IA des jeux et apprentissage par renforcement dans un environnement multijoueur

Léo Lacoste, Olivier Dupont, Antoine Courtens, Quentin Fabriès

version du 07/05/2021

Encadré par Monsieur Tristan Vaccon

Sommaire

1 Introduction

L'Intelligence artificielle (IA) est en pleine expansion depuis maintenant plusieurs années. C'est un vaste domaine complexe qui suscite une certaine curiosité sur son fonctionnement. Notre projet consiste à utiliser l'une des nombreuses branches de l'IA à savoir le Machine Learning. On trouvera égalementle terme d'apprentissage automatique (en français). Ici, nous allons nous intéresser à une méthode d'apprentissage du Machine Learning qui est l'apprentissage par renforcement. Pour comprendre cela, l'article suivant nous a permis d'y voir un peu plus clair. Notamment sur le fonctionnement de cette méthode fondée sur les erreurs précédentes de l'IA : https://www.lebigdata.fr/reinforcement-learning-definition

Pour notre projet nous avons dans un premier temps comme buts de nous familiariser avec Python et l'environnement Gym Retro puis de comprendre le principe des algorithmes DQN. Par la suite nous souhaiterions entraîner une IA contre elle-même, dans un environnement multijoueur, sur le célèbre jeu Pong sorti par l'entreprise Atari en 1972. L'objectif ultime est de réussir à implémenter l'apprentissage d'une IA dans un environnement multijoueur plus complexe tel que Street Fighter 2 ou Mortal Kombat.

2 Organisation

2.1 Gestionnaire de versions

Nous n'avons pas encore commencé à implémenter nos algorithmes mais nous utiliserons Git pour gérer les différentes versions de notre projet. Cela nous permettra de travailler sur différentes tâches tout en gardant une version fonctionnelle et livrable.

2.2 Outils collaboratifs

De plus, nous utiliserons Trello qui est un outil de collaboration permettant d'affecter les différentes fonctionnalités à implémenter, de gérer les deadlines, etc. en fonctions des différents membres de l'équipe. Nous avons également mis en place un serveur Discord pour tout échange de documentations dans le but de communiquer facilement entre les membres de l'équipe.

2.3 Répartition du travail

Pour la première partie de ce projet l'objectif était de commencer à se former sur la notion d'apprentissage en IA et à apprendre les différents algorithmes d'apprentissage. De plus nous devions débuter sur l'environnement Gym Retro qui gravite autour du projet. Pour cela nous nous sommes répartis les tâches. Nous sommes quatre au sein du groupe. Deux d'entre nous ce sont occupés de l'environnement Gym Retro et de l'utilisation des ROMs (Quentin Fabriès et Antoine Courtens) et les deux autres se sont formés sur les algorithmes et les techniques d'apprentissage (DQN, SARSA, réseaux de neurones, etc) (Olivier Dupont et Léo Lacoste) Nous avons ensuite mis en commun notre travail lors d'une réunion pour que chacun puisse s'imprégner des travaux des autres membres de l'équipe.

Lors de la deuxième partie de projet nous avons tous les quatre fait un grand nombre de tentatives. Les bibliothèques Python TensorFlow et Keras ont été utilisées pour la partie intélligence artificielle. Ainsi de nombreuses recherches ont été menées concernant l'implémentation de réseau de neurones avec Gym/Gym Retro. Nous avons mis en commun nos différentes expérimentations afin de produire les codes de ce projet.

3 Python

Nous avons commencé nos recherches par la compréhension des outils que nous allions utiliser. Nous nous sommes donc intéressés au langage Python qui nous permettra d'interagir avec Gym Retro. Ce langage est très populaire aujourd'hui surtout dans le domaine des mathématiques et de l'intelligence artificielle. Ici nous avons donc commencé par installer un environnement capable d'accueillir ce langage sous Windows 10.

Attention : Nous allons utiliser par la suite une bibliothèque Python appelée Gym Retro, celle-ci fonctionne, au moment où ces lignes sont écrites (janvier 2021) sous les environnements suivants :

— Windows 7, 8, 10
— MacOS 10.13 (High Sierra), 10.14 (Mojave)
— Linux (manylinux1)
— Python 3.6, 3.7, 3.8
Source : Documentation Gym Retro

Dans notre projet nous allons utiliser un environnement utilisant le lan-

gage Python. Ce langage est très populaire aujourd'hui surtout dans le domaine des mathématiques et de l'intelligence artificielle. Ici nous avons donc commencé par installer un environnement capable d'accueillir ce langage sous Windows 10. Pour l'installation sur Windows plusieurs solutions existent. En effet on peut choisir de télécharger une version locale ou portable de Python. Nous avons choisi Python en version locale.

Pour information, la version portable inclut l'IDE Spyder (environnement de développement) mais également tous les paquets nécessaires avec de nombreuses bibliothèques. Cette version portable de Python est disponible sur https://winpython.github.io/. L'environnement Python, Anaconda, pourra également être utilisé. Pour l'installation de Python en local, nous nous sommes rendus sur le site officiel et avons téléchargé la version 3.8.0 pour Windows.

Attention : Lors de l'installation de Python, nous avons veillé à bien cocher la case correspondante à l'ajout d'une variable d'environnement. Celle si nous sera utile par la suite pour taper des commandes relatives à Python dans un terminal.

Lors du second semestre nous avons changé de cap concernant l'environnement Python. En effet nous avons découvert Google Colab et nous avons développé nos codes dessus.

Colab est un outil en ligne développé par Google permettant d'exécuter du code Python. Il est basé sur les notebooks Jupiter bien connus des développeurs travaillant avec ce langage. Il présente l'intérêt pour tous les membres de l'équipe de pouvoir travailler avec la même configuration de Python alors que chacun possède différents environnements. En effet ceux-ci varient d'un ordinateur à un autre (Windows, Linux, composants...). De plus Colab donne accès à un GPU qui peut être utile pour certaines tâches, si l'on n'en possède pas sur sa propre machine.

4 Environnement Gym Retro

Nous allons utiliser Gym Retro. C'est une bibliothèque Python qui permet de faire de l'apprentissage supervisé via de l'intelligence artificielle sur de vieux jeux d'arcades. Elle intègre plusieurs émulateurs de console ainsi que la prise en charge de nombreux jeux connus. Pour notre projet nous utiliserons

l'émulateur pour la console Atari 2600 avec dans un premier tempsle célèbre jeu Pong. Pour faire revivre cet ancien jeu sur nos ordinateurs actuels, il faut insérer une Rom (fichier d'un jeu) dans notre émulateur.

FIGURE 1 – Console Atari 2600

4.1 Installation

Pour installer cette bibliothèque nous allons utiliser le gestionnaire de paquet recommandé pour Python s'appelant Pip. Pip est fourni par défaut dans Python depuis les versions 2.7.9 et 3.4.

1. Nous ouvrons un terminal et lançons la commande **py -3.8 -m pip install gym-retro** **Remarque :** Le -3.8 correspond à la version de

   Python à laquelle va être installer le paquet. Ainsi si plusieurs versions de Python sont installées sur un ordinateur, nous pouvons ainsi choisir la version pour l'installation de Gym Retro comme cela.
2. Après installation nous vérifions que le paquet gym-retro a bien été installé avec la commande **py -3.8 -m pip list** qui fournit la liste de tous les paquets présents selon la version de Python spécifiée.
3. A présent nous démarrons un jeu pour essayer l'émulateur de Gym Retro. En effet plusieurs jeux sont inclus et notamment Airstriker. Pour cela exécutons la commande **py -3.8 -m retro.examples.interactive –game Airstriker-Genesis**. Une fenêtre s'ouvre, nous pouvons à présent jouer au jeu avec les touches du clavier "flèche gauche" et "flèche droite" pour se déplacer et "x" pour tirer.

Remarque : Si pour une raison quelconque nous souhaitions désinstaller

ce paquet, nous aurions pu taper la commande suivante : **py -3.8 -m pip uninstall gym-retro**

Remarque : La librairie Python Gym est similaire à Gym Retro et s'utilise de la même manière. Ainsi sous Gym un environnment comprenant de nombreux jeux Atari peut être installé via cette commande : **py -3.8 -m pip install gym[atari]**. Il suffira par la suite de choisir un jeu parmi la liste proposée sur le site officiel : https://gym.openai.com/envs/#atari

4.2 Ajout et lancement de notre propre Rom Pong

Pour le besoin de notre projet, nous avons eu besoin d'importer un jeu bien spécifique : Pong. Malgré avoir lu et suivi la documentation de Gym Retro mentionnant les Roms de jeux avec les différentes extensions prises en charges, nous avons eu beaucoup de mal à en trouver une qui fonctionne. Nous avons finalement téléchargé une Rom Pong, tirée du jeu Video Olympic d'Atari. La Rom classique du Pong, celle en Noir et Blanc que nous connaissons, n'a pas été trouvée malgré de nombreuses recherches sur Internet.

Recherches sur le jeu Pong

La plateforme gym-retro supporte un certain nombre de jeux à travers différents émulateurs de consoles allant d'Atari 2600 à la Mega Drive en passant par la NES. Les jeux sont alors sous forme de ROMs que nous devons importer dans Gym-retro. Une problématique un peu surprenante peut apparaître : il faut trouver une copie cartouche de notre jeu. Il est alors intéressant de noter que le jeu Pong tel quel n'a pas eu de sortie cartouche sur l'Atari 2600. La console Atari 2600 est une console de jeu à cartouche sortie en 1977 aux USA. Le jeu Pong lui est sorti en Arcade et sur console de jeux mais sur des consoles de jeux dédiées à ce jeu. Pour jouer à Pong sur Atari 2600 on peut le trouver sur la compilation de jeux intitulée "Video Olympics" sortie au lancement de la console. Cette collection de jeux inclut donc une version de Pong. La version de Pong que l'on pourra alors utiliser est celle issue de cette compilation de jeux "Video Olympics".

Pour ajouter Pong à l'environnement Gym Retro, nous nous sommes rendus sur retrostic.com puis nous avons téléchargé le fichier (la Rom) dans un nouveau dossier. Celui-ci comportera toutes les Rom de jeux que nous ajouterons à Gym Retro par la suite. Nous avons ouvert un terminal, nous

FIGURE 2 – Couverture de la cartouche "Video Olympic"

nous sommes placés dans notre dossier contenant nos Roms et avons exécuté la commande suivante (sans oublier le point à la fin) pour ajouter notre Pong : **py -3.8 -m retro.import .**. Pour finir nous l'avons lancé pour nous assurer qu'il était bien fonctionnel avec la commande **py -3.8 -m retro.examples.interactive –game Pong-Atari2600**.

5 L'apprentissage par renforcement

5.1 Introduction

Nous avons effectué plusieurs recherches sur les algorithmes d'apprentissage par renforcement. Nous avons alors commencé par étudier notre sujet afin de comprendre les différentes problématiques qui vont se poser. Pour mieux comprendre ces problématiques notre recherche s'est orientée sur les différents travaux qui ont été faits dans ce domaine.

Le travail de Volodymyr Mnih et d'autres chercheurs a permis de montrer que l'on pouvait utiliser l'apprentissage par renforcement pour jouer à des jeux Atari. Les différentes recherches qui découlent de ce travail nous per-

mettent ainsi de mieux appréhender notre sujet (page Wikipédia de l'apprentissage par renforcement : https://fr.wikipedia.org/wiki/Apprentissage_par_renforcement)

L'algorithme Deep Q-Networks (DQN), un algorithme d'apprentissage par renforcement, est au cœur de la solution proposée afin de laisser jouer une IA à la place d'un humain à des jeux Atari. Avec la familiarisation de l'environnement GymRétro qui est le résultat de ces différents travaux, la compréhension de cet algorithme et de ces variantes est un élément essentiel pour nous permettre de faire combattre une IA face à elle-même. Ainsi, avant de proposer une IA qui se confronterait à elle-même, on pourrait d'abord essayer de programmer notre IA pour que celle-ci puisse battre celle déjà implémentée dans les différents jeux Atari et donc dans celui qui nous intéressera plus particulièrement c'est à dire le jeu Pong.

5.2 DQN : un algorithme capable d'apprendre à jouer aux jeux vidéo

Un article du journal : "LeMonde" relate alors la découverte de Mnih et d'autres chercheurs. Cette équipe a donc créé un algorithme capable d'apprendre à jouer par lui-même à des jeux dits "classiques". (Source : Le Monde)

L'équipe de chercheurs explique alors la différence fondamentale entre cet algorithme DQN et une autre IA. DQN ne connaît pas les règles du jeu à l'avance. On peut citer le chercheur Volodymyr Mnih sur le principe de l'algorithme : "Ce que nous avons créé, c'est un algorithme capable d'apprendre directement de ses expériences – et donc plus proche de la manière dont les humains apprennent, et dont nos cerveaux construisent des modèles". Leurs différentes expérimentations de leur algorithme sont alors effectuées sur des jeux de la console Atari 2600. Pour déterminer la meilleure action à entreprendre DQN s'appuie sur le compteur de score qui était un élément présent dans tous les jeux à l'époque et calcule quelle action serait la meilleure et quelle action permettrait de rapporter le plus de points. On peut noter que le jeu Pong dispose d'un compteur de score, on pourra alors utiliser cet algorithme pour essayer d'avoir le meilleur score.

On peut cependant s'interroger sur la capacité de cet algorithme à être performant dans ce jeu. L'étude publiée par Nature (une des revues scientifiques les plus anciennes et les plus réputées dans le monde) montre des différences de performances de l'algorithme en fonction des jeux. L'algorithme

peut alors parfois “comprendre” comment jouer au jeu, (l'algorithme selon l'étude fonctionne bien sur le jeu Breakout ou le jeu du casse-briques, DQN a réussi à connaître la meilleure stratégie à adopter pour faire le maximum de points). On peut alors s'interroger sur la capacité de DQN à comprendre le jeu Pong. Même si le jeu Pong dispose d'un compteur de score, celui-ci n'augmente que de 1 à chaque échange gagné face à l'adversaire. Comment pourra-t-il déterminer le meilleur coup possible alors que tous les coups qu'il peut jouer peuvent lui apporter le même nombre de points. La présence d'une IA face à DQN nous apporte encore plus de questions. Comment l'IA dans Pong est-elle programmée ? Comment agit-elle face à un joueur et dans notre cas ici comment un algorithme d'apprentissage va se comporter face à elle.

5.3 Battre Pong ?

Essayer de battre l'ordinateur sur Pong semblait être un bon point de départ. Cependant les différents travaux issus du travail sur l'algorithme DQN mis au point par les chercheurs du laboratoire Deep Mind de Google ont permis de notamment dépasser ce qu'un humain pouvait faire face à la machine dans différents jeux Atari et qualifiant même les capacités de surhumainesquand ce dernier venait à jouer à Pong. Cette partie, semblait être un bon point de départ pour comprendre les capacités de notre programme face à un adversaire avec une intelligence artificielle précalculée, mais cela risque aussi de nous dériver vers des problématiques qui ne sont pas le cœur de notre sujet. Nous ne voulons pas que notre algorithme affronte un autre pour devenir le meilleur. Nous voulons que l'adversaire en face de lui soit lui-même, que celui-ci apprenne et devienne meilleur au fil des échanges. (Sources : Article de Venturebeat sur les performances de l'IA sur le jeu Pong, Rapport sur lequel se base l'article : le site arxiv.org)

5.4 Un agent et un environnement les bases de l'intelligence artificielle

Un environnement correspondra dans notre projet à une image de notre jeu Pong. Un agent pourra agir sur cet environnement à l'aide d'une action. L'action correspond à l'entrée que l'on va effectuer sur la manette pour contrôler l'élément représentant le joueur dans le jeu. L'agent en effectuant une action peut avoir une récompense. Notre récompense sera par exemple un point en plus lorsque que l'échange est gagné dans Pong. On cherchera alors à maximiser cette récompense. Un des problèmes de ce processus c'est que ce dernier va énormément se répéter.

La fonction Q va prendre un état et une action et nous permettre de connaître la récompense que l'on peut avoir au cours d'un échange. On estime que l'état correspond soit à une observation soit à une image de notre jeu. Il faut pour cela que notre fonction Q soit "parfaite". Lorsque l'on se retrouve alors à un état on peut effectuer un certain nombre d'actionson va alors choisir l'action où la récompense donnée par la fonction Q qui sera la meilleure. La fonction Q estime alors la récompense que l'on aura, pas seulement au prochain état en choisissant une action mais celle que l'on aura au bout d'un échange en prenant une action.

Nous allons détailler cette fonction Q. Soit "s" un état, a une action, R(s,a) correspond à la fonction de récompense et enfin notre fonction Q(s,a) retournant la valeur Q. La fonction Q va mesurer, pour une paire (état, action), son efficacité mais sur le long terme. L'algorithme de Q-learning estime alors Q(s,a) pour chaque paire état-action :

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[R(s_t, a_t) + \gamma max_{a \in A(s_{t+1})} Q(s_{t+1}, a) - Q(s_t, a_t)],$$

FIGURE 3 – Fonction Q

"st" est l'etat courant, st+1 est le prochain état en ayant choisi l'action à l'état courant st. A(st+1) correspond à l'ensemble des actions possibles à l'état st+1, y est le facteur d'actualisation (discount factor) et alpha correspond au taux d'apprentissage. Le discount factor varie entre 0 et 1. Si celui-ci est plus petit que 1 cela signifie que l'agent préfère la récompense courante plutôt que les recompenses passées.

On revient alors à notre DeepQ-Network, où pour un état ou une image de notre jeu on va estimer si une action nous apportera une grande récompense. La valeur que l'on obtient c'est la valeur Q. On a parlé de fonction Q "parfaite" mais pour que celle-ci devienne comme telle, il faut entraîner la fonction Q. Pour cela il faut jouer au jeu, on stocke ce que l'on a fait durant l'échange. Et on va utiliser cette expérience acquise pour jouer de nouveaux échanges. Cet entraînement va nous permettre d'obtenir une fonction Q maximale.

Une vidéo intéressante sur le travail de l'équipe composée de Volodymyr Mnih et d'autres chercheurs sur l'apprentissage par renforcement pour jouer à des jeux Atari. Le rapport sur lequel se base la vidéo.

5.5 Application à notre problème

Voici une réponse apportée aux problématiques de l'IA face à elle-même énoncées dans notre sujet. Deux agents qui agissent sur l'environnement. L'un veut maximiser sa récompense et donc obtenir le plus de points. L'autre souhaite que l'adversaire ait le moins de points possibles.

6 Autres algorithmes et techniques

Pour l'instant l'algorithme DQN reste l'un des meilleurs choix en termes d'implémentation mais pour savoir cela, nous nous sommes quand même penchés sur les autres choix possibles. L'algorithme SARSA est une variante de l'algorithme DQN mais qui a pour objectif de garantir une stratégie sûre ainsi qu'un chemin vers le but initial puisque l'un des problèmes de DQN est qu'il va tester un nombre de choix possibles en les effectuant. Or, il se peut qu'un de ses choix soit fatal. Dans l'exemple d'un jeu de combat le meilleur choix est peut-être de ne pas garder sa garde mais par conséquent il y a risque vis-à-vis de l'adversaire.

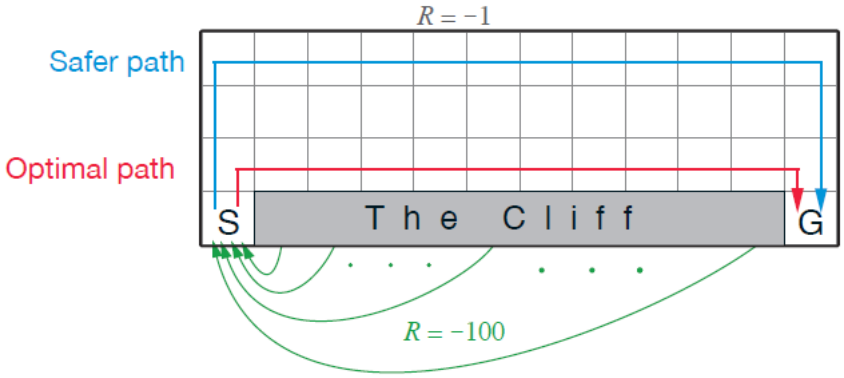


FIGURE 4 – Comparaison des algorithmes DQN (Q-Learning) ici en rouge et SARSA ici en bleu, avec un exemple

Source : medium.com

L'exemple ci-dessus est le parcours depuis une case S jusqu'à G sans tomber dans le vide (The Cliff). On voit que DQN donne un chemin optimal en coût et en temps mais qu'il y a des risques sur la recherche de chemin qu'il

tombe dans le vide, ce qui peut être génant dans un jeu vidéo s'il fait une faute grave qui lui coute la défaite. SARSA est plus lent et moins optimal mais il garantit un chemin « sûr ». Nous avons également commencé à regarder la notion de réseaux de neurones pour la partie de projet qui sera une machine qui joue contre elle-même et qui apprend de ses erreurs. Nous nous sommes inscrits pour le cours Droit et Conduite de projet sur cette notion avec le challenge AI4Industry qui sera pour nous un bon moyen de commencer à se former sur les réseaux de neurones.

7 Avancement du projet au premier semestre

À ce stade du projet, nous avons installé notre environnement Python avec la librairie Gym Retro. Nous avons cherché et trouvé une ROM du jeu Pong. Nous avons fait des recherches sur cette librairie et avons fait quelques codes pour la prendre en main. Nous avons par la suite fait des recherches pour comprendre les algorithmes de renforcement, en particulier DQN. Au second semestre nous allons essayer d'implémenter une IA à l'aide de cet algorithme.

8 Implémentation du code

Lors du second semestre, nous avons travaillé sur la partie implémentation d'un algorithme DQN avec l'environnement Gym. Le but étant de réussir à battre dans un premier temps le jeu Pong avec une IA. Puis dans un second temps nous voulions réussir à entrainer une autre IA contre celle de départ. Nous avons eu du mal à comprendre le fonctionnement de Gym/Gym Retro, ce n'est vraiment qu'à la fin du projet que nous commencions à l'appréhender correctement. Cela en partie à cause d'une documentation "officielle" que nous avons trouvée très succincte et peu détaillée. Il nous a fallu croiser beaucoup de sites web, d'articles, de vidéos et de codes afin de comprendre certaines méthodes et certains résultats. De plus la partie réseau de neurones n'a pas été simple à comprendre et surtout à mettre en pratique.

8.1 Lecture des variables de la RAM pour apprendre à jouer

Pour commencer nous avons implémenté l'environnement Pong et récupéré les variables importantes qu'il renvoie au cours du jeu. Nous avons ensuite cherché à comprendre comment étaient intégrées les ROMs dans l'environnement Gym Retro. Pour transformer une ROM de jeu en un environnement

d'apprentissage par renforcement, il est nécessaire de recourir à l'intégration du jeu dans l'environnement d'apprentissage.

L'intégration correspond à l'interface entre l'émulation du support vidéo-ludique et l'environnement d'apprentissage. Pour apprendre et jouer correctement, l'agent a besoin de travailler avec plusieurs éléments du jeu, comme l'affichage de celui-ci, les adresses des variables internes à la mémoire ou encore des sauvegardes d'états de jeu (dit "save states"). L'intégration permet donc de récupérer ces éléments afin de lire leurs valeurs à chaque étape de l'apprentissage.

L'intégration d'un jeu est importée par Gym/Gym Retro depuis différents fichiers d'intégrations JSON. Ces fichiers d'intégrations regroupent de façon structurée les éléments nécessaires à l'apprentissage des agents :

1. data.json : adresses mémoires des variables de jeu internes à la mémoire que peut lire l'API Python de Gym Retro
2. metadata.json : définit l'état du jeu de départ pour l'apprentissage et des informations utiles au débogage.
3. scenario.json : informations du jeu relatives à l'apprentissage par renforcement qui utilisent les variables dans data.json : fonctions de récompense et condition de fin

La plateforme Gym Retro propose un outil graphique d'intégration de jeux (disponible sur Linux).

Cet outil d'intégration (figure 5) assez sobre mais utile permet de lire la mémoire interne du jeu pendant l'émulation. Cela nous permet donc de rechercher par notification de changement des valeurs les adresses mémoires des variables souhaitées. On scanne la mémoire du jeu afin de trouver par directives de critères comme "Is", "Increased", "Decreased", "Changed" ou "Unchanged" les différents changements de valeurs.

La figure 6 nous montre que pour la directive "Increased", l'outil a scanné 961 variables différentes dans la mémoire. Au moment du scan, ces variables ont été observées sur l'augmentation ou l'incrémentation de leurs valeurs. Les champs nous indiquent leurs valeurs (elles changent à chaque mise à jour du jeu), leurs adresses mémoire ainsi que leurs types de données.

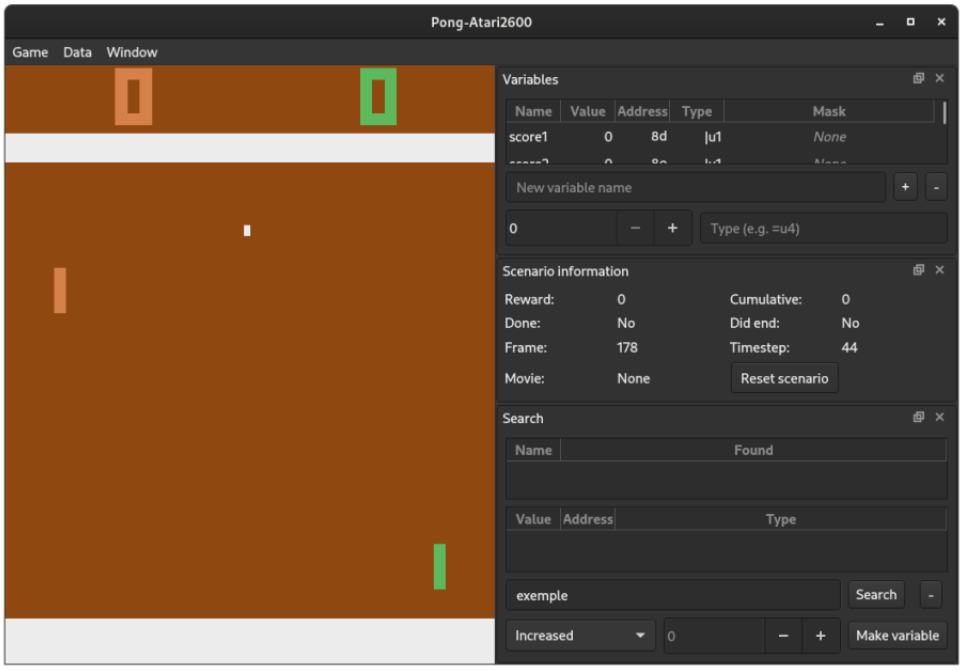


FIGURE 5 – Aperçu de l'outil graphique de Gym Retro

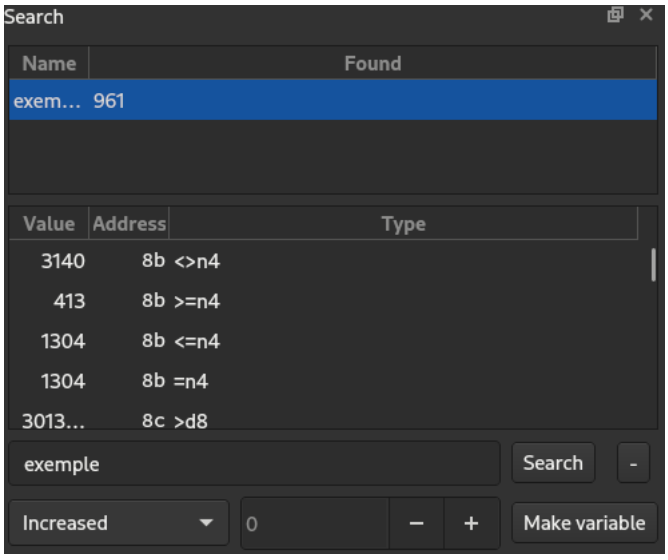


FIGURE 6 – Lecture de la mémoire via l'outil graphique

Il est important de bien être attentif aux changements de valeurs afin de repérer les bonnes adresses mémoires. Il est possible de vérifier que l'on ait trouvé les bonnes variables en changeant leurs valeurs et en observant le com-

portement du jeu résultant. Une fois nos variables finalement trouvées, l'outil permet d'exporter directement au format JSON les fichiers d'intégrations comprenant les adresses mémoires des nouvelles variables de jeu, qui seront importées et utilisées par Gym Retro pour l'apprentissage. L'outil d'intégration permet également de créer et d'exporter un scénario (pour l'apprentissage) à l'aide des variables trouvées, ainsi que des sauvegardes d'états du jeu à la manière d'un émulateur.

8.2 Lecture des images pour apprendre à jouer

L'API Gym possède déjà différentes intégrations pour le support Atari 2600. Dans notre cas, l'intégration et la rom du jeu Pong est disponible en deux versions. Une version ou nous pouvons observer la Ram et l'autre où nous pouvons récupérer les images en couleurs du jeu. Respectivement ces Roms se nomment Pong-ram-v0 et Pong-v0. Nous avons choisi d'utiliser la deuxième pour la bonne raison qu'un agent humain jouant à Pong regarde les images du jeu (déplacements de la balle) et non pas la Ram du jeu. Nous avons voulu que notre agent intelligent ait le même comportement qu'un agent humain.

Remarque : Un agent intelligent est une entité autonome capable de percevoir son environnement grâce à des capteurs et aussi d'agir sur celui-ci via des effecteurs afin de réaliser des buts. Dans notre cas l'agent correspond à un joueur (IA) jouant contre le jeu Pong. Source : https://fr.wikipedia.org/wiki/Agent_intelligent

Gym met à disposition pour l'utilisateur une implémentation d'environnements de jeux. Celui-ci contient toutes les informations destinées à l'apprentissage par renforcement d'un agent dans un jeu-vidéo : les états du jeu perçus par l'agent, l'ensemble des actions qu'il peut prendre ou encore les récompenses reçues selon une action donnée. L'environnement du jeu évolue continuellement tout au long d'un scénario.

Quelques définitions sur l'ensemble des notions gravitantes à un environnement de jeu :

1. Étape : une étape du jeu, décrite dans notre cas comme une image dans le temps ("frame" dans sa version anglophone)
2. Épisode ou Scénario : Un épisode / scénario est décrit comme une succession finie d'étapes du jeu. La fin d'un scénario est invoquée par une condition d'arrêt

3. Observation : un état observable de l'environnement de jeu à une étape donnée (état de la mémoire, lors de la lecture de la RAM ou l'image courante, lors de la récupération des images)
4. Action : action entreprise par l'agent sur l'environnement qui prend effet lors de la prochaine étape. Correspond aux "contrôles", aux boutons associés au périphérique d'entrée. C'est la console Atari 2600 dans notre cas.
5. Récompense : gain associé à l'agent, obtenu grâce à l'action qu'il a choisie précédemment dans le jeu.

8.3 Un agent "pas intelligent" jouant à Pong

Nous allons voir dans cette partie comment utiliser les outils fournis par Gym et visualiser une partie de Pong avec un agent qui n'a pas été formé. Notre environnement Gym a été mis en place comme montré ci-dessous. Nous récupérons également des variables utiles que nous allons par la suite détailler. Nous initialisons l'environnement du jeu Pong comme suit :

```
#Environnement (Roms disponiblent sur https://gym.openai.com/envs/#atari)
env = gym.make('Pong-v0')
```

Nous récupérons deux variables importantes qui sont le nombre d'actions que peut effectuer le joueur ainsi que les actions elles même. Ici nous obtenons 6 actions possibles soit : NOOP, FIRE, RIGHT, LEFT, RIGHTFIRE et LEFTFIRE.

```
#Nombre d'actions disponibles pour le jeu chargé
actions = env.action_space.n
print("Nombre d'actions qu'un joueur peut choisir sur Pong :",actions)

#Les actions disponibles pour le jeu chargé
print("Les actions disponibles pour un joueur sont :")
env.unwrapped.get_action_meanings()
```

On peut également demander au jeu (en tant qu'agent) de réaliser une action spécifique avec le code suivant :

```
action = random.choice([0,1,2,3,4,5])
observation, reward, done, info = env.step(action)
```

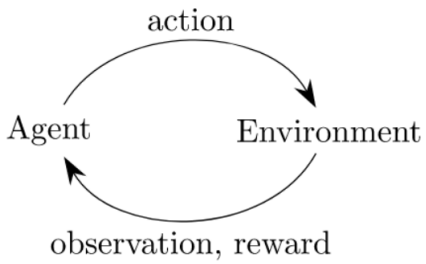
La variable action représente une action entre 0 et 5 et env.step(action) permet d'indiquer au jeu l'action à effectuer. Il est à noter que 0 correspond à l'action "NOOP", 1 à "FIRE" etc... À la suite de cela, l'environnement nous renvoie quatre variables qui nous seront utiles pour construire notre futur réseau de neurones. À savoir :

1. "observation" (objet) : contient les données image à une étape de jeu

```
Donnees d'observation -> env.observaton_space.shape :
(nb pixels largeur : 210, nb pixels hauteur : 160, nb composantes pixels : 3)
```

2. "reward" (float) : récompense obtenue par l'agent à une étape du jeu.
3. "done" (boolean) : si le jeu est terminé ou non. Condition de fin d'un épisode, déterminée dans le fichier d'intégration de la ROM du jeu.
4. "info" (dictionnaire) : informations supplémentaires sur l'étape de jeu actuelle fournies par l'environnement Gym (valeurs des variables d'intégrations...).

Le schéma ci-dessous synthétise comment les données circulent entre l'agent et l'environnement.



Remarque : L'attribution de la récompense (reward) pour un agent jouant à Pong fonctionne comme ceci :

1. S'il remporte un échange, il obtient +1.0
2. S'il fait des échanges avec le jeu, il obtient 0.0
3. S'il perd un échange, il obtient –1.0

Ainsi nous sommes en mesure de faire jouer un premier agent, pas très intelligent, puisqu'il se contente de réaliser des actions de manières aléatoires. Notre code permet de générer une vidéo (disponible sur notre git de projet) montrant le jeu Pong jouant contre cet agent. En la visionnant on remarquera

que l'agent ne marque aucun point ou alors s'il réussit ce n'est seulement que du hasard. Le but étant maintenant de doter notre agent d'une intelligence artificielle le permettant de jouer et d'apprendre de lui-même à marquer des points contre l'adversaire. Pour cela nous allons utiliser un algorithme DQN (Deep Q-Networks), dont ses principes ont été vu plus haut dans ce mémoire.

8.4 Un agent intelligent jouant à Pong

Construire un réseau de neurones type Deep Q-Networks nécessite plusieurs éléments.

Tout d'abord construire une modélisation de réseau de neurones appropriée à l'apprentissage par renforcement pour Pong sur lequel faire travailler l'algorithme DQN.

Également trouver une politique adéquate pour l'algorithme DQN afin d'optimiser les choix d'actions de l'agent dans l'environnement.

Et enfin pour augmenter les performances de l'algorithme une "loss function" pour déterminer les erreurs de prédictions ainsi qu'un optimisateur pour corriger les poids du modèle et paramètres de l'algorithme afin de minimiser les erreurs de prédictions.

Pour prédire l'action que l'agent intelligent doit choisir pour marquer des points, il doit interpréter la direction de la balle pour pouvoir la renvoyer avec sa raquette. En pratique nous allons utiliser un réseau de neurones à convolution pour prédire la bonne action à jouer selon les situations. Nous allons récupérer chaque image du jeu en temps réel, les interpréter et à la suite de cela le modèle sera capable de prédire l'action la plus pertinente à jouer. Source : https://web.stanford.edu/class/psych209/Readings/MnihEtAlHassibis15NatureControlDeepRL.pdf

Comme l'indique le papier de recherche (page 11), un modèle bien entrainé arrive à obtenir plus ou moins 19 points.

Lors de l'implémentation du code, plusieurs tentatives ont été faites. Aussi bien en lisant les données dans la Ram qu'en utilisant les images du jeu. Nous avons utilisé les bibliothèques Keras et TensoFlow pour créer nos modèles et nos agents avec Python. Dans un premier temps nous avons rédigé le code ci-dessous afin d'avoir un modèle et un agent DQN.

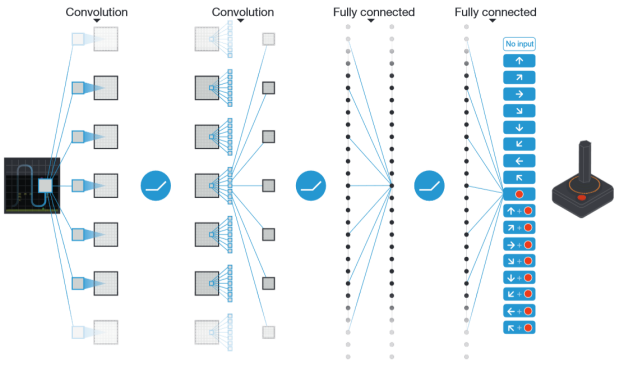


FIGURE 7 – Schéma tiré du papier de recherche représentant un réseau de neurones

```
#Définition de l'environnement Gym
ENV_NAME = 'Pong-v0' #@param {type:"string"}
NOMBRE_APPRENTISSAGES = 2 #@param {type:"number"}
NOMBRE_TEST =  2#@param {type:"number"}
env = gym.make(ENV_NAME)

#Définition d'une fonction qui créée un modèle.
def getModel(env):
  model = Sequential()
  model.add(Flatten(input_shape=(1,) + env.observation_space.shape))
  model.add(Dense(16))
  model.add(Activation('relu'))
  model.add(Dense(16))
  model.add(Activation('relu'))
  model.add(Dense(16))
  model.add(Activation('relu'))
  model.add(Dense(env.action_space.n)) #Nombre d'actions disponibles
  model.add(Activation('linear'))
  return model

#Définition d'une fonction qui créée un agent.
def getAgent(env,model):
  memory = SequentialMemory(limit=500, window_length=1)
  policy = BoltzmannQPolicy()
  dqn = DQNAgent(model=model, nb_actions = env.action_space.n, memory = memory,
                 nb_steps_warmup=10, target_model_update=1e-2, policy=policy)
  return dqn

model = getModel(env)
agent = getAgent(env,model)
```

getModel() permet de créer un réseau de neurones à plusieurs couches en prenant en compte le nombre d'actions qu'il est possible de faire lorsque l'on joue à Pong. getAgent() permet de créér un agent DQN à l'aide d'un modèle. Nous faisons ensuite le code suivant pour entraîner notre modèle avec NOMBRE_APPRENTISSAGE = 50000.

```
#Entrainement
agent.compile(Adam(lr=1e-3), metrics=['mae'])
agent.fit(env,nb_steps=NOMBRE_APPRENTISSAGES, visualize=False, verbose=2)
```

Pour terminer nous le testons avec la ligne ci-dessous pour vérifier qu'il est fiable.

```
scores = agent.test(env,nb_episodes=NOMBRE_TEST,visualize=True )
```

En affichant dans la console avec NOMBRE_APPRENTISSSAGE = 50000 et NOMBRE_TEST = 20 nous obtenons ceci :

```
 10863/20000: episode: 9, duration: 40.163s, episode steps: 1139, steps per second:  28, episode reward: -20.000, mean reward: -0.018 [-1.000,  1.000], mean action:
 12504/20000: episode: 10, duration: 58.815s, episode steps: 1641, steps per second:  28, episode reward: -18.000, mean reward: -0.011 [-1.000,  1.000], mean action:
 13764/20000: episode: 11, duration: 47.595s, episode steps: 1260, steps per second:  26, episode reward: -21.000, mean reward: -0.017 [-1.000,  0.000], mean action:
 15004/20000: episode: 12, duration: 47.125s, episode steps: 1240, steps per second:  26, episode reward: -20.000, mean reward: -0.016 [-1.000,  1.000], mean action:
 16352/20000: episode: 13, duration: 51.035s, episode steps: 1348, steps per second:  26, episode reward: -21.000, mean reward: -0.016 [-1.000,  0.000], mean action:
 17379/20000: episode: 14, duration: 38.916s, episode steps: 1027, steps per second:  26, episode reward: -21.000, mean reward: -0.020 [-1.000,  0.000], mean action:
 18487/20000: episode: 15, duration: 42.025s, episode steps: 1108, steps per second:  26, episode reward: -21.000, mean reward: -0.019 [-1.000,  0.000], mean action:
 19799/20000: episode: 16, duration: 49.286s, episode steps: 1312, steps per second:  27, episode reward: -20.000, mean reward: -0.015 [-1.000,  1.000], mean action:
done, took 733.069 seconds
<tensorflow.python.keras.callbacks.History at 0x7f027a5f4650>Testing for 20 episodes ...
Episode 1: reward: -21.000, steps: 1004
Episode 2: reward: -21.000, steps: 999
Episode 3: reward: -21.000, steps: 1024
Episode 4: reward: -21.000, steps: 1007
Episode 5: reward: -21.000, steps: 1009
Episode 6: reward: -21.000, steps: 1023
Episode 7: reward: -21.000, steps: 1013
Episode 8: reward: -21.000, steps: 1017
Episode 9: reward: -21.000, steps: 1034
Episode 10: reward: -21.000, steps: 1014
Episode 11: reward: -21.000, steps: 1015
Episode 12: reward: -21.000, steps: 1027
Episode 13: reward: -21.000, steps: 1018
Episode 14: reward: -21.000, steps: 1008
Episode 15: reward: -21.000, steps: 1022
Episode 16: reward: -21.000, steps: 1015
Episode 17: reward: -21.000, steps: 1012
Episode 18: reward: -21.000, steps: 1016
Episode 19: reward: -21.000, steps: 1020
 13%|▉       | 128/1021 [00:00<00:00, 1275.79it/s]Episode 20: reward: -21.000, steps: 1019
```

La variable de récompense (reward) varie de –18 à -21 lors de l'entrainement ce qui montre que le modèle obtient au maximum 3 points. Mais lors de la phase de test, il reste à -21. Par conséquent notre modèle n'est pas bon ou nous ne l'utilisons pas correctement. Nous l'avons entraîné 20000, 50000 puis 650000 fois (+7h d'exécution) pour établir le fait que ce n'était pas l'entrainement qui était responsable de nos mauvais résultats.

Nous avons essayé d'implémenter, avec la documentation de TensorFlow, un autre code en utilisant cette fois ci, non pas les images comme entrée du modèle mais les variables de la RAM. Le code commenté qui a été implémenté sera disponible sur notre git en complément de ce mémoire car trop long à détailler ici. Au final ce code ne nous a pas trop convaincu non plus, l'agent peine à marquer des points. Là encore peut-être que notre modèle n'a pas été utilisé correctement. Lien du tutoriel : https://www.tensorflow.org/agents/tutorials/1_dqn_tutorial

Pour finir nous avons essayé le code disponible sur la documentation de keras-gym. Celui-ci nous a donné un résultat très satisfaisant. La partie de Pong avec notre agent intelligent a été enregistrée. Elle est disponible dans notre git de projet. Il a fallu plus de 10h pour entrainer ce modèle.

Code disponible : https://keras-gym.readthedocs.io/en/stable/notebooks/atari/dqn.html

Pour information, les parties de code ci-dessus ont été rassemblées dans git disponible à cette adresse : https://git.unilim.fr/lacoste37/ProjetM1.

9 Conclusion

Ce projet a été un vrai challenge. Pour la majorité de l'équipe nous n'avions ni bases en Python ni bases d'intelligence artificielle. Nous avons donc tout appris, aussi bien à manipuler Python et son environnement que les concepts non vus en cours comme l'apprentissage par renforcement (DQN). Le problème majeur rencontré fut le manque de documentation autour des librairies utilisées pour la construction de réseaux de neurones. Nous avons tout de même réussi à avoir un agent intelligent jouant à Pong. Du fait que cette partie était assez difficile, nous n'avons pas pu intégrer un agent de ce type sur un jeu plus compliqué comme Street Fighter par exemple. Ce projet a été très formateur.

Ce projet, bien qu'ayant eu des difficultés, nous a permis d'avoir une expérience dans le domaine de l'IA et plus particulièrement dans l'apprentissage par renforcement. Nous n'avons pas pu réaliser tout ce qu'on voulait faire (comme tester avec un autre jeu ou un autre algorithme de renforcement), mais ce projet a quand même complété les modules liés à l'IA cette année et nous a montré l'intérêt de travailler dessus.

10 Grille de participation

Etudiant		Organisation, coordination		Production		
Nom	Prénom	Coordination des RDV	Participation active aux RDV	Code	Rapport Final	Préparation de la soutenance
COURTENS	Antoine	4	4	4	3	4
DUPONT	Olivier	4	4	3	4	4
FABRIES	Quentin	5	4	4	4	4
LACOSTE	Léo	4	4	4	3	4

11 Références

1. **Apprentissage par renforcement :** https://fr.wikipedia.org/wiki/Apprentissage_par_renforcement

2. **Article du journal "Le Monde" : "Un algorithme capable d'apprendre à jouer aux jeux vidéo"** https://www.lemonde.fr/pixels/article/2015/02/25/un-algorithme-capable-d-apprendre-a-jouer-aux-jeux-video_4583358_4408996.html

3. **Article de "Venturebeat" sur les performances de l'IA sur le jeu Pong :** https://venturebeat.com/2018/11/16/openai-and-deepmind-ai-system-achieves-superhuman-performance-in-pong-and-

4. **Rapport sur lequel se base l'article de Venturebeat :** https://arxiv.org/pdf/1811.06521.pdf

5. **Pourquoi le jeu Pong n'est pas vraiment sur Atari 2600 :** https://atariage.com/forums/topic/131764-why-cant-i-find-pong/

6. **Vidéo Youtube sur le travail de l'équipe composée de Volodymyr Mnihet d'autres chercheurs sur l'apprentissage par renforcement pour jouer à des jeux Atari :** https://www.youtube.com/watch?v=rFwQDDbYTm4&ab_channel=YannicKilcher

7. **Le rapport sur lequel se base la vidéo :** https://arxiv.org/pdf/1312.5602.pdf

8. **Algorithme SARSA :** https://medium.com/gradientcrescent/fundamentals-of-reinforcement-learning-navigating-cliffworld-with-sarsa-an

9. **Rom du Pong :** https://www.retrostic.com/roms/atari-2600/video-olympics-pong-sports-6632

10. **Console Atari 2600 :** https://fr.wikipedia.org/wiki/Atari_2600

11. **Article sur l'apprentissage par renforcement :** https://www.lebigdata.fr/reinforcement-learning-definition

12. **Configuration requise de Gym Retro :** https://retro.readthedocs.io/en/latest/index.html

13. **WinPython :** https://winpython.github.io/

14. **Python :** https://www.python.org/

15. **Utilisation de Pip :**
https://docs.python.org/fr/3.6/installing/index.html

16. **Image de couverture :** https://www.pcmag.com/news/video-kids-react-poorly-to-atari-2600

17. **Papier de recherche :**
https://web.stanford.edu/class/psych209/Readings/MnihEtAlHassibis15NatureControlDeepRL.pdf

18. **TensorFlow :** https://www.tensorflow.org/?hl=fr

19. **Keras :** https://keras.io/

20. **Gym Environnemnt Atari :**
https://gym.openai.com/envs/#atari

21. **Pong DQN :** https://gym.openai.com/envs/#atari