**中北大学**

**算法程序设计实验周**

**说 明 书**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **学 院:** | 大数据学院 | | | | | |
| **专 业:** | 计算机科学与技术 | | | | | |
| **题 目:** | 探寻宝藏 | | | | | |
| **成 绩:** | **出勤情况(0.1)** | **算法演示(0.4)** | | **说明书(0.2)** | **答辩(0.3)** | **总体评价** |
|  |  | |  |  |  |
| **指导教师:** | 李玲 | | | | | |
| **学生姓名：** | 张华健 | | **学号** | | 1707004531 | |

 2019 年 6月 23 日

# 1.设计目的

1.1 项目说明

传说HMH大沙漠中有一个M\*N迷宫，里面藏有许多宝物。某天，Dr.Kong找到了迷宫的地图，他发现迷宫内处处有宝物，最珍贵的宝物就藏在右下角，迷宫的进出口在左上角。当然，迷宫中的通路不是平坦的，到处都是陷阱。Dr.Kong决定让他的机器人卡多去探险。

但机器人卡多从左上角走到右下角时，只会向下走或者向右走。从右下角往回走到左上角时，只会向上走或者向左走，而且卡多不走回头路。（即：一个点最多经过一次）。当然卡多顺手也拿走沿路的每个宝物。Dr.Kong希望他的机器人卡多尽量多地带出宝物。请你编写程序，帮助Dr.Kong计算一下，卡多最多能带出多少宝物。

1.2 设计目标

此次算法设计目标是通过利用动态规划的方法使得探寻宝藏问题的解决效率得到提高，并使其空间资源得到优化。通过可视化与动态化的方法使得此问题的解决方法得到更好的呈现。

1.3 项目设计进度说明

2019年05月22日 ~ 06月09日 布置题目，并对题目进行分析；

2019年06月10日 ~ 06月17日 算法设计及编码；

2019年06月18日 ~ 06月19日 对所涉及算法进行调试；

2019年06月20日 对算法进行分析测试；

2019年06月21日 撰写实验周说明书；

2019年06月21日 验收、成绩考核。

# 2.需求分析

2.1 项目需求

本项目旨在辅助卡多完成迷宫寻宝并得出最多的宝藏路线。将采用双进程dp问题的解决思路对卡多的路线进行提前规划，同时通过可视化与动态化的方式演示卡多寻宝的过程。

2.2 项目业务要求

（1）需要设计路线帮助卡多得到最多的宝藏。

（2）通过可视化的方式将卡多寻路的方式表现出来。

（3）需要良好的交互界面。

# 3.总体设计

3.1 总体设计

此问题分为三部分，第一部分：进行动态建表的过程（即函数find\_ans），当找到最大宝藏同时备忘表已建立。第二部分：通过此备忘表进行回溯查询 （即find\_route），得到标记两条路的二维迷宫表。第三部分：将得到的迷宫数组与迷宫路线数组传入可视化的类中进行可视化设计。

3.3 算法的详细设计思想

（1）设计内容：

传说HMH大沙漠中有一个M\*N迷宫，里面藏有许多宝物。某天，Dr.Kong找到了迷宫的地图，他发现迷宫内处处有宝物，最珍贵的宝物就藏在右下角，迷宫的进出口在左上角。当然，迷宫中的通路不是平坦的，到处都是陷阱。Dr.Kong决定让他的机器人卡多去探险。但机器人卡多从左上角走到右下角时，只会向下走或者向右走。从右下角往回走到左上角时，只会向上走或者向左走，而且卡多不走回头路。（即：一个点最多经过一次）。当然卡多顺手也拿走沿路的每个宝物。Dr.Kong希望他的机器人卡多尽量多地带出宝物。请你编写程序，帮助Dr.Kong计算一下，卡多最多能带出多少宝物。

（2）设计要求：

输入

第一行： K 表示有多少组测试数据。

接下来对每组测试数据：

第1行: M N

第2~M+1行： Ai1 Ai2 ……AiN (i=1,…..,m)

【约束条件】

2≤k≤5 1≤M, N≤50 0≤Aij≤100 (i=1,….,M; j=1,…,N)

所有数据都是整数。 数据之间有一个空格。

输出

对于每组测试数据，输出一行：机器人卡多携带出最多价值的宝物数

符合探寻宝藏题目要求，实现寻找最优宝藏路径；

要求算法设计合理，有较高的算法效率；

注意程序的实用性、可读性；

设计符合面向对象的设计原则和思想。

3.4 源代码

Mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include<QKeyEvent>

namespace **Ui** {

class **MainWindow**;

}

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

public:

explicit **MainWindow**(QWidget \*parent = nullptr);

~***MainWindow***();

void **find\_ans**(int k,int total, int row, int col);

void **find\_route**(int k, int total, int row, int col);

private slots:

void **on\_up\_clicked**();

void **on\_mazegroup\_currentIndexChanged**(int index);

void **init\_string**();

void **initMaze**();

void **on\_left\_clicked**();

void **on\_right\_clicked**();

void **on\_down\_clicked**();

void **on\_pgup\_clicked**();

void **on\_pgdn\_clicked**();

void **on\_nextMaze\_clicked**();

void ***keyPressEvent***(QKeyEvent \*event);

void **on\_beforeMaze\_clicked**();

void **on\_runMaze\_clicked**();

void **on\_pushButton\_clicked**();

private:

Ui::MainWindow \*ui;

int c; //数据组数

int m[5]; //行

int n[5]; //列

int maze[5][55][55]; //迷宫

int dp[110][55][55]; //表

int mark[5][55][55]; //路线

};

#endif // MAINWINDOW\_H

Mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include<QMovie>

#include<QDebug>

MainWindow::**MainWindow**(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

QMovie \*movie =new QMovie("E:/image/mazesearch .gif");

ui->logo->setMovie(movie);

movie->start();

connect(ui->begin,SIGNAL(clicked()),this,SLOT(initMaze()));

//connect(ui->mazegroup,SIGNAL(on\_mazegroup\_currentIndexChanged()),)

}

MainWindow::~***MainWindow***()

{

delete ui;

}

void MainWindow::**on\_mazegroup\_currentIndexChanged**(int index)

{

c=index+1;

ui->mazewidget->setC(c,0);

qDebug()<<c;

}

void MainWindow::**init\_string**(){

QString str;

str=ui->mazeText->toPlainText();

QStringList list=str.split(" ");

memset(m,0,sizeof(m));

memset(n,0,sizeof(n));

int num=0;

for(int i=0;i<c;i++){

m[i]=list.at(num).toInt();

num++;

n[i]=list.at(num).toInt();

num++;

for(int col=1;col<=m[i];col++){

for(int row=1;row<=n[i];row++){

maze[i][col][row]=list.at(num).toInt();

num++;

}

}

if(num>=(list.size()-1))

break;

}

for (int k=0;k<c;k++) {

for(int col=1;col<=m[k];col++){

for(int row=1;row<=n[k];row++){

qDebug()<<maze[k][col][row];

}

}

}

}

void MainWindow::**find\_ans**(int k,int total, int row, int col) {

for (int l = 2; l<total; l++) {

for (int i = 1; i <= row; i++) {

for (int j = 1; j <= col; j++) {

if (i != j&&l >= i&&l >= j) {

dp[l][i][j] = max(max(dp[l - 1][i - 1][j], dp[l - 1][i][j - 1]), max(dp[l - 1][i - 1][j - 1], dp[l - 1][i][j])) + maze[k][i][l - i] + maze[k][j][l - j];

}

qDebug() << dp[l][i][j] << "\t";

}

qDebug() << endl;

}

qDebug() << endl;

}

dp[total][row][col] = max(dp[total - 1][row][row - 1], dp[total - 1][row - 1][row]) + maze[k][row][col];

}

void MainWindow::**find\_route**(int k, int total, int row, int col) {

int max[3][1];

max[0][0] = row;

max[1][0] = row;

max[2][0] = dp[total][row][col];

mark[k][row][col] = 3;

//printf\_s("%d\n", max[2][0]);

for (int l = total; l > 2; l--) {

int r1 = maze[k][max[0][0]][l - max[0][0]];

if (l == total) {

r1 = 0;

}

// qDebug() <<"r1="<< r1 <<endl;

int r2 = maze[k][max[1][0]][l - max[1][0]];

// qDebug() << "r2=" << r2 << endl;

max[2][0] = max[2][0] - r1 - r2;

// qDebug()<< "testmax" << max[2][0]<<endl;

for (int i = 1; i <= row; i++) {

for (int j = i; j <= col; j++) {

if (dp[l - 1][i][j] == max[2][0]) {

max[0][0] = i;

max[1][0] = j;

// qDebug() <<"test"<< max[0][0] << "\t" << max[1][0] << endl;

break;

}

}

}

mark[k][max[0][0]][l - max[0][0] - 1] = 1;

mark[k][max[1][0]][l - max[1][0] - 1] = 2;

//qDebug() << max[0][0] << "\t" << max[1][0] << "\t" << max[2][0] << endl;

}

//cout << endl;

}

void MainWindow::**initMaze**() {

init\_string();

memset(mark, 0, sizeof(mark));

for (int k=0;k<c;k++) {

memset(dp, 0, sizeof(dp));

find\_ans(k,n[k] + m[k], m[k], n[k]);

find\_route(k,n[k] + m[k], m[k], n[k]);

mark[k][1][1] = 3;

for (int i = 1; i <= m[k]; i++) {

for (int j = 1; j <= n[k]; j++) {

qDebug() << mark[k][i][j] << "\t";

}

qDebug()<<"\n";

}

}

ui->mazewidget->setMaze(maze,mark,c,m,n);

ui->mazewidget->*updateGL*();

}

void MainWindow::**on\_left\_clicked**()

{

ui->mazewidget->setX(1);

}

void MainWindow::**on\_right\_clicked**()

{

ui->mazewidget->setX(-1);

}

void MainWindow::**on\_up\_clicked**()

{

ui->mazewidget->setZ(1);

}

void MainWindow::**on\_down\_clicked**()

{

ui->mazewidget->setZ(-1);

}

void MainWindow::**on\_pgup\_clicked**()

{

ui->mazewidget->setY(-1);

}

void MainWindow::**on\_pgdn\_clicked**()

{

ui->mazewidget->setY(1);

}

void MainWindow::**on\_nextMaze\_clicked**()

{

ui->mazewidget->setnext(true);

}

void MainWindow::***keyPressEvent***(QKeyEvent \*event){

switch (event->key())

{

case Qt::Key\_F1: //F1为全屏和普通屏的切换键

on\_nextMaze\_clicked();

break;

case Qt::Key\_F2:

on\_beforeMaze\_clicked();

break;

case Qt::Key\_F5:

on\_runMaze\_clicked();

break;

case Qt::Key\_Escape: //ESC为退出键

close();

break;

case Qt::Key\_W:

on\_up\_clicked();

//m\_zPos+=0.1f;

break;

case Qt::Key\_S:

on\_down\_clicked();

//m\_zPos-=1.0f;

break;

case Qt::Key\_Q: //按下PageUp视角向上转

on\_pgup\_clicked();

break;

case Qt::Key\_E: //按下PageDown视角向下转

on\_pgdn\_clicked();

break;

case Qt::Key\_D: //Right按下向左旋转场景

on\_right\_clicked();

break;

case Qt::Key\_A: //Left按下向右旋转场景

on\_left\_clicked();

break;

case Qt::Key\_Enter:

on\_pushButton\_clicked();

break;

}

ui->mazewidget->*updateGL*();

}

void MainWindow::**on\_beforeMaze\_clicked**()

{

ui->mazewidget->setnext(false);

}

void MainWindow::**on\_runMaze\_clicked**()

{

ui->mazewidget->runMaze();

}

void MainWindow::**on\_pushButton\_clicked**()

{

ui->mazewidget->resize(1366,768);

ui->mazewidget->full();

}

Maze.h

#ifndef MAZE\_H

#define MAZE\_H

#pragma once

#include<QWidget>

#include<QGLWidget>

#include<gl/GL.h>

#include<gl/GLU.h>

#include<iostream>

#include<algorithm>

#include<string>

//row 行 col 列

using namespace std;

class **Maze** : public QGLWidget

{

Q\_OBJECT

public:

explicit **Maze**(QWidget \*parent = nullptr);

~***Maze***();

void **setMaze**(int myMaze[][55][55],int value[][55][55],int c,int m[],int n[]);

protected:

void ***initializeGL***(); //初始化界面

void ***resizeGL***(int w,int h); //窗口刷新时重置

void ***paintGL***(); //绘图函数

//void keyPressEvent(QKeyEvent \*event);//键盘响应事件

// void inputKey(int key, int x, int y);//键盘响应事件

void **orientMe**(float ang);

void **moveMeFlat**(int direction);

//void keyPressEvent(QKeyEvent \*event);//键盘响应事件

private:

void **Map**(); //初始化场景

void **buildLists**(); //初始化显示列

private:

int myMaze[5][55][55]; //迷宫宝藏

int value[5][55][55]; //迷宫宝藏的价值

int c;

int m[5];

int n[5];

int count=0; //计数器

bool nextMaze; //是否全屏显示

bool run=false; //是否走迷宫

QString m\_FileName; //图片路径

GLuint m\_Texture; //存储纹理

QString m\_BlackName; //背景路径

GLuint m\_Black; //背景

QString m\_run1Name; //路径

GLuint m\_run1; //路径

QString m\_run2Name;

GLint m\_run2;

GLuint m\_Fog; //雾的模式

GLuint m\_Box; //保存盒子的显示列表

constexpr static const float m\_PIOVER180 = 0.0174532925f; //实现度和弧度直接的折算

GLfloat m\_xPos; //储存当前位置

GLfloat m\_zPos;

GLfloat m\_yRot; //视角的旋转

GLfloat m\_LookUpDown; //记录抬头和低头

public:

void **setX**(double x); //控制移动

void **setY**(double y);

void **setZ**(double z);

void **setnext**(bool full); //下一个

void **setC**(int c,int count); //总数

void **runMaze**(); //走迷宫

void **full**();

};

#endif // MAZE\_H

Maze.cpp

#include "maze.h"

#include<QTimer>

#include<QKeyEvent>

#include<QDebug>

Maze::**Maze**(QWidget \*parent) : QGLWidget(parent)

{

nextMaze=false;

m\_xPos=0.0f;

m\_yRot=0.0f;

m\_zPos=0.0f;

m\_Fog = 0;

m\_FileName="E:/image/maze.jpg";

m\_BlackName="E:/image/blackmaze.jpg";

m\_run1Name="E:/image/apple.png";

m\_run2Name="E:/image/banana.png";

QTimer \*timer=new QTimer(this);

connect(timer,SIGNAL(timerout()),this,SLOT(upadteGl()));

timer->start(10);

}

Maze::~***Maze***(){

}

void Maze::**setMaze**(int (\*myMaze)[55][55],int (\*value)[55][55],int c,int m[],int n[]){

qDebug()<<"kaishi";

for (int k=0;k<c;k++) {

for(int col=1;col<=m[k];col++){

for(int row=1;row<=n[k];row++){

this->myMaze[k][col][row]=myMaze[k][col][row];

this->value[k][col][row]=value[k][col][row];

qDebug()<<this->myMaze[k][col][row]<<"+"<< this->value[k][col][row];

}

}

}

for(int i=0;i<c;i++){

this->m[i]=m[i];

this->n[i]=n[i];

}

}

void Maze::***initializeGL***(){

m\_Texture = bindTexture(QPixmap(m\_FileName)); //载入位图并转换成纹理

m\_Black=bindTexture(QPixmap(m\_BlackName)); //背景纹理

m\_run1=bindTexture(QPixmap(m\_run1Name)); //路径

m\_run2=bindTexture(QPixmap(m\_run2Name));

glEnable(GL\_TEXTURE\_2D); //启用纹理映射

buildLists(); //创建显示列表

glClearColor(0.0f, 0.0f, 0.0f, 0.0f); //黑色背景

glShadeModel(GL\_SMOOTH); //启用阴影平滑

glClearDepth(1.0); //设置深度缓存

glEnable(GL\_DEPTH\_TEST); //启用深度测试

glDepthFunc(GL\_LEQUAL); //所作深度测试的类型

glHint(GL\_PERSPECTIVE\_CORRECTION\_HINT, GL\_NICEST); //告诉系统对透视进行修正

GLfloat LightAmbient[] = {0.5f, 0.5f, 0.5f, 1.0f}; //环境光参数

GLfloat LightDiffuse[] = {1.0f, 1.0f, 1.0f, 1.0f}; //漫散光参数

GLfloat LightPosition[] = {0.0f, 0.0f, 2.0f, 1.0f}; //光源位置

glLightfv(GL\_LIGHT1, GL\_AMBIENT, LightAmbient); //设置环境光

glLightfv(GL\_LIGHT1, GL\_DIFFUSE, LightDiffuse); //设置漫射光

glLightfv(GL\_LIGHT1, GL\_POSITION, LightPosition); //设置光源位置

glEnable(GL\_LIGHT1); //启动一号光源

GLfloat fogColor[] = {0.5f, 0.5f, 0.5f, 1.0f}; //雾的颜色

glFogi(GL\_FOG\_MODE, GL\_EXP); //设置雾气的初始模式

glFogfv(GL\_FOG\_COLOR, fogColor); //设置雾的颜色

glFogf(GL\_FOG\_DENSITY, 0.05); //设置雾的密度

glHint(GL\_FOG\_HINT, GL\_DONT\_CARE); //设置系统如何计算雾气

glFogf(GL\_FOG\_START, 1.0f); //雾的开始位置

glFogf(GL\_FOG\_END, 5.0f); //雾的结束位置

glEnable(GL\_FOG); //启动雾效果

}

void Maze::***resizeGL***(int w, int h){

glViewport(0, 0, (GLint)w, (GLint)h); //重置当前的视口

glMatrixMode(GL\_PROJECTION); //选择投影矩阵

glLoadIdentity(); //重置投影矩阵//设置视口的大小

gluPerspective(45.0,((GLdouble)w/(GLdouble)h),0.1,100.0);

glMatrixMode(GL\_MODELVIEW); //选择模型观察矩阵

glLoadIdentity();

*updateGL*();

}

void Maze::**buildLists**(){

m\_Box = glGenLists(1); //创建两个显示列表的空间

glNewList(m\_Box, GL\_COMPILE); //开始创建第一个显示列表

glBegin(GL\_QUADS);

glNormal3f(0.0f, 0.0f, 1.0f);

glTexCoord2f(1.0f, 1.0f);

glVertex3f(1.0f, 1.0f, -1.0f); //右上(顶面)

glTexCoord2f(0.0f, 1.0f);

glVertex3f(-1.0f, 1.0f, -1.0f); //左上(顶面)

glTexCoord2f(0.0f, 0.0f);

glVertex3f(-1.0f, 1.0f, 1.0f); //左下(顶面)

glTexCoord2f(1.0f, 0.0f);

glVertex3f(1.0f, 1.0f, 1.0f); //右下(顶面)

glTexCoord2f(0.0f, 0.0f);

glVertex3f(1.0f, -1.0f, 1.0f); //右上(底面)

glTexCoord2f(1.0f, 0.0f);

glVertex3f(-1.0f, -1.0f, 1.0f); //左上(底面)

glTexCoord2f(1.0f, 1.0f);

glVertex3f(-1.0f, -1.0f, -1.0f); //左下(底面)

glTexCoord2f(0.0f, 1.0f);

glVertex3f(1.0f, -1.0f, -1.0f); //右下(底面)

glTexCoord2f(1.0f, 1.0f);

glVertex3f(1.0f, 1.0f, 1.0f); //右上(前面)

glTexCoord2f(0.0f, 1.0f);

glVertex3f(-1.0f, 1.0f, 1.0f); //左上(前面)

glTexCoord2f(0.0f, 0.0f);

glVertex3f(-1.0f, -1.0f, 1.0f); //左下(前面)

glTexCoord2f(1.0f, 0.0f);

glVertex3f(1.0f, -1.0f, 1.0f); //右下(前面)

glTexCoord2f(0.0f, 0.0f);

glVertex3f(1.0f, -1.0f, -1.0f); //右上(后面)

glTexCoord2f(1.0f, 0.0f);

glVertex3f(-1.0f, -1.0f, -1.0f); //左上(后面)

glTexCoord2f(1.0f, 1.0f);

glVertex3f(-1.0f, 1.0f, -1.0f); //左下(后面)

glTexCoord2f(0.0f, 1.0f);

glVertex3f(1.0f, 1.0f, -1.0f); //右下(后面)

glTexCoord2f(1.0f, 1.0f);

glVertex3f(-1.0f, 1.0f, 1.0f); //右上(左面)

glTexCoord2f(0.0f, 1.0f);

glVertex3f(-1.0f, 1.0f, -1.0f); //左上(左面)

glTexCoord2f(0.0f, 0.0f);

glVertex3f(-1.0f, -1.0f, -1.0f); //左下(左面)

glTexCoord2f(1.0f, 0.0f);

glVertex3f(-1.0f, -1.0f, 1.0f); //右下(左面)

glTexCoord2f(1.0f, 1.0f);

glVertex3f(1.0f, 1.0f, -1.0f); //右上(右面)

glTexCoord2f(0.0f, 1.0f);

glVertex3f(1.0f, 1.0f, 1.0f); //左上(右面)

glTexCoord2f(0.0f, 0.0f);

glVertex3f(1.0f, -1.0f, 1.0f); //左下(右面)

glTexCoord2f(1.0f, 0.0f);

glVertex3f(1.0f, -1.0f, -1.0f);

glEnd();

glEndList(); //第一个显示列表结束

}

void Maze::***paintGL***(){

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT); //清除屏幕和深度缓存

Map();

}

void Maze::**Map**(){

glBindTexture(GL\_TEXTURE\_2D, m\_Texture); //选择纹理

qDebug()<<"cout="<<count<<"c="<<c;

for (int y=-10;y<10;y++) {

for (int z=-10; z<10; z++) //循环来控制画盒子

{

for (int x=-10; x<10; x++)

{

if(y<7&&y>-9&&x<9&&x>-9&&z<9&&z>-9)

continue;

if(z==9&&x>-3&&x<3&&y<9&&y>1)

continue;

if(y==7&&x<9&&x>-9&&z<9&&z>-9){

if(x>0&&z>0&&myMaze[count][x][z]>0){

if(run&&(value[count][x][z]>0||(x==1&&z==1))){

if(value[count][x][z]!=2){

qDebug()<<"run1="<<value[count][x][z];

glBindTexture(GL\_TEXTURE\_2D, m\_run1);

glLoadIdentity();//设置盒子的位置

glTranslatef(0.0f+(float(x)\*4.0f+m\_xPos),

10.0f-float(y)\*2.0f+m\_yRot, -50.0f+float(z)\*4.0f+m\_zPos);

glCallList(m\_Box);

}else{

qDebug()<<"run2="<<value[count][x][z];

glBindTexture(GL\_TEXTURE\_2D, m\_run2);

glLoadIdentity();//设置盒子的位置

glTranslatef(0.0f+(float(x)\*4.0f+m\_xPos),

10.0f-float(y)\*2.0f+m\_yRot, -50.0f+float(z)\*4.0f+m\_zPos);

glCallList(m\_Box);

}

}else{

qDebug()<<"myMaze="<<myMaze[count][x][z];

glBindTexture(GL\_TEXTURE\_2D, m\_Black);

glLoadIdentity();//设置盒子的位置

glTranslatef(0.0f+(float(x)\*4.0f+m\_xPos),

10.0f-float(y)\*2.0f+m\_yRot, -50.0f+float(z)\*4.0f+m\_zPos);

glCallList(m\_Box);

}

}

}else{

glBindTexture(GL\_TEXTURE\_2D, m\_Texture);

glLoadIdentity();//设置盒子的位置

glTranslatef(0.0f+(float(x)\*2.0f+m\_xPos),

10.0f-float(y)\*2.0f+m\_yRot, -40.0f+float(z)\*2.0f+m\_zPos);

glCallList(m\_Box); //绘制盒子

}

}

}

}

}

void Maze::**runMaze**(){

this->run=true;

qDebug()<<"run";

*updateGL*();

}

void Maze::**setX**(double x){

m\_xPos+=x;

*updateGL*();

qDebug()<<x;

}

void Maze::**setY**(double y){

m\_yRot+=y;

*updateGL*();

qDebug()<<y;

}

void Maze::**setZ**(double z){

m\_zPos+=z;

*updateGL*();

qDebug()<<z;

}

void Maze::**setnext**(bool next){

qDebug()<<next;

if(next){

count++;

if(count>=c-1){

count=c-1;

}

}else{

count--;

if(count<=0){

count=0;

}

}

run=false;

*updateGL*();

}

void Maze::**setC**(int c, int count){

this->c=c;

this->count=count;

qDebug()<<"c="<<c<<"count="<<count;

}

void Maze::**full**(){

*resizeGL*(1366,768);

}

Main.cpp

#include "mainwindow.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

# 4.实验周设计体会

这是双进程DP问题，首先，假设出发点为A 终点为B 那么，根据题目给出的条件，可以推出A->B的动态转移方程为 dp[i][j] = max(dp[i-1][j],dp[i][j-1]) + a[i][j]; 由于，同理可得B的情况，那么，题目的意思是A->B 然后 B -> A我们可以假设同时从A点出发，得到两条不同路径，这个是一样的效果。所以，我们可以得到一个动态转移方程dp[i][j][p][q] = max(dp[i-1][j][p-1][q],dp[i-1][j][p][q],dp[i][j-1][p-1][q],dp[i][j-1][p][q-1]) 因为 每次只能移动一步，即 i+1 或j+1 那么 i+j是移动的步数 因为从A点开始移动的，经过相同的步数，肯定能得到i+j = p+q

经过观察发现此问题可以优化一维，转化为三维数组。此问题可以看成过两个人同时从起点向终点前进，但要保证路线不能相交，除了在终点，两人速度相同，如果某一时刻在相同的行或者列，则他们一定会在相同的列或行，因此保证他们行不相同（k>=i&&k>=j&&i!=j），既可以保证列不相交。因此建立三维数组dp[k][i][j]进行记录。（k:总步数，i：第一个人的行数，j:第二个人的行数）dp[k][i][j]：走到第k步（在起点时是第二步），A走到i行，B走到j行时所得的最大宝藏数，则A在的列为k-i,B为k-j。因此进行推算得状态转移方程为；dp[l][i][j]=max(max(dp[l-1][i-1][j],dp[l-1][i][j-1]),max(dp[l-1][i-1][j-1],dp[l-1][i][j]))+a[i][l-i]+a[j][l-j]，最后需加上右下角的宝藏。

# 5.参考文献

[1] （美）Anany Levitin 著，潘彦 译﹒算法设计与分析基础（第2版）﹒北京：清华大学出版社，2007

[2] 王红梅 胡明﹒算法设计与分析（第2版）﹒北京：清华大学出版社，2013

[3] 刘汝佳﹒算法竞赛入门经典﹒北京：清华大学出版社，2014

[4] 王晓东.计算机算法设计与分析（第三版）.北京：清华大学出版社，2014

[5] 王晓东﹒算法设计与分析习题解答（第3版）﹒北京：清华大学出版社，2014

[6] 邹恒明﹒算法之道（第2版）﹒北京：机械工业出版社，2012

[7] 王秋芬 吕聪颖 周春光﹒算法设计与分析﹒北京：清华大学出版社，2011

[8] cly116. Qt Creator中的3D绘图及动画教程.<https://blog.csdn.net/cly116>.

[9] [南笺](https://www.cnblogs.com/xiehuan-blog/).[Qt5学习教程](https://www.cnblogs.com/xiehuan-blog/articles/9029080.html).<https://www.cnblogs.com/xiehuan-blog/articles/9029080.html>