计算概论（B）小课题结题报告

题目： 三国杀小游戏

程序名称： sanguosha.app

姓名：韩子钊 沈士恒

学号：1600011301 1500015941

目录：

- 摘要
- 功能总结
- 代码结构总结
- C++类总结
- 游戏进程总结
- 可视化总结

摘要：

关键词：游戏，纸牌，三国杀

本课题旨在用Qt Widget Application编写一个mac平台的三国杀本地人机对战游戏。玩家通过控制一名角色，进行标准三国杀4~8人局的游戏。（游戏简介可参考：三国杀）由于控制该游戏进程远比预想复杂，我们缩减部分手牌（诸如与判定相关的卡牌并未加入），缺省了武将和个人技能，但仍较为完整地实现了游戏其余的全部功能，包括一个极为简单但较为合理的AI（或者叫应对系统），该AI基本使得其能力与初级人类玩家相比，使得游戏难度较为平衡，不会出现一边倒的情况。相比与现实的纸牌游戏，我们会在信息窗口中显示玩家当前的全部选项，对新手极为友好，即使第一次玩，也能很快上手。
游戏程序在os x上用Qt Creator写成，编译用*Qt 5.7.0 clang 64 bit*.

功能总结：

点击Start按钮，弹出对话框选择游戏人数，确认后开始游戏进程。最底下从右到左依次是玩家头像和玩家血量（以勾玉显示），手牌区，装备区（四条文字框）。上方是其余的七位电脑玩家，从玩家开始逆时针递增。若玩家数小于8，则空出玩家右手边若干位。每个电脑玩家头像右上方显示剩余手牌，右边是血量，下方是装备区。游戏进行中，信息通过弹出的**Game Message**显示给玩家，玩家可以上下滚动，浏览游戏历史；装备区上方有托管选项，可以加快游戏进程，增加游戏可玩性。游戏开始后随机分派游戏身份，随后洗牌，分配初始手牌，由主公开始第一个回合。
当游戏进行到某玩家回合时：
1. 该玩家头像处会有红色边框显示。
2. **Game Message**会显示玩家全部可选择的动作，玩家通过向textinput框中输入选项数字，随后回车确认选项。
3. 弃牌会在游戏中央区堆叠起来，牌出尽后会重新洗牌（游戏会卡顿数秒)。
4. 该玩家头像处会有红色边框显示。
5. 游戏支持一定程度的撤回功能（如：出**杀**后，选择目标时，可取消，退回到上一次选择位置）。 当判断游戏结束时，会弹出很大的游戏结束信息，宣布哪方获胜。
当某角色死亡后，在其头像上出现红色标记“阵亡”，并在其头像右上方显示其身份。 游戏通过点击"退出游戏"按钮退出。

代码总结

Summary:
151 text files.

C++ Source:
- Number of Files: 75
- Lines of Code: 3866
C/C++ Header:
- Number of Files: 76
- Lines of Code: 1539

文件结构（tree命令）

源文件目录

结构总览

```
.
├── Base of Event & SkillEffect
├── Cards
│   ├── EquipCard
│   │   └── WeaponCard
│   └── NonEquipCard
├── Core
├── Event
```

```
|    └──── public\ class\ Event
├──── Player
└──── SkillEffect
```

各结构下文件：

（1）

```
── Core
   ├──── Cheat.cpp
   ├──── Cheat.h
   ├──── Exception.cpp
   ├──── Game.cpp
   ├──── Game.h
   ├──── GamePointer.h
   ├──── Lib.h
   ├──── main.cpp
   ├──── mainwindow.cpp
   └──── mainwindow.h
```

（2）

```
── Base of Event & SkillEffect
   ├──── MoveStruct.cpp
   ├──── MoveStruct.h
   ├──── MoveVisibilityEffect.cpp
   ├──── MoveVisibilityEffect.h
   ├──── PreUseStruct.cpp
   ├──── PreUseStruct.h
   ├──── PrimitiveEvent.cpp
   ├──── PrimitiveEvent.h
   ├──── PrimitiveMoveEvent.cpp
   ├──── PrimitiveMoveEvent.h
   ├──── Static.cpp
   ├──── Static.h
   ├──── TargetStruct.cpp
   ├──── TargetStruct.h
   ├──── Transform.cpp
   ├──── Transform.h
   ├──── Trigger.cpp
   ├──── Trigger.h
   ├──── UseStruct.cpp
   ├──── UseStruct.h
   └──── ValueChangeEvent.h
```

（3）

```
── Cards
   ├──── Card.cpp
  ├──── Card.h
   ├──── CardFilter.h
   ├──── CardInfo.cpp
```

```
├── CardInfo.h
├── CardType.cpp
├── CardType.h
├── EquipCard
│   ├── DefensiveHorse.cpp
│   ├── DefensiveHorse.h
│   ├── EightDiagram.cpp
│   ├── EightDiagram.h
│   ├── OffensiveHorse.cpp
│   ├── OffensiveHorse.h
│   ├── RenWangShield.cpp
│   ├── RenWangShield.h
│   └── WeaponCard
│   ├── Axe.cpp
│   ├── Axe.h
│   ├── CiXiongSword.cpp
│   ├── CiXiongSword.h
│   ├── CrossBow.cpp
│   ├── CrossBow.h
│   ├── EquipCard.cpp
│   ├── EquipCard.h
│   ├── Halberd.cpp
│   ├── Halberd.h
│   ├── IceSword.cpp
│   ├── IceSword.h
│   ├── QiLinBow.cpp
│   ├── QiLinBow.h
│   ├── Spear.cpp
│   ├── Spear.h
│   ├── WeaponCard.cpp
│   └── WeaponCard.h
├── NonEquipCard
│   ├── CounterCard.cpp
│   ├── CounterCard.h
│   ├── DelayedTrick.cpp
│   ├── DelayedTrick.h
│   ├── Dismantle.cpp
│   ├── Dismantle.h
│   ├── Divination.cpp
│   ├── Divination.h
│   ├── Dodge.cpp
│   ├── Dodge.h
│   ├── Duel.cpp
│   ├── Duel.h
│   ├── Friendship.cpp
│   ├── Friendship.h
│   ├── Harvest.cpp
│   ├── Harvest.h
```

```
   |   ├── Lightning.cpp
   |   ├── Lightning.h
   |   ├── NonEquipCard.cpp
   |   ├── NonEquipCard.h
   |   ├── Peach.cpp
   |   ├── Peach.h
   |   ├── RandomShot.cpp
   |   ├── RandomShot.h
   |   ├── Slash.cpp
   |   ├── Slash.h
   |   ├── SouthernIntruder.cpp
   |   ├── SouthernIntruder.h
   |   ├── Thievery.cpp
   |   ├── Thievery.h    | ├── TimeWalk.cpp
   | ├── TimeWalk.h     |   ├── WeaponGambit.cpp
   |   └── WeaponGambit.h
   ├── StarterDeck.cpp
   ├── Zone.cpp
   └── Zone.h

 (4)
── Event
   ├── Event.cpp
   ├── Event.h
   ├── EventBegin.cpp
   ├── EventBegin.h
   ├── EventEnd.cpp
├── EventEnd.h
   └── public class Event
   ├── CounterEvent.cpp
   ├── CounterEvent.h
   ├── DamageEvent.cpp
   ├── DamageEvent.h
   ├── DeathEvent.cpp
   ├── DeathEvent.h
   ├── DiscardEvent.cpp
   ├── DiscardEvent.h
   ├── DyingEvent.cpp
   ├── DyingEvent.h
   ├── HpReductionEvent.cpp
   ├── HpReductionEvent.h
   ├── JudgeEvent.cpp
   ├── JudgeEvent.h
   ├── MoveEvent.cpp
   ├── MoveEvent.h
   ├── NeedUseEvent.cpp
   ├── NeedUseEvent.h
   ├── NeedYieldEvent.cpp
```

```
    ├─── NeedYieldEvent.h
    ├─── PhaseEvent.cpp
    ├─── PhaseEvent.h
    ├─── RecoverEvent.cpp
    ├─── RecoverEvent.h
    ├─── ResolveEvent.cpp
    ├─── ResolveEvent.h
    ├─── TurnEvent.cpp
    ├─── TurnEvent.h
    ├─── UseEvent.cpp
    ├─── UseEvent.h
    ├─── YieldEvent.cpp
    └─── YieldEvent.h

  (5)
─── Player
    ├─── ClientPlayer.cpp
    ├─── ClientPlayer.h
    ├─── ConsolePlayer.cpp
    ├─── ConsolePlayer.h
    ├─── Player.cpp
    └─── Player.h

  (6)
─── SkillEffect
 ├─── RegisterSkillEffect.cpp
 ├─── RegisterSkillEffect.h
 ├─── RemoveSkillEffect.cpp
 ├─── RemoveSkillEffect.h
 ├─── SkillEffect.cpp
 └─── SkillEffect.h
```

---

c++类总结

类的从属关系：

Card:

```
class CardType --> {
class EquipCard;
class NonEquipCard;
}

class EquipCard --> {
class WeaponCard;
class DefensiveHorse;
class OffensiveHorse;
class EightDiagram;
```

```
class RenWangShield;
}

class WeaponCard ---> {
class Axe; "贯石斧"
class CiXiongSword; "雌雄双股剑"
class CrossBow; "诸葛连弩"
class Halberd; "方天画戟"
class IceSword; "寒冰剑"
class QiLinBow; "麒麟弓"
class Spear; "丈八蛇矛"
}

class Trigger ---> {
class AxeSkill;
......(The same as the one above + Skill)
class HarvestInit;
class HarvestCleanup;
}

class NonEquipCard ---> {
class CounterCard; "闪"
class DelayedTrick; "延时类：乐不思蜀，闪电"
class Dismantle; "过河拆桥"
class Divination; "无中生有"
class Duel; "决斗"
class Friendship; "桃园结义"
class Harvest; "五谷丰登"
class Peach; "桃"
class RandomShot; "万箭齐发"
class Slash; "杀"
class SouthernIntruder; "南蛮入侵"
class Thievery; "顺手牵羊"
class WeaponGambit; "借刀杀人"
}
```

Base of Event:

```
class PrimitiveEvent --> {
class PrimitiveMoveEvent;
class EventBegin;
class EventEnd;
class RegisterSkillEffect;
class RemoveSkillEffect;
class ValueChangeEvent;
}
```

Event:

```
class Event ---> {
class TurnEvent;
```

```
class PhaseEvent;
class UseEvent;
class YieldEvent

class MoveEvent;
class NeedUseEvent;

class CounterEvent;
class Judgement;
class DamageEvent;
class HpReductionEvent;
class DyingEvent;
class DeathEvent;
class RecoverEvent;
class ResolveEvent;
class DiscardEvent;
}
```

几个重要的类：
Game *game 是控制整个游戏的类，在Game.cpp建立，在最重要的头文件 “Lib.h” 中被声明为 extern Game *game;
其它文件多通过 #include “Lib.h” 或间接包含该头文件，在Game被构造的同时，game被设置为指向该新建的Game类，从而 *game 在全局都可以访问。随时可以通过game调用玩家人数，当前玩家，各牌区。。。理论上所有游戏内参数。包括：

```
int nPlayer;
vector<CardType*> cardTypes;
GameValue<int> curPlayer;
GamePointer<TurnEvent> curTurn;
vector<Player*> players;
Zone deck,discardPile,resArea;
vector<PrimitiveEvent*> timeline;
vector<Event*> stack;
vector<Static*> staticTable[nStaticValues][3];
```

Card类及其延伸包括：class Card, class class CardInfo, class CardType, class Zone.
一个排区内的牌，从Zone->next开始到Zone->prev，以链表形式存储Card * ，该链表是双向循环链表. Card类主要包含：

```
Zone* zone;
Card* prev;
Card* next;
vector<SkillEffect*> skills;
Card(CardInfo *c,Card *p);
virtual ~Card();
void erase();
void insert(Card *p);
void move(Card *p);
```

```
bool isVisibleTo(Player *p);
string toString(Player *p);
```

控制角色的类：class ClientPlayer（Player最基础的类） –> class ConsolePlayer，class Player虽不是class ClientPlayer的子类，但基本来自ClientPlayer。ConsolePlayer是最重要的类之一。主要函数有:

```
virtual void inform(string message);
int getChoice(string reason, const vector<string> &choices);
vector<int> getMultiChoice(string reason, const vector<string> &choices,
int max, const vector<bool> &mandatory);
```

游戏进程中最重要的事件一半写在了ConsolePlayer.cpp中，最重要的是函数

void ConsolePlayer::inform(string message);
int ConsolePlayer::getChoice(string reason, const vector &choices)

每回合，角色通过inform函数获取信息，对于人类玩家，inform函数还负责通过Game Message输出信息。
每回合角色可进行的操作显示和选择，通过getChoice函数进行。

游戏进程总结

游戏的进程，主要通过两个文件控制：ConsolePlayer.cpp, Game.cpp.
Game.cpp中的构造函数负责构造所有初始参数，并调用TurnEvent::execute(), 根据从Event继承来的onTiming(),execute()会先进入onTiming(beforeTurn)再进入onTiming(beginTurn)，随后调用各阶段的PhaseEvent (new PhaseEvent(player,phase))->happen()，随后进入onTiming(endTurn)和onTiming(afterTurn)。PhaseEvent会根据各种情况来决定调用具体事件中多态化的happen()函数。具体的事件类，可参考
类的从属关系中的Event。
而每个事件发生后与各玩家的交互，由ConsolePlayer负责，包括AI。信息的具体输入输出由一个Qt的类 CGlobal中的函数CGlobal::QOUT(string)实现。

可视化总结

我们学习了Qt中的一些部件，程序中主要使用了以下部件：

QLabel
QPushButton
QTextEdit
QInputDialog
QScrollArea

与图片有关的部分则使用了：

QPixmap
QPainter
QPalette
QPen
QFont

与可视化有关的部分，大部分都写进了mainwindow.h & mainwindow.cpp中。
在mianwindow.h中定义了一个类class CGlobal，在其中加入了我们需要的可视化widget和一些全局变量。

```
class CGlobal
{
public:
CGlobal();
~CGlobal();
public:
static MainWindow * ww;
static int inputtemp ;
static bool inputbutton_clicked ;
static void QOUT(std::string a) {
QString qs = QString::fromStdString(a);
CGlobal::ww->textedit->insertPlainText(qs);
CGlobal::ww->textedit->moveCursor(QTextCursor::End);
}
static int CurrentPlayer ;
static int HumanPlayer ;
static QLabel * otherplayerzonelabel[7];
static QLabel * HumanplayerHP[5];
static QLabel * otherplayerHP[7][5];
static QPixmap * deepgreen;
static QPixmap * green;
static QPixmap * yellow;
static QPixmap * orange;
static QPixmap * red;
static QPixmap * smalldeepgreen;
static QPixmap * smallgreen;
static QPixmap * smallyellow;
static QPixmap * smallorange;
static QPixmap * smallred;
static QLabel Cur[7];
static QLabel * rebelwin;
```

```
static QLabel * lordwin;
static QLabel * renegadewin;
static QPixmap * generalpix[25];
static QLabel * otherplayergeneral[7];
static QLabel Dead;
static QLabel * Humangeneral ;
static QLabel * otherplayernumber [8] ;
static QLabel * humanplayernumber [8] ;
static QLabel * roleslabel[8];
static QPixmap * rolespix[4];
static QTextEdit * cardnum[7];
static int CurrentTurn;
static int AutoOrNot;
};
```

具体可视化的效果见游戏或ppt