程序设计A　Lab2

TA:

- 14302010005@fudan.edu.cn 姜卓立
- 14302010040@fudan.edu.cn 武多才
- 14302010042@fudan.edu.cn 何培剑

1 摘要

本次lab主要涉及以下2个部分：

1. 继续重构lab11
2. 尝试RoboCode编程

2 重构lab11

结合课堂上的 知识，本次重构给大家三个考虑的基本点。

2.1 如何设计使得易于向迷宫中添加个体

回顾一下我们的迷宫，一开始有玩家，然后又有了怪物，然后又出现了不同种类的怪物。这些个体都会处理其它个体与自己接触的事件。那要如何设计才可以使得我们可以之后也可以很方便的添加其它的个体呢，比如再添加一个NPC，玩家只要接触到NPC就会触发对话。

课上讲过战场战士的列子，这就很熟悉了，首先我们要有一个超类Entity，然后他有handleEvent方法。如下：

```
public abstract class Entity {
  void handleEvent(Entiy other);
}
```

在主逻辑中，每个我们只面向entity编程，当一个entity走到一个格子（Place）时，就触发在该格子里的其它所有个体的handleEvent方法。核心代码如下：

```
private void triggerEvents(Place dst, Entity curEntity) {
        // trigger event for every entity on this place:dst
        int num = dst.getEntiesNum();
        for (int i = 0; i < num; i++) {
            Entity ob = dst.getEntity(i);
            ob.handleEvent(curEntity);
        }
 }
```

自然，添加新的entity后，我们的主逻辑核心是不需要修改的。如此做到可扩展新个体，而不去修改复杂的游戏流程逻辑。

2.2 还要区分出不会处理接触事件的个体。

很可惜的是，并不是所有个体都会处理接触事件，例如迷宫游戏的宝箱，任何个体接触到它都不会发生任何事（原来的要求是按按键p捡宝箱的）。这要怎么办。一种方法是宝箱依然继承Entity类，但是handleEvent里不做任何处理，就可以达到该要求。

这样可以实现功能，优点是简单，但是造成一点逻辑缺陷。

另一种方法也是课上举过的例子：使用接口。我们定义HasEvent接口，要处理接触事件的个体就实现该接口，否则不实现。如下：

```
// enity
public abstract class Entity {


}

//
public interface HasEvent {
  void handleEvent(Entiy other);
}

// Player
public class Player extends Enity implements HasEvent {
  void handleEvent(Entiy other) {
  }
}

// Treasure
public class Treasure extends Enity {

}

// core main logic
private void triggerEvents(Place dst, Entity curEntity) {
        // trigger event for every entity on this place:dst
        int num = dst.getEntiesNum();
        for (int i = 0; i < num; i++) {
            Entity ob = dst.getEntity(i);
            if (ob instanceof HasEvent) // has event ?
                ((HasEvent)ob).handleEvent(curEntity);
        }
 }
```

在这之后，可以方便的扩展新的有事件和无事件的entity子类。

2.3 保持游戏逻辑中立

之前的lab一直是做针对游戏逻辑内部的设计。之后我们需考虑使用UI将游戏展现出来，一个好的设计要使得其游戏控制逻辑与具体的UI形式是无关的，保持中立性。

为了实现这一点我们可以使用外观模式，即我们需要有一个类将封装所有的对该游戏系统的调用。

提供一些建议：

1. 分包，将游戏分为ui包和core包。

2. 写一个类封装所有ui层可能对游戏核心（core包）产生的调用。例如：

```
public class Maze {

  // some methods demo
  // 1. createGame
  // 2. startSomeLevel
  // 3. movePlayer
  // 4. moveMonster
  // 5. getGameView
  // 6. getScoreMessage
  // 7. isGameOver
  // other methods
}
```

如此做之后，需要注意如下问题。

1. 所有的信息展示只能在ui包内实现，core包内不应当产生任何形式输出，而应只是维护状态。
2. core包内不应当包含任何与特定UI形式相关的内容。例如不能产生对fx或swing等UI组件的依赖，例如lab11中HasIcon中的char需要重构，这是针对cmd UI的素材，现在需要将其换成抽象的自定义Icon。
3. UI包内所有类不能直接调用core包内的类，只能调用前面的外观Maze类

如果发现自己写着写着犯了以上问题，你的设计就耦合在一起了，而使得core不能直接被不同的UI复用。

3 RoboCode编程

这部分我们尝试RoboCode编程。一是体会RoboCode良好的设计，使得我们可以方便向战场中添加新的robot，其次是RoboCode编程本身十分有趣且具有挑战性。看一些介绍：

> 1. Robocode is a programming game, where the goal is to develop a robot battle tank to battle against other tanks in Java or .NET.
> 2. Schools and universities are using Robocode as part of teaching how to program, but also for studying artificial intelligence (AI).
> 3. Creating a robot can be easy. Making your robot a winner is not. You can spend only a few minutes on it, or you can spend months and months.

下面提供一个非常好的教程。

3.1 Install

If Java has been correctly installed and configured, you should be able to run the installer by double-clicking it(robocode-a.b.c.d-setup.jar, a.b.c.d is version number, download it from ftp). If the installer doesn't display, run the jar file from command line.

3.2 The Robot Editor

The first step is to open up the Robot Editor. From the main Robocode screen, click on the **Robot** menu, then select **Source Editor**.

When the editor window comes up, click on the **File** menu, then select **New Robot**.

In the dialogs that follow, type in a name for your robot, and enter your initials.

Voila! You now see the code for your own robot.

3.3 A new robot

1.
```
1  package pkg;
2  import robocode.*;
3
4  public class YourRobotNameHere extends Robot {
5
6      public void run() {
7          while (true) {
8              ahead(100);
9              turnGunRight(360);
10             back(100);
11             turnGunRight(360);
12         }
13     }
14
15     public void onScannedRobot(ScannedRobotEvent e) {
16         fire(1);
17     }
18 }
```

We're only concerned with lines 8-11 and 16 here... you won't need to change anything else. Not that much, right?

By the way, if you're **really** concerned about the rest of it, here it is:

- `import robocode.*;` – Says that you're going to use Robocode objects in your robot.
- `public class MyFirstRobot extends Robot` – Says the object I'm describing here is a type of `Robot`, named `MyFirstRobot`.
- `public void run() { ... }` – The game calls your run() method when the battle begins.
- `{ ... }` – "Curly brackets" (**{ }**) group things together. In this case, they're grouping together all the code for the robot.

3.4 Let's move somewhere

Let's add a couple lines so that it will do something.

First, we'll examine the run() method:

```
while(true) {
    ahead(100);
    turnGunRight(360);
    back(100);
    turnGunRight(360);
}
```

`while(true) { ... }` means: "Do the stuff inside my curly brackets, forever".

So this robot will:

1. Move ahead 100 pixels.
2. Turn the gun right by 360 degrees.

3. Move back 100 pixels.
4. Turn the gun right by 360 degrees again.

The robot will continue doing this over and over and over, until it dies, due to the `while(true)` statement.

Not so bad, right?

3.5 Fire at will!

When our radar scans a robot, we want to fire:

```
public void onScannedRobot(ScannedRobotEvent e) {
    fire(1);
}
```

The game calls your `onScannedRobot()` method whenever – during one of the actions – you see another robot. It sends along an event that can tell us lots of information about the robot – its name, how much life it has, where it is, where it's heading, how fast it's going, etc.

However, since this is a simple robot, we're not going to look at any of that stuff. Let's just fire!

3.6 Compile your robot

First, save your robot by selecting the **Save** in the **File** menu. Follow the prompts to save your robot.

Now, compile it by selecting **Compile** in the **Compiler** menu.

3.7 Start the battle

If your robot compiles without any errors, you can start a new battle with your robot.

Start a new battle by selecting **New** in the **Battle** menu. (If you cannot see your robot, you might have to refresh the list of robots by selecting Options -> Clean robot cache). Add your robot to the battle together with at least one other robot as e.g. sample.Target, and press the **Start Battle** button to let the games begin!

3.8 Coding in Intellij

Robocode also supports developing robots using external IDEs like e.g. Eclipse, IntelliJ IDEA, NetBeans, Visual Studio etc., which supports the developer much better than the robot editor in Robocode.

Here is the tutorial of coding in Intellij: https://www.youtube.com/watch?v=naGH2ikkR-g

4 声明