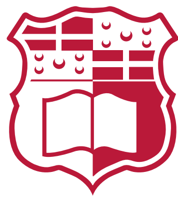

Multiplayer Tile Game

CPS2002: Software Engineering
Practical Assignment

Luke Collins
University of Malta

Stefania Damato
University of Malta

20th May, 2018

Contents

1 Introduction **4**

2 Initial Game Implementation **5**

2.1 Test Coverage 6

3 Enhancements **7**

3.1 Different Map Types 7

3.2 One Map in Memory 8

3.3 Teams 9

3.4 Test Coverage after Enhancements 9

Bibliography **11**

I

Introduction

This documentation discusses the implementation of a Tile Game in the following git repository: https://stefaniatadama.github.io/cps2002-assignment.

The Jenkins page for this project can be found here, and javadocs can be found here. A precompiled jar file can be found in the root directory of the repository, so to run the game, simply navigate to the root in a terminal, and run the command:

```
java -cp game.jar edu.um.cps2002.tile_game.Launcher
```

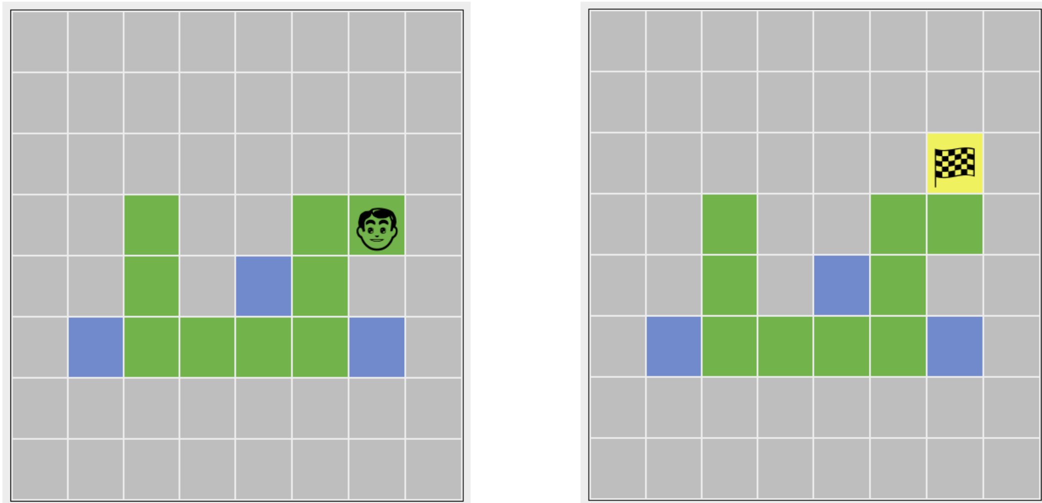

Figure 1.1: A screenshot of gameplay

The interactions with the repository were all done using git flow. Therefore as git flow practices dictate, most commits were performed in appropriate feature branches, merging into develop at the end of a feature implementation, and finally into master for releases. Unfortunately, for the second part of the assignment (meaning the development of the basic game), all feature branches were deleted since this is what git flow does by default when a feature branch is closed. Therefore the only commits visible in the history are the merges into develop. For the third part, feature branches were kept using the `-k` tag (k for 'keep').

There are two important tagged releases: release 1.1 for the second part of the assignment, and 2.0 for the third part.[1]

[1] Release 1.0 had a tiny bug which we fixed.

II

Initial Game Implementation

For the second part of the assignment, we were required to create a multiplayer game in which players compete to find the treasure on a map. Tiles on the map can be of type grass, water or treasure. The players always start on a grass tile, and if they step on a water tile they go back to their initial position. The player who finds the treasure tile first wins. Our initial implementation involved an abstract class `Map` with two subclasses `GameMap` and `PlayerMap`. `GameMap` held the complete map generated in the beginning of the game, while each player had an instance of `PlayerMap` which only held information about tiles that the player has already explored. The UML class diagram of this part is shown in figure 2.1.

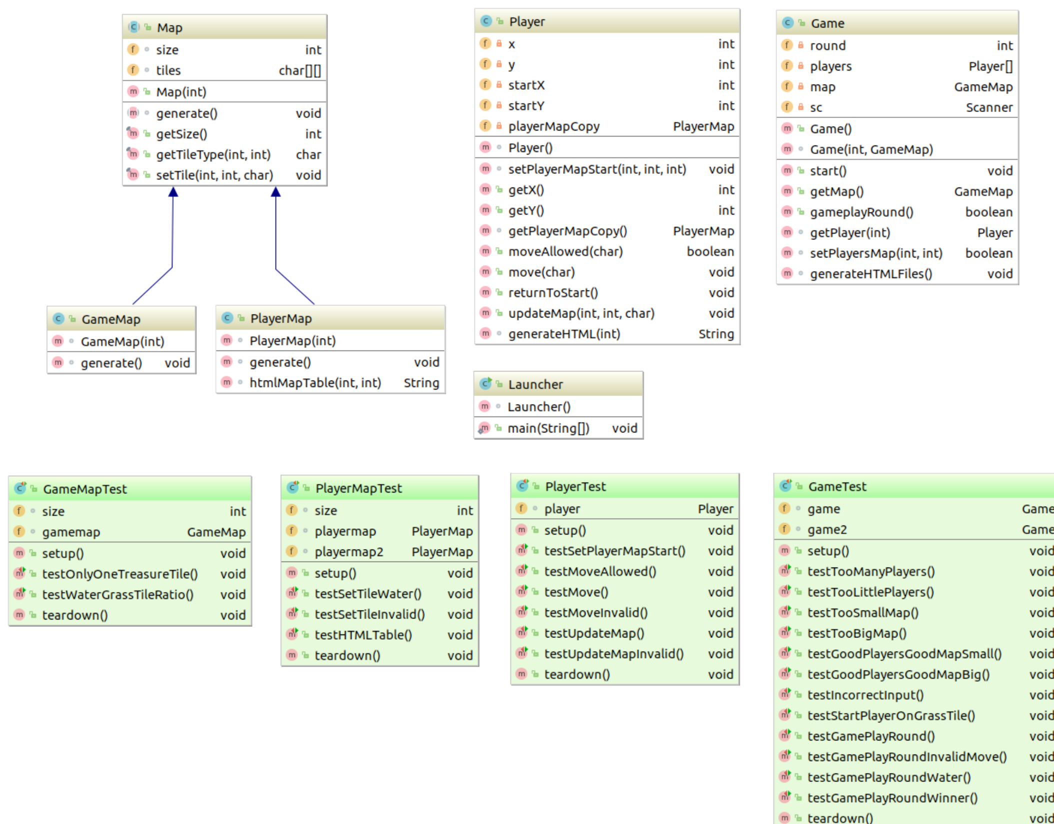

Figure 2.1: UML Diagram for initial implementation of the game

2.1 Test Coverage

Code coverage metrics are given in table 2.1. The `Launcher` is not tested due to the fact that it simply intialises an instance of the `Game` class and runs methods which are tested elsewhere. Moreover, testing it would require knowledge of the randomly generated map and initial player positions, which are not known before runtime. Other lines which were not tested involve throwing an exception, or writing of the HTML files which would require comparing files to be tested.[2]

Package	Class, %	Method, %	Line, %
`edu.um.cps2002.tile_game`	83.3% (5/6)	86.2% (25/29)	86.4% (153/177)

Class	Class, %	Method, %	Line, %
`Game`	100% (1/1)	87.5% (7/8)	89.6% (60/67)
`GameMap`	100% (1/1)	100% (2/2)	100% (18/18)
`Launcher`	0% (0/1)	0% (0/2)	0% (0/10)
`Map`	100% (1/1)	100% (4/4)	100% (11/11)
`Player`	100% (1/1)	90% (9/10)	86% (37/43)
`PlayerMap`	100% (1/1)	100% (3/3)	96.4% (27/28)

Table 2.1: Code coverage for the initial implementation of the game

[2]Note that the actual generation of HTML code was tested, it was simply testing writing to files which was omitted.

III

Enhancements

The third part of the assignment involved the implementation of three enhancements to the game program described in the previous part, and the use of an appropriate design pattern in each case.

3.1 Different Map Types

The first improvement required us to support different types of maps in the game, namely a *safe* map type and a *hazardous* map type. More map types could possibly be added in the future, so the design was required to facilitate any future additions. To meet these requirements, the **factory method** design pattern was employed. This design pattern defines an interface for creating an object, but let subclasses decide which class to instantiate.[3] Since the user decides at runtime what type of map to use, we can't predict the subclass of `Map` to instantiate. The factory method is therefore ideal since it encapsulates the knowledge of which `Map` subclass to create and moves it out of the framework to be dealt with at runtime. Also, adding more map types simply involves creating new subclasses which extend `Map`, and an extra line in the `createMap()` method.

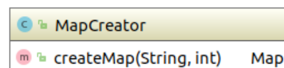

Figure 3.1: Map Creator UML Block

The factory method `createMap()` is given in listing 3.2.

```
1 public class MapCreator {
2     public GameMap createMap(String type, int size){
3         if(type.equals("H"))
4             return new HazardousMap(size);
5         if(type.equals("S"))
6             return new SafeMap(size);
7         return null;
8     }
9 }
```

Listing 3.1: `MapCreator` class

[3]Since this design pattern was only to be used for `Map` objects, the interface classes `Creator` and `Product` were omitted, and only the `MapCreator` class was implemented. In general, the use of interfaces in design patterns is a matter of object-oriented design, and have nothing to do with the design patterns themselves.[1]

3.2 One Map in Memory

For the second improvement, the game was required to store precisely one instance of the map in memory, yet still to present each player with a version of the map in which only explored tiles are visible. To this end, the **singleton** design pattern was employed, ensuring that the program has only one instance of `Map` and with global access . To avoid the `Player` class having to store the map in any form,[4] the `Tile` class was created to keep track of which players in the game have visited that particular tile. The `Map` class was then modified to store an array of `Tile`s instead of `char`s.

In our implementation, all `Map` objects are created by the `MapCreator` class (due to the previously implemented design pattern). Therefore the singleton pattern was applied to the `MapCreator` class by declaring the returned `Map` static, and returning a reference to the map already declared should any subsequent calls to `createMap()` be made.[2]

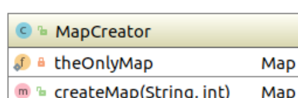

Figure 3.2: Map Creator UML Block

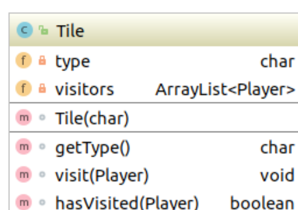

Figure 3.3: Map Creator UML Block

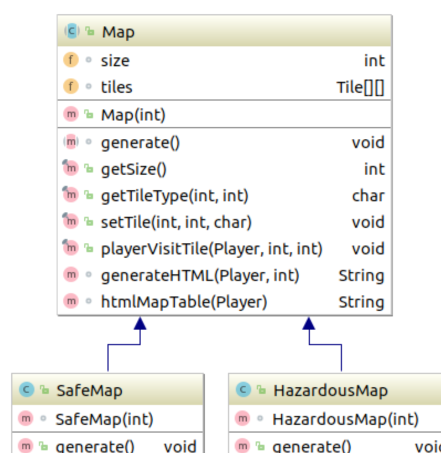

Figure 3.4: Map Class Hierarchy

```
public class MapCreator {

    private static Map theOnlyMap;

    public synchronized Map createMap(String type, int size){

        if(theOnlyMap != null)
            return theOnlyMap;
        if(type.equals("H"))
            theOnlyMap = new HazardousMap(size);
        if(type.equals("S"))
            theOnlyMap = new SafeMap(size);

        return theOnlyMap;
    }
}
```

Listing 3.2: `MapCreator` class

[4]Using a two-dimensional array of booleans felt like cheating.

3.3 Teams

The third improvement involved the addition of teams to our game. Each player belongs to a team, and if a player uncovers a tile, the other players in the same team can also see the uncovered tile.

Due to our design choice in which `Tiles` keep track of the players who have visited them, this could easily have been done by changing the `ArrayList<Player>` in the `Tile` class to an `ArrayList<int>`, which would store the team number of each player which has visited the tile, instead of a reference to individual players. However since it is in the spirit of the assignment to make use of design patterns, we opted to use the **observer** design pattern.

The observer pattern takes care of one-to-many relationships by establishing *subjects* and *observers*. Similar to the relationship between a newspaper publisher and a reader, whenever an observer undergoes a change of state, it informs the subject which then informs all other observers. In our case, the team is the subject and the players are the observers. When a player uncovers a tile, it notifies the team, which then updates the rest of the players with this new information. This relationship can be seen in the UML class diagram below.

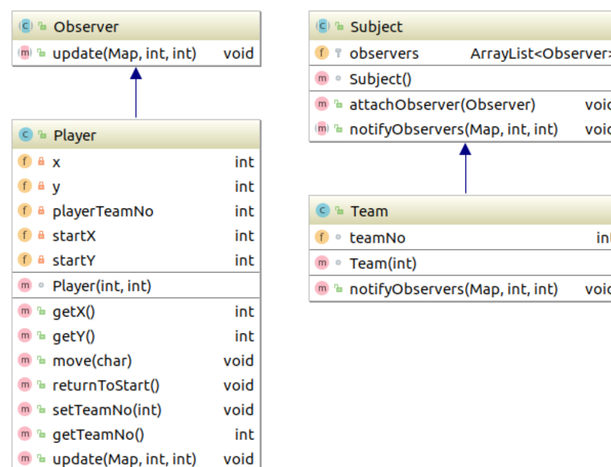

Figure 3.5: Abstract and concrete observer/subject classes

A screenshot of the enhancement in action can be seen in figure 3.6.

3.4 Test Coverage after Enhancements

Code coverage metrics of the program after the enhancements are given in table 3.1. Similar remarks to those of section 2.1 can be made about the `Launcher` class and the HTML file writing method in the `Game` class. Some of the old tests had to be redesigned to work within the changes to the game's structure, such as the inclusion of teams and the new `Tile` class.

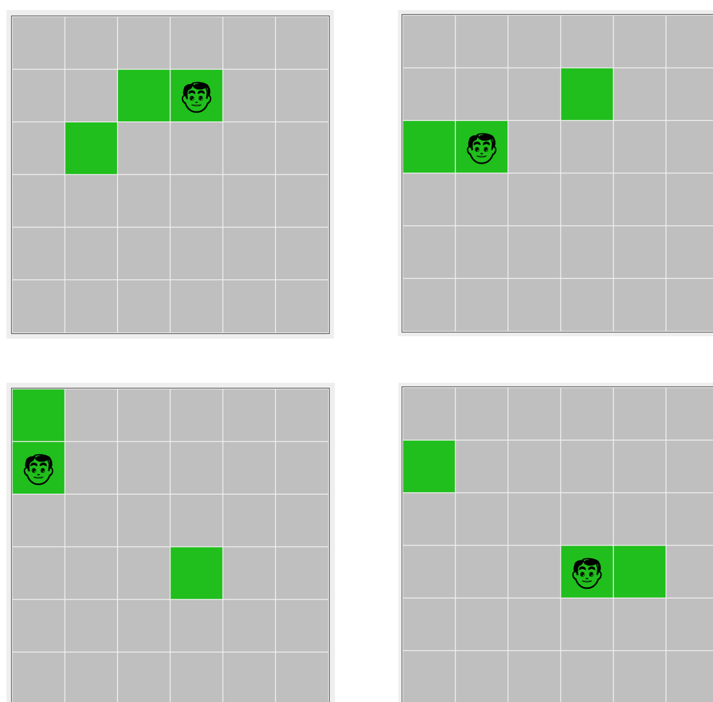

Figure 3.6: Illustration of teams after one round of gameplay. The player in the top left moved to the right, revealing this tile to his teammate on the top right. Similarly, the player on the top right moved left, revealing the new tile to his teammate. In the same way, the two players at the bottom share newly explored tiles. Note that the initial position of players is not shared with teammates.

Package	Class, %	Method, %	Line, %
`edu.um.cps2002.tile_game`	90.9% (10/ 11)	90.2% (37/ 41)	89.7% (217/ 242)

Class	Class, %	Method, %	Line, %
`Game`	100% (1/ 1)	88.9% (8/ 9)	91.7% (88/ 96)
`HazardousMap`	100% (1/ 1)	100% (2/ 2)	100% (20/ 20)
`Launcher`	0% (0/ 1)	0% (0/ 2)	0% (0/ 10)
`Map`	100% (1/ 1)	85.7% (6/ 7)	85% (34/ 40)
`MapCreator`	100% (1/ 1)	100% (2/ 2)	87.5% (7/ 8)
`Observer`	100% (1/ 1)	100% (1/ 1)	100% (1/ 1)
`Player`	100% (1/ 1)	100% (8/ 8)	100% (27/ 27)
`SafeMap`	100% (1/ 1)	100% (2/ 2)	100% (20/ 20)
`Subject`	100% (1/ 1)	100% (2/ 2)	100% (5/ 5)
`Team`	100% (1/ 1)	100% (2/ 2)	100% (6/ 6)
`Tile`	100% (1/ 1)	100% (4/ 4)	100% (9/ 9)

Table 3.1: Code coverage of the game after enhancements

Bibliography

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, first edition, November 1994.

[2] markspace (StackOverflow User). *Combining the Factory Method and Singleton Design Patterns*. https://stackoverflow.com/a/50379458/5664775, May 2018. Last Accessed: 20th May, 2018.