Université Sorbonne Paris Nord

L1 Informatique
Année 2024-2025

Mini-projet du module
Programmation Impérative 2 (Info/DL)

Jeux de déplacement

Consignes

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 la semaine du 5 mai à votre chargé de TP.

Vous devez rendre par l'ENT, avant le dimanche 16 mai 2025 23h59 une archive contenant

- votre code (fichier .h, .c et Makefile uniquement) et éventuelles fichiers de données
- un fichier README.txt indiquant le nom des auteurs (votre monôme/binôme), description du programme, usage du programme, caractériques de votre code... et tout ce que vous jugez utile à transmettre avec rendu (difficultés rencontrées, bugs corrigés ou non etc).

Lors des séances de TP de la semaine du 12 mai 2025, vous devez venir présenter, lors d'une mini-soutenance,

- une exécution de votre code de la partie A du Mini-Projet
- répondre à des questions concernant des lignes de code précises et l'organisation globale du code.
- présenter l'état de votre partie B (qui est à rendre en fin de semaine)

Sur les 3 séances de 3h de TP, votre chargé de TP prendra des notes sur votre progression et pourra vous aider et vous guider.

Cadre du mini-projet

On s'intéresse aux jeux de déplacement où un pion (ou deux pions), appelé souvent personnage, se déplace à l'écran lorsque le joueur appuie sur les flèches du clavier [1]. On distingue deux types de jeux :

- le type 1 : où le pion ne bouge que si le joueur appuie sur une flèche : comme les jeux de plateaux (Super Mario...), de sport (Tennis, Billard...), les jeux de labyrinthe (Zelda...)
- le type 2 : où le pion est toujours en mouvement : le joueur donnant la direction de déplacement (Snake, Pacman...) et le pion continue selon cette direction.

On considère des jeux qui se déroulent dans une grille 2D rectangulaire de $n \times m$ cases. On dirige un (ou deux) pions représenté à l'écran par une case de couleur qui se déplace dans une zone cernée des murs ou dans un labyrinthe.

Ce projet se divise en deux parties :

- Partie A : cette partie est un TP classique avec des questions dans le but de construire un jeu de déplacement minimaliste : cette partie est à résenter achevée pour la dernière séance de TP.
- Partie B : sur la base de la partie A (à finir au plus tôt donc !), vous proposerez un jeu de déplacement plus élaboré : cette partie est à présenter (finie ou non) pour la dernière séance de TP.

Le rendu porte sur les deux parties.

1. Il peut y avoir deux pions également, déplacés par un deuxième joueur en utilisant des lettres du clavier comme E, S, F, X par exemple.

Exercice d'introduction

Exercice 1 : Prise en main d'un exemple ultra-minimaliste

Décompresser l'archive `Jeu_exemple.tgz` qui contient
- `README.txt` : qui donne les indications sur le jeu
- `Makefile`
- `main_jeu.c` : qui contient le main
- `Fonctions_jeu.h` et `Fonctions_jeu.c` qui contiennent les fonctions du jeu.

Liser le README qui vous indique comment installer et joueur. Installer le jeu et jouer. Vous pouvez constater que ce jeu est ultra-minimaliste (est-ce un jeu ?...) : en appuyant sur les flèches du clavier, le joueur fait changer les couleurs d'un carré et peut visualiser le temps qui passe depuis sa dernière action.

Le but de ce programme est de vous fournir les outils nécessaire à un jeu de mouvement même sans que le pion ne bouge ici.

Il y a trois outils utilisé ici
- les "séquences d'échappements ANSI' pour mettre de la couleur dans le terminal
- la libraire `ncurses` pour utilser les flèches du clavier
- le principe de la programmation événementiel qui est la base de la plupart des jeux de mouvements.

Séquences d'échappements ANSI

Pour afficher des couleurs dans le terminal, 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]
- "\33[1E" : va en début de la ligne suivante **A préférer à** $\backslash n$.

Q 1.1. *(Cette question n'a pas à être rédigée à l'écrit, mais sachez y répondre à l'oral).* Quelle est la ligne de code dans ce programme qui permet d'écrire un carré rouge et pourquoi écrit-elle un carré ?

2. Pour d'autres couleurs et d'autres commandes, voir la page `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.

Librairie ncurses

Pour utiliser les flèches du clavier, le programme utilise la librairie `ncurses.h` [3]

Attention pour utiliser cette librairie (comme pour math.h), il faut 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].

Attention, une étrangeté de ncurses fait qu'après la première utilisation de getch, l'écran devient noir... Il faut donc afficher après cette première utilisation.

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 1.2. *(Cette question n'a pas à être rédigée à l'écrit, mais sachez y répondre à l'oral).*
Quelle valeur retournée par getch correspond à l'absence de pression sur une touche pendant une itération du jeu ?

Programmation événementielle

Le principe de la programmation événementielle est de faire un programme qui scrute en permanence des évènements extérieurs au programme pour s'activer en réaction avec ces évènements.
Ici les vénements sont principalement le fait que le joueur appuie sur une touche que l'on récupère par la commande getch.

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

Pour mettre en œuvre la programmation évènementielle, on utilse une boucle qui engloble l'ensemble du cœur du programme. Cette boucle commence par récupérer la saisie éventuelle d'une touche par l'utilisateur. Si aucune touche n'a été appuyée, le programme se poursuit sans évènement particulier : par exemple, un pion poursuit sa course ; ou alors il ne se passe rien. Sun une touche a été pressée, les conséquences de cette pression sont calculées, changeant alors l'affichage pour représenter le déplacement d'un pion.

Q 1.3. *(Cette question n'a pas à être rédigée à l'écrit, mais sachez y répondre à l'oral).*
Dans le programme, comment sont différenciées les options 1 et 2 du jeu (voir le fichier README.txt pour connaître les 2 options).

Partie A : un jeu minimaliste

Le but du jeu de cette Partie A est de partir du jeu ultra-minimaliste fourni pour concevoir un jeu minimaliste où un pion se déplace dans une grille, est bloqué par des murs, doit éviter des pièges et doit atteindre un trésor.
Vous pouvez être tenté d'améliorer ce jeu dès à présent... Ce n'est pas interdit ! Mais attention, il faut que la Partie A marche avant la dernière séance de TP !!!

Exercice 2 : Codage de la grille

On veut coder une structure permettant de gérer et afficher la grille du jeu

Q 2.1. Créer un fichier `Grille.h` définissant :
- un type enum listant des élèments qui peuvent apparaître dans une grille de jeu : RIEN, MUR, PION, PIEGE...
- une structure nommée `Grille` contenant
  - deux entiers n et m donnant les dimensions de la grille rectangulaire
  - un tableau à deux dimensions remplit d'éléments du jeu

Q 2.2. Créer un fichier `Grille.c` définissant les fonctions (déclarées dans `Grille.h`) suivantes
- `Grille_initialiser` prenant en entrée un deux entiers n et m qui alloue une Grille retourne un pointeur sur une Grille. Le tableau de la grille est également alloué.
- `Grille_desallouer` qui désalloue la structure et son contenu.
- `Grille_vider` qui met l'élément RIEN partout dans la grille
- `Grille_redessiner` qui dessine à l'écran ce qui est codé dans le tableau de la Grille. Chaque case de la grille correspond à **deux espaces**[6] de fond d'une couleur différentes pour chaque élément.

Vous pouvee aussi ajouter d'autre fonction qui permette de placer un Pion ou un mur dans la grille.

Q 2.3. Tester vos fonctions en remplissant un tableau de Grille "à la main" et en l'affichant. Pour cela, vous pouvez partir du code du jeu minimaliste en ajoutant l'affichage de la grille à chaque itération de la boucle

Exercice 3 : Déplacement du pion

Q 3.1. Créer le fichier `Pion.h` et `Pion.c` qui permette de coder un pion. Les données principales sont l'emplacement `(x,y)` du pion, mais on peut utilement conserver aussi l'emplacement précédent `(x_old,y_old)` du pion pour le faire revenir à la place précédent s'il a foncé dans un mur.

Q 3.2. Créer des fonctions permettant d'allouer un pion, de le désallouer, de placer un pion à une case donnée.

6. 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.

Q 3.3. On ajoute un type enum permettant d'avoir les évènement du jeu concernant le pion : HAUT, BAS, DROITE, GAUCHE...
Créer une fonction `Pion_deplacer` qui, étend donné un Pion et un évènement, déplace le pion dans la direction indiquée, tout en sauvegardant la position précédente en cas de déplacement interdit détecté par la suite.

Q 3.4. Tester vos fonctions en ajoutant un Pion dans le programme qui sera manipulé par la bouche événementielle. A chaque appuie d'une touche, déplace le Pion dans ses données **et** le déplace dans le tableau de la grille. Ainsi, lorsque la grille sera déplacée, on verra le pion se déplacer

Q 3.5. Ajouter dans `Fonctions_jeu` une fonction permettant de réaliser le déplacement du pion. Pour cela
- un Pion se heurtant dans un mur, revient à la place précédente et s'immobilise
- un Pion passant sur un BUT (par exemple un trésor) récolte le BUT et poursuit son chemin
- un Pion tombant dans un piège... fait arrêter le jeu.

Exercice 4 : Avoir une jolie grille de jeu

Q 4.1. Ajouter une fonction `Grille_charger_fichier` qui retourne un pointeur sur Grille (alloué) qui est initialisée à partir d'un fichier dont le nom est passé en paramètre. Ce fichier donne l'aspect initial de la grille en début de jeu. Voici un exemple de fichier possible qui donne une grille 5 case par 10 cases contenant des murs tout autour de la grille, un pion en case (2,3), un trésor (B) et un piège (Q) à éviter.

```
5 10
MMMMMMMMMM
M P      M
M    Q   M
M      B M
MMMMMMMMMM
```

Q 4.2. Tester votre fonction en affichant des grilles différentes.

Exercice 5 : Finaliser le jeu minimaliste de la partie A

L'ensemble des fonctions et des struct manipulés jusqu'ici vous ont donné le jeu de partie A décrit en début de section.

Q 5.1. Tester votre jeu qui doit fonctionner autant en option 1 (jeu pas à pas) qu'en option 2 (jeu non stop).

Exercice 6 : (Bonus) Mise en place du UNDO

Dans cet exercice, on désire ajouter le fait qu'à tout moment, le joueur peut appuyer sur

la touche DEL (ou SUPPR ou Z) du clavier entraînant que le Pion revienne sur ses pas (et remette les trésors à sa place).

Cette fonction UNDO bien connue dans les éditeurs de texte, est plutôt rare dans les jeux ! Pourtant elle serait bien utile pour pouvoir revenir à un embranchement de labyrinthe pour continuer l'exploration ailleurs sans se faire manger par un monstre ; ou retenter sa chance pour faire un meilleur saut sur une plateforme.

Pour gérer cette fonction UNDO, il vous :

- créer une liste chaînée d'évènement où les évènements "inverses" à celui fait par le pion sont stochés en tête de liste
- et où un évènement UNDO provoque la suppression de l'évènement stockée en tête de liste et son applicatio.

Q 6.1**.** Tester votre processus UNDO dans différentes situations ! -

Partie B : A vous de jouer !

Dans cette deuxième partie, il vous est demandé de créer un jeu plus élaboré à partir de votre code de la partie A. **Pensez déjà à bien faire fonctionner la partie A (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 !
Si elle fonctionne, vous serez évalué sur votre réalisation finale.
Si elle n'est pas achevée ou fonctionne imparfaitement, vous serez évalué sur votre ambition et ce que vous aurez mis en œuvre pour essayer d'y parvenir : n'hésitez donc pas à choisir une variante qui vous semble intéressante.

Voici quelques idées possibles pour vos variantes :

- Pacman : Le pion doit manger tous les pilules d'un labyrinthe sans se cogner à des fantômes qui bougent ératiquement dans un coin du labyrinthe.
- Snake : Un serpent est au départ une case qui va s'allonger peu à peu. Des fruis apparaissent à des emplacements aléatoires. 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.
- Pong : deux joueurs font des échanges comme au ping-pong ou au tennis. Les pions sont des barres situées à gauche et à droite de l'écran. La balle part d'un côté et l'adversaire doit heurter la balle. Elle rebondit selon un angle dû à l'impact. Le premier à rater la balle a perdu.
- Zelda-like : un pion se déplace dans un labyrinthe où des objets sont visibles ou invisibles. Il faut trouver la sortie en actionnant ces objets, en évitant des pièges... et tout cela en un temps limités.
- Mario-like : un pion se déplace de plateaux en plateaux en allant à gauche, à droite ou en sautant. Il doit prendre de l'élan pour certains sauts. Et il peut récupérer des trésors en chemin. Le but est d'atteindre la sortie.
- ... et à vous de proposer d'autres idées !