

Programmation séquentielle

Puissance 4 – Travail évalué

Gregory Trollet, Hepia-ISC, 2024-2025

11.03.2025

But du travail pratique

Le but de ce travail pratique est de réaliser un jeu de puissance 4. Le jeu doit implémenter trois modes différents :

- un mode deux joueuses ;
- un mode un joueuse où l'ordinateur joue aléatoirement ;
- un mode un joueuse “intelligent-e” où l'ordinateur essaie de gagner ou de ne pas perdre.

Le rendu visuel du jeu sera réalisé en mode texte “ASCII Art” tel qu’illustré ci-dessous (vous pouvez copier-coller les symboles depuis la version text de cet énoncé :

Column number? (starts at 1):

4

			X			
		X	O			
O	X	X	O			
X	O	X	O			

1 2 3 4 5 6 7

Player one won!

En outre, une batterie de tests (**testbed**) permettant de valider que votre implémentation est correcte vous est fournie, de même que le programme fonctionnel.

Ainsi, vous pourrez vous assurer que votre implémentation produit exactement l’affichage souhaité. Ce dernier point est particulièrement important, car si l’affichage exact n’est pas respecté, tous les tests de validation échoueront.

Syntaxe du programme à implémenter

Le programme à implémenter doit impérativement respecter la syntaxe suivante :

```
Usage: puissance4 <mode> <row> <col>
    mode specifies the mode: 1 = single player game (random),
                             2 = single player game (AI), 3 = two players game
    row  specifies the number of rows (>= 4)
    col  specifies the number of columns (>= 4)
```

À noter que l’ordre des arguments est fixé. Voici un exemple d’exécution avec le mode deux joueuses sur une grille de 6x7 (lignes x colonnes) :

```
./puissance4 3 6 7
```

Un autre exemple avec une partie contre l’ordinateur “intelligent” sur une grille de 11x9 :

```
./puissance4 2 11 9
```

Règles du jeu

Le jeu se joue à deux. Chacun-e leur tour, les joueuses choisissent une colonne. Quand le choix est fait, une pastille “tombe” au bas de la colonne spécifiée. La première joueuse qui aligne quatre pastilles soit verticalement, soit horizontalement, soit en diagonale a gagné.

- Le plateau de jeu est composé d’un nombre arbitraire de lignes et colonnes ; le nombre de lignes ou colonnes doit être supérieur ou égal à quatre.
- À chaque joueuse correspond une pastille ; celle-ci est représentée par le caractère **X** pour la première et **O** pour la deuxième.
- Chacun-e leur tour les joueuses choisissent une colonne à l’aide du clavier ; la première colonne est numérotée 1.
- Si une colonne invalide est spécifiée, le programme doit re-demander à l’utilisateurice d’entrer une colonne (valide).
- Une fois une colonne sélectionnée, la pastille va se placer en bas de la colonne sur la première case non-occupée (si toute la colonne est occupée il est impossible de poser la pastille).
- La première joueuse à avoir aligné quatre pastilles horizontalement, verticalement, ou en diagonale, a gagné.
- Si toutes les cases sont remplies et qu’aucun-e joueuse n’a gagné la partie se termine par un match nul.

Mode ordinateur aléatoire

Programmer le jeu pour une seule personne face à l'ordinateur. Dans la première version de cette variante l'ordinateur jouera toujours au hasard.

Mode ordinateur “intelligent”

Dans une version plus sophistiquée l'ordinateur sera en mesure d'empêcher la victoire de son adversaire s'il gagne au prochain coup, ou gagnera s'il est en mesure de gagner au prochain coup. Dans tous les autres cas l'ordinateur jouera au hasard.

Cahier des charges

1. Programmer le jeu entre un humain et un ordinateur :
 - l'ordinateur jouera toujours un coup au hasard.
2. Programmer une “intelligence artificielle” un peu plus évoluée :
 - l'ordinateur gagnera s'il le peut au prochain coup ;
 - l'ordinateur empêchera la joueuse de gagner au prochain coup s'il le peut.
3. Programmer le jeu à deux joueuses humaines :
 - afficher l'état de la partie à l'écran où une case libre est représenté par un `\` (espace), et les cases de chacun des joueuses par un `X` ou un `O` respectivement. Le tableau de jeu sera affiché comme dans l'exemple fourni ;
 - la victoire doit être vérifiée après chaque coup ;
 - un match nul est possible si aucun-e des joueuses n'a gagné et qu'il n'y a plus de case libre.
4. Les paramètres du jeu sont lus sur la ligne de commande selon la syntaxe définie au paragraphe *Syntaxe du programme à implémenter*.

Important: à cause de la “mise en mémoire tampon” des chaînes de caractères lues ou affichées (avec `printf()` ou `scanf()`) vous aurez besoin de la fonction suivante pour vider cette mémoire tampon pour lire les entrées comme il faut :

```
void flush_input() {
    int c;
    while ((c = getchar()) != '\n' && c != EOF);
}
```

Critères d'évaluation

- Forme : qualité du dépôt Git, structure du code, documentation, **2 points**
- Tests : tests **unitaires** correctement mis en place, **1 point**
- Fonctionnalités : le code répond au cahier des charges et passe tous les tests automatisés, **4 points**

Ce travail aura un coefficient de 0.5 par rapport aux deux autres évaluations du semestre.

Le rendu est fixé au **vendredi 4 avril, 23h59**. Vous devez me partager votre dépôt avec comme accès *developer*.

Tests automatisés

Paramètres d'entrée

Il est très important de respecter le format de la ligne de commande comme indiquées ci-dessus.

Lae joueuse doit rentrer une colonne à la fois. On supposera qu'iel rentrera toujours un nombre entre 1 et le nombre de colonnes (pour simplifier).

Début de la partie

1. Si `M` et `N` correspondent au `row` et `col` des paramètres d'entrée. Vous devez afficher la taille du plateau de jeu sur la première ligne

Board size is `MxN` (rows x col)

2. Vous devez afficher en "ASCII Art" l'état du tableau au début de la partie :

```

|_|_|_|_|_|_|
|_|_|_|_|_|_|
|_|_|_|_|_|_|
|_|_|_|_|_|_|
|_|_|_|_|_|_|
|_|_|_|_|_|_|
|_|_|_|_|_|_|
|_|_|_|_|_|_|
1 2 3 4 5 6 7
```

En cours de partie

1. Lorsqu'un-e joueuse doit jouer, la chaine de caractères suivante doit être affichée :

Column number? (starts at 1):

2. Après chaque coup (que ce soit d'un-e joueuse ou de l'ordinateur), le tableau de jeu doit aussi être affiché :

x	0					
1	2	3	4	5	6	7

Victoire/défaite/match nul

Un certain nombre de tests automatisés sont fournis avec le code. Il est impératif que le format de sortie soit scrupuleusement respecté pour pouvoir passer les tests. En d'autres termes :

- Lorsque le joueur 1 a gagné, le programme affichera :

Player one won!

- Lorsque le joueur 2 a gagné, le programme affichera :

Player two won!

- Lorsque l'ordinateur a gagné, le programme affichera :

Computer won!

- Lorsque le résultat est un match nul, le programme affichera :

It is a draw.

Tests automatisés fournis

Un squelette comprenant les tests automatisés vous est fourni sur la page dans l'archive `skeleton_for_students.tar.gz`.

À l'intérieur du répertoire se trouve un **Makefile**. Celui-ci sera utilisé pour compiler votre programme et nettoyer les fichiers finaux et intermédiaires générés. Ce **Makefile** doit contenir les cibles suivantes :

- **puissance4**: cible par défaut permettant de générer l'exécutable du jeu ; le nom de l'exécutable doit impérativement être **puissance4** ;
- **clean**: cible permettant de supprimer l'exécutable et les fichiers intermédiaires générés (et aussi les fichiers candidats du **testbed**) ;

- **tests**: cible permettant d'exécuter la batterie de tests ; vous ne devriez pas avoir à modifier les règles de cette cible (c'est déjà fait pour vous).

Important: Il est impératif d'initialiser la graine de votre générateur de nombre aléatoire à 0 pour que les tests passent avec succès (`srand(0)`).

Il est **impératif que votre programme passe tous les tests !** Vous pouvez vérifier simplement ceci en exécutant “make tests” dans le répertoire racine du travail pratique. En cas d'exécution sans erreur, vous devriez obtenir un affichage similaire à ceci :

```
$ make tests
make -C testbed
make[1]: Entering directory 'TheDirOnMyLovelyMachine/testbed'
make -C 2players
make[2]: Entering directory 'TheDirOnMyLovelyMachine/testbed/2players'
=====[2 player tests]=====
test1
-----
test2
-----
test3
-----
test4
-----
test5
-----
make[2]: Leaving directory 'TheDirOnMyLovelyMachine/testbed/2players'
make -C rand_ai
make[2]: Entering directory 'TheDirOnMyLovelyMachine/testbed/rand_ai'
=====[random AI tests]=====
test1
-----
test2
-----
test3
-----
test4
-----
make[2]: Leaving directory 'TheDirOnMyLovelyMachine/testbed/rand_ai'
make -C smart_ai
make[2]: Entering directory 'TheDirOnMyLovelyMachine/testbed/smart_ai'
=====[smart AI tests]=====
test1
-----
test2
-----
```

```
test3
-----
test4
-----
test5
-----
make[2]: Leaving directory 'TheDirOnMyLovelyMachine/testbed/smart_ai'
make[1]: Leaving directory 'TheDirOnMyLovelyMachine/testbed'
```

Notes importantes

- Vous trouverez sur dans l'archive `skeleton_for_students.tar.gz` un exécutable (`puissance4`) qui devrait être conforme au standard que nous avons défini dans ce document. N'hésitez pas à l'exécuter pour vérifier le format de sortie.
- Écrivez un code aussi modulaire et réutilisable que possible.
- Mettez votre code dans plusieurs fichiers séparés ; le projet sera compilé avec le `Makefile` fourni dans le squelette.
- Essayez d'écrire vos propres tests pour chacune des fonctions que vous concevez.