

# Ma première IA jouant à un jeu vidéo

PROJ104 Rapport de projet

Henri Besancenot

Axel Daboust

Clément Gilli

Iliass Khoutaibi

Guided by Pascal Blanchi

2024

# Table des matières

- ① Introduction
- ② Formalisation du problème
  - Théorèmes fondamentaux
- ③ Q-Learning
  - L'algorithme
  - Résultat
  - Limites du Q-Learning
- ④ Deep Q-Learning
  - From Q-Learning to Deep Q-Learning
  - Preprocessing
  - Double Deep Q-Learning

# Introduction

## Introduction

Dans le cadre de ce projet, nous avons eu l'opportunité de travailler dans le domaine de l'IA appliqué aux jeux vidéos. L'objectif était d'entraîner des IA à jouer à des jeux, en commençant par des jeux simples comme le Cart-pole, puis complexifier progressivement les jeux en passant par des jeux Atari et finalement arriver jusqu'à des jeux comme Mario Bros et Doom où les environnements sont très complexes.

# Formalisation du problème

### Petite remarque...

L'ensemble des démonstration et des lemmes fournis dans cette présentation se retrouveront dans le compte rendu mis-projet. On admet provisoirement les résultats.

- ① Introduction
- ② Formalisation du problème
  - Théorèmes fondamentaux
- ③ Q-Learning
  - L'algorithme
  - Résultat
  - Limites du Q-Learning
- ④ Deep Q-Learning
  - From Q-Learning to Deep Q-Learning
  - Preprocessing
  - Double Deep Q-Learning



## Théorème (Algorithme Q-learning)

$$Q_{k+1}(s_t, a) = Q_k(s_t, a) + \alpha[r(s_t, a) + \beta \max_{a'} Q_k(s_{t+1}, a') - Q_k(s_t, a)] \mathbb{1}_{(S_t=s_t, A_t=a)}$$
*où  $s_{t+1}$  correspond à l'état du système après l'action  $a$ .*

## Théorème (Algorithme Q-learning)

$$Q_{k+1}(s_t, a) = Q_k(s_t, a) + \alpha[r(s_t, a) + \beta \max_{a'} Q_k(s_{t+1}, a') - Q_k(s_t, a)] \mathbb{1}_{(S_t=s_t, A_t=a)}$$
*où  $s_{t+1}$  correspond à l'état du système après l'action  $a$ .*

## Théorème (Équation de Bellman)

$$Q_{\pi}(s, a) = r(s, a) + \beta \sum_{s', a'} Q_{\pi}(s', a') \cdot \pi(a'|s') \cdot p(s'|s, a)$$

# Théorèmes fondamentaux

## Théorème (Algorithme Q-learning)

$$Q_{k+1}(s_t, a) = Q_k(s_t, a) + \alpha[r(s_t, a) + \beta \max_{a'} Q_k(s_{t+1}, a') - Q_k(s_t, a)] \mathbb{1}_{(S_t=s_t, A_t=a)}$$

où  $s_{t+1}$  correspond à l'état du système après l'action  $a$ .

## Théorème (Équation de Bellman)

$$Q_\pi(s, a) = r(s, a) + \beta \sum_{s', a'} Q_\pi(s', a') \cdot \pi(a'|s') \cdot p(s'|s, a)$$

## Théorème (Équation de Bellman★)

$$Q^\star(s, a) = r(s, a) + \beta \sum_{s'} \max_{a'} Q^\star(s', a') \cdot p(s'|s, a)$$

# Q-Learning

- ① Introduction
- ② Formalisation du problème
  - Théorèmes fondamentaux
- ③ Q-Learning
  - L'algorithme
  - Résultat
  - Limites du Q-Learning
- ④ Deep Q-Learning
  - From Q-Learning to Deep Q-Learning
  - Preprocessing
  - Double Deep Q-Learning

Voici une implémentation ***primaire*** de l'algorithme **Q-learning** que le PROJ104 team a conçu afin de résoudre les environnements simples, notamment Cartpole et Mountain Car.

---

## Algorithm Q-Learning simple

---

- 1: Initialiser l'environnement, les variables d'environnement et le nombre d'épisode M
- 2: **for** i allant de 1 à M **do**
- 3:     Initialiser l'état s
- 4:     **while** l'agent n'a pas perdu **do**
- 5:         Executer l'action a qui vérifie  $a_t = \operatorname{argmax}_a Q(s_t, a)$
- 6:         Récupération des dans  $s_{t+1}$  observations et du la récompense
- 7:         Actualisation des variables d'environnement
- 8:         Actualisation de la matrice en appliquant Bellman
- 9:         Actualisation de  $s_t$  avec  $s_{t+1}$
- 10:     **end while**
- 11: **end for**

- ① Introduction
- ② Formalisation du problème
  - Théorèmes fondamentaux
- ③ Q-Learning
  - L'algorithme
  - **Résultat**
  - Limites du Q-Learning
- ④ Deep Q-Learning
  - From Q-Learning to Deep Q-Learning
  - Preprocessing
  - Double Deep Q-Learning



# Résultats

En implémentant cet algorithme, nous avons très rapidement eu des résultats corrects. Par exemple, pour le jeu du pendule, il arrive à se stabiliser en haut directement et y reste sans jamais perdre. Il faut noter que l'algorithme converge très rapidement vers une solution optimale.

# Résultats

En implémentant cet algorithme, nous avons très rapidement eu des résultats corrects. Par exemple, pour le jeu du pendule, il arrive à se stabiliser en haut directement et y reste sans jamais perdre. Il faut noter que l'algorithme converge très rapidement vers une solution optimale.

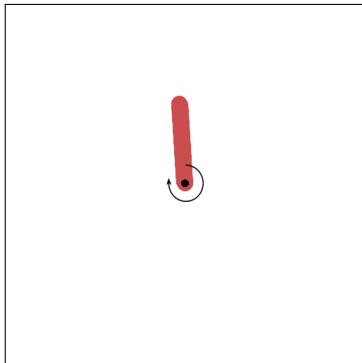


Figure – Pendulum

# et pour Cartpole?

De même pour CartPole, l'agent continue d'accumuler progressivement des rewards jusqu'à atteindre **1000 points** en moyenne!

# et pour Cartpole?

De même pour CartPole, l'agent continue d'accumuler progressivement des rewards jusqu'à atteindre **1000 points** en moyenne !

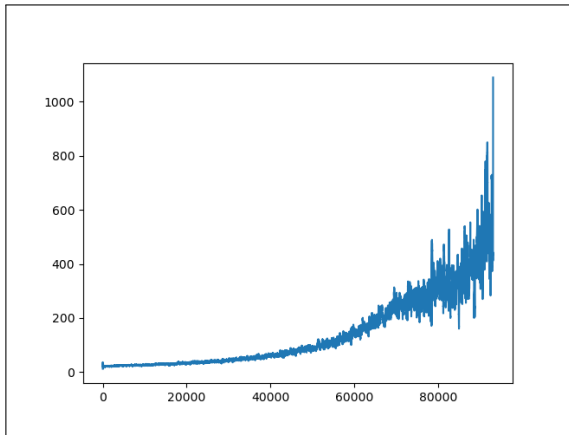


Figure – Means de Cartpole

- ① Introduction
- ② Formalisation du problème
  - Théorèmes fondamentaux
- ③ Q-Learning
  - L'algorithme
  - Résultat
  - Limites du Q-Learning
- ④ Deep Q-Learning
  - From Q-Learning to Deep Q-Learning
  - Preprocessing
  - Double Deep Q-Learning

## Limites du Q-Learning

Le Q-learning nous a donc permis d'entraîner notre IA sur tous ces environnements assez simples. Plusieurs problèmes se posent si on passe à des environnements plus complexes :

## Limites du Q-Learning

Le Q-learning nous a donc permis d'entraîner notre IA sur tous ces environnements assez simples. Plusieurs problèmes se posent si on passe à des environnements plus complexes :

- l'algorithme repose sur la manipulation d'une matrice de taille proportionnelle au nombre d'actions possible ainsi que de la taille de l'environnement.

## Limites du Q-Learning

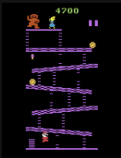
Le Q-learning nous a donc permis d'entraîner notre IA sur tous ces environnements assez simples. Plusieurs problèmes se posent si on passe à des environnements plus complexes :

- l'algorithme repose sur la manipulation d'une matrice de taille proportionnelle au nombre d'actions possible ainsi que de la taille de l'environnement.
- l'algorithme du Q-learning ne nous permet pas de traiter des environnements de type RGB comme Pong et Breakout puisque une action prise à chaque itération est liée à une analyse d'une image.



# Action space très large

On voit que dans l'exemple de donkey kong, Action space est égal à 18, ce qui est trop pour l'algorithme précédent !



This environment is part of the [Atari environments](#). Please read that page first for general information.

Action Space	Discrete(18)
Observation Space	Box(0, 255, (210, 160, 3), uint8)
Import	<code>gymnasium.make("ALE/DonkeyKong-v5")</code>

Figure – Action space de Donkey Kong

# Observation = Image

Action Space	Discrete(4)
Observation Space	Box(0, 255, (210, 160, 3), uint8)
Import	<code>gymnasium.make("ALE/Breakout-v5")</code>

Figure – Breakout observation space

Action Space	Discrete(2)
Observation Space	Box([-4.8000002e+00 -3.4028235e+38 -4.1887903e-01 -3.4028235e+38], [4.8000002e+00 3.4028235e+38 4.1887903e-01 3.4028235e+38], (4,), float32)
import	<code>gymnasium.make("CartPole-v1")</code>

Figure – Cartpole observation space

# Deep Q-Learning

- ① Introduction
- ② Formalisation du problème
  - Théorèmes fondamentaux
- ③ Q-Learning
  - L'algorithme
  - Résultat
  - Limites du Q-Learning
- ④ Deep Q-Learning
  - From Q-Learning to Deep Q-Learning
  - Preprocessing
  - Double Deep Q-Learning



## Deep Q-Learning

Heureusement, le domaine de l'intelligence artificielle a connu plusieurs développements ces dernières années notamment en 2015 avec la publication de recherches scientifiques importantes comme : **Playing Atari with Deep Reinforcement Learning** où ils s'intéressent à la méthode du **Deep Q-learning**.

# From Q-learning to Deep Q-learning

Comme nous l'avons dit plus tôt, nous voulons à présent nous attaquer à des environnements plus complexes.

# From Q-learning to Deep Q-learning

Comme nous l'avons dit plus tôt, nous voulons à présent nous attaquer à des environnements plus complexes.

On va remplacer cette matrice  $Q$  par **un réseau de neurones** renvoyant "virtuellement" les coefficients d'une matrice selon un environnement donné.



# Deep Q-Learning sans images

---

## Algorithm Deep Q-Learning

---

- 1: Initialiser un replay Memory  $D$  de capacite  $N$
- 2: Initialiser un reseau de Neuronne  $Q$
- 3: **for**  $i$  allant de 1 à  $M$  **do**
- 4:     Initialiser l'état  $s_1 = x_1$
- 5:     **while** l'agent n'a pas perdu **do**
- 6:         avec une perobabilite  $\epsilon$  choisir une action random  $a$
- 7:         sinon  $a = \max_{a'} Q(s, a')$
- 8:         Executer l'action  $a$  et observer la reward  $r$  et la nouvelle image  $\tilde{x}$
- 9:         Actualiser  $\tilde{s} = s, a, \tilde{x}$
- 10:        Enregistrer  $(s, a, r, \tilde{s})$  dans  $D$
- 11:        Prendre aleatoirement des transition  $(\phi, a, r, \tilde{\phi})$  de  $D$
- 12:        
$$y = R(s, a) + \gamma \cdot \max_{a'} Q(\tilde{s}, \tilde{a})$$
- 13:        Performer la descente du gradient à  $(y - Q(\phi, a_j))^2$
- 14:     **end while**
- 15: **end for**

# Application- Lunar Lander

Voici la reward de **Lunar Lander**

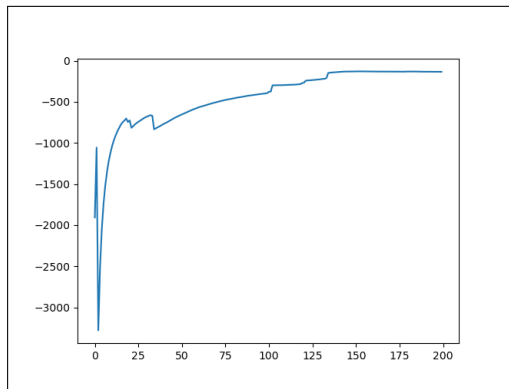


Figure – Rewards de Lunar Lander

**Peut-On appliquer l'algorithme  
précédent à des  
environnements avec des  
images ?**

**MAIS**

# MAIS

Si on traite directement les images ...

## MAIS

Si on traite directement les images ...

L'analyse sera très **lourde** et **complexe** pour nos réseaux de neurones

## MAIS

Si on traite directement les images ...

L'analyse sera très **lourde** et **complexe** pour nos réseaux de neurones  
Il faut passer par un étape intermédiaire de simplification d'image :

## MAIS

Si on traite directement les images ...

L'analyse sera très **lourde** et **complexe** pour nos réseaux de neurones

Il faut passer par un étape intermédiaire de simplification d'image :

**Le Preprocessing**



- ① Introduction
- ② Formalisation du problème
  - Théorèmes fondamentaux
- ③ Q-Learning
  - L'algorithme
  - Résultat
  - Limites du Q-Learning
- ④ Deep Q-Learning
  - From Q-Learning to Deep Q-Learning
  - **Preprocessing**
  - Double Deep Q-Learning

## Étapes du Preprocessing

Après de recherches extensives, nous avons trouvé quatre étapes importantes du preprocessing à suivre afin d'accélérer la convergence de notre Q matrice :

## Étapes du Preprocessing

Après de recherches extensives, nous avons trouvé quatre étapes importantes du preprocessing à suivre afin d'accélérer la convergence de notre Q matrice :

### ① Grayscale

## Étapes du Preprocessing

Après de recherches extensives, nous avons trouvé quatre étapes importantes du preprocessing à suivre afin d'accélérer la convergence de notre Q matrice :

- 1 Grayscale
- 2 Resizing

## Étapes du Preprocessing

Après de recherches extensives, nous avons trouvé quatre étapes importantes du preprocessing à suivre afin d'accélérer la convergence de notre Q matrice :

- ① **Grayscaleing**
- ② **Resizing**
- ③ **Frame-Skipping**

## Étapes du Preprocessing

Après de recherches extensives, nous avons trouvé quatre étapes importantes du preprocessing à suivre afin d'accélérer la convergence de notre Q matrice :

- ① **Grayscale**
- ② **Resizing**
- ③ **Frame-Skipping**
- ④ **Frame-Stacking**

L'image reçue devient grise.

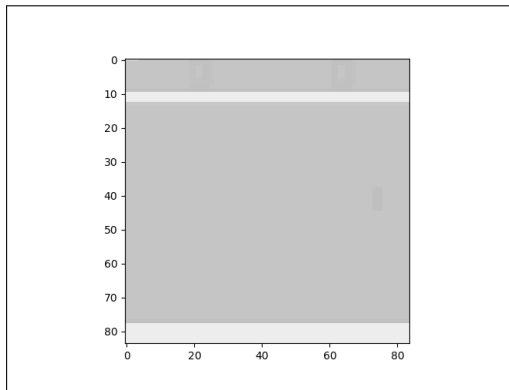
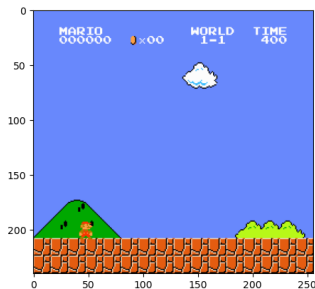


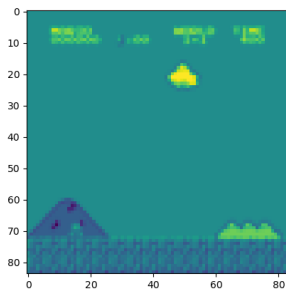
Figure – Grayscaled Pong

# Resizing

On modifie la taille de notre matrice image, on change la taille souvent à **84x84**, une valeur convenable qu'on a trouvé expérimentalement après plusieurs essais.



(a) Mario sans resize



(b) Mario avec du resize



# Frame-Skipping

pour une valeur  $n$  qui représente le Frame-skipping, il s'agit de répéter une action  $n$  fois, afin d'explorer le maximum de possibilité et de nouveaux espaces d'observation.

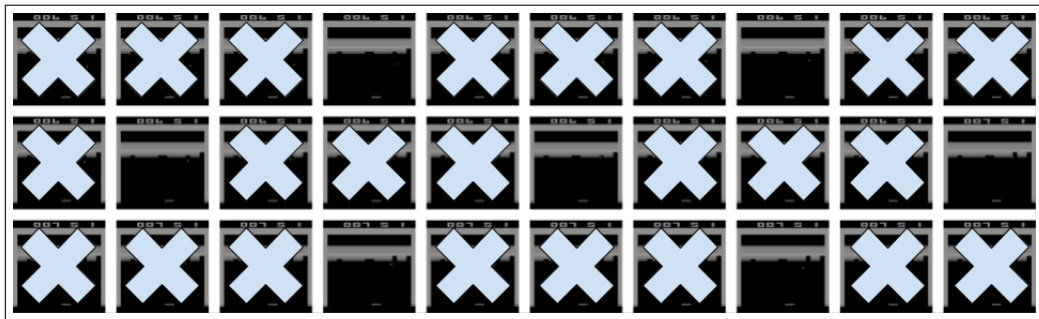
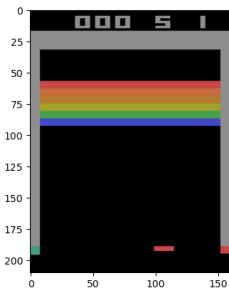


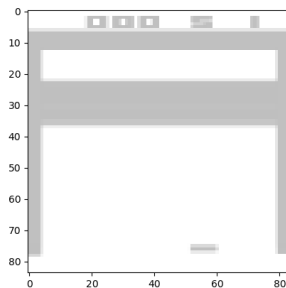
Figure – Frame-skipping pour  $n = 4$

# Frame-Stacking

pour la même valeur  $n$  du Frame-Skipping, on stack les  $n$ -dernières frames pour obtenir un tensor de taille **[ $n \times 84 \times 84$ ]**. Le Frame-Stacking permet à l'agent d'analyser  $n$  actions et  $n$  espaces d'observation à chaque itération au lieu d'une seule action et un seul espace d'observation.



(a) Breakout sans preprocessing



(b) Breakout avec du preprocessing

Pour optimiser la convergence, nous nous sommes à nouveau appuyés sur le papier de 2015 qui se limite seulement à l'assignation de la fonction de récompense  $R$  dans le cas où un épisode est terminé. L'équation de Bellman devient :

Pour optimiser la convergence, nous nous sommes à nouveau appuyés sur le papier de 2015 qui se limite seulement à l'assignation de la fonction de récompense  $R$  dans le cas où un épisode est terminé. L'équation de Bellman devient :

## Théorème (Équation de Bellman)

$$Q(s, a) = R(s, a) + \gamma \cdot (1 - \text{terminated}) \cdot \max_{a'} Q'(s', a')$$

## Algorithm Deep Q-Learning

- 1: Initialiser un replay Memory  $D$  de capacite  $N$
- 2: Initialiser un reseau de Neuronne  $Q$
- 3: **for**  $i$  allant de 1 à  $M$  **do**
- 4:     Initialiser l'état  $s_1 = x_1$  et l'état preprocessed  $\phi = \phi(s_1)$
- 5:     **while** l'agent n'a pas perdu **do**
- 6:         avec une perobabilite  $\epsilon$  choisir une action random  $a$
- 7:         sinon  $a = \max_{a'} Q(s, a')$
- 8:         Executer l'action  $a$  et observer la reward  $r$  et la nouvelle image  $\tilde{x}$
- 9:         Actualiser  $\tilde{s} = s, a, \tilde{x}$  et prerocess  $\tilde{\phi} = \phi(\tilde{s})$
- 10:        Enregistrer  $(\phi, a, r, \tilde{\phi})$  dans  $D$
- 11:        Prendre aleatoirement des transition  $(\phi, a, r, \tilde{\phi})$  de  $D$
- 12:

$$y = \begin{cases} R(s, a) & \text{si est terminale} \\ R(s, a) + \gamma \cdot \max_{a'} Q(\tilde{s}, \tilde{a}) & \text{sinon} \end{cases}$$

- 13:     Appliquer la descente du gradient à  $(y - Q(\phi, a_j))^2$
- 14: **end while**

**Peut-on améliorer encore la vitesse de convergence de  
notre Matrice**

**Peut-on améliorer encore la vitesse de convergence de  
notre Matrice**

Oui! En utilisant deux réseaux de neurones!

Peut-on améliorer encore la vitesse de convergence de  
notre Matrice

Oui! En utilisant deux réseaux de neurones!  
C'est ce qu'on appelle le **Double Deep Q-Learning**



- ① Introduction
- ② Formalisation du problème
  - Théorèmes fondamentaux
- ③ Q-Learning
  - L'algorithme
  - Résultat
  - Limites du Q-Learning
- ④ Deep Q-Learning
  - From Q-Learning to Deep Q-Learning
  - Preprocessing
  - Double Deep Q-Learning

# Double Deep Q-Learning

## Deep Q-Learning

Une variante a été utilisée ici, qui est d'utiliser un deuxième réseau de neurones lors de l'apprentissage.

## Deep Q-Learning

Une variante a été utilisée ici, qui est d'utiliser un deuxième réseau de neurones lors de l'apprentissage.

Sa principale motivation est un critère de stabilité et de convergence. Dans l'équation de Bellman.

## Deep Q-Learning

Une variante a été utilisée ici, qui est d'utiliser un deuxième réseau de neurones lors de l'apprentissage.

Sa principale motivation est un critère de stabilité et de convergence. Dans l'équation de Bellman.

Un deuxième réseau, souvent appelé "target", servira à la recherche de la valeur maximale accessible depuis le nouvel état.

# Double Deep Q-learning

## Deep Q-Learning

Une variante a été utilisée ici, qui est d'utiliser un deuxième réseau de neurones lors de l'apprentissage.

Sa principale motivation est un critère de stabilité et de convergence. Dans l'équation de Bellman.

Un deuxième réseau, souvent appelé "target", servira à la recherche de la valeur maximale accessible depuis le nouvel état.

Il sera d'ailleurs actualisé avec les poids du réseau principal au bout d'un certain nombre d'épisodes d'entraînement.

- Posons  $s_t$  un état de notre environnement à l'instant  $t$ ,  $Q_{eval}$  notre réseau principal,  $r_t$  la reward et  $a_t$  l'action.

- Posons  $s_t$  un état de notre environnement à l'instant  $t$ ,  $Q_{eval}$  notre réseau principal,  $r_t$  la reward et  $a_t$  l'action.
- Posons  $s_{t+1}$  le nouvel état de notre environnement à l'instant  $t+1$  et  $Q_{target}$  notre réseau target.



- On évalue  $Q_{eval}(s_t, a_t)$  et on applique **L'équation de Bellman** à  $Q_{target}$

- On évalue  $Q_{eval}(s_t, a_t)$  et on applique **L'équation de Bellman** à  $Q_{target}$
- Ensuite on applique la descente du gradient à  $(Q_{eval}(s_t, a_t) - Q_{target})^2$

- On évalue  $Q_{eval}(s_t, a_t)$  et on applique **L'équation de Bellman** à  $Q_{target}$
- Ensuite on applique la descente du gradient à  $(Q_{eval}(s_t, a_t) - Q_{target})^2$
- Enfin, après N-épisodes, on met à jour les poids de  $Q_{target}$  avec ceux de  $Q_{eval}$

# Application : Car Racing

Voici les résultats de l'entraînement de l'environnement : **Car-racing** avec le double Deep Q-Learning avec une vidéo d'exécution.

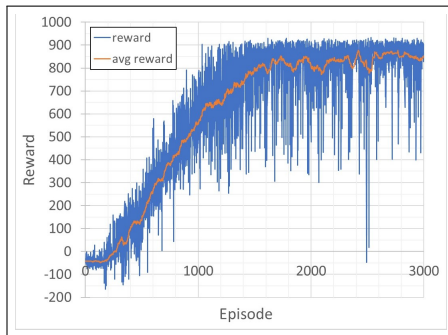


Figure – Rewards de Car-racing

# Application : Pong/Breakout

Voici les résultats de l'entraînement des environnements : **Pong** et **Breakout**  
avec des vidéos : **Breakout** et **Pong**

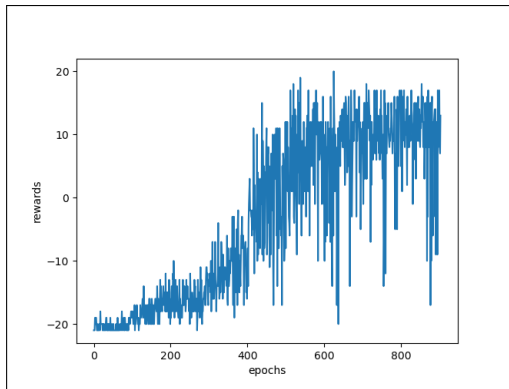


Figure – Rewards de Pong

On a quelques problèmes avec Mario et Doom et cela est du une autre fois au nombre importants d'actions des environnements. Voici quelques vidéos d'execution apres des entrianements.

# Future Work

# Conclusion and Future Work

Finalement, avec cette approche de Deep Q-learning, nous visons à étendre nos recherches et applications à des jeux encore plus complexes (Mario/Doom...), en améliorant constamment nos algorithmes pour gérer des environnements dynamiques et imprévisibles.



# Conclusion and Future Work

Finalement, avec cette approche de Deep Q-learning, nous visons à étendre nos recherches et applications à des jeux encore plus complexes (Mario/Doom...), en améliorant constamment nos algorithmes pour gérer des environnements dynamiques et imprévisibles.

Notre prochaine étape consiste à explorer des techniques plus avancées de renforcement profond :

- DDPG ou le Deep Deterministic Policy Gradient

# Conclusion and Future Work

Finalement, avec cette approche de Deep Q-learning, nous visons à étendre nos recherches et applications à des jeux encore plus complexes (Mario/Doom...), en améliorant constamment nos algorithmes pour gérer des environnements dynamiques et imprévisibles.

Notre prochaine étape consiste à explorer des techniques plus avancées de renforcement profond :

- DDPG ou le Deep Deterministic Policy Gradient
- PPO ou le Proximal Policy Optimization

# The End

Lien vers le Gitlab du Team Proj104 : ia-jeu