# 1 – Préliminaires

Le code source du projet est disponible sur mon dépôt public : <https://github.com/NajibXY/DRL>

J’ai utilisé Python 3.8, PyTorch dans un environnement virtuel Pipenv.

La méthodologie de réplicabilité est décrite dans README.md et les dépendances sont enregistrées dans requirements.txt.

# 2 – Deep Q-network sur CartPole

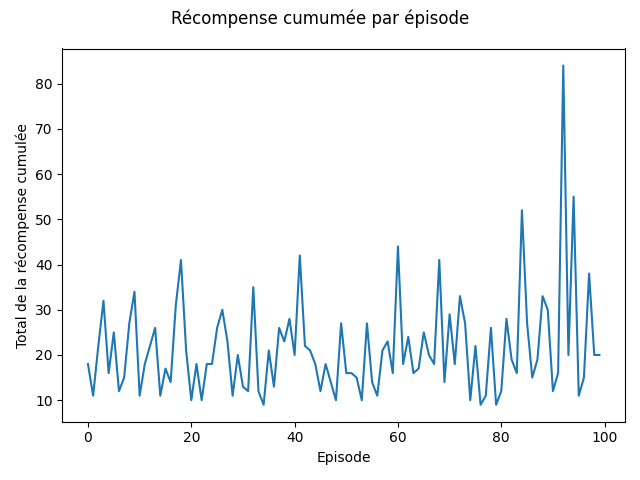
## 2.1 – Début

* Le code de l’agent aléatoire est disponible dans le fichier random\_agent.py. On peut changer l’environnement dans lequel il agit en modifiant la valeur default de la ligne :

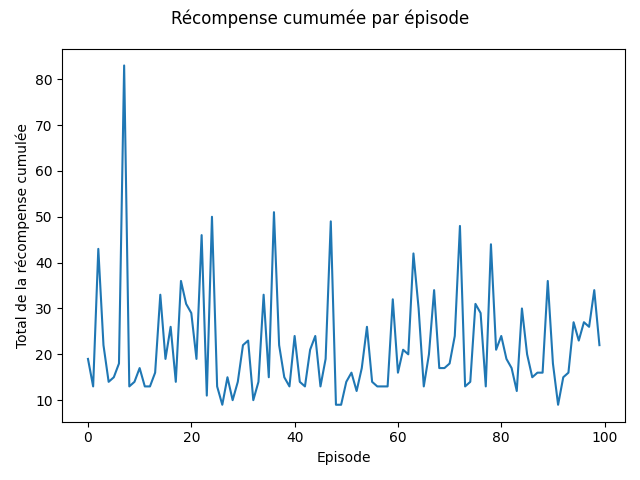
parser**.**add\_argument**(**'env\_id'**,** nargs**=**'?'**,** default**=**'CartPole-v1'**,** **help=**'Select the environment to run'**)**

* Afin de visualiser l’évolution de la récompense, j’ai utilisé matplotlib pour afficher la récompense cumulée par épisode afin d’avoir un aperçu global des performances de l’agent sur un total de 100 épisodes

***Exemple de résultat n°1 :***



***Exemple de résultat n°2 :***



Sans surprise, les résultats sont chaotiques et oscillent aléatoirement, il n’y a pas d’apprentissage.

## 2.2 – Experience replay

### Interaction

La structure d’une interaction est définie par la classe Interaction en tant que tuple classique (e, a, s, r, f).

### Buffer

La classe Buffer implémente quant à elle le buffer, un buffer a un size et un tableau d’interactions (données) duquel on pop l’interaction à l’indice 0 à chaque fois que l’on rajoute une interaction si cela nous fait dépasser la limite de taille.

La classe buffer possède donc une méthode .append(), mais également une méthode .sample() permettant d’implémenter la fonction de sampling pour récupérer un minibatch de données aléatoires dans les interactions stockées par le buffer, et ce en utilisant la méthode .sample(x, n) du package random où n est la taille du sample : random**.**sample**(**self**.**buffr**,** **).**