INF101 : Projet 2023 - Les tours de Hanoi

Le projet se réalise en **binôme** sur votre temps de travail **personnel** (pas en cours). Vous devrez rendre **votre code et votre rapport** à votre enseignant-e de TP selon ses instructions, **avant** le dernier TP lors duquel aura lieu une **soutenance**. Le projet compte dans la note de CC2 (20% de la note d'UE).

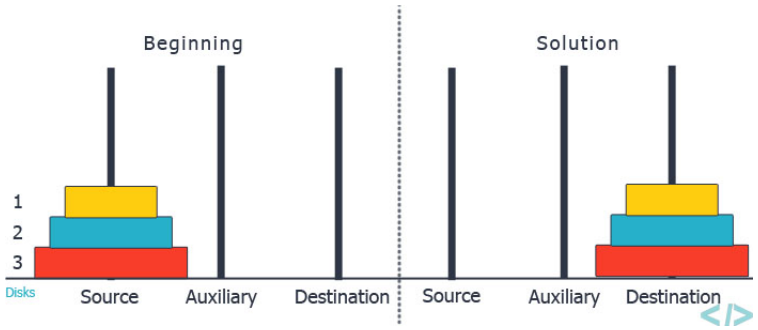
1 Introduction

Dans ce projet on se propose de réaliser un jeu des tours de Hanoi.Le début est plus guidé que la fin, des options sont suggérées, mais vous pouvez (et devez) aussi laisser libre cours à votre imagination pour ajouter des fonctionnalités.

Règles. Les tours de Hanoi sont un jeu de réflexion consistant à déplacer des disques de différents diamètres, d'une tour de départ à une tour d'arrivée, en passant par une tour intermédiaire, en un minimum de coups, avec ces contraintes:

- on ne peut déplacer qu'un seul disque à la fois
- on ne peut pas placer un disque sur un disque plus petit que lui

Dans la configuration de départ, les disques sont empilés en ordre de taille sur la tour de gauche. Dans la configuration finale, ils doivent être empilés dans le même ordre mais sur la tour de droite. Dans le problème classique, il y a 3 disques, et il suffit de 7 coups pour les déplacer d'une tour à l'autre. Dans le problème général à n disques, il faut $2^n - 1$ coups au minimum pour déplacer tous les disques.



Structures de données. On note n le nombre de disques. On représente ces disques par des numéros : le plus petit disque porte le numéro 1, le plus grand disque porte le numéro n.

On représente le plateau de jeu par une liste `plateau`. Celle-ci contient 3 listes, une par tour (départ à l'indice 0, auxiliaire à l'indice 1, arrivée à l'indice 2). Dans chaque liste représentant une tour, le numéro du disque le plus bas est à l'indice 0, et le dernier élément est le numéro du disque le plus haut. S'il n'y a aucun disque sur une tour, la liste correspondante est vide.

Par exemple, sur l'image ci-dessus, la configuration de départ est représentée par la liste `plateau` suivante: $[[3, 2, 1], [], []]$. La tour de départ (indice 0) contient donc tous les disques, le plus gros en bas, puis le moyen, puis le petit en haut; la tour auxiliaire et la tour d'arrivée ne contiennent aucun disque (listes vides). Si on déplace le petit disque sur la tour auxiliaire, on obtient la liste `plateau` suivante: $[[3, 2], [1], []]$. La tour de départ ne contient plus que les disques numéro 3 (le plus grand) et 2 (le moyen); la tour auxiliaire contient le petit disque (numéro 1); la tour d'arrivée ne contient aucun disque. La configuration d'arrivée est $[[], [], [3, 2, 1]]$ (tous les disques sur la tour de droite).

2 Partie A : plateau de jeu et listes

1. Écrire une fonction `init(n)` qui reçoit en argument un entier n (le nombre de disques), et qui renvoie la liste représentant la configuration initiale du plateau, c'est-à-dire avec les n disques (numérotés de 1 à n) empilés dans l'ordre décroissant de taille (et donc aussi de numéro), sur la tour de gauche (à l'indice 0 de la liste `plateau`); les autres tours (milieu et droite) sont vides.

2. Écrire une fonction `nbDisques(plateau, numtour)` qui reçoit la configuration courante du plateau et un numéro de tour (entre 0 et 2, supposé correct), et qui renvoie le nombre de disques sur cette tour.

3. Écrire une fonction `disqueSup(plateau, numtour)` qui reçoit la configuration courante *plateau* et un entier *numtour*, et renvoie le numéro du disque supérieur de cette tour (ou -1 si elle est vide ou incorrecte).

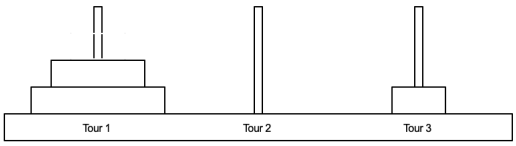
4. Écrire une fonction `posDisque(plateau, numdisque)` qui reçoit la configuration courante du plateau et un numéro de disque (entre 1 et n, supposé correct, donc disque forcément présent), et qui renvoie sa position, c'est-à-dire le numéro de la tour sur laquelle il est placé.

5. Écrire une fonction `verifDepl(plateau, nt1, nt2)` qui reçoit en argument une configuration du plateau, une position initiale et une position finale, et qui renvoie un booléen indiquant si ce déplacement est autorisé (c-à-d qu'il y a bien un disque dans la tour $nt1$, et que la tour $nt2$ est vide ou contient un disque plus grand que celui qu'on veut y poser). On appellera donc la fonction `disqueSup`.

6. Écrire une fonction `verifVictoire(plateau, n)` qui reçoit en argument la configuration du plateau et le nombre de disques, et qui vérifie si on a atteint la solution, c'est-à-dire les n disques empilés dans l'ordre décroissant de taille (de n à 1) sur la tour de droite, les autres tours étant vides.

3 Partie B: graphisme avec Turtle

Spécifications de l'interface On veut dessiner les tours de Hanoï dans une fenêtre `turtle`. N'oubliez pas qu'on peut changer la vitesse de déplacement de la tortue. Par exemple l'affichage ci-dessous est obtenu avec des disques et un plateau d'épaisseur 20, des tours d'épaisseur 6, un diamètre de 40 pour le petit disque, un incrément de 30 entre les diamètres de 2 disques successifs, et un écart horizontal de 20 entre 2 tours (en plus de la taille du plus grand disque), et avec le bord du plateau. La hauteur des tours permet d'y empiler $n + 1$ disques. Le plateau est positionné en $(-300, -200)$. Vous pouvez bien sûr adapter ces valeurs pour améliorer l'affichage à votre guise.



1. Écrire une fonction `dessinePlateau(n)` qui dessine le plateau de jeu vide pouvant recevoir n disques (ce qui influence l'écartement entre les tours - les disques ne doivent pas se chevaucher - et leur hauteur - la tour doit rester visible même si elle contient n disques -).

2. Écrire une fonction `dessineDisque(nd, plateau, n)` qui reçoit un numéro de disque, la configuration du plateau, et le nombre total de disques. Cette fonction trouve la position du disque *nd* sur le plateau, et le dessine aux bonnes coordonnées. *Il faut donc calculer les coordonnées du disque en fonction des paramètres.*

3. Écrire une fonction `effaceDisque(nd, plateau, n)` qui reçoit les mêmes paramètres, et qui efface le disque numéro *nd. Indice: il suffit de dessiner en blanc par dessus ; attention cependant à bien retracer l'axe de la tour si on l'efface. Remarque: cette fonction est très similaire à la précédente, on pourra factoriser le code dans une fonction auxiliaire.*

4. Écrire une fonction `dessineConfig(plateau, n)` qui reçoit en argument la configuration du plateau et le nombre de disques, et qui dessine les disques en conséquence (on suppose le plateau et les tours déjà dessinées). On appellera la fonction `dessine_disque` pour tracer chaque disque.

5. Écrire une fonction `effaceTout(plateau, n)` qui reçoit en arguments la configuration du plateau et le nombre total de disques, et qui efface tous les disques (mais pas les tours) en appelant la fonction `effaceDisque`.

6. Suggestions d'améliorations : on pourra ajouter une image de fond, des couleurs de tours/disques, du texte, etc.

4 Partie C: interactions avec le joueur

On va maintenant s'intéresser à l'interaction avec l'utilisateur. Pour simplifier, on se limitera au départ à une interaction au clavier : le programme demandera à l'utilisateur de saisir le numéro de la tour de départ (il ne peut déplacer que le disque au sommet de cette tour) et de la tour d'arrivée. Il faut l'empêcher de choisir une tour de départ vide, ainsi qu'une tour d'arrivée contenant un disque plus petit que le disque déplacé.

1. Écrire une fonction `lireCoords(plateau)` qui reçoit en argument la configuration du plateau, qui demande à l'utilisateur le numéro de la tour de départ (filtrée jusqu'à ce qu'elle soit correcte entre 0 et 2, et que la tour correspondante ne soit pas vide), puis le numéro de la tour d'arrivée (filtrée pour être correcte entre 0 et 2, et pour ne pas contenir un disque plus petit que le disque déplacé). La fonction redemande jusqu'à obtenir des coordonnées correctes, puis renvoie ces 2 numéros (tour de départ, tour d'arrivée). *Attention à ne pas bloquer sur la lecture de la tour d'arrivée si le disque au sommet de la tour de départ ne peut pas être déplacé car aucune tour d'arrivée n'est possible. Il faut alors relire aussi le numéro de la tour de départ.*

2. Écrire une fonction `jouerUnCoup(plateau,n)` qui appelle la fonction `lireCoords` pour récupérer le déplacement que le joueur souhaite faire ; puis qui déplace effectivement le disque grâce aux fonctions `dessineDisque` et `effaceDisque` de la partie B. Cette fonction **modifie** aussi la liste `plateau` reçue en argument pour refléter la nouvelle configuration.

3. Écrire une fonction `boucleJeu(plateau,n)` qui reçoit en arguments la liste `plateau` contenant une configuration des disques, ainsi que le nombre n de disques, et qui interagit avec l'utilisateur pour déplacer des disques **jusqu'à** ce que celui-ci ait gagné. On utilisera la fonction `verifierVictoire` pour savoir quand arrêter, et la fonction `jouerUnCoup` pour effectuer chaque tour de jeu (modification de l'affichage et de la liste `plateau`). La fonction doit aussi compter le nombre de coups joués par l'utilisateur, et renvoyer ce compteur en fin de boucle (victoire).

   - **Option1** : afin d'éviter des parties trop longues, on pourra ajouter un nombre maximum de coups autorisés et arrêter le jeu si le joueur ne gagne pas avant cette limite. Dans ce cas la fonction doit renvoyer non seulement le nombre de coups joués, mais aussi un booléen indiquant si le joueur a gagné (`True`) ou perdu (`False`).
   - **Option2** : ajouter la possibilité pour le joueur d'abandonner la partie, par exemple en saisissant une tour de départ égale à -1 (pour éviter de lui poser la question à chaque tour pour savoir s'il veut continuer à jouer, avant de lire les coordonnées). Dans ce cas attention à bien autoriser cette valeur -1 lors du filtrage de la saisie utilisateur de la fonction `lireCoords`. La boucle de jeu doit alors renvoyer en plus du nombre de coups une valeur indiquant si le joueur a gagné, perdu (trop de coups), ou abandonné.

4. Écrire un programme principal qui demande à l'utilisateur le nombre de disques n souhaité, puis qui initialise la liste décrivant le plateau en position initiale, le dessine dans la fenêtre `turtle`, puis lance la boucle de jeu. Le programme se termine quand la boucle de jeu se termine, soit parce que l'utilisateur a gagné, soit parce qu'il a abandonné avant la fin, soit parce qu'il a perdu (selon les options que vous aurez choisi de coder ou pas). Le programme affiche un message de défaite ou de victoire selon les cas, avec le nombre de coups joués et le nombre de coups minimal possible.

L'exemple ci-dessous illustre quelques interactions du programme principal souhaité. Les coordonnées sont filtrées : si la tour de départ est vide, on redemande le numéro de la tour de départ ; si la tour d'arrivée est occupée, on redemande les 2 coordonnées (pour éviter un blocage quand aucune tour d'arrivée n'est disponible). On a ici choisi d'autoriser la valeur -1 pour la tour de départ, ce qui déclenche une demande de confirmation d'abandon, puis la sortie de la boucle de jeu.

```
Bienvenue dans les Tours de Hanoi
Combien de disques? 3
Coup numéro 1
Tour de départ? 1
Invalide, tour vide.
Tour de départ? 0
Tour d'arrivée? 1
Je déplace le disque 1 de la tour 0 à la tour 1
Coup numéro 2
Tour de départ? 0
Tour d'arrivée? 1
Invalide, disque plus petit.
Tour de départ? 0
Tour d'arrivée? 2
Je déplace le disque 2 de la tour 0 à la tour 2
Coup numéro 3
Tour de départ? -1
Tu souhaites abandonner (o/n)? o
Abandon de la partie après 2 coups.
Au-revoir.
```

Suggestions d'améliorations. On pourra ajouter l'affichage du nombre de coups dans la fenêtre graphique; l'interaction à la souris (l'utilisateur clique sur le disque qu'il veut déplacer, puis sur la tour de destination) ; et des boutons (par exemple pour abandonner la partie, pour afficher les règles du jeu ou le nom de l'équipe de développement, etc).

5 Partie D: annulation de coups

On se propose maintenant de permettre au joueur de revenir en arrière en annulant un ou plusieurs coups. Pour cela on va enregistrer à chaque tour l'état courant du jeu dans un dictionnaire `coups`. Les clés de ce dictionnaire sont les numéros de coups (le premier coup est le numéro 1). Les valeurs de ce dictionnaire sont les configurations `plateau` à chaque étape (en clé 0: plateau initial). Ce dictionnaire contient donc tout l'historique de la partie.

1. Écrire une fonction `dernierCoup` qui reçoit un dictionnaire `coups` et qui renvoie le dernier coup joué, sous la forme d'une paire (tour de départ, tour d'arrivée). **Indice:** quel est le numéro du dernier coup ?

2. Écrire une fonction `annulerDernierCoup` , qui reçoit le dictionnaire `coups` et qui annule le dernier coup. Il faut donc utiliser `dernierCoup` et inverser le dernier coup, mais aussi penser à supprimer la dernière configuration du dictionnaire, et décrémenter le compteur de coups (égal à la plus grande clé de ce dictionnaire).

3. Modifier la fonction `boucleJeu` pour y autoriser l'annulation du dernier coup.

4. **Option**: on pourra aussi permettre au joueur d'enregistrer la configuration courante de sa partie (le dictionnaire `coups`) dans un fichier, en utilisant par exemple le module `pickle`, et de recharger plus tard une partie à partir d'un fichier pour la reprendre au même endroit.

6 Partie E: comparaison des scores et temps de jeu

On va maintenant modifier le programme pour permettre à plusieurs joueurs de jouer à tour de rôle, et enregistrer leurs scores (voire leur temps de jeu) dans un dictionnaire. Cela a l'avantage d'offrir un stockage structuré des informations, et de permettre ensuite de les comparer. On n'enregistre que les parties gagnées.

1. Écrire une fonction `sauvScore` qui enregistre le score du joueur (nom, nombre de disques, nombre de coups) dans un dictionnaire. **Attention**: un même joueur peut jouer plusieurs parties. Quelles sont les clés du dictionnaire?

2. *Option:* on pourra aussi enregistrer la durée de la partie c-à-d le temps de jeu en secondes et/ou minutes (voir le module `time`, et en particulier la fonction `localtime()`).

3. Modifier le programme principal de jeu pour qu'en cas de victoire, il demande son nom au joueur et appelle la fonction `sauvScore` pour enregistrer son score dans le dictionnaire.

4. Écrire une fonction `afficheScores` qui reçoit un dictionnaire de scores et affiche un tableau des meilleurs scores (plus petit nombre de coups), dans l'ordre (meilleur joueur en haut), pour un nombre de disques donné ou pour chaque nombre de disques possible. L'affichage pourra se faire dans la console ou dans la fenêtre `turtle`, au choix.

5. Écrire une fonction `afficheChronos` qui affiche le classement en durée de jeu (si vous avez stocké cette information).

6. Écrire une fonction `reflexionMoy` qui calcule le temps moyen de réflexion par coup, par joueur. Pour cela il faut récupérer les informations de toutes les parties gagnées par ce joueur (on suppose que les noms des joueurs sont uniques), avec le temps total et le nombre de coups.

7. Écrire une fonction qui afficher un classement des joueurs qui jouent le plus vite (moins de réflexion par coup, ce qui ne veut pas forcément dire qu'ils ont été plus efficaces en nombre de coups).

8. Ajouter un menu à votre programme principal pour que le joueur puisse choisir d'afficher les scores, temps, classement, etc.

9. *Option*: on pourrait aussi enregistrer le nombre de victoires et défaites pour chaque joueur, pour faire ensuite des statistiques sur le pourcentage de parties gagnées.

10. *Option:* on pourra stocker les scores dans un fichier, et les charger depuis ce fichier, avec le module `pickle`. Dans ce cas on ajoutera la date de chaque partie.

11. *Option:* utiliser le module `matplotlib` pour tracer des graphiques de statistiques intéressantes sur les parties gagnées.

7 Partie F: jeu automatique, fonction récursive

Dans cette partie on souhaite créer une fonction capable de trouver la liste optimale des mouvements à jouer pour déplacer n disques de la tour de départ à la tour d'arrivée.

1. Écrire une fonction qui résout automatiquement le problème des tours de Hanoï avec n disques. Elle doit renvoyer la liste des mouvements à effectuer.

2. Écrire une fonction qui reçoit une liste de déplacements (supposés corrects) et qui les anime dans `turtle`.

3. Ajouter un bouton "voir solution" dans l'interface, ou une option dans la boucle de jeu au clavier. On peut aussi prévoir d'animer la solution quand le joueur perd ou abandonne la partie.

4. Options: ajouter la possibilité pour le joueur de demander un indice: dans ce cas le programme lui suggère un coup à jouer (textuellement dans la console, ou visuellement en surlignant le disque à déplacer).