SINF1252 Projet 1: jeu de dames

2015

1 Énoncé

1.1 Description du projet

Lors de ce premier projet, vous devez implémenter, en langage C, un jeu de dames en mode console. Pour ce faire, le squelette du programme (que vous **devez** utiliser) vous est fourni.

Dans le squelette, vous trouverez deux grandes parties: le programme principal (`programme.c`) et le système de jeu (`dames.h` et `dames.c`). Le programme principal a pour but de gérer l'interaction entre les joueurs et le système de jeu, c'est dans cette partie que vous devez lire au clavier les coups que le joueur veut effectuer, lui communiquer l'état du damier, etc. Cette partie du programme ne peut donc pas mémoriser d'information relative à l'état de la partie (quel joueur doit jouer, position des pions, etc), ces informations doivent se trouver dans le système de jeu. Le système de jeu (`dames.c` et `dames.h`) doit, lui, exposer au programme principal les fonctions permettant de jouer (déplacement, charger une partie, etc.). La liste des ces fonctions est décrite dans `dames.h`, fichier que vous ne **pouvez <u>strictement</u> pas** modifier.

Voici une liste des structures et fonctions documentées dans `dames.h`, référez-vous au fichier en question pour plus de détails concernant le contenu des structures et la signature des fonctions.

- `struct game`: représente les informations relatives au jeu: l'état du damier, les mouvements effectués, le joueur actuel
- `struct coord`: représente des coordonnées (x, y)
- `struct move`: représente un mouvement
- `struct move_seq`: représente une séquence au sein d'un mouvement (voir ci-dessous pour une explication plus détaillée)
- `new_game()`: créer un nouveau jeu
- `load_game()`: charge un jeu à partir d'un damier donné

- `free_game()`: libère les ressources associées à un jeu
- `apply_moves()`: applique une série de mouvements à un jeu
- `is_move_seq_valid()`: vérifie si une séquence de mouvement est valide
- `undo_moves()`: annule les n derniers coups joués et restaure l'état du damier. Il faut donc implémenter un système qui mémorise les coups joués **à l'aide d'une liste chaînée** (champ `moves` dans `struct game`).
- `print_board()`: affiche l'état du damier sur la console

La fonction permettant de jouer un coup (`apply_moves`) prend une liste chaînée en paramètre, liste que vous devez construire en vous basant sur les structures `move` et `move_seq` qui vous permettent de passer en paramètre l'ensemble des déplacements (case par case) d'un pion lorsque vous jouez un coup. Cette liste chainée doit être construite **dans l'ordre**: le premier élément de la liste est le premier coup à jouer, etc.

1.2 Description du damier

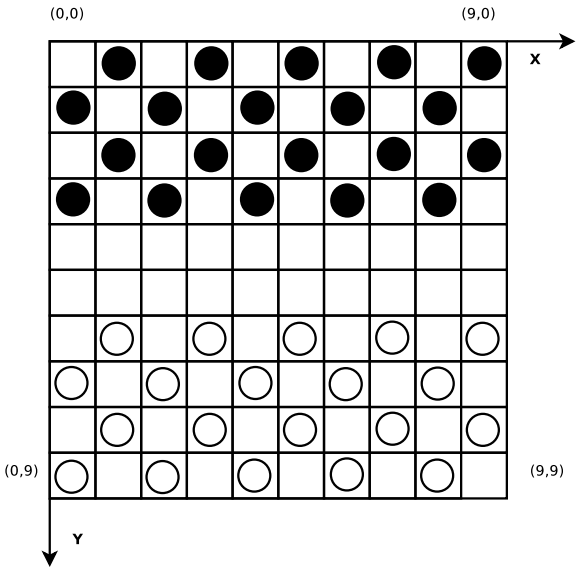

Figure 1: Damier initial

Le damier (champ `board` dans `struct game`) se représente comme une matrice dont les axes des abscisses et des ordonnées sont numérotés de 0 à 9 en partant depuis le coin supérieur gauche du damier. En début de partie, les blancs se trouvent toujours en dessous alors que les noirs se trouvent au dessus. Les coordonnées d'une case se représente de manière suivante : `(abscisse,ordonnée)`. La

case tout en haut à gauche a donc les coordonnées $(0, 0)$, la case en bas à gauche $(0, 9)$, en bas à droite $(9, 9)$. Voyez la Fig. 1 pour une illustration.

Chaque case est représentée par une valeur entière v (`int`) dont la signification est la suivante:
Soit `00000CTP` représentant les 8 bits de poids faible de v. Les trois derniers bits ont la signification suivante:

- Le bit C est le bit de couleur: 0 = noir, 1 = blanc.
- Le bit T est le bit de type de pièce: 0 = pion, 1 = dame.
- Le bit P est le bit de présence: 0 = case vide, 1 = case remplie.

Ainsi, une case où se trouve un pion blanc aura comme valeur `00000101 (= 0x5)` et une dame noire aura `00000011 (= 0x3)`. Les bits autres que les bits `C, T, P` sont toujours à 0.

1.2.1 Définition des mouvements et des séquences

Une séquence est le déplacement d'un pion ou d'une dame d'une case vers une autre, par exemple le déplacement de $(0, 0)$ vers $(1, 1)$. Une séquence est représentée par la structure `mov_seq` contenant les coordonnées de départ (`c_old`) et d'arrivée (`c_new`). Un mouvement pouvant comporter de multiples prises, le pion peut donc passer par plusieurs cases. Pour ce faire, un mouvement (structure `move`) comprend une liste chaînée de séquences. Le mouvement (`struct move`) $(0, 0)$ vers $(2, 2)$ vers $(4, 0)$ comprendra donc deux `move_seq` : un $(0, 0)$ vers $(2, 2)$ et un autre $(2, 2)$ vers $(4, 0)$ (cela signifie donc que deux pions ennemis en $(1, 1)$ et en $(3, 1)$ ont été capturés).

1.2.2 Exemples

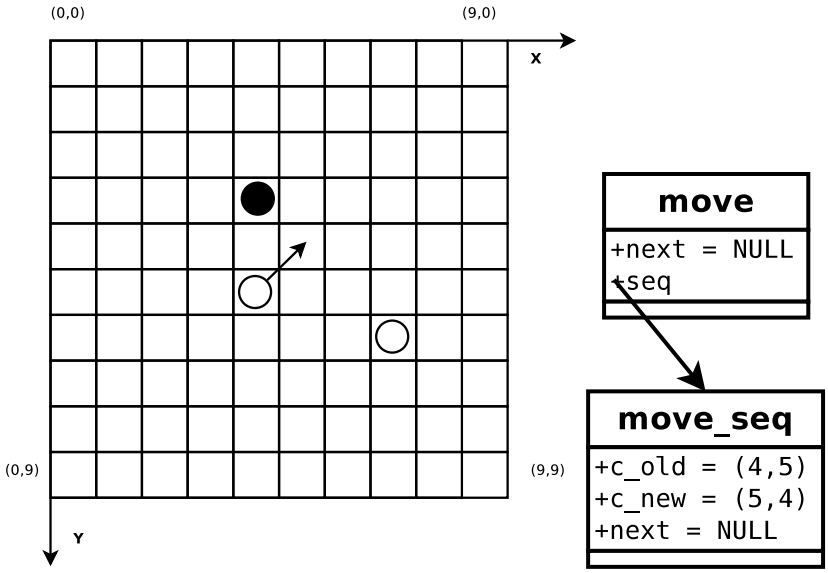

Figure 2: Mouvement et description des structures à passer à apply_moves

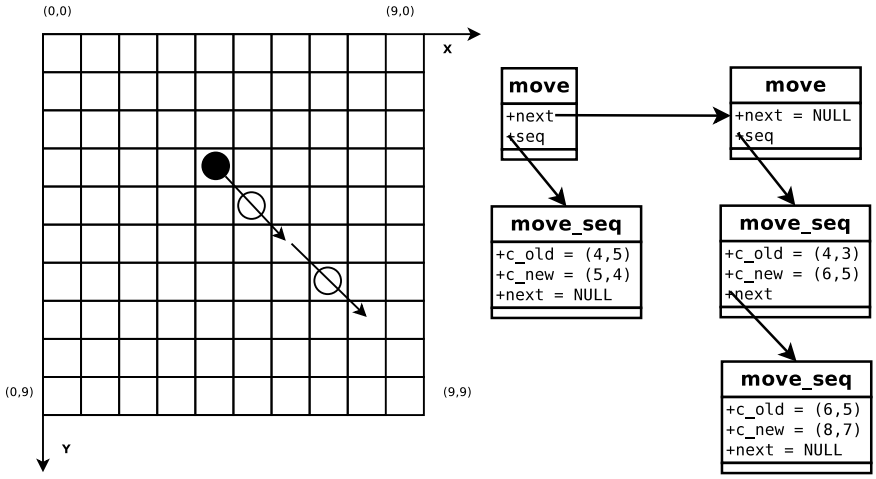

Figure 3: Suite de la Fig. 2: double capture et description des structures à passer à apply_moves

1.3 Règles du jeu

Les règles sont les règles du jeu de dames international, et sont disponibles ici : `http://www.ffjd.fr/Web/index.php?page=reglesdujeu`. Vous ne devez pas respecter le point 10 rendant la prise obligatoire, ainsi que les règles

de match nul. On considère qu'un joueur est gagnant si l'autre joueur n'a plus de pièces ou que l'autre joueur a le trait mais ne peut plus effectuer de mouvement valide. Tout mouvement effectué après le mouvement donnant la victoire à l'un ou l'autre joueur ne doit pas être pris en compte.

1.4 Tests

Afin de prouver le bon fonctionnement de votre implémentation, nous vous demandons de fournir un fichier test.c contenant les tests que vous avez effectués pour valider votre implémentation. Vous devez utiliser la librairie CUnit [1]. Voyez le chapitre du cours correspondant pour plus de détails : `http://sites.uclouvain.be/SystInfo/notes/Outils/html/cunit.html`.

1.5 Makefile

Votre programme doit être accompagnée d'un Makefile permettant d'effectuer les commandes suivantes:

- `make`: compile l'ensemble des composants (jeu + test)
- `make jeu`: compile le jeu uniquement (doit produire un exécutable nommé `programme`)
- `make test`: compile les tests uniquement (doit produire un exécutable nommé `test`)

1.6 Indices

- Commencez par écrire les tests. Un membre du groupe peut se concentrer sur les tests pendant que l'autre implémente le code.
- Même si la liste chainée `moves` donnée en paramètre à la fonction `apply_moves` doit spécifier dans l'ordre les mouvements à effectuer (i.e. le premier élément de la chaine est le premier mouvement), l'implémentation de la liste chainée `moves` dans `struct game` est laissée libre.
- Après avoir écrit les fonctions `new_game`, `load_game` et `free_game`, il est fortement conseillé de commencer par l'écriture de la fonction `is_move_seq_valid` qui pourra être réutilisée à divers endroits de votre code.

2 Délivrables

Vous devez remettre votre code **commenté** (étant donné qu'il n'y a pas de rapport à fournir, votre code doit être suffisamment clair et commenté) dans une archive

[1] `http://cunit.sourceforge.net/`

.tar.gz sur `http://crp.info.ucl.ac.be/SINF1252/` (vos logins seront distribués au cours des semaines qui viennent). N'oubliez pas d'inclure un **Makefile** ! **Un code qui ne compile pas ou qui n'a pas de Makefile ne sera PAS évalué !**

Votre dossier doit contenir tous les fichiers source nécessaires à la compilation et l'utilisation de votre implémentation du jeu ainsi que les fonctions de test que vous avez défini pour valider votre implémentation. Ces fichiers sont:

- `dames.h`
- `dames.c`
- `programme.c`
- `test.c`
- `Makefile`

La **deadline** pour la remise du projet est le mercredi 11 mars à 18h00.

3 Évaluation

Ce projet sera évalué et comptera dans la note finale. Les critères de cotations (non-exhaustifs) sont les suivants:

- Qualité du code (indentation, respect d'une convention,...);
- Commentaires;
- Tests de la valeur de retour des fonctions;
- Exhaustivité et qualité des tests;
- Gestion de la mémoire (efficacité, memory leak, etc.).