pollen robotics

Rapport de Stage

Intégration d'algorithmes d'apprentissage dans une borne d'arcade robotisée

Etudiant : David Albert

Maître de stage : M. Pierre Rouanet

Tuteur : M. Jean-Philippe Kotowicz

Table des matières

Remerciements

Je voudrais tout d'abord remercier mon maître de stage et CTO de l'entreprise Pierre Rouanet pour le temps qu'il m'a consacré. Merci à lui de m'avoir guidé tout au long de ce stage sur les différentes tâches ainsi que de m'avoir fait découvrir de nombreuses choses autour du language Python. Le temps passé à discuter des différentes techniques d'entraînement des modèles et des résultats obtenus m'a fortement aidé à cerner le sujet et ses difficultés.

Je voudrais ensuite remercier Matthieu Lapeyre (CEO de Pollen Robotics) pour son aide autour de la création de la voix d'Arcadeep. Matthieu m'a permis d'avoir à disposition tous les outils nécessaires à la réalisation de mon travail. Merci également d'avoir battu le meilleur score sur le jeu Flappy Bird Duo et ainsi d'avoir permis de tester les limites de l'IA.

Pour terminer, je voudrais remercier toute l'équipe de *Pollen Robotics* ainsi que l'équipe de *Luos Robotics* qui ont permis que ce stage se passe dans les meilleures conditions de travail possibles.

Introduction

J'ai effectué mon stage de spécialité dans l'entreprise Pollen Robotics. Ce stage a duré onze semaines au total. Durant ces semaines de stage j'ai travaillé sur un projet tout nouveau pour l'entreprise qui est la création d'une borne d'arcade. L'idée de Pierre et Matthieu (CTO et CEO de Pollen) était d'utiliser les nouvelles technologies ainsi que les récentes recherches dans le domaine de l'intelligence artificielle (notamment le domaine de l'apprentissage par renforcement) pour créer une borne de jeux d'arcade qui serait originale par son design mais surtout par son interaction avec le joueur. Une interaction que l'on souhaitera la plus complète et intelligente possible.

La borne de jeux d'arcade, dont le nom est Arcadeep, a pour but d'être vendu à des particuliers. Le projet n'ayant quasiment pas été commencé avant mon arrivée, mon objectif n'était pas de faire une version commercialisable mais plutôt de voir si l'idée était réalisable et s'il était possible d'apporter quelquechose de nouveau et de pertinent au joueur.

J'ai donc eu différentes tâches à réaliser autours de ce projet. J'ai passé environ la moitié du temps de mon stage à comprendre et travailler avec des outils d'apprentissage par renforcement pour faire apprendre à un agent (dans mon cas la borne) à jouer à certains jeux. L'autre partie de mon stage a consisté à implémenter des outils d'interaction entre Arcadeep et l'utilisateur. Je parlerais dans ce rapport de ces deux principaux axes. Tout d'abord je présenterais brièvement l'entreprise Pollen Robotics puis j'expliquerais dans une première partie les méthodes que j'ai utilisé pour faire apprendre un agent à jouer à certains jeux tels que Flappy Bird et Pacman. Je présenterais ensuite Arcadeep et les différentes formes d'interactions intelligentes qui la compose à ce jour.

Présentation de l'entreprise

Pollen Robotics est une entreprise fondée en mai 2016 par d'anciens chercheurs de l'Inria, experts en Intelligence Artificielle (IA) et en Robotique, et notamment internationalement reconnus pour le développement du robot humanoïde Poppy.

Pollen Robotics réalise des produits et services nécessitant l'intervention de machine intelligentes, capables d'agir en milieu ouvert et d'interagir avec des utilisateurs non-experts.

Son expertise est exploitée au travers de 3 activités commerciales :

- la commercialisation de la plateforme robotique Reachy, un bras robotique humanoïde open source à destination des équipes R&D et innovation de l'industrie,
- l'usage de Reachy pour créer des animations dans les espaces publiques.
- la prestation de service pour la réalisation de brique technologique robotique et IA sur mesure.

Pollen Robotics a également inventé la solution technologique modulaire Robus (breveté) et a créé une spin-off nommée Luos Robotics pour en assurer le développement commercial.

Activité de R&D et d'innovation

La recherche menée chez Pollen Robotics a pour but de mieux comprendre comment créer des robots capables d'agir et d'interagir dans un milieu ouvert en interaction avec le grand public.

Il s'agit d'un enjeu scientifique majeur car les technologies qui ont permis la robotisation massive des usines sont en échecs dès lors que le milieu d'intervention ou la tache à exécuter ne peuvent être parfaitement connus et programmés en amont par un ingénieur. Leur mécatronique hyper précise mais rigide et leur intelligence reposant principalement sur des systèmes de décision se heurtent à la complexité et à l'imprévu des événements ou environnements inconnus. Afin de pouvoir répondre à ces contraintes extrêmement complexes, il est nécessaire de développer de nouvelles approches permettant de construire automatiquement des représentations du monde de plus haut niveaux.

Pollen Robotics, cherche à rendre les robots opérationnels dans un milieu ouvert, typiquement dans des lieux publiques (gares, hôpitaux, villes, fermes...) ou au sein de l'habitation privée.

Pour cela, il faut utiliser des techniques totalement différentes de celles employées pour les robots industriels. Il faut favoriser l'apprentissage et l'émergence de comportements avancés au travers de l'apprentissage automatique (reinforcement learning par exemple). Il faut également que leur conception mécatronique permette de s'adapter physiquement au monde réel pour interagir efficacement avec un environnement ouvert tout sauf uniformisé. Enfin, un aspect souvent laissé de côté est l'interaction de ces machines avec le grand public (et potentiellement des animaux) qui doivent pouvoir communiquer facilement et naturellement

avec des robots de manière ponctuelle ou répétée, imprévue, hors d'un cadre pré-établi. Ainsi des robots agissant dans un milieu ouvert doivent pouvoir communiquer leurs intentions, leurs besoins et leurs états aux êtres humains autour d'eux, de manière suffisamment simple, naturelle et agréable afin de faciliter leur acceptation et leur utilisation.

Ainsi, faire sortir les robots des usines requiert d'avoir une approche globale en mêlant des innovations software, hardware et interfaces homme-machine pour créer des robots de services et sociaux utiles et performants. Ces enjeux scientifiques forment les principales thématiques de R&D chez Pollen Robotics :

- Créer des intelligences artificielles permettant à des robots d'apprendre à réaliser différentes taches et à évoluer dans un milieu ouvert, en interaction étroite avec des humains,
- la réalisation de plateforme robotique et de brique mécatronique permettant à des robots de pouvoir se déplacer et manipuler des objets en milieu ouvert,
- et des briques d'interaction agissant sur le comportement et les émotions affichées du robot afin de faciliter son acceptation et son interaction avec le grand public (personnes non formées à la robotique).

FIGURE 1 – Pollen Robotics

Part I

Méthodes d'apprentissage par renforcement

L'apprentissage par renforcement est un domaine de l'intelligence artificielle très populaire de nos jours. Il constitue, avec l'apprentissage supervisé et l'apprentissage non supervisé, l'un des trois principaux sous-domaines de l'apprentissage automatique. Les méthodes d'apprentissage par renforcement se sont nettement popularisées ces dernier temps parce-qu'elles ont montré qu'elles pouvaient aboutir à de très bons résultats dans certaines applications. Un exemple bien connu car il est remarquable est la victoire d'AlphaGo (modèle créé par DeepMind) qui a battu le coréen Lee Se-Dol, champion du monde de jeu de Go. Pendant mon stage j'ai étudié certaines méthodes récentes d'apprentissage par renforcement dans le but de créer une intelligence artificielle capable de progresser au fur et à mesure des parties sur différents jeux.

Dans cette première partie nous ferons un bref état de l'art de l'apprentissage par renforcement. Tout d'abord, nous introduirons le concept général et les enjeux de ce type d'algorithmes et les notations utilisées. Puis nous nous pencherons sur la partie algorithmique en introduisant les méthodes basées sur des échantillons et les méthodes utilisant des approximations de fonction. Enfin nous verrons quelques astuces et outils pour améliorer les résultats et simplifier l'apprentissage.

1 Les fondamentaux de l'apprentissage par renforcement

L'apprentissage par renforcement est le fait d'apprendre ce qu'il faut faire dans une situation donnée. C'est le fait de comprendre les conséquences de ses actions. Les conséquences, positives ou non, sont perçues sous la forme d'un simple signal récompense. Ce processus d'apprentissage en interagissant avec l'environnement est probablement le plus utilisé par l'enfant dans la première année de sa vie.

Nous posons ici les concepts mathématiques de base de ce type d'apprentissage.

1.1 Système d'apprentissage par renforcement

Un système d'apprentissage par renforcement se compose de trois parties.

- **L'environnement** : Comme son nom l'indique il s'agit du contexte dans lequel l'agent (défini ci-après) évolu. Dans la majeure partie des cas, l'environnement est amené à

évoluer et on définit donc l'espace des états de l'environnement par $\mathcal{S}$. On appelle *modèle de l'environnement* la façon dont celui-ci évolue au cours du temps, on le note $p(s', r \mid s, a)$.

- **L'agent :** Il sagit d'une entité pouvant agir selon un comportement qui lui est propre. Ses actions son étroitement liées à sa perception de l'environnement. Dans le cadre de l'agent, plusieurs termes peuvent être défini. L'agent observe l'état de l'environnement, on a donc un espace d'observation $\mathcal{O}$. De plus, l'agent peut exécuter différentes actions, on a donc l'espace des actions possibles défini par $\mathcal{A}$. Pour finir, le comportement de l'agent (qui sera appelé stratégie par la suite) est représenté par la fonction $\pi : \mathcal{S} \to \mathcal{A}$.
- **La récompense :** Le dernier élément d'un système d'apprentissage par renforcement est la récompense. La récompense est un signal perçu par l'agent et émis par l'environnement : $r \in \mathbb{R}$

Citons, à titre d'exemples, quelques systèmes d'apprentissage par renforcement.

Ex 1. Une voiture autonome (l'agent) circulant (les actions) sur un circuit (l'environnement) avec comme récompense le nombre de tours de circuit qu'elle a réalisé. L'ensemble des états de ses capteurs forment l'observation.

Ex 2. Un chien (l'agent) qui reçoit de la nourriture (récompense) quand il fait les tâches que son maître lui dis (l'environnement). L'observation dans ce cas est ce qu'il a vu et entendu.

Ex 3. La borne de jeu Arcadeep (l'agent) qui gagne des points (récompense) quand elle gagne un jeu ou que le joueur s'amuse (l'environnement).

Ces exemples pourront être réutilisés par la suite pour appuyer les notions futurs.

La dynamique du système

Tout système d'apprentissage par renforcement évolu au cours du temps sous la même dynamique. A chaque instant t :

- L'agent reçoit un état $s_t \in \mathcal{S}$ et une récompense r_t et émet une action $A_t \in \mathcal{A}$.
- L'environnement reçoit une action $A_t \in \mathcal{A}$ et émet un état $s_{t+1} \in \mathcal{S}$ et r_{t+1}.

Ce qui se résume avec le schéma suivant :

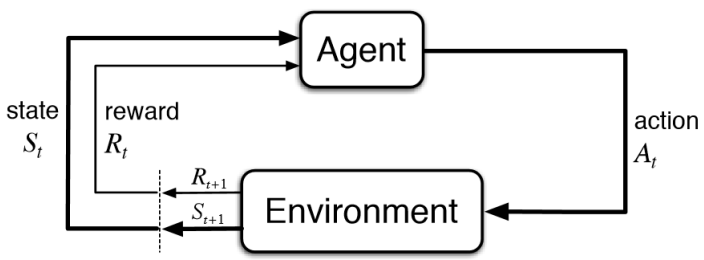


FIGURE I.1 – Schéma de la dynamique d'un système d'AR

1.2 Accomplir un objectif

Nous nous concentrons désormais sur l'agent. Le but des algorithmes d'apprentissage par renforcement est de trouver une stratégie π pour l'agent pour qu'il réalise au mieux son objectif. Mais alors une question se pose. Comment définir l'objectif.

Définir l'objectif

Dans un système d'apprentissage par renforcement, nous pouvons poser l'heuristique suivante :

> **Heuristique de l'objectif**
>
> Tout objectif peut être formulé comme le fait de ***maximiser la récompense cumulée*** au cours du temps.

Ainsi, il n'y a pas besoin de définir l'objectif de manière explicite. Il nous suffit simplement de définir une fonction de récompense r adéquate et une fonction de récompense cumulée G_t. Dans le cas général, nous utilisons $G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k$ avec $\gamma \in [0, 1]$ et la possibilité d'avoir soit $T = +\infty$ ou $\gamma = 1$ (mais pas les deux).

Le paramètre γ est appelé *discount factor*. Dans le cas où $\gamma < 1$, il permet d'accorder plus d'importance aux récompenses à court terme. Ce processus est naturel chez l'animal qui préfèrera en général une récompense à court terme.

Pour accomplir l'objectif ainsi défini par r et G_t, on défini une fonction de valeur. La fonction de valeur n'est autre que la récompense que l'agent peut espérer récupérer à long terme à partir d'un état $s \in \mathcal{S}$. Elle défini donc à quel point être dans cet état est bien pour l'agent. Nous pouvons utiliser les deux fonctions de valeur suivante :

- Fonction de **valeur d'état** : $\forall s \in \mathcal{S}, \quad v(s) = \mathbb{E}\{G_t \mid S_t = s\}$
- Fonction de **valeur d'action** : $\forall (s, a) \in \mathcal{S} \times \mathcal{A}, \quad q(s, a) = \mathbb{E}\{G_t \mid S_t = s, A_t = a\}$

Dans la suite de la partie théorique, nous utiliserons la fonction de valeur v. Cependant, tout ce qui sera présenté, fonctionne également avec q comme fonction de valeur.

En notant v_π la fonction de valeur associée à la stratégie π et π_* la stratégie optimale, nous pouvons poser le but de tout algorithme d'apprentissage par renforcement de façon mathématique.

$$\forall s \in \mathcal{S}, \ \pi_*(s) = arg \max_{\pi} v_\pi(s)$$

En d'autre termes, pour accomplir l'objectif il faut trouver la stratégie π qui maximise la récompense à long terme v_π.

Nous pouvons réécrire la fonction de valeur tel que suit :

$$\begin{aligned} v_\pi(s) &= \mathbb{E}\{G_t \mid S_t = s\} \\ &= \mathbb{E}_\pi\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+2} + ... \mid S_t = s\} \\ &= \mathbb{E}_\pi\{r_{t+1} + \gamma G_{t+1} \mid S_t = s\} \\ &= \mathbb{E}_\pi\{r_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t \sim \pi(S_t)\} \ \ (Bellman\ equation\ (i)) \end{aligned}$$

L'équation de Bellman nous donne une relation entre la valeur à un état s et les valeurs aux états suivants s'. Cette équation est la base de nombreuses méthodes d'approximation de la fonction de valeur v_π que nous verrons par la suite.

On peut ainsi déduire les **4 principales Equations de Bellman :**

$$
\begin{aligned}
&(i) \quad v_\pi(s) = \mathbb{E}\{r_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t \sim \pi(S_t)\} \\
&(ii) \quad \max_a v_*(s) = v_{\pi_*}(s) = \mathbb{E}\{r_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a\} \\
&(iii) \quad q_\pi(s,a) = \mathbb{E}\{r_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a\} \\
&(iv) \quad q_*(s,a) = q_{\pi_*}(s,a) = \mathbb{E}\{r_{t+1} + \gamma \max_{a'} q_\pi(S_{t+1}, a') \mid S_t = s, A_t = a\}
\end{aligned}
$$

Dans certains cas, nous connaissons un modèle parfait de l'environnement (en général le MDP suivi). Dans ce cas, on applique des méthodes de programmation dynamique pour construire une stratégie optimale.

Dans le cas contraire, le modèle de l'environnement n'est pas parfaitement connu (*model-free*). On utilisera des méthodes itératives pour construire de manière approchée la stratégie optimale. Ces méthodes sont par exemples les méthodes Monte-Carlo et TD. Dans la suite du rapport, nous parlerons essentiellement de ce cas.

2 Méthodes basées sur des échantillons

A partir de maintenant, nous considérons que le modèle de l'environnement n'est pas parfaitement connu. L'idée des méthodes *model-free* est d'utiliser l'expérience passée pour construire la politique π. Pour ceci, on va approximer la fonction de valeur optimale v_* en utilisant des exemples de séquences vécues de type état/action/récompense/état suivant, etc. Les trois méthodes de mise à jour de la fonction de valeur en utilisant des échantillons sont la méthode TD-learning, la méthode SARSA et la méthode de Q-learning.

2.1 TD-learning, SARSA et Q-learning

TD-learning

Pour mettre à jour la fonction de valeur v, l'algorithme TD utilise des séquences d'échantillons du type $S_t,\ A_t,\ R_t,\ S_{t+1}$.

On rappelle la première équation de Bellman :

$$\forall s \in \mathcal{S} \quad v_\pi(s) = \mathbb{E}_\pi\{r_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s\}$$

Les méthodes itératives issuent de la programmation dynamique utilisent cet équation pour mettre à jour la fonction de valeur :

$$\forall s \in \mathcal{S} \quad v_{k+1}(s) = \mathbb{E}_\pi\{r_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s\}$$

Toutefois, dans notre cas, nous nous basons simplement sur les échantillons aléatoires du type $S_t,\ A_t,\ R_t,\ S_{t+1}$ et pas sur un modèle exact. La règle de mise à jour de la fonction de valeur dans le cas de l'algorithme TD s'écrit :

$$v_{t+1}(S_t) = \underbrace{r_{t+1} + \gamma v_t(S_{t+1})}_{TD-Target}$$

Cependant cette méthode de mise à jour est assez brutale et engendre beaucoup de bruit. C'est pourquoi il faut mieux faire la moyenne pondérée entre la valeur actuelle et la valeur cible. Ce qui aboutit à une réécriture sous la forme :

$$\begin{aligned} v_{t+1}(S_t) &= (1-\alpha_t)v_t(S_t) + \alpha_t(r_{t+1} + \gamma v_t(S_{t+1})) \\ &= v_t(S_t) + \alpha_t \underbrace{[r_{t+1} + \gamma v_t(S_{t+1}) - v_t(S_t)]}_{TD-error} \end{aligned}$$

La méthode TD combine donc l'approximation itérative des méthodes de programmation dynamique avec l'utilisation d'échantillons propre aux méthodes *model-free*. Les méthodes SARSA et Q-learning se distinguent de la méthode TD-learning car elles utilisent la fonction de valeur q et donc sont basées sur l'équation de Bellman (iii). Ce qui distingue chacune d'entre elle n'est que la règle de mise à jour qui se présente de la façon suivante :

SARSA

La règle de mise à jour de q pour la méthode SARSA est la suivante :

$$q_{t+1}(s,a) = q_t(S_t, A_t) + \alpha_t(r_{t+1} + \gamma q_t(S_{t+1}, A_{t+1}) - q_t(S_t, A_t))$$

Q-learning

Quand à l'algorithme du Q-learning, sa règle de mise à jour est :

$$q_{t+1}(s,a) = q_t(S_t, A_t) + \alpha_t(r_{t+1} + \gamma \max_{a'} q_t(S_{t+1}, a') - q_t(S_t, A_t))$$

Ces méthodes n'utilisent que la séquence suivante pour mettre à jour leur fonction de valeur. Cependant, en fonction de l'environnement, il peut être astucieux que la mise à jour ne prenne pas en compte uniquement la dernière séquence d'action, récompense, état mais n séquences consécutives. C'est pourquoi la généralisation de chacune de ces méthodes est intéressante.

2.2 n-step learning

On rappelle la formulation théorique du gain total : $G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k$.

Les méthodes TD, SARSA et Q-learning peuvent être généralisées au cas où on ne considère plus uniquement la récompense suivante mais les n récompenses suivantes. Dans le cas du TD-learning, contrairement à la méthode dites *1-step*, la TD-target n'est plus $r_{t+1} + \gamma v(S_{t+1})$ mais la valeur $G_t^{(n)}$. $G_t^{(n)}$ est un estimateur de la valeur $v_\pi(S_t)$ et est égale à :

$$G_t^{(n)} = r_{t+1} + \gamma r_{t+2} + ... + \gamma^{n-1} r_{t+n} + v_{t+n-1}(S_{t+n})$$

La mise à jour de la fonction de valeur se fait donc de la façon suivante :

$$v(S_t) \leftarrow v(S_t) + \alpha(G_t^{(n)} - v(S_t))$$

Remarque : Dans presque tous les cas, $G_t^{(n)}$ est un estimateur biaisé de $v_\pi(S_t)$. L'unique cas pour lequel ce n'est pas vrai est quand $t + n \geq T$.

Remarque 2 : La méthode **Monte-Carlo** consiste à mettre à jour chaque état étant donné une séquence complète de récompenses observées de l'étape t à la fin de l'épisode. On peut considérer la méthode Monte-Carlo comme étant une méthode de type ∞-step TD-learning.

Remarque 3 : Plus n est grand et plus le biais est faible mais plus la variance est grande. La méthode 1-step TD-learning est propice aux MDPs tandis que dans le cas où l'environnments n'est pas markovien, il est parfois mieux d'aller vers un peu plus de Monte-Carlo (augmenter le nombre de steps).

De la même manière, il existe des généralisations des méthodes SARSA et Q-learning que nous ne détaillerons pas dans ce rapport.

3 Approximation de fonctions et Deep-Reinforcement Learning

Les méthodes vues précédemment sont applicables dans le cas où il y a un nombre n fini d'actions et un nombre m fini d'états possibles. Ainsi, dans ce cas particulier et à condition que les valeurs n et m ne soient pas trop grandes, nous pouvions représenter la fonction de valeur sous la forme d'une matrice. Dans le cas particulier où le nombre d'états est relativement élevé ou que l'espace $\mathcal{S}$ est continue, nous chercherons une approximation de la fonction de valeur. Dans certains cas une représentation polynomiale est suffisante mais dans notre cas nous utiliserons des réseaux de neurones artificielle qui sont de bons outils pour approcher des modèles non linéaire.

Dans ce cas, la fonction de valeur gardée en mémoire sera une fonction paramétrique que l'on notera v_θ (respectivement q_θ). Le but de l'approximation de fonction est de généraliser le cas des états que nous avons vu à des états que nous n'avons pas encore vu pour pouvoir prédire l'action à effectuer dans le cas d'états encore jamais explorés. Pour cela, on cherchera à optimiser les paramètres θ du modèle en utilisant les méthodes de mise à jour vu précédemment.

3.1 Mise à jour des paramètres de la fonction de valeur

Nous resterons dans le cas général pour lequel on considère un algorithme utilisant le gain cumulé après n étapes comme estimateur de la fonction de valeur.

$$G_t^{(n)} = r_{t+1} + \gamma r_{t+2} + ... + \gamma^{n-1} r_{t+n} + v_\theta(S_{t+n})$$

La règle de mise à jour pour l'algorithme TD-learning s'écrit désormais :

$$v_{\theta_{t+1}}(S_t) = v_{\theta_t}(S_t) + \alpha_t \underbrace{\left[G_t^{(n)} - v_{\theta_t}(S_t)\right]}_{\delta_t = TD-error} \nabla_\theta v_\theta(S_t)$$

En d'autre terme, cela revient à minimiser la fonction de coût $\mathcal{L}(\theta) = \frac{1}{2}\left[G_t^{(n)} - v_\theta(S_t)\right]^2$

De la même façon les mises à jour pour Q-learning et SARSA est :

$$q_{\theta_{t+1}}(S_t) = q_{\theta_t}(S_t) + \alpha_t \left[G_t^{(n)} - q_{\theta_t}(S_t, A_t)\right] \nabla_\theta q_\theta(S_t, A_t)$$

Avec $G_t^{(n)}$ les estimateur de q_π pour chacune des deux méthodes.

3.2 Deep Q-Learning

L'algorithme DQN a été introduit en 2015 par une équipe de DeepMind qui a, grâce à ce papier, a réalisé ce que l'on appelle du *end-to-end learning*. Le *end-to-end learning* stipule que l'agent ne reçoit comme information que l'image entière et non pas des caractéristiques prédéfinies par le programmeur. L'agent va donc apprendre à comprendre les caractéristiques importantes de l'image et choisir une action en fonction de l'image qu'il reçoit.

Algorithm 1 Deep Q Learning with Experience Replay

procedure DQN($n\_episodes, T, C, \epsilon, \gamma$)
  Initialize replay memory D to capacity N
  Initialize action-value function Q_θ with random weights θ
  Initialize target action-value function $\hat{Q}_{\theta^-}$ with weights $\theta^- = \theta$
  for $episode = 0,\ n\_episodes$ **do**
    Initialize state s_0
    for $t = 0,\ T$ **do**
      With probability ϵ select a random action a_t,
      otherwise select $a_t = arg\max_a(Q_\theta(s_t, a))$
      Execute action a_t and observe reward r_t and state s_{t+1}
      Store transition (s_t, a_t, r_t, s_{t+1}) in D
      Set

$$y_j = \begin{cases} r_j \ \ if\ episode\ terminates\ at\ step\ j+1 \\ r_j + \gamma \max_{a'} \hat{Q}_\theta(s_{t+1}, a') \ \ otherwise \end{cases}$$

      Perform a gradient descent step on $(y_j - Q_\theta(s_j, a_j))^2$
      Every C steps reset $\hat{Q} = Q$
      Set $s_{t+1} = s_t$

Pour bien comprendre cet algorithme, je l'ai réimplémenté grâce à la librairie TensorFlow et ai joué avec différents paramètres pour tester leur influence sur l'apprentissage.

4 Quelques astuces pour le DRL

Ces dernières années, DeepMind a mené et mène encore une sorte de course pour atteindre les meilleures performances en Deep Reinforcement Learning sur les jeux Atari. Le modèle DQN de 2015 a été de nombreuses fois repris et amélioré grâce à des astuces plus ou moins complexes qui ont aboutit à de meilleurs résultats pour la plupart des jeux Ataris. La conclusion, montré dans le papier Rainbow [2], est que la combinaison de plusieurs astuces permet d'obtenir des agents plus performants. J'ai donc implémenter en complément de l'algorithme DQN, certaines de ces astuces.

4.1 Improve Data Efficiency : Experience Replay

Une première astuce est ce qu'on appelle l'Experience Replay. L'experience replay est une astuce purement statistique. Elle permet d'utiliser de manière efficace les informations des échantillons tirés. En effet, le principe est de réutiliser des échantillons passés qui ont été sauvegarder sous la forme (s_t, a_t, r_t, s_{t+1}) pour apprendre avec les mêmes échantillons de

multiples fois. Cette pratique permet d'accélérer l'apprentissage car au lieu de mettre à jour le réseau pour chaque échantillon une fois, on va le met à jour plusieurs fois pour un même échantillon. De plus, nous prenons des lots d'échantillons de manière aléatoire dans le buffer. De cette manière, nous évitons d'utiliser des échantillons qui se sont produits à des instants rapprochés et peuvent donc être corrélés.

Prioritized Experience Replay

Une légère variante du simple Experience Replay expliqué ci-dessus est la version *prioritized*. Dans cette version nous ne tirons plus chaque échantillon avec la même probabilité mais avec une probabilité dépendant de son importance. Un échantillon est considéré important si son erreur $(G_t^{(n)} - v_\theta(S_t))$ est importante.

4.2 Imitation Learning

Lors de l'apprentissage de la réalisation d'une nouvelle tâche, il y a deux possibilités pour apprendre à bien la faire. Prenons l'exemple de l'apprentissage chez un enfant à faire le ménage. Dans le premier cas, c'est le cas étudié jusqu'ici, les parents de l'enfant lui demandent de faire de ménage. L'enfant va se débrouiller tout seul avec les ustensiles que ses parents lui ont donné. Quand les parents reviennet, ils émettent un avis sur ce qu'a fait l'enfant (l'avis peut être sous forme de récompense/punition). La fois suivante, si l'enfant a reçu un avis positif il va tenter de faire la même chose et si l'avis était mauvais il va tenter de s'améliorer. La seconde fois les parents re-émette un avis et ainsi de suite. On comprend vite que dans le cas où l'enfant ne sait pas du tout comment faire le ménage. Il peut être fastidieux pour lui d'apprendre tout seul si les parents ne lui expliquent pas ou ne montrent pas comment faire. C'est ce principe d'imitation d'un agent expert que l'on appelle *Imitation Learning*.

L'*imitation learning* nécessite donc d'avoir un ensemble de solutions dites experte. Ce sont des solutions qui ne sont pas forcément optimale mais qui peuvent être utilisées pour accélérer l'apprentissage de l'agent. En général, l'agent va donc reproduire le comportement de l'expert pendant quelques épisodes et s'améliorer très rapidement vers le niveau de ce dernier. Ensuite l'agent pourra commencer une phase d'apprentissage par renforcement pour acquérir un niveau supérieur à son superviseur. Par exemple, pour le jeu de Go, nous ne savons pas modéliser la stratégie optimale mais il est possible d'avoir accès à des parties de joueurs professionnels qui ont donc des stratégies très bonnes.

Plusieurs techniques existent pour imiter un expert. Pour ma part j'ai utilisé la méthode la plus simple à implémenter qui est le *Behavior Cloning*. Le *Behavior Cloning* est très simple. Nous devons disposer d'un ensemble de n couples de données observation/action (o, a) générées à partir d'un agent expert. L'idée est simplement de faire de l'apprentissage supervisé pour ajuster les paramètres de la fonction de valeur q_θ. Nous cherchons donc à minimiser la fonction de coût :

$$(Softmax\ Loss) \quad \mathcal{L}(\theta) = -\sum_i \mathbb{1}_{\{x=i\}}(a) \times \log([q_\theta(o)]_i)$$

L'*Inverse Reinforcement Learning (IRL)* et le *Generative Adversarial Imitation Learning (GAIL)* sont d'autres techniques pour faire de l'*imitation learning*.

5 Outils pour l'apprentissage par renforcement

On définit parfois l'intelligence d'un agent comme étant sa capacité à accomplir des objectifs dans un grand nombre d'environnements. Certaines personnes considèrent donc qu'une mesure de l'intelligence d'un agent pourrait avoir la forme suivante :

$$\Upsilon(\pi) = \sum_{\mu \in E} 2^{-K(\mu)} V_{\mu}^{\pi}$$

avec :

- π représente l'agent (sa politique de décision)
- V_{μ}^{π} représente la valeur obtenue par l'agent π dans l'environnement μ
- $K(\mu)$ représente la complexité de Kolmogorov de l'environnement μ

Cette formule est évidemment très réductrice de ce qu'est l'intelligence. Cependant si nous la considérons tel quel il en résulte que pour pouvoir avoir des agents intelligents ils doivent être compétents dans de nombreux environnements. Pour cela, ils doivent être entraînés dans des environnements divers. C'est pourquoi la communauté scientifique a mis en place des outils simples et open-source pour entraîner des agents dans des environnements de toutes sorte. Durant mon stage, j'ai principalement utilisé deux frameworks qui sont Gym et Stable Baselines.

(a) Gym

(b) Stable Baselines

FIGURE I.2 – Outils

5.1 Gym

Gym (fig. I.2a) est une framework fournissant un ensemble d'outils pour développer et comparer des algorithmes d'apprentissage par renforcement. Ce qui fait de cette framework un incontournable est qu'elle contient un grand nombre d'environnements de simulation. De plus elle permet, de manière simple, d'en créer de nouveau. Ces environnements sont standardisés et vont donc permettre à un même agent d'évoluer dans plusieurs environnements différents. Pour me familiariser avec la conception de ce type d'environnements, j'ai créé dès le début du stage une reproduction simple du jeu Flappy Bird sous le modèle Gym. J'ai choisi ce jeu car je savais qu'avec un état de l'environnement bien configuré, il est relativement simple d'apprendre une stratégie parfaite.

5.2 Stable Baselines

La seconde framework avec laquelle j'ai beaucoup travaillé est Stable Baselines (fig. I.2b). Stable Baselines fournit une implémentation pour les principaux modèles d'apprentissage par renforcement. Elle est étroitement liée à Gym car pour l'entrainement, les agents doivent évoluer dans des environnements Gym ou du moins des environnements ayant les mêmes fonctions caractéristique. Il est à noter, que Stable Baselines implémente ses modèles grâce à Tensorflow et donc fournit des outils propres à cette librairie tel que la possibilité de visualiser l'avancement de l'apprentissage grâce à l'outil TensorBoard. Pour plus d'information sur Tensorflow, voir mon rapport de stage Technicien.

Part II

Arcadeep, la borne d'arcade intelligente

Le projet d'Arcadeep a pour but de fournir une expérience nouvelle aux joueurs. L'idée initiale pour y aboutir est de créer une borne de jeux d'arcade qui puisse prendre vie de différentes façons. A mon arrivée dans l'entreprise, la borne avait déjà été montée. La borne initiale est contitué d'un écran, d'enceintes et de la possibilité de jouer à deux joueurs (un côté rouge et un jaune). Chaque joueur possède un joystick et six boutons. Le côté jaune est réservé à l'IA qui, grâce à des servomoteurs, peut faire bouger de façon automatique le joystick jaune et certains boutons.

Le travail autour de la borne d'arcade s'est fait petit à petit. Le but était qu'au fur et à mesure des semaines, la borne d'arcade acquiert des nouvelles fonctionnalités. Pour cela, j'ai principalement travaillé autour de trois objectifs. Le premier étant d'implémenter une interface graphique pour permettre à l'utilisateur de naviguer entre les différentes possibilités qu'offre la borne Arcadeep et d'entrainer les premiers modèles d'IA. Le second était de faire en sorte qu'Arcadeep puisse communiquer avec le joueur. Le dernière objectif et celui sur lequel j'ai passé le plus de temps a été de mieux comprendre la phase d'entraînement des modèles d'IA.

1 Les fonctionnalités de base des bornes d'arcade

Avant de faire d'Arcadeep une borne de jeu d'arcade de nouvelle génération, il a fallu en faire une borne de jeux d'arcade "basique". Par là, j'entend une borne de jeux auxquels on peut jouer à différents jeux avec différents niveaux de difficultés. En plus de celà, Pierre et Matthieu voulaient qu'on ait la possibilité de choisir entre trois modes de jeux qui sont : jouer tout seul, l'IA joue toute seule et nous jouons en confrontation avec l'IA. Avec ces modes de jeux, nous pouvions déjà avoir une idée de l'intérêt de jouer contre une IA ou de regarder une IA jouer comme un fond d'écran.

1.1 Les jeux

Au démarrage de la borne, le joueur peut choisir parmis un ensemble de jeux. Actuellement, il y a cinq jeux auxquels on peut jouer qui sont : *Casse-Brique*, *Pacman*, *Flappy Bird*, *Tetris* et *Pong*.

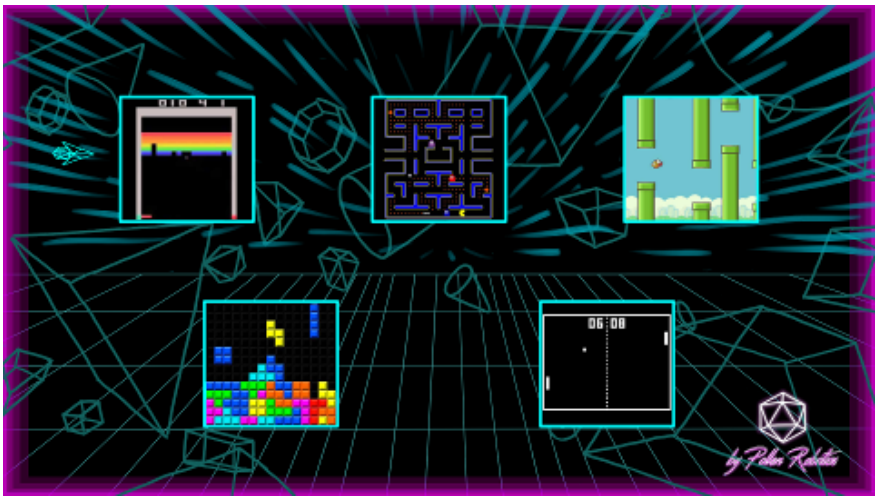


FIGURE II.1 – Choix d'un des cinq jeux

Parmi ces jeux, il y en a trois que nous avons récupéré des environnements Gym. C'est jeux sont Pong, Casse-Brique et Pacman. Quand à Tetris et Flappy Bird, nous les avons développé nous même. Le jeu Tetris avait été implémenté par Pierre avant mon arrivée. Pour ma part, j'ai implémenté le jeu Flappy Bird. J'ai créé une version minimale de Flappy Bird dès le début de mon stage dans le but de me familiariser avec la création d'environnements Gym, puis je l'ai repris pour en faire des améliorations et des versions différentes tel que le mode Duo dans lequel le joueur humain joue en collaboration avec l'IA (voir code 2). J'ai créé ce mode car il nous a finalement paru que le fait de jouer avec la borne pouvait être aussi stimulant que de jouer contre elle.

FIGURE II.2 – Flappy Bird en mode Duo

1.2 Les différents modes

Comme expliqué auparavant, nous avons fait en sorte que l'utilisateur puisse choisir entre trois modes de jeux :

- le mode Humain Solo
- le mode IA Solo
- le mode Versus

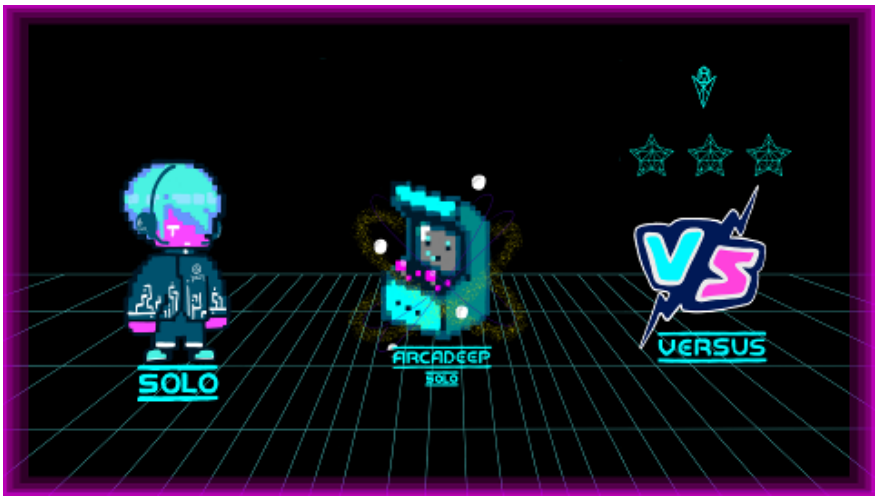


FIGURE II.3 – Choix du mode de jeu

Dans le cas du mode Versus, il est en plus possible de choisir le niveau de l'IA contre laquelle nous jouons. Les IA (sauf celle de Tetris) présentes au départ ont été entraînées avec l'algorithme A2C implémenté dans la librairie Stable Baselines. Les différents niveau représentent simplement un avancement de l'apprentissage.

Le cas de Tetris est plus compliqué. C'est un jeu pour lequel les algorithmes d'apprentissage par renforcements n'aboutissent pas facilement à de bons résultats. En effet, le fait de créer une ligne nécessite d'avoir placé plusieurs pièces de façon intelligente. De plus la mécanique du jeu n'est pas simple à comprendre pour un algorithme. Contrairement aux cinq autres jeux, la conséquence de ses actions peut être très lointaine et le lien entre une action et le résultat n'est pas toujours évident. J'ai passé quelques temps à étudier ce cas et ai testé de nombreuses techniques pour entraîner une IA à jouer à Tetris avec le moins d'informations possible.

1.3 Autres fonctionnalités

A l'heure actuelle, la borne joue également la musique et certains sons des jeux, nécessaires à l'immersion du joueur.

Comme le but est que la borne soit originale et qu'elle prenne vie, plusieurs choses ont ensuite été envisagées. Nous nous sommes finalement penché sur deux aspects principaux : le 'visage' d'Arcadeep qui consiste en un panneau d'affichage de LED RGB et le dialogue avec Arcadeep. Pour ma part j'ai travaillé uniquement sur le dialogue.

2 La voix d'Arcadeep (voir code 3)

Du point de vue du développement logiciel, en plus du jeu FlappyBird et de quelques modules de l'environnement d'Arcadeep, j'ai travaillé sur la voix de la borne. Le but était de faire parler Arcadeep pour qu'elle partage ses émotions et qu'elle puisse communiquer de manière sérieuse ou non avec le joueur. Ce n'est pas une chose facile de communiquer avec des humains de manière naturelle. De plus, j'ai tenté de faire en sorte que la voix d'Arcadeep soit indépendante de l'application décrite précédemment. Ainsi j'ai développé, grâce aux conseils de mon maître de stage, une application Flask pour la gestion de la voix d'arcadeep.

2.1 Le dialogue avec DialogFlow

Une manière efficace de créer un agent conversationnel est d'utiliser la librairie DialogFlow. DialogFlow est une API Google fournissant un ensemble d'outils nécessaires à la création de chatbots. L'outil est relativement facile à prendre en main et permet de prédéfinir un ensemble d'interactions possibles avec l'utilisateur.

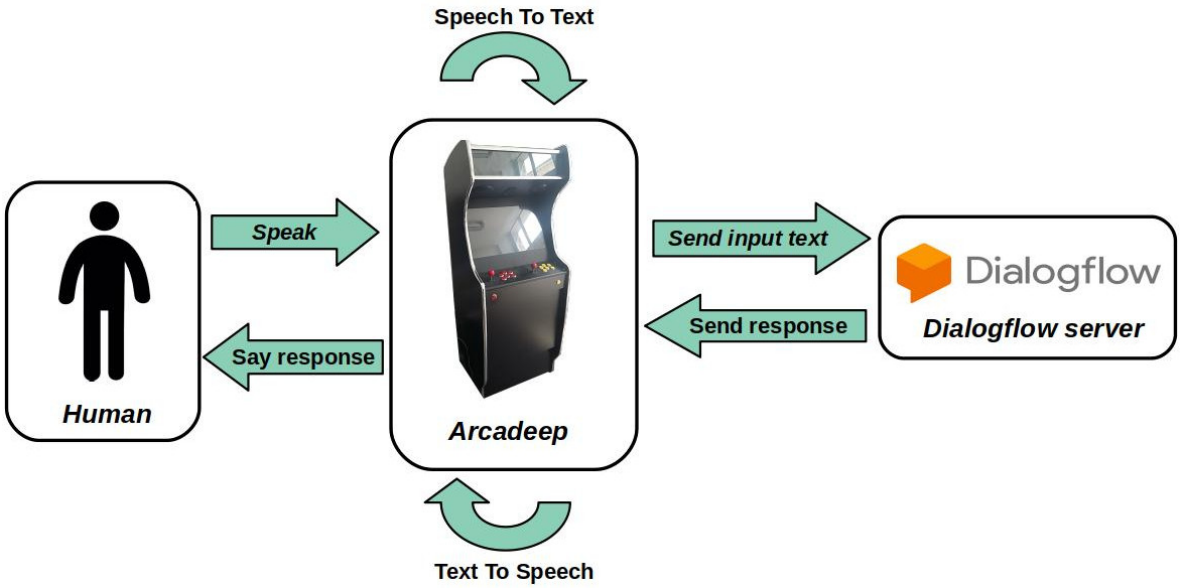


FIGURE II.4 – Scéma du mécanisme de discussion avec DialogFlow

3 L'apprentissage par renforcement sur les jeux Atari

Un des objectifs principaux du projet était d'appliquer des méthodes d'apprentissage par renforcement pour faire apprendre à Arcadeep à jouer à des jeux. En plus de les appliquer il a fallu que je comprenne l'utilité de chaque paramètre pour optimiser l'apprentissage et aboutir à des résultats intéressant. Dans le but de bien comprendre le modèle DQN j'ai décidé d'en refaire une implémentation avec la librairie *TensorFlow* (voir code 1) pour pouvoir avoir le controler sur tous les paramètres utilisés. J'ai donc dans un premier temps fait de nombreux tests sur les paramètres du modèle et comparer les différents résultats. Cependant, j'ai vite compris que les méthodes d'apprentissage n'était pas très stable et parfois, l'apprentissage était compliquer à analyser. C'est pourquoi j'ai par la suite développé des outils de visualisation pour tenter de mieux comprendre les résultats.

3.1 Expérimentations sur Flappy Bird avec *features*

J'ai tout d'abord réalisé de nombreux tests sur le jeu Flappy Bird avec le modèle DQN. Dans les premiers tests, j'ai utilisé une version de l'environnement pour laquelle l'observation de l'agent est le scalaire $\Delta y = y_{centre\_flappy} - y_{centre\_next\_plateforme}$ (voir fig.II.5). Ainsi il est aisé d'estimer une stratégie raisonable pour Flappy qui est d'aller vers le bas si $\Delta y > 0$ et de voler si $\Delta y < 0$. Cette stratégie (que l'on appellera *Bot*) a d'ailleurs été utilisé comme point de comparaison et pour expérimenter le behavior cloning.

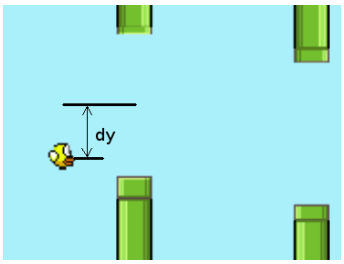


FIGURE II.5 – Exemple d'observation Δy de l'agent

Un exemple de test que j'ai réalisé est de comparer les résultats d'apprentissage lorsque l'on fait varié le taux d'exploration. J'ai d'abord fixé les paramètres du modèle tel que suit : $\alpha = 0.00025$, $\gamma = 0.9$, $batch\_size = 32$, $memory\_size = 10000$ et $C = 200$. Ensuite, j'ai joué sur les paramètres ϵ_{debut} et ϵ_{fin} puis tenter de comparer et expliquer les résultats. Les taux d'exploration utilisés sont tel que suit :

- $\epsilon_{debut} = 1.0$ et $\epsilon_{fin} = 0.1$
- $\epsilon_{debut} = 1.0$ et $\epsilon_{fin} = 0.8$
- $\epsilon_{debut} = 0.1$ et $\epsilon_{fin} = 0.01$

Pour ces trois choix de paramètre ϵ, les résultats sont représentés sur la figure ci-après. A titre indicatif j'ai rajouté sur la courbe d'évolution de la récompense, la récompense moyenne du *Bot* en rouge et la récompense moyenne des meilleurs parties des employers de Pollen Robotics en noir.

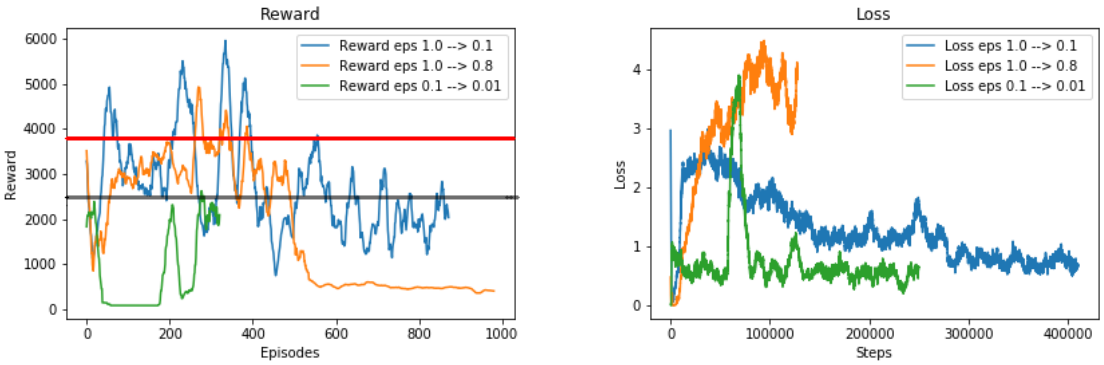


FIGURE II.6 – Comparaison de la méthode en fonction du taux d'exploration

Interprétations : Il n'est pas simple d'expliquer clairement les résultats obtenus lors des trois apprentissages. Globalement, on voit tout de même que la courbe d'exploration correspondant à une décroissance linéaire de 100% à 10% d'exploration (courbe bleue) semble aboutir

à de meilleurs résultats (reward plus élevée et fonction de coût décroissante). Ce choix de paramètre d'exploration est d'ailleurs celui du papier DQN original [1].

On constate également que dans une première partie d'apprentissage la courbe orange fournit des résultats globalement meilleurs que les joueurs humains. Cependant après un peu plus de 500 épisodes, ils se dégradent nettement. Il n'est pas simple d'expliquer ce changement. Pour expliquer cela, il faut se pencher plus en détail sur ce qu'a appris le réseau de neurones q_θ dans ce cas là. Pierre m'as vivement conseillé d'afficher la fonction de valeur. L'avantage de l'environnement Flappy Bird que j'utilise ici est que l'observation n'est qu'un scalaire. Ainsi, c'est un des rares cas où une représentation direct de la fonction q_θ est possible. J'ai donc affiché l'évolution de la fonction q_θ dans ce cas (voir fig II.7).

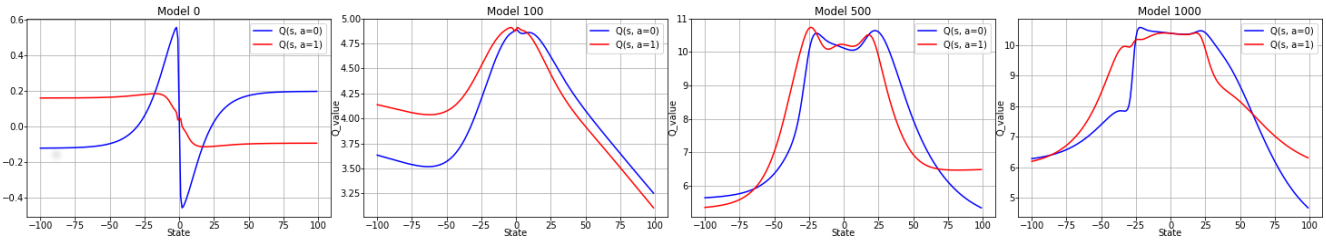


FIGURE II.7 – Evolution de la fonction q_θ dans le cas ϵ varie entre $100\% \rightarrow 80\%$

Interprétations : On peut remarquer que la fonction de valeur q_θ a, dès l'épisode 100, une forme intéressante. Plus on est proche de 0 et plus la valeur est élevée, ce qui est cohérent car plus Flappy est proche du centre de la plateforme et plus il a des chances de survivre longtemps et donc avoir un gain (et fonction de valeur) élevée. On remarque également que les courbes $q_\theta(s, \underbrace{0}_{Ne\ rien\ faire})$ et $q_\theta(s, \underbrace{1}_{Voler})$ sont légèrement décalées. C'est ce décalage qui permet d'obtenir une stratégie déterministe très bonne. En effet, du fait que $\forall s < 0, \quad q_\theta(s,1) > q_\theta(s,0)$ et $\forall s > 0, \quad q_\theta(s,1) < q_\theta(s,0)$, la stratégie déterministe engendrée est celle du *Bot*.

Après 1000 épisodes on distingue un applatissement surprenant de la fonction de valeur autour de 0. La largeur de l'applatissement est environ égale à :

$$min\_size\_ouv - \frac{epaisseur\_bird}{2} = 60 - 7.5 \approx 50px$$

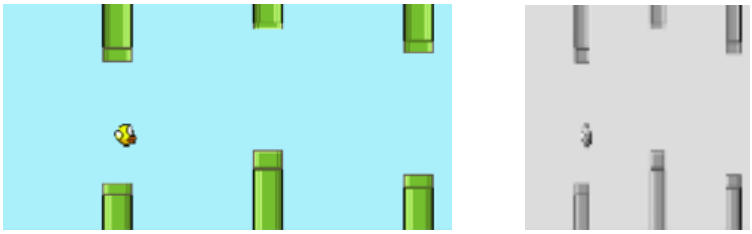
Cette largeur est environ la largeur du passage en les deux tubes. En d'autre termes, tant que Flappy se situe entre les deux tubes, l'espérance du gain total reste le même. Le problème c'est qu'en même temps que cet applatissement, les résultats de Flappy ont chuté. Ce résultat est probablement dû au fait que la part d'exploration est trop élevée.

3.2 Expérimentations sur FlappyBird CNN

Dans les environnements Flappy Bird que j'ai développé, l'un d'eux a pour but de faire du *end-to-end learning*. Dans le cas où un agent apprend à jouer à des jeux vidéos, le *end-to-end learning* désigne le fait que l'agent doit choisir l'action en ne connaissant que l'image d'entrée du jeu et non des caractéristiques prédéfinies par le programmeur comme le Δy de la section d'avant. Cette tâche est compliquée car elle nécessite que l'IA apprenne à détecter

et caractériser les éléments importants de l'image puis qu'à partir de ces éléments elle apprenne l'action à réaliser.

J'ai donc tenter de faire apprendre à un agent à jouer à Flappy Bird en ne prenant comme entrée que l'image. Pour simplifier la tâche j'ai décidé de simplifier l'image d'entrée en la transformant en noir et blanc (une seule chaine) et en la redimensionnant en $84px \times 84px$.



(a) Vue de l'humain (b) Vue de l'IA

FIGURE II.8 – Le jeu vu par un humain et par l'IA

Le réseau de neurones utilisé dans ce cas est un réseau de neurones convolutifs dont l'architecture est semblable au papier DQN original. Avec la même procédure que dans l'exemple d'avant j'ai donc tenté de comparer différents paramétrages.

Comparaison de deux entrainement

J'ai tout d'abord confronté 2 types de paramétrages totalements différents.

- **Test 1 :** $\alpha = 0.00025$, $\gamma = 0.975$, $batch\_size = 32$, $\epsilon_{debut} = 1.0$, $\epsilon_{fin} = 0.1$, exp_replay : Simple Buffer , $memory\_size = 10000$ et $C = 200$.
- **Test 2 :** $\alpha = 0.001$, $\gamma = 0.975$, $batch\_size = 32$, $\epsilon_{debut} = 1.0$, $\epsilon_{fin} = 0.1$, exp_replay : Prioritized, $memory\_size = 50000$ et $C = 10000$.

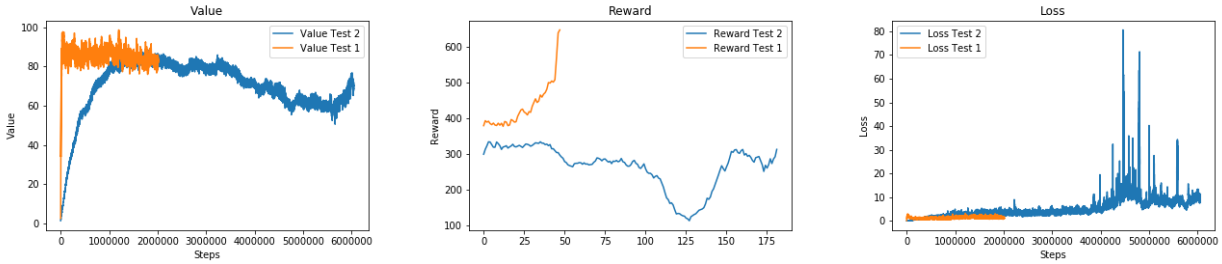


FIGURE II.9 – Comparaison de la méthode en fonction du taux d'exploration

Interprétations : On peut comparé la vitesse de convergence vers une bonne fonction de valeur. Celle du *Test 2* est très rapide en comparaison au *Test 1*. De plus, la reward du *Test 1* augmente de manière significative vers de bons résultats.

Dans ce cas, la fonction de valeur q_θ ne peut pas être représentée aussi simplement que dans l'exemple de Flappy avec Δy.

J'ai donc dû utiliser d'autres outils plus complexe pour tenter de comprendre l'évolution de q_θ.

Analyse en profondeur du premier test

Les résultats ici sont les résultats pour le premier test. Le score du meilleur modèle est plutôt bon (en moyenne la moitié des meilleurs score humain).

Grad-CAM

L'une des techniques est d'utiliser ce qu'on appelle "Grad-CAM" [3]. Cette méthode, permet de visualiser, pour chaque couche, les zones influençant majoritairement les résultats de sortie du réseau de neurones. Plus la zone est rouge et plus la zone est considérée avoir un effet sur la sortie.

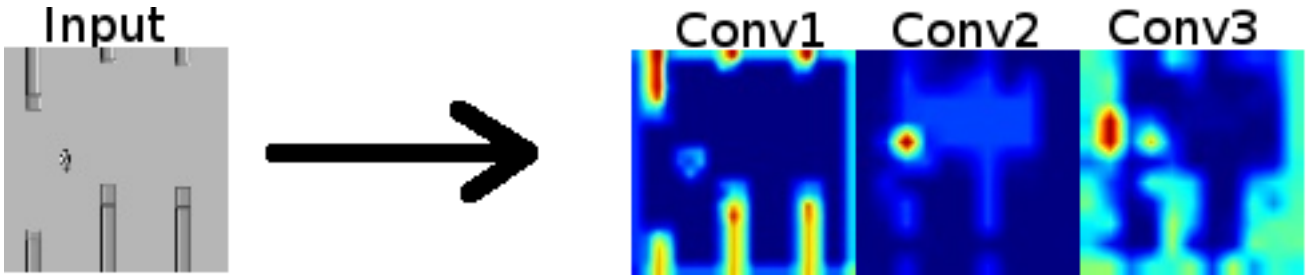


FIGURE II.10 – Gradient-weighted Class Activation Mapping for Action 0

On se rend compte que la première couche du réseau convolutif v_θ porte attention principalement aux plateformes et probablement un peu a Flappy également. La deuxième couche semble se focaliser plus sur Flappy et sur les espaces libres ou plutôt "sans danger". Dans la figure II.10, cela correspond à la zone de l'image *conv2* qui est de couleur bleue la moins foncée. Quand à la dernière couche de convolution, il est plus difficile de l'analyser.

Occlusion Sensitivity

Une seconde technique est l'occlusion sensitive. Cette technique permet de visualiser les parties de l'image qui affectent le réseau de neurones pour chaque action. Par exemple dans la figure II.11, on voit que pour le fait d'aller vers le bas, ce qui ce trouve dans la zone devant et en dessous de Flappy est intéressante. Dans l'autre cas (action d'aller vers le haut), c'est la zone plus au dessus qui est importante.

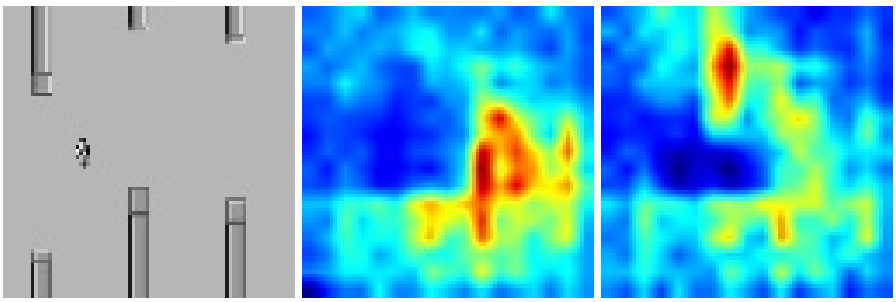


FIGURE II.11 – De gauche à droite : entrée du réseau, résultat pour l'action 0 (aller en bas), résultat pour l'action 1 (aller en haut)

Conclusion des tests

Certains affichages sont intéressant et permettent de mieux comprendre le processus d'apprentissage par renforcement sur le jeux flappy bird. Cependant tous les résultats précédent sont en général basés sur un seul apprentissage et donc doivent être considérés avec grandes précautions. En effet, les apprentissages se font uniquement sur des échantillons de trajectoires. Ainsi, les courbes de récompenses portent également une grande part d'aléatoire. Une meilleure façon de procéder pour pouvoir mieux comparer les résultats d'apprentissage aurait été de lancer de nombreux entraînement avec les mêmes paramètres et d'en faire une moyenne. Etant donné que le temps et les ressources nécessaires pour entraîner un un modèle, je n'ai pas pu mettre en oeuvre cela avant la fin de mon stage.

Conclusion

Pour conclure, ce stage m'a permis de travailler sur un projet très intéressant, nouveau et qui nécessite de mêler de nombreuses technologies. J'ai ainsi pu découvrir de nombreuses technologies parfois très utilisées par les entreprises du numérique tel que la création de chatbots avec DialogFlow qui dans un autre contexte que celui de mon stage est souvent utilisé pour du service client en ligne. Pour réalisé ce stage, j'ai donc tenté d'utiliser au mieux mes compétences pour faire avancer la création de cette entité intelligente qu'est Arcadeep.

De plus, la grosse partie recherche autour de l'apprentissage par renforcement m'a permis d'avoir une meilleure vue d'ensemble du domaine et de mieux appréhender ses limites et ses difficultés. Pour cette aspect, je penses qu'un point important du stage a été de voir que cette thématique est encore précoce et que, bien qu'elle peut aboutir à de bons résultats, elle nécessite une grosse puissance de calcul et un long entraînement instable qui peut être plus difficile à utiliser tel quel.

Enfin, j'ai eu la chance de travailler avec une équipe formidable qui m'a permis de découvrir véritablement ce qu'est de travailler au sein d'une start-up en robotique.

Bibliographie

[1] *Human-level control through deep reinforcement learning*, Mnih et al, https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf, Feb 2015.

[2] *Rainbow : Combining Improvements in Deep Reinforcement Learning*, Hessel et al, https://arxiv.org/pdf/1710.02298.pdf, 2017.

[3] *Grad-CAM :Visual Explanations from Deep Networks via Gradient-based Localization*, R. Selvaraju et al, https://arxiv.org/pdf/1610.02391.pdf, Oct 2016.

[4] *Reinforcement Learning : An Introduction*, Richard S. Sutton and Andrew G. Barto, 2018

Codes

- **Code 1** - Implementation DQN et Behavior Cloning - *Google Cloud*
- **Code 2** - Environnement Flappy Bird - *Github*
  → https://github.com/davHub/flappy-bird-env
- **Code 3** - Voix d'Arcadeep - *Github*
  → https://github.com/davHub/arcade-voice