****

**面向对象程序设计上机实验报告**

实验6：动态游戏规则的存取实现

**学院名称 智能与计算学部**

**专 业 软件工程**

**学生姓名 陈昊昆**

**学 号 3021001196**

**年 级 2021级**

**班 级 3班**

**时 间 2022年 4月23日**

1. **实验目的**

理解文件的读写过程；

培养参数统一配置的思维；

练习使用DEBUG工具调试程序；

1. **实验内容**

1) 修改前面实验的一款游戏，游戏中每个对象配置从一个配置文件中读取，配置文件格式类似如下：

#整个游戏的配置文件

#这个文件里面可能有注释，以#开头表示注释

#基地的属性

#每行形式为 [对象类型].[属性]=[整数] 其中对象类型和EnumObjectType 中的枚举类型字符完全相同。

#[属性]取值为 health range strength

#测试的时候，属性的整数值可能会改变

base.health=1000

base.range=0

base.strength=0

#重型坦克的属性

heavyTank.health=500

heavyTank.range=100

heavyTank.strength=100

mediumTank.health=300

mediumTank.range=50

mediumTank.strength=50

......

2) 程序必须从配置文件中动态的读取配置规则，完成每个对象的攻击情况，并且通过测试用例。

1. **程序实现**

管理玩家信息

**public** **class** BattleField {

**private** **static** Param *param*=**null**;

**private** **static** List<Player> *allPlayer*=**new** ArrayList<Player>();

**private** **static** List<GameObject> *allGameObjectList* = **new** ArrayList<GameObject>();

// 初始化方法，读取指定配置文件，并将其解析为 Param 对象

**public** **static** **void** init(String config) **throws** FileNotFoundException, IOException {

*param*=**new** Param(config);

}

**public** **static** Param getParam() {

**return** *param*;

}

**public** **static** List<Player> getAllPlayer() {

**return** *allPlayer*;

}

**public** **static** List<GameObject> getAllGameObject(){

**return** *allGameObjectList*;

}

**public** **static** **void** addGameObject(GameObject gameObject) {

*allGameObjectList*.add(gameObject);

}

// 创建一个名为 name 的新 Player 对象，并将其添加到 allPlayer 静态变量中，然后返回该对象

**public** **static** Player createPlayer(String name) {

Player player=**new** Player(name);

*allPlayer*.add(player);

**return** player;

}

// 创建一个 GameObjectVo 对象，表示一个玩家的基地，然后将其传递给 GameBase 构造函数，返回构造出的 GameBase 对象

**public** **static** GameBase createGameBase(Player player,**int** x,**int** y) {

GameObjectVo gameObjectVo=**new** GameObjectVo(player,EnumObjectType.***base***,x,y);

**return** **new** GameBase(gameObjectVo);

}

}

读取游戏参数配置文件，并将其解析为参数对象

**public** **class** Param {

**public** **static** **final** String ***HEALTH*** = "health";

**public** **static** **final** String ***RANGE*** = "range";

**public** **static** **final** String ***STRENGTH*** = "strength";

// 存储所有参数的 Map 对象，其中键为字符串类型，值为另一个 Map 对象，键为字符串类型，值为整型数值

**private** Map<String, Map<String, Integer>> allParamsMap = **new** HashMap<String, Map<String, Integer>>();

**public** **int** getIntParam(EnumObjectType objectType, String paramName) {

// 获取指定对象类型和参数名的整型参数值

**if** (!allParamsMap.containsKey(objectType.toString())) {

**throw** **new** IllegalArgumentException(objectType.toString());

}

Map<String, Integer> paramMap = allParamsMap.get(objectType.toString());

**if** (!paramMap.containsKey(paramName)) {

**throw** **new** IllegalArgumentException(paramName);

}

**return** paramMap.get(paramName);

}

// 构造函数，根据指定配置文件名读取配置文件，并将其解析为参数对象

**public** Param(String configName) **throws** FileNotFoundException, IOException {

// **TODO** Auto-generated constructor stub

**try** (Reader reader = **new** FileReader(configName);

// LineNumberReader 必须在其他流基础上构建

LineNumberReader lineReader = **new** LineNumberReader(reader);) {

String line = lineReader.readLine();

**while** (line != **null**) {

**if** (line.startsWith("#") || line.trim().length() == 0) {

} **else** {

String[] strs = line.split("=");

String left = strs[0];

//String s="";

**int** paramValue = Integer.*valueOf*(strs[1].trim());

String[] names = left.split("\\.");

String type = names[0];

String paramName = names[1];

Map<String, Integer> paramMap = **null**;

**if** (allParamsMap.containsKey(type)) {

paramMap = allParamsMap.get(type);

} **else** {

paramMap = **new** HashMap<String, Integer>();

}

paramMap.put(paramName, paramValue);

allParamsMap.put(type, paramMap);

}

line = lineReader.readLine();

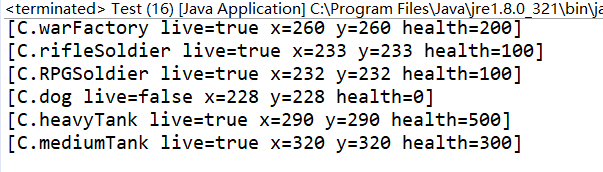
}

}

}

}

1. **实验结果**



1. **实验中遇到的问题及解决方法**

添加已存在的游戏对象导致的异常：在 addGameObject() 方法中判断是否已经存在相同的游戏对象，如果已经存在则抛出异常或不进行添加。

创建已存在的玩家或游戏对象导致的异常：在 createPlayer() 和 createGameBase() 方法中，可以通过遍历 allPlayer 和 allGameObjectList 列表判断是否已经存在相同的对象，如果已经存在则抛出异常或不进行创建。