Intelligence artificielle et programmation Python

publié le 12/04/2022 - mis à jour le 16/06/2022

TraAM 2021 - 2022

Descriptif :

Scénario Traam faisant suite à l'expérimentation d'un jeu pour découvrir l'intelligence artificielle, le but étant d'aborder l'IA sous l'angle de la programmation afin de mieux rendre compte de la notion d'apprentissage automatique (machine learning)

Sommaire :

Sommaire des TraAms "Intelligence Artificielle"

Cet article fait suite à un premier article rendant compte du début de l'expérimentation qu'il est utile de consulter afin de bien saisir l'ensemble de la démarche.

Découvrir l'intelligence artificielle à partir d'un jeu .

Fiche synoptique

Thématique

Apprentissage automatique : découvrir l'apprentissage par renforcement

Niveau concerné

Seconde

Problématique

Popularisée par les matchs Deep Blue contre Kasparov à l'aube du 21ème siècle, la rivalité entre machine et être humain dans le domaine des jeux a toujours alimenté les fantasmes de notre inconscient collectif.
Depuis quelques années, les intelligences artificielles surpassent l'être humain lors de confrontations sur des jeux de stratégies comme les échecs ou le jeu de go et les récents progrès utilisant en particulier les réseaux neuronaux et l'apprentissage profond interrogent sur les modèles qui ont permis ces avancées remarquables.
Plus modestement, dans un contexte pédagogique de classe, il peut paraître pertinent de se poser les questions suivantes :

Comment une machine peut-elle apprendre à gagner contre un humain ?

Peut-on programmer une forme d'intelligence artificielle avec les élèves ?

Contenu

- statistiques, probabilités
- échantillonnage
- représentations graphiques : arbres, schémas
- algorithmiques et programmation Python

Nombre d'heures utilisées

Environ 2 heures

Outils et ressources

- article initial de Martin Gardner ;
- Animation du Palais de la Découverte : article du site et notice de jeu
- Article de Bastien Masse sur le jeu Hexapawn
- Article de la médiathèque de Riom Limagne et Volcans
- Vidéo de démonstration de l'expérience par Vsauce2 : The Game That Learns
- pour jouer en ligne contre l'IA (animation proposée par William laidet, membre du groupe Traam) : http://tableauxmaths.fr/SNT/Hexapawn/

Page suivante : "*Narration de l'expérimentation*"

Narration de l'expérimentation

Intentions

Après avoir mis en évidence le phénomène d'apprentissage à l'aide de l'arbre de jeu élagué avec le tableur, la suite de l'expérimentation a consisté en une mise en œuvre algorithmique et une tentative de programmation de l'IA papier de Martin Gardner.

Les objectifs de la séance étaient les suivants :

- présenter la modélisation du jeu Hexapawn en langage Python (représenté par une liste de listes ici) ;
- leur faire compléter le cas d'une partie entre deux joueurs aléatoires, simuler celle-ci un grand nombre de fois afin de retrouver la fréquence de victoire de chaque couleur établie de manière empirique lors de la découverte du jeu, et de manière théorique avec l'étude de l'arbre de jeu
- leur faire manipuler la fonction de jeu d'intelligence artificielle pour réaliser plusieurs affichages graphiques afin de mesurer la qualité de l'apprentissage en fonction du retour informatif choisi (punition ou récompense)
- leur faire trouver, à partir de la liste des parties gagnantes pour l'IA, une stratégie gagnante d'Hexapawn qui est un jeu résolu pour les noirs.

La complexité de la modélisation en langage Python (plusieurs dizaines de lignes de code), les contraintes de temps et de programme (les listes Python ne sont pas au programme de seconde) ont fortement limité la part de programmation dévolue aux élèves.

Il a donc fallu tenir compte de leur niveau en programmation et leur donner suffisamment de tâches à réaliser pour que la séance demeure intéressante et atteigne ses objectifs.

Cette séance s'est déroulée sur une heure de cours en demi-classe, en salle informatique, sur un support de type notebook dans l'environnement Capytale (code d'accès : e4e6-513443).

Version pdf du notebook élève (PDF de 918.4 ko)
Notebook Capytale converti au format pdf

Modélisation du jeu

Le plateau de jeu a été modélisé sous la forme d'une liste de listes, le plateau initial étant défini par `plateau = [["N","N","N"],[" "," "," "],["B","B","B"]]`.

Après quelques explications sur le repérage des cases, je leur ai présenté les fonctions informatiques intégrant les règles du jeu et permettant la réalisation d'une partie complète. Quelques exemples, insérés dans des cellules à exécuter, étaient répartis dans le notebook afin d'illustrer mes propos et favoriser la représentation du jeu dans la machine.

Cette partie "magistrale" a permis de rappeler le fonctionnement du jeu et a mis en évidence la complexité que peut représenter la conversion d'un jeu d'apparence simple en programme informatique.

Partie entre deux joueurs aléatoires

La fonction de simulation d'une partie `partie_alea_vs_alea()` a été présentée et les élèves devaient compléter la portion de code relative aux tours des noirs, en faisant l'analogie avec la même portion de code relative aux blancs :

```
import random
def partie_alea_vs_alea():
    """renvoie une partie aléatoire entre deux joueurs qui jouent au hasard"""
    tour = 0
    plateau = [["N","N","N"],[" "," "," "],["B","B","B"]]
    vainqueur = ""
    partie_en_cours = True
    while partie_en_cours == True:
        if tour % 2 == 0: # au tour des blancs de jouer
            coup_choisi= random.choice(coups_possibles(plateau,"B")) # choix d'un coup au hasard
            plateau= plateau_joue(plateau, 'B', coup_choisi) # coup joué
            if fin_de_partie(plateau, "B") == True: # vérification d'une éventuelle victoire des blancs
                partie_en_cours= False # arrêt de la partie
                vainqueur= "B" # affectation du nom du vainqueur dans la variable vainqueur
        if tour % 2 == 1: # au tour des noirs de jouer
            ...
            ...
            ...
            ...
            ...
        tour = tour + 1 # on passe au tour suivant
    pass #return vainqueur # remplacer pass par l'instruction mise en commentaire
```

Cette tâche n'a pas posé de problème, les élèves n'ayant qu'à faire un copier/coller de code et modifier le nom de la couleur, mais des erreurs d'indentation ont tout de même été observées, notamment dans la place du `return` qui a souvent été aligné avec les instructions de la boucle.

Il était ensuite demandé aux élèves de construire une fonction de simulation prenant en paramètre le nombre de parties et renvoyant la fréquence de victoires de chaque couleur.

La fonction était en partie construite et les élèves devaient compléter, avec une plus grande autonomie que précédemment, le corps de la fonction qui contenait deux structures conditionnelles symétriques, avec incrémentation d'un compteur.

```
def echantillon(nb_parties):
    """renvoie la fréquence de victoire de chaque couleur après simulation de nb_parties parties"""
    victoires_blancs = 0
    victoires_noirs = 0
    for _ in range(nb_parties):
        ...
        ...
        ...
        ...
    pass # remplacer l'instruction pass par une instruction return ....
```

Après lecture attentive des spécifications de l'algorithme, la plupart des élèves a réussi à compléter la fonction. Les écueils se sont situés à plusieurs niveaux et on retrouve les erreurs classiques inhérentes à l'apprentissage de Python :

- erreurs dans l'orthographe des variables : oubli de "s" dans les variables `victoires_blancs` et `victoires_noirs` ;
- confusion entre opérateur d'affectation `=` et opérateur de comparaison `==` ;
- indentation des blocs de structure conditionnelle non respectée ;
- oubli des deux points terminant l'en-tête des blocs de structure conditionnelle ;
- difficulté à traduire l'incrémentation des compteurs en `victoires_blancs = victoires_blancs + 1`

L’appel de cette fonction a permis de mettre en évidence la notion de fluctuation d’échantillonnage, les élèves m’interpelant en me disant qu’ils n’avaient pas la même fréquence que moi au tableau ou que leur voisin(e).

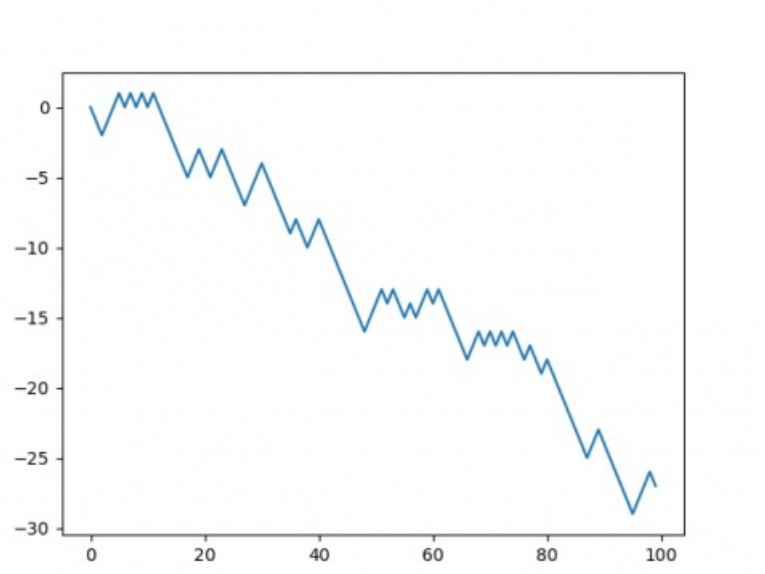
Après leur avoir fait estimer oralement la dispersion des valeurs, je leur ai demandé d’augmenter progressivement la taille de l’échantillon, jusqu’à 15 000 tirages et ils ont pu constater, en comparant leurs valeurs, la stabilisation des fréquences, et percevoir, sous une forme expérimentale, la loi des grands nombres.

J’en ai alors profité pour leur rappeler que le modèle, c’est-à-dire l’arbre de jeu que nous avions étudié dans les séances précédentes donnait les probabilités de $\frac{259}{432}$ et $\frac{173}{432}$ et que les variations mesurées étaient de l’ordre de quelques millièmes pour les échantillons de taille 15 000. Nous avons ainsi validé le principe d’estimation d’une probabilité par une fréquence observée sur un échantillon.

Afin de mieux visualiser l’évolution des victoires de chaque couleur, un diagramme à ligne brisée a été proposé et a permis de mieux saisir la tendance pour le cumul des victoires. Celui-ci était construit selon le principe suivant :

- pour une victoire des **blancs**, on se décale d’une unité vers la droite et d’une unité vers le **bas** ;
- pour une victoire des **noirs**, on se décale d’une unité vers la droite et d’une unité vers le **haut** ;

Le cumul progressif des victoires blanches était alors caractérisé par une tendance baissière de la courbe :

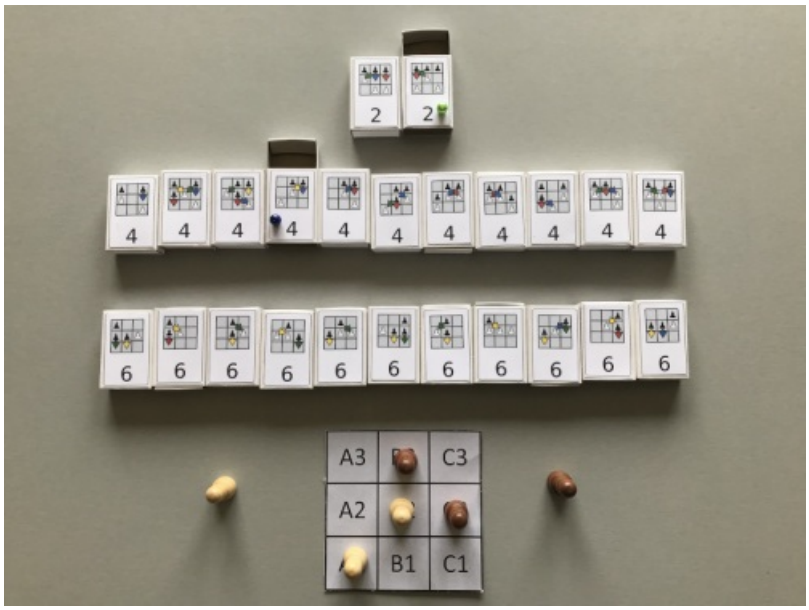


Graphique illustrant l’évolution des victoires dans le cas des joueurs aléatoires (blancs vers le bas, noirs vers le haut)

Page suivante : "*Narration de l’expérimentation : Joueur aléatoire contre une IA*"

Joueur aléatoire contre une IA

Historiquement, le jeu d’Hexapawn avait été créé par Martin Gardner pour simuler une intelligence artificielle en papier, composée de 24 boîtes d’allumettes contenant des perles de couleur.



IA papier de Martin Gardner avec les 24 boîtes d'allumettes.

Chaque boîte correspond à l'une des situations du plateau de jeu (ou son symétrique) et les couleurs aux différents coups que la machine peut réaliser dans cette situation.

Dans sa configuration initiale (celle que nous avons modélisée dans le tableur), lorsque c'est au tour de la machine de jouer, elle tire une perle au hasard dans la boîte correspondant à la situation et qui lui indique le coup qu'elle doit jouer. Cette perle est mise de côté et, à la fin de la partie, la règle est la suivante :

- si la machine a gagné, toutes les perles sont remises dans leurs boîtes ;
- si elle a perdu, la perle qui l'a fait perdre est éliminée et le coup associé ne peut plus être joué

En multipliant les parties, le retrait progressif des perles perdantes va permettre à la machine d'apprendre les bons coups en élaguant l'arbre de jeu pour ne conserver que les branches gagnantes.

Cette méthode d'apprentissage se base sur un système de *punition* : si l'IA perd, elle est punie (confiscation de perles) de façon à ce qu'elle ne reproduise plus l'erreur qu'elle a commise pour perdre. À force elle ne commet plus d'erreur.

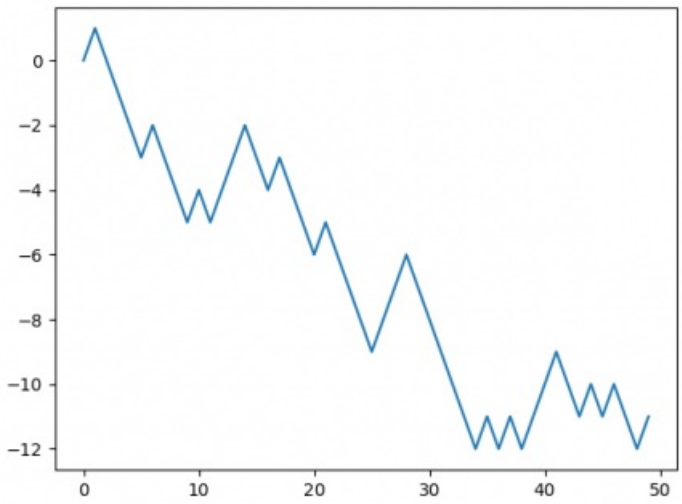
Mais on peut aussi envisager de procéder par *récompense*. Lorsque que l'IA gagne, on rajoute une ou plusieurs perles de la couleur du coup gagnant dans la dernière boîte (on aurait aussi pu rajouter des perles dans les autres boîtes sélectionnées lors de la partie). Ainsi, l'IA a plus de chance de reproduire ce coup gagnant plus tard car la probabilité de tomber sur cette perle gagnante est augmentée.

Dans cette partie les élèves ont lu avec moi le code de la fonction `sequence_alea_vs_ia`, afin de saisir comment les récompenses et punitions étaient gérées d'un point de vue informatique.

Cette fonction prend en paramètre un nombre de parties et dispose de deux paramètres optionnels permettant la gestion de l'apprentissage : `punition`, réglée initialement à `True` et `recompense`, réglée initialement à `False`, afin d'être en accord avec le modèle original.

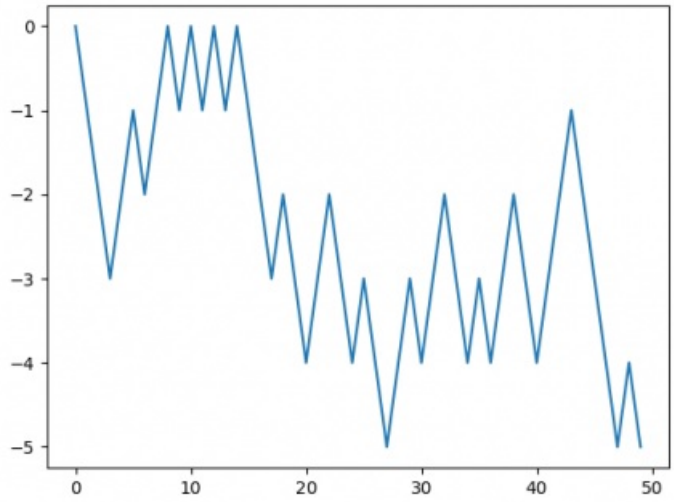
Ensuite, ils on fait des appels de cette fonction et de la fonction `graphique` pour mesurer l'effet de la méthode d'apprentissage sur la courbe des victoires :

- `punition = False` et `recompense = False` : on retrouve la partie aléatoire à deux joueurs, il n'y a pas d'apprentissage et les victoires des blancs se cumulent :

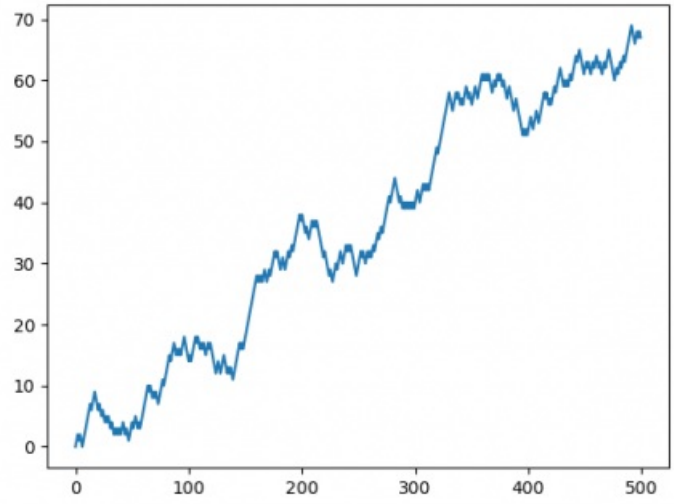


Courbe d'apprentissage en l'absence de retour informatif

- `punition = False` et `recompense = True`. Sur un nombre limité de parties, l'apprentissage n'est pas flagrant. En revanche, dès que le nombre de parties devient important, on note l'évolution des victoires des noirs :



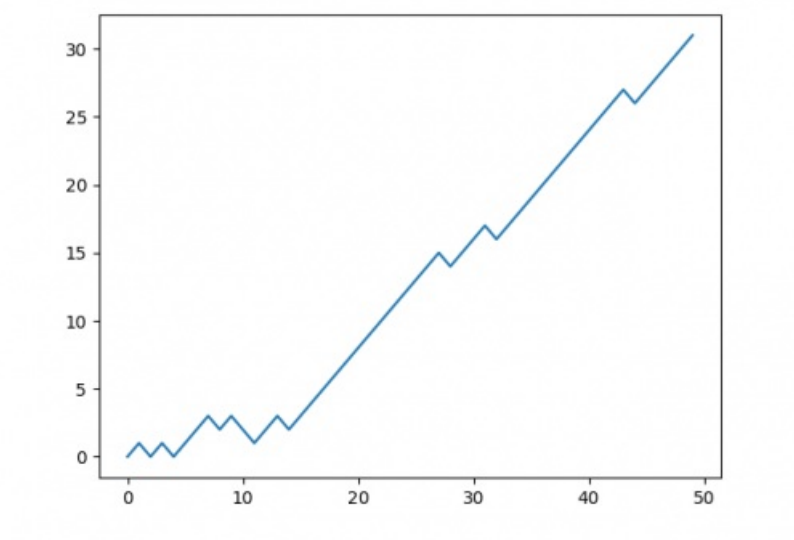
Sans punition et avec récompense, petit nombre de parties



Sans punition et avec récompense, grand nombre de parties

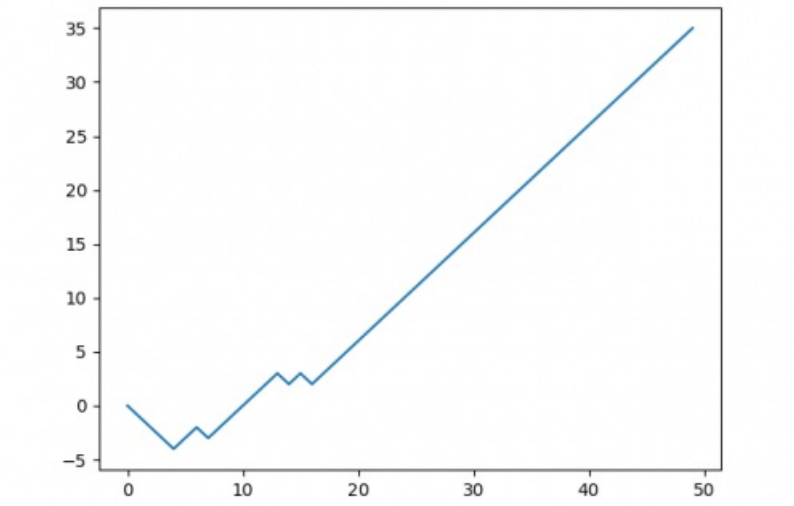
- `punition = True` et `recompense = False`. C'est la situation de la machine de Gardner. On note un certain

nombre de défaites (une dizaine), avant de voir des victoires quasiment systématiques :



Avec punition et sans récompense : conditions de la machine de Gardner

- `punition = True` et `recompense = True`. Le cumul des deux systèmes n'est pas forcément beaucoup plus efficace, peut-être la machine gagne-t-elle plus vite et les défaites disparaissent complètement plus tôt mais ce n'est pas probant avec les paramètres choisis (j'avais choisi de rajouter trois fois le mouvement gagnant dans la dernière "boîte").



Avec punition et avec récompense : combinaison des deux feedbacks.

D'un point de vue du déroulement en classe, cette séquence s'est bien déroulée et les élèves ont bien apprécié de faire varier les différents paramètres de la fonction afin d'en retirer les conclusions attendues.

Prolongement : recherche d'une stratégie gagnante

Cette partie n'a pas été faite en classe. Elle consiste à utiliser une dernière fonction listant les parties gagnantes pour la machine et à se servir des résultats renvoyés pour rédiger une stratégie gagnante au jeu d'Hexapawn qui est résolu pour les noirs.

L'utilisation de cette fonction, avec les différents paramètres de feedbacks, permet alors de faire émerger une trentaine de parties types qui illustrent les meilleurs choix de l'IA en fonction du choix des blancs à chaque tour.

Une rédaction de la stratégie gagnante est alors possible. Cette partie a été donnée en devoir à la maison aux élèves afin de conclure l'expérimentation.

Version pdf du notebook professeur (PDF de 1 Mo)

Page suivante : "*Réussites, freins et limites*"

Réussites, obstacles et limites

L'activité en elle-même s'est bien déroulée mais le manque de temps pour l'apprentissage de la programmation et les limitations du programme de seconde dans ce domaine ont nettement réduit l'ambition initiale qui était de faire programmer cette forme d'IA par les élèves.
En effet, enseigner la programmation demande du temps : l'acquisition d'un nouveau langage nécessite de nombreuses heures de manipulation et il est indispensable de proposer des travaux pratiques très réguliers afin d'entretenir un niveau de maitrise qui permette aux élèves d'être autonomes dans ce langage.
Malgré un nombre important d'heures passées en salle informatique dans la période qui précédait l'expérimentation, mes élèves ne maitrisaient pas assez les concepts algorithmiques et les fondamentaux du langage Python pour envisager de leur laisser davantage de code à produire.
Une plus grande anticipation, avec une initiation au langage très tôt dans l'année et une répartition plus étalée des apprentissages, aurait sans doute autorisé une ambition plus haute.
Cependant, la composante graphique de l'activité a permis de mettre en évidence l'effet de l'apprentissage en comparant les parties aléatoires et les parties avec intelligence artificielle et en faisant varier les options récompense/punition. Les remarques et observations des élèves m'ont convaincu que le concept de "machine qui apprend" a été saisi.

Document joint

Ressources de l'expérimentation (Zip de 8.1 Mo)

Cette archive contient l'intégralité des fichiers sources des documents (.tex, .py, .ipynb, .ods) utilisés pour la mise en œuvre de l'expérimentation