Mini-Project 2: Nim

Louis Poulain--Auzéau & Anya-Aurore Mauron
Scipers: 301545 & 295924
Artificial Neural Network, EPFL Lausanne, Switzerland

June 2022

1 Introduction

We chose to only respect the word limit, and not have extra figures in the report. In this document, 'random player' stands for a player who plays at random every time. 'Optimal player' stands on the contrary for a player who follows the optimal policy at each move. If we talk of 'Optimal player with $\varepsilon = x$', we mean a player who plays randomly with probability x and follows the optimal policy otherwise. The terms 'agent', 'Q-agent' and 'QL-player' designate all the player who plays by following the Q-learning algorithm (again, ε stands his probability to play randomly). Finally the term 'decrease of exploration' refers to the following formula, used by a Q-agent to decrease its ε during the training

$$\varepsilon(n) = \max\Big(\varepsilon_{\min}, \varepsilon_{\max}(1 - \frac{n}{n^*})\Big),$$

where n is the number of games played and $\varepsilon_{\min}, \varepsilon_{\max}$ are constants pre-determined, respectively equal to 0.1 and 0.8. For n^* big, $\varepsilon(n)$ will start close to ε_{max} and linearly decreases to ε_{min}. More exactly, $\varepsilon(1) = \varepsilon_{max}(1 - 1/n^*)$ and $\varepsilon(n) = \varepsilon_{min}$ for $n \geq n^*$. It means that for n^* big, there will exploration games than for low ones.

2 Q-learning

If the question does not specify it otherwise, the results are obtained by training a Q-agent with $\varepsilon = 0.1$ against an optimal player with $\varepsilon = 0.5$. The results will also be averaged on 5 simulations to take into account the stochasticity of the problem. It doesn't mean, though, that the results should be interpreted as they are. There still remain discrepancies.

Solution 1 Figure 1 shows that the average reward indeed goes up during the training to reach about 0.25 after 20000 games. This means that the Q-learning player eventually wins 62.5% of the games so we conclude that the algorithm seems correct and the Q-learning agent learns to play NIM.

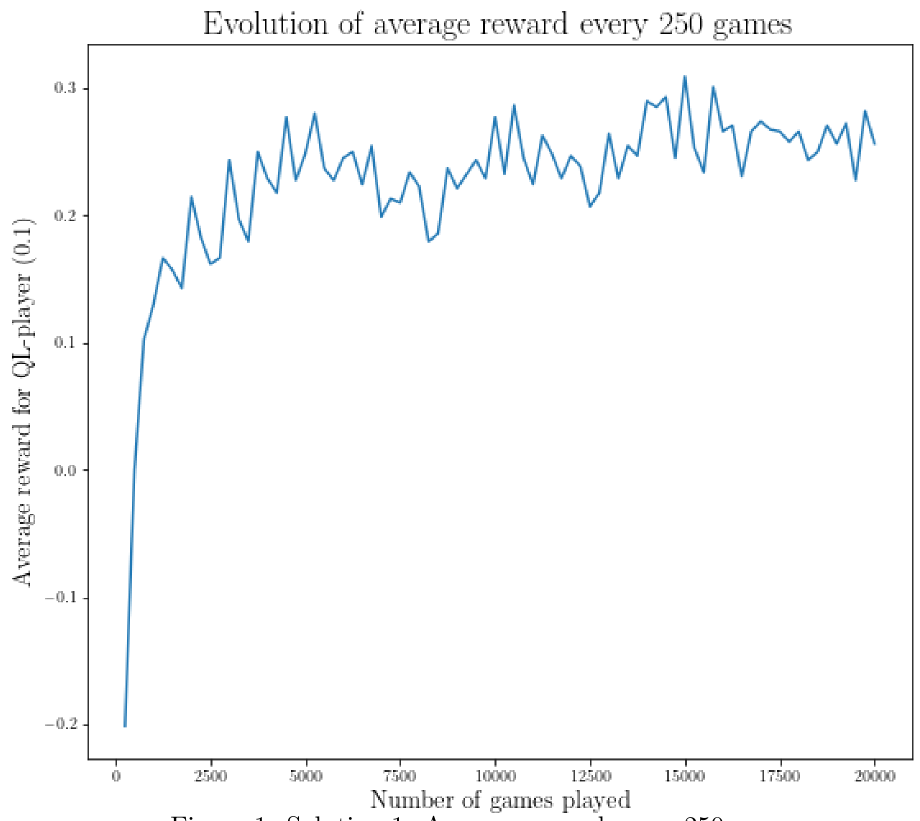


Figure 1: Solution 1. Average reward every 250 games.

Solution 2 On Figure 2 we can observe that, surprisingly, decreasing the exploration level doesn't seem to have an influence on the final averaged reward. The only difference is that the early performances are much better with little to no exploration. This was expected since with no exploration the agent will prefer the greedy policy and will find quickly how to win.

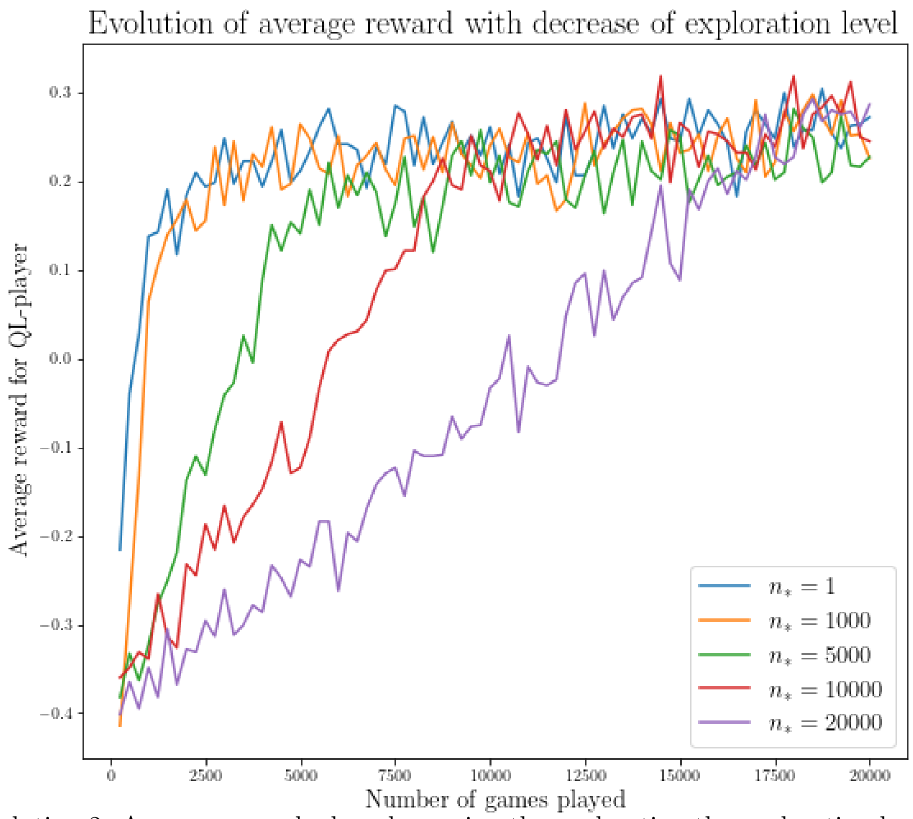


Figure 2: Solution 2. Average reward when decreasing the exploration the exploration level for several n^*.

Solution 3 We see on Figure 3, as expected, that the performance against the random player is very high (90% of games won at the end) for every n^*, which for this setting doesn't seem to play a role. On the other hand, the performance against the optimal player is very bad (5% of the games are won eventually for most of the n^*). We see, though, that the higher n^*, the better the performance (except for 1000 and 5000). Indeed, since the agent will explore a lot, he will have a better understanding of all options that he can takes. Since he doesn't train against an optimal player, it helps him when playing against an optimal player.
Compared to the previous question, we see that decreasing exploration doesn't seem to influence the result when playing against a totally random player ($\mathrm{M}_{\mathrm{rand}}$) and a semi-random player (the training player).

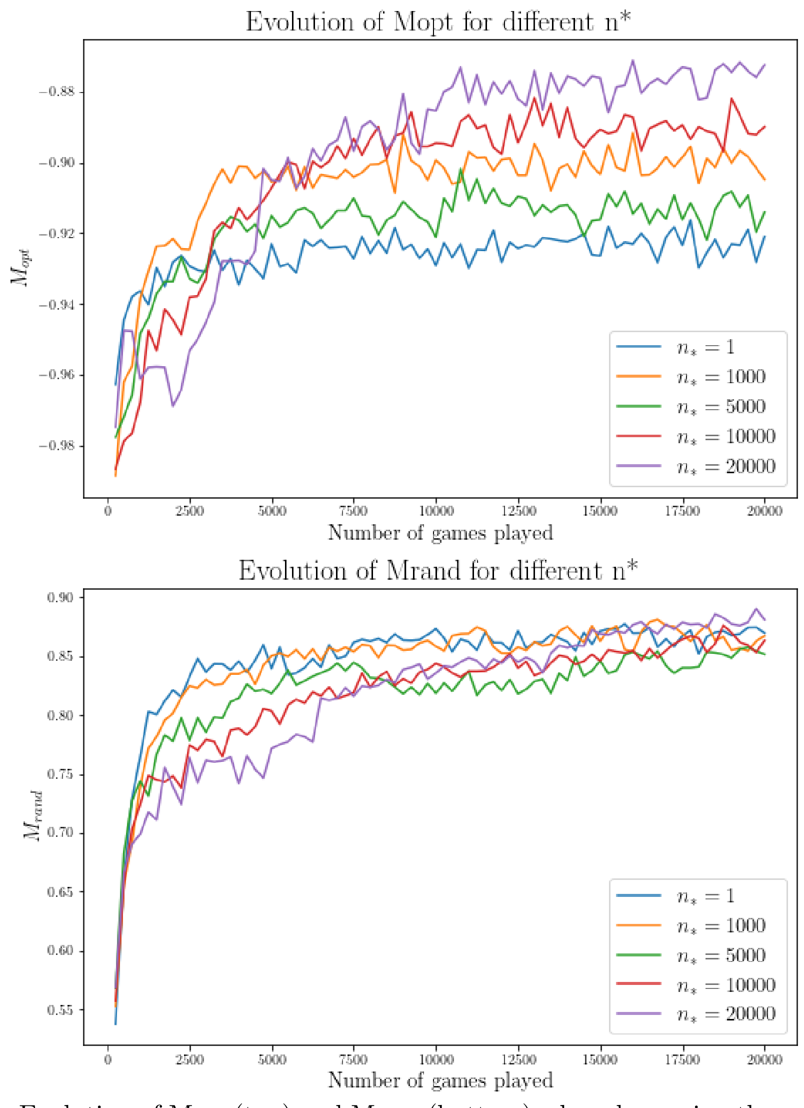


Figure 3: Solution 3. Evolution of $\mathrm{M}_{\mathrm{opt}}$ (top) and $\mathrm{M}_{\mathrm{rand}}$ (bottom) when decreasing the exploration level for several n^*.

Solution 4 In this question, we use $n^* = 20'000$. First, the performance against an optimal player seems related to having trained against a player with very small ε_{opt}. Indeed, as soon as $\varepsilon_{\text{opt}} = 0.1$, the performance is bad (only 20% of the games are won) and for higher ε_{opt}, the performance does not even seem to increase. On the other hand, the performance against a random player is very bad for $\varepsilon_{\text{opt}} = 0$, and relatively good for all other values. In fact, for $\varepsilon_{\text{opt}} = 0.01$, the performance is not that good and almost identical as for $\varepsilon_{\text{opt}} = 0.8$. Finally for $\varepsilon_{\text{opt}} = 0.05$ and 0.1, the performances are very good (around 0.96 for both).

The good performance for $\varepsilon_{\text{opt}} = 0$ against the optimal player but bad against the random one is due to the agent being trained on configurations played by an optimal player (hence it should perform well against the optimal player). In fact, those configurations only cover a small part of all possible configurations. Hence the bad performance against a random player. Indeed, if the random player's leads to a configuration for which the Q-agent only has Q-values close to 0 or similar, then he will play 'randomly'. This aspect can also be seen for $\varepsilon_{\text{opt}} = 0.01$: very good against the optimal but not that good against the random.

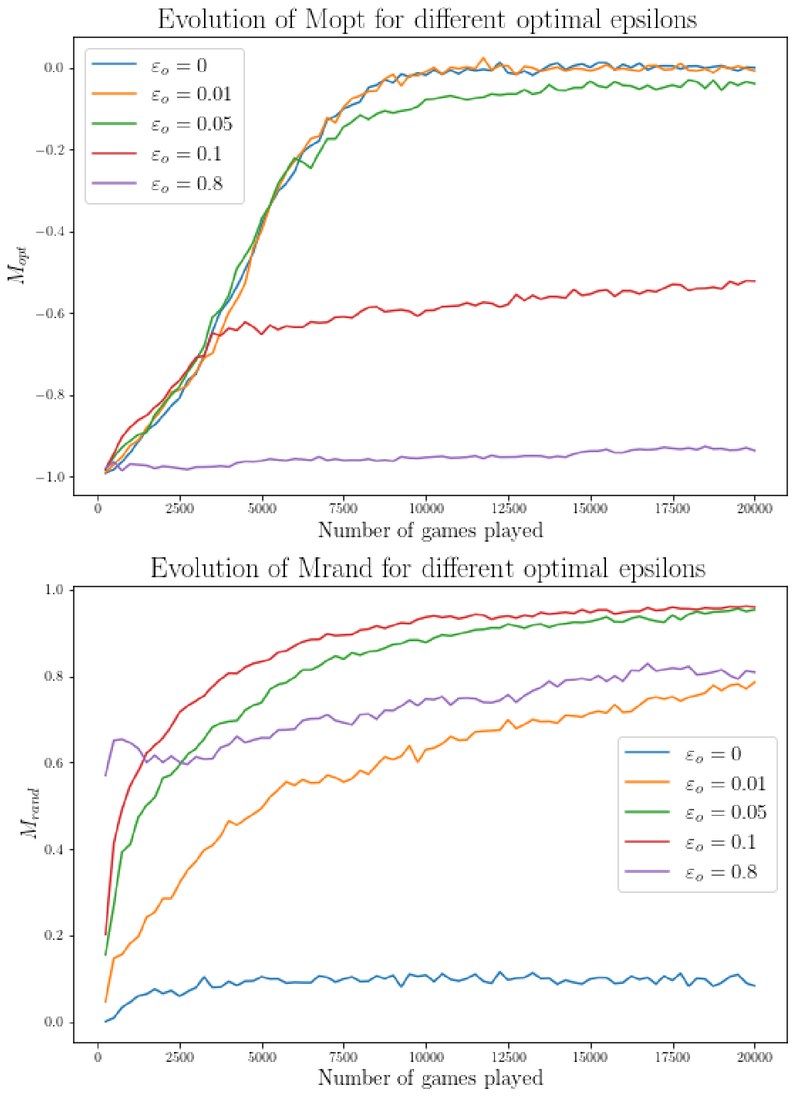


Figure 4: Solution 4.

Solution 5 If we consider the best values as a whole then we obtained $M_{\text{opt}} = -0.04$ and $M_{\text{rand}} = 0.95$ (for $\varepsilon_{\text{opt}} = 0.05$). Independently, we achieved $M_{\text{opt}} = -4.8e-4$ (for $\varepsilon_{\text{opt}} = 0$) and $M_{\text{rand}} = 0.96$ (for $\varepsilon_{\text{opt}} = 0.1$). In question 20, we use the first two values.

Solution 6 No. Indeed, by training against an optimal player, the Q-agent will probably learn a biased version of the game since his opponent will not play some of the available moves. Then he will only update the relevant Q-values, and leave some of them at 0. On the other hand, by playing against a random player, there is a greater chance that every configuration is seen during training and hence every Q-value is updated. Then, even if both players learn the optimal policy, their Q-values will differ for the moves that haven't been updated during the training against the optimal player. Moreover, learning the optimal policy doesn't mean that the agent has reached a set of fixed Q-values. Indeed, if the agent only has a greedy policy, then he chooses the action with the highest Q-value, regardless of the values of the rest. So two players could be playing the same, without having exactly the same set of Q-values.

Solution 7 From Figure 5, we see that as soon as ε is bigger than 0, the performance against the optimal player is drastically reduced. On the other hand, it seems to have less impact against a random player, even though the difference can still be seen. Moreover, for a high ε, we see that the performance on 20000 is worse than for lower ε but the shape of curve seem to indicate that for more games the performance would be equally good. However, against the optimal player, the performance is very bad. Indeed, the higher the exploration, the lesser chance the player has to find a good policy in 'early games'.

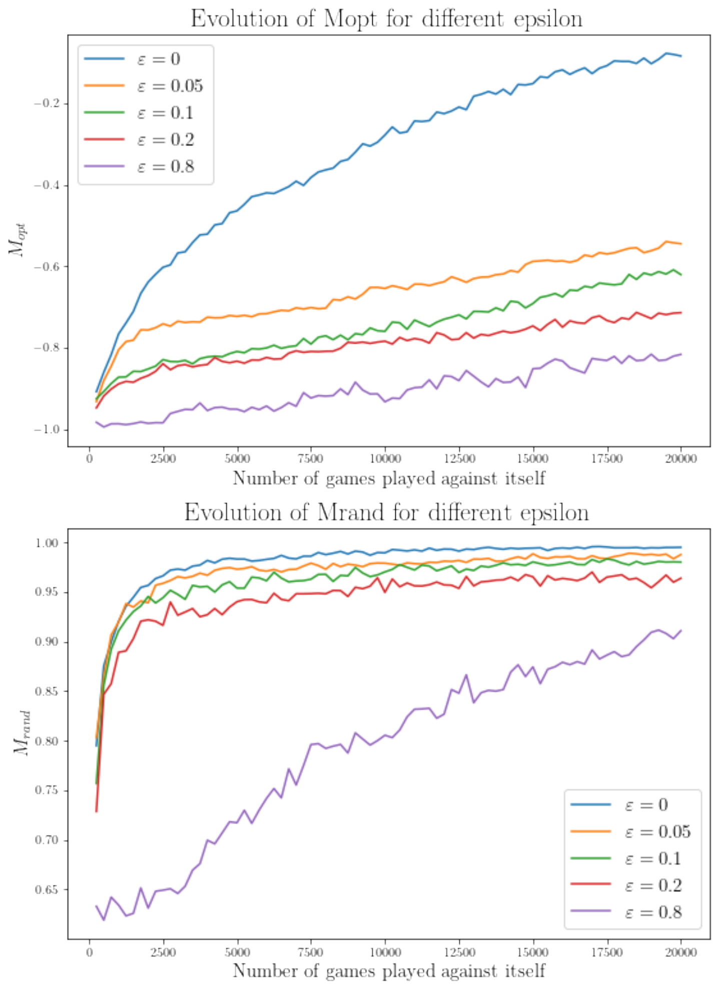


Figure 5: Solution 7.

Solution 8 Against the optimal player, it seems that increasing n_* doesn't improve the results. In fact, for early games, the performance is better for low n_* and at the end of the training, all performances except for $n_* = 1$ are at the same level. On the second figure, we can see that the differences are less pronounced between low n_* and high ones, but again, having no exploration seems to produce better results.

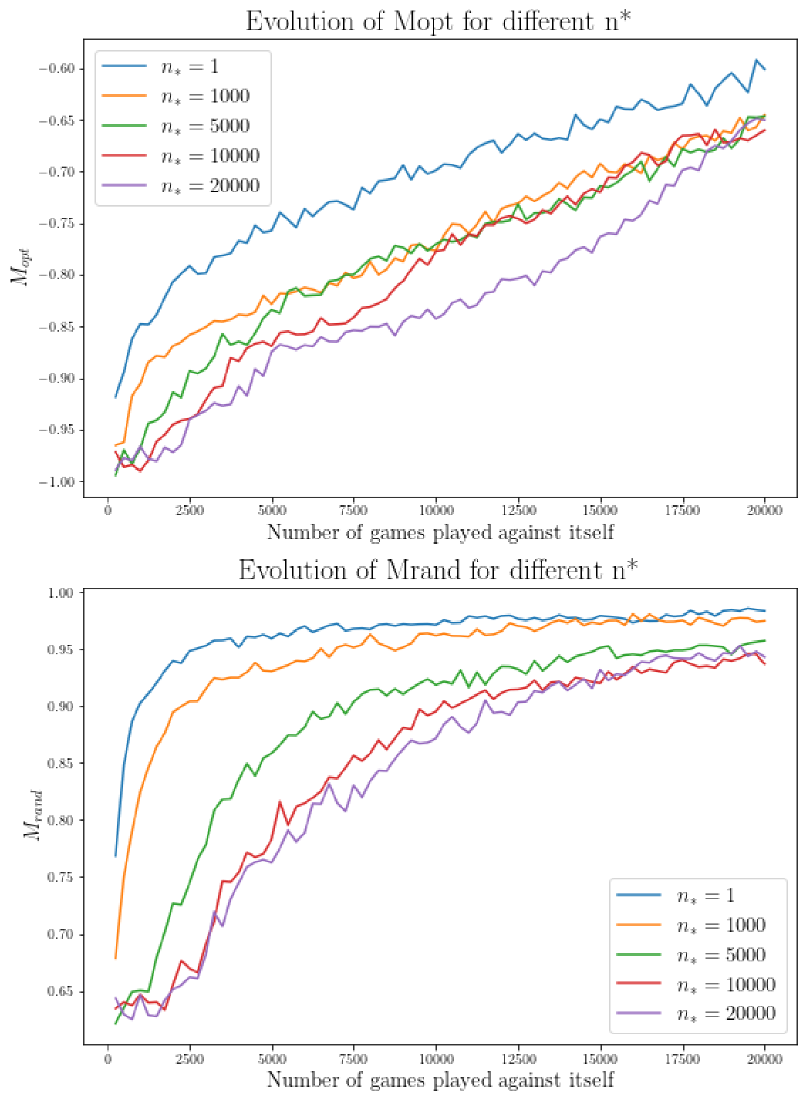


Figure 6: Solution 8.

Solution 9 The best values we could achieve are $M_{\text{opt}} = -0.6$ and $M_{\text{rand}} = 0.98$ (for $n^* = 1$).

Solution 10 For the first configuration (Figure 7, top), it is clear that the action taken is the best one as it results in a win. The second plot (Figure 7, middle) shows that the player has a perfect understanding for this situation and would eventually win the game. Indeed, after his move, there will only be 1 stick in each heap, forcing the other player to lose the game. Finally for the last plot (Figure 7, bottom), we can use the optimal policy to determine if the preferred move is the correct one. The future position should have a nim-sum of 0 to ensure that the QL-player will win. Here:

- action 030 leads to the configuration 002. We have: $000_2 \oplus 000_2 \oplus 010_2 = 010_2 \neq 000_2$, so taking this action would result to a loss;
- action 020 leads to the configuration 012. We have: $000_2 \oplus 010_2 \oplus 010_2 = 000_2$, so taking this action would result to a loss;
- action 010 leads to the configuration 022. We have: $000_2 \oplus 010_2 \oplus 010_2 = 000_2$ so taking this action would eventually result in winning (if the following Q-values are also the right ones);
- action 002 leads to the configuration 030. We have: $000_2 \oplus 011_2 \oplus 000_2 = 011_2 \neq 000_2$, so taking this action would result to a loss;
- action 001 leads to the configuration 031. We have: $000_2 \oplus 011_2 \oplus 001_2 = 001_2 \neq 000_2$, so taking this action would result to a loss;

Hence, the best choice is indeed the move that is preferred by the QL-player.

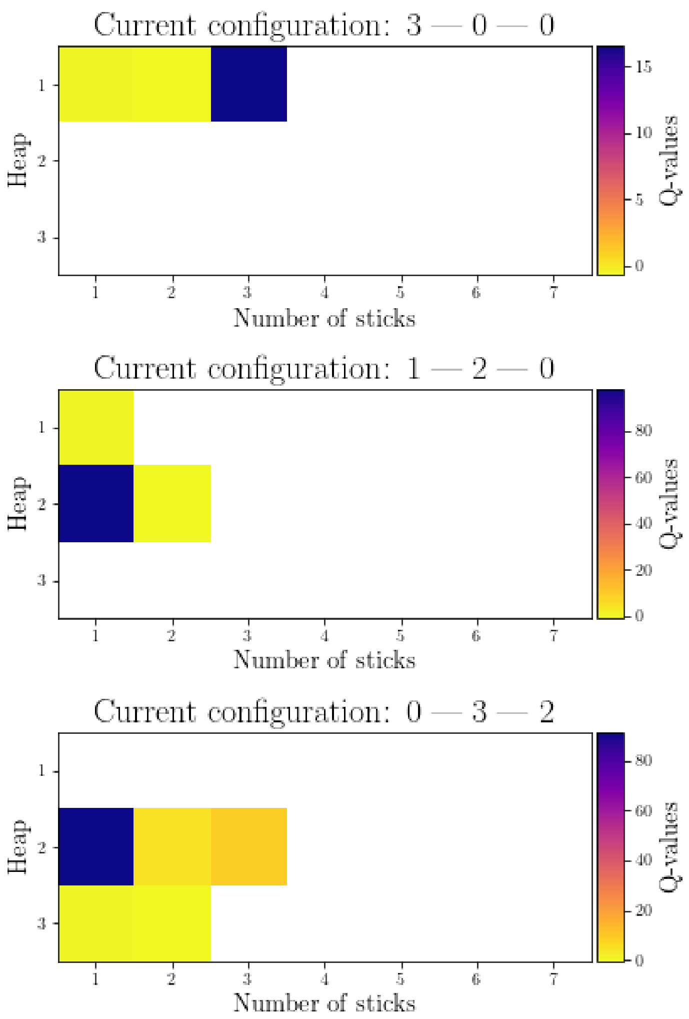


Figure 7: Solution 10.

3 Deep Q-learning

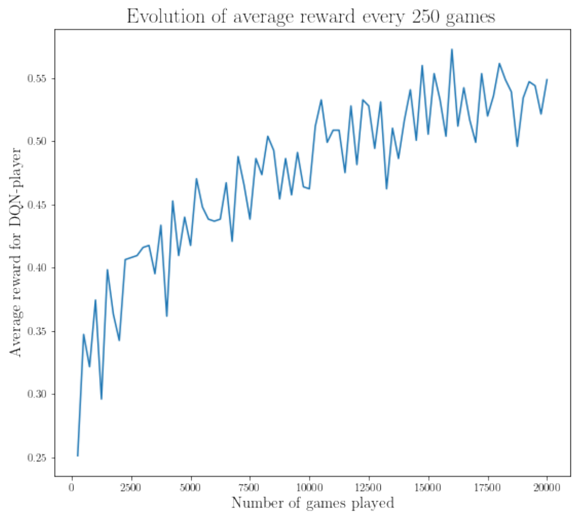
3.1 Implementation Details

The DQN algorithm is implemented with the Pytorch library [1]. The main structure of the optimization and the replay buffer was inspired by the DQN tutorial given by Pytorch. The state representation and neural network architecture are implemented as described on the project sheet. The free parameters are also set as given, except for the learning rate of ADAM optimizer, which was set to 10^{-4}.

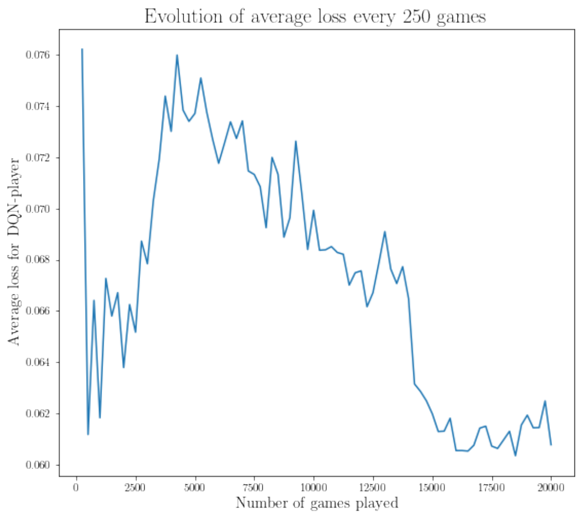
Moreover, the following results are computed from an averaging over 5 trials, as in the previous part. Since it is a mean, the results in the text are given with a precision of ± 0.05.

3.2 Learning from experts

Solution 11 Several combinations of learning rates and of ϵ are tested. The best result was obtained with $\epsilon = 0.1$ and $lr = 10^{-4}$. That way, the reward was the highest (around 0.55) and the loss decreases, see figure 8. The agent does learn how to play Nim, as the reward is increasing through the games.



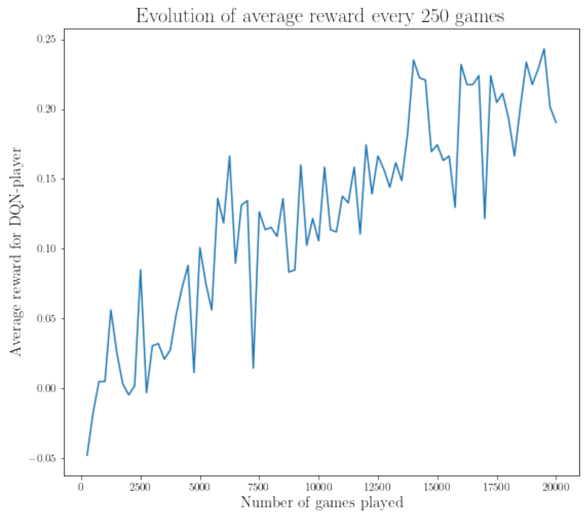
(a)



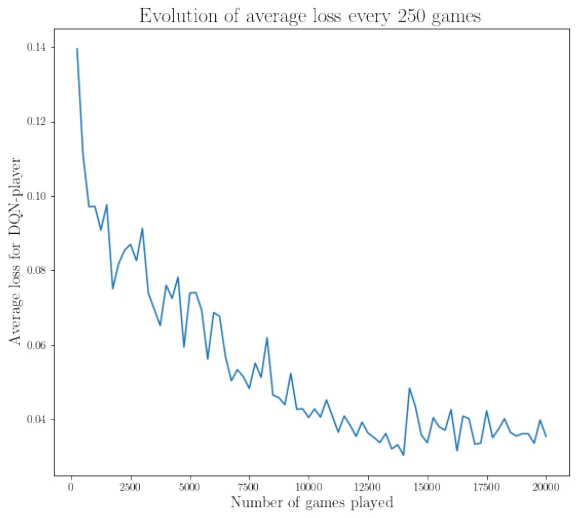
(b)

Figure 8: Solution 11.

Solution 12 Without the replay buffer and with batch size of 1, the agent reaches a lower average reward: 0.25 instead of 0.55, see figure 9. The average loss is still decreasing, which shows that the optimization is working.



(a)



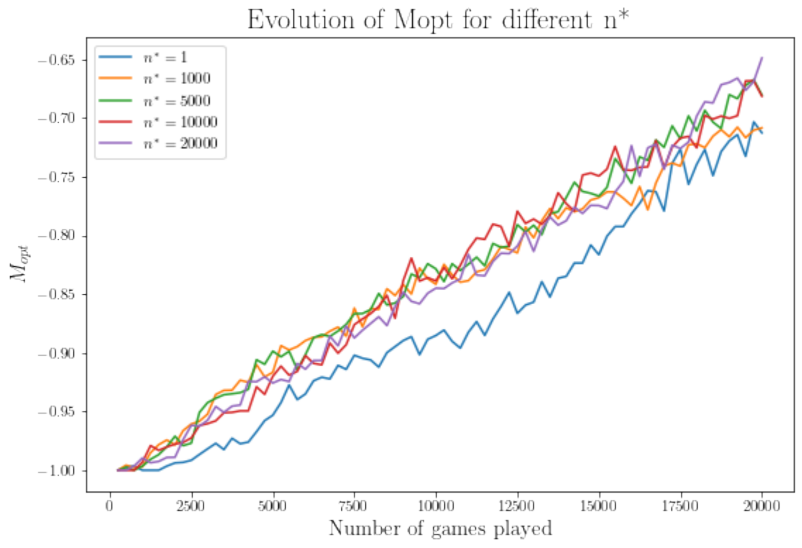
(b)

Figure 9: Solution 12.

Solution 13 The evolution of M_{opt} and M_{rand} is plotted for $n^* = 1, 1000, 5000, 10'000, 20'000$ in figure 10. For all $n^* \geq 1000$, the evolution of M_{opt} is approximately the same. It approximately grows linearly with respect to the number of games. The value M_{opt} reaches a value of -0.65 after $20'000$ games. For $n^* = 1$, the evolution is slower. The agent performs a bit less well, but at the end of the $20'000$ games, still reaches a value $M_{opt} = -0.7$.

Concerning M_{rand}, its evolution is not linear. For all n^*, M_{rand} grows rapidly until 7'500 games, and then stabilizes. We observe that the evolution of M_{rand} is really similar for $n^* \geq 1'000$. At the end of the 20'000 games, M_{rand} reaches a value of about 0.9. For $n^* = 1$, its evolution is slower. However, it still reaches about the same value after 20'000 games.

In this figure, we see that more exploration in the beginning only helps a little bit getting a better M_{opt} and M_{rand} at the end of the 20000 games. Therefore decreasing ε while training only helps a bit, compared to having a fixed ε, after 20 000 games. From now on, we choose $n^* = 1000$ since we have seen that it reaches a good performance.



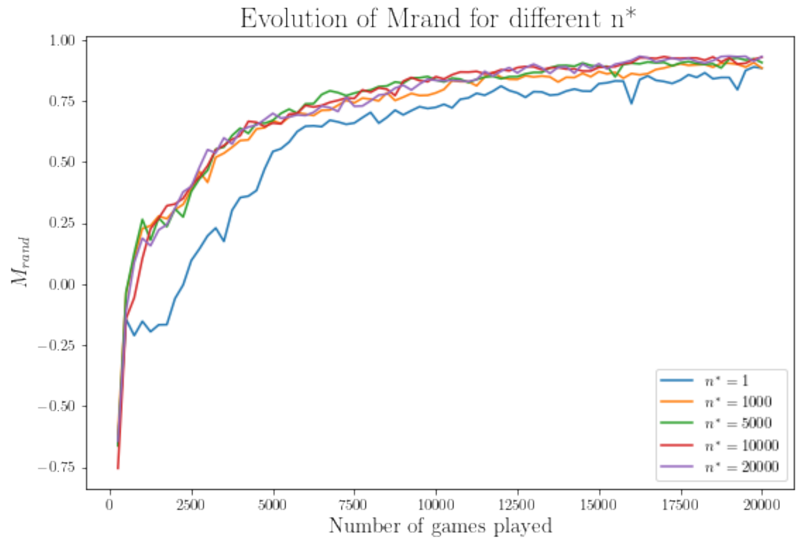


Figure 10: Solution 13.

Solution 14 We observe in figure 11 the evolution of M_{opt} and M_{rand} for $\varepsilon_{opt} = 0, 0.2, 0.4, 0.6, 0.8, 1$. We know that the optimal player has $\varepsilon_{opt} = 0$ and the player who only takes random moves has $\varepsilon_{opt} = 1$. In between, the player takes random moves with probability ε_{opt}.

This is coherent with the result we obtained. For the evolution of M_{opt}, the smaller the value ε_{opt}, the stronger the increase of M_{opt} throughout the games. In the end, M_{opt} reaches the highest value of about -0.2 for $\varepsilon_{opt} = 0$. The lowest value after 20 000 games is about -0.95 for $\varepsilon_{opt} = 1$. This behaviour is due to the fact that the DQN agent could learn a lot better how to win against the optimal player, when training against it. When its opponent is the random player, the DQN agent learns how to play against random moves, but as they are many configurations of the game, it does not learn how to play against the optimal moves.

The evolution of M_{rand} is the best for $\varepsilon_{opt} \geq 0.2$. Indeed, M_{rand} converges to its highest value of about 0.8 for $\varepsilon \geq 0.2$. However, the bigger ε_{opt}, the faster the convergence of M_{rand}. For $\varepsilon_{opt} = 0$, the DQN agent only learns how to play against the optimal player, which results in a poor performance when playing against a random player. As the DQN agent does not see a big variety of moves, it performs poorly when its opponent plays completely randomly. It results with an evolution of M_{rand} staying below -0.25.

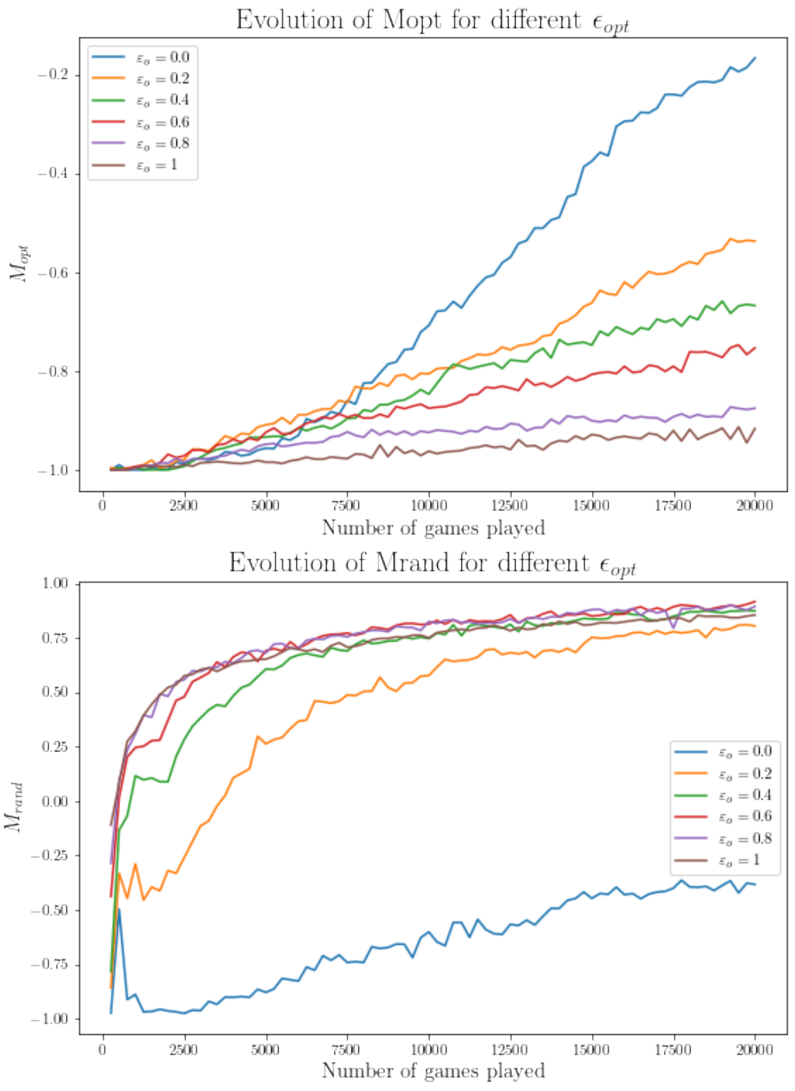


Figure 11: Solution 14.

Solution 15 For different n^* and training against a player with $\varepsilon_{opt} = 0.5$, the highest value of M_{opt} that could be achieved after 20000 games is about -0.65. For all $n^* \geq 1000$, M_{opt} converges approximately to that value. For various ε_{opt} and $n^* = 1000$, the highest value of M_{opt} is -0.2 for $\varepsilon_{opt} = 0$.

For different values n^* and training against an opponent with $\varepsilon_{opt} = 0.5$, the highest value of M_{rand} is about 0.9. For all n^*, M_{rand} converge to that value. The highest value of M_{rand} for different ε_{opt} is about 0.8, and is reached by $\varepsilon_{opt} \geq 0.2$ after 20 000 games.

3.3 Learning by self-practice

Solution 16 The fastest evolution of M_{opt} is with $\varepsilon = 0.2$; it reaches about $M_{opt} = -0.4$, see figure 12. Then, the bigger ε, the slower the evolution of M_{opt} through the games. Finally, for $\varepsilon = 0.0$, the DQN agent looses at every game, since $M_{opt} = -1$. Furthermore, for $\varepsilon > 0$, the evolution of M_{rand} is approximately the same. The value M_{rand} reaches about 0.9 after 20′000 games. For $\varepsilon = 0.0$, M_{rand} stagnates below 0. The agent could not learn how to play with $\varepsilon = 0.0$ since it did not see a big variety of states. Thus it must surely take some forbidden moves.

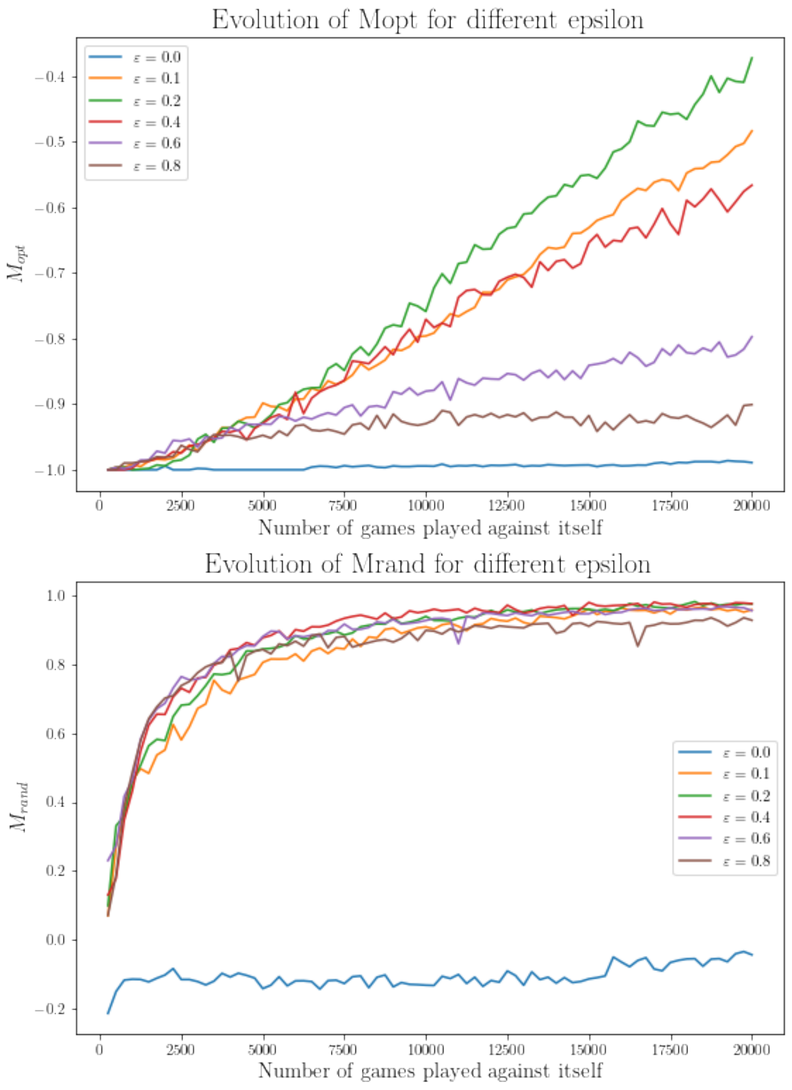


Figure 12: Solution 16.

Solution 17 In figure 13, M_{opt} has the best evolution for $n^* \in \{1'000,\ 5'000,\ 10'000\}$. It reaches about -0.35 at the end of the 20'000 games. The evolution of M_{rand} is approximately the same for $n^* \in \{1'000,\ 5'000,\ 10'000\}$. For $n^* = 1$, the growth is a bit slower, but M_{rand} still reaches the same value of about 0.95 at the end of the 20'000 games. Hence, decreasing ε helps playing against the optimal player, but not so much against the random player.

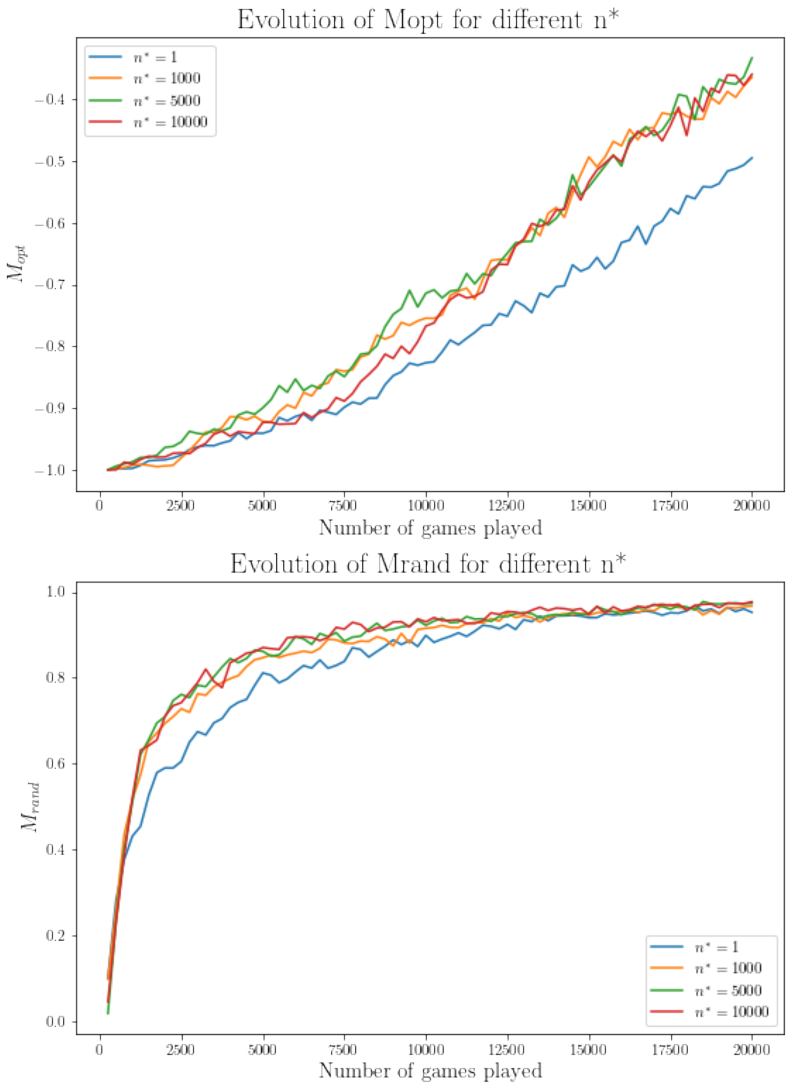


Figure 13: Solution 17.

Solution 18 Concerning M_{opt}, the highest value that could be achieved was about -0.35, with $\varepsilon = 0.2$ in solution 16, and $n^* \in \{1'000,\ 5'000,\ 10'000\}$ in solution 17. Moreover, concerning M_{rand}, the highest value that could be attained was about 0.95 in solution 17.

Solution 19 The best performance was for example obtained at solution 17 with $n^* = 5'000$. We train the DQN agent against itself for 20'000 games and visualize the Q-values of all actions, see figure 14. We use the same arrangements of games as in solution 10. The DQN player chooses the correct actions in order to win (see solution for more explanation). Thus, the agent learnt how to play the game well.

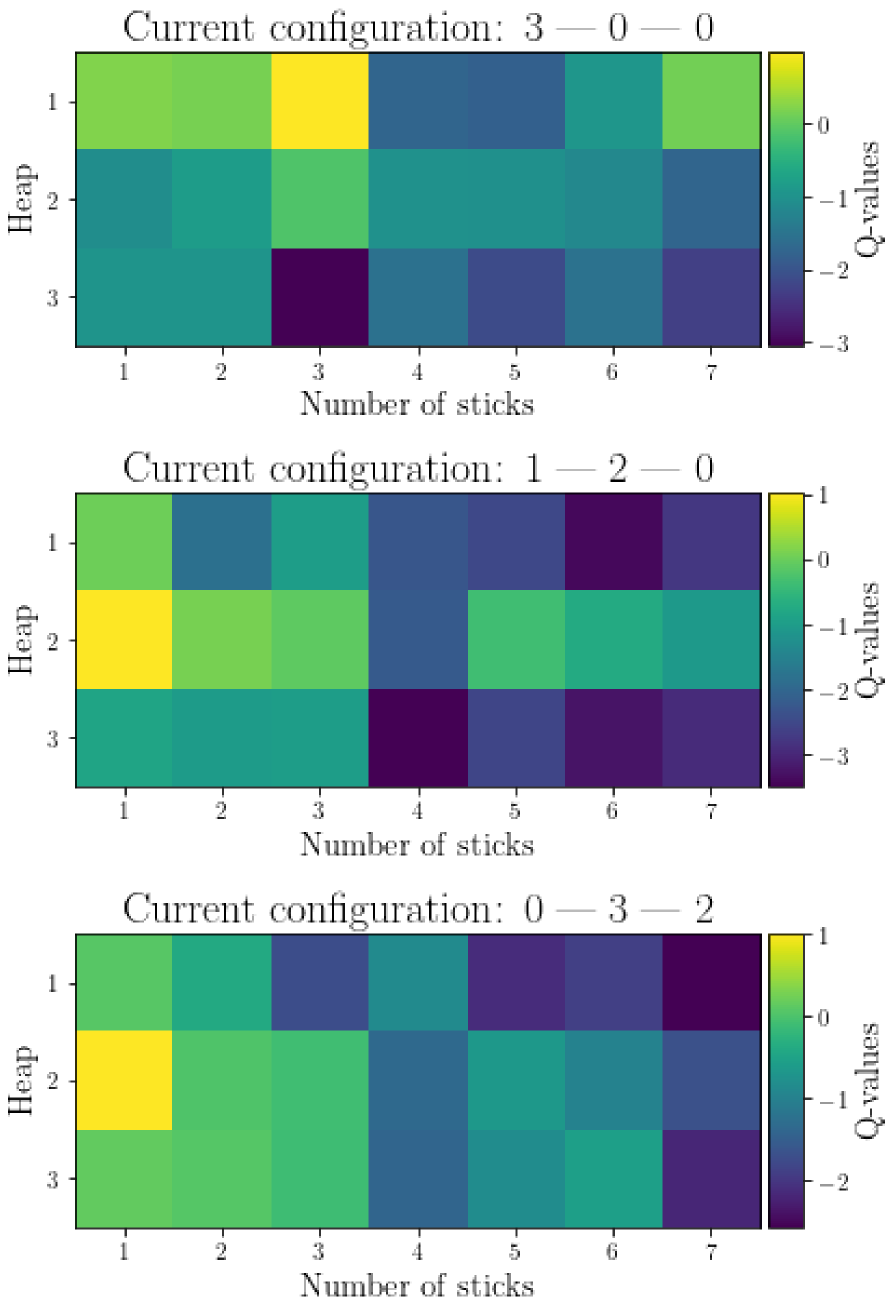


Figure 14: Solution 19.

4 Comparing Q-Learning with Deep Q-Learning

Solution 20 Table 1 presents the best M_{opt} and M_{rand} final result for each player (Q-learning and DQN), against the expert or self practicing.

		M_{opt}	M_{rand}	T_{train}
Q-Learning	expert	-0.04	0.95	$\sim$ 7'000
	self-practice	-0.6	0.98	10'000
DQN	expert	-0.2	0.9	15'000
	self-practice	-0.35	0.95	16'000

Table 1: Solution 20. Best performances of Q-learning and DQN and their corresponding training time

Solution 21 Against an expert player, the Q-learning agent learned better how to play compared to the DQN agent. Indeed, the value M_{opt} is closer to zero when using Q-learning compared to using DQN learning. However, when self-practicing, the DQN agent plays better than the Q-learning agent. Overall, both players obtained similar good results for M_{rand}, with a slightly better performance for the Q-learning player though. In addition, the training time is clearly shorter for the Q-learning agent. Learning from experts, the Q-learning agent learns about twice as fast than the DQN agent. But this was expected since the DQN agent is allowed to take forbidden moves.

Therefore, overall, the Q-learning agent learns better how to play Nim compared to the DQN agent. The only exception would be for self-practicing against the optimal player making no random moves. Nevertheless, the Q-learning agent beats the DQN agent on any other aspects.

To conclude, both players present different implementations that could achieve reasonable results. Both players showed similar behaviours for the different training types (changing exploration level, ε_{opt} or ε). The Q-learning player implementation has the particularity of saving the q-values for each possible moves, which can become a problem if the number of moves is very big. On the other hand, DQN employs a neural network, which has the advantage of having a reasonable number of parameters. Nevertheless, we were constrained to implement the DQN player so that it could take unavailable actions. That might be one of the reasons that the Q-learning player performed generally better than the DQN agent.

References

[1] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.