

Travaux dirigés 7 : pour aller plus loin

1 PGCD (récursivité)

Rappel : pour a et d deux entiers naturels, on dit que d divise a , et on note $d \mid a$, s'il existe un entier q tel que $a = dq$. En particulier $d \mid 0$ quel que soit d . Le plus grand diviseur commun (PGCD) de deux entiers naturels a et b est le plus grand entier d tel que $d \mid a$ et $d \mid b$. Cette définition donne un moyen effectif de trouver le PGCD de deux entiers (un algorithme) : il suffit de chercher tous les diviseurs de a et de b et de prendre le plus grand d'entre eux.

1. Que vaut $\text{PGCD}(0, 0)$?
2. Si seulement un des entiers a et b est nul, que vaut $\text{PGCD}(a, b)$?
3. Si aucun des deux entiers a et b n'est nul, jusqu'à quelle valeur doit-on chercher des diviseurs de a et de b ?
4. Comment tester si un nombre en divise un autre, en C ?
5. Écrire une fonction non récursive qui calcule le PGCD de deux entiers naturels.
6. Rappeler l'algorithme d'Euclide pour le calcul du PGCD.
7. Écrire une fonction récursive pour le calcul du PGCD de deux entiers.
8. À votre avis quelle est la méthode la plus rapide ?

2 Les struct ne sont pas si complexes et point difficiles

9. Proposer un type utilisateur pour les nombres complexes en notation algébrique ($a + ib$, $a, b \in \mathbb{R}$).
10. Donner une fonction d'addition et une fonction de multiplication entre ces nombres complexes (les tester).
11. Proposer un type utilisateur pour les points de l'espace en coordonnées réelles : \mathbb{R}^3 .
12. Écrire une fonction `est_dans_sphere` prenant en argument un point c , un réel r , et un point p , qui renvoie vrai si le point p appartient à la sphère de centre c et de rayon r , faux sinon.
13. Définir une fonction `distance` qui retourne la distance entre deux points de l'espace passés en arguments. Simplifier la fonction `est_dans_sphere` en faisant appel à cette fonction `distance`.
14. Proposer un type utilisateur pour représenter une sphère dans l'espace.
15. Définir une fonction `collision_spheres` qui prend en entrée deux sphères et retourne `true` si les deux sphères ont une intersection non vide, `false` sinon.

3 Le jeu de la tablette de chocolat

Une tablette (ou plaquette) de chocolat est une matrice de carreaux de chocolat que l'on peut couper selon les lignes ou les colonnes qui séparent les carreaux. La hauteur et la largeur se comptent en nombre de carreaux. On a peint en vert un carreau dans le coin d'une tablette. Deux joueurs coupent tour à tour la tablette, au choix, selon une ligne ou une colonne (pas nécessairement près du bord). L'objectif est de ne pas être le joueur qui se retrouve avec le carreau vert (la tablette de hauteur 1 et de largeur 1).

Nous allons programmer ce jeu en faisant jouer à l'ordinateur le rôle de l'opposant.

3.1 Type utilisateur (optionnel)

Définir un type utilisateur qui servira à représenter la tablette (faire simple!).

3.2 Le tour de jeu et l'arbitre

Le joueur commencera la partie, puis l'opposant (l'ordinateur) jouera et ainsi de suite. À chaque fois l'une des deux dimensions de la tablette sera modifiée, jusqu'à ce qu'il y ait un perdant. Vous adopterez la structure suivante pour le programme :

- Une variable sera utilisée pour déterminer à qui vient le tour de jouer (joueur ou opposant);
- Une boucle réalisera le tour de jeu;
- Une fonction `partie_perdue()` prenant en argument la tablette et renvoyant vrai lorsque celle-ci se réduit au carré vert, servira à construire la condition de sortie de la boucle;
- Après quoi un affichage annoncera au joueur s'il a gagné ou perdu.
- À l'intérieur de la boucle, selon le tour, il sera fait appel à une fonction `joueur()` ou bien à une fonction `opposant()` (ces fonctions doivent modifier la tablette).

Vous êtes libre du choix de la taille initiale de la tablette.

3.3 Affichage

Réaliser une fonction d'affichage de la tablette qui sera appelée au début de chaque tour de jeu. Vous pouvez dans un premier temps vous contenter d'un affichage numérique de ses dimensions, et par la suite améliorer votre programme avec un affichage plus graphique.

3.4 Les joueurs

Le joueur étant l'humain entre le clavier et l'écran, il faudra lui faire saisir le choix de son coup, en deux temps : couper des colonnes ou couper des lignes ? Combien ? À vous de déterminer la manière de réaliser cette saisie.

Vous pouvez commencer à tester le fonctionnement de votre programme en utilisant la fonction `joueur()` également pour l'opposant.

L'opposant jouera au hasard. Pour cela vous pouvez vous aider d'une fonction `nombre_aleatoire` qui tire au hasard un nombre entre zéro et son argument (un entier positif). Cette fonction utilisera `rand`. N'oubliez pas d'initialiser le générateur de nombres aléatoires dans le `main()` ni d'inclure les bons fichiers d'en-tête (fonctionnalités supplémentaires).

Tester votre programme. Avez-vous prévu les cas où l'un des deux joueurs n'a pas le choix ? Est-ce que les coups invalides sont rendus impossibles ? Vous pouvez réutiliser et améliorer des fonctions vues précédemment (`choix_utilisateur()`, par exemple).

4 Améliorations

4.1 Tablette aléatoire

Dans votre programme la tablette est initialisée au départ. Pouvez-vous utiliser une fonction sans argument, `tablette_aleatoire` qui renvoie une tablette aléatoire ?

4.2 L'opposant parfait

Pouvez-vous trouver la meilleure manière de jouer (cela dépend de la tablette de départ) ? Si oui, faites en profiter votre opposant. Vous pouvez également le faire se tromper de temps en temps, en introduisant pour cela une dose de hasard...

4.3 Plusieurs tablettes

On peut utiliser plusieurs tablettes pour ce jeu : chaque joueur choisit sur quelle tablette jouer son tour et le perdant sera celui recevant pour son tour toutes les tablettes à la dimension 1×1 .