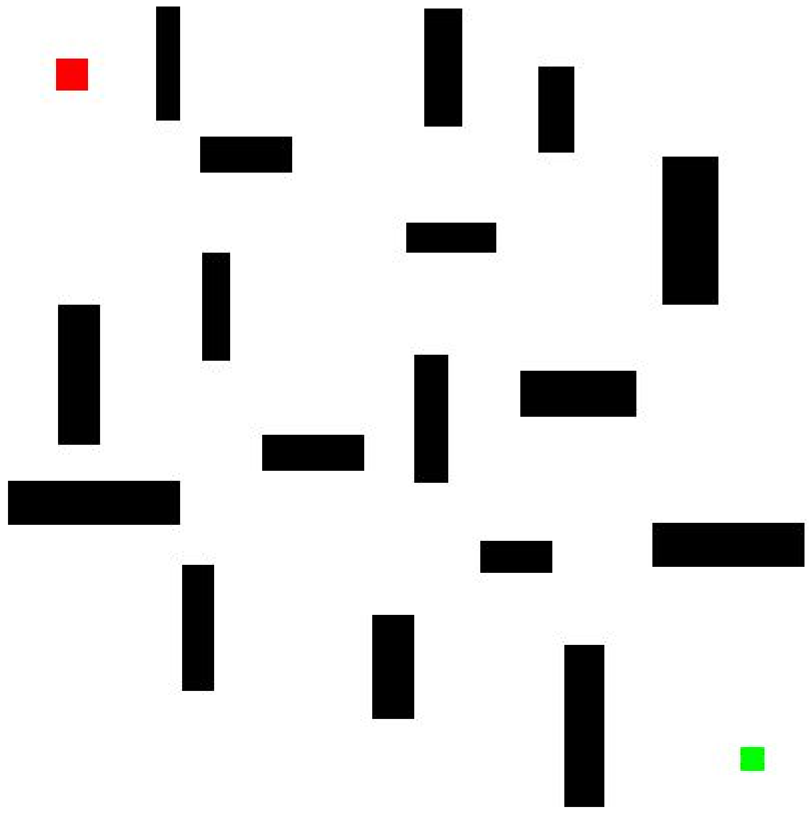
Task-2

You will be given an image consisting of a start point(green) and an end point(red) and some obstacles in black. You have
to write an RRT(Randomly-exploring Random Tree) path planning code to find a path between the two given points.
The image we were given for this task was



Img-1

I have to use RRT pathfinding algorithm.
BASICS OF RRT-
The rrt algorithm is designed to work efficiently both in convex and non convex space.
An RRT grows a tree rooted at the starting configuration by using random samples from the search space. As each sample is drawn, a connection is attempted between it and the nearest state in the tree. If the connection is feasible (passes entirely through free space and obeys any constraints), this results in the addition of the new state to the tree.

The length of the connection between the tree and a new state is frequently limited by a growth factor. If the random sample is further from its nearest state in the tree than this limit allows, a new state at the maximum distance from the tree along the line to the random sample is used instead of the random sample itself. The random samples can then be viewed as controlling the direction of the tree growth while the growth factor determines its rate. This maintains the space-filling bias of the RRT while limiting the size of the incremental growth.
RRT growth can be biased by increasing the probability of sampling states from a specific area. Most practical implementations of RRTs make use of this to guide the search towards the planning problem goals. This is accomplished by introducing a small probability of sampling the goal to the state sampling procedure. The higher this probability, the more greedily the tree grows towards the goal.

(source-https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree)

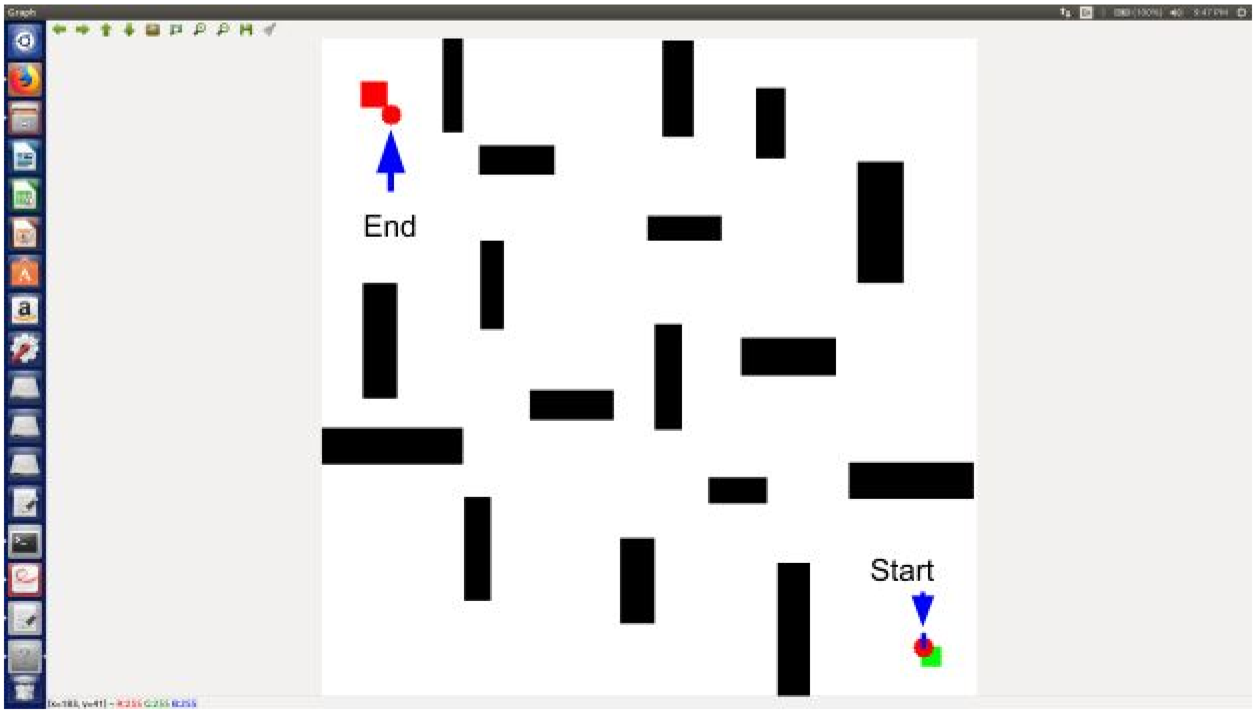
Code-

```
#include "opencv2/highgui/highgui.hpp"     //adding necessary libraries
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/core/core.hpp"
#include<stdio.h>
#include<iostream>
#include<cstdlib>
#include<cmath>
#include<stack>


using namespace cv;
using namespace std;

struct POINT        //structure consisting a point node and index of its
mother
        {
        Point p;
        int mother;
        };

POINT from_to[2];       //to store start and end points
int i=0;
double thresh=8;        //max length it can go in one step
//i'm using array coz i don't know how to use list library in c++
POINT nodes[3000];      //here i will store the nodes
bool done=false;        //if i reach the end point then done will be true
POINT start,end;
stack<Point> path;

Mat graph=imread("rrt.jpg",1); //reading the image
//by left click select the start point and end point
void onMouse(int evt,int x,int y,int flags,void* param)
       {
        if(evt==CV_EVENT_LBUTTONDOWN)
        {
        from_to[i].p.x=x;
        from_to[i].p.y=y;
        circle(graph,from_to[i].p,6,Scalar( 0, 0, 255 ),-1);
        ++i;
        }
  }
  //this function returns a random point from the search space
POINT randPt()
  {
        POINT random;
        int randX=rand();
        randX=randX % graph.cols;

        int randY=rand();
        randY=randY % graph.rows;

        random.p.x=randX;
        random.p.y=randY;

        return random;
  }

double Dist(POINT p1, POINT p2)     //calculate distance between two
points
  {
        double dist=(double)
sqrt((pow(p1.p.x-p2.p.x,2.0)+pow(p1.p.y-p2.p.y,2.0)));
        return dist;
  }
  //if the node is black i.e on obstacle then returns 0 else 1
int isvalid(POINT node,POINT nn)
  {
        if(graph.at<Vec3b>(node.p.y,node.p.x)[0]>250 &&
graph.at<Vec3b>(node.p.y,node.p.x)[1]>250 &&
graph.at<Vec3b>(node.p.y,node.p.x)[2]>250)
        return 1;
        else
        return 0;
  }

//from the list of previous nodes it returns the nearest node
POINT nearestNode(POINT node)
  {
        POINT nn;
        double minDist=600;
        int d;
        for(d=0;d<3000 && (nodes[d].p.y!=0 && nodes[d].p.x!=0);d++)
        {
        if(minDist>Dist(node,nodes[d]))
        {
                minDist=Dist(node,nodes[d]);
                nn.p=nodes[d].p;
                nn.mother=d;
        }

        }
        return nn;
  }
```

```
// it finds a node using the random point
void node()
  {
        POINT Node;
        POINT rnd=randPt();
        POINT nn=nearestNode(rnd);
        double dist=Dist(rnd,nn);

        if(dist<=thresh)
        {
        Node=rnd;
        Node.mother=nn.mother;
        }
        else
        {

        Node.p.x=nn.p.x+(thresh*((rnd.p.x-nn.p.x)/dist));
        Node.p.y=nn.p.y+(thresh*((rnd.p.y-nn.p.y)/dist));
        Node.mother=nn.mother;
        }

        cout<<" @@@@@@@@@ isvalid = "<<isvalid(Node,nn)<<endl;

        if(isvalid(Node,nn))
        {

        line(graph,Node.p,nn.p,Scalar(255,0,0),2);
        int j;
        for(j=0;j<1600;j++)
        {
                if(nodes[j].p.x==0 && nodes[j].p.y==0)break;
        }
        nodes[j]=Node;
        if(Dist(Node,end)<16)
        {
        done= true;
        }
        }
        cout<< "**** done "<<endl;

  }

int main()
  {
        namedWindow("Graph",0);
        setMouseCallback("Graph",onMouse,NULL);// callback function
        Point temp;

        for(int z=0;z<3000;z++) //initializing the array
        {
        nodes[z].p.x=0;
        nodes[z].p.y=0;
        nodes[z].mother=0;
        }
        cout<<">>>>>>> Choose the start point and the end point in the graph
by left click ---"<<endl;
        while(1)
        {
        imshow("Graph",graph);
        waitKey(10);

        if(i>=2)break;
        }
        start=from_to[0];
        end=from_to[1];


        cout <<" Start Point = "<<start.p<<" End Point = "<<end.p<<endl;
        cout <<" random point = "<<randPt().p<<endl;
        cout <<" dist = "<<Dist(start,end)<<endl;

        nodes[0]=start;
        long int k=0;
        while(1)
        {
```

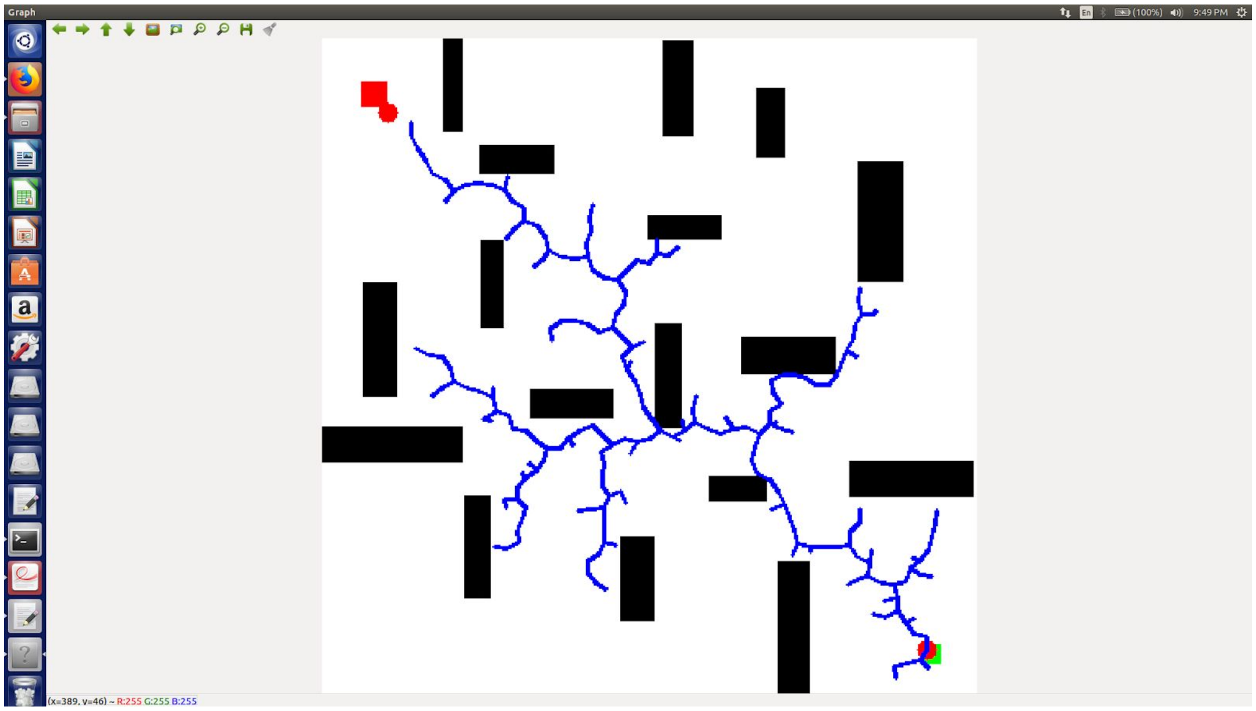
```
        node();
        imshow("Graph",graph);
        ++k;
        waitKey(1);
        if(done)break;
        }        }
        cout<<"\n\n\n\n>>>>>>>> press any key to see the path--"<<endl;
        waitKey(0);


        int a;
        for(a=0;a<3000;a++)
        {
        cout<<" #######################  mother_index =
"<<nodes[a].mother<<"  a = "<<a<<endl;
        if(nodes[a].p.x==0 && nodes[a].p.y==0)break;
        }

        cout<<" a = "<<a<<endl;



        POINT tmp=nearestNode(end);

        line(graph,tmp.p,end.p,Scalar(238,130,238),2);
        path.push(end.p);

        do
        {
        path.push(tmp.p);
        line(graph,tmp.p,nodes[tmp.mother].p,Scalar(238,130,238),2);
        tmp=nodes[tmp.mother];
        }while(tmp.mother!=0);
        line(graph,tmp.p,start.p,Scalar(238,130,238),2);
        path.push(start.p);
        cout<<"\n\n\n\n>>>>>>> and the consisting points of the path are
stored in the stack 'path'\n\n"<<endl;

        imshow("Graph",graph);
        waitKey(0);
  }
```

And here is one output --



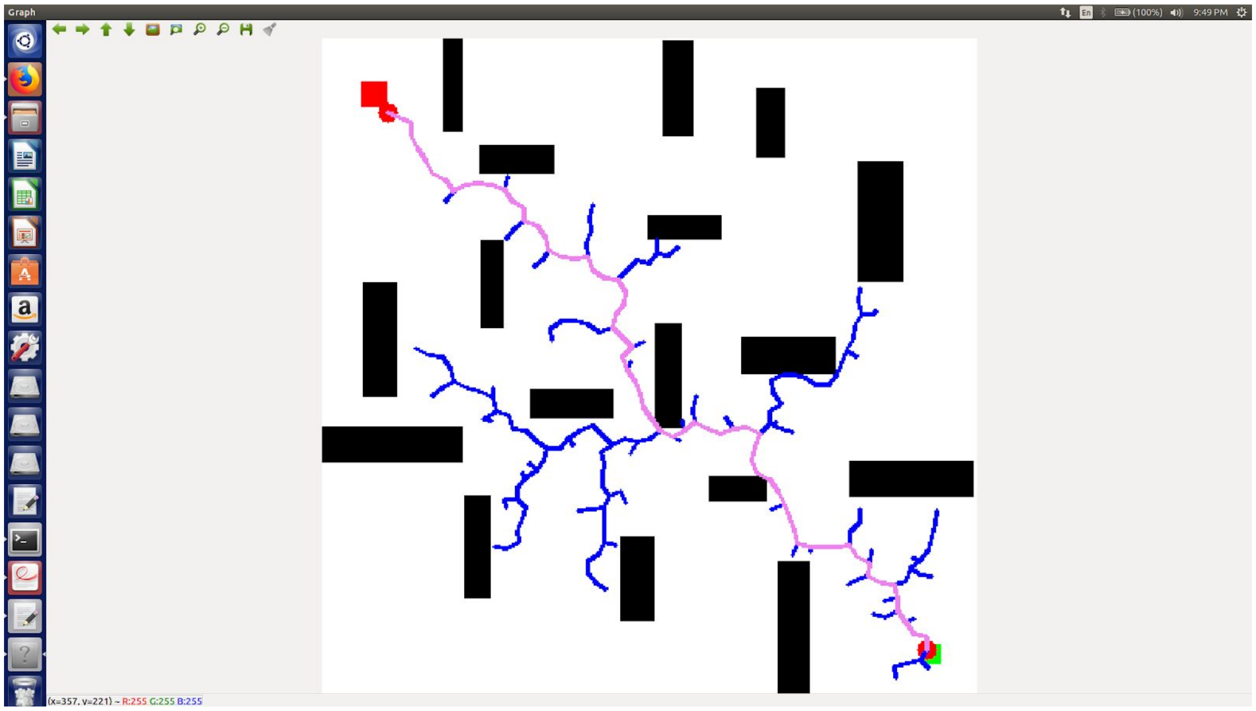
Img-2

Start and end points are chosen by left click of mouse



Img-3

Searching randomly avoiding randomly.



Img-4

Final path - in violet colour.