TD : Trolls et châteaux

I. Un défi en dimension 1

L'objectif de ce TP est vous initier de façon ludique à la notion d'intelligence artificielle. Pour cela, votre travail sera dédié à une noble tâche : défendre votre château contre un troll !

a. Deux châteaux et un troll

Le jeu « Trolls et châteaux » est un jeu à deux joueurs qui se joue au tour-par-tour. Chaque joueur incarne le seigneur d'un château et doit faire son possible pour éloigner un troll en maraude de ses terres.

Au début de la partie, le plateau de jeu est constitué des éléments suivants :

- Deux châteaux
- Un chemin rectiligne entre ces deux châteaux, divisé en un nombre impair de cases
- Un troll situé au milieu de ce chemin (sur la case centrale)

b. Comment éloigner un troll

Pour se défendre contre les trolls, chaque château est équipé d'une catapulte et d'une certaine réserve de pierre.

A chaque tour de jeu, chaque joueur décide secrètement du nombre de pierres qu'il souhaite lancer sur le troll. Lorsque chacun a fait son choix, on résout le tour de jeu de la façon suivante :

- Si les deux joueurs lancent le même nombre de pierres, le troll ne bouge pas
- Si un joueur lance plus de pierres que l'autre, le troll se déplace d'une case en direction du joueur qui a lancé le moins de pierre à ce tour

Remarques :

- Les deux châteaux commencent avec le même nombre de pierres dans leurs réserves
- Il n'est pas possible d'ajouter des pierres à cette réserve
- A chaque tour, chaque joueur lance au moins une pierre
- Il n'est pas possible de lancer plus de pierres qu'il n'y en a dans la réserve

c. Fin de partie

Il existe deux conditions qui entraînent la fin de la partie :

- Le troll atteint l'un des châteaux
- L'un des joueurs n'a plus de pierres

Condition 1 : Le troll atteint l'un des châteaux

Si le troll atteint l'un des châteaux, le seigneur de ce château perd instantanément la partie (même s'il lui reste des pierres).

Condition 2 : L'un des joueurs n'a plus de pierres

Si l'un des joueurs n'a plus de pierres, la partie s'arrête, et le troll est déplacé d'autant de cases qu'il reste de pierres au joueur adverse (un peu comme si le joueur adverse lançait ses pierres une à une et provoquait à chaque fois le déplacement du troll).

Une fois ce déplacement effectué, deux cas de figure sont possibles :

- Si le troll est à égale distance des deux châteaux (case centrale du chemin), il y a match nul
- Si le troll est plus proche d'un château que de l'autre, il se dirige vers le château le plus proche et le joueur concerné perd la partie

d. Avis aux amateurs de chocolats

Votre travail dans le cadre de ce TD est de créer une intelligence artificielle capable de déterminer à chaque tour le nombre de pierres à envoyer sur le troll.

Chaque IA sera confrontée aux autres dans diverses simulations afin de déterminer quelle est la plus efficace pour chasser le troll. Il vous est bien entendu possible d'envoyer plusieurs versions de votre code.

Si ce projet suscite suffisamment d'engouement, des classements seront régulièrement mis en ligne sur le site du cours afin de présenter les résultats des dernières simulations.

Outre le prestige, le créateur de la meilleure IA recevra une boite de chocolats.

II. Fichier source

a. Récupération du fichier source

Localisez le fichier source disponible sur le support en ligne du cours et enregistrez-le dans votre répertoire de travail (votre fichier de travail et ce fichier source doivent être au même endroit).

Pour pouvoir utiliser les éléments déclarés dans le fichier source, tapez les lignes suivantes au début de votre fichier de travail :

```
import troll
```

b. Représentation du problème

Pour représenter ce problème en Python, vous allez utiliser des objets de type `Partie`.

La notion d' "objet" est très pratique en programmation, notamment pour structurer l'information. Dans notre cas, cela va nous permettre de caractériser facilement l'état d'une partie.

Pour créer un objet `p` de type `Partie` avec `x` cases et `y` pierres, la syntaxe est la suivante :

```
p = troll.Partie(x, y)
```

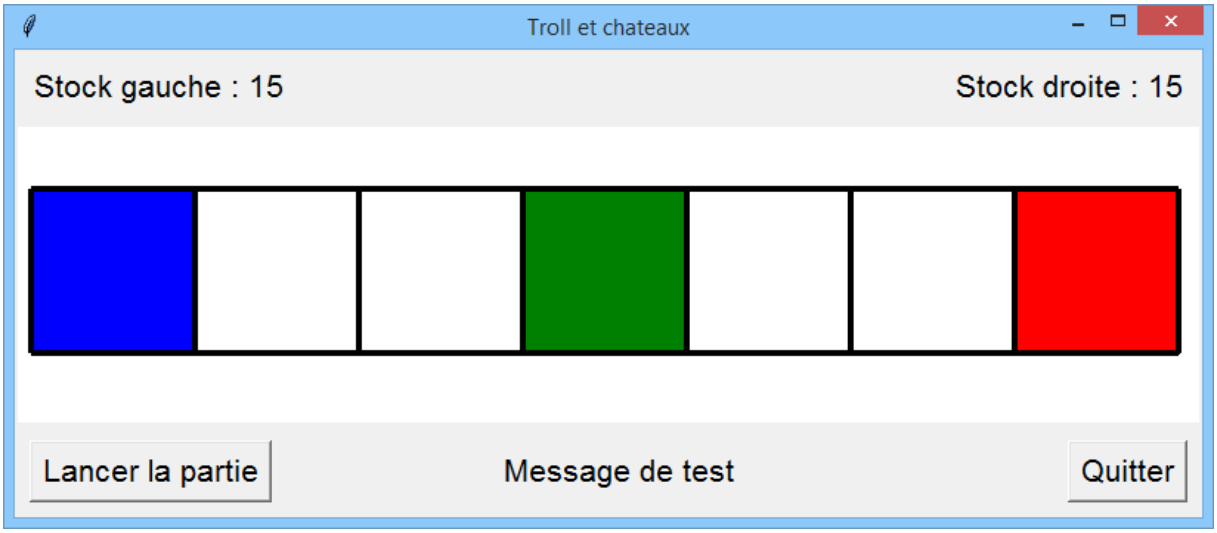
Question 1. Créer un objet de type `Partie` avec 7 cases et 15 pierres, et utiliser la fonction `print` pour l'afficher.

Dans un objet, l'information est stockée sous forme de "propriétés". Pour obtenir la valeur d'une propriété, la syntaxe est simple `nom_de_l_objet.nom_de_la_propriete` (attention à ne pas oublier le point entre les deux)

Un objet `p` de type `Partie` comporte ainsi les éléments suivants :

- Un entier `nombreCases` qui représente le nombre de cases entre les deux châteaux
- Un entier `positionTroll` qui indique la position du troll sur ce chemin
- Un entier `stockInitial` qui indique le nombre initial de pierres
- Un entier `stockGauche` qui indique le nombre de pierres restantes dans la réserve du château de gauche
- Un entier `stockDroite` qui indique le nombre de pierres restantes dans la réserve du château de droite
- Une liste `coupsPrecedents`, qui décrit les coups joués depuis le début de la partie :
    - Les coups sont listés du plus ancien au plus récent
    - Chaque coup est représenté par un couple de valeurs, qui contient
        - le nombre de pierres joué par le joueur de gauche à ce tour
        - le nombre de pierres joué par le joueur de droite à ce tour

c. Exemple



La partie qui débute dans la configuration ci-dessus est décrite par l'objet suivant :

```
----- Partie en cours -----

[Parametres initiaux]
   Nombre de cases : 7
   Stock initial : 15

[Etat de la partie]
   Position du troll : 3
   Stock gauche : 15
   Stock droite : 15

[Coups precedents]
[]
```

Si `p.positionTroll` atteint 0, le joueur de gauche a perdu. Si ce nombre atteint 6, c'est le joueur de droite qui a perdu.

Question 2. Utiliser la syntaxe `nom_de_l_objet.nom_de_la_propriete` pour accéder aux différentes valeurs stockées dans l'objet défini à la question 1.

d. Tour de jeu

Pour interagir avec un objet, on utilise les "méthodes" de cet objet : ce sont des fonctions qui sont liées à cet objet et aux informations qu'il contient.

Par exemple, un objet de type `Partie` possède une méthode `tourDeJeu`, qui permet comme son nom l'indique de jouer un tour. Cette fonction prend deux arguments : le nombre de pierres lancées par le joueur de gauche, et le nombre de pierres lancées par le joueur de droite.

Par exemple, si `p` est un objet de type `partie`, `p.tourDeJeu(2, 4)` modifie la partie p pour prendre en compte le lancer de 2 pierres du côté gauche, et 4 pierres du côté droit.

Question 3. Utiliser la méthode `tourDeJeu` pour simuler "manuellement" plusieurs parties.

e. Simulations

Le fichier source contient en outre une fonction `troll.jouerPartie`, qui attend les arguments suivants :

- Un entier `nombreCases` qui correspond au nombre de cases entre les deux châteaux pour cette simulation
- Un entier `stockInitial` qui correspond au stock initial de pierres dans chaque château pour cette simulation
- Une fonction `strategieGauche`, qui correspond à l'IA utilisée par le joueur du château de gauche
- Une fonction `strategieDroite`, qui correspond à l'IA utilisée par le joueur du château de droite

Vous avez également à votre disposition de deux exemples de stratégies basiques, intitulées `troll.strategieExemple1` et `troll.strategieExemple2`.

Question 4. Utiliser la fonction `troll.jouerPartie` avec les stratégies d'exemple pour déterminer comment fonctionnent ces stratégies.

f. Interface graphique

Le fichier source contient également une class `troll.GUI`. Les objets issus de cette classe permettent de jouer des parties de Trolls et Châteaux de manière plus graphique.

Le constructeur de cette classe prend 4 arguments :

- Le nombre de cases entre les deux châteaux pour cette simulation
- Le stock initial de pierres
- La stratégie du joueur de gauche
- La stratégie du joueur de droite

Question 5. Créer un objet de type `troll.GUI` pour visualiser un affrontement entre les stratégies `troll.strategieExemple1` et `troll.strategieExemple2`.

III. Implémentation

a. Création d'une intelligence artificielle

C'est maintenant à vous de jouer : votre objectif est d'écrire une fonction qui étant donné l'état d'une partie renvoie le nombre de pierres à lancer.

Cette fonction doit :

- prendre deux arguments :
  - Un objet de type `Partie`, correspondant à la partie en cours
  - Une liste d'objets de type `Partie`, qui contient les parties précédentes (si vous voulez étudier les coups de votre adversaire pour adapter vos réponses)
- renvoyer un nombre entier correspondant au nombre de pierres à lancer
- être écrite comme si vous étiez le joueur du château de gauche (le programme de simulation s'occupera d'adapter ce code s'il doit être utilisé pour le château de droite).

Question 6. Ecrire une ou plusieurs fonctions, et confrontez là à celles de vos camarades pour savoir qui est le meilleur chasseur de trolls !

Remarque : Vous pouvez réfléchir à plusieurs, mais il n'y a qu'une seule boite de chocolats à gagner.

b. Simulations et classement

Chaque IA va affronter toutes les autres sur 4 catégories :

- 7 cases et 15 pierres
- 7 cases et 30 pierres
- 15 cases et 30 pierres
- 15 cases et 50 pierres

Pour chaque catégorie, chaque IA affronte chacune des autres sur 1000 matchs : l'IA qui remporte le plus de matches est déclarée vainqueur et gagne 3 points. En cas de match nul, chaque IA gagne 1 point.

Si une IA propose un coup invalide (c'est-à-dire un nombre de pierres négatif ou nul, ou un nombre supérieur au stock de pierres restantes), cet IA est disqualifiée pour cet affrontement et perd 1 point.

Si les deux IA proposent simultanément un coup invalide, elles sont toutes deux disqualifiées et perdent toutes deux 1 point.

La fonction `troll.jouerPlusieursParties` permet de simuler ces affrontements multiples. Elle prend les mêmes arguments que la fonction `troll.jouerPartie`.

Question 7. Tester la fonction `troll.jouerPlusieursParties`.