

**实验报告**

|  |  |
| --- | --- |
| **课程名称**: | 设计模式 |
| **开课学期**: | 2020-2021学年 第2学期 |
| **专业**: | 软件工程 |
| **班级年级**: | 2018级 |
| **学生姓名**: | 宋行健 |
| **学号:** | 222018321062006 |
| **实验教师:** | 王晓蒙 |

**计算机与信息科学学院 软件学院**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验项目名称 | | 实验3 创建型模式2 | | | |
| 实验时间 | | 2021/3/30 | Type | \*验证性□设计性 □综合性 | |
| 1. 思路：  首先对老师发的代码进行逐句研读，了解代码架构，并绘制类图。通过类图的展示，程序的逻辑逐渐明显。程序主要被封装为3个主要控制类，分别为GameLoopTimer分管整个游戏的时间流转，GameController分管键盘消息的预处理，GameManager分管初始化构建游戏中的对象和生产管理对象。  另外，程序中建立了一个抽象类Entity，让Coin、Player、Map、Number这些实体类来继承整个抽象Entity类。之所以这样做的原因是上述的这些实体类都有两个共同的主要属性，Image图片和Point2D位置。另外在Entity类中还有一个核心方法，isOverlap(Entity e)，用来判断两个对象是否碰撞重叠，在该程序中主要用在判断是**否接到金币**和**金币是否落地**。  2. 类图：    3. 代码分析  3.1 原型模式设计  在代码中主要是Coin类使用了原型模式，在GameManager类中首先创建了Coin的原型，在GameManager中的refresh()方法中进行Coin原型的深复制，即创建了另一个单独的Coin对象。在深复制的时候，通过copy.setPosition语句重新设置深复制后的Coin的位置。    图 1 Coin的原型创建（GameManager.java）    图 2 Coin深复制（GameManager.java）  另外，在Coin类中，对clone函数进行了重写，使得它可以进行深复制，如果不重写的话，就会变成复制指针，就是浅复制了，当修改浅复制后的Coin的位置，会将原始的Coin对象的位置也修改，就不能实现接金币游戏中产生金币的要求。    图 3 clone函数重写（Coin.java）  3.2 单例模式设计  **（一）饿汉模式**  在程序中，GameController和GameManager使用的是饿汉模式创建的。是在类加载后直接创建实例。另外，这两个单例模式的使用均为饿汉模式中的直接赋初值的方法，这两个单例对象都比较简单，因此这种方式比较适合。      **（二）懒汉模式**  在程序中，Render类是是使用懒汉模式创建的单例，是按需在调用getRender() 方法的时候根据情况创建。在getRender()中判断当前实例是否已经被创建，如果为空则调用构造器创建对象。这个单例模式，使用了**双重检查**这种线程安全的方式，但是效率较低，每次只允许一个线程访问单例对象。  如果改进的话，我认为Render也可以使用“饿汉模式”进行创建，因为这个对象并不是很复杂。    3.3 碰撞检测    图 4 碰撞代码  上图展示的是碰撞检测的代码，通过水平方向和竖直方向各两个判断，判断两个对象是否重叠（碰撞）。其示意图如下所示。只有在同时满足这四个判定条件时，碰撞检测才为真。    图 5 碰撞检测示意图  4. 改进  4.1 逻辑改进  **（一）金币重力加速度**  首先，对于金币的产生，它们是从上掉落的，按说应该是符合自由落体运动的，即在掉落是是有加速度的，为了模拟这一物理过程，我为金币增加了一个恒定的加速度，setSpeed(speed + 1)，在每个计时器单位内速度增加1。    **（二）人物启动加速度**  人物启动加速度的原理与金币自由落体的原理是相同的，都是通过在每个时间单位内增加速度的数值，这里我采用的都是恒定加速度。不过人物启动加速度的时候要注意一个细节，就是在人物停止的时候要将储存的速度重置。    **（三）人物停止，脚步停止**  在程序中我发现了这样一个小Bug，就是人物在停止时依然会进行运动的动画，这是因为Rider的move()函数是一直在运行的，其中的lookList.nextFrame();语句会一直切换帧。为了解决这个问题，我在Rider的DIRECTION中又添加了一个状态STOP，当人物对象停止时设置状态为 STOP，同时在move()中开辟一个条件分支，当状态为STOP的时候不进行动画帧的切换。但是后来又遇到了一个问题就是在停止前不知道人物的朝向，为了解决这个问题，我又建立了一个变量proDir来储存人物在停止前的朝向。    **（四）金币掉落密度调节**  在程序运行时，我发现掉落产生的金币过多，游戏没有挑战性，我想让金币产生的少一点。为了解决这个问题，我在金币生成的函数里加了一层过滤。这个过滤的思想和第一次实验中减慢对象速度的原理是一样的。我设置了两个int类型的变量，coinSpeedController是一个跟随计时器在不断增加的数，coinSpeed是控制金币产生量的变量，coinSpeed越大，金币产生的速度越慢。这里主要用到了整除取模的原理进行选择性过滤产生金币。    4.2 性能改进    这里的如果file为空会报一个Null错误，所以为了提高代码的强壮性，可以将其改为Objects.requireNonNull(file.listFiles())。    这里在生成金币的时候，有可能出现copy复制的对象为Null的情况，为了避免这种情况发生，应该把下面的两句话放在try中，和copy一起进行。    这里在初始化积分牌临时变量的时候可以使用基础变量int替代，int为基本变量类型，而Integer是他的封装，这里只是为了实现一个临时记分的功能，因此为了效率更高，使用int就可以了。  5. 运行结果    6. 总结  该项目是原型模式和单例模式的融合实现。其中Coin的生成使用的是原型模式，最初构建一个Coin原型，之后通过对这个原型进行深复制从而产生金币。对于Render，GameController和GameManager使用的是单例模式，因为这些类的对象在整个程序中都只能有一个，否则就会出现互相影响的问题。  在GameManager类中首先创建了Coin的原型，在GameManager中的refresh()方法中进行Coin原型的深复制，即创建了另一个单独的Coin对象。在深复制的时候，通过copy.setPosition()语句重新设置深复制后的Coin的位置。另外，在Coin类中，对clone函数进行了重写，使得它可以进行深复制，如果不重写的话，就会变成复制指针，就是浅复制了，当修改浅复制后的Coin的位置，会将原始的Coin对象的位置也修改，就不能实现接金币游戏中产生金币的要求。  在程序中，GameController和GameManager使用的是单例模式中的饿汉模式方法创建的，采用的是在类加载后直接创建实例的方式。而Render类是是使用懒汉模式创建的单例，是按需在调用getRender()方法的时候根据情况创建。在getRender()中使用了双重检查判断当前实例是否已经被创建，如果为空则调用构造器创建对象。  另外，老师在代码中使用**速度（speed）**的方式进行对人物对象的移动是十分巧妙的。一方面是使代码更加简洁直观，另一方面可以减少卡顿。这些都是值得学习和积累经验的地方。 | | | | | |
| 教师评阅 | 实验内容（60%）： | | | |  |
| 实验思路（20%）： | | | |  |
| 实验总结（20%）： | | | |  |
| 实验成绩： | | | | |