

Exercise 3 – A Monopoly™ game – Design patterns (8 points)

Objective of the exercise is to simulate a simplified version of the Monopoly™ game.

In the following, a summary of the rules – to necessary respect in the design and implementation – is provided. Other constraints and details are free to define by the students, and set as design hypotheses.

- The game is played on a circular game board, composed of 40 positions on the board, indexed from 0 to 39
 - A set of players is given, each with name and initial position. Dice are rolled and players' positions on the game board will change
 - If a player reaches position 39 and still needs to move forward, he'll continue from position 0. In other words, positions 38, 39, 0, 1, 2 are contiguous
 - Each player rolls two dice and moves forward by a number of positions equal to the sum of the numbers told by the two dice
 - A player's turn ends after having moved to its new position
 - The same position can be occupied by more than one player
 - If a player gets both dice with the same value, then he rolls the dice and moves again. If this happens three times in a row, the player goes to jail and ends his turn
 - Jail can be a situation the player is in, or a place he pays visit to. The board therefore has a Visit Only / In Jail cell at position 10. A Go To Jail cell is also present, at position 30
 - If at the end of a basic move, the player lands on Go To Jail, then he immediately moves to the position Visit Only / In Jail and is in jail. His turn ends
 - If after moving, the player lands on Visit Only / In Jail, he is visiting only and is not in jail
 - While the player is in jail, he still rolls the dice on his turn as usual, but does not move until either:
 - (a) he gets a both dice with the same value, or
 - (b) he fails to roll both dice with the same value for three times in a row (i.e., his previous two turns after moving to jail and his current turn)
- If either (a) or (b) happens in the player's turn, then he moves forward by the sum of the dice rolled positions and his turn ends. He does not roll the dice again even if he has rolled a both dice with the same value.

1. Introduction

The problem is a game of Monopoly. Before the game starts, the number of players, their name and position are unknown.

There is only one board for the game, on which players can roll two dice and move forward. The board is composed of 40 cases, numbered from 0 to 39 and forming a cycle. Only two of them are specials : the “Visit Only / In Jail” (at position 10) and “Go To Jail” (at position 30) cases. The rest are classic cases, not doing anything.

2. Design hypotheses

The board is unique, always with the same case layout.

Players have the same characteristics : a name and a position (and the initial one). Their position is represented by a piece which they can choose at the start. They can make only one action : rolling the dice. They are determined by the board (two special cases) and if they make a double (rolling again the dice) or three doubles (go to jail).

So we have six classes : *Board*, *Piece*, *Case*, *Dice*, *Player*, and the *Game* itself. Having such classes allow us later to change the number of dice, create new cases, change rules, ... without having problems of development.

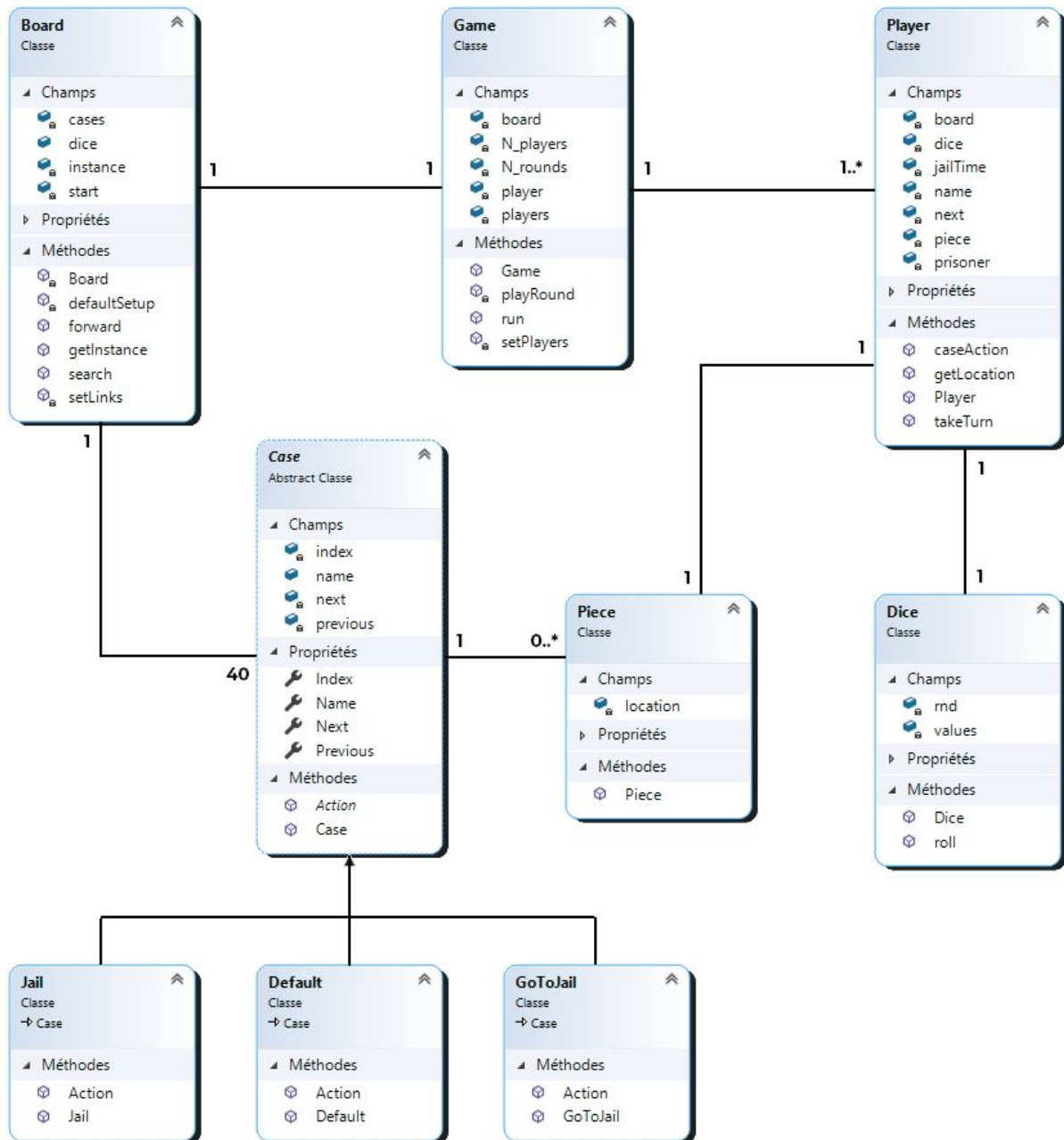
We use a singleton pattern for the *Board*, because there is only one board, and we don't want any duplicate.

For *Player* and *Case*, we use the iterator pattern. *Game* and *Board* are their iterator. It's a great way to design it : there are n players and they play one by one. We make a linked list of players in the *Game* class, and at the end of the turn, we move to the next player thanks to his *next* property.

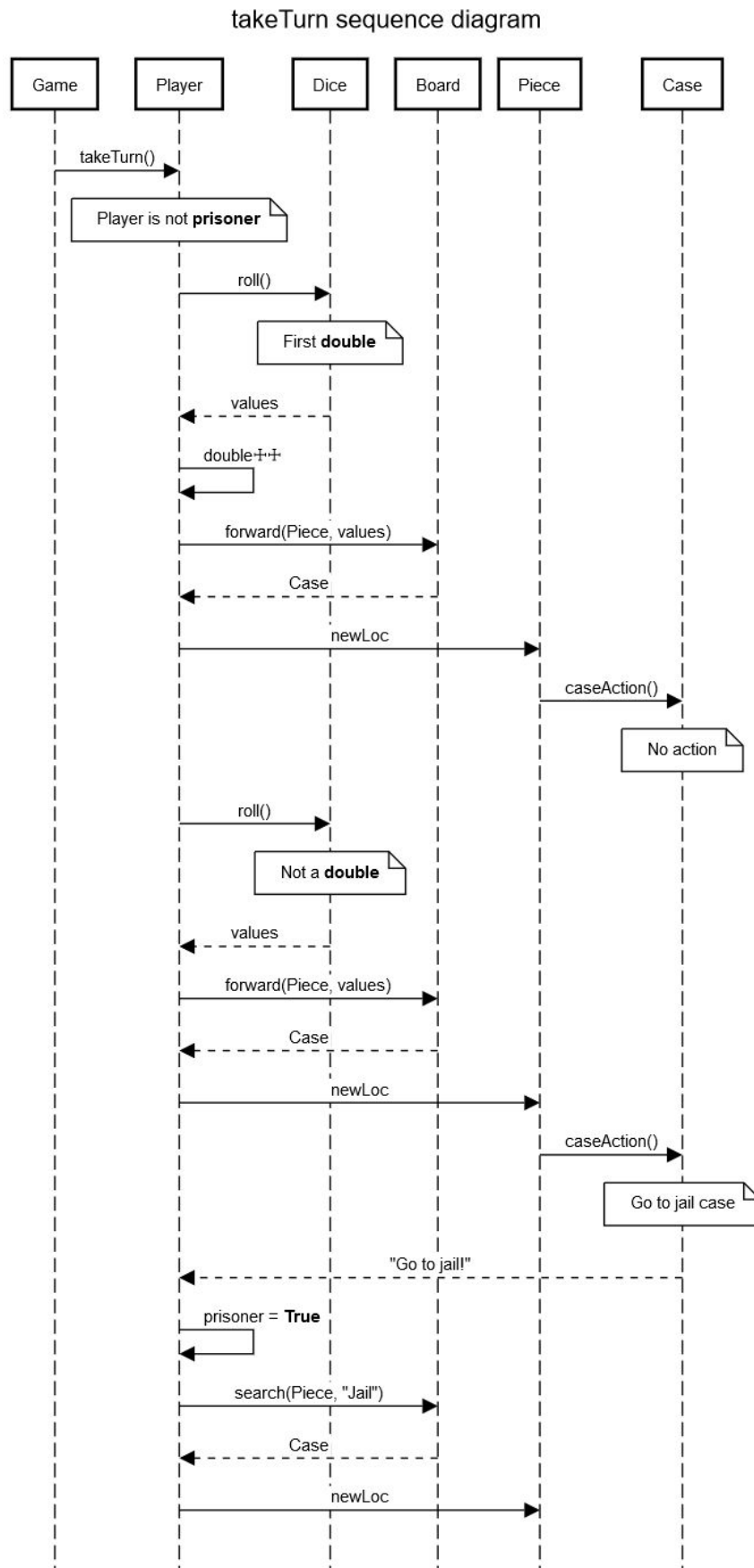
It is the same for the cases. There are 40 cases, in the *Board* constructor we make a doubly linked list out of them. “Doubly” because maybe some cards or cases have as an action to “Go back to ...” instead of “Go to ...”.

3. UML diagrams

a. Class diagram of the solution



b. Sequence diagrams

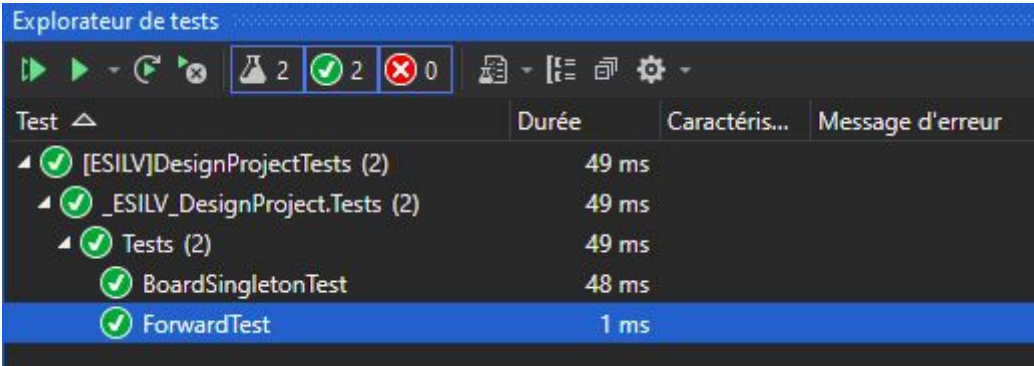


4. Test cases

We have two unit tests :

- Testing that the board is indeed a singleton. We create two board objects *board1* and *board2* and we check that they both are linked to the same instance, in fact that they are equal.
- Testing that all the methods to push forward the piece are well functioning. We create a board and a player with his piece and his dice. We get his index before the play and we roll the dice. We sum the values with the index and in parallel we use the forward mechanism on his piece. At the end we watch if the sum we got matches with the final index.

Both unit tests worked well.



Test	Durée	Caractéris...	Message d'erreur
✓ [ESILV]DesignProjectTests (2)	49 ms		
✓ _ESILV_DesignProject.Tests (2)	49 ms		
✓ Tests (2)	49 ms		
✓ BoardSingletonTest	48 ms		
✓ ForwardTest	1 ms		

5. Additional / Final remarks

There is a *Dice* instance in the *Board* class, it is because of the way randomness acts in the native package *Random*. It is pseudo-random, and when instantiated in the *Player* class, the dice were giving only doubles.