Multi-agents Ultimatum Game with Reinforcement Learning

Tangui Le Gléau, Xavier Marjou, Tayeb Lemlouma, Benoit Radier

Multi-Agents Ultimatum Game with Reinforcement Learning

Tangui Le Gléau[1,2], Xavier Marjou[1], Tayeb Lemlouma[2], and Benoit Radier[1]

[1] Orange Labs
{tangui.legleau,xavier.marjou,benoit.radier}@orange.com
[2] IRISA
tayeb.lemlouma@irisa.fr

Abstract. *The Ultimatum Game* is an experimental economics game in which an agent has to propose a sharing partition of a limited amount of resources to other agents who have to accept it or not. If the offer is rejected per consensus, the process of sharing is abandoned. So all agents have to guess what are the best decisions (offer and vote) to optimise their respective gain. We focus on an iterated multi-agent version of Ultimatum Game also known as the Pirate Game, a riddle in which pirates have to share coins according to specific rules. To solve such game, we employ a multi-agent model. In particular, we design a new kind of Artificial Neural Network model able to output an integer partition of discrete finite resources, trained by a Reinforcement Learning agent to identify an acceptable offer to the voting agents. We take an interest in evaluating the performances against several kinds of voting behaviours. The results are close to theoretical optima for all tested scenarios thus demonstrating the flexibility of our method.

Keywords: Multi-Agents Systems · Ultimatum Game · Reinforcement Learning · Game Theory

1 Introduction

Numerous use-cases involve resource sharing between multiple actors in multi-agents systems (MAS) where the involved actors try to gain a common quantity of interest and reach consensus [1]. For instance, in the context of telecommunication, one given Mobile Network Operator (MNO) can share 5G network resources with vertical industries [2], thereby allowing them to request the reservation of a network slice in the MNO infrastructure [3].

As modern networks (e.g. IoT and UAV) become more decentralized and autonomous, network agents entities often incorporate Artificial Intelligence (AI) strategies to achieve local decisions and maximize the network performance [4]. Some approaches like [5] propose to solve such network resource management with the application of the Reinforcement Learning (RL) approach [6], which is a trial and error process where an agent can identify the best sequence of actions to maximise its cumulative reward. In such approaches, resources management

depends solely on the MNO sharing of its infrastructure. In several use-cases, the demanding agents can accept or refuse an offer (i.e. a sharing distribution), which is a critical assumption for the MNO in case the total demand exceeds its available resource capacity.

Obviously, for an offerer agent, the design of sharing strategies is crucial as failing to satisfy the demands of other agents could result in excluding the offerer from the business. For dynamic and unpredictable environments (e.g. in hybrid human-agent groups [7]), it is extremely difficult to completely specify the offerer strategy at the time of its design and before runtime applications [8]. Indeed, the design of a MAS based on the understanding of actions and interactions within artificial agents remains a challenging task. In this respect, the Ultimatum Game (UG) paradigm can be considered as a viable candidate to capture such complex interactions in a dynamic and heterogeneous environment [7]. In this context, we investigate a multi-agent version of UG.

UG had already been studied in particular with a multi-agent setting [9–13]. In this paper, we tackle the problem of UG with discrete finite resources which is a rather interesting hypothesis of the game. We focus on the application of a Reinforcement Learning approach to model the behaviour of a given proposer (i.e. offerer) and voter agents in the multi-agents version of UG with discrete resources.

From a more complex and more attractive perspective, we focus on a particularly interesting version of the UG, which is inspired by a curious riddle called the *Pirate Game* where several agents are hierarchically ranked. The principle of this riddle is as follows: n_P pirates have to share n_C coins. There is a hierarchy among pirates: $P_1, P_2, ..., P_{n_P}$ and the proposer is the Highest Rank Pirate (HRP) who initiates the offer of coins partition. Then, all pirates (including the proposer) vote for or against the partition. If the partition is accepted, the game is over and the coins are distributed; if it is rejected, the HRP is eliminated and the next HRP has to propose a new partition; and so forth. It is worth noting that all the pirates are considered as rational players (meaning that they prefer one coin rather than zero regardless of any feeling of injustice, fairness, etc.). Moreover, each pirate knows his rank in the hierarchy and prioritises to survive then to maximise its profit. The PG can be summarised as an iterated game of maximum n_P rounds in each of which the remaining pirates play an ultimatum game. The rationality of agents leads to one interesting equilibrium that is demonstrated in appendix A. The Pirate Game (PG) presents many motivating properties as in many real-world applications, it involves multiple agents (the pirates) trying to share a pool of common finite resources (the coins); it has a well-defined set of rules; it benefits from a mathematical modelling, which is a perfect baseline for evaluating the obtained results. In this work, in order to model such MAS, we propose an Artificial Neural Network (ANN) base model able to output a discrete partition. Thanks to an Reinforcement Learning (RL) algorithm (REINFORCE algorithm described of Section 2), the ANN is trained to have the highest chances to generate a partition that will be accepted by most rational players involved in the sharing.

The paper is structured as follows: after a brief background on Reinforcement Learning methods (section 2), we present our model in section 3, in particular, a method to generate a stochastic integer partition. Then, after introducing some scenarios of behaviours that we experiment (section 4), we show the results of the RL agent learning in section 5. Finally, we discuss our results and provide some perspectives in section 6.

2 Background on Reinforcement Learning

RL is a trial and error process that generates an optimal policy $\pi(a|s)$ which relates the best action a to a current state s (each action delivers a single reward r_t) so that the cumulated reward R_t is maximal.

2.1 Value-based methods

A popular category of value-based algorithms is *Temporal-Difference* learning (TD-Learning) [6] which consists, for each state s, to evaluate a function value $V(s)$ representing the reward expectancy from the state s and according to a policy π: $V(s) = \mathbb{E}[R_t|s_t = s]$. The principle of TD-learning is to evaluate the mean of value at each state. The update is computed at each step t of episodes:

$$V(s_t) \leftarrow V(s_t) + \alpha(r_{t+1} + \gamma V(s_{t+1}) - V(s_t))$$

where $0 < \alpha < 1$ is a learning rate and $0 < \gamma \leq 1$ is a discount-rate which attenuates the importance of future rewards.

2.2 Policy-based methods

The core idea of such methods is to directly determine a policy that relates the optimal actions to the states (e.g. REINFORCE algorithm [14]). The principle is to find an optimal policy $\pi^*(a|s)$ modelised with parameters θ: π_θ, the problem consists in finding optimal parameters θ^* such as:

$$\theta^* = \underset{\theta}{\operatorname{argmax}}\ J(\theta), \quad \text{where } J(\theta) = \sum_{\tau} P(\tau|\theta)R(\tau)$$

with τ a sequence of states and actions (called a trajectory) and $R(\tau)$ the cumulative reward of trajectory τ. Then, the gradient is computed:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log(\pi_\theta(\tau_i))R(\tau_i)$$

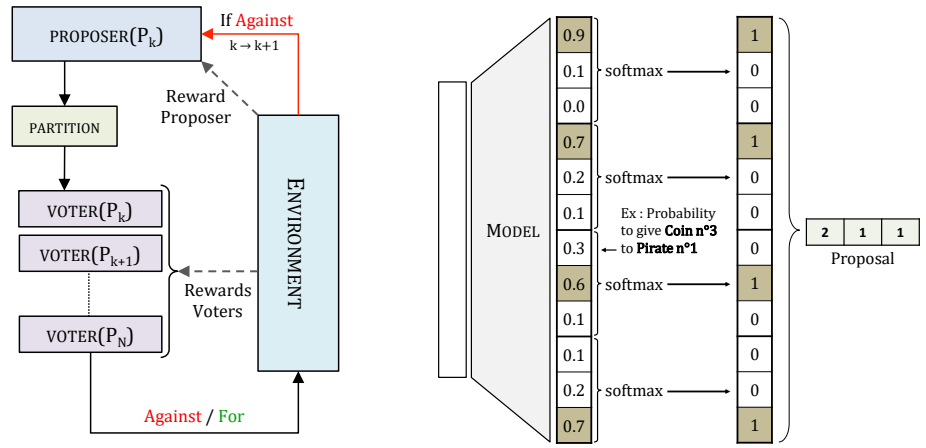
Then, the algorithm consists of sampling trajectories thanks to current stochastic policy π_θ in order to compute the gradient $\nabla_\theta J(\theta)$ to finally update the parameters with gradient descent.

3 Model

As presented previously at the outset of this paper, we implement a model using RL to solve the riddle. This means that our proposed model learns the best decisions to fulfil priorities induced by the game rules: survive first, then earn coins.

3.1 Players

In order to solve the PG with our proposed model, we introduce two functions: a *proposer* function to propose a partition and a *voter* function allowing each agent to vote for or against the offer. Both functions are shared by all pirates because every pirate's goal is to find out his optimal behaviour whatever his rank. Then, at any time during the learning process, any pirate is able to compute an offer according to his hierarchical situation or vote according to the proposed partition, the proposer's rank and his own.



(a) The models of proposer and voter are trained together toward the equilibrium

(b) Model of Proposer: neural network outputting a distribution of integer partition ($n_P = 3,\ n_C = 4$)

Fig. 1: Architecture of algorithm

Proposer As mentioned above, the *proposer* function is generic, therefore it takes as input at least the rank of the proposer. The use of a neural network which is a priori not necessary is motivated by the possibility to use a more complex input to represent the state (for example, the history of previous behaviours

using recurrent hidden state or communication protocols between agents of different hierarchy, see Section 6). The output is an integer partition describing the number of coins n_C with a maximum number of pirates n_P terms. Several techniques can be considered: first, we can use a network output of n_P neurons in each of which there would be the number of coins assigned to each pirate. The problem of this method is that neurons values must be integers where the total sum is a specific integer equals to the number of resources. Hence, there are major constraints which are technically difficult to impose. Another possibility would be to normalise the output of n_P neurons, multiply it by n_C and take the closest integer for each neuron. Unfortunately, this method can lead to an issue of rounded values as the sum would differ from expected n_C with a difference up to 50% in the worst cases, which would be difficult to adjust. To overcome this problem, we propose a new encoding of the integer partition. We use an output Y of $n_P n_C$ neurons, and we apply one softmax every n_P neurons in such a way that we obtain n_C vectors of probabilities (of n_P values). We can formally link the output Y and $f(c,p)$ the probability to give coin c to pirate p :

$$Y[i] = f(\lfloor \frac{i}{n_P} \rfloor,\ i \bmod n_P) \Leftrightarrow f(c,p) = Y[n_P c + p]$$

Then, the partition P of integer n_C with n_P terms can be formally written as follows:

$$P[i] = Card\{c,\ i = \underset{x}{\operatorname{argmax}}\ f(c,x)\} \quad \forall i \in [0, n_P - 1]$$

The proposed model ensures that every coin is attributed to one and only one pirate, which is the definition of a partition. Moreover, the model provides stochasticity which is interesting for training and allows to use the REINFORCE algorithm.

Figure 1b depicts an example of the model used for the partition proposal when $n_C = 4$ coins and $n_P = 3$ pirates.

Voters To model and train the voter function, we use a value-based algorithm (simple Monte-Carlo) which evaluates the mean score of coins for each state proposer id and voter id). Thanks to this baseline, each voter can compare the number of coins proposed to the number of coins he could get in the next step. Then he can choose to accept or to reject the offer. When at least 50% of the voters accept the offer, the offer is declared accepted, otherwise, the offer is declared rejected.

3.2 Training

Both the proposer and voter functions are trained simultaneously. For the proposer, we use the REINFORCE algorithm with deep learning. The environment provides a reward of $+\exp(c)$ with c the number of coins when the offer is accepted and a reward of -1 for death (elimination) when the offer is rejected. We use the exponential function to insist on the greediness of pirates, which somehow corresponds to an increasing marginal utility.

4 Experimented scenarios

In this section, we present the scenarios we want to study in addition to the Pirate Game (version with hierarchical ranks as mentioned in the introduction). For reproducibility, the code has been made available[1].

4.1 Selfish and rational vote

In this scenario, all pirates vote according to the rules of the original PG [15]. Each pirate is purely rational and only considers its interests, hence voting selfishly. As mentioned in the introduction, the rationality of agents leads to an equilibrium (demonstrated in appendix A).

4.2 Prosocial votes

In this scenario, we ignore the assumption of rationality. Each pirate (except the proposer who remains selfish) votes for a proposal if and only if the partition looks equally fair to all pirates, otherwise votes against it. Hence, the pirates' vote becomes a prosocial vote.

To this purpose, the voters decide according to the result of a Jain's index calculation [16]. If a system allocates resources to n contenting users, such that the i^{th} user receives an allocation x_i, then the index for the system is:

$$J(x_1, x_2, ...x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{(n \sum_{i=1}^{n} x_i^2)}$$

When this index approaches 1, a distribution is considered equally fair to all parties; when this index approaches $1/n$, a distribution is considered completely unfair. For our work, we decided arbitrarily that when the Jain's index of the distribution proposal is equal to the theoretical maximum possible Jain's index (provided in appendix B), a voter accepts the proposed partition, otherwise it votes against it. We can easily guess that the optimal strategy will converge towards a partition where each integer of coins distribution will differ from maximum one coin.

4.3 Prosocial vote with partial observation

In this scenario, we extend the previous scenario with a partial observation of the proposal: we suppose that each pirate knows only what he received. So the proposer who has to guarantee only a majority of votes can adapt more easily his offer.

[1] `https://github.com/tlgleo/Pirate_Riddle/blob/master/pirates_riddle.ipynb`

5 Results

For each behaviour of voters, our REINFORCE training algorithm provides the proposer with a policy which is a probabilities distribution over possible partitions.

The results are presented using tables displaying the number of coins received by each pirate according to his rank and the number of living pirates. The upper row of a table represents the distribution when n_P pirates are alive with the first pirate P_1 being the HRP. The second row represents the distribution when $n_P - 1$ pirates are alive with P_2 being the HRP; and so forth. A white bin represents a case where a HRP is eliminated, meaning that this dead pirate will not receive any coins.

The learning graphs represent two curves: the orange one is the reward of RL algorithm which is the mean number of coins received by the proposer. The blue one shows the evolution of the euclidean distance between the predicted TD-learning of voter function and theoretical solutions (when a deterministic optimal solution exists).

5.1 Selfish votes only

This sub-section presents the results obtained when all pirates vote according to the original rules of the PG. Figure 2a shows that the proposer's reward grows quickly and is close to the optimal solution as soon as the training reaches 2000 episodes[1]. As mentioned previously, the results may seem counter-intuitive but we demonstrated in appendix A that this is the best strategy for the proposer being faced with rational agents.

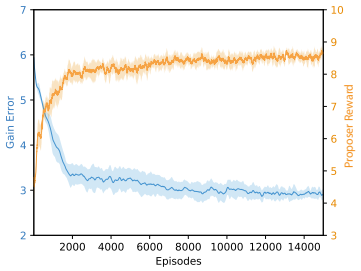
Figure 2b shows that the proposed distribution is close to the optimal solution.

5.2 Prosocial vote

In this part, we present the evaluation results obtained when only the HRP votes selfishly and when all the other agents vote prosocially by accepting the partition plan if and only if Jain's index is maximal. Figure 3b shows that the new behaviour of the voters leads to a distribution which is fairer for the different pirates.

As expected, the only cases where the selfish vote of the proposer is enough to win most resources are when there are only one or two pirates in the MAS (represented by the last two rows in 3b). In all other cases, the selfish vote of the proposer agent is thwarted by the prosocial votes of the other agents.

[1] The training of REINFORCE algorithm can be seen here: `https://youtu.be/gh4USNJVWuw`

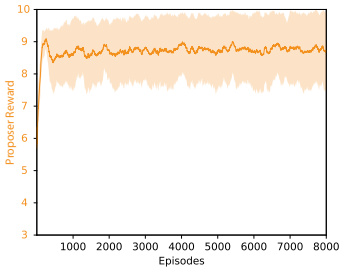


(a) Learning curves of RL agents, shaded areas represent confidence intervals (95% over 10 runs). The orange curve is the reward of the proposer and the blue one shows the voter TD-learning error (with theoretical equilibrium).

Pirate rank					
P1	P2	P3	P4	P5	
8.0±0.1	0.0±0.0	1.0±0.1	0.0±0.0	1.0±0.0	♟♟♟♟♟
	9.0±0.0	0.0±0.0	1.0±0.0	0.0±0.0	♟♟♟♟
		9.0±0.0	0.0±0.0	1.0±0.0	♟♟♟
			9,9±0.3	0,1±0.3	♟♟
				10.0±0.0	♟

(b) The number of coins for each situation (one to five players). The table shows the means and standard deviations from probabilities distribution learned by our proposer RL agent in 15 000 episodes.

Fig. 2: Results with the PG original behaviour (i.e. with purely rational players) with $n_P = 5$ pirates and $n_C = 10$ coins. Our RL agent is looking for the best probabilities distribution to earn the highest score.



(a) Learning curves

Pirate rank					
P1	P2	P3	P4	P5	
2.0±0.0	2.0±0.0	2.0±0.0	2.0±0.0	2.0±0.0	♟♟♟♟♟
	3.0±0.2	2.2±0.5	2.8±0.6	2.0±0.3	♟♟♟♟
		4.0±0.3	3.0±0.2	3.0±0.2	♟♟♟
			10.0±0.1	0.0±0.1	♟♟
				10.0±0.0	♟

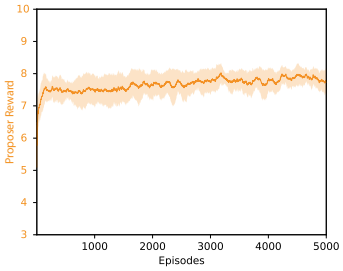
(b) Learned distribution

Fig. 3: Results in mixed mode (highest rank is selfish, the voters are prosocial with $n_P = 5$ pirates, $n_C = 10$ coins

5.3 Partial observation

In this scenario, each voter knows only his part of the plan. We can see that our agent succeeds to achieve a good strategy. The best one consists in selecting enough pirates and convince them to vote for his sharing. Since the observation is partial which means that they only know their part, the proposer only has to concentrate the resources on the minimal number of agents, for example with two of the five pirates and give the minimal number that requires the optimal value of Jain's index. Figure 4 shows the results of the distribution. Thought

the choice of voters doesn't change the optimal strategy, we can notice that the learning algorithm deterministically selects the same pirates to give them the minimum number of coins.



(a) Learning curves

Pirate rank					
P1	P2	P3	P4	P5	
5.0±0.0	3.0±0.0	2.0±0.0	0.0±0.0	0.0±0.0	♟♟♟♟♟
	8.0±0.0	2.0±0.0	0.0±0.0	0.0±0.0	♟♟♟♟
		7.0±0.0	3.0±0.1	0.0±0.1	♟♟♟
			10.0±0.0	0.0±0.0	♟♟
				10.0±0.0	♟

(b) Learned distribution

Fig. 4: Results in prosocial mode with partial observation ($n_P = 5$ pirates, $n_C = 10$ coins)

6 Discussion and Next Steps

In this section, we discuss the salient technical aspects of our proposed model and some perspectives raised.

6.1 Adaptability of the model

The empiric work shows that the same ANN model can predict near to optimal offers in different scenarios without knowing a priori the voting criteria of the other players, which demonstrates a great deal of flexibility.

The results also illustrate the divide and conquer proverb: the proposer generally obtains a bigger reward when all other agents vote selfishly than when all other agents vote prosocially. In the selfish scenario, the proposer earns $n_C - \lfloor (n_P - 1)/2 \rfloor$ coins whereas in the prosocial scenario, he wins $\lfloor n_C/n_P \rfloor \pm 1$ coins. In other words, the lesson learned by the players is that they gain more rewards by collaborating than by acting selfishly.

One of the most outstanding results is the one of Section 5.2: an HRP willing to act selfishly is forced to propose a prosocial offer when two or more lower-rank pirates vote prosocially, otherwise the offer is rejected. In other words, the adaptability of the model also leads to a high influenceability.

6.2 Communication between players

In the experimented scenarios, the agents do not communicate and their behaviour is assumed to be purely rational. This means that the decisions of agents

are based on their unique personal interest without communicating with other agents before voting. Thanks to this rationality, their behaviour can be predicted therefore allowing proposers to adapt the offers. It could be interesting to introduce a communication canal directly linked to a specific action such as voting against the next offer. This canal should be binary and would be used as an implicit message addressed to a specific agent. The agent would learn to interpret the received message and then change his vote to encourage the proposer to modify his offer. For this purpose, it would be interesting to use an LSTM layer to take into account the previous messages [17, 18] and observe how the agents adapt their offers. As described in [19], introducing communication can leads to implicit collaboration between agents.

7 Conclusion

In this work, we addressed a multi-agent version of Ultimatum Game, an economics game in which a proposer offers a partition of resources and voters accept it or not. In the version called the Pirate Game, the players are hierarchically ranked. If the vote is rejected, the proposer is eliminated from the game leaving his role to his successor. We showed that a Reinforcement Learning agent can solve such UG, which means that the agent gains the ability to decide optimally when processing a given offer of resources sharing so it earns enough and stays "alive".

For technical conclusions, we succeeded to implement a learning algorithm able to solve a discrete problem where the solution is represented by an integer partition which is not trivial. This same proposed approach can address similar real-world problems such as resource affectation, allocation, sharing, etc. Finally, we showed empirically that the reinforcement learning paradigm was a robust mean to address these problems. Specifically, it converges to an equilibrium even in a case where several adversary agents try to earn payoff with a different hierarchical situation.

References

1. L. Ding, Q. Han, X. Ge, and X. Zhang. An Overview of Recent Advances in Event-Triggered Consensus of Multiagent Systems. *IEEE Transactions on Cybernetics*, 48(4):1110–1123, April 2018.
2. White Paper 5G for Connected Industries and Automation. *5G ACIA*, 2nd edition, 2 2019.
3. White paper etsi gana model in 5g network slicing: Programmable monitoring, gana knowledge planes. *ETSI*, 2019.
4. N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y. Liang, and D. I. Kim. Applications of Deep Reinforcement Learning in Communications and Networking: A Survey. *IEEE Communications Surveys Tutorials*, 21(4):3133–3174, Fourthquarter 2019.
5. Rongpeng Li, Zhifeng Zhao, Qi Sun, I Chih-Lin, Chenyang Yang, Xianfu Chen, Minjian Zhao, and Honggang Zhang. Deep reinforcement learning for resource management in network slicing. *IEEE Access*, 6:74429–74441, 2018.

6. Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. 2018.
7. Fernando P. Santos, Jorge M. Pacheco, Ana Paiva, and Francisco C. Santos. Evolution of Collective Fairness in Hybrid Populations of Humans and Agents. In *AAAI Technical Track: Multiagent Systems*, pages 6146–6153, 2019.
8. Sandip Sen and Gerhard Weiss. "learning in multiagent systems. In *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, pages 259–298, 199.
9. Steven De Jong, Karl Tuyls, and Katja Verbeeck. Artificial agents learning human fairness. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, pages 863–870. International Foundation for Autonomous Agents and Multiagent Systems, 2008.
10. Yu-Han Chang and Rajiv Maheswaran. The social ultimatum game and adaptive agents. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pages 1313–1314, 2011.
11. Steven De Jong, Simon Uyttendaele, and Karl Tuyls. Learning to reach agreement in a continuous ultimatum game. *Journal of Artificial Intelligence Research*, 33:551–574, 2008.
12. Fang Zhong, Steven O Kimbrough, and DJ Wu. Cooperative agent systems: Artificial agents play the ultimatum game. In *Proceedings of the 35th Annual Hawaii International Conference on System Sciences*, pages 2207–2215. IEEE, 2002.
13. Fernando P Santos, Francisco C Santos, Francisco Melo, Ana Paiva, and Jorge M Pacheco. Learning to be fair in multiplayer ultimatum games. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 1381–1382, 2016.
14. Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, May 1992.
15. Ian Stewart. A puzzle for pirates. *Scientific American*, 280(5):98–99, 1999.
16. Rajendra K. Jain, Dah-Ming W. Chiu, and William R. Hawe. A Quantitative Measure Of Fairness And Discrimination For Resource Allocation In Shared Computer Systems. Technical report, Digital Equipment Corporation, September 1984.
17. Jakob Foerster, Ioannis Alexandros Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2137–2145, 2016.
18. Jakob N Foerster, Yannis M Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate to solve riddles with deep distributed recurrent q-networks. *arXiv preprint arXiv:1602.02672*, 2016.
19. Kris Cao, Angeliki Lazaridou, Marc Lanctot, Joel Z. Leibo, Karl Tuyls, and Stephen Clark. Emergent communication through negotiation. *CoRR*, abs/1804.03980, 2018.

A Proof of Optimal Selfish partition

For n_P pirates and n_C coins, the optimal proposal P following the game rules and satisfying the rational priorities of pirates is:

$$P_{n_P}[i] = \begin{cases} n_C - \lfloor (n_P - 1)/2 \rfloor, & \text{if } i = 0 \\ 1, & \text{if } i = 2(k+1) \\ 0, & \text{if } i = 2k+1 \end{cases}$$

with $\lfloor \ \rfloor$ denoting the floor function.

We can show this by backward induction on the number of pirates n_P:

- **Base Case**: for $n_P = 1$, there is only one possible partition: $P[0] = n_C$
- **Induction step**: let us assume the assertion is valid for n_P, let us show the assertion is valid for $n_P + 1$.
  The proposer (decision maker) who will naturally vote for himself needs to convince only $\lfloor n_P/2 \rfloor$ other pirates to guarantee the majority of acceptance ie his survival. Then to fulfil the priority of maximal gain, it is optimal to choose the $\lfloor n_P/2 \rfloor$ lowest gains in the partition P_{n_P} and to offer them only one coin more (this will convince them thanks to their rationality) and offer zero coins to others. In the partition P_{n_P}, there are $\lfloor n_P/2 \rfloor$ null gains, so we choose to offer one coin to them and zero to others. In the new partition P_{n_P+1}, it doesn't change the explicitation since the incrementation of index (highest rank is always 0) compensates the parity modification. Finally, the decision maker takes for him the remaining coins: $P_{n_P+1}[0] = n_C - \lfloor n_P/2 \rfloor$. Therefore, the propriety is verified for $n_P + 1$.

B Maximal Jain's index of an integer partition

We can show that the maximal value of the Jain's index of any partition of the integer n_C in n_P terms is given by:

$$\overline{J(n_P, n_C)} = \left[1 + \frac{d(1-d)}{x^2}\right]^{-1}$$

where

- x is the ratio $x = n_C/n_P$
- d the decimal part of x: $d = x - \lfloor x \rfloor$

When n_C is divisible by n_P, $d = 0$, so $\overline{J(n_P, n_C)} = 1$. If not, $\overline{J(n_P, n_C)} < 1$. For example, $\overline{J(4, 10)} \approx 0.962$ with for example the optimal partition [4 4 3 3] .