* Work process

To develop this software project, a Waterfall Model was followed as it is explained next.

* 1. Analysis (Knowing the game):

To know the game and have an idea about how to develop the project, two actins were taken:

\* First the physical cards have been checked.

\* Then an app of the game was player.

* 1. Designing the UML:

The UML designing process had 3 versions which are described next:

\*First one with the classes included and the main methods.

\*Second one with less classes to avoid redundancy but more methods.

\*Before designing the last version, it was necessary to try some methods in code to know what would be the practical option, and after that the final version was defined (This is an exception of the engineering software model used).

* 1. Implementation

For coding, the main idea was to follow the design, so, it was started by creating the classes with the main variables and methods and some basic functionality to test the design.

After making some changes in the UML design the main functionality of the implementation was defined and coded, adding some interesting features to make more personal the game (which will be explained later).

Finally, some extra functionalities were added.

In this stage the Waterwall model was combined with a Kanban model in order to apport ideas and work to the team and moreover to work with the testing stage continuously.

* 1. Testing

As was mentioned above, the testing part came together with the implementation work, in order to define what was need to solve, but at the same time it was not a impediment to continue working in another features of the code.

A final test process was followed in separate for each member of the group in order to improve the performance of the code.

* 1. Documentation and Deployment

Finally, the documentation stage was set up, creating the required documentation.

* Teamwork

The team work in a collaborative way, starting early the designing and code stages in order to be able to present some extra features and to provide a personalized work.

Every stage has the participation of each member of the team.

The test process was taken separately in order to increase the scope of the detection errors stage.

* Design choices

After analysing the UML design, the team decided to finally maintain 9 classes in the code, the class with major functionalities will be game and the rest of classes will support the minor processes such as; updating lists, displaying features and so on.

Classes

* + - Main: Only to make run the program.
    - UserInterface: To have control of the interaction with the players by scanner method.
    - Game method: This class is defined as the more active one, were most of the principle functionality has been placed.
    - Player: This class is the second one in term of managing functionalities for the project.

For cards, the program it is managing inheritance, where the father class is Card and the child classes are CardKeeper, CardRule, CardGoal.

* + - Card: This class has the basic attributes and method to inherit to its child classes.
    - CardKeeper: Are the basic ones to define the keepers of the game.
    - CardRule: The main difference with the other card classes is related to the fact that this class has the attribute which because is going to take different names of cards as rules and limit to determine the limit cards in the game, for example:

Card Rule🡪 Limit keepers (2,3,4).

Card Rule🡪 Limit play (0 which means play all,2,3,4).

Card Rule🡪 Limit hand (0,1,2).

Card Rule🡪 Limit draw (2,3,4,5).

* + - CardGoal: (15 combinations of pair keepers) This class is special because is composed for 2 card keepers, this decision was taken on the bases of providing scalability possibilities to the project, on the top of that, the combination of the Card Keepers is made randomly to add some interesting, personal and not predictive features to the game, therefor if in the future it would necessary to have more or less than 15 Card Goals, it would be a matter of only changing a integer variable.

There is not going to have the same combination of pairs per game.

* + - RuleArea: This class will have the List of Rule Cards that are in play. It works with the class CardRule, therefor it is just allowed to have in the rule area a max of 4 cards for each type of Card rule (example: It i not possible to have 2 limit draw cards because there is a contradiction).
* Special elements in your code you want to address and last but not least a small manual for a user to play your game (you can assume this user has played a normal game of Fluxx before)

Steps to play FLUXX

* 1. **Choose the number of players 🡪** between 0 and 6.
  2. **Write down the nick name of each player**🡪 It is not allowed to have repetitive nicknames for a single game.
  3. **Start playing.**

**Per each turn** (automatically in the order that the players register the nicknames).

The player will be able to type:

3.1. **help:** to display all input options.

3.2. **done or anything else:** to continue playing.

If the player type **help** the displayed options are:

\* Type '1' to display all the keepers on the playing field.

\* Type '2' to display the current goal.

\* Type '3' to display all current rules.

\* Type 'help' to display all input options.

\* Type 'done' to continue with your turn.

The first time by typing 1,2 or 3 the program will show that there is not keepers, goal or rules in game yet, but, after the first turn the program will display the corresponding features.

If the player type done or anything else, the hand cards for that specific player will be displayed, for example:

0: Goal Bread + The Rocket

1: Goal 5 Keepers

2: Goal Television + Milk

3: Keeper The Moon

Choose a card

By choosing a card the play continue with its normal dynamic.