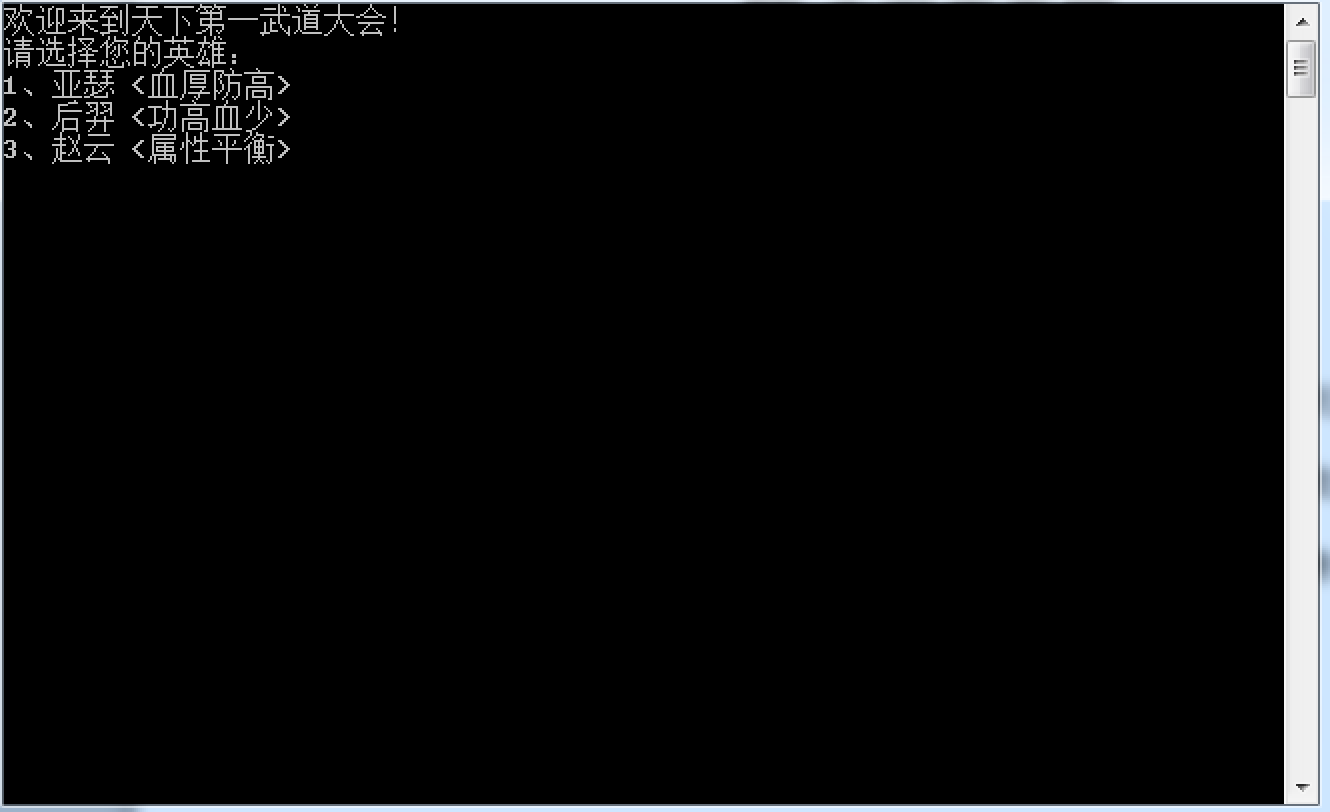
**PK小游戏案例**

# 1 案例介绍

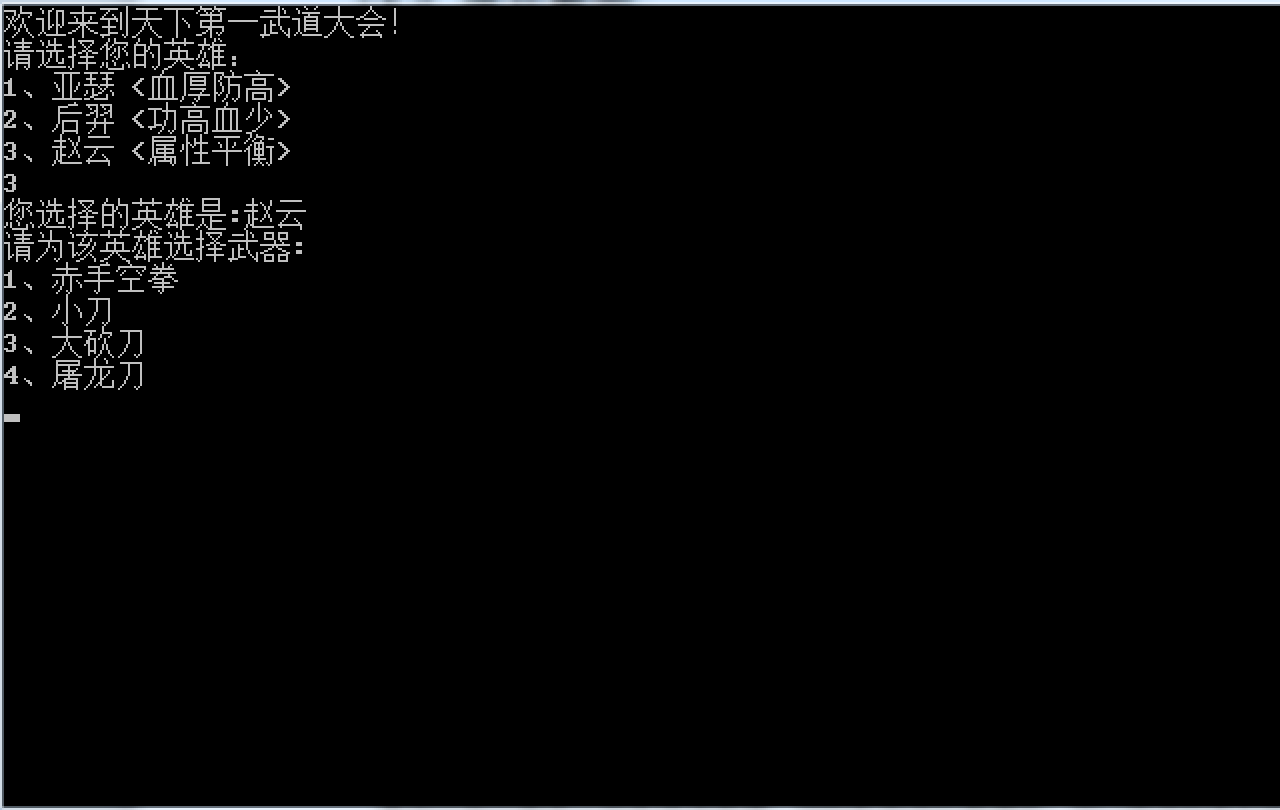
## 1.1 PK小游戏简介

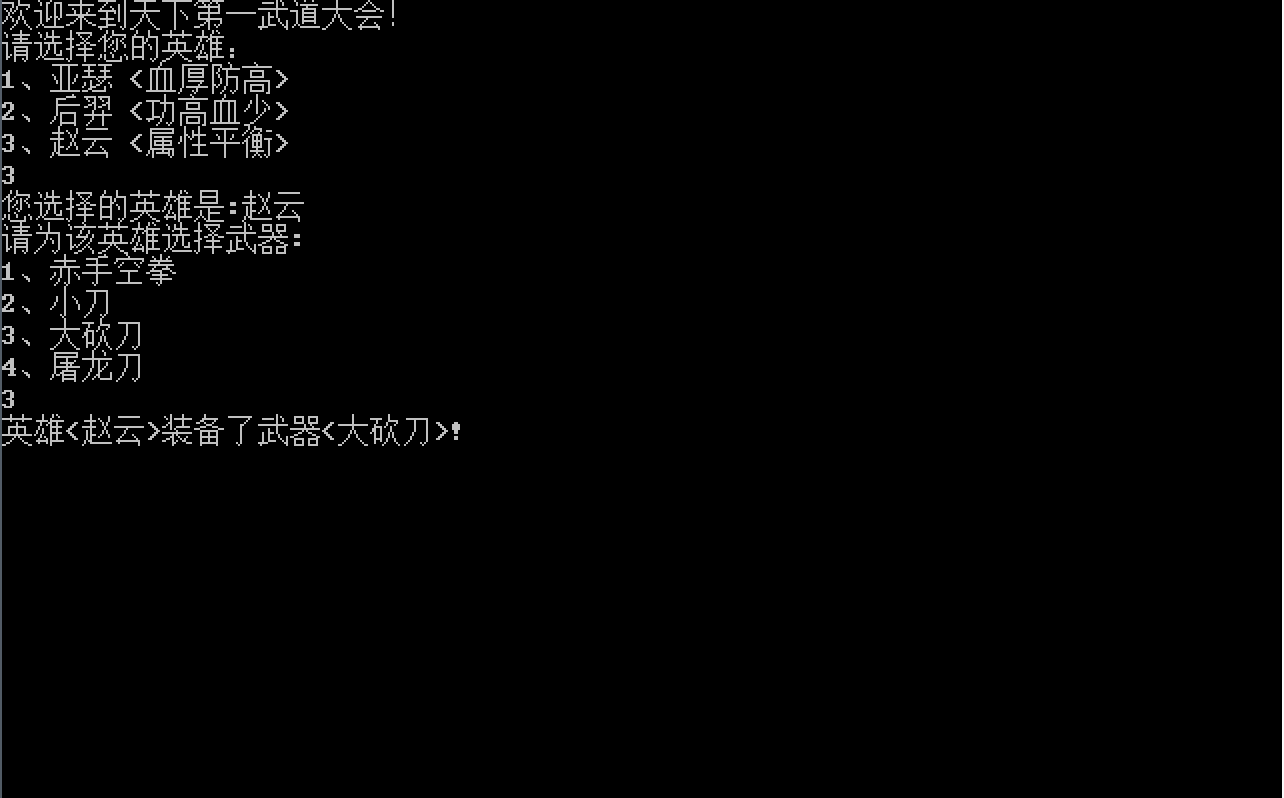
PK小游戏是最早的挂机类游戏的鼻祖，我们今天所看到所有的挂机类游戏，几乎原理都和我们这款小游戏思路一致，在游戏中添加各种随机事件会添加游戏的乐趣，当然对于PK而言我们更希望自己能够获得胜利。

## 1.2 项目截图



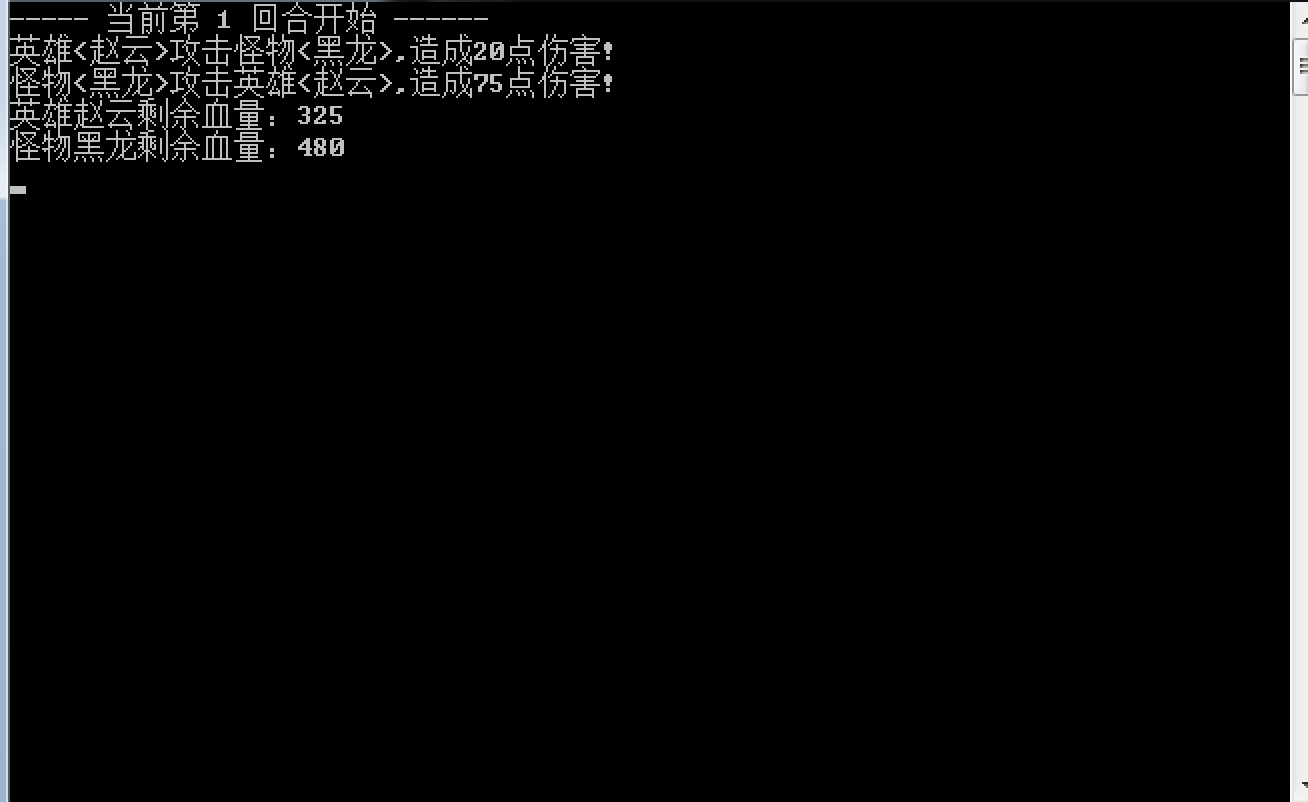
游戏开始界面是让玩家挑选英雄，当玩家挑选好英雄后，可以挑选玩家的武器，当然我们设计的比较简单，是让玩家挑选武器而不是战斗中获得武器，后期开发可以让玩家击杀一定英雄升级武器或者提供更好的武器获取方式。



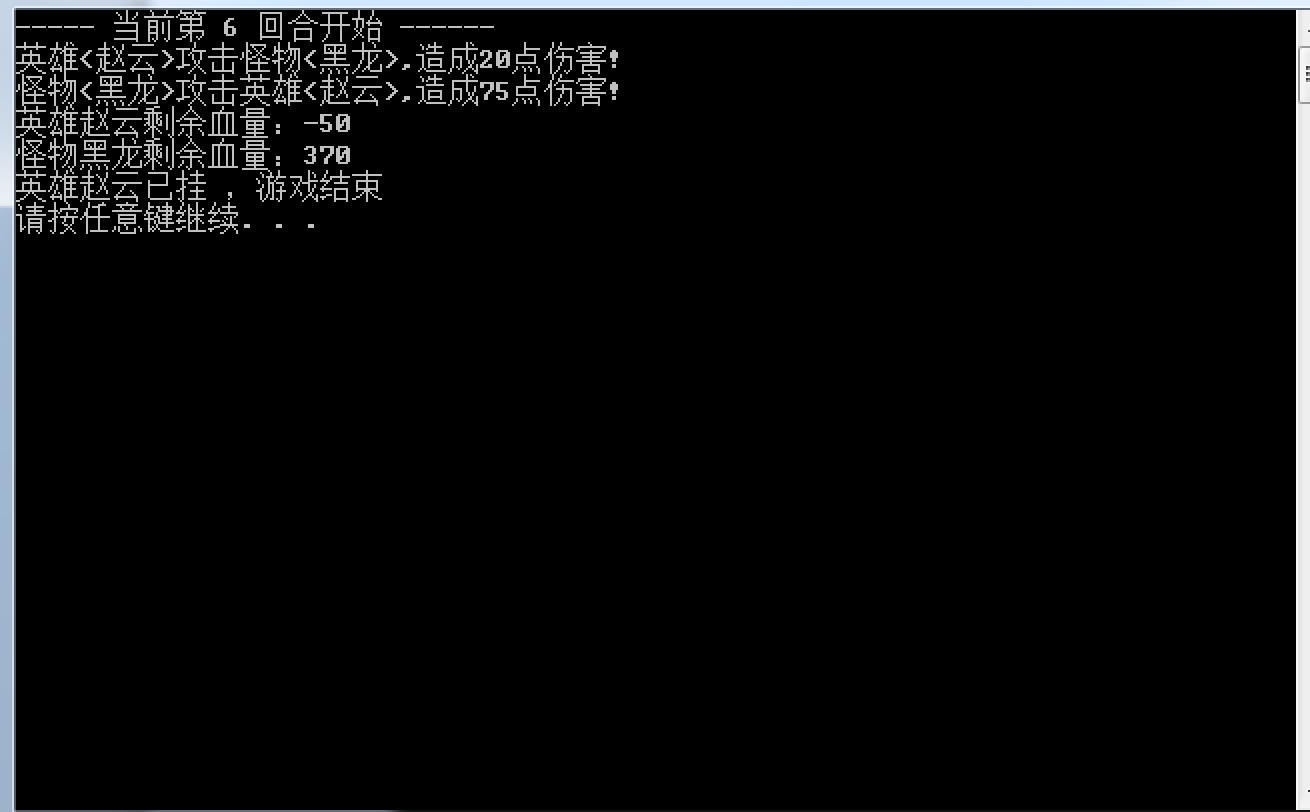


例如我们选择了赵云，并带上一把大砍刀，之后系统会为我们随机分配敌人。并进行战斗

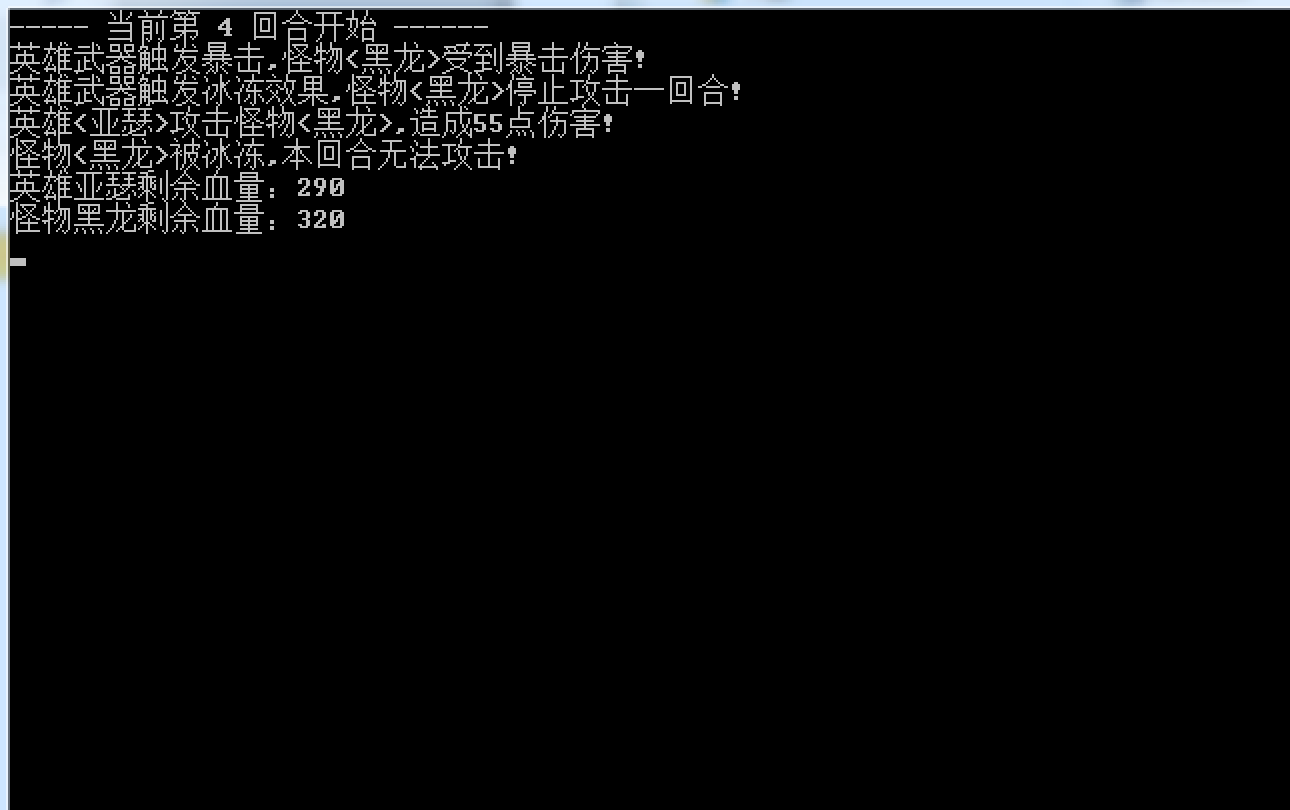
战斗时统计回合数量，玩家和敌人的剩余血量以及战斗数据。



直到一方血量小于等于0，战斗结束。



当然游戏中武器是一大亮点，我们在武器中添加了暴击、吸血、冰冻等效果来丰富游戏的玩法。



这些都可以由开发人员集思广益添加新元素，使游戏更加丰富多彩。

## 1.3游戏元素

游戏中元素分为：玩家、敌人、武器以及文件管理

**玩家**：

也就是我们是英雄Hero，对抗敌人的主角

**敌人**：

和玩家PK的对象，我们可以成为Monster怪物

**武器：**

玩家手里拿的武器，特殊武器会具有不同的特殊效果，如暴击、冰冻、吸血等。

**文件管理**

用于和文件进行沟通，本案例中的数据来源于.CSV文件，当然里面的数值由开发人员制定。

## 1.4 游戏规则

玩家上来选择人物、武器，接下来由系统随机分配一个敌人进行PK。每轮PK的数据显示到屏幕中，并由玩家点击任意键进行下一轮PK。

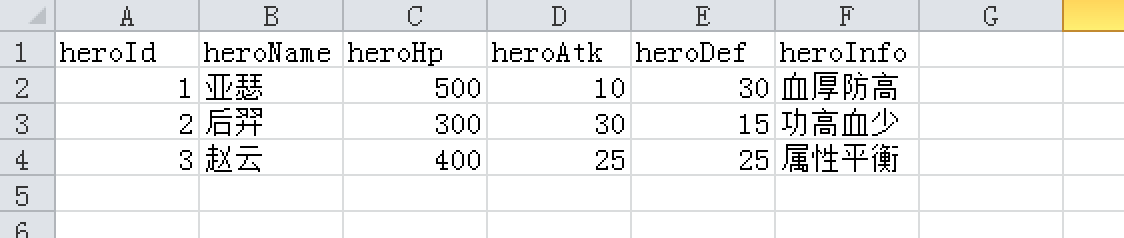
当有任意一方出现扑街，视为游戏结束。并提示胜利者。

# 2 游戏制作

## 2.1 CSV文件制作

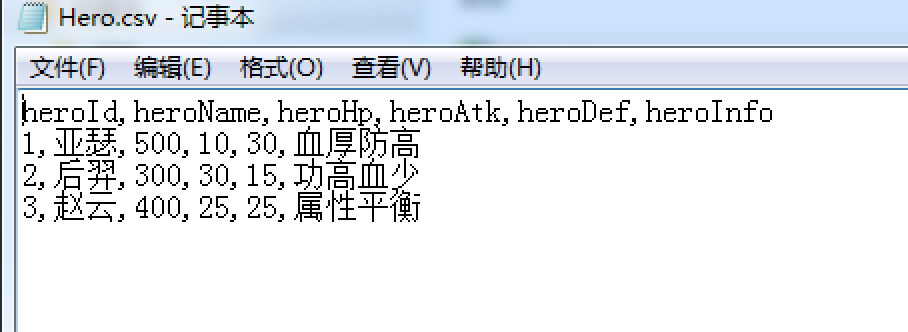
创建一张Hero英雄表，利用Excel，并且加入英雄属性字段

英雄字段（英雄ID、英雄名称、英雄血量、英雄攻击力、英雄防御力、英雄简介）



在Excel中点击另存为，格式中选择CSV为后缀的，保存该表（CSV为纯文本格式，内部的字符串都是以 ‘,’逗号进行分割）。后期可以利用记事本对CSV文件进行编辑

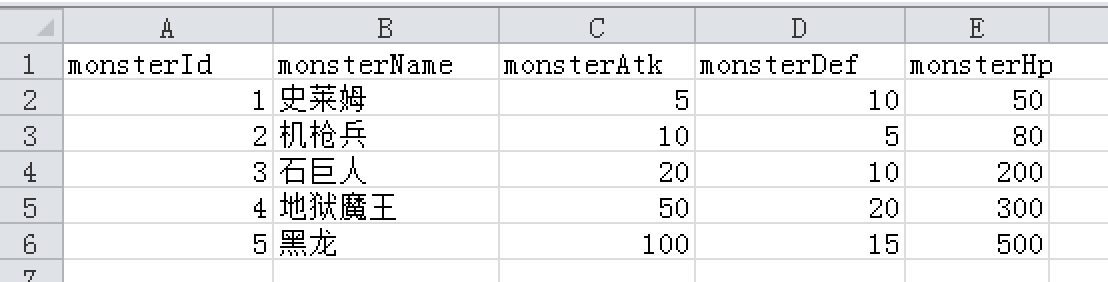
内部格式如下：



同理我们可以制作其他表：

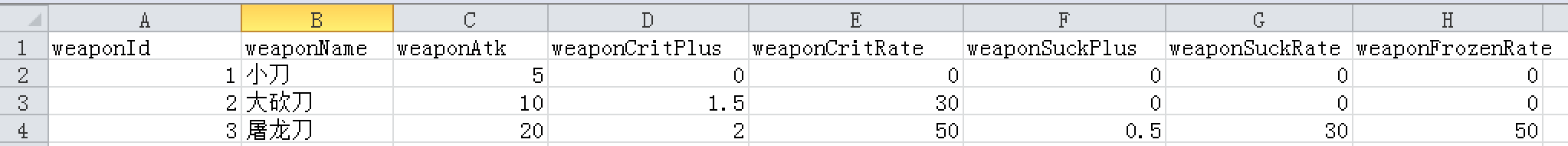
Monsters表如下：

Monsters字段（怪物Id、怪物名称、怪物攻击力、怪物防御力、怪物血量）



Weapons表如下：

Weapons字段（武器Id、武器名称、武器攻击力、武器暴击系数、武器暴击率、武器吸血系数、武器吸血率、武器冰冻率）



至此我们所需游戏中的数据已经制作完成。

## 2.2 创建项目

打开VS创建项目，项目名称任意，选择路径进行保存。

## 2.3 解析文件

本案例中难点之一就是解析文件的数据，并放入都项目中进行使用

首先我们希望上面的数据用索引的方式可以读取，那么就需要利用map结构进行封装，例如我想读取英雄表中第一个英雄的姓名，那么map的格式可以如下map<string,string> m

第一个string为key，代表英雄属性，第二个string为value，代表英雄具体属性的值，

比如我想访问英雄的姓名 可以使用 m[“heroName”]来访问

当然我们发现这样访问的话只适用于1个英雄，当我们的表中有多个英雄，就需要利用二级的map来访问具体的英雄了，map<string , map<string,string>> M 大map中的第一个string为英雄的id，第二个value就是具体的英雄字段了

于是我们比如想访问第一个英雄，那么可以利用 M[“1”][“heroName”]，代表访问1号英雄的名称，那么想访问2号英雄的名称，就可以使用M[“2”][“heroName”]，以此类推

因此可以将这个对文件做处理的事情都放在一个类中去做，fileManage，文件管理类，在这个类中

我们可以封装一个方法，如：

loadCSVData(*string* path, *map*<*string*, *map*<*string*, *string*>> &mapFileData)

这个方法中参数1 代表解析的路径名称，参数2 代表解析后的数据将放入的容器。

当然读数据，我们还是利用C++中的文件流进行读取操作。

*ifstream* ifs(path); //输入流对象 读取文件

if (!ifs.*is\_open*())

{

*cout* << "文件打开失败" << *endl*;

}

*string* firstLine;

ifs >> firstLine;

利用上面代码可以读取到文件的第一行数据，例如hero.csv中我们可以读取到该表的第一行字段。

当然我们读取的第一行字段是利用了逗号进行了拼接，所以我们需要分割字符串，并且可以将分割的数据放入到vector容器中

void FileManager::parseLineToVector(*string* line, *vector*<*string*>&v)

{

//解析单行到容器中

//分割字符串到容器中

int pos = -1;

int start = 0;

// hero\_id,hero\_name,hero\_hp,hero\_att,hero\_def,hero\_crit

while (true)

{

pos = line.*find*(",", start); //从0开始查找 ','

if (pos == -1)

{

*string* tmp = line.*substr*(start, line.*size*() - start); //查找最后一个分割

v.*push\_back*(tmp);

break; //找不到break返回

}

*string* tmp = line.*substr*(start, pos - start); //找到了,进行分割 参数1 起始位置，参数2 截取长度

v.*push\_back*(tmp);

start = pos + 1;

}

}

之后解析其他数据，并且放入到大容器中

*string* data;

while (ifs >> data)

{

*vector*<*string*> dataV; // 1 小刀 10,20

parseLineToVector(data, dataV);

//创建map容器 保存当前行数据 map<string ,string> m key 属性 value 值

*map*<*string*, *string*> mSingleLineData; //map容器 临时小容器

for (int i = 0; i < dataV.*size*(); i++)

{

mSingleLineData.*insert*(*make\_pair*(csvV[i], dataV[i]));

}

//大容器中插入数据

mapFileData.*insert*(*make\_pair*(dataV[0], mSingleLineData));

}

测试： *cout* << mapFileData["1"]["hero\_name"] << *endl*;

## 2.4 创建武器类

武器类是我们游戏的难点之二，如果所有武器都写在同一个类中，那么不利用后期的拓展维护以及会加大我们的工作量。我们所学习C++就是要学习他的复用性，可以利用多态来创建一个抽象类（Weapon），这个类中不实现任何函数，只是做声明，而在他的子类中做实现操作，每把武器我们都可以单独作为一个类，毕竟武器之间也是有属性区别的。

//武器基类

class Weapon

{

public:

//获取基础伤害

virtual int getBaseDamage() = 0;

//暴击效果 返回值大于0 触发暴击

virtual int getCrit() = 0;

//获取吸血 返回值大于0 触发吸血

virtual int getSuckBlood() = 0;

//冰冻效果 返回true 代表冰冻

virtual int getFrozen() = 0;

//触发概率

virtual bool isTrigger(int rate) = 0;

public:

int baseDamage; //基础攻击

*string* weaponName; //武器名称

int critPlus; //暴击系数

int critRate; //暴击率

int suckPlus; //吸血系数

int suckRate; //吸血率

int frozenRate; //冰冻率

};

在多态中，武器基类只是做声明，不需要做任何实现，由派生类实现

接下来可以声明我们想要创建的武器以及实现了，

例如我们可以声明一个小刀类：

class Kinfe:public Weapon

{

public:

Kinfe();

//获取基础伤害

virtual int getBaseDamage();

//暴击效果 返回值大于0 触发暴击

virtual int getCrit();

//获取吸血 返回值大于0 触发吸血

virtual int getSuckBlood();

//冰冻效果 返回值大于 代表冰冻

virtual int getFrozen();

virtual bool isTrigger(int rate); //是否触发

};

实现.cpp文件如下：

Kinfe::Kinfe()

{

FileManager fm;

*map*<*string*, *map*<*string*, *string*>> weaponData; //武器表容器

fm.loadCSVData("./Weapons.csv", weaponData);

*string* id = weaponData["1"]["weaponId"];

weaponName = weaponData[id]["weaponName"]; //名称

baseDamage = *atoi*(weaponData[id]["weaponAtk"].*c\_str*()); //基础伤害

critPlus = *atoi*(weaponData[id]["weaponCritPlus"].*c\_str*()); //暴击加成

critRate = *atoi*(weaponData[id]["weaponCritRate"].*c\_str*()); //暴击率

suckPlus = *atoi*(weaponData[id]["weaponSuckPlus"].*c\_str*()); //吸血加成

suckRate = *atoi*(weaponData[id]["weaponSuckRate"].*c\_str*()); //吸血率

frozenRate = *atoi*(weaponData[id]["weaponFrozenRate"].*c\_str*()); //冰冻率

}

//获取基础伤害

int Kinfe::getBaseDamage()

{

return this->baseDamage;

}

//获取暴击

int Kinfe::getCrit()

{

if (isTrigger(critRate))

{

return this->baseDamage \* this->critPlus;

}

else

{

return 0;

}

}

//吸血

int Kinfe::getSuckBlood()

{

if (isTrigger(suckRate))

{

return this->baseDamage \* this->suckPlus;

}

else

{

return 0;

}

}

int Kinfe::getFrozen()

{

if (isTrigger(frozenRate))

{

return 1;

}

else

{

return 0;

}

}

bool Kinfe::isTrigger(int rate)

{

int num = *rand*() % 100 + 1; //1~100

if (num < rate)

{

return true;

}

return false;

}

同理我们可以创建其他的武器，来做不同的实现。

## 2.5 英雄类实现

英雄有自己的名称、攻击力、防御力等属性，以及攻击怪物，穿戴装备等行为，因此我们可以设计如下的英雄类

class Monster;

class Hero

{

public:

Hero(int heroId);

void Attack(Monster \*monster);

void EquipWeapon(Weapon \*weapon);

public:

int heroHp;

int heroAtk;

int heroDef;

*string* heroName;

*string* heroInfo;

Weapon \*pWeapon;

};

实现如下：

Hero::Hero(int heroId)

{

FileManager fm;

*map*<*string*, *map*<*string*, *string*>> heroData; //武器表容器

fm.loadCSVData("./Hero.csv", heroData);

*string* tmpId = *std*::*to\_string*(heroId); //int 转string

*string* id = heroData[tmpId]["heroId"];

this->heroName = heroData[id]["heroName"]; //名称

this->heroAtk = *atoi*(heroData[id]["heroAtk"].*c\_str*()); //攻击力

this->heroDef = *atoi*(heroData[id]["heroDef"].*c\_str*()); //防御力

this->heroHp = *atoi*(heroData[id]["heroHp"].*c\_str*()); //血量

this->heroInfo = heroData[id]["heroInfo"]; //英雄介绍

this->pWeapon = *NULL*; //武器开始为空

}

//英雄攻击

void Hero::Attack(Monster \*monster)

{

int crit = 0;

int addHP = 0;

int forzen = 0;

int damage = 0;

if (this->pWeapon == *NULL*){

damage = this->heroAtk; //武器为空，赤手空拳

}

else{

//基础伤害

damage = this->heroAtk + this->pWeapon->getBaseDamage();

//计算吸血

addHP = this->pWeapon->getSuckBlood();

//是否定身

forzen = this->pWeapon->getFrozen();

//是否暴击

crit = this->pWeapon->getCrit();

}

if (crit){

damage = damage + crit; //如果暴击，伤害加上武器的暴击伤害

*cout* << "英雄武器触发暴击,怪物<" << monster->monsterName << ">受到暴击伤害!" << *endl*;

}

if (forzen){

*cout* << "英雄武器触发冰冻效果,怪物<" << monster->monsterName << ">停止攻击一回合!" << *endl*;

}

if (addHP){

*cout* << "英雄武器触发吸血效果,英雄血量增加" << addHP << "血!" << *endl*;

}

monster->isFrozen = forzen;

int trueDamage = (damage - monster->monsterDef > 0 ? damage - monster->monsterDef : 1);

monster->monsterHp -= trueDamage; //真实打到怪物上的伤害

this->heroHp += addHP; //吸血

*cout* << "英雄<" << this->heroName << ">攻击怪物<" << monster->monsterName << ">,造成" << trueDamage << "点伤害!" << *std*::*endl*;

}

//装备武器

void Hero::EquipWeapon(Weapon \*weapon)

{

if (weapon == *NULL*)

{

return;

}

this->pWeapon = weapon;

*cout* << "英雄<" << this->heroName << ">装备了武器<" << this->pWeapon->weaponName << ">!" << *endl*;

}

## 2.6 怪物类实现

怪物是我们的角色之一，但是怪物除了有攻防血基本属性，我们应该额外加一个是否被冰冻的属性，来判断怪物自身的状态，因此设计如下：

class Hero;

class Monster

{

public:

Monster(int monsterId);

void Attack(Hero \*hero);

public:

*string* monsterName;

int monsterHp;

int monsterAtk;

int monsterDef;

bool isFrozen;

};

.cpp实现如下：

Monster::Monster(int monsterId)

{

FileManager fm;

*map*<*string*, *map*<*string*, *string*>> monsterData; //武器表容器

fm.loadCSVData("./Monsters.csv", monsterData);

*string* tmpId = *std*::*to\_string*(monsterId); //int 转string

*string* id = monsterData[tmpId]["monsterId"];

this->monsterName = monsterData[id]["monsterName"]; //名称

this->monsterAtk = *atoi*(monsterData[id]["monsterAtk"].*c\_str*()); //攻击力

this->monsterDef = *atoi*(monsterData[id]["monsterDef"].*c\_str*()); //防御力

this->monsterHp = *atoi*(monsterData[id]["monsterHp"].*c\_str*()); //血量

this->isFrozen = false; //冰冻状态

}

void Monster::Attack(Hero \*hero)

{

if (this->isFrozen){

*std*::*cout* << "怪物<" << this->monsterName << ">被冰冻,本回合无法攻击!" << *std*::*endl*;

return;

}

//计算对英雄的伤害

int damage = this->monsterAtk - hero->heroDef > 0 ? this->monsterAtk - hero->heroDef : 1;

hero->heroHp -= damage;

*cout* << "怪物<" << this->monsterName << ">攻击英雄<" << hero->heroName << ">,造成" << damage << "点伤害!" << *endl*;

}

## 2.7 战斗实现

所有的角色类都设计好后，就差和用户进行交互了，在入口函数中我们可以添加战斗函数，来进行最后一步的实现：

void Fighting()

{

FileManager fm;

*map*<*string*, *map*<*string*, *string*>> heroData; //英雄表容器

fm.loadCSVData("./Hero.csv", heroData);

*cout* << "欢迎来到天下第一武道大会！" << *endl*;

*cout* << "请选择您的英雄：" << *endl*;

char buf[1024];

sprintf(buf, "1、%s <%s>", heroData["1"]["heroName"].*c\_str*(), heroData["1"]["heroInfo"].*c\_str*());

*cout* << buf << *endl*;

sprintf(buf, "2、%s <%s>", heroData["2"]["heroName"].*c\_str*(), heroData["2"]["heroInfo"].*c\_str*());

*cout* << buf << *endl*;

sprintf(buf, "3、%s <%s>", heroData["3"]["heroName"].*c\_str*(), heroData["3"]["heroInfo"].*c\_str*());

*cout* << buf << *endl*;

int select = 0;

*cin* >> select;

*getchar*();

Hero hero(select);

*cout* << "您选择的英雄是:" << hero.heroName << *endl*;

*cout* << "请为该英雄选择武器:" << *endl*;

*map*<*string*, *map*<*string*, *string*>> WeaponData; //英雄表容器

fm.loadCSVData("./Weapons.csv", WeaponData);

*cout* << "1、赤手空拳" << *endl*;

sprintf(buf, "2、%s ", WeaponData["1"]["weaponName"].*c\_str*());

*cout* << buf << *endl*;

sprintf(buf, "3、%s ", WeaponData["2"]["weaponName"].*c\_str*());

*cout* << buf << *endl*;

sprintf(buf, "4、%s ", WeaponData["3"]["weaponName"].*c\_str*());

*cout* << buf << *endl*;

*cin* >> select;

*getchar*();

Weapon \* weapon = *NULL*;

switch (select)

{

case 1:

*cout* << "no zuo no die why you try！";

break;

case 2:

weapon = new Kinfe;

break;

case 3:

weapon = new BroadSword;

break;

case 4:

weapon = new DragonSword;

break;

}

hero.EquipWeapon(weapon); //装备武器

int id = *rand*() % 5 + 1; //1~5

//随机创建一个怪物

Monster monster(id);

int round = 1;

while (true)

{

*getchar*();

*system*("cls");

*cout* << "----- 当前第 " << round << " 回合开始 ------" << *endl*;

if (hero.heroHp <= 0)

{

*cout* << "英雄" << hero.heroName<< "已挂 ，游戏结束" << *endl*;

break;

}

hero.Attack(&monster);

if (monster.monsterHp <= 0)

{

*cout* << "怪物" << monster.monsterName << "已挂，顺利通关" << *endl*;

break;

}

monster.Attack(&hero);

*cout* << "英雄" << hero.heroName << "剩余血量：" << hero.heroHp<< *endl*;

*cout* << "怪物" <<monster.monsterName << "剩余血量：" << monster.monsterHp << *endl*;

if (hero.heroHp <= 0)

{

*cout* << "英雄" << hero.heroName << "已挂 ，游戏结束" << *endl*;

break;

}

round++;

}

}

最后加入随机数种子，随机出一个敌人与玩家战斗，当然设计者可以有更好的思路完善本案例

*srand*((unsigned int)*time*(*NULL*));

至此，本案例制作结束。