2015

复旦大学微电子

[需求与设计报告]

制作人员：朱晨畅 陈德政 罗吕根 韩昌佑

目录

1.系统概述

1.1 软件系统的用途

Fate/undefined 是一款平面战棋类游戏。这款游戏将型月世界观中的第五次圣杯战争作为背景，参照 DND 3R 设定集对比 Fate 系列的设定进行了一定的二次创作。双方各操纵 6 枚棋子在棋盘上厮杀，以对方全灭作为胜利条件。

此游戏在开发的全过程中，除了对 Fate 系列中人物原本设定的参考外，游戏人物数据和场景设计、战斗形式和动画、系统构建方式等完全为原创。另外，此游戏完全非商业化，游戏中所用的所有图片和音效素材均来自网络或提取自 Fate 系列原版动画。此游戏的开发仅作为复旦大学 2015 年春季 2012 级微电子学专业《软件设计与开发》专业课程的课程设计以及同学间关于软件开发技术的交流与学习而存在。

现 Fate/undefined 最新版已通过 α 测试，其内部版本号为 2.1.7。初版完成度约为 90%，可满足平时与朋友休闲娱乐的基本要求，后续可能会继续对其更新和维护。

开发平台为 Qt 5.4.1/Visual Studio 2013。

1.2 系统开发的过程

2015.4.6

0.若编译失败，一个较大的原因是 x86x64 的原因。

解决方法：

打开项目属性页，点开右上角的配置管理器，设置活动平台和目标平台为 x64,。保存退出后点击运行，应该会提示 qt 版本未设置问题，此时再去 qt project 菜单设置 qt 版本即可。

1.暂时是分成了 GameWindow，MapUnit，PanelMainMenu，PanelOperate，PanelHeadUp，PlayerController，Element，ElementPool，Constant 这几个类。创建项目时 GameWindow 的头文件自动变成了 gamewindow.h，暂未改正。

GameWindow：最主要的类。main 函数生成后，所有其他的面板型的对象都以他作为父对象（或者“祖父”对象），也就是说 gamewindow 的对象一旦回收，所有的面板对象也将被一一析构。另外 GameWindow 中定义了内聚的子类 MapUnit，即地图块。

MapUnit：地图块，当前赋予其 4 种可能的材质，用枚举表示。重载了 Item 所必须的设置碰撞/选择框的 boundingrect 方法、将自己绘制出来的 paint 方法以及设置边缘的 shape 方法。

PanelMainMenu：主菜单界面。暂设两个按键，play 和 option，option 功能未添加。点击 play 后会调用 GameWindow 的 initGame 方法，开始关于地图的一系列初始化。

PanelOperate：Element 的操作面板。对 Item 点击左键弹出菜单已基本实现，但因时间原因仍没有实现只针对 Element 的筛选功能。现阶段对所有 Item（Element 和地图块，MapUnit）都可以左键在右下方弹出菜单，点击空白（如灰色背景格子）处或点击右键可隐藏（delete）菜单。尚未添加任何按钮。

PanelHeadUp：正上方的面板。由于时间问题，没有对其进行细节处理，只是左上角预留了一个没有绑定功能的按钮。双方各六个 Element 槽也未实现。

PlayerController：借鉴网上代码只实现了 30 帧刷新功能，定义了未使用的 bIsMyTurn 成员。并且在构造函数里添加了一个 Element 的创建。

Element：未添加任何功能，和 MapUnit 一样重载了三种方法。

ElementPool：功能尚未实现。应该是近期完成的目标。

2.代码中所有"{"都是另起一行的，这是因为 Qt 自带的代码、网上的代码对成员函数的定义大部分都是这样做的，故并未改正。

3.部分变量命名不规范，望加指正。

记录员 陈德政

2015.4.7

0.关于 Qt 的提醒（科普）：

1）Qt 中不用 C++自带的 STL，比如 vector，而是使用 Qt 带的 QVector。

2）Qt 中，GraphicsItem 就是其中可以交互的单位；GraphicsScene 是存储背景图片或者说大地图（暂用 2000*1000）的数据结构，其本身不能附着任何面板、对话框，仅仅是一个大数据库；GraphicsView 是提供给用户用来观察前者的接口，可以定义视口大小，也可以外挂上别的面板。

3）gamewindow.qrc 文件中存储的是所有导入的文件的资源配置文件，生成可执行文件的时候会将对应图片压成二进制信息。资料显示：“qrc 列出的资源文件必须位于 .qrc 文件所在目录或者其子目录下”，故之后将图标资源文件夹拷了一份放在了 Resources 文件夹下。

1.已实现只针对 Element 的点击弹窗事件，可用鼠标正反键点击地板和空白而不会崩溃。另外说明一点：弹窗拥有一个 0.2s 的延迟，只是为了防止弹出太过突兀。另外添加了没有实现很好效果的滚轮事件处理功能，也就是在视口中如果上下滚轮，由于弹出的面板是悬浮在 GraphicsView 上的，也就是在视口上方悬浮，故不会随之移动，此时应该将其 delete 掉，但现在只实现了鼠标

只有在面板中上下滚轮时才使其消失的效果。更好的处理方法已经想到，需要将 View 独立成一个新的类，将会改日进行测试。
2.ElementPool 已经实现，随 PlayerController 构造时一同被构造，故多个玩家会有多个 ElementPool。里面存放的是玩家 Element 的指针。弹出面板的事件中测试的就是 Item 是否在 ElementPool 中。
记录员 陈德政

2015.5.1

这个版本就是将 1.0.1 中 gamewindow 类中的 MapUnit 分出来定义在 MapUnit.h 和 MapUnit.cpp 中了。
记录员 罗吕根

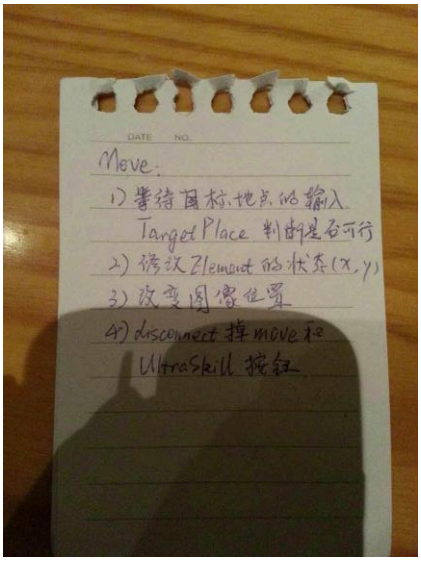
2015.5.4

//修改菜单
PanelOperate.cpp
PanelOperate.h
gamewindow.qrc
Resource 文件夹内附 4 张 WarcraftIII 的图标
//修改返回键
gamewindow.cpp
PanelHeadUp.h
PanelHeadUp.cpp

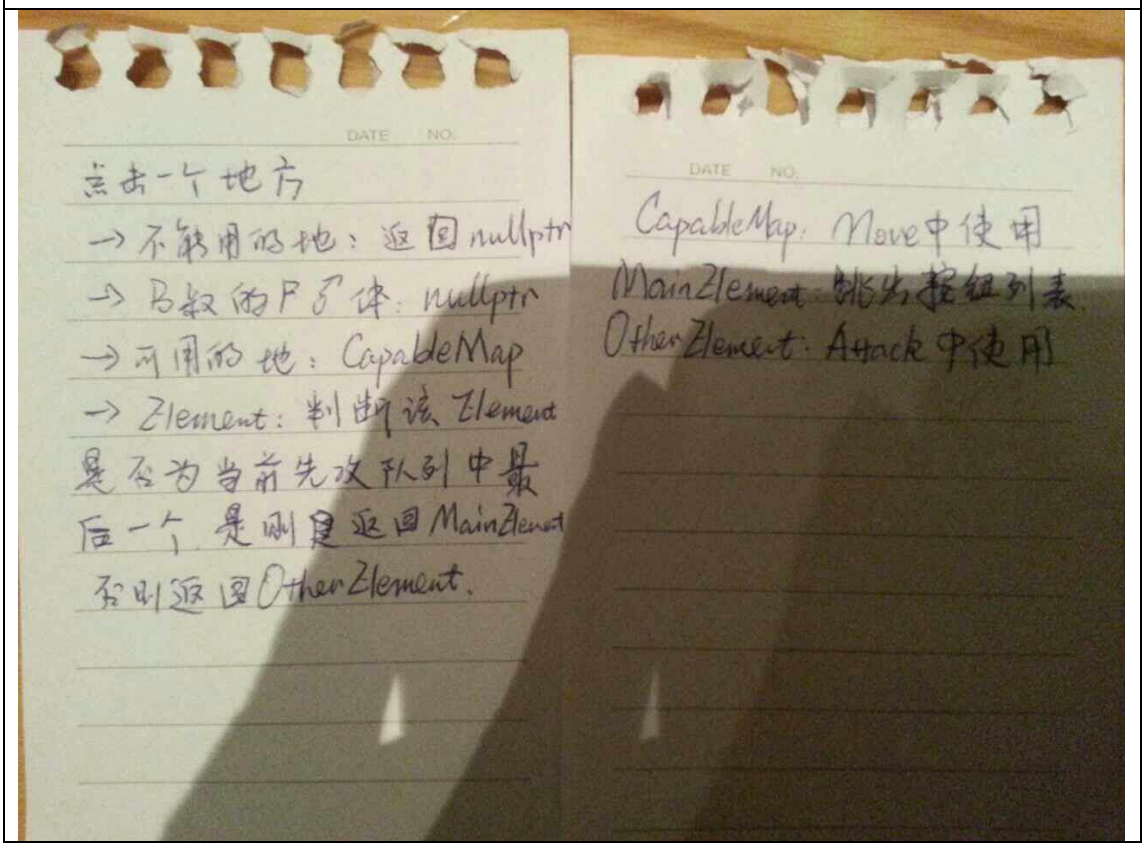
记录员 朱晨畅

2015.5.5

将增添内容加以整合,并添加了 view 的 wheelevent 功能,修改了部分类的定义。 具体修改如下:
1. 将 MapUnit 从 GameWindow 类中独立出来,作为一个单独的类,其余不变。
2. 增加左上角返回菜单功能。
3. 点击 Element 后弹出由四个图标构成的 PanelOperate,为之后的攻击,移动,取消,特殊技的实现预留了按钮。
4. PanelOperate 基本功能草图如下：

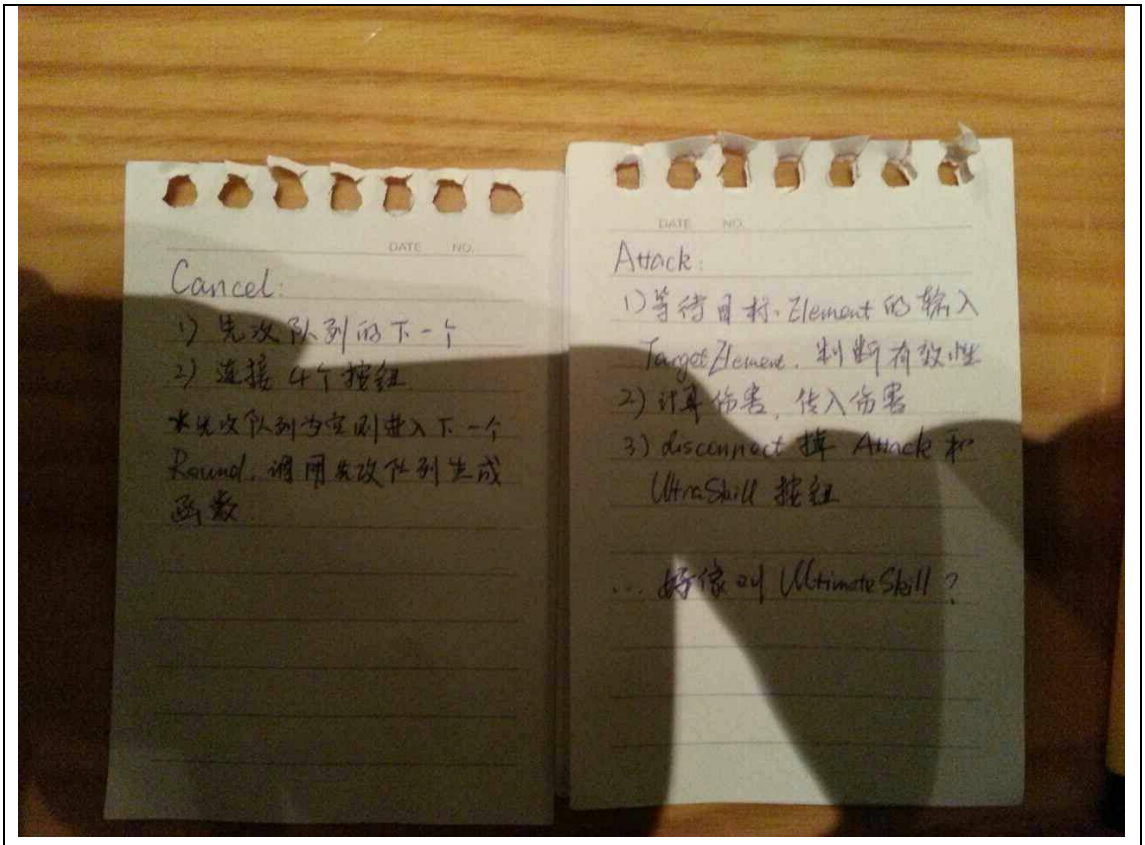


Move:
1) 等待目标地点的输入
Target Place 判断是否可行
2) 获取 Element 的状态(x, y)
3) 改变图像位置
4) disconnect 掉 move 和
UltraSkill 按钮



点击一个地方
→ 不能用的地：返回 nullptr
→ 已死的 FF 体：nullptr
→ 可用的地：CapableMap
→ Element：判断该 Element
是否为当前先攻队列中最
后一个，是则是返回 MainElement
否则返回 OtherElement.

CapableMap：Move 中使用
MainElement：能为按钮列表
OtherElement：Attack 中使用



Drawn by 朱晨畅

5.	将 view 独立为一个新的类,定义为 GameView,继承于 QGraphicView。另外 PanelOperate 设为 GameView 的子对象（就是说 GameView 消亡了它也会跟着消亡）而不再是 GameWindow 的子对象。原本为了实现点击出现 PanelOperate 故重载了 GameWindow 的 mouseEvent,但这样做不好实现滚轮事件,故取消了 GameWindow 的 mouseEvent 的重载,而是用了 GameView 的 mouseEvent 重载,现在已能任意滚动滚轮消除 PanelOperate。
6.	增加了新的头文件 function.h，任何有用的辅助函数都可以放在里面，现在在里面放了原本定义在 gamewindow 里的 wait()函数(通过以上一些修改也使得 gamewindow 的代码长度大大缩短)。
7.	GameView 有 controller 指针的成员，构造时需要 controller，这是因为只有 controller 才能使用 elementpool 的查找功能接口，而 view 的鼠标事件需要用这个接口。相应的 PlayerController 的构造就不需要 GameView。为此 gamewindow 中初始化各个元素的顺序有些调整。
8.	用到的图片都放在了 Resource 下的 IconSrc 文件夹内，并且默认的 qrc 文件中存放的所有资源所用的路径都改成了相对的 **Resource/IconSrc/…**,直接打开项目就能正常使用引用的资源而不用修改路径。
	记录员 陈德政

2015.5.6

主要是调整了整个程序的类的构架,简化类图可参考**类简单说明.pptx**,具体修改如下:	
1.	多个 Element 构成 ElementPool,多个 MapUnit 构成 MapInfo(注意 MapInfo 不再是静态类)，GameContent 包含了 ElementPool 和 MapInfo，其在初始化时会对 ElementPool 和 MapInfo 分别初始化，GameContent 是唯一一个可以访问游戏信息（地图和 Element）的类，通过它的 Map()和 Element()接口可以访问到 MapInfo*以及 ElementPool*。

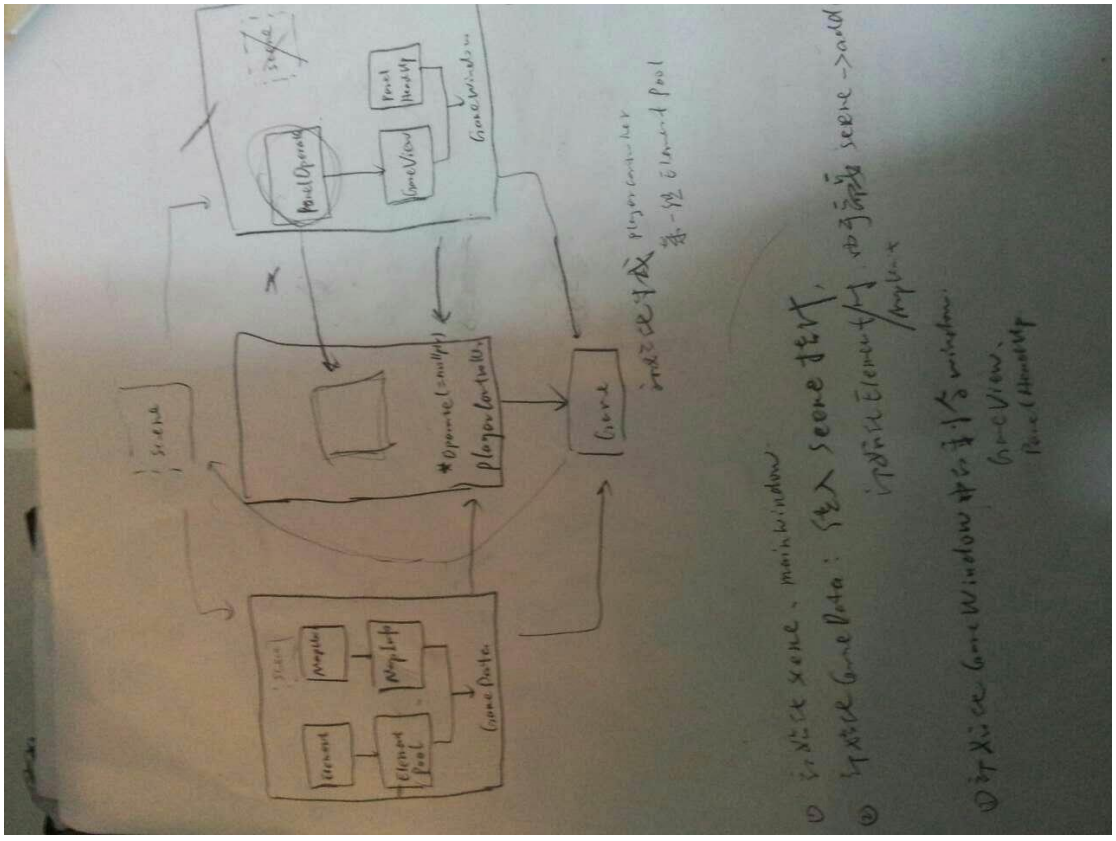
2. 删除 PlayerController，将 GameView 更名为 PlayerViewController，其实就是 View 将会兼有作为 QGraphicsView 的功能和 PlayerController 的与 Element 控制相关的功能，这么做算是尝试多种情况后的一种妥协，这是因为：

（1)在 View 的鼠标点击事件中,必须要能够访问 ElementPool 以检查是否是所需的 Element 以便决定是否生成 PanelOperate；

（2）若使用独立的 PlayerController 对象，由于 PanelOperate 以 View 为父对象，而 PanelOperate 的按键事件应该能触发 PlayerController 的一系列操作，这就需要将 PlayerController 与 PanelOperate 之间 connect 起来，但一方面 PlayerController 与 PanelOperate 并无直接关系，若想 connect 必将以 View 作为桥梁，另一方面能 connect 的只有 QObject，这就必须要将 PlayerController 声明为 QObject，但这样做之后，connect 仍会有报错（可能是因为不是典型的 QObject，如 QGraphicsView 等）。

3. ElementPool 在初始化时会创造 2*6（定义在 Constant.h 中）个 Element，装在 ElementPool 对象的 QVector（仅有一个）中，前六个为 Host 的，后六个为 Client 的。Element 有名为 Index 的成员，Host 为 1~6，Client 为-1~-6，区分 Host 和 Client 的方法是 PlayerViewController 中有名为 State 的结构（具体见源码，不难理解），此结构中存放一系列与游戏进行过程有关的状态变量（许多是 bool 型），其中有一个名为 IsHost 的 bool 型变量，此变量在网络连接时初始化，Host 一方的为 true，Client 一方的为 false，默认为 true，在 View 的鼠标点击弹 PanelOperate 事件中，将会检查是否是本方的 Element，只有本方的才会弹出 PanelOperate（此功能已实现，由于默认 IsHost 为 true，故只有左边 6 个 Element 能弹菜单，右边的不行），另外在 View 初始化后也会将视野移向索引为 1 的 Element 处，对于 Client 则是-1。

4. 手稿如下：



Drawn by 陈德政

5. 访问 ElementPool 中的任意元素：
在 PlayerViewController 中，_Content->Elements()返回 ElementPool*，
_Content->Elements()->get_Element(index)就能访问任意（双方）Element
对于 Element，可用接口如下：
返回其索引 1~6，-1~-6
int Index(){ return _Index; };
返回所在行数（从 0 计数）
int Xcell(){ return _Xcell; };
返回所在列数（从 0 计数）
int Ycell(){ return _Ycell; };

6. 访问 MapInfo 中的任意一块 MapUnit
在 PlayerViewController 中，_Content->Map()返回 MapInfo*，
对于 MapInfo，其中可用接口如下：
检查某行某列（从 0 开始计数）地图块是否存在
bool check_MapUnit_exist(int x, int y)
返回加上一个偏移量后的某行某列处的 MapUnit 的指针（设计此接口的考虑是方便某 Element 为中心的范围操作）
MapUnit* get_MapUnit_around(int x, int y, int offset_x, int offset_y)
返回地图宽度（比最右的MapUnit的Xcell大1）
int get_map_width()
返回地图高度（比最下的MapUnit的Ycell大1）
int get_map_height()
返回地板材质
QVector<emFLOOR>& get_floor_info()
返回QVector*，此接口一般不要使用（初始化中使用了此接口）
QVector<MapUnit*>& get_MapUnit_vector()

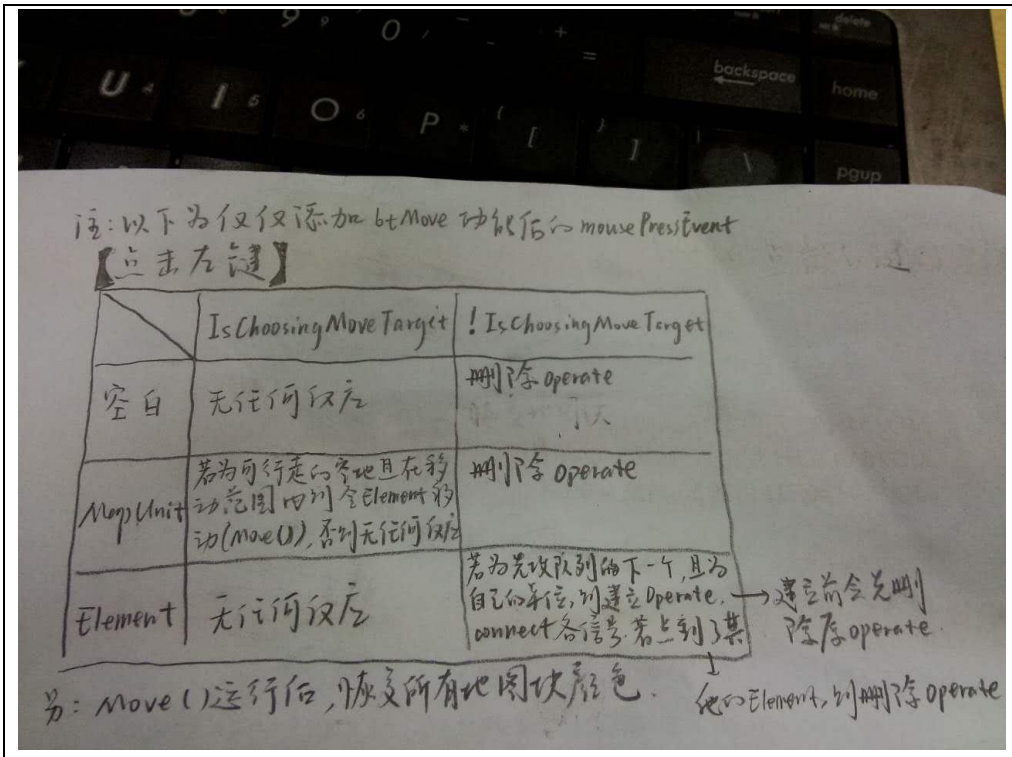
7. 所有常数定义都放在Constant.h中，尽量不要使用所谓凭空出现的“魔数”。

记录员 陈德政

2015.5.11

1. 完成了Move和Attack的大部分功能,包括点击Move按钮后的MapUnit高亮(用到了BFS,能判断墙阻隔的作用),Element的移动,点击Attack按钮后的敌方Element高亮(连接了Character类，调用了其中的是否在射程内的一些接口)，Attack后生命值减少(调用了一些Character中的接口)

2. 下为mousePressEvent设计草稿:



Drawn by 陈德政

3.	构建了一个新类DamageLabel，其以PlayerViewController为父对象，构造函数中输入damage的值并显示在屏幕正中央(0为“MISS”)。在PlayerViewController中有show_damage的函数，用于构造DamageLabel并设置3000ms后deleteLater的计时器

记录员 朱晨畅、陈德政

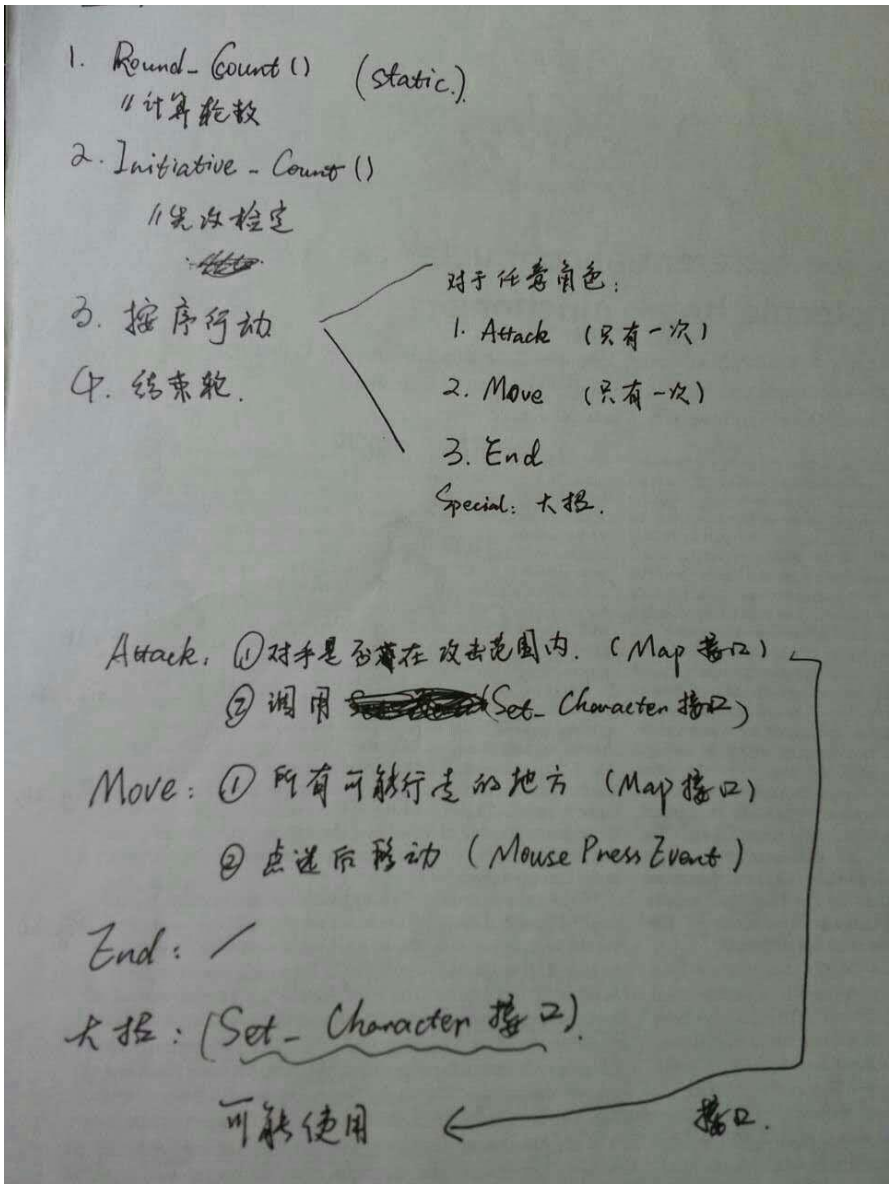
2015.5.13

导入了各职阶的图片，其余微调

记录员 陈德政

2015.5.16

1.	游戏循环基本建立，initNewRound和initQueue的实现
2.	游戏循环参考草图：



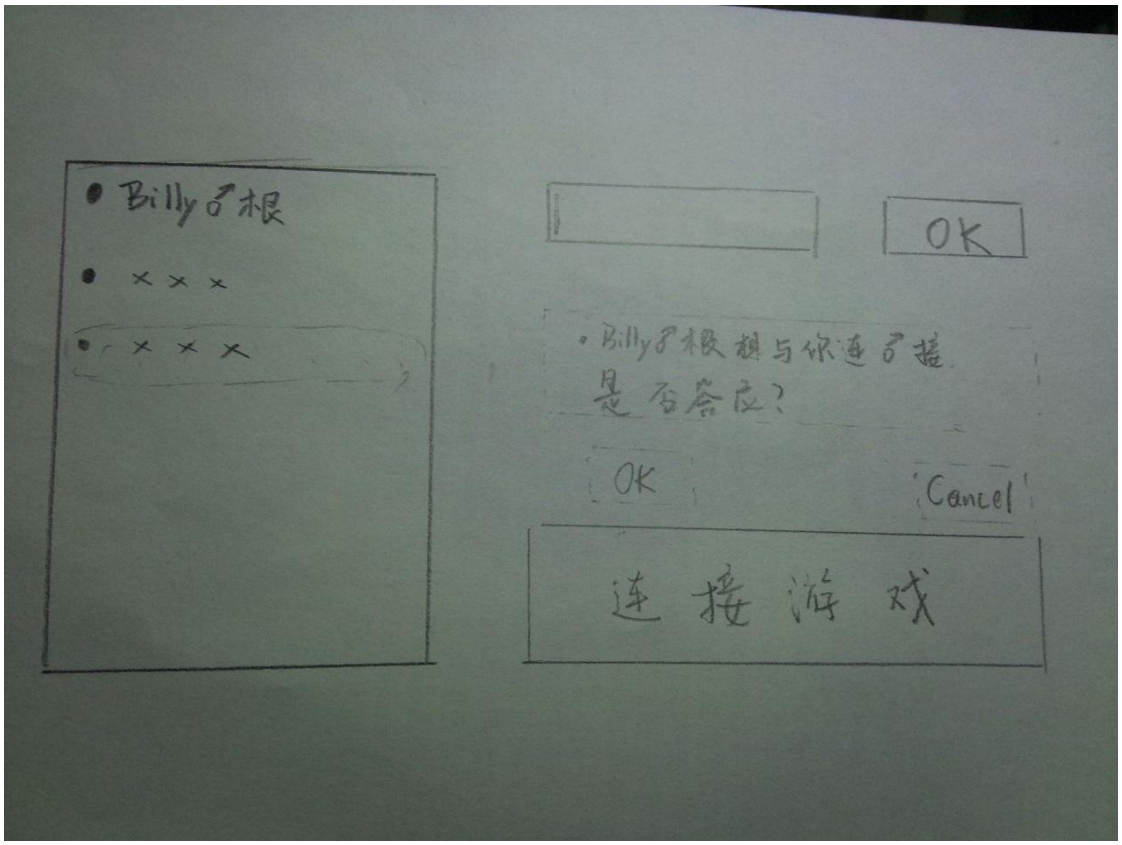
Drawn by 朱晨畅

3.	默认的AI，点击Cancel
4.	血条初步实现，素材尚需调整（**许多素材都需要重新编辑和组织**）

5. MultiPlayer进入游戏的流程已初步设想好，尚未实现真正联网

基本流程如下：

1) 点击play，隐藏所有原先的Option和Play按钮，出现新的ListWidget和TextEdit框以及“OK(Ok)”“连接游戏(Connect)”按钮
2) 左边ListWidget显示所有能连接的客户端，右边TextEdit可以输入自己的名字。按“OK”键，可将自己的名字加入List。（**注：这里用的是QListWidget，类似于QGraphicsItem，每一栏就是一个QListWidgetItem，生成后需要用QListWidget将其一个一个加入进去**）
3) 点击左边的列表中的非自己的项后，且输入的名字有效（不为空且不为重复）时，右侧的第二个按钮“连接游戏”有效
4) 点击连接游戏后，按钮上方显示“等待对方确认”
5) 对方：“连接游戏”上方显示原先被隐藏的面板，确认(Confirm)是否与对方连接，选是则进入游戏，选否(Cancel)则断开连接



Drawn by 朱晨畅

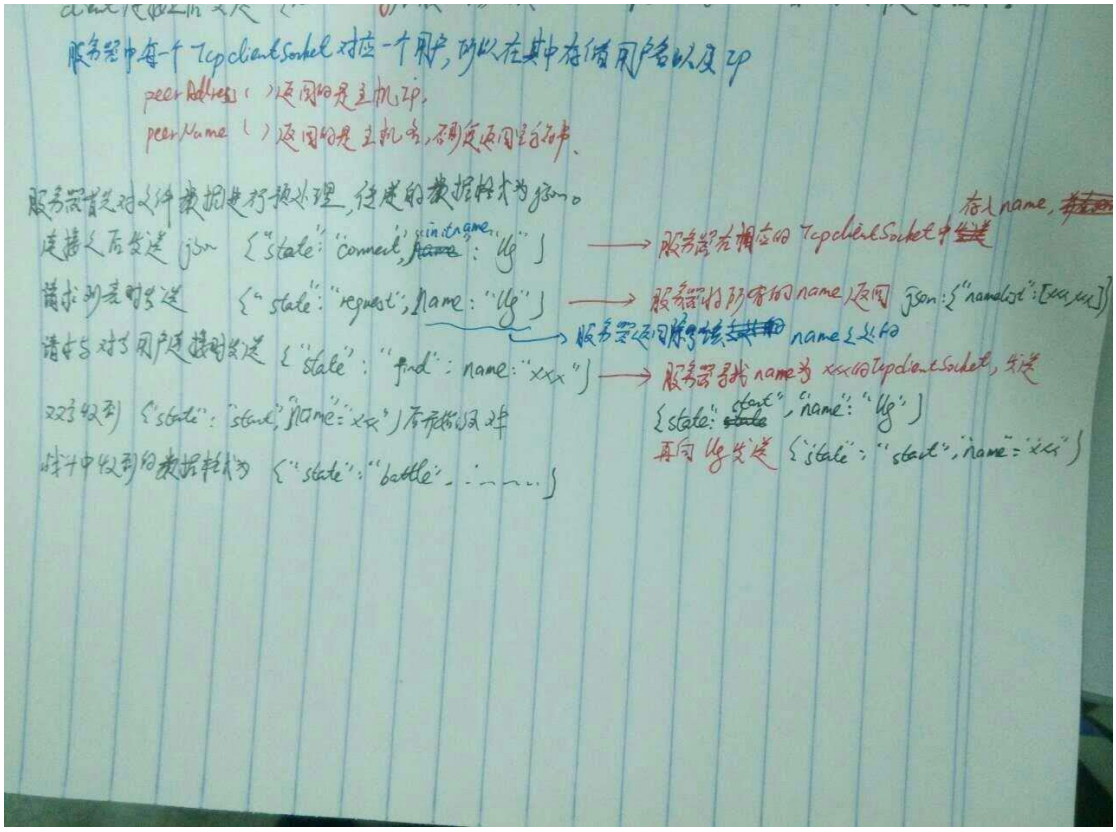
记录员 朱晨畅、罗吕根、陈德政

2015.5.18

该版本舍弃了单机模式，是一个将大部分逻辑处理移入服务器的大更新；

【此版本用了socket的指针所以可能会造成一系列的网络安全问题,不过没有做好友列表前无所谓请自动忽略】

1. 点击上面的OK后不会将自己名字存进左边的【这样没什么意义不是么= =】，只是会将自己的名字发送给服务器，如果有重名部分服务器会返回一个error值，则必须要重新输入，此时连接游戏的按钮unable。如果不重名则选择左边的某个在线的人则可以开始对战，新增刷新在线用户按钮。【这个到时候也可以做成自动刷新】

2. 用于连接的tcpclient放在了gamewindow中，用指针传递给了PanelMainMenu，【所以其构造函数也发生了相应的变化】
3. 在点击OK时向服务器发送名字，如果名字重复则需要重新输入名字，此部分在PanelMainMenu的InputName里面。【返回名字错误的动作还没有加上去】
4. 点击connect由于还没有选择对战玩家的功能，所以暂时默认让点击connect的玩家和名字为”llg”的对战，此时发送的数据见程序；
5. 当两者都确认开始时【就是建立PanelMainMenu】时会向服务器发送确认，服务器等两人都准备好了之后开始进入游戏循环，先发的人永远都在左边，谁先发暂时不随机。
6. 进入游戏后客户端采取事件驱动，即受到消息后才能进行相应的动作，暂时只能点击cancel之后才能进入下一个棋子【删除了之前点击完attack后move后就会自动跳转】，攻击造成的伤害、棋子是否死亡都放在服务器判定，由服务器判定胜负，返回相应的事件，对战中如果有一个玩家掉线则直接通知对手胜利。
7. 服务器的中存放了每一局的信息，每一局都是一个chessboard，根据playername找到玩家所在的棋局，server的封装并不是很好不过暂时不用管。如果一局结束会删除该棋局。
8. 具体的逻辑见各种事件以及相应的信号槽。
9. 至于单人对战和双人对战：要添加双人对战只用新建一个PanelMainMenu【可以把之前的那个直接拿来用】，根据用户选择直接判定新建哪种PanelMainMenu。
10. 网络传输设计草稿：    Drawn by 罗吕根
记录员 罗吕根

2015.5.18

1. server修复了伤害显示错误的bug；

2. 客户端做成单人模式和双人模式都可以使用，单人模式接口在SingalPlayerController中，双人模式在PlayerViewController中；
需要改进： 1. 双人模式的进入后需要选择对手； 2. 双人对战不允许退回面板； 3. 进入双人模式之后可以返回前页； 4. 网络连接中断返回最开始页面，提示网络中断； 5. 增加网络连不上的通知； 6. 增加各种动作；
记录员 罗吕根

2015.5.19

1. 加入了singlePlayer下的沿Move路径移动的动画（技术支持：朱晨畅），在网络部分尚需完善。其由串行动画组QSequentialAnimationGroup实现，代码较为集中。
2. 将notePossibleAttackTarget改回了void型，并将移动后攻击自动next的设定删除。
3. 将QTimer全部变为QTimer::singleShot。
4. 与“singal”相关的所有东西都改成了“single”，不知服务器中会不会有影响。
5. 加入了主菜单中的exit功能。
6. Move时，任何Element与墙有相同的阻挡效果。
记录员 朱晨畅、陈德政

2015. 5.19

1.将notePossibleAttackTarget改回了bool型。
2.增加选择NameList中的对手进行游戏，新增Dialog通知是否同意对战。
3.名字如果不重复会有来自服务器的通知。
4.双人对战的动作完成。
5.新增刷新用户列表功能，button图标欠缺。
6.新增应用程序的图标，可以在Debug文件里面见到exe文件的图标。
记录员 罗吕根

2015.5.20

1.Element类的ElementResoursePath数组修改，以及paint函数的修改。
2.SinglePlayerController.h中新定义的类，加入类的声明，在SimplePlayerController类中加入两个private变量。
3.在SimplePlayerController类的构造函数中加入新增变量的初始化。
4.在notePossibleMoveTargetMapUnit，move和attack中修改与新增与动画有关的语句，详情查看程序内注释。
5.修了Set_Character.cpp的一点小BUG，Death函数修改至无问题。
6.修改show_Damage的颜色和大小。
记录员 朱晨畅

2015.5.21

主要修改了界面效果:

1. 把Operate和上面的四个按钮效果做了调整,后续应该会进一步修改.初步设想是Operate能访问到State,在每一次创建Operate的时候会根据当前HasAttacked、HasMoved、HasUltimateSkilled、Element的类型创建相应的效果，不能使用的功能将变成灰色（disable）。目前只实现了鼠标停留（hover）在四个按钮上的发光效果。另外添加了UltimateSkilled基本接口
2. MultiPlayer界面的大修改。基本布局调整，增添了刷新和返回前一级主菜单的按钮，但选择MultiPlayer后返回主菜单再进入SinglePlayer时存在的BUG，尚未解决，应为网络相关。另外增加了文本输入框输入前灰色，点击输入框输入时变为白色的效果（实际是设置样式表，当输入框focus时颜色不同）。
记录员 陈德政

2015.5.24

小幅度修改Character_Element 重写Set_Character Element中增加了tomb_note和active_note函数 GameState内新增变量 修改或新增了SingalPlayerController的Move，Attack，UltimateSkill相关和MousePressEvent等函数 PanelHeadUp的set_health大改，加入了血条减少和变色动画
附上棋子Ex技能设定: Saber： 30面板伤害及以上开启大招 大招射程10，必中固定25伤害 Archer： 25面板伤害及以上开启大招 大招射程2，修改一定数据，打击4次或直到MISS或直到enemy死亡 Rider： 3格内不论敌我都进行先攻检定，若结果不超过15则受到等于当前血量数值的伤害 Assassin： 50%几率闪避所有伤害（除了Caster的攻击） Caster： 3次任意门，传送距离15，寻路算法同移动 Berserker： 2回合后（死亡round+2的round的计算先攻队列时）复活，并且该回合处于无敌状态
修改:将音乐和音效,operate按键变灰的内容整合
记录员 朱晨畅、罗吕根、陈德政

2015.5.26

1.新增联网模式下的Ex技能；

2.修改saber走路的音效和archer攻击的音效；
3.服务器做出了大量改动；
4.还有少量不会固定出现的bug，例如Caster的移动问题，之后得到α版测试时再改进；
记录员 罗吕根

2015.5.27

本次更新由于是整合故内容较多：
1. 随机AI的加入，虽然有少许BUG（如有时候周围没有敌人时会不采取任何行动且mapunit的note不会被消除），但已经基本能正常运行，下一步应该是增加主界面singleplayer的page，可以选择难度。
2. 类的改名，PanelMainMenu -> PanelMenu，主菜单新建了名为GameMainMenu的类，注意PanelMenu中修改较多，但应该较好理解（将各个部件划归到各个Page上进行管理是关键，这样做deleteLater主page后，上面的所有部件都会被释放掉）。
3. 网络的整合(主要是对playerviewcontroller的修改)，基本没有问题，差加入联网时的音效。
4. GameMainMenu不是View的子对象，而是直接为gamewindow的子对象，点击左上角mainmenu键后，原来view上的一切鼠标和滚轮操作均会无效，若之前出现过operate，则会将其关闭(调用resetOperate)，之前处于选择mapunit和element的状态不会受到影响，点击GameMainMenu上的resume后仍保持原状，可继续选择目标。
5. option的改音量功能已实现，在GameMainMenu中也可以改音量(新增的MusicPlayer类是设想将背景音乐和音效都包装好，但现在还没有实施)。
6. option的换地图功能已实现，默认会进入Churchyard地图。地图信息在MapUnit.h和MapInfo.h中被定义为const static。另外地图的缩略图和地形贴图虽已经找全，但不是特别理想，以后会有时间再改，再者地图块的透明度降低了，调为了70%不透明，原本是50%。
7. 两个controller都粗略对各个函数做了划分，使用VS2013的ctrl+M,ctrl+L的折叠功能（技术支持：朱晨畅）更容易找到所需的函数。
8. Element换了一套新的头像,外圈更厚,便于区分。
9. Resource文件夹的组织有了些变化。
记录员 朱晨畅、罗吕根、陈德政

2015.5.27

1.新增联网模式下的地图选取，以邀请者选择的地图为准；
2.新增网络模式下攻击、移动的音效；
3.修复双人对战后不能继续开始一场对战的bug；【测试时服务器崩溃过，不过原因不详，还有待发现bug原因】
4.新增一方在游戏中退出，对手直接通知胜利的功能；
记录员 罗吕根

2015.6.1

1. 新增了PanelBattleInfo类和ElementBattleInfo类，PanelBattleInfo上面每一栏都是一个ElementBattleInfo，主要接口为: 1) PanelBattleInfo类的RefreshAllBattleInfo函数.主要作用是刷新面板上所有棋子的信息，目前只在Next和Attack中被使用，详见singleplayercontroller的这两个函数。在造成伤害后会更新HP数值，死亡时会将图标变成灰色（此死亡是指Death(get_round_death())，B叔的复活没有测试，不过应该问题不大）。另外在大招造成伤害或死亡的语句后也应将此函数添加进去 2) PanelBattleInfo类的RefreshNewRoundInfo函数，主要作用是载入新的先攻队列，其只在initNewRound中被使用
2. 选择AI难度的页面和逻辑已经初步做好，目前可以选择四个难度（加入了Hard难度）。由于时间和精力问题没有认真去美化界面，后续应该会有所调整。
3. 先攻队列由QQueue型改成了QVector，没有太大影响
4. 开始游戏、新round时的文字动画，以及next，initNewRound和AI动作都加了延迟。
5. 修复了移动时穿过Element的BUG，主要是修改了get_Element_with_pos函数。
6. 网络部分的微调。
7. 不知道是不是电脑的原因，进游戏之后（选择AI难度进入游戏主界面后）BGM会出现BUG
记录员 朱晨畅、罗吕根、陈德政

2015.6.1

1.修改Rider和Archer大招，Rider大招改为与攻击动作时机相同，Archer大招改为状态切换。
2.SinglePlayerController到PlayerViewController成吨的移植；
3.网络部分大量的修改，现在的服务器非常稳定；
4.少量图片的修改；
5.增加WASD以及拖拽的方式控制视角的功能；
需要修改的地方： Archer攻击距离不同动画也应该不同； 增加更困难的AI； 调整大招音效，现在暂定Berserker复活时有音效；
记录员 陈德政

2015.6.2

1.新增Lunatic难度，但是仍有BUG；
2.修改_Scene长宽，使其随地图大小改变；
3.新增随地图更换背景；
4.新增随地图换BGM；
记录员 罗吕根、陈德政、韩昌佑

2015.6.7

1. 【注】：为测试方便，set_character.cpp L85修改了saber的初始大招数目为2，原始为0.同样的，L121修改了rider的初始大招数目为2，原始为0.还修改了SinglePlayerController.cpp L365，将saber大招范围由10改为了100，另外还有Element.h L26修改了saber先攻值由2变为了12。

2.	全部按键的逻辑都放在operate里判断，btUltimateSkillPressed里不再提供任何大招能否释放的判断（见SinglePlayerController和PlayerViewController中的btUltimateSkillPressed函数以及PanelOperate的构造函数），**具体可见PanelOperate.cpp**。类似的，btAttackPressed以及btMovePressed也不提供逻辑判断（即只要按键处于有效状态，点击就能触发）
3.	Saber大招特效已更新
4.	Assassin大招特效已更新（SinglePlayerController）
5.	将PlayerViewController中的内容大部分做了整合，没有做整合的有： 1) assassin的闪避动画。在attackAnimation函数中 2) archer的音效。archerAnimation并未删除，里面简单写了播放对应音效的代码。可能还需修改 3) 新round、next的延时没有加入网络中，可以考虑是否加入
6.	根据当前鼠标位置动态出现operate已经实现。
7.	柳洞寺地图已加入。
8.	Lunatic难度已更新。
9.	尚需实现的东西：offline模式、许多大招图标、saber和archer头像。
	记录员 罗吕根、韩昌佑、陈德政

2015.6.7

1.将assassin的闪避动画、archer的音效以及大招之后换头像进行了完善；
2.将每个人的大招图片进行了替换；
3.更改了saber的头像；
4.柳洞寺地图的微调；
5.原multiplayer模式分成offline模式和online模式；
记录员 陈德政、罗吕根

2.系统需求说明

2.1 系统总体功能

2.1.1.系统总体功能

本系统主要实现了单机 AI、单机双人对战、联机对战、设置更改和退出系统等功能。其中单机 AI 实现 Tutorial、Easy、Normal、Hard、Lunatic 五种难度，设置更改中加入地图选择、背景音乐和音效

2.1.2 系统目标

Fate/undefined 是一款战棋类游戏，本系统主要目标是用户能够在单机或联机的情况下顺利进行游戏。

2.2 环境需求

2.2.1.开发时的软硬件环境

操作系统	64 位 win7/win8	编译环境	vs 2013
插件	Qt5.4	硬件配置	CPU：i3 及以上

2.2.2.运行时的软硬件环境

操作系统	64 位 win xp/win 7/win 8	推荐配置	CPU：core 2 以上

2.3.系统功能概述

1、 SinglePlayer：点击 SinglePlayer 按钮即可开始与 AI 战斗的新游戏，点击时可以选择 AI 难度，从 Tutorial 到 Lunatic 不等。

2、 MultiPlayer：点击 MultiPlayer 按钮即可开始与别的玩家对战的新游戏，点击时可以选择在同一台机器上(offline)或是联网寻找玩家(online)。

3、 Option：点击 Option 按钮即可进入选项页面，在此页面中可以调整背景音乐和音效的声音大小，同时可以选择对战地图。

4、 Exit：点击 Exit 按钮即可退出游戏。

5、 左键调出面板：当一枚棋子在其行动轮内，该棋子会被高亮。若用左键点击这枚棋子，则会出现操作面板。操作面板的位置会由棋子当前的位置决定，一般会出现在右下角，若是棋子太靠下或是太靠右则会出现在左上角或是右上角等方向。面板上含有移动、攻击、Ex 技能和结束行动轮四个按钮。

6、 标注移动范围：当一枚棋子点击移动按钮时，这枚棋子可以选择的移动范围将会被高亮。移动范围基于棋子周围地形、其他棋子位置和自身属性而定，通过广度优先搜索实现标注。

7、 移动：当一枚棋子在移动范围高亮的状态下点击某一处高亮的地图区域，这枚棋子将会遵循最短路径被移动到该区域。最短路径通过一次与标注移动范围时相似的广度优先搜索记录路径，并以连续动画将其串行成平滑移动。

8、 标注攻击对象：当一枚棋子点击攻击按钮时，这枚棋子可以攻击的对象将会被高亮。攻击对象基于棋子自身的攻击范围这一属性而定，通过遍历所有棋子实现标注。

9、 攻击：当一枚棋子在攻击对象高亮的状态下点击某一高亮攻击对象，这枚棋子将会对该对象进行一次攻击。这次攻击会调用 Attack 函数，并分为攻击棋子是近战还是远程、被攻击棋子是否死亡、对手是否发动 Ex 技能等等原因使用不同的动画，并精心调整了时间轴使得动作看起来非常连贯。

10、 Ex 技能：一枚棋子只有在合适的情况下才能发动 Ex 技能。在这种情况下点击 Ex 技能按钮会发动 Ex 技能，此时将会根据棋子自身属性调用不同的 Ex 技能函数，具体效果视情况而定。

11、 结束行动轮：一枚棋子随时可以通过点击结束行动轮按钮结束自己的行动轮，此时该棋子取消高亮。在先攻列表上该棋子的后一枚棋子将被高亮，并能够进行诸如移动、攻击等操作。

12、 右键取消标注：当一枚棋子在移动范围或是攻击对象被标注的情况时，并不能调出该棋子的操作面板。此时右键能够取消该棋子的标注情况并使得左键能够调出面板。

13、 左键点击拖动：在无关处左键点击地图并按住拖拽可以小范围移动地图，能够很好解决地图过大而不易看清局势的情况。

14、 WSAD 移动：任何时候按 WSAD 按钮可以实现向上下左右方向移动地图，其效果与左键点击拖动十分相似。

15、 AI 行为：AI 分为 Tutorial、Easy、Normal、Hard、Lunatic 五个难度，其行为模式各

不相同，综合而言 AI 难度逐步上升，为其设计的等效血量和等效距离等等各类高级数据的公式愈发复杂而完整。

16、 左键打开战斗信息面板：在游戏内界面点击中间 logo 按钮可以打开战斗信息面板。战斗信息面板中将会显示当前轮数、棋子的先攻队列、棋子血量和棋子 Ex 技能当前状态等信息，便于玩家作出决策。

17、 左键打开游戏主菜单：在游戏内界面点击 Main Menu 按钮可以打开游戏主菜单，其中包含于 Option 界面相似的对于背景音乐和音效的音量调整，以及继续游戏和返回主界面选项。

18、 血量显示：游戏内界面上方的 HeadUp 面板中有棋子的血量显示。当一枚棋子由于 HP 的增加或减少而导致血量变化时，HeadUp 面板中对应的血条长度和颜色会发生变化，这是通过 20 帧的颜色和长度重置实现的。当 HP 降为 0 时原本为彩色的棋子头像会变为黑白，当某些棋子释放 Ex 技能时头像和血量也会产生一定的变化。

19、 联机游戏：通过 MultiPlayer->Online 按钮可以进入联机模式，通过服务器端寻找对战对手。联机界面上方需要输入自己的名字，随后便可以查看同样在服务器端的所有对手并进行邀请。此时对方会弹出是否答应邀请的对话框。若对方答应，则双方按照邀请方所选择的地图进行对战。

3.系统设计

3.1 系统级设计决策

为实现此游戏项目，首先应将系统分解为几个子系统，而且：

- 子系统间应该有明确、简单的接口
- 子系统之间应该有较小的依赖性，而且能相对独立地设计各个子系统。即软件设计中单独模块的高内聚和模块间的低耦合

3.2 系统总体设计

3.2.1 系统设计思想

3.2.1.1 系统构思

1. 主体

游戏中所有对象的父对象是 gamewindow，其继承于 Qt 中的 QMainWindow。其首先在 main 函数中被创建并进入事件循环，接着由它为父对象按照一定的顺序初始化创建一系列游戏必要的组件对象，并开始游戏循环，且在结束游戏时被删除。其即为整个游戏的窗口，也可以说是代表了游戏本体。但在游戏中，其虽然处于“门面”的地位，构建了一切所有的游戏元素，但出于游戏结构的合理性的考虑，其本身并不参与游戏循环，而只是起到一个初始化所有游戏对象、为所有游戏对象提供平台的作用。起到游戏循环重要作用的是它创建的 **GameViewController 对象**

2. 角色系统

1) **Character_Element：棋子通用基础信息**
   包含了所有棋子通用的基本属性的设置。包括生命值、AB、AC、重击、用随机数计算的伤害等
2) **Set_Character：棋子特殊信息**
   根据不同的职阶（Choose）分配人物不同的属性，并且根据不同人物加入各自独特的特殊技

3. 资源管理系统

1) **MapInfo：地图资源信息**
   管理地图的全部信息并提供地图相关的接口。包括各个地图块的位置和类型的存储、改变/回复当前地图块的颜色、访问某个地图块的接口等
2) **ElementPool：全部棋子资源信息**
   提供外部访问全部棋子的接口。包括游戏中所有 12 个棋子指针的存储、按照需求访问某指定棋子的接口等

4. 界面系统

- **进入游戏的菜单页面组**
  主要是对多种模式选择的管理
  1) **MainPage：主页面**
     最基础的页面，刚进入游戏时显示此界面，让玩家选择是进行单人还是多人游戏，或者跳转入调节音量和切换地图的本地设置界面，又或者结束游戏
  2) **SingleplayerPage：多种难度 AI 选择页面**
     单人模式中分多种等级 AI 可供选择，目前有 5 个难度可供选择：训练（Tutorial），简单（Easy），普通（Normal），困难（Hard），疯狂（Lunatic）
  3) **MultiplayerPage：离线/在线模式选择页面**
     多人模式中分离线/在线模式可供选择，离线模式即为一台主机相同输入设备下两个玩家轮流控制各自棋子对战的模式，在线模式即为通过访问远程服务器与其他在线玩家进行网络对战的模式
  4) **OnlinePage：在线模式连接页面**
     在线模式连接页面应能满足实现网络在线对战的一些基本功能要求，如玩家名字的输入、当前在线玩家的显示和刷新、请求连接的按钮和请求回复界面等
- **游戏主界面**
  显示游戏界面所有的玩家需要知道的信息，绘制出地图以及棋子的图形，同时展现棋子行动的一切动画。
  1) **Scene：场景**
     使用的是 Qt 自带的 QGraphicsScene。用于承载所有在其上附着的地图块和棋子，作为地图可视界或者说场景而存在
  2) **GameViewController：游戏视窗**
     重载了 Qt 自带的 QGraphicsView。是连接资源管理系统和 Scene 的桥梁，同时也是用户与整个系统交互的**门面**，且参与构建游戏循环本身，**是整个游戏系统的核心**。其主要有以下三大功能：
     A. 用于在屏幕上显示资源、观察 Scene，作为镜头、游戏视角本身而存在
     B. 接受用户的键盘鼠标操作并在视窗中作出即时的响应
     C. 接受游戏流程的控制并在视窗中作出即时的响应

【注】由于单机和在线模式显示的机制不同，故需分为两个用于界面显示的类，**分别为 SinglePlayerController 和 PlayerViewController，此处为了便于说明主界面的构造先不予区分。**

3) **PanelHeadUp：抬头面板**

主要职责是实时地显示所有棋子当前的血量、生存状态，并提供游戏主菜单和游戏战斗信息面板的开关按钮

4) **GameMainMenu：游戏主菜单**

在进入游戏后，点击抬头面板的 MainMenu 按钮可弹出此菜单。在此菜单中可即时调节本地的背景音乐和音效的音量，并能选择是否退出游戏或返回游戏

5) **PanelBattleInfo：游戏战斗信息面板**

在进入游戏后，点击抬头面板的中心按钮可弹出此菜单，再次点击即可取消。在此菜单中显示了双方所有棋子的即时的战斗信息，包括双方每个棋子的先攻顺序（AttackSequence）、每个棋子的血量和大招状态，故此面板承载了重要的对战信息，是玩家在实际对战中制定作战策略的重要参考

6) **PanelOperate：游戏操作面板**

其为 GameViewController 的用户交互功能实现的具体延伸部件，玩家点击 GameViewController 所观察到的当前可操控的自己的棋子即可弹出此战斗控制面板，此面板提供了攻击（Attack）、移动（Move）、特殊技（UltimateSkill）、防御/结束动作（Defend/Cancel）四种操作，也是玩家能够操作棋子进行对战的唯一平台

5. 游戏循环控制系统

1) **GameState：游戏状态存储结构**

游戏循环控制的基本思想是事件驱动，事件的分歧由游戏当前的状态决定，状态分为玩家本地的游戏状态以及全局游戏的状态，故需要用一个结构维护当前游戏瞬间所有的状态信息，包括玩家对棋子的各种操作

2) **GameViewController：游戏视窗**

游戏视窗兼具参与游戏循环控制的本领，其通过考察到不同的 GameState 信息，配合发送与接收到的不同的信号，在视窗中展现不同的响应结果，完成游戏循环

6. AI 系统

SinglePlayerController：单人游戏视窗

单人模式中的视窗（也是离线模式的视窗），提供了 AI 的接口，并且在 gamewindow 初始化 SinglePlayerController 时会根据游戏进入界面中选择的 AI 等级调用不同难度的 AI 处理函数操纵 AI 与玩家对战

7. 动画系统

1) **单人模式和离线模式**

单人模式中，动画系统没有用独立的类去包装，而是分散在各个需要加入动画的对象的动作中，如 gamewindow 中 gamestart 提示语的上浮动画，PanelHeadUp 上的

血条变化的动画，GameViewController 中新 round 提示语的滑动动画和棋子的攻击移动特殊技动画等

2) **在线模式**

在线模式中，由于战斗双方的战斗逻辑是在服务器上完成的，且战斗动画必须在双方的游戏界面上都能按要求正常被播放，故战斗动画被额外分割了出来。

8. 网络系统

- **服务器端**

在线模式中才会使用的在服务器端运行的所有程序

1) **Character_Element、Element、ElementPool：服务器端的角色系统**

由于 Online 模式下所有的战斗数据的处理都是经过服务器的，所以在服务器建立了一个和客户端类似的没有 AI 信息的角色系统。

2) **chessboard：服务器的棋局存储结构**

chessboard 中有对战双方的玩家名、双方的棋子信息以及游戏的各种逻辑，当两玩家通过 Online 模式进行对战时会新建一个 chessboard 来存储该棋局并利用该 chessboard 对象进行各种逻辑。

3) **tcpserver：服务器的图形界面**

tcpserver 主要负责显示服务器图形界面，显示用户的登录与下线，显示服务器的端口号。

4) **tcpclientsocket：服务器与客户端连接的接口**

客户端存在 Tcpclient 与服务器连接，响应的，对于每一个连接上服务器的主机都会对应一个 tcpclientsocket 来收发数据。

5) **server：服务器信息处理中枢**

server 包含了各个连接到服务器的 tcpSocket 以及正在进行的棋局（chessboard）。当 tcpclientsocket 收到数据后，会对数据进行简单的分类然后将数据传给 serve 调用不同的处理方式。对于 chessboard 的更改也是在 server 中完成的。

- **客户端**

在线模式中才会使用的在客户端运行的程序

1) **TcpClient：与服务器沟通的接口**

在线模式中才会使用，客户端收发的各种数据都是通过 TcpClient 完成的。TcpClient 收到服务器传来的信息先会解析信息的内容再调用不同的信号函数通知 PlayerViewController 等完成相应的动作。

2) **PlayerViewController：玩家游戏视窗**

在线模式中的视窗，基本功能类似，但考虑到了网络的数据传输故将其中的动画部分分割了出来

3) **manager：请求公告栏的接口**

manager 不是一个单独的类而是一个建立在 gamewindow 里的指针。manager 是 QNetworkAccessManager 类的指针，用来向 http 服务器请求最新的公告（例如游戏版本更新等等）。同时由于使用 Tcp 协议要显示没有连接上服务器需要增加繁琐的步骤，所以采取了向相应的服务器发送 http 请求来判断是否已经连接上服务器。

3.2.1.2 关键算法与数据结构

■ 关键算法

1. 游戏初始化

【gamewindow.cpp/PanelMenu.cpp/GameContent.cpp/GameViewController.cpp】

游戏初始化是一系列有序且逻辑严密的连续动作，主要是 gamewindow 对各个对象的构建和游戏循环的启动。按照时间顺序，游戏初始化首先可以分为两个重要部分：

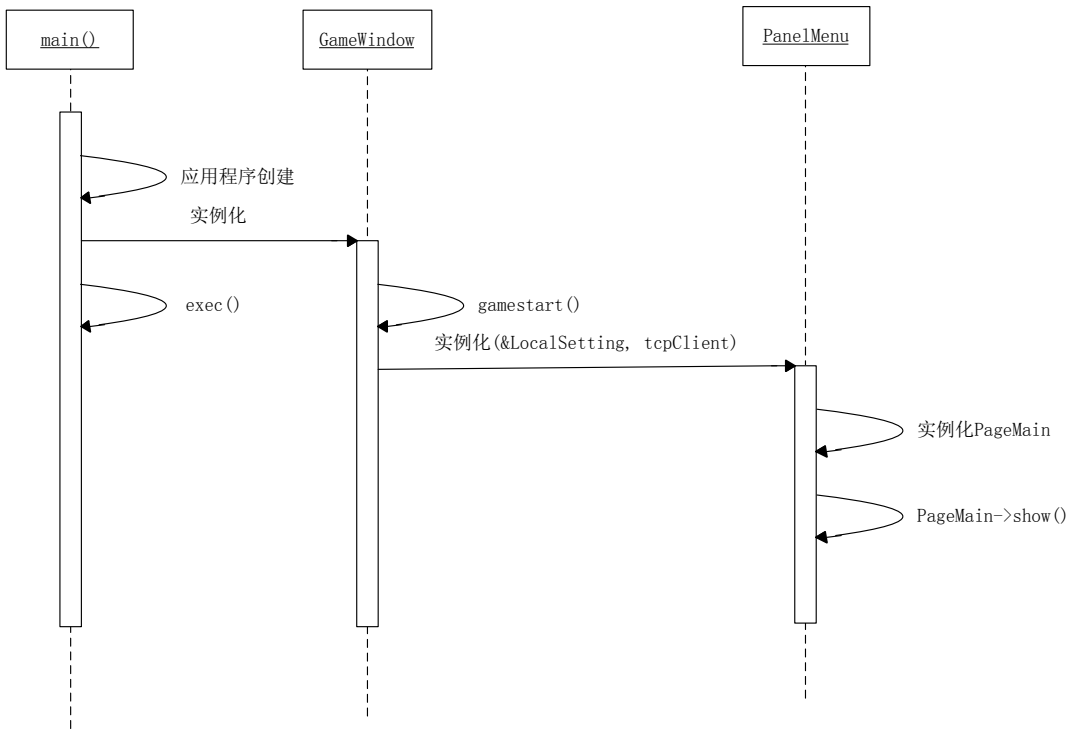
- 进入游戏界面
- 实际开始游戏（单人/离线/在线）

下面分别叙述这两个过程中游戏初始化的流程

A. 进入游戏界面

此阶段从游戏 main 函数调用开始到生成游戏进入主界面为止

对象生命周期图如下：

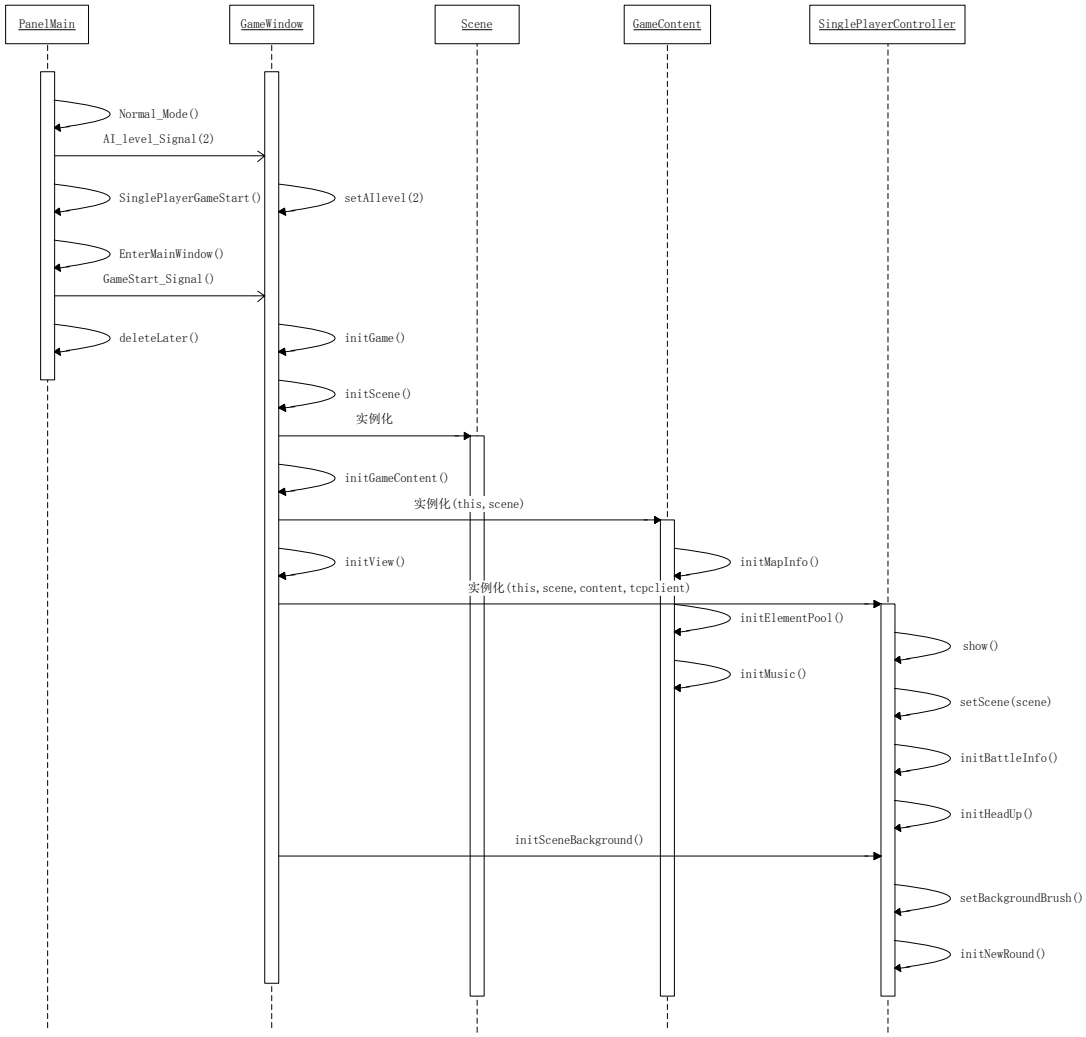


最终结果是 PageMain 被显示出来，玩家可选择想进行的游戏模式或者调节本地设置，又或者结束游戏

B. 实际开始游戏

此阶段从 GameWindow 接受游戏进入主界面选择的模式信息开始到实际游戏循环启动为止

对象生命周期图如下：



具体顺序即为：

GameWindow 接受 PanelMenu 的 AIlevel 的信号并设置 AI 等级

→

初始化 Scene

→

以 Scene 为参数，初始化 GameContent

→

以 Scene 和 GameContent 为参数，单人模式还会以 AIlevel 为参数，联机模式还会以 TcpClient 为参数，初始化 GameViewController

其中：

GameContent 初始化时会初始化 MapInfo 和 ElementPool 以及 Music

GameViewController 初始化时会初始化 PanelBattleInfo 和 PanelHeadUp，且在一定延时后会初始化 NewRound，开启游戏循环

2. 进入游戏界面的页面管理

【PanelMenu.h/PanelMenu.cpp】

PanelMenu 在创建出来时，其上没有创建任何具体按键的子对象，而是以**页面**的形式管理整个进入游戏界面，在各个页面上分别创建子对象，显示页面和隐藏页面时，其上的所有子部件也跟着被显示或隐藏，如此可大大简化页面跳转逻辑并令程序可读性大大增强

PanelMenu.h 中的页面定义如下：

```
//Pages
QFrame *_PageMain;
QFrame *_PageSinglePlayer;
QFrame *_PageMultiPlayer;
QFrame *_PageOnline;
QFrame *_PageOption;
```

而在其后定义了以下槽函数：

```
//slot
void JumpToPageSinglePlayer();
void JumpToPageMultiPlayer();
void JumpToPageOnline();
void JumpToPageOption();
void ReturnToPageMultiplayer();
void ReturnToMainMenu();
```

当接收到跳转某页面的按键点击事件信号后，就会调用上述对应的槽函数，而在每个槽函数中只对页面进行 hide/deleteLater 或 show/new 处理，所有页面上的子部件新建时父对象都是对应页面，这样根据 Qt 的资源管理机制，当父对象的页面 show 时，所有子部件也会分别调用 show 指令，父对象页面 hide 时，所有子部件也会分别调用 hide 指令，父对象页面 deleteLater 时，所有子部件也会按照创建顺序被一一释放

另外，PageMain 由于地位的特殊性，其在 PanelMenu 构造时就被创建，在 PanelMenu 存活期间永远不会被删除，在生成并显示其他页面时，它只会被 hide 而不是 deleteLater，相对的，所有其他的页面创建时总会 new 出新的页面，返回时总会被 deleteLater 回收

3. 切换地图

【PanelMenu.cpp/MapInfo.h/MapInfo.cpp】

地图切换的总流程分成两部分：

- **选择需载入的地图信息**
- **按照所选地图信息正确初始化地图**

下面分别从这两个层面叙述地图切换的流程：

A. 选择需载入的地图信息

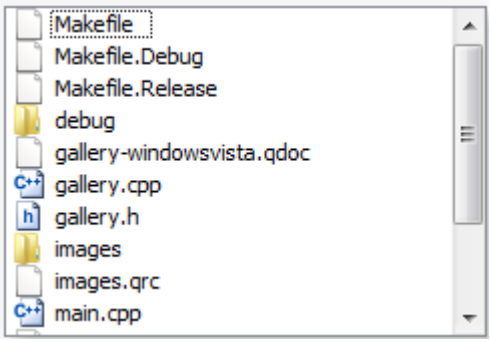
切换地图的功能在 PageOption 中，PageOption 中定义了 QListWidget 的部件 _MapChoose，其中 QListWidget 是 Qt 自带的用于承载 QListWidgetItem 的矩形框列表外形的容器，比较常见的 QListWidget 的使用有下拉列表等。此处设想的选地图

界面是**矩形框中横向分布各个地图的缩略图（单行），每个缩略图下方有此地图的名字，用鼠标点击后即可实现地图的切换。**

为实现此显示效果，QListWidget 的设置如下：

```
_MapChoose->setViewMode(QListView::IconMode);
_MapChoose->setFlow(QListView::LeftToRight);
_MapChoose->setWrapping(false);
_MapChoose->setSelectionMode(QAbstractItemView::SelectionMode::SingleSelection);
_MapChoose->setMovement(QListView::Static);
_MapChoose->setIconSize(QSize(100, 100));
```

第一行第二行为设置模式，默认的 QListWidget 是 QListView::ListMode 模式，如图：



各个 QListWidgetItem 是从上自下排布，图标在左文字在右

而 QListView::IconMode 则是从左自右排布，图标在上文字在下，符合要求，如图：

第三行设置列表不会折叠，以免出现多行显示的效果，故设置“是否折叠”属性为 false

第四行设置 QListWidgetItem 只能单选，以免在设置选择逻辑时出现歧义

第五行设置 QListWidgetItem 不可移动。如果没有这行，用鼠标就可以拖拽 QListWidgetItem，而且拖拽时整个程序会变得非常缓慢，严重影响用户体验。考虑到拖拽改变 QListWidgetItem 位置的无必要性，故此处禁止了拖拽

第六行设置了图标的大小，基本刚刚填充 QListWidget 高度的大部分，较美观

QListWidget 的_MapChoose 设置完毕后，将各个 QListWidgetItem 添加入其中，并对其 currentTextChanged 事件进行捕捉，实时改变当前设置中所用的地图：

```
QObject::connect(_MapChoose, &QListWidget::currentTextChanged, this,
&PanelMenu::SetCurrentMap);
```

此处 SetCurrentMap 中改变了 GameWindow 在初始化 PanelMenu 时传入的 LocalSetting 的引用的参数，LocalSetting 中保存了地图的序号信息：

```
typedef struct ST_LocalSettings
{
    int MapIndex;
    int BackgroundIndex;
    int BackgroundMusicVolume;
    int EffectVolume;
}LocalSettings;
```

MapIndex 存放了当前地图的索引，在后面 GameWindow 初始化 MapInfo 时会以这个索引为参数，选择对应的地图文件进行初始化。

B. 按照所选地图信息正确初始化地图

MapInfo.h 中定义了静态常量的地图设置信息：

```
const static MapFile AllMapFileInfo[] =
{
    MapFile{ "Church yard", ":/IconSrc/Map_Churchyard",
"Resources/MapFile/Churchyard.txt" },
    MapFile{ "Einzbern Castle", ":/IconSrc/Map_EinzbernCastle",
"Resources/MapFile/EinzbernCastle.txt" },
    MapFile{ "Fuyuki Bridge", ":/IconSrc/Map_FuyukiBridge",
"Resources/MapFile/FuyukiBridge.txt" },
    MapFile{ "Liyudou Temple", ":/IconSrc/Map_LiyudouTemple",
"Resources/MapFile/LiyudouTemple.txt" }
};
```

其中，MapFile 为自定义的结构类型，用于存储完整地图的信息：

```
typedef struct STMapFile
{
    QString MapName;
    QString ThumbnailPath;
    QString FileName;
}MapFile;
```

用上一步得到的 Index 去访问 AllMapFileInfo 静态数组，即可获得对应的地图完整信息 MapFile，在 MapInfo 中，调用 initMapInfo 即可读入对应地图 txt 文件并初始化地图，其中地图的 txt 文件也规定了一定的格式以便于连续读入：

以 Church_Yard.txt 地图文件为例，其格式如下：

```
Church_Yard
20 15
1 3 1 5 1 7 1 9 1 11 1 13
20 3 20 5 20 7 20 9 20 11 20 13
11 11 12 11 12  11 12 11 13 13  13 13 11 12 11  12 11 12 11 11
11 11 11 11 11  11 13 11 11 11  11 11 11 13 11  11 11 11 11 11
11 11 12 11 12  11 11 11 11 11  11 11 11 11 11  12 11 12 11 11
11 11 11 11 11  13 11 11 11 11  11 11 11 11 13  11 11 11 11 11
11 11 12 11 12  13 11 11 11 13  13 11 11 11 13  12 11 12 11 11
11 11 11 11 11  11 11 11 11 11  11 11 11 11 11  11 11 11 11 11
11 11 12 11 12  11 11 13 11 11  11 11 13 11 11  12 11 12 11 11
11 11 11 11 11  11 11 13 11 13  13 11 13 11 11  11 11 11 11 11
11 11 12 11 12  11 11 13 11 11  11 11 13 11 11  12 11 12 11 11
11 11 11 11 11  11 11 11 11 11  11 11 11 11 11  11 11 11 11 11
11 11 12 11 12  13 11 11 11 13  13 11 11 11 13  12 11 12 11 11
11 11 11 11 11  13 11 11 11 11  11 11 11 11 13  11 11 11 11 11
11 11 12 11 12  11 11 11 11 11  11 11 11 11 11  12 11 12 11 11
11 11 11 11 11  11 13 11 11 11  11 11 11 13 11  11 11 11 11 11
11 11 12 11 12  11 12 11 13 13  13 13 11 12 11  12 11 12 11 11

//11:墓地地板->唯一可以走的地方
//12:墓碑1
//13:砸坏的墓碑1
//14:砸坏的墓碑2
```

具体的格式定义在MapInfo.h中：

```
typedef struct STMapInfo
{
    QString MapName;
    int MapWidth;
    int MapHeight;
    QVector<QPair<int, int> > BirthPoints;
    QVector<emFloor> MapFloorInfo;
}stMapInfo;
```

第一行为地图名

第二行为地图大小（Width，Height 的顺序）

第三行为左边玩家(黄色)的出生点，按照两两一组的顺序依次是 Saber，Archer，Rider，Assassin，Caster，Berserker（这也是程序内部 6 个棋子的序列顺序）的坐标，坐标统一从 1 开始

第四行为右边玩家（蓝色）的出生点（类似于左边玩家的设定）

第五行开始为地图信息，每个可用 ifstream 读入的单独的 int 为一个地图块的类型信息，比如是普通的可通行陆地还是毁坏的不可通行的陆地，按照地图宽度和高度读取 Width * Height 个地图块信息后即停止读取，故后面可以写一些对地图块信息的注释等

4. 音量调节

【PanelMenu.cpp/GameMainMenu.cpp/GameContent.cpp】

游戏中总共有两个地方可以调节背景音乐和音效的音量：

- **进入游戏界面的 PageOption 中**
- **游戏主界面里 GameMainMenu 中**

下面分别从这两个层面叙述音量调节的流程

A. 进入游戏界面的 PageOption 中

前面也可以看到，LocalSetting 中除了存储了玩家所选的地图的索引信息外，还有 BackgroundMusicVolume 以及 EffectVolume 这两个成员，在 PageOption 中，构建了两个 QSlider（滑动条）成员：

```
//Widget on OptionPage
QSlider *_slBackgroundMusicVolume;
QSlider *_slEffectVolume;
```

在构建完毕后，将它们的 valueChanged 事件信号与对应的 changeVolume 的槽函数相连接：

```
    QObject::connect(_slBackgroundMusicVolume, &QSlider::valueChanged,
this, &PanelMenu::ChangeBackgroundMusicVolume);
    QObject::connect(_slEffectVolume,    &QSlider::valueChanged,    this,
&PanelMenu::ChangeEffectVolume);
```

【注意信号与槽函数都有一个 int 参数，需要传递的也正是这个信号】

而槽函数中对 LocalSetting 的对应变量进行调节，进入游戏后 GameContent 按照 LocalSetting 的值调节音量，如此即可调节进入游戏后的音量

B. 游戏主界面里 GameMainMenu 中

进入游戏之后，需要即时调节音量，基本方法和 A 一致，只是需注意要即时调节音量则需在对应的槽函数中就调用 GameContent 中的改变音量的函数。另外需注意，音量是有默认值的，故在 gamewindow 被初始化时，需**一次性调用**设置默认属性的函数 DefaultSettings()：

```
void GameWindow::DefaultSettings()
{
    _LocalSettings.BackgroundMusicVolume = 50;
    _LocalSettings.EffectVolume = 50;
    _LocalSettings.MapIndex = 1;
    _LocalSettings.BackgroundIndex = 1;
}
```

5. 游戏胜负判断

【SinglePlayerController.cpp/PlayerViewController.cpp】

单机和在线的游戏胜负判断类似，此处只介绍单机模式的胜负判断

单机模式里起主要判断胜负功能的函数为 void SinglePlayerController::CheckVictory()

所用局部变量定义如下：

变量	含义
bool flag_Miku	黄色方所有棋子是否全部死亡
bool flag_Rin	蓝色方所有棋子是否全部死亡
QString text	一方胜利时需要显示的文本

每当可能造成伤害时，调用 CheckVictory 函数即可判断游戏胜负。

流程如下：

A. 将 flag_Miku 和 flag_Rin 赋为 true

B. 若一方的某枚棋子并未死亡（其中 Berserker 棋子并未在复活后再次死亡）便会使得对应 flag 变量变为 false。

C. 根据双方 flag 变量的情况给 text 赋值，若其中一方胜利则会并通过一段动画显示胜利和返回主界面。

6. 回合战斗流程控制

【SinglePlayerController.cpp/PlayerViewController.cpp】

整个战斗流程从微观和宏观角度可分别视为：

- **微观——每个棋子的战斗流程**
- **宏观——先攻队列的循环**

以下从这两个层面分别说明游戏的回合战斗流程控制机制

A. 微观——每个棋子的战斗流程

轮到某个棋子时，其基本回合流程为：一个回合内可以并且只能进行最多一次攻击以及最多一次移动，先后顺序没有要求，EX 技能因棋子的不同而要求和消耗有区别，每回合以“防御/结束回合”作为回合的结束

程序在实现此功能时，无论是单机模式或者是在线模式，都是通过前面提到的 GameState 状态信息结构来控制的，其定义如下：

```
typedef struct _GameState
{
    bool IsHost;
    bool IsChoosingAttackTarget;
    bool IsChoosingMoveTarget;
    bool HasMoved;
    bool HasAttacked;
    bool HasUltimateSkilled;
    QVector<int> AttackSequence;
    int WhileUltimateSkilling;
    int Round;
    int AILevel;
}GameState;
```

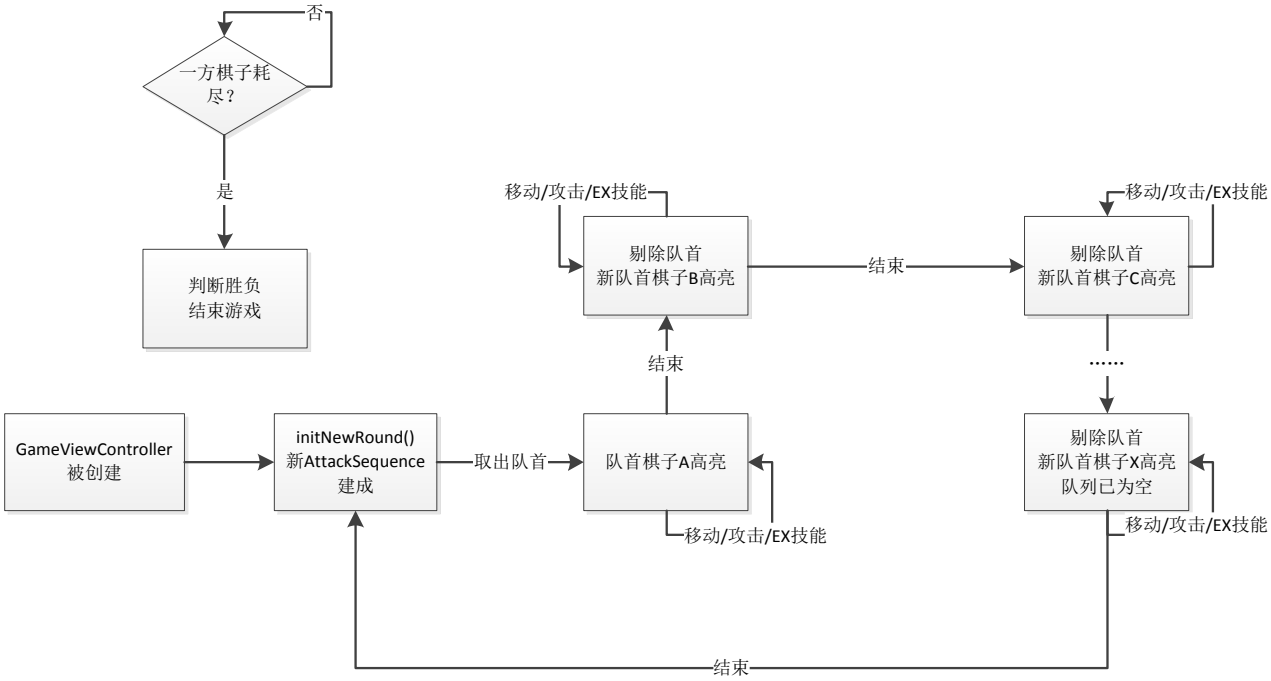
其中，与棋子回合流程控制相关的主要为第二到第六个状态变量，即：IsChoosingAttackTarget（是否正在选取进攻目标），IsChoosingMoveTarget（是否正在选取移动目标），HasMoved（本回合是否已经移动过），HasAttacked（本回合是否已经移动过），HasUltimateSkilled（本回合是否已经释放过 EX 技能）。为实现流程的有序控制，必须针对相应的游戏状态情况开启或禁用部分操作按钮，此部分在 PanelOperate 的介绍中会详细说明，这里仅说明**由这 5 个状态变量控制的所有行动逻辑**：

1) 移动的按钮总是在 HasMoved 为 true 时被禁用
2) 攻击的按钮总是在 HasAttacked 为 true 时被禁用
3) EX 技能的按钮（主动释放型 EX 技能）总是在 HasUltimateSkilled 为 true 时被禁用，另外不同棋子有不同的 EX 技能启动和存在形式
4) 点击移动按键时，令 IsChoosingMoveTarget 为 true，此时调用 notePossibleMoveTarget 函数，标记所有能移动的地图块。只有在 IsChoosingMoveTarget 为 true 的条件下点击高亮地图块才能移动，移动完毕后令 HasMoved 为 true，类似的逻辑还有 Caster 的 EX 技能
5) 点击攻击按键时，令 IsChoosingAttackTarget 为 true，此时调用 notePossibleAttackElement 函数，标记所有在射程内的能攻击的目标。只有在 IsChoosingAttackTarget 为 true 的条件下点击高亮的敌人目标才能攻击，攻击完毕后令 HasAttackd 为 true，类似的逻辑还有 Saber 的 EX 技能
6) 只有当 IsChoosingMoveTarget 和 IsChoosingAttackTarget 均为 false 时，点击当前可操作棋子才能弹出 PanelOperate
7) 每个棋子结束当前轮而令下一个棋子获得活动权时，HasMoved、HasAttacked、HasUltimateSkilled 这 3 个变量都恢复为 false

B. 宏观——先攻队列的循环

每进行新的一轮时，所有棋子需进行一次先攻值检定并进行排序，生成所谓先攻队列。此过程是在 initNewRound 中进行的，initNewRound 的 initQueue 会排空 GameState 中的 AttackSequence（QVector<int>型，int 为对应棋子的序号，范围为 1~6 和-1~-6），并重新装载 AttackSequence。紧接着高亮队首的棋子，令其获得活动权，之后按照前面所说的行动逻辑作出行动后以“防御/结束回合”结束此棋子本轮的活动，这时会自动调用 Next()函数，令 AttackSequence 去除队首元素，高亮新队首元素，接着新的队首元素作出行动。如此不断进行直至队列为空之后，再一次调用 initNewRound 建立新的先

攻队列……这样，游戏循环便能保持持续运转，直至一方棋子耗尽时结束游戏，流程图见下：



7. 棋子移动范围判断

【SinglePlayerController.cpp/PlayerViewController.cpp】

单机和在线模式的移动范围判断类似，此处主要介绍单机模式下的移动范围判断的算法。考察的函数为：void SinglePlayerController::notePossibleMoveTargetMapUnit()，其作用是找到当前先攻队列的队首的元素、也即目前正在进行移动操作的棋子其所有可以到达的地图块，并令其高亮标记（绿色）

其中局部变量定义如下：

变量	含义
Element* element	当前行动的棋子
int temp_x	广搜队列队首元素的 X 坐标
int temp_y	广搜队列队首元素的 Y 坐标
int temp_speed	element 的等效速度
QVector<int> List_X	X 坐标的广搜队列
QVector<int> List_Y	Y 坐标的广搜队列
QVector<int> List_Pace	已行动等效步数的广搜队列
int head	广搜队列队首
int tail	广搜队列队尾
int temp_pace	广搜队列队首元素的已行动等效步数
int deltaX	X 坐标偏移量枚举数组
int deltaY	Y 坐标偏移量枚举数组

bool MultiIgnore[40][40]	广搜队列去重判断数组
MapUnit* temp_MapUnit	根据广搜队列队首元素和坐标偏移量计算出是否为可移动区域的待判断地图块

具体流程如下：

A. 取出处于先攻队列最前位的棋子 element。

B. 以 element 为中心开始**广度优先搜索**，在搜索过程中会将可行的地图块高亮并放入广搜队列末尾。

8. AI

【SinglePlayerController.cpp】

单人模式中，可以与 AI 进行对战，由_State.AILevel 控制选择 AI 的难度。其基本流程为在 1000ms 延迟后调用 notePossibleMoveTargetMapUnit 函数，高亮 AI 可以移动的范围，并在 1500ms 延迟后调用对应难度的 **AI 动作（AI_Action）函数**。

目前设置了 5 个等级的 AI 可供挑战，也就对应了 5 个 AI_Action 函数，这 5 个难度分别为：

- **训练（Tutorial）**
- **简单（Easy）**
- **普通（Normal）**
- **困难（Hard）**
- **疯狂（Lunatic）**

下面分别从这 5 个层面对 AI 的工作流程进行叙述

A. 训练（Tutorial）

维护函数：void SinglePlayerController::**AI_Action_TutorialMode**()

基本 AI 工作流程：

Tutorial AI 会在 1000ms 后调用 Cancel 函数取消行动。故实际为消极 AI，所有轮到 AI 的回合均以跳过结束

B. 简单（Easy）

维护函数：void SinglePlayerController::**AI_Action_EasyMode**()

此模式的基本设计思想是 AI 智能最低，不考虑不同棋子的区分或相克，没有策略，只会攻击在移动射程内的敌人，无法攻击时随机移动。此模式中 AI 不会释放 EX 技能

函数中的局部变量定义如下：

变量	含义
Element* element	当前行动的棋子
QVector<Element*> Alive_Element_Vector	仍然活着的敌人
int num_sig	判断敌对势力为黄还是蓝
int temp_step	element 的速度
MapUnit* target_MapUnit	可能成为移动目标的地图块
int target_Element	可能成为目标的地方棋子在 Alive_Element_Vector 中的位置
MapUnit* temp_MapUnit	用于判断能否成为 target_MapUnit 的暂时地

	图块
int Action_Num	为了延迟设置的参数
int Dis_Delay	为了延迟设置的参数

基本流程如下：

1) 取出处于先攻队列最前位的棋子 element。
2) 判断其阵营并取出敌对阵营所有可被攻击的敌方棋子，存放在 Alive_Element_Vector 中。
3) 将 Alive_Element_Vector 中所有的元素**按照血量**从多到少排序。
4) 搜索所有先前被 notePossibleMoveTargetMapUnit 函数高亮的 MapUnit，寻找一个可以攻击到血量相对较低的敌方棋子的地点，标记该地点和所攻击棋子，存放在 target_MapUnit 和 target_Element 中。
5) 如果无法通过移动攻击到任何一名敌人，就**随机选取一个地点**进行移动。
6) 通过 target_MapUnit 和 target_Element 的值进行函数调用的延迟设定，其中可能调用 Move、notePossibleAttackTargetElement、Attack 和 Cancel 等函数。

C. 普通（Normal）

维护函数：void SinglePlayerController::**AI_Action_NormalMode**()

此模式的基本设计思想是 AI 有一定的智能，进攻时会考虑到不同棋子的不同特性和相克，且每个棋子会持续朝向目标敌对棋子移动和攻击，远程棋子会在靠近目标后保持在距离目标棋子射程范围左右的区域内移动和攻击。此模式中棋子不会释放 EX 技能

函数中的局部变量定义如下：

变量	含义
Element* element	当前行动的棋子
QVector<Element*> Alive_Element_Vector	仍然活着的敌人
int num_sig	判断敌对势力为黄还是蓝
int temp_step	element 的速度
MapUnit* target_MapUnit	可能成为移动目标的地图块
int target_Element	可能成为目标的地方棋子在 Alive_Element_Vector 中的位置
bool whether_Element	判断当前地图块是否能够攻击到 target_Element 所代表的敌方棋子的参数
int min_Dis	element 与 target_Element 所代表的敌方棋子的最小距离
int max_Dis	element 与 target_Element 所代表的敌方棋子的最大距离
MapUnit* temp_MapUnit	用于判断能否成为 target_MapUnit 的暂时地图块
int Action_Num	为了延迟设置的参数
int Dis_Delay	为了延迟设置的参数

基本流程如下：

1) 取出处于先攻队列最前位的棋子 element。
2) 判断其阵营并取出敌对阵营所有可被攻击的敌方棋子，存放在

Alive_Element_Vector 中。

3) 将 Alive_Element_Vector 中所有的元素**按照等效血量从多到少排序**，其中等效血量为根据 element 和敌方棋子的 AB、AC、当前 HP 等数值所折算出的一个高级数据。
4) 搜索所有先前被 notePossibleMoveTargetMapUnit 函数高亮的 MapUnit，寻找一个可以攻击到等效血量相对较低的敌方棋子的地点，标记该地点和所攻击棋子，存放在 target_MapUnit 和 target_Element 中。若存在相似的许多可以攻击到同一枚地方棋子的地图块，则根据 element 本身的特性决定。若 element 的攻击范围为 1，那么尽可能逼近敌人。若 element 的攻击范围大于 1，那么在能攻击到敌人的情况下尽可能保持与敌人的距离等于其攻击范围。
5) 如果无法通过移动攻击到任何一名敌人，就寻找等效血量最少的棋子并向其移动。
6) 通过 target_MapUnit 和 target_Element 的值进行函数调用的延迟设定，其中可能调用 Move、notePossibleAttackTargetElement、Attack 和 Cancel 等函数。

D. 困难（Hard）

维护函数：void SinglePlayerController::**AI_Action_HardMode()**
void SinglePlayerController::**AI_Action_HardMode_UltimateSkill_Saber()**
void SinglePlayerController::**AI_Action_HardMode_UltimateSkill_Caster()**

此模式的基本设计思想是：AI 智能较高，一般的战斗策略与 normal 类似，即：进攻时会考虑到不同棋子的不同特性和相克，且每个棋子会持续朝向目标敌对棋子移动和攻击，远程棋子会在靠近目标后保持在距离目标棋子射程范围左右的区域内移动和攻击。另外此模式中棋子会在 EX 技能条件满足时释放 EX 技能

AI_Action_HardMode()中局部变量定义如下：

变量	含义
Element* element	当前行动的棋子
QVector<Element*> Alive_Element_Vector	仍然活着的敌人
int num_sig	判断敌对势力为黄还是蓝
int temp_step	element 的速度
MapUnit* target_MapUnit	可能成为移动目标的地图块
int target_Element	可能成为目标的地方棋子在 Alive_Element_Vector 中的位置
bool whether_Element	判断当前地图块是否能够攻击到 target_Element 所代表的敌方棋子的参数
int min_Dis	element 与 target_Element 所代表的敌方棋子的最小距离
int max_Dis	element 与 target_Element 所代表的敌方棋子的最大距离
MapUnit* temp_MapUnit	用于判断能否成为 target_MapUnit 的暂时地图块
int Action_Num	为了延迟设置的参数
int Dis_Delay	为了延迟设置的参数

基本流程如下：

1) 取出处于先攻队列最前位的棋子 element。
2) 判断其阵营并取出敌对阵营所有可被攻击的敌方棋子，存放在 Alive_Element_Vector 中。
3) 将 Alive_Element_Vector 中所有的元素按照等效血量从多到少排序，其中等效血量为根据 element 和敌方棋子的 AB、AC、当前 HP 等数值所折算出的一个高级数据。
4) 判断 element 是否有能力释放 EX 技能，若能则进入相应判断部分，判断是否使用 EX 技能及 EX 技能释放对象，包括 AI_Action_HardMode_UltimateSkill_Saber 和 AI_Action_HardMode_UltimateSkill_Caster 两个需要指定目标的函数，此处通过巧妙设置延迟使得流程看起来非常顺畅。
5) 搜索所有先前被 notePossibleMoveTargetMapUnit 函数高亮的 MapUnit，寻找一个可以攻击到等效血量相对较低的敌方棋子的地点，标记该地点和所攻击棋子，存放在 target_MapUnit 和 target_Element 中。若存在相似的许多可以攻击到同一枚地方棋子的地图块，则根据 element 本身的特性决定。若 element 的攻击范围为 1，那么尽可能逼近敌人。若 element 的攻击范围大于 1，那么在能攻击到敌人的情况下尽可能保持与敌人的距离等于其攻击范围。
6) 如果无法通过移动攻击到任何一名敌人，就寻找等效血量最少的棋子并向其移动。通过 target_MapUnit 和 target_Element 的值进行函数调用的延迟设定，其中可能调用 Move、notePossibleAttackTargetElement、Attack 和 Cancel 等函数。

E. 疯狂（Lunatic）

维护函数：void SinglePlayerController::**AI_Action_LunaticMode()**
void SinglePlayerController::**AI_Action_LunaticMode_UltimateSkill_Saber()**
void SinglePlayerController::**AI_Action_LunaticMode_UltimateSkill_Caster()**

此模式的设计思想为：AI 智能较高，战斗策略除了与 hard/normal 类似的部分外，还加入了先攻击后撤离的机制：每次轮到 AI 棋子时，都会做一次先攻击后移动还是先移动后攻击的判定，如此即能大大增加 AI 策略性。另外此模式中棋子会在 EX 技能条件满足时释放 EX 技能

AI_Action_LunaticMode()中局部变量定义如下：

变量	含义
Element* element	当前行动的棋子
QVector<Element*> Alive_Element_Vector	仍然活着的敌人
int num_sig	判断敌对势力为黄还是蓝
int temp_step	element 的速度
MapUnit* target_MapUnit	可能成为移动目标的地图块
int target_Element	可能成为目标的地方棋子在 Alive_Element_Vector 中的位置
bool whether_Element	判断当前地图块是否能够攻击到 target_Element 所代表的敌方棋子的参数
int min_Dis	element 与 target_Element 所代表的敌方棋子的最小距离
int max_Dis	element 与 target_Element 所代表的敌方棋子的最大距离

MapUnit* temp_MapUnit	用于判断能否成为 target_MapUnit 的暂时地图块
int Action_Num	为了延迟设置的参数
int Dis_Delay	为了延迟设置的参数
bool UltimateSkill_and_Move	为 Rider 而设置，判断 Ex 技能之后是否进行移动
bool move_after_attack	判定移动和攻击的先后顺序而设置的变量
bool is_Archer	判定是否是 Archer 棋子，如果是，后续需进行 Ex 技能的判定

基本流程如下：

1) 取出先攻队列最前端的棋子 element。
2) 判断其阵营并取出敌对阵营所有可被攻击的敌方棋子，存放在 Alive_Element_Vector 中。
3) 将 Alive_Element_Vector 中所有的元素按照等效血量从多到少排序，其中等效血量为根据 element 和敌方棋子的 AB，AC，当前 HP，在先攻队列中的排序，以及攻击范围等数值所折算出的一个高级数据。
4) 判断 element 是否有能力释放 EX 技能，若能则进入判断部分，判断是否使用 EX 技能及 EX 技能释放对象，包括 AI_Action_LunaticMode_UltimateSkill_Saber 和 AI_Action_HardMode_UltimateSkill_Caster 两个需要指定目标的函数，此处通过巧妙设置延迟使得流程看起来非常顺畅。
5) 进行先攻击后移动还是先移动后攻击的判定，然后设置 move_after_attack 参数进行后续操作。这也是本个 AI 与 HardMode 最大的不同之处。而且，在此处需进行是否 Archer 的判定，如果是 Archer 原始形态，如果距离 1 以内有敌人，需进行 Ex 技能之后再攻击和移动，如果是 Ex 技能状态下的 Archer，需判断攻击范围 1 以外 10 以内是否有敌人，如果有，需恢复原始状态之后进行攻击和移动。
6) 首先判断不移动是否有攻击对象，如果有的话将 move_after_attack 设置为 true，然后选择攻击对象。
7) 如果 move_after_attack 为 true，就执行攻击动作，完成攻击动作之后，选择离攻击对象最远的地点进行躲避，选择完移动目标之后，执行移动动画，结束此次操作。
8) 如果 move_after_attack 为 false，就首先选择移动目标，执行移动动作，然后判断是否有敌人在攻击范围内，如果有，选择等效血量最少的敌人进行攻击，执行攻击动作，如果没有敌人在攻击范围内，向等效血量最少的敌人的方向移动。移动动作和攻击动作通过 target_MapUnit 和 target_Element 进行选择对象，若二者不为空，就执行相应的动作。
9) 此外，这里还调用了 Move，notePossibleAttackTargerElement 以及 cancel 函数执行相应动画操作。

9. 游戏战斗信息面板构建与更新

【PanelBattleInfo.h/PanelBattleInfo.cpp/ElementBattleInfo.h/ElementBattleInfo.cpp】

点击 PanelHeadUp 可弹出游戏战斗信息面板，可显示当前先攻队列信息以及每个棋子的 HP、EX 技能的状态。

由于设想中 PanelHeadUp 上的中心按钮处于 PanelHeadUp 和 PanelBattleInfo 之上、刚好各覆盖一半的效果，故需令 PanelBattleInfo 处于 PanelHeadUp 之下。如下图所示：

程序中是通过先初始化 PanelBattleInfo 后初始化 PanelHeadUp 的方式实现的，且 PanelBattleInfo 在创建后不会被 deleteLater 掉，而只会 hide（隐藏）或 show（显示），换句话说，此面板在创建开始就一直存在着。故程序的重点在于如何随**战斗进程不断刷新其上的信息**。由于需要知道 GameState 中的先攻队列信息，且由于需要显示每个棋子的生命值和 EX 技能状态，而这两个信息在 Element 的_character 中，故初始化 PanelBattleInfo 时必须输入 GameState 的参数（指针或引用均可，程序中用的是指针），并能访问 Element 的信息，即需要#include “Element.h”

PanelBattleInfo 的基本构成为（如下图）：

第一行为当前游戏的回合数，数据来自 GameState，每次 initNewRound 时需要刷新此信息

从后面第二行开始每一行都是双方的某一个棋子的生命值和 EX 技能状态的信息，从上至下为先攻队列的顺序，金黄色边框表明当前被授予活动权的棋子。当棋子死亡（包括 Berserker 假死）后，当前轮的 PanelBattleInfo 中显示头像变为灰色，HP 固定为 0，若此棋子在活动棋子之后，则需要在轮到此棋子活动时自动跳过此棋子。并在下一轮开始时 initNewRound 时不加入此棋子

为实现上述功能，将每一栏即每个棋子的战斗信息定义为了名为 ElementBattleInfo 的对象，换句话说，PanelBattleInfo 上承载了各个棋子对应的 ElementBattleInfo，每一个 ElementBattleInfo 包含棋子的序号、HP 值、EX 技能状态的信息，并有一个用于刷新当前信息的接口：

```
void RefreshBattleInfo();
```

而 PanelBattleInfo 有以下两个主要接口：

```
void RefreshNewRoundInfo();
void RefreshAllBattleInfo();
```

前者**只在新一轮开始的时候被调用**，类似于 AttackSequence 的行为，排空 PanelBattleInfo 中所有的 ElementBattleInfo，并按照 AttackSequence 的顺序重新按顺序装填 ElementBattleInfo，之后会自动调用 RefreshAllBattleInfo

后者**在所有改变游戏中棋子生命值和 EX 状态的地方均会被调用**，此接口调用时，会调用所有其上的 ElementBattleInfo 的 RefreshBattleInfo 接口，刷新所有棋子的信息，虽然原本只用刷新某一个棋子的信息即可，这样做效率上会有损失，但本身对最多 12 个棋子的 ElementBattleInfo 刷新信息几乎不耗费时间，而且统一刷新省却了判断当前是哪个 ElementBattleInfo 的麻烦，而且使得**底层的 ElementBattleInfo 对上层的 GameViewController 不可见**，在棋子频繁的状态信息变换中，刷新任何棋子当前信息只需不断调用此接口即可

10. 游戏操作面板的创建

【GameViewController.cpp/PanelOperate.h/PanelOperate.cpp】

前面也已经提到过，游戏操作面板是玩家进行对战的唯一介入方式，关于游戏操作面板的说明主要分为两个部分：

- **鼠标事件呼出 PanelOperate**
- **PanelOperate 的构造过程**

以下分别从这两个方面进行说明

A. 鼠标事件呼出 PanelOperate

由于单机和在线模式此处的逻辑相同，故只以 SinglePlayerController 即单机模式为主进行说明。

SinglePlayerController 中重载的 mousePressEvent(QMouseEvent *e)中的关键条件判断如下：

```
if (this->isInteractive()){
    if (e->button() == Qt::LeftButton){
        if (!_State.IsChoosingMoveTarget && !_State.IsChoosingAttackTarget &&
    (_State.WhileUltimateSkilling == 0)){
            if (itemSelected != NULL && index != 0 && (((index > 0 && _State.IsHost)
        || (index < 0 && !_State.IsHost) || (_State.AILevel == -1)))&&
        CheckIsNextAttacker(index)) {
                //创建PanelOperate
            }
        }
    }
}
```

即满足以下四个条件：

1) 当View为可用时（开启主菜单时会禁止View的所有鼠标和键盘事件）
2) 当点击鼠标左键时（鼠标右键是撤销选择状态）
3) 当前不处于正在选择移动目标、正在选择攻击目标、正在选择EX技能目标的状态（即最一般的状态）

4) 当前选择的棋子不为空、当前选择的棋子为自己的棋子或者为离线模式（双方棋子都能控制）、当前棋子为先攻队列的队首

的时候，才会创建PanelOperate

B. PanelOperate 的构造过程

Operate 面板的构建分为两个重要部分，一是面板的生成，二是其上按钮的生成。

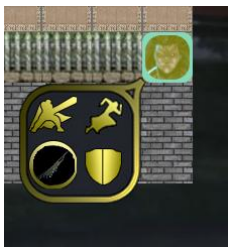
1) 对于面板的生成，出于完善游戏体验的目的，设计了根据当前鼠标位置确定弹出 PanelOperate 形态的设定，保证不会出现弹出的面板在 View 边框外的情况。如下图：

具体实现即为四组判断条件，根据当前鼠标指针的位置而分别创建不同位置处不同形态（实际是一张图的不同旋转角度）的 PanelOperate

当鼠标 x 坐标位于 0~（View 宽度-面板宽度）**以内**，且 y 坐标也位于 0~（View 高度-面板高度）**以内**时，弹出的面板在右下角，此为最常见的情形

当鼠标 x 坐标位于 0~（View 宽度-面板宽度）**以内**，而 y 坐标位于 0~（View 高度-面板高度）**以外**时，弹出的面板在右上角

当鼠标 x 坐标位于 0~（View 宽度-面板宽度）**以外**，而 y 坐标位于 0~（View 高度-面板高度）**以内**时，弹出的面板在左下角



当鼠标 x 坐标位于 0~（View 宽度-面板宽度）**以外**，且 y 坐标也位于 0~（View 高度-面板高度）**以外**时，弹出的面板在左上角

2) 对于按钮的生成，其严格考察了了当前的状态信息（GameStart中），并以此判断某操作是否应该被开放或被禁止。值得注意的是，经过之前代码的重构后，为了使程序逻辑更加清晰，**绝大部分按钮使能和禁止的逻辑都集中在了面板PanelOperate按钮的创建和设置过程中**，而不是在对应的按键对应的槽函数（或者说回调函数）中，换句话说，**凡是被使能了的操作按钮点击后就一定能发挥其作用**，如此也能简化其他函数的代码，避免逻辑的重复。具体逻辑如下：

◆ **对于Saber和Caster，其EX技能消耗为整个回合：**

```
if (state.HasMoved || state.HasUltimateSkilled)
    _btMove->setDisabled(true);
if (state.HasAttacked || state.HasUltimateSkilled)
    _btAttack->setDisabled(true);
if (state.HasMoved || state.HasAttacked || state.HasUltimateSkilled
|| !EX_state)
    _btUltimateSkill->setDisabled(true);
```

此逻辑即为所要求的：

【移动过】或【EX技能释放过】则禁止移动操作按钮，

【攻击过】或【EX技能释放过】则禁止攻击操作按钮

【移动过】或【攻击过】或【EX技能释放过】或【EX技能可使用次数为0】则禁止EX技能操作按钮

◆ **对于Archer，其EX技能一回合一次且必须在攻击和移动之前释放：**

```
if (state.HasMoved)
    _btMove->setDisabled(true);
if (state.HasAttacked)
    _btAttack->setDisabled(true);
if (state.HasMoved || state.HasAttacked || state.HasUltimateSkilled)
    _btUltimateSkill->setDisabled(true);
```

此逻辑即为所要求的：

【移动过】则禁止移动操作按钮，

【攻击过】则禁止攻击操作按钮

【移动过】或【攻击过】或【EX技能释放过】则禁止EX技能操作按钮

◆ **对于Rider，其EX技能消耗一个攻击动作**

```
if (state.HasMoved)
    _btMove->setDisabled(true);
if (state.HasAttacked || state.HasUltimateSkilled)
    _btAttack->setDisabled(true);
if (state.HasAttacked || state.HasUltimateSkilled || !EX_state)
    _btUltimateSkill->setDisabled(true);
```

此逻辑即为所要求的：

【移动过】则禁止移动操作按钮，

【攻击过】或者【EX技能释放过】则禁止攻击操作按钮

【攻击过】或【EX技能释放过】或者【EX技能可使用次数为0】则禁止EX技能操作按钮

◆ **对于Assassin，其EX技能永久存在**

```
if (state.HasMoved)
    _btMove->setDisabled(true);
if (state.HasAttacked)
    _btAttack->setDisabled(true);
```

此逻辑即为所要求的：

【移动过】则禁止移动操作按钮，

【攻击过】则禁止攻击操作按钮

EX技能永久有效

◆ **对于Berserkeer，其EX技能在使用过之前永久有效，使用过之后永久无效**

```
if (state.HasMoved)
    _btMove->setDisabled(true);
if (state.HasAttacked)
    _btAttack->setDisabled(true);
if (!EX_state)
    _btUltimateSkill->setDisabled(true);
```

此逻辑即为所要求的：

【移动过】则禁止移动操作按钮，

【攻击过】则禁止攻击操作按钮

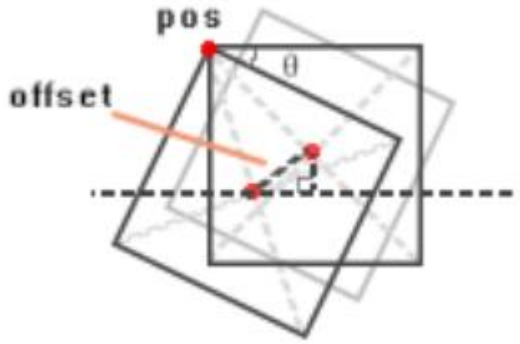
【EX技能释放次数为0】则禁止EX技能操作按钮

11. QGraphicsItem 旋转后偏移量补偿

【SinglePlayerController.cpp/PlayerViewController.cpp】

此算法只在投射物动画中被使用，用于弥补 Qt 本身旋转操作带来的副作用。

Qt 中，QGraphicsItem 可以设置其旋转操作的属性值，即 setRotation，但其中不足为：此旋转的中心为默认的 QGraphicsItem 的**左上角**，也即 Item“所在”的点，如此旋转后，实际显示的图片会与正常图片产生一个偏移量，如下图：



根据三角函数的知识可以计算出，图中 offset_X 和 offset_Y 分别为：

offsetX = cos(PI / 4 - rad / 2)*cos((PI - rad) / 2)*sqrt(2)*CELL_SIZE;

offsetY = -1 * sin(PI / 4 - rad / 2)*cos((PI - rad) / 2)*sqrt(2)*CELL_SIZE;

其中 PI 为π，rad 为 θ / 180 * π，CELL_SIZE 为 QGraphicsItem 的图片边长

令实际的旋转后的 QGraphicsItem 的 pos 其 x 和 y 分别加上这两个偏移量则能按照要求正确显示位置（尤其用于动画）

12. 键盘输入视点控制

【SinglePlayerController.cpp/PlayerViewController.cpp】

由于 QGraphicsView 默认自带的滚动效果很差，鼠标滚轮滚动为台阶式而不是连续式，而且横向滚轮需加 Alt 键控制操作繁琐，另外自带的滚动条比较难看且拖动行为不友好，故在本游戏中完全重新设计视窗或者说游戏镜头的控制，其基本目标为：**连续移动镜头视点，操作简单**

为实现上述目标，总共设计了两种观察方式：

- **键盘 WSAD 连续控制镜头**
- **鼠标左键拖拽连续控制镜头**

其中第二种是对第一种的补充，用于满足**只使用鼠标完成游戏**的玩家的需求而设计的。以下分别从这两个方面说明视点控制的机理

A. 键盘 WSAD 连续控制镜头

实现此操作模式的基本思路为：重载 View 的 KeyPressEvent 和 KeyReleaseEvent，当 KeyPressEvent 事件出现时，根据按键的类型（四种，WSAD）开启对应的（四种，上下左右）循环计数的计时器，它们的计数间隔相当小，而且每一次计时时间到都会调用对应的使镜头视点朝某一方向移动 dx 微小距离的函数，而在 KeyReleaseEvent 触发时，关闭所有的计时器，由于计时间隔非常小，每一次镜头的移动又相当小，故玩家感受到的是比较顺畅的连续动画效果。综上，如此即可实现根据按键时间的长短连续移动镜头不同长度的距离的效果。此操作有一个弊病是不可以同时按下两个方向键，若同时按下两个方向键程序的处理是忽略后按下的按键。

B. 鼠标左键拖拽连续控制镜头

设置当前 View 为可拖拽模式：

```
this->setDragMode(QGraphicsView::ScrollHandDrag);
```

如此即可开启此拖拽控制方式。但由于开启后，鼠标在 View 上移动时始终为拖拽手型，较为难看，故在 main 函数中设置了**应用程序的全局指针图片**，其会覆盖所有指针，算是一定程度上弥补了自带的拖拽模式的缺陷：

```
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    GameWindow w;
    w.show();
    QCursor cursor(QPixmap(":/IconSrc/cursor"), 1, 1);
    a.setOverrideCursor(cursor);
    return a.exec();
}
```

13. 移动动画效果

【SinglePlayerController.cpp/PlayerViewController.cpp】

单机和在线模式的移动动画相似，此处主要介绍单机模式下的移动动画的构建流程。

维护函数：void SinglePlayerController::**Move(MapUnit* target)**

函数中局部变量定义如下：

变量	含义
QVector<QVector<MapUnit*>> List_MapUnit	行动路径的广搜队列，List_MapUnit[i]代表第 i 个元素从初始位置到现在的路径，存在一个 QVector 里
QVector<MapUnit*> tempMapUnitRoute	准备存到 List_MapUnit 中的临时变量
int nowX	target 的 X 坐标
int nowY	target 的 Y 坐标
QSequentialAnimationGroup *AnimeGroup	移动动画
QPropertyAnimation *MoveAnime	从一个地图块移动到路径 QVector 中的下一个地图块的动画
其余变量	参照 notePossibleMoveTargetMapUnit 函数中的变量含义

基本流程如下：

A. 判断 target 是否被高亮
B. 进行与 notePossibleMoveTargetMapUnit 相似的广搜，不同的是此次多了行动路径的广搜队列，储存从起始地图块到当前地图快的最短路径
C. 在广搜队列中寻找 target，并读取其路径用于制作动画

14. 攻击动画效果

【SinglePlayerController.cpp/PlayerViewController.cpp】

单机和在线模式的攻击动画相似，此处主要介绍单机模式下的攻击动画的构建流程。

维护函数：void SinglePlayerController::**Attack(Element* enemy)**

函数中局部变量定义如下：

变量	含义
QParallelAnimationGroup* ParallelPartofAttackAnimation	攻击动画的平行部分
QSequentialAnimationGroup* AllAttackAnimation	总攻击动画
QParallelAnimationGroup* AttackTowardEnemyGroup	攻击方棋子的动画
QPropertyAnimation *AttackForward	攻击范围为 1 的棋子向敌方棋子移动，攻击范围大于 1 的棋子拥有自己的特殊弹道效果
QPropertyAnimation* AttackTowardEnemy_Back	攻击范围为 1 的棋子从敌方棋子的位置移回
QSequentialAnimationGroup* BeAttacked	被攻击方棋子的动画
QPropertyAnimation *DeathAnime	若被攻击方棋子死亡则需要播放的动画

基本流程如下：

A. 设置 AttackForward 为向对方棋子移动，加入 AttackTowardEnemyGroup。
B. 若棋子为远程则重新设置 AttackTowardEnemyGroup。

C. 设置 AttackTowardEnemy_Back 并加入 ParallelPartofAttackAnimation，若棋子为远程则忽略该动画。
D. 设置 BeAttacked 为随机抖动动画并加入 ParallelPartofAttackAnimation。
E. 将 AttackTowardEnemyGroup 和 ParallelPartofAttackAnimation 按照先后顺序加入 AllAttackAnimation，开始动画。
F. 在 AllAttackAnimation 结束后若敌方棋子死亡则设置 DeathAnime 并播放

15. 血条动画效果

【PanelHeadUp.cpp】

当棋子受到攻击损失生命值时，HeadUp 上的人物头像下的血条将会按照百分比有相应**减短**和**变色**，而且此过程是动画化的

维护函数：void PanelHeadUp::**set_health(int _Index, int HP, int max_HP, bool Death, bool Death_to_Alive)**

此函数的局部变量定义如下：

变量	含义
int idx_temp	血条编号
double deltaX	血条所需像素变化量
double delta	真实血量百分比
int new_Green	真实血量 RGB 的 G 值
int new_Red	真实血量 RGB 的 R 值
double old_delta	显示血量百分比
int old_Green	显示血量 RGB 的 G 值
int old_Red	显示血量 RGB 的 R 值
QPropertyAnimation* MoveAnime	血条长度变化
double now_delta	当前血量百分比
int now_Green	当前血量 RGB 的 G 值
int now_Red	当前血量 RGB 的 R 值

基本流程如下：

A. 获取血条编号。
B. 获取 deltaX，计算动画所需时间。
C. 获取旧 RGB 值和新 RGB 值。
D. 用 singleshot 做一个 20 帧的颜色刷新，计算出当前血条长度并据此更新 RGB 值。
E. 如果角色死亡，在血条动画的最后需要将头像变灰并将血条 Item 隐藏。
F. 如果 Berserker 复活，在血条动画之前将头像变红并显示血条 Item。

16. Saber EX 技能及其动画效果

【SinglePlayerController.cpp/PlayerViewController.cpp】

Saber 的 EX 技能是 Excalibur，中文译名为“誓约胜利之剑”，在 fate 系列中的设定是“藉由将所有者的魔力变换成光，再从往下挥的剑的前端如光束一般地放出来破坏万物”，由于 Qt 本身的限制，不能像游戏引擎那样使用特效和粒子效果，而又为了增强游戏性改善玩家的游戏体验，本游戏中对 Excalibur 的效果做了最大程度的还原，具体动

画分为以下几个部分：

- 巨大剑气的淡入动画
- 巨大剑气旋转角度对准敌人后快速飞向敌人的移动动画
- 巨大剑气在贴近敌人之后慢速移动摧毁敌人的移动动画
- 巨大剑气在快速移动阶段朝旋转角度为中心的 90 度范围的扇形区域内不断发射割裂空气引起的波动的动画
- 波动淡出的动画
- 巨大剑气在慢速移动阶段朝旋转角度为中心的 180 度范围的扇形区域内不断发射大量撞击摩擦引起的火花的动画
- 火花淡出的动画

主要维护函数：

```
void UltimateSkill_Saber(Element* enemy);
```

主函数，包括剑气图像的创建，快速阶段的音效，快速/慢速飞行动画

```
void UltimateSkill_Saber_Effect_Fast();
```

辅助函数，完成每一次快速飞行阶段计时时间到向周围发出波动的效果

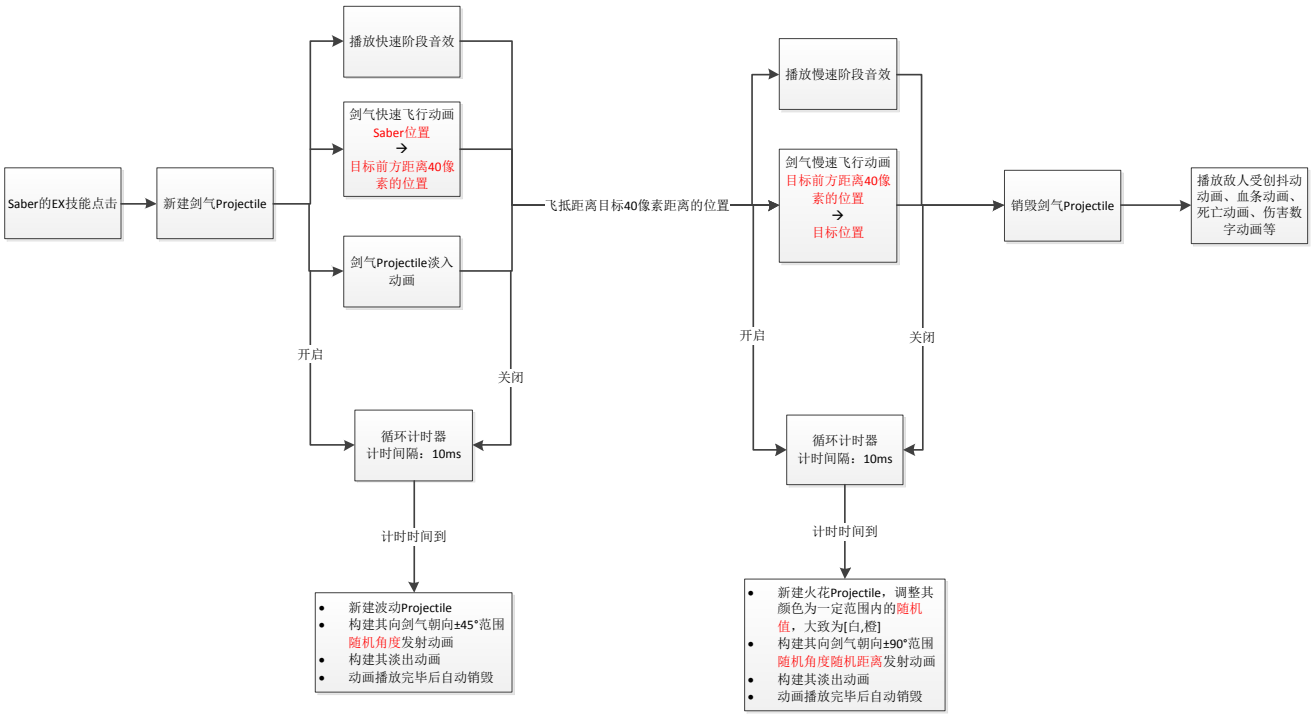
```
void UltimateSkill_Saber_Effect_Slow();
```

辅助函数，完成每一次慢速飞行阶段计时时间到向周围发出火花的效果

```
void UltimateSkill_Saber_Sound_Effect();
```

辅助函数，慢速阶段的音效

流程图如下：



17. Archer EX 技能

【SinglePlayerController.cpp/PlayerViewController.cpp】

Archer 的 EX 技能是切换近远程，以满足不同战局的要求。此技能的触发逻辑在前面 PanelOperate 中已经说明，即所有动作之前，且一回合最多一次。

维护函数：

void Remote_Archer();

void Melee_Archer();

两个函数对 Archer 的 Character 属性进行了不同发展方向（近战 or 远程）的修改

18. Rider EX 技能及其动画效果

【SinglePlayerController.cpp/PlayerViewController.cpp】

Rider 的 EX 技能效果是 Rider 周围一定范围内的所有其他敌我棋子先全部变成灰色并播放受击抖动动画，然后进行先攻判定，判定小于 15 的棋子播放死亡的渐隐动画。其在单机模式和在线模式两种情况下虽调用情况不同但动画效果一致，故这里只说明单机模式下 Rider 的 EX 技能动画效果

维护函数：void SinglePlayerController::**UltimateSkill_Rider()**

此函数局部变量定义如下：

变量	定义
Element* element	当前行动的棋子
QVector<Element*> element_vector	受到 EX 技能影响的所有妻子
QSequentialAnimationGroup* BeAttacked	被攻击动画
QPropertyAnimation *DeathAnime	死亡动画

基本流程如下：

A. 判定 Rider 可以使用 EX 技能
B. 找到所有在 EX 技能范围内的棋子（不包括本身）
C. 将所有棋子执行 tomb_note，调整颜色
D. 制作被攻击的抖动动画 BeAttacked
E. 进行先攻判定，判定小于 15 的棋子将执行死亡动画并隐藏
F. 所有剩余棋子执行 alive_note，调整颜色

19. Assassin EX 技能动画效果

【SinglePlayerController.cpp/PlayerViewController.cpp】

Assassin 的 EX 技能为“心眼”，即被普通攻击时有一半几率使此次攻击自己的 AC 值提高到 30，此时，除了 Caster 以外（Caster 的 AB 为 100，故即使 AC 变为 30 只要没有触发 1/20 的故有 miss 也是必中），所有其他棋子的攻击均为必落空。游戏中为了增强用户体验，给 Assassin 成功触发 EX 技能躲过 Caster 以外的所有人攻击加了利用快速分身闪避攻击的动画效果，用此动画代替正常的受击抖动动画。具体动画效果由以下几个部分组成：

- Assassin 自己瞬间消失，并在一定延迟后快速回复不透明状态的动画
- 在与攻击者到 Assassin 的连线垂直的方向上在 Assassin 两边分别创造一个分身，并构建两个分身向 Assassin 的位置快速移动的动画
- 两个分身由全透明到半透明的转变动画

主要维护函数：

```
void Attack(Element* enemy);
```

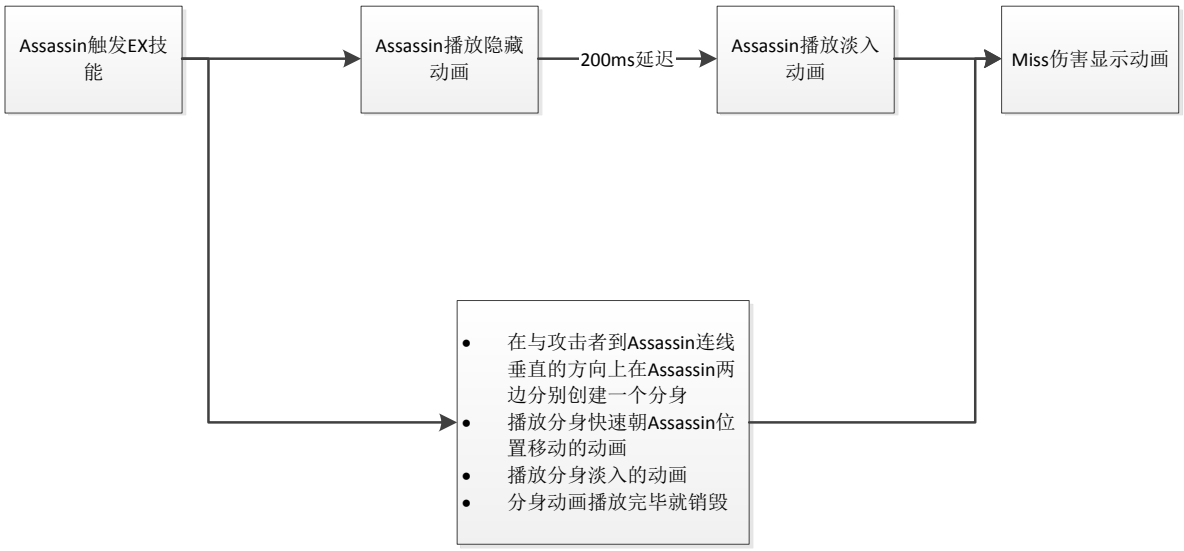
在受击时判断当前 AC 是否为 30 而且攻击者不为 Caster，即触发了 Assassin 的 EX 技

能且攻击者不是 Caster，若是则将原本的受击抖动动画替换为新的闪避动画

```
void UltimateSkill_Assassin_Sound_Effect();
```

闪避时的音效

流程图如下：



20. Caster EX 技能及其动画效果

【SinglePlayerController.cpp/PlayerViewController.cpp】

Caster 的 EX 技能效果为创建一个任意门并瞬间移动至 15 格以内所有可以移动的目标地图块，其中瞬移的动画为“淡入→淡出”。其在单机模式和在线模式两种情况下虽调用情况不同但动画效果一致，故这里只说明单机模式下 Caster 的 EX 技能动画效果

维护函数：void SinglePlayerController::**noteUltimateSkillTarget_Caster()**
void SinglePlayerController::**UltimateSkill_Caster(MapUnit* target)**

函数局部变量定义如下：

变量	定义
QSequentialAnimationGroup *AnimeGroup	EX 技能动画组
QPropertyAnimation *OpacityAnime	消失动画
QPropertyAnimation *MoveAnime	移动动画
QPropertyAnimation *OpacityAnime_2	出现动画

基本流程如下：

A. 执行 noteUltimateSkillTarget_Caster 函数，其仅将 notePossibleMoveTargetMapUnit 的步数调整为了 15，其余基本相同。
B. 若选定一个高亮地图块则执行 UltimateSkill_Caster(MapUnit* target)函数。
C. UltimateSkill_Caster(MapUnit* target)函数分别定义了从当前位置消失、从当前位置移动到指定位置、在指定位置出现三个动画，并依次加入 AnimeGroup，其中移动动画仅为了调整 Caster 棋子的位置而存在，其本身不可见。
D. 执行 AnimeGroup 并取消高亮地图块的高亮。

21. Berserker EX 技能

【SinglePlayerController.cpp/PlayerViewController.cpp/Element.cpp……】

Berserker 的 EX 技能分散在不同的函数中，并没有一个特定的函数描述该技能。基本流程如下：

A. 在 Berserker 死亡的时候向 Element 类中传入 Round_Death 死亡轮。

B. 当 Round 当前轮与 Round_Death 死亡轮相差正好为 2 的时候通过 Death 函数的特判复活，在先攻队列检定的时候通过特判调整血条长度

C. 在复活轮受到致命伤害时通过 Death 函数的特判将血量回复到 15 点

22. 网络通信

网络通信部分主要分为以下几个部分：

- **数据发送和接收**
- **游戏之前的沟通**
- **游戏进行过程**
- **请求游戏公告**

以下将从这四个方面叙述网络通信的运作机制

A. 数据发送和接收

服务器和客户端发送的数据格式为 json,构建 json 并发送的基本程序为：

```
QVariantMap answer;
answer.insert("state", "当前状态");
QString json = QJsonDocument::fromVariant(answer).toJson();
tcpSocket->write(json.toLocal8Bit(),json.toLocal8Bit().length();
```

服务器接收 json 的基本程序如下：

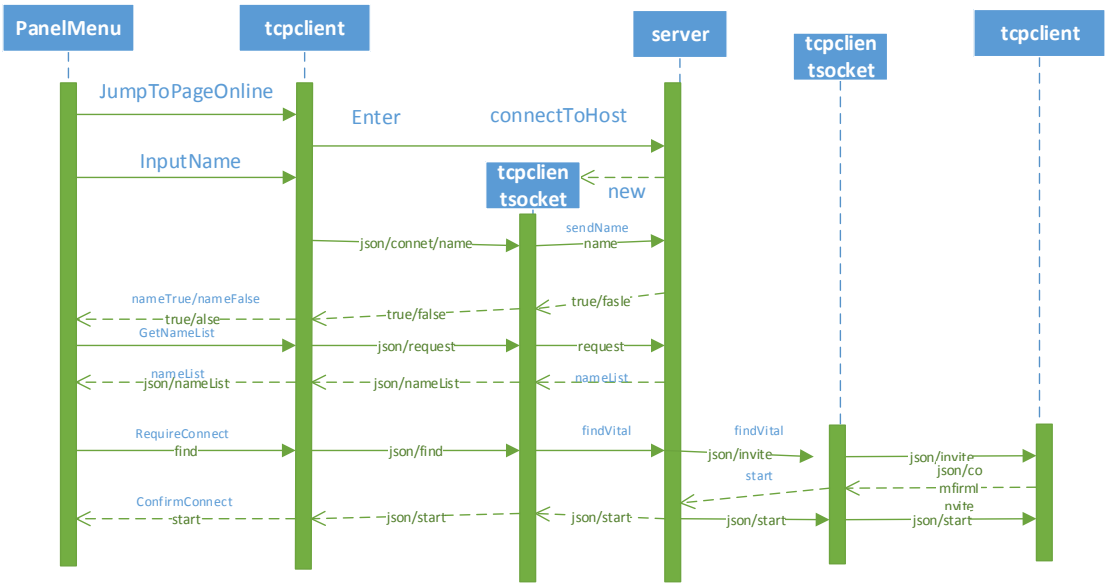
```
tcpSocket->read(buf, length);
buf[length] = '\0';
QString msg = QString::fromLocal8Bit(buf);
QJsonParseError error;
QJsonDocument jsonDocument = QJsonDocument::fromJson(msg.toUtf8(), &error);
if (error.error == QJsonParseError::NoError){
    if (jsonDocument.isObject()){
            QVariantMap result = jsonDocument.toVariant().toMap();
```

由于发送的消息中最后没有'\0'而将 buf 转为 QString 格式是读到'\0'为止，所以需要有 buf[length] = '\0';以便防止 buf 原本就有默认值而造成 msg 的值并不准确。通过 QJsonDocument 类将 msg 转为 json,同时由于无论是发送还是接收都是用的 Utf8 编码，所以也支持中文输入。

得到 json 后，将 json 转成 QVarantMap，因为 Qt 中，QVariantMap 比较像脚本语言中的对象，比较灵活，所以可以传递数组或任意格式的变量。

B. **游戏之前的沟通**

进行 Online 游戏之前，玩家要进行一系列的沟通，其流程图如下：



游戏开始之前与服务器的沟通如上所示，都是在 PanelMenu 中进行，利用 tcpclient 与服务器对应的 tcpclientsocket 进行沟通，大致过程为：

客户端：	**服务器：**
① 连接到服务器	创建相应的 tcpclientsocket
②发送自己的名字	判断名字是否有效并返回
③收到名字有效，向服务器请求名单	返回除该用户外的在线用户名单
④请求与某用户对战	转发邀请给被邀请用户
被邀请用户收到邀请并同意	返回游戏开始消息给双方
⑤收到游戏开始信号	

用到的函数分别为：

PanelMenu::JumpToPageOnline()	Server::incomingConnection(qintptr socketDescriptor)
PanelMenu::InputName()	Server::sendName(const QString& name, TcpClientSocket*client)
PanelMenu::GetNameList()	Server::nameList(TcpClientSocket*client)
PanelMenu::RequireConnect()	Server::findVital(const QString &name, const QString& fromname,const int&map)
PanelMenu::getInvite(QString name,int map)收到邀请 PanelMenu::ConfirmInvite()同意	Server::start(const QString& to, TcpClientSocket*client)
PanelMenu::ConfirmConnect()	

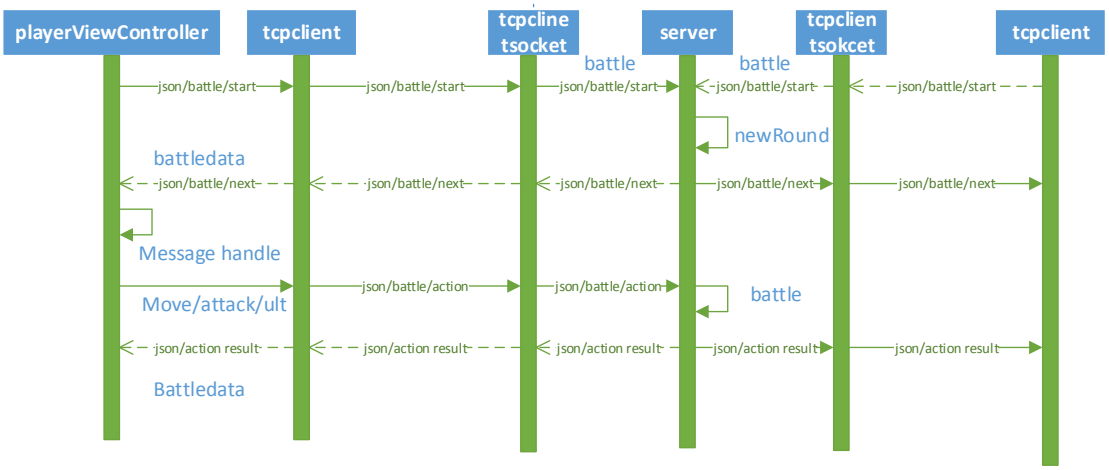
特殊情况：1、玩家输入名字无效则不会请求名单；2、被邀请用户拒绝则不会向服务器发送任何消息；

C. **游戏进行过程**

游戏的开始信号为双方都收到服务器传来的 start 信号。

双方向服务器发送{{"state","battle"},{"action","start"}},当服务器收到二者的信息后便开始游戏循环。

战斗时的流程图如下：



服务器和客户端收发数据还是通过 socket,但是处理数据服务器都是通过 battle 函数进行处理，客户端都是通过 battledata 进行处理，分别将收到的信息转化成相应的结果。

每次客户端只要有 Move/Attack/Ult/Cancel 都会传递给服务器，服务器再把数据的结果后发给参与该棋局的两玩家，两边的 battleData 根据收到的结果改变客户端的 UI、角色系统。

特殊情况：1、当有个玩家退出游戏，则直接向服务器发送 Lose 的消息，服务器将胜利消息发给其对手，然后删除该局；2、玩家网络断开后，服务器删除对应的 socket 并且发送胜利信息给其对手并删除该棋局。

D. **请求游戏公告**

为了显示游戏有没有连接上服务器，给玩家一个比较直观的感受，在 gamewindow 中定义了指针：

```
QNetworkAccessManager*manager;
```

在 gamestarth()函数里，通过 manager 通过 get 方法向服务器发送请求，即请求存在 http://10.134.141.152/inform.txt 中的公告，（公告由管理员改写）在 PanelMenu 中通过 getHttpReply 函数得到该公告的文本并显示在主界面左下角。由于这段文字本身是"未连接到服务器…",所以若请求失败则表示未连接上服务器。

请求公告：

```
manager->get(QNetworkRequest(QUrl("http://10.134.141.152/inform.txt")));
```

接收公告：（因为有中文所以要改成 Utf9 格式）

```
QTextCodec *codec = QTextCodec::codecForName("utf8");
QString all = codec->toUnicode(reply->readAll());
```

■ 客户端数据结构

1. TcpClient 类

1) 功能说明：客户端的 socket，用于和服务器连接，继承 QObject

2) 初始化构造方式：

输入变量：

变量名	变量类型	变量结构	变量意义
parent	QWidget	指针	TcpClient 的父类

内部参数初始化列表

变量名	类型	结构	种类	初值
status	bool	普通变量	private	false
ip	QString	普通变量	private	"10.134.141.152"
HasConnected	bool	普通变量	public	false
userName	QString	普通变量	public	""
enemyName	QString	普通变量	public	""
tcpSocket	QTcpSocket *	指针	Public	nullptr
port	int	普通变量	private	8010
serverIP	QHostAddress *	指针	private	new QHostAddress()

3) 备注：服务器建立在 ip 10.134.141.152 之上，端口为 8010。

2. ButtonBase 类

1) 功能说明：自定义的一个按钮类，可以实现点击或者鼠标放在上面的特效，继承 QPushButton

2) 初始化构造方式

内部参数初始化列表

变量名	类型	结构	种类	初值
_ButtonPicture	QPixmap *	指针	private	new QPixmap();
__PressPicture	QPixmap *	指针	private	new QPixmap();
__ReleasePicture	QPixmap *	指针	private	new QPixmap();

3) 备注：无

3. Character_Element 类和 Character 类

1) 功能说明：Character_Element 类是所有人物的基类，是最底层的类之一。然后，以 Character_Element 为基础衍生出一个新的类，命名为 Character.

2) 类的结构说明：

Chatacter_Element 类

属性	含义

HP	生命值
Max_HP	最大生命值
Initiative	先攻值
Speed	移动速度
Range_Inc	射程增量
AB	攻击加值
AC	防御等级
X/Y	位置坐标
Ability_Bonus	伤害值加成
Dice_Num	伤害骰数量
Dice_Type	伤害骰种类
Critical_Hit_Range	暴击范围

Character 类

属性	含义
Cause_Damage	该单位已造成的伤害数
Whether_Strokes	是否已发动大招攻击
Num_Strokes	剩余可发动大招次数
Round	死亡之后所经过战斗轮数
Whether_Strokes	是否已发动大招攻击
Round_Death	记录死亡时间
Choose	用来选择 Character 所对应的人物，从 1-6 以此代表 Saber，Archer，Rider，Assassin，Caster 和 Berserker。大招以及构造函数，都会按照 Choose 进行初始化

4. Element 类

1) 功能说明：提供各个棋子的 UI、动作接口，继承 QObject 和 QGraphicsItem

2) 初始化构造方式：

输入变量：

变量名	变量类型	变量结构	变量意义
x	int	普通变量	棋子的横坐标
y	int	普通变量	棋子的纵坐标
index	int	普通变量	棋子的种类

内部参数初始化列表

变量名	类型	结构	种类	初值
surface_index	int	普通变量	public	index

_character	Character	类对象	public	_character.set_Choose(abs(index));
				_character.Initial();
				_character.set_X(x - 1);

				_character.set_Y(y - 1); 四个接口初始化
_Index	int	普通变量	private	index
IsAttackNoted	bool	普通变量	private	false
_Effect	QGraphicsColorizeEffect *	指针	private	nullptr

3) 备注：_Effect 在棋子被标记的时候会用到，棋子的 UI 是通过自身继承的 Paint 等函数创建的。

5. ElementBattleInfo 类

1) 功能说明：战斗状态栏的类，继承 QFrame

2) 初始化构造方式：

输入变量：

变量名	变量类型	变量结构	变量意义
state	GameState *	指针	游戏的状态标记
element	Element*	指针	棋子的信息
parent	QWidget *	指针	父类

内部参数初始化列表

变量名	类型	结构	种类	初值
_State	GameState *	普通变量	private	false
_Element	Element*	普通变量	private	element
_ElementIcon	QLabel*	普通变量	private	new QLabel(this)
_HPNumText	QLabel *	普通变量	private	new QLabel(this)
_UltimateSkillState	QLabel *	普通变量	private	new QLabel(this)
_DeathEffect	QGraphicsColorizeEffect *	普通变量	private	nullptr
_ActiveNote	QFrame *	普通变量	private	nullptr
IsDead	bool	普通变量	private	false

3) 备注：上述的各个变量组成了战斗状态栏的各行状态，分别棋子图标、棋子血量、棋子大招数、棋子死亡标记。

6. ElementPool 类

1) 功能说明：存放棋局中各个棋子的信息。

2) 初始化构造方式：

内部参数初始化列表：

变量名	类型	结构	种类	初值
m_Elements	QVector<Element*>	向量	private	nullptr

3) 备注：各种棋子的初始化是在 chessboard 的 initElement 中进行的，通过 ElementPool 的 addElement 函数来添加棋子。与服务器的区别是有一个 void restoreAllEnemyElementColor(bool IsHost);来控制 UI。

7. GameContent 类

1) 功能说明：存放游戏的音乐播放器、地图、棋子以及游戏的设置。继承 QObject
2) 初始化构造方式：

输入变量：

变量名	变量类型	变量结构	变量意义
scene	QGraphicsScene*	指针	存放游戏的场景
localsettings	LocalSettings*	指针	游戏的设置
parent	父类	指针	游戏父类

内部参数初始化列表：

变量名	类型	结构	种类	初值
player	QMediaPlayer *	指针	public	nullptr
playlist	QMediaPlaylist *	指针	public	nullptr
player_battle	QMediaPlayer *	指针	public	nullptr
playlist_battle	QMediaPlaylist *	指针	public	nullptr
_Map	MapInfo*	指针	private	nullptr
_Elements	ElementPool*	指针	private	nullptr
_Scene	QGraphicsScene*	指针	private	scene
_Settings	LocalSettings *	指针	private	localsettings

3) 备注：构造函数里面通过三个函数 initMapInfo()、initElementPool()、initMusic()来初始化。

8. GameMainMenu 类

1) 功能说明：游戏内操作面板类，继承 QFrame。
2) 初始化构造方式：

输入变量：

变量名	变量类型	变量结构	变量意义
content	GameContent *	指针	见 GameContent*类
parent	QWidget *	指针	父类

内部初始化列表：

变量名	类型	结构	种类	初值
_btResume	ButtonBase *	指针	private	new ButtonBase(this)
_btQuit	ButtonBase *	指针	private	new ButtonBase(this)
_slBackgroundMusicVolume	QSlider *	指针	private	new QSlider(Qt::Horizon

				tal, this)
_slEffectVolume	QSlider *	指针	private	new QSlider(Qt::Horizontal, this)
_Content	GameContent *	指针	private	content

3) 备注：GameMainMenu 分别是返回游戏、退出、设置背景音乐音量、设置音效音量。

9. GameState 类

功能说明：存放游戏进行的状态

变量名	类型	功能
IsHost	bool	表示是否是主机
IsChoosingAttackTarget	bool	是否当前是自己的回合
IsChoosingMoveTarget	bool	是否处于点击了 Attack 键而准备选择攻击目标的状态
HasMoved	bool	是否处于点击了 Move 键而准备选择移动目标的状态
HasAttacked	bool	是否已经攻击过
HasUltimateSkilled	bool	是否已经放过大招
AttackSequence	QVector<int>	攻击队列
WhileUltimateSkilling	int	是否处于奇怪的大招状态
Round	int	回合数
AILevel	int	AI 难度

10. GameWindow 类

1) 功能说明：游戏界面的底层类，为 UI 的核心，继承 QMainWindow

2) 初始化构造方式：

变量名	类型	结构	种类	初值
tcpclient	TcpClient*	指针	public	new TcpClient(this)
_Menu	PanelMenu*	指针	private	nullptr
_Scene	QGraphicsScene*	指针	private	nullptr
_View	PlayerViewController*	指针	private	nullptr
_Single	SinglePlayerController *	指针	private	nullptr
_Content	GameContent*	指针	private	nullptr
_LocalSettings	LocalSettings	类对象	private	nullptr
_MainMenu	GameMainMenu *	指针	private	nullptr
manager	QNetworkAccessManager*	指针	private	new QNetworkAccessMan

				ager(this)
AILevel	int	普通变量	private	nullptr
SingleOrMulti	bool	普通变量	private	false

3) 备注：其它指针分别在 gameStart 函数中建立或者是经过与 UI 交互后创建，其它指针的功能见相应的类。

manager 为 QNetworkAccessManager*指针，用于向服务器发送 http 请求。

11. LocalSettings 类

功能说明：存放游戏的设置

变量名	类型	功能
MapIndex	int	地图编号
BackgroundIndex	int	地图背景编号
BackgroundMusicVolume	int	背景音乐
EffectVolume	int	效果音

12. MapInfo 类

1) 功能说明：存放地图的信息。
2) 初始化构造方式：

内部初始化列表：

变量名	类型	结构	种类	初值
CurrentMapFileInfo	stMapInfo	类变量	private	
MapUnitVector	QVector<MapUnit*>	向量	private	

3) 备注：两变量的初始化见相应的类。

13. MapUnit 类

1) 功能说明：存放地图块信息，继承 QObject、QGraphicsItem
2) 初始化构造方式：

变量输入：

变量名	变量类型	变量结构	变量意义
x	int	普通变量	该地图块横坐标
y	int	普通变量	该地图块纵坐标
eType	emFloor	枚举类型	该地图块种类
index	int	普通变量	棋子的标记

内部初始化列表：

变量名	类型	结构	种类	初值
_Index	int	普通变量	private	index

_Xcell	int	普通变量	private	x
_Ycell	int	普通变量	private	y
_eType	emFloor	枚举类型	private	eType
_IsPassable	bool	普通变量	private	false
IsNoted	bool	普通变量	private	false
_ColorEffect	QGraphicsColorizeEffect *	指针	private	nullptr

3) 备注：无

14. PanelBattleInfo 类

1) 功能介绍：战斗状态栏以及回合显示，继承 QFrame
2) 初始化构造方式：

变量输入：

变量名	变量类型	变量结构	变量意义
content	GameContent *	指针	
state	GameState *	指针	
parent	QWidget *	指针	父类

变量意义见相应的类

内部初始化列表：

变量名	类型	结构	种类	初值
__Content	GameContent *	指针	private	content
_State	GameState *	指针	private	state
_RoundNumText	QLabel *	指针	private	new QLabel(QString{std::to_string(_State->Round).c_str()} this);
AllElementInfo	QVector<ElementBattleInfo*>	向量	private	

3) 备注：AllElementInfo 是通过 RefreshNewRoundInfo() 进行初始化的，PanelBattleInfo 类是通过继承的 paintEvent 函数绘制出的。

15. PanelHeadUp 类

1) 功能介绍：显示战斗界面的标题，继承 QFrame
2) 初始化构造方式：

内部变量初始化：

变量名	类型	结构	种类	初值
GraphicsItem	QVector<QPushButton*>	向量	public	
_btMainMenu	ButtonBase *	指针	private	new

				ButtonBase(this)
_btPanelBattleInfo	ButtonBase *	指针	private	new ButtonBase(this)
HealthItem	QVector<QWidget *>	向量	private	
SHAnime	Set_Health_Animation*[12]	指针数组	private	
SHAnime_color	Set_Health_Animation*[12][100]	指针数组	private	
msec_50	QTimer*[12][100]	指针数组	private	
SHAnime_death	Set_Health_Animation*[12]	指针数组	private	

3) 备注：后面三个变量为动画变量，几个变量构成的功能为显示血条掉血的动作。

16. PanelMenu 类

1) 功能介绍：游戏的主界面、Option 界面、单人游戏模式选择界面、多人游戏模式选择界面、Online 模式界面，继承 QFrame。

2) 初始化构造方式：

变量输入：

变量名	变量类型	变量结构	变量意义
tcpclient	TcpClient*	指针	客户端 socket，用于联网
localsettings	LocalSettings*	指针	游戏设置
parent	QWidget *	指针	父类

内部变量初始化：

变量名	类型	结构	种类	初值
_LocalSettings	LocalSettings *	指针	public	localsettings
information	QString	指针	private	QStringLiteral("未连接到服务器....")
playlist	QMediaPlaylist *	指针	private	new QMediaPlaylist
player	QMediaPlayer *	指针	private	new QMediaPlayer
tcpPoint	TcpClient *	指针	private	tcpclient
_PageSinglePlayer	QFrame *	指针	private	nullptr
_PageMultiPlayer	QFrame *	指针	private	nullptr
_PageOption	QFrame *	指针	private	nullptr
_PageMain	QFrame *	指针	private	new QFrame(this)
_btSinglePlayer	ButtonBase *	指针	private	new ButtonBase(_PageMain)
_btMultiPlayer	ButtonBase *	指针	private	new

				ButtonBase(_PageMain)
_btOption	ButtonBase *	指针	private	new ButtonBase(_PageMain)
_btExit	ButtonBase *	指针	private	new ButtonBase(_PageMain)
inform	QLabel*	指针	private	new QLabel(_PageMain)

3) 备注：PanelMenu 中还有很多父类为_PageSinglePlaye 之类的类的子类的指针，所以没有列出来，这些指针在 PageSinglePlaye 等创建出来时也会相应创建出来。

17. PanelOperate 类

1) 功能介绍：游戏内操作面板的类，继承 QLabel。
2) 初始化构造方式：

变量输入：

变量名	变量类型	变量结构	变量意义
x	qreal	普通变量	操作面板出现地点横坐标
y	qreal	普通变量	操作面板出现地点纵坐标
state	const GameState&	常量引用	游戏进行的状态
type	Operate_type	普通变量	操作的按钮类型
EX_state	bool	普通变量	大招状态
parent	QWidget *	指针	父类

内部变量初始化：

变量名	类型	结构	种类	初值
_btAttack	ButtonBase *	指针	private	new ButtonBase(this)
_btMove	ButtonBase *	指针	private	new ButtonBase(this)
_btUltimateSkill	ButtonBase *	指针	private	new ButtonBase(this)
_btCancel	ButtonBase *	指针	private	new ButtonBase(this)
_Offset	QPoint *	指针	private	_Offset = new QPoint

3) 备注：根据 type 的不同_Offset 的会根据不同的坐标初始化。

18. SinglePlayerController 类

1) 功能介绍：提供了单机游戏的控制台，处理单机模式下以及的各种数据，继承 QGraphicsView
2) 初始化构造方式：

变量输入：

变量名	变量类型	变量结构	变量意义
scene	QGraphicsScene*	指针	游戏场景
content	GameContent*	指针	游戏的音乐、地图及设置
ailevel	int	普通变量	游戏模式
parent	QWidget *	指针	父类
IsHost	bool	普通变量	是否是主机

内部变量初始化：

变量名	类型	结构	种类	初值
_Content	GameContent *	指针	private	content
_Operate	PanelOperate *	指针	private	nullptr
_Scene	QGraphicsScen*	指针	private	scene
_timer	QTimer *	指针	private	nullptr
_BulletPicture_Archer	Element*	指针	private	new Element(0, 0, 13)
_BulletPicture_Caster	Element*	指针	private	new Element(0, 0, 14)
_BulletPicture_Saber	Element*	指针	private	new Element(0, 0, 15)

3) 备注：还有些变量并未初始化而是在游戏进行中因为某些条件被创建的临时变量，这些变量都不在上述范围之内。
4) 后面三个变量分别为三种棋子大招时调用的弹道图片。

19. PlayerViewController 类

1) 功能介绍：提供了网络游戏的控制中心，处理网络模式（Online 模式）下的各种数据，继承 QGraphicsView。
2) 说明：由于 PlayerViewController 类是从 SinglePlayerController 中分离出来的一个与之相似的控制中心，所以数据结构大致类似，下面只指出数据结构不同的地方：

① 构造函数中将用于选择游戏模式的 ailevel 改成了用于网络数据传递的 TcpClient*tcpclient;

② 在内部变量中新增是否开始游戏的一个标识性变量 bool gamestart 表示游戏是否开始运行，初始化为 false。

■ 服务器数据结构

1) 服务器网络部分类

tcpserver 类：

1) 功能说明：服务器界面，继承 QDialog 类
2) 内部参数初始化列表

变量名	变量类型	变量结构	变量种类	初值

ContentListWidget	QListWidget	指针	private	QListWidget(this)
PortLabel	QLabel	指针	private	QLabel(QStringLiteral("端口："))
PortLineEdit	QLineEdit	指针	private	QLineEdit
CreateBtn	QPushButton	指针	privet	QPushButton(QStringLiteral("创建服务器"))
mainLayout	QGridLayout	指针	private	QGridLayout(this)
port	int	指针	private	8080
server	int	指针	private	nullptr

3) 类关联

4) 备注：server 在 tcpserver::slotCreateServer()中被创建，即点击创建服务器的 CreateBtn 后创建 server。

server 类：

1) 功能说明：服务器的逻辑处理中枢以及各棋局的存储器，继承 QTcpServer

2) 初始化构造方式：

输入变量：

变量名	变量类型	变量结构	变量意义
parent	QObject	指针	server 的父类
port	int	普通变量	服务器端口号

内部参数初始化列表：

变量名	类型	结构	种类	初值
tcpClientSocketList	QList<TcpClientSocket*>	QList	private	nullptr
battleBoard	QList<chessboard*>	QList	private	nullptr
userMap	QMap<QString, TcpClientSocket*>	QMap	private	nullptr

3) 备注：tcpClientSocketList 用于存放各个连接进来的用户的 socket

battleBoard 存放各个正在进行中的棋局

userMap 存放了用户的 socket 以及相应的用户名

tcpclientsocket 类：

1) 功能说明：存放服务器端对应的各个客户端的 socket，继承 QTcpSocket

2) 初始化构造方式:

内部参数初始化列表：

变量名	变量类型	变量结构	变量意义
name	QString	普通变量	客户端名称
_HasStartGame	bool	普通变量	表示是否正在游戏

3) 备注：无

2) 服务器角色系统类

说明：服务器角色系统类与客户端角色系统类类似，不过服务器角色系统去除了客户端大量的 UI 部分的代码，让其运行效率更高。

chessboard 类

1) 功能说明：存放棋局信息
2) 初始化构造方式：

输入变量：

变量名	变量类型	变量结构	变量意义
newplayer1	QString	普通变量	玩家 1 名称
newplayer2	QString	普通变量	玩家 2 名称

内部参数初始化列表：

变量名	类型	结构	种类	初值
player1	QString	普通变量	public	newplayer1
player2	QString	普通变量	public	newplayer2
Round	int	普通变量	public	0
readyPlayer	int	普通变量	public	0
_Elements	ElementPool*	指针	public	nullptr
AttackSequence	QQueue<int>	队列	public	nullptr

3) 备注：readyPlayer 用来判断是否游戏开战的双方都准备好了，若准备好了即 readyPlayer2 则开始进行游戏循环。

ElementPool 类

1) 功能说明：存放棋局中各个棋子的信息。
2) 初始化构造方式：

内部参数初始化列表：

变量名	类型	结构	种类	初值
m_Elements	QVector<Element*>	向量	private	nullptr

3) 备注：各种棋子的初始化是在 chessboard 的 initElement 中进行的，通过 ElementPool 的 addElement 函数来添加棋子。

Element 类

1) 功能说明：存放服务器中棋局的棋子信息。
2) 初始化构造方式：

输入变量：

变量名	变量类型	变量结构	变量意义
x	int	普通变量	棋子初始 x 坐标
y	int	普通变量	棋子初始 y 坐标
camp	int	普通变量	棋子阵营
choose	int	普通变量	棋子类型

内部参数初始化列表：

变量名	类型	结构	种类	初值
_Camp	int	普通变量	private	camp

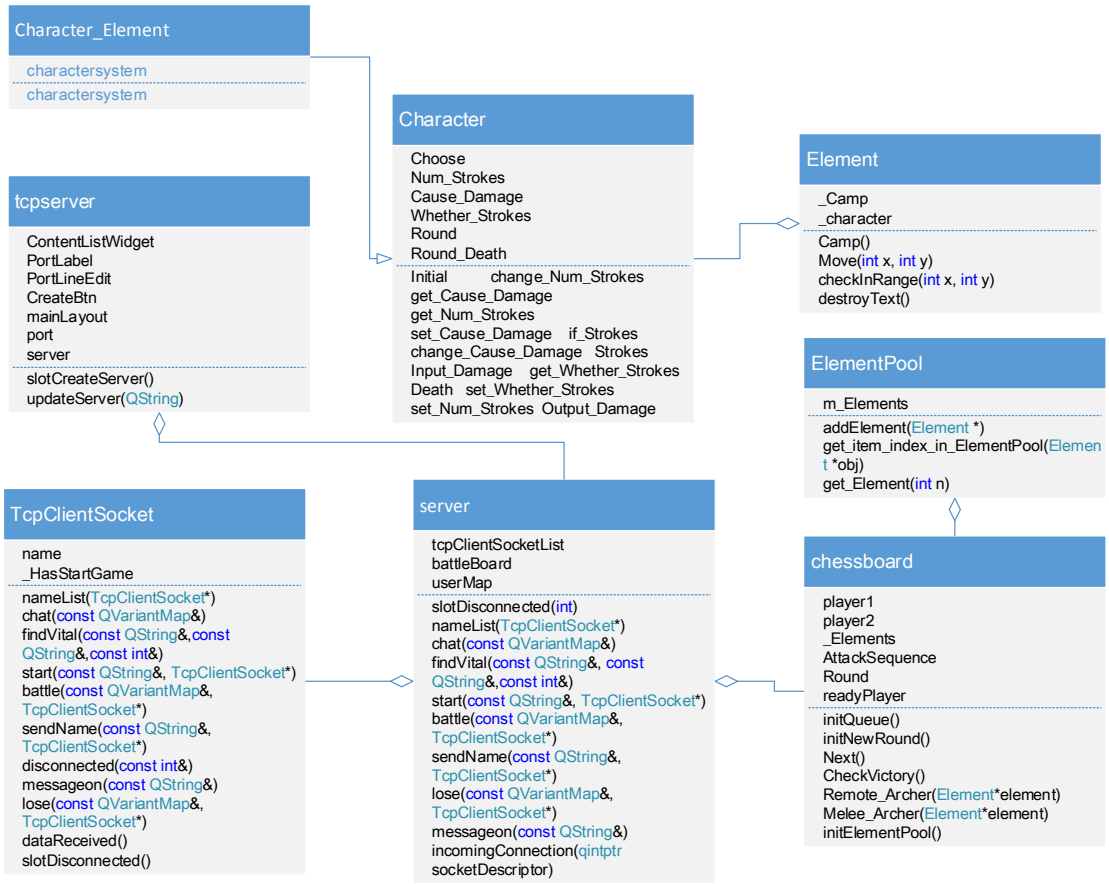
_character	Character	类对象	public	通过_character的
				_character.set_Choose(choose);
				_character.Initial();
				_character.set_X(x);
				_character.set_Y(y);
				接口来初始化

3) 备注：Element 类是棋子的信息，虽然大多数功能其实可以直接合并在 Character 类中但是为了保持客户端和服务器的一致便于维护所以仍然采取了 Character 和 Element 类分开的做法。
Element 类就是客户端去除 UI 部分的 Element 类。

Character 和 Character_Element 类

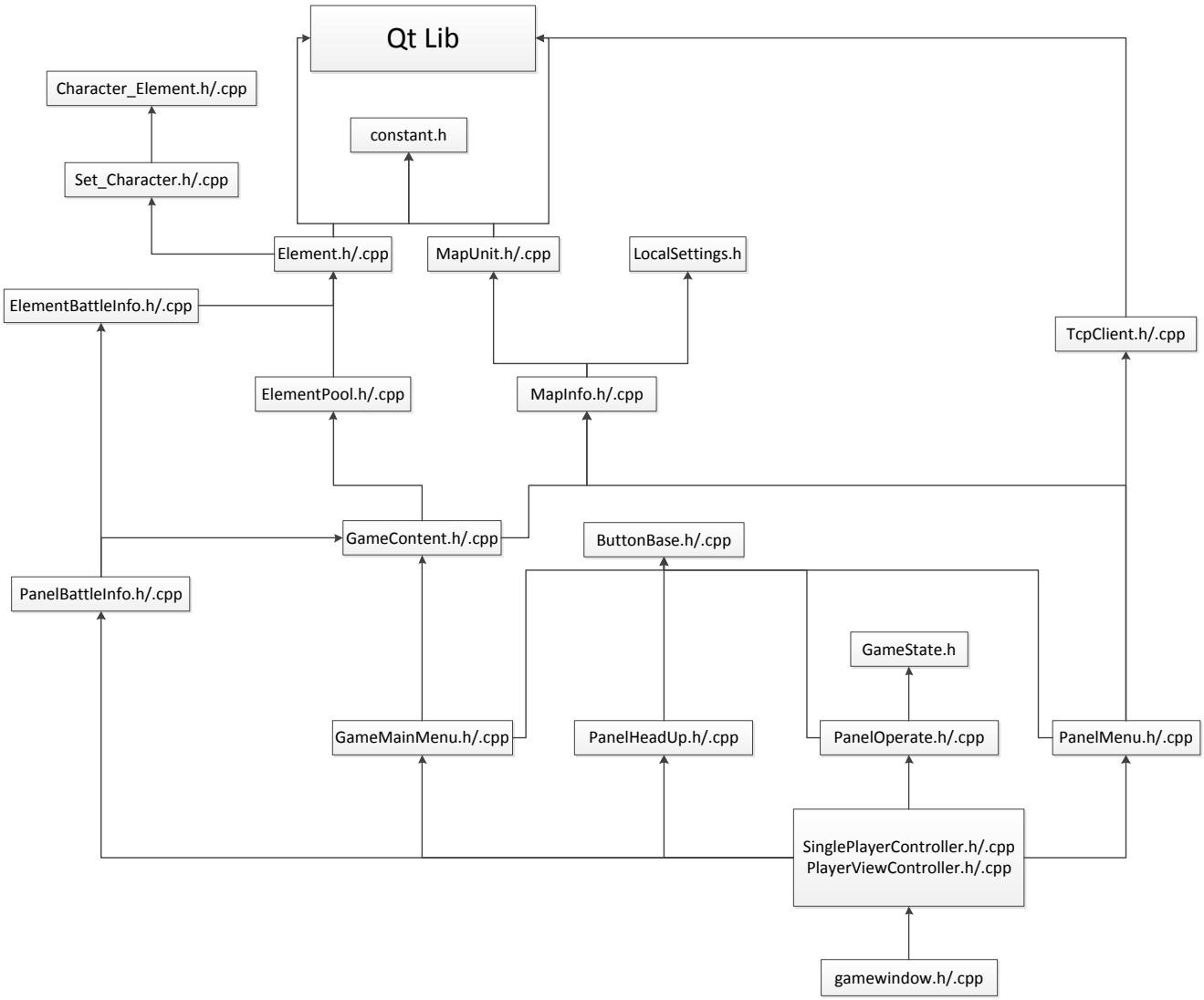
这两个类客户端和服务器一样所以在客户端部分详细说明，这里就不再赘述。

类图为：



3.2.2 系统体系结构

■ 代码结构



3.2.3 系统动态行为

在整个系统的运作中，主要发生以下动态行为

- 游戏初始化
- 进入游戏界面页面切换控制
- 游戏抬头面板的血量显示变化
- 游戏操作面板控制棋子行动
- 每个棋子先攻权轮换的游戏循环控制
- 战斗信息面板的刷新
- 敌人 AI 的对策
- 网络数据传输

其中大部分在关键算法中已有说明，故此处不再赘述

系统内存泄露问题

由于该系统使用了大量的指针申请空间并且每一个界面的切换都是伴随着新一次的空间申请，所以内存泄露对于该系统来说是一个不得忽视的问题。
对于内存泄露问题，我们小组的解决方法如下：

1、 利用 Qt5.4 的内存管理机制：

Qt5.4 中规定一个对象被释放时，首先会将其子类的空间释放，例如我们在给指针申请 GameMainMenu 对象时，GameMainMenu 中某些元素申请空间时会将 GameMainMenu 作为他们的父类，如下代码所示：

```
QLabel* text_BGM_volume = new QLabel("Background VOL", this);
.....
_slBackgroundMusicVolume = new QSlider(Qt::Horizontal, this);
.....
```

当 GameMainMenu 被释放则会先将 text_BGM_volume 以及_slBackgroundMusicVolume 的空间释放在将自身的空间释放。

2、 利用 Qt5.4 的 deletelater()函数

Qt5.4 的 deletelater()函数与 C++的 delete()相似，都是释放指针申请的空间。不过 deletelater()指调用 deletelater()函数的函数执行完毕后回到事件循环后将该对象删除或者等到该函数所在线程结束后释放对象。在基类利用完毕后我们利用 deletelater()释放空间避免内存泄露。

例如在 PanelMenu.cpp 中 EnterMainWindow()函数：

因为 this->hide();只是将 this 隐藏起来但仍然存在堆中，于是调用 this->deleteLater();将

这部分空间清除。

```
void PanelMenu::EnterMainWindow()
{
    emit GameStart_Signal();
    this->hide();
    this->deleteLater();
}
```

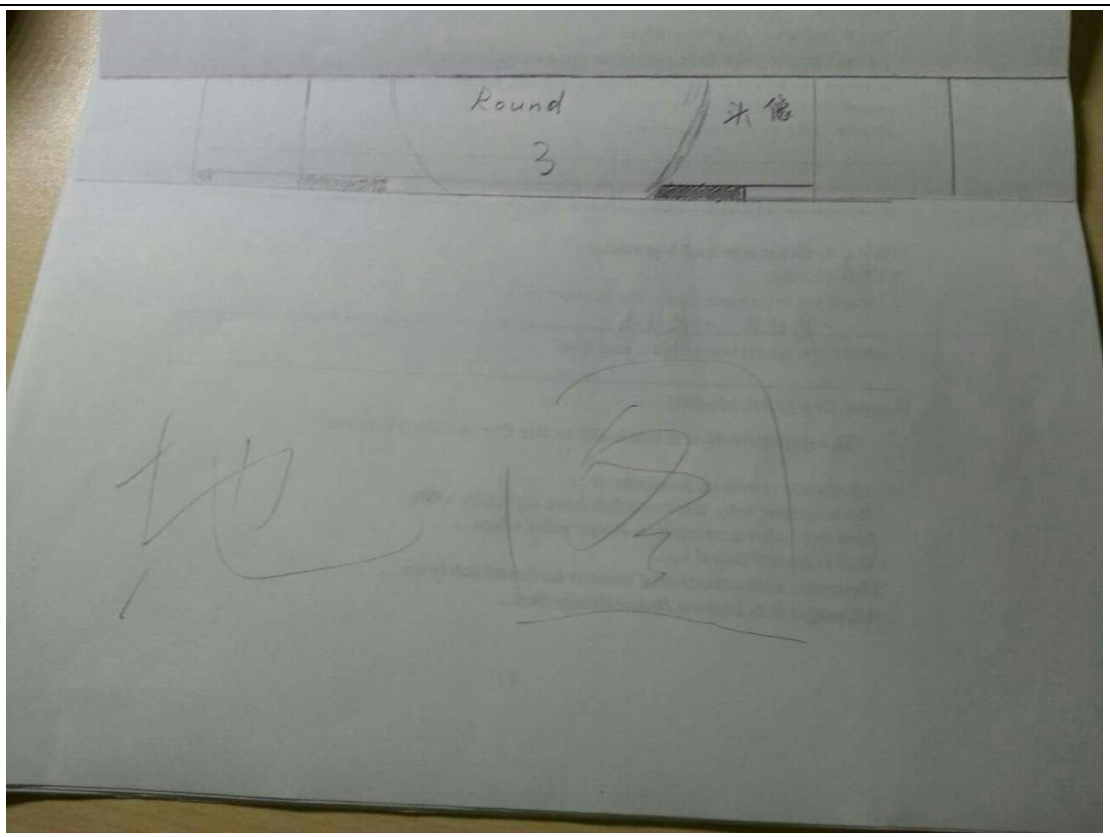
在 gamewindow 中从游戏界面返回到主界面时的 returnPanelMenu 也是做了防止内存泄露的处理：

```
void GameWindow::returnPanelMenu()
{
    //防止内存泄露开始删东西
    if (SingleOrMulti){
        _View->deleteLater();
        singleToMulti();
    }
    else{
        _Single->deleteLater();
    }
    _Scene->deleteLater();
    _Content->deleteLater();
    disconnect(manager, 0, 0, 0);
    this->gameStart();
    this->show();
}
```

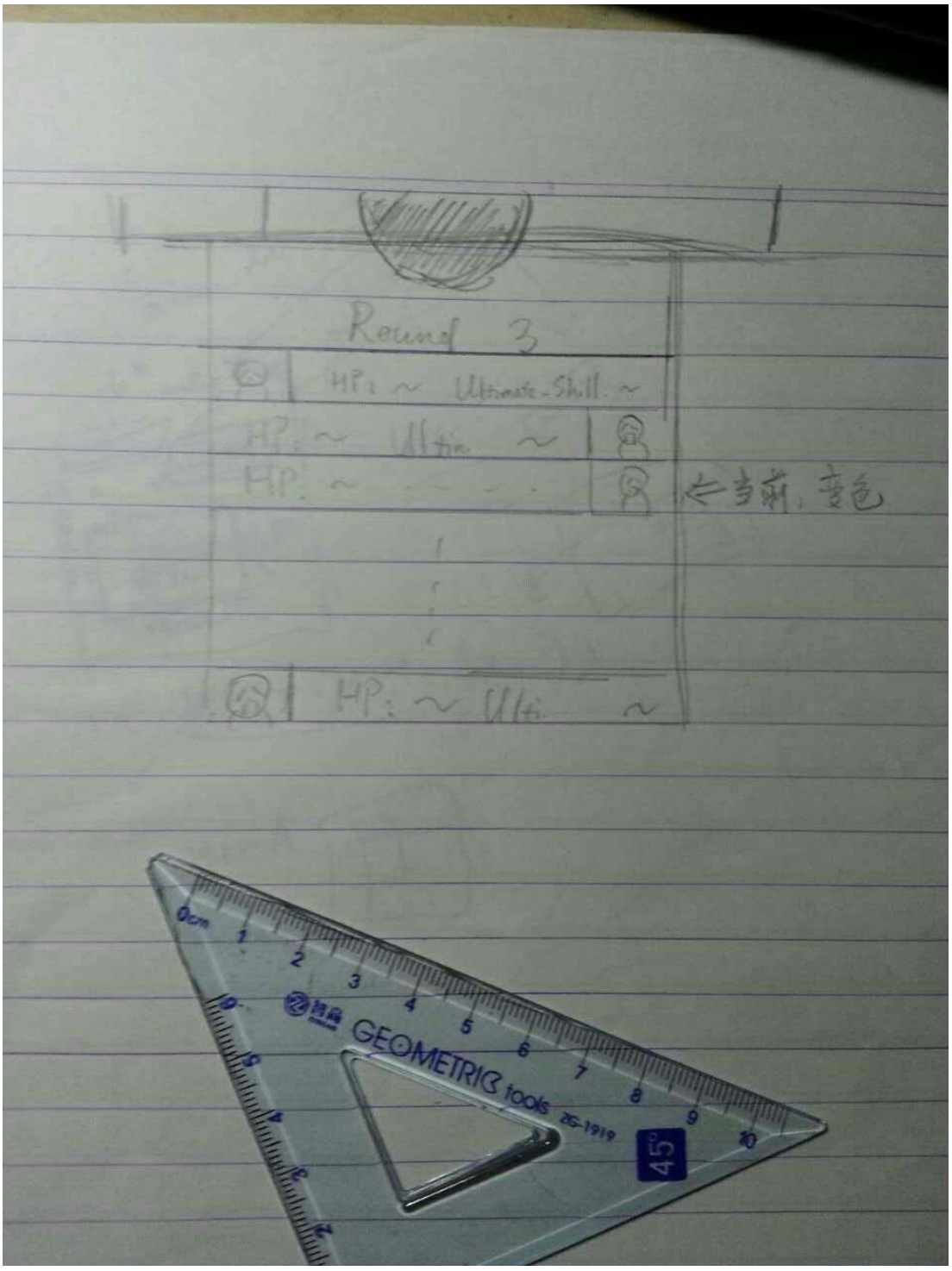
3.3 用户界面设计

3.3.1 用户界面草图

以下为**游戏主界面**和**战斗信息面板**的设计草图（By 朱晨畅）



Drawn by 朱晨畅


Round 3
HP: ~ Ultimate-Skill ~
HP: ~ Ultim ~
HP: ~
HP: ~ Ulti ~
当前角色
0cm
GEOMETRIC tools
45°

Drawn by 朱晨畅

3.3.2 用户界面说明

主界面

主界面上有四个按钮，从左至右分别为 Singleplayer，Multiplayer，Option，Exit，分别对应单人游戏、多人游戏、操作设定和退出游戏，前三个按钮在点击后会跳转至对应界面。左下角为公告栏，当有游戏版本更新等信息时会在左下角显示，如果未连接上服务器也会在左下角显示出来。

左下角显示版本号：

左下角显示未连接：

Singleplayer 页面

Singleplayer 界面左上角为返回主界面的按钮，下方从左至右分别为 Tutorial，Easy，Normal，Hard，Lunatic，分别对应 AI 难度的练习模式、简单模式、中等模式、困难模式和疯狂模式，点击后会选择对应难度的 AI 进行游戏。

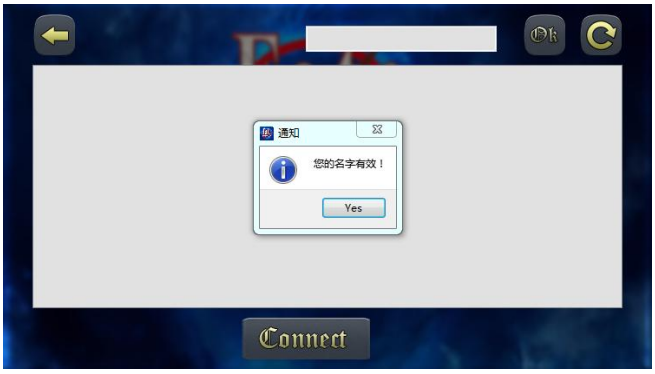
Multiplayer 页面

MultiPlayer 界面左上角为返回主界面的按钮，下方从左向右分别问 Online、Offline 模式。Offline 模式即为在同一台电脑上两位玩家的对战，方便只有一台电脑却有两个玩家的对战，点击即进入游戏。Online 模式为网络模式，即寻找在线用户与之对战。

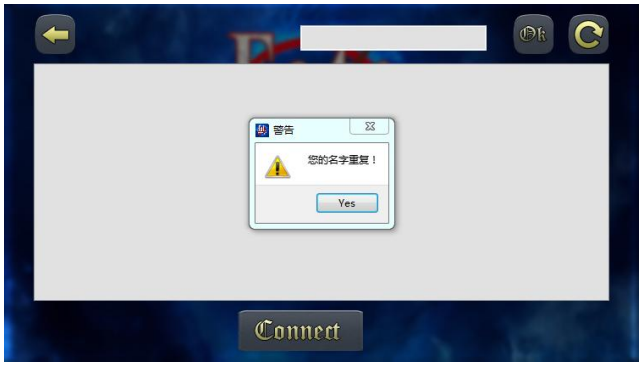
Online 页面

进入 Online 模式界面后，首先需要在左上角的文本框中输入自己的名字，点击 **ok** 按钮则向服务器发送自己的名字，若名字有效则会返回提示，如果和其他人重复则会返回名字重复的提示，若无任何反应则是网络异常未连接上服务器。当名字有效后会收到在线用户的列表显示在下方。若之前已经输入过名字，再进入时重新输入名字自己的用户名还是不会变。自己输入名字有效后若点击右上方的刷新按钮则会刷新在线用户列表。

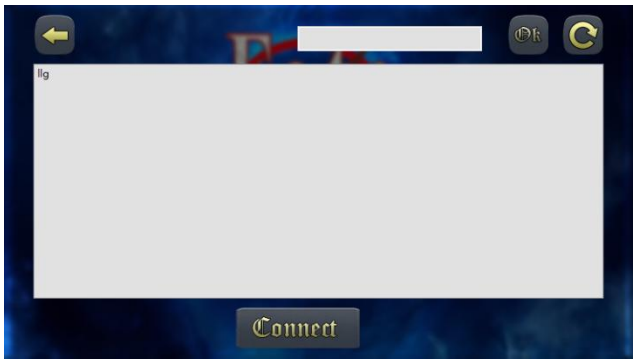
输入的名字有效：



输入的名字已经存在：

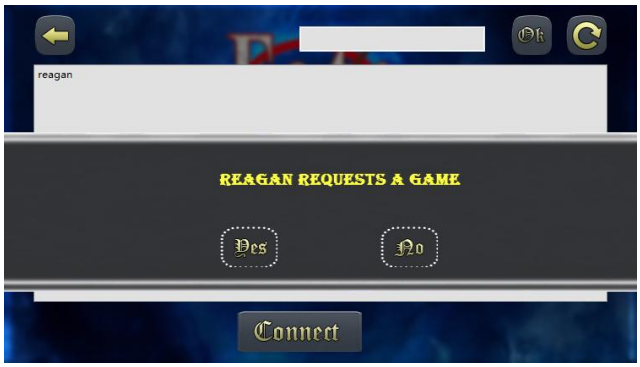


用户列表中显示在线用户：



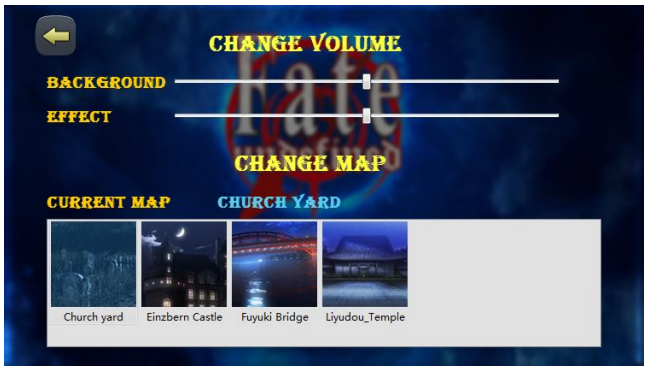
点击列表中的某位在线用户，再点击 **connect** 便会向对方发送游戏申请，对方收到后会显示出一个 Dialog 提示。

收到玩家 Reagan 的邀请：



点击 YES 后双方即可开始对战，点击 NO 则双方都无法开战。

Option 页面

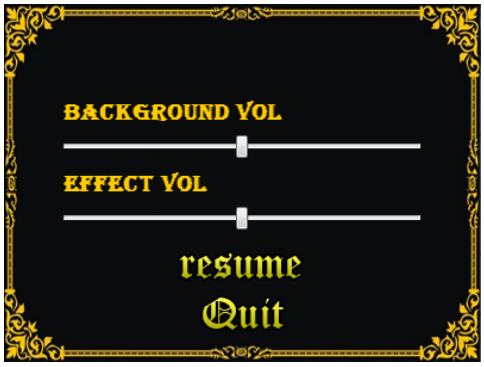


Option 界面左上角为返回主界面的按钮，音量调整区域中拥有两个滑条，分别对应背景音乐和效果音。地图选择中 Current Map 显示当前选择的地图名字，下方可以选择地图。

游戏内界面

游戏内界面有黄蓝框的棋子分别为双方的 6 枚棋子。WASD 或是鼠标左键拖拽都可以移动界面。上方左侧 Main Menu 按钮可以调出**游戏主菜单**，中间游戏图标按钮可以调出**战斗信息面板**。

游戏主菜单



游戏主菜单提供了游戏背景音量调整和效果音量调整，与 Option 界面的音量调整区域作用相同。下方 resume 按钮取消主菜单返回当前游戏，Quit 表示放弃游戏返回主界面。

战斗信息面板

战斗信息面板中包含双方棋子的许多信息，从上至下为先攻队列，黄框表示当前行动的棋子，HP 与 EX 分别表示该棋子当前 HP 与 EX 技能的情况，最上方 Round 提示当前轮数。