Reinforcement Learning : Autonomous Driving

BACHER Quentin, PIERRON Marie

December 7, 2025

Contents

1 Introduction

1.1 Problem Description and Game Context

This project aims to develop an autonomous driving policy using **Deep Reinforcement Learning (DRL)** within a custom **Pygame** simulation environment. The fundamental objective is to train the agent (the vehicle) to **maximize the distance traveled** over a 20-second episode while strictly adhering to **safety constraints**, including avoiding collisions, running red lights, and lane departures. The agent interacts with an environment whose internal state is continuous (position, speed), but which is interfaced with the algorithm via a normalized/discrete state space and a discrete action space.

The approach of autonomous driving via **Reinforcement Learning (RL)** is particularly relevant because it is difficult to model the problem as a supervised learning task due to strong interactions with the environment (other vehicles, pedestrians) and the uncertainty in the behavior of other actors. RL allows the agent to learn from its own experience and handle stochastic control problems. Recent surveys (e.g., Ravi Kiran et al., 2020) emphasize both the promise of DRL for driving and the practical difficulties around safety, validation, and sample efficiency.

Within the DRL framework, the agent operates in an environment where the perceived state is used for **Planning**, which is the hardest task as it integrates recognition and prediction to choose the best sequence of future actions. This project focuses on learning an effective Planning policy.

The implementation of DRL for autonomous driving faces several significant challenges that complicate its successful application:

- **Safe-Exploration**: Balancing exploration with safety constraints (collisions are catastrophic).
- **Sample Complexity**: DRL often requires many interactions before reaching good performance.
- **Credit Assignment**: Consequences of actions may appear many time steps later (Sutton & Barto, 2018).
- **Action Discretization**: Discretizing steering and throttle can lead to jerky behavior if granularity is insufficient (Mnih et al., 2015).
- **Function Approximation**: Large state-action spaces require deep networks that generalize across similar situations (Mnih et al., 2015).

1.2 Environment and Agent Definition

1.2.1 The Agent and the Environment

The agent is the **autonomous vehicle** (*car*), and the environment is the **driving simulation** (*DrivingEnv*). The environment is continuous in its internal state, but the agent interacts using a discrete/normalized observation vector and discrete actions.

1.2.2 State Space (S)

The state is a normalized vector of dimension $N = 8$, where values are typically between 0 and 1, calculated by the `_get_observation()` function:

$$S = (obs\_type, obs\_dist, car\_speed\_norm, steer\_norm, car\_x\_norm, car\_y\_norm, lir, time\_left).$$

- `obs_type`: Type of the nearest obstacle (normalized $[0, 1]$).
- `obs_dist`: Normalized distance to the nearest obstacle ($\max 300$m).
- `car_speed_norm`: Vehicle speed (normalized by 25.0 km/h).

- `steer_norm`: Current steering angle (normalized $[0, 1]$, 0.5 is straight).
- `car_x_norm`: Normalized X position.
- `car_y_norm`: Normalized Y position (lateral position within the lane).
- `lir`: "Light is red" (1.0 if the front traffic light is red, 0.0 otherwise).
- `time_left`: Remaining episode time (normalized $[1, 0]$).

Why an 8-dimensional observation? We intentionally use a *compact* observation to keep learning tractable and sample-efficient. The vector combines (i) **safety-critical cues** (nearest obstacle type and distance, red-light indicator, lateral position), (ii) **control state** (speed and steering), and (iii) **progress/time** signals (position and remaining time). This pragmatic design reduces sample needs while preserving key decision variables.

1.2.3 Action Space (A)

The action space is strictly **discrete**, consisting of $5 \times 3 = 15$ possible combinations, derived from 5 steering commands (from -0.6 to $+0.6$) and 3 throttle commands (-1.0 braking, 0.0 coasting, $+1.0$ accelerating).

Why discretize controls into 5×3? DQN-style methods are naturally designed for discrete actions (Mnih et al., 2015). Using 5 steering levels provides enough granularity to correct lane position without exploding the action count, while 3 throttle levels capture the main longitudinal decisions (brake/coast/accelerate). The resulting 15 actions keep the search space manageable while enabling non-trivial driving behavior.

1.2.4 Reward Function (R)

The reward function is **dense**, granting positive rewards for speed and safe distance maintenance, while imposing **severe, episode-terminating penalties** (up to -10.0) for dangerous events (pedestrian/car collision, running a red light, or leaving the road). Empirically, overly large reward magnitudes caused unstable training (high variance of returns), so rewards are clipped to remain within a moderate range. The project's success is defined by high and stable cumulative reward in evaluation mode ($\epsilon = 0$), while keeping catastrophic failures low.

1.3 Project Goals

The project's success is defined by achieving a stable, high cumulative reward over the maximum episode length (20 seconds) while operating in **evaluation mode**. Performance is primarily measured by the average total reward and the avoidance of episode-ending penalties such as collisions and road departures. We set an ambitious internal target of $\mathbf{500}$ cumulative reward to encourage rapid learning and strong performance.

2 Methodology

2.1 Algorithms and Equations

The methodological focus is entirely on Deep Reinforcement Learning (DRL), specifically implementing the **Double Deep Q-Network** (**Double DQN** / **D-DQN**) algorithm (van Hasselt, Guez & Silver, 2016), building on the original DQN framework (Mnih et al., 2015). Double DQN is chosen over standard DQN for its enhanced stability and its ability to mitigate the **overestimation of Q-values**. It achieves this by employing two separate networks: the **Policy Network** (θ) selects the optimal action in the next state, while the separate **Target Network** (θ^-) evaluates the value of that chosen action. This decoupling provides a more stable learning target.

The core of this value-based approach is the **Action-Value Function** $Q(s,a)$, which estimates the expected discounted return of taking action a in state s:

$$Q_\pi(s,a) = \mathbb{E}_\pi \left[\sum_{k=0}^{H-1} \gamma^k r_{k+1} \,\middle|\, s_0 = s, a_0 = a \right].$$

Why Double DQN Instead of Standard DQN? Standard DQN can suffer from **overestimation bias** because the same network is used to both *select* and *evaluate* the next action. Double DQN reduces this bias by using the policy network for action selection and the target network for action evaluation, producing more reliable temporal-difference targets and typically improving training stability in noisy environments.

To further stabilize learning, the target network is updated via a **Soft Update** mechanism (Polyak-style averaging), with a factor $\tau = 0.005$, preventing abrupt shifts in the learning signal:

$$\theta_{\text{target}} \leftarrow \tau \cdot \theta_{\text{policy}} + (1-\tau) \cdot \theta_{\text{target}}.$$

This soft update is preferred over a hard update because it ensures the target network parameters evolve gradually. This gradual change contributes significantly to training stability by maintaining a lower variance in the loss function.

Why Off-Policy Learning (DQN) Instead of On-Policy Methods? Our setting benefits from **off-policy** learning because it allows reusing past experience multiple times. With a replay buffer, transitions collected under exploratory behavior (ϵ-greedy) can be sampled repeatedly, improving sample efficiency and reducing temporal correlations in sequential driving data (Sutton & Barto, 2018; Mnih et al., 2015). We use an experience replay buffer to store past transitions and train the network on randomly sampled mini-batches, which breaks temporal correlations in driving trajectories and improves sample efficiency and training stability. This is particularly relevant here because unsafe exploration (collisions, red-light violations, road departures) can be frequent during early training: off-policy learning lets the agent learn from both successful and failed trajectories without requiring fresh rollouts at every update.

Why DQN Rather Than Tabular Q-Learning? Although DQN follows the same principle as Q-learning, **tabular Q-learning** becomes impractical in our setting because the state is continuous/ normalized (positions, speeds, distances), implying an extremely large number of possible configurations. A sufficiently fine discretization would create a huge table and slow learning. DQN replaces the table with a neural network that approximates $Q(s,a)$ and generalizes across similar states, which is essential for our 8-dimensional observation space.

2.2 Environment and Agent Roles

The simulation relies on two key classes:

- **DrivingEnv (Environment)**: This class formalizes the sequential decision process as a **Markov Decision Process (MDP)**. Its function is to manage the internal dynamics of the car and the simulation world, specifically: accepting an action a_t from the agent, calculating the resulting next state s_{t+1} and the reward r_t based on the vehicle dynamics and safety constraints, and determining if the episode is done or truncated.

- **DQNAgent (Agent)**: This class encompasses learning intelligence. Its function is to select the action a_t (using the ϵ-greedy strategy), to store the transition $(s_t, a_t, r_t, s_{t+1}, \text{done})$ in the *replay memory*, and to execute the learning step (*replay*) by minimizing the loss between the predicted $Q(s, a)$ and the Double DQN target.

Road boundaries (design choice). To keep training focused on obstacle-handling (red lights, pedestrians, and vehicles), we introduced **road boundaries** that act as a soft constraint on lateral motion. Instead of immediately terminating an episode when the car touches or slightly exceeds the road limits, the environment **clamps the vehicle back inside the lane** and increments a boundary-hit counter. We then apply a **small penalty** when the boundary is touched, encouraging the agent to correct its trajectory and remain centered while avoiding the overly harsh learning signal of frequent early terminations. This choice improves learning stability by preventing the agent from spending most episodes failing due to lane departures, while still preserving an incentive to drive safely within the road.

2.3 Network Structure and Function

The DQN Network Architecture (`DQNNetwork`) utilizes a straightforward yet deep **Multilayer Perceptron (MLP)** as the function approximator for the Q-function. The network is designed to efficiently map the 8-dimensional state vector input to the 15 discrete output Q-values.

Network Architecture:

- **Input Layer**: Accepts the **8-dimensional** normalized state vector.
- **Hidden Layers (3)**:
  1. **Layer 1**: 256 neurons. Handles initial complex feature extraction.
  2. **Layer 2**: 256 neurons. Further processes the high-dimensional internal representation.
  3. **Layer 3**: 128 neurons. Condenses features before mapping to output actions.
- **Activation Function**: All hidden layers incorporate the **Rectified Linear Unit (ReLU)** activation function, introducing non-linearity.
- **Output Layer**: A fully-connected linear layer that produces the **15 Q-values**, each serving as an estimate of the expected discounted return $Q(s, a)$ for one of the 15 possible actions in the current state.

This deep architecture enables the network to learn the complex, non-linear relationship between the perceived environment state and the optimal driving actions.

2.4 Parameters and Training Setup

Table 1: Key Double DQN Algorithm Parameters

Category	**Parameter**	**Value**	**Description**
Learning	Discount Factor (γ)	0.99	Set high to emphasize future rewards, encouraging long-term planning.
	Learning Rate (lr)	1.00×10^{-5}, 1.00×10^{-4}	Compared to study speed/stability trade-off (Adam optimizer).
	Loss Function	MSELoss	Minimizes the mean squared TD error between predicted and target Q-values.
Exploration	Initial Epsilon (ϵ_{start})	1.0	Starts with 100% random action selection to build initial experience.
	Epsilon Minimum (ϵ_{min})	0.05	Minimum probability of random exploration maintained after decay.
	Epsilon Decay Rate	0.995	Exponential factor applied after each episode to gradually reduce exploration.
Memory/Stability	Replay Memory Size	150	Stores past transitions to reduce temporal correlations and enable off-policy reuse.
	Batch Size	128	Number of samples randomly drawn from the replay memory for each training step.
	Target Update Mechanism	Soft Update($\tau = 0.005$)	Gradually updates the target network weights, enhancing stability (preferred over hard updates).

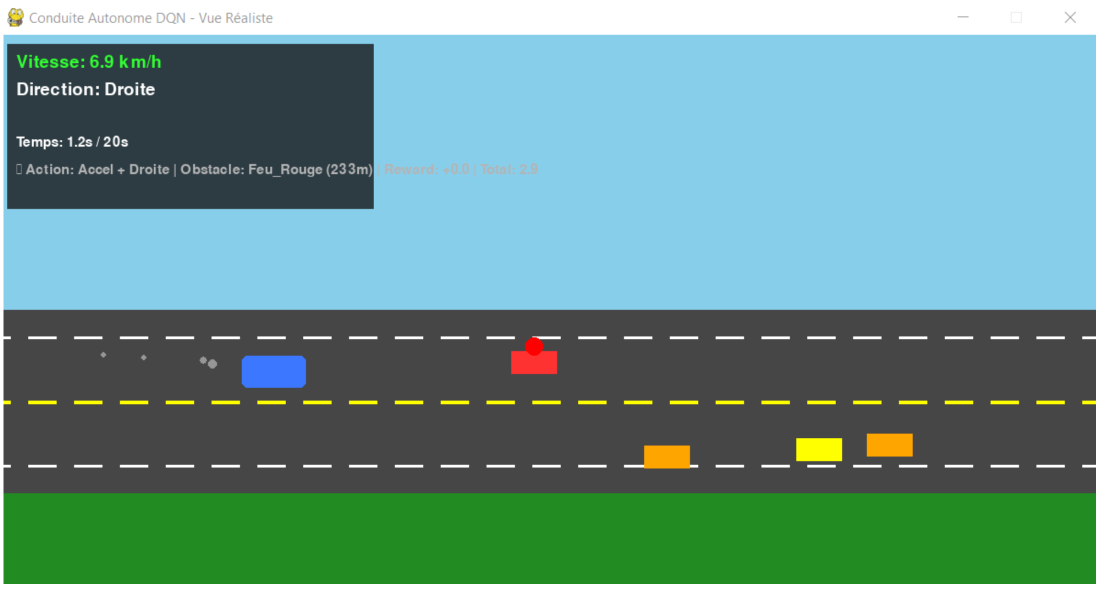
3 Results

3.1 Evaluation Metrics

The evaluation of the trained Deep Q-Network agent's performance is meticulously tracked using the `plot_training_results` function, which generates four key graphs for assessing learning progress:

- **Rewards per Episode (Raw)**: Shows the inherent variability and general trend of the returns.
- **Smoothed Rewards (Window=20)**: Applies a moving average to visualize the policy's **convergence** over episodes. A steadily increasing and stabilizing curve indicates successful learning.
- **Training Loss**: Displays the average Mean Squared Error per episode, expected to **decrease** as the Policy Network's predictions stabilize relative to the Target Network's estimations.
- **Evaluation Scores**: A critical metric that periodically assesses the performance of the pure (non-exploratory) policy, directly reflecting the actual quality of the learned behavior.

3.2 Policy and Reward Visualization



```
Vitesse: 6.9 km/h
Direction: Droite

Temps: 1.2s / 20s
 Action: Accel + Droite | Obstacle: Feu_Rouge (233m) | Reward: +0.0 | Total: 2.9
```

Figure 1: Visualization example

The Policy and Reward Visualization phase provides essential qualitative insight into the agent's learned behavior via the `demo_agent` function. This function offers a **visualization** (in the end) of the running policy within the Pygame environment, displaying crucial data points as the simulation progresses. Key metrics presented include:

- The speed and current steering angle of the vehicle.
- The specific action chosen by the agent (decoded from the Q-function output using `ACTION_MAP[action]`).
- The instantaneous and cumulative rewards received.

Furthermore, the visualization immediately displays terminal messages upon failure, clearly indicating the cause of the episode's termination, such as a collision or road departure.

3.3 Results of Simulation with Learning Rate

We are comparing Learning Rates (lr) of 10^{-5} and 10^{-4} to identify the optimal balance between training stability and convergence speed in our DRL setup.

- A **low lr (10^{-5})** ensures high stability and smooth weight updates, preventing oscillation, but risks slow convergence or getting stuck in a suboptimal, shallow minimum.
- Conversely, a **higher lr (10^{-4})** promotes faster learning but carries a significant risk of instability and divergence, where large steps cause the optimization to overshoot the optimum.

3.3.1 Analysis of Results with lr $= 10^{-5}$

The D-DQN training results using a very low Learning Rate (lr $= 10^{-5}$) confirm a **highly stable but slow convergence process**, typical of a cautious optimization strategy.

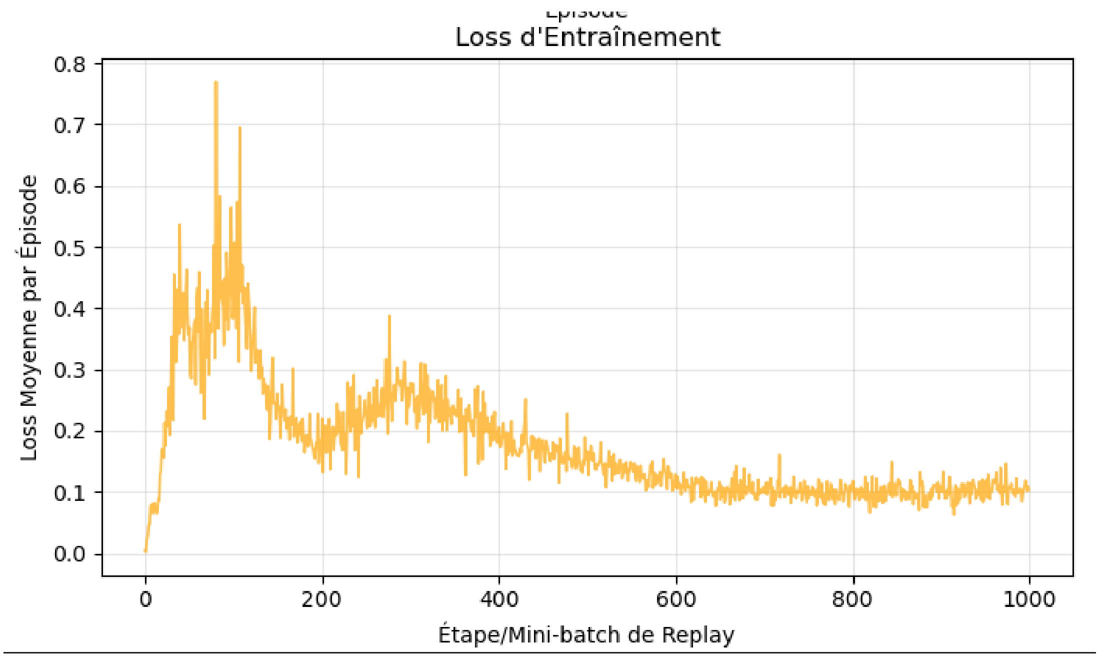


Figure 2: Loss curve

- **Training Stability and Loss**: Stability is evidenced by the **Loss Trend**, which decreases slowly and smoothly from around 0.54 to stabilize around 0.10 after Episode 600. This indicates the low lr successfully prevents divergence or oscillation, although this stability is achieved at the cost of convergence speed.

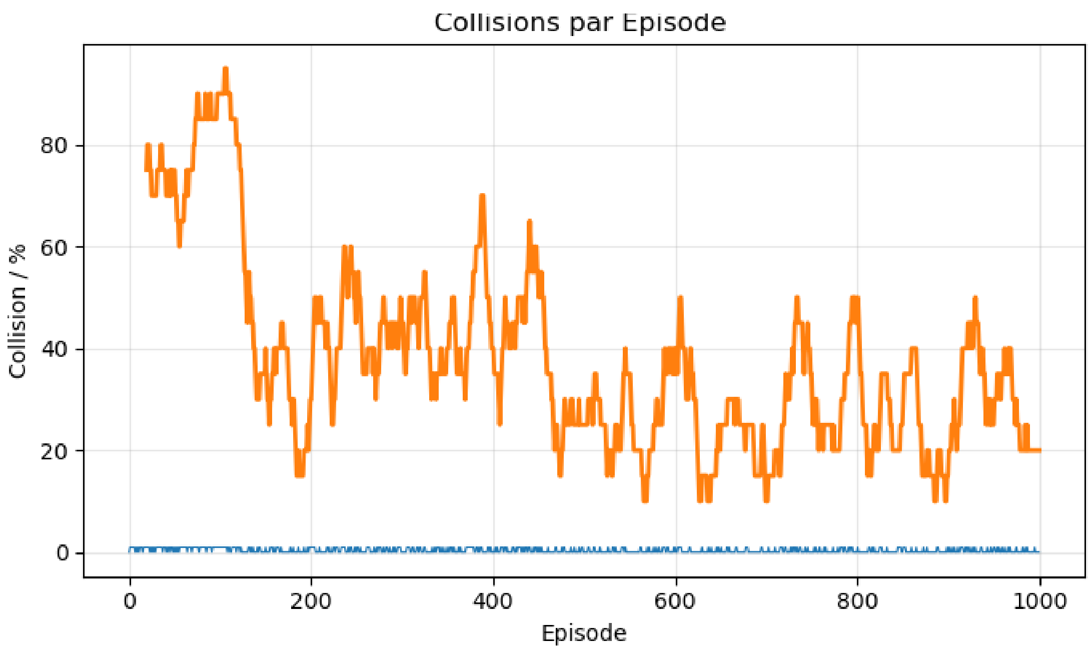


Figure 3: Collision curve

- **Collision Analysis**: The collision analysis reveals the policy's fundamental fragility despite the stable loss function. Initial collisions, peaking near 90%, rapidly drop to the $20-50\%$ range after Episode 200, confirming basic learning. However, the collision rate persistently spikes up to 50% throughout late training, indicating a **lack of robust safety generalization** even with minimal exploration.

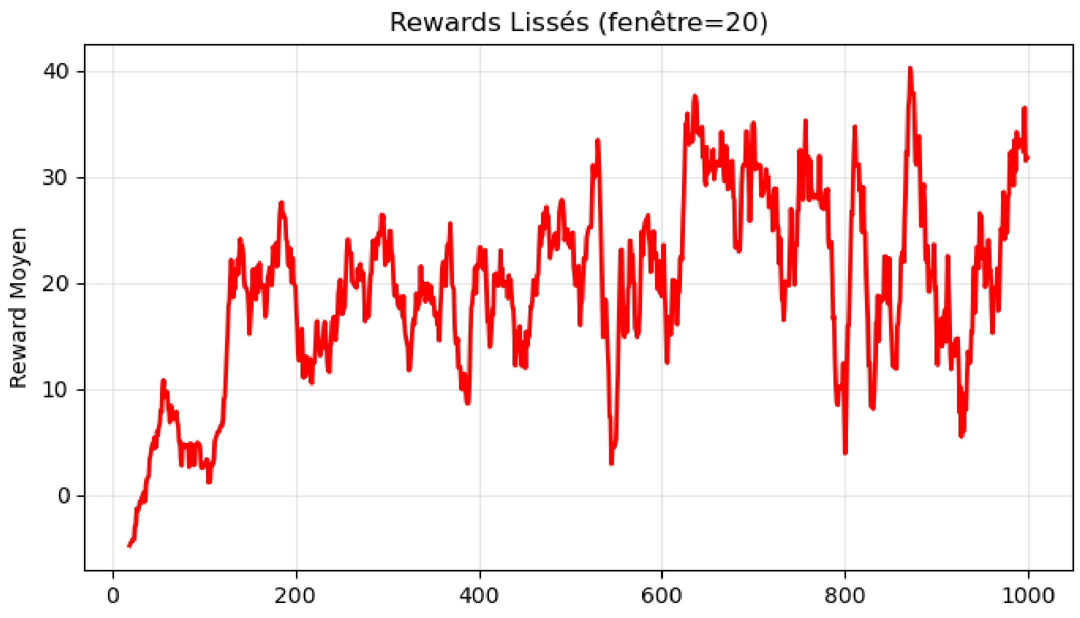


Figure 4: Rewards curve (Average Mobile)

- **Policy Performance**: While raw episode rewards (R) are volatile, the **Evaluation Scores** show moderate improvement from initial poor performance (around -2 to 2) to the $20-35$ range in mid-training. Critically, in the late-training phase, the evaluation scores remain inconsistent $(\max 49.6)$, **failing to reach the target of 500.**

This suggests the extremely small step size was **insufficiently aggressive** to optimize the Q-function to master the full, complex, 20-second driving task. Ultimately, lr $= 10^{-5}$ provided a foundational, moderately successful policy, but its caution led to **incomplete convergence** within 1000 episodes, necessitating a comparison with lr $= 10^{-4}$ to attempt to break this performance plateau.

3.3.2 Analysis of Results with lr $= 10^{-4}$

Analysis of Results with lr $= 10^{-4}$ The training run with lr $= 10^{-4}$ demonstrated a clear trade-off between speed and stability.

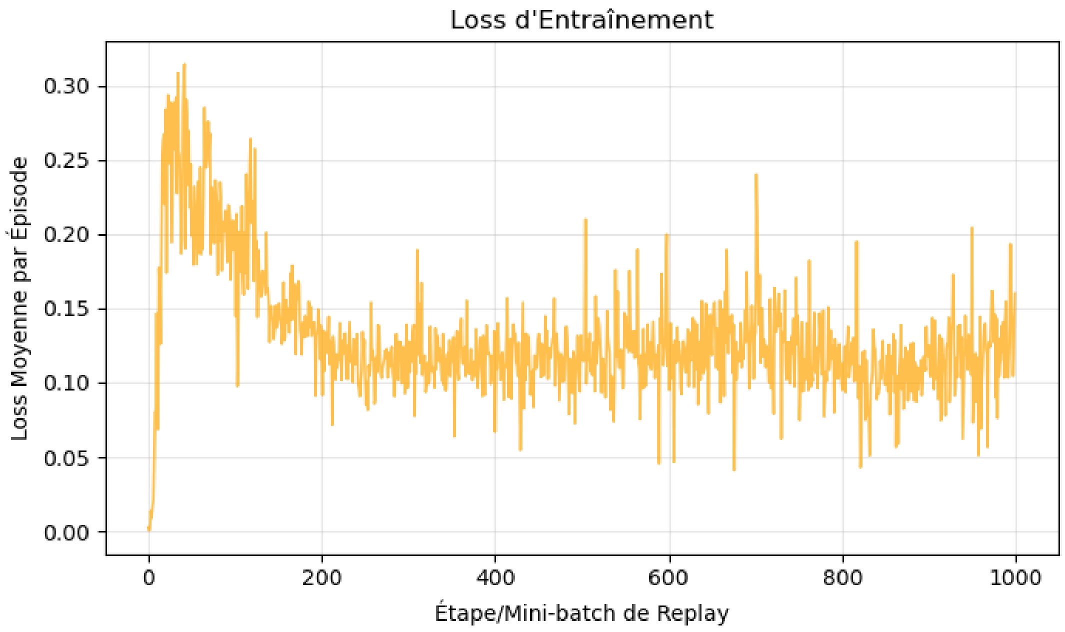


Figure 5: Loss curve

- **Loss Oscillation (Stability)**: The loss function decreased faster but exhibited significant **oscillation** (e.g., jumping from 0.062 to 0.160), confirming that the large learning steps were often overshooting the optimal minimum, leading to confirmed instability.

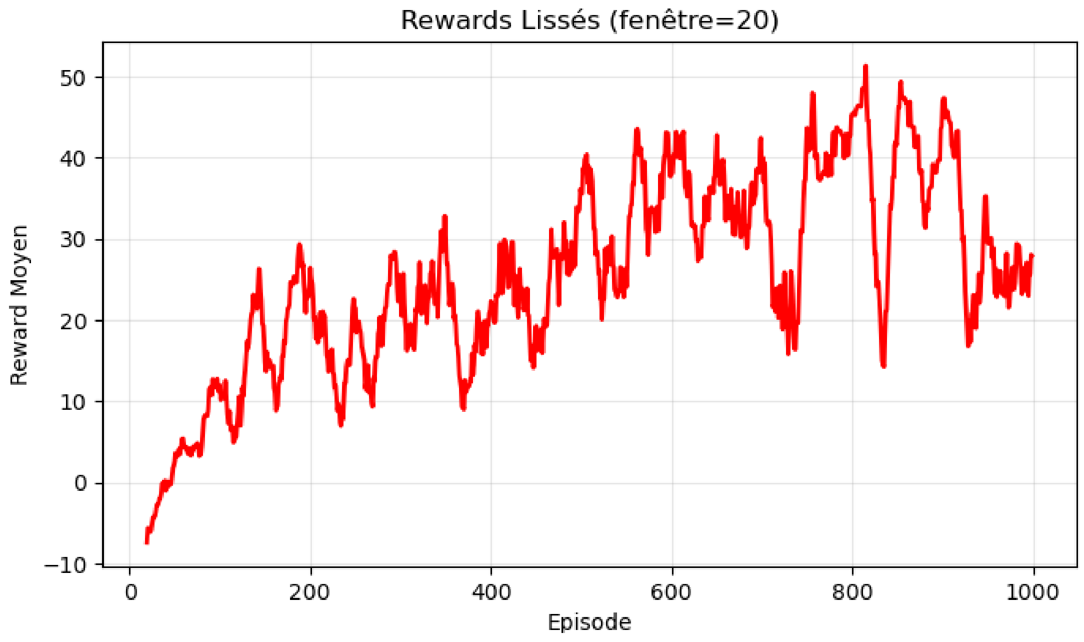


Figure 6: Rewards curve (Average Mobile)

- **Peak Performance**: The agent achieved a **higher peak evaluation score ($\max 59.0$)**. In fact, Raw rewards (R) are highly volatile, suggesting periods of rapid policy improvement and successful long-distance travel, quickly followed by failure.This indicates that the aggressive rate allowed the policy to reach a qualitatively better state than was possible with the slower rate.

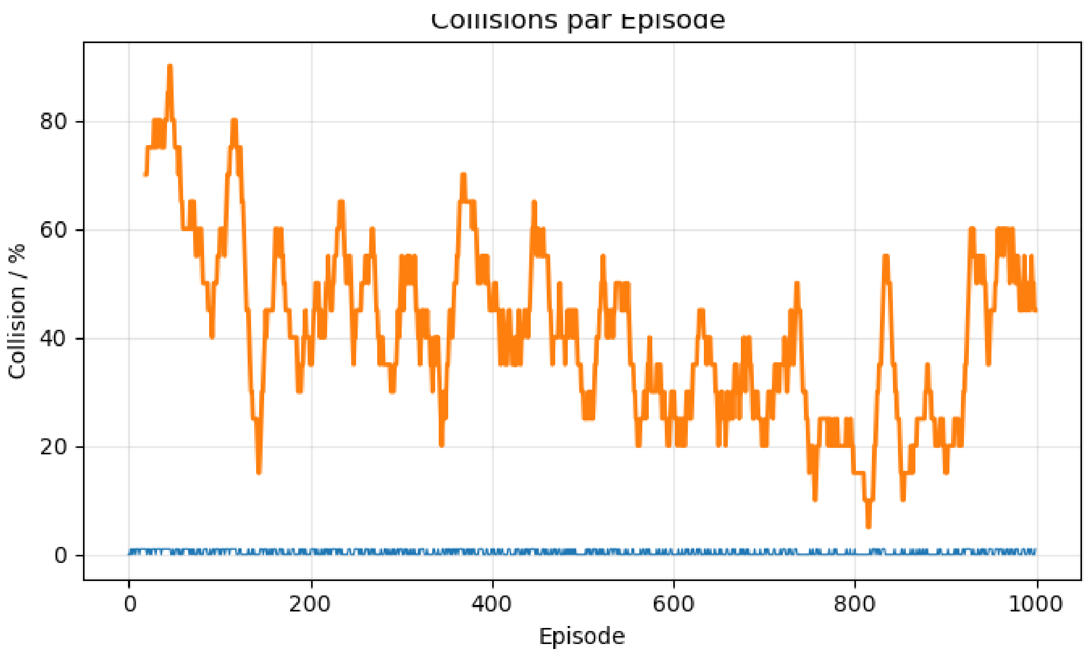


Figure 7: Collision curve

- **Collision Rate (Safety)**: Despite the high peak reward, the collision rate remained persistently high, frequently spiking to $\mathbf{40\%}-\mathbf{50\%}$ throughout late training. This persistent policy instability is confirmed by severe negative Eval scores (e.g., -32.5).

The experiment confirmed the core learning rate hypothesis:

- The aggressive rate (lr $= 10^{-4}$) achieved higher peak performance but suffered from **policy instability** and loss oscillation.
- The extremely low rate (lr $= 10^{-5}$) achieved **high stability** but led to **incomplete convergence** and a suboptimal peak performance $(\max 49.6)$.

Neither Learning Rate nor Learning Rate successfully trained the agent to consistently avoid catastrophic failure (collisions) or to reach the cumulative target reward of $\mathbf{500}$ in 1000 episodes. A Learning Rate between 10^{-5} and 10^{-4} would likely be necessary to find the optimal balance between speed and stability required for robust navigation.

4 Discussion

By implementing a Double DQN agent in a custom Pygame driving environment (compact 8-dimensional state and discrete action space $5 \times 3 = 15$) was instructive because it highlighted the gap between an RL training run that "works" and a policy that is genuinely robust from a safety standpoint. A key concern in autonomous driving discussed in the DRL literature (e.g., Ravi Kiran et al., 2020; Sallab et al., 2017).

4.1 Learnings (what we learned from the experience).

A key takeaway is the **sensitivity of deep RL training** to hyperparameters, especially the learning rate. In our experiments, lr $= 10^{-5}$ yielded smoother and more stable learning curves but slower progress, while lr $= 10^{-4}$ enabled higher peaks at the cost of stronger oscillations and occasional regressions. Monitoring loss, smoothed rewards, and collision rate also showed that a decreasing loss does not automatically imply a safe policy, since Q-values can remain fragile near critical states (short distances, red lights, obstacle proximity). Finally, reward design mattered strongly: keeping rewards **bounded and informative** (dense shaping plus terminal penalties) was necessary to avoid unstable training.

4.2 Limitations and challenges.

The main limitation is **incomplete safety generalization**: even after learning, collisions remain non-negligible and volatile, suggesting that the learned Q-function is not reliably optimal near dangerous states. A plausible explanation is the combination of (i) rare but highly informative terminal penalties, (ii) ϵ-greedy exploration that can generate unsafe actions, and (iii) a compact observation focusing on the nearest obstacle only, which limits anticipation.

In addition, training is **computationally slow** in practice because learning requires many interaction steps and repeated updates before the policy improves consistently. This difficulty is amplified by the fact that our environment is **dynamic and stochastic**: each episode can present different combinations of events (e.g., pedestrians may or may not appear, the agent may need to handle one vehicle or several vehicles, and obstacle placements vary). This episode-to-episode variability increases the **variance of returns** and makes credit assignment harder, since the same action can lead to different outcomes depending on the scenario. As a consequence, learning curves become noisier and the policy may overfit to frequently encountered situations while remaining fragile on rarer but safety-critical cases.

Additional constraints come from the **discrete control** (quantized steering/throttle may induce less smooth behavior) and the **partial state representation** (limited context prevents multi-object planning). Finally, the environment itself remains **simplified**: the road is straight, which reduces the diversity of driving situations and may limit how well the policy would transfer to more realistic conditions. A natural extension would be to design a **winding road** (curvature, turns) to introduce richer dynamics and force the agent to learn more realistic lateral control. Practically, implementing and debugging the environment, instrumenting metrics (including collisions), and stabilizing training also represented a significant development effort.

4.3 Potential improvements.

Several extensions could improve both performance and safety:

- **Imitation Learning / Behavior Cloning**: pretraining from simple "expert" trajectories (even rule-based demonstrations: brake at red lights, stay centered, avoid close obstacles) can reduce unsafe exploration and improve early safety (e.g., DAgger-style approaches by Ross et al., 2011).

- **Shared memory / experience pooling**: training multiple environment instances and feeding a shared replay buffer increases transition diversity and exposes the learner to rare critical events more often (distributed replay ideas such as Ape-X: Horgan et al., 2018). A simpler variant is to keep a small buffer dedicated to critical transitions or to use prioritized replay (Schaul et al., 2016).
- **Richer observations**: adding multiple obstacles at different ranges or relative-speed cues would support more reliable planning.
- **Scheduling and stability tuning**: learning-rate scheduling and systematic tuning of replay/batch/update settings could help overcome performance plateaus while reducing instability.

5 Conclusion & References

This project introduced a custom **Pygame** autonomous driving environment and trained a **Double DQN** policy from a compact 8-dimensional normalized state and a **discrete** 15-action control space. The results show that the agent learns basic behaviors (forward progress, reasonable speed regulation, initial obstacle-handling strategies), but achieving **safety-reliable** driving remains difficult: catastrophic failures (collisions and terminal events) stay too frequent, preventing the agent from reaching the internal cumulative reward goal of $\mathbf{500}$ within the considered training budget.

Among the two learning rates tested, **lr** $= 10^{-4}$ produced the **best-performing model** in terms of peak policy quality: it reached a higher maximum evaluation score (about 59 versus 49.6 for **lr** $= 10^{-5}$), suggesting faster and more effective optimization of the Q-function and better short-term driving capability. In other words, the larger step size helped the agent escape the slower training plateau observed with **lr** $= 10^{-5}$ and discover policies that can occasionally travel farther and handle obstacles more successfully. However, this improved peak performance came at the cost of **reduced stability**, with stronger loss oscillations and persistent late-training collision spikes, indicating that safety generalization remains incomplete.

Beyond raw performance, the main contribution is methodological: the experiments highlight the importance of (i) stabilization mechanisms in deep Q-learning (target network, replay buffer, Double DQN target computation, gradient clipping), (ii) a bounded and informative reward function, and (iii) careful instrumentation via dedicated metrics (smoothed rewards, loss, and collision rate). For improved robustness and reduced exploration cost, promising extensions include **imitation learning** as a safety-aware warm start, and **shared experience** (or prioritized replay) to emphasize rare but critical transitions.

In terms of control, we deliberately relied on a **discrete** action set (15 actions) because it provides a stable and tractable interface between the learning algorithm and the vehicle dynamics. This choice reduced the difficulty of simultaneously learning fine-grained steering and speed regulation, which would have required a more complex continuous-control setup. As future work, improvements can be pursued *within the same discrete framework* (e.g., better state information, imitation learning warm-start, and replay strategies focusing on rare critical events) before considering more realistic continuous-control methods.

References

- Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2nd ed.). MIT Press.
- van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double Q-learning. *AAAI*.
- Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2016). Prioritized experience replay. *ICLR*.
- Ravi Kiran, B., Sobh, I., Talpaert, V., Mannion, P., Al Sallab, A. A., Yogamani, S., & Pérez, P. (2020). Deep reinforcement learning for autonomous driving: A survey.
- Sallab, A. E., Abdou, M., Perot, E., & Yogamani, S. (2017). Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19)