CartomENS-IA Project
Final Report

cartomensia@ens-lyon.fr

https://graal.ens-lyon.fr/cartomensia

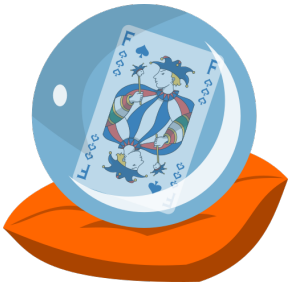
https://www.facebook.com/cartomensia

https://twitter.com/CartomENS_IA

Team members:

Béthune Louis
Blondeau-Patissier Lison
Coiffier Guillaume
Deschamps Quentin
Guépin Florent
Léchine Ulysse
Pouget Stéphane
Vacus Robin
Valque Léo
Benoit Tristan
Champseix Nicolas
Devevey Julien
Fernandez Simon
La Xuan Hoang
Pitois François
Rochette Denis
Valentin Herménégilde
Vavrille Mathieu

ENS DE LYON



LA BÂTARDE .com

Contents

List of Figures

Part 1

General Project Description

1.1 La Bâtarde

La Bâtarde is a trick taking game[1] created by Jules Marconnier. This is a mix between many other cards game like "La Belotte", "L'ascenseur", "Le wist"...

Moreover this game contains parts that are an auction game.

The main principle of this game is to evaluate the strength of our cards accurately, and play in accordance with it. So despite the randomness that is inherent to every card game, luck is not so important, and the game is deeply strategic.

More information can be found on the official website : http://www.labatarde.com.

Rules of the game

La Bâtarde is played using a special deck of 36 cards. There are four suits, Spades, Hearts, Diamonds and Clubs, as in usual card games, but the ranking of the card (from strongest to weakest) is the following:

- First we have the king's court: King, Queen, Knight and Jack
- Then come the artists: Fool, Musician and Juggler
- Finally the animals: Dog and Cat.

At the beginning of a round, a suit is chosen to be the trump[2]. The order of the cards of this suit is then modified: artists become stronger than the king's court. We can also play with No-Trump (no suit is chosen to be the trump) or All-Trump (all suits are trumps).

The trump is chosen during a bet phase. Each player estimates from his or her hand how many tricks can be done in the current round, and which trump would be best. They then make bets according to some specific rules we will not detail here. Two mecanisms will be relevant for us, the méchoune and the choune. When a player makes an announcement in a given suit, another player can decide to méchoune (in a real life game: saying "Méchoune!") to keep this trump suit (players can still bet on how many tricks they will get). This will double the points for everyone. The player who made the méchouned announcement can then decide to choune to double points again.

After the trump suit has been selected and each player has made a bet about the number of tricks they would take during the current round, the game phase can start. Players play cards in turn and gain tricks when they put the strongest card.

The points are the difference between the number of tricks announced and the number of tricks actually done; the goal is to get the lesser score.

Detailed rules of the game can be found on the official website of *La Bâtarde* [3]

[1] A card game in which play of a hand centers on a series of units of play, called tricks, which are each evaluated to determine a taker of that trick.

[2] The trump suit is the suit that is stronger than all other.

[3] http://www.labatarde.com/les-rgles-du-jeux

1.2 Goals and expected results

1.2.1 Main goals

The goal of the CartomENS-IA project is to create a software that allows players to play to *La Bâtarde* online, against other humain players, or offline against artificial intelligences (AI).

Constraints:

- The player must be able to play online or offline, on a computer or a phone (running on Android).
- If a player disconnects from an online game, he or she should be replaced by an AI.
- The offline AI for mobile devices should have a low energy consumption.
- The AI must have different levels of difficulty.
- In both platforms, the user interface should be ergonomic. The original game has a strong visual identity, so we will try to do justice to it.
- Tutorials should be available to new players in order to make them discover the rules of the game and to provide them a small training.

Artificial Intelligence:

Our work will contain an important part of research to develop an AI for this game, because as of today no such AI exists.
We wish to apply recent innovations as Monte Carlo Tree Search or Neural Network to design a powerful AI, and eventually find an optimal strategy. We will also try classical methods such as deterministic heuristics, which are lighter in terms of computing power.
The AI will have several levels of difficulty, from sandbox to very hard. Moreover, we will design a light version of the AI that can run on mobile devices with low energy consumption.

1.2.2 Related Work

There are other projects from the ENS de Lyon on the same theme, as Cartomancer[4] or Macaks[5]. Cartomancer is an open-source project, to develop a card-game edition program. Macak is a simulator for the game *Magic: the gathering*.

Software for card games exist since decades. There is a lot of examples: poker, spider solitaire... we can find them as computer programs, web applications or phone applications.

We studied several other resources, mainly for the AI part, where we will test neural networks, Monte-Carlo method, and expert systems. These resources are detailed in the related work package.

We can also draw inspiration from the AIs of "Belote"[6].

Finally, we studied some design patterns to create the architecture of the GUI and the core that rules the game.

1.2.3 Scientific and Market Spin-offs

Depending on the success of the research and development of the AI, it could be used as a foundation to tackle other card games or games in general.

Furthermore, this software will be of interest to Jules Marconnier. Indeed, it allows him to market his game and facilitate access for new players. We can also imagine the software being taken over after the end of the project to include other modules such as online tournaments, challenges against the AI, etc.

The targeted users are average people with no specific competence in computer science, who want to play or learn to play *La Bâtarde* . The software should be easy to use, and not too heavy in running time or resources. The tutorials will be pedagogical enough for a beginner.

This project requires 25 dollars to get a Google play account and release the application to the app store. 100 euros have also been spent to buy 5 copies of the deck of cards.

[4] https://jumplyn.com/#!/project/594c313f2661be5c0cd709f3/details
[5] https://jumplyn.com/#!/project/594c2ded0870203fd4f8ddca/details
[6] https://www.belote.com

1.3 Team

We are master students in computer science, currently studying at the École Normale Supérieure de Lyon, in France. The list of members can be found in appendix A.

1.4 Work Schedule

1.4.1 General Schedule

First part of the project

The following figure is a modification of the original schedule we proposed at the beginning of the project. It depicts roughly how the work was organized during the whole project.

Week	1	2	3	4	5	6	7	First report
Communication	Website							Website done
Core	Modules							V1
GUI		Ergonomics	Low Level		Basics			V1
IA								Playable Version
Mobile device client	Apprenticeship							
Network / Server								
Tutorial								

Figure 1.1: Schedule of the project from start to the first report

Week	8	9	10	11	12	13	14	15	16	17	18	19	20
Communication	Screenshots			Report		Break		Report	Communication Interludes				
Core	Round number Méchoune		Game initialisation			Break							
GUI	Clean up		Sounds Scores Menu			Break		Méchoune					
IA	Monte-Carlo Bot deter					Break		V1	Debug Improvements				
Mobile device				V1		Break							
Tutorials	Theory				V1	Break							
Network	Implementation				V1	Break		Méchoune			V2		

Figure 1.2: Schedule from first report to end of project

1.4.2 Interludes

The **Interludes** is an event organized by the **ENS de Lyon** that gather passionates about board and role-play games from the four French ENS. It happened at week 20 in the schedule (9–11 February). For this occasion, a tournament of *La Bâtarde* was organized, and we presented a beta version of our software. We let people play and collected feedbacks, which were then used to improve the software.

It was nice to see that some improvements we had already thought of, such as putting non-playable cards in grey, were suggested by users too: we could check that our ideas were indeed correct and that it would profit users. Several members of the team participated in the Tournament, which was ultimately won by our project leader, Louis Bethune. You can find picture of the event in Figures B.1, B.2 and B.3, in appendix.

We could also let our AI play against real players in real-life games, both during the Interludes and during ordinary sessions of the board-game club of the ENSL, where Jules frequently comes to organize sessions of *La Bâtarde* . This allowed us to test the IA even before the online version of the game was completed. You can see a picture of the setting in Figure B.4.

Part 2

Presentation of the Work Packages

We divided the work in several Work Packages (WP):

- **WP Communication: Leader:** B. Lison;
  Members: B. Louis, C. GUIllaume, F. Simon, L. Ulysse, P. François, P. Stéphane.

  The main goal of this work package is to present and document our work, mainly with our website[1]. We also encouraged each work-package to keep tracks of the general progression by taking detailed notes at every work package meeting. Finally, we were in charge of the external communication.

- **WP Core: Leader:** R. Denis;
  Members: B. Tristan, B. Louis, C. GUIllaume, D. Quentin, F. Simon, L. Hoang, V. Mathieu.
  This work package is in charge od the core architecture of the CartomENS-IA project, it is the central platform. As a main objective, it has the game engine, with all the rules of the game "La bâtarde". Each platform — mobile and PC — can have its own core.

- **WP Artificial Intelligence: Leader:** D. Quentin;
  Members: B. Tristan, B. Louis, B. Lison, C. Nicolas, L. Ulysse, V. Robin, V. Herménégilde, V. Léo.
  The main objective of this work package is to create an Artificial Intelligence capable of playing *La Bâtarde* . We will explore different possibilities given by game theory: Monte-Carlo trees, min-max methods... The AI should be efficient on both parts of the game: the evaluation of the cards to make auctions and the game phase. The others objectives are to create different levels of difficulties and to design an AI capable of replacing a player who left in the middle of a game.

- **WP GUI: Leader:** C. GUIllaume,
  Members: B. Louis, C. Nicolas, D. Julien, P. François, R. Denis, V. Léo, V. Mathieu.
  The GUI package aims at developing a program in order to play *La Bâtarde* with a computer. We decided to use the C++ programming language, and the SFML graphic library. Sprites and graphical resources were provided by J. Marconnier, creator of the game. Therefore, the visual identity of our client is close to the one of the physical game. Previews of the program can be found at the appendix of this document (section C).

- **WP Mobile Device Client: Leader:** B. Tristan;
  Members: B. Lison, D. Julien, G. Florent, L. Hoang, P. Stéphane, V. Robin.
  This work package aims to develop an application to play the game with a mobile device, with the same expectations as the computer client. The application will target many different resolutions and versions on the Android platform. The limited resources of a smartphone will be used wisely: firstly, the AI will fit the operational capacity of a smartphone; secondly,

[1] graal.ens-lyon.fr/cartomensia

the network communication will be smart enough to avoid wasting too much energy; and finally, the images will be adapted to the resolution so that no memory is wasted on loading over-detailed images.

- **WP Tutorial: Leader:** V. Mathieu;
  avoid being overtaken. **Members:** B. Lison, L. Hoang, P. François.
  The tutorial package aims at creating an interactive tutorial for new players. The goal is to learn the rules of the game in an interactive way, by playing the game. The architecture created allows us to create easily a new tutorial with a simple language.

- **WP Network and server: Leader:** F. Simon;
  Members: R. Denis, V. Herménégilde.
  This work package aims to build a server and to set up clean communication protocols between the different applications. It will be used to connect players together, set up games. Also, the server will be there to replace a player if one leaves the game, in which case, an AI will play instead to keep the game running. The server will be responsible for the authentication and security of the communications. This work package will have to design communication protocols and standards that all applications will use to communicate. This work package needed fewer people thant the others.

Part 3

Final Status

3.1 General Impression

The full schedule established at the beginning of the year has been respected. All functionalities planned in proposal had been implemented. Moreover we got time to add more features, suggested by project members, other students, or Jules Marconnier. We never released an official V2 of our work because all versions after the V1 were playable, due to an incremental work. Today, we can play both offline or online, on a server currently hosted at ENS de Lyon. The connections are authenticated and secure : it is impossible to cheat. The AI plays better than a beginner (if often beat them), and plays well enough to add challenge to an expert (after many tests on experimented players). The GUI is a real success : it is beautiful and ergonomic (according to the new players that test the software during Interludes). There is animations, noises, and trophies. Finally, the mobile device client offers similar features, adapted to the particular constraints of a phone (screen size, memory limits).

3.2 WP0: Communication

On the first week, we created a bilingual (French and English) website[1] presenting our team and work. It explains the context and ambitions of CartomENS-IA, and is regularly updated. We let the proposal online for interested users, along with a summary. Captions of the game and news about our current state of work were also updated regularly. Finally, users can download the different versions on the website.

We also designed a logo for the application :

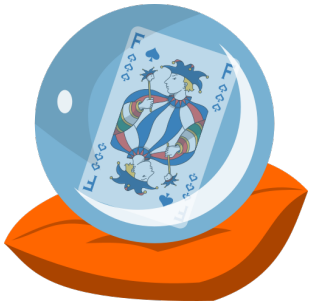


Figure 3.1: The CartomENS-IA logo

We maintained an overleaf file containing meeting reports of each of our work meetings. It allowed us to keep an historic of our decisions, research and issues; moreover it made easier for everyone to understand the work done by a team, and thus to help them. This has been really helpful for the redaction of this report.

[1] graal.ens-lyon.fr/cartomensia

For the Interludes, a week-end of games organized in Lyon between the ENS (9–11 February), we presented our application during the *La Bâtarde* tournament organized by Jules Marconnier. People could come and play against the AI, which allowed us to collect feedbacks in return.

We were also in charge of the public demo on March, 2nd, during which we made a short presentation of our work as well as a demo of the application.

All of these events were announced on our website.

Finally, we created a Facebook page[2] and a Twitter account[3], to inform users of every news from CartomENS-IA. Notably, we could announce the release of the last version of CartomENS-IA, available on the website, and of the mobile version, available on the Android Market.

The website will also be of use for playing online: users will need to log in to the website to create an account, in order to play online.

3.3 WP1: Core

This WP was the most important one at the beginning of the project, because choices made here will have for consequence the whole architecture of the project. So two meetings were necessary before a first functional skeleton was proposed, using part of a code previously made by some members of the WP. Then the team extended the skeleton to obtain a fully functional core. The goal was to provide a clearly understandable interface, with fully documented code, for the network, the AI, and the GUI.

The core contains utilities such as `Card`, `Bet` or `CardStack`, but it essentially consists in the `Kernel` class. This class rules the game, asks players for moves, and notify them when an action has been done. Moreover, it can be initialized in a particular state, or with a file, saving its own state in a file, and contains other informations such as statistics on the players. This is especially useful for the AI package, that could need to repeat the same situation several times in a row (for instance, in order to train an AI).

The `GameState` class is a view on the game that a player can access. Each player has its own `GameState`. It can only be modified by the Kernel (a notification is sent at each move validated by Kernel). The player can ask its ID, its hand, the current trump, the history of trumps, the current stack of cards, etc., according to the rules of the game. However, he cannot ask for the game of other players.

That way, we have a `GameState` that does not need full information on the game to work. It is a good feature because it means the remote server does not need to send the hands of each player to the local player. So, technically, it is impossible to intercept information on the network and acquire information: it is a protection against cheating.

We also made a mother class named `Player`, that is meant to be derived. This class provides methods as `play()` or `bet()`, which are called by the Kernel. When designing an AI, the AI team only needs to implement those methods, using the `GameState` view.

Moreover one of the derived class, named `ParallelPlayer`, allows the player to play asynchronously : the Kernel who calls the method blocks on a condition variable and wait for an answer. It allows the GUI or the network to not directly depend on the Kernel, and just listen to it on an infinite loop. These choices lead to serious issues when handling the Choune and the Méchoune, two rules of the game that are highly asynchronous.

We found a compromise to handle constraints of network, GUI, AI packages, some of which were opposite. It resulted in a complex architecture, and despite several meetings we were never completely satisfied with our choices. Instead of spending too much time finding the best architecture, we took a good one, and we implemented it quickly, so that other packages could begin to work (cf Figure 3.2).

In order to handle multi-player games over the network the Kernel class was split in a `Kernel` and a `RemoteKernel`. In a multi-player game, the Kernel is in the server side, where the `RemoteKernel`, in the client side, is the interface between the Kernel and the GUI. Using the exact same strategy, the Player class was split in a `Player` and a `RemotePlayer`.

The Méchoune is an action that a player can do at any time in "La bâtarde". It was rather difficult to manage it. The Kernel loop changed and become partially asynchronous.

[2] https://www.facebook.com/cartomensia/
[3] https://twitter.com/CartomENS_IA

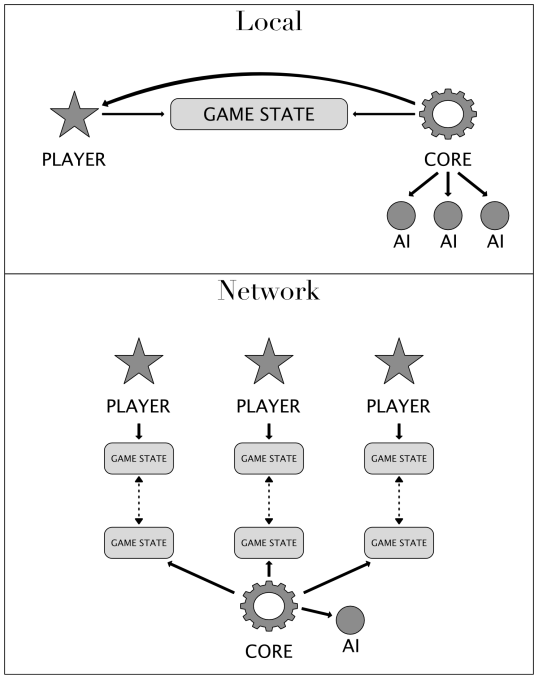


Figure 3.2: Core architecture

3.4 WP2: Artificial intelligence

As *La Bâtarde* has not already been studied, we cannot know what could be the best approach, so we decided to explore different kinds of artificial intelligences.

Since the start of the project, the work of the team has mostly been theoretical. We did a lot of research in various domains of AI, and we selected three promising branches on which we performed further investigation. We therefore subdivided the WP in three groups. Each of them should create an AI corresponding to its domain: deterministic bot, Monte-Carlo method or neural network method.

3.4.1 AI - Knowledge engine

To help different AIs, we built an engine which deduce information from previous tricks. For example, if a player trumps, we deduce he has no card of the suit being played in his hand anymore. This engine keeps this information on all players and AIs can get this information. It can also generate random hands which respect this information. It's easy to record the cards already played, or to detect if a player doesn't have a given color.

But drawn hands randomly according to these constraints is hard. We are suspicious that the generalization of this problem may be NP-complete. However on real-life situations there is no too much constraints, except at the end of a round when the number of remaining cards is low. Thank to this, a randomized backtrack is quick enough for an intensive use.

3.4.2 AI - Deterministic Bot

This sub-work package works on deterministic bots that are able to play to La Bâtarde using heuristic algorithms. We divided the work in two part: the bets and the game itself.

For the bets, we only have a pseudo code, which try to evaluate the hand of our boot. Two questions need to be answered: how many tricks can we expect to win and how easy it is to respect a given bet. For now, the possibility for the players to use méchoune and choune is not taken into account.

For the evaluation, we use some probabilities on the distribution to give a score to each card. The probabilities are computed for the different trumps and only depend on our hand, with some

approximation. The score, between zero and one, is an estimation of the probabilities for this card to make a trick. Then we add all the scores to have an estimation for the all hand.

We currently have implemented this estimation for one trump but not yet for all trump and no trumps, which are expected easier. For the one trump, some improvements could be done for some kind of hand - for example a hand with many cards in one color - where the evaluation is not satisfactory.

The evaluation of risk is not yet implemented. It is meant to compute a risk indicator among the bets. We will need, for a given bet, to give a score to each card according to your hand's ability to avoid playing this card or to avoid being overtaken.

As for the card-playing part, we now have an algorithm which slightly better than a beginner. We haven't yet implemented the bet part so the tests have been done with the bet of the Monte-Carlo AI. We hope to have better result with deterministic bets because of the increase of logic between the two phases of the game.

The algorithm is based on case-disjunction. First we compute some easy fact : essentially if the bot needs tricks, if it is the first player or not, if the other players want tricks and a more difficult to evaluate : are we going to make less or more tricks than expect. Depending on the result the bot plays instantaneously if the command is easy, for example "play the smallest trump". Otherwise for more complex commend as "play the smallest card with a good probabilities to take the trick" we have implemented the computations of the probabilities. This computation contains some approximations and parameters which could be change to modify the level of the bot.

At this stage, fixed with the Monte-Carlo the bot can be used for a beginner level. To make it independent the biggest work is on the bets. If many improvements could be done it's the only one which is really important, others ones will change the behavior but not add capacities to the bot.

3.4.3 AI - Monte-Carlo

We have searched information about Monte-Carlo method [4] for AI. According to the scientific articles we found, Monte-Carlo Tree-search [5] is not efficient on incomplete information. Flat Monte-Carlo is better on this problems.

So we have implemented a Flat Monte-Carlo, it knows how to bet and how to play. For the betting phase we do it like this : the Monte-Carlo chooses the suit of the trump, it deals possible hands to the other players (using the knowledge engine) and then it simulates N games (N=2000 for now) played by bots (random-playing bots for now) such random games are called "descent".

After it did all those games it can build a table of how many points it expects to gain (remember it is bad to gain points) depending on the bet (for instance if when the simulated trump was heart the player did 4 tricks 9 times out of 10 then in the table the bet "4 hearts" will have a low value, meaning that it is a good bet).

However we can do better, indeed when we deal a hand to the other players it would not be clever to deal them hands that do not match what they said, for instance if a player announces "8 spades" it probably means that he has a lot of spades in his hand, so dealing him a hand with no spades should be almost out of the question (he could still bluff so we still allow ourselves unlikely hands). To palliate to this problem when we do a "descent" we affect a coefficient of credibility to this descent, the farther the other players are from the bet they announced the less credible the descent should be.

After all this work the Monte-Carlo AI will bet what minimizes the standard deviation around his bet so the bet that minimize the average loss for the score.

```
1: for i = 1 to nbMaxDescents do
2:    deal random cards to the other players
3:    for each possible trump t do
4:       set current trump to t
5:       simulate a round where everyone plays randomly
6:       store the scores
7:    end for
8: end for
9: Return the bet b that minimizes average loss
```

Algorithm 1: Pseudocode for the Monte Carlo : Bet Phase

For the playing phase the Monte-Carlo algorithm does approximately the same thing except

that instead of doing descent depending on the suit of the trump it will do descents depending on the first card it can play. Then it will play the card that is most likely to make it do his bet. There is no further complications to this algorithm yet.

```
Record actions made by other players so far
Deduce the cards they cannot have in Knowledge Engine 𝒦
for i = 1 to nbMaxDescents do
    for each card c do
        deal random cards to the other players according to 𝒦
        play the card c
        simulate a round where everyone plays randomly
        store the scores
    end for
end for
Return the card c that minimizes average loss
```

Algorithm 2: Pseudocode for the Monte Carlo : Card Phase

A pragmatic problem to solve was to make the descents faster so we could do more of them and thus be more precise, in order to do this a "fast" version of the kernel (which is needed to check the rules) has to be implemented, work has been done to achieve this but to this day the "fast" kernel still has some bugs.

Moreover we see that everyone plays randomly. It's quick to simulate, but not very realistic. So we can expect to improve the results by replacing these random players by clever ones. Actually two ways are explored : using the deterministic bot (previous section) or a neural network (next section).

Currently, the AI you can play against in CartomENS-IA is an implementation of this Monte-Carlo algorithm. The number of descents is tuned according to the wanted difficulty : 50 for easy, 400 for medium and 2000 for hard.

3.4.4 AI - Neural Network

To refresh the reader's memory I remind him/her of the general approach of neural networks.

La Bâtarde is giving us two challenges : make a coherent bet at the beginning of the game and achieving this bet with the set of cards we have. Since it can be seen as two disjoint problems, we could develop two different AIs, say "bet AI" and "card AI". Card AI could have very good results in winning even if the bet is completely unrealistic. However we could also develop card AI in symbiosis with bet AI. For instance, if card AI is best at playing aggressively it should work better with a bet IA which makes high bets.

A remark was made in the last report by the reviewer : on what pieces of data can we train our neural network ? As pointed out in the last report, we do not know what move is the best in general (otherwise a neural network would be useless anyway) this is why our first idea consisted in using Q-learning [6] to train our neural network. Using the Q-learning paradigm we would not need to have data to train our neural network on, the neural network would (very schematically) play against itself and figure out the best moves on its own. I don't detail more since this approach has been described in the first report and that we had to drop it because of pragmatic constraints anyway.

Now even though this approach was dropped we thought of another approach to use neural networks in combination with the Monte-Carlo AI. To describe it I need to explain a "problem" in the current use of Monte-Carlo. When the Monte-Carlo does its "descents" (after it distributed random hands to the players and where it plays against them) it considers itself and the players as random AIs that is to say not very good. This is a problem as we would like the simulated games to be as close to real ones as possible and this is not the case when we say everyone plays at random. A naive idea is to make the players play as Monte-Carlo AIs in the descents (so the AI would recursively call itself on one less cards) however this approach is naive indeed as the computation time required to do this would simply be gigantic. A good but non optimal idea is to simulate players inside the descents as deterministic bot IA, this -we expect- is far better than considering random players and gives much more credibility to the played games inside the descent.

Now the idea using neural networks works like this : if we can train a neural network to take as input a hand and to output the card that the current Monte-Carlo IA would play with this hand,

then we can simulate the players inside the descent as using this neural network to play and it will be as if they were playing using the Monte-Carlo IA. We have now a version 2 of the Monte-Carlo algorithm, but now we can train another neural network to take as input a hand and to output the card that the 2nd version of the Monte-Carlo would play with this hand, then we can simulate players inside the descent as using this new neural network and they'll be even more intelligent than last time ! Of course we can do this as much as we want and we'll stop when the successive AIs stop getting better. This idea has not been yet implemented but we feel confident it would give good results. We will personally go as far as saying that - assuming the neural network simulates the Monte-Carlo accurately enough- then this approach is very close to the optimal way of playing *La Bâtarde* , a corollary is that if it does not work (that is to say the IA still gets beaten on average a fair amount of time by human players) then the level cap in *La Bâtarde* is low. One would not be able to get very good at this game since its skill will get overshadowed by the stochastic processes in the game.

Further development

The Monte-Carlo IA answers to the initial demand : an AI able to bet and play with different levels. But all the works started are not finished and it could be interesting to work on some. The main ideas are to use neural networks to improve the Monte-Carlo AI and to add the betting part to the deterministic bot. That could be done with the mobile device team to have a light and efficient bot for mobile, the current one is not really efficient and the Monte-Carlo IA is too energy intensive to implement on a mobile device.

3.5 WP3: Graphical User Interface (GUI)

In this package, we worked in close collaboration with the Core team (see 3.3), which implemented the rules of the game. In its current state, the computer client allows the user to play a full game of *La Bâtarde* . We also implemented language support [4] and some graphical options such as a fullscreen mode and different window resolutions. Overall, the GUI package represents about 6500 lines of code.

3.5.1 Graphic design

The graphic design was mostly made by François Pitois, member of the team, who provided us the necessary image files, and some previews of what the GUI would look like in its final state.
The creator of *La Bâtarde* , Jules Marconnier, allowed us to use his graphical resources, so that our program would not differ too much from the original game. Therefore, cards displayed in the program are the exact same one as the original physical game. On the other hand, we also took some liberties on other graphical parts, such as the fonts and the layouts.

3.5.2 Programming choices

The whole GUI code has been developed using the C++ language and the SFML library [7]. We used a oriented object paradigm. The code architecture is depicted in figure 3.3. Note that this is a simplified version of it, since there are actually more dependencies between files, and some minor files or classes have been omitted. The full documentation can be generated using Doxygen.

[4] Current version supports French, English, German, Spanish and Portuguese

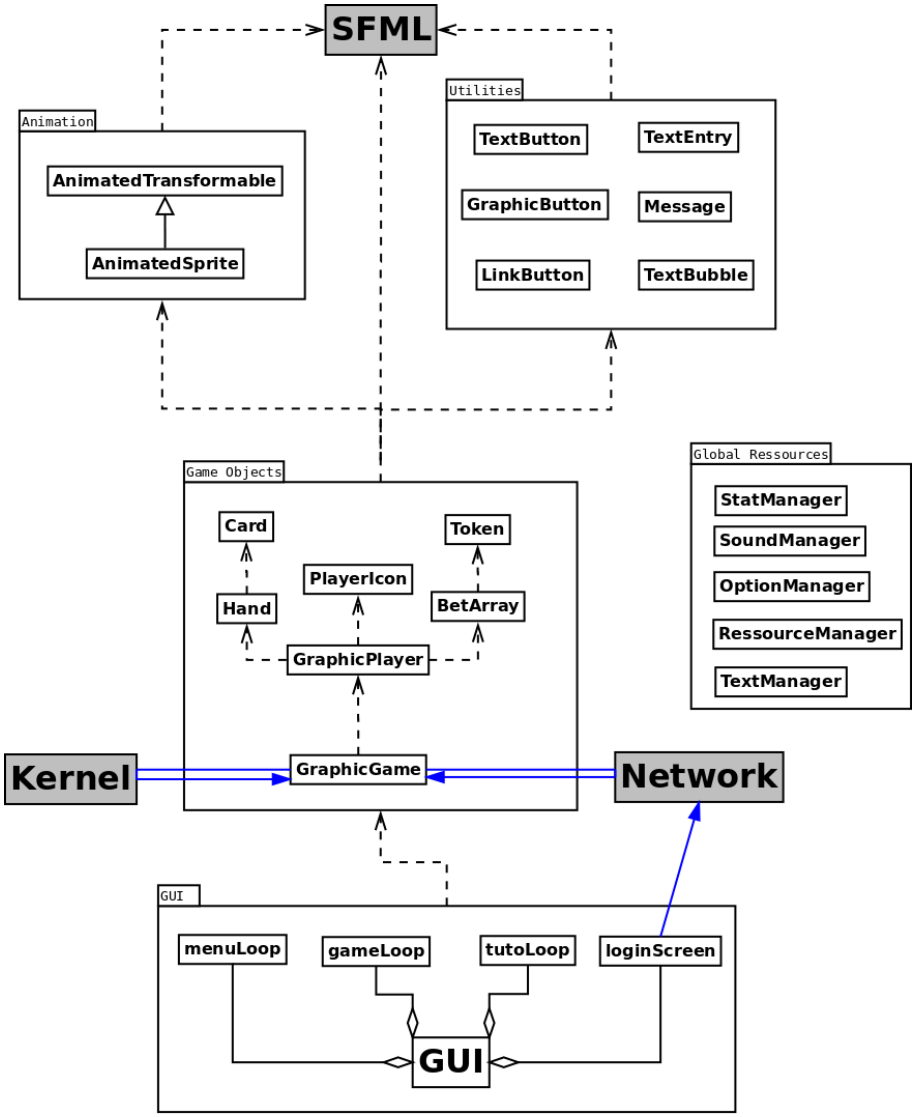


Figure 3.3: GUI branch architecture

3.5.3 The GUI class

The whole code of the package is articulated around the `GUI` class. This class implements some event loops, which are run by the program in order to render the interface and allow the player to interact with it. Several types of loop exist:

- The menu loop, which mostly displays messages and buttons.
- The game loop, which is the interface where the game occurs. This is the main interface of the program.
- The option loop, which allow us to change graphical and game preferences.
- The tutorial loop, which is a scripted game loop in which we can trigger events as we desire. It has been implemented using a simple script language (see 3.7 for more information).
- The login screen, which makes a link with the authentication part of the network work-package (see 3.8 for more information).

These loops are standard SFML loops, where each element displayed on the screen is updated 60 times per second. Because of their numbers, the menu loops have been re-factored into a unique loop that takes graphical objects as arguments and display them.

3.5.4 Animation

The GUI of CartomENS-IA directly rely on the SFML library. This allow us to display, move and transforms sprites and pictures on the screen. However, transformations implemented in SFML are instantaneous. In order to have fancy movements for cards and other objects, we had to implement those movements by ourselves. That is the purpose of the `AnimatedTransformable` class, which inherits from the SFML Transformable class. Every displayable object of the GUI then inherits from `AnimatedTransformable`.
`AnimatedSprite` is also a daughter class of `AnimatedTransformable` that take it to the next level, but only uses images, like cards, and not text nor buttons.

3.5.5 Utilities and game objects

Utility objects and game objects correspond to every object of the program that have something to display on the screen. Utilities are at a lower level in the architecture. They consist in basic objects that are in every GUI, such as messages, buttons, text entries, etc.
Those basic classes were implemented very early in the project. With them, we were able to create more complex objects, which are used in the game. The list of the game objects goes as follow :

Card : A sprite representing a card of *La Bâtarde* . The sprites are the official card designs of the physical game. This class should not be mistaken with the Core::Card class card. Actually, it contains graphical content, and a Core::Card that carries the information such as its color or its value.

Hand : A class containing several cards and displaying them in an arc. This is what is used to display cards on the edges of the screen (see the "Visuals" section in appendix C).

GraphicPlayer : This class holds every information possible on a player during the game, including its Hand instance, its icon, and its bets.

DropDownMenu : A menu that can be retracted to avoid overloading the game screen. It contains various information about the current state of the game, such as the current trump, the last trick played or the score and the rank of the four players.

BetArray : A selector array that allow the player to make his bet. It is divided into two parts: we choose the trump of the bet in the bottom part, and the number of tricks on the top part. This class is in communication with the core thanks to the `GraphicGame` class, which allow us to prevent players from betting something they are not allowed to bet.

GraphicGame : The "main" class of a round of *La Bâtarde* . At the launch of a round, this class is created, and instantiates four `GraphicPlayer`, a `DropDownMenu` and a `Core` in order to play. It also implements the progress of the game, such as transitions between bet phase and play phase, by calling the corresponding methods in the core.

ScoreDisplay: A window that displays at the end of a round or a game, making a summary of all the scores.

3.5.6 Resource management

Resource management is one of the trickiest part of the GUI's architecture. Indeed, it implements several features that needed to be thought correctly in order to be easily implemented. There are three classes dedicated to the resource management. There are meant to be instantiated only once, as global variables.

RessourceManager : This class contains every external resource file that is necessary to run the game. This includes the images (usually .png) and the font files (.tff). Those file are stored in some unordered maps, and can be accessed via pointers. This allow us to make great savings in memory space. Indeed, each sprite created by SFML needs a Texture object to be drawn of the screen. With this system, we instantiate the sprite by calling the `RessourceManager`. If the texture was loaded, then the manager returns it. Otherwise, it loads it in the map before returning it. This prevents any risk of loading the same texture twice.

TextManager : The `TextManager` class contains every piece of text that exists in the program. This is useful for the support of different languages. With this architecture, we instantiate a Message or a Button with a key, which is a string describing the text. The `TextManager` contains unordered maps that goes from keys to real strings. Given the key and the language in which the program is configured, the `TextManager` returns the correct text to be displayed by the class that sent the request. Text data are loaded from external txt files.

OptionManager : The `OptionManager` is meant to hold in its private attributes any option that are contained in the game. Options are written in an external .txt file to be saved between two runs of the program. The only place in the code where setters for the options are used should be the option loop of the `GUI` class. Options are loaded in the constructor of `OptionManager`, and are written back in the file in the destructor of the class. If the .txt file is missing or corrupted, we load the game with some default parameters, and it will be recovered when the program is closed.

SoundManager : The `SoundManager` class has been added rather late in the project. It stores every sound effects and music of the game, and allow us to play them whenever we want in the code.

StatManager : In order to improve the player's life into the game, statistics about games played are stored in the `StatManager` class. This class also handles some funny achievements that happen when the player does some particular actions during the game. A utility class `TrophyDrawer` is meant to display those achievements on the screen. All of these data are loaded from an external profile file, and written back at the end of the program.

3.5.7 Interfacing with the core package

In the computer software of CartomENS-IA, the GUI package is meant to be the main package. At launch of the program, we indeed instantiate a GUI, that will then instantiate a core and some artificial intelligences if necessary. The only places in the code where the core is called are in the `GraphicGame` class methods, the game loop and the main file.

3.5.8 Interfacing with the network package

The GUI of CartomENS-IA communicates over the network in two places :

- On the login screen, where we initiate a TCP communication with the distant server, and we send username and passwords to the remote database.
- During the game loop, where we retrieve actions of other players over the network and send our own. All of this is handled by the `ServerCommunication` class provided by the Network team (see 3.8).

3.6 WP4: Mobile device client

Unlike the PC version of the project, which is implemented in C++, the mobile device version was made in the Java language. It was developed with the Android Studio IDE[5].

3.6.1 Team Schedule in relating to the rest of the project

After several exercise sessions in order to make the team used to the Java language, we started developing our client. At first, we mostly copied the architecture of the PC client, since it had several weeks of advance (which was planned by the schedule). This allowed us to avoid loosing any time. However, as the developing went on, we gradually changed our code architecture in order to make better use of the particularities of our programming language and our device.

3.6.2 Graphical interface

Here is a little demo of our current status: https://www.youtube.com/watch?v=5W-8RRURIN8.

[5] https://developer.android.com/studio/features.html

Technical choices: animations are done by drawing bitmaps into Android canvas. We chose to have a thread dedicated to the drawing, which handle the canvas through Android `surface holder`. The bitmaps are handled by an independent `Resource Manager` which ensures that each picture is loaded only once. Loaded bitmaps that stop being used are unloaded to improve performance. The only case in which we do not use the canvas is to display the score ; here we take advantage of the `activity` class.

Animation model: we started from the `AnimatedTransformable` object built by WP GUI, but ended with a deeply reformed version, according to our own needs and philosophy. This object is the building block of our graphic interface. It can be animated by combining one rotation, one translation and one scaling in a desired period of time.

Object oriented programming All our other objects are derived from - or own - `AnimatedTransformable`. Here is a non exhaustive list of the high level objects that are directly used in the main "World" class.

Hand computes the position of the main player's cards - main methods are about adding, removing or selecting cards;

Board handles the flow of cards in and out the board;

BidInterface recovers the bid from the main player;

Player displays useful information about the other players - as their bid, their current tricks and so on.

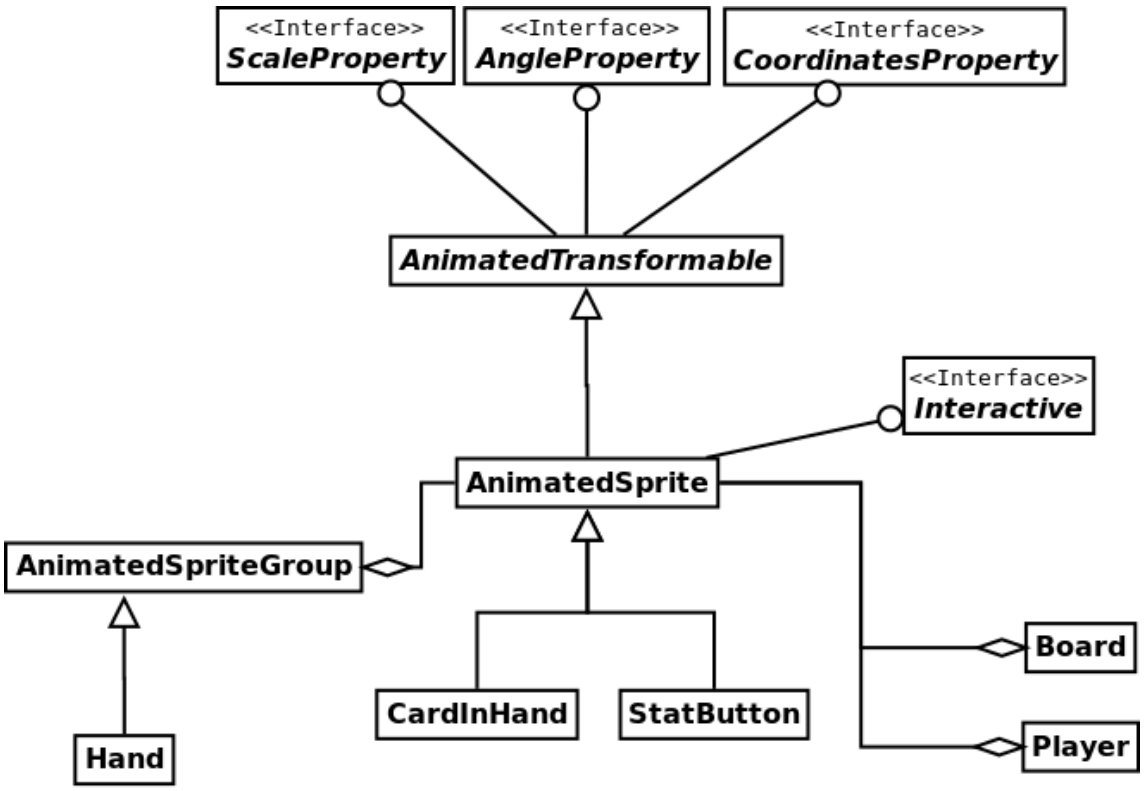


Figure 3.4: Mobile device architecture

3.6.3 Game Ergonomy choices

Gameplay: We have considered the classical drag-and-drop gesture, but we have eventually chosen another way of playing a card. The player has to first select a card by clicking on it. Then, the card appears to be selected by scaling and going up, slightly outspreading the other cards.

- clicking a second time on the selected card results in playing it;
- clicking on another card unselects the first card and selects the new one;
- clicking anywhere else on the screen unselects any card.

We have made this choice to prevent the player from misclicking and playing an unwanted card, while keeping the gameplay simple.

In game screen layout: Due to the lack of space inherent to any mobile app, we have chosen to exclude several elements from the screen:

- cards in other players hands;
- player's bids tokens;
- cards from previous tricks.

However, we include a small window for each player displaying:

- his icon;
- his number of already taken tricks for this deal;
- his bid for this deal.

3.6.4 Core

There were no reason to have a different core in the application than in the PC client. Hence, we used a direct translation of the PC core as a black box. Core events take place into a stack. Events regularly pop from the stack and are handled by the graphic interface. As a consequence, there is a desynchronization between what is visible on the screen and the real gamestate.

3.6.5 Final state of the work package

Unfortunately, the work of the Mobile Device Team could not go as far as the work of the rest of the project. There is still features to implement, that will probably be done in the future.

Méchoune We lack the time to implement the méchoune mechanics. This needs a deep modification of the Java core in order to handle asynchronous queries.

Network : Due to the fact that we are some weeks behind the rest of the project, we did not had time to implement the network part. However, it is planned that our code will communicate using TCP protocol with the CartomENS-IA server.

AI : No strong AI is currently available in the mobile version. A player discovering the game can dominate the current bots after only a few games.

3.7 WP5: Tutorial

Figure 3.5: Preview of the tutorial mode in CartomENS-IA

The tutorial package is here to help people that don't know how to play *La Bâtarde* . The goal is to have an interactive tutorial where new players will learn all the basics of the game, and maybe some advanced strategies. To make it interactive, we created an environment where the player will learn the rules by playing. There are be some parts of text, and other parts where the player is asked to play a card/make a bet.

The work was essentially divided in two parts: one theoretical part where designed the scripted games used to make people learn the rules, and another part where we implemented those games. For the implementation, we did not use the core because all the moves were forced. For the design of the games, we choose to divide the tutorial in several points:

- The cards: explanation of the order of the cards, with or without trumps;
- The tricks: the main part of the game, we learn how we can play a card, and how trumps work.
- The bets: here we learn how to bet at the beginning of each round
- Flow of the game: to finish the tutorial, we explain the general setting of the game (e.g. the number of cards in hands), and the last things players need to know before being able to play.

The figure 3.7 shows a preview of the tutorial interface (in French). As well as the rest of the application, the tutorial is translated in many languages.

To implement the tutorial, we chose to create a grammar (a kind of scripting language) that allows us to explain the basic steps of the tutorial. That way, tutorials are not directly encoded in C++. Here is an example of a series of instructions written in our script:

```
Play 2 Queen club
Play 3 Musician club
Message Pause tuto7
Message tuto8
```

The messages needs to be clicked to continue the tutorial when we have the argument `Pause`, a card is played with the command `Play`.

This grammar allows us to do all the basic operations. A harder tutorial was the one about the cards, because it is very specific: we have to draw the cards, and we want to move the cards. As all these functions were not useful in other tutorials, we decided to "hardcode" them (meaning, these functions cannot be called by the grammar).

Further work can be done in the tutorial for advanced players. If someone can find a good strategy it will be easy to implement the tutorial associated to this strategy. however the game seems to offer few possibilities of simple efficient strategies.

3.8 WP6: Network and server

The Network and server package is split in two parts:

- The Network part, in charge of handling the game when playing online with other players,
- The Server part, in charge of managing the pre-game phase : the lobby phase, authentication and cryptographic protocols.

3.8.1 Network

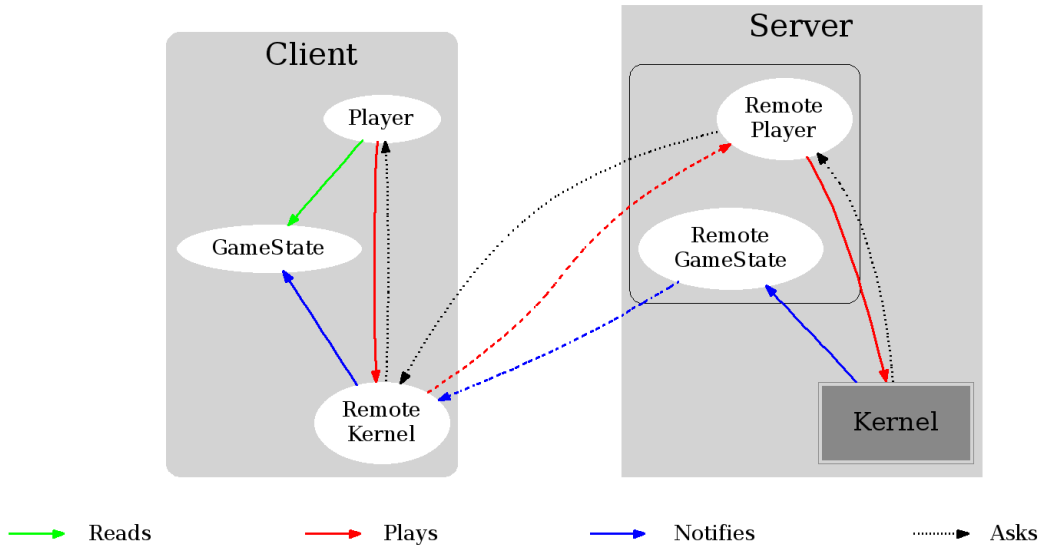


Figure 3.6: Network architecture

One of the major goals of CartomENS-IA was to develop an application on which we could play with several players. For that we needed a way to connect several players in one game, via an internet connection. But once the core had been written, we also wanted to have one unique interface for the kernel and the players, so that we did not have to rewrite another core for multiplayer games. This has been achieved by making alternate versions of the `Kernel`, `Player` and `GameState` objects that used the same interface as the original ones, but are just interface that connect a client and a server.

On the server there is a regular `Kernel`, with four `Players` and their corresponding `GameStates`. Some `Players` are actually `RemotePlayers`, and have a `RemoteGameState`. These two objects are connected via a TCP connection to a `RemoteKernel` that runs on the client's side (fig. 3.8.1). This `RemoteKernel` has the actual `Player` and its `GameState`. Then when the Kernel wants the Player to play, it asks the `RemotePlayer`, that forwards the request to the `RemoteKernel`, the `RemoteKernel` asks the Player and forwards the answer to the `RemotePlayer`, that can answer to the server's Kernel. Similarily, the information the `Kernel` gives to the `RemoteGameState` is transmitted to the client's `GameState`, so that the `Player` has access to the information of the current game.

3.8.2 Méchoune

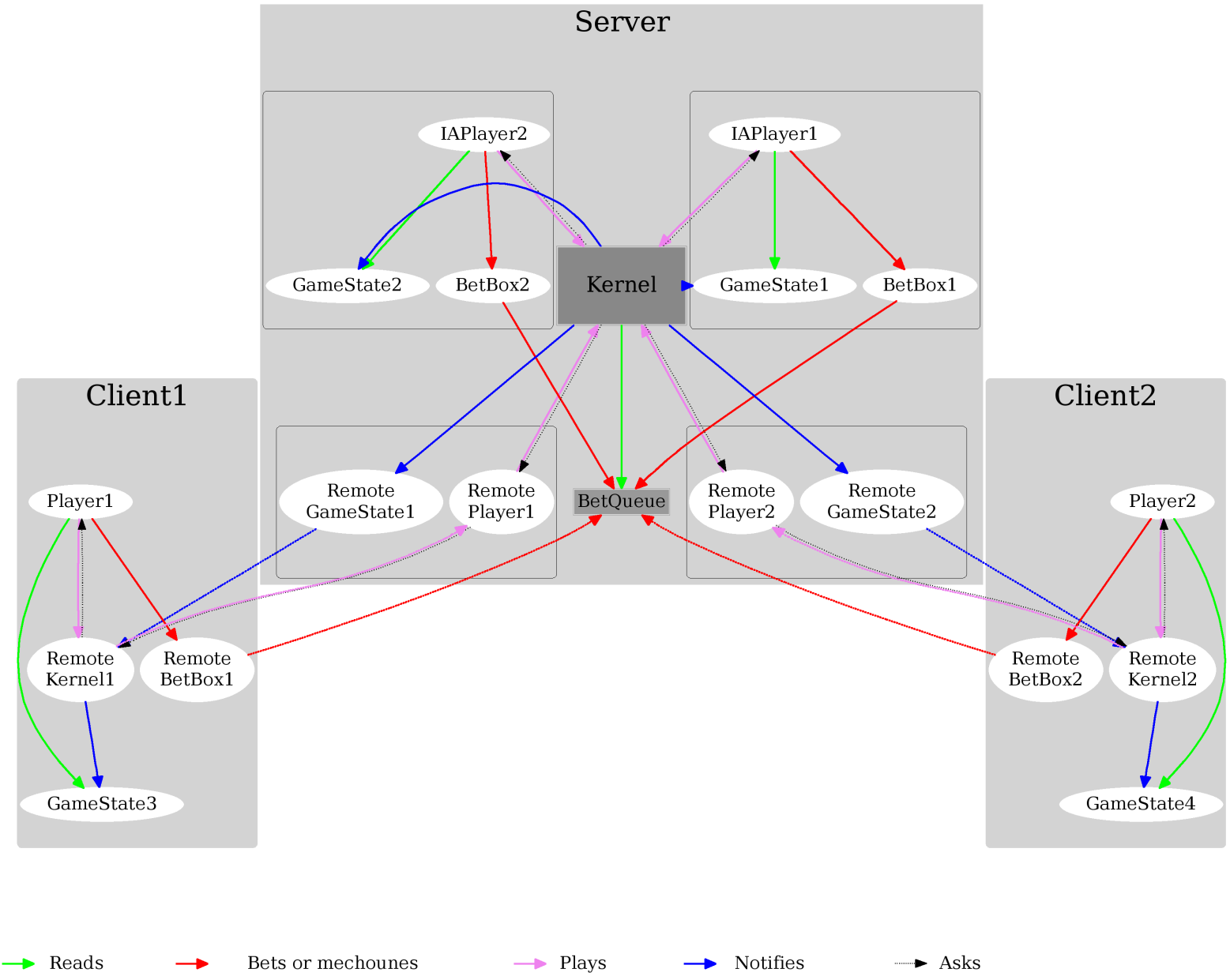


Figure 3.7: Network architecture with the méchoune

One part that has been hard to implement in the core and the network is the méchoune. The méchoune is an action any player can do during the bet phase. So it is very different from the bets and the cards played, for which we can ask to every player what does he bet or play at their turn since they can only bet something or play a card during their turn. For the méchoune we cannot asks every player whether they méchoune since that would imply that they cannot méchoune at every moment. This was however the first chosen solution, asking each player whether they have méchoune the previous bet before accepting the next one. So this requires blocking calls to method méchoune of the `Players`, giving them all the time they need to reply. However in a normal game there is no delay and anyone can speak at any moment.

We had to let the human players (local and remote) méchoune at any moment during the bet phase, but we still have to ask the IA players to méchoune sequentially, since they are not parallel. The solution that implemented works with a queue where everyone writes their bet when asked or the fact that they méchoune. Then we cannot have blocking calls when asking players to bet

or we could miss a méchoune. That means that we have to notify a player that they must bet and wait for something to arrive in the queue, a bet or a méchoune. For the IA player there are two solutions: assume that they will reply fast enough and make a blocking call or make the call in a separate thread and wait for something to arrive in the queue. We chose the first solution, even if the IA player does not reply instantaneously. When it replies, we put the actions that were received during the call to the IA in the queue and then the action of the IA to treat them in the right order. The only problem this causes is that when the IA computes its bet, the `Kernel` cannot notify everyone that a second player is méchouning, bit we still enforce the rules on the order of treatement of actions.

Now we must be able to do this remotely, that is to allow each remote human player to write in the queue, but we cannot do it via the `RemotePlayer` since we cannot have several `RemotePlayer` listening at the same time in the same thread.

In the final architecture (fig. 3.8.2), each Player has a `BetBox` that can write directly in the `BetQueue`, directly for the local players or via the TCP socket for the remote players. That way, the bet and the méchoune is transparent for the player, on the server or on the client.

3.8.3 Server

Connection and cryptography

When a new user opens a TCP connection with the server, the very first thing done is a key exchange protocol. The messages exchanged during this phase are not encrypted. Using the `libsodium`[6] cryptography library and its implementation of elliptic curve public key algorithms [1], the client and server generate their asymmetric public/private key pair. Then, they send their public key to each other and derive two new symmetric keys, using the XSalsa20 protocol[2]: one used to send messages, and one used to receive messages. With this first exchange done, the client and server can cypher each message with strong cryptographic functions, in addition to the use of a nonce. This has two advantages, authentication and secrecy.

Authentication: once the key exchange is done, both the client and the server are sure that only the entity that did the key exchange can properly cypher the messages. This way, we are sure that no third agent can pretend to be the server and lie to the player, or pretend to be the player and send false informations to the server.

Secrecy: because each message is cyphered, nobody can spy on the conversation. The nonce is a counter concatenated with the message, and incremented at each send operation so that it is unique for each message. The nonce is used so that the same message is never encoded twice the same way. This way, even if somebody is listening, they will not be able to deduce the content of the messages or the key, even if many similar messages are exchanged.

Authentication

To be allowed to play CartomENS-IA online, each user must authenticate with a login and a password. This phase happens after the connection, so each message is fully cyphered.

In order to authenticate, the user enters a login and a password. For security reasons, the clear password is never sent and never stored. The server uses a `SQLite` database to store the username and a salted hash of the password[3]. Because the database contains only a salted version of the password, an attacker stealing the database could not deduce the user's clear password.

First of all, the user sends their username. The server looks in the database and sends back the salt associated to the username, if it exists. With this salt, the user hashes their password with the salt to get the salted password.

At this point, if the user simply sent their salted password to the server, an man-in-the-middle (MITM) attacker could get the salted password and use it to authenticate later on. We wanted to avoid this. We wanted a protocol that did not allow a MITM attacker to authenticate without the original user. In order to do this we added a new step.

The server sends a random string to the client and the user hashes their salted pass with this random string, getting a key. Then they send this key to the server. The server receives the key,

[6] https://github.com/jedisct1/libsodium

hashes the salted password stored in its database with the random string, and compares with what the user sent. This way, the key exchanged changes at each authentication because it depends on the random string, so in order to authenticate, knowledge about previous authentication is not sufficient.

Once the user is authenticated, the server allows them to create or join a game.

Account Creation

The requests for an account creation are handled by the server that makes sure that the salted pass and the pass are cyphered before sent. In theory, any entity could ask for the creation of a new account, but we added the possibility to create an account from the website. Once all the informations are given, the website calls a `Python` script that uses the `libsodium` cryptography library to salt the password before sending it to the server. The server then adds the username, its salt and its salted pass to the database.

Lobbies and multiplayer games

After authentication, users can ask different things:

- Create a new empty game: The server builds a new empty Lobby, and adds the user to the lobby
- Join a given lobby : The client sends the id of the lobby that they want to join, the server checks if it exists, if there is room left, and adds the user to this lobby
- Join matchmaking : Join a random lobby that has room for an additional player.

Once the player is affected to a lobby, it receives information before the game starts. All members of a lobby are notified when a new player joins their lobby. This way they know how many players are still needed to start the game. They also receive the lobby id so that they can share it with their friends to play together in the same lobby.

The master of the lobby can then ask to start the game. If there is less than the number of players needed, the server automatically fills the lobby with IAs. This allows players to play even if they do not find a lobby of human players.

Conclusion

All the objectives targeted by this project has been fulfilled. We even got the time to add features or easter eggs. Moreover this project allowed students to train and progress in various topics, including programming languages, design pattern, management of a team...
Globally, all of us are satisfied with the quality of the final result.

The computer client and the mobile device client will be publicly released soon.

Currently the CartomENS-IA server is running on the server of the team **Avalon**, at the ENS de Lyon. It is enough for now, because we are not expecting a huge number of players. However, this may change in the future.

The code of the client part is open source, but the code of the server will remain hidden to prevent the creation of a concurrent server. The code and the resources (images, textures that belongs to Jules Marconnier) are protected by a license.

There is still a lot of work to do, so it is not over. In fact, there is currently talks with Jules Marconnier about the continuation of the project later on over the incoming months or years. A small part of the team is ready to handle this. We need to maintain the server, add features, correct undiscovered bugs, or modify the application in order to scale when necessary.

One day, the application may be put on the market and make money using advertising or any other way. Today, it is free, because we think that we have better chances to build a community of players that way.

Bibliography

[1] Bernstein, D. J. Curve25519: new diffie-hellman speed records. In *International Workshop on Public Key Cryptography* (2006), Springer, pp. 207–228.

[2] Bernstein, D. J. The salsa20 family of stream ciphers. In *New stream cipher designs*. Springer, 2008, pp. 84–97.

[3] Biryukov, A., Dinu, D., and Khovratovich, D. Argon2: the memory-hard function for password hashing and other applications, 2015.

[4] contributors, W. Monte carlo tree search — wikipedia, the free encyclopedia, 2018. [Online; accessed 12-January-2018].

[5] creativity research group, C. Monte carlo tree search (mcts) research hub., 2010-2013. [Online; accessed 5-November-2017].

[6] Heinrich, J., and Silver, D. Deep reinforcement learning from self-play in imperfect-information games. *arXiv preprint arXiv:1603.01121* (2016).

[7] SFML. Sfml website, 2018. [Online; accessed 12-January-2018].

Appendix A

Team

Leader: Bethune Louis

Team:

- Benoit Tristan
- Bethune Louis
- Blondeau-Patissier Lison
- Champseix Nicolas
- Coiffier Guillaume
- Devevey Julien
- Deschamps Quentin
- Fernandez Simon
- Guépin Florent
- La Xuan Hoang
- Léchine Ulysse
- Pitois François
- Pouget Stéphane
- Rochette Denis
- Vacus Robin
- Valentin Herménégilde
- Valque Léo
- Vavrille Mathieu

Appendix B

Pictures from the Interludes

Figure B.1: The poster for the event

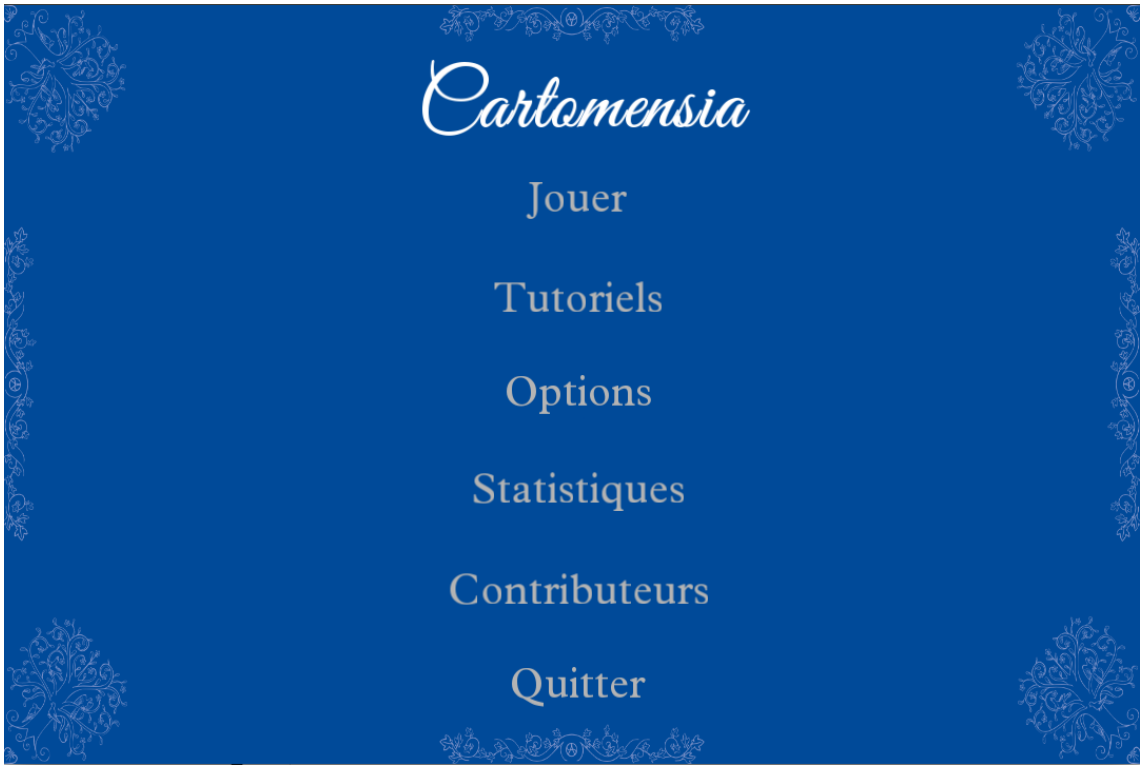
Figure B.2: The **La Bâtarde** tournament

Figure B.3: Prizes for the tournament

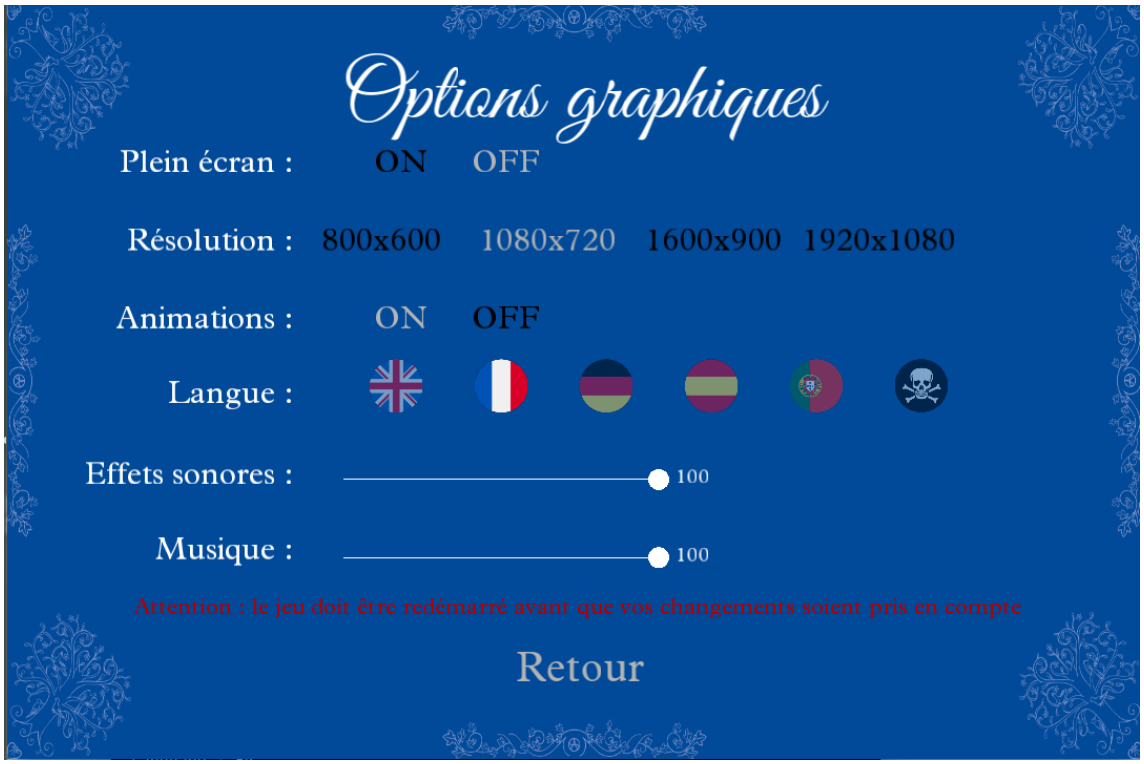
Figure B.4: Setting for real-life games

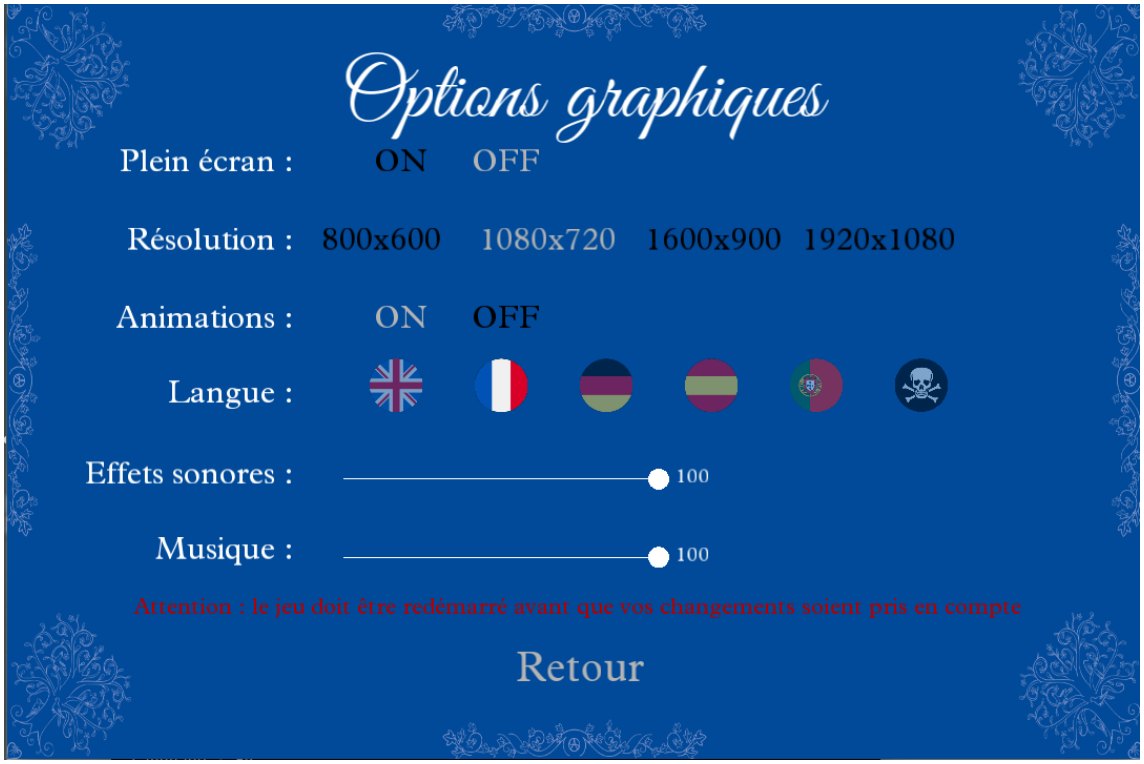
Appendix C

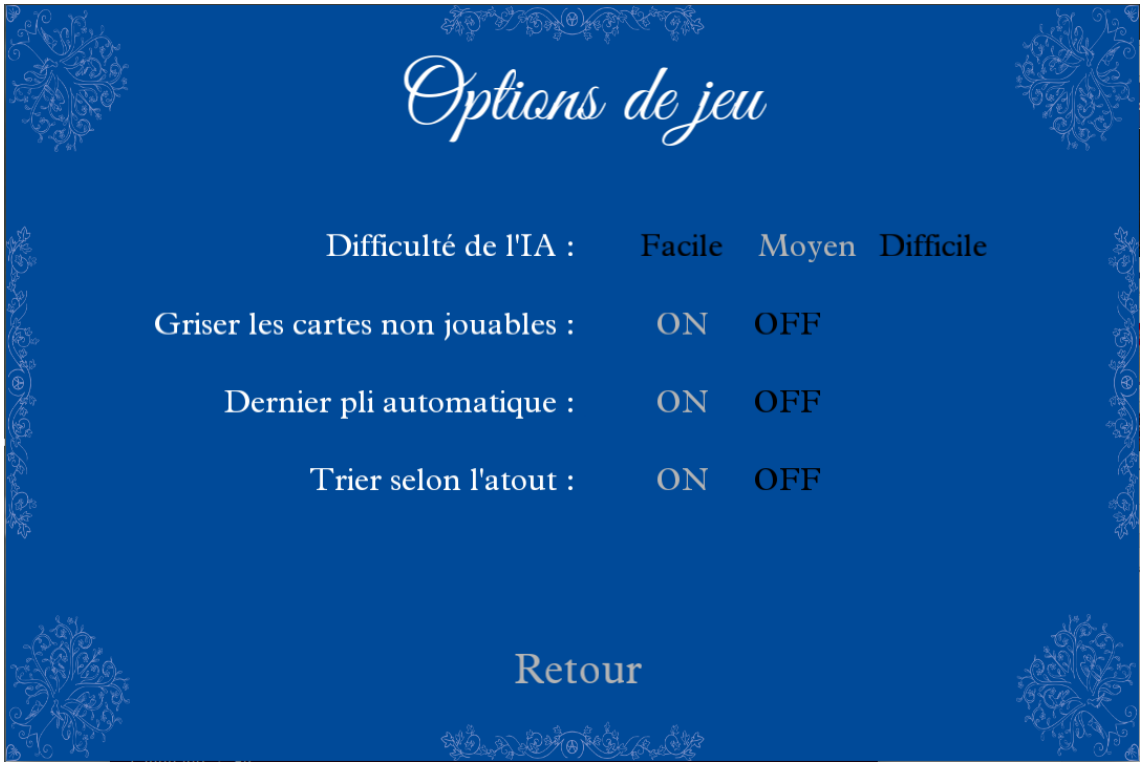
Visuals




Cartomensia
Nom de compte
Mot de passe
Jouer hors-ligne
Se connecter
Quitter


Options graphiques
Plein écran : ON OFF
Résolution : 800x600 1080x720 1600x900 1920x1080
Animations : ON OFF
Langue :
Effets sonores : 100
Musique : 100
Attention : le jeu doit être redémarré avant que vos changements soient pris en compte
Retour


Options graphiques
Plein écran : ON OFF
Résolution : 800x600 1080x720 1600x900 1920x1080
Animations : ON OFF
Langue :
Effets sonores : 100
Musique : 100
Attention : le jeu doit être redémarré avant que vos changements soient pris en compte
Retour


Options de jeu
Difficulté de l'IA : Facile Moyen Difficile
Griser les cartes non jouables : ON OFF
Dernier pli automatique : ON OFF
Trier selon l'atout : ON OFF
Retour


Succès et Statistiques
0 Meilleur score
0 Meilleure place
24 Nombre de parties
0 Nombre de victoires
0 Nombre de seconde place obtenues
0 Nombre de troisième place obtenues
0 Nombre de dernière place obtenues
24 Nombre de parties quittées
12 Nombre de manches
46 Nombre total de points reçus
10 Nombre de manches méchounées jouées
8 Nombre de manches chounées jouées
4 Moyenne de points par manche
15 Nombre de plis obtenus
1 Moyenne de plis par manche
1 Nombre de plis gagnés avec un chat de trèfle
Retour


Bienvenu dans ce tutoriel pour apprendre à jouer à La Bâtarde.


A vous de jouer !
Cliquez sur une carte de trèfle.


Cartes : 6
Manche 2/10
Atout : SANS ATOUT
Choune
Pari
Score
Total
Vous 2 8 20
Ouest 0 4 4
Nord 0 0 8
Est 0 4 4
Manche suivante


0
TOUT
ATOUT
0
TOUT
ATOUT
2
SANS
ATOUT
TOUT
ATOUT


TOUT
ATOUT