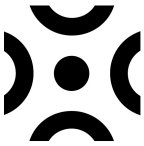
Initiation à la programmation en `C`

Jeu des Multi Dames

Licence 2 Info – 2025-2026

DM



1 Introduction du jeu *MultiDames*

MultiDames est un jeu de plateau inspiré des dames, qui se joue à **2 à 4 joueurs** sur un plateau carré (par exemple 8×8 cases). Le but est de marquer le plus de points possible en récupérant des pions grâce à des sauts successifs.

1.1 Mise en place

- Le plateau est initialement rempli de pions de couleurs : 34 pions blancs, 20 rouges et 10 noirs.
- Chaque joueur commence avec un **score nul**.
- Au premier tour, chaque joueur retire un pion blanc de son choix du plateau et le compte dans son score.

1.2 Déroulement d'une partie

La partie se joue en tours successifs. Au tour d'un joueur, trois situations sont possibles :

1. **Saut de pions (si possible)**
   - Le joueur choisit un pion (peu importe sa couleur) permettant au moins un saut.
   - Si ce pion peut sauter au moins un autre pion, il **doit** commencer une suite de sauts jusqu'à qu'aucun autre saut ne soit possible.
   - Chaque saut suit la règle :
     - le pion actif saute par-dessus un pion adjacent (horizontalement, verticalement ou en diagonale),
     - la case opposée doit être libre,
     - le pion sauté est retiré du plateau et sa valeur est ajoutée au score du joueur.
2. **Arrêt volontaire**
   - Un joueur peut choisir de s'arrêter (c'est-à-dire de ne plus participer aux tours suivants).
   - Mais il ne peut s'arrêter que s'il reste **au moins un autre joueur actif** dans la partie.
3. **Fin de la partie**
   - Si, au début du tour d'un joueur, aucun pion du plateau ne peut sauter, la partie s'arrête immédiatement.
   - Dans ce cas, le joueur courant subit une pénalité : la somme des valeurs des pions restants sur le plateau est soustraite de son score.

Les pions blancs sont de valeur 1, les rouges de valeurs 5 et les noires de valeurs 8.

1.3 Victoire

Le joueur ayant le score le plus élevé gagne la partie.

2 Conception

Le programme à produire est à réaliser dans un unique fichier `multi_dames.c` et doit permettre de jouer au jeu entre 2 et 4 joueurs. Ce dernier doit être compilé avec les options de compilation présentées en cours, à savoir `-Wall` et `-std=c17`. Ces options doivent vous accompagner tout au long de la réalisation du DM.

2.1 Interface utilisateur

L'utilisateur est considéré comme bienveillant et ne tentera pas volontairement de faire planter le programme. Il est cependant nécessaire de lui indiquer le format de ce qu'il doit taper, nombre, caractère ou chaine de caractères. On pourra tout à fait se restreindre à l'utilisation d'entiers comme seules interactions avec l'utilisateur : deux valeurs de 1 à 8 (ou 0 à 7) pour une position, un 0 ou 1 pour une question oui/non.

Malgré sa bienveillance, l'utilisateur peut également se tromper et entrer une valeur incorrecte bien que dans le format attendue (position invalide, saut impossible...).

L'affichage est libre. Voici un exemple extrait (présenté en trois colonnes) de l'exécution d'un prototype pour ce jeu :

```
Score:
    J1  J2  J3  J4
    1   1   1   1
Tour: 2
Joueur 1 (1)
Plateau:
   1 2 3 4 5 6 7 8
  ----------------
1 |  x + o o + + x
2 |+ x x + + + x +
3 |x +   + x x x +
4 |x + +   x x + o
5 |+ + + +   o + +
6 |x + o x x + x +
7 |+ + x o + x x o
8 |o o + + x + + o
```

```
Arrêter ? (1=oui, 0=non) 0
Position du pion sauteur ?
1 3

   1 2 3 4 5 6 7 8
  ----------------
1 |  x[+]o o + + x
2 |+ x x + + + x +
3 |x +   + x x x +
4 |x + +   x x + o
5 |+ + + +   o + +
6 |x + o x x + x +
7 |+ + x o + x x o
8 |o o + + x + + o
Sauts possibles: 1 1, 3 3.
Entrer un saut: 3 3
```

```
   1 2 3 4 5 6 7 8
  ----------------
1 |  x   o o + + x
2 |+ x   + + + x +
3 |x +[+]+ x x x +
4 |x + +   x x + o
5 |+ + + +   o + +
6 |x + o x x + x +
7 |+ + x o + x x o
8 |o o + + x + + o
...
```

2.2 Structures de données

Pour réaliser ce programme, on a besoin de structure de données représentant l'état du jeu.

Cela passe dans un premiet temps par la représentation de l'état du plateau.

```c
#define TAILLE  8
typedef struct {
    // 0 si vide, 1 si blanc, 2 si rouge et 3 si noir
    int pion[TAILLE][TAILLE];
} Plateau;
```

Chaque position sur le plateau est vide ou possède un pion de couleur donnéee.

Pendant la partie, il faut retenir le score de chaque joueur ainsi que son état, s'il est toujours dans la partie ou bien s'il a décidé de s'arrêter.

```c
typedef struct {
    int etat;  // 1 si dans la partie, 0 sinon
    int score;
} Joueur;
```

Enfin, l'état du jeu est représenté par l'état du plateau, celui des joueurs mais aussi le nombre de ces derniers, le joueur à qui c'est le tour et enfin un compteur de tour. On a en effet de distinguer le premier tour (où les joueurs retirent chacun un pion sans pouvoir s'arrêter) des tours suivants. À partir du second tour, le joueur en cours peut saisir un pion pour le faire sauter de case vide en case vide. On propose de compléter l'état du jeu par la position de ce pion s'il est saisi.

```c
#define MAX_JOUEURS  4
typedef struct {
    Plateau plateau;
    Joueur joueur[MAX_JOUEURS];
    int nb_joueurs;
    int joueur_courant;
    int tour;
    int pion_est_saisi; // 1 si un pion est saisi, 0 sinon
    int pion_i, pion_j; // la ligne et colonne du pion saisi (0 sinon)
} Jeu;
```

2.3 Fonctions

Il est attendu et fortement recommandé de découper le programme en fonctions. Parmi ces fonctions, les suivantes **doivent être présentes et fonctionnelles** :

- `int jeu_capturer(Jeu *jeu, int i, int j)` : le joueur courant capture la pièce se trouvant à la ligne i et la colonne j (numéroté entre 0 et 7)
- `int jeu_saisir_pion(Jeu *jeu, int i, int j)` : utilisée en début de tour pour saisir le pion permettant de faire des saut (par le joueur courant)
- `int jeu_sauter_vers(Jeu *jeu, int i, int j)` : fait sauter le pion saisi vers la case (vide) en position (i,j) en capturant le pion entre les deux positions au passage
- `int jeu_arreter(Jeu *jeu)` : utilisée en début de tour pour arrêter de jouer (tant que le joueur courant n'est pas le dernier joueur)
- `int jeu_joueur_suivant(Jeu *jeu)` : passe au joueur suivant
- `void jeu_charger(Jeu *jeu)` et `void jeu_ecrire(Jeu *jeu)` : fonctions pour les tests présentés dans la section suivante

Ces fonctions changent l'état du jeu et, à l'exception des fonctions de tests, elles **ne doivent pas utiliser les entrées sorties (`scanf`/`printf`)**. Ces fonctions doivent renvoyer un code d'erreur :

- 0 si l'action à effectuer n'est pas possible (position invalide, saut impossible, dernier joueur)
- 1 si tout s'est bien passé

Vous êtes libres (et encouragés) à découper votre code plus en profondeur. Par exemple, on *pourrait* avoir les fonctions :

- `int plateau_pion_peut_sauter(Plateau *plateau, int i, int j)` pour vérifier si un pion peut sauter.
- `int case_est_valide(int i, int j);` pour vérfier si une position désigne un case dans le plateau et non à l'extérieur
- `void jeu_afficher(Jeu *jeu)` pour afficher le plateau à l'utilisateur
- `void jeu_capturer_pion(Jeu *jeu, int i, int j)` pour que le joueur courant capture un pion

3 Tests

Afin de tester vos fonctions et votre programme, des tests automatiques vont être mis en place sur Platon. Un test automatique remplacera la fonction `main` de votre programme et pourra par exemple être de la forme :

```c
int main(void) {
    Jeu jeu;
    int i, j;
    jeu_charger(&jeu);
    scanf("%d%d", &i, &j);
    if (!jeu_sauter_vers(&jeu, i, j)) {
        printf("Action impossible\n");
    }
    jeu_ecrire(&jeu);
    return 0;
}
```

Le programme sera exécuté avec comme entrée l'état du jeu dans le format textuel suivi d'une position. Le sortie obtenue sera comparée à celle attendue.

Les fonctions testés sont celles exigées dans la partie précédente

Les fonctions `void jeu_charger(Jeu *jeu)` et `void jeu_ecrire(Jeu *jeu)` sont nécessaires pour ces tests. Ces deux fonctions liront et écriront depuis/vers les flux standards d'entrée/sortie l'état du jeu sous forme textuel. (Elles ne servent cependant pas à communiquer avec le joueur)

Le format textuel imposé est le suivant :

```
<nb joueurs> <tour> <joueur courant>
<etat J0> <score J0>
<etat J1> <score J1>
...
<pion est saisi> <pion i> <pion j>
<p_00> <p_01> <p_02> ... <p_07>
...
<p_70>          ...        <p_77>
```

où `p_ij` désigne le contenu de la case à la ligne `i` et la colonne `j`.

Ci-contre à droite, un exemple pour une partie en cours de 4 joueurs. Le 3e joueur s'est arrêté, et le 4e est en train de faire sauter le pion se trouvant à la position `(4,2)` (5^e ligne et 3^e colonne).

```
4 3 3
1 15
1 12
0 14
1 4
1 4 2
2 2 3 1 1 2 3 1
3 2 1 3 1 0 0 3
2 1 1 1 1 1 0 2
3 1 1 0 3 2 2 1
2 3 1 0 0 2 1 2
1 0 0 0 0 1 0 2
0 1 2 0 0 0 0 2
1 2 2 0 1 0 2 0
```

L'état du jeu est donc donné par une suite de `3 + 2 * nb_joueurs + 3 + 64` entiers positifs ou nuls.

4 Consignes de rendu

Le devoir maison est à réaliser **en binôme au sein d'un même groupe de TP**. Il doit être rendu sous la forme d'archive ZIP avant le **31 octobre 2025 16:00**.

L'archive doit contenir :

- Un seul fichier source `multi_dames.c`
- Un fichier texte `LisezMoi` indiquant :
  - votre groupe de TP
  - vos noms
  - comment compiler le programme
  - comment jouer
  - les difficultés rencontrées et comment elles ont été surmontés (le cas échéant)

5 Évaluations

Ce DM est évalué en deux temps. La première évaluation est basé sur le rendu et est automatique. La seconde évaluation sera sous la forme d'un TP noté.

5.1 Tests automatiques

Ce rendu ne fait l'objet que d'une évaluation automatique testant le bon format du rendu et le bon fonctionnement des fonctions de votre programme. Aucune correction manuelle n'interviendra : si une fonction n'est pas conforme aux spécifications (signature, comportement attendu, format d'E/S), les tests associés échoueront.

Le respect strict des conventions indiquées (types, noms de fonctions, format des données en entrée/sortie) est donc indispensable pour valider les tests correspondants.

Une archive invalide, un fichier mal nommé ou un programme ne compilant pas sera considéré comme un rendu nul et ne fera l'objet d'aucun test.

5.2 TP Noté

Ce DM constitue un travail préparatoire au prochain TP noté. Il a pour objectif de vous familiariser avec la manipulation des tableaux, des structures, ainsi que l'organisation d'un projet C en plusieurs fonctions.

Le TP noté sera l'occasion de réinvestir ces compétences à travers d'exercices de programmation relatifs au jeu MultiDames, par exemple :

- fonctions de manipulation du jeu
- extensions possibles des règles
- variantes d'implémentation

En d'autres termes, le bon déroulement du TP noté suppose d'avoir réellement réalisé ce DM : il s'agit d'une étape intermédiaire indispensable dans votre progression.