Machine Learning Project

Report 1

Yannou Ravoet, r0637112

Robrecht Peeters, r0627461

2019 – 2020

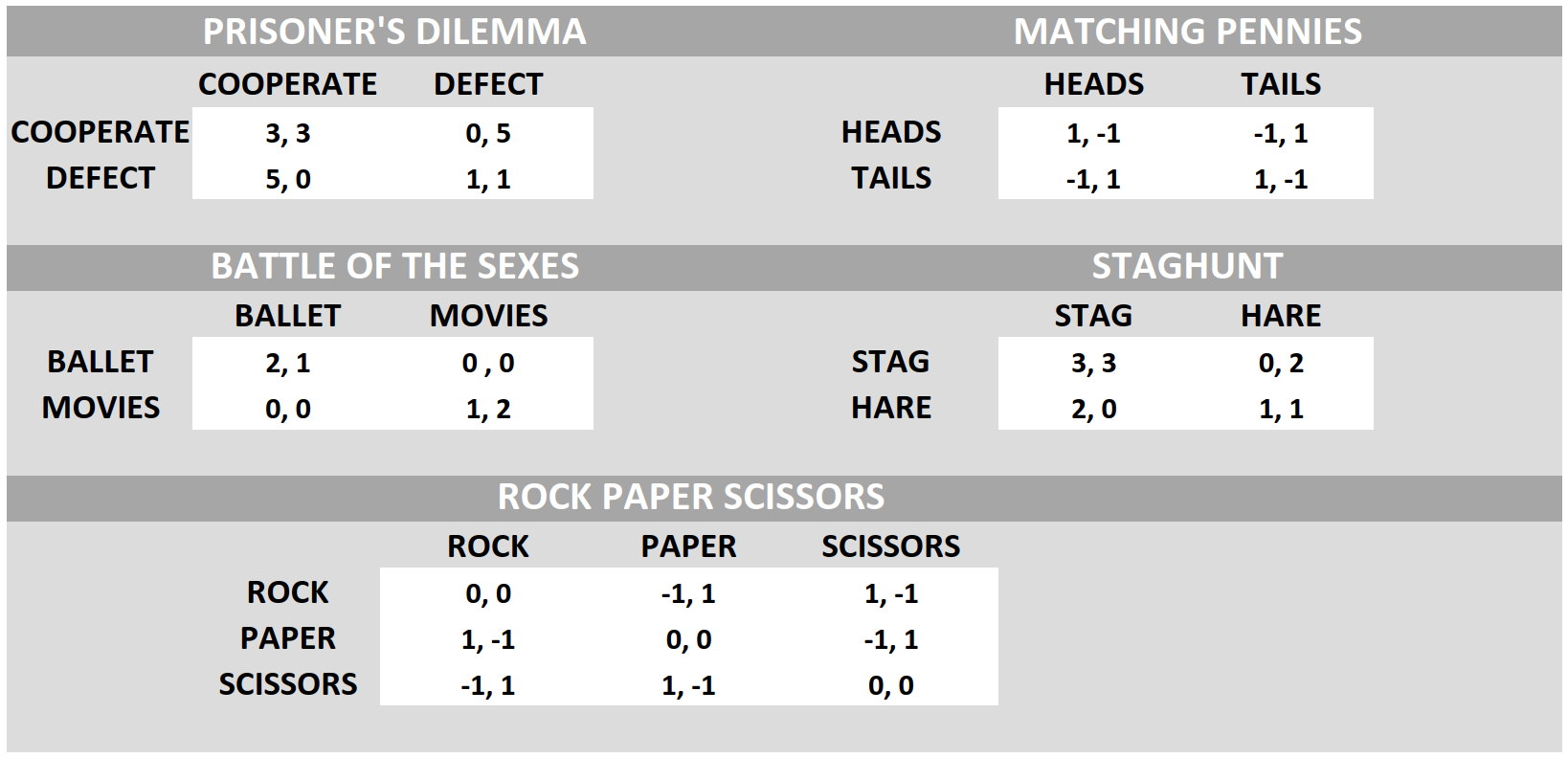
# 1 Introduction

In this report we take our first steps to combining the fields of Reinforcement Learning and Game Theory in a Multi-Agent setting. We look at the matrix games known as the prisoner's dilemma, matching pennies game, battle of the sexes, rock paper scissors and as a bonus the stag hunt. The payoff matrices for each game are visualized in Figure 1. In section 2 we discuss the Nash equilibria and Pareto optimal states for each game. Section 3 discusses the reinforcement learning part of the project. After discussing the used algorithms, the implementation of each algorithm is briefly discussed as well as their application on the games. Section 4 summarizes the literature that was used to research and implement the different game theory and reinforcement learning concepts and algorithms. Section 5 draws a conclusion and Section 6 makes an estimate as to which algorithms might be interesting for the next challenge: Kuhn and Leduc Poker.

# 2 The games

A ***Nash equilibrium*** occurs when both players chose the best response given that they know the action of the other players. In other words, none of the players can react better given that the other players do not change their strategy. We make a distinction between a pure strategy Nash equilibrium, wherein each player chooses a single action deterministically, and a mixed strategy Nash equilibrium where (at least one of) the players choose a set of actions with probabilities between 0 and 1.

A state (a selection of actions for each player) is ***Pareto optimal*** if none of the players can receive a better reward without diminishing another player’s reward. This can be seen as a state in which a player could convince all other players to change state (since the new state is either better or equally as good for them).



**Figure 1:** Payoff matrices for all games discussed in the report

In the prisoner's dilemma there is only 1 Nash equilibrium: both players ‘defect’. The state in which both players ‘cooperate’ is not a Nash equilibrium, even though it is a Pareto Optimal state, since each player can improve his reward given that the other player keeps ‘cooperating’. There is thus but 1 Nash equilibrium.

In the matching pennies game, we are dealing with a competitive zero-sum game where one player’s increase in reward corresponds to the other player’s decrease in reward. There is only a mixed strategy Nash equilibrium where both players choose ‘heads’ or ‘tail’ with a 50-50 probability distribution. Each player then loses as much as they win, resulting in a reward of 0. By definition, each state of a zero-sum game is Pareto-optimal since there is no way to increase a player’s reward without diminishing another players reward. There is thus but 1 Nash equilibrium.

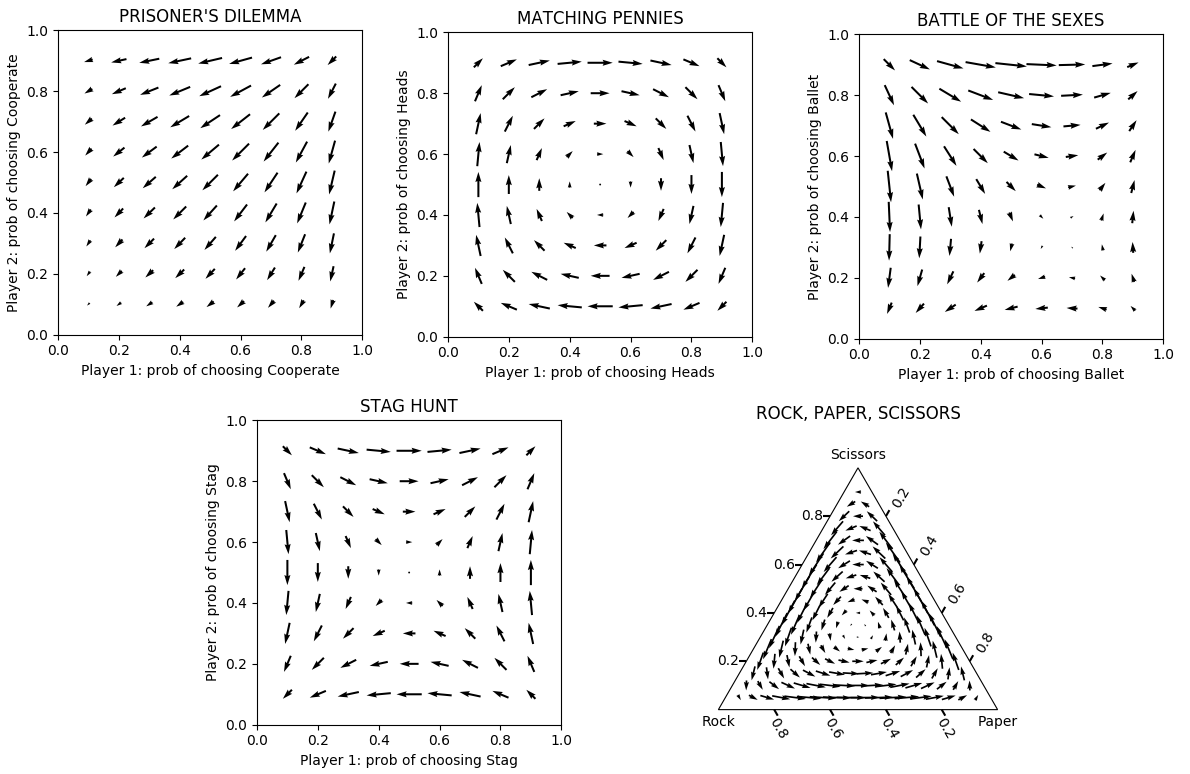
In the coordination game battle of the sexes, there are 2 pure strategy Nash equilibria: if both players choose ‘movies’ or both players choose ‘ballet’. There is also a mixed Nash equilibrium where both each player chooses his/her preferred activity 2/3’s of the time and the other activity 1/3 of the time. This leaves each player with an average reward of 2/3 (2/3\*2\*1/3 + 1/3 \* 1 \* 2/3). Both pure strategy Nash equilibria are Pareto-optimal. There are thus 3 Nash equilibrium.

In rock paper scissors, there is no pure strategy Nash equilibrium, since none of the state consists of a best response for both players. However, the players can mix the actions with 33% probability each to achieve the mixed Nash equilibrium with a reward of 0. Again, since this is a zero-sum game, each of the 9 states is Pareto-optimal. There is thus but 1 Nash equilibrium.

Lastly, in the stag hunt, another coordination game, if both players choose the same action ‘stag’ or ‘hare’ they reach a pure strategy Nash equilibrium. The difference with battle of the sexes is that the ‘stag’-’stag’ equilibrium is better for both players and is thus the only Pareto optimal state. The mixed strategy Nash equilibrium for the stag hunt consists of choosing the ‘hare’ or ‘stag’ action respectively 1/3 and 2/3 of the time.

Replicator dynamics are used to the change in policy populations over time. The intuition is that policies that are more present (a bigger population) or are simply better (a higher reward) have a higher chance of surviving. In a game form, an action will thus increase more if the probability of choosing action rises or if the reward (or fitness) of action rises relative to the average reward of all possible actions. We can also express this change in the probability of choosing action in a 2-player game, with payoff matrices and for player 1 and 2 respectively.

Figure 2 shows each of the Nash equilibria discussed in phase plots of population evolutions based on the payoff matrices of the games. Replicator dynamics are used to calculate the change in policy populations over time.



**Figure 2:** Quiver plot of each game showing the strategy evolution with replicator dynamics.

# 3 Independent learning in benchmark matrix games

## 3.1 Algorithm overview

***Q-Learning*** is used in single agent Markovian environments[[1]](#footnote-1) to maximize an agent’s reward (payoff in game theory). In multi-agent environments, the future state is not completely determined by the present state but also on the actions of other agents (for which the agent (initially) has no probability distribution). As such, Q-learning is not longer guaranteed to converge to optimal q-values. The traditional Q-learning update equation updates the q-values of the selected action , based on its reward and (discounted) future potential. An interesting side effect about using temporal difference learning algorithms such as Q-Learning in one-shot games, is that the ***influence of future rewards is eliminated*** (since there never are any possible actions to predict a future reward from).

Clearly these updates are independent of the agent’s policy. The two main policies are the epsilon-greedy and Boltzmann exploration scheme. The epsilon-greedy exploration scheme will, with a chance select the best action it knows, and with a chance it selects a random action. The Boltzmann exploration scheme will instead use a temperature value to define a normalize the probability distribution over the possible actions. As such, actions with a sub-optimal reward are not simply ignored percent of the time, but rather each action is picked at a frequency corresponding to their expected reward.

One interesting thing to note is that ***mixed strategy Nash equilibria are never achieved*** (even with an exploration rate of 0). The reason being that the agents are constantly changing their q-values based on previous rewards and mixed strategy equilibria are thus unstable (since with a probability p or (1-p) the agent will lose/win and adapt their q-values). The only way to reach a mixed strategy Nash equilibrium would be to start in one (especially if the probability p is not representable as a computer number) and have a learning rate of 0.

If we were to train the agents in an ***iterated play environment***, there is a higher chance of them hovering around the mixed strategy equilibrium, since the iterations average out the wins and losses out (whereas in one-shot learning, a couple of wins in row, could drive the q-values toward a pure strategy Nash equilibrium). Similarly, there is also a higher chance to end up in a Pareto optimal state in a coordination game (for example the ‘stag’-’stag’ state in the stag hunt game), since when we learn over multiple iteration, the other agents action choices are indirectly learned (as a form of noise in the environment) and taken into account when updating the q-values before going to the next training episode. In a way, iterated play ***mimics coordination*** between the players.

***Frequency Adjusted Q-learning*** (FAQ) tries to compensate for over-selection of a dominant action (due to the action having a higher selection probability) by dividing the learning rate in the update rule by the probability of the action whose q-values are being updated. An extra factor is also included to make sure we never reach a learning rate > 1, which could lead Q-values to become higher than the actual rewards.

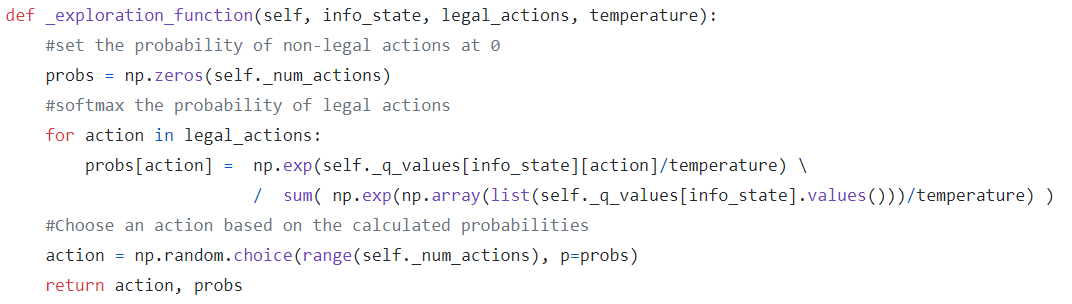
***Lenient Frequency Adjusted Q-learning*** (LFAQ) tries to introduce a concept of leniency towards other agents. This is especially useful in coordination-games, where the previous Q-learning algorithms are susceptible to early mis-coordination (action choices that do not lead to a Pareto optimal state) which can drive the agents away from the Pareto optimal state. The simplest way to implement this leniency towards other agents (leniency towards the fact that the other agents will not always choose the action leading to a Pareto optimal state), is to evaluate actions during iteration before training on the iteration that resulted in the best reward (which will often be the Pareto optimal state). For example, in the stag hunt, there is a fairly high chance that if we evaluate the agent’s actions 10 times, a state will occur where both actions choose ‘stag’. This is then the best reward and this action is used to update the q-values of both agents. The probability of choosing stag then rises for both agents. Repeating this procedure leads to the Pareto optimal state of ‘stag’ - ‘stag’. We can also intuitively feel that in later training episode, the probability of a Pareto optimal outcome rises, and thus the value can decrease to increase training efficiency.

## 3.2 Algorithm implementation

The open\_spiel library includes an implementation of epsilon-greedy Q-learning in the QLearner class. We implemented a child class from the QLearner class so that we can more easily change the exploration function and implement exploration annealing over training iterations.

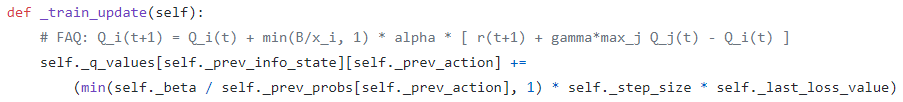
When we use ***EpsilonGreedy\_QLearner*** agents with an epsilon value annealing down to 0, the agents’ action choices converge to pure strategy Nash equilibria. An epsilon value > 0 cannot converge to a fixed strategy, since there is an inherit uncertainty as to whether each agent will stick to the strategy (and a Nash equilibrium is defined when the other player sticks to the same strategy). The smaller the epsilon value, the higher the probability that both agents will choose an action that leads to a Nash equilibrium state. The implementation of the epsilon-greedy exploration function is the same as in the open\_spiel library.

The ***Boltzmann\_QLearner*** agents can never have temperature value of 0 (since we divide by the temperature). Therefore, Boltzmann Q-learning can only hover around action probabilities that are very close to a pure strategy Nash equilibrium, but can never quite reach the equilibria since there is always some randomness left. One interesting problem we came across in the implementation was that for extremely low temperature values (such as t<0.0028), the resulting action probabilities overflow (and thus crash the program). This sets a lower bound temperature value, which was empirically found to be around 0.003. The implementation of the Boltzmann exploration function can be found in Figure 3.



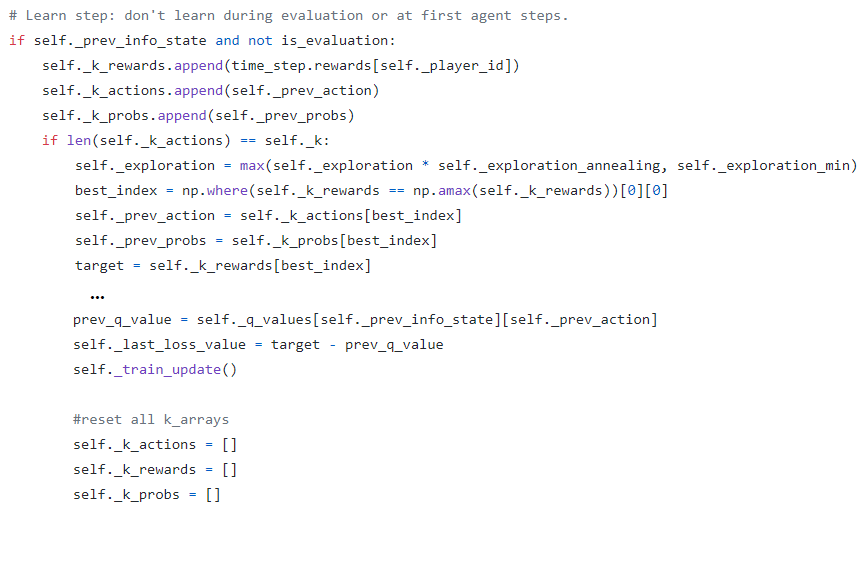
**Figure 3:** The python implementation of the Boltzmann exploration function. The Boltzmann\_QLearner inherits from the tabular\_QLearner.

The ***Boltzmann\_FAQLearner*** agents have to save the probability of the previously selected action in order to balance the learning rate between actions (actions that are selected less often have a higher learning rate). Instead of only saving the probability of the previous action, we save the output probabilities of the Boltzmann exploration function, since we will need these probabilities in the Boltzmann\_LFAQLearner. A -value is also saved to adapt the training update (so that the learning rate is never greater than 1). The value for was chosen equal to the learning rate , if was small enough. Otherwise, a smaller value of was chosen to have a more desirable path of convergence. An implementation of the FAQ training update step can be found in Figure 4.



**Figure 4:** The python implementation of the FAQ update rule. The Boltzmann\_FAQLearner class inherits from the Boltzmann\_QLearner.

Finally, the ***Boltzmann\_LFAQLearner*** agents have a -value and save a list of κ actions, probabilities and rewards. After κ-episode of evaluation (no q-value updates), the best rewards and corresponding action and probability is used to perform an FAQ-learning update on the q-values. The list of actions, probabilities and rewards are then emptied, and the process restarted. As such, a fairly high number of training iterations is required to achieve convergence. A higher learning rate was also chosen, since there is a high chance that the agent choses the optimal action at most of the training iterations (as the benchmarked games do not have that many different state-action combinations). An implementation of the LFAQ training update step can be found in Figure 5.



**Figure 5:** The python implementation of the LFAQ update rule. The Boltzmann\_LFAQLearner class inherits from the Boltzmann\_FAQLearner.

Figure 6 shows a trajectory plot of each of the 4 algorithms overlaid on a quiver plot of the replicator dynamics. ……………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………

FIGURE 6: Trajectory plots of each algorithms for a couple of games

# 4 Literature review

To get to know the open\_spiel, we mostly looked at the examples and algorithm implementations in the open\_spiel/python directory of the GitHub repository. To get to know the basic concepts of game theory such as payoff matrices, mixed and pure strategy Nash equilibria, Pareto-optimality and replicator dynamics, the introduction sections of the survey on Evolutionary Dynamics in Multi-Agents Learning and the papers on FAQ and LFAQ [1]–[3] were used along with online resources such as the YouTube series ‘Game Theory 101’ [4]. The different Q-learning algorithms and different exploration strategies were taken from the FAQ and LFAQ papers [2], [3], [5].

For the research on the algorithms we want to implement to solve Kuhn and Leduc Poker ………………….

………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………

# 5 Conclusion

………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………

# 6 Future work

………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………

# References

[1] D. Bloembergen, K. Tuyls, D. Hennes, and M. Kaisers, “Evolutionary Dynamics of Multi-Agent Learning: A Survey,” *J. Artif. Intell. Res.*, vol. 53, pp. 659–697, Aug. 2015, doi: 10.1613/jair.4818.

[2] M. Kaisers and K. Tuyls, “Frequency Adjusted Multi-agent Q-learning,” p. 8.

[3] D. Bloembergen, M. Kaisers, and K. Tuyls, “Lenient Frequency Adjusted Q-learning,” p. 8.

[4] W. Spaniel, *Game Theory 101*. 2019.

[5] A. Kianercy and A. Galstyan, “Dynamics of Boltzmann Q-Learning in Two-Player Two-Action Games,” *Phys. Rev. E*, vol. 85, no. 4, p. 041145, Apr. 2012, doi: 10.1103/PhysRevE.85.041145.

1. In a Markovian environment future states are independent of past states given the present state. Thus, if an agent performs action in state, it will always have the same probability of arriving in state no matter the states [↑](#footnote-ref-1)