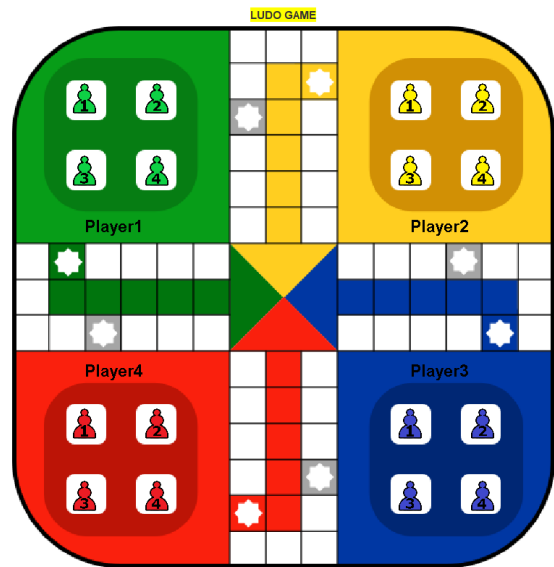


# Programming a Ludo Game

Project report

Florian CLOAREC  
Théo DURR



LP2A - Introduction to object conception and programming

Spring 2021

Université de Technologie de Belfort-Montbéliard



## Contents

|          |   |          |
|----------|---|----------|
| <b>1</b> | <b>Introduction</b>   | <b>2</b> |
| 1.1      | Project Analysis . . . . .  | 2        |
| 1.2      | Organization . . . . .  | 2        |
| 1.2.1    | Tools used . . . . .  | 2        |
| 1.2.2    | Documentation of the source files . . . . .                         | 2        |
| <b>2</b> | <b>The game components</b>  | <b>2</b> |
| 2.1      | The Players . . . . .   | 3        |
| 2.1.1    | Humans . . . . .  | 3        |
| 2.1.2    | Artificial Intelligences . . . . .                                  | 4        |
| 2.2      | The Coordinates System . . . . .                                    | 4        |
| 2.3      | The board . . . . .   | 4        |
| 2.3.1    | The pieces & the dice . . . . .                                     | 5        |
| <b>3</b> | <b>The game engines</b>   | <b>5</b> |
| 3.1      | Managing collisions . . . . .                                       | 5        |
| 3.2      | Gamemodes . . . . .   | 5        |
| 3.2.1    | 4 Artificial Intelligences . . . . .                                | 5        |
| 3.2.2    | 4 Players . . . . .   | 5        |
| 3.2.3    | 1 Player versus 3 Artificial Intelligences . . . . .                | 6        |
| <b>4</b> | <b>The graphical user interface (GUI)</b>                           | <b>6</b> |
| 4.1      | Aspect of the game . . . . .  | 6        |
| 4.2      | Interactions with the user . . . . .                                | 6        |
| 4.3      | Translating relatives coordinates in absolute coordinates . . . . . | 6        |
| <b>5</b> | <b>Conclusion</b>   | <b>6</b> |
| 5.1      | Improvements paths . . . . .  | 6        |
| 5.2      | Benefits of this project . . . . .                                  | 6        |
|          | <b>Appendices</b>   | <b>7</b> |
| <b>A</b> | <b>Unified Modeling (UML) Diagram</b>                               | <b>7</b> |
| A.1      | Engine . . . . .  | 7        |
| <b>B</b> | <b>Simplified algorithm</b>   | <b>8</b> |

# 1 Introduction

## 1.1 Project Analysis

First of all, we performed an analysis of the subject by defining, the behaviour, the constraints or even the signature of each class and represented all of this in an **UML Class Diagram**.

## 1.2 Organization

### 1.2.1 Tools used

We also used some tools like IntelliJ Idea as IDE, for time saving tasks like compiling, creating packages, making unit tests, and so on. As a **Version Control System (VCS)**, we used *Git*. It allowed us a better collaboration, a way to backup our files and, last but not least : It is a tool widely spread in the computer science jobs. So performing on this tool is a real benefit for the future.



Moreover, we decided to write this report in latex, as it is easier to insert some code, and mathematical formulas.

### 1.2.2 Documentation of the source files

As we used IntelliJ Idea, documenting the code was really simple. We trigger the snippet by typing `/**`, then by pressing `<Tab>` and it generates the documentation template, all we need to do is to fill in the description of the parameters and the return, and to provide a description of the function.

---

```
// Example of documentation generation
/**
 * PUT DESCRIPTION HERE
 * @param position PUT PARAMETER DESCRIPTION HERE
 * @return PUT RETURN DESCRIPTION HERE
 */
```

---

# 2 The game components

This project is split into two main components : *game engines* and *graphical user interface (GUI)*. The engines are the core of the game. They ensure IAs are following certain rules, as well as player, who are restricted in their choices with the GUI by displaying only "rules friendly" moves. We will detail some of the fonctionnalities.

Before explaining the engines, we need to talk about the components needed for it. They play a main role in the OOP. Yet, they are quite simple. For a summarized view of the structure, see APPENDIX X

## 2.1 The Players

The players are the entities playing the game. They have several actions such as moving a Piece, throw a dice, or pass. However, they are two types of player : real players (Humans) and Bots (Artificial Intelligence), their actions would be different, for instance when moving a piece. By following OOP principles, all the player types inherit from an abstract parent class called `Player`.

### 2.1.1 Humans

Humans are described in the child class `Human`. The difference between them and AIs results in the actions with the board, such as throwing a dice, choosing the piece to move, as there is some GUI implemented in the class. Here is the code for the `choosePiece` method :

---

```
// Human.java
public Piece choosePiece() {
    List<Piece> movablePiece = this.getMovablePieces();
    if (movablePiece.size() == 0) {
        JOptionPane.showMessageDialog(null, this.getName() + " : You can't move any
            piece, you have to pass");
        return null;
    } else {
        movablePiece.sort(new SortByBestMove());
        Collections.reverse(movablePiece);
        String[] stringMovablePiece = new String[movablePiece.size() + 1];

        for (int i = 0; i < movablePiece.size(); i++) {
            stringMovablePiece[i] = "" + movablePiece.get(i).getNumber();
        }
        stringMovablePiece[movablePiece.size()] = "pass";
        String result = (String) JOptionPane.showInputDialog(null, this.getName() + "
            : Choose the piece that you want move " + this.getDice().getValue() + "
            cases forward", this.getName() + " : Choose Piece",
            JOptionPane.QUESTION_MESSAGE, null, stringMovablePiece,
            stringMovablePiece[0]);

        if (result == null) {
            System.exit(1);
        }

        int intResult;
        try {
            intResult = Integer.parseInt(result);
        } catch (NumberFormatException e) {
            return null;
        }
        return this.getPieces()[intResult - 1];
    }
}
```

---

We can see that there is some GUI code inside the `Human` class. We found it was a good compromise between code organization and code efficiency.

### 2.1.2 Artificial Intelligences

Artificial Intelligences (AIs) work slightly different : there is no GUI code, as they are managed by the computer. So the work here was to implement the reflexion of the AI. As the main goal of this project was not to do machine learning, we simply implement a procedural set of `if/else` statements in a separate java class:

---

```
// SortByBestMove.java
public class SortByBestMove implements Comparator<Piece> {
    @Override
    public int compare(Piece piece1, Piece piece2) {
        if (piece1 == null) {
            return -1;
        } else if (piece2 == null) {
            return 1;
        } else if (piece2.isAtStable() && !piece1.isAtStable()) {
            return -1;
        } else if (piece1.isAtStable() && !piece2.isAtStable()) {
            return 1;
        } else if (!piece1.isAtImmuneSquare() && piece2.isAtImmuneSquare()) {
            return 1;
        } else if (!piece2.isAtImmuneSquare() && piece1.isAtImmuneSquare()) {
            return -1;
        } else {
            return piece1.getPosition().getProgress() -
                piece2.getPosition().getProgress();
        }
    }
}
```

---

## 2.2 The Coordinates System

Each position object is composed of a color which is the color of the player and an integer "progress" which is the number of squares covered from the starting square. In the `PositionConstant` class, we make the link between the name of the important squares and the value of progress to which they correspond. Thus the stable is worth -1, the departure 0 and the arrival or Home is worth 57. Thanks to this system, the progression of each player on the game board is easy to manage, because it is enough to increment the number of squares you want to advance. But the problem arises when you want to compare the positions of the pieces of the other players, because each position is given in a base relative to each player represented by the color of this player. To do this we need to create a function that allows to convert a position from one color to another. To do this, using modular arithmetic and basic operations, we calculate the position in the new base (See subsection 5.2). Once this value has been calculated, the method creates a new `Position` object with this calculated value and the desired color.

## 2.3 The board

The board of the game is quite simple. It contains all the players (no matter which type they are), and a Dice. Its complexity only results in its constructor, which is depending of the gamemode (See subsection 3.2). The `Board` class contains some methods, such as `getPiecesAtCoordinates` :

---

```
/**
 * Computes the pieces at given coordinates
 * @param position the position to check
 * @return List of piece if found, null either
 */
public List<Piece> getPiecesAtCoordinates(Position position){
    List<Piece> result = new ArrayList<>();
    for (Player player : getPlayers()) {
        for (Piece piece : player.getPieces()) {
            if (piece.getPosition().equals(position)) {
                result.add(piece);
            }
        }
    }
    if (result.isEmpty()) {
        return null;
    } else {
        return result;
    }
}
```

---

### 2.3.1 The pieces & the dice

## 3 The game engines

They are 3 engines in total. Each engine is called depending of the gamemode. By following OOP principles, all the engines inherit from an abstract parent class called **Engine**.

### 3.1 Managing collisions

Managing collisions is a central feature of the game, it ensure pieces go to the stable according to the rules of the game (and according to the ??)

This feature is required by the rules of the game, we implemented it, but with more time, we could have tested it. We might expect some bugs.

### 3.2 Gamemodes

#### 3.2.1 4 Artificial Intelligences

This gamemode was not required in the specifications, but we made it for *experimental purposes*. It allowed us to check if everything was working properly without having a GUI. The four players are AIs and are playing by following the rules.

#### 3.2.2 4 Players

This gamemode unlieve the entire potential of the GUI, by allowing four players to play in the same time on the same computer. Each player plays one after the other.

### **3.2.3 1 Player versus 3 Artificial Intelligences**

## **4 The graphical user interface (GUI)**

### **4.1 Aspect of the game**

### **4.2 Interactions with the user**

### **4.3 Translating relatives coordinates in absolute coordinates**

## **5 Conclusion**

### **5.1 Improvements paths**

### **5.2 Benefits of this project**

# Appendices

## A Unified Modeling (UML) Diagram

### A.1 Engine



## B Simplified algorithm

