# Final Project – A MonopolyTM game – Design Patterns and Software Development Process

Table des matières

[Final Project – A MonopolyTM game – Design Patterns and Software Development Process 1](#_Toc92549774)

[1. Introduction 1](#_Toc92549775)

[2. Design Hypotheses 2](#_Toc92549776)

[3. UML diagrams 3](#_Toc92549777)

[a. Class diagram of the solution 3](#_Toc92549778)

[b. Sequence diagrams 3](#_Toc92549779)

[4. Test cases 3](#_Toc92549780)

[Singleton Test case 4](#_Toc92549781)

[Test case “Go to jail” after 3 doubles dices in a row 4](#_Toc92549782)

[Test case “Go to jail” case 4](#_Toc92549783)

[Test case “Go out of jail” after a double dice 5](#_Toc92549784)

[Test cases overview 5](#_Toc92549785)

[5. Conclusion/ Final remarks 5](#_Toc92549786)

## Introduction

As ESILV 4th year students, we attended the ‘Design Pattern and Software Development Process’ course. Along the course, we learnt theory and practice of many design patterns, processes and tools relevant to our studies. This project brings to a close this course, offering a final opportunity to use our newly acquired skills through a practical project : the goal was to simulate a simplified version of the Monopoly game.

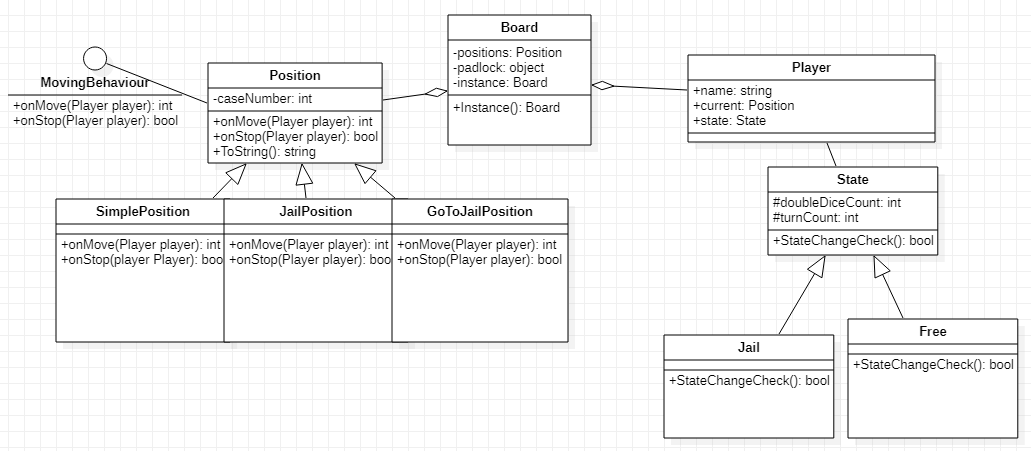
The concept offers several possibilities for design patterns. After exploring possibilities, we went with the one that seemed most fitting to us as explained below. We put into practice the good manners we were accustomed to regarding the modelling. You shall find all explanations in the corresponding sections.

As for the simplified version of the game, it consists of a similar board, having 40 possible positions that the players travel across during the game. The part where the positions represent famous avenues has been removed, then there is no buying or deals at all during the game. The only remaining game features are the jail, located at tile 10, and the go-to-jail tile at the 30th position. All the moving around also remains the same as in the original game (2 dices, moving by the sum of the dices, 1 double and the player plays again, 3 doubles make the player go to jail, 1 double to go out of jail or after 3 turns).

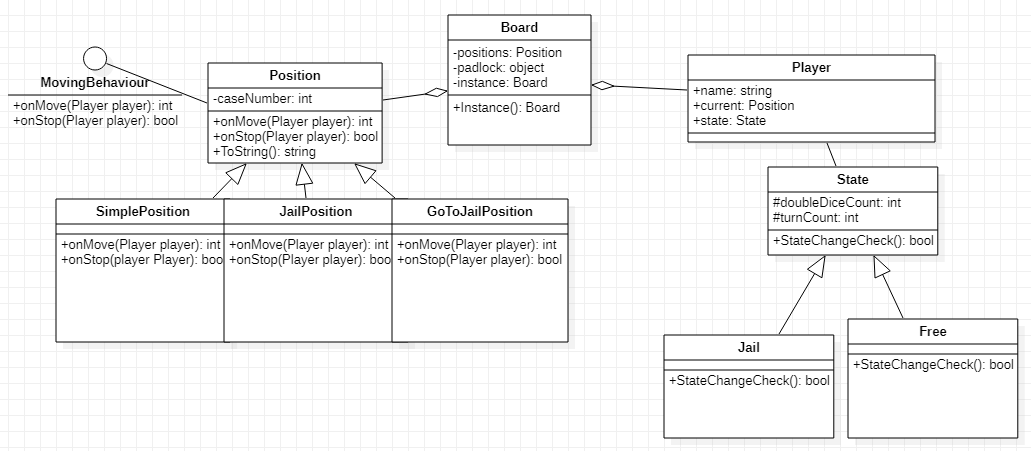
Let’s move on to the design hypothesis.

## Design Hypotheses

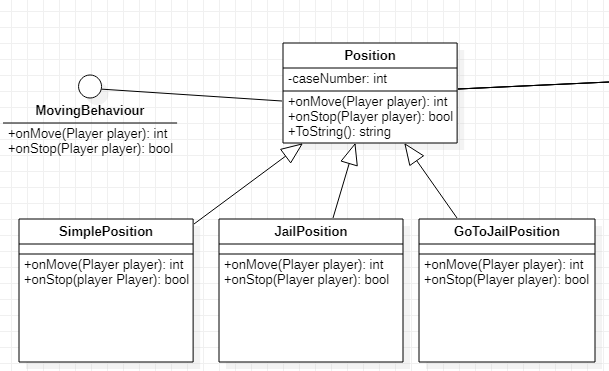
Singleton pattern



State pattern

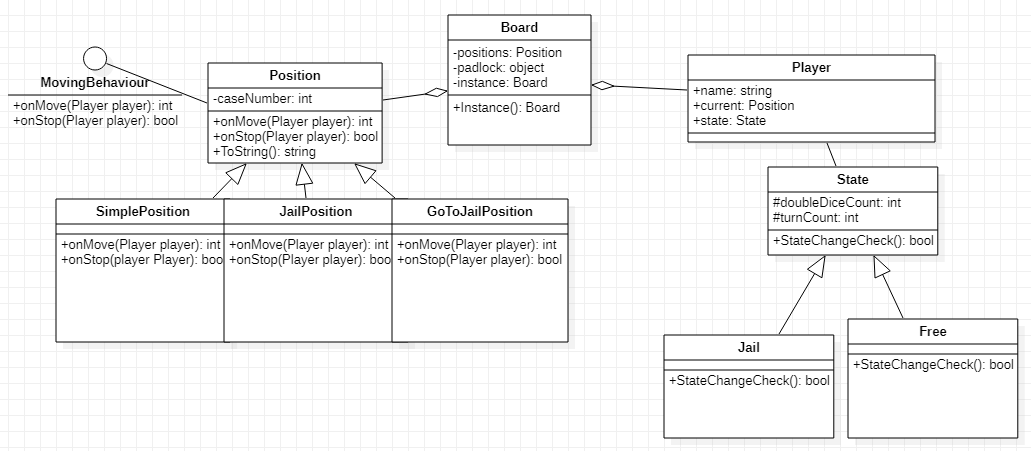


Strategy pattern:



## UML diagrams

### Class diagram of the solution



*This class diagram was made on StarUML*

### Sequence diagrams

## Test cases

Once we had implemented all our patterns, we ran the program to see if the game was working correctly.

We then decided to test more precisely the 4 major functionalities of our project:

* The creation of a unique Board thanks to our Singleton pattern
* The “3 doubles dice in row” case leading to Jail case
* The “double dice” case that allows the player to get out of jail
* The case “Go to jail”

In order to do these tests, we created another Unit Test Project on Visual Studio.

### Singleton Test case

Une image contenant texte

Description générée automatiquement

In order to trigger the exception set up in the Singleton Pattern, we tried to create another instance of a board.

We obtained the expected result, only one board was created thanks to our Singleton Pattern protecting the uniqueness existence of a board.

### Test case “Go to jail” after 3 doubles dices in a row

Une image contenant texte, capture d’écran, écran

Description générée automatiquement

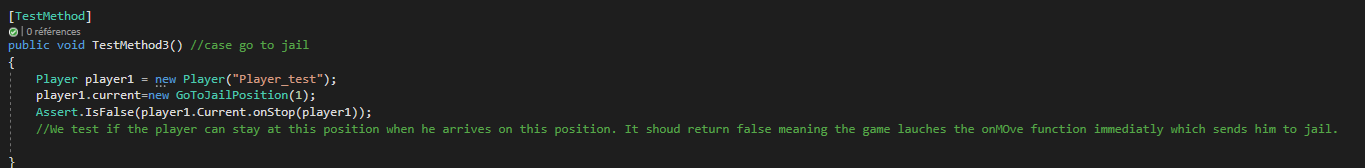
We created two different cases to test our State Pattern. The first one with a player having done 2 double dices to see if the Boolean method StateChangeCheck() was returning False, and the second one with the same player having done 3 double dices in a row, to check if the method was returning True.

We obtained the expected results for both tests, for the first one nothing happened and the state of player was still “Free” and for the second test, the player’s state changed from Free to Jail.

### Test case “Go to jail” case

In the game, if a player stops on the position 30, which is the Go-to-jail position, he does not stay here until the next turn like other basic positions. Instead, the player has to move immediately to jail, and end its turn there even if he did a double. In order to test if that event was going as expected, we created a player and an instance of this position. We made the player stop on that instance by setting it as its current position, then called the onStop method. This method, implemented on all types of positions, is the one designed to return true if the player’s turn is over or false if not. We indeed expected it to return false as a player can not stop on the go-to-jail case, hence tested it using Assert.IsFalse.

The test was successful. Here is the code snippet:



### Test case “Go out of jail” after a double dice

In Monopoly, players might end up in ‘jail’. There are two ways to get out : either make a double on the player’s turn, or leave after 3 turns have gone by. We wanted to test that the first exit scenario was well implemented. To test that, we created a player and immediately set is State as Jail. We set the variable DoubleDiceCount to 1, imitating the case where the player just made a double. Then, we called the StateChangeCheck method, designed to return true if the player needs to change state (jail to free or free to jail). In our case, the player is supposed to leave jail, so is state should go from Jail to Free, and the function needs to return true. That’s what we test using the Assert.IsTrue function.

The test was successful. Here is the code snippet:

Une image contenant texte

Description générée automatiquement

### Test cases overview

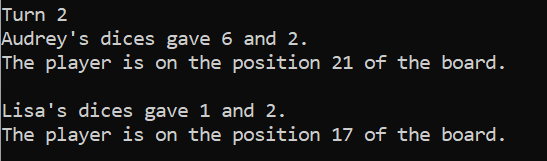
We then tested each Unit Test and as we can see below on the picture, we obtained the expected results as they were all successful.

Une image contenant texte, capture d’écran, moniteur, écran

Description générée automatiquement

## Conclusion/ Final remarks

To put it in a nutshell, the project allowed to us to apply what we had seen along the course. We successfully implemented 3 design patterns : Singleton, State, and Strategy patterns. Our project meets all requirements and works well, allowing us to simulate a game of the simplified Monopoly without interfering. The program prints some information along the game, like the value of the dices, players’ positions after the turn:



The game stops on its own after 50 turns.

On an improvement perspective, we could have added features to our project to make it more enjoyable: chose the number and names of players with the console, add a visual of the board throughout the game…

We are quite satisfied by our project, both in terms of final results and the technical experience and improvement. We had a great time working as a team to make this project become real within a few weeks to meet the deadline and are pleased to submit our work.