高级语言程序设计

实验报告

南开大学 计算机科学与技术

张祯颉

2213919

计算机科学卓越班

2023 年 5 月 12 日

高级语言程序设计大作业实验报告

目录

作业题目

图形化小游戏——一款类似 Slay-Antiyoy 的战棋小游戏

开发软件

Visual Studio 2022，EasyX 库

课题要求

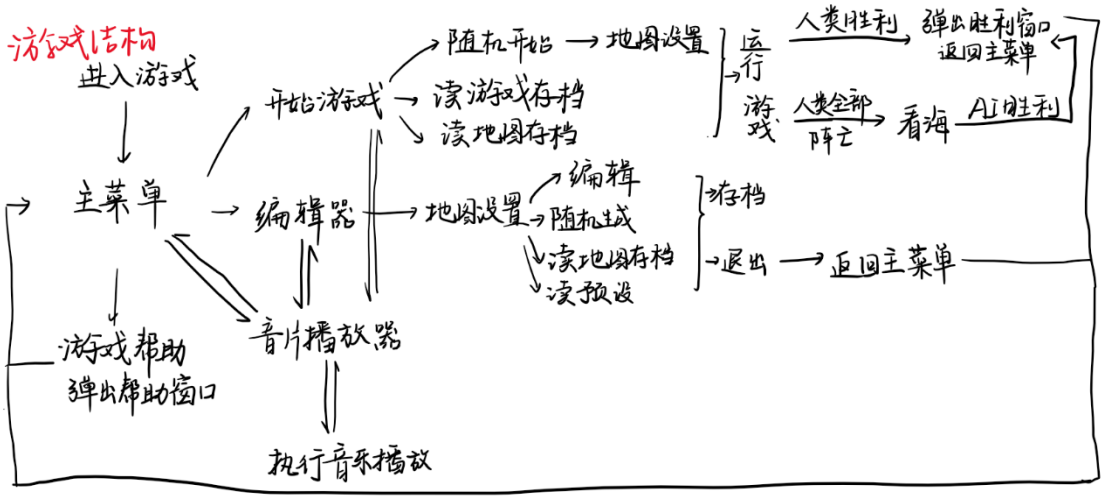
1）基于 C++的图形化

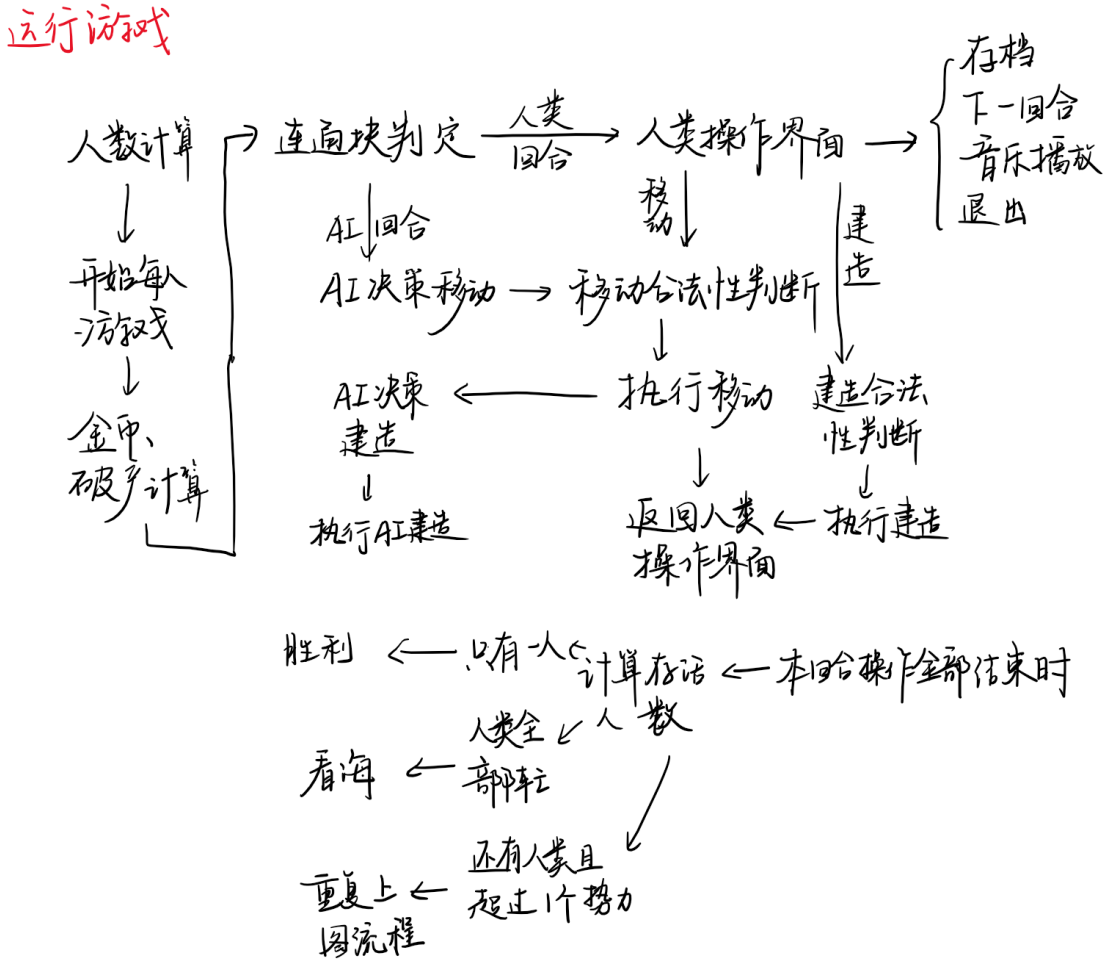
2）其他要要求不限

主要流程

运行流程：

游戏由四大部分构成：进行游戏、地图编辑器、音乐播放器、游戏帮助，三部分并行运行在进行游戏和地图编辑器大类中还有子类。流程图如下：

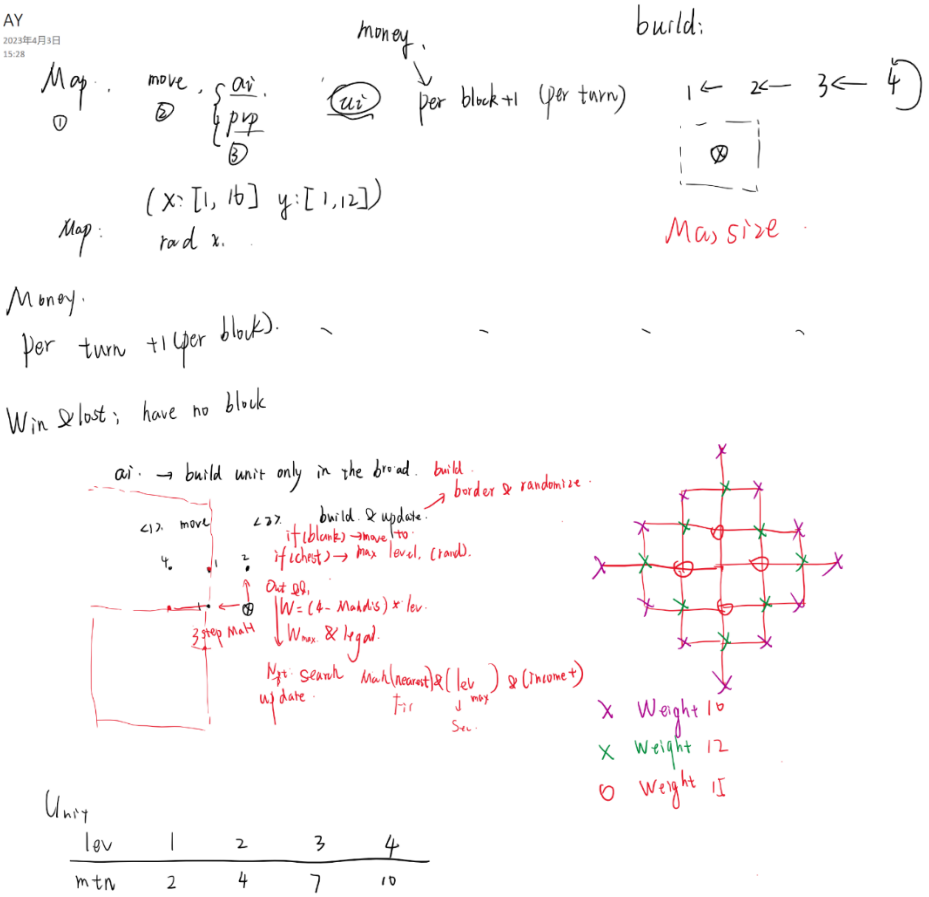




开发流程：

立项和初期规划：

确定制作战棋游戏大概于 2023 年 4 月初。由于 Yiotro 在游戏 Slay 基础上开发的 Antiyoy（几乎）是仅有的兼具看海模式（即没有玩家，全程 ai 对决，人类观看）、地图编辑器、自由选择人数单机多人模式于一体的战棋游戏，且其的游戏规则极其简单，熟练掌握却很难（尤其是高手对局时），用作者 Yiotro 的话来说就是“Easy to learn, hard to master.”同时，这款游戏的大小仅有 7Mibs，在使用并不高配的手机的高中时代，这款游戏成为高中宿舍集体生活中少有的多人同时和单人兼具的娱乐方式，遂决定以此为蓝本，制作一款类似的战棋游戏。



早期规划的截图（使用 OneNote 所做的草稿）

ai 权值计算的初稿。

控制台代码编写：

由于此前几乎没有接触过可视化编程，初高中长期使用基于控制台的编程方式，遂决定先编写游戏规则和游戏运行模块的代码再进行图形化。

游戏规则和游戏运行是项目最核心的部分，也是项目出现 bug 最多的部分，测试和修复 bug 消耗大量时间。在 4 月 23 日才完成基本的控制台代码，可以使用 txtprintChessBoard()打印的棋盘进行一次正常的游戏了。

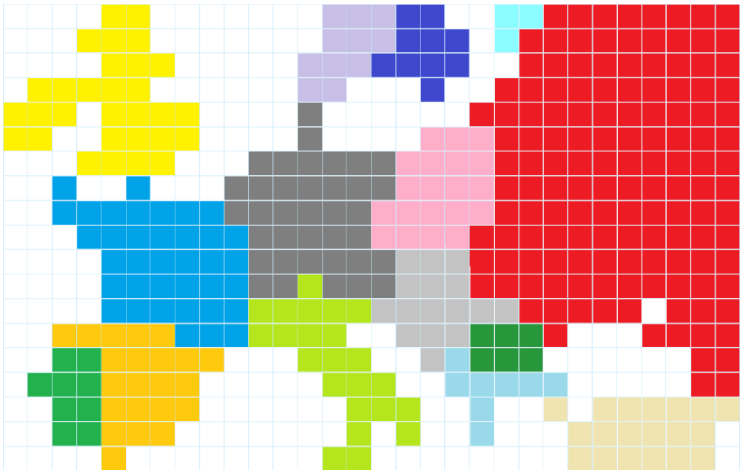
```
void ConJudge()//并查集判定连通块
{
    memset(f,0,sizeof(f));
    for(int i=1;i<=n;i++)//枚举每一个点
    {
        for(int j=1;j<=m;j++)
        {
            if(ma[i][j]>0)//如果有颜色
            {
                if(ma[i-1][j]==ma[i][j])//这点的上面有
                {
                    if(q[ma[i][j]][r[i][j]])//曾经有颜色
                    {
                        if(q[ma[i][j]][r[i][j]]==1)//最后一块了
                        {
                            cash[ma[i-1][j]][r[i-1][j]]+=cash[ma[i][j]][r[i][j]];
                        }
                        q[ma[i][j]][r[i][j]]--;
                    }
                    r[i][j]=r[i-1][j];//换色
                    q[ma[i][j]][r[i][j]]++;//加入上面
                    if(ma[i][j]==ma[i][j-1]&&r[i][j]!=r[i][j-1])//上面和左边都有且原来的编号不一致，涉及重新连通
                    {
```

作为游戏运行核心的 Conjudge()函数

大量的注释因为这里是 bug 的高5发区，该段代码光重构就重构了四次

图形化代码编写：

控制台代码完成后，需要查阅大量资料从头学习 EasyX 的使用。决定先编写按钮类，因为游戏中有大量按钮，使用统一的按钮结构体可以有效减轻代码长度和工作量。此后，先完成“毛坯版”的游戏菜单和游戏的编写，其他部分暂时保留控制台。此时游戏运行又出现了很多 bug。修复完成大约在 4 月 29 日左右。



游戏的背景图，使用 Excel 和画图软件的填色桶画格子的。
图片是抽象化的西班牙内战结束至二战爆发前的欧洲局势。

其他功能的图形化编写：

Word 等软件打开文件夹存读档，但是由于 EasyX 只能打开一个窗口只好作罢，此后创造性的想出了存档插槽这一折中办法，实现分别存档。

```
/*
游戏存档结构：
1.地图大小 n,m
2.人类和人机数量 h_p和a_p
3.20*20矩阵，ma[i][j]地图归属
4.20*20矩阵，r[i][j]，即各点处于的连通块状态
5.20*20矩阵，army[i][j]，即各点目前的军队
6.10*400矩阵，存下cash数组
7.10*400矩阵，存下q数组
8.当前回合
校验功能貌似不需要了
*/
```

最新的存档结构，校验功能是最初进行任意文件存读档时准备的，后来使用插槽，不需要了。

存档结构发生过多次改变，起初将 n 和 m 写反导致存读档错误，后来又因为 cash 和 q 数组的误判导致存读档后金币数据极其诡异，后面认为直接以文本形式存档即可，因为地图很小，文本形式也不会占用过大的空间。

音乐播放器：

EasyX 使用的 MciSendString 的 api，最开始由于网易云音乐下载的 mp3 格式是加密过的无法直接使用 MciSendString("play")打开，后来只能将其转换为.wav 格式，导致文件大小极其夸张。此外，在我的电脑上 MciSendString("repeat")无法正常使用，所以只能写一个函数来控制音乐的随机播放。

编写完之后发现声音音量时空，故而重新转换了一次.wav 压音量，导致文件体积更大。音乐包括玩梗和历史向音乐两种。下面是具体列表：

游戏内名称	音乐名称
脱粪の小曲	一般男性脱糞シリーズ
1444 の小曲	The Voyage（欧陆风云 4 主题曲）
1836 の小曲	Rule the World（维多利亚 3 主题曲）
1936 同盟国の小曲	HOI4 Main Theme Alies（钢铁雄心 4 主题曲兼同盟国主题曲）
1936 苏维埃の小曲	Comintern Theme（钢铁雄心 4 共产国际主题曲）
1936 洗头佬の小曲	Axis Theme（钢铁雄心 4 某种意识形态的主题曲）
凯南开大学の小曲	来源于一个 b 站视频"电棍：凯南开大学校歌"（只有补档）
只因你太美の小曲	只因你太美

蓝色为历史向音乐，均为瑞典游戏公司 Paradox Interactive 制作的诸多大战略游戏的背景音乐。

棕色为玩梗音乐，包括日本现代年轻人的亚文化，中国的一些亚文化和著名网红的成名曲。

在获取音乐播放时长上，最初采用了笨方法：即手动建立一个数组来记录音乐播放时长，后来发现可以调用 mciSendString 的 status 中 position 和 length 两个 api 来完成，同时可以获取当前播放时长。

小修补：

此后进行了一些小修补，比如返回键、音乐播放器的自定义功能等，由于 mciSendString 的 api 无法使用 L"String", _T"String", ch.c_str()中的任何一种形式的字符串，无法加入播放自定义音乐名称和路径的功能。

并加入了几张地图预设。

代码介绍

代码部分分为 tools.cpp，common.cpp，UI.cpp，main.cpp 四个子 cpp 文件，其中主要功能都在 main.cpp 中。

tools.cpp（外部引入的经典代码模板）

这个部分主要是一些常用工具，包括透明抠图，高质量文字等（其中的播放音乐功能本机不明原因无法使用。

```
}

void putimageTMD(int  picture_x, int picture_y, IMAGE* picture, int tmd) //x为载入图片的X坐标，y为Y坐标
{
    if (picture_y < 0)return;
    DWORD* dst = GetImageBuffer();    // GetImageBuffer()函数，用于获取绘图设备的显存指针，EASYX自带
    DWORD* draw = GetImageBuffer();
    DWORD* src = GetImageBuffer(picture); //获取picture的显存指针
    int picture_width = picture->getwidth(); //获取picture的宽度，EASYX自带
    int picture_height = picture->getheight(); //获取picture的高度，EASYX自带
```

```
// 载入PNG图并去透明部分
void _putimagePNG(int  picture_x, int picture_y, IMAGE* picture) //x为载入图片的X坐标，y为Y坐标
{
    DWORD* dst = GetImageBuffer();    // GetImageBuffer()函数，用于获取绘图设备的显存指针，EASYX自带
    DWORD* draw = GetImageBuffer();
    DWORD* src = GetImageBuffer(picture); //获取picture的显存指针
    int picture_width = picture->getwidth(); //获取picture的宽度，EASYX自带
    int picture_height = picture->getheight(); //获取picture的高度，EASYX自带
    int graphWidth = getwidth();       //获取绘图区的宽度，EASYX自带
    int graphHeight = getheight();     //获取绘图区的高度，EASYX自带
    int dstX = 0;    //在显存里像素的角标

    // 实现透明贴图 公式： Cp=α p*FP+(1-α p)*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);   //G
```

```
void setFont(const char* fontName, int fontWidth, int fontHeight) {
    LOGFONT f;
    gettextstyle(&f);
    f.lfHeight = fontHeight;                    // 设置字体高度为 48
    f.lfWidth = fontWidth;
    strcpy(f.lfFaceName, fontName);    // 设置字体为“黑体”(高版本 VC 推荐使用 _tcscpy_s 函数)
    f.lfQuality = ANTIALIASED_QUALITY;    // 设置输出效果为抗锯齿
    setbkmode(TRANSPARENT);
    settextstyle(&f);
}
```

8　部分代码截图

common.cpp（常量定义和窗口加载函数）

这个部分主要是一些常量的定义和窗口加载函数。后来因为头文件连接出错，导致常量定义移动到了主函数下。

```
#include "common.h"



void _InitWin()
{
    initgraph(WIN_W, WIN_H, EX_SHOWCONSOLE);
    setbkcolor(COLOR3);
    cleardevice();
    setbkmode(0);
}
```

代码截图

UI.cpp（按钮结构体定义）

这个部分主要定义了按钮的创建、绘制、触发等函数，节约了编写代码的时间。

```
#include"UI.h"

struct Button* CreateButton(int x, int y, int w, int h, const char* text)
{
    Button* pNewBut = (Button*)malloc(sizeof(Button));
    memset(pNewBut, 0, sizeof(Button));
    pNewBut->x = x;
    pNewBut->y = y;
    pNewBut->w = w;
    pNewBut->h = h;
    strcpy(pNewBut->text, text);
    return pNewBut;
}

bool UpdataButton(ExMessage msg, struct Button* button)
{
    if (msg.x > button->x && msg.x<button->x + button->w && msg.y>button->y && msg.y < button->y + button->h) {
        if (msg.message == WM_LBUTTONDOWN) {
            return true;
        }
        button->flag = true;
    }
    else {
        button->flag = false;
    }
    return false;
}
```

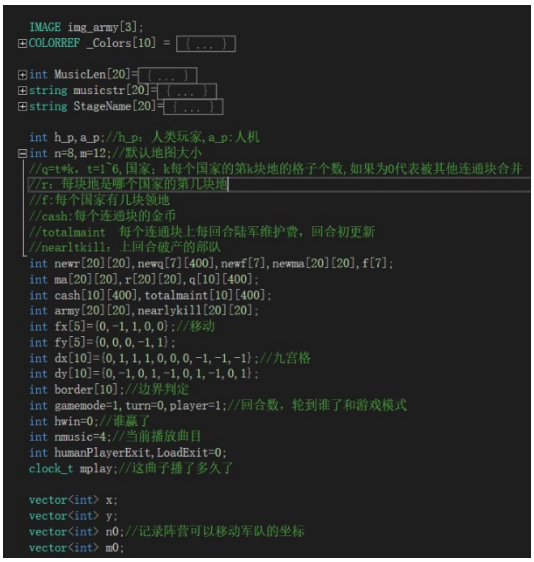
```
void DrawButton(const struct Button* button) {
    setlinecolor(BLACK);
    setlinestyle(0, 1);
    if (button->flag) {
        setfillcolor(COLOR2);
    }
    else {
        setfillcolor(COLOR1);
    }
    settextcolor(LIGHTCYAN);
    settextstyle(button->h * 2 / 3, 0, "黑体");
    fillrectangle(button->x, button->y, button->x + button->w, button->y + button->h);
    outtextxy(button->x + (button->w - textwidth(button->text)) / 2, button->y + button->h / 6, button->text);

}
```

代码截图

main.cpp（主函数）

这个部分由几类共 47 个函数组成，构成了程序的大部分内容，实现了程序的大部分功能。

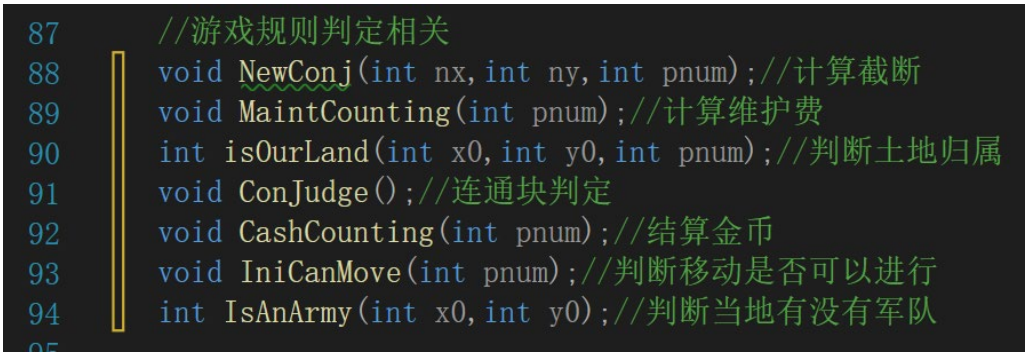
1、常量和全局变量定义。



```
IMAGE img_army[3];
COLORREF _Colors[10] = { ... }

int MusicLen[20]={ ... }
string musicstr[20]={ ... }
string StageName[20]={ ... }

int h_p,a_p;//h_p：人类玩家,a_p:人机
int n=8,m=12;//默认地图大小
//q=t%k，t=1~6,国家；k每个国家的第k块地的格子个数,如果为0代表被其他连通块合并
//r：每块地是哪个国家的第几块地
//f:每个国家有几块领地
//cash:每个连通块的金币
//totalmaint  每个连通块上每回合陆军维护费，回合初更新
//nearltkill：上回合破产的部队
int newr[20][20],newq[7][400],newf[7],newma[20][20],f[7];
int ma[20][20],r[20][20],q[10][400];
int cash[10][400],totalmaint[10][400];
int army[20][20],nearlykill[20][20];
int fx[5]={0,-1,1,0,0};//移动
int fy[5]={0,0,0,-1,1};
int dx[10]={0,1,1,1,0,0,0,-1,-1,-1};//九宫格
int dy[10]={0,-1,0,1,-1,0,1,-1,0,1};
int border[10];//边界判定
int gamemode=1,turn=0,player=1;//回合数，轮到谁了和游戏模式
int hwin=0;//谁赢了
int nmusic=4;//当前播放曲目
int humanPlayerExit,LoadExit=0;
clock_t mplay;//这曲子播了多久了

vector<int> x;
vector<int> y;
vector<int> n0;//记录阵营可以移动军队的坐标
vector<int> m0;
```

本部分定义了大部分全局变量和常量，如地图颜色，地图归属，兵种位置，音乐名称等。

2、游戏规则判定



```
87      //游戏规则判定相关
88      void NewConj(int nx,int ny,int pnum);//计算截断
89      void MaintCounting(int pnum);//计算维护费
90      int isOurLand(int x0,int y0,int pnum);//判断土地归属
91      void ConJudge();//连通块判定
92      void CashCounting(int pnum);//结算金币
93      void IniCanMove(int pnum);//判断移动是否可以进行
94      int IsAnArmy(int x0,int y0);//判断当地有没有军队
```

游戏的基础规则判定部分，其中的 Conjudge()函数是整个游戏运行过程中的基石，它使用朴素的并查集（DSU）算法进行连通块判定，也是 bug 的高发区。绝大部分函数都与之相关。

3、操作相关

```
96      //操作与判定
97      int GenMove(int n1,int m1,int dir);
98      int ConductMove(int n1,int m1,int dir);
99      int IsBorder(int x0,int y0,int pnum);
100     int ManDis(int x0,int y0,int x1,int y1);
101     int CanArmyBeCreated(int x0,int y0,int pnum,int level);
102     int ConductCreation(int x0,int y0,int pnum,int dir,int level);
```

分为两类，造兵和移动。GenMove()和 CanArmyBeCreated()是判断是否可以移动或建造军队，ConductMove()和 ConductCreation()是执行移动和建造操作。Mandis()是计算曼哈顿距离，IsBorder()返回与己方领土颜色不同的敌方领土个数。

4、AI（电脑玩家）相关

```
104     //AI相关
105     void AIMoveStep1(int pnum);
106     int AIMoveStep2(int x0,int y0,int pnum);
107     int AICreationStep2(int x0,int y0,int pnum);
108     int AICreation(int pnum);
```

都分为两个 Step，分步解决电脑玩家的移动和建造行为。其中，AIMoveStep1()和 AICreation()都是判定哪些地方应该被执行操作，然后下放到 AIMoveStep2()和 AICreationStep2()中，区别是，AIMovestep2()需要再调用一次 ConductMove()函数，而 AICreationStep2()直接完成操作。

5、游戏交互

```
110     //游戏运行
111     void HumanPlay(int pnum);
112     void RunGame();
113     void StartRandomizedGame();
114     int printChessBoard(int pnum);
115     void MapGeneration(int tal);
116     void CountF();
117     int WhoWin();
118     void Game_Start();
```

主要包括游戏和玩家的交互，HumanPlay()是玩家回合的界面绘制，RunGame()是游戏的运行全程，StartRandomizedGame()和MapGeneration()是随机地图游戏的两个步骤，前者负责绘制图形化选择界面，后者使用随机游走（Random Walk）算法生成一张保证联通的游戏地图。

printChessBoard()绘制了一张游戏地图，CountF()，Whowin()是不同时期判断游戏结束与否的函数，Game_Start()函数是执行不同游戏选项并开始游戏或返回错误信息的函数。

6、调试和人类玩家全体失败后的函数

```
120     //调试和玩家全体阵亡后用
121     void txtprintChessBoard();
122     void Delay(int time);
```

txtprintChessBoard()函数是命令行阶段的棋盘输出函数，后被printChessBoard()函数取代，成为调试工具。Delay()则是玩家全体阵亡后观看电脑操作（看海模式）的延时函数。

7、游戏的附属功能

```
124     //菜单等非游戏本体相关
125     void Edit();
126     void Game_Load();
127     void ConductGameLoad(char ss);
128     void Map_Save();
129     void ConductMapSave(char ss);
130     void Game_Save();
131     void ConductGameSave(char ss);
132     void ChangeMap();
133     void Map_Load(int fla);
134     void ConductMapLoad(char ss);
135     void ConductSpeLoad(int fla);
136     void Menu();
137     void Help();
```

主要包括地图编辑器、存读档、菜单界面和帮助程序。Edit()函数是编辑器界面，ChangeMap()是在编辑器内调整地图大小和游戏玩家情况。Help()是弹出一个帮助窗口。Game_Load(), Game_Save()等

Load&Save 族函数，是存读档的 UI 界面，而 Conduct Load&Save 族函数则执行存读档。Game.sav 是游戏过程中存储的游戏存档，而 Map.sav 是预设或者编辑器的地图存档。其中，Map_Load(int fla)较为特殊，因为其包括两种读取地图存档：开始游戏时读取编辑器存档和在编辑器中读取此前的编辑器存档和读取并修改预设存档进行游戏两种，所以传入一个参数进行区别。

8、音乐播放器

```
//音乐播放
void MusicPlay(int t);
void MusicPlayMenu();
void RandomPlay();
```

MusicPlay(int t)函数是播放第 t 首乐曲，MusicPlayMenu()函数是音乐播放器的 UI 界面。RandomPlay()控制随机播放历史向音乐。

算法或公式

并查集（Disjoint Set Union, DSU）是一种主要用于解决一些元素分组的问题。它管理一系列不相交的集合，并支持两种操作：

合并（Union）：把两个不相交的集合合并为一个集合。

查询（Find）：查询两个元素是否在同一个集合中。

伪代码（不含路径优化和启发式合并）：

初始化：int fa[MAXN];
　　for i:1→n　fa[i] = i;
查询：return (fa[x] == x)?x: find(fa[x]);
合并：fa[find(i)] = find(j);

本实验中主要用于解决地块归属和分类问题，即源码中的 *r[i][j]*相当于 *fa* 数组，源码中的 *ConJudge()*函数则是 Union 操作的体现。

随机游走（Random Walk）算法：从一个或一系列顶点开始遍历一张图。在任意一个顶点，遍历者将以一定概率游走到这个顶点的某个邻居顶点，每次游走后得出一张图，则这张图是连通图。

本实验中的体现：随机生成一张连通的地图。
伪代码：

```
int dis=rand()%4;
if(New node)    position of Node[cnt++]=walk (from:now position by:dis);
```

项目测试

对游戏及代码进行多次测试。

测试项目：

包括但不限于是否能通过编译，控制台形式代码是否能正常运行，图形化游戏是否可以打开，游戏运行是否有超出规则的地方，游戏之外（编辑器、音乐播放器，存读档体系）的功能是否可以正常使用等。

测试结果：

测试中出现过不少于 100 个通过编译后的 bug，修复 bug 消耗了大量时间。例如：移动单位时一个 m 写成 n 导致的地图大小判定错误，音乐播放器的文件路径名使用“/”而不是“\”，许多数组没有进行初始化或开启新一局时没有清零导致的累计错误等等等等。

目前，绝大部分 bug 已经被修复。

收获

EasyX 的图形化可以定义按钮结构体，节约大量时间。

```
struct Button* CreateButton(int x, int y, int w, int h, const char* text)//创建一个按钮
{
    Button* pNewBut = (Button*)malloc(sizeof(Button));
    memset(pNewBut, 0, sizeof(Button));
    pNewBut->x = x;
    pNewBut->y = y;
    pNewBut->w = w;
    pNewBut->h = h;
    strcpy(pNewBut->text, text);
    return pNewBut;
}

bool UpdataButton(ExMessage msg, struct Button* button)//将按钮链接到信号槽 msg
{
```

```
    if (msg.x > button->x && msg.x<button->x + button->w && msg.y>button->y &&
msg.y < button->y + button->h) {
        if (msg.message == WM_LBUTTONDOWN) {
            return true;
        }
        button->flag = true;
    }
    else {
        button->flag = false;
    }
    return false;
}

void DrawButton(const struct Button* button) {//显示按钮
    setlinecolor(BLACK);
    setlinestyle(0, 1);
    if (button->flag) {
        setfillcolor(COLOR2);
    }
    else {
        setfillcolor(COLOR1);
    }
    settextcolor(LIGHTCYAN);
    settextstyle(button->h * 2 / 3, 0, "黑体");
    fillrectangle(button->x, button->y, button->x + button->w, button->y + button->h);
    outtextxy(button->x + (button->w - textwidth(button->text)) / 2, button->y + button->h /
6, button->text);

}
```

MciSendString 可以获取当前音乐的播放剩余时间

```
        s="当前播放：";
        s+=ch[nmusic];
        settextcolor(BLACK);
        //统计长度
        mciSendString("status bk position", sch, 255,0);//目前位置
        lLength=atoi(sch);
        lLength/=1000;
        mciSendString("status bk length", sch, 255,0);//总长
        len=atoi(sch);
        len/=1000;
        lLength=len-lLength;//总长-目前时长=剩余时长
        s+="    ";
    if(lLength/3600>0)
```

```
Musich=lLength/3600,lLength%=3600,s+=to_string(Musich),s+="：";//通常情况下隐去小时分隔符
        if(lLength/60<10)    s+="0";
        Musicm=lLength/60;
        s+=to_string(Musicm);
        s+="：";
        lLength%=60;
        if (lLength % 60 < 10)    s += "0";
        Musics=lLength%60;
        s+=to_string(Musics);
```

注意：这段代码只有音乐文件是.wav 格式时可以使用，其他文件格式（如.mp3 格式）会出现获取总时长错误的 bug。

EasyX 的绘图实际上并不清除此前图片

使用 outtextxy()函数时可以发现，此前输出的数字并不会被抹除。这是因为 EasyX 的绘图实际上是在已有基础上只覆盖需要被改变的像素点。

解决方法很简单，每次重新打印一张背景图来抹除所有像素点，再重新绘图。缺点是：地图较大时，可以出现明显的卡顿，并且没有很好的解决方法。

善用 string 相关函数

c.str(), to_string, stringstream ss 和 sprintf()等函数大大减少了工作量，string 的高度可操作性和衍生库的库函数使编程更方便，尤其是在接入需要 char 类型或 Tchar()或 Lchar()类型函数时。

大规模使用 Button 和 msg 会导致严重卡顿

当地图大小超过 12×12 后，游戏面临严重卡顿。原因是大规模调用 msg 信号和 UpdataButton()函数导致的。同时，EasyX 原生库仅支持一个窗口的绘制。故而，EasyX 不适合进行游戏的编写，编写游戏还需要专门的游戏引擎来编写，而不是 Qt 或者 EasyX。

目前存在的缺陷

1、【严重】【稳定复现】【体验问题】卡顿问题

【现象】前文已述。

【解决】无法解决，建议不要使用过大的地图。

2、【严重】【偶发】【BUG】金币计算错误

【现象】有时会发生某些地块有数万金币

【解决】已修复多次，虽然目前仍然可能出现此问题，但是可能性已经大大下降，若出现此类问题，建议弃档退出游戏重进。

3、【轻微】【偶发】【BUG】部分地块无法向敌方境内造兵

【现象】某些回合，无法向地方境内建造军队。

【解决】再过一回合或存档，或退出游戏重进即可。

4、【轻微】【稳定复现】【体验问题】游戏体积太大

【现象】整个文件过大。

【解决】无法解决，因为音乐文件只能使用.wav 格式，而减少文件体积只能在降低音质上进行，但是测试后发现效果很差。

5、【严重】【偶发】【BUG】进入游戏出现大量错误

【现象】读档报错、读档出空白、音乐无法播放、金币持续错误计算、无法移动等同时出现多个。

【解决】原因不明，可能和系统内存有关。目前已经极少见，若出现请退出重进。