Alexandre Gripari
Alban Falcoz

Flappy Bird RL

I. RL appliqué à Flappy Bird

Observations et actions

Le modèle d'apprentissage repose sur trois observations en entrée :

- **dx** : la distance horizontale entre l'oiseau et le prochain tuyau.
- **dy** : la distance verticale entre l'oiseau et l'ouverture du prochain tuyau.
- **vel_y** : la vitesse verticale actuelle de l'oiseau.

À partir de ces observations, l'agent doit choisir entre deux actions possibles :

- **flap (1)** : effectuer un battement d'aile pour faire monter l'oiseau.
- **skip (0)** : ne rien faire et laisser l'oiseau descendre naturellement.

Bien que le jeu soit potentiellement infini, l'espace des états peut être discrétisé de manière à le rendre fini. Cette caractéristique permet d'appliquer un algorithme d'apprentissage par renforcement de type **Q-Learning**, qui apprend la meilleure action à entreprendre pour chaque état possible afin de maximiser la récompense cumulée au fil du temps.

Reward function

Chaque frame passée en vie rapporte une petite récompense positive afin d'encourager la survie, il l'est également quand il aligner au centre de l'espace entre les tuyaux, tandis qu'une récompense plus importante est attribuée lorsque l'oiseau se rapproche de l'ouverture d'un tuyau et lorsqu'il le franchit complètement. Le saut est légèrement pénalisé afin de limiter les battements d'aile excessifs, tandis qu'une collision entraînant la mort de l'oiseau est fortement sanctionnée.

Q-Learning

Les trois observations en entrée : dx, dy et vel_y sont respectivement définies sur les plages de valeurs suivantes : $[0, 212]$, $[-104, 256]$ et $[-8, 10]$.

L'espace d'état résultant bien que fini, reste très vaste. En considérant toutes les combinaisons possibles, on obtient 213 valeurs pour dx, 361 pour dy et 19 pour vel_y, soit un total de $213 \times 361 \times 19 =$ environ 1,6 million d'états possibles. Comme deux actions peuvent être effectuées à partir de chaque état (flap ou skip), l'espace des paires état-action contient environ 2,9 millions de combinaisons.

Afin de rendre l'apprentissage plus efficace et garentir la convergence, une discrétisation de l'espace d'état a été appliquée. Les variables dx, dy et vel_y ont été réduites à respectivement 24, 36 et 5 intervalles. Cette approximation conduit à un espace d'état de $24 \times 36 \times 5 = 4\,320$ combinaisons possibles, soit 8 640 paires état-action lorsqu'on prend en compte les deux actions possibles. Cette réduction permet d'accélérer l'apprentissage tout en conservant une représentation suffisamment précise de la dynamique du jeu.

Deep-Q learning

Le Deep Q-Learning est une extension du Q-Learning classique, conçue pour résoudre ses limites face aux environnements complexes ou continus. Le DQL remplace la Q-table par un réseau de neurones capable d'estimer les valeurs d'action Q(s,a), ce qui permet de traiter des espaces d'états de grande dimension et des données continues sans devoir les discrétiser, offrant ainsi une meilleure scalabilité et une plus grande capacité de généralisation.

L'architecture d'un Deep Q-Network (DQN) repose sur trois éléments majeurs :

- Le réseau principal, qui approxime la fonction Q(s,a) et prédit les valeurs d'action.
- L'Experience Replay, un buffer qui stocke les expériences passées (s,a,r,s'). Lors de l'entraînement, des lots d'expériences sont tirés aléatoirement pour casser la corrélation temporelle et stabiliser l'apprentissage.
- Le Target Network, un second réseau mis à jour périodiquement à partir du réseau principal, permettant de calculer les valeurs cibles et d'éviter les oscillations lors de l'apprentissage.

Le processus d'entraînement suit alors plusieurs étapes : initialisation du modèle et des hyper-paramètres (learning rate, la taille des couches cachées, …), interaction avec l'environnement selon une politique ε-greedy (exploration/exploitation), stockage des expériences, puis mise à jour du réseau via la descente de gradient sur des mini-lots d'expériences. Les poids du réseau cible sont régulièrement synchronisés avec ceux du réseau principal.

Le DQL a donné des résultats très satisfaisants, dépassant les 1000 pipes très régulièrement et ne mourrant que dans des situations assez rares.

Dans le cadre d'une version du jeu où les pipes accélèrent petit à petit, le DQL a montré sa capacité à généraliser, faisant des scores plus élevés que le simple Q learning. Cela montre que le Deep Q-Learning ne se contente pas d'apprendre des associations état-action spécifiques, mais parvient à extraire des représentations plus abstraites de l'environnement, lui permettant d'adapter son comportement plus facilement face à de nouvelles situations.

II. Code

Récupérer le code : `git clone https://github.com/Alexandre-Gripari/FlapPyBird-RL.git`

Tout doit etre executer à partir de la racine du projet. !

Q-learning

Lancer un entrainement : `python -m src.q_learning.q_learning`

Faire jouer l'agent :

```
python -m src.q_learning.agent_q --q_matrix_path=CHEMIN_VERS_LA_MATRICE.npy
```

Changer les paramètres : Tous les paramètres (hyper-paramètres/discretization/nombre d'épisodes) sont définis dans une dataclass (TrainingConfig) il est donc possible de changer les valeurs par defaut ou d'instancer cette classe avec les paramètres souhaités.

Deep-Q

Lancer un entrainement : `python -m src.deep_q_learning.deep_q_learning`

Faire jouer l'agent : `python -m src.deep_q_learning.agent_deep_q_learning`