Software Engineering

CPS2002

Assignment Report

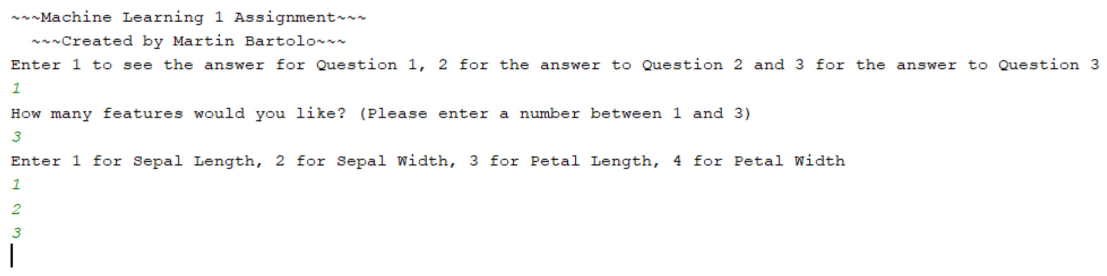
Martin Bartolo - 0218300L
Mikhail Cassar - 0319599M

BSc (Hons) Computing Science and Mathematics

Assignment due 27th May 2019

Contents

Diagram Example



```
~~~Machine Learning 1 Assignment~~~
  ~~~Created by Martin Bartolo~~~
Enter 1 to see the answer for Question 1, 2 for the answer to Question 2 and 3 for the answer to Question 3
1
How many features would you like? (Please enter a number between 1 and 3)
3
Enter 1 for Sepal Length, 2 for Sepal Width, 3 for Petal Length, 4 for Petal Width
1
2
3
```

Code Snippet

```java
public class Main{
  public static void main (String args[]){
    System.out.println("Hello World!");
  }
}
```

Introduction

The aim of this assignment was to collaboratively work on a software project, with the main focus being on rigorous software testing and the use of Git. Our first task was to set up our environments, namely our Git repository on Github and our Jenkins environment on the University Jenkins server. First, we initialised our Git repository and ensured that each team member could commit changes and push and pull them from Github. When this was ensured, we set up our Jenkins environment to work with Maven and scan for changes from Github every few minutes. Whenever changes are found, they are built and run with a detailed code coverage report and test results being displayed using the Emma plug-in. Our progress at this point can be seen by viewing the "Part1" tag on our Github repository. After completing our set-up, we were ready to start working on the two remaining tasks which will be discussed in detail throughout the remainder of this report.

Enhancements

Different Map Types

The design pattern chosen for this enhancement was the **factory design pattern**. Since this enhancement was centred around the map's creation then it was clear that a creational design pattern had to be chosen. The main things which needed to be kept in mind were that 2 initial map types had to be implemented while more map types could easily be added to the future. Therefore, the Factory design pattern was the most suitable for the task.

The class diagram of the enhancement can be seen below.

****Add diagram here****

Our implementation is centred around a *Map* interface which contains each method which will be used by the different map types. Each type of map then has a class which implements this main *Map* interface as can be seen in the class diagram above. This design allows for the easy implementation of additional map types in the future.

A factory method is present in the *MapCreator* class which is passed the map's type and its size. This class has a creator subclass for every map type, allowing for easy creation of additional map types in the future. An instance of the correct creator subclass will then be made and used to create the map. This can be seen in the code snippet below.

```
//Factory Method
Map createMap(String type, int mapSize){

    MapCreator creator;

    if(type.equals("safe")){
        creator = new SafeMapCreator(mapSize);
    }
    else if(type.equals("hazardous")){
        creator = new HazardousMapCreator(mapSize);
    }
    else{
        creator = null;
        System.err.println("Invalid map type");
    }

    if (creator != null) {
        return creator.create();
    }
    else{
        return null;
    }
}
```

The map is then created in the appropriate creator subclass by setting the map size and calling the generate method in the map type's class (either the *SafeMap* or *HazardousMap* class in our implementation but more can easily be added). A code snippet of this create method for the "safe" map type is shown below

```
//Method to create and return a safe map
@Override
public Map create(){
    //getInstance method used here to obtain a
    //static instance of the hazardous map
    SafeMap map =  SafeMap.getInstance();
    map.setMapSize(mapSize);
    map.generate();
    return map;
}
```

In the generate method for the safe map, the maximum amount of water tiles is set to 10% and then rounded down to the nearest integer. This code snippet is shown below.

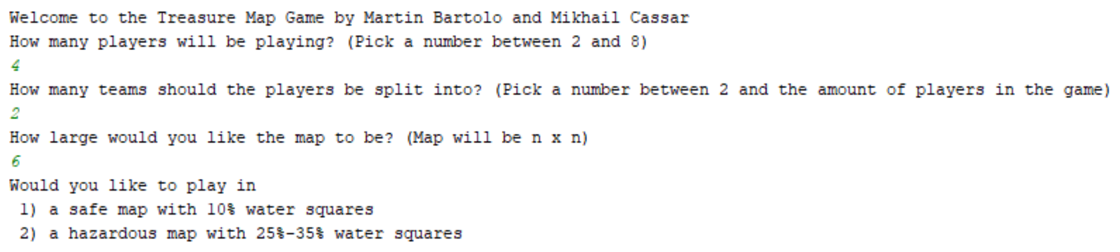
```java
//The maximum number of water tiles in a map is set to
//10% of the total tiles
int waterMaxTiles = (int) Math.floor((mapSize*mapSize) * 0.1);
```

In the generate method for the hazardous map, the maximum amount of water tiles is set to 25% and then rounded up to the nearest integer. This code snippet is shown below.

```java
//The maximum number of water tiles in a map is set to
//25% of the total tiles
int waterMaxTiles = (int) Math.ceil((mapSize*mapSize) * 0.25);
```

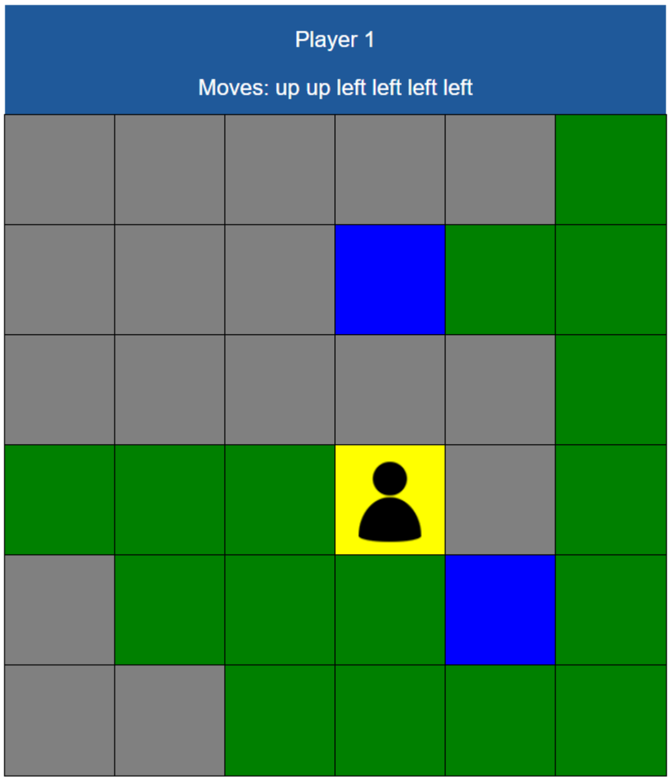
Apart than these changes, the map is generated as it was in the generate method in the basic version of the program before the enhancements.

When running the program, the user is asked what map type they would like to play with. This can be seen below.



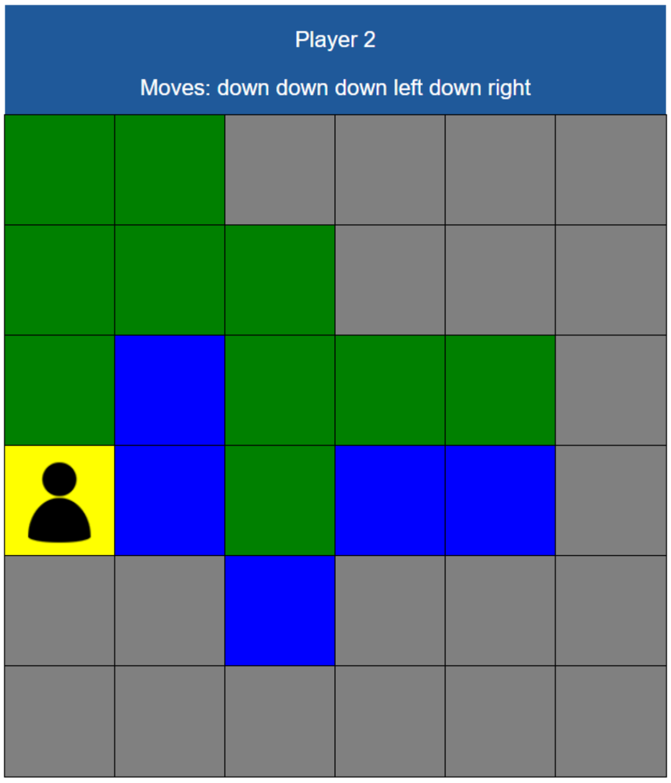
```
Welcome to the Treasure Map Game by Martin Bartolo and Mikhail Cassar
How many players will be playing? (Pick a number between 2 and 8)
4
How many teams should the players be split into? (Pick a number between 2 and the amount of players in the game)
2
How large would you like the map to be? (Map will be n x n)
6
Would you like to play in
 1) a safe map with 10% water squares
 2) a hazardous map with 25%-35% water squares
```

After the user chooses the map type, the map is created as discussed on the previous page and then the game can be played. On the next page is a comparison between the ending screen of a game played by 4 players in 2 teams on a 6x6 safe map and a game played by the same players on a 6x6 hazardous map. Notice the different amounts of water tiles encountered between the 2 map types.


Player 1

Moves: up up left left left left

Ending screen of game using a safe map


Player 2

Moves: down down down left down right

Ending screen of game using a hazardous map