
Renforcement learning pour LeekWars

Rapport final

Encadrant :
Mathieu Lafourcade

Etudiants :
Juliat Thomas - 22108957 Aissaoui Lucas - 22104312
Cauty Alexandre - 22115525 Schnuriger Loïc - 22015463

Remerciements

Nous tenons à remercier notre tuteur, monsieur Mathieu Lafourcade pour l'aide qu'il nous a apporté tout au long de notre projet.

Table des matières

1 Introduction	3
1.1 Sujet	3
1.2 Leek Wars	3
1.3 Objectifs	3
2 Modélisation du problème	4
2.1 Actions	4
2.1.1 Actions d'attaques	4
2.1.2 Actions de déplacement	4
2.2 États	4
2.2.1 Map	4
2.2.2 Armes	5
2.2.3 Puces	6
2.2.4 Caractéristiques	6
2.3 Espace d'états	7
3 Partie technique	8
3.1 Générateur de combats	8
3.2 Cycle d'entraînement	9
3.3 Sauvegarde des modèles	9
4 Partie stratégique	11
4.1 Stratégie de renforcement du modèle	11
4.2 Stratégie de sélection des actions	12
4.3 Stratégie d'entraînement	12
4.3.1 Entraînement contre une IA existante	12
4.3.2 Entraînement automatique	13
5 Problèmes rencontrés	14
6 Résultats	14
7 Conclusion	15

1 Introduction

1.1 Sujet

"L'objet de ce sujet est de créer une IA qui apprend à se battre efficacement, via une approche combinant apprentissage par renforcement (association état courant et action(s) à réaliser) et réseaux de neurones profonds (pour évaluation/classification de l'état courant). L'efficacité de l'IA se mesurera à son classement dans le site du jeu."

1.2 Leek Wars

Leek Wars est un jeu de programmation dans lequel vous devez créer le plus puissant poireau et détruire vos ennemis ! Développez votre propre intelligence artificielle grâce au LeekScript, un langage facile à apprendre, et devenez le meilleur éleveur !

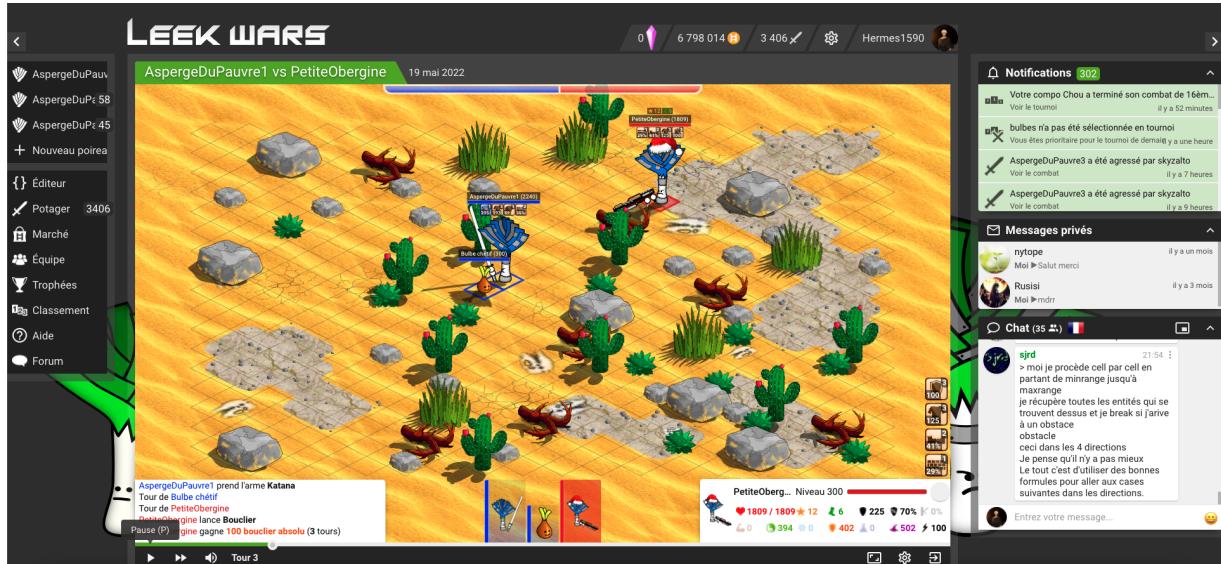


Figure 1 – Interface du jeu

1.3 Objectifs

L'objectif pour notre groupe est de réussir à mettre au point un système capable de construire par renforcement des IA uniquement en interpréter les résultats obtenus à la suite des combats réalisés par cette dernière.

2 Modélisation du problème

2.1 Actions

Il existe 24 armes et 98 puces qu'il est possible d'acheter avec la monnaie du jeu. Un poireau niveau 301 peut utiliser 4 armes et 20 puces pour un combat. Ces armes et puces peuvent faire des dégâts, soigner ou appliquer divers effets sur les poireaux et ont chacune une portée et une zone d'effet. Toutes les armes et puces nécessitent un certain nombre de points de tour à chaque utilisation. Certaines puces ne peuvent pas être utilisées plus d'une fois par tour et la plupart ont un temps de recharge de plusieurs tours.

Chaque poireau possède donc des points de tour (PT) et de mouvement (PM) qui sont réinitialisés à chaque début de tour (sur la figure 5 on peut voir que notre poireau à 21 points de tour et 4 points de mouvement).

2.1.1 Actions d'attaques

Il existe plusieurs actions qui utilisent les points de tour.

Pour attaquer avec une arme il faut, en premier, l'équiper avec la fonction `setWeapon(weapon)`. Cette fonction utilise 1 PT à chaque fois qu'elle est utilisée même si l'arme est déjà équipée. Nous avons donc nous-mêmes défini une fonction qui vérifie que l'arme n'est pas déjà équipée et qui nous évite ainsi de perdre inutilement des points de tour.

Après avoir équipé l'arme, la fonction `useWeapon(entity)` permet d'attaquer l'entité donnée en paramètre de la fonction. Comme expliqué plus haut, chaque arme à une zone d'effet et une portée, il faut donc être bien placé pour pouvoir toucher l'ennemi.

Il existe plusieurs puces qui permettent de rajouter des PT pendant un nombre de tours limité.

2.1.2 Actions de déplacement

Il existe deux principales actions de déplacement, `moveAwayFrom(entity, [mp])` et `moveToward(entity, [mp])` qui permettent respectivement de s'éloigner et se rapprocher d'une entité. On peut, pour ces deux fonctions, spécifier un nombre de MP maximum à utiliser. Un déplacement d'une case coûte 1 MP. Il existe plusieurs puces qui permettent de rajouter de MP pendant un nombre de tours limité.

2.2 États

Plusieurs fonctions prédéfinies sont disponibles en leekScript, chaque fonction utilise un certain nombre d'opérations qui peut aller de 5 à plusieurs dizaines de milliers. Le nombre maximal d'opération pendant un tour est de 20 millions, au-delà l'IA plante et le poireau ne fait plus rien jusqu'au prochain tour.

Le nombre d'opérations est donc un facteur important à prendre en compte dans notre projet.

2.2.1 Map

Il existe 6 terrains de combats différents dans le jeu. Un terrain est un carré de 17 cases de côté. Chaque case est numérotée de 0 à 612 et a des coordonnées (x;y). La case de coordonnées (0;0) est au milieu du terrain et la case numéro 0 est dans un angle du terrain.

Chaque terrain a des obstacles placés différemment, la plupart des armes et des puces ne peuvent pas toucher un autre poireau à travers les obstacles.

Le terrain est choisi aléatoirement à chaque combat.



Figure 2 – Exemple de terrain

2.2.2 Armes

Leek Wars compte 25 armes. les armes doivent être prises en main pour être utilisées. On peut avoir 2 à 4 armes sur un poireau selon son niveau, mais on ne peut en utiliser qu'une seule à la fois. Un poireau porte plusieurs armes mais ne peut se servir que de celle qu'il a dans les mains. Ainsi, il faut utiliser la fonction setWeapon(weapon) pour changer l'arme qui est dans les mains du poireau. Un changement d'arme coûte 1 PT, et ce, même si cette arme est déjà équipée. On doit donc tester si une arme est déjà équipée pour éviter de l'équiper inutilement.



Figure 3 – Armes disponible sur Leek Wars

2.2.3 Puces

Leek Wars compte 98 puces. Les puces, contrairement aux armes, n'ont pas besoin d'être "prises en main". Il suffit d'équiper une puce pour pouvoir l'utiliser. Pour se servir d'une puce, il faut se servir de la fonction useChip(chip, entity). Celle-ci prend deux paramètres : la puce à utiliser et la cible.

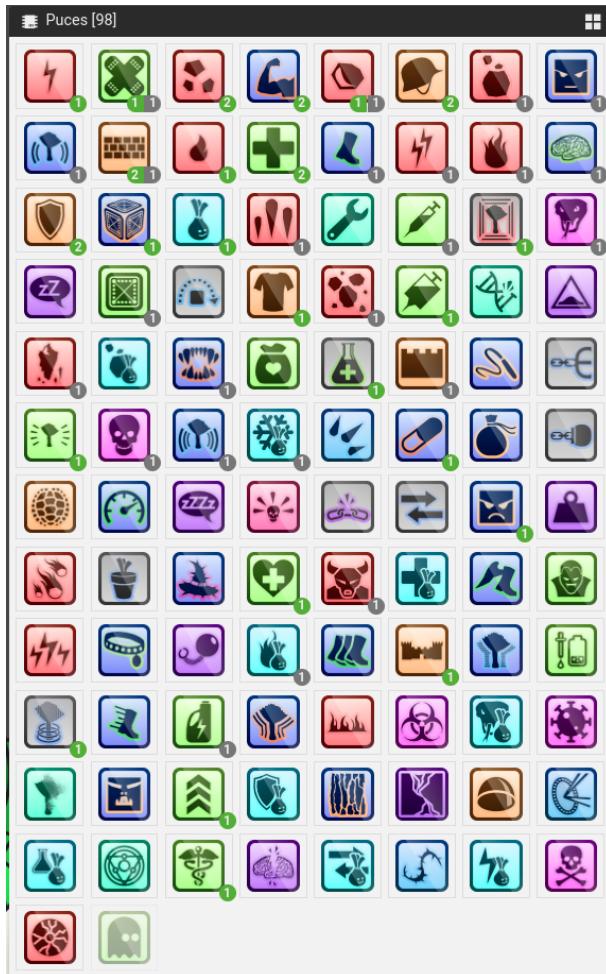


Figure 4 – Les caractéristiques

2.2.4 Caractéristiques

Un poireau possède plusieurs caractéristiques qui peuvent être augmentées grâce à des points de capitale que l'on obtient en passant des niveaux.

Points de Vie	Force	Sagesse	Résistance	Agilité	Science	Magie	Fréquence	PT PM

Figure 5 – Les caractéristiques

2.3 Espace d'états

Il existe plusieurs types de combat :

- Le combat solo
- Le combat 4 contre 4
- Le combat 6 contre 6
- Le battle royale où le but est d'être le dernier survivant parmi 10 poireaux.

Pour notre projet nous nous concentrerons sur le combat solo, il oppose deux poireaux qui possèdent chacun 4 armes, 20 puces, 10 valeurs pour leurs caractéristiques sur une map de 613 cases. On peut estimer la taille de l'espace d'état à :

$$\binom{4}{25} \cdot \binom{20}{98} \times 613 \times [...] \times 2 \approx 10^{12}$$

3 Partie technique

3.1 Générateur de combats

Le générateur de combat est une pièce importante de notre projet, en effet c'est grâce à ce dernier que nous allons pouvoir exécuter les combats nécessaires pour renforcer notre IA. Le code source de combat est disponible sur GitHub à l'adresse suivante : <https://github.com/leek-wars/leek-wars-generator>. Pour que le générateur fonctionne, il faut également installer le langage Leek Script dans le dossier du même nom : <https://github.com/leek-wars/leekscript>.

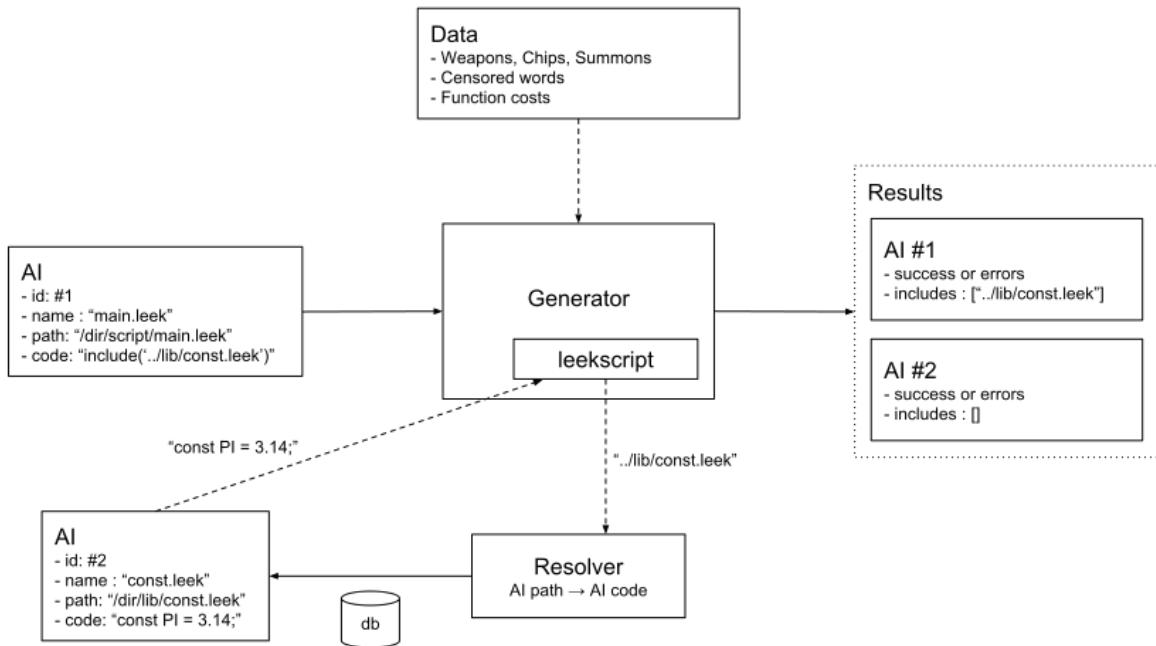


Figure 6 – Générateur de combats

3.2 Cycle d'entraînement

Pour notre sujet, l'apprentissage par renforcement consiste, pour un poireau, à apprendre les actions à prendre, à partir d'expériences, de façon à optimiser une récompense quantitative au cours du temps. Le poireau est plongé au sein d'un combat, et prend ses décisions en fonction de son état courant. En retour, le modèle qu'utilise le poireau est modifié en fonction du résultat des actions exécutées. L'algorithme va attribuer une récompense (positive ou négative) à chaque action. L'objectif était que notre modèle s'améliore au fil des combats.

Pour entraîner notre modèle, l'algorithme de renforcement va répéter le cycle suivant :

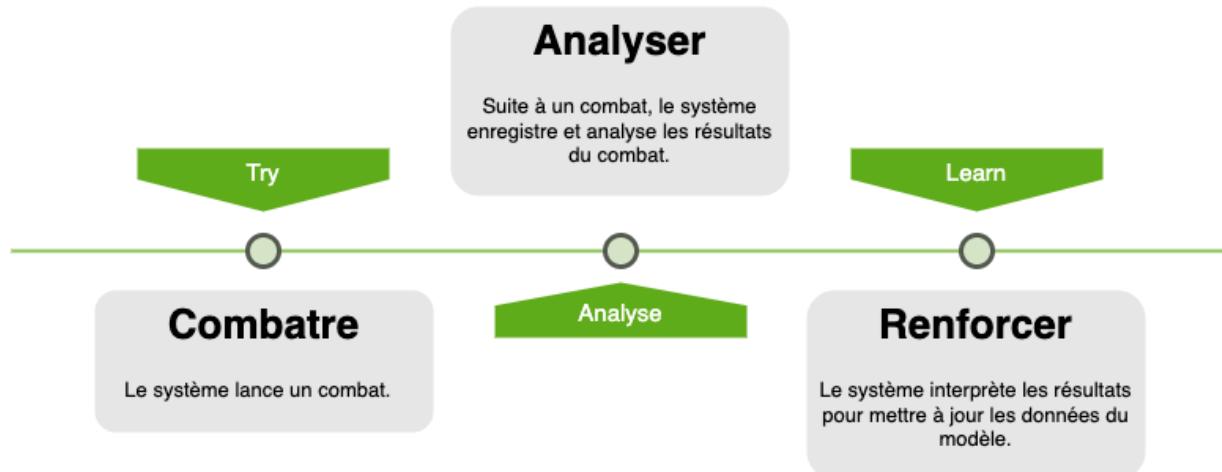


Figure 7 – Cycle de renforcement

3.3 Sauvegarde des modèles

Les données obtenues à la suite d'une série de combat sont enregistrées dans un fichier texte (tab.leek) qui sera utilisé par la suite pour jouer en ligne avec une IA capable d'interpréter la table de renforcement.

Le fichier tab.leek contient deux tableaux d'entier. Le premier tableau d'entier est notre tableau d'association qui représente la booléanisation des états.

Le deuxième tableau représente la table de renforcement, qui enregistre les poids associés à chacune des actions disponible de notre IA. Chaque ligne correspond à un état et chaque colonne à une action. LeekScript étant un langage relativement pauvre au niveau de la manipulation de données, nous utilisons simplement un tableau à une dimension avec un système de modulo pour simuler des lignes.

```
1 var tabRL = [];
2 [1089, 1815, 646, 1103, 2509, 1105, 1275, 2598, 2267, 957,
3 3639, 1082, 618, 1851, 3394, 210, 3118, 2634, 1589, 979,
4 654, 378, 2659, 426, 2969, 2004, 1139, 632, 3663, 1134,
5 #...
6 1893, 744, 3029, 1998, 2945, 3552, 428, 3088, 3230, 3242]
```

Ces deux variables sont la base de notre système. Le style de jeu de notre IA dépend uniquement de la valeur de ces deux dernières.

4 Partie stratégique

Jusqu'à présent nous avions simplement mis en place les algorithmes existant sur le renforcement d'IA, c'est à partir de cette étape que nous avons fait des choix stratégiques sur l'implémentation du renforcement pour pallier des problèmes techniques, pour personnaliser au mieux la méthode de renforcement et ainsi être en adéquation avec notre sujet.

4.1 Stratégie de renforcement du modèle

C'est la stratégie de renforcement du modèle qui définit quand, comment et en quelle quantité notre IA allait recevoir des récompenses pour l'action qu'elle effectue et qui allait définir quel type d'IA qu'elle allait devenir. La récompenser seulement lorsqu'elle ne perd pas de point vie ou lorsqu'elle en gagne en fera naturellement une IA fuyarde avec comme objectif de fuir l'ennemi pour ne prendre aucun dégât. Pour notre stratégie de renforcement, nous avons opté pour une IA agressive qui obtient une récompense que lorsqu'elle a infligé des dégâts et en quantité équivalente au nombre de dégâts infligés.

Pour ce qui est de la partie technique du renforcement, lorsque l'IA tombe dans un nouvel état du système elle crée une nouvelle branche dans sa table de renforcement contenant un tableau de taille de toutes les actions possibles de l'IA, et chaque index représente une action. A la création de la branche, toutes les actions ayant la même chance d'être choisi, tous les éléments du tableau sont initialisés à 1. L'IA va ensuite choisir un index aléatoire du tableau et va exécuter l'action qui lui est associé, suite au résultat de l'action, la valeur de l'élément se trouvant à l'index de l'action réalisée va être incrémentée avec la récompense donnée à l'IA. Si l'IA se retrouve sur la même configuration du système, l'IA ne créera pas de branche mais ira voir le tableau qui correspond à l'état du système et cette fois-ci tira au sort parmi les index du tableau mais avec une pondération en fonction des valeurs des index, de ce fait, une action ayant été récompensée auparavant aura plus de chances d'être choisie à nouveau.

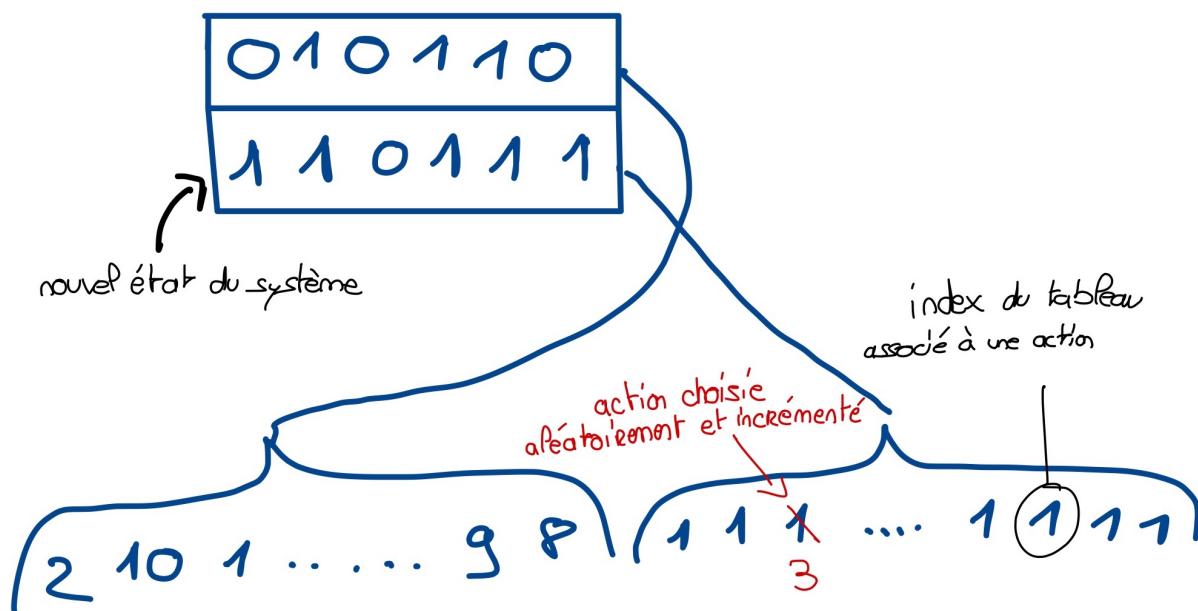


Figure 8 – Schéma de la stratégie de renforcement

Nous avons réalisé pendant nos tests que notre stratégie n'était pas optimale, effectivement les dégâts moyens dans le jeu étant de 100 points de dégâts, si lorsque nous incrémentions une action par rapport aux dégâts qu'elle a infligés sur l'adversaire, l'incrémantation et donc la pondération appliquée était trop violente, et la meilleure action n'était peut-être pas être choisi. Notre IA s'enfouffrait souvent dans un choix non optimal, pour pallier ce problème nous avons tout simplement appliqué une division par 10 entre les dégâts infligés à l'ennemi et la récompense donnée à l'IA.

4.2 Stratégie de sélection des actions

Suite à l'entraînement de notre IA lors de la mise en situation réelle contre de vrai joueurs, la sélection d'une action se fait simplement en prenant l'action liée à l'index du tableau qui possède la plus grande valeur, le tableau est bien entendu celui qui est associé à l'état du système. Il se peut que l'état du système dans lequel se trouve notre IA durant un combat ne soit jamais apparu pendant tout l'entraînement de notre IA, pour choisir l'action la plus optimale à prendre, l'IA va trouver l'état du système connu qui se rapproche le plus de celui actuel et va continuer son exécution comme si c'était celle-ci.

4.3 Stratégie d'entraînement

Nous avons imaginé deux stratégies principales pour entraîner notre modèle. La première consiste à confronter une IA équipée d'un modèle vierge et de la faire s'entraîner contre une IA codée par un humain. Le but était que cette dernière devienne meilleure que l'IA codée par l'humain. La deuxième méthode est celle imaginée avec notre encadrant M. Lafourcade. Cette méthode va confronter une IA équipée d'un modèle vierge face à elle-même. Nous allons voir plus en détails les deux scénarios de chaque stratégie.

4.3.1 Entraînement contre une IA existante

Dans le scénario de cette méthode, la première entité est l'IA contenant le modèle et la deuxième entité est le poireau équipé de l'IA contre laquelle le premier poireau va se battre et renforcer son modèle. Ci-dessous un extrait du fichier JSON est contenant cette partie du code :

```
1 {
2     "..."
3
4     "entities": [
5         [
6             {
7                 "..."
8                 "name": "IA1",
9                 "level": 301,
10                "ai": "test/ai/IA-Train.leek",
11                "cell": 12,
12                "id": 1,
13                "..."
14            }
15        ],
16        [
17            {
18                "..."
19                "name": "IA2",
20                "level": 301,
21                "ai": "test/ai/BOT-Train.leek",
22                "cell": 52,
23                "id": 2,
24                "..."
25            }
26        ]
27    "..."
28 }
29 }
```

4.3.2 Entraînement automatique

On peut remarquer que dans l'extrait JSON suivant les deux entités sont équipées de la même IA qui utilise le même modèle. Cette stratégie permet de renforcer deux fois plus vite car l'on interprète les deux fichiers de résultats pour renforcer le même modèle.

```
1 {
2     "..."
3
4     "entities": [
5         [
6             {
7                 ...
8                 "name": "IA1",
9                 "level": 301,
10                "ai": "test/ai/IA-Train.leek",
11                "cell": 12,
12                "id": 1,
13                ...
14            }
15        ],
16        [
17            {
18                ...
19                "name": "IA2",
20                "level": 301,
21                "ai": "test/ai/IA-Train.leek",
22                "cell": 52,
23                "id": 2,
24                ...
25            }
26        ]
27    ],
28    ...
29 }
```

L'avantage de cette méthode est que l'on renforce deux fois le même modèle ce qui permet de converger plus rapide vers un modèle efficace. L'inconvénient est que notre IA n'est jamais confrontée à une IA humain et de ce fait elle laisse des vulnérabilités qu'un humain peut remarquer en regardant quelques combats.

5 Problèmes rencontrés

Nous avons rencontré des problèmes sur la taille du fichier leekScript. LeekWars bride les fichiers par un nombre maximum de caractères. Cette situation nous a contraints à tester des combats avec une IA peu entraînée sur LeekWars.

Lors de la phase

6 Résultats

Suite à l'entraînement de notre IA nous l'avons testé sur différentes IA proposées par le jeu, et malheureusement elle n'arrivait à gagner contre aucune d'entre elles. En regardant les combats plus précisément, nous avons remarqué que notre IA ne faisait pas beaucoup d'action sensée. Cela est dû au problème cité précédemment, effectivement étant grandement limité dans la taille du fichier leekScript, seulement 140 états peuvent être définis, or un état est défini par 30 bits, le nombre d'états possible est donc de $30^2 = 900$. Déjà que nous avons dû grandement réduire la taille de la définition d'état pour éviter d'avoir trop d'état du système différent, n'avoir que si peu d'état possible parmi ce qu'il reste rend très problématique l'entraînement de l'IA.

7 Conclusion

Du fait de problèmes techniques expliqués précédemment notre IA n'a pas pu atteindre nos objectifs, tout de même, le développement d'une intelligence en machine learning sur un langage sans aucune bibliothèque, ne pas juste utiliser une fonction sans la comprendre comme sur un basique code python de machine learning nous a permis de comprendre les rouages des intelligences artificielles, de réaliser toute la complexité de celle-ci. Nous avons compris à quel point les réglages initialaux d'une IA peuvent modifier leur résultat à l'entraînement. Faire face à des problèmes techniques de taille et à trouver des solutions pour les résoudre. Pour conclure, d'un point de vue algorithmique nous sommes tout a fait ravie de la phase d'apprentissage de notre IA malgré le fait que celle-ci n'est pas performante, certes notre IA n'est pas capable de gagner des combats mais tous les mécanismes d'apprentissage sont fonctionnels et le seul frein à notre réussite est une barrière créée par le développeur du jeu qui hors de notre portée.