南京大学

本科教学课程报告

课程名称 数据库原理与应用

开课院系 信息管理学院

题　　目 一个基于数据库的卡牌类游戏设计

学生姓名 林定康 191820116

白雪童 191820003

陈文杰 191820019

帅奕航 191820172

授课教师 谈晓洁

目录

数据库设计报告

一、需求分析

（一）收集用户需求

本次大作业我们将设计一个基于数据库的卡牌类游戏。游戏的基本流程如下图所示，经过开始界面后，会有每次的战斗，和不同的人物进行 PK，成功后循环进入下一个战斗，失败则结束游戏；成功击败所有人物则闯关成功。这些部分，包括游戏的机制和前端等部分，我们将使用 python 来实现。

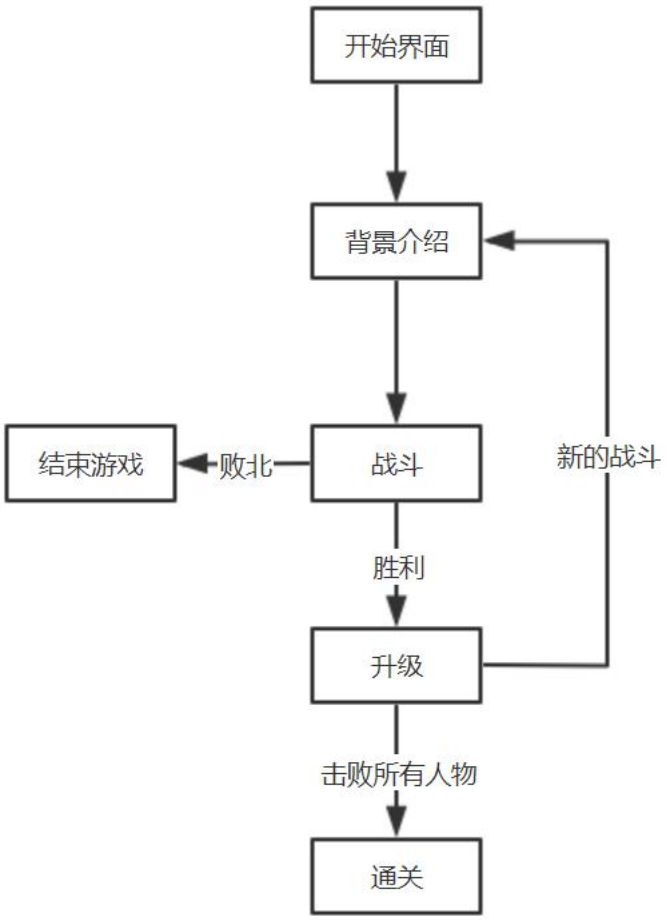

图 1 游戏基本流程图

通过数据库实现的部分主要在于对战斗的信息进行管理、查询和更新，这一部分是用户需求的重点。

打斗的规则有：

1. 玩家和人物打斗的过程中，玩家首先开始一轮回合，抽取三张卡牌，对自己或对方的的攻、防、血量等造成影响。三张卡牌的操作结束后，玩家回合结束，人物回合开始，同样抽取三张卡牌并进行操作。
2. 玩家和人物都具有各自的技能，这些技能是依附于人物和玩家一直起作用，区别于只在一个回合发生作用的卡牌。技能是对自己或对方的的攻、防、血量，或者对卡牌的属性造成影响。

3. 玩家和人物都具有各自的属性，属性与属性之间是相克的，属性被克制的人物的技能在战斗中失效。

4. 玩家和人物对战过程中，只要有一方的血量将为零则战斗结束，战斗结束后血量值不为零的一方获胜。

二、概念结构设计

（一）数据字典

1.数据流图

通过用户的需求描述，我们总结为如下的数据流图来描述一个战斗回合：

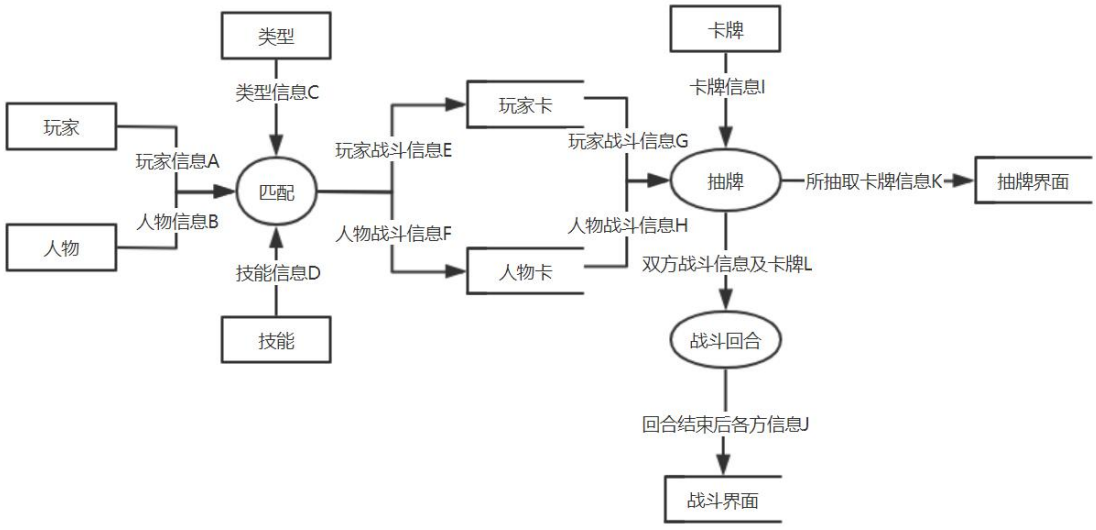

图 2 战斗回合数据流图

2.数据项描述

2.1 cno
数据项：cno
含义说明：唯一标识每个人物和玩家
别名：人物/玩家编号
类型：字符型
长度：8
取值范围：0 至 99999999
取值含义：目前为各个人物或玩家的出现次序的顺序号，0 在前为人物，反之为玩家
与其他数据项的逻辑关系：Cno->character, cno->hair, cno->attack, cno->defend, cno->level, cno->ctype，是 all-key 关系“人技匹配”的外键。

2.2 character
数据项：character
含义说明：人物的名字
别名：人物
类型：人物名

长度：20
取值范围：all
取值含义：人物的名字
与其他数据项的逻辑关系：cno->character

2.3 hair
数据项：hair
含义说明：血量
别名：发量
类型：整型
长度：int
取值范围：0-9999
取值含义：设定血量有限，但随着游戏开发，可能会提高上限
与其他数据项的逻辑关系：cno->hair, sno->hair, cardno->hair

2.4 attack
数据项：attack
含义说明：显示攻击值，数值是对目标的血量的减少值
别名：攻击值
类型：整型
长度：int
取值范围：0-9999
取值含义：设定攻击值有限，但随着游戏开发，可能会提高上限
与其他数据项的逻辑关系：cno->attack, sno->attack, cardno->attack

2.5 defend
数据项：defend
含义说明：显示防守值，数值是对攻击数值的减少值
别名：防守值
类型：整型
长度：int
取值范围：0-9999
取值含义：设定防守值有限，但随着游戏开发，可能会提高上限
与其他数据项的逻辑关系：cno->defend, sno->defend, cardno->defend

2.6 level
数据项：level
含义说明：显示级数，数值人物或者玩家的级别
别名：级别
类型：整型
长度：int
取值范围：0-9999
取值含义：设定级别数有限，但随着游戏开发，可能会提高上限
与其他数据项的逻辑关系：cno->level

2.7 ctype,restrict,restricted
数据项：ctype
含义说明：人物的类型属性
别名：类型/克制/被克制
类型：字符型
长度：2
取值范围：(“智”,”攻”,”防”)
取值含义：智代表智力型人物或玩家，防代表防御型人物或玩家，攻代表攻击型人物或玩家
与其他数据项的逻辑关系：cno->ctype

2.8 sno
数据项：sno
含义说明：唯一标识每一个技能
别名：技能号
类型：字符型
长度：8
取值范围：字母和数字
取值含义：标识号
与其他数据项的逻辑关系：sno->skill, sno->hair, sno->attack, sno->defend, sno->process, sno->object, 是 all-key 关系“人技匹配”的外键。

2.9 skill
数据项：skill
含义说明：技能的名字
别名：技能
类型：字符串
长度：20
取值范围：all
取值含义：技能的名字
与其他数据项的逻辑关系：sno->skill

2.10 process
数据项：process
含义说明：记录技能的各种值对对象产生作用的运算
别名：运算操作
类型：字符型
长度：2
取值范围：(“+”,”-”,”*”,”/”)
取值含义：+为加操作，-为减操作，*为乘操作，/为除操作
与其他数据项的逻辑关系：sno->process

2.11 object
数据项：object

含义说明：是卡牌操作的作用对象
别名：作用对象
类型：字符型
长度：20
取值范围：(“card”,”character”)
取值含义：card 是指对象为卡牌，character 是指对象为人物
与其他数据项的逻辑关系：sno->object

2.12 cardno
数据项：cardno
含义说明：卡牌号
别名：唯一标识卡牌
类型：字符型
长度：8
取值范围：0-9999
取值含义：标识卡牌
与其他数据项的逻辑关系：cardno->card, cardno->hair, cardno->attack, cardno->defend

2.13 card
数据项：card
含义说明：卡牌的名称
别名：卡牌名
类型：字符型
长度：20
取值范围：all
取值含义：卡牌的名称
与其他数据项的逻辑关系：cardno->card

3.数据结构描述：

3.1 人物
数据结构：人物
含义说明：定义了一个人物的有关信息
组成：cno, character, hair, attack, defend, level

3.2 玩家
数据结构：玩家
含义说明：定义了一个玩家的有关信息
组成：cno, character, hair, attack, defend, level

3.3 人物类型
数据结构：任务类型
含义说明：说明了玩家和类型的匹配关系
组成：cno, ctype

3.4 类型相克
数据结构：类型相克
含义说明：规定了几种人物类型之间的相克关系
组成：restrict, restricted

3.5 技能
数据结构：技能
含义说明：定义了一个技能的相关信息
组成：sno, skill, hair, attack, defend, process, object

3.6 人技匹配
数据结构：人技匹配
含义说明：规定了人物和技能之间的匹配关系，即什么人物拥有什么技能
组成：cno, sno

3.7 人物卡牌
数据结构：人物卡牌
含义说明：定义了一个人物卡牌的相关信息，任务卡牌是人物可以抽取的卡牌
组成：cardno, card, hair, attack, defend

3.8 玩家卡牌
数据结构：玩家卡牌
含义说明：定义了一个玩家卡牌的相关信息，玩家卡牌是玩家可以抽取的卡牌
组成：cardno, card, hair, attack, defend

4.数据流：

4.1 玩家信息
数据流：A
说明：将玩家信息提供战斗匹配
数据流来源：玩家
数据流去向：匹配
组成：cno, character, hair, attack, defend, level

4.2 人物信息
数据流：B
说明：将人物信息提供战斗匹配
数据流来源：人物
数据流去向：匹配
组成：cno, character, hair, attack, defend, level

4.3 类型信息
数据流：C

说明：将类型和人物之间的匹配关系以及类型之间相克关系传递给战斗匹配
数据流来源：类型
数据流去向：匹配
组成：cno, ctype, restrict, restricted

4.4 技能信息
数据流：D
说明：将技能和人物之间的匹配关系以及技能的相关属性传递给战斗匹配
数据流来源：技能
数据流去向：匹配
组成：cno, sno, skill, hair, attack, defend, process, object

4.5 玩家战斗信息
数据流：E，G
说明：是玩家和技能匹配以及自己的类别信息
数据流来源：匹配
数据流去向：玩家卡
组成：玩家(cno, character, hair, attack, defend, level)，技能(sno, skill, hair, attack, defend, process, object), ctype, restricted

4.6 人物战斗信息
数据流：F，H
说明：是人物和技能匹配以及自己的类别信息
数据流来源：匹配
数据流去向：人物卡
组成：人物(cno, character, hair, attack, defend, level)，技能(sno, skill, hair, attack, defend, process, object), ctype, restricted

4.7 卡牌信息
数据流：I
说明：人物卡牌和玩家卡牌的各项信息
数据流来源：卡牌
数据流去向：抽牌
组成：cardno, card, hair, attack, defend

4.8 双方战斗信息及卡牌
数据流：L
说明：双方当前的各个属性值、类型、技能的相关信息、所抽取卡牌的相关信息
数据流来源：抽牌
数据流去向：战斗回合
组成：人物(cno, character, hair, attack, defend, level)，玩家(cno, character, hair, attack, defend, level)，技能(sno, skill, hair, attack, defend, process, object), ctype,restricted，卡牌(cardno, card, hair, attack, defend)

4.9 回合结束后各方信息
数据流：J
说明：战斗回合结束后，双方的各类属性值
数据流来源：战斗回合
数据流去向：战斗界面
组成：人物(cno, character, hair, attack, defend, level)，玩家(cno, character, hair, attack, defend, level)

4.10 所抽取卡牌信息
数据流：K
说明：所抽取卡牌的各项信息
数据流来源：抽牌
数据流去向：抽牌界面
组成：cardno, card, hair, attack, defend

5.数据存储

5.1 玩家卡
数据存储：玩家卡
说明：记录玩家参与战斗时的各项属性
流入数据流：E
流出数据流：G
组成：玩家(cno, character, hair, attack, defend, level)，技能(sno, skill, hair, attack, defend, process, object), ctype, restricted
数据量：每个战斗开始前一张
存取方式：用 python 实现部分，呈现到界面上

5.2 人物卡
数据存储：人物卡
说明：记录人物参与战斗时的各项属性
流入数据流：F
流出数据流：H
组成：人物(cno, character, hair, attack, defend, level)，技能(sno, skill, hair, attack, defend, process, object), ctype, restricted
数据量：每个战斗开始前一张
存取方式：用 python 实现部分，呈现到界面上

5.3 抽牌界面
数据存储：抽牌界面
说明：记录双方所抽取的各三张牌的信息
流入数据流：K
流出数据流：无
组成：cardno, card, hair, attack, defend
数据量：每一个回合 3 张

存取方式：用 python 实现部分，呈现到界面上

5.4 战斗界面
数据存储：战斗界面
说明：记录一个回合结束之后的各方西信息
流入数据流：J
流出数据流：无
组成：人物(cno, character, hair, attack, defend, level)，玩家(cno, character, hair, attack, defend, level)
数据量：每个回合结束时两张
存取方式：用 python 实现部分，呈现到界面上

6.处理过程

6.1 匹配
处理过程：匹配
说明：对于玩家和人物的战斗信息进行匹配
输入：ABCD
输出：EF
处理：根据玩家表、人物表、人物类型表、类型相克表、技能表、人机匹配表，来匹配玩家和人物的战斗信息。

6.2 抽牌
处理过程：抽牌
说明：给回合开始方抽取三张卡牌
输入：GHI
输出：LK
处理：如果是玩家的回合，就从玩家卡牌随机抽取三张卡牌；如果是人物的回合，就从人物卡牌随机抽取三张卡牌

6.3 战斗回合
处理过程：战斗回合
说明：使用三张卡牌进行攻击、防御、加血操作等
输入：L
输出：J
处理：用卡牌的信息对双方的相对应的属性值进行运算

（二）概念结构模型

我们将概念抽象为六个实体：人物、人物卡牌、玩家、玩家卡牌、技能、类型。

根据需求分析，我们抽象出七个联系：

①人物抽取人物卡牌：是多对多的联系，一个人物可以抽取多个卡牌，一个卡牌可以被多个人物抽取。这个是 m:n 的联系。

②玩家抽取玩家卡牌：是多对多的联系，一个玩家可以抽取多个卡牌，一个卡牌可以被多个玩家抽取。这个是 m:n 的联系。

③人物属于某种类型：是多对一的联系，一个人物属于固定的一种类型，一种类型下可以包含多个人物。这个是 n:1 的联系。

④玩家属于某种类型：是多对一的联系，一个玩家属于固定的一种类型，一种类型下可以包含多个玩家。这个是 n:1 的联系。

⑤人物拥有某些技能：是一对多的联系，一个人物可以拥有多个技能，一种技能只能属于一个玩家。这个是 1:n 的联系。

⑥玩家拥有某些技能：是一对多的联系，一个玩家可以拥有多个技能，一种技能只能属于一个玩家。这个是 1:n 的联系。

⑦类型与类型相克：是一对一的联系，一种类型克制另一种类型，每一种类型也被一种类型克制。这个是 1：1 的联系。

在这个概念结构模型中，三种关系（一对一，一对多，多对多）都被应用到联系当中。

再将前面在数据字典中定义的数据项作为属性添加到相应的主体和联系上，特别注意的是 restrict 和 restricted 是在联系“相克”上的属性。得到的 E-R 图如下：

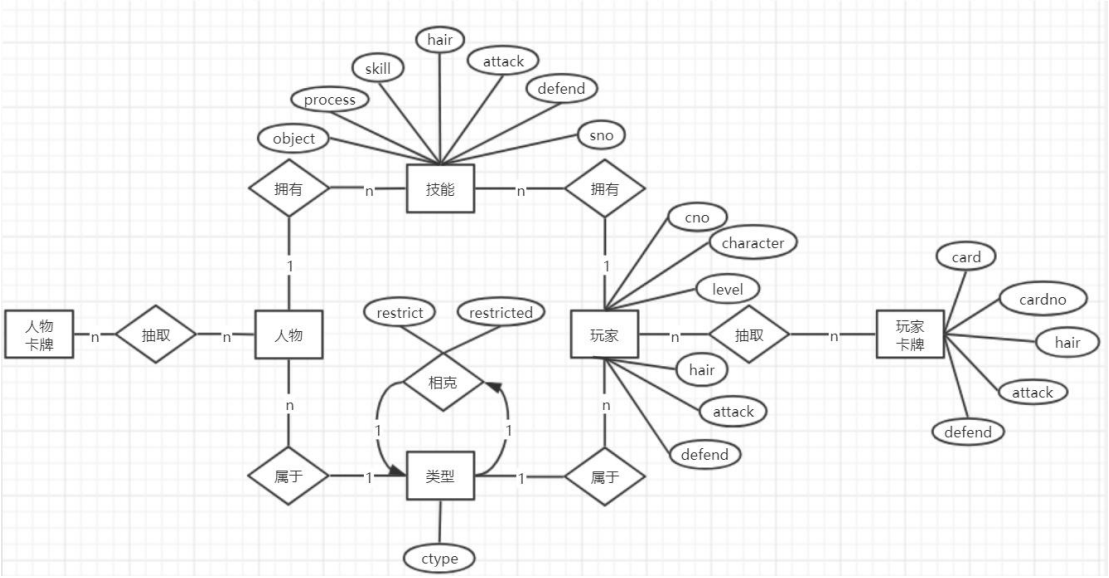

图 3 ER 图

三、逻辑结构设计

按照概念结构模型的设计构建关系数据模型。经过优化，得到 8 个关系：

人物(cno, character, hair, attack, defend, level)

玩家(cno, character, hair, attack, defend, level)

人物类型(cno, ctype)

类型相克(restrict, restricted)

技能(sno, skill, hair, attack, defend, process, object)

人技匹配(cno,sno)

人物卡牌(cardno, card, hair, attack, defend)

玩家卡牌(cardno, card, hair, attack, defend)

其中：人物、玩家、类型相克、技能、人物卡牌和玩家卡牌都是非主属性完全函数依赖于主键，而人物类型和人技匹配为 all-key。此逻辑模型做到了：

①所有非主属性对每一个码都是完全函数依赖；

②所有的主属性对每一个不包含它的码，也是完全函数依赖；

③没有任何属性完全函数依赖于非码的任何一组属性。
因此此逻辑结构设计的范式为 BC 范式。

四、数据库实施

1.搭建数据库

```
create database MBG

create table figure
(Cno varchar(8) primary key,
charac varchar(20),
hair int,
attack int,
defend int,
lv int
)
create table player
(Cno varchar(8) primary key,
charac varchar(20),
hair int,
attack int,
defend int,
lv int
)
create table cha_type
(Cno varchar(8) primary key,
ctype char(2) check (ctype in ('攻','防','智'))
)
create table type_relation
(res char(2) check (res in ('攻','防','智')),
resed char(2)check (resed in ('攻','防','智')),
primary key (res,resed)
)
create table skill
(Sno varchar(8) primary key,
skillname varchar(20),
attack int,
defend int,
process char(2) check (process in ('+','*')),
obj varchar(20) check (obj in('card','character'))
)
create table fs_match
(Cno varchar(8),
Sno varchar(8),
```

```
primary key (Cno,Sno)
)
create table f_card
(cardno varchar(20) primary key,
cardname varchar(20),
hair int,
attack int,
defend int
)
create table p_card
(cardno varchar(20) primary key,
cardname varchar(20),
hair int,
attack int,
defend int
)
```

2.数据库完整性控制

2.1 实体完整性：

每一个表中都设置了一个专门用来标志每一条记录的序号，并将其设置为了主键，确保了数据库的实体完整性，如 figure、player、cha_type 表都以 Cno 为主键，skill 表以 Sno 为主键，f_card、p_card 表都以 cardno 为主键，fs_match、type——relation 表以各自表格的两个元素的组合为主键。

2.2 参照完整性：

为了便于使用数据框格式从数据库中提取数据，并且拆分表格改进数据库的范式，此数据库中没有定义参照完整性约束。

2.3 用户定义的完整性：

设置了属性上的约束条件。

在表格 cha_type 和 type_relation 中，对人物的类型做的限制，即人物只能拥有“攻”、“防”、“智”这三种类型，不允许使用除此之外的类型对表进行插入或修改操作。

在表 skill 中，对 process 和 obj 进行了限制。列 process 为对应技能的对数值的操作，限制为“+”或“*”，即技能只能对数据进行加法或者乘法的操作。列 obj 表示技能作用的对象，限制为“card”或“character”，即技能作用的对象只能是一张卡牌或者一个游戏角色。

2.4 断言：

```
create assertion ass_on_figure check (100>=(select COUNT(*)from figure))
create assertion ass_on_player check (100>=(select COUNT(*)from player))
create assertion ass_on_type_relation check (3>=(select COUNT(*)from type_relation))
```

对数据库附加了三条断言，前两条是为了避免使得游戏机制过于复杂，对人物玩家的数量进行了限制，即 figure、player 表的记录条数都不允许超过 100。否则会有太多角色需要

用户来记忆，降低游戏的可玩性。

第三条是对 type_relation 表进行的限制，由于“攻”、“防”、“智”这三种类型人物相克关系是确定的，所以 type_relation 表是不需要修改的表格，将这张表格的记录条数限制为原本就规定好的 3 条。

2.5 触发器

```
create trigger insert_or_update_cha
before insert or update on figure
referencing new row as newrecord
for each row
begin
if (newrecord.lv<1)or(newrecord.lv>6)
then newrecord.lv=1
end if
end;

create trigger insert_or_update_player
before insert or update on player
referencing new row as newrecord
for each row
begin
if (newrecord.lv<1)or(newrecord.lv>6)
then newrecord.lv=1
end if
end;

create trigger insert_or_update_cha_balance
before insert or update on figure
referencing new row as newrecord
for each row
begin
if (newrecord.hair>30)or(newrecord.attack>30)or(newrecord.defend>30)
then newrecord.charac='unbalanced'
end if
end;

create trigger insert_or_update_player_balance
before insert or update on player
referencing new row as newrecord
for each row
begin
if (newrecord.hair>30)or(newrecord.attack>30)or(newrecord.defend>30)
then newrecord.charac='unbalanced'
end if
```

```
end;
```

对数据库增加四条触发器，对 figure、player 表进行了限制。前两条为了维护游戏规则，根据规则，一共进行六轮战斗即能通关，所以人物和玩家等级不允许超过 6 或者小于 1，在错误设置的情况下，将其修改为一个默认等级 1 级。后两条为了维护游戏平衡，对角色的发量值、攻击值和防御值做出限制，因为某项过高的数值会导致玩家过于强大或者角色太难击败。一旦修改或插入操作中某一项的值超过了 30，即会将该条记录的名称先命名为“unbalanced”，由后续开发人员进行讨论后再进一步考虑如何处理这条打破游戏平衡的记录。

3.视图的建立

```
create view FC_match1
as
select *
from figure

create view FC_match2
as
select *
from player

create view FPcard_match1
as
select *
from f_card

create view FPcard_match2
as
select *
from p_card

create view viewskill
as
select *
from skill

create view viewcha_type
as
select *
from cha_type

create view viewfs
as
select *
```

```
from fs_match
```

以上视图的建立是为了配合前端的代码工作，生成的视图中仅仅提供了前端代码所需要的数据表，一方面便于前端开发者直接调用，简化操作，另一方面在之后的数据库维护和进一步开发中，可以将不同视图的权限分给不同层次的管理者或用户，保证其他数据的机密和安全，一定程度上确保最小访问权原则。

4.数据库安全性

```
grant all privileges
on table cha_type,f_card,figure,fs_match,p_card,player,skill,type_relation
to manager1,manager2,manager3,manager4
with grant option;

grant all privileges
on table figure,cha_type,player
to cha_manager;

grant all privileges
on table f_card,p_card
to card_manager;

grant all privileges
on table skill,fs_match
to skill_manager;

grant select
on table cha_type,f_card,figure,fs_match,p_card,player,skill,type_relation
to gameplayer
```

主要通过对用户的授权来实现数据库的安全性控制。

在目前的情况下，将数据库的管理情况分为三级，即开发人员——管理者——游戏玩家。开发人员有四位，所以将所有表的所有权限给了 manager1 到 manager4，并允许开发人员进行授权操作。管理者分为卡牌管理者、角色管理者、技能管理者，卡牌管理者负责 f_card、p_card 表，角色管理者负责 figure、cha_type、player 表，技能管理者负责 skill、fs_match 表，三种管理者在协商一致的情况下分别对各自的负责的表格进行操作，由于需要管理者对数据库进行更新或修改，所以授予了全部权限。对于游戏玩家，为了使其了解游戏的机制、玩法和内容，所以允许查看所有表格的数据。

五、游戏应用开发与搭建

1.后端设计

1.1 数据库连接

利用 pymssql 中的 GetConnect()函数连接本地的数据库 DBMS，编写查询语句 cur.execute(“select * from ...”)，读取 MBG 数据库中 player、figure、card、skill 等数据表同

时根据需要做表连接返回查询结果，在 Python 内存中生成相应的 DataFrame（数据框）结构数据，为程序的运行提供基本的数据支撑。

1.2 程序主体设计

采用“自顶向下，模块化设计”的程序设计思路，对游戏应用进行解构，根据数据流图和游戏流程图，分离出其需要实现的各项子功能，具体包括：

①人物、卡牌、技能初始化（通过”gne_card（）”、”gne_roe（）”、”gne_skill（）”实现）；

②场景更换与玩家匹配（通过”scene（）、”role_match（）”实现）；

③随机抽牌、发牌（通过”card_shuffle( )”、”get_card( )”实现）；

④人物战斗（涉及到技能与卡牌的使用，通过”use_card( )”、”use_skill( )”实现）；

⑤结果反馈（与人物战斗同步，涉及到对人物属性的总体更改，通过”info_show_(...)”系列函数实现）

1.3 面向对象编程的应用

在数据库设计的过程中，我们绘制了 ER 图，明确了此数据库管理系统的实体，实体属性，以及实体之间的相互关系，完成了数据库系统的抽象建构。

在后端程序设计时，我也采用了这种思路，将人物、技能、类型、卡牌等抽象为面向对象程序设计中的“类”概念，在 Python 中生成了“Role”、“Skill”、“Card”等 Class，每个类都对应有自己的数据成员（即属性），也对应有自己的成员函数（即行为），从而完成封装。同时利用数据库中的数据记录，完成相应类的实例的创建。总的看来面向对象编程与关系数据库的原理有异曲同工之妙。

1.4 与前端的对接

模块化编程，由”__main___”函数实现统一调度，也相应地为前端提供相应的功能接口。

编写 info_show_( )系列函数，为前端界面信息的展示提供相应的获取渠道。

例如，前端需要显示人物当前的信息，那么可以调用我的内置函数，获得人物的名称、发量、攻击值、防御值、类型等属性，并映射到 QT 界面进行实时展示。

1.5 数据库读写

此前所叙述的程序模块主要是将数据读取进内存，并直接在内存中运行，根据用户行为进行数据的操作。

但数据库管理系统涉及到更新操作，所以我们还编写程序实现对数据库的实时更新，具体形成了玩家“存档/独挡”的游戏功能。在玩家通过一个场景后，玩家可以选择存档并退出，这时后端利用 Python 语言将缓存中的数据写入数据库相应的存档表中。玩家再次进入游戏时可以选择读档，同时 Python 读入上次存储的数据，便于用户继续游戏。

2.前端设计

“马保国勇闯南大信管”游戏以数据库作为后台数据调用源，将数据与 python 连接，在 python 中调用 Pyside2 包设计前端可交互的图形界面。python 脚本中将各图形界面封装为类，并将用户、卡牌等信息也封装为类，定义用户操作、卡牌抽取等动作为类中的函数。最终通过传入数据库中的数据，并调用类完成游戏的实现。用户的操作指令通过图形界面传入，并在 python 当中接收。收到用户指令后，py 脚本将自动调用数据库当中相应的表与属

性，对数据库中的数据进行查询和更新。

界面设计分为登陆界面、故事界面、难度选择界面、战斗界面、通关/失败提示界面。

1）登陆界面对应了数据库当中用户的注册与登录，并且进行了相应的安全性检查。即在用户登录界面注册新用户的行为将会在数据库中进行相同实现。

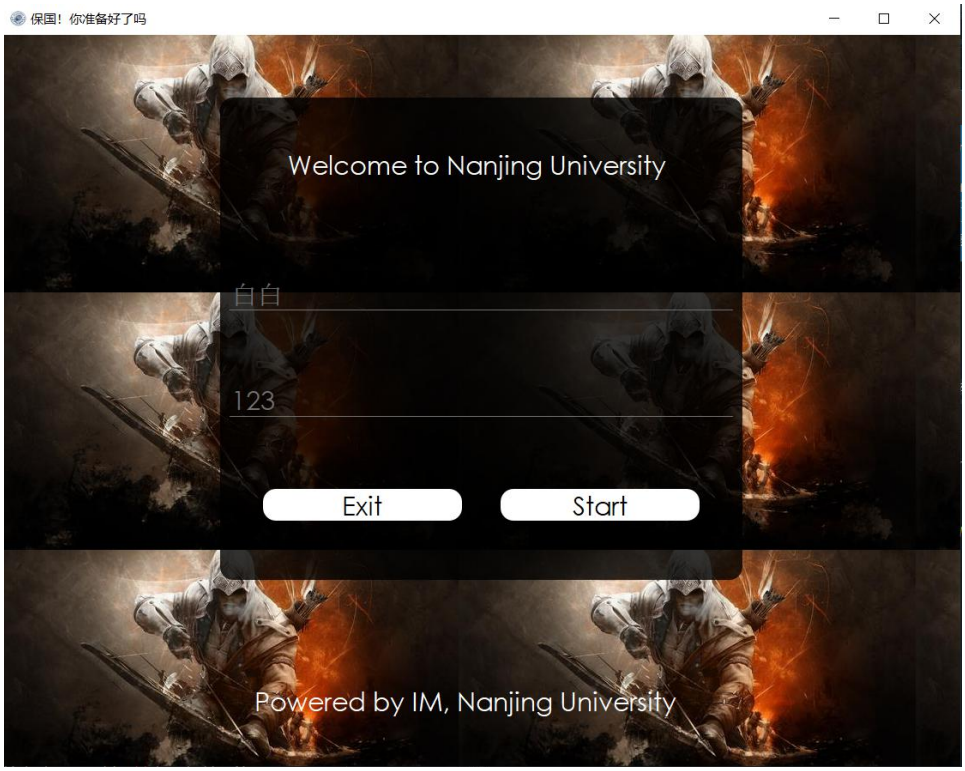

图 4 登陆界面 1

• 安全性检查：

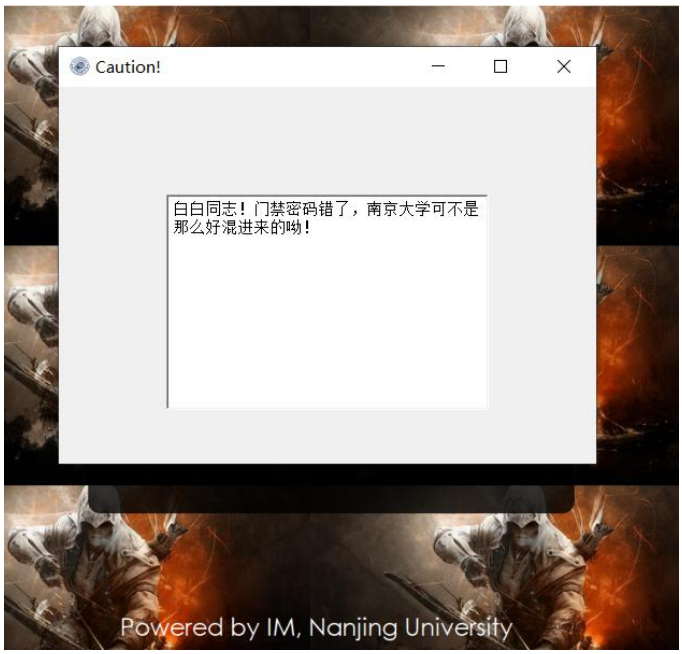

图 5 登陆界面 2（安全性检查）

2）故事界面为了增强用户的体验感、游戏的趣味性而设计，可以调用数据库当中相应的角色并初始化自定义故事情节：

图 6 故事界面 1

图 7 故事界面 2

3）难度选择界面，通过选择三个不同的难度系数，来调用数据库当中对应级别的人物角色和属性，从而达到控制游戏难度的目的

图 8 难度选择界面

4）战斗界面

在战斗界面中，每一回合从数据库中随机抽取人物对应的卡牌，玩家与对手各三张，并读取玩家与对手的各个属性值、技能等。可以通过点击卡牌出牌的方式，对人物属性值进行更改，从而达到增/减发量、攻击值、防御值等效果。属性值的改变可以通过人物头像下方的进度条体现，最终每一回合结束后将玩家的属性在数据库当中进行更新。

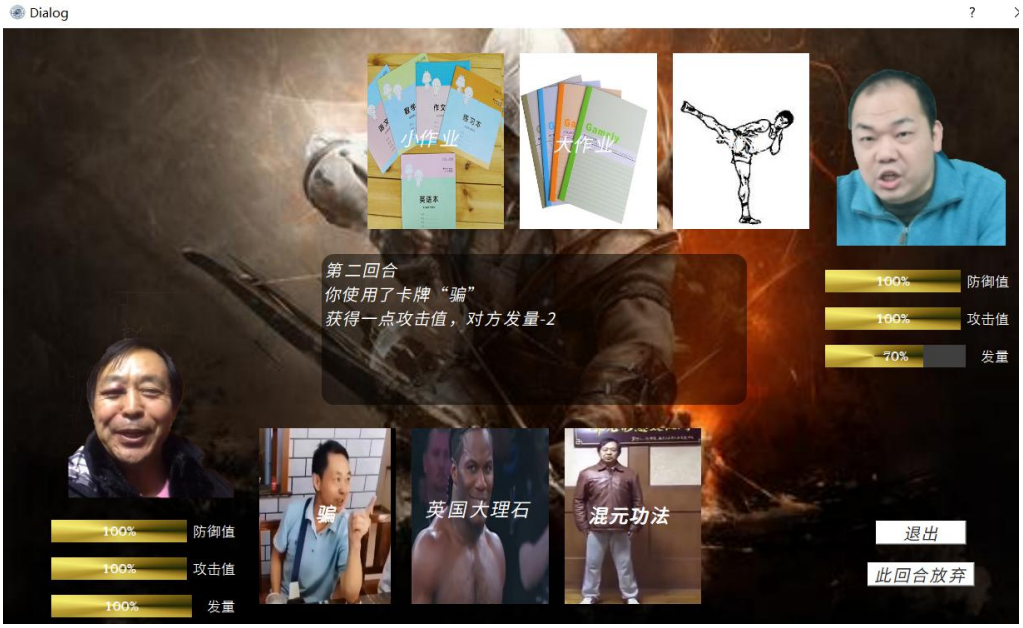

图 9 战斗界面

5）最后是通关/失败界面，也会有相应的故事场景出现。

图 10 通关界面 1

图 11 通关界面 2

六、小组分工

共同完成：想法提出、游戏大体流程、概念结构框架设计

林定康：游戏故事线编写、需求分析、概念结构设计、逻辑结构设计、流程图数据流图 ER 图制图、文稿汇总、PPT 制作。

白雪童：数据库连接、前端设计、界面图片制作与搜集、游戏演示视频录制、数据库数据与应用数据交互。

帅奕航：界面设计、数据库建立、数据库维护、安全性完整性控制设计、数据库连接。

陈文杰：python 游戏程序开发、链接前端、连接数据库、后端部分 PPT 制作、数据库数据与应用数据交互。