Prog1
Projet de Programmation

Préambule

Ceci est le sujet pour vous guider dans la réalisation de votre projet de programmation que vous allez réaliser au cours des dernières séances de TP de ce cours.
Vous pouvez vous associer en **trinôme** pour la réalisation de ce projet. Vous devez communiquer au professeur la composition de ces binômes avant le 1er Décembre

Le projet sera à rendre via moodle avant le 31 Décembre à minuit. Un rendu de contrôle d'avancement vous sera demandé à mi-parcours avant la séance du 11 Décembre.
Sous réserve de problèmes liées aux conditions sanitaires, il y aura des soutenances début Janvier au cours desquelles vous montrerez le résultat de votre projet et vous serez questionnés sur le contenu de votre code.
Lors des séances de TP, vous devez nécessairement vous connecter sur discord. Nous avons créé des salons vocaux d'étude pour chaque groupe. Tous les membres d'un groupe de projet DEVRONT OBLIGATOIREMENT choisir un salon pour chaque groupe de projet et s'y connecter durant toute la séance. Les chargés de TP pourront venir vous voir dans ces salons et vous demander à voir votre avancement, comme ils le feraient en séance de TP traditionnels en salle machine. Quand vous aurez besoin d'assistance, vous envoyer sur le chat textuel de votre groupe un message et le chargé de TP viendra vous voir dans votre salon privé d'étude.

1 Présentation

Le but du projet est de programmer un jeu de Dames dans le terminal. Si vous ne connaissez pas ou avez oublié les règles du jeu, aller sur wikipedia. Vous pouvez aussi trouver facilement des jeux gratuits sur portable ou sur internet.
Un exemple d'exécutable codé par mes soins est disponible sur moodle (une version pour windows, une pour linux et une pour mac, prenez celle qui correspond à votre système). Testez le pour vous donner une idée de ce que vous allez devoir coder.
Je vous fournis aussi quelques fichiers dans lesquels vous allez coder.

- le fichier `structures.c` contient tous les types `struct` et `enum` nécessaires. Tout est déjà codé et il ne faudra pas modifier le code de ce fichier (vous pouvez a la limite ajouter des choses si vous le jugez nécessaire).
- le fichier `affichage.c` contient des fonctions déjà écrites pour vous faire gagner du temps et qui permettent notamment d'afficher dans le terminal le damier de jeu. A priori, vous n'aurez pas à la modifier dans un premier temps (sauf a vouloir customiser l'apparence du jeu). Le fichier `couleurs.c` est un fichier contenant des noms de constantes pour les couleurs affichables dans un terminal. elles sont utilisées dans `affichage.c`
- Le fichier `gameplay.c` est l'endroit où vous allez passer le plus de temps. Il contient tout une liste de fonctions que vous aurez à modifier et dont le rôle attendu est décrit en commentaires avant chaque fonction. **Vous êtes libres d'ajouter toute fonction auxiliaire que vous jugerez utile**.
- le fichier `interfaceJeu.c` contient des fonctions qui ont trait aux menus et déroulement du jeu. Vous y travaillerez une fois que `gameplay.c` sera avancé.
- Le fichier `main.c` contiendra l'exécutable final faisant appel aux fonctions de `interfaceJeu.c`. Avant cela, ce main vous servira à tester vos fonctions. Un fichier `tests.c` contient des méthodes de tests pour la première semaine, vous pourrez aussi procéder de cette façon pour chaque semaine.

Vous trouverez des infos supplémentaires sur ces fichiers à la section 3.2 de ce document.

A propos de l'organisation des fichiers et de la compilation Afin de maintenir une lisibilité satisfaisante du code d'un programme, il est fréquent de découper le programme en plusieurs fichiers comme ci dessus. Comme nous n'avons pas encore vu en cours la programmation modulaire, et pour éviter les complications, j'ai opté pour une solution plus simple mais pas forcément orthodoxe et conforme aux usages recommandés en C. Ici, pas de fichier header .h (voir cours sur compilation modulaire), et donc **le fichier `main.c` sera le seul a devoir être compilé**, il inclut les autres fichiers dans son préambule via la directive `#include` qui agit comme un copié collé du fichier passé en argument.

A propos du travail a plusieurs Bien sur les contraintes liées au confinement ne faciliteront pas le travail a plusieurs. Je vous autorise donc a ne pas vous mettre en trinôme si vous ls désirez. Si vous travaillez à plusieurs, il vous faudra surement bien vous organiser et utiliser discord pour faire du partage d'écran. Il est préférable que chacun ait les fichiers sur son ordi pour pouvoir travailler entre les séances. Donc, même si un seul d'entre vous montre son écran, les autres recopient le code pour tout avoir aussi sur leur machine. A partir de la semaine 2, où certaines fonctions plus difficiles sont à coder, il peut être intelligent aussi de se repartir les taches, chacun bosse sur une fonction différente et vous faites un point tous les quart d'heure pour suivre votre avancement par exemple.

2 Plan de développement

Vous avez quatre TP que vous pourrez consacrer à l'avancement de votre projet. Bien sûr, pendant ces TP vous pouvez demander de l'assistance aux chargés de TP, mais comme le rendu est noté, vous n'aurez évidemment pas de correction toute faite pour telle ou telle fonction.
Dans les différents fichiers sont donnés des en-têtes de fonctions qu'il faut compléter qui sont précédés d'un commentaire indiquant ce que la fonction est censée faire ainsi que le numéro de semaine S1,S2,... pour laquelle ce code devrait être écrit. Vous pouvez bien sur aller plus vite. Pour les fonctions qui retournent autre chose que `void`, et afin de ne pas provoquer une tonne de Warnings a la compilation, j'ai souvent mis une valeur par défaut (comme `return -1` pour une fonction qui retourne `int`).

Pour commencer Pour commencer je vous conseille de bien lire ce document en entier en examinant les fichiers fournis, en commençant par le fichier `structures.c`. En commentaires j'ai inscrit le rôle de chaque structure. Plus de détails se trouvent à la section 3.2.
Ensuite allez voir le fichier `gameplay.c`. Vous remarquerez qu'afin de vous aider à démarrer, j'ai écrit le code des premières fonctions. Remarquez son code qui fait usage de la façon unidimensionnelle choisie pour représenter le tableau.

Progression possible La progression que j'imagine est la suivante
- Semaine 1 : Lecture du sujet. Se familiariser avec les structures. Ecriture de toutes les fonctions de base get/set/is et la fonction `isMovePionValide`.
- Semaine 2 : is MangePion Valide et ses variantes pour la Dame. Comprehension du type `move` a base de liste chainées.
- Semaine 3 : les move et l'interfaceJeu pour arriver a un jeu a peu pres fonctionnel pour deux joueurs humains
- Semaine 4 : la partie calcul du meilleur move dans `gameplay.c` et eventuellement joueur CPU.
- Ensuite des tests pour peaufiner votre projet avant la remise.

3 Conseils

3.1 Progression

Je vous conseille de ne pas essayer de tout coder tout de suite et de suivre le plan proposé. En particulier :
- ne pas vouloir faire le menu du début ou ce genre de choses, concentrez vous sur les fonctions demandées qui permettront de faire une interface de jeu ensuite.
- il vaut mieux se concentrer si les pions d'abord, et si vous n'arrivez pas a coder les fonctions relatives aux dames ; passer à la suite pour y revenir ensuite.
- ne pas vouloir coder le joueur CPU. Déjà si ca marche avec deux joueurs humains c'est bien.
- Dans les règles du jeu de dames, une règle importante est qu'un joueur doit toujours faire un coup dans lequel il capture des pièces adverses s'il le peut (et même il doit jouer le coup qui capture le plus de pièces possibles). Ceci nécessite de coder une fonction qui calcule le meilleur coup possible et je vous conseille de garder ça pour la semaine 3 ou 4 aussi.

3.2 Quelques détails sur les fichiers déjà fournis

Ici je vous explique avec un peu plus de détail que dans les commentaires les codes fournis.
- La structure centrale pour lancer un jeu est `jeu`. Elle contient comme champs :

- deux entiers `NB_Lig` et `NB_Col` pour les dimensions du plateau. Un entier `NB_LigInit` qui designe le nombre de lignes avec des pions en début de partie. Le damier typique sur lequel on joue en France est 10/10/4
- un pointeur `plateau` vers le tableau représentant le plateau de jeu. Attention on a choisi de modéliser le plateau comme un tableau de dimension 1. Ce tableau a tailleNB_Lig*NB_Col : les `NB_Col` premières cases représentent la première ligne (celle du bas du plateau), puis les `NB_Col` suivantes la deuxieme ligne, etc.. Par exemple la case en haut a gauche a pour indice `NB_Col(NB*Lig-1)`. Voir la fonction `getCaseVal` par exemple.
- un tableau `tabJ` de taille 2 contenant les deux struct joueur
- un entier `jCourant` qui vaudra a un moment donné de l'execution 0 ou 1 et qui désignera l'indice (dans le tableau `tabJ`) du joueur dont c'est le tour actuellement.

L'objet `jeu` sera crée au lancement du programme, et nous le manipulerons dans les fonctions via un pointeur afin de ne pas faire des copies inutiles. Ainsi si ce pointeur est `jeu* g`, on accèdera au joueur courant par l'instruction `g->tabJ[g->jCourant]` (une fonction `getJoueurCourant` déjà ecrite dans `gameplay.c` retourne exactement ceci).

- Le plateau est stocké sous la forme d'un tableau de `caseVal`. Ce type enuméré deja donné indique les valeurs possibles pour une case du plateau : `Vide`, `PB` et `DB` pour un pion blanc ou une dame blanche, `PN` et `DN` pour un pion noir ou une dame noire. Il y a aussi `PBC`,`PNC`,`DBC`,`DNC` qui servent a noter les pieces capturées. Ces valeurs seront utiles quand on testera des mouvements de prise : les pieces capturées ne sont pas supprimées tout de suite, elles sont temporairement marquées avec ces valeurs et ne sont supprimées du plateau qu'à la fin du tour quand le mouvement est bien validé. Notez que les valeurs du type `caseVal` sont énumérées dans un ordre qui fait que `Vide=0` et que les versions capturées ont une valeur opposée à celle de leur équivalent non capturé (pa exemple `PBC` vaut -1 et `PB` vaut 1) .
- On a aussi des types `couleur` et `typePiece` qui sont utiles pour des fonctions qui interrogent la valeur des pièces présentes. Le type `position` désigne juste des coordonnées du plateau et sont pratiques pour écrire les fonctions de `gampleay.c`.
- Enfin les structures `etape` et `move` permettent de manipuler les propositions de mouvements de pièce. Comme un mouvement sur le plateau a une taille varaible (si on mange plusieurs pièces adverses par exemple) il s'agit d'une structure de liste chainée. Une `etape` est la donnée d'une `position` et d'un pointeur vers l'`etape` suivante. Un `move` est alors la donnée d'une premiere etape `first` et d'une etape de fin `fin`. Un `move` a aussi un champ `score` qui permet d'avoir accès directement au nombre de pièce capturées sans reparcourir la liste. Un mouvement simple sans prise aura un score de `0`.

3.3 A propos de l'affichage

Au début du fichier `affichage.c` se trouve une ligne

```
1 //#define AFFICHAGEJOLI
```

Si vous enlevez les `//`, ce ne sera plus un commentaire et ca changera l'apparence des pieces pour les remplacer par des icônes plus jolies. Mais il est fort possible que selon votre système (en particulier sous Windows) et son paramétrage, cela cause des problèmes d'affichage. Essayez et remettez la ligne en commentaire si vous constatez un affichage peu lisible.

3.4 Tests

Vous le savez, il est fortement recommandé de bien tester les fonctions que l'on écrit au fur et à mesure de l'avancement. Une bonne pratique est même de faire des fonctions dédiées au test. Dans le fichiers `tests.c` vous trouverez des fonctions qui permettent de tester si vos fonctions déjà écrites fonctionnent. J'ai mis un exemple de telle fonction `testsS1()` que vous pouvez regarder et qui permet de tester certaines des fonctions à coder au TP1. vous n'avez qu'à appeler cette fonction dans votre main pour voir ce qui se passe. Je vous recommande de faire ainsi pour chaque semaine par la suite.