circle's borrowed given code SingleCircleWorld and Rectangle's test code SingleRectangleWorld.

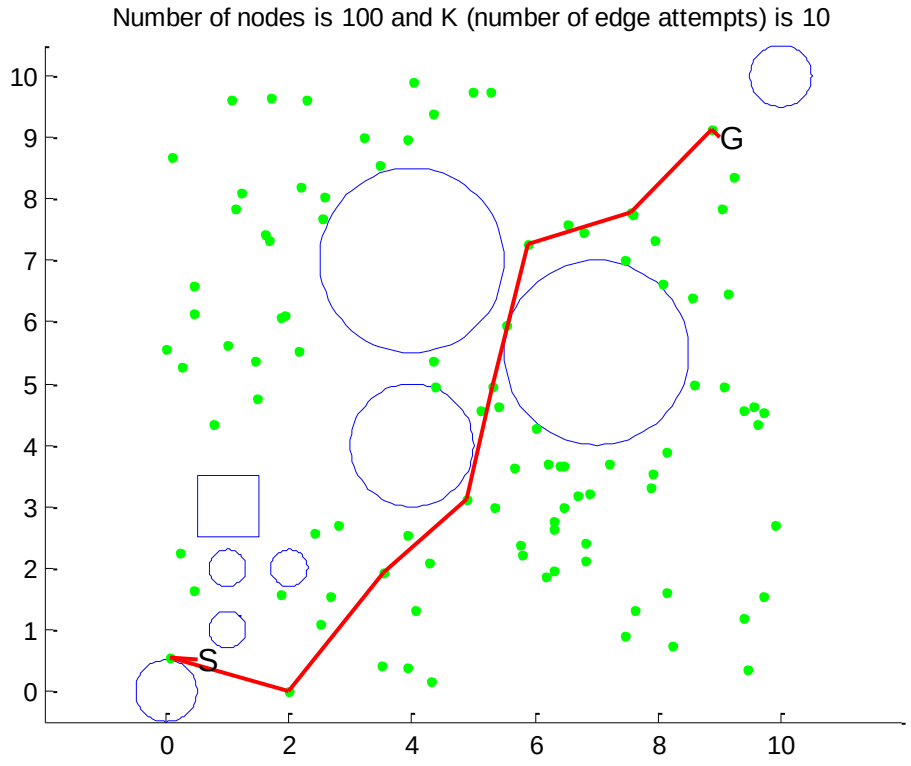
4. Implement a sub-class of World that aggregates obstacles and makes use of object oriented design so that it does not need to differentiate different obstacles when it prints them to file and checks for collisions.

Subclass MyWorld is implemented to read the specified problem specification file and aggregates obstacles to a list. By using pointers to base class obstacle, it does not need to tell obstacles apart to ask collision queries and write them to the PRM list and later to the Matlab file.

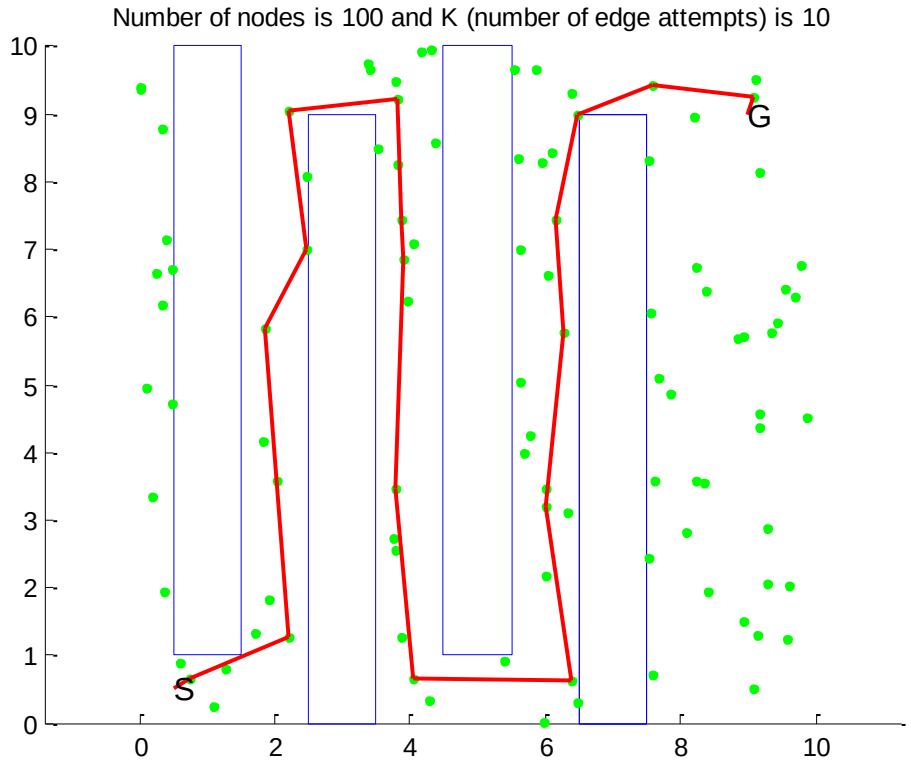
Results:

Run problems using MyWorld:

```
el2310@el2310-VirtualBox:~/prm-files$ make
make: `main' is up to date.
el2310@el2310-VirtualBox:~/prm-files$ ./main -h
Usage: ./main [-h help] [-p inputfile with problem spec] [-n number of nodes] [-
e number of edges per node] [-s step size for collision check] [-w world model]
[-D always display edges]
el2310@el2310-VirtualBox:~/prm-files$ ./main -p problem1.txt -w MyWorld

Parameter setup
===============
#nodes=100
#edges per node10
step size for collision check 0.1
xStart=0.5
yStart=0.5
xGoal=9
yGoal=9
xMin=0
xMax=10
yMin=0
yMax=10
worldModel="MyWorld"

Reading problem specification file "problem1.txt"

Found a path with 10 nodes, look at it by running dispPRM in matlab

el2310@el2310-VirtualBox:~/prm-files$
```



```
DID NOT FIND A PATH, you may want to try again

el2310@el2310-VirtualBox:~/prm-files$ ./main -p problem2.txt -w MyWorld

Parameter setup
===============
#nodes=100
#edges per node10
step size for collision check 0.1
xStart=0.5
yStart=0.5
xGoal=9
yGoal=9
xMin=0
xMax=10
yMin=0
yMax=10
worldModel="MyWorld"

Reading problem specification file "problem2.txt"

Found a path with 19 nodes, look at it by running dispPRM in matlab

el2310@el2310-VirtualBox:~/prm-files$
```



```
el2310@el2310-VirtualBox:~/prm-files$ ./main -p problem3.txt -w MyWorld

Parameter setup
===============
#nodes=100
#edges per node10
step size for collision check 0.1
xStart=0.5
yStart=0.5
xGoal=9
yGoal=9
xMin=0
xMax=10
yMin=0
yMax=10
worldModel="MyWorld"

Reading problem specification file "problem3.txt"

Found a path with 10 nodes, look at it by running dispPRM in matlab

el2310@el2310-VirtualBox:~/prm-files$
```

Number of nodes is 100 and K (number of edge attempts) is 10