# Rapport de projet LP2A : LudoGame



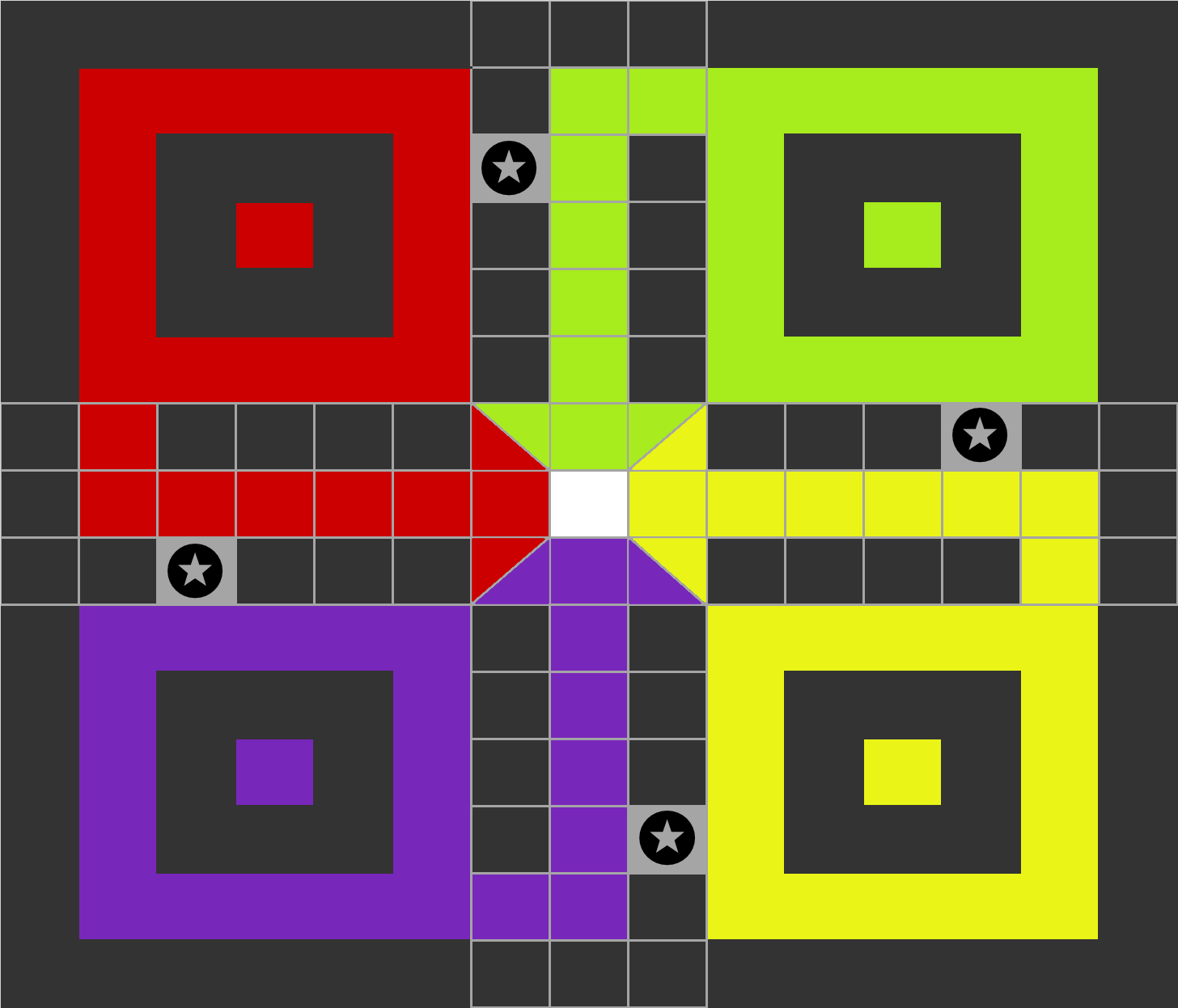


Table des matières

[Rapport du projet d’IFB : Jeu de Belote Coinchée en C dans la console 1](#_Toc70348823)

[Introduction 1](#_Toc70348824)

[Sources utilisées 2](#_Toc70348825)

[Description of development and implementation choices 2](#_Toc70348826)

[“UML” diagram 3](#_Toc70348827)

[The different classes used 3](#_Toc70348828)

[path 3](#_Toc70348829)

[Piece 3](#_Toc70348830)

[Player 3](#_Toc70348831)

[Position 4](#_Toc70348832)

[Square 4](#_Toc70348833)

[SquareArray 4](#_Toc70348834)

[Team 4](#_Toc70348835)

[Scene builder and JavaFX 4](#_Toc70348836)

[Global operation of the code 5](#_Toc70348837)

[The GUI (main) 5](#_Toc70348838)

[The Controllers 5](#_Toc70348839)

[Difficulties encountered 6](#_Toc70348840)

[Conclusion 6](#_Toc70348841)

# 

# Introduction

The aim of this project was to develop a Ludo game using Java and object oriented programming methods, the specification were to “implement the Ludo game using JAVA respecting the original rules” with a compulsory 4 players mod and an optional 1 player vs 3 “COMs”. As a quick reminder Ludo is a Board game in which 4 players each have 4 pawns and the aim is to bring all their pawns to the end of their path, by rolling a dice and moving their pawns... We also needed to implement a graphical interface.

## Sources utilisées

<https://www.belote.com/regles-et-variantes/regle-belote-coinche/>

<http://www.ffbelote.org/regles-coinche/#7>

<https://mon.gameduell.fr/gd/s03.do?gametype=bel&top>

<https://openclassrooms.com/fr/courses/19980-apprenez-a-programmer-en-c>

<https://openclassrooms.com/fr/courses/2342361-gerez-votre-code-avec-git-et-github>

<https://www.youtube.com/watch?v=x7IIDycK04M>

<https://stackoverflow.com/>

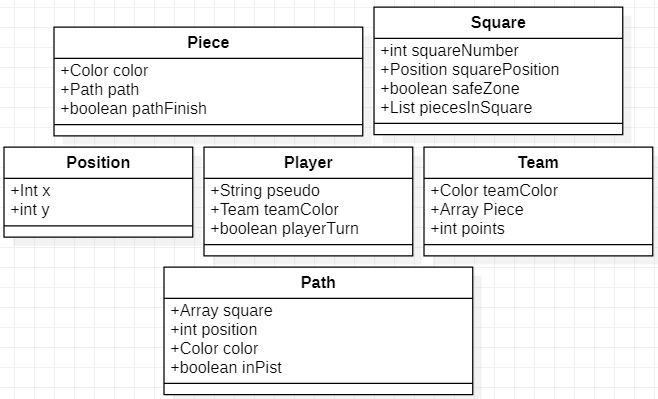
# 

# Description of development and implementation choices

In this part we will try to explain how we organized the project structure, we will describe the different classes we created and their utility.

So of course before starting to develop our ludo game we created a diagram in order to define what classes we were going to need and what attribute was needed for each class.

## “UML” diagram



## 

## The different classes used

Our game is based on several classes which are designed to respect the pillar of abstraction and encapsulation in oriented object programming, indeed most of the methods are simple to use and easy to reuse and modify and attributes of each class are private and can be accessed from the outside using getters and setters.

Here is a quick description of each class we created.

### 

### path

This class is used to define the path that the pawn will follow depending on their color, this class specifically contains the method use to initialized the path of the pawn depending on their color, this path is created and store thanks to an array.

### Piece

This class is used to define the pawns, the attributes are a color, a path, and a boolean telling us if the pawn have reached the home. The class mainly contains constructor, getters and setters used to access the information concerning a pawn.

### Player

The player class is used to define a player with a nickname and a Team, it mainly contains getter and setter the main part of the work is done in the Team class. In this class we also find the method used to initialise the player by defining their nickname and team color.

### Position

The class position is used to manage coordonate of the pawn on the board and is especially useful to display the pawns with the graphical interface.

### Square

This class is used to define a box of the Ludo Board, we chose to define a box with the following attribute: A int number representing the number of the box, a position, a boolean defining if the case is a safe zone or not.

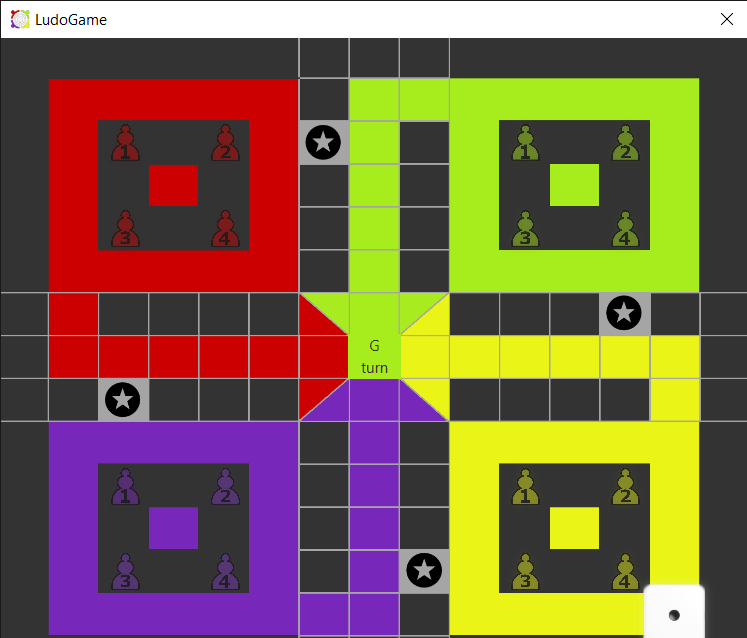
### SquareArray

The Class squareArray is a class that contains static methods, the class itself contains 2 methods initSquares() and initSquaresColor(). The initSquares method allows the creation of an array of 52 squares which represent the squares of the game board. The purpose of this function is to initialize all the squares of the board in order to use them to create the path of each piece. Thus the coloured squares of each piece will be missing in its Path, that's why we created the function initSquaresColor which is used to initialize the coloured squares of the pieces according to the colour that we send it.

### Team

The class Team is used to associate a player a color and the corresponding pawns, so every player possesses an array of 4 pawns corresponding to his team color.

### Scene builder and JavaFX



*We decided not to use the default libraries for the GUI, but rather to use JavaFX because the libraries are easier to use even if you have to download them and they are more recent and therefore have a more up-to-date style. We decided to use the Scene Builder application, which allowed us to set up the entire graphical interface more easily and to modify it as we wished. So we had to create .fxml files to set up the whole interface with Scene Builder, and then class controllers to interact with the user interface.*

## Global operation of the code

### The GUI (main)

The GUI class is the most important one in our project, it is a class that does everything and not much at the same time. As explained we decided to work with JavaFX which is easier to use and much more qualitative, so we used the SeneBuilder application which will create the .fxml file. So we have the GUI class which will retrieve the fxml file of the menu, will prepare the window with the name, the icon, etc... and then display it to the user.

### The Controllers

As the .fxml file must be connected to a class called controller which will allow it to interact with the window, this controller will declare all the controls of which we have declared their ID.

Thus the class MenuController is the controller of the menu, the class will thus include all the text zones to enter the nickname as well as the method of the button to launch the game which functions more or less like what there is in the class GUI. The particularity of this Class is that it must communicate with the controller of the game to transmit the nickname of the players to him and for this it is necessary to recover and launch a method of the controller of the game in the controller of the Menu.

One thus has a second controller GameBoardController, which makes it possible to launch all the operation of the game. We had previously built the whole project in console and then made the graphical interface, which caused us a lot of problems during the assembly of the project because we had to readapt all our code in console for the display because a large part had become obsolete. Moreover, we can find in the sources a .text file containing our main.java test for the console game.

So the game controller has 3 big methods that set up the whole game. First we have initialize() which will set up all the players, determine which player will start the game, but also save the basic position of the pieces.

*Secondly we have the throwDice() method which will throw the dice and return a value between 1 and 6. But this function will also take into account who has to play and thus activate the counters that are playable, if none of the player's counters are playable the function makes him skip his turn.*

*And thirdly we have the MovePiece() method which will apply to each selected piece, it will move the selected piece, then it will check if there are pieces to send back to the start in the arrival square. Then it will check if the player has just won or not and will deactivate all the pieces of the player who has just played. It also checks if the player is allowed to replay under certain conditions and stops the game if three players have won.*

# Difficulties encountered

We encountered quite a lot of difficulties, it was the first time we did a project using Oriented object programming, so even with the lecture that we had, the architecture of the project was not obvious to us, and even if the project was pretty simple in theory, we really struggle to put everything together and make it work… We also struggled with the graphical interface, we tried to use Swing and AWT to create the graphical interface, but with the given time we prefered to set up javaFX which was more practical to use.

*We had a lot of trouble implementing the GUI, as we had already designed the project in console and had to adapt it to our GUI in JavaFX. We realised that we should have started by making the GUI and then implementing all the methods related to the project*

# Conclusion

During this project, we have learnt a lot about java and oriented object programming,and after numerous difficulties we managed to develop a functional Ludo board game with a pretty good graphical interface which can be played with four players and will respect the rules. The game could obviously be improved in many ways, we probably didn’t use the full potential of oriented object programming here, but overall, I think we can consider this project a success.