Projet de programmation impérative

Le jeu de dames

Version **provisoire** du 12 mars 2018

Le but du projet est d'écrire un programme C offrant à l'utilisateur plusieurs fonctionnalités liées au jeu de dames : simulation d'une partie, enregistrement et reprise d'une partie, etc. Le jeu de dames est un jeu de société d'antique origine opposant deux joueurs. Une partie se déroule en déplaçant des pièces sur un plateau de 10×10 cases, appelé *damier* ; les joueurs jouent alternativement, en déplaçant une de leurs pièces sur le damier, ce déplacement pouvant entraîner la prise de certaines pièces de l'autre joueur ; on dit qu'un joueur a *le trait* si c'est à son tour de jouer ; le but du jeu est de capturer toutes les pièces de l'adversaire. La variante du jeu sur laquelle vous travaillerez est **le jeu de dames international** (JDDI) : vous devez consulter les règles sur le site de la Fédération Française de Jeu de Dames (FFJD) :

`http://www.ffjd.fr/Web/index.php?page=reglesdujeu`

Vous pouvez aussi consulter la page de Wikipédia consacrée au jeu :

`https://fr.wikipedia.org/wiki/Dames`

mais en cas de contradiction entre les deux sites, c'est celui de la FFJD qui fera autorité.

1 Les bases du programme

1.1 Les pièces des joueurs

Il y a un seul type de pièce : le *pion* (d'où la lettre choisie pour le désigner). Un pion peut être promu en *dame* (d'où la lettre choisie pour désigner un pion promu) s'il atteint une certaine zone du damier.

On affecte par convention la valeur 0 à l'un des deux joueurs et la valeur 1 à l'autre joueur. On associe à chaque pièce du jeu un caractère selon le tableau suivant :

Pièce	Joueur 0 (pièce non promue)	Joueur 0 (pièce promue)	Joueur 1 (pièce non promue)	Joueur 1 (pièce promue)
Pion	p	d	P	D

Une pièce se déplace sur le damier selon son statut (non promu, promu) et selon le joueur auquel elle appartient. Vous devez donc définir une structure *piece* contenant un champ *joueur* et un champ *statut*, puis écrire les fonctions suivantes :

- *piece_creer* qui prend en arguments un joueur et un statut, et renvoie une pièce.
- *piece_joueur* qui prend en argument une pièce et renvoie son joueur.
- *piece_identifier* qui prend en argument un caractère et renvoie la pièce correspondante.
- *piece_caractere* qui prend en argument une pièce et renvoie le caractère qui lui est associé.
- *piece_afficher* qui prend en argument une pièce et affiche le caractère qui lui est associé.

1.2 La liste des coups joués

On souhaite pouvoir stocker la suite des coups, ou *mouvements*, effectués depuis le début de la partie, sous la forme d'une liste doublement chaînée. Un mouvement est un déplacement (valide) d'une pièce sur le tablier, accompagné éventuellement d'une promotion et/ou de la capture d'une ou plusieurs pièces adverses.

Attention ! Le déplacement d'une pièce peut être complexe dans la mesure où la pièce déplacée peut occuper des positions intermédiaires au cours de son déplacement (dans le cas de prise multiple).

Un mouvement contient les informations suivantes : les coordonnées des positions (initiale, intermédiaires et finale) occupée par la pièce déplacée, un booléen permettant de savoir si il y a eu une promotion, et les informations sur les éventuelles captures de pièces adverses.

1.3 La liste des configurations

On souhaite également pouvoir stocker la suite des configurations successives depuis le début de la partie, sous la forme d'une liste doublement chaînée. Cela s'avérera indispensable pour les tests de fin de partie.

1.4 La partie

Une partie de jeu de dames se déroule sur un *damier* de $10 \times 10 = 100$ cases. Vous utiliserez un tableau à deux dimensions de 100 cases. Chaque case du damier peut être *vide* ou contenir une pièce.

Voici, par exemple, le damier au début de la partie :

```
                      0  1  2  3  4  5  6  7  8  9
Joueur 1->         9     P     P     P     P     P
Joueur 1->         8  P     P     P     P     P
Joueur 1->         7     P     P     P     P     P
Joueur 1->         6  P     P     P     P     P
                   5     .     .     .     .     .
                   4  .     .     .     .     .
Joueur 0->         3     p     p     p     p     p
Joueur 0->         2  p     p     p     p     p
Joueur 0->         1     p     p     p     p     p
Joueur 0->         0  p     p     p     p     p
```

Dans la représentation ci-dessus, les cases occupées ou susceptibles d'être occupées par une pièce (les cases "foncées" du damier) sont signalées par le caractère de la pièce (si la case est occupée) ou un point (si la case est inoccupée). Les cases qui ne peuvent pas accueillir de pièce (les cases "claires" du damier) sont signalées par un espace vide.

Vous définirez une structure *partie* contenant :
- un damier,
- la liste des coups joués depuis le début de la partie,
- la liste des configurations successives du damier depuis le début de la partie,
- un booléen permettant de savoir si c'est le joueur 0 ou le joueur 1 qui a le trait.

Vous devrez aussi définir les fonctions suivantes :
- *case_vide* teste si une case du plateau est vide.
- *modifier_case* prend en arguments un pointeur sur une partie, une pièce et les coordonnées d'une case, et modifie le damier en plaçant la pièce passée en argument sur la case passée en argument.
- *changer_joueur* prend en argument un pointeur sur une partie et change le joueur qui a le trait.
- *afficher_plateau* prend en argument un pointeur sur une partie et affiche le damier.
- *modif_damier* prend en arguments un pointeur sur une partie et un déplacement (supposé valide) ; elle modifie le damier (en particulier, elle supprime les pièces capturées) et ajoute le mouvement correspondant dans la liste des coups déjà joués.

- *annuler_mouvement* prend en entrée un pointeur sur une partie et annule le mouvement précédent ; cette fonction modifie la liste des coups déjà joués et replace, si besoin est, les pièces capturées.
- *saisie_case* ne prend pas d'argument et récupère des coordonnées ; les valeurs comprises entre 0 et 9 sont admises ; d'autres valeurs peuvent être ajoutées pour gérer l'annulation des coups, ou le fait de quitter la partie.
- *partie_creer* ne prend pas d'argument et alloue la mémoire nécessaire pour stocker une partie ; elle initialise la liste des coups joués.
- *partie_detruire* prend en argument un pointeur sur une partie et libère l'espace mémoire qu'elle occupe.
- *partie_sauvegarder* prend en argument un pointeur sur une partie et une chaîne de caractères (le nom et le chemin d'un fichier) et sauvegarde la partie dans le fichier. Voir section 1.6.
- *partie_charger* prend en argument une chaîne de caractères (le nom et le chemin d'un fichier *.plt* à charger) et renvoie un pointeur sur une partie. Voir section 1.7.
- *partie_nouvelle* ne prend pas d'argument et renvoie un pointeur sur une nouvelle partie initialisée comme dans l'exemple ci-dessus.
- *partie_jouer* contient la boucle principale du jeu. Tant que les joueurs ne décident pas de quitter la partie, on récupère leurs mouvements, on vérifie qu'ils sont valides. S'ils sont valides, les mouvements sont effectués et on passe au joueur suivant. Tant que les mouvements ne sont pas valides, le même joueur continue à jouer. Un joueur peut annuler le(s) mouvement(s) précédent(s). Lorsqu'un joueur décide de quitter la partie, le jeu propose de la sauvegarder.
- *replay_charger* prend en argument une chaîne de caractères (le nom et le chemin d'un fichier *.part* à charger) et renvoie un pointeur sur une partie. Voir section 1.8.
- *replay_jouer* prend en argument une partie et la rejoue depuis le début. Voir section 1.8.

1.5 Les déplacements

Les mouvements constituent une part importante du projet. Une fonction doit permettre de tester si un déplacement est valide. Vous devrez donc écrire la fonction suivante :

- *deplacement_valide* qui prend en argument une suite de coordonnées de cases (**et ce que vous jugerez utile d'ajouter**), teste le statut de la pièce de la case de départ et vérifie si ce déplacement est valide.

1.6 Enregistrer

Vous devrez donner aux joueurs la possibilité d'enregistrer leur partie en écrivant dans un fichier (dont le nom est donné par un joueur) le contenu du plateau. Afin d'identifier le format du fichier, on le fait commencer par la ligne suivante : "PL" Le fichier contenant le début de partie est donc :

```
PL
 P P P P P
P P P P P
 P P P P P
P P P P P
 . . . . .
. . . . .
 p p p p p
p p p p p
 p p p p p
p p p p p
```

1.7 Charger une partie à partir d'un fichier

Vous avez défini le type *partie* (voir la section 1.4 ci-dessus). Vous devrez offrir la possibilité de remplir automatiquement une partie à partir d'un damier enregistré dans un fichier de type *.plt*. Le format d'un fichier *.plt* est le suivant : un en-tête contenant le mot "PL", puis le contenu du damier en ASCII (voir l'exemple du damier pour l'enregistrement dans la section 1.6).

Vous écrirez donc une fonction *charger_partie* qui prend en argument le chemin du fichier à charger et renvoie un pointeur vers un damier correctement initialisé. Cette fonction renverra *NULL* si le fichier ne respecte pas le format *.plt*,

1.8 Rejouer une partie

Vous devrez aussi offrir la possibilité de rejouer une partie enregistrée dans un fichier de type *.part* contenant les déplacements de pièces effectués. Le format d'un fichier *.part* est le suivant : un en-tête contenant le mot "PR" puis une ligne par coup joué, chaque ligne contenant les coordonnées de la case de départ (de la pièce déplacée) puis celles de sa case d'arrivée.

2 Exécution du programme

Le programme doit pouvoir être exécuté de plusieurs manières différentes. Soit *jddi* le nom du fichier exécutable.

- Exécution par défaut : *jddi*
  Le programme lance une nouvelle partie.
- Exécution à l'aide d'un fichier : *jddi NomFichier*
  Le programme détecte s'il s'agit d'un fichier de type *plt* ou de type *part* en lisant la première ligne. En fonction de la première ligne, le programme donne la possibilité au joueur de jouer sur le plateau chargé (*plt*) ou rejoue la partie (*part*).

A chaque tour, le joueur a la possibilité de :
- déplacer une de ses pièces,
- annuler le coup précédent,
- sortir du programme.

Si le joueur décide de sortir du programme, on lui propose de sauvegarder la partie dans le fichier de son choix (il peut refuser).

Fin de partie et autres contraintes Les contraintes qu'il faut faut prendre en compte, en particulier pour tester la fin de la partie, sont parfois assez subtiles. Vous devrez indiquer très précisément dans votre compte rendu celles que vous aurez traitées. Il va de soi que l'appréciation de votre travail dépendra en partie des règles et contraintes que vous aurez été capables de prendre correctement en compte

3 Compte rendu

Le projet doit être fait en **trinôme**. Un trinôme doit être entièrement composé d'étudiants de L1 informatique (groupes de TD-TP du mardi matin) ou entièrement composé d'étudiants de L1 mathématiques et DL (groupes de TD-TP du mardi après-midi). **Le compte rendu devra être envoyé à vos chargés de TD au plus tard le vendredi 6 avril 2018** (cette date est susceptible d'être avancée).

Le compte rendu de projet se présentera sous la forme d'un fichier *.tar.gz* contenant :

- un makefile permettant de compiler le programme ;
- un fichier *notes.txt* dans lequel vous expliquerez comment compiler, exécuter le programme et les différentes commandes du programme ;
- un répertoire *Plateaux/* contenant vos fichiers *.plt* ;
- un répertoire *Parties/* contenant vos fichiers *.part* ;
- un répertoire *src/* contenant vos fichiers *.c* et *.h* ;
- un rapport de quelques pages dans lequel vous décrirez le projet, indiquerez les difficultés que vous aurez rencontrées et, le cas échéant, les moyens que vous aurez utilisés pour les surmonter ; le rapport ne contiendra pas de code mais pourra renvoyer à certaines fonctions du code.

Tout manquement à cette liste sera sanctionné et aura un impact sur la note finale.

J'attire également votre attention sur quelques points particulièrement importants.

- Tous vos programmes doivent être **commentés et indentés** proprement.
- Tout espace mémoire alloué au cours du programme doit être **libéré** avant la fin de l'exécution.
- La rédaction du rapport doit être soignée, tout particulièrement **la clarté et la précision de l'expression**.
- Le rapport doit contenir impérativement **la répartition du travail dans le trinôme** ; l'absence de cette information, qui devra être aussi précise que possible, pénalisera tous les étudiants du trinôme.