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 the stag hunt. The payoff matrices for each game are visualized in Appendix A. In section 2 we discuss the Nash equilibria and Pareto optimal states for each game. Section 3 the implemented independent learning algorithms and population dynamics. Section 4 summarizes the literature that was used, and Section 5 draws a conclusion of the report. 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 uses a probability distribution over the actions. 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. Intuitively, this corresponds to 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).

In the prisoner's dilemma there is only 1 Nash equilibrium since both players have a strictly dominating strategy: ‘defect’. The only Pareto Optimal state is when both players ‘cooperate’. However, since this is not the dominating strategy, this is not a Nash equilibrium. 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 his 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 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 . Both pure strategy Nash equilibria are Pareto-optimal. There are thus 3 Nash equilibria.

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 choose each action 1/3 of the time 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, the two pure strategy Nash equilibria are defined by the players choosing the same action: the payoff dominant ‘stag’ or risk dominant ‘hare’ action. 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 when both players play each action ½ of the time, a mixed strategy Nash equilibrium is found. There are thus 3 Nash equilibria.

The fact that we only get odd numbers of Nash equilibria, is due to the *oddness theorem* stating that nearly all finite games have an odd and finite number of equilibria. If not, this is usually due to a weakly dominant strategy (a strategy that is not always better, but never worse).

# 3 Benchmarking in matrix games

## 3.1 Independent learning

***Q-Learning*** is used in single agent Markovian environments[[1]](#footnote-1) to maximize an agent’s reward (payoff in game theory). However, in a multi-agent environment the future state is not entirely determined by the present state but also by the actions of other agents (for which the agent (initially) has no probability distribution). Therefore Q-learning is no 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. A side effect 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).

(1)

The update rule is clearly independent of the agent’s policy. The two policies we implemented 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 select a random action. The *Boltzmann exploration* scheme will instead use a temperature value to define a probability distribution over the possible actions. Thus, actions with a sub-optimal reward are not simply ignored percent of the time, but rather each action is picked with a frequency in line with its expected reward. A side effect of this parameter, is that the agent can never quite reach a pure strategy Nash equilibrium, since we cannot set (divide by zero) and there is thus always some randomness left.

(2)

One interesting thing to note is that ***mixed strategy Nash equilibria are never achieved*** (even with an exploration rate of 0) since agents are constantly changing their q-values based on previous rewards and mixed strategy equilibria are thus unstable (since with a probability or 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 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 training episodes containing multiple iterations average out the wins and losses (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 part of the environment. In a way, iterated play thus *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 chosen action, mimicking a simultaneous update of all actions, as is the case in population dynamics. An extra factor is also used to ensure we never reach a learning rate , which could lead q-values to become higher than the actual rewards.

(3)

***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, 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 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 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.

Appendix D goes over the code implementation of each of the algorithms discussed above. Appendix C shows the learning trajectories for each of the learning algorithms on all the 2x2 matrix games. Figure 1 shows the learning trajectory of each algorithm overlaid on their equivalent population dynamic for the ……. game.

## 3.1 Population dynamics

Dynamics are used to model a change in policy in different populations through time. The equations for each of the discussed dynamics can be found in Appendix F. The intuition behind ***Replicator Dynamics*** *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 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. This can also be expressed for a matrix game with payoff matrices and for player 1 and 2 respectively.

***Boltzmann Q-learning dynamics*** describe the change in policy of Q-learning agents with a Boltzmann exploration scheme and an infinitesimally small learning rate. The dynamics can be divided in an exploitation and exploration part, where the exploration part can be further subdivided in the exploration of a single strategy and the exploration of the entire policy. In the equation below: is the chosen action and is any legal action.

***FAQ-learning* *dynamics*** are derived from the FAQ-learning update rule and can be seen as a weighted average between cross learning (replicator dynamics) and exploration.

In ***LFAQ-learning dynamics*** the fitness function of a population changes (due to playing iterations before training). The resulting fitness of a strategy is expressed as the expected maximum payoff of that strategy. The dynamics update equation is the same as for the FAQ-learning dynamics with a replacement of the fitness function by .

Figure 1 shows phase plots of the different population dynamics and the learning trajectory of their corresponding independent agent implementation on the ……. game. Appendix E goes over the code implementation for the different learning dynamics.

FIGURE 1: trajectory plot on top of phase plot for each independent learning implementation and its corresponding dynamics for the …. game.

# 4 Literature review

To get to know the OpenSpiel library, we mostly looked at the examples and algorithm implementations in the OpenSpiel /python directory of the library itself. 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], [5]. The different independent learning algorithms and their corresponding dynamics can be found in the FAQ and LFAQ papers [2], [3], [6].

For the research on the algorithms we want to implement to solve Kuhn and Leduc Poker we mostly used the ‘related work’ and ‘Poker – Current Methods’ sections of [7] and the original papers of the discussed algorithms [8], [9].

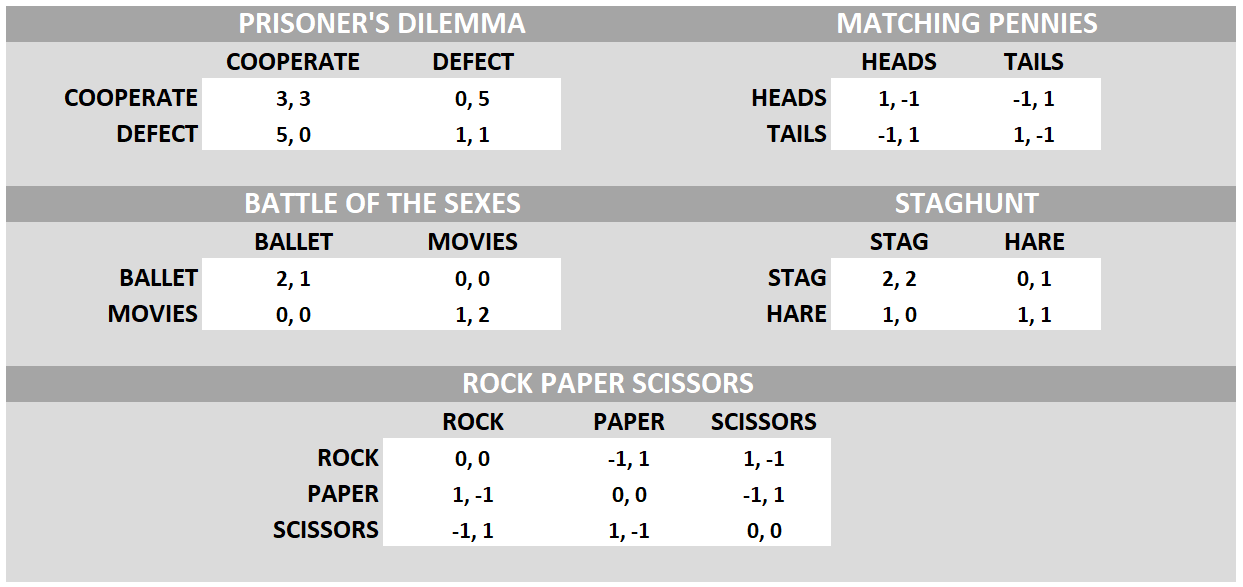
# 5 Conclusion

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

# 6 Future work

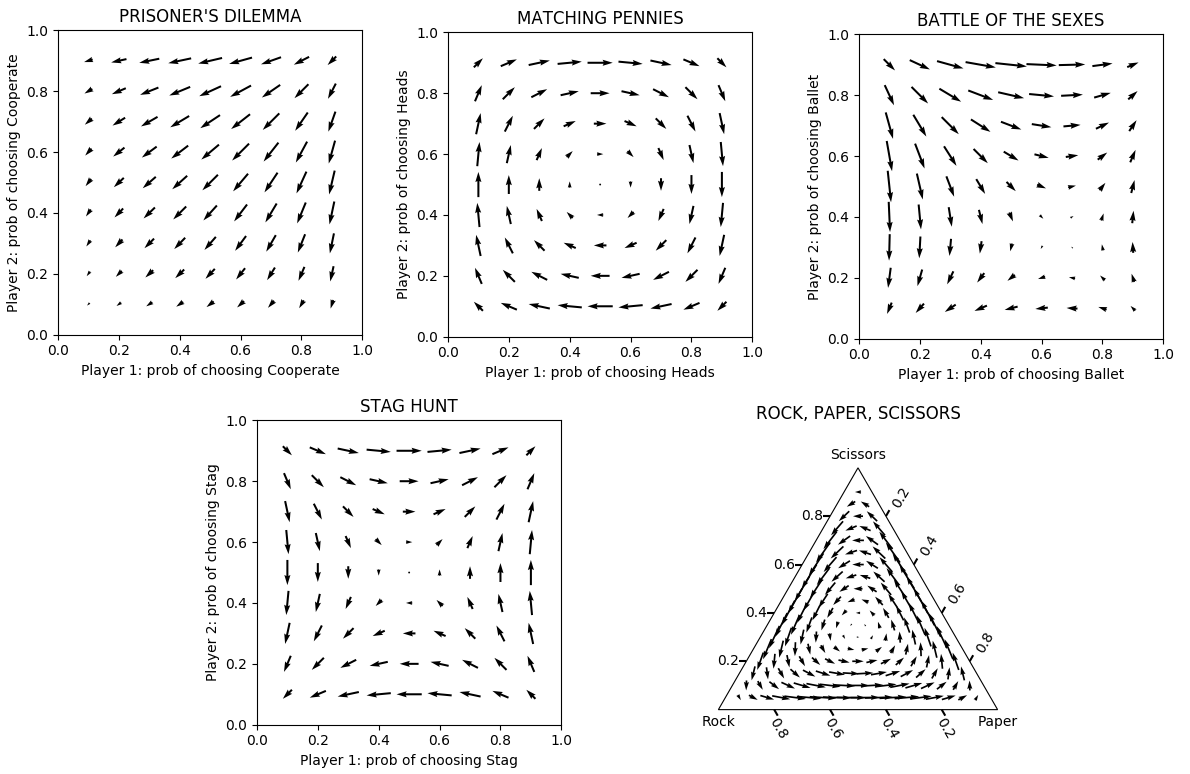
Kuhn and Leduc poker are incomplete information game and thus we are looking for an algorithm we can run on an extensive-form game, where we can use chance events. Since we need to output a lookup table to compete against the other agents of the course, we decide that the self-play Counterfactual Regret Minimization (CFR) algorithm would be interesting. If a more efficient implementation is necessary (although certainly not likely for Kuhn Poker), we can look at the Monte Carlo variant (MCCFR). Although the strategies from other groups will be static, we cannot train on them, and thus Fictitious Play is unlikely to be a good contender.

# APPENDIX A: Payoff matrices for all games



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

# APPENDIX B: Replicator Dynamics phase plots for all games



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

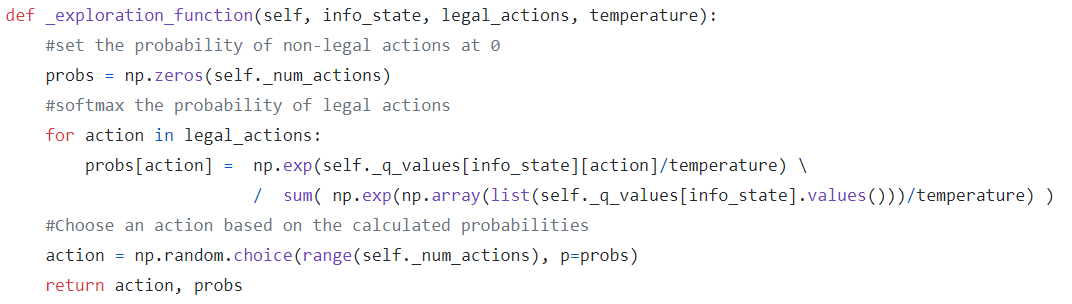
# APPENDIX C: Trajectory plots of all independent learning algorithms on the corresponding phase plots for the 2x2 matrix games.

# APPENDIX D: Code implementation of the independent learning algorithms

The OpenSpiel library includes an implementation of epsilon-greedy Q-learning. We implemented a child class from this implementation to add exploration annealing and make a change of exploration function easier. This tabular\_QLearner class forms the base class for all other algorithms we implemented.

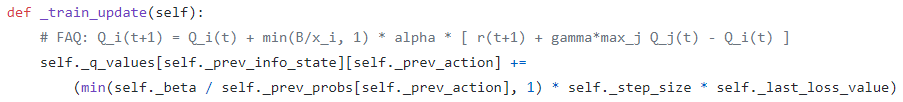
The ***EpsilonGreedy\_QLearner*** agents are a copy of the OpenSpiel QLearner with the added option to do exploration annealing.

The ***Boltzmann\_QLearner*** changes the exploration function as seen in Figure 3. 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.



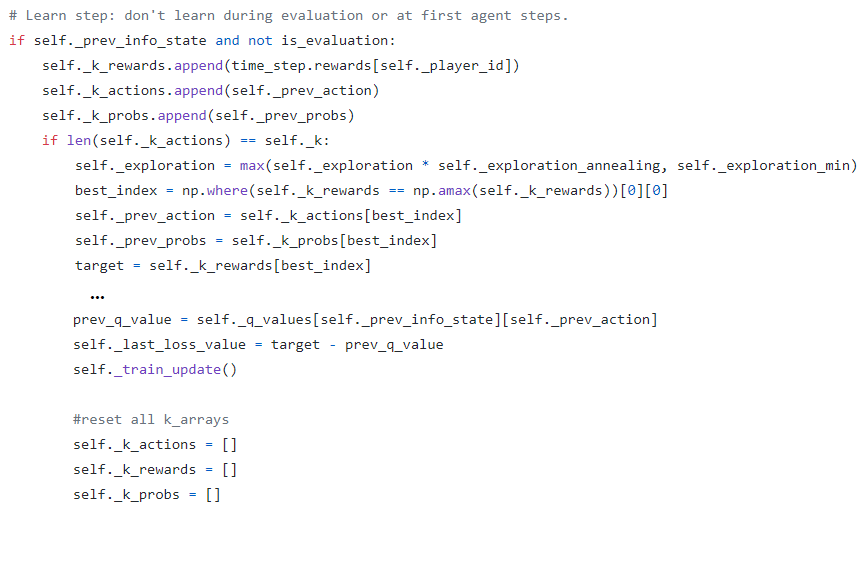
**Figure 3:** The python implementation of the Boltzmann exploration function. The Boltzmann\_QLearner inherits from the tabular\_QLearner.

The ***Boltzmann\_FAQLearner*** agents 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-episodes of evaluation (no q-value updates), the best reward and corresponding action and probability is used to perform one FAQ-learning update on the q-values. The list of actions, probabilities and rewards are then emptied, and the process reiterates. As such, more training iterations are 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. A -value around 10 was used and is constant whilst training. As the benchmarked games do not have that many different state-action combinations, this value is likely to contain an iteration with an optimal action choice for both agents. 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.

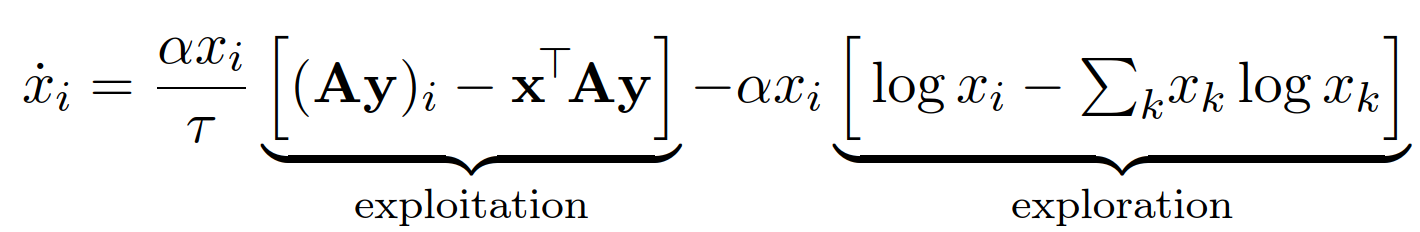
# APPENDIX E: Code implementation of the population dynamics

# APPENDIX F: All equations together

* **Q-learning**
* **Boltzmann exploration scheme**
* **FAQ-learning**
* **LFAQ-learning**

-

* **Replicator Dynamics**
* **Q-learning Dynamics (**with Boltzmann exploration**)**

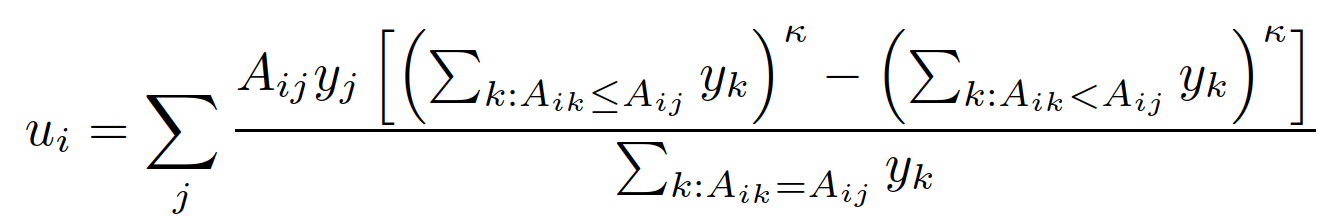


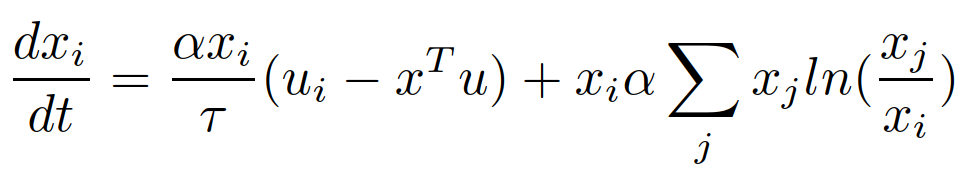
* **FAQ-learning Dynamics**

A close up of a logo

Description automatically generated

* **LFAQ-learning Dynamics**





# 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] O. Salazar, *19 2 The Replicator Equation 1329*. .

[6] 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.

[7] J. Heinrich, “Reinforcement Learning from Self-Play in Imperfect-Information Games,” University College London, London, 2017.

[8] N. Burch, M. Lanctot, D. Szafron, and R. G. Gibson, “Efficient Monte Carlo Counterfactual Regret Minimization in Games with Many Player Actions,” p. 9.

[9] M. Zinkevich, M. Bowling, M. Johanson, and C. Piccione, “Regret Minimization in Games with Incomplete Information,” p. 14.

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)