Université Sorbonne Paris Nord

L1 Informatique
Année 2023-2024

Mini-projet du module "Programmation 2"

Le jeu du serpent

Mini-Projet à réaliser en binôme (ou en monôme) au sein de votre groupe de TP.
Vous devez avoir indiqué votre binôme au plus tard le 7 mai à votre chargé de TP.

Vous devrez rendre par l'ENT, **AVANT** *le début de la dernière séance de TP du 16 mai 2024, une archive contenant*

- *votre code (fichier .h, .c et Makefile uniquement)*
- *un fichier txt indiquant les caractériques de votre code*

Lors d'une mini-soutenance lors de cette dernière séance de TP, vous montrerez une exécution de votre code et vous répondrez à des questions concernant des lignes de code précises et l'organisation globale du code.

Introduction

On s'intéresse au jeu du serpent, bien connu sous le nom de Snake. Le jeu se passe dans une grille rectangulaire de $n \times m$ cases. On dirige un serpent consitué d'une file de cases qui doit manger un fruit apparaissant aléatoirement sur une case de la grille. A chaque fruit mangé, le serpent s'allonge. Le jeu s'arrête si le serpent se heurte lui-même ou heurte un bord de la grille. L'objectif du jeu étant de manger un maximum de fruits.
En plus d'implémenter ce jeu, on considérera des variantes dans une deuxième partie.

1 Première partie : jeu à un serpent

Exercice 1 : Codage de la grille

Dans ce premier exercice, on veut coder une structure permettant d'afficher la grille du jeu et de tirer un fruit au hasard.

Q 1.1. Créer un fichier `Grille.h` définissant une structure nommée `Grille` contenant
- deux entiers n et m donnant les dimensions de la grille rectangulaire
- un tableau à deux dimensions de chaînes de caractères.
- les coordonnées de la case contenant un fruit

Q 1.2. Créer un fichier `Grille.c` définissant les fonctions (déclarées dans `Grille.h`) suivantes
- `Grille_allouer` prenant en entrée un deux entiers n et m qui retourne un pointeur sur une Grille, dont le tableau est également alloué, c'est à dire que chacune de ses cases contient une chaîne de caractères de taille 8.
- `Grille_vider` qui met **deux espaces** [1] de fond noir dans les cases du tableau.
- `Grille_tirage_fruit` qui tire au sort les coordonnées d'une case où apparaîtra le fruit et les stocke dans les champs de Grille.
- `Grille_remplir` qui remplit les cases du tableau des deux espaces de la couleur correspondant aux éléments du jeu : pour l'instant il n'y a à placer que les deux case de couleur de fond rouge correspondant au fruit.
- `Grille_desallouer` qui désalloue la structure et son contenu.

Q 1.3. Créer un main dont la ligne de commande contient les dimensions de la grille, qui alloue et vide le tableau, tire un fruit et le place dans le tableau. Créer un `Makefile` et tester votre programme.

Q 1.4. Ajouter la fonction `Grille_redessiner` qui affiche la grille en affichant le contenu des cases de son tableau. On veut également que la grille soit entourée d'un liseret fait d'espaces de fond d'une couleur de votre choix.

Pour afficher la grille, on va utiliser les possibilités du terminal texte et des séquences ASCII dites "séquences d'échappements ANSI" (ANSI Escape Sequences). Ces séquences sont des suites de caractères ASCII qui peuvent être envoyés au terminal par la commande `printf`.
Le caractère ASCII pour ESC peut être écrit (en octal) \33 : on fait suivre immédiatement ce code d'autres caractères qui composent une commande. Ces caractères ne sont pas affichés mais ils ont une action sur le terminal :
- "\33[2J" : efface le terminal
- "\33[H" : déclare que le prochain caractère sera écrit en haut à gauche du terminal.
- "\33[XXm" où XX est un nombre décimal : déclare que les caractères suivants sont de couleur de fonds noir si XX=00, rouge si XX=41, vert si XX=42... [2]

1. On peut remarquer que les cases d'un terminale sont rectangulaires, ce qui n'est pas très joli... une astuce est de représenter graphiquement une case de la grille par deux cases rectangulaires : cela donne un carré approximatif.
2. Pour d'autres couleurs et d'autres commandes, voir la page

- "\33[1E" : va en début de la ligne suivante **A préférer à** $\backslash n$ **pour la suite**.

Voici un exemple

```
1  for (i=0;i<10;i++)
2    printf("\33[42m  ");
```

qui affiche à l'écran un liseret vert de 10 cases (chaque case étant composée de 2 espaces).

Q 1.5**.** Tester votre affichage pour différentes tailles et tirage au sort de position du fruit.

Exercice 2 : Codage des sections du serpent

Dans cet exercice, on va considérer un struct nommé `Serpent` qui va permettre de coder les informations concernant un serpent.

Q 2.1**.** Créer les fichiers `Serpent.h` et `Serpent.c` pour qu'un `Serpent` contienne les coordonnées de la tête du serpent et une liste chaînée des sections qui vont composer le serpent.
Créer les fichiers `Liste_Section.h` et `Liste_Section.c` qui permettent la manipulation d'une liste chaînée de sections du serpent. Une `Section` est composée de sa taille (en nombre de cases) et de la couleur de la section (i.e. une chaîne de caractères de taille 8 contenant deux cases de la couleur donnée par les séquences d'échappement).
Vous pouvez réutiliser les fichiers `liste_int.h` **et** `liste_int.c` que vous pouvez trouver sur l'ent avec ce document : il s'agit d'une liste contenant uniquement des entiers : il vous suffit d'utiliser "cherche et remplace" pour transformer une liste d'entiers en une liste de section.

Q 2.2**.** Adapter le `main` et `Makefile` à ces nouveaux fichiers.

Q 2.3**.** Implémenter les fonctions nécessaires à la gestion des sections du serpent pour créer une section, désallouer une section, créer une liste, ajouter une section en tête, ajouter en queue et désallouer une liste.
Vos fonctions doivent utiliser un nombre d'opérations borné par une constante (comme vu au cours 6 et TD 7).

Q 2.4**.** On veut tester la validité de nos fonctions. Pour cela, on ajoute au main un serpent avec des sections de tailles et couleurs variables (ajouteées en tête et en queue).

Q 2.5**.** Pour visualiser un serpent, on ajoute dans `Grille.h` et `Grille.c` une fonction `Grille_remplir` qui prend en paramètre une grille et un serpent et qui remplit la grille d'un serpent en représentant ses sections. On va ici le **dessiner horizontalement** : en effet, il s'agit ici simplement d'un test intermédiaire (appelé *test unitaire*) pour vérifier le bon fonctionnement de votre code.

`https://gist.github.com/fnky/458719343aabd01cfb17a3a4f7296797`. Il est à noter que ces séquences datent de l'époque ds téléscripteurs pour formater les pages papier issus des premiers terminaux.

Exercice 3 : Déplacer un serpent limité à une case

Laissons de côté pour cet exo la liste chaînée des sections du serpent. On va faire se déplacer un serpent limité à une case en utilisant les curseurs du clavier. On va pour cela utiliser la librairie `ncurses.h`[3]

Q 3.1. Téléchargez, regardez et exécutez le fichier `Exemple_clavier.c` qui vous propose un code d'exemple qui sera utile pour comprendre le fonctionnement du clavier avec `nurses`. Noter que pour compiler cet exemple (et donc votre code) vous devez effectuer la dernière étape de compilation avec l'option `-lncurses`.

Nous utilisons ici uniquement les aspects de gestion des interruptions système correspondant au clavier de ncurses (qui a de nombreuses autres fonctionnalités). Une interruption système désigne le fait qu'un programme est mis en pause par le micro-processeur quelques instants pour effectuer une autre tâche : ici cette autre tâche est de récupérer une éventuelle frappe sur le clavier.

Contrairement à `scanf`, nous allons utiliser `getch` qui renvoie un code dès qu'une touche a été utilisée sur le clavier. Si vous utilisez `getch` sans autre commande préalable, le programme est mis en attente jusqu'à ce qu'une touche soit appuyée. Mais dans l'exemple fourni, la commande `halfdelay(x)` force `getch` a s'interrompre au bout de `x` dixièmes de seconde[4]. Le programme retrouve alors la main pour faire autre chose : ici dans l'exemple, on affiche un compteur. On peut faire varier ce temps `x` pour avoir un jeu du serpent plus ou moins rapide pour le joueur.[5].

Le reste des commandes de `ncurses` utilisées dans l'exemple sont nécessaires pour faire fonctionner `getch` en lien avec la fenêtre du terminal. Pensez à toujours les mettre en début et fin de programme (si vous les oublier en fin de programme, le terminal va dysfonctionner par la suite).

Q 3.2. Créer les fichiers `Fonctions_Jeu.h` et `Fonctions_Jeu_1_serpent.c`. Ce fichier .h va contenir les entêtes des fonctions de gestion des jeux à coder dans les différentes parties du projet. Le code de cette partie est à mettre dans `Fonctions_Jeu_1_serpent.c`. Adapter le main pour qu'il contienne en ligne de commande : les dimensions de la grille, le temps de delai désiré pour accélérer/ralentir le jeu, ansi qu'une quatrième valeur indiquant à quel jouer veut jouer le joueur (pour l'instant, il n'y a que le jeu à un serpent).

Q 3.3. Créer dans `Fonctions_Jeu_1_serpent.c`, une fonction permettant de jouer au jeu du serpent (limité à une case). Pour cela, vous créerez une boucle similaire à celle du programme `Exemple_clavier.c` qui sera la structure événementielle du jeu. L'affichage à chaque itération

3. Cette librairie est installé sur les machines de Galilée. Vous pouvez l'installer sous linux en utilisant les commandes `sudo apt-get install libncurses5-dev libncursesw5-dev`. Si vous voulez utiliser des curseurs sous windows, un équivalent à `ncurses` existe avec `conio.h` que vous pouvez trouver ici par exemple `https://www.develop4fun.fr/la-librairie-conio-h-_kbhit-et-getch/`

4. Sous windows avec la librairie `conio.h`, la commande proche de `halfdelay` est `nodelay(win, TRUE);`

5. Remarquer que cette utilisation empêche de descendre en dessous du dixième de seconde pour la réactivité du programme : si c'est suffisant pour Snake, on pourrait avoir envie de davantage de réactivité pour d'autres jeux ou gestion d'un robot etc. Dans ce cas, il faut directement gérer les interruptions système avec des commandes système comme le propose par exemple les librairies SDL ou syst/ .... sous Linux

sera l'appel aux fonctions de `Grille.h` où la fonction `Grille_remplir` reçoit le serpent en paramètre.
A chaque pression d'une touche, il faut mettre à jour les coordonnées de la tête du serpent : ceci va simuler le déplacement du serpent lorsque la grille sera redessinée.

Q 3.4. Ajouter au jeu les actions lorsque le serpent heurte un bord : le programme s'arrête alors.

Q 3.5. Ajouter au jeu l'action de manger un fruit : cela a pour conséquence d'allonger le serpent par l'ajout d'une section de taille aléatoire (entre 1 et 5 cases par exemple) et d'une couleur aléatoire.

Exercice 4 : Codage du déplacement du serpent

Dans cet exercice, on veut gérer le déplacement d'un serpent dans toute sa longueur. Afin de dessiner et redessiner le serpent, il faut conserver en mémoire les emplacements où le serpent a changé de direction. Pour cela, on va ajouter dans le struct `Serpent` de `Serpent.h` une deuxième liste chaînée `Liste_Mouvement` qui contient les coordonnées d'une case et un type enum permettant de coder dans quelle direction (Haut, Bas, Gauche ou Droite) le serpent a obliqué en cette case.

Q 4.1. Créer cette liste chaînée, ajoutez la liste dans `Serpent`. Tester la liste en remplissant un Serpent avec deux ou trois mouvements.

Q 4.2. On veut à présent dessiner le serpent dans la grille. Les informations sont en partie dans la liste des sections et en partie dans la liste des mouvements.
Donnez l'algorithme permettant de dessiner le serpent en partant de sa case de tête en parcourant les deux listes en même temps.

Q 4.3. Tester votre algorithme en entrant "à la main" des données dans les deux listes.

Q 4.4. Ajouter les ajouts à la liste des mouvements dans l'animation de `Fonctions_Jeu_1_serpent.c`. Votre serpent devrait alors avoir un déplacement à chaque pression d'une flèche.

Exercice 5 : Jeu du serpent à un joueur

Pour finir le jeu, il reste à détecter les moments où le serpent se cogne contre lui-même. Pour cela, on va recalculer à chaque itération du jeu, une matrice de conflit qui va permettre de repérer les éléments placée dans la grille.

Q 5.1. On ajoute au programme une `Matrice` $n \times m$, nommée M, dont les éléments sont d'un type `enum element` (i.e. Rien, Fruit,...). Cette matrice indique si une case contient un fruit ou un élément du serpent (et d'autres informations que vous jugerez utiles dans la suite du projet). Décider d'un emplacement pour stocker la matrice.

Q 5.2. A chaque itération, la matrice M ne contient aucune information. En parallèle à l'algorithme de parcours des deux listes pour l'affichage, on remplit M en indiquant les cases

remplies : **c'est à ce moment qu'on peut détecter si le serpent s'est heurté lui-même !**

Q 5.3. Finissez le jeu et testez le !

2 Deuxième partie : Variantes

Dans cette deuxième partie, il vous est demandé de créer une variante au jeu du serpent classique que nous avons codé dans la partie 1. **Pensez déjà à bien faire fonctionner la partie 1 (et à en conserver une version intacte et fonctionnelle, prête à être montrée pour l'évaluation).**

Vous pouvez choisir une variante plus ou moins compliquée ! Vous serez jugée sur votre réalisation finale si elle fonctionne, mais, si elle n'est pas achevée ou fonctionne imparfaitement, sur votre démarcher pour y parvenir : n'hésitez donc pas à choisir une variante qui vous semble intéressante.

Voici quelques idées possibles pour vos variantes :

- Certains fruits peuvent donner (temporairement ou non) des pouvoir démoniaques comme manger une partie de l'autre serpent, épaissir le serpent, avoir un corps en accordéon...
- Faire jouer deux serpents sur la même grille : deux joueurs s'affrontent l'un avec les flèches du clavier, l'autre avec les touches z,q,d,x (qui sont placées correctement sur le clavier) : le gagnant est celui qui a mangé le plus de fruits... ou alors celui qui a touché l'autre...
- Créer une "IA" qui pilote le serpent : elle vise au plus court pour manger les fruits, mais de temps en temps, elle s'égare un peu aléatoirement. Puis faites la jouer en concurrence avec le serpent dirigé par un joueur. On peut mme faire jouer deux IA (différentes ou non) face à face.
- Faire jouer plusieurs serpents par des IAs dans le but que ces serpents ne se touchent jamais
- Et vous pouvez inventer vous-mêmes une variante à ce jeu...