



# LP2A - SKYJO UTBM

12.05.2023

---

THOMAS Yann

ZEDEK Jeremy

## Summary :

<b>Summary :</b>	<b>1</b>
<b>Introduction :</b>	<b>2</b>
<b>Objectives :</b>	<b>2</b>
<b>Key points :</b>	<b>3</b>
I. Conception of the software	3
II. Implementation choice	5
<b>Conclusion :</b>	<b>9</b>

## Introduction :

Skyjo is a popular game that is easy to learn but difficult to master. The objective of the game is to get the lowest score possible by collecting and discarding cards strategically. In this project, we will be implementing a digital version of Skyjo using Java programming language. Our software will allow players to play the game on their computers with a user-friendly way to play on the console. The implementation will involve designing and coding the game logic, user interface and game mechanics. This report will provide an overview of the software conception and implementation choices made for this project.

## Objectives :

The aim of this project is to develop a Skyjo game adapted to the UTBM world. To achieve this, we have assigned the name of a UTBM professor to each card value of the base game, in a purely subjective ranking.

One of our goals is to produce a code that is both readable and as close to reality as possible in order to best replicate the experience of playing Skyjo.

## Key points :

### I. Conception of the software

#### - The different states of the game :

The first step of the program is the main menu, which has three options available : start a game, display game rules, or quit the program. By choosing to start a game, you can choose the number of players that will constitute your game, from two to eight. After choosing, each player has to reveal two of their cards to determine who will start first, the player with the highest score goes first. After all of this, round 1 can begin.

How a turn works ;

The player has two choices, either draw a card or choose to take the card on the discard pile. If they choose to draw, they have the option to discard the card if they don't like it or keep it. Once a card is taken, they must decide which card from their hand they want to exchange it with. Once this choice is made, they discard the replaced card and it's the next player's turn.

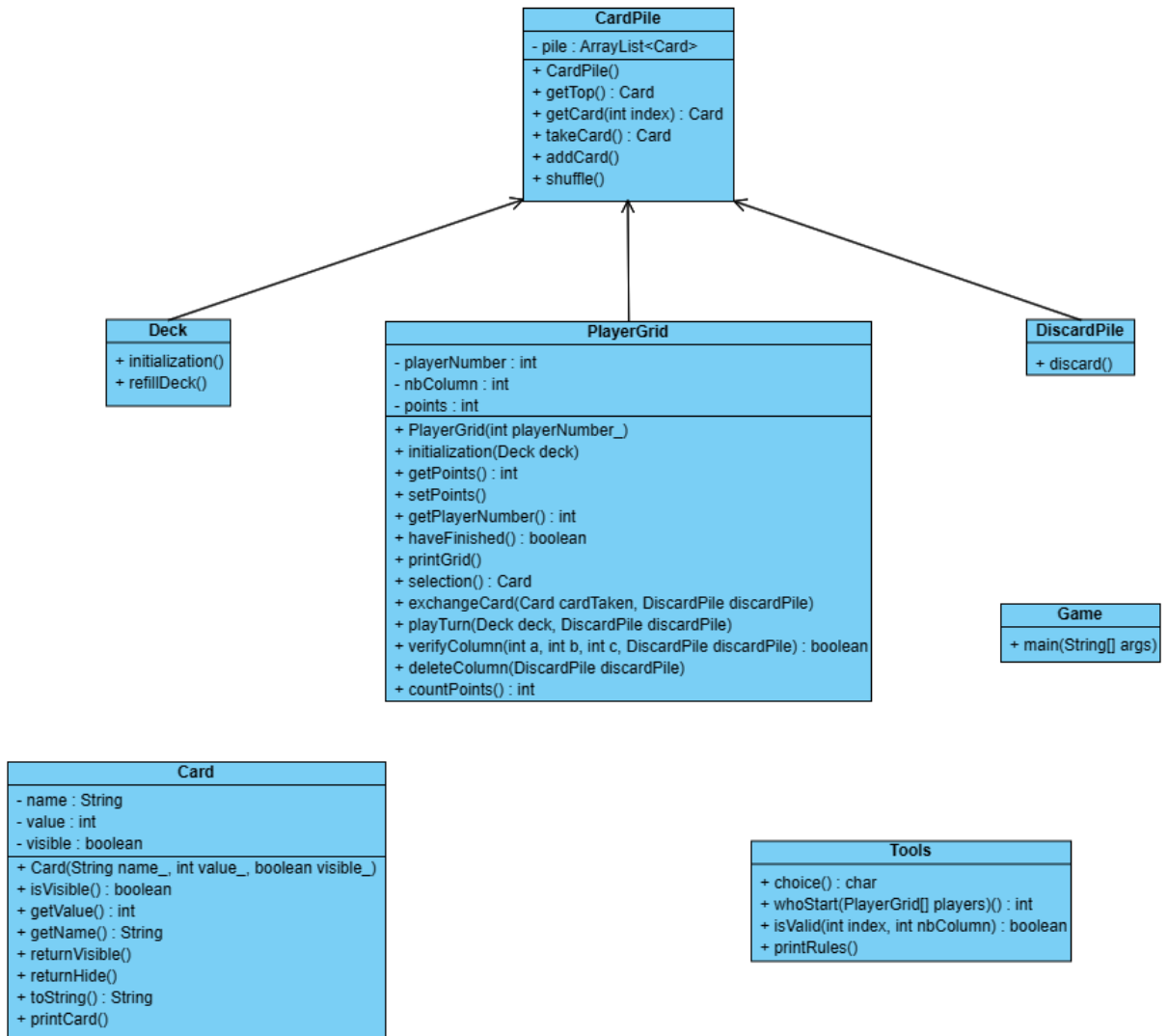
A special mention to the rule that allows a player to discard a column of their hand. This rule is well known and we haven't missed it. It consists of being able to remove a column of your hand by successfully aligning 3 cards with the same value.

This loop repeats until a player reveals all the cards in their hand. Once that is done, the other players have one final turn before the round ends. Once the round is over, it's time to count the points earned.

Points are earned by adding up the values of the cards in a player's hand. If the player who ended the round does not get the lowest score, their points are doubled for that round. After the points are tallied and added to the player's scores, the next round can begin. This loop continues until a player has over 100 points. Once the game is over, the player with the lowest points

wins. At the very end, it's possible to start a new game which would start the program from scratch.

## - The classes and methods :



## II. Implementation choice

### **-Used Libraries :**

During this project, we used a library to implement the desired functions in Java. We specifically used the `java.util` library, but only imported the parts that we needed.

Firstly we imported the "Scanner" which is used to read user inputs from the console. We used it so that the user can choose which card to play or which action to perform during the program's execution.


Additionally, we imported the "ArrayList" tool to store and manipulate a dynamic collection of objects efficiently using native functions like `add()`, `remove()`, `size()`, `clear()`, and `get()`. We found these functions useful during the project's development for our card pile, which is an ArrayList of cards.

Finally, we used the "File" tool to represent and manipulate file paths and directories in Java. We used it to retrieve and read the game rules file that is located in our project folder, where the program looks first. We simply specified the file name and extension to the program, which found it directly.

### **-Data Structure:**

To best implement our project, we chose to use different data structures. First, we wanted our project to be as close to reality as possible, so we created the `CardPile` class which represents our various card piles, including the draw pile, discard pile, and player's grid (which is a pile of cards, spread when displayed on the console).

A card pile is made up of a list of cards, which are objects created in the `Card` class and include a name, a value, and a boolean to indicate if it's visible or not. We chose to use an ArrayList for this list as we found it more convenient. When a game is launched, the ArrayList is filled with 150 card objects which are distributed among the various piles to be used during the game. We do



not generate cards during the game or manipulate abstract objects floating around.

This approach is more realistic, as when we want to draw a card, we take the first card from the "Draw" pile which will be moved from the draw pile to either our grid or the discard pile. This was also easier to code since ArrayLists have native functions to manipulate them, but it's also due to the fact that we worked with concrete objects, which made it easier to track cards and identify issues.

Our program also includes a simpler data structure: an array. This array is used to store the different players currently in the game and to track which player is currently playing.

The class system and object-oriented programming offered by Java were also very useful and allowed us to code fairly simply and quickly without too much difficulty. Object-oriented programming proved to be more interesting for this project than a more classic programming language like C. It offered us numerous advantages, such as the principles of encapsulation, inheritance, polymorphism, and modularity offered by Java, which are mainly the four pillars of object-oriented programming.

### **-Design of the user interface :**

Regarding the user interface, we initially tried to create a GUI using Swing. However, considering the time it would take to learn how to use the necessary tools and the fact that we didn't cover it thoroughly in class, we decided to stick to a console interface, which we tried to make as clean and readable as possible.

To achieve this, our program launches with a menu where the user has three options: start a new game, display the rules, or quit the program. The user enters a letter to make their choice among those listed on the console, and if they do not enter a valid character, the program will indicate so with a message. Once the user enters "P" to play, the game begins.

We decided to clearly separate the players' grids, turns, and rounds. A player's grid is delimited by "-", the start of a turn is marked by a line of "\*",

To aid comprehension, we chose to skip a line between each sentence displayed during the game and multiple lines when a new turn or round begins to mark the separation.

```

-----Player 1 grid-----

*****|*****|*****|*****|
*****|*****|*****|*****|
*****|*****|*****|*****|
*****|*****|*****|*****|
*****|*****|*****|*****|

-----

```

```

-----Player 1 grid-----

*****|Lapostolle (12)|*****|*****|
*****|*****|*****|*****|
*****|*****|Gaud(1)|*****|
*****|*****|*****|*****|

-----

```

(2)


```

/***** PLAYER 2 TURN *****/
-----** Player 2 grid **-----
|1          ||Lannuzel (9)  ||3          ||4          |
|5          ||6          ||7          ||8          |
|9          ||10         ||Turbergue (11) ||12         |
-----

```

(3)





Furthermore, we decided to display the position index of the cards that are face down instead of a series of "\*" to help the user understand their game grid. Once the card is turned over, the index will no longer be displayed, but the player will have understood the positioning system and can continue playing without difficulty.

### **-Tests :**

Throughout the development of the game, we tested our functions to ensure they worked properly. We would code a few functions that worked together and then launch the program with a test version of the main function to test them. If we noticed an error or if the program crashed, we would go back and check our functions, correcting any errors we found until everything worked, and then move on to coding new parts of the program.

Once we finished coding all our functions and the main function, we conducted some test games with people outside the project. This allowed us to verify the readability of our console and also identify bugs where we hadn't thought to look. For example, we realized that the deck was not infinite and needed to be refilled once empty. We also detected a bug that caused the program to crash when trying to flip a card with an index from a previously deleted column, which we fixed.

## Conclusion :

This project has been really interesting and enjoyable to work on because it allowed us to apply the Java classes we had learned during the semester and to learn new things that we hadn't had the opportunity to see in class. It also accustomed us to object-oriented programming, which we have come to appreciate and prefer over classic programming. Although we stuck to the bare minimum and did not code any AI or implement a graphical interface, we tried to produce a complete project, without bugs and clean both in terms of the console and the code itself with its comments.