Assignment 2, part II: Road Fighter Extended

Fecha de entrega: 29 November 2021, 9:00

Objetivo: Inheritance, polymorphism, abstract classes and interfaces.

1. Extensiones

In this assignment, we will extend the code with new functionality. But first a warning:

IMPORTANT: Any of the following, on its own, is sufficient reason to fail:

- breaking encapsulation,
- the use of methods that return lists,
- the use of instanceof or getClass, since identifying the dynamic types of objects is simply a way of avoiding the use of polymorphism and dynamic binding, i.e. avoiding the use of OOP. The use of a "DIY" instanceof (e.g. each subclass of GameElement has a set of methods isX, one for each subclass of GameElement, where in class Y, the method isX returns true if $X = Y$ and false if $X \neq Y$) is even worse since it is simply a clumsier, more verbose, way of doing the same thing.

1.1. Wall game element, `shoot` command and InstantAction interface

Wall. We first add a new type of obstacle called Wall having the same behaviour as Obstacle except for the fact that it has a resistance of 3 instead of 1. Three different symbols – ░, ▒ and █ – are used to represent objects of this class, depending on how many lives the Wall object has left (see the file `symbols.txt` to for the exact symbols). As explained below, the player obtains coins from destroying obstacles by shooting them, in particular, the player obtains 5 coins from destroying a wall (no coins are obtained from destroying a normal obstacle).

`Shoot` command. The `shoot` command removes a life from the first obstacle found in the same lane to the right of the player and costs the player 1 coin. The range of a shot is the visible part of the road, i.e. it has no effect on obstacles that are not inside the visibility. ~~Note that use of the `shoot` command does not involve a cycle.~~ Here follows a sample execution:

Different representations of a Wall

```
Command >
q
[DEBUG] Executing: q
Distance: 6
Coins: 10
Cycle: 3
Total obstacles: 1
Total coins: 0
  ═════════════════════════════════════════════════════════════════
      >                       █                       ¦
   ─────── ─────── ─────── ─────── ─────── ─────── ─────── ───────
                                                      ¦
   ─────── ─────── ─────── ─────── ─────── ─────── ─────── ───────
                                                      ¦
  ═════════════════════════════════════════════════════════════════

Command >
s
[DEBUG] Executing: s
Distance: 6
Coins: 9
Cycle: 4
Total obstacles: 1
Total coins: 0
  ═════════════════════════════════════════════════════════════════
      >                       ▓                       ¦
   ─────── ─────── ─────── ─────── ─────── ─────── ─────── ───────
                                                      ¦
   ─────── ─────── ─────── ─────── ─────── ─────── ─────── ───────
                                                      ¦
  ═════════════════════════════════════════════════════════════════

Command >
s
[DEBUG] Executing: s
Distance: 6
Coins: 8
Cycle: 5
Total obstacles: 1
Total coins: 0
  ═════════════════════════════════════════════════════════════════
      >                       ░                       ¦
   ─────── ─────── ─────── ─────── ─────── ─────── ─────── ───────
                                                      ¦
   ─────── ─────── ─────── ─────── ─────── ─────── ─────── ───────
                                                      ¦
  ═════════════════════════════════════════════════════════════════

Command >
s
[DEBUG] Executing: s
Distance: 6
Coins: 12
Cycle: 6
Total obstacles: 0
Total coins: 0
  ═════════════════════════════════════════════════════════════════
      >                                               ¦
   ─────── ─────── ─────── ─────── ─────── ─────── ─────── ───────
                                                      ¦
   ─────── ─────── ─────── ─────── ─────── ─────── ─────── ───────
                                                      ¦
  ═════════════════════════════════════════════════════════════════
```

After executing the shoot action, the game elements should be updated and any dead objects eliminated. You may, or may not, choose to also move the player after the shot (but before updating the other game elements).

Collider interface. The `shoot` command only affects obstacles. To implement a behaviour which only affects certain game elements, we can use dynamic binding. To this end, to the Collider interface we add a method receiveShot() that can be given a default empty body in GameElement to be overwritten in the appropriate GameElement subclasses.

InstantAction interface. The `shoot` command does not create a bullet object that travel towards the obstacle and then collides with it but, instead, acts instantaneously like a laser. We create a new interface called InstantAction to represent the propagation of an effect to a game element without there being a collision.

```
package es.ucm.tp1.logic;

public interface InstantAction {
  void execute(Game game);
}
```

The implementation of the execute method in each concrete instantaneous action, in this case the ShootAction, determines which game element (or elements) is affected by that action, in order to call the appropriate method, in this case receiveShot(), on that game element (or those game elements). The instantaneous action objects should be created in the corresponding command.

1.2. The SuperCoin game element

Next, we create a new game element subclass called SuperCoin, represented by the symbol $, which gives the player 1000 coins if the car drives over it. There can be only one supercoin per game; one way of ensuring this is by endowing the SuperCoin class with a static boolean attribute. When a supercoin is present on the road, the information `Supercoin is present` must be displayed each time the road is displayed. Here follows an example in which the car drives over a supercoin:

```
[DEBUG] Executing:
Distance: 26
Coins: 5
Cycle: 4
Total obstacles: 20
Total coins: 11
Supercoin is present
Elapsed Time: 0.71 s
  ══════════════════════════════════════════════════════════════════
                     ¢                       ☺       ░       ░
   _______ _______ _______ _______ _______ _______ _______ _______
      >              $
   _______ _______ _______ _______ _______ _______ _______ _______
      ░       ¢              >>>      ¢       ¢
  ══════════════════════════════════════════════════════════════════

Command >

[DEBUG] Executing:
Distance: 25
Coins: 5
Cycle: 5
Total obstacles: 20
Total coins: 11
```

```
Supercoin is present
Elapsed Time: 1.66 s
  ═════════════════════════════════════════════════════════════════
              ¢                               ░       ░
   _______ _______ _______ _______ _______ _______ _______ _______
      >       $                       ☺                       ¢
   _______ _______ _______ _______ _______ _______ _______ _______
      ¢              >>>      ¢       ¢                       ░
  ═════════════════════════════════════════════════════════════════

Command >

[DEBUG] Executing:
Distance: 24
Coins: 1005
Cycle: 6
Total obstacles: 20
Total coins: 11
Elapsed Time: 5.04 s
  ═════════════════════════════════════════════════════════════════
      ¢                               ░       ░              >>>
   _______ _______ _______ _______ _______ _______ _______ _______
      >                                               ¢       ¢
   _______ _______ _______ _______ _______ _______ _______ _______
             >>>      ¢       ¢                       ░
  ═════════════════════════════════════════════════════════════════
```

1.3. The Turbo game element

When the car passes over this game element, is represented by the symbol >>>, it jumps forward three squares; note that this does not change the requirement that the car is always displayed in the leftmost column. If, in the process of jumping forward three squares, the car passes over some other game element, this has no effect (there is no collision or coin grabbing).~~, including if that game element is on the target square of the jump. In the latter case, the car and the other game element share the same square (the car is displayed).~~ However, if there is a game element on the target square of the jump there is a collision. Implementing this requires checking for collisions twice in each cycle. This extra collision detection could either take place between moving the other game elements and printing, or at the start of the next cycle before moving the player. Here follows an example in which the car drives over a turbo:

```
Command >

[DEBUG] Executing:
Distance: 25
Coins: 5
Cycle: 5
Total obstacles: 20
Total coins: 11
Supercoin is present
Elapsed Time: 1.38 s
  ═════════════════════════════════════════════════════════════════
              ¢                               ░       ░
   _______ _______ _______ _______ _______ _______ _______ _______
      >       $                       ☺                       ¢
   _______ _______ _______ _______ _______ _______ _______ _______
      ¢              >>>      ¢       ¢                       ░
  ═════════════════════════════════════════════════════════════════

Command >
a
[DEBUG] Executing: a
Distance: 24
Coins: 5
Cycle: 5
```

```
Total obstacles: 20
Total coins: 11
Supercoin is present
Elapsed Time: 5.58 s
  ══════════════════════════════════════════════════════════════════
      ¢                               ░       ░              >>>
   _______ _______ _______ _______ _______ _______ _______ _______
      $                       ☺                       ¢       ¢
   _______ _______ _______ _______ _______ _______ _______ _______
      >      >>>      ¢       ¢                       ░
  ══════════════════════════════════════════════════════════════════

Command >

[DEBUG] Executing:
Distance: 20
Coins: 5
Cycle: 6
Total obstacles: 20
Total coins: 11
Supercoin is present
Elapsed Time: 6.50 s
  ══════════════════════════════════════════════════════════════════
      ░       ░              >>>
   _______ _______ _______ _______ _______ _______ _______ _______
                      ¢       ¢       ¢                       ░
   _______ _______ _______ _______ _______ _______ _______ _______
      >               ░                               ¢       █
  ══════════════════════════════════════════════════════════════════

Command >
```

1.4. The `clear` command and the `cheat` command

We now create two commands for use in debugging.

The `clear` command. This command eliminates all game elements (except the car) from the road. The abbreviated form of this command is simply 0.

The `cheat` command. This command is used to add an advanced game element, i.e. an object of one of the GameElement subclasses appearing in the list given below, some of which have already been introduced and the rest of which will be introduced later in this document.

1. Wall
2. Turbo
3. SuperCoin
4. Truck
5. Pedestrian

The execution of this command results in the addition of a GameElement object of the chosen subtype in a random lane of the last visible column, after first deleting any game elements already present in that column. The advanced game element to be added is identified by its name: `wall`, `turbo`, `super`, `truck` and `ped`, respectively.

Since this is a debugging command, to avoid spending too much time implementing it, it is acceptable to use a method in GameElementGenerator (called from the execute method of the CheatCommand class) which contains a switch on the possible advanced game element names, in each case of which a GameElement of the type corresponding to that name is created (before calling a method of Game to introduce this new GameElement object into the game). However, a better implementation would be to introduce the following code:

- a NAME constant in each GameElement subclass similar to the NAME constant in each Command subclass,
- a name attribute in GameElement similar to the name attribute in Command,
- a parse method in GameElement similar to the parse method in Command that uses this name attribute and is inherited by each of the GameElement subclasses,
- a static attribute AVAILABLE_GAMEELEMENTS in GameElementGenerator similar to the AVAILABLE_COMMANDS attribute of Command containing one object of each GameElement subclass,
- a no-argument constructor in each GameElement subclass to be used to create the objects of the AVAILABLE_GAMEELEMENTS attribute; notice that the coordinates of these objects remain uninstantiated,
- a static method getGameElement in GameElementGenerator similar to the getCommand method of Command; when this method receives an uninstantiated object returned by one of the parse methods of the objects in the AVAILABLE_GAMEELEMENTS array, it creates a corresponding instantiated object[1], where the row number is obtained by calling getRandomLane of Game

Note that the AVAILABLE_GAMEELEMENTS array could also be used to implement the `info` command similarly to the way in which the AVAILABLE_COMMANDS array is used to implement the `help` command.

1.5. The **`wave`** command

This command, which costs the player 5 coins, pushes all the game elements within the visibility (except the car) one square to the right on the road. ~~If a game element has another game element immediately to its right, the command has no effect. Note, however, that the order of updating of the game elements on the board may affect how many of them can be pushed to the right. You should try to update the elements in order to maximise the number that can be pushed to the right.~~ To implement this command, you should create another instantaneous action as was done for the `shoot` command. The abbreviated form of this command is `"w"`. After executing the wave action, the game elements should be updated and any dead objects eliminated. You may, or may not, choose to also move the player after the wave has occurred (but before updating the other game elements).

Here follows an example of the use of this command:

[1] How? For example, by calling a createObject method that is defined in each subclass of GameElement and which creates (instantiated) objects of that subclass.

```
Command >

[DEBUG] Executing:
Distance: 6
Coins: 5
Cycle: 4
Total obstacles: 4
Total coins: 1
  ═════════════════════════════════════════════════════════════════
                      ¢       ░                       ¦
   _______ _______ _______ _______ _______ _______ _______ _______
      >       ░                                       ¦
   _______ _______ _______ _______ _______ _______ _______ _______
      ░                                       ░      ¦
  ═════════════════════════════════════════════════════════════════

Command >
w
[DEBUG] Executing: w
Distance: 6
Coins: 0
Cycle: 4
Total obstacles: 4
Total coins: 1
  ═════════════════════════════════════════════════════════════════
                              ¢       ░               ¦
   _______ _______ _______ _______ _______ _______ _______ _______
      >               ░                               ¦
   _______ _______ _______ _______ _______ _______ _______ _______
              ░                                       ░
  ═════════════════════════════════════════════════════════════════
```

1.6. The **`grenade`** command and the Grenade game element

This command, which costs the player 3 coins, creates a grenade object at the position on the road provided as a parameter to the command; in fact, two parameters must be provided, the x and y coordinates, where the x coordinate is relative to the position of the car and where the value of x is less than or equal to the visibility. If there is already an object at the position provided as a parameter, the command has no effect. As for any command with parameters, the GrenadeCommand class will need a parse method that is different to the one in the Command class, i.e. it will need to overwrite the parse method of the Command class (and, to avoid writing *fragile code*, parse methods for commands with parameters should not return the value this).

The behaviour of a grenade object is as follows:

- It contains a cycle counter initialised to 3, which explodes when the value reaches 0[2].

- When the grenade explodes it causes damage of 1 life to all the obstacles (including walls, trucks and pedestrians) in its vicinity.

The impact of the explosion should be implemented by introducing a method receiveExplosion in the *Collider* interface. In those classes that are affected by the explosion, since the effect is the same as that of a shot, receiveExplosion can simply call receiveShot. A grenade is repesented by the symbol ð followed by, in square brackets, the number of cycles left until it explodes.

We would also like to implement the propagation of the explosion as an instantaneous action. However, the instantaneous actions defined so far either assume that the position of

[2] *hint*: ensure that a grenade whose counter is zero is considered dead, then call the explode method from the onDelete method of the Grenade class

origin of the action is that of the player (ShootAction) or that the instantaneous action has no position of origin (WaveAction), whereas the position of origin of the explosion action is clearly that of the grenade object causing the explosion. For this type of instantaneous actions, we can simply define a constructor in the action class and use it to pass the action the position of its origin. In the case of the grenade, the ExplodeAction object should be so created by the constructor of the Grenade class (where an object of the Grenade class is itself created in the execute method of the GrenadeCommand class). After executing the grenade action, the game elements should be updated and any dead objects eliminated. You may, or may not, choose to also move the player after the grenade has been placed (but before updating the other game elements).

Here follows an example of the use of the `grenade` command:

```
Command >
g 5 1
[DEBUG] Executing: g 5 1
Distance: 25
Coins: 2
Cycle: 5
Total obstacles: 20
Total coins: 11
Supercoin is present
Elapsed Time: 30.52 s
  ════════════════════════════════════════════════════════════════
              ¢                               ░       ░
   _______ _______ _______ _______ _______ _______ _______ _______
      >       $                       ☺      ð[3]             ¢
   _______ _______ _______ _______ _______ _______ _______ _______
      ¢              >>>      ¢       ¢                       ░
  ════════════════════════════════════════════════════════════════

Command >

[DEBUG] Executing:
Distance: 24
Coins: 1002
Cycle: 6
Total obstacles: 20
Total coins: 11
Elapsed Time: 38.18 s
  ════════════════════════════════════════════════════════════════
      ¢                               ░       ░             >>>
   _______ _______ _______ _______ _______ _______ _______ _______
      >                              ð[2]             ¢       ¢
   _______ _______ _______ _______ _______ _______ _______ _______
             >>>      ¢       ¢                       ░
  ════════════════════════════════════════════════════════════════

Command >

[DEBUG] Executing:
Distance: 23
Coins: 1002
Cycle: 7
Total obstacles: 20
Total coins: 11
Elapsed Time: 38.80 s
  ════════════════════════════════════════════════════════════════
                              ░       ░             >>>
   _______ _______ _______ _______ _______ _______ _______ _______
      >               ☺      ð[1]             ¢       ¢       ¢
   _______ _______ _______ _______ _______ _______ _______ _______
     >>>      ¢       ¢                       ░
  ════════════════════════════════════════════════════════════════

Command >
```

```
[DEBUG] Executing:
Distance: 22
Coins: 1002
Cycle: 8
Total obstacles: 20
Total coins: 11
Elapsed Time: 39.66 s
  ════════════════════════════════════════════════════════════════════════
                                                     >>>
   _______  _______  _______  _______  _______  _______  _______  _______
      >                                   ¢        ¢        ¢
   _______  _______  _______  _______  _______  _______  _______  _______
      ¢        ¢                          ░                          ←
  ════════════════════════════════════════════════════════════════════════
```

1.7. Interface Buyable

There are various commands that share the characteristic of having a cost, so we can represent this behaviour in another interface called Buyable to be implemented by the Command class:

```java
package supercars.control;

import supercars.logic.Game;

public interface Buyable {

    public int cost();

    public default void buy(Game game){
        // TODO add your code
    };
}
```

The body of the buy() method will decrease the coins of the player by the appropriate amount. In this case, instead of specifying the default behaviour in the abstract class Command, as we could have done, we have lifted it to the interface itself by using a default method. If the default behaviour is that of the commands that do not (resp. do) have a cost, the commands that do (resp. do not) have a cost will need to overwrite this default behaviour.

1.8. Game elements that move

All the game elements created until now (except the player) do not move. We now create two game elements that have their own movement.

The Truck game element: objects of this class, represented by the symbol "←", advance from right to left one square each turn. They can collide with the player in the same way as an obstacle but they do not collide with obstacles. If, on a given cycle, the truck shares a square with another game element, either of the two can be represented on the screen as occupying that square (this will be dictated by the order of the game elements, itself dictated by the order of introduction into the game). You must not allow the player and a truck to pass through each other without colliding; as for the implementation of the Turbo command, this will involve checking for collisions twice on every cycle.

Here follows an example:

```
[DEBUG] Executing:
Thunder hit position: (0 , 0)
Distance: 98
Coins: 30
Cycle: 2
Total obstacles: 1
Total coins: 0
Elapsed Time: 0,41 s
  ═════════════════════════════════════════════════════════════════
                              ←
   _______ _______ _______ _______ _______ _______ _______ _______
      >
   _______ _______ _______ _______ _______ _______ _______ _______

  ═════════════════════════════════════════════════════════════════

Command >
q
[DEBUG] Executing: q
Thunder hit position: (2 , 1)
Distance: 97
Coins: 30
Cycle: 3
Total obstacles: 1
Total coins: 0
Elapsed Time: 0,84 s
  ═════════════════════════════════════════════════════════════════
      >       ←
   _______ _______ _______ _______ _______ _______ _______ _______

   _______ _______ _______ _______ _______ _______ _______ _______

  ═════════════════════════════════════════════════════════════════

Command >

[DEBUG] Executing:
Thunder hit position: (0 , 0)
Distance: 96
Coins: 30
Cycle: 4
Total obstacles: 1
Total coins: 0
Elapsed Time: 1,92 s
  ═════════════════════════════════════════════════════════════════
      @
   _______ _______ _______ _______ _______ _______ _______ _______

   _______ _______ _______ _______ _______ _______ _______ _______

  ═════════════════════════════════════════════════════════════════
[GAME OVER] Player crashed!
```

The Pedestrian game element: Pedestrians are obstacles, represented by the symbol ☺, that cross the road constantly in the same column (and therefore move diagonally from right to left on the screen), alternating between moving from top to bottom and from bottom to top. If the car collides with a pedestrian, the game ends and, in addition, the player loses all his or her points. You must not allow the player and a pedestrian to pass through each other without colliding; as for the implementation of the Turbo command, this will involve checking for collisions twice on every cycle.

Here follows an example:

```
Command >

[DEBUG] Executing:
Thunder hit position: (1 , 0)
Distance: 77
```

```
Coins: 45
Cycle: 23
Total obstacles: 1
Total coins: 0
Elapsed Time: 15,05 s
  ═════════════════════════════════════════════════════════════════

   _______ _______ _______ _______ _______ _______ _______ _______
      >               ☺
   _______ _______ _______ _______ _______ _______ _______ _______

  ═════════════════════════════════════════════════════════════════

Command >
a
[DEBUG] Executing: a
Thunder hit position: (5 , 1)
Distance: 76
Coins: 45
Cycle: 24
Total obstacles: 1
Total coins: 0
Elapsed Time: 15,84 s
  ═════════════════════════════════════════════════════════════════

   _______ _______ _______ _______ _______ _______ _______ _______

   _______ _______ _______ _______ _______ _______ _______ _______
      >       ☺
  ═════════════════════════════════════════════════════════════════

Command >

[DEBUG] Executing:
Thunder hit position: (0 , 1)
Distance: 75
Coins: 0
Cycle: 25
Total obstacles: 0
Total coins: 0
Elapsed Time: 16,49 s
  ═════════════════════════════════════════════════════════════════

   _______ _______ _______ _______ _______ _______ _______ _______

   _______ _______ _______ _______ _______ _______ _______ _______
      @
  ═════════════════════════════════════════════════════════════════
```

If the pedestrian is hit by a bullet or a grenade, the player loses all their coins but does not die and the game continues.

1.9. Thunder Action

All the game elements and actions introduced so far are either created at the beginning of the game or via a user command. We now introduce an instantaneous action that can occur during a game without any user command: Thunder which falls on a random square of the visible part of the road. If a thunderclap falls on an obstacle it eliminates it.

Here follows an example:

```
Command >

[DEBUG] Executing:
Thunder hit position: (4 , 0)
Distance: 94
Coins: 6
```

```
Cycle: 6
Total obstacles: 45
Total coins: 25
Supercoin is present
Elapsed Time: 2,77 s
  ══════════════════════════════════════════════════════════════════
  ══════════════════════════════════════════════════════════════════
              ←                       ¢
   _______ _______ _______ _______ _______ _______ _______ _______ 
      >                               ▪                       ¢
   _______ _______ _______ _______ _______ _______ _______ _______ 
              ░                               ░
  ══════════════════════════════════════════════════════════════════
  ══════════════════════════════════════════════════════════════════

Command >

[DEBUG] Executing:
Thunder hit position: (4 , 1) -> Obstacle hit
Distance: 93
Coins: 6
Cycle: 7
Total obstacles: 45
Total coins: 24
Supercoin is present
Elapsed Time: 3,27 s
  ══════════════════════════════════════════════════════════════════
  ══════════════════════════════════════════════════════════════════
                              ¢
   _______ _______ _______ _______ _______ _______ _______ _______ 
      >                                               ¢       ¢
   _______ _______ _______ _______ _______ _______ _______ _______ 
      ░                               ░
  ══════════════════════════════════════════════════════════════════
  ══════════════════════════════════════════════════════════════════
```

Observe that the position of the thunderclap is relative to the column containing the player.

1.10. Additions to the GameElementGenerator class

To the factory class GameElementGenerator we add a method to be called on each cycle to determine if and where to create objects during the game:

```
  public static void generateRuntimeObjects(Game game) {
    // Note we use this method to create and inject new objects or actions on runtime.

    if (game.getLevel().hasAdvancedObjects()) {
      game.execute(new ThunderAction());
    }

  }
}
```

We also need to add the creation of the new game elements Wall, Turbo, Supercoin, Truck and Pedestrian to the generateGameElements method which then has the following form:

```
public static void generateGameElementss(Game game, Level level) {

  for(int x = game.getVisibility()/2; x < game.getRoadLength(); x ++) {

    game.tryToAddObject(new Obstacle(game, x, game.getRandomLane())
        , level . obstacleFrequency());
        game.tryToAddObject(new Coin(game, x, game.getRandomLane())
             , level . coinFrequency());


        if (level . hasAdvancedObjects()) {
            game.tryToAddObject(new Wall(game, x, game.getRandomLane()), level.
                  advancedObjectsFrequency());
            game.tryToAddObject(new Turbo(game, x, game.getRandomLane()), level.
                  advancedObjectsFrequency());
            if (! SuperCoin.hasSuperCoin()) {
                game.tryToAddObject(new SuperCoin(game, x, game.getRandomLane())
                     , level . advancedObjectsFrequency());
            }
            game.tryToAddObject(new Truck(game, x, game.getRandomLane()), level.
                  advancedObjectsFrequency());
            game.tryToAddObject(new Pedestrian(game, x, 0), level.advancedObjectsFrequency());

        }
    }
}
```

Observe the use of the hasAdvancedObjects() method which returns true if the frequency of apparition of advanced objcts is more than 0. Recall that even when the frequency of apparition of advanced objects is 0, the appearance of objects can be forced using the `cheat` command.

1.11. Additions to the Level class

We have not yet specified the frequency of appearance of the new game elements referred to as advanced objects. Like the other such frequencies, this value will be stored as an attribute in the Level class. In modifying the Level class, we also add a new level called ADVANCED, leaving the configuration information for the different levels as shown in Table 1.1.

Nivel	length	width	visibility	Obs freq	Coin freq	advObjFreq
TEST	10	3	8	0.5	0.5	0
EASY	30	3	8	0.5	0.5	0
HARD	100	5	6	0.7	0.3	0
ADVANCED	100	3	8	0.3	0.3	0.1

Tabla 1.1: Configuration for each level of difficulty.

2. Important observations

To ensure that the code of the classes Game and GameElementsContainer is suitably generic, you can manifest whether or not they handle any concrete game elements or instantaeous actions by not importing whole packages, only individual classes (which, in any case, is the best policy). If you do so and your code is suitably generic, the class Game should only need the following import statements if the last two were in the same package (as shown here), it would not be necessary to import them:

```
import java.util . Random;
import es.ucm.tp1.control.Level;
import es.ucm.tp1.logic. gameElements.GameElement;
import es.ucm.tp1.logic. gameElements.Player;
import es.ucm.tp1.logic. actions. InstantAction;
import es.ucm.tp1.logic. GameElementGenerator;
import es.ucm.tp1.logic. GameElementContainer
```

and the class GameElementContainer should only need the following:

```
import java.util . ArrayList;
import java.util . List;
import supercars.logic . gameElements.GameElement;
```

The **Available game elements** are as follows, where we write the advance objects in upper case and the others in lower case:

```
[DEBUG] Executing: i
Available objects:
[Car] the racing car
[Coin] gives 1 coin to the player
[Obstacle] hits car
[GRENADE] Explodes in 3 cycles, harming everyone around
[WALL] hard obstacle
[TURBO] pushes the car: 3 columns
[SUPERCOIN] gives 1000 coins
[TRUCK] moves towards the player
[PEDESTRIAN] person crossing the road up and down
```

The **Available commands** are as follows:

```
[DEBUG] Executing: h
Available commands:
[h]elp: show this help
[i]nfo: prints game element info
[n]one | []: update
[q]: go up
[a]: go down
[e]xit: exit game
[r]eset [<level> <seed>]: reset game
[t]est: enables test mode
[s]hoot: shoot bullet
[g]renade <x> <y>: add a grenade in position x, y
[w]ave: do wave
Clear [0]: Clears the road
Cheat <AO-name>: Removes all elements of last visible column and adds advanced object AO
```

3. Testing

To help you with your coding, we provide you with some tests. Providing input/output tests such as these is a compromise since, on the one hand, they can be of great help to the students in debugging their code, but on the other hand, if is difficult, if not impossible, to avoid the obligation to produce exactly the same output imposing too many restriction on the structure of that code. In particular, regarding movement and display, the tests suppose your code has the following structure:

- check for collisions

- move the player
- check for collisions
- move the other game elements
- display the state of the game

whereas the following structure could be considered more reasonable, though slightly more complicated to implement[3]:

- move the player
- check for collisions
- move the other game elements
- check for collisions
- display the state of the game

Note that the main difference between these two code structures is that the order of *check for collisions* and *display the state of the game* is inverted. This means that, in the structure, expected by the tests, for game elements that move, such as a pedestrian, at the point where a portion of the road must be displayed, the state of the game after a car has run into a pedestrian is different to the state of the game after a pedestrian has run into a car. Of course, in a real-world game involving near real-time movement, any difference between the visualisation of these two states would likely be imperceptible. Nevertheless, in the english group, the tests are provided as an aid but it is not obligatory for your code to pass them. However, you should check whether the tests resolve undespecified aspects of this problem statement and, where they do, use this resolution. For example, if a move up or down command would take the car off the road, does this lead to an "unrecognised command" message or does the car simply advance in the same lane? See test `05_easy_s100` for the answer.

Take into account that:

- We have changed the nomenclature. The new format is as follows:
  - `00_easy_s666.txt`: This is the input of test case number `00` with level `easy` and seed `666`.
  - `00_easy_s666_output.txt`: This is the output of the same test case.

3.1. Test cases

For Assignment 2 we provide more test cases than for Assignment 1. Below is a list of all the test cases provided along with a description of the objective of each one.

- `00_easy_s666.txt`: Check the execution of basic commands.
- `01_easy_s666.txt`: Play a game with level `EASY` and seed 666 which ends in a crash.

[3]For example, if you are not careful, the code implementing explosions could even lead to an infinite loop.

- `02_easy_s100.txt`: Play a game with level `EASY` and seed 100 which ends in a win for the player.
- `03_test_s100.txt`: Play a game with level `TEST` and seed 100 which ends in a win for the player.
- `04_test_s100.txt`: Play a game with level `HARD` and seed 100 which ends in a win for the player.
- `05_easy_s100.txt`: Test the handling of commands that could cause the player to leave the road.
- `06_easy_s25.txt`: Test the basic `reset` command.
- `07_easy_s37.txt`: Test the advanced `reset` command.
- `08_easy_s37.txt`: Test the `info` command.
- `09_easy_s37.txt`: Test the `shoot` command.
- `10_easy_s37.txt`: Test the `supercoin` game element.
- `11_easy_s37.txt`: Test the `turbo` command.
- `12_easy_s37.txt`: Test the `clear` command, both at level `EASY` and level `ADVANCED`.
- `13_easy_s37.txt`: Test the `wave` command.
- `14_easy_s37.txt`: Test the `grenade` command and the Grenade game element.
- `15_easy_s37.txt`: Test the `cheat` command.

3.2. Batch execution of the tests

To facilitate the use of this larger number of test cases, we also provide a script `testsPR2.sh` that you can use to execute all the test cases against your implementation. The script can be used in Linux, MacOS (be sure to use a terminal executing a bash shell) or Windows (you will need to use Cygwin or WSL).

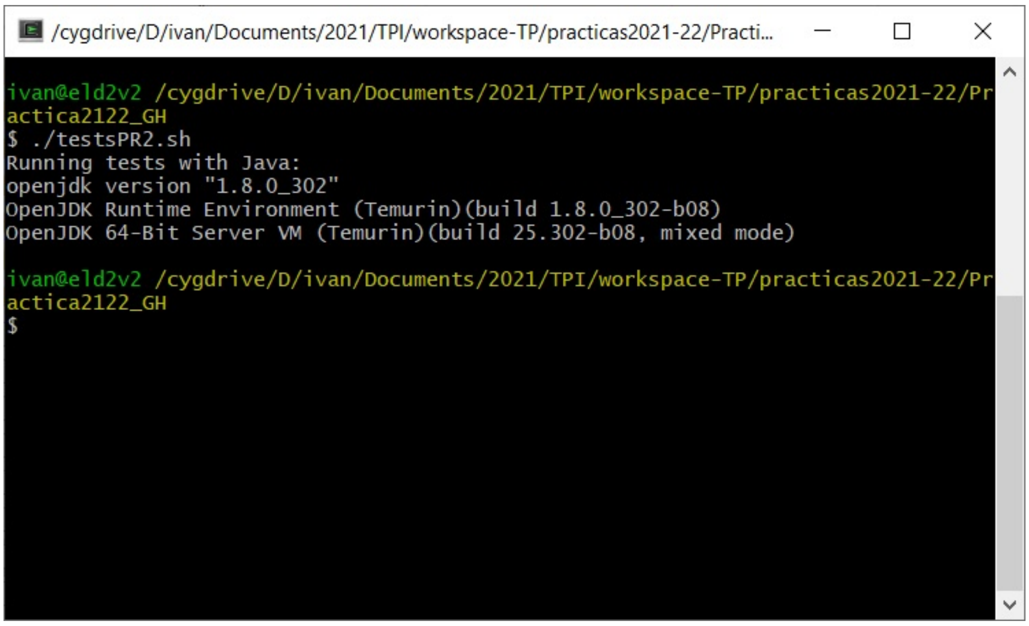
To execute the tests using the script:

1. Place the script `testsPR2.sh` in the root of the Eclipse project.
2. Create a directory called `test` in the root of the Eclipse project and place the tests in this directory
3. Open a terminal and execute the script.

The script creates a directory called `output` in which it writes the output files (with name suffix `_output.txt`), each containing the output of one of the test cases, as well as files (with name suffix `_diffs.txt`) containing the differences between the actual output with the expected output. If you have difficulty interpreting the content of these difference files, you can consult the following resources:

- StackOverflow: Interpreting git diff output.
- What is git diff and how do we read the output.

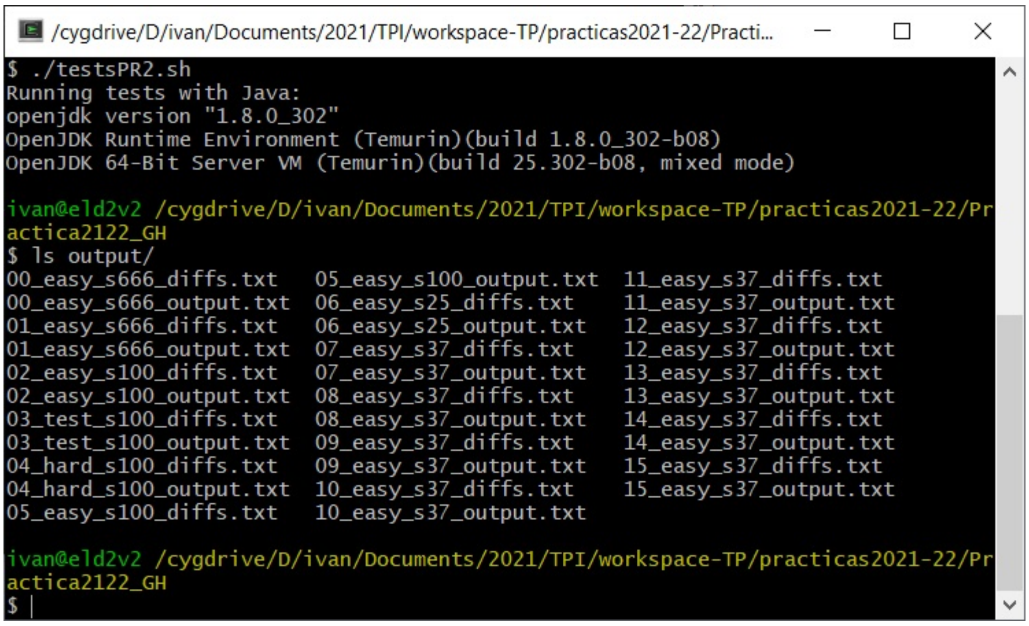
Figures 1 and 2 show a Cygwin terminal with the output of the script.



```
/cygdrive/D/ivan/Documents/2021/TPI/workspace-TP/practicas2021-22/Practi...
ivan@eld2v2 /cygdrive/D/ivan/Documents/2021/TPI/workspace-TP/practicas2021-22/Pr
actica2122_GH
$ ./testsPR2.sh
Running tests with Java:
openjdk version "1.8.0_302"
OpenJDK Runtime Environment (Temurin)(build 1.8.0_302-b08)
OpenJDK 64-Bit Server VM (Temurin)(build 25.302-b08, mixed mode)

ivan@eld2v2 /cygdrive/D/ivan/Documents/2021/TPI/workspace-TP/practicas2021-22/Pr
actica2122_GH
$
```

Figura 1: Ejecuting the test script in Cygwin



```
/cygdrive/D/ivan/Documents/2021/TPI/workspace-TP/practicas2021-22/Practi...
$ ./testsPR2.sh
Running tests with Java:
openjdk version "1.8.0_302"
OpenJDK Runtime Environment (Temurin)(build 1.8.0_302-b08)
OpenJDK 64-Bit Server VM (Temurin)(build 25.302-b08, mixed mode)

ivan@eld2v2 /cygdrive/D/ivan/Documents/2021/TPI/workspace-TP/practicas2021-22/Pr
actica2122_GH
$ ls output/
00_easy_s666_diffs.txt   05_easy_s100_output.txt  11_easy_s37_diffs.txt
00_easy_s666_output.txt  06_easy_s25_diffs.txt    11_easy_s37_output.txt
01_easy_s666_diffs.txt   06_easy_s25_output.txt   12_easy_s37_diffs.txt
01_easy_s666_output.txt  07_easy_s37_diffs.txt    12_easy_s37_output.txt
02_easy_s100_diffs.txt   07_easy_s37_output.txt   13_easy_s37_diffs.txt
02_easy_s100_output.txt  08_easy_s37_diffs.txt    13_easy_s37_output.txt
03_test_s100_diffs.txt   08_easy_s37_output.txt   14_easy_s37_diffs.txt
03_test_s100_output.txt  09_easy_s37_diffs.txt    14_easy_s37_output.txt
04_hard_s100_diffs.txt   09_easy_s37_output.txt   15_easy_s37_diffs.txt
04_hard_s100_output.txt  10_easy_s37_diffs.txt    15_easy_s37_output.txt
05_easy_s100_diffs.txt   10_easy_s37_output.txt

ivan@eld2v2 /cygdrive/D/ivan/Documents/2021/TPI/workspace-TP/practicas2021-22/Pr
actica2122_GH
$ |
```

Figura 2: Files generated by execution of the test script in Cygwin.