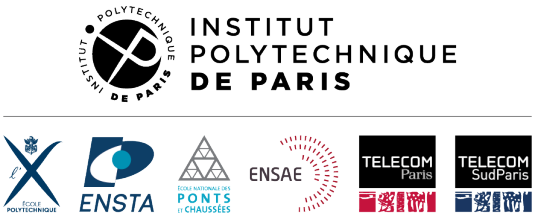
Reinforcement Learning Project
Paper : "*Rainbow: Combining Improvements in Deep Reinforcement Learning*"

Amine Razig
ENSAE - Polytechnique
amine.razig@polytechnique.edu

Abdelilah Younsi
École Polytechnique
abdelilah.younsi@polytechnique.edu



Project summary

This project is based on the foundational academic article *"Rainbow: Combining Improvements in Deep Reinforcement Learning"*Matteo Hessel [2018]. The objective is to explore and analyze the various enhancements proposed for the DQN algorithm, evaluating their theoretical and practical impacts.

The project is structured around three core elements. First, we summarize the key contributions of Matteo Hessel [2018], detailing the integration of distinct improvements in deep reinforcement learning. We provide theoretical insights into the mathematical foundations of these techniques and their interactions.

Next, we conduct a detailed experimental study by reproducing the main results from the paper. we analyze the effectiveness of the Rainbow agent compared to its individual components. This step involves implementing the algorithm and validating its performance across different games.

Finally, we extend the work by proposing our own experiments to evaluate the approaches.

All our code, theoretical explanations, and experimental results are available in this Github repository[1], ensuring the reproducibility of our work.

1 Introduction

The objective of this report is to explore various concepts of reinforcement learning from a mathematical perspective, with somes expériments to conclude. We precise that our explanations are

[1] **GitHub Repository link** : $https://github.com/arazig/Deep-Reinforcement-Learning$

intended to facilitate the understanding of the paper *Rainbow: Combining Improvements in Deep Reinforcement Learning*, certains éléments resterons a préciser.

Below is an overview of the key topics covered in this report:

1. Fundamental Concepts: Agents, Environments, Rewards, and Markov Decision Processes (MDPs)
2. Q-value Iteration and Learning
3. Transition from Tabular Q-Learning to Deep Networks
4. The Concept of Experience Replay
5. Target Networks and the Limitations of Deep Q-Networks (DQN)
6. The Six Core Components of the Rainbow Algorithm
7. The Rainbow Approach: Combining Multiple Improvements
8. Experimental Evaluation and Performance Analysis
9. Proof and Insights on Double Q-Learning Theoreme

2 Basic Concepts of Agents, Environments, Rewards, and MDPs

To introduce this report, we first explain the reinforcement Learning framework for learning through interaction with an environment. The core components of RL include the agent, the environment, states, actions, and rewards. These components are formalized using the concept of a Markov Decision Process (MDP), which provides a mathematical foundation for sequential decision-making.

2.1 Agent and Environment

In RL, an agent is an entity that interacts with an environment by taking actions based on the current state. The environment responds to these actions by transitioning to a new state and providing a reward to the agent. This interaction occurs over discrete time steps $t = 0, 1, 2, \ldots$. At each time step t, the agent observes the current state $S_t \in \mathcal{S}$, where $\mathcal{S}$ is the set of all possible states. Based on this observation, the agent selects an action $A_t \in \mathcal{A}(S_t)$, where $\mathcal{A}(S_t)$ is the set of available actions in state S_t. The environment then transitions to a new state S_{t+1} and provides a reward $R_{t+1} \in \mathcal{R}$, where $\mathcal{R}$ is the set of possible rewards.

2.2 Markov Decision Process

The interaction between the agent and the environment is typically modeled as a Markov Decision Process. An MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where:

- $\mathcal{S}$ is the state space,
- $\mathcal{A}$ is the action space,
- $\mathcal{P}$ is the state transition probability function,
- $\mathcal{R}$ is the reward function,
- $\gamma \in [0, 1]$ is the discount factor.

The state transition probability function $\mathcal{P}$ defines the probability of transitioning to state s' and receiving reward r given the current state s and action a:

$$\mathcal{P}(s', r|s, a) = \mathbb{P}(S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a).$$

An important point is that the Markov property assumes that the future state and reward depend only on the current state and action, and not on the history of past states and actions:

$$\mathbb{P}(S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a, H_t) = \mathbb{P}(S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a),$$

where $H_t = (R_t, S_{t-1}, A_{t-1}, \ldots)$ represents the history up to time t.

2.3 Rewards and Returns

The agent's goal is to maximize the cumulative reward, or return, over time. The return G_t at time t is defined as the sum of discounted future rewards:

$$G_t = \sum_{t'=t+1}^{\infty} \gamma^{t'-t-1} R_{t'},$$

where γ is the discount factor that determines the importance of future rewards. A discount factor γ close to 1 encourages the agent to prioritize long-term rewards, while a smaller γ focuses on short-term rewards.

In some settings, the return may be defined over a finite horizon T:

$$G_t^T = \sum_{t'=t+1}^{T} R_{t'}.$$

Alternatively, in episodic settings, the return is well-defined if the episode terminates after a finite number of steps.

2.4 Value Functions

To evaluate the performance of a policy π, which is a mapping from states to actions, we define the state-value function $v_\pi(s)$ and the action-value function $q_\pi(s, a)$. The state-value function $v_\pi(s)$ represents the expected return when starting from state s and following policy π:

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s].$$

Similarly, the action-value function $q_\pi(s, a)$ represents the expected return when starting from state s, taking action a, and then following policy π:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a].$$

These value functions satisfy the Bellman equation, which expresses the relationship between the value of a state (or state-action pair) and the values of its successor states:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} \mathcal{P}(s', r|s, a) \left[r + \gamma v_\pi(s') \right],$$

$$q_\pi(s, a) = \sum_{s',r} \mathcal{P}(s', r|s, a) \left[r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s', a') \right].$$

2.5 Optimal Policy and Value Functions

The goal of RL is to find an optimal policy π^* that maximizes the expected return for all states. The optimal state-value function $v^*(s)$ and the optimal action-value function $q^*(s, a)$ are defined as:

$$v^*(s) = \max_\pi v_\pi(s),$$

$$q^*(s, a) = \max_\pi q_\pi(s, a).$$

These optimal value functions satisfy the Bellman optimality equation:

$$v^*(s) = \max_a \sum_{s',r} \mathcal{P}(s', r|s, a) \left[r + \gamma v^*(s') \right],$$

$$q^*(s, a) = \sum_{s',r} \mathcal{P}(s', r|s, a) \left[r + \gamma \max_{a'} q^*(s', a') \right].$$

The optimal policy π^* can be derived from the optimal action-value function q^* by selecting the action that maximizes $q^*(s, a)$ for each state s:

$$\pi^*(s) =_a q^*(s, a).$$

To recap what we said in this recall of the basics of RL, the MDP framework provides a rigorous mathematical foundation for RL, where the agent interacts with the environment to maximize cumulative rewards. The concepts of value functions and the Bellman equations are central to understanding how RL algorithms, such as Q-learning and Deep Q-Networks (DQN), operate.

These concepts will be essential for understanding the Rainbow algorithm, which combines several improvements to DQN to achieve state-of-the-art performance in RL tasks. Indeed, to understand well the purpose of the method proposed in the paper of interest Matteo Hessel [2018] we will, in the next parts of the report, continue to explain fondamentals reinforcement learning concepts.

3 Learning by Q-value Iteration

3.1 Q-Value Iteration Process

Q-value iteration is an iterative algorithm for computing the optimal action-value function $Q^*(s, a)$, which represents the maximum expected cumulative reward achievable by taking action a in state s and subsequently following the optimal policy. The process refines Q-values using the Bellman optimality equation:

$$Q_{k+1}(s, a) = \mathbb{E}\left[r + \gamma \max_{a'} Q_k(s', a')\right]$$

3.2 Exploration vs. Exploitation

Balancing exploration and exploitation is critical in reinforcement learning to avoid suboptimal policies. This dilemma arises because an RL agent must decide between trying new actions to discover their effects (exploration) and leveraging known information to maximize rewards (exploitation). Intuitively speaking, if a person visits a new city and wants to find the best restaurant, he might try different restaurants (exploration) to discover which ones offer the best food and experience. This helps him gather information about various options. Once he's found a restaurant he likes, he keeps going back to it (exploitation) because he knows it's good. This maximizes his immediate satisfaction based on past experiences. However, too much exploitation can cause him to miss out on even better options that he hasn't discovered yet. Therefore, balancing these two strategies is essential for optimizing outcomes in both reinforcement learning and real-life decision-making.

3.2.1 Strategies for Balancing Exploration and Exploitation

In the RL modelisations, balancing exploration and exploitation is crucial for discovering optimal policies, and two widely-used strategies for achieving this balance are the ϵ-Greedy Strategy and Boltzman Exploration, each offering distinct mechanisms to navigate the trade-off between exploring new actions and exploiting known rewards.

- **ϵ-Greedy Strategy**:
  - With probability ϵ, select a random action (exploration).
  - With probability $1 - \epsilon$, select $a^* = \arg\max_a Q(s, a)$ (exploitation).
- **Boltzmann (Softmax) Exploration**:
  - Uses a softmax distribution to select actions based on their Q-values:

$$P(a|s) = \frac{e^{Q(s,a)/\tau}}{\sum_{a'} e^{Q(s,a')/\tau}}$$

  - The temperature parameter τ controls the randomness:
    - High τ: Actions are chosen more uniformly, encouraging exploration.
    - Low τ: Actions with higher Q-values are favored, encouraging exploitation.

3.3 Temporal Difference (TD) Learning and Q-Learning

Temporal Difference learning is a fundamental concept in reinforcement learning that combines ideas from Monte Carlo sampling and dynamic programming. It updates Q-values incrementally

using TD errors, which measure the difference between the current Q-value estimate and the TD target. This approach allows for more efficient learning compared to methods that wait for the final outcome to update values.

- **The TD error** is given by the difference between the current Q-value estimate and the TD target term, the formula is given by :

$$\delta = \underbrace{r + \gamma \max_{a'} Q(s', a')}_{TD\ target} - \underbrace{Q(s, a)}_{Q-values\ estimate}$$

- **Q-Learning Update Rule**: The Q-learning update rule adjusts Q-values toward the TD target using a learning rate α. The update rule is:

$$Q(s, a) \leftarrow Q(s, a) + \alpha\delta$$

Here, α determines how much the new information (TD error) influences the Q-value update. A higher α means the agent learns more quickly from new experiences but can also lead to instability if α is too high. This incremental update helps the agent gradually improve its policy over time.

Algorithm 1 Temporal Difference Error and Q-Learning Update Rule

Input: State s, Action a, Reward r, Next state s', Learning rate α, Discount factor γ
Output: Updated Q-value $Q(s, a)$
Compute TD Error:
$\delta \leftarrow r + \gamma \max_{a'} Q(s', a') - Q(s, a)$ {δ is the TD error}
where:
  $r + \gamma \max_{a'} Q(s', a')$ {TD target}
  $Q(s, a)$ {Current Q-value estimate}
Update Q-value:
$Q(s, a) \leftarrow Q(s, a) + \alpha\delta$ {Q-Learning update rule}
where:
  α {Learning rate controlling update magnitude}
  δ {TD error}
Return: Updated $Q(s, a)$ =0

- **Off-Policy Learning**: Q-learning is an off-policy method, meaning it learns the optimal policy while following a behavior policy, such as an ϵ-greedy policy. This allows for flexible exploration.

4 Replacing Tabular Q-Learning with Deep Networks

4.1 Limitations of Tabular Q-Learning

Tabular Q-learning stores Q-values in a lookup table, where each entry $Q(s, a)$ corresponds to a unique state-action pair. While effective for small, discrete environments (e.g., grid worlds with 10 states), it fails catastrophically in large or continuous spaces:

- **Scalability Issues**: The table size grows exponentially with the number of states and actions. For example:
  - Atari games (e.g., *Pong*) have $\sim 10^{1000}$ possible screen states. Storing Q-values for all combinations is impossible.
  - Real-world robotics tasks involve continuous sensor data (e.g., lidar, camera feeds), making tabular representation infeasible.
- **No Generalization**: Tabular methods cannot generalize across similar states. For instance, if an agent learns to avoid walls in one maze layout, it cannot transfer that knowledge to a slightly different maze.

4.2 Deep Q-Networks : Neural Networks as Function Approximators

In 2015, a new approach was presented by Volodymyr Mnih [2015]. DQN replaces the Q-table with a neural network $Q(s, a; \theta)$, where θ represents network weights. The network takes a state s (e.g., raw pixels) as input and outputs Q-values for all actions.

Generalization: The network learns patterns (e.g., edges, colors, object positions) from pixels and applies them to unseen states. For example, if a DQN learns that "walls are bad" in one maze, it can avoid walls in any maze with similar visual features.

Scalability: The network's fixed number of parameters (e.g., 1 million weights) can represent Q-values for exponentially many states.

As an example for this in *Breakout*, the DQN learns to associate the ball's trajectory with high Q-values for moving the paddle toward it, even if the exact pixel configuration is new.

4.3 How DQN Solves Scalability

A major result of the paper Volodymyr Mnih [2015] is the highlighting that DQNs allow us to leverage neural networks' ability to compress state information. Indeed, the achitecture of the networks provide two main effects :

The first one is the **parameter sharing**: Similar states (e.g., two frames of *Pong* where the ball is slightly shifted) activate overlapping network weights, allowing the agent to share knowledge across states.

The second is the **feature extraction**: Convolutional layers automatically detect hierarchical patterns (e.g., edges $\rightarrow$ shapes $\rightarrow$ game objects), reducing reliance on handcrafted state representations.

5 Concept of Experience Replay

5.1 Problem of correlated Data and Non-I.I.D. Learning

In online Q-learning, transitions are generated sequentially (e.g., consecutive frames in a game). This leads to two critical issues:

- Consecutive transitions (s_t, a_t, r_t, s_{t+1}) and $(s_{t+1}, a_{t+1}, r_{t+1}, s_{t+2})$ are highly correlated. For example, in Pac-Man, adjacent frames differ only slightly as the agent moves. As a consequence, Gradient updates become biased toward recent experiences, destabilizing training. The network "forgets" older patterns and overfits to recent ones.
- Stochastic gradient descent assumes training data is independent and identically distributed (i.i.d.). Sequential transitions violate this assumption, leading to inefficient and oscillatory updates.

5.2 Solution 1: Experience Replay

Experience replay addresses these issues by storing transitions (s_t, a_t, r_t, s_{t+1}) in a replay buffer D of fixed capacity N. During training, mini-batches B are sampled uniformly from D, breaking temporal correlations.

5.3 Solution 2: Prioritized Experience Replay

Standard experience replay samples transitions uniformly from the replay buffer $\mathcal{D}$. However, not all transitions are equally valuable: some experiences are more "surprising" or informative than others. **Prioritized Experience Replay** addresses this by prioritizing transitions based on their temporal difference error δ, which measures how much the agent's current Q-value estimate deviates from the target, as we explained before.

5.3.1 Motivation for Prioritization

- **Learning Efficiency**: Transitions with high $|\delta|$ (e.g., rare successes or catastrophic failures) provide more learning signal. Prioritizing these accelerates convergence.
- **Adaptive Focus**: The agent spends more computational resources on experiences it currently misunderstands.

5.3.2 Mathematical Formulation

Let the priority p_i of transition i be proportional to its TD error:

$$p_i = |\delta_i| + \epsilon$$

where ϵ is a small constant to ensure non-zero probabilities.
Transitions are sampled with probability:

$$P(i) = \frac{p_i^\alpha}{\sum_j p_j^\alpha}$$

where $\alpha \in [0, 1]$ controls prioritization strength ($\alpha = 0$ recovers uniform sampling).

6 Concept of Target Networks and Limitations of DQN

6.1 Target Networks in Deep Q-Learning

In Deep Q-Networks, the agent uses a neural network $Q(s, a; \theta)$ with parameters θ to approximate Q-values. However, directly updating $Q(s, a; \theta)$ using its own predictions as targets introduces instability. This is because the **TD target** $r + \gamma \max_{a'} Q(s', a'; \theta)$ depends on the same parameters θ being optimized, creating a feedback loop akin to "chasing one's own tail."

6.1.1 Solution: Target Networks

To stabilize training, DQN employs a **target network** $Q(s, a; \theta^-)$, which is a delayed copy of the online network $Q(s, a; \theta)$. The target network's parameters θ^- are updated periodically (e.g., every C steps) by copying θ, while the online network θ is updated continuously.

The TD target is now computed using the target network:

$$y = r + \gamma \max_{a'} Q(s', a'; \theta^-)$$

The loss function becomes:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}} \left[(y - Q(s, a; \theta))^2 \right]$$

7 The Six Components of Rainbow

So far, we've introduced the mathematical intuition behind the main concepts widely used in RL. We can now focus on the 6 algorithms that make up the Rainbow model. The main idea is to explain how the strengths of each method are combined by integrating the different components into a single model.

First, we'll concentrate on the theoretical explanation of the model. Then, in a second step, we'll describe our experiments and comment on our results.

Let's start with an overview of the six key components of Rainbow.

Double Q-learning

This notion of Double Q-leanring was first introduced by Hado van Hasselt [2016]. The idea is the following. Their paper addresses the overestimation bias in Q-learning, particularly when combined with deep neural networks. So, the authors propose Double DQN, a modification that reduces this bias and improves performance.

The principal motivation behind the approach of DQN is a phenomen of overestimation in Q-learning.

Indeed, since the Q-learning's target is given by:

$$Y_t^Q = R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \theta_t),$$

which uses the same values for both action selection and evaluation, leading to overestimation. This overestimation is represented by the theorem 1. Based on the paperHado van Hasselt [2016] we also reformulate the proof of this theorem :

Theorem 1: Even with unbiased estimates, the maximum Q-value is biased upward:

$$\max_a Q_t(s, a) \geq V_*(s) + \sqrt{\frac{C}{m-1}},$$

where C is the variance of the Q-value estimates and m is the number of actions.

This overestimation bias, formalized in Theorem 1, highlights the need for methods like Double Q-learning to decouple selection and evaluation.

So, as we said, the idea behind DQN is to decouple action selection and evaluation using two sets of weights θ and θ', it leads us to the following formula:

$$Y_t^{\text{DoubleQ}} = R_{t+1} + \gamma Q(S_{t+1}, _a Q(S_{t+1}, a; \theta_t); \theta'_t).$$

This reduces overestimation by using θ'_t to evaluate the action selected by θ_t. Then, the authors adapt Double Q-learning to DQN by using the target network θ^- for evaluation. This modification retains the DQN architecture while reducing overestimation:

$$Y_t^{\text{DoubleDQN}} = R_{t+1} + \gamma Q(S_{t+1}, _a Q(S_{t+1}, a; \theta_t); \theta_t^-).$$

Double DQN successfully mitigates the overestimation bias in Q-learning, leading to more accurate value estimates and improved performance in complex environments.

Prioritized Replay in Rainbow

While Section 5.3 introduced prioritized experience replay (PER) using TD errors, Rainbow adapts this mechanism to align with its distributional learning framework. Instead of prioritizing transitions based on 1-step TD errors δ, Rainbow uses KL divergence between the predicted and target return distributions as the priority signal. This modification better reflects the agent's uncertainty in distributional value estimates.

For a transition with n-step return $d_t^{(n)}$, the priority is computed as:

$$p_t \propto \left(D_{\text{KL}} \left(\Phi d_t^{(n)} \| d_t \right) \right)^{\omega},$$

where ω controls the prioritization strength ($\omega = 1$ recovers proportional prioritization). This KL-based priority directly measures how "surprising" the transition is under the current distributional model, enabling more robust credit assignment in stochastic environments.

Rainbow retains two key PER components:

Stochastic Prioritization: Transitions are sampled with probability $P(i) \propto p_i^\alpha$, where α balances uniform vs. prioritized sampling.

Importance Sampling (IS): Updates are weighted by $w_i = \left(\frac{1}{N \cdot P(i)} \right)^\beta$ to correct sampling bias, with β annealed from β_{initial} to 1 during training.

This integration ensures that transitions contributing most to distributional errors (e.g., partial observability or rare events) are replayed more frequently, while IS weights stabilize convergence. Combined with multi-step returns, KL prioritization allows Rainbow to focus on transitions where both immediate rewards and long-term value distributions are poorly modeled.

Dueling Network Architecture

The dueling network architecture introduces an additional structural modification to traditional Q-networks by decoupling value estimation from advantage learning. The advantage learning help choose an action that results in a better state-value function compared to the mean contribution which is the value function itself. This architectural innovation, first proposed by Ziyu Wang [2016], addresses fundamental challenges in credit assignment and learning efficiency through explicit separation of state valuation and action-dependent advantages.

The core insight comes from the observation that many states require precise estimation of state value without needing detailed action discrimination. The network achieves this through parallel streams (networks) estimating the state value function $V(s)$ and advantage function $A(s, a)$, which combine to produce Q-values through a specialized aggregation layer. The advantage function plays a crucial role in measuring the relative importance of each action compared to the state's average value:

$$A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s)$$

Through Value and advantage saliency maps, this decomposition allows the network to learn which states are intrinsically valuable (in the long run because of the visual patterns the value stream learns) while efficiently identifying actions that outperform the average behavior in those states (in the short run because of the visual patterns the advantage stream learns).

The network implements this through twin streams emerging from shared convolutional features:

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left(A(s, a; \theta, \alpha) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a'; \theta, \alpha) \right)$$

This aggregation mechanism resolves a critical identifiability challenge inherent in the additive decomposition $Q = V + A$. The naive formulation suffers from unobservability - any constant shift in advantage values could be offset by opposing changes in state value estimates, leaving Q-values unchanged. The mean subtraction operation enforces a natural constraint on the advantage function:

> **Identifiability Lemma**: For any Q-function decomposition $Q(s, a) = V(s) + A(s, a)$, the solution can be unique when $\mathbb{E}_{a \sim \pi}[A(s, a)] = 0$. The mean subtraction operator in the equation above satisfies this condition by construction.

This constraint stabilizes learning by anchoring the value stream to estimate true state values while allowing the advantage stream to capture relative action preferences. The architecture retains the original Q-learning interface, enabling seamless integration with existing reinforcement learning algorithms while providing an efficient State Valuation, a robust action selection and lead to better generalization. In fact :

- The shared value estimates accelerate learning in states where action choices have minimal impact
- The advantage stream's relative measurements reduce sensitivity to small Q-value fluctuations
- And separate value estimation prevents overfitting to transient action-specific variations

Distributional Reinforcement Learning

Distributional Reinforcement Learning (Distributional RL) redefines the value-based learning paradigm by modeling the full probability distribution of returns $Z(s, a)$ rather than merely estimating its expectation $Q(s, a) = \mathbb{E}[Z(s, a)]$. Introduced by Marc G. Bellemare [2017], this approach captures the intrinsic uncertainty of future rewards, enabling more robust policy evaluation and mitigation of overestimation bias.

The return distribution is approximated using a fixed discrete support with $N_{\text{atoms}} \in \mathbb{N}^+$ atoms spanning a predefined range $[v_{\min}, v_{\max}]$. The support is a vector of equally spaced values:

$$z^i = v_{\min} + (i-1)\frac{v_{\max} - v_{\min}}{N_{\text{atoms}} - 1}, \quad \text{for } i \in \{1, \ldots, N_{\text{atoms}}\}.$$

For each state-action pair (S_t, A_t), the distribution d_t is parameterized by a probability mass vector $p_\theta(S_t, A_t) = [p_\theta^1, \ldots, p_\theta^{N_{\text{atoms}}}]$, where p_θ^i represents the likelihood of the return being z^i. These probabilities are predicted by a neural network, with softmax normalization applied independently across atoms for each action to ensure $\sum_i p_\theta^i(S_t, A_t) = 1$.

Distributional Bellman Update. The return distribution satisfies a distributional Bellman equation:

$$Z(S_t, A_t) \overset{D}{=} R_{t+1} + \gamma_{t+1} Z(S_{t+1}, A_{t+1}^*),$$

where $A_{t+1}^* =_a \mathbb{E}[Z(S_{t+1}, a)]$ is the greedy action. To implement this recursively:

1. **Target Action Selection**: Compute A_{t+1}^* using a target network $\overline{\theta}$:

$$A_{t+1}^* =_a \sum_{i=1}^{N_{\text{atoms}}} z^i \cdot p_{\overline{\theta}}^i(S_{t+1}, a).$$

2. **Shift and Scale**: Apply the Bellman update to the target distribution:

$$\hat{Z}^i = R_{t+1} + \gamma_{t+1} z^i \quad \text{for each atom } z^i.$$

3. **Projection**: Map the transformed atoms $\{\hat{Z}^i\}$ onto the fixed support using an L_2-projection operator Φ. This is because the applied equivalent Bellman operator makes the support of the new value distributions and the old one disjoint.

Learning Objective. The network parameters θ are optimized by minimizing the Kullback-Leibler (KL) divergence between the projected target distribution $\Phi d_t'$ and the predicted distribution d_t:

$$\mathcal{L}(\theta) = \mathbb{E}\left[D_{\text{KL}}\left(\Phi d_t' \,\|\, d_t\right)\right],$$

where $d_t' \equiv (R_{t+1} + \gamma_{t+1} z, \;\; p_{\overline{\theta}}(S_{t+1}, \overline{a}_{t+1}^*))$ and $d_t = (z, p_\theta(S_t, A_t))$. This reduces to cross-entropy loss over the discrete support, enabling efficient optimization.

Below is the algorithm in the originale paper Marc G. Bellemare [2017]:

Algorithm 2 Categorical Algorithm

Require: A transition $s_t, a_t, r_t, s_{t+1}, \gamma_t \in [0, 1]$
$Q(s_{t+1}, a) := \sum_i z_i p_i(s_{t+1}, a)$
$a^* \leftarrow \arg\max_a Q(s_{t+1}, a)$
$m_i = 0, \quad i \in \{0, \ldots, N-1\}$
for $j \in \{0, \ldots, N-1\}$ **do**
  # Compute the projection of $\hat{T}z_j$ onto the support $\{z_i\}$
  $\hat{T}z_j \leftarrow [r_t + \gamma_t z_j]_{V_{\min}}^{V_{\max}}$
  $b_j \leftarrow (\hat{T}z_j - V_{\min})/\Delta z$ # $b_j \in [0, N-1]$
  $l \leftarrow \lfloor b_j \rfloor, \quad u \leftarrow \lceil b_j \rceil$
  # Distribute probability of $\hat{T}z_j$
  $m_l \leftarrow m_l + p_j(s_{t+1}, a^*)(u - b_j)$
  $m_u \leftarrow m_u + p_j(s_{t+1}, a^*)(b_j - l)$
end for
return $-\sum_i m_i \log p_i(s_t, a_t)$ #Cross-entropy loss
=0

Multi-step Learning

Multi-step learning enhances temporal credit assignment by considering sequences of future rewards when updating Q-value estimates. While traditional Q-learning uses a 1-step temporal difference (TD) target, Rainbow employs n-step returns to strike a balance between the high bias of 1-step methods and the high variance of Monte Carlo rollouts. This approach accelerates reward propagation while maintaining stable learning.

n-Step Return Target

For a transition at time t, the n-step return $G_t^{(n)}$ aggregates rewards over n steps and bootstraps from the value of the state n steps later:

$$G_t^{(n)} = \sum_{k=0}^{n-1} \gamma^k R_{t+k+1} + \gamma^n \max_{a'} Q(S_{t+n}, a'; \theta^-),$$

where θ^- denotes the target network parameters. This provides a richer learning signal by directly incorporating observed rewards while retaining the long-term value estimate.

Integration with Distributional RL

In Rainbow's distributional formulation, the n-step target becomes a distribution over returns. Let $Z(S_{t+n}, a^*)$ denote the return distribution for the optimal action a^* in state S_{t+n}, selected using the online network θ. The target distribution is constructed by:

1. Accumulating the discounted n-step rewards:

$$R_t^{(n)} = \sum_{k=0}^{n-1} \gamma^k R_{t+k+1}.$$

2. Shifting and scaling the bootstrap distribution $Z(S_{t+n}, a^*; \theta^-)$ by γ^n.
3. Projecting the resulting distribution onto the predefined support .

The projected target distribution $\Phi d_t^{(n)}$ is then used to compute the KL divergence loss.

Noisy Nets

Finally, the idea behind the last extension of DQN is to mprove exploration by adding parametric noise to the weights of a neural network. Unlike traditional exploration methods like ϵ-greedy or entropy regularization, which rely on random perturbations of the agent's actions, NoisyNet introduces structured noise directly into the network's parameters. By doing this, we allows the agent to explore more efficiently by inducing state-dependent stochasticity in the policy.

Here is a brief explanation of working of this method. First, in a standard neural network, a linear layer is defined as:

$$y = wx + b,$$

where x is the input, w is the weight matrix, b is the bias, and y is the output.

In NoisyNet, the weights and biases are replaced with noisy versions:

$$y = (\mu^w + \sigma^w \odot \varepsilon^w)x + (\mu^b + \sigma^b \odot \varepsilon^b),$$

where:

- μ^w and μ^b are the learnable mean parameters for weights and biases,
- σ^w and σ^b are the learnable standard deviation parameters,
- ε^w and ε^b are noise variables sampled from a fixed distribution (e.g., Gaussian),
- $\odot$ represents element-wise multiplication.

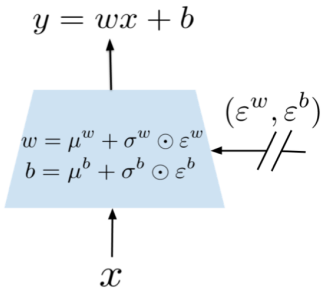


Figure 1: **Graphical representation of a noisy linear layer**(source : Meire Fortunato [2018]). *The input x corresponds to the state s in the rest of the report. The parameters μ^w, μ^b, σ^w, and σ^b are learnable, while ε^w and ε^b are noise variables. The noisy weights and biases are computed as $w = \mu^w + \sigma^w \odot \varepsilon^w$ and $b = \mu^b + \sigma^b \odot \varepsilon^b$, respectively. The output y is computed as $y = wx + b$.*

In Deep Q-Networks, NoisyNet replaces the ϵ-greedy exploration strategy. Instead of randomly selecting actions with probability ϵ, the agent greedily selects actions based on the noisy Q-values generated by the network.

And so, the loss function for NoisyNet-DQN is:

$$\bar{L}(\zeta) = \mathbb{E}_{\varepsilon,\varepsilon'}\left[\mathbb{E}_{(s,a,r,y)\sim D}\left[\left(r + \gamma \max_{b\in\mathcal{A}} Q(y, b, \varepsilon'; \zeta^-) - Q(s, a, \varepsilon; \zeta)\right)^2\right]\right],$$

where:

- ζ represents the parameters of the noisy network,
- ε and ε' are noise samples for the online and target networks,
- D is the replay buffer,
- $Q(s, a, \varepsilon; \zeta)$ is the noisy Q-value function.

8 The Rainbow approach

Matteo Hessel [2018] integrate all the aforementioned components into a single agent, which we call Rainbow. Below, we explain how each component is integrated and how they work together.

First, we replace the 1-step distributional loss with a **multi-step variant**. The target distribution is constructed by contracting the value distribution in S_{t+n} according to the cumulative discount and shifting it by the truncated n-step discounted return. The target distribution is defined as:

$$d_t^{(n)} = (R_t^{(n)} + \gamma^{(n)} z, p_{\theta_{t+n}}(S_{t+n}, a^*_{t+n})),$$

where $R_t^{(n)}$ is the truncated n-step discounted return, $\gamma^{(n)}$ is the cumulative discount, and a^*_{t+n} is the greedy action selected by the online network.

The resulting loss is the Kullback-Leibler (KL) divergence between the projected target distribution $\Phi_z d_t^{(n)}$ and the predicted distribution d_t:

$$L = D_{\text{KL}}(\Phi_z d_t^{(n)} \| d_t),$$

where Φ_z is the projection onto the support z.

We combine the multi-step distributional loss with **double Q-learning** by using the greedy action in S_{t+n} selected according to the online network as the bootstrap action a^*_{t+n}. This action is then evaluated using the target network, which helps reduce overestimation bias.

In standard proportional prioritized replay, transitions are prioritized based on the absolute TD error. However, in Rainbow, we prioritize transitions using the **KL loss**, which is the loss being minimized by the algorithm:

$$p_t \propto \left(D_{\text{KL}}(\Phi_z d_t^{(n)} \| d_t)\right)^{\omega},$$

where ω is a hyperparameter that controls the degree of prioritization. Using the KL loss as a priority is more robust in noisy stochastic environments because the loss can continue to decrease even when the returns are not deterministic.

The network architecture is a **dueling network** adapted for use with return distributions. The network has a shared representation $f_\xi(s)$, which is fed into two streams:

- A **value stream** v_η with N_{atoms} outputs.
- An **advantage stream** a_ψ with $N_{\text{atoms}} \times N_{\text{actions}}$ outputs.

For each atom z_i, the value and advantage streams are aggregated as in dueling DQN, and then passed through a softmax layer to obtain the normalized parametric distributions used to estimate the return distributions:

$$p_\theta(s,a) = \frac{\exp(v_\eta^i(\phi) + a_\psi^i(\phi,a) - \bar{a}_\psi^i(s))}{\sum_j \exp(v_\eta^j(\phi) + a_\psi^j(\phi,a) - \bar{a}_\psi^j(s))},$$

where $\phi = f_\xi(s)$ and $\bar{a}_\psi^i(s) = \frac{1}{N_{\text{actions}}} \sum_{a'} a_\psi^i(\phi, a')$.

Finally, we replace all linear layers in the network with their **noisy equivalents**, as described in Equation (4). Within these noisy linear layers, we use factorised Gaussian noise to reduce the number of independent noise variables. This allows the agent to explore more efficiently by automatically tuning the noise parameters via gradient descent.

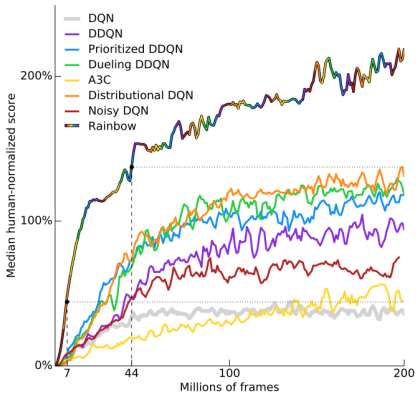
Summary

Rainbow integrates the following components:

- **Multi-step distributional loss** for more accurate value estimation.
- **Double Q-learning** to reduce overestimation bias.
- **Prioritized experience replay** using the KL loss for robust prioritization.
- **Dueling network architecture** adapted for return distributions.
- **Noisy networks** for efficient exploration.

9 Experiments

General performance of rainbow



According to the figure presented in the original paper, Rainbow significantly outperforms the other methods in terms of human-median normalized score, achieving higher performance across a wide range of environments. This demonstrates Rainbow's superior ability to leverage a combination of advancements in deep reinforcement learning for better learnng.

In the next section we present our personal experiments, we aim to explore Rainbow performance and try to understand the effect of each of its components.

Our ablation experiment

We conducted an ablation study on the Rainbow reinforcement learning model to analyze the impact of removing its key components on training performance over 20,000 frames in the CartPole environment. The results are summarized in the figure.

Our findings indicate that Categorical DQN is the most critical component, as its removal drastically reduces performance. n-step returns and the Dueling network also play significant roles, as their absence leads to noticeable drops in learning efficiency. In contrast, Double Q-learning, Noisy layers, and Prioritized Experience Replay contribute to improved stability and performance but are not strictly essential for achieving high scores in this task.

Through this study, we were able to see how different components of Rainbow contribute to training efficiency, especially in a simple environment like CartPole. Some elements, like Categorical DQN, turned out to be crucial, while others, such as Noisy Layers or Prioritized Experience Replay, mainly helped with stability but were not absolutely necessary.

That being said, CartPole is quite a basic game, and Rainbow's advantages become even more evident in more complex environments. To explore this further, we adapted a fully functional code on our GitHub page[2], making it compatible with the Gymnasium library. This ensures that all our experiments are fully reproducible and easy to run.

We also test Rainbow on more challenging games like LunarLander and Pac-Man, where the impact of each component becomes even clearer. These experiments helped us better understand which mechanisms are truly essential when dealing with more complex learning tasks.

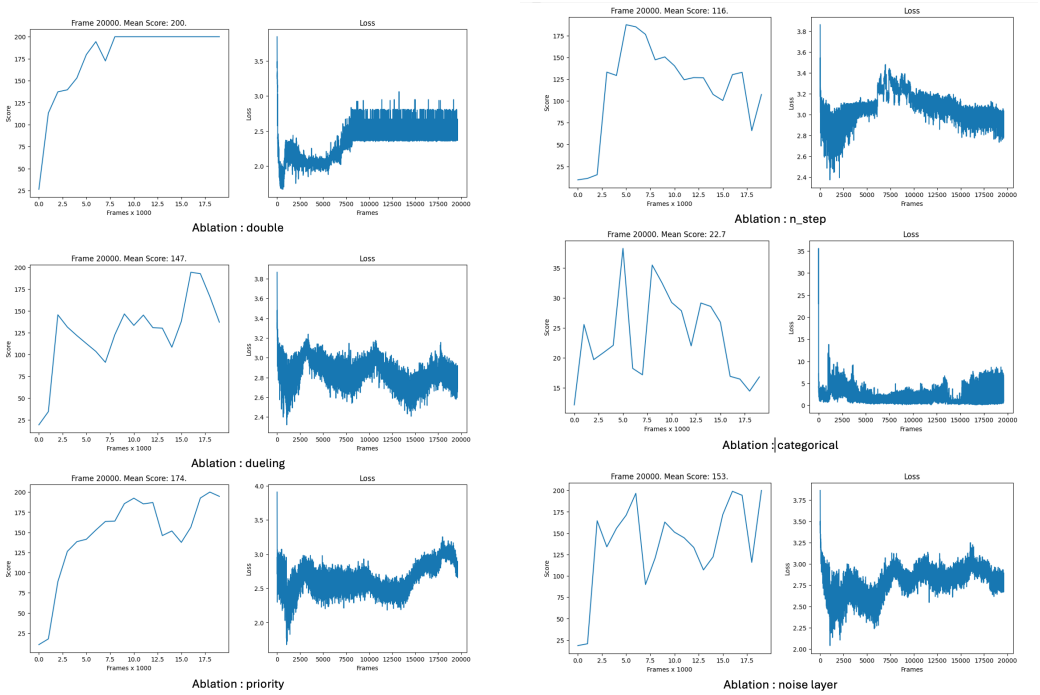


Figure 2: Ablation studies on the Rainbow model. These plots show the effect of removing certain components on training over 20,000 frames on CartPole.

Atari Pacman Experiment

To further evaluate Rainbow's performance in more complex environments, we tested the algorithm on the classic Atari game Pacman. We ran the Rainbow agent for 10 epochs, with each epoch consisting of 1000 steps. The results of this experiment can be visualized in the video folder available in our GitHub repository. The video format provides a more intuitive understanding of the agent's

[2] **GitHub Repository link** : $https://github.com/arazig/Deep-Reinforcement-Learning$

behavior and learning progress compared to static performance graphs, especially in a visually dynamic game like Pacman. Interested readers can increase the number of epochs and steps to observe the agent's continued improvement, eventually reaching a point where it successfully wins the game. This experiment demonstrates Rainbow's capability to learn effective policies in environments with complex visual inputs, sparse rewards, and challenging dynamics.

10 Conclusion

In this report, we investigated the Rainbow algorithm, which integrates six key improvements to DQN: Double Q-learning, Prioritized Experience Replay, Dueling Networks, Distributional RL, Multi-step Learning, and Noisy Networks. Our theoretical analysis detailed how these components address different limitations of traditional DQN while working synergistically.

Our experimental results confirmed that Rainbow outperforms both vanilla DQN and individual components across various environments. Our ablation studies identified Categorical DQN (Distributional RL) as the most critical component, followed by n-step returns and Dueling Networks. Experiments on CartPole, LunarLander, and Pac-Man demonstrated Rainbow's effectiveness in environments of increasing complexity.

These findings emphasize the value of combining multiple algorithmic improvements rather than focusing on isolated techniques. Rainbow's success illustrates how addressing various challenges in reinforcement learning—from exploration to overestimation to credit assignment—yields compounding benefits when integrated properly.

Future work could explore extending Rainbow to continuous action spaces or combining it with recent innovations such as transformer architectures or hierarchical approaches. Our implementation provides a foundation for further research and applications in reinforcement learning.

Appendix

Proof of Theorem 1

Proof. We aim to show that for a state s where all true optimal action values are equal, i.e., $Q_*(s,a) = V_*(s)$, and the Q-value estimates $Q_t(s,a)$ are unbiased but not all correct, the maximum Q-value is biased upward. Specifically:

$$\max_a Q_t(s,a) \geq V_*(s) + \sqrt{\frac{C}{m-1}},$$

where $C = \frac{1}{m}\sum_a (Q_t(s,a) - V_*(s))^2$ is the variance of the Q-value estimates, and m is the number of actions.

Defintion of the errors :

First, we define the errors. Let $\epsilon_a = Q_t(s,a) - V_*(s)$ be the estimation error for each action a. By assumption, the errors are unbiased:

$$\sum_a \epsilon_a = 0,$$

and the variance of the errors is:

$$\frac{1}{m}\sum_a \epsilon_a^2 = C.$$

The idea of the proof is to reason by the absurd. So we suppose, for contradiction, that the maximum error is less than $\sqrt{\frac{C}{m-1}}$, i.e.,

$$\max_a \epsilon_a < \sqrt{\frac{C}{m-1}}.$$

Let $\{\epsilon_i^+\}$ be the set of positive errors (size n) and $\{\epsilon_j^-\}$ be the set of strictly negative errors (size $m-n$), such that $\{\epsilon_a\} = \{\epsilon_i^+\} \cup \{\epsilon_j^-\}$.

Then we analyze the cases : If $n = m$, all errors are non-negative. However, since $\sum_a \epsilon_a = 0$, this implies $\epsilon_a = 0$ for all a, which contradicts $\sum_a \epsilon_a^2 = mC$. Therefore, $n \leq m-1$.

Bound of the Positive and Negative Errors :

For the positive errors $\{\epsilon_i^+\}$, we have:

$$\sum_{i=1}^{n} \epsilon_i^+ \leq n \max_i \epsilon_i^+ < n\sqrt{\frac{C}{m-1}}.$$

Using the constraint $\sum_a \epsilon_a = 0$, the sum of the absolute values of the negative errors $\{\epsilon_j^-\}$ satisfies:

$$\sum_{j=1}^{m-n} |\epsilon_j^-| < n\sqrt{\frac{C}{m-1}}.$$

This implies:

$$\max_j |\epsilon_j^-| < n\sqrt{\frac{C}{m-1}}.$$

Hölder's Inequality

We use Hölder's inequality, we bound the sum of squares of the negative errors:

$$\sum_{j=1}^{m-n} (\epsilon_j^-)^2 \leq \sum_{j=1}^{m-n} |\epsilon_j^-| \cdot \max_j |\epsilon_j^-| < n\sqrt{\frac{C}{m-1}} \cdot n\sqrt{\frac{C}{m-1}} = \frac{Cn^2}{m-1}.$$

The total sum of squares of all errors is:

$$\sum_{a=1}^{m} \epsilon_a^2 = \sum_{i=1}^{n} (\epsilon_i^+)^2 + \sum_{j=1}^{m-n} (\epsilon_j^-)^2 < n\frac{C}{m-1} + \frac{Cn^2}{m-1} = \frac{Cn(n+1)}{m-1}.$$

Since $n \leq m-1$, we have:

$$\sum_{a=1}^{m} \epsilon_a^2 < mC,$$

which contradicts the assumption $\sum_a \epsilon_a^2 = mC$. Therefore, our initial assumption is false, and:

$$\max_a \epsilon_a \geq \sqrt{\frac{C}{m-1}}.$$

Tightness of the Bound

The bound is tight because equality holds when:

$$\epsilon_a = \sqrt{\frac{C}{m-1}} \quad \text{for } a = 1, \ldots, m-1,$$

and

$$\epsilon_m = -\sqrt{(m-1)C}.$$

This satisfies both $\sum_a \epsilon_a = 0$ and $\sum_a \epsilon_a^2 = mC$.

The proof demonstrates that even with unbiased Q-value estimates, the maximum Q-value is biased upward due to the inherent structure of the maximization operator in Q-learning. This result motivates the need for methods like Double Q-learning to mitigate overestimation.

□

Experiments on different games

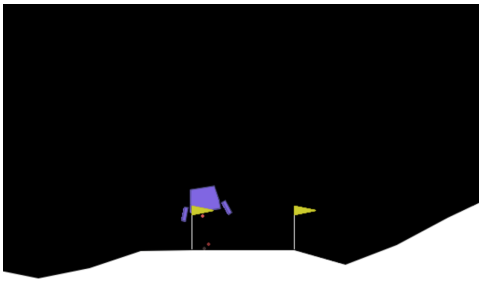


Figure 3: Lunarlander Image 1

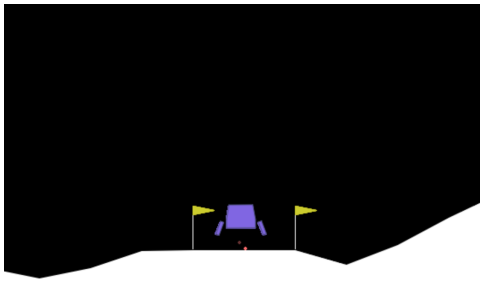


Figure 4: Lunarlander Image 2

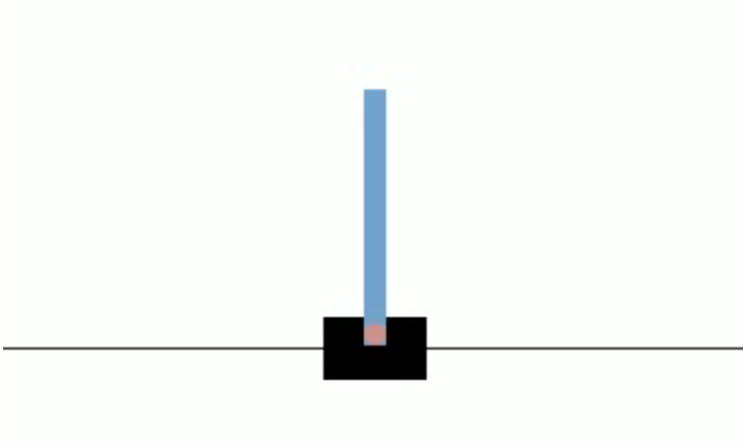


Figure 5: Cartpole

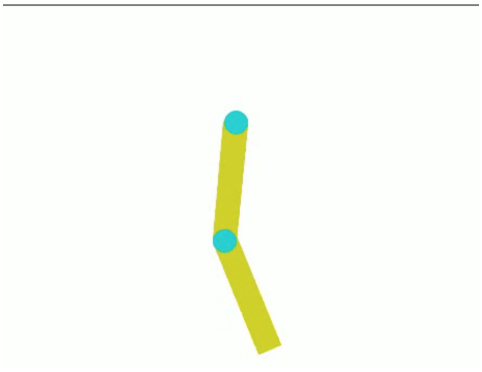


Figure 6: Acrobot Image 1

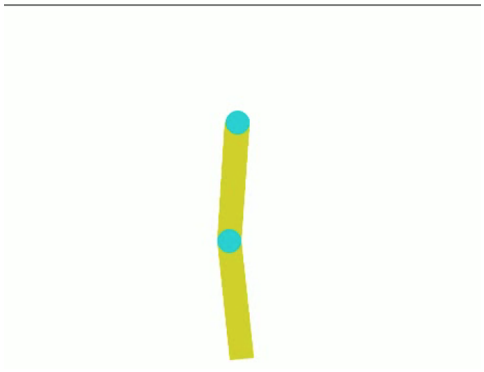


Figure 7: Acrobot Image 2

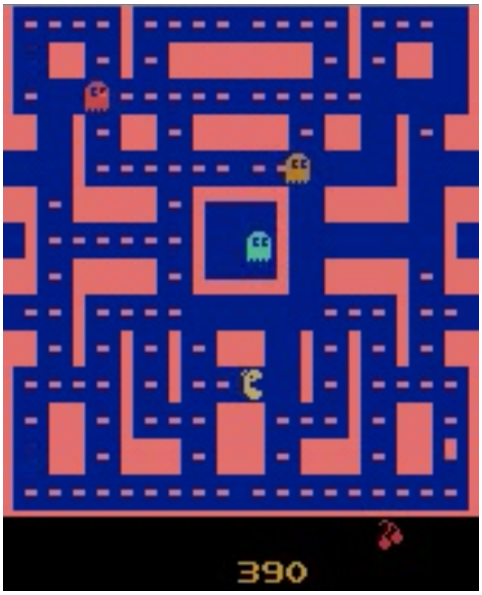


Figure 8: Pacman Image 1

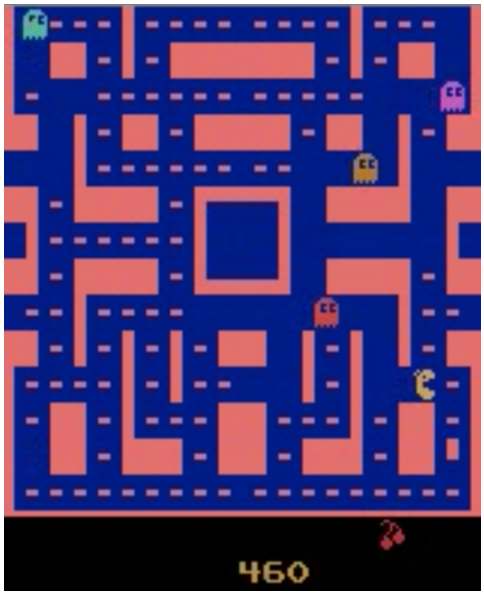


Figure 9: Pacman Image 2

GitHub

Instructions to run the code are available in the README file of the following github:

GitHub Repository link : $https://github.com/arazig/Deep-Reinforcement-Learning$

References

David Silver Hado van Hasselt, Arthur Guez. Deep reinforcement learning with double q-learning. *AAAI*, 30, 2016.

Rémi Munos Marc G. Bellemare, Will Dabney. A distributional perspective on reinforcement learning. *ICML*, 2017.

Hado van Hasselt Tom Schaul Georg Ostrovski Will Dabney Dan Horgan Bilal Piot Mohammad Azar David Silver Matteo Hessel, Joseph Modayil. Rainbow: Combining improvements in deep reinforcement learning. 2018.

Bilal Piot Jacob Menick Ian Osband Alex Graves Vlad Mnih Remi Munos Demis Hassabis Olivier Pietquin Charles Blundell Shane Legg Meire Fortunato, Mohammad Gheshlaghi Azar. Noisy networks for exploration. *ICLR*, 2018.

Ioannis Antonoglou David Silver Tom Schaul, John Quan. Prioritized experience replay. *ICLR*, 2016.

David Silver Andrei A. Rusu Joel Veness Marc G. Bellemare Alex Graves Martin Riedmiller Andreas K. Fidjeland Georg Ostrovski Stig Petersen Charles Beattie Amir Sadik Ioannis Antonoglou Helen King Dharshan Kumaran Daan Wierstra Shane Legg Demis Hassabis Volodymyr Mnih, Koray Kavukcuoglu. Human-level control through deep reinforcement learning. *Nature*, 518, 2015.

Matteo Hessel Hado van Hasselt Marc Lanctot Nando de Freitas Ziyu Wang, Tom Schaul. Dueling network architectures for deep reinforcement learning. *ICML*, 2016.