2. 单例模式，设计游戏中的配置管理器类

```
/*
 * Singleton.h
 *
 *  Created on: 2017年7月29日
 *      Author: xiaoquan
 */

#ifndef SINGLETON_H_
#define SINGLETON_H_

class Manager {
private:
    // 静态成员变量,提供全局惟一的一个实例
    static Manager *m_pStatic;
public:
    Manager();
    virtual ~Manager();

    // 静态成员函数,提供全局访问的接口
    static Manager * GetManager();

    void test();
};

#endif /* SINGLETON_H_ */
```

```cpp
/*
 * Singleton.cpp
 *
 *  Created on: 2017年7月29日
 *      Author: xiaoquan
 */

#include "Singleton.h"
#include<iostream>
#include<cstdlib>


// 类的静态成员变量要在类体外进行定义
Manager* Manager::m_pStatic = NULL;

Manager::Manager() {
    // TODO Auto-generated constructor stub

}

Manager::~Manager() {
    // TODO Auto-generated destructor stub
}

// 静态成员函数,提供全局访问的接口
Manager * Manager::GetManager(){
    if(NULL == m_pStatic){
        m_pStatic =  new Manager();
    }
    return m_pStatic;
}

void Manager::test(){
    std::cout << "Test Manager!\n";
}
```

4. 简单工厂模式，生成各种角色。

```
/*
 * Factory.h
 *
 *  Created on: 2017年7月29日
 *      Author: xiaoquan
 */

#ifndef FACTORY_H_
#define FACTORY_H_

//抽象角色  代表角色的抽象
class Actor{
public:
    Actor(){}
    virtual ~Actor(){}
};

//各种实例角色

//代表Player角色的实现
class Player:public Actor{
public:
    Player();
    ~Player();
};

//代表Demon角色的实现
class Demon:public Actor{
public:
    Demon();
    ~Demon();
};

//代表Fairy角色的实现
class Fairy:public Actor{
public:
    Fairy();
    ~Fairy();
};

//代表Civilian角色的实现
class Civilian:public Actor{
public:
    Civilian();
    ~Civilian();
};

//工厂的抽象类,生产角色
class Factory {
public:
    Factory(){}
    virtual ~Factory(){}
```

```
    virtual Actor * CreateActor(){ return new Actor();}
};

// 生产Player
class PlayerFactory:public Factory{
public:
    PlayerFactory();
    ~PlayerFactory();

    virtual Actor * CreateActor();
};

// 生产Demon
class DemonFactory:public Factory{
public:
    DemonFactory();
    ~DemonFactory();

    virtual Actor * CreateActor();
};

// 生产Fairy
class FairyFactory:public Factory{
public:
    FairyFactory();
    ~FairyFactory();

    virtual Actor * CreateActor();
};

// 生产Civilian
class CivilianFactory:public Factory{
public:
    CivilianFactory();
    ~CivilianFactory();

    virtual Actor *CreateActor();
};

#endif /* FACTORY_H_ */
```

```cpp
/*
 * Factory.cpp
 *
 *  Created on: 2017年7月29日
 *      Author: xiaoquan
 */

#include "Factory.h"
#include<iostream>

using namespace std;

Player::Player(){
    cout << "Construct a Player! \n";
}
Player::~Player(){
    cout << "Destruct a Player!\n";
}

Demon::Demon(){
    cout << "Construct a Demon!\n";
}

Demon::~Demon(){
    cout << "Destruct a Demon!\n";
}

Fairy::Fairy(){
    cout << "Construct a Fairy!\n";
}

Fairy::~Fairy(){
    cout << "Destruct a Fairy!\n";
}

Civilian::Civilian(){
    cout << "Construct a Civilian!\n";
}

Civilian::~Civilian(){
    cout << "Destruct a Civilian!\n";
}

PlayerFactory::PlayerFactory(){
    cout << "Construct a PlayerFactory!\n";
}

PlayerFactory::~PlayerFactory(){
    cout << "Destruct a PlayerFactory!\n";
}

Actor* PlayerFactory::CreateActor(){

    return new Player();
```

```
}

DemonFactory::DemonFactory(){
    cout << "Construct a DemonFactory!\n";
}

DemonFactory::~DemonFactory(){
    cout << "Destruct a DemonFactory!\n";
}

Actor * DemonFactory::CreateActor(){
    return new Demon();
}

FairyFactory::FairyFactory(){
    cout << "Construct a FairyFactory!\n";
}

FairyFactory::~FairyFactory(){
    cout << "Destruct a FairyFactory!\n";
}

Actor * FairyFactory::CreateActor(){
    return new Fairy();
}

CivilianFactory::CivilianFactory(){
    cout << "Contruct a CivilianFactory!\n";
}

CivilianFactory::~CivilianFactory(){
    cout << "Destruct a CivilianFactory!\n";
}

Actor * CivilianFactory::CreateActor(){
    return new Civilian();
}
```