**Final Project – A Monopoly™ game**

1. **Introduction**

For the *« Design Pattern & Software Development Process »* course, we carried out this project as a group of 2, Yanis DAHMOUCHE and François COUTAU. The goal of the task is to simulate a simplified version of the famous Monopoly game. You will find our code [here](https://github.com/Franciskid/MonopolyQuickConsoleGame).

**Rules**

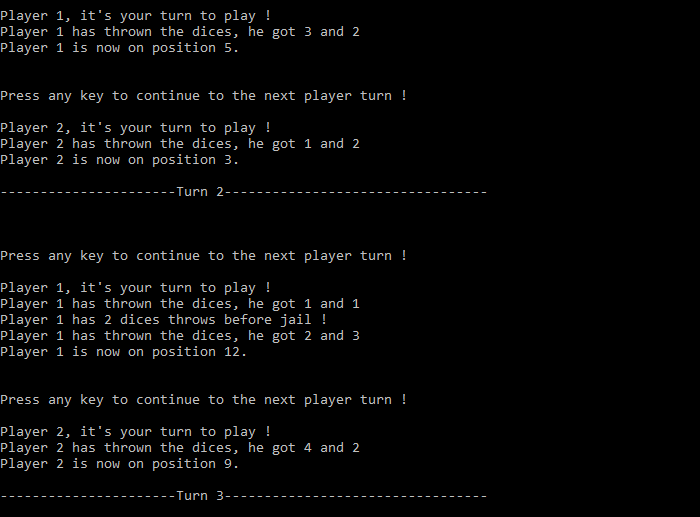
In this game, the players can only move throughout the board, which consists of 40 blocks. Each player has 2 dices in his hands, which he throws at every turn and moves forward of the sum indicated by the 2 dices. Players can go to jail if they throw the dices with the same value 3 times in a row or if they land on box number 30 (the « go to jail » box). They stay in prison for 3 turns unless they manage to throw the dices with the same value.

**Concrete objective**

The program needs to be implemented with some design patterns, as best implemented as we can and in a useful way.

1. **Design Hypotheses**

There were no specifications on the interface we had to do. So we decided to make a simple console application and update with a comments every time someone does something.



This allowed us to implement the observer pattern which, as its name indicated, observes every action done throughout the game and lets the user know by writing each action to the console.

We also implemented a View for the Monopoly game, this allows us to print a recap of the game whenever we desire, it looks like that :

Text

Description automatically generated

The assignment also let us decide which programming languages we could use so we chose *C#*.

Other than that, the assignment wasn’t pretty detailed on the way to do things so we were pretty free to do things as we wanted to.

1. **UML diagrams**
   1. **Class diagram of the solution**

A screenshot of a computer

Description automatically generated with low confidence

This is our class diagram.

As you can see we have implemented two design patterns for this project. The observable pattern which allows us to see and inform the user every time something happens during the game. It is very useful to better control to one place what will be displayed on the screen instead of looking at ‘Console.WriteLine’s dispatched through the whole code, which can get messy very fast.

We also used the Singleton pattern to make sure only 1 instance of the monopoly game could run at once.

* 1. **Sequence diagram**

1. **Test cases**

We used the built-in unit tester in Visual Studio to make the test-cases. We tested 4 functions and built-in functionnalities and design-patterns.

* 1. **Monopoly singleton test**

A screenshot of a computer

Description automatically generated with medium confidence

Test to check if there is only 1 instance of the monopoly class. We check the reference of the 2 objects.

* 1. **Monopoly observer test**

Graphical user interface, application

Description automatically generated

Test to check whether the game observer writes in the console what is expected when the game is initialized.

* 1. **Player observer test**

Graphical user interface, text, application

Description automatically generated

Test to check whether the player observer writes in the console what is expected when the player’s position is changed.

* 1. **Dice test**

Graphical user interface, text, application

Description automatically generated

Test to check whether the ‘Total’ property of the Dice class returns the expected value

1. **Conclusion**

This project was very instructive as it forced us to make the cleanest code we could do and the easiest to understand for anyone looking at it. We only used 2 defined design-patterns (we did use a view pattern for our main Monopoly class but it’s not very interesting), the Observer pattern and the Singleton pattern. We don’t feel like we could have used any more patterns, indeed we did have some ideas and we could have implemented some others but that would not have been very useful and would have only added unnecessary complexity to the project.