## 第一题

目前已经整理成.exe文件，附在了作业压缩包中；

PS：代码在ＶＳ中运行时需要包含ｇｒａｐｈｉｃｓ．ｈ头文件，且需要设置多字节字符集

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<stdio.h>

#include<iostream>

#include<string.h>

#include<stdlib.h>

#include<graphics.h> //引入了graphics图形库用以优化界面

using namespace std;

constexpr auto BLACK\_CHESS = 3;;

constexpr auto WHITE\_CHESS = 2;;

constexpr auto EMPTY = 0;;

constexpr auto CHESS\_WIDTH =25;;

//瓦片地图类，用于graphics中存放贴图和加载贴图

class TileMap //用于制作棋盘的瓦片地图

{

private :

IMAGE MAP001;

IMAGE MAP002;

IMAGE MAP003;

IMAGE MAP004;

IMAGE MAP005;

IMAGE MAP006;

IMAGE MAP007;

IMAGE MAP008;

IMAGE MAP009;

public:

//加载所需要的图片资源

void LoadImg()

{

loadimage(&MAP001, "../res/MAP001.png");

loadimage(&MAP002, "../res/MAP002.png");

loadimage(&MAP003, "../res/MAP003.png");

loadimage(&MAP004, "../res/MAP004.png");

loadimage(&MAP005, "../res/MAP005.png");

loadimage(&MAP006, "../res/MAP006.png");

loadimage(&MAP007, "../res/MAP007.png");

loadimage(&MAP008, "../res/MAP008.png");

loadimage(&MAP009, "../res/MAP009.png");

}

//生成瓦片地图

void MapInit(int chessWidth)

{

chessWidth++;

initgraph(48 \* chessWidth, 48 \* chessWidth);

//填充中心部分

for (int i = 1; i < chessWidth - 1; i++)

{

for (int j = 1; j < chessWidth - 1; j++)

{

putimage(48 \* i, 48 \* j, &chessmap.MAP001);

}

}

chessWidth--;

//填充四个边角

{

putimage(0, 0, &chessmap.MAP006);

putimage(48 \* chessWidth, 48 \* chessWidth, &chessmap.MAP008);

putimage(0, 48 \* chessWidth, &chessmap.MAP009);

putimage(48 \* chessWidth, 0, &chessmap.MAP007);

}

//填充四条边

{

for (int i = 1; i < chessWidth; i++)

{

putimage(48 \* i, 0, &chessmap.MAP002);

}

for (int i = 1; i < chessWidth; i++)

{

putimage(48 \* i, 48\*chessWidth, &chessmap.MAP005);

}

for (int i = 1; i < chessWidth; i++)

{

putimage(0, 48\*i, &chessmap.MAP003);

}

for (int i = 1; i < chessWidth; i++)

{

putimage(48\*chessWidth, 48 \* i, &chessmap.MAP004);

}

}

}

}chessmap;

//棋盘类，对应的有棋子的绘制和最终胜利条件的检测

class ChessBoard

{

public:

int chessWidth;

int chessboard[CHESS\_WIDTH][CHESS\_WIDTH] = {};

int chessNumber;

//根据鼠标点击的位置和回合数绘制相应的棋子的位置

void ChessPut(int gameround)

{

MOUSEMSG m;

while (true)

{

m = GetMouseMsg();

int pos = MouseHitXY(m);

if (m.uMsg == WM\_LBUTTONDOWN)

{

if (pos == 0) {}

else

{

int flag = 1;

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

{

if (pos == SaveChess[i]) flag = 0;

}

if (flag == 1)

{

SaveChess[gameround] = pos;

PrintChess(pos, gameround);

break;

}

}

}

}

}

//胜利条件的判定

bool VictoryDef() const

{

int Victory = 0;

//水平检测

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

{

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

{

if (chessboard[i][j] == chessboard[i + 1][j] && chessboard[i][j] == chessboard[i + 2][j] && chessboard[i][j] == chessboard[i + 3][j] && chessboard[i][j] == chessboard[i + 4][j] && chessboard[i][j] >1)

Victory = 1;

}

}

//竖直检测

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

{

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

if (chessboard[j][i] == chessboard[j][i + 1] && chessboard[j][i] == chessboard[j][i + 2] && chessboard[j][i] == chessboard[j][i + 3] && chessboard[j][i] == chessboard[j][i + 4] && chessboard[j][i] >1)

Victory = 1;

}

//斜向检测

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

{

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

if (chessboard[j][i] == chessboard[j + 1][i + 1] && chessboard[j][i] == chessboard[j + 2][i + 2] && chessboard[j][i] == chessboard[j + 3][i + 3] && chessboard[j][i] == chessboard[j + 4][i + 4] && chessboard[j][i] > 1)

Victory = 1;

}

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

{

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

{

if (chessboard[i][j] == chessboard[i + 1][j + 1] && chessboard[i][j] == chessboard[i + 2][j + 1] && chessboard[i][j] == chessboard[i + 3][j + 3] && chessboard[i][j] == chessboard[i + 4][j + 4] && chessboard[i][j] > 1)

Victory = 1;

}

}

for (int i = chessWidth; i >= 0; i--)

{

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

{

if (chessboard[j][i] == chessboard[j + 1][i - 1] && chessboard[j][i] == chessboard[j + 2][i - 2] && chessboard[j][i] == chessboard[j + 3][i - 3] && chessboard[j][i] == chessboard[j + 4][i - 4] && chessboard[j][i] > 1)

Victory = 1;

}

}

//返回gameover的值

if (Victory == 1) return false;

else return true;

}

private:

//处理鼠标点击位置的信息

int MouseHitXY(MOUSEMSG m) const

{

int eps = 10;

int HitX = 0,HitY=0;

//x值处理

for (int i = 1; i <= chessWidth; i++)

{

if (m.x < 48 \* i + eps && m.x>48 \* i - eps)

HitX = i;

}

for (int i = 1; i <= chessWidth; i++)

{

if (m.y < 48 \* i + eps && m.y>48 \* i - eps)

HitY = i;

}

if (HitX && HitY)

return (HitX \* 100 + HitY);

else

return 0;

}

//具体画出黑色和白色的点，并且保存点的数据；

void PrintChess(int pos,int gameround)

{

int HitX = pos / 100, HitY = pos % 100;

if (gameround % 2 == 0)

{

setfillcolor(BLACK);

chessboard[HitX][HitY] = BLACK\_CHESS;

}

else

{

setfillcolor(WHITE);

chessboard[HitX][HitY] = WHITE\_CHESS;

}

solidcircle(48 \* HitX, 48 \* HitY, 20);

}

int SaveChess[200];

}Board;

//玩家类

class Player

{

public:

Player(char\* name) //player的构造函数

{

PlayerName = name;

}

void NamePrint()

{

cout << PlayerName;

}

private:

char\* PlayerName;

};

int main()

{

//初始化加载游戏资源

bool gameover = 1;

int GameRound = 0;

chessmap.LoadImg();

cout << "请输入棋盘的大小（大小不超过16）" << endl; //读取棋盘的大小

scanf("%d", &Board.chessWidth);

//登记相应的玩家信息

char a[20],b[20];

printf("Player1，请输入您的名字：\n");

scanf("%s", a);

Player Player1(a);

Player1.NamePrint();

cout << ",您执黑，先行。"<< endl;

printf("Player1，请输入您的名字：\n");

scanf("%s", b);

Player Player2(b);

Player1.NamePrint();

cout << ",您执白，后手。" << endl << endl;

Sleep(1000);

cout << "即将进入游戏" << endl << endl;

Sleep(3000);

//生成对应的棋盘

chessmap.MapInit(Board.chessWidth);

//开始游戏，进入游戏循环

while (gameover)

{

Board.ChessPut(GameRound);

gameover = Board.VictoryDef();

GameRound++;

}

if (!gameover)

{

if (GameRound % 2 == 0)

{

cout << "白字获胜" << endl;

cout << "congratulations！";

Player2.NamePrint();

}

else

{

cout << "黑子获胜" << endl;

cout << "congratulations！";

Player1.NamePrint();

}

}

getchar();

Sleep(1000);

closegraph();

return 0;

}

## 第二题

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<iostream>

#include<stdio.h>

using namespace std;

//构造studentData类用于存放数据

//根据题目的要求和需要，这里使用浅拷贝即可实现要求的功能

class Student\_Data

{

public:

//构造函数调用

//这里意外报错，后来发现是由于VS版本较高导致的安全性问题，无法直接使用char\*，而是需要使用const char

Student\_Data(int id, const char\* name, double score) :StudentID(id), StudentName(name), StudentScore(score) {}

void test();

void max(Student\_Data list[], int num)

{

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

{

int flag = 0;

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

{

if (list[i].StudentScore < list[j].StudentScore)

flag = 1;

}

if (flag == 0)

{

Student\_Best = list[i];

}

}

}

//打印学生信息的函数

void Student\_Data\_Print()

{

cout << StudentID << " " << StudentName << " " << StudentScore << endl;

}

private:

int StudentID;

const char\* StudentName;

double StudentScore;

}Student\_Best(0,"null",0.0);

void Student\_Data::test()

{

cout << StudentID << endl;

}

int main()

{

//这里随即编造了一些数据

Student\_Data Student\_List[10] =

{

Student\_Data(1001,"xiaoming",98.7),

Student\_Data(1002,"xiaohong",93.5),

Student\_Data(1003,"xiaogang",97.3),

Student\_Data(1004,"xiaoli",95.4),

Student\_Data(1005,"xiaozhang",99.0),

Student\_Data(1006,"xiaohua",98.6),

Student\_Data(1007,"xiaomahu",91.2),

Student\_Data(1008,"xiaoqiang",94.7),

Student\_Data(1009,"xiaoliu",99.5),

Student\_Data(1010,"xiaoma",96.5)

};

Student\_List[10].max(Student\_List, 10); //这里显示读取了无效数据，大概可能是由于读取的数据中仅仅使用到了score

Student\_Best.Student\_Data\_Print();

}

## 选做题（１）

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<iostream>

#include<stdio.h>

using namespace std;

class Product

{

public:

Product(int id, int num, int price) :EmployeeID(id), ProductNum(num), ProductPrice(price) {}

//声明静态变量总价格和总的数量

static double Total;

static int TotalNumber;

void total()

{

Total += ProductNum \* ProductPrice;

TotalNumber += ProductNum;

}

static double PriceAverage();

private:

int EmployeeID;

int ProductNum;

double ProductPrice;

};

//定义静态函数——用于求平均值

double Product::PriceAverage()

{

return (Total / TotalNumber);

}

//初始化静态变量

double Product::Total = 0;

int Product::TotalNumber = 0;

int main()

{

Product product01[3]

{

Product(101,5,23.5),

Product(102,12,24.5),

Product(103,100,21.5)

};

int n = 3;

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

{

product01[i].total();

}

cout << "总销售款为:" << Product::Total << "元" << endl;

cout << "平局销售单价为:" << Product::PriceAverage() << "元" << endl;

return 0;

}