面向对象与程序设计——实习四

目录

[实习题目 2](#_Toc11247)

[题目分析及设计思路 2](#_Toc23796)

[1、 任务一（五子棋） 2](#_Toc26693)

[2、 任务二（2048） 3](#_Toc19427)

[3、 任务三（封装） 3](#_Toc11852)

[UML图和设计说明 3](#_Toc17780)

[关键代码和截图 4](#_Toc21056)

[1、 任务1（五子棋） 4](#_Toc19654)

[2、 任务2（2048） 6](#_Toc29543)

[3、任务三（动态链接库） 8](#_Toc4632)

[遇到的问题及解决 12](#_Toc16054)

实习题目

1、 QT 绘图练习。实现五子棋游戏。棋盘上的横行线从上到下用阿拉伯数字 1——15 标记，纵行线则从左到右用英文字母 A——O 按字母顺序标记。在棋盘上有 5 个比较特殊的交叉点，用和棋盘横纵线颜色相同的实心小圆点标示出来。

2、 QT 贴图练习。实现 2048 游戏。在 4x4 棋盘中分布一些带数字的方块，每次控制所有方块向同一个方向运动，两个相同数字的方块撞在一起之后合并成为他们的和，每次操作之后会在空白的方格处随机生成一个 2 ，最终得到一个“2048”的方块就算胜利了。如果 16 个格子全部填满并且相邻的格子都不相同也就是无法移动的话，那么恭喜你，gameover，可以点击 try again。

重新开始。

1. 定义一个 C++三角形类，能够计算三角形的周长和面积，制作成 DLL；分装成 CLR 类库，在 C#中调用并测试；封装成 python 类库，在 python 中调用并测试。

题目分析及设计思路

1. **任务一（五子棋）**

首先，游戏的界面分为棋盘、棋子和行列索引号三个部分，应先绘制棋盘再绘制棋子：

1. 五子棋的棋盘由14×14规格的网格组成，通过循环绘制带边框的正方形即可，即 得到整个棋盘格。
2. 棋子分为黑白两色，通过绘制对应颜色的圆即可表示。每个棋子具有位置（行列号） 和颜色的属性，可以通过vector数组动态储存所有已存在棋盘上的棋子信息。
3. 行列索引号1...15和A...O通过QLabel控件设置，由于横纵各15个索引，逐一设置 比较繁琐，可以建立数组存储这30个标签，遍历数组即可。

接下来是游戏的实现逻辑：

1. 重写鼠标点击函数以实现“下棋”这一动作，对鼠标点击之处经过四舍五入，得到 对应的落子点。若该落点处不存在棋子，就在以落点为圆心绘制执棋方颜色的圆。并构造一 个新的棋子实例，装入vector数组。
2. 每落一个棋子，都应检查是否终结战局。以当前所下棋子为中心，向周围八个方 向 分别检查相邻的同色棋子个数，属于同一直线的两方向上的个数相加。若相加结果达到 4（当前棋子不计入）即代表对战结束，赢家为当前的执棋方，弹出信息框后，玩家点击确 定后，清空棋子数组，自动进入新的棋局；若没有达到4，说明对战未结束，应该交换棋权。
3. **任务二（2048）**

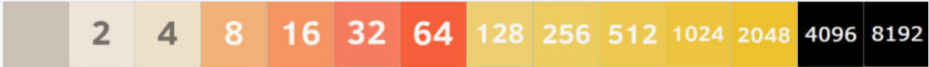


图 1

首先，为每一个色块贴图。所提供的位图分辨率是1255 x 90，共包含14个格子，则每个格子约算90×90像素。

1. 将2048.bmp放到项目的.pro文件所在目录下。创建一个长度为16的QPixmap二 维 数组，16个位图都加载自2048.bmp。
2. 游戏界面由16个方块组成。展示方式是通过QPainter的drawPixmap函数嵌套循环 绘制。利用QRect选定需要显示出来的位图的某一部分。读取位图的起始位置根据游戏 进 展变化而变化。

游戏实现逻辑与任务一的五子棋实现流程比较相似：

1. 创建int类型的数组pieces[4][4]，数组中存放当前色块存放的数值是2的多少次幂（0,1,2,3...,13），每个色块位图的显示也将依赖于该数组，界面上在QRect(Edge+i\*95,Edge+j\*95,90,90)处绘制的色块正好对应起始点是90\*pieces[i][j]的位图。使用0初始化整个数组（即对应位图最开头的空白格）。
2. 左右移动是通过重绘来模拟完成的，并非真的位图在界面上平移。重写键盘函数，按上下左右箭头，表示向不同方向的“移动”。这一步通过switch语句完成，每次读取到有效按键，都将跳转到相应函数执行移动操作Up( )，Down( )，Right( )，Left( )。
3. 以Up( )为例，在函数中，分为移动->相消->移动3个步骤。首先从上到下遍历pieces[4][4]的所有元素，把移动方向上最远端的空格赋予当前元素的pieces指数值然后将当前元素置0；接下来从上到下遍历pieces[4][4]前3行元素，若存在某元素和元素下方的元素值相同，则当前元素+1，下方元素置0。若有元素==11则游戏胜利；最后再进行一次移动的操作，防止经过相消后移动不完全的情况出现（例如，2222—>4040，事实上应该是4400）。
4. 每一次按键，完成移动相消后。在剩余的空格中随机产生一个2，并遍历pieces数组判断当前是否满格。若已满格且不存在相邻元素，则判定游戏失败，弹出提示框。重新初始化pieces数组和各变量，开始新的游戏。
5. **任务三（封装）**
6. 实现三角形类；
7. 封装C++代码为动态链接库DLL；
8. 用CLR封装并在C#中调用；
9. 在Python中利用ctypes库调用x64位生成的DLL；

UML图和设计说明

1. Piece类是棋子类，包含数据成员position和isblack。Position用于储存棋子的位置（QPoint），isblack表示棋子是否为黑色，由于棋子只有黑白两色，因此直接设置该变量为bool类型变量，方便了后续对颜色的判断；mainwindow类的数据成员包含QVector数组对象（用于储存棋子）、当前执棋方Cur\_round（黑色为true，白色为false）。函数成员则包含了绘制（棋盘和棋子）、判断、鼠标函数的重写。

mainwindow类中用到了Piece类型的变量，所以关系为依赖关系。

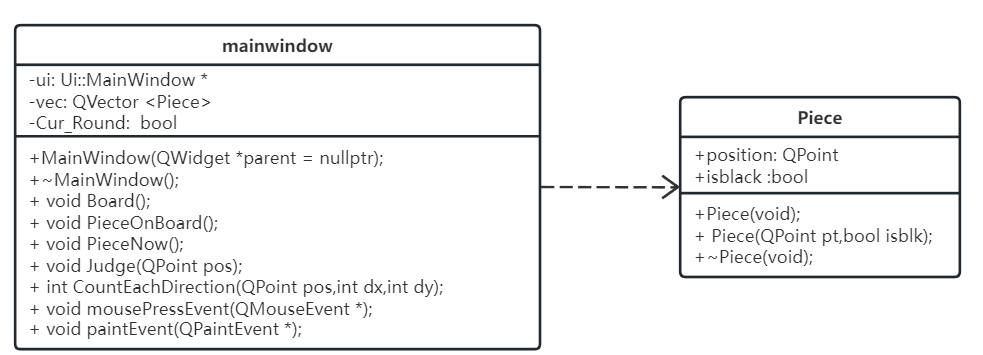


图 2

1. 仅有mainwindow类，UML图略。mainwindow类的结构如下，程序函数的功能见注释：

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

public:

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

~***MainWindow***();

void **drawPieces**();//绘制16个色块

void **BackGround**();//绘制白色背景

void ***paintEvent***(QPaintEvent \*e);//重写绘制函数

void ***keyPressEvent***(QKeyEvent \*e);//重写键盘函数

void **Rand2**();//随机在空格处产生2

void **Judge**();//以及满格的情况下 判断是否还能继续

void **Init**();//初始化 包括pieces数组清零 变量重置

//处理键盘响应的函数:

void **Down**();

void **Up**();

void **Right**();

void **Left**();

Ui::MainWindow \*ui;

int Edge=50;//边缘

//存储16个棋子上的数字是2的多少幂次 第一个空白格为0

//对应位图的显示格数

int pieces[4][4];

QPixmap \* imgs[4][4];

bool over; //游戏结束标识 主要为了 失败后按键无效

bool win; //胜利标识

};

关键代码和截图

1. **任务1（五子棋）**

界面绘制逻辑是：a.绘制棋盘；b.绘制天元和小星；c.绘制棋盘上已存在的棋子；d.绘制随鼠标而移动的当前棋子。其中，棋子的颜色是棋子的属性，棋子储存在动态数组vec中。

//更新绘制界面

void MainWindow::***paintEvent***(QPaintEvent \*e){

Board();

PieceOnBoard();

PieceNow();

update();

}

除去绘制工作外，核心代码逻辑在于判断对局结果，在以下两个函数中实现。

void MainWindow::**Judge**(QPoint pos){

qDebug()<<"Judge"<<endl;

//定义8个方向 顺时针从正右方向开始

int count1=CountEachDirection(pos,1,0);

int count2=CountEachDirection(pos,1,1);

int count3=CountEachDirection(pos,0,1);

int count4=CountEachDirection(pos,-1,1);

int count5=CountEachDirection(pos,-1,0);

int count6=CountEachDirection(pos,-1,-1);

int count7=CountEachDirection(pos,0,-1);

int count8=CountEachDirection(pos,1,-1);

qDebug()<<count1<<count5<<count1+count5<<endl;

qDebug()<<count2<<count6<<count2+count6<<endl;

qDebug()<<count3<<count7<<count3+count7<<endl;

qDebug()<<count4<<count8<<count4+count8<<endl;

//两两配对 检查是否五子连线

if(count1+count5>3||count2+count6>3||count3+count7>3||count4+count8>3){

//信息对话框

if(Cur\_Round){

QMessageBox::information(NULL,"OVER","黑方获胜", QMessageBox::Ok);

}

else{

QMessageBox::information(NULL,"OVER","白方获胜", QMessageBox::Ok);

}

//战局结束之后 新开一局

vec.clear();//清空所有棋子

return;

}

//若没有结束

Cur\_Round=!Cur\_Round;//交换棋权

}

//计算每个方向的相邻棋子数

int MainWindow::**CountEachDirection**(QPoint pos,int dx,int dy){

QPoint nextPoint;

nextPoint.setX(pos.x()+dx);

nextPoint.setY(pos.y()+dy);

int num=0;//此方向上的同色相邻棋子数

bool nextExist;//此方向上的下一个点是否存在

do{

nextExist=false;

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

{

if(vec[i].isblack==Cur\_Round&&vec[i].position==nextPoint){

num++;

nextExist=true;

break;//存在就可以跳出循环了

}

}

nextPoint.setX(nextPoint.x()+dx);

nextPoint.setY(nextPoint.y()+dy);

}while(nextExist);

return num;

}

运行截图如下所示，图 3是棋盘的界面，可见棋盘的横纵编号和天元小星。图 4是游戏示例，可见黑子率先连成五个，对战结束并提示“黑方胜利”。

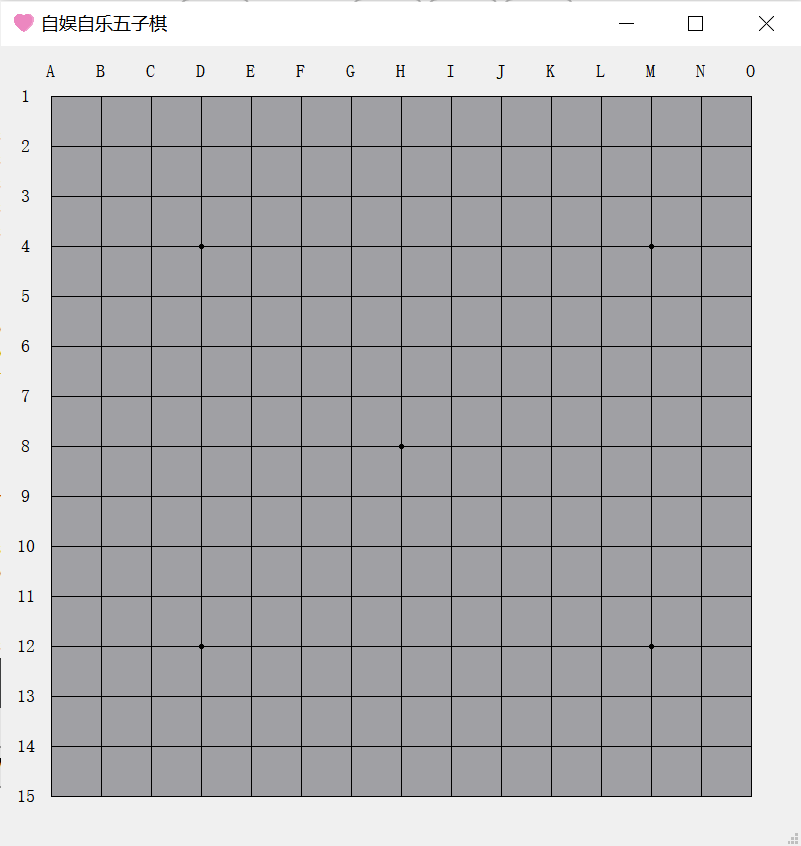
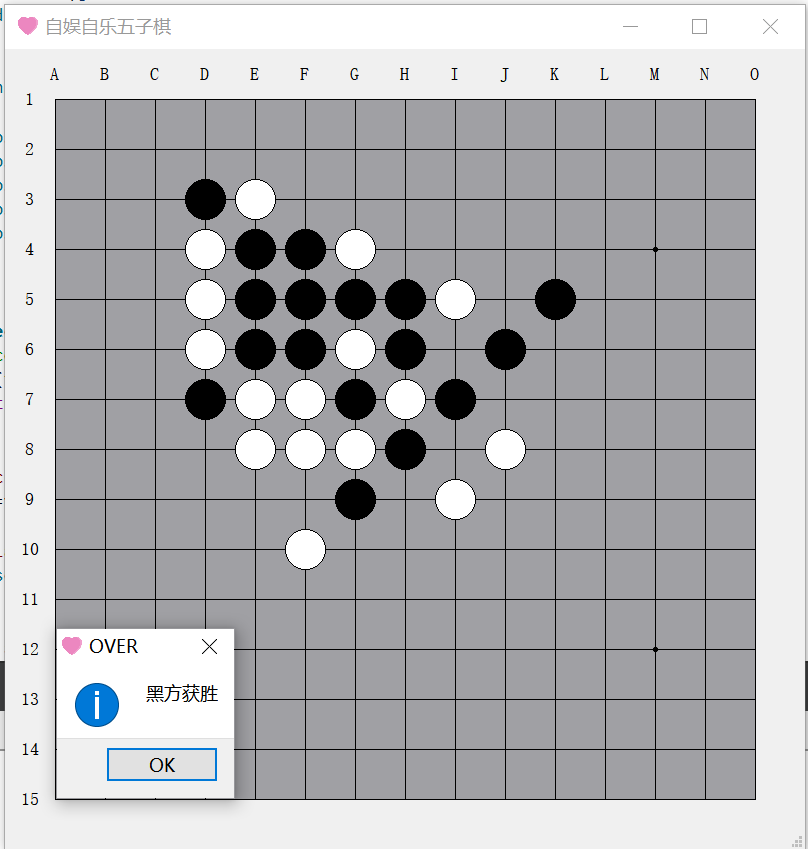
 

图 3  图 4

1. **任务2（2048）**

绘制函数如下。重绘函数***paintEvent***(QPaintEvent \*e)中包含绘制背景的函数 BackG

round()、绘制色块的函数**drawPieces**()和updata()更新函数。BackGround()和**drawPieces**()二者的实现也如下所示。

除了绘制以外，核心逻辑在于处理玩家操作。重写响应键盘函数正是是处理玩家行为消息的函数，首先识别游戏状态是否结束。游戏在进行的状态下，若识别按键为WSAD之一，则跳转至相应的处理函数。若为其他按键视为无效。处理完移动和消除操作后，若胜利，则提示玩家胜利；若游戏仍在继续，则在此时的空格子中随机选取一个赋值为1（对应2的1次方：2）。

其中，具体对玩家按键的处理函数是Up( ),Down( ),Right( ),Left( )。下文代码以Left()函数为例，包括三个循环嵌套，分别对应移动、相消、移动。

void MainWindow::***paintEvent***(QPaintEvent \*e){

BackGround();

drawPieces();

update();

}

void MainWindow::**drawPieces**(){

QPainter p(this);

// pixmap = pixmap.scaled(392,392.00/pixmap.width()\*pixmap.height());//按照宽度等比例放大

for(int i=0;i<4;i++)

{

for(int j=0;j<4;j++)

{

QPixmap img=\*imgs[i][j];

p.drawPixmap(QRect(Edge+i\*95,Edge+j\*95,90,90),img,QRect(pieces[i][j]\*90,0,90,90));//第一个参数是在窗口显示的位置 第三个参数是位图的显示矩形

}

}

}

//绘制白色背景

void MainWindow::**BackGround**(){

QPainter p(this);

p.setBrush(Qt::white);

p.setPen(Qt::white);

p.drawRect(20,20,435,435);

}

void MainWindow::***keyPressEvent***(QKeyEvent \*e)

{

if(over){

return;//已结束 按键无效

}

switch (e->key()) {

case Qt::Key\_Up: Up(); break;

case Qt::Key\_Down: Down(); break;

case Qt::Key\_Right: Right(); break;

case Qt::Key\_Left: Left(); break;

default: return;//除了4个箭头 其他按键不起作用

}

if(win==true){

QMessageBox::information(NULL,"GAME OVER","游戏胜利", QMessageBox::Ok);

Init();

return;

}

//每移动一次 随机生成新的2 然后judge是否结束

Rand2();//随机在空格子中生成2

}

void MainWindow::**Left**(){

qDebug()<<"left"<<endl;

//实现移动

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

for(int j=0;j<4;j++){

//为0跳过 不做操作

if(pieces[i][j]!=0){

//找到前面最远的空位[m][j]

for(int m=0;m<3;m++){

if(pieces[m][j]==0){

pieces[m][j]=pieces[i][j];

pieces[i][j]=0;

break;

}

}

}

}

}

//实现相邻的同种色块相消 从上往下消

for(int i=0;i<3;i++){

for(int j=0;j<4;j++){

if(pieces[i][j]==0){continue;};

if(pieces[i][j]==pieces[i+1][j])//相消

{

pieces[i][j]+=1;

pieces[i+1][j]=0;

}

if( pieces[i][j]==11){

win=true;

over=true;

}

}

}

//相消完成后再衔接一次移动 否则会出现 2222->4040 应该是4400

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

for(int j=0;j<4;j++){

//为0跳过 不做操作

if(pieces[i][j]==0){continue;}

//找到前面最远的空位[m][j]

for(int m=0;m<3;m++){

if(pieces[m][j]==0){

pieces[m][j]=pieces[i][j];

pieces[i][j]=0;

break;

}

}

}

}

}

运行结果如下，图 5是打开游戏的初始化界面；图 6是胜利画面（由于游戏水平有限未达到2048的战绩，修改程序的胜利条件为pieces[i][j]==2，即达到4就胜利）；图 7是游戏失败的界面，此界面只发生在下述情况：所有格子都被填满而且无相邻两格的数值相同。

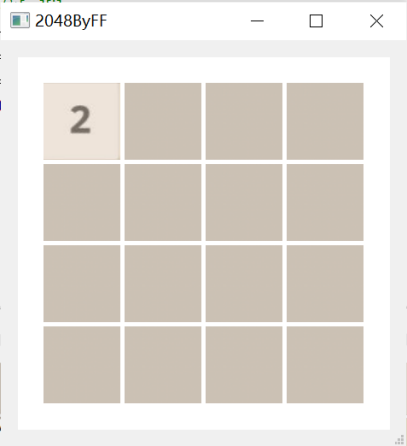
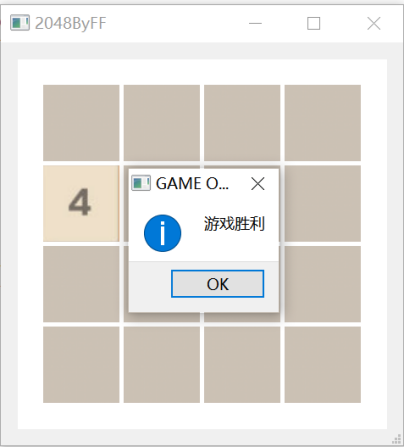
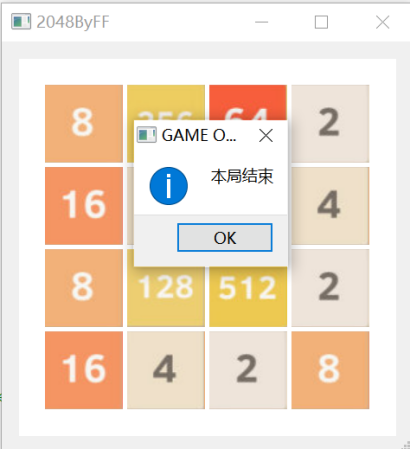
  

图 5  图 6  图 7

1. **任务三（动态链接库）**

首先创建DLL项目，创建CTriangle类，分为Triangle.h和Triangle.cpp。生成成功后将在文件夹中找到Triangle.dll和Triangle.lib，如图 8所示。

**Triangle.h**

#pragma once

#ifdef TRIANGLE\_EXPORTS

#define MY\_DLL\_EX \_\_declspec(dllexport)

#else

#define MY\_DLL\_EX \_\_declspec(dllimport)

#endif

class MY\_DLL\_EX CTriangle

{

public:

CTriangle(double a, double b, double c);

double Perimeter();

double Area();

private:

double a, b, c;

};

**Triangle.cpp**

#include"pch.h"

#include<cmath>

#include"Triangle.h"

CTriangle::CTriangle(double a, double b, double c):a(a),b(b),c(c)

{

}

double CTriangle::Perimeter()

{

return a + b + c;

}

double CTriangle::Area()

{

double p = 0.5\*(a + b + c);

return sqrt(p\*(p - a)\*(p - b)\*(p - c));

}

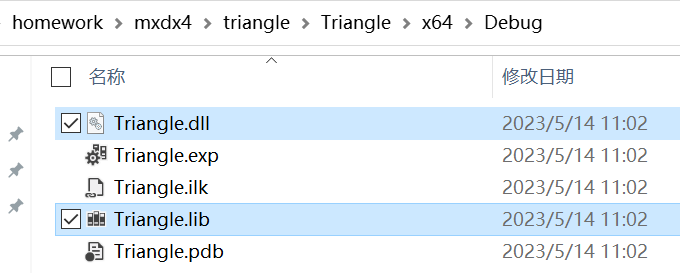


图 8

创建CLR项目TRGLibrary，封装上述CTriangle类，在TRGLibrary.h中引入Triangle.h的相对路径，并创建命名空间Trianglelib。生成成功后将在文件夹中找到TRGLibrary相关dll文件，如图 9所示。

#pragma once

#include"../Triangle/Triangle.h"

using namespace System;

namespace Trianglelib {

public ref class Triangle

{

public:

//一切行动用指针：

Triangle(double a, double b, double c) { ptr = new CTriangle(a, b, c); }

~Triangle() { delete ptr; }

double Area() { return ptr->Area(); }

double Perimeter() { return ptr->Perimeter(); }

private:

CTriangle \*ptr;//指向已写好的Traingle对象

};

}

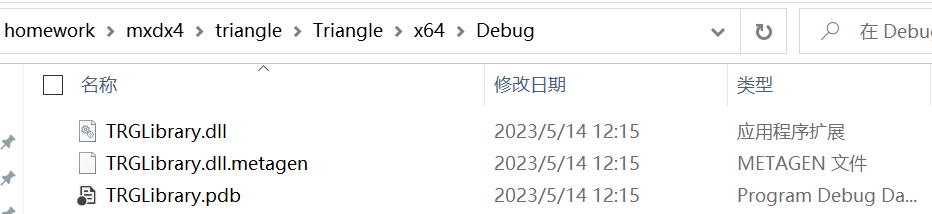


图 9

创建一个C#项目，将TRGLibrary添加至引用，根据需要修改和添加库目录和包含目录等。在Program.cs中用using关键字添加命名空间Trianglelib。创建一个以3，4，5为边长的三角形并计算边长。成功生成一个exe可执行文件。将其在文件夹中打开，得到边长结果12，如图 11所示。

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Trianglelib;

namespace ConsoleApp1

{

class Program

{

static void Main(string[] args)

{

Triangle c = new Triangle(3, 4, 5);

System.Console.WriteLine(c.Perimeter());

}

}

}

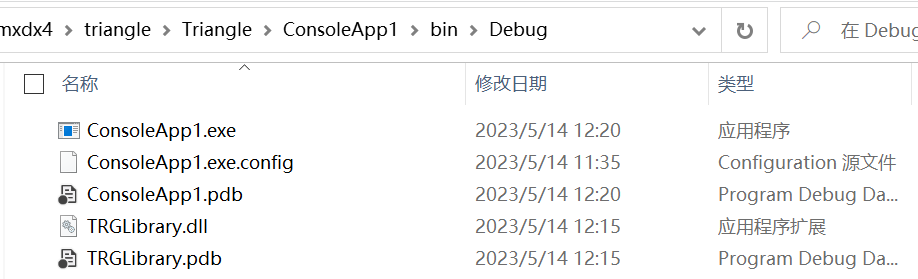


图 10

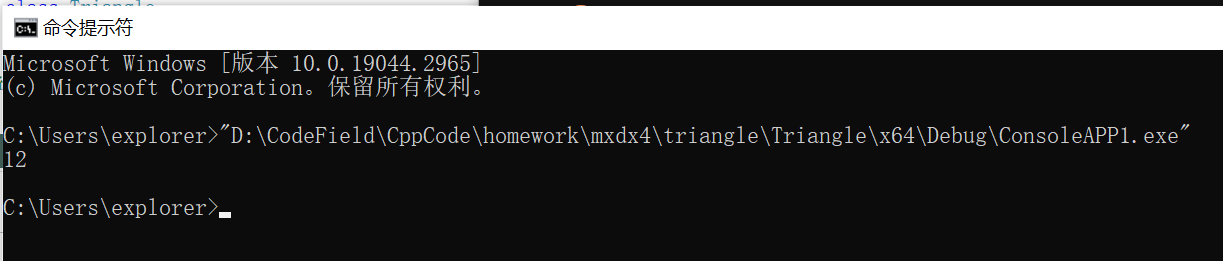


图 11

接下来为Python生成dll文件。创建DLL项目Dlltest，添加t.h，t.cpp，testcpp.cpp。运行成功后，将生成的Dlltest.dll。创建test.py文件，把Dlltest.dll复制到python文件同一目录下。结果如图 12所示。

**t.h**

#pragma once

extern "C" {

class CTriangle

{

public:

CTriangle(double a, double b, double c);

double Perimeter();

double Area();

private:

double a, b, c;

};

}

**t.cpp**

#include"pch.h"

#include<cmath>

#include"t.h"

CTriangle::CTriangle(double a, double b, double c) :a(a), b(b), c(c)

{

}

double CTriangle::Perimeter()

{

return a + b + c;

}

double CTriangle::Area()

{

double p = 0.5\*(a + b + c);

return sqrt(p\*(p - a)\*(p - b)\*(p - c));

}

**testcpp.cpp**

#include"pch.h"

#define DLLEXPORT extern "C" \_\_declspec(dllexport)

#include "t.h"

DLLEXPORT int getArea() {

CTriangle t1 = CTriangle(3,4,5);

double Area = t1.Area();

return Area;

}

DLLEXPORT int getPerimeter() {

CTriangle t1 = CTriangle(3, 4, 5);

double Perimeter = t1.Perimeter();

return Perimeter;

}

**test.py**

import ctypes

# Load the CTriangle.dll library

ct = ctypes.CDLL('./Dlltest.dll')

area=ct.getArea()

perimeter=ct.getPerimeter()

print('area',area)

print('perimeter',perimeter)

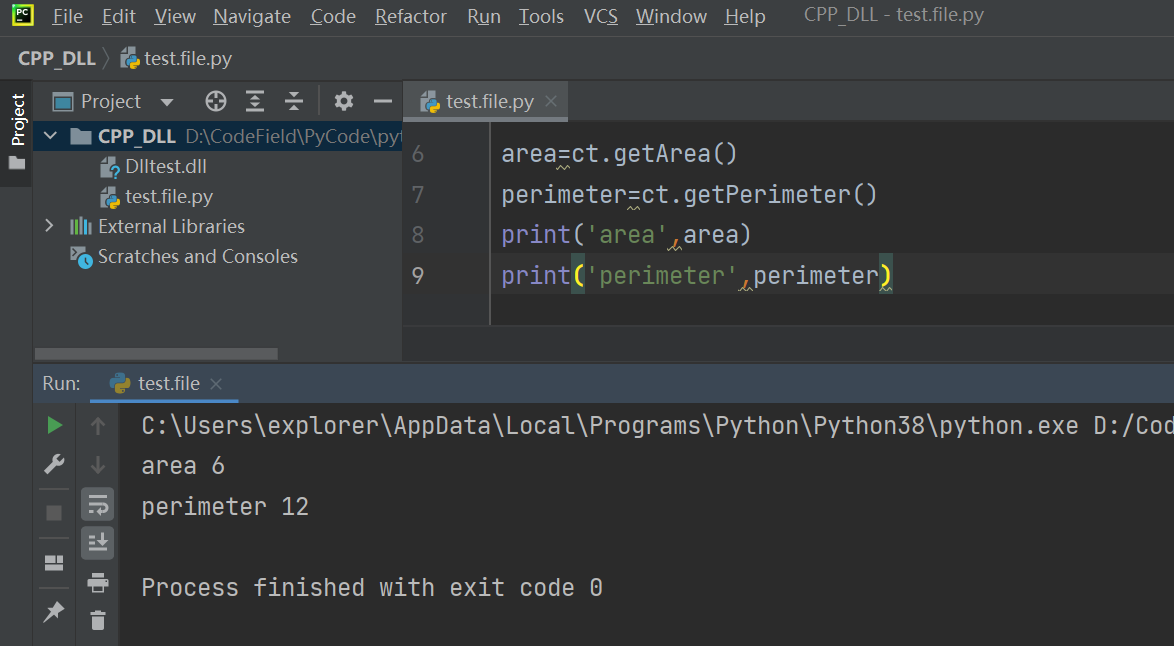


图 12

遇到的问题及解决

1. 任务1中，找不到方法实现按照矩形裁剪的方式显示位图，并让位图像跑马灯一样后移。

解决：查阅文档找到QRect作为参数形式的drawPixmap()函数形式，实现矩形显示；将pieces数组的元素值设为2的幂数，而不是本身数值大小。QRect的参数与pieces元素数值一一对应，通过重绘函数即可实现位图显示随游戏进程的实时变化。

1. 完成前两个任务后，试图将程序分享给他人试玩，但由于他人并不具备程序运行所需配置，因此无法直接运行exe文件。

解决：参考 <https://www.jianshu.com/p/0806c5350bdd> 生成dll文件后，打包给其他电脑，

1. 经过与他人分享程序，被找到任务一中的BUG：五子棋在窗体内除了已存在棋子的点位，其余任何位置都可以落子，以至于棋子甚至可以超出棋盘。

解决：重新添加落棋子的判断语句，在鼠标函数重写中限制落子只能位于棋盘内部。

1. 遇到找不到库文件报错失败。

解决：在项目属性中的VC++目录中修改添加所需库的路径。首先在电脑搜索该库的位置，若存在，则复制文件路径到C++项目属性的库目录中。也存在电脑中本身不存在所需库文件的情况，如下图。经查阅，是因为未安装python3.8的Debug版本。此时不需要卸载现有python，只需在控制面板打开程序并更改，勾选Debug选项。再将库目录更新即可。

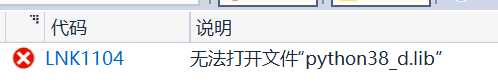
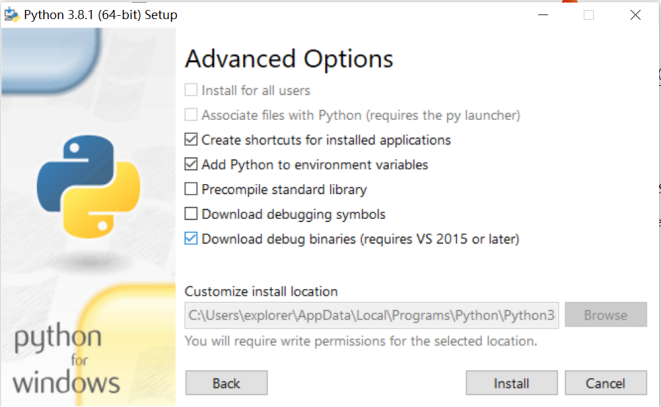
 

图 13 图 14

5、python中调用DLL文件的方式：<https://www.cnblogs.com/FHC1994/p/11421229.html>

此外，过程中还遇到很多其他问题，没有及时一一记录。