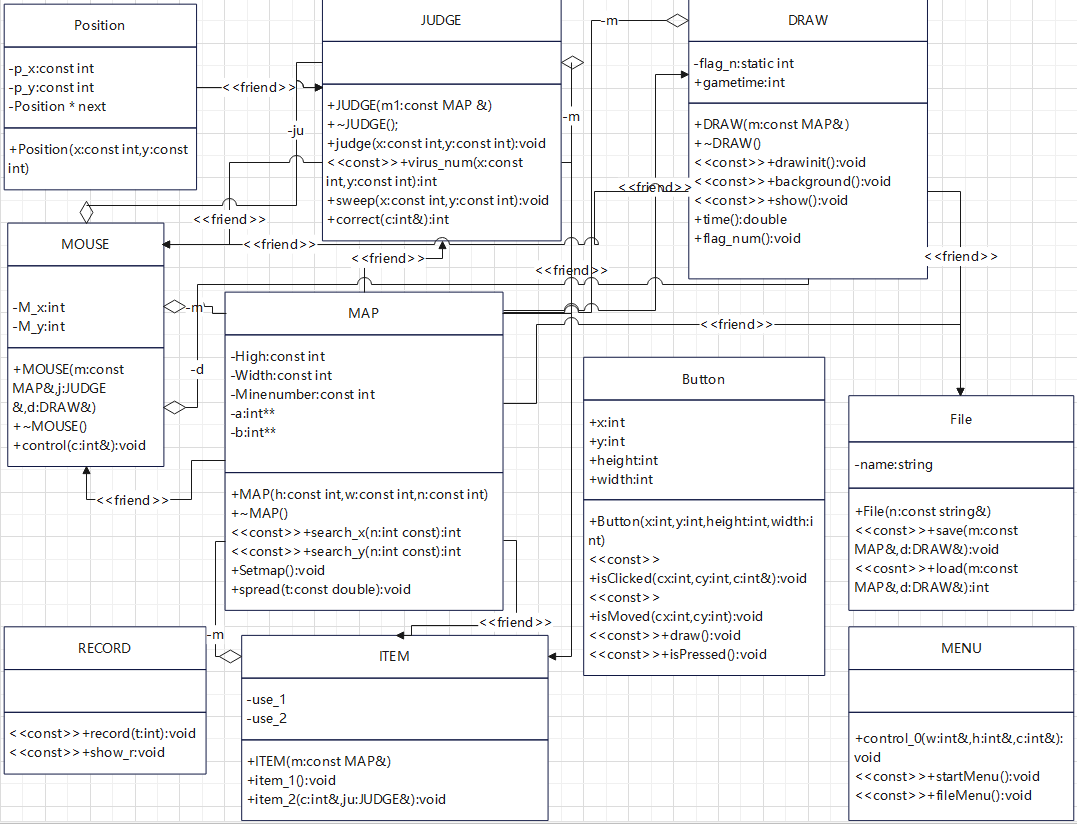
# 《扫除病毒》游戏设计说明书

类图



类的数据成员及成员函数

class MENU

{

public:

void control\_0(int& w, int& h, int& n, int& c);//实现鼠标与菜单交互的函数

void startMenu()const;//显示主菜单界面的函数

void fileMenu()const;//显示游戏模式菜单界面的函数

};

class File

{

private:

string name;//存档文件名称

public:

File(const string& n) { name = n; };//构造函数

void save(const MAP& m, DRAW& d)const;//通过文件保存游戏存档的函数

int load(const MAP& m, DRAW& d)const;//通过文件读取游戏存档的函数

};

class Button

{

public:

Button(int x, int y, int height, int width);//构造函数

int x;//储存按键的横坐标的数据

int y;//储存按键的纵坐标的数据

int height;//储存按键的长度的数据

int width;//储存按键的宽度的数据

void isClicked(int cx, int cy, int& c)const;//按键点击事件的函数

void isMoved(int cx, int cy)const;//鼠标移动事件的函数

void draw()const;//画出按键的函数

void isPressed()const;//键盘交互事件的函数

};

class Position//储存方块的位置信息，用于广度优先搜索中链表的实现

{

private:

const int p\_x, p\_y;//储存结点的横纵坐标的数据

Position\* next;//储存下个结点的地址的指针

public:

Position(const int x = 1, const int y = 1) :p\_x(x), p\_y(y)//构造函数

{

next = 0;

};

friend class JUDGE;//友元类

};

class MAP

{

private:

const int High, Width, Minenumber;//地图基本信息，长度、宽度和病毒数量

int\*\* a = 0;//二维动态数组,储存地图信息，是否有病毒，以及方块周围的病毒数量

int\*\* b = 0;//二维动态数组,储存地图信息，方块是否被翻开，以及是否被标记

public:

MAP(const int h = 9, const int w = 9, const int n = 50);//构造函数

~MAP();//析构函数

int search\_x(const int n)const;//搜索第n个方块所在的横坐标的函数

int search\_y(const int n)const;//搜索第n个方块所在的纵坐标的函数

void Setmap();//通过洗牌算法实现随机布雷，完成地图的初始化的函数

void spread(const double t);//病毒扩散的函数 ，使用随机数，在病毒周围随机产生新的病毒

friend class JUDGE;//友元类

friend class MOUSE;//友元类

friend class DRAW;//友元类

friend class File;//友元类

friend class ITEM;//友元类

};

class JUDGE

{

private:

const MAP m;//游戏地图对象

public:

JUDGE(const MAP& m1);

~JUDGE();

void judge(const int x, const int y);//（鼠标左键点击后）判断该位置是否有病毒的函数

int virus\_num(const int x, const int y)const;//得到该位置九宫格内的病毒数的函数

void sweep(const int x, const int y);//清扫方块的函数（若某个位置无病毒，清扫并判断其周围的区域是否有病毒，若无继续清扫，使用广度优先搜索算法）

int correct(int& c);//根据病毒数目的变化，保证显示数字的正确，以及判定游戏的胜负的函数

friend class MOUSE;

friend class ITEM;

};

class DRAW

{

private:

const MAP m;//游戏地图对象

static int flag\_n;//储存游戏中剩余的标记数的数据

public:

DRAW(const MAP& m);//构造函数

~DRAW();//析构函数

double gametime = 0;//储存时间的数据，时间精确到毫秒

void drawinit()const;//图片载入的函数

void background()const;//地图背景的显示函数

void show()const;//游戏画面显示和实时更新的函数

double time();//游戏时间记录和显示的函数

void flag\_num();//显示剩余可用的标记数的函数

friend class MOUSE;

friend class File;

};

class MOUSE

{

private:

const MAP m;//游戏地图对象

JUDGE ju;//游戏判断对象

const DRAW d;//游戏绘图对象

int M\_x, M\_y;//储存鼠标在地图中的位置

public:

MOUSE(const MAP& m, JUDGE& j, DRAW& d);//构造函数

~MOUSE();//析构函数

void control(int& c);//鼠标控制的函数，包括鼠标移动，左键，右键点击

};

class RECORD

{

public:

void record(int t = 180)const;//储存游戏记录的函数

void show\_r()const;//游戏记录的显示函数

};

class ITEM

{

private:

const MAP m;// 游戏地图对象

int use\_1;//道具1是否在使用的数据

int use\_2;//道具2是否在使用的数据

public:

ITEM(const MAP& m);//构造函数

void item\_1();//包含道具1：口罩的函数

void item\_2(int& c, JUDGE& ju);//包含道具2：消毒剂的函数

};

主要技术难点及实现方案：

难点1：游戏多界面的处理，完整的游戏结构。与平时程序的线性结构不同，在游戏中需要实现在主界面进行不同的选择，在游戏中能够返回到主界面，进行再次选择，达成完整的闭环。

实现方案：采用了循环语句和选择语句相结合的方式，在主函数中定义与界面选择有关的变量，在界面函数中通过传地址的方式改变变量，再通过变量的值作为判断依据，来在不同界面实现选择和循环，从而达成游戏结构的闭环。

难点2：游戏中无法避免的不同模块之间的交互，例如在游戏中鼠标点击会导致地图数据改变，地图数据又与显示游戏画面关系密切，存档的保存和读取要与地图数据交互，道具功能的实现又需要与地图和判断模块相互作用，各种模块关系密切，数据传递较多。

实现方案：在部分类中声明了友元类，通过使用友元类，在一个类中可以调用另一个类的函数，也可以利用另一个类的数据进行操作和判断，必要时会改变另一个类的数据，从而更便捷地在不同模块中进行交互，为安全性考虑，同时将大多数对象引用的传递声明为const，保证了不同模块之间不会随意互相改变数据，防止导致数据的混乱。

难点3：玩家的操作，在图形界面中实现人机交互

实现方案：游戏利用了easyX图形库实现图形界面，同时也利用库中的函数getmessage（），对玩家的鼠标数据进行读取，读取到鼠标的位置、是否点击、点击左键或右键等数据，通过判断语句完成玩家与游戏的交互，还利用了\_getch()函数，读取玩家通过键盘输入的数据，以此在游戏中加入了道具的使用。除此之外，我还利用了图形库中的InputBox（）函数，实现了玩家可以在输入框中输入符合要求的数字，从而选择游戏中是否保存，读取存档，以及使玩家能够自定义游戏地图的大小。

难点4：扫雷玩法的核心算法1：如何在自定义大小的地图上实现快速且随机的布雷

实现方案：我先用动态的二维数组实现了地图大小的自定义。对于布雷的算法：最初的想法是直接用随机数在全部的方块中随机产生一个个雷，当产生的雷重复时再次刷新产生，直到不与已有的雷重复，这种方式的弊端是当雷的数量较多时极易产生重复，需要不断重复刷新，直到产生不重复的雷，速度较慢，无法实现快速布雷。

在学习和不断尝试之后，我选择了用洗牌算法来实现随机布雷：对于在方块总数为n大小的地图内部x个雷，先设定前x个方块全部为雷，然后将第1个方块与与包括它在内的后面全部方块（n个）中的随机一个进行交换，再将第2个方块与包括它在内的后面全部方块（n-1个）中的随机一个进行交换，以此类推，交换x次以后，所有方块中有雷的概率相等，均为x/n，从而实现了快速随机布雷，该算法时间复杂度为（n），随地图总大小和地雷总数的变化，时间改变较小，因此也便于玩家自定义地图的大小和游戏难度。

该算法的实现代码如下：

void MAP::Setmap()

{

int i, j, temp = 0;

srand(time(0));

for (i = 1; i <= High; i++)//初始化全部为0

{

for (j = 1; j <= Width; j++)

{

a[i][j] = 0;

b[i][j] = 0;

}

}

i = 1, j = 1;

//洗牌算法实现快速随机布雷

for (int k = 1; k <= Minenumber; k++)

{

a[i][j] = 9;

j++;

if (j == Width + 1)

{

j = 1;

i++;

}

}

for (i = 1; i <= High; i++)

{

for (j = 1; j <= Width; j++)

{

cout << a[i][j] << ' ';

}

cout << endl;

}

i = 1, j = 1;

for (int k = 1; k <= Minenumber; k++)//先在前x个位置布雷，然后对于前x个位置，每个位置和包括它在内的之后每个位置进行交换，经过x次交换后，每个位置有雷的概率都是x/总数，实现等概率且不重复地布雷

{

int m = rand() % (High \* Width - k) + k;

temp = a[i][j];

a[i][j] = a[search\_y(m)][search\_x(m)];

a[search\_y(m)][search\_x(m)] = temp;

j++;

if (j == Width)

{

j = 1;

i++;

}

}

}

难点5：扫雷玩法的核心算法2：实现点击后自动清空一片无雷的区域。在实现该效果之前，我的游戏中对于空方块，即点开后上面没有数字的方块只能一个个点击，不能像传统Windows上的扫雷游戏一样点击后直接清空周围一片无雷的区域。

实现方案：采用了广度优先搜索算法，利用链表来实现。用一个类来储存位置信息（横纵坐标）和下一个结点的地址，设置一头一尾两个指针，最初两指针指向同一个地址，而在之后的循环中，头指针会不断移动到下一个结点，并对下一个结点的方块周围的方块依次进行判断，是否是需要翻开的方块，若是，则将尾指针延伸，储存其位置信息，尾指针不断延伸至不符合要求无法翻开的方块，而头指针在循环中不断后移，当两个指针指向的地址再次相同时，循环结束。利用这种方式，巧妙地实现了对于一个空的方块，清除它周围一片没有病毒的区域。

该算法的实现代码如下

void JUDGE::sweep(const int x, const int y)//动态指针

{

Position p(x, y);

Position\* head = &p, \* tail = &p;//前后两个指针，用于实现广度优先搜索

Position\* plist = &p;//指向p，用于最后内存的释放

do

{

if (head->next != 0)//首指针后移

{

head = head->next;

}

int x\_, y\_;

for (int i = 0; i < 9; i++)//遍历一个矩形区域

{

x\_ = head->p\_x + i % 3 - 1;

y\_ = head->p\_y + i / 3 - 1;

if (x\_ >= 1 && y\_ >= 1 && x\_ <= m.Width && y\_ <= m.High)//&& (x\_ != head->p\_x || y\_ != head->p\_y)

{

if (m.a[y\_][x\_] == 9)

{

//保持原样

}

else if (virus\_num(x\_, y\_) > 0 && m.b[y\_][x\_] != 2 && m.b[y\_][x\_] != 1)

{

m.a[y\_][x\_] = virus\_num(x\_, y\_);

m.b[y\_][x\_] = 1;

}

else if (virus\_num(x\_, y\_) == 0 && m.b[y\_][x\_] != 2 && m.b[y\_][x\_] != 1)//遇到满足条件的方块，尾指针延伸

{

m.a[y\_][x\_] = 10;

m.b[y\_][x\_] = 1;

//Position p\_(x\_, y\_);//有作用域的限制

tail->next = new Position(x\_, y\_);//注意要delete避免内存泄露!!!

tail = tail->next;

}

}

}

} while (head != tail);

head = plist->next; // 此时head指向plist的下一个结点，第一个结点不能删除（不是new得到的）。

while (head != NULL)

{

tail = head->next;

delete head;

head = tail;

}

plist = NULL; // 只是释放了其内存空间，指向链表的指针值并没有被置为空

}

难点6：如何恰当使用文件I/O流，完成对游戏存档的保存和读取。

实现方案：由于存档所需要存储的主要信息只有地图的数据和时间数据，而在地图类中使用了动态二维数组，无法将整个对象输出到文件中保存，我选择了直接用循环将动态数组的数据输出到文件中，同时记录时间的函数返回保存时的时间一并输出到文件中，读取存档时直接按顺序再次将文件中内容读入即可。