Apprendre à une machine à jouer... sans ordinateur !

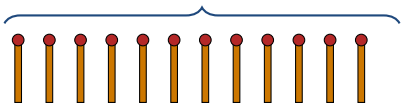
Mise en situation. On parle beaucoup d'intelligence artificielle (IA) dans les médias : des ordinateurs "intelligents" qui apprendraient tout seuls, parfois présentés comme inquiétants. Mais en réalité, une IA n'est jamais qu'une machine qui répète des opérations très simples, des millions de fois, pour finir par trouver une bonne stratégie – sans jamais "comprendre" ce qu'elle fait.
Dans cette activité, vous allez fabriquer une machine capable d'apprendre, **toute seule et sans ordinateur**, à devenir imbattable au **jeu de Nim**. Vous découvrirez ensuite comment on peut traduire cette idée en quelques lignes de code Python.

1. Les règles du jeu de Nim

Définition

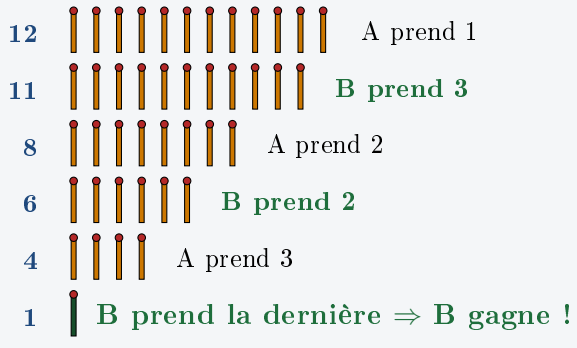
On dispose de 12 allumettes sur la table. Deux joueurs s'affrontent et jouent chacun à leur tour. À chaque tour, un joueur retire **1, 2 ou 3 allumettes** (à son choix). Le joueur qui prend la **dernière allumette** a **gagné** (celui qui ne peut plus jouer, faute d'allumette, a donc perdu).

12 allumettes de départ



Exemple

Exemple de partie. A commence avec 12 allumettes. B connaît la stratégie (laisser un multiple de 4 à l'adversaire).



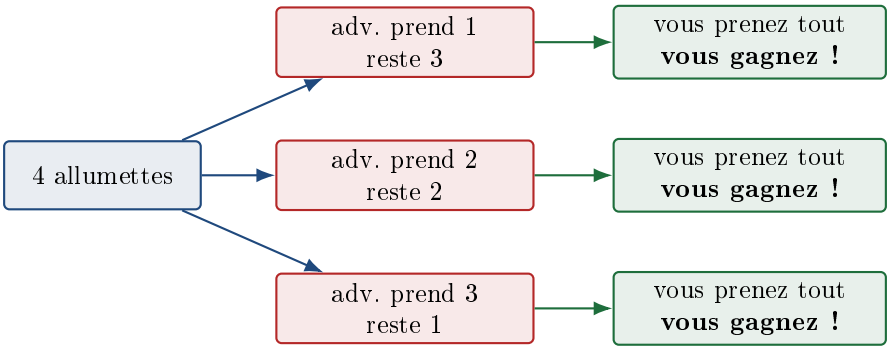
▶ À vous de jouer !

Jouez plusieurs parties de Nim par deux, avec 12 allumettes (ou 12 jetons, ou 12 gommes...). Le joueur A commence.

1. Essayez de repérer une "bonne" façon de jouer. Y a-t-il un nombre d'allumettes restantes qui vous semble dangereux à laisser à l'adversaire ?
2. Recommencez en laissant volontairement **4 allumettes** à votre adversaire. Que se passe-t-il, quel que soit son choix (1, 2 ou 3) ?

Coup de pouce

Si vous laissez exactement 4 allumettes à l'adversaire, quel que soit son choix (il prend 1, 2 ou 3), il vous reste toujours 1, 2 ou 3 allumettes – vous pouvez alors prendre le reste et gagner !

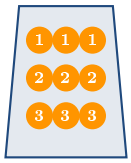


2. Une machine qui apprend en perdant

Nous allons maintenant fabriquer une machine (sans ordinateur !) capable d'apprendre à jouer, uniquement par essais et erreurs.

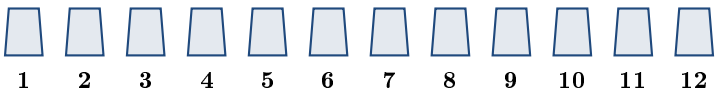
Définition

Matériel : 12 gobelets numérotés de 1 à 12, et un grand nombre de jetons. Dans **chaque** gobelet, on place **3 jetons "1"**, **3 jetons "2"** et **3 jetons "3"** (soit 9 jetons par gobelet), quel que soit le numéro du gobelet.



gobelet

Chaque gobelet (numéros 1 à 12) contient exactement 3 jetons "1", 3 jetons "2" et 3 jetons "3", quelle que soit sa position.

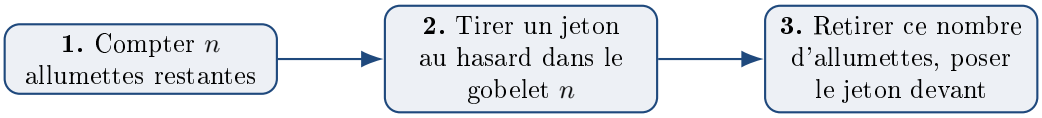


Les 12 gobelets numérotés, tous identiques au départ de l'apprentissage.

Exemple

Déroulement d'une partie. Un humain joue contre la machine. L'humain commence et retire 1, 2 ou 3 allumettes. Puis, à son tour, la machine :

1. compte le nombre n d'allumettes restantes sur la table ;
2. secoue le gobelet numéro n et tire un jeton au hasard ;
3. retire le nombre d'allumettes indiqué sur le jeton, et pose ce jeton devant le gobelet correspondant (pour se souvenir du coup joué).



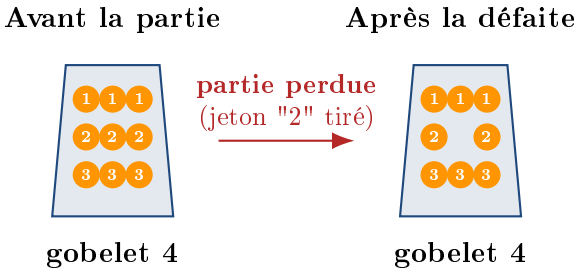
Définition

La règle d'apprentissage. Pendant la partie, la machine note à chaque tour le gobelet utilisé et le jeton tiré. À la fin de la partie :

- si la machine **gagne**, elle remet **tous** les jetons tirés pendant la partie dans leurs gobelets d'origine ;
- si la machine **perd**, elle ne remet **aucun** des jetons tirés pendant cette partie : ils sont définitivement écartés.

Remarque

Si le jeton tiré indique un nombre d'allumettes supérieur à ce qu'il reste sur la table (par exemple tirer "3" alors qu'il ne reste que 2 allumettes), la machine prend simplement toutes les allumettes restantes.



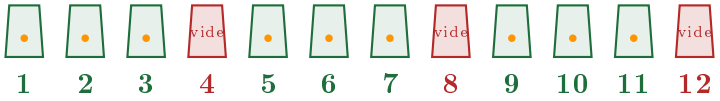
Un seul jeton "2" a été tiré du gobelet 4 pendant cette partie perdue : il n'est pas remis (les jetons des autres gobelets utilisés pendant la même partie sont eux aussi écartés).

▶ À vous de jouer !

Par groupes, fabriquez le matériel et jouez plusieurs parties contre la "machine" (l'un de vous joue le rôle de la machine, en suivant strictement les règles ci-dessus, sans réfléchir stratégiquement !).

1. Combien de parties avez-vous dû jouer avant que la machine ne se mette à gagner systématiquement ?
2. Que se passe-t-il, au fil des parties, dans le gobelet numéro 4 ? Pourquoi, à votre avis ?
3. Une fois la machine "entraînée", regardez les jetons restants dans chaque gobelet. Retrouvez-vous la stratégie que vous aviez identifiée dans l'activité précédente ?

État attendu des gobelets après un long entraînement



Les gobelets 4, 8 et 12 correspondent à des positions perdantes : quel que soit le jeton tiré, la machine finit toujours par perdre lorsqu'elle y passe. Au fil des parties, tous leurs jetons sont donc progressivement écartés.

Remarque

La machine ne "comprend" jamais les règles du jeu : elle ne fait que répéter un geste mécanique (tirer un jeton au hasard) et une punition simple (retirer un jeton après une défaite). Pourtant, après quelques parties, elle devient **imbattable**. C'est exactement le principe de l'**apprentissage par renforcement**, une des méthodes utilisées par certaines IA modernes (par exemple pour apprendre à jouer aux échecs ou au jeu de Go).

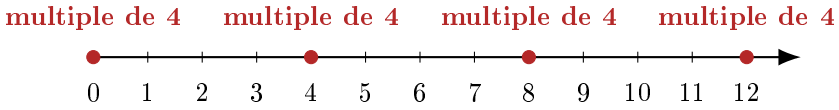
3. Passons au code : simuler la stratégie en Python

On ne va pas coder l'apprentissage complet (il faudrait des listes, qui ne sont pas encore au programme), mais on peut coder **la stratégie gagnante** une fois qu'elle est découverte, et faire jouer l'IA contre vous.

Exemple

Rappel de la stratégie gagnante. D'après l'activité précédente, un joueur gagne à coup sûr s'il parvient à laisser à son adversaire un nombre d'allumettes multiple de 4 (4, 8 ou 12). En Python, on teste si un nombre n est un multiple de 4 grâce à l'opérateur **modulo %**, qui donne le **reste** d'une division euclidienne :

```
n = 9
reste = n % 4
print(reste)  # affiche 1, car 9 = 2*4+1
```



Les nombres "dangereux" à laisser à l'adversaire sont les multiples de 4 : 4, 8 et 12.

▶ À vous de jouer !

1. Sans exécuter le code, donner la valeur affichée pour `n = 11`, puis pour `n = 12`.
2. Modifier le code pour tester si `n` est un multiple de 4 (c'est-à-dire si `reste` est nul), en utilisant une structure `if`.

Coup de pouce

En Python, l'égalité se teste avec le symbole `==` (et non `=`, réservé à l'affectation d'une variable) :

```
if reste == 0:
    print("multiple de 4")
```

4. Coder l'IA "joueur parfait"

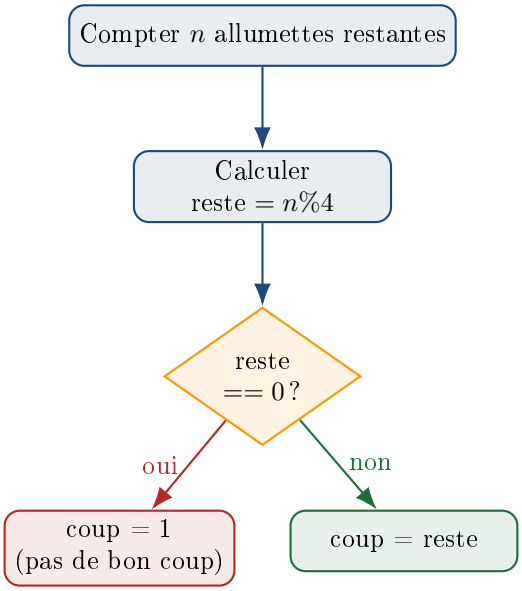
Exemple

Voici un programme complet où l'**IA choisit toujours le bon coup**. À chaque tour, elle calcule le reste de la division du nombre d'allumettes restantes par 4, et en déduit combien de jetons elle doit prendre pour ramener ce reste à zéro pour son adversaire.

```
n = 12  # nombre d'allumettes au depart
print("Il reste", n, "allumettes.")

# --- Tour de l'IA ---
reste = n % 4
if reste == 0:
    coup = 1  # aucun bon coup, l'IA va perdre : elle joue 1
else:
    coup = reste

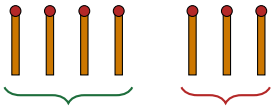
print("L'IA prend", coup, "allumette(s).")
n = n - coup
print("Il reste", n, "allumettes.")
```



▶ À vous de jouer !

1. Faire fonctionner ce programme "à la main" en supposant que $n = 12$ au début (comme dans le code). Combien d'allumettes l'IA prend-elle ? Combien en reste-t-il ?
2. Reprendre le calcul en supposant cette fois que $n = 7$ au début du tour de l'IA. Combien d'allumettes doit-elle prendre ?
3. Pourquoi le programme joue-t-il "coup = 1" lorsque le reste est nul ? Que peut-on en déduire sur les chances de victoire de l'IA dans ce cas précis ?

Cas $n = 7$



4 allumettes laissées **coup = 3**

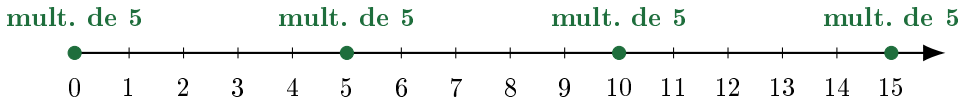
Remarque

Ce programme ne "joue" qu'un seul tour. Pour simuler une partie complète (plusieurs tours, en alternant avec un joueur humain qui entre son coup au clavier avec `input()`), il faudrait une boucle `while n >0:`. C'est une piste d'approfondissement possible si le temps le permet !

5. Pour aller plus loin (facultatif)

Définition

Et si on changeait le nombre d'allumettes de départ ? La stratégie "viser un multiple de 4" dépend du fait qu'on peut retirer 1, 2 ou 3 allumettes à chaque tour. Si les règles changent (par exemple, on peut retirer 1, 2, 3 **ou 4** allumettes), le nombre "dangereux" à laisser à l'adversaire change aussi.



Avec 1 à 4 allumettes prenables par tour, les nombres dangereux deviennent les multiples de 5.

▶ À vous de jouer !

1. Si l'on peut désormais retirer 1, 2, 3 ou 4 allumettes à chaque tour, quel est le nouveau nombre "magique" (multiple de quel nombre) qu'il faut laisser à l'adversaire pour gagner à coup sûr ?
2. Modifier le programme de la partie 4 pour qu'il fonctionne avec cette nouvelle règle.

Coup de pouce

Avec un maximum de k allumettes prenables par tour, le nombre magique devient un multiple de $k + 1$. Dans le code, il suffit de remplacer tous les `4` par `5` !