Projet de programmation impérative

Le jeu de Shôgi

(Version très provisoire du 21 mars 2017)

Le but du projet est d'écrire un programme C offrant à l'utilisateur plusieurs fonctionnalités liées au jeu de Shôgi : simulant d'une partie, enregistrement et reprise d'une partie, etc. Le Shôgi est un jeu de société traditionnel opposant deux joueurs. Comme dans le jeu déchecs, il se joue sur un plateau, appelé *tablier*et le but du jeu est de capturer le roi adverse en déplaçant des pièces sur le tablier. Les deux caractéristiques du Shôgi qui le distinguent le plus nettement du jeu d'échecs occidental sont les suivants :

- une pièce peut être *promue* quand elle atteint une certaine zone du tablier et
- un joueur, quand il capture une pièce de l'adversaire, se l'approprie, la place dans sa *réserve* de pièces et peut la réutiliser pour son compte en la replaçant ultérieurement sur le tablier.

Les règles peuvent être consultées sur le site de la Fédération Française de Shôgi :
`http://ffshogi.e-monsite.com/pages/le-shogi/regles-du-jeu.html`
Vous pouvez aussi consulter la page de Wikipédia consacrée au jeu :
`http://fr.wikipedia.org/wiki/Shogi`

1 Les bases du programme

1.1 Les pièces

Il y huit types de pièce différents : pion (*Pawn*, d'où la lettre choisie pour le désigner), lance (*Lance*), cavalier (*kNight*), fou (*Bishop*), tour (*Rook*), général d'argent (*Silver general*), général d'or (*Golden general*) et roi (*King*). 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 1 (pièce non promue)	Joueur 1 (pièce promue)	Joueur 0 (pièce non promue)	Joueur 0 (pièce promue)
Pion	p	d	P	D
Lance	l	j	L	J
Cavalier	n	c	N	C
Fou	b	f	B	F
Tour	r	t	R	T
Général d'argent	s	a	S	A
Général d'or	g	–	G	–
Roi	k	–	K	–

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

- *piece_creer* qui prend en arguments un joueur, un type 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 réserve de chaque joueur

Il est indispensable de stocker, pour chaque joueur, les pièces capturées à son adversaire. Ces pièces constituent sa *réserve*. Vous utiliserez des cases "artificiellement" ajoutées au tablier pour recueillir la réserve de chaque joueur (voir la section 1.4 ci-après).

1.3 La liste des coups joués

On souhaite pouvoir stocker la suite des coups, ou *mouvements*, précédents sous la forme d'une liste doublement chaînée. Un mouvement peut être un parachutage (déplacement d'une pièce depuis la réserve vers une case vide du tablier) ou un déplacement (valide) d'une pièce sur le tablier.

Un coup contient quatre informations : les coordonnées de départ, les coordonnées d'arrivée, un booléen permettant de savoir si il y a eu une promotion, et un booléen permettant de savoir si une pièce a été capturée ou non.

1.4 La partie

Une partie de Shôgi se déroule normalement sur un *tablier* de $9 * 9 = 81$ cases. Vous utiliserez un tableau à deux dimensions de $11 * 11 = 121$ cases ; cet accroissement "artificiel" du tableau vous permettra de représenter les réserves de pièces des deux joueurs. Chaque case du tablier peut être *vide* ou contenir une pièce.

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

```
                      0  1  2  3  4  5  6  7  8  9 10
                  0   .  .  .  .  .  .  .  .  .  .  .
Joueur 0->        1   .  L  N  S  G  K  G  S  N  L  .
Joueur 0->        2   .  .  R  .  .  .  .  .  B  .  .
Joueur 0->        3   .  P  P  P  P  P  P  P  P  P  .
                  4   .  .  .  .  .  .  .  .  .  .  .
                  5   .  .  .  .  .  .  .  .  .  .  .
                  6   .  .  .  .  .  .  .  .  .  .  .
Joueur 1->        7   .  p  p  p  p  p  p  p  p  p  .
Joueur 1->        8   .  .  b  .  .  .  .  .  r  .  .
Joueur 1->        9   .  l  n  s  g  k  g  s  n  l  .
                 10   .  .  .  .  .  .  .  .  .  .  .
```

Dans la représentation ci-dessus, la tablier à proprement parler est constitué des "cases" dont les indices de colonne et de ligne sont compris entre 1 et 9. Les cases de la ligne 0 et de la colonne 0 (à l'exception des deux cases qui ont un de leurs indices égal à 10) sont destinées à accueillir la réserve de pièces du joueur 0 et les cases de la ligne 10 et de la colonne 10 (à l'exception des deux cases qui ont un de leurs indices égal à 0) sont destinées à accueillir la réserve de pièces du joueur 1. Ces réserves, constituées des pièces capturées à l'adversaire, sont vides au début de la partie !

Vous définirez une structure *partie* contenant :
— un tablier (avec la réserve de chaque joueur),
— la liste des coups joués,
— un booléen permettant de savoir si c'est au joueur 0 ou au joueur 1 de jouer.

On devrez aussi définir les fonctions suivante :
— *case_vide* teste si une case du plateau est vide.

— *modifier_case* prend en argument un pointeur sur une partie, une pièce et des coordonnées et modifie le tablier.
— *changer_joueur* prend en argument un pointeur sur une partie et change le joueur.
— *afficher_plateau* prend en argument un pointeur sur une partie et affiche le tableau avec la réserve de chaque joueur.
— *deplacement* prend en arguments un pointeur sur une partie, des coordonnées de départ et d'arrivée. Appeler cette fonction suppose que le déplacement est valide. Modifie le tablier, ajoute le déplacement dans la liste des coups et si une pièce est capturée, elle est ajoutée à la réserve du joueur concerné.
— *annuler_deplacement* prend en entrée un pointeur sur une partie et annule le coup précédent. Modifie la liste des coups précédents et replace la pièce éventuellement capturée.
— *saisie_case* ne prend pas d'argument et récupère des coordonnées. Les valeurs comprises entre 0 et 10 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. Initialise les listes.
— *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 caractère (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 leur déplacement, on vérifie qu'ils sont valides. S'ils sont valides, les déplacements sont effectués et on passe au joueur suivant. Tant que les déplacement ne sont pas valides, le même joueur continue à jouer. Un joueur peut annuler le(s) déplacement(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 caractère (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 déplacements constituent une part importante du projet. Un parachutage sera assimilé à un déplacement (depuis une case dont un des indices est 0 ou 10). Pour chaque type de pièce, une fonction doit permettre de tester si un déplacement est valide. Vous devrez donc écrire les fonctions suivantes :

— *deplacement_valide_pion*, *deplacement_valide_tour*, *deplacement_valide_cavalier*, *deplacement_valide_tour*, *deplacement_valide_roi*, *deplacement_valide_reine*, qui prennent en argument les coordonnées de la case de départ, les coordonnées de la case d'arrivée et un statut, et renvoient 1 si le déplacement est valide, 0 sinon.
— *deplacement_valide* qui prend en arguments les coordonnées des cases de départ et d'arrivée (**et ce que vous jugerez utile d'ajouter**), teste le type et le statut de la pièce de la case de départ et appelle la fonction adéquate.

L'implantation de ces fonctions devra éviter au maximum de dupliquer du code. La définition de fonctions supplémentaires est donc recommandée.

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
..........
.LNSGKGSNL.
..R.....B..
.PPPPPPPPP.
..........
..........
..........
.ppppppppp.
..b.....r..
.lnsgkgsnl.
..........
```

1.7 Charger les parties 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 tablier 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 tablier en ASCII (voir l'exemple du tablier pour l'enregistrement dans la section 1.6).

Vous écrirez donc une fonction *charger_partie* qui prendra en argument le chemin du fichier à charger et renvoie un pointeur vers un tablier 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. Pour cela, on définit un format de fichier contenant les déplacements de pièces effectuées.

2 Exécution du programme

Le programme doit pouvoir être exécuté de plusieurs manières différentes. Imaginons que le nom du fichier exécutable soit *shogi*.

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

3 Partie avancée du projet

Les éléments de la partie avancées seront comptabilisés uniquement si le reste du projet a été programmé.

Déplacements avancés et fin de Partie

- Être capable de détecter si le roi est en échec ou en échec et mat.
- L'empêcher de se déplacer dans une case qui le met en échec.

Programmation avancée

4 Rendu

Le projet sera envoyé 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*
- Vos programmes doivent être commentés et indentés proprement.
- Le nombre de *malloc* et de *free* effectué doit être le même à la fin du programme.
- Un rapport de quelques pages dans lequel vous décrirez le projet, ce que qui vous a posé problème et lorsque c'est le cas, comment vous avez résolu ce problème. Le rapport ne doit pas contenir de code, mais peut faire référence à certaines fonctions dans le code.
- **Le rapport doit contenir impérativement la répartition du travail dans le trinôme. Tout oubli de cette partie pénalisera tous les étudiants du trinôme**.

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