2017夏小学期大作业说明文档

李仁杰

2016013271

**重要说明**:

本次大作业一共有三个工程，分别是：

Gwent\_Console

Gwent\_Client

Gwent\_Server

其中，Gwent\_Console提供了一个可以玩的控制台程序，Gwent\_Server是服务端程序，Gwent\_Client是客户端程序

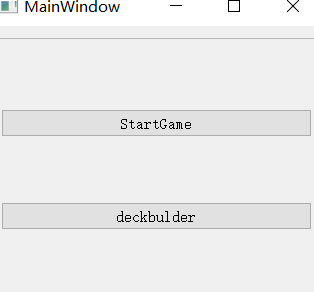
服务端与客户端本来应该通过网络链接，但是由于时间关系我没有写网络模块，我在客户端上添加了一个输入窗口用于输入本来应该由服务端发送的字符串，服务端会在控制台下打印出本来应该输出到客户端的字符串。字符串的格式为(to%userid%)command#param1$para2$para3…。（为了方便测试，服务端也会输出一些交互信息。）当客户端收到服务端的字符串之后，会对图形界面做出更新。

**测试说明:**

测试卡牌效果，考察逻辑架构建议使用Gwent\_Console。如果要测试客户端和服务端，可以运行客户端，然后手动把客户端输出的命令通过服务端的输入命令窗口输入进服务端，当客户端返回一个值的时候，把他输入到服务端的控制台窗口。

**界面说明**

**1.Client主界面**

:

点击StartGame会弹出一个交互窗口，一个命令输入窗口(用于输入本来应该由服务端传输过来的数据)。

**1.1.1交互窗口**



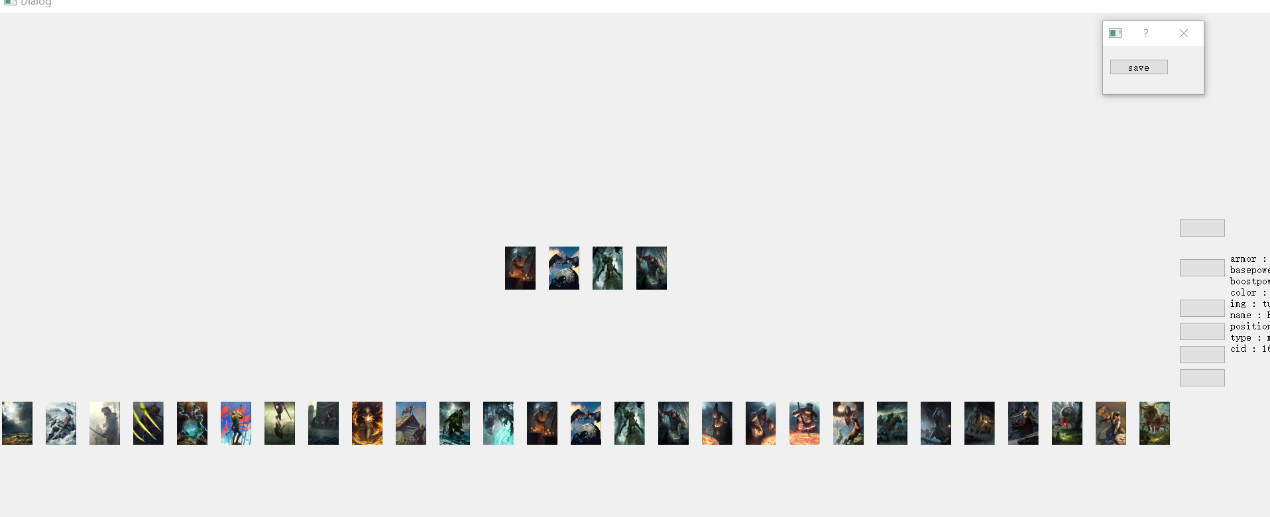
左边显示场面，六个按钮代表六个横排，当鼠标点击一张卡/横排按钮时，右边会显示它的属性，上面可以设置得分，右上方和右下方可以显示双方得分。

**1.1.2命令输入窗口**



文本编辑框用于输入命令，点击submit确认命令并提交给交互窗口，下方显示历史命令记录和用户交互窗口的输出(这些输出本来应该发送给服务端)，输出的格式是select\_*type* *value。*

**1.2套牌编辑器**

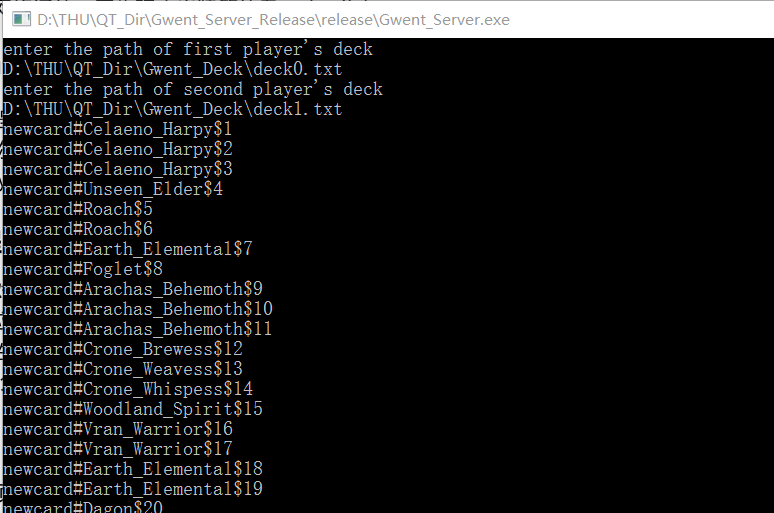


点击套牌编辑器，会弹出两个窗口，大窗口的下端是目前的牌，点击下端的牌可以把他加入你的套牌，你的套牌显示在中间排(row\_1\_0)。点击中间排的牌可以把他移除你的套牌。点击小窗口的save按钮可以把你的套牌保存为txt格式的文本文件。你也可以不使用套牌编辑器直接创建文本文件，然后手动输入或从剪切板粘贴，只要你描述套牌的字符串符合规范，这个套牌也可以被载入。

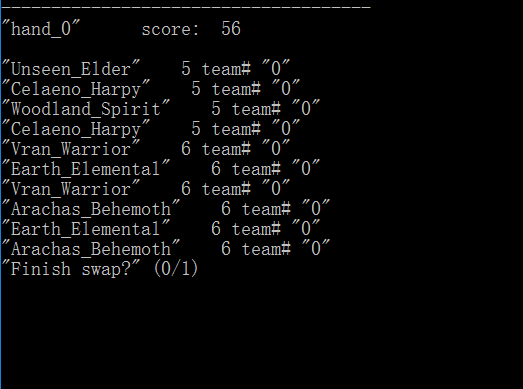
注:为了方便测试效果，程序对套牌没有限制，也就是说你可以带任意数量的任意卡，如果你带了多张领袖卡，那么保证在游戏开始时能被抽到的只有一张。

**2．Server主界面**

Server的主界面是一个控制台。最开始输入双方套牌的绝对路径。



形如上图的输出是发送给客户端的命令字符串



形如上图的输出是方便测试的少量交互。

**3.控制台**

这是一个可以玩的控制台版本，用于方便考察逻辑和卡牌机制的实现效果。

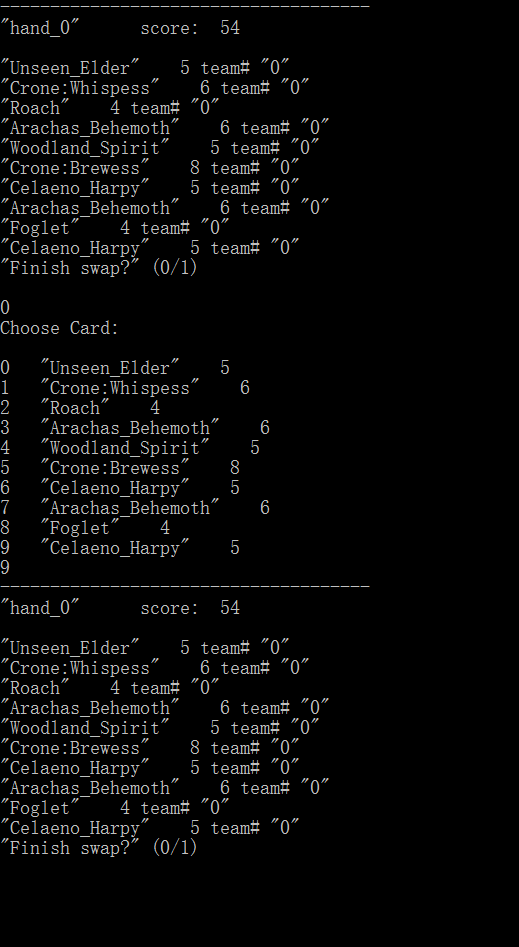
注意由于最开始的想法是用Client和Server进行通信，所有控制台版本不会检查输入，如果输入不合法程序大概率会崩溃。

**3.1**

在程序开始的时候，程序会要求你输入双方套牌的绝对路径，请输入正确的路径并且保证套牌中的卡牌的名字正确，在Gwent\_Deck目录下有两套已经组建好的套牌，可以测试大部分卡牌效果，也可以自己组建套牌来测试卡牌效果，允许出现在套牌中的卡牌名字在ALLCARD.txt中储存(如果ALLCARD.txt中没有某些要求实现的卡牌的名字，是ALLCARD中漏掉了，可以自行加上，注意“蔽日浓雾”的卡牌名字是*Impenetrable\_Frog*)。

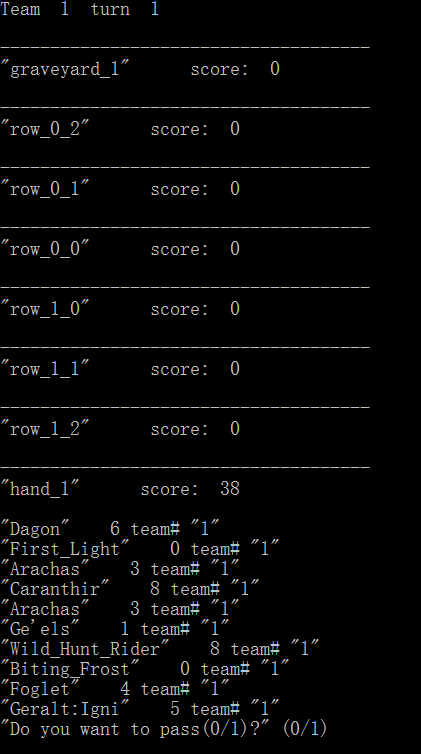
**3.2**

在一局(Round)游戏开始前，程序会显示当前的局数**3秒**(请不要以为是程序进入了死循环)。然后程序会显示当前手牌并询问是否需要调度手牌



hand\_0 代表当前选择是否调度的是第0个玩家，score是你手牌的战力之和，如果你选择调度手牌(0)，那么程序会要求你选择一张卡，你只需要输入这张卡前面的数字就好了。

**3.3**



回合(Turn)开始的时候，程序会在最上端显示当前操作的玩家和回合数，程序会打印当前的场面，row\_k\_0/1/2分别是第k个玩家的近战、远程、攻城排，graveyard是墓地，hand是手牌，然后你可以根据提示进行游戏。注意当程序要求输入一个数字的时候，程序会检测数字是否在规定的范围内，但是不会检测你输入的是不是数字，如果你输入的不是数字，程序大概率会崩溃。

3.3

一局(Round)游戏结束之后程序会打印这局的对局信息三秒钟，正常游戏结束后，程序会打印胜利者和小局比分并进入休眠。

**架构：**

**Console**

**1.SI\_Object类**

继承自QObject，里面用一个QMap<SI\_String,SI\_String>维护了对象的“属性”，并且提供了设置属性，得到属性的操作。[SI\_String是我包装的一个字符串类型，在本程序中它是QString]

2.

战场的操作与信息获取在Field类里，任何改变场上信息的操作都通过Field类里面的信号完成。信号分为多层，最高层比如造成伤害，打出卡牌。中间层比如调整护甲、战力。底层比如调整一个对象的属性、调整一张卡牌的位置。

每类信号都有两类，比如打出卡牌：

\_playCard信号链接playCard槽，用于把一张卡打出去(调整位置)。

playCard槽执行结束之后，会发送playCard\_信号，用于造成打出这张卡牌的后续效果和通知需要知道场上是否打出了一张卡牌的对象。

**3.卡牌**

**3.1Card基类**

继承自SI\_Object，Card基类里面有一些基本的卡牌信息，并且提供了一些基本的槽(比如这张卡牌被打出了)的虚函数，这些槽在Card基类的构造函数中被链接到信号上，当场上发生相应时间的时候会被调用。

**3.2卡牌类**

每一张不同的卡牌都是一个类，继承自Card基类，如果卡牌有相应的效果，只需要override相应的槽函数就可以了，如果卡牌的效果不止和自己有关，可以自己定义一些槽函数，覆盖 Card类中的 \_\_doConnect函数并且在这个函数中对新增的槽函数做相应的链接，这个函数会在相应的卡牌被实例化的时候自动调用。

具体的卡牌类使用了QMetatype，所以可以通过一个存有卡牌名字的字符串来实例化。具体的卡牌类只能使用Card::factory(Game\* game\_ptr ,SI\_String cardName)进行实例化并且用 .destroy() 成员摧毁。

添加一张卡牌需要添加一个新的类，也就是一个头文件和一个源文件，不需要对除此之外的代码做出任何修改。

**4.天气**

天气的实现和卡牌相同，也是使用一个Weather基类并用其factory函数来实例化一个具体的天气。

**5 Game类型**

Game管理游戏的进度，里面有一个field成员和两个用于用户交互的 UserInteraction成员。

游戏流程控制通过 \_startGame , \_startRound, \_startTurn, \_endTurn, \_endRound, \_endGame消息来控制。

**Client:**

主体架构和Console相同，不过Card基类没有派生类，因为客户端不进行计算，也就不会发动效果。

Card类里面加了一个Pushbutton，CardSet类里面加了一个Widget，每当场上局面变化的时候，CardSet就会遍历自身里面有的那些Card对象，并且更新自己的Widget，所以不需要额外的代码来管理。

**Server**

和Console基本相同

**Client和Server的交互**

Client可以收到的字符串有以下几种:

newcard#cardName$id实例化一个名字是cardName的卡片并对应到id

setplace#id$cardSetName 把一个id的卡片对象的位置移动到cardSetName对应的CardSet里

setproperty#id$proName$proVal 把id对应的卡片的proName属性设置为proVal

setscore#cardSetName$val 把cardSetName对应的CardSet的得分设置为val

setwinround#k$x 把第k个用户的胜利局数显示为x

inputcard#id1$id2$id3$... 从id列表中选择一张卡并返回id

inputrow#a$b$c$... 从rowName列表中选择一张卡并返回id

inputorder#rowName... 从rowName对应的row里面选择一个位置并且返回它

每个项目目录下的DoxygenFile/html/index.html有相应项目的继承关系文档