TP - tableaux - allumettes

Objectif(s)

- ✯ Travailler sur des fonctions manipulant des tableaux

Exercices obligatoires

Exercice 1

Question 1

Écrivez une fonction `sont_valeurs_positives` qui prend en paramètre un tableau d'entier et sa taille et renvoie 1 si toutes les valeurs du tableau sont positives, 0 sinon.

Écrivez une fonction de test pour valider votre code, avec plusieurs tests.

Question 2

Écrivez une fonction `compte_nombres_avant_moins1` qui prend un tableau et sa taille en argument, et renvoie le nombre de cases parcourues avant de rencontrer la valeur -1. Si la valeur -1 n'est pas atteinte avant la fin du tableau, la valeur -1 sera renvoyée.

Écrivez une fonction de test pour valider votre code, avec plusieurs tests.

Exercice 2 – Le jeu des allumettes, celui de Fort-Boyard...

Un jeu d'allumettes classique, joué dans l'émission Fort Boyard est *la course à 20* (parfois appelé à tort Jeu de Nim, un jeu plus complexe).

La course à 20 se joue sur un simple tas de 20 allumettes. Chaque joueur à son tour retire une, deux ou trois allumettes du tas. Le joueur prenant la **dernière** allumette **gagne**.

Question 1

Écrivez d'abord une fonction `demande_joueur` qui prend en paramètre le nombre d'allumettes restant et demande au joueur de saisir un nombre parmi un, deux ou trois, et retourne ce nombre. Si l'utilisateur choisit un nombre invalide (différent de 1, 2 ou 3, ou plus grand que le nombre d'allumettes restant), la fonction le sermonne et lui demande à nouveau. Cette fonction ne modifie pas le nombre d'allumettes du jeu, la fonction principale fera cette modification en tant qu'arbitre.

Question 2

Écrivez une fonction principale qui demande alternativement aux deux joueurs le nombre d'allumettes qu'ils doivent retirer et met à jour le jeu, puis annonce le vainqueur.

Nous allons maintenant écrire des bots pour ce jeu (on pourrait dire des IA, mais c'est un bien grand concept pour ce qui va suivre). Les bots devront avoir la même interface que `demande_joueur`, pour pouvoir s'y substituer dans le programme principal.

Question 3

Écrivez une fonction `bot_idiot` qui prend en paramètre le nombre d'allumettes du jeu et retourne un entier tiré au hasard entre 1 et 3 (mais pas plus le nombre d'allumettes restant dans le tas, bien sûr !). C'est un bot bien facile à battre !

Question 4

Pour gagner à ce jeu, il suffit de toujours laisser à l'adversaire un nombre d'allumettes multiple de quatre. Écrivez une fonction `super_bot` qui prend en paramètre le nombre d'allumettes du jeu, et retourne le nombre d'allumettes selon cette stratégie si c'est possible, le résultat d'un appel à la fonction précédente si ce n'est pas possible.

Question 5

Malheureusement, contre un tel bot, le jeu est frustrant, car si on commence, le bot gagne systématiquement. Écrivez un troisième bot `bot_moyen` qui joue au hasard tant qu'il reste plus de 12 allumettes, et de façon optimale ensuite. On fera bien appel aux fonctions précédentes.

Renforcements

Exercice 3 – Le vrai jeu de Nim

Dans le vrai jeu de Nim, appelé aussi jeu de Marienbad, il y a au départ plusieurs tas d'allumettes. C'est ce jeu qui a donné naissance au Nimrod, l'un des premiers jeux vidéos de l'histoire.

À son tour, chaque joueur peut retirer **autant d'allumettes** qu'il le souhaite, mais dans un seul tas. De nouveau, le joueur qui retire la dernière allumette du dernier tas **gagne**.

Le jeu sur un seul tas n'a pas de sens puisque le premier joueur gagne systématiquement en prenant toutes les allumettes. Le jeu sur deux tas est moins trivial, mais une stratégie gagnante est facile à identifier (l'avez-vous ?). À partir de trois tas, le jeu devient plus intéressant. Une situation initiale standard consiste en 4 tas, de tailles 1, 3, 5 et 7.

Question 1

Décrivez comment décrire l'état d'une partie :

— si on fixe à l'avance le nombre maximum de tas.

— si on se laisse la liberté d'un nombre quelconque de tas (soyons réaliste, disons au plus cent).

Écrivez une fonction `initialise_jeu_standard` qui initialise une partie comme ci-dessus, et une fonction `initialise_jeu_aleatoire` qui initialise une partie avec un nombre de tas aléatoire (entre 2 et 10) qui comportent des nombres d'allumettes aléatoires (entre 1 et 20).

Question 2

Écrivez à nouveau la fonction `demande_joueur` qui demande au joueur de jouer un coup, et vérifie la validité du coup.

Question 3

Écrivez une fonction `jouer_partie` qui permet à deux joueurs de faire une partie dans une interface de jeu jouable. Une ligne représentera un tas, chaque allumette étant représentée par une étoile. Vous pouvez effacer l'écran avec la commande : `printf ("\33[H\33[2J");`. Vous pouvez aussi insérer des pauses dans l'affichage en utilisant les commandes `sleep` et `usleep` : vous aurez les détails de ces commande avec un petit `man 3 usleep` dans la console.

Question 4

Écrivez une fonction de bot aléatoire, mais qui remporte la partie si c'est possible en un coup.

Question 5

Écrivez un programme principal permettant de lancer une partie contre un joueur humain ou un bot aléatoire, au choix.

Question 6

Implémentez la fonction qui permet à un bot de gagner sur deux tas. (indice : les situations où les deux tas sont de même taille sont particulières)

Question 7

Implémentez la vraie stratégie gagnante que vous pouvez trouver sur la page du Marienbad.