程序设计实习大作业实验报告

艾逸文	李磊	李享
物理学院	信息科学技术学院	物理学院
2100011301	2100011470	2100011451

2023 年 7 月 8 日

目录

1 程序功能介绍

1.1 程序目的

该程序为一款游戏，模仿最近比较热门的“羊了个羊”微信小游戏。实现了羊了个羊的大部分功能，自行选择游戏难度等其他功能以增加游戏的趣味性。为用户提供休闲娱乐，益智以及社交的平台。

1.2 程序功能

(1) 主界面可以选择游戏难度。
(2) 洗牌道具可以将所有卡牌打乱并在随机位置重新生成卡牌
(3) 撤回道具可以将卡槽中最右侧的卡牌放回点击之前的位置
(4) 游戏上方有进度条以提示游戏完成度
(5) 游戏中有退出游戏功能，退出游戏前需要玩家确认退出游戏

1.3 程序规则

(1) 用户选择游戏中使用的图标，选择游戏难度。
(2) 开始游戏之后用户可以点击最上层的图标并加入卡槽中。
(3) 当下层图标上方覆盖的图标均被移除后下方图标变为最上层图标。
(4) 卡槽中出现三个相同的卡牌即可消去这三个相同的卡牌。
(5) 如果卡槽已满，则游戏失败。如果所有卡牌均被移除，则游戏胜利。
(6) 游戏过程中可以选择道具，不同道具有不同的功能。

2 项目各模块与类设计细节

在实现本程序时, 我们设置了如下几个类:
class Card; class Game; class Slot; class GameOverBox;
class ConfirmBox; class Menu; class Bar; class Hyperlink;
接下来依次介绍.

2.1 Card: 卡片类

class Card 是 QPushButton 的子类, 所有在本游戏中可以点击, 以及已消除的卡片都属于这个类.

class Card 有如下的成员变量:

```
int posx, posy, orix, oriy;    // 记录一开始生成的位置以及当前所在的位置
enum CardType type;            // 代表这张卡片的类型
int in_heap;                   // 卡片是否在堆当中
std::vector<Card *> covering;  // 这张卡片所覆盖的其它卡片
std::vector<Card *> covered;   // 这张卡片被哪些其它卡片所覆盖
Game * game;                   // 指向游戏状态的一个指针, 用来和游戏进行交互
```

其中, 成员变量 type 是一个 CardType 枚举类型, 包括 ClickableCard(表层可点击的卡片), CoveredCard(被覆盖的卡片), SlotCard(槽中的卡片), EliminatedCard(已消除的卡片) 四种.

class Card 有一些成员函数来维护对应的卡片的状态, 以下列举出其中比较重要的成员函数, 以及一些与 Card 类相关的全局函数. 这些函数中多处涉及卡牌图片的明暗切换, 实现方法为: 每种卡牌都有两个亮度的图片, 每次切换时需重新设置按钮的样式表 (setStyleSheet() 函数) 更改贴图.

```
Card::Card(...);
/*
class Card的构造函数. 初始化卡片的位置, 名称等信息; 设定卡片类型为
ClickableCard; 将卡片的基类QPushButton用参数parent(指向卡片所处
界面的Qdialog *指针)初始化, 也就是将卡片与游戏界面绑定; 设置卡片的
序列号; 调用setup_card()实现卡片的显示; 检查已有卡片是否有被本卡片
遮盖的, 并调用cover_card()设置覆盖关系.
*/

static void setup_card(...);
/*
调用QPushButton::setGeometry()设置卡片的位置和大小; 根据卡片名称
设置相应的背景图片; 调用show()使按钮出现.
*/

void remove_upper_card(Card * upper_card);
/*
对一张被盖住的卡片, 删掉盖在其上面的某张卡upper_card. 具体操作:
在covered中查找并删除upper_card这个元素; 若删除后covered为空,
则使当前卡片变亮且可点击.
*/

void Card::remove_lower_card(Card * lower_card);
```

```
/*
当某卡牌(this)压住的另一张牌(lower_card)被破坏时会调用此函数,
函数会在covering中查找并删除lower_card.
*/

void remove_card(void);
/*
当玩家点到一张卡牌时会调用此函数, 函数会遍历cover, 并对其中的
卡片都调用remove_upper_card(), 随后将covering清空.
*/

bool overlap(Card * old_card, Card * new_card)
{
    return abs(old_card -> posx - new_card -> posx) < CARD_SIZE
        && abs(old_card -> posy - new_card -> posy) < CARD_SIZE;
}
/*
判断先后生成的两张卡牌是否具有叠放关系, 供其他函数使用.
*/

void cover_card(Card * upper_card, Card * lower_card);
/*
让upper_card盖住lower_card. 具体来说: 在upper_card->covering
中增加lower_card; 在lower_card->covered中增加upper_card;
将lower_card->type设为CoveredCard, 并使其变暗, 禁止点击.
*/
```

2.2 Game: 游戏状态类

class Game 记录整个游戏的所有信息, 程序的主要逻辑在这个类中实现. class Game 是 QDialog 的子类, 有如下的成员变量

```
Ui::Game * ui;                    // 用于实现界面的GUI
std::vector<Card *> all_cards;    // 记录本局游戏的所有卡片
Slot * slot;                      // 本局游戏的卡槽
Bar * move, * cover;              // 进度条指针
int cards_clickable, cards_in_slot, cards_eliminate;
            // 分别为可点击卡片, 槽中卡片, 已消除卡片
int card_nums, card_types, cards_in_heap;
            // 分别为总卡片数, 卡片种类数, 每个堆初始的卡片种类数
int length, width, max_num_card;
            // 分别为界面长度, 宽度, 每行每列放的最多卡片数
int shuffle_left,retreat_left,crash_left;
            // 三种道具剩余可使用次数
```

class Game 有一些成员函数来维护游戏状态, 以下列举出其中比较重要的成员函数:

```
explicit Game(...);
/*
class Game的构造函数(相关的参数会在之后介绍主菜单时介绍), 初始化相关的成员
变量, 背景, 进度条, 然后生成卡片, 首先生成堆中的卡片, 然后再生成其它的卡片.
在生成卡片的同时确定覆盖关系, 将卡片和update函数(接下来会介绍的函数)连接,
并且允许最上层的卡片被鼠标点击.
*/

void on_confirmBox_clicked(void);
/*
该函数是返回按钮, 在点击返回按钮时, 会跳出一个窗口让玩家选择是返回主菜单还是
继续游戏.
*/

void on_retreat_clicked(void);
/*
该函数是撤回按钮, 在点击撤回按钮时, 卡槽当中的最后一张卡片会被移动到它最开始
生成的时候的位置. 在卡槽没有任何卡片的时候不会有任何的效果
*/

void on_crash_clicked(void);
/*
该函数是破坏按钮, 在点击破坏按钮时, 会选择三张同样类型的卡片直接消除, 不论
它是否被其它的卡片覆盖. 卡槽和堆当中的卡片不会被破坏.
*/

void update(Card * chosen);
/*
该函数是状态更新函数, chosen是被点击的卡片, 它更新game当中的状态和进度
条, 并且把卡片插入槽中.
*/

void update_tail(void);
/*
该函数也是状态更新函数, 与上一个函数分开是因为要保证卡片先移动到槽中再进行
下一步的操作, 该函数在slot.cpp中的insert_card函数中调用. 它检查卡片移
入槽中之后是否有三张同样的卡牌, 如果有, 就直接消除. 如果卡槽满了, 就弹出
fail窗口, 如果所有卡片都消除了, 就弹出win窗口.
*/
```

同时, 为了确保程序的正确性, 我们还写了一些一致性检查函数以确保每次状态的更新都是自洽的, 他们分别是 class Game, class Card, class Slot 的成员函数 void Game::consistency_check(void), void Slot::slot_concheck(int

cards_in_slot), void Card::cardconcheck(void); 在每次状态更新 (比如在每次点击一张卡片或者使用特殊道具的时候) 的时候调用, 确保状态的转移是正确的.

2.3 Slot: 卡槽类

class Slot 维护被点击并加入卡槽的卡牌

class Slot 有如下的成员变量:

```
int size;                       // 卡槽允许的最大容量
int curr_size;                  // 当前卡槽中的卡牌
Game * game;                    // 指向游戏状态的指针, 用于与游戏交互
std::vector<Card *> cards;      // 指向卡槽中卡牌的指针
```

class Slot 有一些成员函数来实现点击卡牌之后对卡槽状态的维护, 还有对游戏状态的判断。以下列出一些重要的成员函数

```
std::vector<Card *>::iterator Slot::find_slot(Card * card)
/*
该函数寻找新加入的卡牌应该插入的位置。如果卡槽中有与新加入卡牌种类相同的卡牌
, 则新加入的卡牌应该插入与之相同种类卡牌的后方; 如果卡槽中没有与新加入卡牌同
种类的卡牌, 则新加入的卡牌应该插入所有卡槽卡牌的后方。card是新加入的卡牌的
指针, 返回合适插入位置的iterator。
*/

std::vector<Card *>::iterator Slot::check_slot(void)

/*
该函数检查卡槽中是否有3个相同的卡牌。如果有3个相同的卡牌, 则返回这三个相同
卡牌第一个卡牌的iterator; 否则返回卡槽的end()
*/

void Slot::remove_cards(std::vector<Card *>::iterator card_it, bool win)
/*
该函数将卡槽中3个相同的卡牌消除, 并将剩余的卡牌重新左对齐。只有当卡槽中存在
3个相同种类卡牌是才会调用该函数。card_it是卡槽中3个相同种类卡牌中指向第一
个卡牌的iterator.
*/

void Slot::insert_card(Card * card, std::vector<Card *>::iterator place)
/*
该函数将一个新的卡牌插入到卡槽中。place为新卡牌应当插入到位置。
*/
```

2.4 GameOverBox: 游戏结束框类

游戏结束框是 QDialog 的子类, 在游戏结束的时候弹出, 在关闭该框或者点击该框的返回按钮时, 都会退回到主菜单中, 游戏结束框有失败和胜利两种情况. 失败当且仅当槽中的卡片有 7 张, 并且无法消除. 胜利当且仅当所有的卡片都被消除.

2.5 ConfirmBox: 确认退出框类

确认退出框是 QDialog 的子类, 在用户在游戏过程中点击返回按钮时弹出. 该框有两个选项, 一是继续游戏, 二是直接返回主菜单, 析构所有和游戏相关的对象. 直接关闭该窗口会继续游戏.

2.6 Menu: 菜单类

菜单类是 QDialog 的子类, 在游戏一开始运行时就会弹出. 主菜单有 5 个按钮, 其中一个是返回按钮, 点击这个按钮弹出一个确认退出框, 点击退出会退出游戏, 点击返回或者关闭会回到主菜单. 剩余 4 个按钮是四种模式, 分别为 easy, medium, hard, hell. 点击他们会进入对应难度的游戏界面.

下面的结构体代表了传给游戏类的参数:

```
struct GameParameters
{
    int cardNum;              // 总卡片数量
    int cardType;             // 总卡片类型数量
    int cardInHeap;           // 每个堆中卡片数量
    int numShuffle;           // 打乱道具可使用次数
    int numRetreat;           // 撤回道具可使用次数
    int numCrash;             // 破坏道具可使用次数
};


void Menu::startGame(const GameParameters& params)
{
    Game game(params.cardNum, params.cardType, params.cardInHeap,
              params.numShuffle, params.numRetreat, params.numCrash);
    game.exec();
     /* 游戏是通过exec函数运行起来的 */
}
```

2.7 Bar: 进度条类

进度条是 QPushButton 的子类, Game 中设有两个 Bar 类型的成员变量 - 色条和边框. 游戏开始时, 会先在界面上部生成边框的图片. 点击一张卡牌后, Game::update() 会根据卡牌数目增加色条的长度, 然后重新绘制边框; 点击撤回道具并且可撤回时, 也会缩短色条的长度, 然后重新绘制边框. 这样可实现" 边框始终覆盖住色条" 的效果.

2.8 Hyperlink: 超链接类

超链接类是 QDialog 的子类, 当游戏中道具使用次数用完之后并再次点击道具按钮时弹出. 点击看广告得道具按钮之后, 将自动打开指定的网页, 同时自动关闭超链接窗口, 游戏中获得额外的道具. 如果点击返回或者关掉超链接窗口, 则返回游戏, 但是不会获得额外的道具.

3 项目总结与反思

3.1 项目总结

本项目的实现逻辑比较简单, 但是要求设计一些互相联系紧密的类, 同时还要对不同的类进行维护. 在维护时, 代码编写者需要有清晰的认知, 保证所有的变量关系都是正确的, 尤其是在实现卡片覆盖关系的时候, 要仔细的维护指针之间的关系. 除此之外, 程序的各种细节也需要注意, 因为这会极大的影响游戏体验. 我们花了大量的时间做各种图片和音频, 调整图片之间的关系, 使得界面更加好看.

git 的使用也在我们的项目中起到了举足轻重的作用, 它帮助我们控制代码版本, 防止出现混乱, 让我们三人的团队合作变得更加顺利.

3.2 项目反思

反思: 我们经常会遇到一些因为数据维护的问题出现难以预测的问题导致程序崩溃, 而且的程序崩溃地方和错误开始的源头可能会相距很远, 在调试的时候会遇到一些困难. 之后我们采取了代码分块和一致性检查的办法将错误的源头修复. 如果在一开始的时候就能做好代码的模块化工作, 那我们会节省大量的时间.

4 小组分工

艾逸文 (组长): 负责项目的组织, 负责图形化界面的建立, 处理音频, 图像和超链接.

李磊: 实现游戏的核心逻辑 (class Game, class Slot, class Card) 同时撰写对应部分的实验报告

李享: 负责卡牌点击和运动，进度条, 音效等细节, 制作背景图片, 同时撰写对应部分的实验报告

5 参考文献

[1] https://houbb.github.io/games/ylgy/#/game

[2] https://github.com/liyupi/yulegeyu

[3] https://doc.qt.io/qt-6/stylesheet-reference.html