Android 移动应用开发期中项目报告

三国演义电子词典

组号： 09

小组成员：

目录

1. 概述

本次项目意在开发一款以三国为背景的三国电子辞典应用，并在此基础上添加三国牌游戏模块，集学习与娱乐于一体。

该应用基于 Java 语言以及 Python 语言，利用 Android Studio 软件，对三国人物信息进行展示的同时，还提供对人物信息进行增删改查、背景音乐播放与昆特牌游戏功能。

在人物信息的存储上，使用了数据库技术，提供完整的 4500 个三国人物的信息；在人物信息检索上，提供条件筛选批量搜索服务；在游戏模块上， 以游戏《The Witcher 3: Wild Hunt》中的昆特牌小游戏为基础设定，以三国武将作为玩家方，与我们训练出来的 AI 进行昆特牌对战，游戏过程中双方利用有限数量的手牌进行三局两胜的对局。

2. 背景

本次项目设定使用人群较为广泛，主要针对对三国历史感兴趣或对三国历史有学习需要的人群。

在内容上，项目以三国为主题背景，在实现数据库以提供完整的 4500 个三国人物资料的基础上，除了提供基础的增删人物，查找人物以及修改人物信息的四个功能之外，还提供了信息检索批查找人物、背景音乐以及三国牌游戏这三个服务，用户可在游戏过程中加深对三国人物的印象，此外在细节处还支持手指左右滑动以利用返回界面或保存信息的功能，以及支持设置不同的内容展示视图。

在技术上，本项目在整体上以 Java 语言为主体，以 Android Studio 为软件基础进行开发。在数据库功能上，利用了爬虫技术从网上获取了资源后， 利用 sqlite，实现了 4500 个的三国人物信息资料库；在信息检索部分，利用了 SQL 语言，实现了人物信息的直接搜索与条件搜索；在背景音乐功能上， 利用了多线程技术，实现了后台播放；在页面滑动部分，利用到了属性动画技术，实现了手指向右滑动可以返回上一页面的功能；在游戏功能上,参考昆特牌的机制进行了实现。本项目比其他同类应用相比，优势在于：

- 在三国信息电子辞典部分，存储三国人物数量更多，资料更为完善，信息检索功能提供了批量筛选的服务，功能更为完善，具有实用性。
- 在信息处理上，对不同的信息进行了必填与选填的限制，保障了应用保存的信息的完整度的同时，对信息的欠缺有一定的容忍度。
- 数据库采用 SQLite，轻便可嵌入到软件中，同时将大量数据放在远程网络服务器上的数据库中，有需要才会下载到本地数据库，这样可减小本地数据库的负担，使其运行更稳定快捷，运行更为稳定.
- 在游戏模块部分，改变了昆特牌游戏此前只在 PC 端或主机端运营而完全没有涉猎于移动设备中的局面，使得玩家在安卓移动设备上也可以进行游戏，具有创新性。
- 在兼容性上，兼容 Android4.4 及以上版本。

- 在使用上，UI 界面信息清晰明了，使用提示充分，游戏运行无卡顿，支持左右滑动与长按，方便操作，用户使用体验良好。

3. 项目说明

3.1. 功能模块

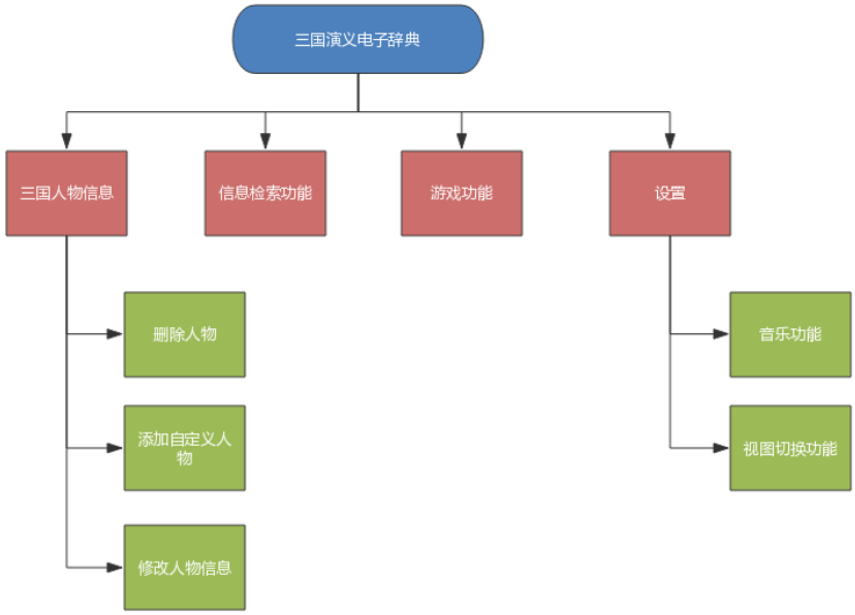


图 1 项目功能模块

3.2. 逻辑流程

3.2.1. 主界面总逻辑

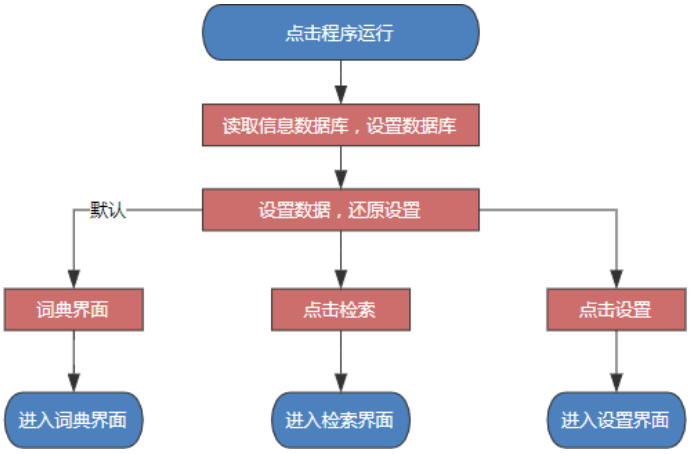


图 2 辞典主页逻辑流程图

3.2.2. 辞典主页逻辑流程

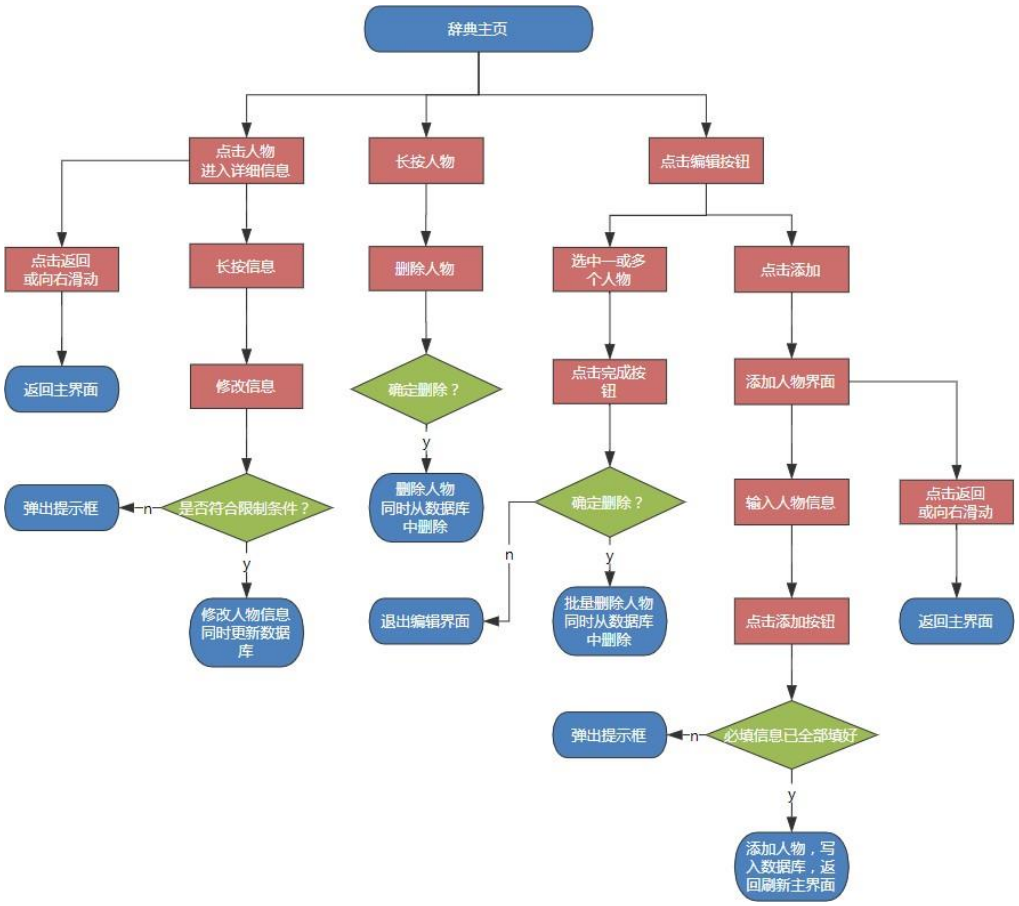


图 3 辞典主页逻辑流程图

3.2.3. 检索界面逻辑流程

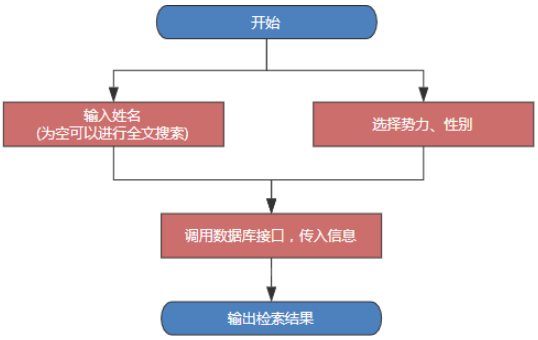


图 4 检索界面逻辑流程图

3.2.4. 设置界面逻辑流程

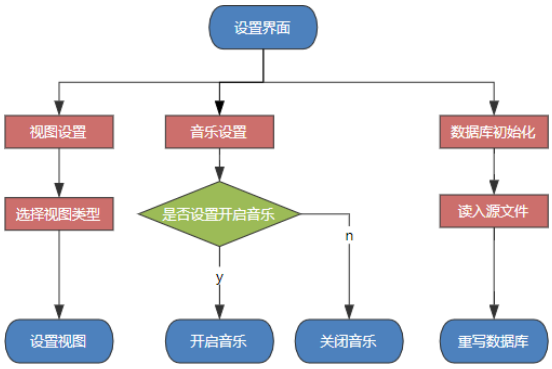


图 5 检索界面逻辑流程图

3.2.5. 游戏模块逻辑流程

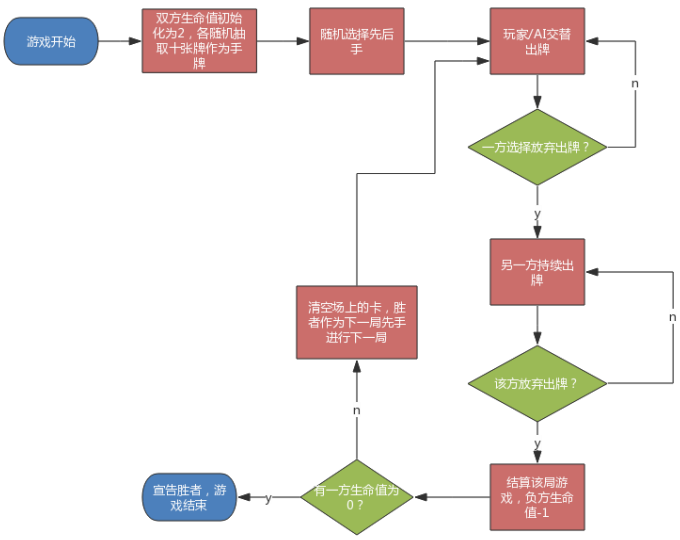


图 6 游戏模块逻辑流程图

3.3.技术说明

3.3.1. 滑动模块

在这一模块的功能中，除了使用到 OnTouchEvent 触屏事件外，更主要运用的技术是属性动画 Property Animation。安卓的动画类型可分为视图动画与属性动画两大类，而属性动画是在安卓 3.0 版本后出现的新特性。与视图动画相比，属性动画的作用对象不再局限于 View 对象，同时动画效果的种类也更多。

属性动画的原理是在一定的时间间隔内，通过不间断地对一个 View 的属性值进行改变并进行相对应的设置，以达到出现动画效果的目的，在这某种程度上类似于.gif 图的原理，在实现过程中，只需要设置一个 View 的初始值和结束值， 属性动画就会随时间的变化而逐渐把 View 的属性从初始值变化到结束值，而这个属性可以是任意对象的任意属性。

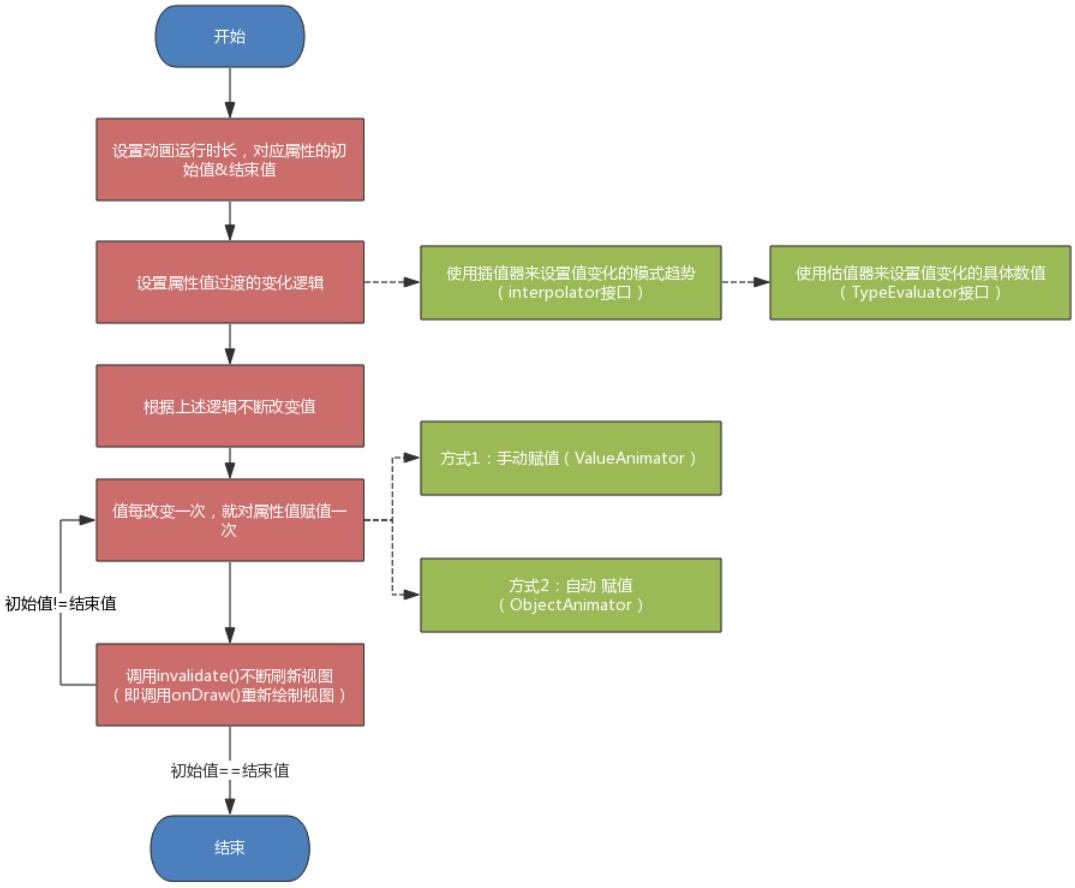


图 7 属性动画的工作原理逻辑

在属性动画中，最重要的两个类分别是 ValueAnimator 类以及 ObjectAnimator 类。前者是属性动画机制中最为核心的一个类，负责实现动画的原理，也即不断地控制值的变化并手动进行赋值，而后者则直接对对象的属性值进行改变操作，其本质上是不断地控制值的变化后不断地自动地赋予对象对应的属性，更加智能、自动化程度更高。两者区别在于间接或直接地对对象的属性进行操作。在本次项目中，使用的是 ObjectAnimator 类，在使用其的过程中，

主要调用 get 函数与 set 函数。

在具体的实现过程中，首先，获取当前的根布局与手机屏幕的尺寸信息：

```
//设置滑动返回
decorView = getWindow().getDecorView();// 获得手机屏幕的宽度和高度，单位像素
DisplayMetrics metrics = new DisplayMetrics();
getWindowManager().getDefaultDisplay().getMetrics(metrics);
screenWidth = metrics.widthPixels;
screenHeight = metrics.heightPixels;
```

图 8 根布局等信息的获取

而后，重载 OnTouchEvent 函数，对手指的按下、滑动以及抬起分别进行对应的处理：

```
@Override
public boolean onTouchEvent(MotionEvent event) {
    if(event.getAction() == MotionEvent.ACTION_DOWN){       // 按下
         downX = event.getX();
    }else if(event.getAction() == MotionEvent.ACTION_MOVE){ // 手指滑动
        float moveDistanceX = event.getX() - downX;
        if(moveDistanceX > 0){//向右滑动
            decorView.setX(moveDistanceX); //设置界面的X到滑动到的位置
        }
    }else if(event.getAction() == MotionEvent.ACTION_UP){   // 抬起手指
        float moveDistanceX = event.getX() - downX;
        if(moveDistanceX > screenWidth / 2){
            Continue_Move(moveDistanceX);
        }else{ //距离没超一半，恢复
            ObjectAnimator.ofFloat(decorView, "X", moveDistanceX, 0).setDuration(300).start();
        }
    }
    return super.onTouchEvent(event);
}
```

图 9 对 onTouchEvent 函数的重载

其中，函数 Continue_Move()实现的是滑动动画效果：

```
private void Continue_Move(float moveDistanceX){
    // 从当前位置移动到右侧。
    ValueAnimator anim = ValueAnimator.ofFloat(moveDistanceX, screenWidth);
    anim.setDuration(500); // 结束时间
    anim.start();
    anim.addUpdateListener(new ValueAnimator.AnimatorUpdateListener() {
        @Override
        public void onAnimationUpdate(ValueAnimator animation) {
            float x = (float) (animation.getAnimatedValue());
            decorView.setX(x);//不断设置根布局的x
        }
    });
    anim.addListener(new AnimatorListenerAdapter() {
        @Override
        public void onAnimationEnd(Animator animation) {
            Returns_null();
        }
    });
}
```

图 10 函数 Continue_Move()

此外，为了能够直接在滑动过程中看到即将返回的界面，需要设置 Activity 的背景为透明，为此，声明以下 style 并在 AndroidManifest.xml 中为对应的 Activity 进行主题 theme 的设置：

```
<style name="translucent" parent="Theme.AppCompat.Light.NoActionBar">
    <item name="colorPrimary">@color/colorPrimary</item>
    <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
    <item name="colorAccent">@color/colorAccent</item>
    <item name="android:windowBackground">@color/translucent</item>
    <item name="android:colorBackgroundCacheHint">@null</item>
    <item name="android:windowIsTranslucent">true</item>
</style>
```

图 11 Activity 透明背景主题

3.3.2. 游戏模块

游戏界面实现：

基础牌局由敌方攻城列-敌方远程列-敌方近战列-我方近战列-我方远程列-我方攻城列，考虑到最少需要 6 个 VIew 来展示牌局，因此选择了使用六个横向布置的 RecyclerView 来作为牌局展示，每个 RecyclerView 的右边用一个 TextView 显示这一列的总点数，双方近战列之间的空隙用于显示剩余双方剩余生命值、双方总点数，以及进入手牌/墓地界面的入口，如下

图 12 游戏界面

后续由其他成员优化的过程中，考虑到两点：

卡牌尺寸太小，可能会出现看不清的情况。将卡牌尺寸还原，增加两个垂直滚动，如下

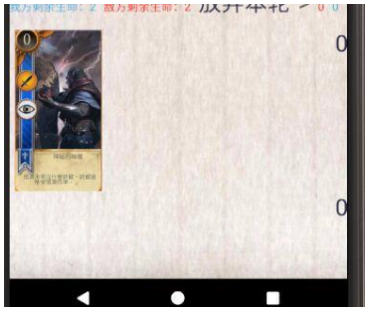


图 12 优化后游戏界面

进入手牌/墓地界面的接口原本是一个 > 箭头，click 监听的面积较小，将

其改为“选择手牌”四个字，避免出现点很多次都没有点到那个箭头的情况。

AI 实现思路：

- 双方手牌数量均有限，因此“手牌差距”是对战局影响最大的因素，AI 优先使用间谍来增加手牌/使用诱饵替换自己场上的间谍/使用医生复活墓地中的间谍。
- 避免被玩家利用田忌赛马的思路(即玩家部署大量间谍，保存大点数牌而 AI 打出大点数牌)在第一局失去大量手牌，当 AI 与玩家的总点数悬殊高于 30 点时 AI 自动放弃本局，减小沉没成本。
- 当 AI 打出医生卡且墓地中没有间谍时，AI 会选择复活一张“净收益”最高的卡，即这张卡打出之后 **AI 增加的点数+玩家减少的点数**最大。当墓地中的卡大于 6 张以上 AI 才会选择打出医生卡。
- 玩家方近战点数大于 30 时，AI 会选择使用火灼卡烧毁玩家近战列点数最高的卡。
- AI 优先打出普通卡，将能带来大量收益的“集群召唤”类卡(即打出 A 卡后，从卡组与手牌中找到所有与 A 卡同名的卡立刻打出)尽可能留到后面打。
- AI 方近战列点数大于 40 时，使用领导号角使近战列点数倍增。

以上是 AI 在出牌时的选择顺序，即从上至下执行，当执行了这些条目中的某一条打出一张牌时，AI 这一轮出牌结束，不再向下执行，等待玩家出牌。

3.3.3. 爬虫模块

在本次项目中，提供的 4500 个三国人物资料源自课程给定的网站，在这其中使用到了爬虫技术。

- Scrapy data flow
  Scrapy 使用了 Twisted 作为框架，Twisted 有些特殊的地方是它是事件驱动的，并且比较适合异步的代码。对于会阻塞线程的操作包含访问文件、数据库或者 Web、产生新的进程并需要处理新进程的输出(如运行 shell 命令)、执行系统层次操作的代码(如等待系统队列),Twisted 提供了允许执行上面的操作但不会阻塞代码执行的方法。

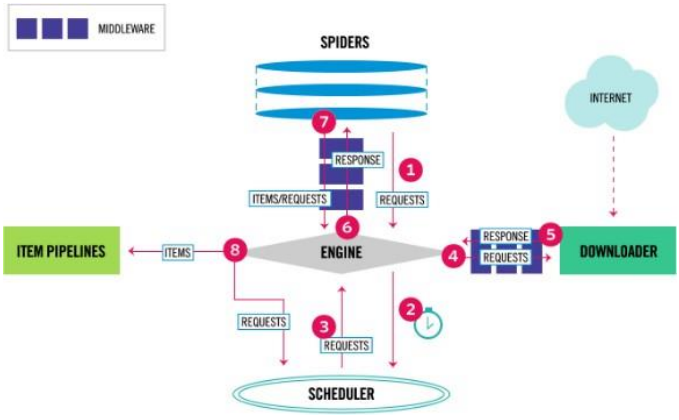


图 13 Scrapy 架构图

Scrapy 数据流是由执行的核心引擎(engine)控制，流程是这样的：

1、爬虫引擎获得初始请求开始抓取。

2、爬虫引擎开始请求调度程序，并准备对下一次的请求进行抓取。

3、爬虫调度器返回下一个请求给爬虫引擎。

4、引擎请求发送到下载器，通过下载中间件下载网络数据。

5、一旦下载器完成页面下载，将下载结果返回给爬虫引擎。

6、引擎将下载器的响应通过中间件返回给爬虫进行处理。

7、爬虫处理响应，并通过中间件返回处理后的 items，以及新的请求给引擎。

8、引擎发送处理后的 items 到项目管道，然后把处理结果返回给调度器，调度器计划处理下一个请求抓取。

9、重复该过程（继续步骤 1），直到爬取完所有的 url 请求。

- 页面分析

通过浏览器的检查功能，我们发现三国资料是通过 iframe 便签取得别的地方的页面，真正的网址是域名加上下图红框中的后缀，如图。

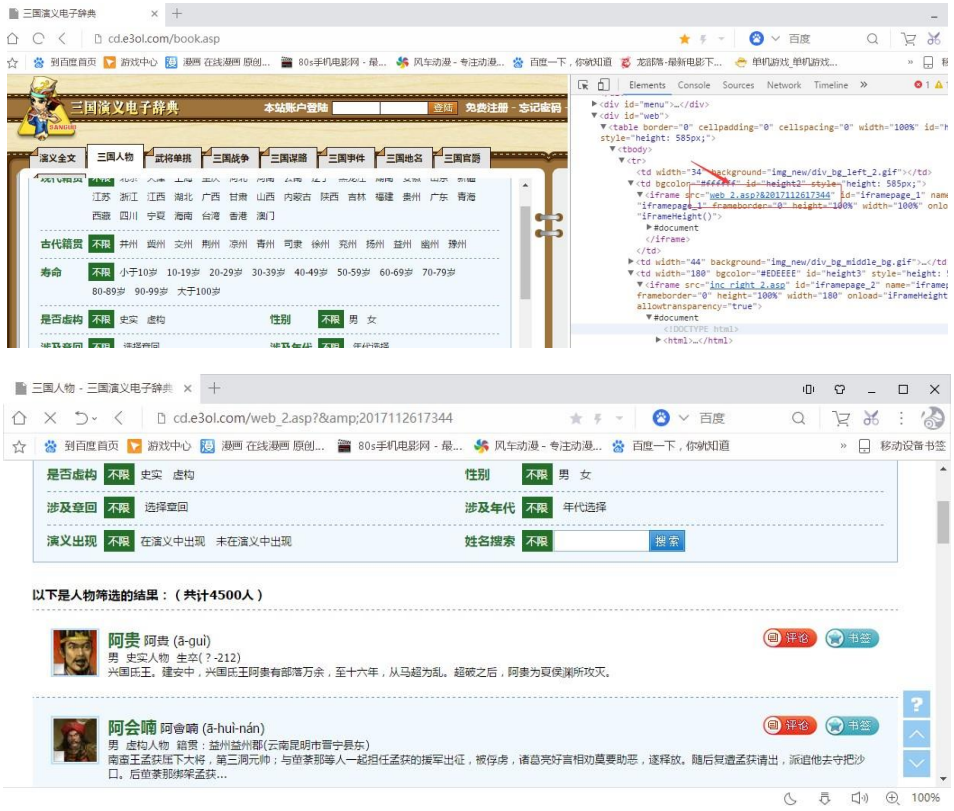


图 14 网站分析

由于我们需要获取的信息是人物的名字、信息、图片，所以要通过浏览器可以获取这些信息在网页中的位置，我们可以分析得到对应的 xpath 表达式：

1） 姓名

图 15 网页中名字的位置

xpath 表达式：

```
//*[@id="web"]/div[@class="book_2_list"]/div[2]/div[1]/span/a/text()
```

2) 信息

图 16 网页中信息的位置

xpath 表达式：

```
//*[@id="web"]/div[@class="book_2_list"]/div[2]/div[3]/text()[1]
//*[@id="web"]/div[@class="book_2_list"]/div[2]/div[3]/text()[2]
```

3） 图片 URL

图 17 页中图片的位置

这个地方我们只能先获取 style 的 xpath 的路径，然后对里面的内容进行提取截取 url。

Style 的 xpath 表达式：

```
//*[@id="web"]/div[@class="book_2_list"]/ div[1]/div/@style
```

- 翻页分析

1） 点击不同势力，只有箭头所指数字有变化

图 18 翻页分析 1

2） 点击下一页，url 最后多了个后缀，以及最后的数字随着页码变化而变化

图 19 翻页分析 2

- 实现代码

  以爬取魏国所有人为例：

  1） item.py 中定义好要存储的信息

```python
class SGItem(scrapy.Item):
    name = scrapy.Field() # 名字
    desc1 = scrapy.Field() # 信息1
    desc2 = scrapy.Field() # 信息2
    img_url = scrapy.Field() # 图片url
    country = scrapy.Field() # 势力
```

图 20 item.py 中的定义

2） 在自定义的派虫类 dmoz_spider.py 中写爬虫动作

```python
class SG(scrapy.Spider):
    name = "sanguo"
    # 允许的域名
    allowed_domains = ["cd.e3ol.com"]

    # 起始URL，这里按照规律将一个势力所有页面的url放了进去，就不用通过代码提取来分页
    start_urls = [
        "http://cd.e3ol.com/web_2.asp?a1=&a2=34&a3=&a4=&a5=&a6=&a7=&a8=&a9=&a10=& \
        a12=&pageno={}".format(i) for i in range(1, 11)
    ]

    def parse(self, response):
        country = "魏"
        for sel in response.xpath('//*[@id="web"]/div[@class="book_2_list"]'):
            item = SGItem()
            # 提取名字
            item['name'] = sel.xpath('div[2]/div[1]/span/a/text()').extract()[0]

            # 提取信息描述
            b = sel.xpath('div[2]/div[3]/text()[1]').extract()
            b[0] = " ".join(b[0].split())
            item['desc1'] = b[0]
            item['desc2'] = sel.xpath('div[2]/div[3]/text()[2]').extract()[0]

            # 提取图片url
            a = sel.xpath('div[1]/div/@style').extract()
            a[0] = a[0][21:-1]
            item['img_url'] = a[0]

            # 设置势力
            item['country'] = country

            yield item
```

图 21 爬虫动作

3.3.4. 数据库模块

SQLite 是 D.Richard Hipp 用 C 语言编写的开源嵌入式数据库引擎，它是一款轻型的数据库，是遵守 ACID 的关系型数据库管理系统，它的设计目标是嵌入式的，而且由于其占用资源低(占用内存只需几百 K)、处理速度快等特点，SQLite 的特点大致如下：

1.轻量级：使用 SQLite 只需要带一个动态库，就可以享受它的全部功能，而且那个动态库的尺寸想当小。

2.独立性：SQLite 数据库的核心引擎不需要依赖第三方软件，也不需要所谓的“安装”。

3.隔离性：SQLite 数据库中所有的信息（比如表、视图、触发器等）都包含在一个文件夹内，方便管理和维护。

4.跨平台：SQLite 目前支持大部分操作系统，不至电脑操作系统更在众多的手机系统也是能够运行，比如：Android 和 IOS。

5.多语言接口：SQLite 数据库支持多语言编程接口。

6.安全性：SQLite 数据库通过数据库级上的独占性和共享锁来实现独立事务处理。这意味着多个进程可以在同一时间从同一数据库读取数据，但只能有一个可以写入数据。

此处 SQLit 数据库主要因为 Android 中内置了,相对于服务器端的 MySQL 这类数据库，把一些文件信息直接保存到本地读取速度回更加的快，也省去了再去访问服务器数据库的时间。

Android 提供了一个 SQLiteOpenHelper 类。我们需要定义一个继承自 SQLiteOpenHelper 的子类(在实例中为 DatabaseOpenHelper 类)，并在此类中重写 onCreate()和 onUpgrade()方法。

```
public class DatabaseHelper extends SQLiteOpenHelper {
    private static final String name = "project_data.db";
    private static final int version = 1; //数据库版本

    public DatabaseHelper(Context context) { super(context, name, null, version); }

    @Override
    public void onCreate(SQLiteDatabase db) {
        Create_role();
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        Drop_Tab("role");
        Create_role();
    }
```

图 22 数据库继承实现

重载完 SQLiteOpenHelper 之后。

需要在这个数据库中创建表，同时也要对更新数据库时删除旧的数据表定义接口，此时实现 3 个接口：

- 创建数据表：使用继承来的 getWritableDatabase()和 SQL 语句进行数据表的创建。
- 删除表：传入数据表名称，使用继承来的 getWritableDatabase()和 SQL 语句删除表。
- 判断表是否存在：通过 getReadableDatabase()和 SQL 语句进行数据表

查询来判断是否存在。

此处考虑到需要读取数据，写入数据，更新数据，删除数据。还要为数据库实现信息的增删改查接口：

- 数据写入入接口：使用 getWritableDatabase()获取一个可写数据库和 ContentValues 作为数据值的存储，将传入的数据进行写入。
- 数据更新接口：使用 getWritableDatabase()获取一个可写数据库、ContentValues 作为数据值的存储、用传入的数据 ID 构造 SQL 的条件语句进行数据的更新。
- 数据读取接口：通过 getReadableDatabase()获取一个可读数据库和传入的所需数据的 ID 或者部分数据信息构造 SQL 语句进行查询，然后使用 Cursor 接收查询得到的数据并将数据放入一个 Data 类中返回。
- 数据删除接口：使用 getWritableDatabase()获取一个可写入的数据库，然后直接通过传入的 ID 进行删除构造 SQL 语句进行删除。

以上接口的具体实现的功能参见代码架构部分，由于都是用字符串转接成 SQL 语句所以代码不在展示。

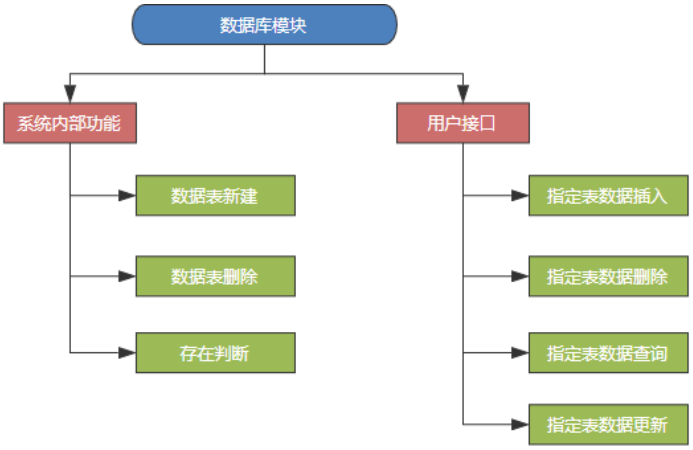


图 23 数据库功能模块

数据库模块还包括数据最开始的数据库写入，数据库源数据使用爬虫爬下来的 json 文件，通过 java 再带的 JSONObject 进行数据的读取和处理。在应用第一次启动的时候进行数据的写入。

这部分的代码主要是读入 json 和 json 转成字符串之后的处理。

```
private ArrayList<Data> Read_Data(String json_name){
    ArrayList<Data>reads=new ArrayList<>();
    try {//将json文件读入buffer
        InputStreamReader isr = new InputStreamReader(getAssets().open(json_name+".json"),"UTF-8");
        BufferedReader br = new BufferedReader(isr);
        String line;
        StringBuilder builder = new StringBuilder();
        while((line = br.readLine()) != null){
            builder.append(line);
        }
        br.close();
        isr.close();
        JSONObject testjson = new JSONObject(builder.toString());//builder读取了JSON中的数据。
        //直接传入JSONObject来构造一个实例
        JSONArray array = testjson.getJSONArray(json_name);          //从JSONObject中取出数组对象
        for (int i = 0; i < array.length(); i++) {
            JSONObject role = array.getJSONObject(i);     //取出数组中的对象
```

图 24 读入 json 文件（json 转字符串处理略）

后期每次第一次启动需要写入的数据太多，耗时过长，此时使用另一种方式进行数据库准备，首先我们先提取之前已经写入好的数据库，放入资源文件，使用 byte 读写的方式将数据库整体移到安装目录下。

```
private void copyDataBase() throws IOException {//直接复制数据库到对应路径
    String databaseFilenames = DATABASE_PATH + dbName;//路径和名字
    File dir = new File(DATABASE_PATH);
    if (!dir.exists())//判断文件夹是否存在，不存在就新建一个
        dir.mkdir();
    FileOutputStream os = null;
    try {
        os = new FileOutputStream(databaseFilenames);//得到数据库文件的写入流
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
    InputStream is = Main.this.getResources().openRawResource(R.raw.project_data);//得到数据库文件的数据流
    byte[] buffer = new byte[8192];
    int count = 0; //进行数据库写入
    try {
        while ((count = is.read(buffer)) > 0) {
            os.write(buffer, 0, count);
            os.flush();
        }
    } catch (IOException e) {
    }
    try {
        is.close();
        os.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

图 25 直接复制数据库到对应路径代码

3.3.5. 背景音乐模块

音乐模块使用安卓自带的 MediaPlayer，Android 的 MediaPlayer 包含了 Audio 和Video 的播放功能，在 Android 的界面上，Music 和Video 两个应用程序都是调用 MediaPlaer 来实现的。

主要实现步骤为：

- 首先创建 MediaPlaer 对象；
- 然后调用 setDataSource()方法来设置音频文件的路径；
- 再调用 prepare()方法使 MediaPlayer 进入到准备状态；
- 调用 start 方法就可以播放音频。

此处因为 APP 的需要所以，把系统所需的音频文件进行内置到资源文件夹 raw 中，然后打包成 apk，不使用从路径读取的方式。

在界面开始时先调用数据库模块读取设置数据表，然后从表中获取到之前的音乐设置信息，来确定是否需要进行音乐的播放。整个步骤在界面创建时需要执行到第三步，最终交由数据库的读出数据是否要进行第四部。

首先要进行音乐的准备也就是前三步

```
mediaPlayer=MediaPlayer.create(this,R.raw.music_main);
try {
    mediaPlayer.prepare();
    mediaPlayer.setLooping(true);
} catch (Exception e) {
    e.printStackTrace();
}
```

图 26 音频数据的准备

然后去读取数据库，先判断这个数据表是否存在，不存在则创建新表写入默认值

```
if(!databaseHelper.tabIsExist("settings")){
    databaseHelper.Create_settings();
    databaseHelper.Insert_settings(music,display);
}
```

图 37 判断设置数据库是否存在

```
int display_music=databaseHelper.Select_settings();//读取设置数据库
display=display_music%10;
music=display_music/10;

if(music==1){ //如果音乐为开
    mediaPlayer.start(); //开始播放
    mediaPlayer.setOnCompletionListener(new MediaPlayer.OnCompletionListener() {
        @Override  //设置循环播放监听
        public void onCompletion(MediaPlayer arg0) {
            mediaPlayer.start();
            mediaPlayer.setLooping(true);
        }
    });
}
```

图 28 判断是否开启音乐，并设置音乐循环播放。

同时对于应约播放开关进行监听，如果根据开关打开和关闭进行音乐播放和暂停，同时在修改了设置之后将数据写入设置数据库

```
switch_music.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton compoundButton, boolean b) {
        if(b){
            Toast.makeText(Main.this,"音乐打开",Toast.LENGTH_SHORT).show();
            music=1;
            databaseHelper.Updata_settings(music,display);
            mediaPlayer.start();
        }
        else {
            Toast.makeText(Main.this,"音乐关闭",Toast.LENGTH_SHORT).show();
            music=0;
            databaseHelper.Updata_settings(music,display);
            if(mediaPlayer.isPlaying())
                mediaPlayer.pause();
        }
    }
});
```

图 29 音乐开关监听

3.3.6. 主界面模块

主界面模块使用了 TabHost+Fragment 实现类似微信、淘宝中的底部导航栏，在整个主界面模块中承载了 3 个页面，同时减少了切换界面所带来的延迟和调用数据库接口的次数，此处因为使用了爬虫获取数据库，所以数据库比较大。

此处在信息展示中图片的展示使用了 Glide 架构进行加载 URL 图片，好处就是相对于使用 HTTP 连接进行架构的加载，其代码量更为少，不过主要的一点就是现在 android 中访问网络不能再主线程中进行，也就要新开线程进行获取图片，此时获取图片的方式为异步获取。

图 30 Glide 架构功能列表

主界面和搜索界面采用了数据隔离的理念，主界面的显示数据可以根据我们的使用自己添加或者从搜索数据库中添加。主界面和搜索界面分别使用自己的数据库。主界面的数据库进行删除操作不会修改到搜索数据库，搜索界面的数据库会接纳主界面数据的信息修改和添加，但是在最后重新初始化数据库的时候搜索数据库会还原成最初模式，重主界面数据库中添加的内容也会删除，同时也确保在信息展示界面载入时不会因为需要载入过多信息而崩溃（测试过如果读取过多信息会出现闪退）。

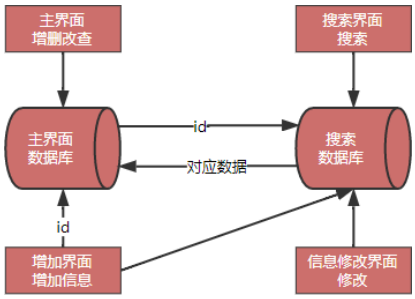


图 31 信息展示主界面与搜索界面数据库关系

三个界面分别为：

- 信息展示界面：

  信息展示界面使用recycleview，进行信息展示，同时为recycleview 增加了过度同动画，同时使用 LayoutManager 对于设置界面改变展示布局的方式作出相应。

  在主界面打开时调用了调用了数据库中获取全部数据(第一次开启需要写入数据)：

```
if(!databaseHelper.tabIsExist("main")) {//主界面数据库
    databaseHelper.Create_main();
    Insert_main_Data();
}
datas=databaseHelper.Select_main_data();
```

图 32 查询主数据表获取数据

在 recycleview 适配器中作出了兼容改变布局的调整同时使用 Glide 对我们数据库存入的抓包获取到的图片 URL 进行加载。

```
else if(choose==1){//界面2
    ImageView imageView = viewHolder.getView(R.id.item1_image);
    ImageView country = viewHolder.getView(R.id.item1_country);
    ImageView choose = viewHolder.getView(R.id.item1_choose);
    TextView name = viewHolder.getView(R.id.item1_name);
    Glide.with(mContext).load(data.getUrl()).into(imageView);
    //本地无缓存使用url获取
    if(data.getCache()==0)Glide.with(mContext).load(data.getUrl()).into(imageView);
    else imageView.setImageBitmap(data.getBitmap());
    name.setText(data.getName());
    country.setImageBitmap(data.getCountry_image(mContext));
    choose.setVisibility(View.INVISIBLE);
}
```

图 33　recycleview 适配器填充数据（部分）

信息展示模块中实现了增删功能：

图 xx 进入信息展示界面编辑

图 xx 信息编辑界面

在信息展示界面进入信息编辑功能之后可以通过点击加号进入信息增加人物界面，单击人物后会出现标记(此处记录点击的 position 进行更改)，在使用确认编辑之后会将人物删除，此时删除的时候回调用数据库的删除接口同时删除数据库对应数据。

```
else{//宫格模式显示设置标识
    ImageView choose = viewHolder.getView(display==1?R.id.item1_choose:R.id.item2_choose);
    if (choose.getVisibility()==View.INVISIBLE) {
        choose.setVisibility(View.VISIBLE);//设置标识可见性
        choose_list.add(String.valueOf(position));//移入标识数组
    } else {
        choose.setVisibility(View.INVISIBLE);
        choose_list.remove(String.valueOf(position));//移出标识数组
    }
}
```

图 34 增加删除标识代码（部分）

确认删除的时候有一个注意点，当我们删除了 ArrayList 的 data 之后，此位置之后的数据的位置会发生前移改变，所以需要需要倒序删除， 也就是从最后一个被选中的数据开始进行删除

```
Collections.sort(choose_list); //排序
for (int j = choose_list.size() - 1; j >= 0; j--) {//从大到小删除避免位置改变
    Data tmp = datas.get((int) Integer.valueOf(choose_list.get(j)));
    if (tmp != null) //从数据库中移除
        databaseHelper.Delete_main(tmp.getId());
    datas.remove(tmp);//从数据List中移除
    commonAdapter.notifyDataSetChanged();//适配器注意变化
}
```

图 35 倒序删除数据

- 搜索界面：

搜索界面整体由两个 radiogroup 和一个 searchview 构成搜索条件输入部分，然后由一个 recyceview 构成搜索到的数据展示部分。

图 36 搜索界面

SearchView 本身带了输入和提交，我们只需要对其进行重载即可，然后在提交时获取两个 radiogroup 选择的位置。之后调用数据库模块中实现的查询接口进行查询，将返回的数据展示在 recycleview 上，此处 recyceview 功能和信息展示界面类似。此时搜索会限制返回信息少于 200 条。

此处额外一点，信息检索功能建立在独立的数据库模块上，检索出的信息需要点击添加按钮才会到信息展示的主界面，主界面只是拥有部分数据。

- 设置界面：

  设置界面最为简单，此处描述一下功能，就是由一个 radiogroup 和一个 switch 构成， radiogroup 通过选择改变 recycleview 的 LayoutManager，然后修改 recycleview 的显示方式，switch 直接接入到上述音乐模块中。最后是一个按钮调用数据库接口删除数据表然后重新写入进行还原。

3.3.7. 其他界面

其他界面主要是具体信息展示修改，和增加人物界面，这两个界面主要都是一些输入，和显示控件，这两个界面主要就是对信息进行增改，此处只要调用上述的数据库接口即可实现。

这个界面关键点对于相机和相册读入的图片进行裁剪以及相机和相册权限的申请，此处一点就是由于 SQLite 的限制一个人的信息包括图片之和不能超过 2m 导致我们必须就图片进行裁剪。

3.4. 代码架构

3.4.1. 总架构

图 37 总 UML 类图

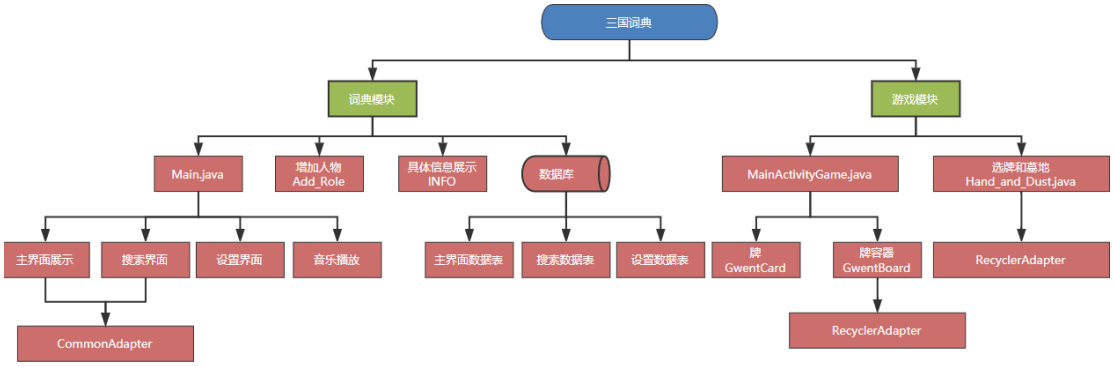


图 38 总代码阅读器

3.4.2. 游戏模块

在游戏模块中，整体架构如下图

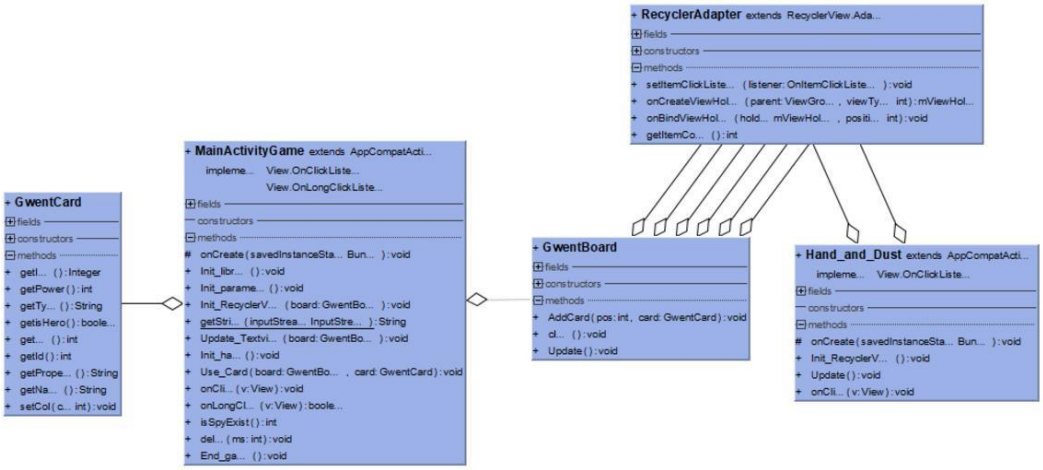


图 39 游戏模块代码架构

下面具体介绍各类与函数的作用:

- GwentCard：卡牌类，存放卡牌的属性
- GwentBoard：游戏牌局，使用 6 行 RecyclerView 展示牌局
  - AddCard(int pos,GwentCard card) 向指定位置添加卡牌
  - clear() 清空 board
  - Update() 更新 board

- Hand_and_Dust：玩家手牌与墓地，使用两行 RecyclerView 展示手牌与墓地
  - Init_RecyclerView() 初始化 RecyclerView
  - Update() 更新 VIew

- MainActivityGame：游戏主体
  - Init_Library() 初始化双方卡组
  - Init_Parameter() 初始化参数
  - Init_Hand() 初始双方手牌
  - Use_Card(GwentCard card) 使用传入卡牌，并结算产生的效果
  - Update_TextView() 更新界面
  - deley(int ms) 延时函数
  - End_game() 发出游戏结束信号

3.4.3. 数据库模块

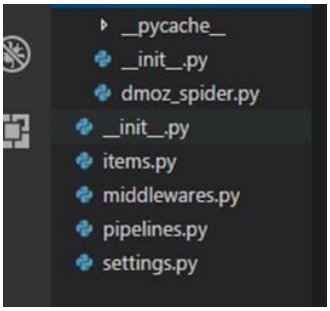
图 40 数据库模块代码架构

部分函数解释：

- private void Insert(Data data)：传入一个 Data 类然后插入搜索数据库
- public int Insert_back_id(Data data)：传入一个 Data 类然后调用 Insert 进行插入，插入后，查询其在搜索数据库的 ID,作为结果返回用于设置 arraylist 中 data 的 id
- public void Updatas(Data data)：根据 Data 的 ID 更新其在搜索数据库中的所有数据。
- public void Delete(Data data)：根据 Data 的 ID 删除其在搜索数据库中的存在。
- public Data Select(int id)：传入数据 ID，从搜索数据库中读取数据
- Public ArrayList<Data> Search_Data (String name,int country,int sex)：根据输入条件检索出搜索数据库中的对应数据，name 为名字进行模糊搜索，country 为势力，sex 为性别。此接口供搜索界面使用。
- public void Create_role()：创建搜索用的搜索数据库
- public void Create_main()：创建主界面用的只包含 ID 的数据库
- private void Insert_main(Data data)：向主界面数据库中插入 ID 信息。

- public ArrayList<Data> Select_main_data()：读出主界面中的所有 ID，并调用搜索数据库中的函数 public Data Select(int id)，读取相应数据。
- public void Delete_main(int id)：从 main 数据库中删除对应的 ID 信息。
- public void Drop_Tab(String tabName)：删除对应的表

3.4.4. 爬虫模块

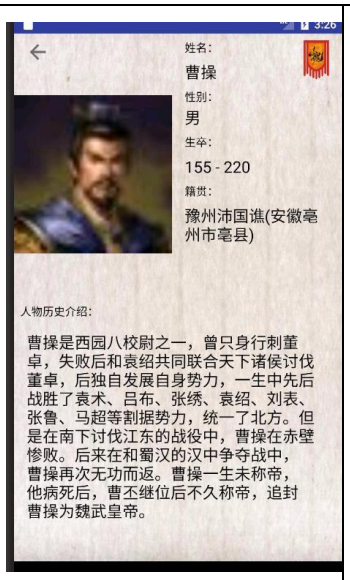
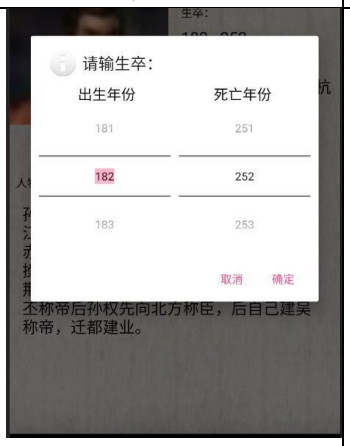
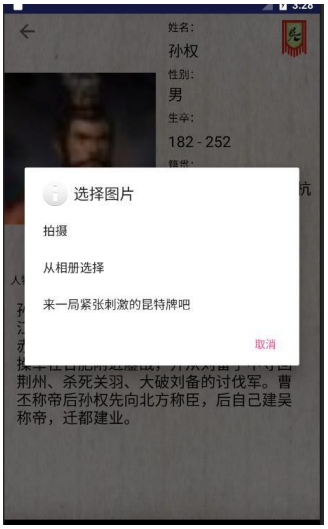
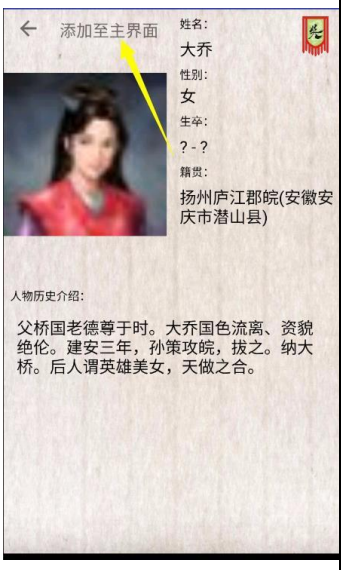
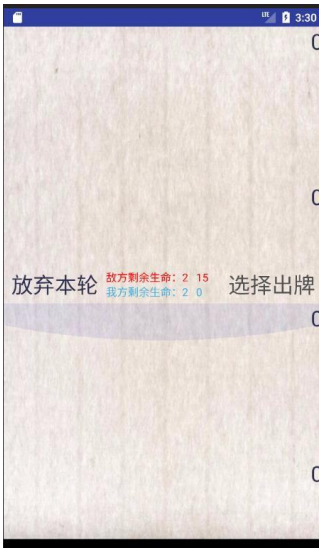


- items.py 负责数据模型的建立，类似于实体类。
- middlewares.py 自己定义的中间件。
- pipelines.py 负责对 spider 返回数据的处理。
- settings.py 负责对整个爬虫的配置。
- spiders 目录负责存放继承自 scrapy 的爬虫类。

4. 应用效果

动态效果参照视频

		
显示模式 1	显示模式 2	显示模式 3

<table>
<tr><td> 
</td><td> 
</td><td> 
</td></tr>
<tr><td>人物信息展示</td><td>人物信息添加</td><td>信息删除</td></tr>
<tr><td> 
</td><td> 
</td><td> 
</td></tr>
<tr><td>信息修改（部分）</td><td>信息修改（部分）</td><td>信息检索</td></tr>
<tr><td> 
</td><td> 
</td><td> 
</td></tr>
<tr><td>检索信息添加至主界面</td><td>游戏主界面</td><td>游戏选牌界面</td></tr>
</table>

5. 开发过程中遇到的问题及解决办法

- **问题 1**：不清楚如何在 Android 中实现回合制

  由于昆特牌是一个回合制游戏， 双方交替进行， 原本设想使用 while(1)的死循环并设置跳出条件来执行回合制，但是 Android 的 Activity 只有如 onCreate，onStart，onResume 这样的生命周期，在这些里面写死循环会导致 Activity 创建/开始失败，而 Activity 又没有 onRunning 这样的生命周期，因此很困扰。

  解决办法：交替发送广播实现回合制进行，Activity 创建后发送广播通知先手方出牌进行游戏，先手方出牌完毕后发送广播通知后手方出牌，如此交替，直到双方均放弃出牌后发送广播结算本局

- **问题 2**：SQLite 数据库中单次读取信息容量限制。

  实现数据库时读取数据经常崩溃，后再查询资料得知 SQLite 数据库中，单次查询返回的数据只能有 2M。

  解决办法：实现数据库时每条数据都使用单独查询，也就是将先查询符合条件的 ID 然后将这些 ID 存储起来在使用，然后按照 ID 一个一个去查询数据并返回

- **问题 3**：人物信息数据量大

  在爬虫与数据库实现的过程中，发现 4500 个人物的图片扒下来数据量有点大，不利于 app 的运行。

  解决办法：将 url 扒下来存在服务器数据库里， app 中的数据库只打包部分人的数据与图片，然后 app 中的查询会远程查询服务器里的数据库，如果想从远程数据库添加人物到本地数据库才会下载图片和人物资料， 这样就省了不少空间

- **问题 4**：url 获取图片效果不稳定

  使用自己写的 url 获取图片的函数时，因为 http 用法不是很了解导致每次都新开线程进行图片获取，但是最终有的图片没办法更新到对应的 imageview 上。

  解决办法：使用 google 推荐的 Glide 插件进行获取图片并更新解决了图片更新与显示同步的问题。

- **问题 5**：数据库一次查询返回数据过多导致崩溃

  一开始写了一个全数据查询，可以获得整个数据的所有数据，结果在运行时，查询了部分数据之后就闪退了。

  解决办法：通过测试是返回的数据量过多导致的，所以此处就不使用全数据查询了，控制了查询所能返回的数据上限增加稳定性。

6. 思考及感想

张镓伟：

这次 project 是在众多作业中艰难前行。同时这次 project 不仅是对前面所学知识的一次大整合，也是拓展新知识的机会。我在这次 project 中主要负责数据部分。通过查找记录，我们很快确定了使用轻型数据库 sqlite，因为它占用资源低，可以嵌入到 App 中，不像 mysql 那些需要另外安装。一开始我们打算的是手扒十几二十个人的数据做个 demo，不过后面想着既然都做了，就趁此机会学习一下爬虫，期末 project 说不定也有机会用到。所以我花了一些时间去学习爬虫并且将网站上的所有人物数据扒了下来。这次实验也是一次头脑风暴吧，感觉收获挺多的。

黄洁莹：

这次的 project 主要带给我的感受是承前半学期之知识，拓后半学期之眼界。在这次的 project 中，在实现上，我个人除了使用到了之前学习到的知识完成了基础功能中的小部分工作外，还通过学习 Android 中的触发事件与属性动画部分，实现了项目中页面滑动的效果。在学习的过程中理解了平时使用手机时常用到的左右滑动的原理与处理过程，以及在 java 实现动画方便有了更深的体会。而后期学习用户手册如何编写的过程中，看课程主页上给的那些例子和教程的时候对手机应用的用户手册有了新的体会，原来平时随便看看而已的应用说明背后还是要花开发人员很大的功夫去整理和编写的。

张子豪：

这次 project 中我主要负责的是昆特牌游戏部分。前前后后利用零零碎碎的时间写了大概两周，由于开始设计的时候代码架构没有太设计好， 让六个 ArrayList 分别存放而没有用一个数组存放，导致后续写代码的时候有一些地方需要写六次相同的代码，大大增加了代码量，但其实用数组存了话一个 for 循环就能搞定，代码可读性也会高很多，这让我以后对写 project 的时候在代码架构方面要更加仔细。

另外同样由于架构的问题，将手牌页面放在了另一个 Activity，本来计划 AI 出牌时 new 一个线程让它 sleep 来达到延时的目的，但是由于玩家出牌是在另一个 Activity，sleep 的话就无法及时返回牌局，因此最后这个延时方法没有调用。

蔡荣裕：

这次 project，主要负责部分界面的设计和数据库的实现，界面设计是修改幅度最大的，也是最花时间的，但是很多控件都是之前使用过的所以也不存在什么困难的地方。这次实验中更多的精力放在了不同界面转换时接口的设计和数据库设计接口的部分，SQLite 的使用不是很难，编程也只是普通的 SQLite，主要就是要给队友留好接口，做到只要告诉他们把 data 类放进去就可以实现什么什么功能，数据库创建时直接调用什么函数就可以。这些设计和调试还是比较花时间的。同时数据库因为要从 url 获取图片所以对于多线程的用法也有了一定的理

解，但是最终还是没有自己实现一个稳定的 http 获取图片还是比较。

最后对于整个 project 整合也是我来做，感触颇深，就算使用 GitHub 每个人的代码也是重口难调，最终整合的时候还是需要问每个人都做了什么，从那边可以把代码接进去，团队协作对于接口还是真的很重要的，不然真的很累。

7. 参考资料

http://cd.e3ol.com/book.asp
https://doc.scrapy.org/en/master/topics/architecture.html
还有很多博客就不一一列出了。