Université Nice Sophia Antipolis

STUDY AND RESEARCH WORK

Report: Puerto Rico

TEAM ROCKET

JANIN Rémi
DIALLO Elhadj Mamadou Foula
PAJANY CARPIN CAOUNDIN Allan

Teacher
Mr **MALAPERT Arnaud**

January 23th 2019

TABLE OF CONTENTS

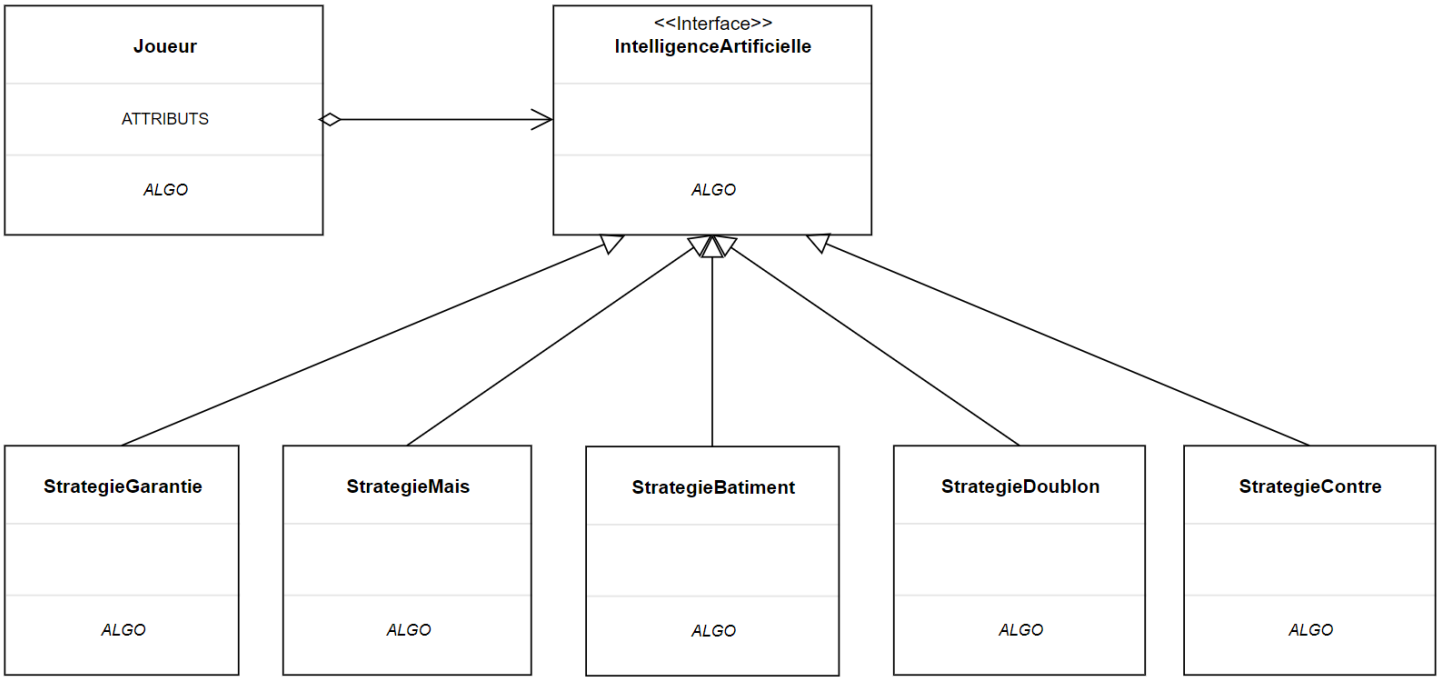
INTRODUCTION

Following our Puerto Rico board game development project, we will present in this report our research work and our news implementations concerning the management of several artificial intelligences.

1. DEVELOPMENT

Since we wanted to have rich strategies for our A.I. and were missing most of the large buildings and almost all of the small purple buildings, we decided before we started implementing A.I. to add all the missing buildings.

Then, in order to manage and manipulate our strategies, we used the design pattern strategy as follows:



Picture 1: Design pattern strategy in our code

2. A.I. GUARANTEE

2.1 Description

It has a unique strategy where its goal is to obtain victory points per load as quickly as possible, thus producing and loading the most barrels. We have reduced random choices as much as possible.

It follows the following procedure:

- ❖ Role choice:
  - **SETTLER -> BUILDER -> MAYOR -> CRAFTSMAN -> CAPTAIN**

    If the chain is not respected, one chooses either the role with the highest number of doubloons or the first available role.

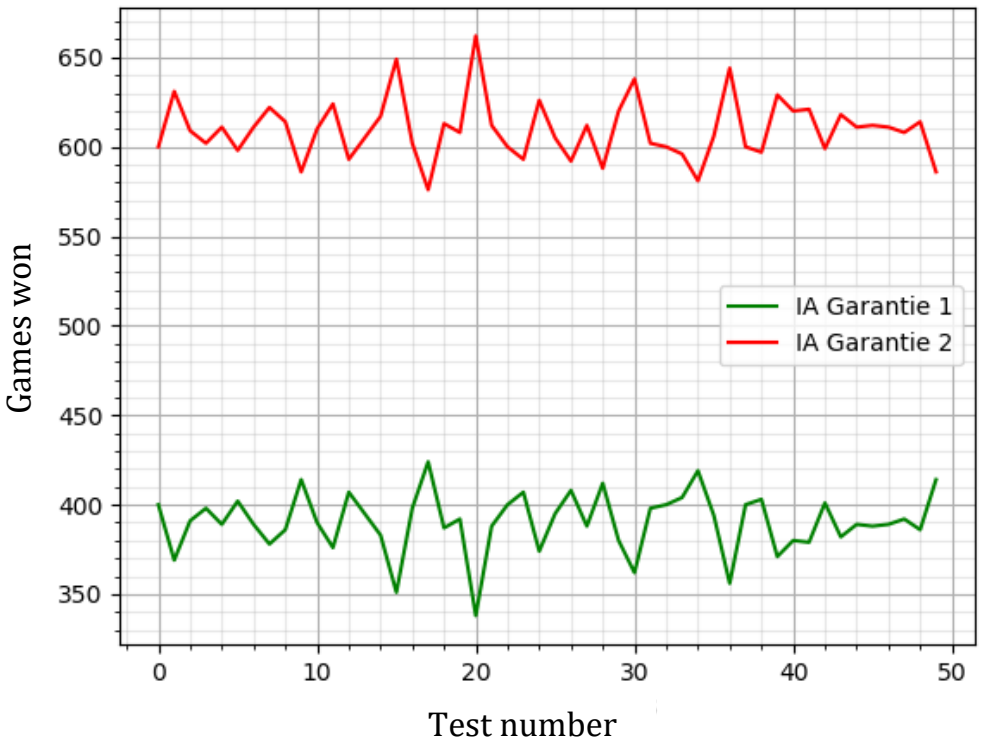
- ❖ Plantation choice: mainly corn, if not planting linked to a built production building, if not first in the list.
- ❖ Building choice: oriented towards the construction of buildings related to its occupied plantations or large buildings (if doubloons is sufficient).
- ❖ Ship choice: choose the ship with the most space and/or having the goods chosen for loading.
- ❖ Barrels choice:
  - Crafstman Phase (privilege): choose the barrel with the highest number.
  - Trader Phase: choose the most expensive barrel that can be sold.
  - Captain Phase: choose the cask that can be loaded and yields the most victory points.
  - Settlers Placement: mandatory as long as we can place them.

2.2 Performances

We test the performance of our A.I. Guarantee on 1000 games played and look at the average of games won, below are the results of 1000 games on 50 tests:

- A.I. Guarantee vs. A.I. Random: A.I. Guarantee generally wins by more than 99%.
- A.I. Guarantee vs himself :

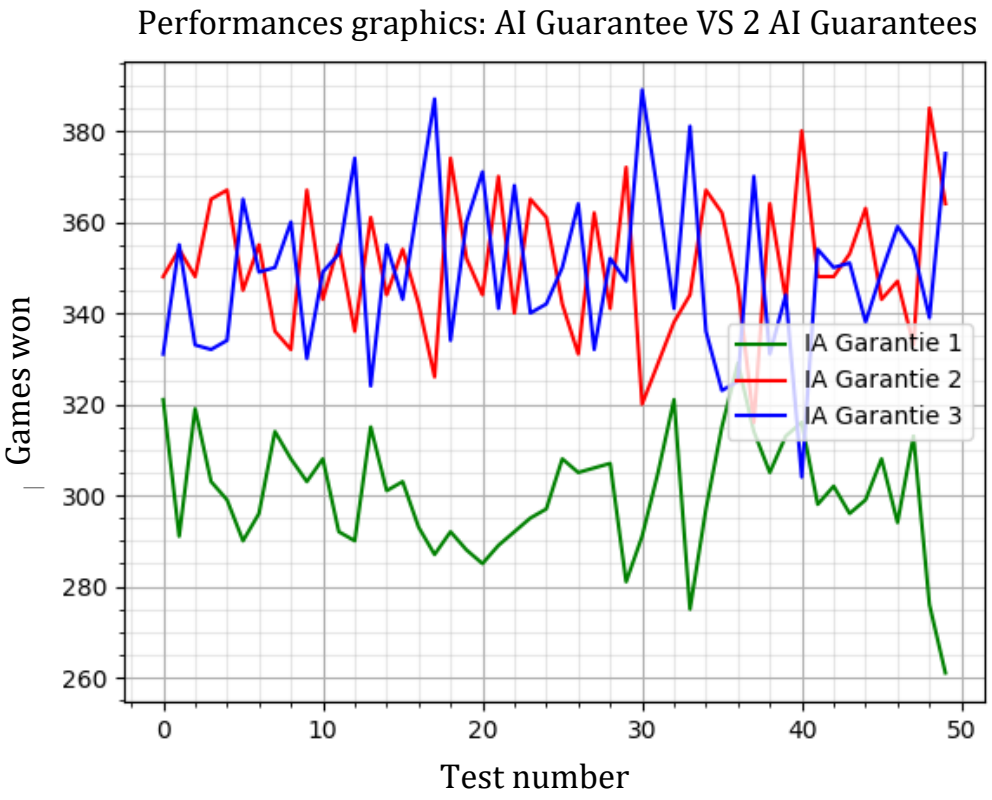
Performances graphics: AI Guarantee VS himself



Picture 2: A.I. Guarantee VS himself test

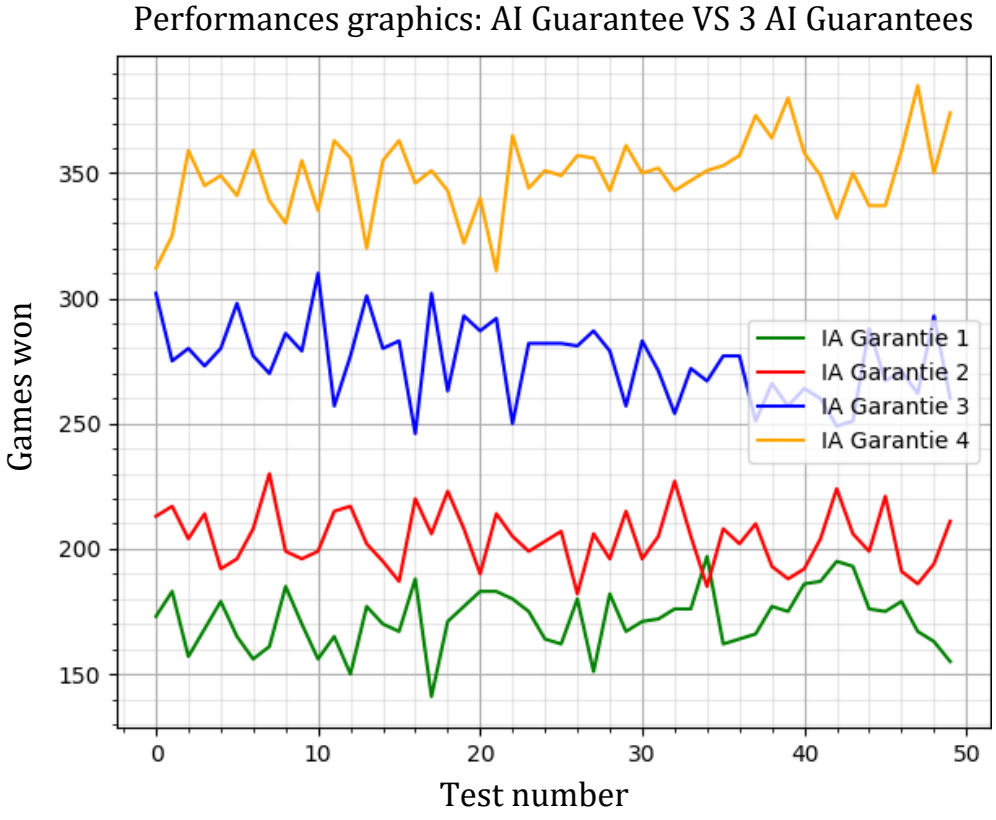
After this test, we were intrigued that the A.I. that starts a game in second position always wins, so we tried testing with a higher number of A.I. to find some correlation.

- A.I. Guarantee vs 2 A.I. Guarantees:



Picture 3: A.I. Guarantee VS 2 A.I. Guarantees test

- A.I. Guarantee vs 3 A.I. Guarantees:



Picture 4: A.I. Guarantee vs 3 A.I. Guarantees test

We get the same relationships every time, the best A.I. are always the ones who start last. Simply, we tried to explain it by the fact that since they act in the same way and according to the rules of the game (when choosing a character, all players perform the same action, except for the prospector), the choice of the first player gives the advantage to the next player when he will have to choose a role, for example if we take the following configuration :

SETTLER -> MAYOR -> CRAFTSMAN -> CAPTAIN

Phase 1:

Bot1 choose settler -> Choose plantation

Bot2 -> Choose plantation

Phase 2:

Bot2 will then no longer have to choose the settler, because *Bot1* already choosen the role and has allowed to *Bot2* to choose a plantation, so it can reach his final goal more quickly. And this done like domino effects, the last to fall win.

3. A.I. AMBITIOUS

3.1 Description

Our A.I. Ambitious had to have as its primary function the analysis of the game part, including its board, the game elements and the players' board, to put it simply, all the observable elements of the game. This would have allowed him to take advantage of or disadvantage other players. All this would have been done by changing strategy as the game evolved, however this method proved to be too Ambitious compared to the time we had, to our knowledge in the field of artificial intelligence and the balance with object-oriented design.

Our current IA Ambitious manages only 4 strategies defined in the following classes:

- ✓ **StrategieMais**: Plant, produce and export only corn.

  SETTLER -> MAYOR -> CRAFTSMAN -> CAPTAIN
- ✓ **StratégieContre (Anticipation)**: Disadvantaging the opponent by forcing him to make random choices. For example, loading ships with goods that other players don't have, or anticipating role choices...

- ✓ **StrategieBatiment:** Build as many buildings as possible in order to have more building victory points than your opponents.

  SETTLER -> MAYOR -> BUILDER

  The settler allows us here to operate quarries (up to 4) in order to obtain discounts when buying buildings.

- ✓ **StrategieDoublon:** Directing his choices towards obtaining doubloons quickly.

  SETTLER -> MAYOR -> CRAFTSMAN -> TRADER -> PROSPECTOR

  When we can sell a goods, we will try to sell the one that costs the most. Then, if the role of Builder is chosen (by strategy, by default or because of another A.I.), we will first have to think about building the "Small and Large Market" and the "Office" which allow respectively to sell a goods for 1 and 2 more doubloons and the same type of goods.

3.2 Impacts on the game

We had to reorganize our code so that A.I. Ambitious could retrieve the information from the game. Indeed, we set up a Game class in which we moved the information previously initIalized in the GameMotor class. We later wanted to implement the "Observer" pattern so that players could observe each other and thus apply the main function of the A.I. Ambitious, but we got stuck on questions of feasibility and redundancy with respect to the current implementation of our code, such as for example, what would be the right time to update strategies, knowing that each time the observable changes, a notification must be sent to the observers who must then update their strategies (A.I. Ambitious)

So we decided to stick with a simple implementation with methods handling the strategy change in the *Joueurs* class and an update method using this change in the *MoteurDuJeu* class. However, these changes didn't directly impact the *StrategieGarantie* class.

3.3 Performances

We keep the same testing methods for all A.I.

- A.I. Ambitious VS Random

```
[INFO] --- exec-maven-plugin:1.6.0:java (default-cli) @ PuertoRico ---

BOT Random 1 a gagné 28 partie(s)       2.8000002%
BOT Random 1 a gagné en moyenne 10 points de victoires grace a ses batiments.
BOT Random 1 a gagné en moyenne 7 points de victoires grace a ses chargements.
BOT Random 1 a gagné en moyenne 0 points de victoires bonus grands bâtiments.
BOT Random 1 a gagné en moyenne 15 doublons.

BOT Ambitieux 1 a gagné 972 partie(s)   97.2%
BOT Ambitieux 1 a gagné en moyenne 10 points de victoires grace a ses batiments.
BOT Ambitieux 1 a gagné en moyenne 33 points de victoires grace a ses chargements.
BOT Ambitieux 1 a gagné en moyenne 1 points de victoires bonus grands bâtiments.
BOT Ambitieux 1 a gagné en moyenne 18 doublons.
```

Picture 5: A.I. Ambitious VS Random

Here the results are quite relevant knowing that we are faced with a random.

- A.I. Ambitious VS himself

```
[INFO] --- exec-maven-plugin:1.6.0:java (default-cli) @ PuertoRico ---

BOT Ambitieux 1 a gagné 439 partie(s)   43.9%
BOT Ambitieux 1 a gagné en moyenne 12 points de victoires grace a ses batiments.
BOT Ambitieux 1 a gagné en moyenne 19 points de victoires grace a ses chargements.
BOT Ambitieux 1 a gagné en moyenne 0 points de victoires bonus grands bâtiments.
BOT Ambitieux 1 a gagné en moyenne 15 doublons.

BOT Ambitieux 1 a gagné 561 partie(s)   56.1%
BOT Ambitieux 1 a gagné en moyenne 12 points de victoires grace a ses batiments.
BOT Ambitieux 1 a gagné en moyenne 21 points de victoires grace a ses chargements.
BOT Ambitieux 1 a gagné en moyenne 0 points de victoires bonus grands bâtiments.
BOT Ambitieux 1 a gagné en moyenne 15 doublons.
```

Picture 6: A.I. Ambitious VS himself

However, this time, after several tests we noticed the same thing as with the bot Guarantees, for 2 A.I. of the same type, the last one always ends up taking the advantage. So they're acting as our guaranteed A.I.

- A.I. Ambitious VS A.I. Guarantee

```
[INFO] --- exec-maven-plugin:1.6.0:java (default-cli) @ PuertoRico ---

BOT Garantie 1 a gagné 830 partie(s)    83.0%
BOT Garantie 1 a gagné en moyenne 10 points de victoires grace a ses batiments.
BOT Garantie 1 a gagné en moyenne 28 points de victoires grace a ses chargements.
BOT Garantie 1 a gagné en moyenne 2 points de victoires bonus grands bâtiments.
BOT Garantie 1 a gagné en moyenne 15 doublons.

BOT Ambitieux 1 a gagné 170 partie(s)   17.0%
BOT Ambitieux 1 a gagné en moyenne 10 points de victoires grace a ses batiments.
BOT Ambitieux 1 a gagné en moyenne 20 points de victoires grace a ses chargements.
BOT Ambitieux 1 a gagné en moyenne 1 points de victoires bonus grands bâtiments.
BOT Ambitieux 1 a gagné en moyenne 17 doublons.
```

Picture 7: A.I. Ambitious VS A.I. Guarantee

In this situation the A.I. AMBITIOUS is not very effective. Our hypotheses are that there is a lack of robustness for changes in strategies, which should be finer, especially in terms of conditions; the strategies themselves are surely flawed; The role choice method needs to be more efficient in the *StrategieContre* class and perhaps we need more strategies.

4. SUMMARY

First of all, we are generally very satisfied with the results, because given the complexity of defining artificial intelligence, we expected rather inconsistent results, as we tried as much as possible to create a behaviour that was as close as possible to human and “dynamic”. An idea to improve our A.I. "dynamically" would be to make it play several games under different conditions, retrieve the data and take the best configurations each time, a bit like machine learning. With a little more knowledge, we could have implemented a more robust ambitious A.I. and choose design methods that might be more suitable for board games (use of fuzzy logic, generic algorithms, neural networks, Min-Max algorithm, finite automata...)

Concerning the development part, although we would have liked to use approaches with different A.I. design patterns and algorithms, we are quite satisfied with the simplicity of its implementation.

CONCLUSION

To conclude, we can say that this project has given us a very enriching experience in how to develop A.I. for a board game first.
Despite our lack of experience in the artificial intelligence domain, we were able to set up basic A.I., based mainly on conditions, which allowed us to see the big number of possibilities that can be obtained without using complex design methods and thus understand the basics of artificial intelligence.