Programmation en C et structures de données

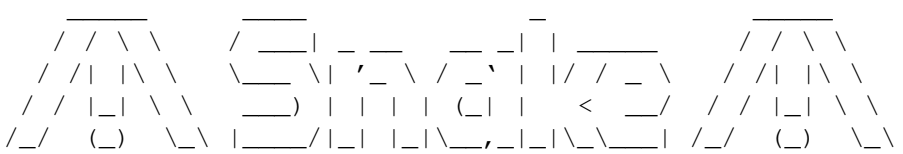
guillaume.revy@univ-perp.fr

CC1, Jeudi 27/10/2022 de 13h45 à 17h45

Ce programme est à réaliser seul, sous environnement GNU/Linux. Il doit être rendu sur Moodle, sous forme d'un unique fichier <nom_prenom.c>. Seuls les documents disponibles sur Moodle sont autorisés. Tout autre document (cours, exercices, ...), ou autre matériel (téléphone portable, ...) est interdit.

Conseils :

- Il est à noter que la qualité primera sur la quantité : un programme court, clair, bien conçu et qui compile sera mieux noté qu'un gros programme mal conçu, incompréhensible, voire qui ne compile pas. Utilisez des noms de variables explicites, indentez votre programme, et n'hésitez pas à le commenter.
- Lisez tout le sujet avant de commencer.
- Veuillez respecter les noms de fonctions et structures proposées dans le sujet.



Le Snake est un jeu dans lequel le joueur dirige un serpent au clavier, sur une grille de dimensions données contenant des pommes et des obstacles. Le but est que le serpent mange les pommes pour grandir. Mais attention, le serpent doit également ne pas sortir de la grille et éviter des obstacles ... et en grandissant, il devient lui-même un obstacle.

L'objectif du TP est d'écrire un programme qui permet de jouer au Snake. Ce programme utilisera la bibliothèque `ncurses` pour l'affichage graphique.

1. Manipulation des coordonnées

Une grille de jeu contient la position du serpent, celles des obstacles et des pommes. Une position est invalide si au moins une des coordonnées est égale à -1, et elle est valide sinon.

► **1.** Définir une structure `position` pour représenter une position sur la grille, c'est-à-dire, les coordonnées entières d'une case sur la grille.

Soit T un tableau de positions de taille n. Ce tableau contient k positions valides, avec $k \in \{0, \cdots, n\}$, puis $n - k$ positions invalides, comme le tableau ci-dessous pour $(n, k) = (8, 5)$:

$$T = \{(0,0), (0,1), (0,2), (0,3), (1,3), (-1,-1), (-1,-1), (-1,-1)\}.$$

► **2.** Écrire une fonction `initialiser` qui initialise un tableau de taille n avec des positions invalides.

► **3.** Écrire une fonction `ajouter` qui ajoute une nouvelle position valide au tableau, après les k premiers éléments. Par exemple :

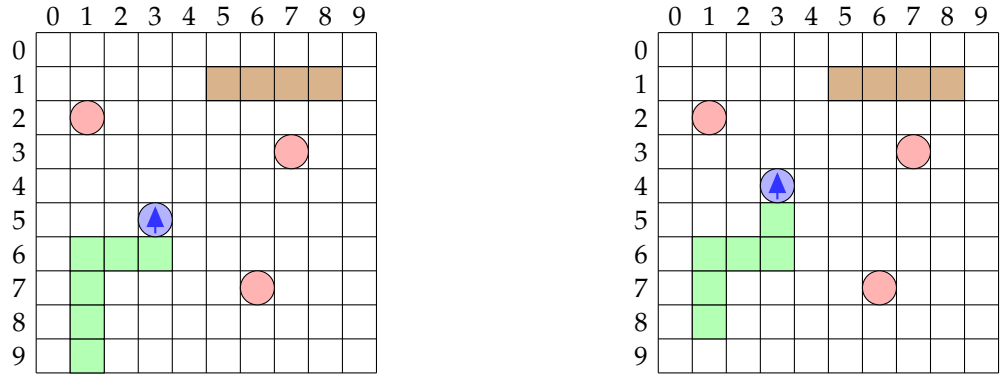
$$\texttt{ajouter}(T, (2,3)) \quad \rightarrow \quad \{(0,0), (0,1), (0,2), (0,3), (1,3), (2,3), (-1,-1), (-1,-1)\}.$$

► **4.** Écrire une fonction `retirer` qui retire et retourne la position d'indice 0 du tableau. Par exemple :

$$\texttt{retirer}(T) \quad \rightarrow \quad (0,0), \{(0,1), (0,2), (0,3), (1,3), (2,3), (-1,-1), (-1,-1), (-1,-1)\}.$$

► **5.** Écrire une fonction `appartient` qui retourne l'indice d'une position dans le tableau, et -1 si elle n'existe pas. Par exemple :

$$\texttt{appartient}(T, (0,1)) \quad \rightarrow \quad 1 \quad \text{et} \quad \texttt{appartient}(T, (0,4)) \quad \rightarrow \quad -1.$$



(a) avant le déplacement (b) après le déplacement

FIGURE 1 – Exemple de grille 10×10.

2. Manipulation de la grille de jeu

Dans ce TP, nous considèrons une grille de jeu de taille 60×30.

Le serpent est représenté par :
- une file de positions, matérialisée par un tableau 1D de positions (comme dans l'Exercice 1), indiquant les cases de la grille occupées par le serpent, et où la position de la tête est la dernière position enfilée,
- une direction,
- et une longueur.

Le serpent ne pouvant occuper que 60×30 cases au total, la file est stockée dans un tableau de taille $60 \times 30 = 1800$. Par exemple, sur le dessin de la Figure 1a représentant un exemple simplifié sur une grille de taille 10×10, le serpent (vert à tête bleue), occupe 7 des 100 cases de la grille. Dans ce cas, la file est :

$$\{(1,9),(1,8),(1,7),(1,6),(2,6),(3,6),(3,5),(-1,-1),\cdots,(-1,-1)\},$$

la longueur est 7 et la direction est *nord*. Comme illustré sur la Figure 1b, après un déplacement, la file sera la suivante :

$$\{(1,8),(1,7),(1,6),(2,6),(3,6),(3,5),(3,4),(-1,-1),\cdots,(-1,-1)\},$$

où $(3,4)$ est la nouvelle position de la tête.

► **1.** Définir une structure `serpent` permettant de représenter le serpent.

► **2.** Écrire une fonction `creer_serpent` qui, étant données la position de la tête, la longueur et la direction, crée et retourne le serpent correspondant, où le corps du serpent est constitué d'une ligne droite.

► **3.** Écrire une fonction `afficher_grille` qui permet d'afficher la grille et le serpent. *Elle pourra être modifiée au cours du TP pour afficher les pommes et les obstacles.*

► **4.** Écrire une fonction `avancer` qui permet de faire avancer le serpent d'une case dans sa direction.

► **5.** Écrire enfin une fonction `changer_direction` qui, étant donnée un entier indiquant une nouvelle direction, permet au serpent de changer de direction.

La grille contient également des pommes. À chaque fois que le serpent mange une pomme, il grandit d'une unité. On considère ici que la grille contient en permanence 20 pommes, placées de manière aléatoire sur la grille et dont on connaît la position. La position des pommes peut être stockée dans un tableau 1D. Sur la Figure 1, il y a 3 pommes : le tableau serait alors

$$T = \{(1,2),(7,3),(6,7)\}.$$

► **6.** Écrire une fonction `creer_pommes` qui permet de créer 20 pommes sur la grille, toutes positionnées de manière aléatoire, et de les stocker dans un tableau.

► **7.** Écrire une fonction `manger` qui, étant donné un tableau de pommes, retourne un entier indiquant si le serpent mange une pomme, c'est-à-dire, si sa tête est sur une case contenant une pomme. Si le serpent mange une pomme, celle-ci est supprimée et une nouvelle est crée.

► **8.** Écrire une fonction `redimensionner_serpent` qui permet de faire grandir le serpent d'une unité.

Finalement, la grille contient des obstacles. Dès que le serpent tape la tête dans un obstacle, vous avez perdu et la partie s'arrête. On considère ici que la grille ne peut contenir que 6 obstacles, en plus de lui-même (et des bords de la grille). Un obstacle est représenté par une position, une longueur et une direction. Sur la Figure 1, l'obstacle est en position $(5,1)$, de longueur 4 et en direction *est*.

► **9.** Définir une structure `obstacle`, qui permet de représenter un obstacle.

► **10.** Écrire une fonction `creer_obstacle` qui, étant données une position, une longueur et une direction, permet de créer un obstacle sur la grille, et une fonction `creer_obstacles` qui permet de créer les 6 obstacles et les stocker dans un tableau.

► **11.** Écrire enfin une fonction `percuter` qui, étant donné un tableau d'obstacles, retourne un entier indiquant si le serpent percute un obstacle.

3. Snake en mode graphique

Pour gagner, on considère que le serpent, de longueur initiale 4, doit manger un total de 30 pommes.

L'objectif de cette dernière étape est de créer le programme principal, en utilisant la bibliothèque `ncurses`. Elle fournit une API pour le développement d'interfaces utilisateur, en utilisant les caractères et couleurs d'un mode semi-graphique. Elle prend notamment en charge la gestion des évènements clavier.

Remarque 1 *Vous pourrez écrire une première version du programme sans utiliser la bibilothèque `ncurses`, et faire l'affichage graphique en mode texte dans un terminal.*

Le listing ci-dessous montre un exemple simple de programme utilisant cette bibliothèque. Pour le compiler, il faut ajouter `-lncurses` à la ligne de compilation.

```
1  #include <ncurses.h>
2  #include <unistd.h>
3
4  int
5  main(void)
6  {
7    int X = 4, Y = 10, action = 'N';
8
9    // ... initialisation de ncurses
10   initscr();
11   cbreak();
12   noecho();
13   keypad(stdscr, TRUE);
14   refresh();
15   curs_set(FALSE);
16   nodelay(stdscr, TRUE);
17
18   // ... boucle principale
19   while(action != 'Q') {
20     if((action = getch()) != ERR)
21       mvprintw(Y, X, "%c %3d ", action, action);
22     if(action == KEY_RIGHT) X++;     // fleche droite
23     else if(action == KEY_LEFT) X--; // fleche gauche
24     else if(action == KEY_UP) Y--;   // fleche haut
25     else if(action == KEY_DOWN) Y++; // fleche bas
26     // ...
27     usleep(125000);
28   }
29
30   mvprintw(Y, X, "... la boucle est maintenant terminee ... appuyez sur Entree");
31   nodelay(stdscr, FALSE);
32   action = getch();
33
34   // ... arret de ncurses
35   endwin();
36
37   return 0;
38 }
```

Ici la fonction `mvprintw` (Ligne 30) est la fonction d'affichage : elle fonctionne de la même manière que `printf`, les deux premiers paramètres étant les coordonnées de l'affichage dans le terminal. Ce programme affiche donc l'action clavier en Y^{e} ligne et X^{e} colonne, tant que celle-ci n'est pas le caractère 'Q'.

► **1.** Modifier l'affichage pour afficher la grille dans le terminal en utilisant cette bibliothèque. *On pourra dissocier l'affichage de la grille de celle du serpent.*

► **2.** Écrire finalement la fonction principale, en utilisant cette bibliothèque. Le déplacement pourra alors se faire en utilisant le pavé directionnel.