南开大学 计算机大类

姓名朱春雯

学号2412452

班级3205c++2024

2025年5月14日

高级语言程序设计

实验报告

目录

[高级语言程序设计大作业实验报告 1](#_Toc21286)

[一. 作业题目 2](#_Toc29969)

[二. 开发软件 2](#_Toc18364)

[三. 课题要求 2](#_Toc742)

[四. 主要流程 2](#_Toc25240)

[1． 整体流程 2](#_Toc20452)

[2． 具体实现 2](#_Toc25354)

[（1）Man类 2](#_Toc22508)

[（2） AI类 3](#_Toc28665)

( 3 ) Chess类………………………………………8

( 4 ) ChessGame类………………………………15

[五. 收获 16](#_Toc23116)

[1． 透明贴图的方法 16](#_Toc10799)

[2． AI思考如何走棋 18](#_Toc16275)

[3． Easyx的用法………………………………………………………… 19](#_Toc16275)

高级语言程序设计大作业实验报告

1. **作业题目**

AI五子棋

1. **开发软件**

Visual Studio 2022（基于easyx）

1. **课题要求**
2. 面向对象。
3. 模型部分
4. 验证
5. **主要流程**
   1. **整体流程**

**将项目分为4个类，分别为：Man;AI;Chess;ChessGame。其中ChessGame用于控制整个游戏，其他类用于实现具体功能。在main函数中实现游戏的主循环等。**

* 1. **具体实现 （部分）**

1. Man类：

1)Man.h:

#pragma once

#include"Chess.h"

class Man

{

public:

void init(Chess\* chess);//利用棋盘数据对棋手进行初始化

void go();

private:

Chess\* chess;

};

2)Man.cpp:

#include "Man.h"

void Man::init(Chess\* chess)

{

this->chess = chess;

}

void Man::go()

{

MOUSEMSG msg;

ChessPos pos;

while (1) {

msg=GetMouseMsg();//获取鼠标点击消息

//判断落子是否有效以及落子功能

if (msg.uMsg == WM\_LBUTTONDOWN && chess->clickBoard(msg.x, msg.y, &pos)) {

break;

}

}

//printf("%d,%d\n", pos.row, pos.col);

//落子

chess->chessDown(&pos, CHESS\_BLACK);

}

Man类中的难点在于获取鼠标点击信息，是一个不太会的知识点

1. AI类：

1)AI.h

#pragma once

#include"Chess.h"

class AI

{

public:

void init(Chess\* chess);//利用棋盘的数据进行AI的初始化

void go();

private:

Chess\* chess;

vector<vector<int>>scoreMap;

private:

void caculateScore();

ChessPos think();

};

2)AI.cpp

#include "AI.h"

void AI::init(Chess\* chess)

{

this->chess = chess;

int size = chess->getGradeSize();

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

vector<int>row;

for (int j = 0;j < size;j++) {

row.push\_back(0);

}

scoreMap.push\_back(row);

}

}

void AI::go()

{

ChessPos pos = think();

Sleep(1000);//停顿

chess->chessDown(&pos, CHESS\_WHITE);

}

void AI::caculateScore()

{

int personNum = 0;//棋手（黑子）在这个方向连续有多少颗棋子

int aiNum = 0;//ai在这个方向有多少颗连续的棋子

int emptyNum = 0;//在该方向空白位的个数

for (int i = 0;i < scoreMap.size();i++) {

for (int j = 0;j < scoreMap[i].size();j++) {

scoreMap[i][j] = 0;

}

}

int size = chess->getGradeSize();

for (int row = 0;row < size;row++) {

for (int col = 0;col < size;col++) {

personNum = 0;

aiNum = 0;

emptyNum = 0;

//对每个点进行计算

if (chess->getChessData(row, col))continue;

for (int y = -1;y <=0;y++) {

for (int x = -1;x <= 1;x++) {

personNum = 0;

aiNum = 0;

emptyNum = 0;

if (y == 0 && x == 0)continue;

if (y == 0 && x != 1)continue;

for (int i = 1;i <= 4;i++) {

int curRow = row + i \* y;

int curCol = col + i \* x;

if (curRow >= 0 && curRow < size && curCol >= 0 && curCol < size && chess->getChessData(curRow, curCol) == 1) {

personNum++;

}

else if (curRow >= 0 && curRow < size && curCol >= 0 && curCol < size && chess->getChessData(curRow, curCol) == 0) {

emptyNum++;

break;

}

else {

break;

}

}

for (int i = 1;i <= 4;i++) {

int curRow = row - i \* y;

int curCol = col - i \* x;

if (curRow >= 0 && curRow < size && curCol >= 0 && curCol < size && chess->getChessData(curRow, curCol) == 1) {

personNum++;

}

else if (curRow >= 0 && curRow < size && curCol >= 0 && curCol < size && chess->getChessData(curRow, curCol) == 0) {

emptyNum++;

break;

}

else {

break;

}

}

if (personNum == 1) {

scoreMap[row][col] += 10;

}

else if (personNum == 2)

{

if (emptyNum == 1) {

scoreMap[row][col] += 30;

}

else if (emptyNum == 2) {

scoreMap[row][col] += 40;

}

}

else if (personNum == 3) {

if (emptyNum == 1)

scoreMap[row][col] += 60;

else if (emptyNum == 2)

scoreMap[row][col] += 200;

}

else if(personNum==4) {

scoreMap[row][col] += 20000;

}

//进行白棋的判断

emptyNum = 0;

for (int i = 1;i <= 4;i++) {

int curRow = row + i \* y;

int curCol = col + i \* x;

if (curRow >= 0 && curRow < size && curCol >= 0 && curCol < size && chess->getChessData(curRow, curCol) == -1) {

aiNum++;

}

else if (curRow >= 0 && curRow < size && curCol >= 0 && curCol < size && chess->getChessData(curRow, curCol) == 0) {

emptyNum++;

break;

}

else {

break;

}

}

for (int i = 1;i <= 4;i++) {

int curRow = row - i \* y;

int curCol = col - i \* x;

if (curRow >= 0 && curRow < size && curCol >= 0 && curCol < size && chess->getChessData(curRow, curCol) == -1) {

aiNum++;

}

else if (curRow >= 0 && curRow < size && curCol >= 0 && curCol < size && chess->getChessData(curRow, curCol) == 0) {

emptyNum++;

break;

}

else {

break;

}

}

if (aiNum == 0) {

scoreMap[row][col] += 5;

}

else if (aiNum == 1) {

scoreMap[row][col] += 10;

}

else if (aiNum == 2) {

if (emptyNum == 1) {

scoreMap[row][col] += 25;

}

else if (emptyNum == 2) {

scoreMap[row][col] +=50 ;

}

}

else if (aiNum == 3) {

if (emptyNum == 1) {

scoreMap[row][col] += 55;

}

else if (emptyNum == 2) {

scoreMap[row][col] += 10000;

}

}

else if (aiNum >= 4) {

scoreMap[row][col] += 30000;

}

}

}

}

}

}

ChessPos AI::think()

{

caculateScore();

vector<ChessPos>maxPoints;

int maxScore = 0;

int size = chess->getGradeSize();

for (int row = 0;row < size;row++) {

for (int col = 0;col < size;col++) {

if (chess->getChessData(row, col) == 0) {

if (scoreMap[row][col] > maxScore) {

maxScore = scoreMap[row][col];

maxPoints.clear();

maxPoints.push\_back(ChessPos(row, col));

}

else if (scoreMap[row][col] == maxScore) {

maxPoints.push\_back(ChessPos(row, col));

}

}

}

}

int index = rand() % maxPoints.size();

return maxPoints[index];

}

对AI的实现较为复杂，需要考虑到棋盘中的每个点，且敌方我方都要考虑，同时Sleep函数的使用使得游戏有了停顿，更人性化

1. Chess类

1 Chess.h

#pragma once

#include<graphics.h>

#include<vector>

using namespace std;

typedef enum {

CHESS\_WHITE = -1, // 白方

CHESS\_BLACK = 1 // 黑方

} chess\_kind\_t;

struct ChessPos {

int row;

int col;

//利用构造函数进行初始化

ChessPos(int r=0,int c=0):row(r),col(c){}

};

class Chess

{

public:

//利用构造函数初始化

Chess(int gradeSize, int marginX, int marginY, float chessSize);

// 棋盘的初始化

void init();

// 判断在指定坐标位置，是否是有效点击

// 如果是有效点击，把有效点击的位置,保存在参数pos中

bool clickBoard(int x, int y, ChessPos\* pos);

// 在棋盘的指定位置, 落子

void chessDown(ChessPos\* pos, chess\_kind\_t kind);

// 获取棋盘的大小（13线、15线、19线）

int getGradeSize();

// 获取指定位置是黑棋，还是白棋，还是空白

int getChessData(ChessPos\* pos);

int getChessData(int row, int col);

// 判断棋局是否结束

bool checkOver();

private:

IMAGE chessBlackImg;

IMAGE chessWhiteImg;

int gradeSize;//棋盘大小（13，15，17，19）

int margin\_x;//棋盘左侧边界大小

int margin\_y;//棋盘上边缘大小

float chessSize;//棋子大小，即格子大小

//表示当前棋局上棋子的分布情况.0为空白；1为黑子；-1为白子

vector<vector<int>>chessMap;//如chessMap[3][5]表示棋盘第三行第五列的落子情况

bool playerFlag;//表示现在该哪一方落子，true:该黑棋落子； flase:该白棋落子

void updateGameMap(ChessPos\* pos);

bool checkWin();//检查输赢的函数，若胜负已分，则返回true

ChessPos lastPos;//定义最后的落子点

};

2 Chess.cpp

#include "Chess.h"

#include<mmsystem.h>

#include<math.h>

#pragma comment(lib,"winmm.lib")

#include<conio.h>

//使图片边缘透明化的接口

void putimagePNG(int x, int y, IMAGE\* picture)

{

// 变量初始化

DWORD\* dst = GetImageBuffer(); // GetImageBuffer()函数，用于获取绘图设备的显存指针

DWORD\* draw = GetImageBuffer();

DWORD\* src = GetImageBuffer(picture); //获取picture

int picture\_width = picture->getwidth(); //获取picture的宽度

int picture\_height = picture->getheight(); //获取picture的高度

int graphWidth = getwidth(); //获取绘图区的宽度

int graphHeight = getheight(); //获取绘图区的高度

int dstX = 0; //在显存里像素的角标

// 实现透明贴图 公式： Cp=αp\*FP+(1-αp)\*BP ， 贝叶斯定理来进行点颜色的概率计算

for (int iy = 0; iy < picture\_height; iy++)

{

for (int ix = 0; ix < picture\_width; ix++)

{

int srcX = ix + iy \* picture\_width; //在显存里像素的角标

int sa = ((src[srcX] & 0xff000000) >> 24); //0xAArrggbb;AA是透明度

int sr = ((src[srcX] & 0xff0000) >> 16); //获取RGB里的R

int sg = ((src[srcX] & 0xff00) >> 8);

int sb = src[srcX] & 0xff;

if (ix >= 0 && ix <= graphWidth && iy >= 0 && iy <= graphHeight && dstX <= graphWidth \* graphHeight)

{

dstX = (ix + x) + (iy + y) \* graphWidth; //在显存里像素的角标

int dr = ((dst[dstX] & 0xff0000) >> 16);

int dg = ((dst[dstX] & 0xff00) >> 8);

int db = dst[dstX] & 0xff;

draw[dstX] = ((sr \* sa / 255 + dr \* (255 - sa) / 255) << 16)

| ((sg \* sa / 255 + dg \* (255 - sa) / 255) << 8)

| (sb \* sa / 255 + db \* (255 - sa) / 255);

}

}

}

}

Chess::Chess(int gradeSize, int marginX, int marginY, float chessSize)

{

this->gradeSize = gradeSize;

this->margin\_x = marginX;

this->margin\_y = marginY;

this->chessSize = chessSize;

playerFlag = CHESS\_BLACK;

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

vector<int>row;

for (int j = 0; j < gradeSize; j++) {

row.push\_back(0);

}

chessMap.push\_back(row);

}

}

void Chess::init()

{

//创建棋盘窗口

initgraph(897, 895, EX\_SHOWCONSOLE);

//显示棋盘图片

loadimage(0, "res/棋盘2.jpg");

mciSendString("play res/start.wav", 0, 0, 0);

//加入黑棋和白棋的图片

loadimage(&chessBlackImg, "res/black.png", chessSize, chessSize, true);

loadimage(&chessWhiteImg, "res/white.png", chessSize, chessSize, true);

//每轮结束，清空棋盘

for (int i = 0; i < chessMap.size(); i++) {

for (int j = 0; j < chessMap[i].size(); j++) {

chessMap[i][j] = 0;

}

}

//黑棋先行

playerFlag = true;

}

bool Chess::clickBoard(int x, int y, ChessPos\* pos)

{

//计算左上角在第几行第几列

int col = (x - margin\_x) / chessSize;//列

int row = (y - margin\_y) / chessSize;//行

int leftTopPosX = margin\_x + col \* chessSize;//左上角X坐标

int leftTopPosY = margin\_y + row \* chessSize;//左上角Y坐标

int offest = chessSize \* 0.4;//设置极限值

bool ret = false;

int len;//棋子到某个角的距离

do {

len = sqrt((x - leftTopPosX) \* (x - leftTopPosX) + (y - leftTopPosY) \* (y - leftTopPosY));

if (len < offest) {

pos->col = col;

pos->row = row;

if (chessMap[pos->row][pos->col] == 0) {

ret = true;

}

break;

}

//右上角判断

int x2 = leftTopPosX + chessSize;

int y2 = leftTopPosY;

len = sqrt((x - x2) \* (x - x2) + (y - y2) \* (y - y2));

if (len < offest) {

pos->col = col + 1;

pos->row = row;

if (chessMap[pos->row][pos->col] == 0) {

ret = true;

}

break;

}

//左下角判断

x2 = leftTopPosX;

y2 = leftTopPosY + chessSize;

len = sqrt((x - x2) \* (x - x2) + (y - y2) \* (y - y2));

if (len < offest) {

pos->col = col;

pos->row = row + 1;

if (chessMap[pos->row][pos->col] == 0) {

ret = true;

}

break;

}

//右下角判断

x2 = leftTopPosX + chessSize;

y2 = leftTopPosY + chessSize;

len = sqrt((x - x2) \* (x - x2) + (y - y2) \* (y - y2));

if (len < offest) {

pos->col = col + 1;

pos->row = row + 1;

if (chessMap[pos->row][pos->col] == 0) {

ret = true;

}

break;

}

} while (0);

return ret;

}

void Chess::chessDown(ChessPos \* pos, chess\_kind\_t kind)//实现落子功能

{

mciSendString("play res/down7.WAV", 0, 0, 0);

int x = margin\_x + chessSize \* pos->col - 0.5 \* chessSize;

int y = margin\_y + chessSize \* pos->row - 0.5 \* chessSize;

//判断下黑棋还是白棋

if (kind == CHESS\_WHITE) {

putimagePNG(x, y, &chessWhiteImg);

}

else {

putimagePNG(x, y, &chessBlackImg);

}

updateGameMap(pos);

}

int Chess::getGradeSize()

{

return gradeSize;

}

int Chess::getChessData(ChessPos\* pos)

{

return chessMap[pos->row][pos->col];

}

int Chess::getChessData(int row, int col)

{

return chessMap[row][col];

}

bool Chess::checkOver()

{

if (checkWin()) {

Sleep(1500);//停顿

if (playerFlag == false) {

//说明该白子走棋，刚刚走起的是黑子，棋手胜利

mciSendString("play res/不错.mp3", 0, 0, 0);

loadimage(0,"res/胜利.jpg");

}

else {

mciSendString("play res/失败.mp3", 0, 0, 0);

loadimage(0, "res/失败.jpg");

}

\_getch();//暂停

return true;

}

return false;

}

void Chess::updateGameMap(ChessPos\* pos)

{

lastPos = \*pos;

chessMap[pos->row][pos->col] = playerFlag ? CHESS\_BLACK : CHESS\_WHITE;

playerFlag = !playerFlag;//实现黑白方交换

}

bool Chess::checkWin()

{

int row = lastPos.row;

int col = lastPos.col;

//先判断水平方向

for (int i = 0;i < 5;i++) {

if (col - i >= 0 &&

col - i + 4 < gradeSize &&

chessMap[row][col - i] == chessMap[row][col - i + 1] &&

chessMap[row][col - i] == chessMap[row][col - i + 2] &&

chessMap[row][col - i] == chessMap[row][col - i + 3] &&

chessMap[row][col - i] == chessMap[row][col - i + 4])

return true;

}

//垂直方向

for (int i = 0;i < 5;i++) {

if (row - i >= 0 &&

row - i + 4 < gradeSize &&

chessMap[row - i][col] == chessMap[row - i + 1][col] &&

chessMap[row - i][col] == chessMap[row - i + 2][col] &&

chessMap[row - i][col] == chessMap[row - i + 3][col] &&

chessMap[row - i][col] == chessMap[row - i + 4][col])

return true;

}

//"/"方向

for (int i = 0; i < 5; i++)

{

if (row + i < gradeSize &&

row + i - 4 >= 0 &&

col - i >= 0 &&

col - i + 4 < gradeSize &&

chessMap[row + i][col - i] == chessMap[row + i - 1][col - i + 1] &&

chessMap[row + i][col - i] == chessMap[row + i - 2][col - i + 2] &&

chessMap[row + i][col - i] == chessMap[row + i - 3][col - i + 3] &&

chessMap[row + i][col - i] == chessMap[row + i - 4][col - i + 4])

return true;

}

//“\”方向

for (int i = 0; i < 5; i++)

{

if (row - i >= 0 &&

row - i + 4 < gradeSize && col - i >= 0 &&

col - i + 4 < gradeSize &&

chessMap[row - i][col - i] == chessMap[row - i + 1][col - i + 1] &&

chessMap[row - i][col - i] == chessMap[row - i + 2][col - i + 2] &&

chessMap[row - i][col - i] == chessMap[row - i + 3][col - i + 3] &&

chessMap[row - i][col - i] == chessMap[row - i + 4][col - i + 4])

return true;

}

return false;

}

了解了透明贴图的方法，通过对四个大方向的判断来检验棋局是否胜负已分是难点

1. ChessGame类

1 ChessGame.h

#pragma once

#include"AI.h"

#include"Man.h"

#include"Chess.h"

class ChessGame //控制

{

public:

//利用构造函数进行初始化

ChessGame(Man\* man, AI\* ai, Chess\* chess);

void play();//控制游戏开始

private: //添加成员

Man\* man;

AI\* ai;

Chess\* chess;

};

2 ChessGame.cpp

#include "ChessGame.h"

ChessGame::ChessGame(Man\* man, AI\* ai, Chess\* chess)

{

this->man = man;

this->ai = ai;

this->chess = chess;

man->init(chess);

ai->init(chess);

}

void ChessGame::play()

{

chess->init();//对棋盘进行初始化

while (1) {

//由棋手先走

man->go();

if (chess->checkOver()) {

chess->init();//对棋盘进行初始化

continue;

}

//ai后走

ai->go();

if (chess->checkOver()) {

chess->init();//对棋盘进行初始化

continue;

}

}

}

这是棋局的核心控制部分，通过这一部分函数创建了众多接口，在一步步地进行完善

五、收获

**1、透明贴图的方法**

**void putimagePNG(int x, int y, IMAGE\* picture)**

**{**

**// 变量初始化**

**DWORD\* dst = GetImageBuffer(); // GetImageBuffer()函数，用于获取绘图设备的显存指针**

**DWORD\* draw = GetImageBuffer();**

**DWORD\* src = GetImageBuffer(picture); //获取picture**

**int picture\_width = picture->getwidth(); //获取picture的宽度**

**int picture\_height = picture->getheight(); //获取picture的高度**

**int graphWidth = getwidth(); //获取绘图区的宽度**

**int graphHeight = getheight(); //获取绘图区的高度**

**int dstX = 0; //在显存里像素的角标**

**// 实现透明贴图 公式： Cp=αp\*FP+(1-αp)\*BP ， 贝叶斯定理来进行点颜色的概率计算**

**for (int iy = 0; iy < picture\_height; iy++)**

**{**

**for (int ix = 0; ix < picture\_width; ix++)**

**{**

**int srcX = ix + iy \* picture\_width; //在显存里像素的角标**

**int sa = ((src[srcX] & 0xff000000) >> 24); //0xAArrggbb;AA是透明度**

**int sr = ((src[srcX] & 0xff0000) >> 16); //获取RGB里的R**

**int sg = ((src[srcX] & 0xff00) >> 8);**

**int sb = src[srcX] & 0xff;**

**if (ix >= 0 && ix <= graphWidth && iy >= 0 && iy <= graphHeight && dstX <= graphWidth \* graphHeight)**

**{**

**dstX = (ix + x) + (iy + y) \* graphWidth; //在显存里像素的角标**

**int dr = ((dst[dstX] & 0xff0000) >> 16);**

**int dg = ((dst[dstX] & 0xff00) >> 8);**

**int db = dst[dstX] & 0xff;**

**draw[dstX] = ((sr \* sa / 255 + dr \* (255 - sa) / 255) << 16)**

**| ((sg \* sa / 255 + dg \* (255 - sa) / 255) << 8)**

**| (sb \* sa / 255 + db \* (255 - sa) / 255);**

**}**

**}**

**}**

**}**

**2、AI思考如何走棋**

**1）AI对落子点进行评分**

**2）AI根据评分进行“思考”**

**3） AI走棋**

**让AI思考起来使该游戏的重难点**

**3、easyx的用法**

**Easyx可以进行贴图（动画播放、透明贴图），获取按键信息，特效等，并能够进行音乐播放，（mcisendstring(非easyx)）**