ANN Mini Project Report

TicTacToe

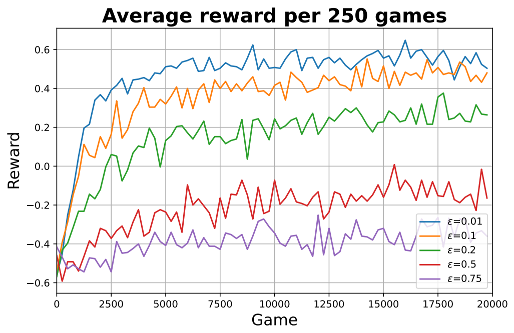
Authors:
Mahammad Ismayilzada (337396)
Abdulkadir Gokce (336709)

June 6, 2022

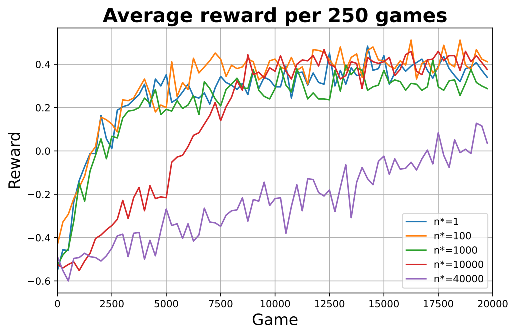
1 Q-Learning

Question 1)

In Fig. 1a we observe that the agent can learn how to win the game, as the average reward exceeds 0.5. We can also deduce that average reward inversely correlated with the ϵ, so greedier agents win games with higher probability.



(a) Q1



(b) Q2

Figure 1: Q1 & Q2

Question 2)

As the training unfolds and n increases, $1 - \frac{n}{n^*}$ decreases and hence ϵ given by

$$\epsilon(n) = \max\left\{\epsilon_{\text{min}}, \epsilon_{\text{max}}\left(1 - \frac{n}{n^*}\right)\right\} \tag{1}$$

starts from $\epsilon_{\text{max}} = 0.8$ and reduces to $\epsilon_{\text{min}} = 0.1$. This allows the agent playing with ϵ-greedy algorithm to explore more at the beginning through taking random actions with high probabilities. At some point ($n \geq n^*$), $\epsilon(n)$ stays the same at ϵ_{min} and after this point it mainly exploits its learned Q-values. From Fig. 1b we observe that $n^* = 100$ and $n^* = 1000$ yields higher average reward, hence decreasing ϵ helps in this case by exploring more at the beginning. However, increasing $n^* > 10000$ any further slows down the training as the agent starts exploiting in a very late stage of the training.

Question 3)

In Fig. 2, one notable difference is that all agents with varying n^*'s are able compete with optimal player, as M_{opt} converged to 0, meaning the agents learn how to compete against the optimal algorithm. However, Fig 1b and M_{rand} didn't converge to an exact value. We also observe that decreasing n^* according to Eq. 2 slowed down the convergence of M_{opt} since agents did not exploit their knowledge in beginning to find the best moves. In the case of M_{rand}, we see that moderate n^* values (i.e. 100) increases the performance as in Q2.

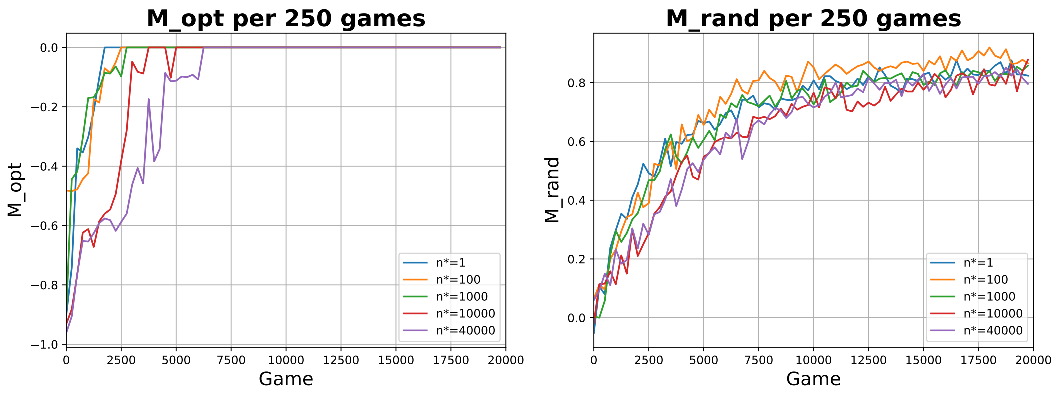


Figure 2: Q3

Question 4)

In this part, we used $n^* = 100$ since it is the best for M_{rand} and second best for M_{opt} in Q3. One striking result in Fig. 3 is that when the agent is trained against random player, $\epsilon_{\text{opt}} = 1$, it cannot compete with optimal player, $\epsilon_{\text{opt}} = 0$. Even for large values of ϵ_{opt} (i.e. $\epsilon_{\text{opt}} = 0.75$), the agent picks up the best actions after playing many games and its M_{opt} converges to 0, whereas the agent trained with $\epsilon_{\text{opt}} = 1$ stuck at $M_{\text{opt}} \approx -0.4$. It is also interesting that the fastest model to reach $M_{\text{opt}} = 0$ is $\epsilon_{\text{opt}} = 0.1$, instead of $\epsilon_{\text{opt}} = 0$. It might be due to not being able explore at the beginning, as optimal player with $\epsilon_{\text{opt}} = 0$ wins the game with least amount of moves. Nevertheless, we see complementary results for M_{rand}, as agents trained against $\epsilon_{\text{opt}} = 0$ and $\epsilon_{\text{opt}} = 0.01$ couldn't escape $M_{\text{rand}} \approx 0.1$. We observe positive correlation of an agent's performance against a random player and ϵ_{opt}. As a result, agents trained against random-like players play well against true random player since they encountered unpredictable moves during training.

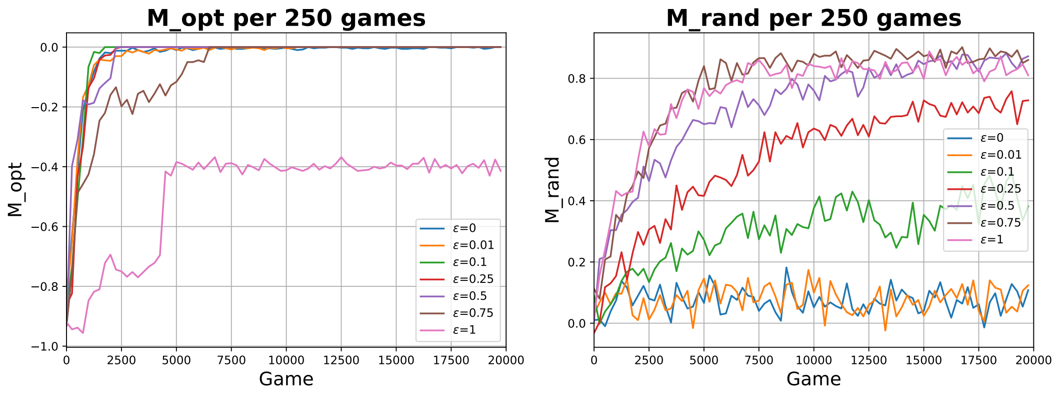


Figure 3: Q4

Question 5)

The best values are $M_{\text{opt}} = 0$ and $M_{\text{rand}} = 0.918$.

Question 6)

No. When playing against optimal player, the uncertainty in the game is lower as the optimal will take the best action, which is most of the unique cases. As a result, if we think about the backup diagrams, majority of the state action pairs won't be visited as optimal player will consistently choose similar actions. On the other hand, random player puts uniform probability on all possible actions, hence there is a positive probability to visit all states if player's $\epsilon > 0$. As a result, learned Q-values will be more uniform. We can also observe this in Fig. 3, where the agents trained against optimal player and random player perform substantially different.

Question 7)

In Fig. 4, all agents have $M_{\text{rand}} > 0.5$ at the end, so they can learn by self-practice. Nevertheless, exploring more clearly helps as there is nice separation between M_{rand} values which positively correlates with ϵ. Furthermore, only the agent obtaining $M_{\text{opt}} = 0$ is trained with $\epsilon = 0.5$, which means that exploration have significant importance in self-learning. When, $\epsilon = 0.1$, the agent cannot compete well against the optimal player, since it tries to exploit before learning the game well.

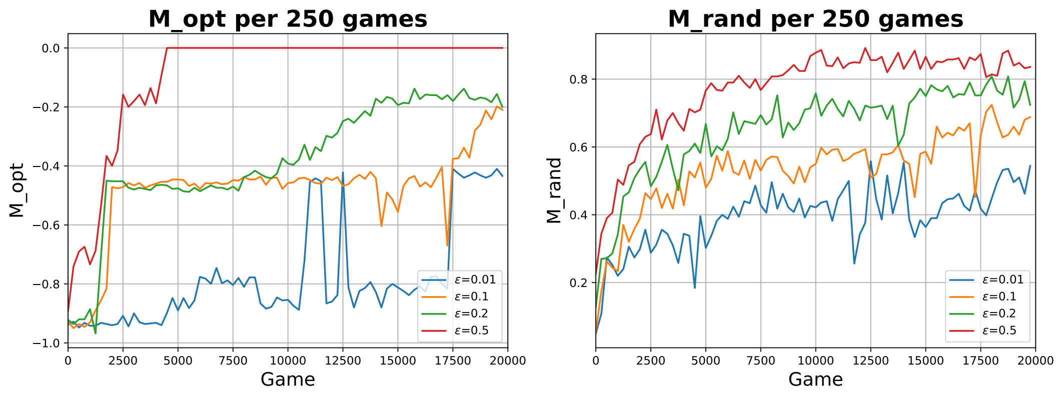


Figure 4: Q7

Question 8)

From Fig. 5, we can deduce that the higher the n^*, the better both the M_{opt} and the M_{rand}. Decreasing agent's ϵ from 0.8 to 0.1 according to Eq. 1 allows agents to explore more aggressively and visit many state-action pairs in beginning. Then, as players' ϵ reduces, they exploit their accumulated Q-values more and choose greedier actions. Hence, at each turn their action becomes more challenging to themselves and they learn how to play better. Contrary to Q3 in Fig. 2, $n^* = 40000$ results in the best M_{opt} and M_{rand}, so slowly decreasing $\epsilon(n)$ is crucial for self-learning.

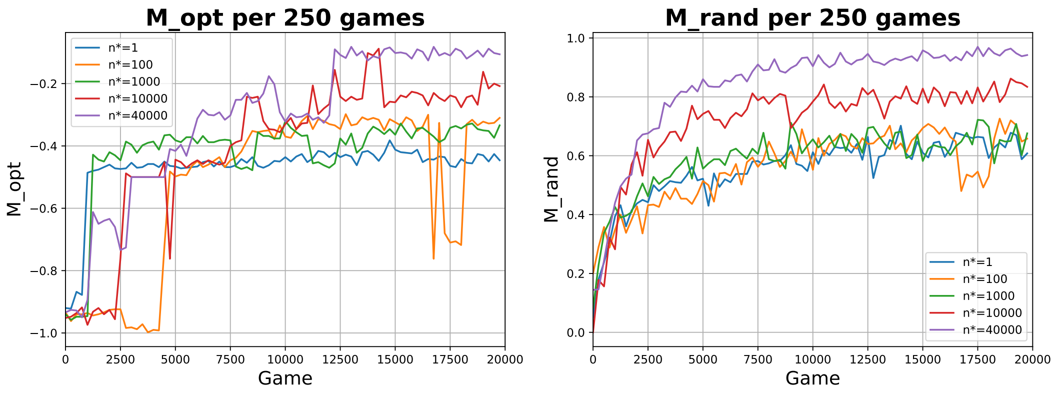


Figure 5: Q8

Question 9)

The best values are $M_{\text{opt}} = -0.082$ and $M_{\text{rand}} = 0.97$.

Question 10)

In this question, we used the agent in Q9 trained with $n^* = 40000$ as has the best M_{opt} and M_{rand}. The image on the left in Fig. 6 show that the highest Q-values is tile between two **X**'s, which lets the agent both block its opponent and prepare a way to win. In the center heatmap we have similar board arrangement and the greediest move is to block **O**, which is the logical action. In this game, the optimal move for the second player when the player chooses a corner is the center tile. Unsurprisingly, the image on the right illustrates that the agent was able to learn it by self-practice as the action with highest Q-value is in the center. From these, we can conclude that the agent can learn playing the game by playing against itself.

2 Deep Q-Learning

Question 11)

In Fig. 7, we can clearly see that the average loss converge to a value higher than 0.2 and the average reward is increasing. This indicates that the agent is learning to play TicTacToe. Also, smaller ϵ results in lower loss and higher reward as expected.

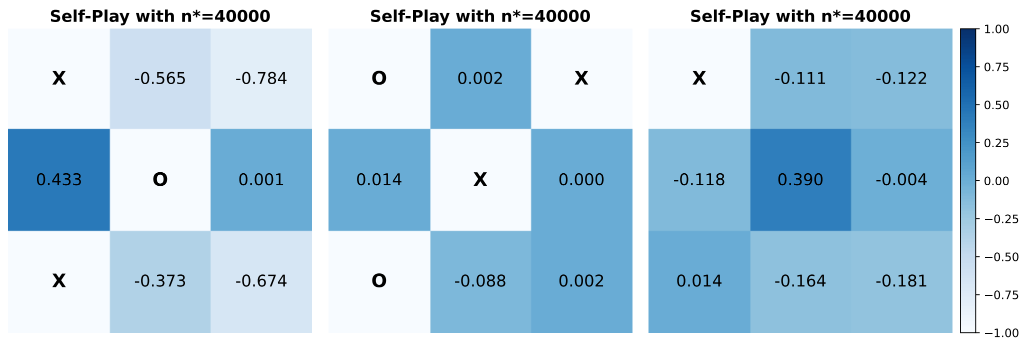


Figure 6: Q10

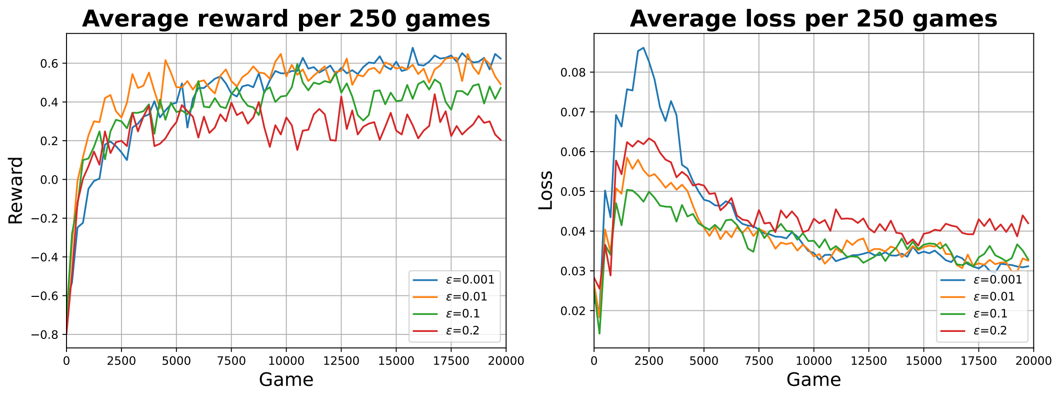


Figure 7: Q11

Question 12)

In Fig. 8, average loss is again decreasing and the average reward is increasing, however, compared to Question 11, loss starts at a higher value and reward converges to a lower value. This indicates replaying more memory and optimizing with a larger batch size is more effective.

Question 13)

In Fig. 9, we observe that varying $\epsilon(n)$ does not really improve M_{rand}, as agents trained with $n^* = 1$ and $n^* = 10000$ have overlapping plots which oscillate around the same mean value, $M_{\text{rand}} \approx 0.9$. Similarly, agents reach $M_{\text{opt}} = 0$ around the same number of played games. These imply that exploring more aggressively in beginning instead of starting with a low ϵ does not speed up the convergence. We think that the reason of the fluctuations in M_{opt} values is the shared network weights, since when the model is optimized, all weights are updated according to the batch sampled from the replay memory. Although the network can

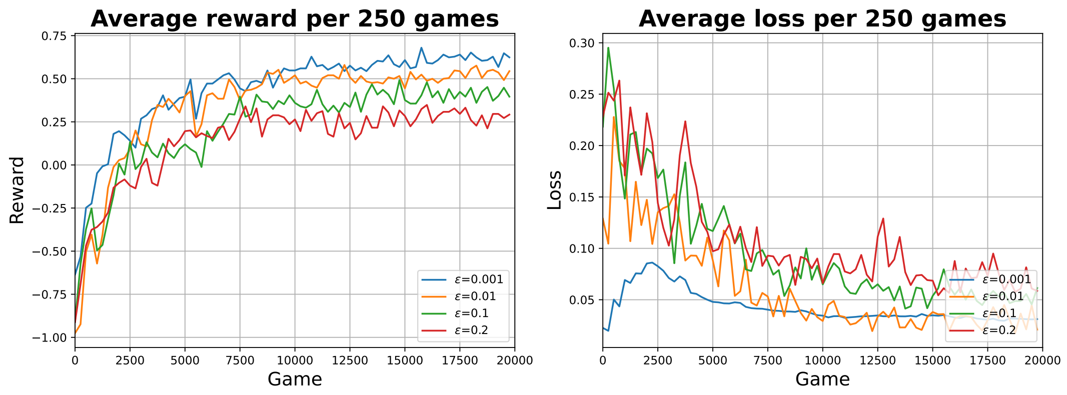


Figure 8: Q12

play against the optimal player without losing any match at some point during the training, it might not be able to select the optimal actions when its weights are updated while playing a non-optimal player. As a result, after reaching $M_{\text{opt}} = 0$, weights are still optimized and the network struggles against the optimal player for some time. Nevertheless, $n^* = 1$ (fixed ϵ) fluctuates more abruptly than $n^* = 10000$ so early exploration might still be beneficial for the network. We also want to mention that although we trained more models with different n^* values, we only report three of them to make the figure readable. Those results also have the characteristics with the ones shown here, hence the above discussion holds for them as well. For full results, you can refer to our Wandb Report.

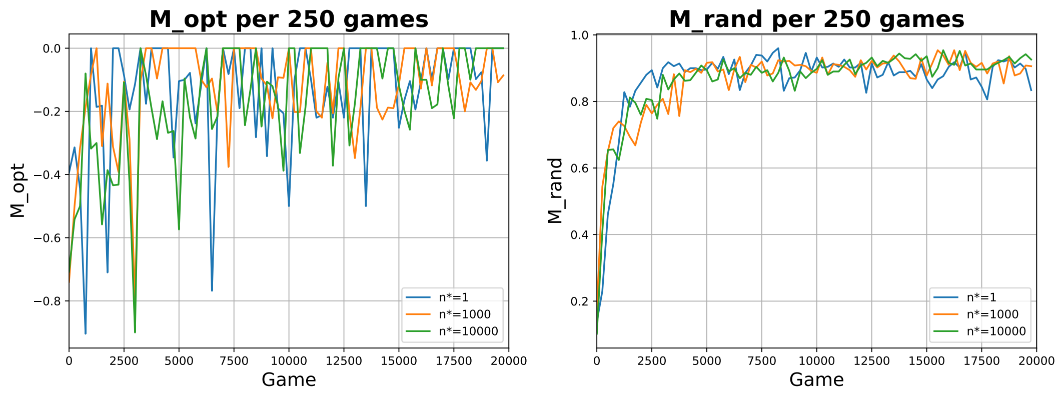


Figure 9: Q13

Question 14)

From Fig. 10, we can deduce that the agent can compete well against an opponent only if it observes similar counter-moves. For example, the agent can play against the optimal player ($\epsilon_{\text{opt}} = 0$) without losing a game when trained against `Opt`(0) or `Opt`(0.1). However, for

$\epsilon_{\text{opt}} = 1.0$, it can consistently find the optimal strategy as the corresponding M_{opt} fluctuates greatly. When the network trained against $\epsilon_{\text{opt}} = 0.5$, it can predict the optimal actions but at some point it may not infer them, its opponent chooses random action half of the time the model weights are shared between the states. In M_{rand} plot we also observe a similar pattern as the agent trained against `Opt`(0) cannot perform well against the random player, `Opt`(1). This is due to the fact that the optimal player chooses from a very limited set of actions (usually a singleton) at a given state whereas the random player is unpredictable. As a result, agents trained against `Opt`(1) and `Opt`(0.5) can quickly learn how to play against `Opt`(1). Lastly, when the model trained against `Opt`(0.1) it can still learn how to play against `Opt`(1) but it requires more games to experience random actions, hence it has a slower convergence for M_{rand}.

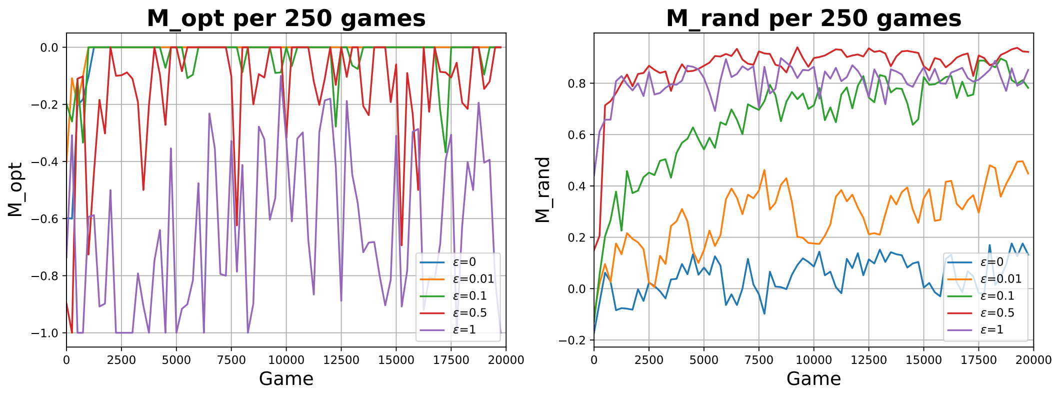


Figure 10: Q14

Question 15)

The best values are $M_{\text{opt}} = 0$ and $M_{\text{rand}} = 0.94$.

Question 16)

The DQN model can learn by self-practice since all agents could obtain $M_{\text{opt}} > -0.2$ and $M_{\text{rand}} > 0.8$ in Fig. 11. However, it is evident that not enough exploration leads to sub-optimal agents, as the agent trained with $\epsilon = 0.01$ has a lower M_{rand} than the others and its M_{opt} changes abruptly, without hitting $M_{\text{rand}} = 0$. Furthermore, the agent trained with $\epsilon = 0.5$ achieves its best M_{opt} and M_{rand} values with fewer number of steps than the others. Therefore, a sufficiently large ϵ that facilitates exploratory actions is required for learning by self-practice.

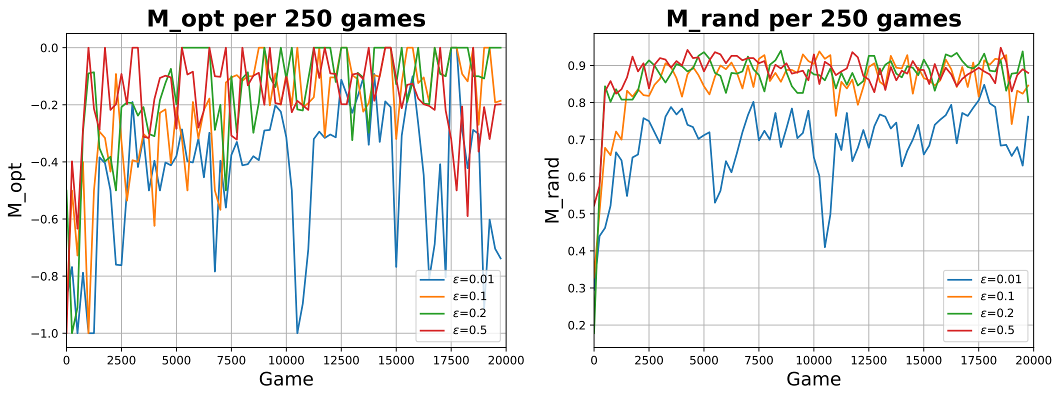


Figure 11: Q16

Question 17)

In terms of best M_{opt} and M_{rand} values, all agents behave similarly by achieving $M_{\text{opt}} = 0$ and oscillating around $M_{\text{rand}} = 0.89$ in Fig. 12. However, we observe that the agent trained with $n^* = 1000$ reaches $M_{\text{opt}} = 0$ in fewer training steps and similarly the agent trained with $n^* = 1000$ obtains $M_{\text{rand}} = 0.9$ before 2500^{th} game. Therefore, decreasing $\epsilon(n)$ may speed up the training although it may not improve the end result. We also tested with more n^* values, but did not report all of them, so they are excluded here for brevity. For full results, you can refer to our Wandb Report.

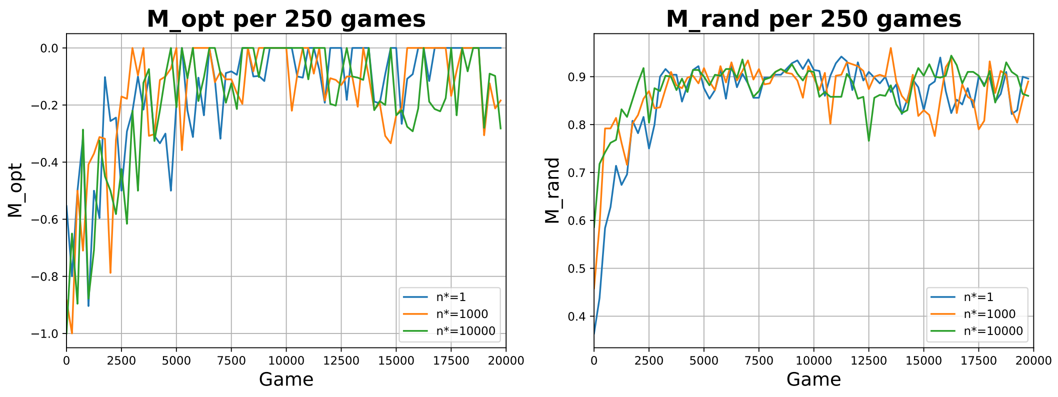


Figure 12: Q17

Question 18)

The best values are $M_{\text{opt}} = 0$ and $M_{\text{rand}} = 0.96$.

Question 19)

In this question, we used the agent in Q18 trained with $n^* = 1000$ as that has the best M_{opt} and M_{rand}. We first consider a normal case where it is player **O**'s turn (leftmost state in Figure 13) and there is one optimal action. We can see that the cell for the optimal action does indeed have the highest Q-value and by a large margin. Second, we consider a tricky state where the optimal action depends on whether the player is **X** or **O** (center and rightmost states in Figure 13). Here we can see that deep Q-learner has in fact generalized to both cases even though under the game conditions of our project, rightmost state would never happen.

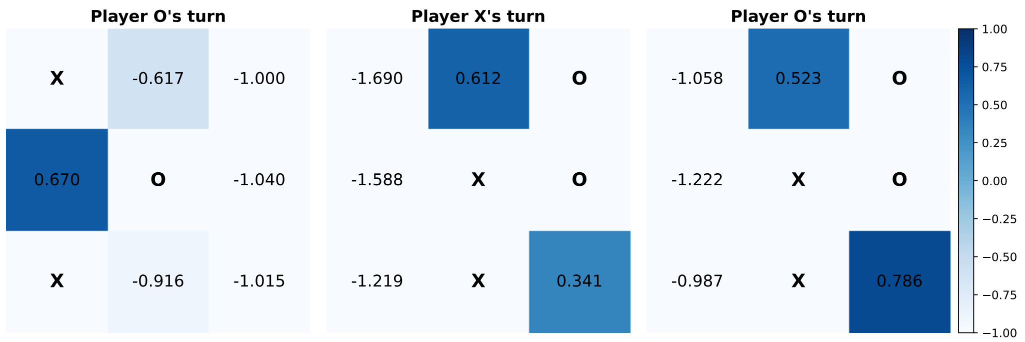


Figure 13: Q19

3 Comparing Q-Learning with Deep Q-Learning

Question 20)

The values of interest are shown in Table 1.

Question 21)

Although Q-learning and DQN models can achieve similar performance levels as in Table 1, their training time in terms of number of played games differs substantially. DQN agents requires approximately a quarter of the number of training steps compared to Q-Learning models, in both learning from experts and learning by self-practice settings. The underlying factor for this is that in DQN, the same network weights is used to map different states to action probabilities. On the other hand, Q-Learning agents create a table of state-action pairs and update the corresponding table entry only when that state-action pair is realized during the game play. Contrary to this, when the policy network is updated with samples from replay memory, states-action values that are not a part of the optimization batch are

Model	M_{opt}	M_{rand}	Training time
Q-Learning, experts	0.0	0.918	4000
Q-Learning, self-practice	0.0	0.97	4250
DeepQN, experts	0.0	0.96	1250
DeepQN, self-practice	0.0	0.96	1000

Table 1: Q20

also updated. As a result, DQN model can generalize well to the other states that it did not observe. Such an example is discussed in Q19 where the image on the right of the Fig. 13 is not possible due to the rules game, but the agent can correctly predict the best actions. Note that in DQN experiments, representing the input states as one-hot encoded tensors also helps. Furthermore, by comparing Figs. 6 and 13, we can deduce that the DQN agent can differentiate the best actions from the rest by putting distinctively more weight on them whereas we can argue that the separations of the state-action values of Q-Learning agent is comparably blurrier. Last but not the least, M_{opt} of DQN agents might fluctuate due to the reasons discussed in Q13. Hence, it might be beneficial to early stop the training, i.e. when $M_{\text{opt}} = 0$.