The Computer Science degree started with an Introduction to Programming (Professor: Jamila Sam), where we learned the basics of programming in JAVA. During the Spring Semester 2019, I attended a mandatory advanced course of Practice to Object-Oriented Programming course (Professor: Michel Schinz). One assignment of that course was a two months project in JAVA, in groups of two, where we would implement a Swiss card game named Jass.
Most of the project was guided by weekly tasks and intermediate tests. The last two weeks where Bonus Part, free of any guidance to let us improve the game and make it as realistic as possible.

I ended obtaining a course grade of 96%, best grade of the class (200 students) for the course and the project.
The Bonus Part could increase our final grade (although not significantly) depending on their ranking :
- **insufficient**
- **enough**
- **good**
- **very good**
- **excellent**

Our project was one of only three to be graded excellent, between more or less 70 propositions.

Of course, such a complex project is probably difficult to fully comprehend at first sight, but I think this project is an excellent way to demonstrate the extent of work and dedication put through my academic studies.

Below is a description for launching the program, as well as a brief explanation of all the different classes in the program.

To launch the program, run the LocalMain class as follow :

The program accepts 4 or 5 arguments. The first 4 specify the players, and the last, optional, specifies the seed to be used to generate the seeds of the different random generators of the program.

Each player is specified by means of a string of characters composed of one to three components, separated from each other by a two-point (:).
The first component consists of only one letter, which can be :
- either h for a human player (local)
- either s for a simulated player (local)
- either r for a remote player.

The second component, optional, is the player's name. By default, the following names are assigned to players, in order : Aline, Bastien, Colette and David.

The third component, also optional, depends on the type of player :
for a simulated player, it gives the number of iterations of the MCTS algorithm (10 000 by default), it must be greater than or equal to 10.
for a remote player, it gives the name or IP address of the host on which the player's server is running (default localhost).

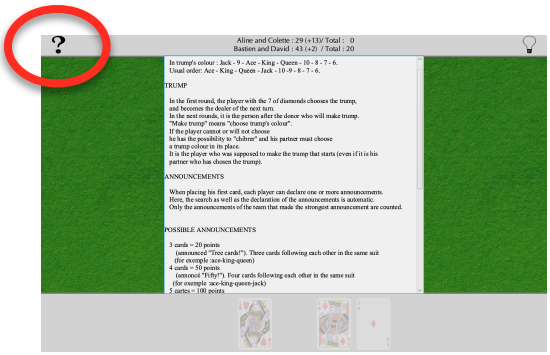
For example, the following arguments : **s h:Marie r:Céline:128.178.243.14 s::20000** specify a game in which the following players participate :
- a simulated player named Aline, with 10,000 iterations,

- a human player named Marie,
- a remote player named Celine whose server is running on the compute whose IP address is 128.178.243.14,
- a simulated player named David, with 20,000 iterations.

If one of the arguments has been entered incorrectly, the console will output this same message, telling where the problem is situated.

To launch the program as a remote player, one need to give its IP address to the local player, and simply launch the RemotePlayer class without any arguments needed. After that, the game will start as soon as the local player launches its program.

Once the game is launched, you can have a look at the rules of the game by clicking on the upper left image (interrogation point).



Here is a very brief explanation of each class :

- **Card, CardSet, Score & Trick :** Classes that represent each of the features in a Jass Game. They contain useful method to act on themselves. (eg : CardSet has a method add(Card card), which adds a given card to the set.

- **PackedCard, PackedCardSet, PackedScore & PackedTrick :** Each feature can be represented by its packed_representation, which is an Integer or Long value. This is a way to make te methods easier. Each method of a feature class (eg. CardSet) can call the method of its packed_representation (eg. PackedCardSet), where all computations will be much easier.

- **Bits32 & Bits64 :** As mentioned, cards, scores, tricks and cardSets will be represented as Integer or Long values. Bits32 & Bits64 are useful classes to act on these values.

- **Jass :** Basic Interface which contains the useful constants of the game

- **PlayerId & TeamId :** Small enumerations representing the different players, their team & team mate.

- **Preconditions :** Basic class with two useful preconditions for the rest of the program.

- **Player :** Interface representing a player, with methods that will define its possible actions.

- **MCTSPlayer, GraphicalPlayer, PacedPlayer :** More complex methods. MCTSPlayer represents a simulated player, who will decide which moves to play thanks to MCTS algorithms. To search for best card to play, he will compute a tree of possible moves to make, ending the turn and computing the possible winning points. He will therefore choose the card which technically could give his team the most points. GraphicalPlayer represents the local player, so it implements the interactions with the screen, and how to show everything going on in the game on the screen. PacedPlayer is simpler and only ensures that a player takes a minimum time to play (otherwise each simulated player would play too fast).

- **TurnState :** Represents the state of a turn. Its attributes are the score, the unplayed cards and the actual trick. This class is used to manipulate a turn's progression.

- **JassGame :** This class implements a complete process of a Jass game. It is the main class that is called by LocalMain to start a game.

- **GraphicalPlayerAdapter, HandBean, ScoreBean, TrickBean :** These classes are used to implement the interactions between the current game in the program and what the local player sees and does on the screen.

- **RemotePlayerClient, RemotePlayerServer, JassCommand, StringSerializer :** When some players play online, they will need to communicate their choices and their moves. This is done thanks to the client and server classes, which communicate over a given port. The JassCommand enumeration is used to give the list of all possible messages types. To be sure to send correct messages, i.e encodable and decodable messages, we use the StringSerializer class.

- **LocalMain, RemoteMain :** Finally, LocalMain and RemoteMain are used to launch the game, as the host local player or as the remote player. LocalMain takes the given arguments into account to launch the wanted game.

All classes and methods in the game are fully and carefully explained through Javadoc.

We used the Bonus Part to add many interesting, creative and fun features to our game. To better understand all these features, here is the original report we had to present to the professor in June.
It's a simple report, but it gives the main ideas, implementation decisions and challenges we faced making this bonus components.
All these new features are strictly our own making. For the most part, our goal was to find how to implement Jass complex rules in our program, as the main program would originally only implement the basic ones. We had no help whatsoever for this bonus part.

BONUS PART :
Choice of the trump:

At the beginning of each turn, the player who starts the turn must first choose the trump of the turn.

Conceptually, this is simply implemented in JassGame, where we already have the first player of the turn assigned to it. Instead of randomly choosing an asset from the setCurrentTrump method, a new abstract method is called chooseTrump for the player in question. A MCTSPlayer chooses the asset using a fairly simple algorithm. For the GraphicalPlayerAdapter, it is the same principle as for cardToPlay: we have created a new ArrayBlockingQueue<Color> that waits to receive a color and returns it. To do this, in GraphicalPlayer, we have a new panel (on the right), which is displayed when it is up to the player to choose, thanks to a Boolean property defined in TrickBean, isTheChooser, which is true only when it is up to the player to choose. The 4 colours are displayed, and as soon as the player clicks on one of them, it is sent to the queue. Of course, since in JassGame the choice of the trump is made after the cards are dealt, the player can see which cards are his, in order to choose the most advantageous trump.

Choosing the trump, or choosing to “chibrer”

The player can "chibrer", and ask his partner to choose for him.

The idea is to add a "chibrer" button to the panel. In JassGame, setCurrentTrump is modified as follows: the currentFirstPlayerOfTheTurn is asked if it wants to chibrer, using a new choseToChibrer method in Player. If the answer is false, chooseTrump from currentFirstPlayerOfTheTurn is called. Otherwise, we call chooseTrump from his partner. choseToChibrer is by default, because a simulated player always returns false. In GraphicalPlayerAdapter, choseToChibrer looks like chooseTrump: an ArrayBlockingQueue<Boolean> that simply waits for an answer. We still create a Boolean property in TrickBean, isAskedToChoose. So, when isTheChooser or isAskedToChoose is true, the colours are displayed. But the button is only visible when isTheChooser is true! If you press the button, you send true to ArrayBlockingQueue<Boolean>. By pressing a colour, if isChooser is true, then we

send false in ArrayBlockingQueue<Boolean> and the colour in question in ArrayBlockingQueue<Color>. Otherwise we just send the colour to ArrayBlockingQueue<Color>.

For communication with a remote player, everything is done the same way as other Player methods.

Make announcements:

At the beginning of each round, the player can make announcements at the time of placing his first card.

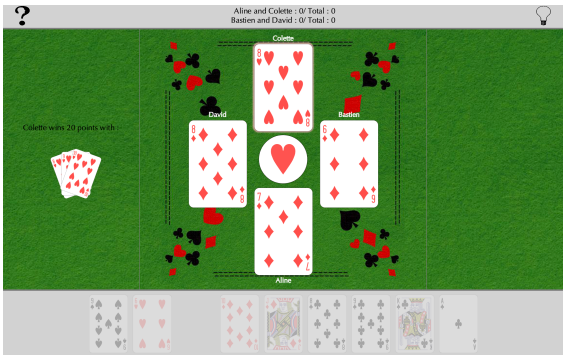
To do this, we have created the Meld classes, which corresponds to a standard announcement (defined by the cards that compose it and the number of points it earns), and MeldSet, which represents a set of separate announcements (which can be announced). These two classes implement the Comparable interface so that the best ad can be selected (only the best Meld of each MeldSet of each player is compared, as in the Jass). The Melds of each MeldSet are stored in a TreeSet in order to preserve the notion of order that is useful when we want to establish a communication about a MeldSet between the client and the server. Indeed we have added in the Player class the methods setWinningPlayerOfMelds and selectMeldSet. The latter taking a MeldSet as argument we had to establish a new representation to send the information to the server:

"pointsMeld1:cardsetMeld1,pointMelds2:cardsetMeld2,... »

(Note that the MeldSet can be empty, this representation can also be empty)

In a similar way to the choice of trumps we have added in GraphicalPlayerAdapter an ArrayBloquingQueue<MeldSet> to communicate with the GraphicalPlayer and properties in ScoreBean to indicate if the player can: make an announcement, know the winning MeldSet, and know who has won.

Making an announcement



Being told who won

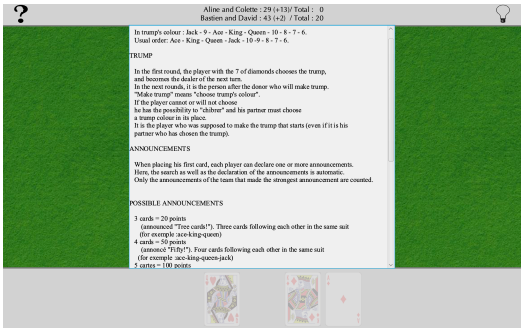
Small original creations:

Graphically, the announcements are displayed on the left of the trickPane, in a rather aesthetic way: cards presented in fan, whose size adapts to the quantity of possible melds, not to exceed on the screen. We implemented a simple feature that make a card come forward if we pass on it with our mouse.

We have also added a hintPlayer argument of the MCTSPlayer type to the GraphicalPlayerAdapter so that if the user exceeds a predefined maximum time to choose the card, announcement or asset, the element is automatically selected by the hintPlayer ! (Which can also be quite nice for testing, one can only decrease the waiting times and doesn't have to play a whole game).

We also added two icons in the scorePane: the first one representing a question mark shows a scrolling panel containing all the rules, the other one representing a bulb shows a green halo around the card that a MCTSPlayer would play (a kind of help/advice for beginners).

Asking for a hint (hint image) and passing the mouse on the card



Clicking on the rule image

These two features are enabled/disabled by clicking on the images. Be careful with the rules, the game doesn't stop.

image sources :

rules_cranium and idea_bulb from pixabay.com

card_table from Canevas.com