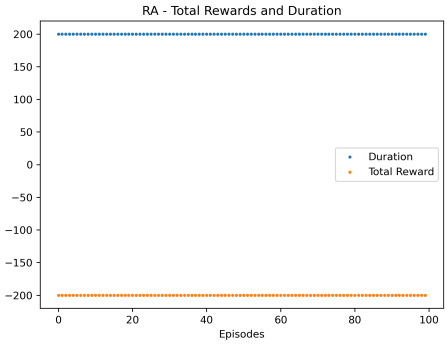
Mini-project 1 : Dealing with sparse rewards in the Mountain Car environment

1 Introduction

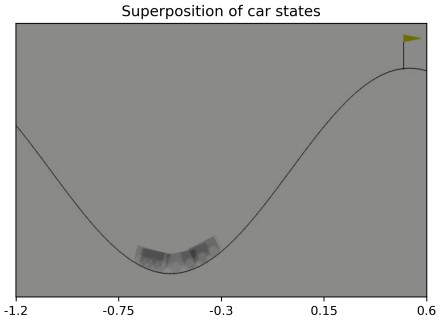
The environment considered in this project is the Mountain Car environment implemented in Gymnasium. A car is placed in a valley at a random location. There are 3 possible actions : accelerate to the left, accelerate to the right, do nothing. The goal is to reach the top of the right hill. The car is underpowered, so it needs to gain momentum by going back and forth before being able to reach the goal. The reward at each step is -1 except when the goal is reached where it is 0.

2 First steps- Random Agent

To gain intuition on the task and the environment of Mountain Car, we first implement a Random Agent that samples from the action space randomly. We run the agent for 100 episodes and present in Fig.1 the duration and reward of each episode. A random agent simply doesn't achieve the task, episode length is trivially the limit for truncation and reward is the opposite (no reward for success).



(a)



(b)

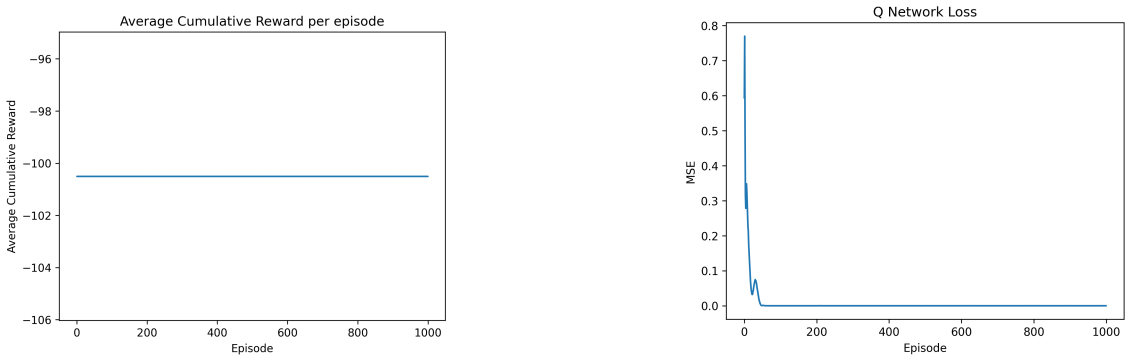
FIGURE 1 – (a) Random Agent performance. (b) Environment visualization of one episode.

3 DQN

In this section, we implement a Deep Q Network agent that estimates Q-values with a 2 layer neural network and an ϵ-greedy action policy. The Agent has a replay buffer which allows it to sample a batch of observed transitions to update the Q-network with the SARSA update rule.

3.1 No auxiliary reward

Because of the sparse reward, the agent usually doesn't solve the task for default episodes truncated at 200 steps. If we allow longer episodes, the agent is equivalent to random until it reaches once the goal. Once it experiences one success, the non trivial reward influence it's learning and it's performances improve [A.1]. This random success triggering learning can happen in 200 steps episodes but is much less probable. In Fig.2 we plot the mean cumulative reward and loss per episode for the DQN agent with default parameters that doesn't solve the task. Its average cumulative reward is then constant for every episode, while its loss decreases as the network learns the local Q-values of the random agent.



(a) (b)

FIGURE 2 – (a) Average cumulative reward per episode and (b) loss of the network learning Q-values from input states along 1000 episodes for DQN agent.

3.2 Heuristic reward function

We create an auxiliary reward function that linearly favours high velocity states to account for the fact that the car needs momentum to reach the top of the mountain, and linearly favours position states close to the goal state. This introduces two scalar parameters : the ratio between velocity and position reward in the heuristic reward, and the scaling of the heuristic reward in the total reward (reward factor). Velocity and position reward are normalized (by the range of accessible values), and their ratio is then set at 0.5.
In Fig.9 we plot the results for DQN agent with the heuristic reward defined and a reward factor of 1. With this reward factor, the heuristic reward per step is of the order and balances the environment reward. The agent quickly solves the task, and its performance clusters around episodes of 150 steps. Of course the agent is not perfect, but after some time it's cumulative number is success is almost linear with a slope close to unity. The behavior of the training loss is surprising, and reflects the fact that the heuristic reward is not linked to the notion of success in an episode and then introduces variability in the SARSA update, scaling with the reward factor [A.2]. Velocity reward accounts for favouring exploration, while position reward introduces bias to the right and breaks the symmetry of the exploration of the precedent agent. This behavior is monitored by the different reward factors [A.2]

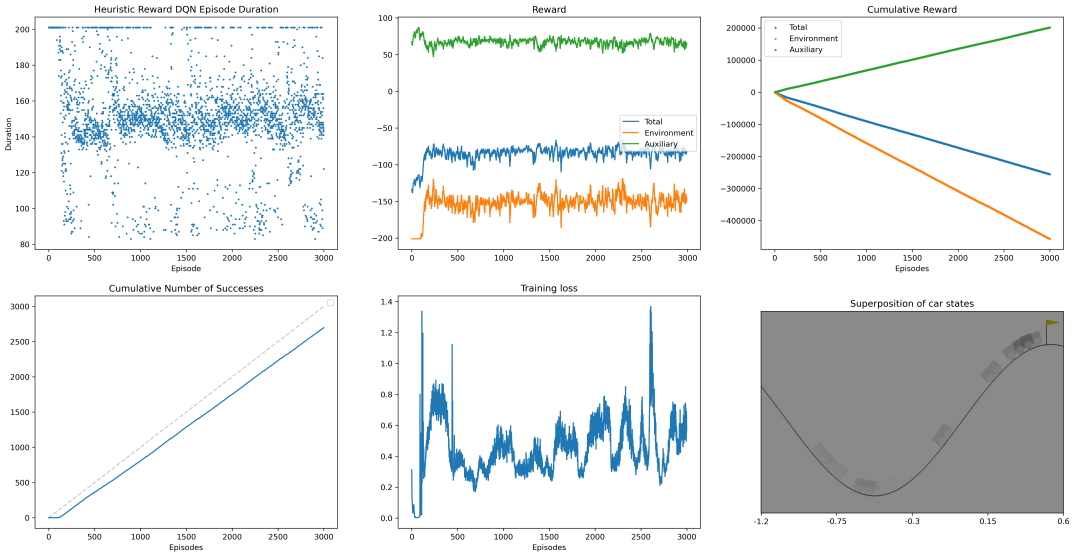


FIGURE 3 – DQN with heuristic reward results for reward factor 1.

3.3 Non domain-specific reward

To deal with sparse rewards with a less environment-specific solution than the heuristic reward, we now implement Random Network Distillation to introduce an intrinsic reward rewarding the agent for reaching less-frequently encountered areas in the state space. In practice, a predictor network is trained to match the predictions of a fixed target network for each state, getting better for more encountered states, intrinsic reward being the squared difference between predictor and target output again introduced with a reward factor.
Before evaluating target and predictor network, we normalize the states so that the reward values doesn't depend on the metric used to describe positions and velocities. This is important to have comparable reward behavior both between different states in an environment and to change environment definition. After evaluating them, we also normalize the squared difference between predictor and target so that the reward doesn't depend on the randomness of the initialization of the fixed target network. As the intrinsic reward is clamped between -5 and 5, a good reward factor would bring it to the order of the environment reward, order 1, so would be around 1/5. A reward factor too small would get back to the case without intrinsic reward, a reward factor too big rewards too much exploration and learn sub optimally even if it can solve the task [A.3].
In Fig.10 we plot the results for DQN agent with RND and a good reward factor of 0.2. The performance and behavior of DQN agent with RND are close to the case with heuristic reward, with cumulative number of success linear with a slope slightly lower than unity. Again, the second source of reward balances the environment reward and enhances exploration. Intrinsic reward is not influence by success, so the agent learns again very quickly and the training loss inherits this variability. In the end, the process of a second source of reward introduces a bias to the same type of behavior (exploratory) and the effects are similar even in reactions to reward factor modification, with the advantage of being generalizable to different environments.

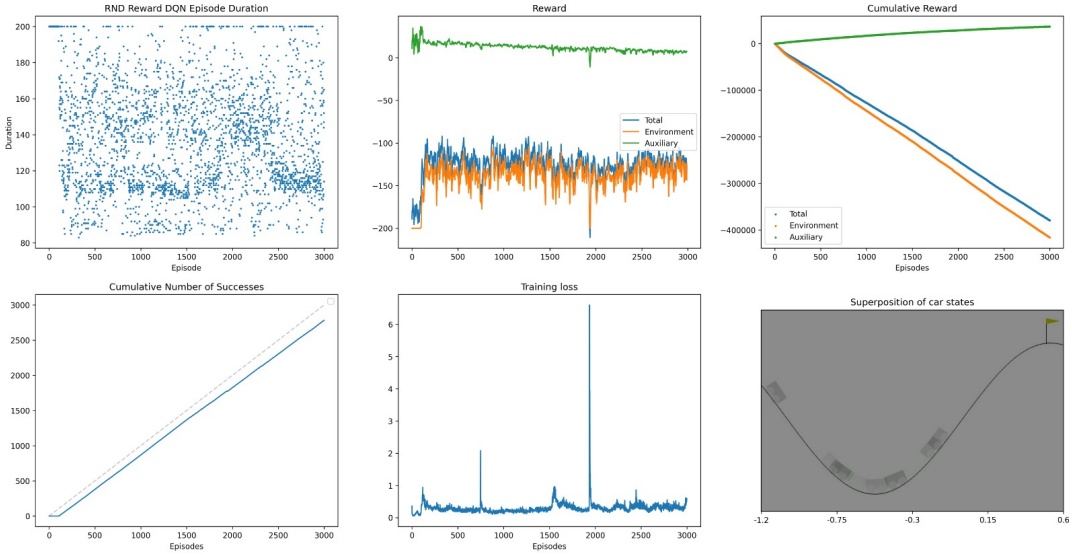


FIGURE 4 – DQN with RND results for reward factor 0.2.

4 Dyna

In this part, we implement the model based approach Dyna. During training, this agent discretzes the state space and build estimates of the transition probabilities, the expected rewards for every state, action pair and Q-values.

4.1 Results

As we can see in Fig. 11, the model-based agent solves the task extremely well and learns consistently with a linear cumulative success trend. We can see the total reward per episode steadily climb and an interesting duration pattern seems to form at the end of training.

Having played around with the discretization size for the position and velocity, we see the difference between smaller and bigger bin sizes. Bigger bin sizes discretise the state space into less bins which allows the model to learn much fast but does not converge to the best solution. Smaller bin sizes learns much more slowly and doesn't seem to converge in learning after a large number of epochs and this is also computationally expensive to scale our transition and expected reward estimates. Dyna seems to solve the

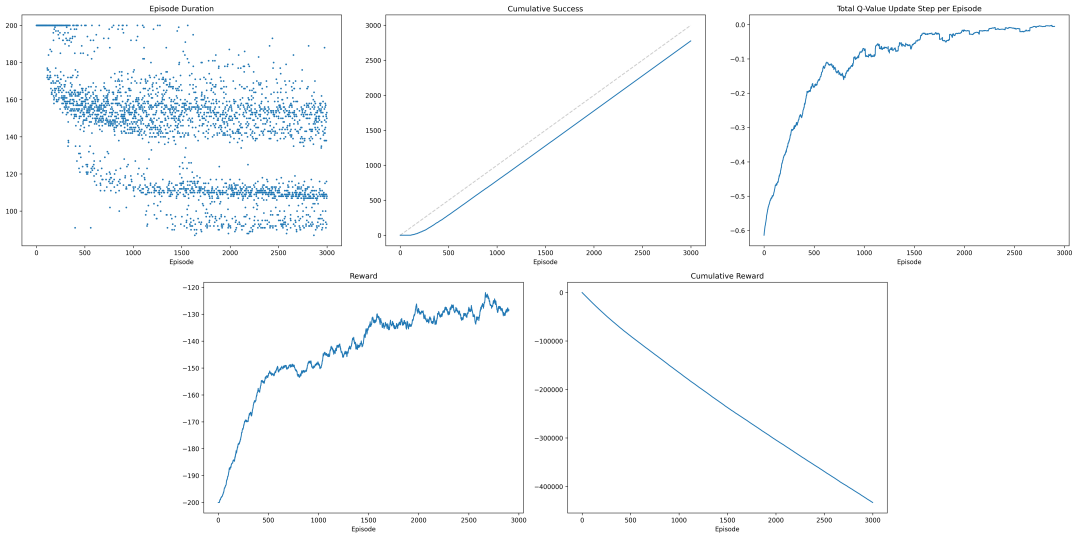


FIGURE 5 – Dyna Results.

task without any auxiliary reward which is due mainly to the state discretization ; this smaller state-space, makes our reward infinitely less sparse and easier for an epsilon-greedy policy to achieve.
We can clearly see a duration pattern to be forming at very large episode numbers : clusters of similar durations being formed and this seems to be due to the agent learning two main optimal trajectories and policies. Perhaps even further differentiation into three optimal trajectories [B.1].
By plotting Q-values and trajectories on the states phase space (Fig.6), we see that the distribution of Q-Values is negative, which makes sense as the environment reward is negative. We remark that trends of high velocity in negative position values are key to making it to the goal, which completely explains the found clusters of durations, which are essentially 2 optimal trajectories being found and correspond to either starting left or starting right after initialization. Starting right seems to be the best policy as it allows for minimal pendular movement and the fastest exiting spiral.

4.2 Comparison with DQN

As a final analysis, we compare in Fig.7 the training and absolute performances of our three main agents : DQN agent with heuristic reward, DQN agent with RND, and Dyna agent. As already pointed before, agents with a second source of reward, not success dependant, starts to learn sooner. This is also clear in plots of the training performance without running average that crops the first episodes (Dyna stagnates shortly in episodes with default -200 reward). But after enough training time, these second reward based agents saturates to lower performances than the Dyna agent. In the end, when testing performances of trained agents on new episodes, the results attained by DQN with second source of reward are similar, while the model based Dyna agent stably reaches higher environment reward.

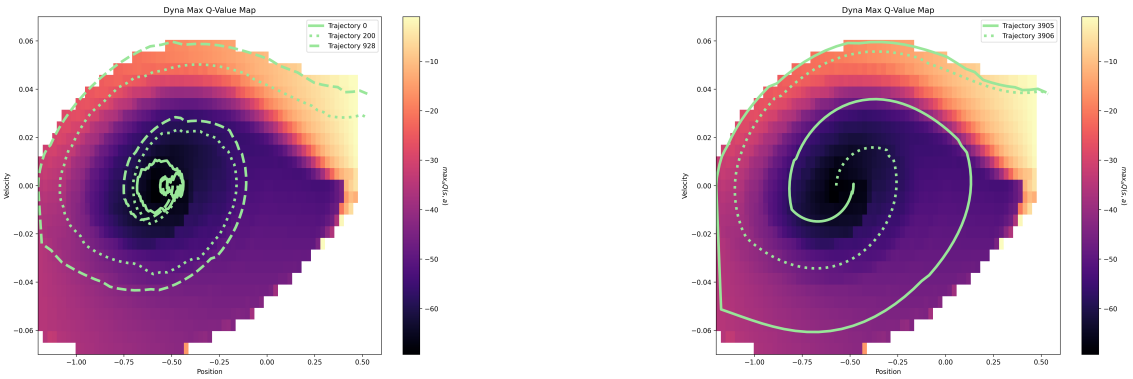


FIGURE 6 – Dyna Maximum Q-Value State Space Map with (a) different learning trajectories and (b) learned optimal trajectories

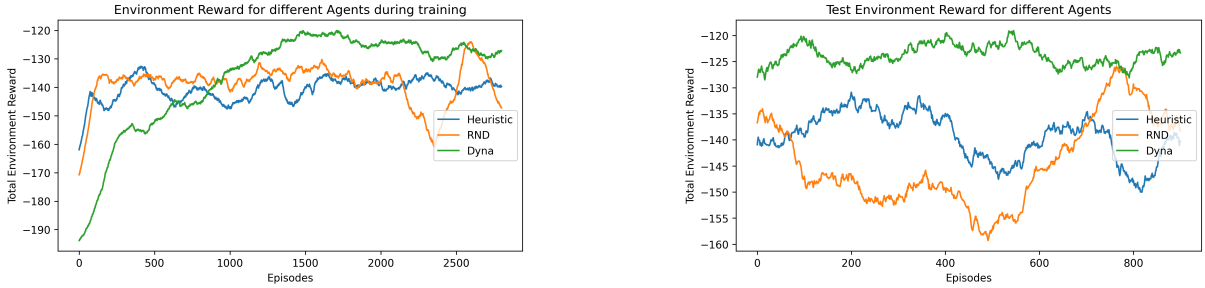


FIGURE 7 – (a) Training performance (environment reward) for DQN with heuristic reward, DQN with RND and Dyna agents. (b) Testing performance (environment reward) of the three trained agents on new episodes with greedy policy.

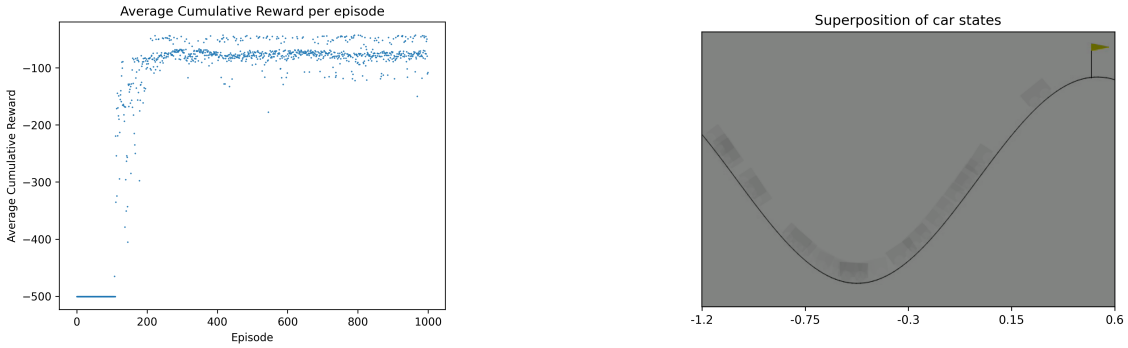
5 Conclusion

In conclusion, DQN leverages deep learning to handle high-dimensional state spaces effectively without needing a model of the environment. Its second source of rewards, heuristic or intrinsic with RND, allows for faster learning in situation of sparse reward by enhancing exploration when needed. This probably makes it more suitable for rapidly changing and continuous environments. On the other hand, Dyna uses discretized observations to reconstruct a representation of the environment through transition, reward and Q-values estimates. Without need for a choice in bias with reward guidance, it adapts to sparse rewards by learning the non reward specific understanding of the environment. In the end, these models are task specific. In the mountain car problem, we observe that model free approaches learns faster but less episode consistent and less optimal solutions. Whilst the model based approach learns slower but more global, episode robust, optimal solutions within the limits of their discretization space.

A DQN

A.1 No auxiliary reward

Fig.8 presents the performance of DQN agent on episodes with truncation at 1000 steps. After learning is triggered, the performance of DQN agent implemented at this point improves then converges and saturates to tasks solved in around more than 200 steps. Some insights on how the agent solves the task is given by a visualisation of the car along a successful episode, it widely and symmetrically explores the position states.



(a) (b)

FIGURE 8 – (a) Average cumulative reward per episode for 1000 episodes truncated at 1000 steps for DQN agent. (b) Environment visualization over one successful episode.

A.2 Heuristic reward function

For a very small reward factor, the agent goes back to the one with no auxiliary reward [3.1]. For a very big one, the heuristic reward takes over the environment one and is enough to solve the task but a lot less optimal. The agent almost doesn't learn by success and the training loss is then even more variable. The balance in the velocity an position rewards becomes important : as the agent gets close to the goal (up the mountain) its speed necessarily decreases (not favoured by speed reward) and it often doesn't make the last step to reach the goal (no experience of success). A way to tackle this problem in this regime would be to look further into the balance between position and velocity bias.

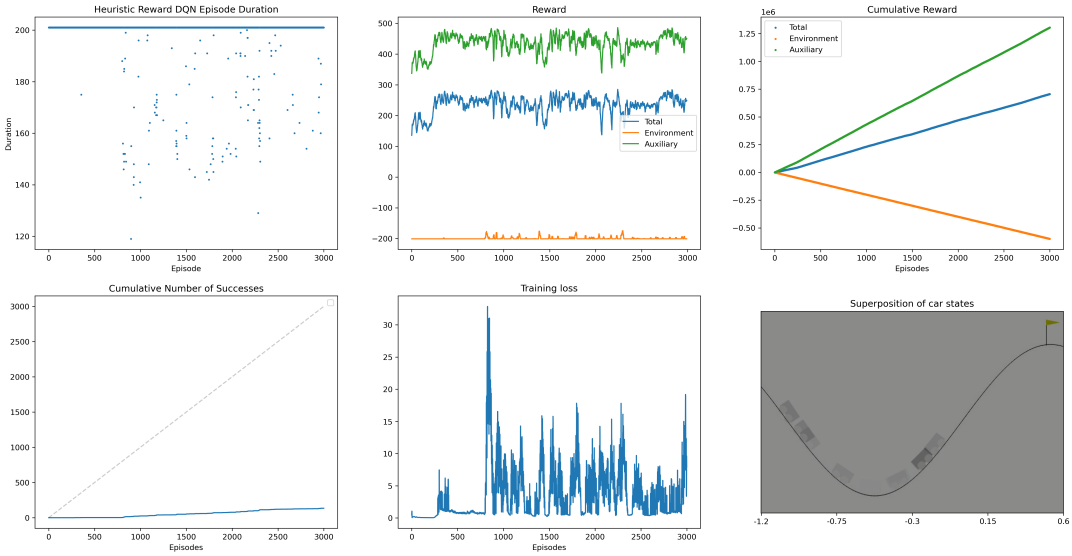


FIGURE 9 – DQN with heuristic reward results for reward factor 5.

A.3 Non domain-specific reward

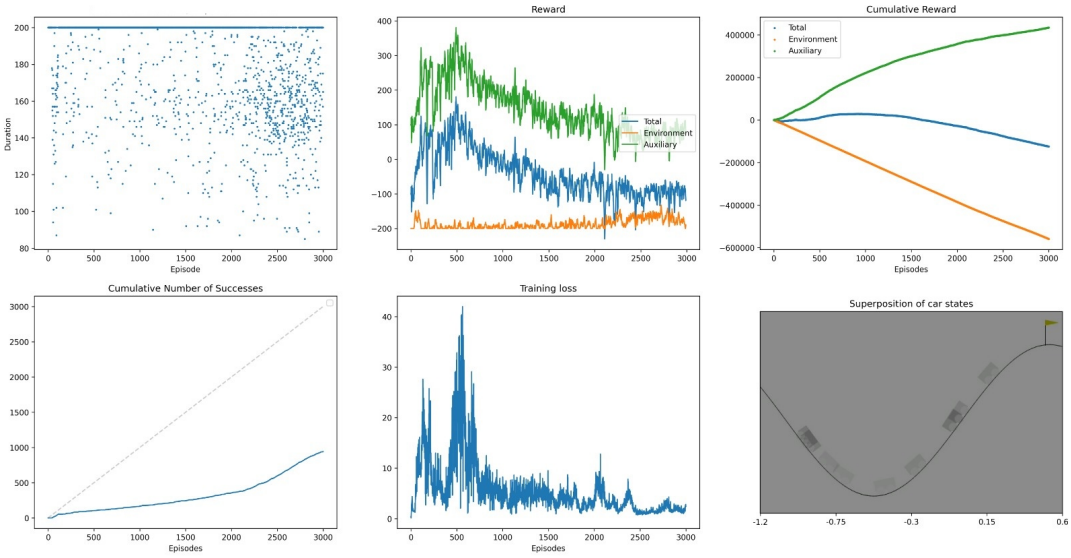


FIGURE 10 – DQN with RND results for reward factor 1.

B Dyna

B.1 results

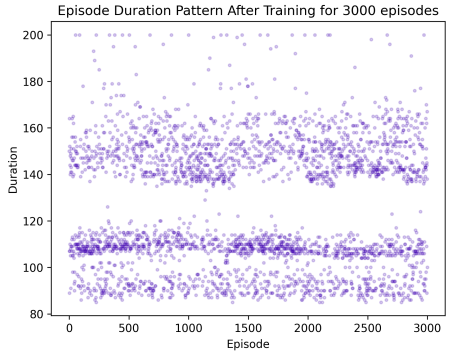


FIGURE 11 – Dyna duration patterns.