```
#include<bits/stdc++.h>
#include<graphics.h>
using namespace std;
#define STEPS 100

int n,backCount=0,graphCount=0;
float X,Y,xSeg,ySeg;
int **maze;

int getXInd(int V)
{
   return V/n;
}

int getYInd(int V)
{
   return V%n;
}

int getVNum(int x,int y)
{
   return y+n*x;
}

class Graph
{
private:
   int V;
   vector<int> *adjList;
public:
   Graph(int v)
```

```
{
    V=v;
    adjList=new vector<int>[V];
}
void addEdge(int u,int v)
{
    adjList[u].push_back(v);
}
int* shortestPath(int src)
{
    int *dist=new int[V];
    int *parent=new int[V];
    for(int i=0;i<V;i++) dist[i]=INT_MAX;
    for(int i=0;i<V;i++) parent[i]= -1;

    queue<int> sP;
    sP.push(src);
    dist[src]=0;

    while(!sP.empty())
    {
        int top = sP.front();
        sP.pop();
        for(auto vector<int> ::iterator it=adjList[top].begin();it!=adjList[top].end();it++)
if(dist[*it]==INT_MAX)
        {
            sP.push((*it));
            parent[(*it)]=top;
            dist[(*it)]=dist[top]+1;
        }
    }
```

```
        return parent;
    }
};

void create_lines()
{
    for(int i=1;i<=n;i++) line(xSeg*i,0,xSeg*i,Y);
    for(int i=1;i<=n;i++) line(0,ySeg*i,X,ySeg*i);
}

void create_stop(int xInd,int yInd,int col=1)
{
    if(xInd==0 && yInd==0) col=2;
    if(xInd==n-1 && yInd==n-1) col=3;

    int xCor=xInd*xSeg;
    int yCor=yInd*ySeg;
    if(col==0) setfillstyle(SOLID_FILL,LIGHTMAGENTA);
    else if(col==1) setfillstyle(SOLID_FILL,BLACK);
    else if(col==2) setfillstyle(SOLID_FILL,LIGHTGRAY);
    else if(col==3) setfillstyle(SOLID_FILL,RED);
    else if(col==4) setfillstyle(SOLID_FILL,GREEN);
    else if(col==5) setfillstyle(SOLID_FILL,LIGHTBLUE);
    floodfill(xCor+1,yCor+1,WHITE);

    create_lines();
}

void color_base()
{
    create_stop(0,0,2);
```

```
    create_stop(n-1,n-1,3);
}
void create_obstacles()
{
    for(int i=0;i<n;i++) for(int j=0;j<n;j++) if(maze[i][j]==1) create_stop(i,j,1);
    return;
}

void create_maze()
{
    X=getmaxx();
    Y=getmaxy();
    xSeg=X/n;
    ySeg=Y/n;

    setfillstyle(SOLID_FILL,LIGHTMAGENTA);
    floodfill(1,1,LIGHTMAGENTA);

    create_lines();
    color_base();
    create_obstacles();

    delay(2000);
}

void startColoring(int* parent,int curr)
{
    if(parent[curr]==-1) return;
    startColoring(parent,parent[curr]);
    graphCount++;
    create_stop(getXInd(curr),getYInd(curr),5);
```

```
    delay(STEPS);
}

bool toCheck(int xX,int yY)
{
    if(xX==0 && yY==0) return true;
    if(xX==n-1 && yY==n-1) return true;
    if(xX>=0 && xX<n && yY>=0 && yY<n) if(maze[xX][yY]==0) return true;
    return false;
}

bool toCheckG(int xX,int yY)
{
    if(xX==0 && yY==0) return true;
    if(xX==n-1 && yY==n-1) return true;
    if(xX>=0 && xX<n && yY>=0 && yY<n) if(maze[xX][yY]==0 || maze[xX][yY]==2) return true;
    return false;
}

bool solveMaze(int currX=0,int currY=0)
{
    if(currX==n-1 && currY==n-1) return true;

    if(toCheck(currX,currY)==true)
    {
        backCount++;
        maze[currX][currY]=2;
        create_stop(currX,currY,4);delay(STEPS);
        if(solveMaze(currX+1,currY)==true) return true;
        if(solveMaze(currX,currY+1)==true) return true;
        if(solveMaze(currX-1,currY)==true) return true;
```

```
        if(solveMaze(currX,currY-1)==true) return true;
        maze[currX][currY]=1;
        create_stop(currX,currY,0);delay(STEPS);

    }
    return false;

}

int main()
{
    cout<<"Enter the Dimension of Maze:\n";
    cin>>n;
    cout<<"Enter Degree/Density of Obstacles:\n";
    cout<<"1 being HIGH\n2 being MEDIUM\n3 being LESS\n";
    int blockNum;
    cin>>blockNum;

    int gdriver=DETECT,gmode;
            initgraph(&gdriver,&gmode,"");

    maze = new int*[n];
    for(int i=0;i<n;i++) maze[i]=new int[n];
    for(int i=0;i<n;i++) for(int j=0;j<n;j++) maze[i][j]=0;
    srand(time(NULL));
    for(int i=0;i<n;i++) for(int j=0;j<n;j++) maze[i][j]=rand()%2;
    for(int k=0;k<blockNum;k++) for(int i=0;i<n;i++) for(int j=0;j<n;j++) if(maze[i][j]==1)
maze[i][j]=rand()%2;
    maze[0][0]=2;maze[n-1][n-1]=3;
    create_maze();
```

```
   solveMaze();
   cout<<"HENCE THE BLOCKS COVERED BY BACKTRACKING IS ->"<<backCount<<endl;

   color_base();
   delay(3000);

   Graph mazeGraph(n*n);
   for(int i=0;i<n;i++) for(int j=0;j<n;j++) if(toCheckG(i,j))
   {
      if(toCheckG(i+1,j)) mazeGraph.addEdge(getVNum(i,j),getVNum(i+1,j));
      if(toCheckG(i-1,j)) mazeGraph.addEdge(getVNum(i,j),getVNum(i-1,j));
      if(toCheckG(i,j+1)) mazeGraph.addEdge(getVNum(i,j),getVNum(i,j+1));
      if(toCheckG(i,j-1)) mazeGraph.addEdge(getVNum(i,j),getVNum(i,j-1));
   }
   int*parent = mazeGraph.shortestPath(0);
   startColoring(parent,getVNum(n-1,n-1));
   color_base();
   cout<<"HENCE THE BLOCKS COVERED BY GRAPH THROUGH  BFS IS ->"<<graphCount<<endl;
   int x=backCount*STEPS;
   cout<<"HENCE THE TIME TAKEN TO SOLVE THE MAZE BY BACKTRACKING IS "<<x<<"
miliseconds"<<endl;
   int y=graphCount*STEPS;
   cout<<"HENCE THE TIME TAKEN TO SOLVE THE MAZE BY GRAPH IS "<<y<<" miliseconds"<<endl;
   cout<<"THIS SHOWS THAT TIME TAKEN BY GRAPH ALGORITHM IS LESS THAN THAT OF
BACKTRACKING BY "<< x-y<< " milliseconds";
   color_base();
   delay(5000);

   return 0;
}
```