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

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.

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 functionnalities.

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 4.3). Once this value has been calculated, the method creates a new Position object with this calculated value and the desired color.

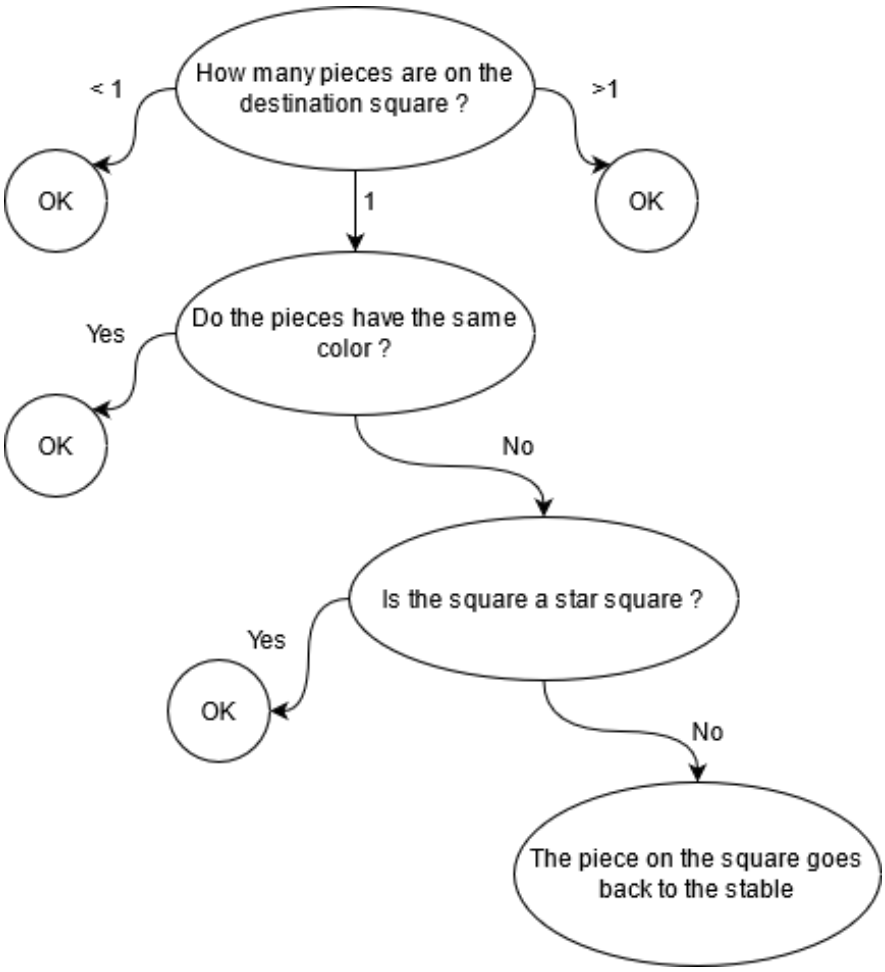
2.3 The board

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 Gamemodes

3.2 Managing collisions



3.3 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.

3.4 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.

Appendices

A Unified Modeling (UML) Diagram