PROJECT REPORT – WEEK 1 ALPER DEMİR-1745850

Market Research

I have researched recently released programming based game, Mouse Run. The game is written in Java and it is an Open Source software ever since its release on April 21, 2013. The game revolves around mice navigating a maze to search and consume cheese while competing with others. The mouse that consumes the most cheese wins. Each of these mouse is programmable and the game host will move the mouse using the logic that is supplied by the programmer.

In order to create a Mouse for the game, the programmer is required to create a java class that will represent the Mouse and provide the logic to it. Depending on the current grid of the maze, the mouse is able to move up, down, left and/or right. The mouse can also plant a bomb at the grid to hinder another mouse progress. The decision to move or to plant a bomb is part of the logic.

A script is provided in each release to help the programmers to compile all their mouse implementations and move them to the host directory. When the game starts, the game host will detect all of the mouse implementations and each mouse will be represented on the maze. The logic of the mouse will determine its movement on the maze.

Since it is very similar to our ideas of the design of the project, this research was very useful.

Here is the tutorial of the game;

Building a Mouse
Here, we will show you how to create a Mouse and eventually get it to run on the Maze. But making it intelligent, we will leave that to you. Here, we assume that you have a proper Java JDK installed and properly configured, and that you have downloaded the No-Source set of Mouse Run. You should unzip the set to a location that is convenient.

If you have properly followed the following steps, you would have succeeded in creating a Mouse.

1. Open up a Text Editor of your choice and enter the following codes in:

```
package mouserun.mouse;

import mouserun.game.Mouse;
import mouserun.game.Grid;
import mouserun.game.Cheese;
```

You would notice that all implementation of the Mouse has to be in the package called **mouserun.mouse**. This will ensure that the game host will be able to detect the Mouse when it is compiled and executed later on. You will also need to import those three classes. You'll find out why later on.

2. Continue adding codes like the following:

```
package mouserun.mouse;

import mouserun.game.Mouse;
import mouserun.game.Grid;
import mouserun.game.Cheese;

public class FooMouse extends Mouse
{

}
```

Not only that all Mouse implementations must be in the same package, they too have to extend from the**Mouse** class. If you have checked the documentation of the Mouse class, you would realize that it is an**abstract class** and that you will need to implement all abstract methods.

3. By checking the documentation, you will know that you will need to implement those methods like the following:

```
package mouserun.mouse;

import mouserun.game.Mouse;
import mouserun.game.Grid;
import mouserun.game.Cheese;

public class FooMouse extends Mouse
{
    public FooMouse()
    {
      super("FooMouse");
    }

    public int move(Grid currentGrid, Cheese cheese)    {
    }

    public void newCheese()    {
    }

    public void respawned()    {

    }
}
```

If you have saved this file and compile it, it will already create a Mouse for you. You will see the mouse on the game interface. But it does not do anything. The mouse is dumber than ever.

But before we move on, lets just see what we have done so far. We have created a constructor for the Mouse which has to be **public**. **super("FooMouse");** will give the name of the Mouse as FooMouse and appear under the Mouse in the game interface.

public int move(Grid currentGrid, Cheese cheese) is probably the most important method in your Mouse class. When the game host has detected that your Mouse has reach a grid, it will need to provide a decision on what to do next. **currentGrid** is the current grid that the Mouse is on and **cheese** is the cheese that is being sought for. With information given by the two objects, your mouse need to make the decision.

public void newCheese() is called when a cheese has been consumed. If you do not wish to do anything, so let it be. This method is normally utilized for cleaning up.

public void respawned() is called when your Mouse has hit a bomb that is not planted by it and has to be relocated to a new location. If you do not wish to do anything, so let it be. This method is normally utilized for recalibration or resetting strategy.

Let's see now what you can do in the **move()** method.

To move Up, do the following:
```
public int move(Grid currentGrid, Cheese cheese)
{
  return Mouse.UP;
}
```

To move Down, do the following:
```
public int move(Grid currentGrid, Cheese cheese)
{
  return Mouse.DOWN;
}
```

To move Left, do the following:
```
public int move(Grid currentGrid, Cheese cheese)
{
  return Mouse.LEFT;
}
```
To move Right, do the following:
```
public int move(Grid currentGrid, Cheese cheese)
{
  return Mouse.RIGHT;
}
```

To plant a Bomb, do the following:

```
public int move(Grid currentGrid, Cheese cheese)
{
  return Mouse.BOMB;
}
```

4. Once when you think you are ready to compile your codes and let the game execute, save your java file as**FooMouse.java** to the directory of the MouseRun folder you have unzipped. The java file should be in the same directory as the **launcher.bat** (on Windows) or **launcher.sh** (on OS X) file.

5. Execute **launcher.bat** file and see your mouse doing things that is based on the logic you have given it.

On OS X however, please open the **Terminal** and enter the following:

```
bash launcher.sh
```

6. You can always test out with the TestMouse bundled with the set. But let us warn you, that mouse is seriously insanely stupid. But oh well...

Changing the Size of the Maze and Number of Cheese
The Maze is dynamic if you don't already know. Is the default Maze size too big for two mice? Or is the default Maze size is too small for 10? Don't worry, you can always change the size of the mouse very easily.

Open up either **launcher.bat** or **launcher.sh** and locate the line

java -cp classes mouserun.GameStarter **20 20 50**

You would notice that there are three numbers as arguments to start the game. These arguments represents the following:

GameStarter [width] [height] [numberOfCheese]

width - Is the number of columns in the Maze.
height - Is the number of rows in the Maze.
numberOfCheese - Is the number of cheese to play for

Save and then execute the script again.

Pitfalls to avoid when creating a mouse

Like we said, creating the mouse is easy. Making it intelligent is not.

After creating and testing a few mice ourselves (yeah, we're learning too!), we've learned to avoid some pitfalls of creating a mouse that is able to think on its own.

1. **Firstly, ensure that you cater to all possible scenarios.**
   You may choose to learn grids as the mouse explore the maze but at times, your mouse may encounter bombs and got displaced. Ensure that you can appropriately cater to situation where it is least expected. Otherwise, you may end up in infinite loop or making the wrong decisions.

   Making the wrong decision is one thing. At least, when you have the ability to, you would move (otherwise, you might just be hitting the wall). Infinite loop in your logic is the worst thing to happen. It will not only affect you, but it will also affect all other mouse. The game has been update to handle situations such as this.

2. **Getting disqualified**
   The game now is no longer forgiving. This is ensure that there is a fair competition with other mouse. Your Mouse is expected to make its decision under a time limit (distribution default 100 microseconds). Your Mouse is also expected not to produce any exceptions during its move. The game will disqualify any mouse from the game if it cause consecutive problems to the game play.