École européenne d'ingénieurs de chimie, polymères et matériaux
Université de Strasbourg

Game Theory Computer Project Report: Cooperation

Caroline Solon, Alexis Lemoine

March 13, 2024

Abstract

In this report we will detail our implementation of the cooperation problem in Microsoft Excel using Visual Basic code. The operation of the simulation program will be explained and the results obtained as a function of the different behaviors will be analyzed.

Contents :

1 Introduction to the problem of cooperation

A brief description of Game Theory

Game theory is a branch of mathematics that offers a formal description o f strategic interactions between individuals or groups. The outcome for each group or individual depends not only on its own actions, but also on those of others. Game theory helps us to understand how different players can act in competitive or cooperative contexts to maximize their gains. Its foundations were laid in the 1920s by mathematicians Ernst Zermelo and Emile Borel, and further developed in the 1940s by Morgenstern and Von Neumann.

Game theory problems can be classified according to different criteria:

Cooperative and non-cooperative games:

- Cooperative: Agents can form alliances to improve their results.
- Non-cooperative: Agents make decisions independently, and the focus is on individual strategies.

Sequential and simultaneous games :

- Sequential : Agents make their decisions with knowledge of those of others.
- Simultaneous : Agents make decisions at the same time, unaware of each other's decisions.

Zero-sum and non-zero-sum games:

- Null sum: Total winnings between players remain constant; if one wins, the other loses.
- Non-zero sum: Wins and losses don't necessarily cancel each other out; players can win or lose at the same time.

Game theory is a central tool in the theorization of many fields, including economics, biology and even international relations.

Cooperation problems

The cooperation of small entities that respect simple rules enables the formation of complex struc- tures that behave in ways that are difficult to predict. Human societies are a case in point, being the result of many individuals behaving in different ways. Given that certain behaviors are favored over others, it is interesting to be able to model them.

Cooperative games can be used to analyze patterns of conflict between collective and individual interests. By modeling different types of behavior, we can simulate, analyze and even predict which strategies agents should adopt to maximize their gains. In real life, cooperation can be explained by reciprocity, (cooperation in the expectation of benefits in return), altruism (if individuals are genetically related, they have an interest in helping each other), social and cultural norms and also certain incentive mechanisms.

One of the best-known cooperation problems is the prisoner's dilemma, formulated in 1950 by Albert Tucker as follows:

Two prisoners, accused of being accomplices to a crime, are held separately by the police. Each is informed that :

- If one of the two confesses and the other doesn't, the former will be rewarded (gain +1) while the latter will be heavily condemned (gain -2),
- If both confess, they will both suffer a light penalty (-1 gain).
- If neither confesses, both go free (win 0).

In this two-agent problem, the dilemma arises from the non-coincidence between individual and collective interests.

Our implementation of the Cooperation Problem

In this report we implement a generalization of the prisoner's dilemma to N individuals. Each agent is exposed to the dilemma and must choose either to cooperate or not to cooperate, i.e. defect. This choice is made without knowing the other's and depends on the agent's personality. The different personalities are also called "rules", which are detailed below. Our model is then a simultaneous non-zero-sum non-cooperative game.

Here's how the points are counted:

- If two agents cooperate, they both earn 3 points.
- If only one agent cooperates and the other chooses to defect, the cooperating agent loses one point and the other gains 4.
- If no one cooperates, the balance sheet is zero.

By default, the model includes 400 agents placed on a *20×20* grid (the world is flat). Each agent initially adopts a "rule", and for a fixed number of rounds each agent must interact with another at least once. Once all the rounds have been completed, the points are counted and it is possible to determine which profile is the winner.

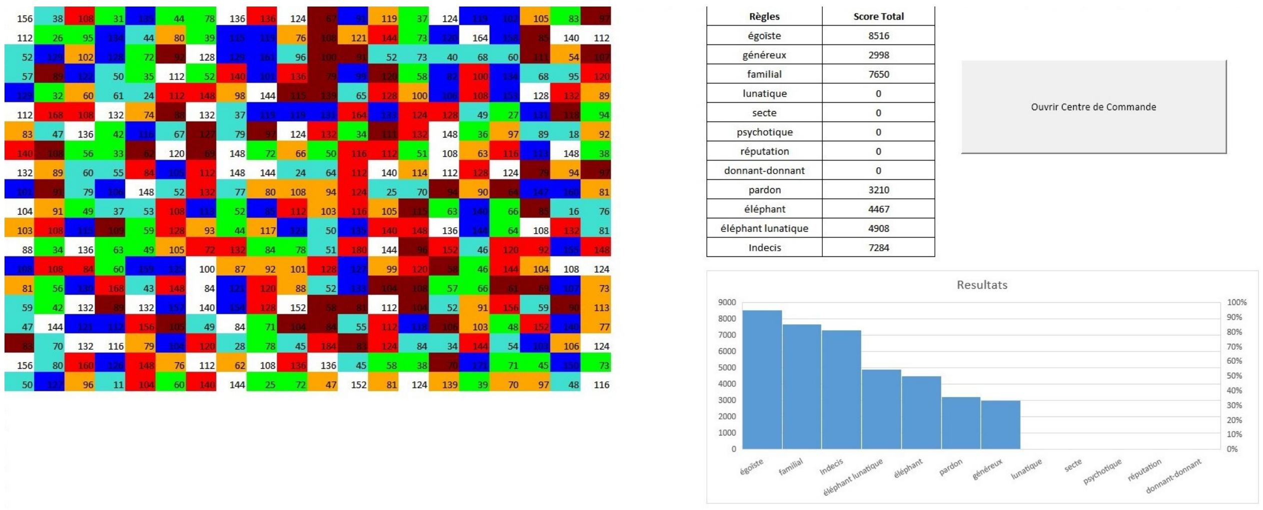
List of rules

- **Selfish**: Always defecting
- **Generous**: Cooperates systematically
- **Familiar**: Systematically cooperates with robots at a distance ≤ 2
- **Lunatic**: randomly chooses to collaborate or not
- **Sectarian**: Sect members are labeled and collaborate only with each other.
- **Psychotic**: This robot is associated with a grudge initially set at 0, which increases every time someone defaults to him. It cooperates systematically at first, then, when the grudge reaches a threshold to be set, it no longer cooperates at all. This threshold is initially set at 20 and can then be manually determined.
- **Reputation**: Each robot has a badboy score initialized at 0, which increases by 1 each time it defaults. The badboy robot systematically cooperates with robots having a badboy score below a threshold, and systematically refuses for the others. This threshold is initially set at 20 and can then be manually determined.
- **Give-and-take**: Always cooperates when meeting a robot for the first time. He then imitates the action of his opposite number when they last met.
- **Forgiveness**: Cooperates the first time, then acts in the same way as the one he met earlier.
- **Elephant**: Remembers the last ten encounters between robots. If the other has collaborated more, it collaborates; if it has defected more, it defects.
- **Lunatic Elephant**: Does the same thing, but randomly draws an action within the last ten encounters.
- ◦ **Undecided**: Alternates between cooperation and non-cooperation

2 Step-by-step and Manual

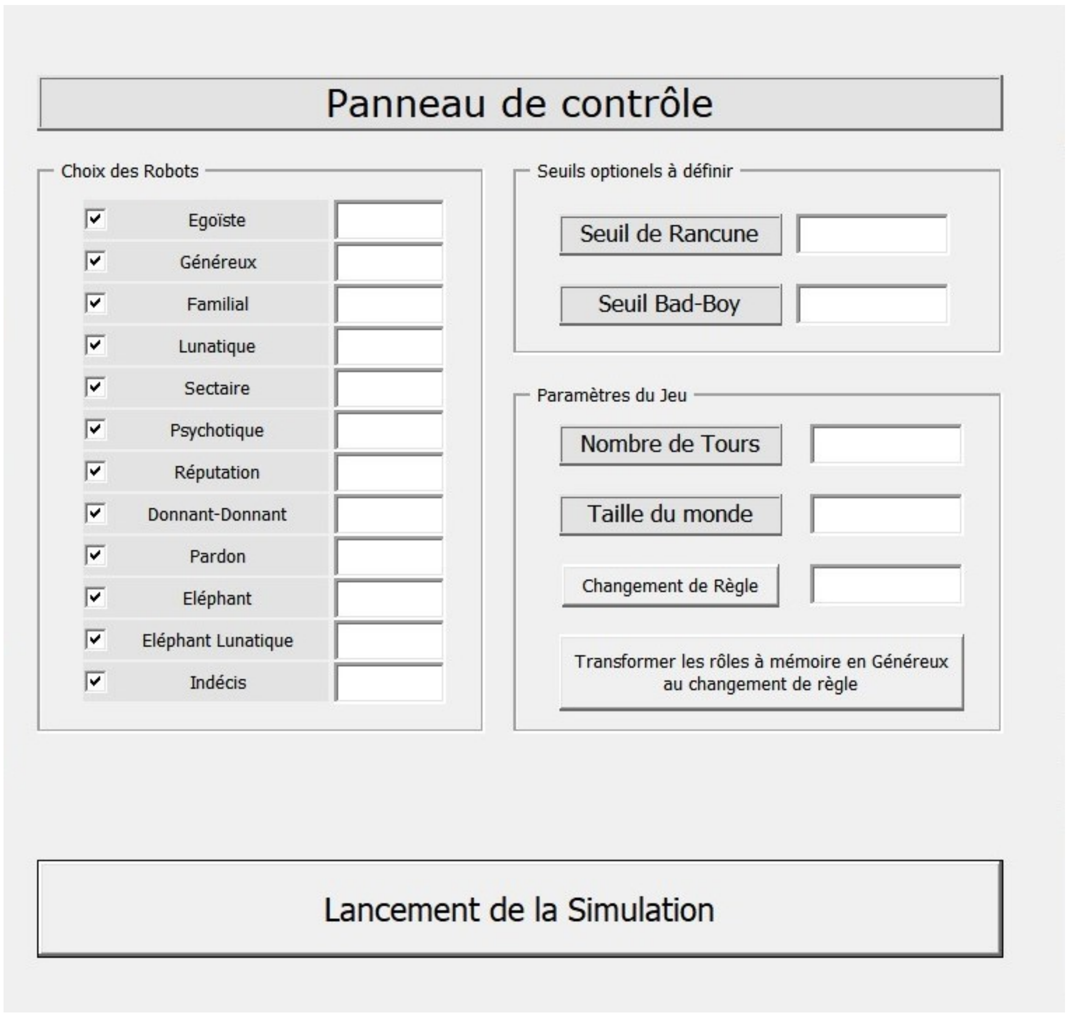
Here is a protocol outlining the steps involved in running the program:

1. Open file Projet informatique_23_24-Coopération-LEMOINE_SOLON.xlsm
2. Check that developer mode is enabled on Excel and that macros are not blocked.
3. Once the file is open, click on the large "Open Command Center" button, and an interface should appear.



Règles	Score Total
égoiste	8516
généreux	2998
familial	7650
lunatique	0
secte	0
psychotique	0
réputation	0
donnant-donnant	0
pardon	3210
éléphant	4467
éléphant lunatique	4908
Indecis	7284

Figure 1 - *Appearance of the Excel worksheet when open*



Panneau de contrôle

Choix des Robots

- ☑ Egoïste
- ☑ Généreux
- ☑ Familial
- ☑ Lunatique
- ☑ Sectaire
- ☑ Psychotique
- ☑ Réputation
- ☑ Donnant-Donnant
- ☑ Pardon
- ☑ Eléphant
- ☑ Eléphant Lunatique
- ☑ Indécis

Seuils optionels à définir

- Seuil de Rancune
- Seuil Bad-Boy

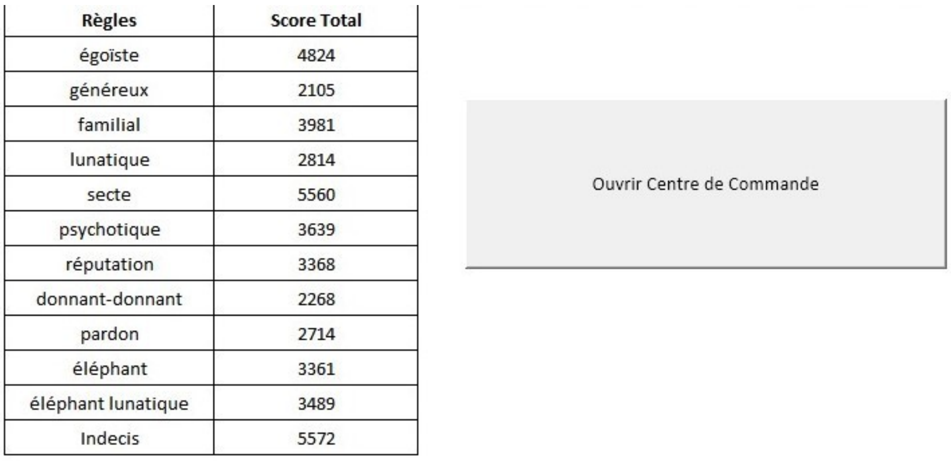
Paramètres du Jeu

- Nombre de Tours
- Taille du monde
- Changement de Règle
- Transformer les rôles à mémoire en Généreux au changement de règle

Lancement de la Simulation

Figure 2 - *Control panel from which simulations are launched*

4. To start the simulation, press "Start simulation". This will launch a simulation with the default parameters, which are listed in the table below. Once the program has finished running, you can visualize the results by looking at the colored grid and viewing the graph which tallies the scores for each rule.



Règles	Score Total
égoïste	4824
généreux	2105
familial	3981
lunatique	2814
secte	5560
psychotique	3639
réputation	3368
donnant-donnant	2268
pardon	2714
éléphant	3361
éléphant lunatique	3489
Indecis	5572

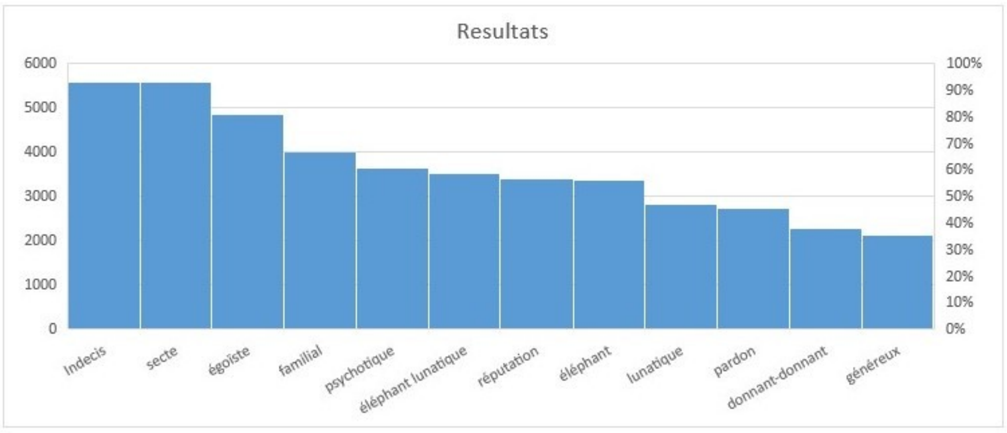


Figure 3 - *Control panel from which simulations are launched*

5. You can change the simulation parameters. To select the roles present, simply check or uncheck them. The input field on the right is used to define the distribution of roles. The probability of each role being selected is defined as follows:

$$\mathbf{P}(\text{Role}) = \frac{\omega}{N}$$

Where ω represents the probability multiplier chosen by the user to be entered in the input field to the right of the role and N the total number of roles present in the simulation. If nothing or a non-integer value is entered, the default value is taken.

6. For the reputation and psychotic roles, you can set the associated thresholds in the appropriate input fields. If nothing or a non-integer value is entered, the default value is taken.

7. The "Number of turns" parameter sets the number of simulation iterations, which is the minimum number of interactions a robot will have during the simulation. If nothing or a non-integer value is entered, the default value is taken.
8. The "Change Rule" button, if pressed, changes the rule of all robots with a score below a certain threshold to the one with the best score after a selectable number of rounds.

$$\text{Threshold} = \text{Max} - \frac{\text{Max - Min}}{4}$$

   Max and Min represent a robot's best score and worst score respectively. If nothing or a non-integer value is entered, the default value is taken.
9. The "Turn memory roles into Generous on rule change" button. Does exactly what it says. It only works if rule change is enabled. *It was noticed rather late that the implementation of this feature was the result of a misunderstanding of the statement.*

Parameters	Default values
Grudge threshold	20
Bad-Boy threshold	20
Presence of a role	TRUE
Probability of being chosen	$\frac{1}{n}$
World size	20 ×20
Number of Towers	40
Rule change	FALSE
Number of rounds at which rule change occurs	40
Transforming memory roles into rule-change generators	FALSE

Table 1 - *Default parameters for simulations*

3 How the Algorithm works

Written description

General structure

After looking at the step-by-step approach, we'll try to understand the program from an overview point of view. The program is made up of two main scripts, the one that codes the interface, from which the constants and parameters of the simulation are chosen and defined as global variables, and the main script that takes these parameters as input to run the simulation, structured as a big loop that runs through all the iterations and all the robots. When the simulation is complete, the results are calculated and the spreadsheet is updated to display them.

Variable declaration

The properties of each robot, such as its score and role, are stored in arrays of the same size as the world, so that each robot can be identified by its coordinates (*i, j*) on the array. For memory-based roles, there are arrays which themselves contain collections listing the robots encountered and their last ten actions. When necessary, functions can browse these arrays of collections to check whether two robots have met and return their past actions.

How simulation works

Once the parameters have been defined and the simulation run, roles are assigned, either randomly or weighted, according to what the user has decided. Each role is identified by the program by an integer ranging from 0 to 11, and can be identified by the user by its color. Once the roles have been defined, for all iterations, each robot makes the choice, stored as a Boolean and according to the rules of its role, to cooperate with or defect from another robot drawn at random, without knowing the latter's choice, which is made in the process. Once both have made their choice, properties (memory, reputation, grudge, etc.) are updated and points are distributed to both robots according to the outcome of the encounter. If the rule change option is activated, and the associated iteration is reached, then roles with a score below the threshold are replaced by the role with the best score. When all iterations (number of rounds) have been completed, the scores for each robot are calculated, grouped by role and displayed on the spreadsheet.

Table	Data type	Information contained
scores(i,j)	Integral	Score for each robot
grid(i,j)	Integral	Robot type index
sectMember(i,j)	Boolean	The robot's sect membership
grudge(i,j)	Integral	Robot grudge score
badboyscore(i,j)	Integral	Robot reputation score
indecis(i,j)	Boolean	Previous action of the undecided
met_robot_names(i,j)	Collection	Names of robots encountered
met_robot_names(i,j,k)	Collection	Robot actions

Table 2 - *Main tables used in the memory simulation*

A general diagram of how the algorithm works is shown below.

General diagram of how the algorithm works

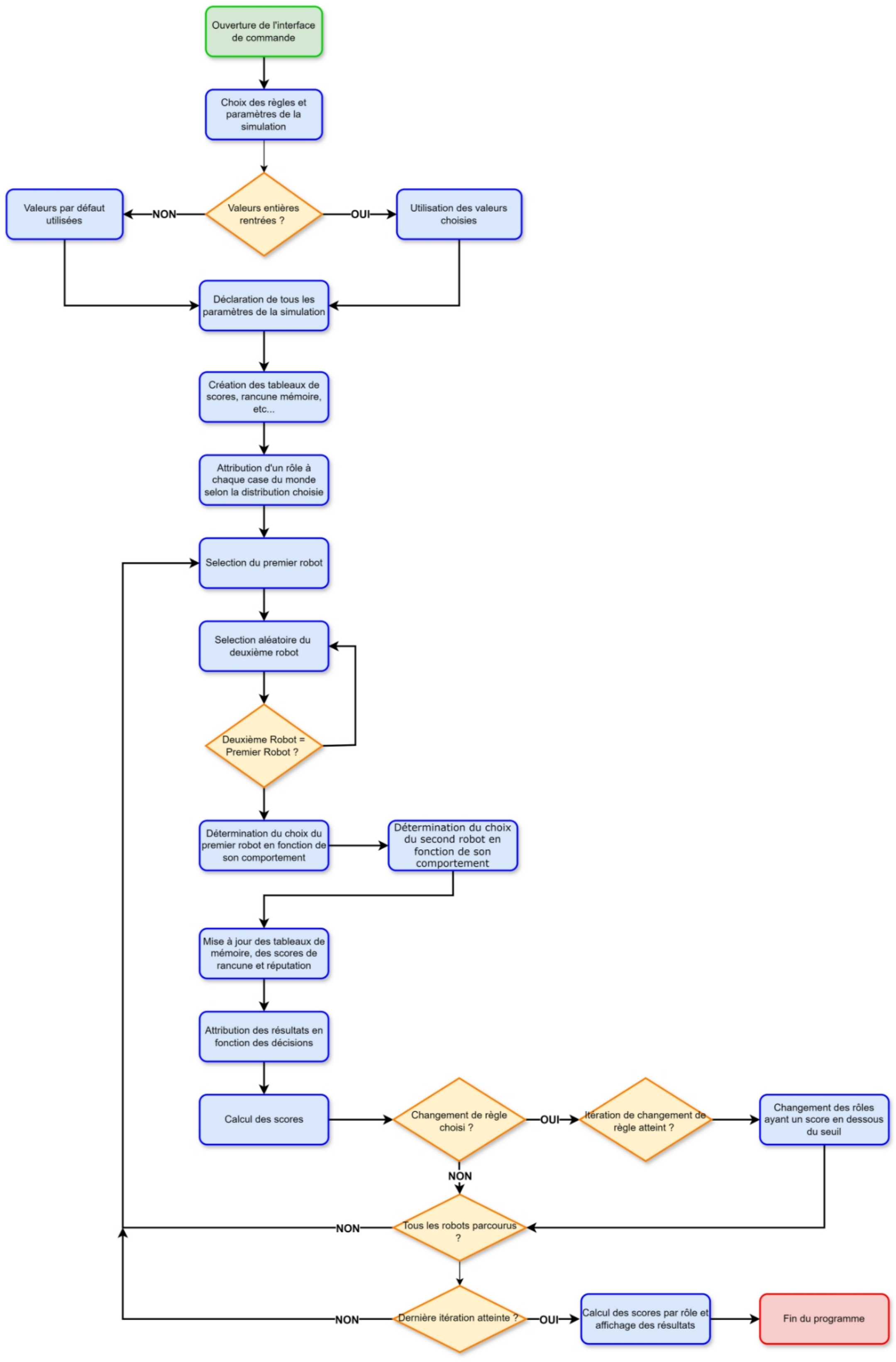

Ouverture de l'interface de commande
Choix des règles et paramètres de la simulation
Valeurs entières rentrées ?
NON
OUI
Valeurs par défaut utilisées
Utilisation des valeurs choisies
Déclaration de tous les paramètres de la simulation
Création des tableaux de scores, rancune mémoire, etc...
Attribution d'un rôle à chaque case du monde selon la distribution choisie
Selection du premier robot
Selection aléatoire du deuxième robot
Deuxième Robot = Premier Robot ?
Détermination du choix du premier robot en fonction de son comportement
Détermination du choix du second robot en fonction de son comportement
Mise à jour des tableaux de mémoire, des scores de rancune et réputation
Attribution des résultats en fonction des décisions
Calcul des scores
Changement de règle choisi ?
OUI
Itération de changement de règle atteint ?
Changement des rôles ayant un score en dessous du seuil
NON
Tous les robots parcourus ?
NON
Dernière itération atteinte ?
NON
OUI
Calcul des scores par rôle et affichage des résultats
Fin du programme

Figure 4 - *Algorithm flow diagram*

4 Results and observations

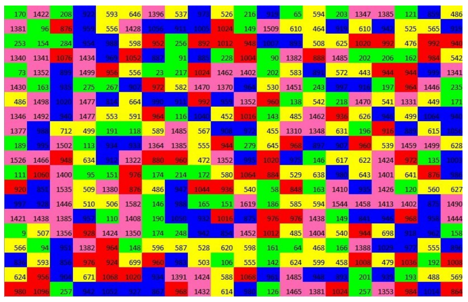
There are too many parameters in the algorithm to be able to reasonably test all cases, but a few general observations can be made, most of which will be made with the default parameters.

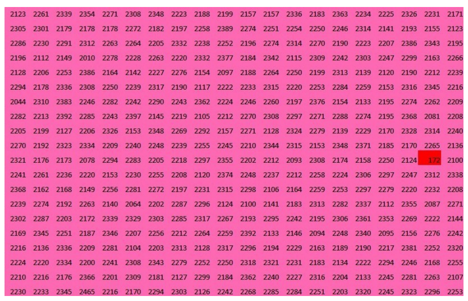
General comments

When you run the simulation with the default settings, you'll find that the sectarian, the indecisive and the selfish most often top the scores. The worst scores are often attributed to the generous, the give-and-take and the forgiving. These low scores for memory roles can be explained by the fact that the relatively low number of rounds selected by default (40) doesn't allow them to use their memory sufficiently, since statistically most of the encounters they make will be one-offs. These robots therefore resemble the behavior of the generous. If we try to limit ourselves to simple roles, then the leading roles are the Generous and the Family. Overall, the game seems to favor non-cooperation.

Influence of the rule change

If we try to keep only the simple roles (generous, family, selfish, moody and sectarian) and introduce a rule change after 40 turns for 400 iterations, we notice that the sectarian is almost totally victorious, with the entire grid except for a few cells becoming sectarian.


(a) Results without rule change


(b) Results with rule change

Generally speaking, the winning role, if not the sectarian, ends up occupying virtually every cell when the rule change is decided.

Influence of world size

The first visible effect of world size is on program performance. When the size of the world is reduced, the program finishes much more quickly, in - installment the larger the world, the poorer the performance. Increasing the world size merely confirms statistically the general results already obtained.

have been observed. Greatly reducing the size of the world tends to produce less conventional results, even if the sect often remains the winner.

Influence of threshold changes

We can see that when we increase the badboy threshold, the performance of the reputation role predictably decreases. We can test this by contrasting the reputation role with the selfish role. With the default settings (Bad-Boy Threshold at 20), the reputation robot wins all the time. When the threshold is raised to 50, the reputation role becomes less distrustful, so the selfish role wins.

To test the grudge threshold, we pit the psychotic against the lunatic. With the default threshold (20), the two roles are more or less equal. If we make the psychotic more susceptible by lowering the grudge threshold to 3, the psychotic wins almost systematically.

Influence of the number of revolutions

It's interesting to look at the influence of the number of turns on memory roles, since their nature means that a higher number of turns is required to make full use of their features. Since memory roles take longer to calculate, we reduce the size of the world to 15×15. Pitting the egoist against the giver, the egoist wins up to around 200 turns, until the giver wins. The same trend can be observed with the forgiving role, except that it takes even more turns for the forgiving role to start winning (around 250). Similar trends can be observed with the elephant roles, except that it takes fewer turns for them to win against the egoist (100).

Influence of proportion change

Unsurprisingly, for all roles, as you increase the proportion of roles, they are more likely to have better scores. For example, with all the rules selected, multiplying the probability of the generous one being chosen by three is enough to make him win.

5 Conclusions

Our program enabled us to put game theory into practice through a generalized simulation of the prisoner's dilemma with N individuals. To make experimentation easier, we added a graphical interface. It's difficult to draw definitive conclusions, given the number of parameters and scenarios to be tested, but our program enabled us to carry out a social experiment without the need to interview anyone human. It's hardly surprising, then, to learn that game theory has been the subject of more rigorous study.