Assignment-4

2023-04-03

Name: Varun Gampa
ID : 773686296

Description:

In this assignment a wildfire was simulated in which an agent 'wumpus', would travel an obstacle region. And after reaching the obstacle the same would set the state of the obstacle as burning.

Another agent, that is the firetruck would traverse the region, finding the burning obstacles. When it reaches the obstacle then it extinguishes the same.

The wumpus moves in a grid world like fashion, following the A-star algorithm. It doesn't have a kinematic model, can move in 8 directions in a step like fashion.

The firetruck uses the PRM planner. It has the kinematic constraints of Ackerman steering.

The obstacle region is a set of tetromino with 10% coverage.

Note that, once an obstacle catches fire, if it is not doused in 10 seconds, then it sets the obstacles within 30 meter distance on fire.

Further if an obstacle is extinguished by the firetruck, it can only be relit by the wumpus.

Constraints in the simulation added: (Settings.py file)

The length of the firetruck is 4.9m, the width is 2.2 meter.
The wheel base is 3m , with a maximum turning radius of 13m. This results in maximum steering angle of 13 degrees. The maximum speed of the car is 10 meters. Infact the car is set to have only two speeds, 10 m/s and 1 m/s. Note that all of this is implemented in the settings.py file

The boundary of the game is 250,250 meters.
For purpose of visualization **pygame was used in which 4 pixels were used to represent one meter**.

The obstacles were in the shape of tetrominos, wherein each square block of the tetromino was of dimension 5mx5m.

The coverage of tetrominos was 10 percent, that is, approximately 62 tetrominos on the field.

The wumpus agent is a circular agent of radius 1 meter. It's speed was set as 0.5m per second (note that the speed as per settings.py is wumpus_step_size/wumpus_speed_limiter)

After an obstacle is set to fire, the fire truck can extinguish it , if it reaches a distance of 10 meters from any of its four sub blocks. Further the fire truck takes 5 seconds to extinguish the obstacle. If an obstacle keeps on burning for 400 seconds without getting extinguished then the obstacle becomes completely burnt.

In the simulation each time step is 0.1 seconds, as a result to simulate 3600 seconds, 36000 loops were done.

PRM algorithm implementation:

PRM algorithm:
In the configuration space, random points are sampled. Number of random points sampled is expected to be a large number as per the problem statement. All the points should be such that, it doesn't lead the vehicle to collide with the obstacles.

for each point:
k-nearest points from the set of these random points are found such that path to all these points is obstacle free. These points are added as edges to the graph. (Bi-directional edges)

In the above step, a graph is built. Now for path planning, from any starting position, the configuration of the starting position is added as a vertex to the graph. In the same manner as above, edges are constructed for the same. Further, in the same manner, the end goal's configuration is also added to the graph.

Next using a graph search algorithm, the set of vertices to traverse through to get to the goal is calculated. (This algorithm is called as the global planner)

Next to travel between any two vertices, a search algorithm is employed which finds the collision free path to travel between two vertices. (This algorithm is called as the local planner)

As a result using the combination of global and local planner PRM is able to find the path between any two points in the configuration space.

PRM implementation for the non holonomic constraint:

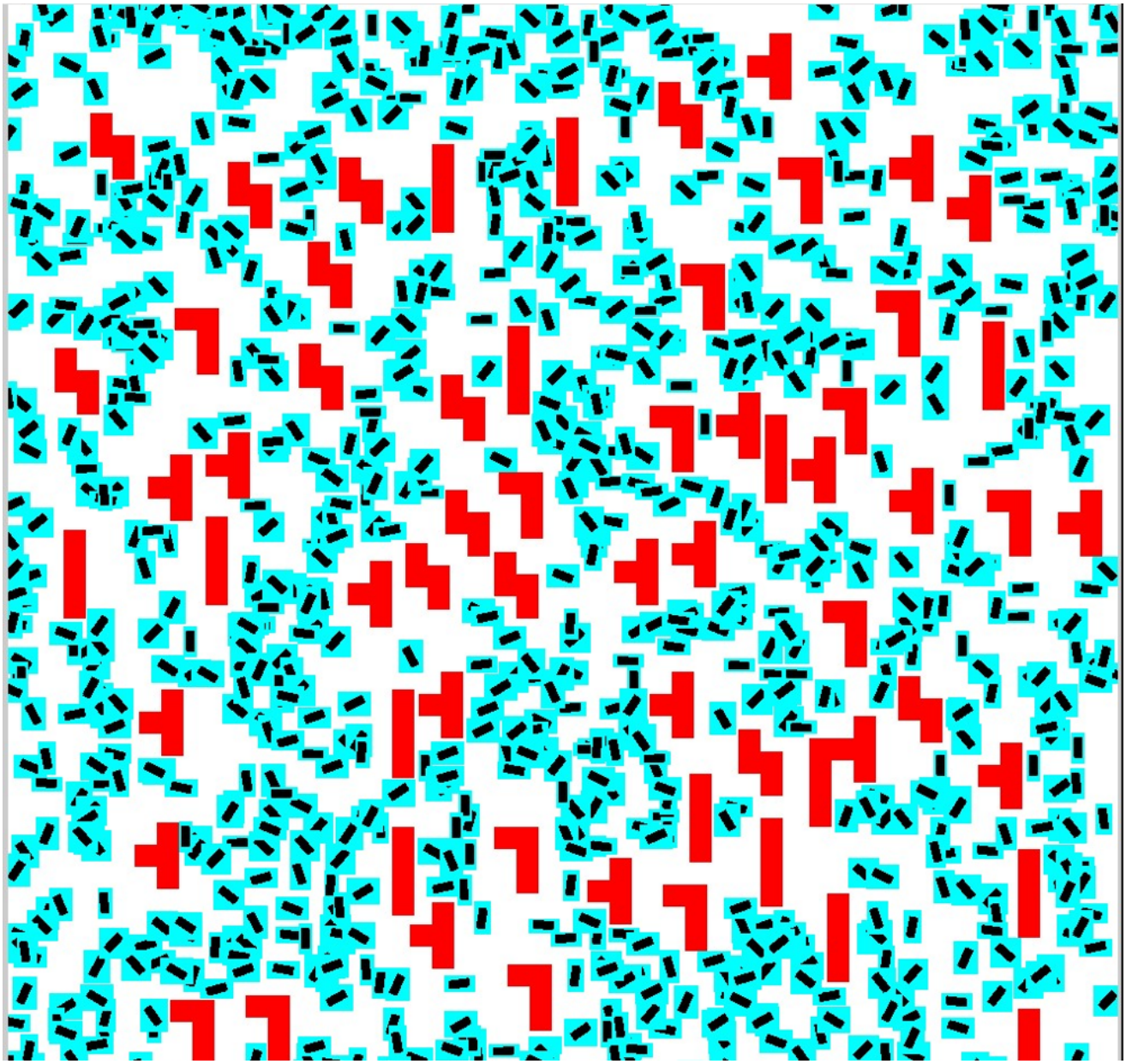
In this section, the specific implementation of PRM is described.

1.

As the first step, obstacle region is generated which is a set of tetrominos. Then random configurations of x, y, theta are chosen. In this code 2000 such points are taken. Each point is such that is doesn't hit the obstacle, as shown : (Note that for each configuration a bounding rectangle is considered with completely enclosed the truck in that configuration with a safety margin of 1.25, and collision check with this rectangle is considered.)

→ Choosing random points logic: With a probability of 0.2, the points' x, y coordinates are chosen from a truncated gaussian distribution with mean centered around any square of any obstacle chosen from the uniform distribution. And theta is chosen from the uniform distribution between -pi to pi. With a probability of 0.8, the points (i.e., configurations) are chosen from the standard uniform distribution.

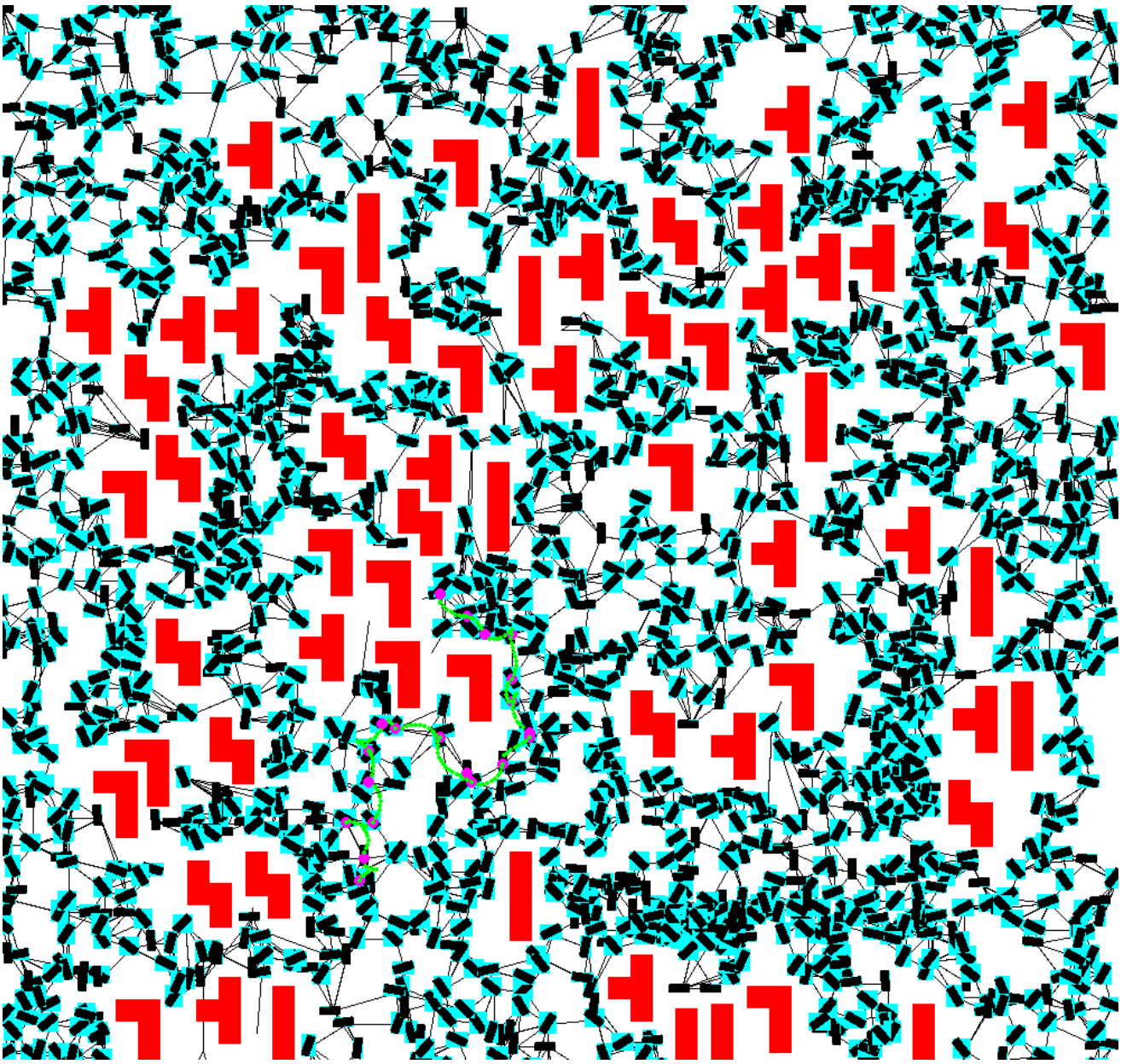
This enables to have more points near the obstacles, such that finer paths required in that region could be found.



2.

Next for each point:
As per the metric of **ReedsShepp** distance from this point, the rest of the points are sorted. Of these, first k points are chosen which can provide a collision free path. In this context, collision free path is chosen as those paths between two points, such that hybrid A-star with kinematic constraints mentioned above can find the path under 0.2 seconds. (Note that if there is a point for which no neighbors can be found, then that point is deleted and replaced by another random point). These k points are assigned as neighbors, resulting in graph generation.

(Resulting graph is shown below)



IMP: Note that if a neighbor is found, the path to the neighbor found by the local planner is also stored.

3.

Next in the finding a path from one configuration to another, global A-star is used, to find the set of vertices to traverse to reach the goal configuration. The weights of each edge is the cost of the local planner to go from one end of edge to the other. (Note that as mentioned above, this is already found when when we employ A-star to check if a neighbor is collision free by finding the path to it under 0.2 seconds)

4.

Using the local planner paths between two vertices saved, the complete path with 0.1 second step size is generated. The same is shown in the above figure,

with the green path being the detailed path.

Simulation results

As per the rules described in the description above, 3600 seconds of simulation is run. Note that this corresponds to 36000 loops for 0.1s step size.

For generating the graph of 1400 points with 5 neighbors, it takes about 8:24 minutes, averaged over 5 iterations. Note that this occurs only once per simulation.

After this the simulation starts, by the wumpus first setting a random obstacle on fire. Which starts to spread as the truck starts to find a burning obstacle and save.

CPU metric

The sum total of wall time used when path planning for the truck, which is finding path from obstacle to save to another is equal to about **0.17568 seconds.**

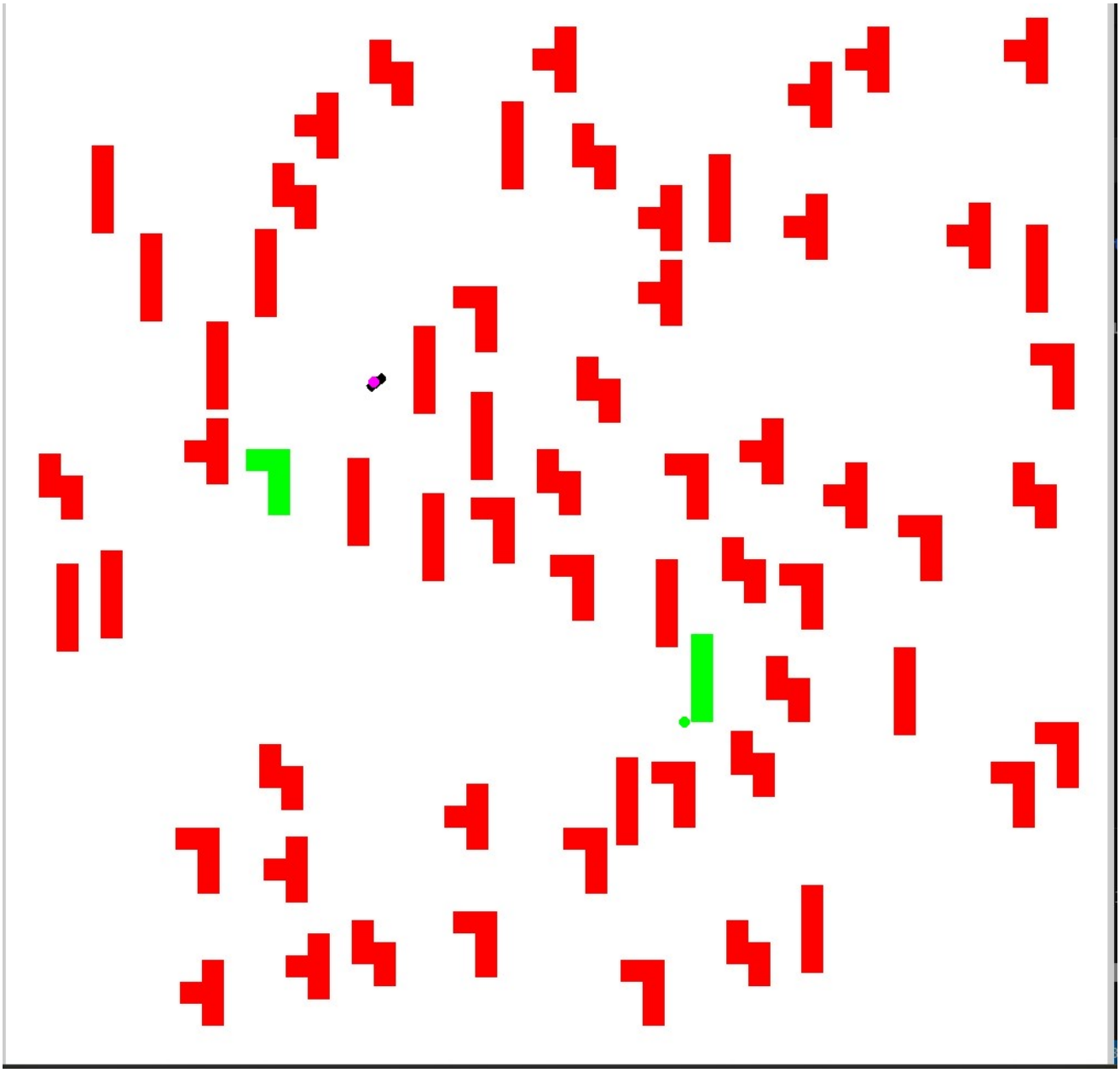
The sum total of wall used when path planning for the wumpus, which is finding path to any unburnt obstacle is about **92.4 seconds.**

The wall clock time to run the simulation without adding any time delays for visualization is about **6:52 minutes**

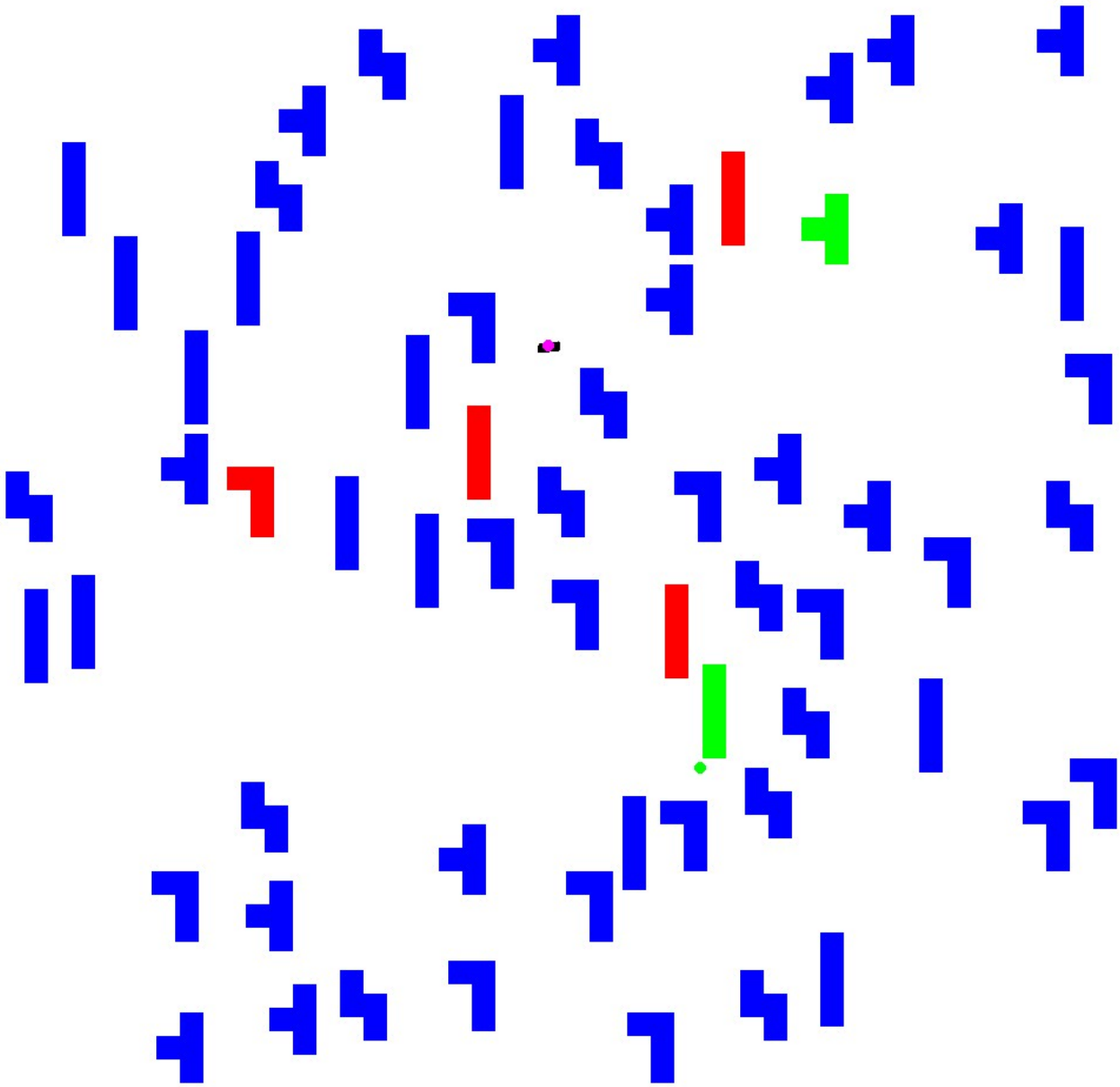
Note that the game usually reaches a steady state in **600 seconds** as per simulation time.

Red is burning obstacles, green is unburnt obstacles, blue is completely burnt obstacles. Green dot is the wumpus, black car is the fire truck.

wildfire spreading image:



Steady state:



((Note that in the video, obstacle complete burning is seen after 1:10 second mark and steady state towards the end))

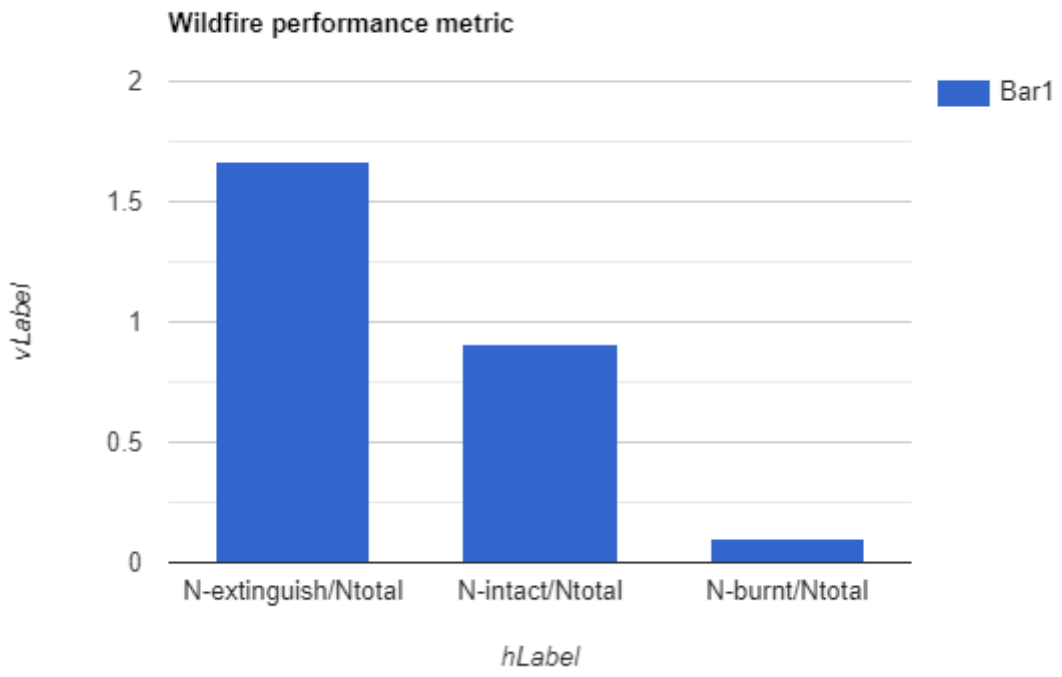
Post simulation number of obstacles not completely burnt is 6.

The number of obstacles completely burnt is 56.

The number of times fire truck puts out fires successfully is 104

The bar graph with metrics is below, where each number above is divided by number of obstacles

Metrics:



Discussion

While PRM takes a long time to setup the graph and the roadmap, but if memory constraints allow for local paths to be save for the children of each vertex, then during run time, PRM is much much faster than A star !. In this case, the fire truck only moves from one configuration to another, as a result local planner is not called as again as it's paths are saved and only the global planner is called. This resulted in PRM planner with kinematic constraints to take only 0.2 seconds out of 6:52 min of simulation, whereas grid based A-star without kinematic constraints took about 90 seconds out of 6:52 min of simulation!

References:

1. https://github.com/ghliu/pyReedsShepp (library for Reeds Shepp) (Note that this is required to run the algorithm)