Rapport technique, Projet final d'IAT

"IA playing Z Invaders"

Nathan BOTTET, Lucas DUFOUR, Mathis FAUCHEUX, Pierre RENAUD

30 avril 2022

Ce rapport technique a pour objectif de présenter les réflexions et le cheminement ayant abouti à la production du projet final d'IAT de notre groupe. Vous trouverez à la suite de ce document des annexes contenant des graphiques utiles à la compréhension générale du projet.

1 Présentation du projet

Le problème que l'on souhaite résoudre est le jeu **Space Invaders**, ici remastérisé en Z Invaders. Notre agent est un vaisseau pour qui le but est de survivre en détruisant le plus de Z possible. Les **quatre actions possibles** pour notre agent sont les suivantes : (0 : aller à gauche, 1 : aller à droite, 2 : tirer, 3 : ne rien faire)

Chaque Z détruit rapporte un point, et la partie continue tant qu'un Z n'a pas touché notre agent (le Z se déplace latéralement puis verticalement en direction de notre agent).

2 QLearning

Comme annoncé dans le sujet, notre but original est de réutiliser un des algorithmes d'**apprentissage par renforcement** vu lors des précédentes séances afin d'obtenir le plus haut score possible au jeu Space Invaders. Nous avons donc choisis de débuter notre résolution en utilisant le **Q Learning**.

2.1 Formalisation

Tout d'abord il est nécessaire de formaliser notre problème en définissant un **nombre fini d'états**. Ces états seront caractérisés selon les variables suivantes :

— distance en x par rapport au Z (discretisé en $(800px/25) * 2 = 64$ valeurs)
— direction latérale du Z (+1 ou -1)
— distance en y entre l'agent et le Z (discrétisé en 16 valeurs)

Ces états vont ainsi être pris en compte dans la **prise de décision de l'action la plus adéquate à effectuer** : c'est la **politique**.

2.2 Algorithme

L'algorithme de QLearning cherche à apprendre une **politique qui maximise la récompense totale**. Ici la **récompense** est représentée par le **point gagné** en touchant le Z, et donc par extension, la **durée totale de la partie** car on sait que la partie se termine quand l'agent à été touché.

Tout d'abord, l'algorithme peut s'exécuter soit en **mode exploration** soit en **mode exploitation**. En effet, quand une première simulation se lance, l'algorithme n'a pas de données pour faire son choix d'action, il va donc "essayer" des actions au "hasard" pour entamer la construction de sa **matrice Q** (fonction d'action-valeur). C'est cette matrice Q qui sera exploitée ensuite par la politique pour déterminer les actions à suivre. Au fur et à mesure de ses choix, Q sera mise à jour et étoffée pour devenir la plus précise possible (=remporter le plus de récompenses).

Nous avons utilisé ainsi utilisé la politique de l'algorithme de QLearning vu précédemment en TP :

$$Q[etat][action]^{n+1} = (1-\alpha)*Q[etat][action]^{n}+\alpha(reward+\gamma*max(Q[prochain\_etat])) \quad (1)$$

2.3 Apprentissage et Hyperparamètrage

Les **hyperparamètres** (HP) sont des variables initialement déclarées (on leur assigne une valeur) qui vont grandement influencer l'apprentissage. Voici une liste non exhaustive des HP les plus déterminants (=l'échantillon que l'on a sélectionné) :

- Learning rate (α : poids donné à l'apprentissage)
- Parametre d'exploration (ϵ_{ini} : fréquence du mode exploration, décroit selon une sigmoide au fur et à mesure des episodes)
- Parametre de récompense future (γ : importance donnée aux récompenses)
- Episodes (nombre de parties effectuées)
- Steps (temps d'une partie)

Pour optimiser au mieux notre apprentissage, il est nécessaire de chercher les **valeurs de ces HP** qui vont donner **le meilleur apprentissage**. Tout d'abord, pour témoigner de l'apprentissage, on peut suivre le nombre de points avant de perdre : en effet, plus l'IA joue bien et apprends, plus elle doit marquer de point avant de perdre. On va donc lancer plusieurs simulation (SM) en faisant varier un HP à la fois sur chacune de nos SM. Nous sélectionnerons les valeurs d'HP qui donnent le plus haut score à la fin des épisodes d'apprentissage indiqués. Dans un soucis de place et de lisibilité nous avons fait le choix de ne pas présenter toutes les courbes d'optimisation d'HP ici (La Figure 1 est un tableau représentant un échantillon des différents essais). Vous trouverez en Figure 2 un exemple sur lequel nous avons représenté plusieurs moyennes (soit plusieurs algorithmes et leur HP) exploitées sur des simulations courtes sans exploration ($\gamma = 0$, $nb\_episodes = 100$, $nb\_steps = 75000$) en parallèle, afin de déterminer la plus optimale en sélectionnant celle qui obtient le meilleur score.

Les HP optimaux sont ainsi les suivants : $(\alpha = 0.1, \epsilon_{ini} = 0.8, \gamma = 0.9, nb\_episodes = 10000, nb\_steps = 15000)$

2.4 Analyse des résultats

Nous avons donc un échantillon des HP les plus optimaux, que nous avons utilisé pour lancer plusieurs mêmes simulations courtes sans exploration en parallèle. Pourtant, nous avons finalement des résultats très peu satisfaisants. En effet en effectuant une moyenne des moyennes glissante sur les 10 derniers épisodes de chaque simulation courte (Figure 2), nous observons un score moyen très faible. Ces résultats sont incohérents, et cela pourrait être lié à plusieurs phénomènes.

Par exemple, il est peut être possible que l'algorithme passe trop de temps en exploration (soit que la fonction qui régit epsilon ne décroisse pas assez vite, voir Figure 3).

Nous avons donc fait le choix de tenter une nouvelle approche.

3 Algorithme Génétique

Ce second choix (qui sort légèrement du sujet de ce projet) nous tenait à cœur et nous a permis d'obtenir d'excellents résultats, c'est pour cela que nous prenons la liberté de présenter une implémentation d'un algorithme génétique (AG).

3.1 Formalisation

La formalisation de cet algorithme se fait par la prise en compte des coordonnées x, y à la fois de l'agent et du Z, de l'état $bullet\_state$ et du sens de déplacement latéral du Z. Ces éléments définissent l'état de notre jeu. Nous conservons les actions définies en début de document

3.2 Algorithme

L'AG que nous avons implémenté repose sur la notion de **générations** de **réseaux de neurones**. En effet, pour diriger la prise de décision de notre IA nous avons mis en place un réseau de neurones (RN) de type 6 input layer, 25 relu, 25 relu et 4 output layer pour les 4 actions possibles. Le but de cet algorithme génétique va être d'**améliorer ce RN à chaque itération/génération**, dans le but d'obtenir le plus haut score.

A chaque génération, nous allons garder seulement les 10% meilleurs réseaux, auxquels nous allons pouvoir appliquer deux actions : la mutation et le crossover, dans le but de "repeupler" en modifiant les mauvais RN à partir des bons sélectionnés précédemment.

La **mutation** consiste tout d'abord à prendre un "bon" RN (dans les 10%) et de copier tous ses poids sur un mauvais RN. Ensuite, pour chaque poids de chaque layer, on tire un nombre entre 0 et 1 , si il est inférieur au mutation rate alors on modifie ce poids en lui rajoutant un valeur aléatoire entre -0.5 et 0.5.

Le **crossover** consiste quand à lui à dupliquer deux "bons" RN sur de mauvais RN, et d'intervertir un poids aléatoire entre les deux.

De plus, pour mieux démarquer les RN entre eux, nous avons accentué la récompense : en effet lorsqu'un agent (donc son RN associé) touche un Z, il gagne également un certain temps donné de simulation en plus afin d'avoir un score encore plus élevé. (Ce procédé à ses limites, on à quelque fois des simulations qui s'éternisent et jouent "à l'infini").

Pour mener à bien ces *très* nombreuses simulation, nous avons parallélisé nos simulation dans des threads.

3.3 Apprentissage et Hyperparamètrage

Comme précédemment, nous avons sélectionné un échantillon d'HP qui semblent pertinents à étudier :

- Nombre de générations (gen)
- Population par génération (pop : nombre de RN)
- Mutation rate ($mr(x)$: facteur affectant le fréquence des mutations)
- Nombre de Steps (s : temps d'une partie)
- Le pas ($p(x)$: nombre de steps gagnés lors d'une réussite, décroit à chaque touche)

Nous avons comme précédemment réalisé des études afin de définir nos HP optimaux, qui sont les suivants : $(gen = 120, pop = 40, mr(x), s = 800, p(x), = 100)$ avec $mr(x)$ une fonction sigmoïde permettant de faire varier le mutation rate tout au long de la simulation. Et $p(x) = s/2 - 20 * score\_actuel$.

3.4 Analyse des résultats

Cette configuration de notre algorithme génétique nous permet d'obtenir des parties allant jusqu'à plus de 1000 points. Pour visualiser l'évolution de l'apprentissage, on représente à chaque générations la moyenne des scores des dix meilleurs RN.

4 Conclusion

Tout d'abord, les résultats non satisfaisant de QLearning sont sans doute dus à un mauvais paramétrage soit de nos états soit de nos HP.

L'AG quand à lui à démontré une excellente capacité à produire des réseaux de neurones permettant de jouer sur des très grandes durées sans perdre. Cependant ce n'est pas le procédé le plus optimal.

En effet un algorithme génétique serait optimal dans un cas ou on aurait plusieurs agents sur une session de jeu, dans nos cas pour une population de 100 agents (soit 100 RN), on doit ouvrir en parallèle 100 sessions de jeu (ce qui est très lourd).