clap程序设计

0. 游戏规则

子弹bullet1
给枪shoot2:消耗一颗子弹
双枪doubleshoot3:消耗两颗子弹
挡block4:格挡单枪和刺
双挡doubleblock5:格挡双枪和刺
反弹rebound6 : 反弹单枪的伤害，格挡刺，需一发子弹
拔剑drawasword7 : 剑可无限次使用，但消耗完可用的剑后不能再连续使用
亮剑brandish8 : 反弹对方刺的攻击，需1把剑
刺stab9 : 需一把剑
砍chop10 : 需两把剑
重挡heavy block11 : 格挡砍
激光枪laser gun12 : 消耗2子弹，造成2伤害
激光炮laser cannon13 : 消耗（对方HP - 1）子弹，一击致命
火箭炮RPG14 : 消耗3子弹，造成2伤害
双火箭炮double RPG15 : 消耗6子弹，造成3伤害
核弹nuclear bomb16 : 消耗5子弹，对目标造成3伤害，对其他人造成1伤害
光剑lightsaber - stab17:需3剑，造成2伤害的普通刺
光刀lightsaber - chop18:需5剑，造成2伤害的普通砍
双砍double chop19 : 需4剑，造成1伤害 鬼斩ghost - chop20 : 需6剑
箭尾斩nock - chop21:需10剑，造成2伤害的鬼斩
缴械disarm22 : 防御除了鬼斩外的剑的伤害并使攻击者下回合无法用剑，需三把剑
镜面反射specular reflection23 : 反弹光类伤害，需三把剑
巨盾giant shield24 : 挡住双砍，鬼斩与箭尾斩
过来comeon25 : 使其他角色的攻击和补给的目标改为你
挑衅provoke26 : 若对方下一回合未攻击你，则其流失一点体力
攻击力比较：激光炮<刺 = 光剑<砍 = 光刀 = 双砍<单枪 = 双枪 = 激光枪<鬼斩 = 箭尾斩<RPG = 双RPG<核弹
核弹只能单独使用；其他攻击可对不同玩家同时进行
多人时不能对一个人使用“双”的攻击

1. 需要了解的东西

1.1 命名规范：（一个简略的规范）

1. 类的首字母要大写，单词与单词之间要大写
2. 变量的首字母小写，单词与单词之间用连字符隔开
3. 函数的首字母小写，其他单词的首字母大写（驼峰命名法）
4. 如果是频繁使用的宏，则为了简便和一般的变量命名法保持相同，如果说像（ACT，STATUS）这样的大宏（管理一大类变量的），则所有字母都大写。这时候要注意到宏的名字不可以和变量名重复，若出现这中情况，则在变量名前面加上“类名_”
5. 头文件和源文件名字全部小写
6. 对于存储的瞬时值、状态值、输入值，这些通常情况都是单个小写单词代表此回合的值，加一个下划线代表下一回合的值，“_htr”则是表示存储单词，如果有同名的函数，则加两个下划线。

1.2 Qt命名规范

1. 几个常用控件的简写：
   pushbutton：btn
   Label：lb
   Spin box：spb
   大体上取每个音节开头的几个不重复的辅音字母即可，其他变量的简写可以在网上搜到，应该基本上是一致的
2. 控件的命名规范：
   控件的功能_控件的简写
   如：help_btn、HP1_lb
   如果有多个此功能的控件，则在其后注明控件所在页码的名称
   如：help_btn_play_page
   对layout控件的命名：
   vertical Layout：vlt
   horizontal Layout：hlt
   layout控件简写_所在页码名称_布局层数(因为会有嵌套布局)_序号
   e.g vlt_play_page_1_2:在player_page的第一层第二个垂直布局
3. 槽函数的命名规范：
   与go_to_slot给出的命名规范一致

1.3 文件读取格式

目前需要读取的文件只有setting类（游戏初始参数）和message类文件（就是信息提示）

所有数据文件都遵循：数据行以#开头，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，每行参数有特定的意义，与其读入规则有关，任意删改文件中的参数数量或调换参数位置将会造成读入bug

1.2.1 setting类文件

以setting.txt为代表，每一个setting类文件存储一类游戏的初始参数，这些参数的存储方式与

```
Initialization::readin_files(const char* filein);
```

中规定的读取方法相一致

1.2.2 message类文件

message类数据行以#开头，所有数据都是字符串，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。
其读取方法为：不断读取数据行，直到遇见开始标志，然后向下继续读取有效行并对其进行操作，直到遇到结束标志
这些参数的读取方法与

```
Initialization::message(const char* filein, const char* startmsg, const char* endmsg);
Finalization::message(const char* filein, const char* startmsg, const char* endmsg);
```

中规定的读取方法相一致

2.参数以及如何访问这些参数

下面说明的这些参数，有些因为过于常用而在"i+.h"中被定义成宏，这些宏在"i+.h"中可以找到其所在类

- 初始值

在游戏正式开始前必须赋值的东西

people 游戏人数

choice 总选择数，默认是26

activechoice 各玩家可做出的有效选择
activechoice[from][number] 第from个玩家的第number个选择是否可用。

Initialization::difficulty 难度十分制等级

Initialization::add_bullet_num 玩家一回合补子弹的子弹数
Initialization::add_bullet_num[from] 第from个玩家一回合补子弹的子弹数
Initialization::add_sword_num 同理

- 游戏总变量

turn 回合数

alivepeople 存活人数

- 输入值

以m存放输入值，m_存放上回合的输入值，m_htr存放历史输入值，

m是一个以Move为原型的对象，其中由两个描述行动的向量组move、at构成

move 指行动类型，at 指行动目标，通常的攻击和挑衅行动都会要求玩家指定行动目标，补给默认行动目标为自身，但会受过来的影响而改变，挑衅的目标不受过来影响，防御和反弹默认行动目标为0

访问方法如下：

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内部访问，是它的私有变量，是连接输入值和状态值的桥梁

- 状态值

HP 体力值 sword 剑数 bullet 子弹数 pause 在cd中的剑数

STATUS 一个包含以上状态值的对象的宏名，但似乎没有用过

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

以上，除了瞬时值之外的变量在全局都可以直接访问，其地位相当于全局变量

3. 游戏框架

3.1 游戏整体运行体系

```
int main()
{
    while(1)
    {
        //玩家选择模式
        //进入模式
        while(1)
        {
            //用特定方法赋初始值
            //将各个向量组长度初始化
            //进入游戏内小循环
            while(游戏并没有结束)
            {
                //回合开始阶段↓
                //延长各个向量组
                        //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↓
                //人头结算
                //刷新存活人数

                //结束阶段
                //保存这回合的行动信息
            }
            //游戏结束了
            //清除向量组
            是否在玩一把?
        }
        是否退出?
    }
}
```

4. 我们可以做的事情

4.1 关卡设计

- mode

1.新手教程

2.闯关模式

3.经典关卡

- method

1. 在游戏开始、结束阶段时的信息提示

在tutorial/level文件夹中创建txt，仿照tutorial-level-1的格式书写

2. 各个关卡的初始参数

在tutorial/level文件夹中创建txt，每行的参数意义可参考tutorial-level-1，以及readinFile函数

3. 修改maxlevel：关卡总数

完成上面这些就可以成功设计关卡并在提示下运行了，简单来说，设计关卡几乎不需要操作源代码

4. 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RULE::rule函数和finalscreen函数（其中应该做好各个关卡的信息），目前来说暂时没有这个需求，也没有提供能这么做的安全对外接口

5. 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

4.2 图形化界面设计