## 一、纸牌游戏功能、目标分析、描述：

**Windows 纸牌游戏布局如下图：**


**功能（游戏玩法）：**

这个游戏使用一副标准的 52 张纸牌。牌桌由 28 张分为 7 堆的纸牌构成。第 1 堆是 1 张纸牌，第 2 堆是 2 张纸牌，以此类推，一直到第 7 堆。每堆纸牌的最上面一张纸牌正面朝上，其余的纸牌则背面朝上。

几个花色堆(有时称为基牌)由从 A 到 K 的不同花色的纸牌组成。它们组成了上面的几
个牌堆，准备在游戏中使用。游戏的最终目标就是将 52 张纸牌组合成各种花色堆。

没有作为牌桌一部分的其余的所有纸牌最初叠放成一堆纸牌(待用堆)。这堆纸牌背面朝
上，可以一张一张地依次取走，正面朝上的放于丢弃堆中。对于丢弃堆的最上面纸牌，可以移动到桌面堆中或者花色堆中。纸牌从待用堆中逐张抽取，直至取空，此时，如果无法进一步移动纸牌，游戏就结束了。

只有与前一张纸牌颜色相反且点数增 1 的纸牌才可以放于桌面堆的最上面。
只有花色相同且点数增 1 的纸牌或者花色堆为空时的 A 纸牌，才可以放于花色堆。
在游戏过程中所产生的桌面堆中的空堆只能由 K 纸牌来填充。
每个桌面堆最上面的纸牌和丢弃堆最上面的纸牌在游戏中总是可以移动的。
当一个牌堆的正面朝上的最下面那张纸牌与目的堆的最上面纸牌之间符合移动规则时(颜色相反，点数增 1)，可以将前面所述的那个牌堆(称为“构成”

(build))的一整套正面朝上的纸牌移动到目的堆,只有这时才允许移动多张纸牌。
桌面堆最上面的纸牌总是正面朝上的。如果一张纸牌从桌面堆移走,则原牌堆的那张背面朝上的最上面的纸牌就可以翻转成正面朝上。

## 二、设计思路

分析纸牌游戏中涉及到的对象，牌、牌堆、游戏面板、以及容纳面板的框架。将其抽象为类，并分析每个类应当具有的行为。

### 牌：

用于创建可视界面的技术已经经历了很多次演变,并且这种趋势还将继续下去。正是因为这种原因，将包含数据值的类（数据类）与用于提供这些值的图形显示的类（视图类）进行分离很有用。这样，就可以根据需要随时修改或替换负责界面显示的类，而保持原来的数据类不变。

**牌类（Card）：**

简单的包含数据与视图两个属性。

**数据类（PlayingCard）：**

类 PlayingCard 应当包含纸牌的数据：点数、花色，以及获取点数与花色的方法。

在软件开发过程中，应该尽可能地创建通用的可复用的类，这些类对于环境条件的要求最小，因此可以从一个应用程序移植到另外一个应用程序。因为类 PlayingCard 仅包含纸牌的数据，不包含任何与所开发的应用程序相关的信息，因此可以很容易地从一个程序迁移到另外一个使用纸牌抽象的程序。

**视图类（CardView）：**

类 CardView 中应当包含的属性是：牌的正反面分别要显示的图像，一个 bool 类型的值（表明当前牌显示正面还是反面），牌的长宽，牌的类型（表明当前牌所在的堆的类型：花色堆或桌面堆等），牌的容器（表明当前牌所在的具体的堆，比如，当前牌在桌面堆，那通过该属性可以知道当前牌在桌面堆七个堆中的哪一个），一个 CardView 指针（指向当前牌在当前堆的下一张牌）。

包含的方法：获取各种属性的方法，绘制牌的方法，以及最重要的：处理鼠标点击以及拖动牌的方法。

### 牌堆——使用继承：

在游戏中，与牌堆相关的许多行为，对于各种牌堆都是公共的，如添加、删除牌的操作，除此之外，每种堆又有自己特有的操作，在这里使用继承来实现，父类 CardPile 给出所有堆共有操作的实现，以及一些其它操作的缺省行为，但这些行为也可以在子类中改写。

**父类 CardPile:**

包含的属性：堆显示的位置，堆显示的长宽，栈（用于容纳当前堆中的所有纸牌，因为所有的堆每次增加或删除牌都是对栈中最上面的牌操作，符合栈的特

性，所以用栈来容纳）
包含的方法：获取当前堆的位置，增加、删除牌，获取堆中最上面的牌，获取堆中牌的数量，判断堆是否为空

**待用堆(DeckPile)：**

DeckPile 包含所有待取用的纸牌，它应当处理鼠标的点击事件，每点击一次则取出一张牌，正面向上显示在丢弃堆（DiscardPile），且在堆为空时应当将废弃堆的牌全部拿来重新等待用户点击。

**丢弃堆（DisCardPile）：**

丢弃堆只需接受来自待用堆的牌，将其在最上层正面向上进行显示，并在待用堆牌的数量为零时，将自己所容纳的牌全部还给待用堆。
丢弃堆只允许拖动最上面的一张牌。

**桌面堆（TablePile）：**

首先，不论何时，桌面堆最上面的牌一定是正面向上显示。
桌面堆中除反面向上的牌，其余均允许拖动，但每次拖动，从点击的那张牌到牌堆最上面的牌都会被拖动。
桌面堆可以接受来自其它牌堆的牌，当其它堆的牌拖动到当前桌面堆时，要判断这次拖动是否合法（若牌堆为空，则只接受点数为 K 的牌；若牌堆不空，则只有颜色不同，且拖动来的牌比当前堆最上面的牌点数小 1，才合法），合法则接受拖动的牌，不合法拖动的牌会返回原位置。

**花色堆（SuitPile）：**

花色堆只允许拖动最上面的牌。
花色堆也可以接受来自其它牌堆的牌，当其它堆的牌拖动到当前花色堆时，同样要判断拖动是否合法（若牌堆为空，可以接受任意点数为 1 的牌；若不空，则拖动的牌既要与堆中所有牌花色相同，而且点数要刚好比堆中最上面的牌点数大 1），合法则接受拖动的牌，不合法拖动的牌会返回原位置。

## 游戏面板（GamePanel）：

一方面显示牌及牌堆，另一方面是为了方便重新开始一局游戏（重新开始游戏只需将原有的面板销毁，重新 new 一个面板即可）。
除以上，最重要的是控制游戏的逻辑（根据玩家的操作，将牌放在合适的位置）。在游戏刚开始时，初始化牌堆及 52 张牌，并将牌放到正确的位置。之后就是不断的根据用户的操作及游戏的规则，不断改变牌的位置。

## 容纳游戏面板的框架（MainFrame）：

包含一个游戏面板，以及一个控制游戏“重新开始”的按钮，当用户按下按钮时，即销毁原有的面板，重新 new 一个面板。

将游戏中的对象抽象为类，以及分析清楚每个类应当具有的属性及行为，接下来就是代码的实现，然后将每个类联系在一起，实现整个游戏的逻辑。

## 胜利判定：

游戏的最终目标是将 52 张纸牌组合成各种花色堆，但是当待用堆与丢弃堆牌的数量均为 0 且桌面堆所有的牌都翻过来（均正面向上）时，一定可以达成最终的目标。所以只需在每次待用堆或丢弃堆牌的数量为 0 或某一桌面堆在没有反面向上的牌时进行判定即可，判定的条件如上，判定成功则玩家胜利，判定失败则游戏继续。

## 三、设计方案：

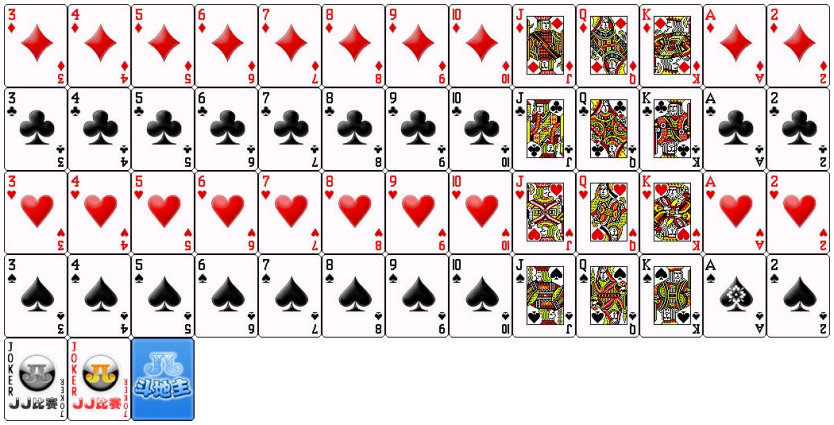

UML.pdf

**UML 类图（见 pdf）：**

**关键代码说明：**

### 1、初始化牌

下面的 init 方法，主要是初始化 52 张牌，每张牌的图像都是来源于如下的图片，所以下面的方法最主要的其实是利用 qt 中的 copy 方法对如下图片进行裁剪，然后匹配对应的数据，从而完成对 52 张牌的初始化。
下面的两个 connect 方法是利用 qt 的信号机制将信号与槽函数连接。具体实现的就是拖动的过程由每张牌自身的拖动（drag）事件来实现，但拖动结束时，会给游戏面板发拖动结束的信号，然后由游戏面板（GamePanel）来处理拖动的结果。第一个 connect 是 GamePanel 用于确定玩家拖动的是哪张牌，第二个 connect 是接收牌发出的拖动结束的信号，然后处理拖动的结果。



```
void GamePanel::init(){
  QPixmap cardsPic(":/res/card.png");
  QSize m_cardSize = QSize(80, 105);
  QPixmap m_cardBackPic(":/res/back.png");
  for (int suit = Suit_Begin + 1, i = 0; suit < Suit_End; suit++, i++)
  {
      for (int pt = Card_Begin + 1, j = 0; pt < Card_SJ; pt++, j++)
      {
```

```
            Card mycard;
            PlayingCard tmp;
            tmp.set((CardSuit)suit,(CardPoint)pt);
            QPixmap pic = cardsPic.copy(j * m_cardSize.width(), i *
m_cardSize.height(), m_cardSize.width(), m_cardSize.height());
            CardView *myview=new CardView(pic,m_cardBackPic);
            myview->hide();
            myview->setParent(this);
            mycard.card=tmp;
            mycard.view=myview;
            myview->aCard=tmp;

connect(myview,&CardView::NotifyDragEvent,this,&GamePanel::OnDragEvent);

connect(myview,&CardView::NotifyDropEvent,this,&GamePanel::OnDropEvent);
            vec.push_back(mycard);
        }
    }
}
```

## 2、打乱牌

如下代码实现的是在游戏未开始时，将牌打乱随机添加到待用堆。初始化牌时，52张牌均被放在一个可变长数组 vec 中，下图 for 循环中 i 即每次循环时，当前 vec 中元素的数量，进入 for 循环后会产生一个大于等于 0 小于 i-1 的随机数 n，然后将 vec 中下标为 n 的元素添加到待用堆中，之后将该元素移除。如此将牌的顺序打乱。

```
void GamePanel::init_DeckPile(){
    qsrand(time(NULL));
    int n = qrand() % 5;
    for(int i=52;i>0;i--){
        int n=qrand()%i;
        m_deckpile->addCard(vec[n]);
        vec.remove(n);
    }
}
```

## 3、在 GamePanel 中处理拖动的结果（“接收方为花色堆的牌”与“接收方为桌面堆的牌”类似，所以没有详细说明）

```
void GamePanel::OnDropEvent(CardView *me){
    //判断玩家将牌拖动到了哪
    if(me->type==1){//接收方为桌面堆的牌
        int i=me->container;//接收方为桌面堆 7 个牌堆中的哪个牌堆
```

```
if(m_tablepile[i]->tfAdd(dragview->aCard.getsuit(),dragview->aCard.getpoint
())){//判断拖动的牌是否可以被该桌面堆接受
            int x=m_tablepile[i]->getPos().x();//之后牌将显示的横坐标
            int
y=m_tablepile[i]->getPos().y()+(m_tablepile[i]->getcnt())*30;//之后牌将显示
的纵坐标
            if(dragview->type==0){//来自待用堆的牌
                dragview->move(x,y);//将牌移动到指定位置
                dragview->raise();//将牌显示在最上层，不会被其它牌遮挡
                Card temp=m_discardpile->getCard();
                m_discardpile->deleteCard();//将拖动的牌在原有牌堆中删除
                m_tablepile[i]->LinkNext(temp);//将牌链接到当前桌面堆最上面
一张牌之后
                m_tablepile[i]->addCard(temp);
            }
            else if(dragview->type==2){//来自花色堆的牌
                dragview->move(x,y);
                dragview->raise();
                Card temp=m_suitpile[dragview->container]->getCard();
                m_suitpile[dragview->container]->deleteCard();
                m_tablepile[i]->LinkNext(temp);
                m_tablepile[i]->addCard(temp);
            }
            else if(dragview->type==1){//来自桌面堆的牌
                //来自桌面堆的牌的特殊之处在于，拖动来的牌可能不止一张
                int cnt=1;CardView *nxt=dragview->next;
                while(nxt!=NULL){//计算拖动的牌的数量
                    nxt=nxt->next;
                    cnt++;
                }
                QStack<Card> stack;
                for(int j=0;j<cnt;j++){//将拖动的牌在原有的堆中中删除，并按
顺序压到栈 stack 中

stack.push(m_tablepile[dragview->container]->getCard());
                    m_tablepile[dragview->container]->deleteCard();
                }
                if(!m_tablepile[dragview->container]->isEmpty()){//判断拖动
的牌原本所在的堆是否为空

if(!m_tablepile[dragview->container]->getCard().view->isEnabled()){//判断原
本所在的堆当前最上面的牌是否背面朝上
                        m_tablepile[dragview->container]->back_card--;
```

```
                    }

m_tablepile[dragview->container]->getCard().view->setDisabled(false);//设置
最上面的牌可用

m_tablepile[dragview->container]->getCard().view->setside(true);//设置最上
面的牌显示正面
                    if(m_tablepile[dragview->container]->back_card<=0)
onTestWin();//若原本所在桌面堆牌的数量为0，则测试玩家是否取得本局胜利
                    qDebug()<<m_tablepile[dragview->container]->back_card;
                }
                while(!stack.empty()){//将拖动的牌移动到正确的位置，并按顺序
链接、添加到对应的堆上
                    Card it=stack.top();
                    stack.pop();
                    it.view->move(x,y);y+=30;
                    it.view->raise();
                    m_tablepile[i]->LinkNext(it);
                    m_tablepile[i]->addCard(it);
                }
            }
        }
    }
    else{//接收方为花色堆的牌
        int i=me->container;

if(m_suitpile[i]->tfAdd(dragview->aCard.getsuit(),dragview->aCard.getpoint()
)){
            if(dragview->next!=NULL) return;
            dragview->move(m_suitpile[me->container]->getPos());
            dragview->raise();
            if(dragview->type==1){
                Card temp=m_tablepile[dragview->container]->getCard();
                m_tablepile[dragview->container]->deleteCard();

                if(!m_tablepile[dragview->container]->isEmpty()){

if(!m_tablepile[dragview->container]->getCard().view->isEnabled()){
                        m_tablepile[dragview->container]->back_card--;
                    }

m_tablepile[dragview->container]->getCard().view->setDisabled(false);

m_tablepile[dragview->container]->getCard().view->setside(true);
```

```
                    if(m_tablepile[dragview->container]->back_card<=0)
onTestWin();
                    qDebug()<<m_tablepile[dragview->container]->back_card;
                }

                m_suitpile[me->container]->addCard(temp);

            }
            else if(dragview->type==0){
                Card temp=m_discardpile->getCard();
                m_discardpile->deleteCard();
                m_suitpile[me->container]->addCard(temp);
            }
        }
    }
}
```

## 4、测试玩家是否胜利

当待用堆与废弃堆牌的数量均为 0，且七个桌面堆中所有的牌均正面向上，则玩家取得胜利

```
void GamePanel::onTestWin(){
    if(m_deckpile->getcnt()==0&&m_discardpile->getcnt()==0){// 待用堆与废弃堆牌的数量均为 0
        bool tf=true;
        for(int i=1;i<=7;i++){
            if(m_tablepile[i]->back_card>0) tf=false;
        }
        if(tf){//提示玩家取得胜利
            QMessageBox box;
            box.setWindowTitle("提示");
            box.setIcon(QMessageBox::Warning);
            box.setText("恭喜您取得本局胜利!");
            box.setStandardButtons(QMessageBox::Yes);
            box.exec();
        }
    }
}
```

## 5、CardView 处理拖动

```
void CardView::mousePressEvent(QMouseEvent *event)
{
    mStartPoint = event->pos();//拖动的起点
}
```

```
void CardView::mouseMoveEvent(QMouseEvent *event)
{
    if ((event->pos() - mStartPoint).manhattanLength() >
QApplication::startDragDistance())//判断是否执行拖动
    {
        QDrag *drag = new QDrag(this);
        QMimeData *mimeData = new QMimeData;
        drag->setMimeData(mimeData);//设置数据

        QSize picSize(width,height);
        QPixmap scaledPixmap = front.scaled(picSize,
Qt::KeepAspectRatio);

        drag->setPixmap(drawPixMap());//设置拖动过程中显示的图像

        emit NotifyDragEvent(this);//向 GamePanel 发出拖动开始的信号，
用于告诉 GamePanel 当前拖动的是哪张牌

        drag->exec(Qt::MoveAction);//开始执行拖动
        delete drag;

        CardView *nxt=next;this->show();//拖动结束后，所有拖动的牌重新
显示
        while(nxt!=NULL){
            nxt->show();
            nxt=nxt->next;
        }

    }
}

QPixmap CardView::drawPixMap(){//绘制拖动过程中显示的图像（因为如果拖
动的是桌面堆的牌，则可能拖动不止一张牌，所以要进行绘制）
    int cnt=1;
    CardView *nxt=next;this->hide();//拖动过程中会显示拖动的牌的图像，
则原图像隐藏
    while(nxt!=NULL){
        nxt->hide();
        nxt=nxt->next;
        cnt++;
    }
    QSize size(width,height+(cnt-1)*30);
    QPixmap res(size);
```

```
    QPainter painter(&res);
    nxt=this;
    for(int i=0;i<cnt;i++){
        painter.drawPixmap(0,i*30,width,height,nxt->front);
        nxt=nxt->next;
    }
    return res;

}
//拖动的代码
void CardView::dragEnterEvent(QDragEnterEvent *event)
{
        event->acceptProposedAction();
}

void CardView::dragMoveEvent(QDragMoveEvent *event)
{
        event->acceptProposedAction();
}

void CardView::dropEvent(QDropEvent *event)
{
    if(type==1||type==2)//只有桌面堆与花色堆可以接收来自其它堆的牌
    {
        emit NotifyDropEvent(this);//向 GamePanel 发出拖动结束的信号，
传递的参数是当前鼠标所在的牌，通过这个参数可以得知玩家将牌拖动到了哪里
    }
}
```