# Clap程序设计

## 需要了解的东西

### 命名习惯（是粗糙的，可以改进^o^）

class 类名，一般带有后缀\_o（因为在class中常常需要定义同名变量名，这样的变量名就没有后缀）

参数的命名习惯，无论是输入值，瞬时值还是状态值（之后详细说明），在每一回合的值都不一样，参考”i+.h”中的宏的定义可知：用单个单词声明本回合的变量，加一个下划线声明上一回合的变量，加\_htr表示全部回合的向量组

### 参数

输入值：m.move(行动类型)与m.at（攻击目标）

瞬时值：所有瞬时值都被封装起来，不需要具体了解

状态值：HP(体力值),bullet(子弹数),sword(剑数),pause(在冷却中的剑数)

turn：回合数

people：初始人数

alivepeople：存活人数

choice：选择数

### 如何访问这些参数

turn,people,alivepeople,choice只需要名字（其实是宏不过这不重要）就可以在include”i+.h”的任何文件中访问（为什么要这个文件可以参考结构图或者后面对各个文件的说明）

输入值

//m[from].at[order] 本回合，第from个玩家的第order个行动的at值

//m\_[from].at[order] 上一回合，第from个玩家的第order个行动的at值

//m[from].move[order] 本回合，第from个玩家的第order个行动的move值

//m\_[from].move[order] 上一回合，第from个玩家的第order个行动的move值

//m\_htr[turn][from].at[order] 第turn回合，第from个玩家的第order个行动的at值

//m\_htr[turn][from].move[order] 第turn回合，第from个玩家的第order个行动的move值

瞬时值只可以在act\_o内部访问

状态值

//HP.状态值[from] 第from个玩家在本回合的HP

//bullet.状态值[from] 第from个玩家在本回合的bullet

//sword.状态值[from] 第from个玩家在本回合的sword

//pause.状态值[from] 第from个玩家在本回合的pause

//HP\_.状态值[from] 第from个玩家在上一回合的HP

//bullet\_.状态值[from] 第from个玩家在上一回合的bullet

//sword\_.状态值[from] 第from个玩家在上一回合的sword

//pause\_.状态值[from] 第from个玩家在上一回合的pause

//status\_htr[turn].状态值[from] 第from个玩家在第turn回合的状态值

//e.g. status\_htr[turn].HP[from] 第from个玩家在第turn回合的HP

### 从main函数看整个程序的框架

int main()

{

srand((unsigned)time(NULL));

//全游戏循环//除非玩家输入指令否则不会退出

while (1)

{

//对各个参数和变量初始化

//回合内小循环//每个回合走一遍

while (看看是否要结束游戏)

{

//对变量进行刷新

//回合开始阶段↓

//显示需要显示的东西

//行动阶段↓

//先引导player做出行动，再让电脑做出行动

//打印行动

//结算阶段↓

//操作玩家们的状态值

//人头结算阶段↓

}

//游戏结束了

Finalization::finalscreen();

move\_history\_o::finalization();

status\_history\_o::finalization();

ACTHistory::finalization();

turn = 0;

if (!Finalization::whether\_restart())

break;

}

}

### 文件的读取格式说明

目前需要读取的文件只有setting这一个

数据行以#开头，所有数据都是整数，读取到行末或者//为止，没有以#开头的为无效行，注释行以//开头，并且所有回车都会被跳过

## 我们可以做的事情

### 毫不留情地指出代码写的不好的地方

包括但不限于：

1.封装不好的地方

2.可以优化的结构

3.难听的命名

…

### 毫不留情地玩弄程序直到出现bug

由于个人能力有限且测试时间短暂所致，程序不可避免出现一些暂时未发现的bug（尤其是玩家不想按规则做事情的时候），但这是可以不断改进的。

出现bug之后，先不要急着重开，将此时的move\_history文件和history文件保存以待之后调试时重现当时的场景

### 设计闯关模式&新手教程

#### 对关卡的任何有意思的构思和想法

#### 在游戏开始、进行中、结束阶段时的信息提示

可以用txt写好交给我封装进相应的类中（也可以自己做，分别是三个显示文件中）

#### 各个关卡的初始参数

写在setting中，并在Initialization::readin()中写明白如何读取它

#### 各个关卡的获胜判定

每个关卡可能有不同的获胜条件，这个需要改变finalization.h中的rule函数和finalscreen函数（其中应该做好各个关卡的信息）

#### 闯关记录的存储文件

该文件应该要做到既可以读也可以写

### 经典模式

经过历史考验的经典模式有以下这几个

双人：

1.我四发你三条

2.六剑四命/六剑五命，禁挑衅

多人：

1.和平

2.三命刀战

3.六命刀战

经典模式可以开发成回合制，与人机对战选择三局两胜等等都可以增加游戏趣味

### 其他模式（搞笑）

简单记录一下高中三年开发出奇奇怪怪的版本，或许有朝一日可以实现？

GCA模式，等级制

嗜血模式

30条命？？

我觉得应该还有别的什么，但是有的可能记不清了

### 英雄模式

最为复杂，也给其他方面（例如人机设计，界面设计等）提出了全方位的要求

并且随着英雄的引入可能程序的固有体系还有要更改的地方，这就给程序设计带来了更高的挑战

建议短期内先不入手

### 图形化界面

#### 游戏进入界面

#### 游戏运行界面

应该包含教程，闯关，经典模式，自定义游戏，挑战模式几部分

需要在屏幕上显示的变量也不过只有状态和行动，所有瞬时值都不需要显示，这也是它们被毫不留情地装起来的原因之一

### 人机设计

#### 群战人机设计

有一些小想法，可以设计一些人格化人机，比如进攻型或者苟活型，挑衅欠打型或者过来不要命型，在困难模式下可以设计人机只对玩家进行攻击或对玩家攻击的概率较高

也可以设计全能的收益计算专家人机，不过困难程度有点大，对算法水平要求高

或者机械学习？？

#### 人机设计要改动的函数

Choose\_your\_action::CPU\_in\_action

### 配乐和界面美化

#### 配乐

最高的期望是每一类型的玩法都有应景的配乐（类似pvz），并且能够有音效（要求是不是有点高ww）

#### 界面美化

各个参数如果都是数字未免乏味，将HP等可视化是一个不错的选择

期望行动能有相应的特效

期望比如受伤死亡之类的能够有特效（类似枪战游戏中被打中的效果）

## 详细解析每一个头文件

### all tools files(工具性文件)

#### i.h

万能头文件

#### i+.h以及对各个变量的描述

包含i.h

包含一系列宏的定义

包含people.h(见后)

包含choice.h(见后)

#### i++.h

包含i+.h

包含基本变量的文件

#### tool.h

##### 数学函数

//最小值函数

int min(int a, int b);

//最大值函数

int max(int a, int b);

//生成1~10的随机数，并将其赋值给n

void arbitrarynum(int& n);

//生成1~range的随机数，并将其赋值给n

void arbitraryrange(int& n, int range);

##### 文件读写函数

// 如果c是回车符，则返回true；否则，返回false。

bool gb\_isCharReturn(char c);

// 从文件file当中读取一行字符并保存到字符串s当中。

void gb\_getLine(string& s, ifstream& file);

//如果s是数据行，则保留s中的数据并返回true，否则返回false

bool gb\_deleteLine(string& s);

//从文件file中读入一个数据行并保存在s中，如果读入成功返回true，读到文件末尾也没有数据行返回false

bool gb\_getDataLine(string& s, ifstream& file);

//从文件file中读入一个数据行并且保存在一个int里，如果成功则返回true，否则返回false

bool gb\_getData(int& n, ifstream& file);

//之前调试时用于测试

void testreadin();

### basic veriaty classes(基本变量文件)

#### choice.h

//可选择的行动的总共数量

//此时初始化为26，并且全局保持不变

//#define choice choice\_o::choice\_

class choice\_o {

public:

static int choice\_;

};

#### people.h

//存储游戏人数

class people\_o {

private:

//人数

//指初始的人数，该值在一局游戏中保持不变

static int people\_num;

//现存活人数

static int alivepeople\_num;

public:

//输入n，将n赋值给people\_num和alivepeople\_num

static void people\_o\_in(int n);

//返回people\_num

static int people\_out();

//返回alivepeople\_num

static int alivepeople\_out();

//刷新alivepoeple\_num（更新存活人数）

static void the\_refresh\_of\_people();

};

#### status.h

//为了调用和改变方便，这里的变量（for now）不定义成private

//存储4个状态值

class status\_o {

public:

vector<int> status\_HP;//生命值

vector<int> status\_bullet;//子弹数

vector<int> status\_sword;//剑数

vector<int> status\_pause;//在CD中的剑数

public:

//默认构造函数，只做了扩充向量长度的工作

status\_o();

//认真的赋值函数，将参数表里的各个值赋给各个变量

status\_o(vector<int> formhp, vector<int> formbullet, vector<int> formsword, vector<int> formpause);

};

//各个回合的状态值

//status\_history\_o::status\_history[turn].status\_HP[from] means 第turn回合第from个玩家的HP

class status\_history\_o {

public:

static vector<status\_o> status\_history;

//将HP0之类的值赋值到status\_history\_o[0]

static void value\_in();

//向量长度的扩充工作

static void initialization();

//将上回合的状态值顺延到下一回合

static void refreshing();

//向量长度清零

static void finalization();

};

#### move.h

//move\_o是玩家和电脑的直接输入

//move\_o::move[order],玩家行动序号为order的行动种类，种类与具体哪一个的对应见clap.txt

//move\_o::at[order],玩家行动序号为order的目标，种类与具体哪一个的对应见clap.txt

//攻击类型需要指定攻击目标，防御和反弹类型不需要指定目标，补给类型默认以自己为目标，其他类型中，过来不需要目标，挑衅需要指定目标

//攻击类型和补给类型的目标会受到过来的影响而改变

class move\_o {

public:

vector<int> move;

vector<int> at;

move\_o();//也是只做了向量长度的扩充工作

};

//各个回合的行动记录

//move\_history\_o::move\_history[turn][from].move/at[order],玩家from在第turn回合的行动

class move\_history\_o {

public:

static vector<vector<move\_o>> move\_history;

static void initialization();//也是只做了向量长度的扩充工作

static void refreshing();//也是只做了向量长度的扩充工作

static void finalization();//向量长度清零

static void cls\_of\_moves\_of\_the\_dead();//将已死亡玩家有可能的行动清零

};

#### turn.h

//回合数

class turn\_o {

public:

static int turn\_;

### functioning classes(功能类函数)

#### 核心变量的传递体系

共有三层传递

输入值：m.move(行动类型)与m.at（攻击目标）

输入值通过act()函数传递到瞬时值

瞬时值：shoot(枪类攻击类型),stab(刀剑类攻击类型),block(防御类型),ca(反弹类型),provoke(是否挑衅),comeon(是否过来)

doblock(是否进行了某种防御),doca(是否进行了某种反弹)

attack.level(攻击力等级),attack.damage(攻击伤害)

add.catagory(补给类型),add.number(一次补给数量)

ds.damagesource(伤害来源),ds.killer(人头收益者),ds.benefit(人头收益数量)

瞬时值通过performattack()函数引起状态值的改变

状态值：HP(体力值),bullet(子弹数),sword(剑数),pause(在冷却中的剑数)

#### act.h

class ACT\_o {

public:

void act();//将move中的变量传递到act的各个顺时值中，

void performattack();//进行结算，通过顺时值对状态值进行改变，并且进行资源的消耗的结算

void head\_gain();//人头收益结算

//（以上不完整，只是三个比较重要的）

class ACTHistory {

public:

static vector<ACT\_o> ACT\_history;

static void initialization();//也就是对向量长度进行初始化

static void refreshing();//也就是对向量长度进行初始化

static void finalization();//清空向量长度

};

#### check.h

此文件里的函数均为内部调用，不需要了解

#### Choose\_your\_action

class Choose\_your\_action {

public:

//一系列指导玩家行动的程序

static void player\_in\_action();

//一系列指导电脑如何行动的程序

static void CPU\_in\_action();

//让电脑和玩家重复存在move\_history文件中的行动

static void repeteaction(const char\* move\_history);

//在文件中打印行动的历史记录（还只是数字）

static void print\_move\_history(const char\*);

};

### screening(显示文件)

#### Initialization

//初始化程序

class Initialization{

public:

static void introduction();//介绍游戏规则

static void readin();//读入

static int defineDifficulty();//初始化难度,将difficulty\_o转化成难度百分比等级，判定人机采用高级算法的概率

//初始各个状态值的返回函数

static vector<int> HP0\_\_();

static vector<int> bullet0\_\_();

static vector<int> sword0\_\_();

static vector<int> pause0\_\_();

private:

static void readin\_file(const char\* settings);//从文件中读入初始设定

static void readin\_keyboard();//从键盘上输入初始设定

private:

static int whether\_introduce;//是否要介绍游戏规则

static vector<int> HP0, bullet0, sword0, pause0;//返回初始各个状态值

static int difficulty\_o;//难度等级

static int bigturn;//暂时没有计

};

#### screening

//游戏中的显示程序

class Screen {

public:

static void print\_actionguide();//打印游戏操作选择并提示“请输入”

static void print\_turn();//打印回合数

static void print\_act();//打印玩家和电脑的行动

static void print\_status();//打印玩家和电脑的状态值

};

#### finalization

//判别游戏是否结束的指令

class RULE {

public:

static bool rule\_();//若可以继续则返回true，结束游戏则返回false

static bool unexpected\_error();//若有资源bug的情况，则返回true

};

//结束显示程序

class Finalization {

public:

static void finalscreen();//游戏结束后显示的东西

static bool whether\_restart();//问玩家要不要接着玩

};

## 对存储文件的说明

setting.txt用于存储初始设定，并且需要读入

history.txt和move\_history.txt用于存储历史记录，目前暂时不用读入