Projet de programmation impérative
le jeu d'échec

5 avril 2016

1 Les bases du programmes

1.1 Les pièces

Une pièce contient une couleur et un type.

— *piece_creer* qui prend en entrée une couleur et un type et renvoie une pièce.
— *piece_couleur* qui prend en entrée une pièce et renvoie sa couleur.
— *piece_identifier* qui prend en entrée un caractère et renvoie la pièce correspondante.
— *piece_caractere* qui prend en entrée une pièce et renvoie le caractère qui lui est associé.
— *piece_afficher* qui prend en entrée une pièce et affiche le caractère qui lui est associé.

1.2 La liste des pièces capturées

On souhaite pouvoir stocker la liste des pièces capturées. Cette liste est simplement chaînée. Fonctionne comme une pile.

1.3 La liste des coups joués

On souhaite pouvoir stocker la liste des coups précédents. Cette liste est doublement chaînée. Un coup contient 3 informations : les coordonnées de départ, les coordonnées d'arrivée et un booléen permettant de savoir si une pièce à été capturée ou non.

1.4 La partie

L'échiquier est un tableau à deux dimensions de $8 * 8$ cases. Chaque case de l'échiquier peut être au choix *vide* ou contenir un pion, une tour, un cavalier, un fou, un roi, une reine.

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

```
                    0  1  2  3  4  5  6  7
NOIR->              0  T  C  F  R  E  F  C  T
NOIR->              1  P  P  P  P  P  P  P  P
                    2  .  .  .  .  .  .  .  .
                    3  .  .  .  .  .  .  .  .
                    4  .  .  .  .  .  .  .  .
                    5  .  .  .  .  .  .  .  .
BLANC->             6  p  p  p  p  p  p  p  p
BLANC->             7  t  c  f  r  e  f  c  t
```

On veut donc définir une structure *partie* contenant :
- un échiquier,
- la liste des pièces capturées,
- la liste des coups joués,
- un booléen permettant de savoir si c'est au joueur noir ou au joueur blanc de jouer.

On définit ensuite la liste de fonctions suivante :
- *case_vide* teste si une case de l'échiquier est vide.
- *modifier_case* prend en entrée un pointeur sur une partie, une pièce et des coordonnées et modifie l'échiquier.
- *changer_joueur* prend en entrée un pointeur sur une partie et change le joueur.
- *afficher_echiquier* prend en entrée un pointeur sur une partie et affiche l'échiquier.
- *deplacement* prend en entrée un pointeur sur une partie, des coordonnées de départ et d'arrivée. Appeler cette fonction suppose que le déplacement est valide. Modifie l'échiquier, ajoute le déplacement dans la liste des coups et si une pièce est capturée, elle est également ajoutée dans la liste concernée.
- *annuler_deplacement* prend en entrée un pointeur sur une partie et annule le coup précédent. Modifie la liste des coups précédents et des pièces pièce capturées.
- *saisie_case* ne prend rien en entrée et récupère des coordonnées. Les valeurs inclues entre 0 et 7 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 rien en entrée et alloue la mémoire nécessaire pour stocker une partie. Initialise les listes.
- *partie_detruire* prend en entrée un pointeur sur une partie et libère l'espace mémoire qu'elle occupe.
- *partie_sauvegarder* prends en entrée un pointeur sur une partie et une chaîne de caractères le nom et le chemin du fichier dans lequel sauvegarder la partie. Voir section 1.6.
- *partie_charger* prend en entrée une chaîne de caractère indiquant où se situe le fichier *.plt* à charger et renvoie un pointeur une partie. Voir section 1.7.
- *partie_nouvelle* ne prend rien en entrée et renvoie un pointeur sur une

nouvelle partie (l'échiquier est comme sur les exemples).
- *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 entrée une chaîne de caractère indiquant où se situe le fichier *.part* à charger et renvoie un pointeur une partie. Voir section 1.8.
- *replay_jouer* prend en entrée une partie et la rejoue depuis le début. Voir section 1.8.

1.5 Les déplacements

Les déplacements constitue une part importante du projet. Pour chaque type de pièce, on souhaite décrire une fonction permettant de tester si un déplacement est valide. On a donc besoin des fonctions suivantes :

- *deplacement_valide_pion*, *deplacement_valide_tour*, *deplacement_valide_cavalier*, *deplacement_valide_tour*, *deplacement_valide_roi*, *deplacement_valide_reine*, qui prennent en entrée les coordonnées de départ et d'arrivée et renvoient 1 si le déplacement est possible, 0 sinon.
- *deplacement_valide* qui prend en entrée les coordonnées de départ et d'arrivée (**et ce que vous voulez**), teste le type de la pièce à la coordonnée 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

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
TCFREFCT
PPPPPPPP
........
........
........
........
pppppppp
tcfrefct
```

La sauvegarde a lieu lorsque les joueurs décident de quitter une partie en cours.

1.7 Charger les parties à partir d'un fichier

Une fois le type *partie* défini, on va vouloir en remplir automatiquement à partir d'un échiquier décrit dans un fichier de type *.plt*.

Format du fichier *.plt* : un en-tête contenant le mot "PL", puis le contenu du plateau en ASCII (voir exemple plateau pour l'enregistrement)

Il faut donc définir une fonction *charger_partie* qui prend en entrée le chemin du fichier à charger et renvoie un pointeur vers un plateau correctement initialisé. Cette fonction renvoie *NULL* si le fichier ne respecte pas le format *.plt*,

1.8 Rejouer une partie

Donnez la possibilité de rejouer une partie. Pour cela, on définit un format de fichier contenant les déplacements de pièces effectuées.

Format du fichier *.part* : Afin d'identifier le format, on commence le fichier par la ligne "PR" chaque ligne contient les coordonnées de départ de la pièce puis les coordonnées d'arrivée.

2 Exécution du programme

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

- Exécution par défaut : *chess*
  Le programme lance une nouvelle partie.
- Exécution à l'aide d'un fichier : *chess 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). Lorsque l'animation montrant une partie est terminée, on sort du programme. Le temps par défaut entre deux plateaux est 1 seconde.
- Exécution paramétrée d'une animation : *chess NomFichier t*
  Le programme vérifie que le fichier est de type *part* et que le paramètre t est un double, Puis exécute une animation en laissant t seconde entre deux plateaux.

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.
- Donner la possibilité de faire un roque (petit et grand).
- Pour les pions, ajoutez la possibilité de faire une prise en passant.

Intelligence Artificielle Écrire un algorithme permettant de résoudre un problème simple de "vie et de mort". Vous pouvez vous inspirez de l'algorithme du minimax, de l'alpha-béta, ou tout ce que vous pourrez trouver.

Interface graphique Créer une interface graphique dans le langage que vous souhaitez. Le jeu se manipulera à la souris, chaque case du plateau étant un bouton.

Programmation avancée Réécrire la structure pièce pour qu'elle contienne un pointeur de fonction, permettant de vérifier si un déplacement est valide. Cet ajout doit permettre de considérablement simplifier la fonction *deplacement_valide*.

4 Rendu

Le projet est à rendre par mail votre chargé de TD **et** votre chargé de TP à la date du Vendredi 08 Avril au soir. Le projet doît être effectué en trinôme.

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.