3ETI, Développement logiciel en C

Présentation du projet:

Le but du projet est de réaliser le fonctionnement d'un jeu d'échec valide.
Plus spécifiquement, il consiste à implémenter l'organisation générale du jeu, et le suivi des règles du mouvement des pièces.

Le projet mettra en avant un ensemble de bonnes pratiques de codage permettant de simplifier la lecture du code, son évolutivité et sa robustesse.
Des aspects généraux de l'informatique seront également mises en pratiques (compilation, utilisation de librairies).

Organisation du code source:

Un certain nombre de fichiers sources vous est fourni. Le projet est lui-même formé de parties comportant deux exécutables.

- Le répertoire **visualiseur**:
  Permet de visualiser l'état d'un échiquier. Il contient un programme indépendant de votre projet qui permet de visualiser le placement des pièces sur l'échiquier.

  Notez que ce programme ne contient aucune *intelligence* au niveau de la connaissance des règles du jeu d'échec, il s'agit uniquement d'un visualiseur d'un état donné.

  Ce code est déjà complet, il ne fait pas partie du cadre du projet à compléter. Il peut être utilisé en tant qu'exécutable simple.
  Pour les étudiants intéressés, le code source vous est entièrement fourni. L'ensemble du code est écrit en C, et il utilise la librairie graphique OpenGL ainsi que le gestionnaire de fenêtre Glut (pour la version 3D).

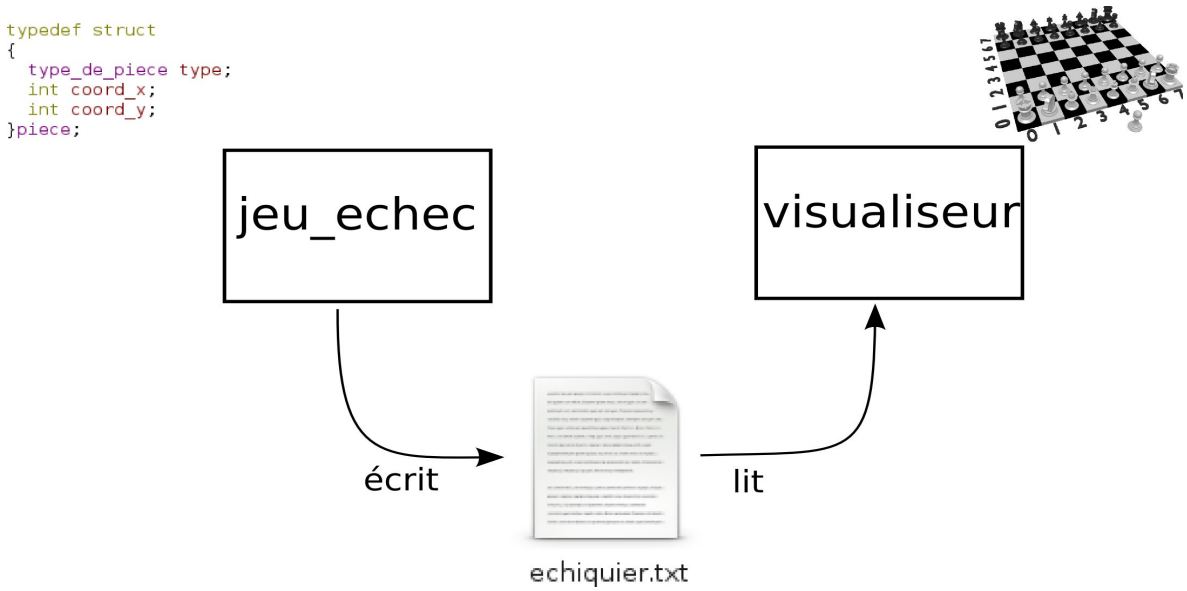
- Le répertoire **jeu_echec**:
  Contient l'ensemble des fichiers permettant de manipuler effectivement le jeu d'échec.

  Il contient le code permettant de
  - stocker l'état d'un jeu d'échec,
  - manipuler le jeu,
  - analyser le suivi des règles du jeu lors de la demande d'une pièce de l'échiquier,
  - communiquer avec l'utilisateur en ligne de commande.

  C'est cette partie du code que vous allez compléter lors de ce projet.

La communication entre ces deux programmes est réalisée à l'aide d'un fichier texte écrit sur le disque dur.

- L'exécutable du jeu écrit à chaque coup dans ce fichier.
- Le visualiseur vient quant à lui lire ce même fichier lorsqu'il est modifié.



Organisation du projet:

Le code source du projet est réparti en différent fichiers.
Chaque fichier représente un niveau **d'abstraction** spécifique.

Au plus haut niveau d'abstraction, l'utilisateur peut définir:

- Le déplacement d'une pièce d'une coordonnée (x,y) vers une autre.
- L'initialisation de l'échiquier à une configuration de départ.
- La sauvegarde, ou le chargement d'un échiquier.

Ces fonctions d'appels de haut niveau qui permettent de manipuler l'échiquier sont situées dans un un fichier spécifique dénommé **API** (Application Programming Interface).

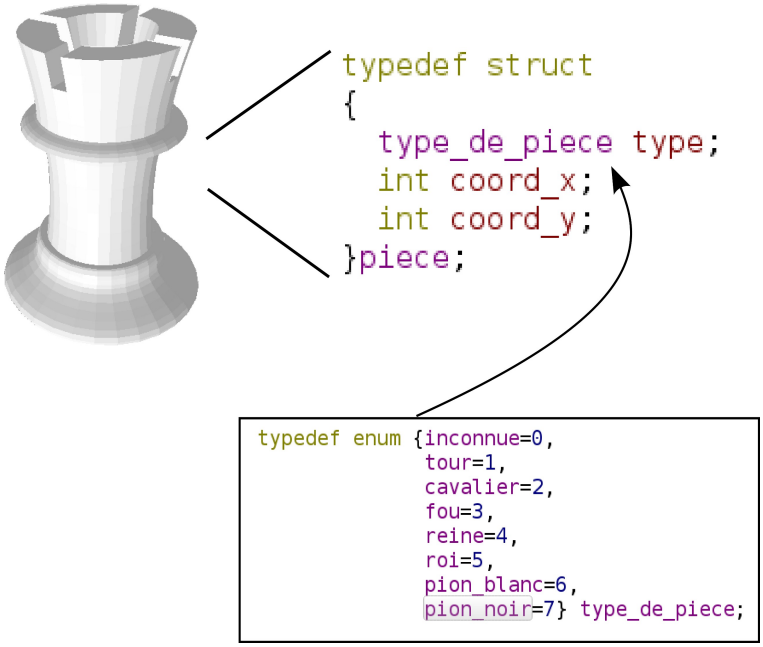
Au niveau d'abstraction plus bas, nous viendrons spécifiquement stocker une pièce en mémoire sous forme de struct, et un jeu sera formé d'un tableau de pièces.

Le détail de ces niveaux d'abstractions est fourni-ci après.

Piece

Le premier niveau d'abstraction est celui représentant une pièce du jeu.
Une pièce est une structure C (struct) contenant:

- Une coordonnée = 2 entiers, l'un pour x, l'autre pour y. Ces coordonnées sont des entiers et doivent être comprises entre 0 et 7.
- Un type définissant de quelle pièce il s'agit. Le type est donné sous forme d'énumération entre (tour, cavalier, fou, reine, roi, pion blanc/noir).



Prenez bien note que la structure unitaire d'une pièce contient ses coordonnées sur l'échiquier.
Une tour aux coordonnées (0,0) et une tour déplacée aux coordonnées (7,0) sont donc deux pièces différentes.

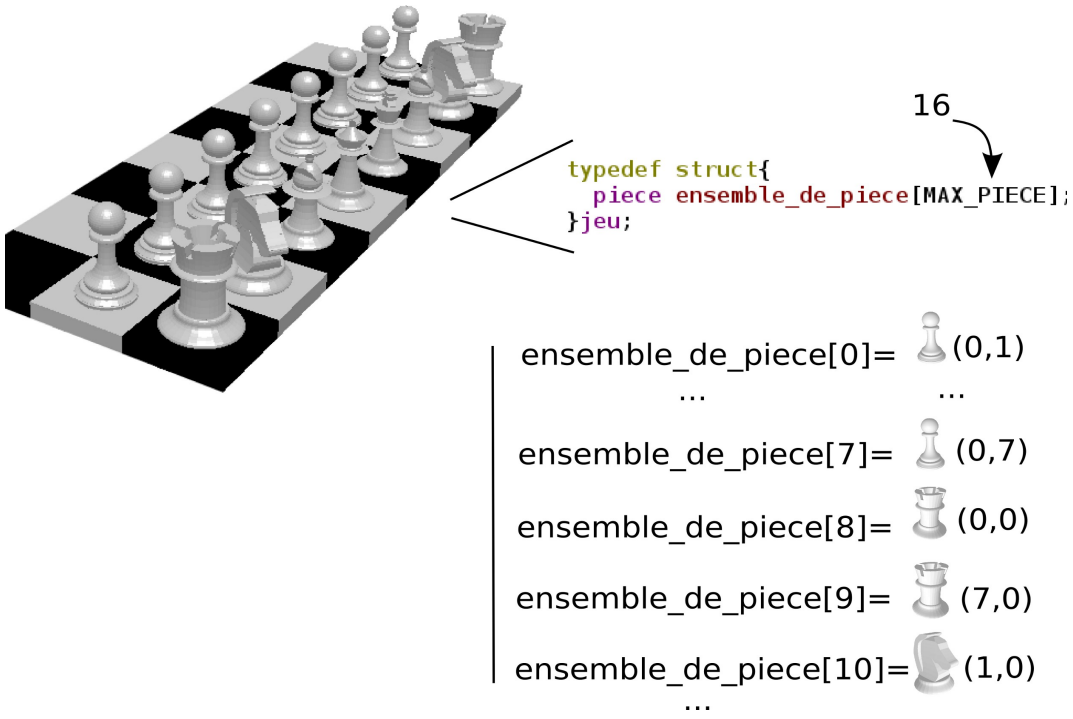
Vous compléterez les fonctions de ce niveau d'abstraction lors de ce projet.

Jeu

Le second niveau d'abstraction est un jeu de pièce.
Un jeu représente l'ensemble des pièces d'un joueur, soit les pièces blanches, soit les pièces noires.
Un jeu est donc un conteneur pour un ensemble de pièces. Dans notre cas, on choisit d'implémenter ce conteneur sous forme de tableau statique.

Un jeu contient au maximum 16 pièces. Le tableau aura donc une dimension d'au moins 16.
Pour simuler les pièces supprimées du jeu, nous placerons des pièces au contenu invalide en fin de tableau.

Il s'agit par contre d'un aspect bas niveau de l'implémentation. Les fonctionnalités de haut niveau doivent pouvoir manipuler un jeu d'échec de manière similaire, même si le stockage est réalisé de manière différente.
On veillera donc à mettre en place des fonctionnalités génériques telles que:
parcourir l'ensemble des pièces valides d'un jeu, trouver ou manipuler une pièce existant à des coordonnées particulières.
De cette manière, seul les fonctions de ce fichier viendront directement manipuler le tableau, et non les fonctions extérieures à ce niveau d'abstraction.

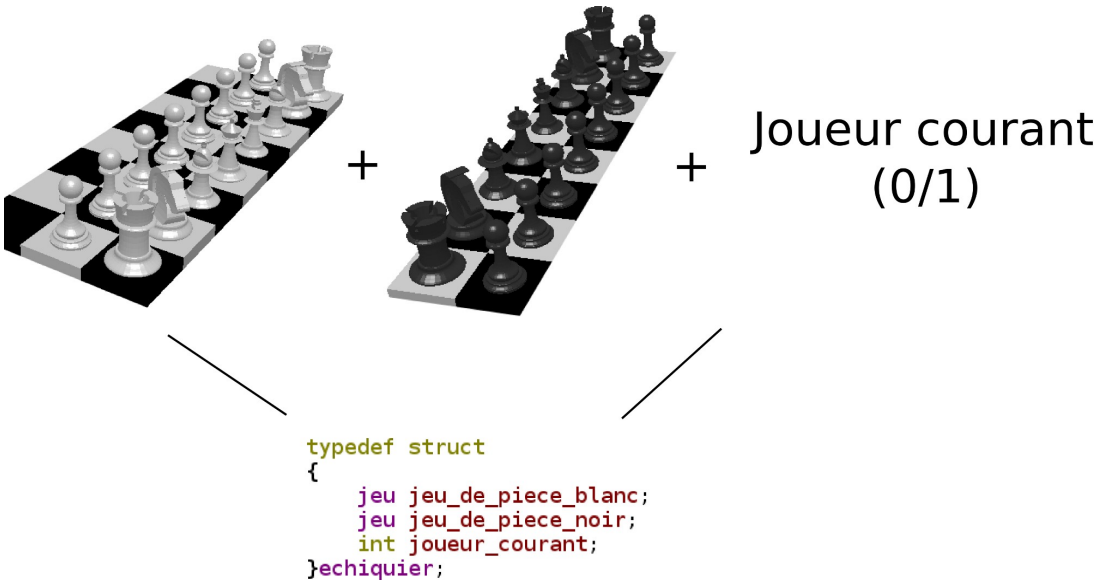


Vous compléterez les fonctions de ce niveau d'abstraction lors de ce projet dans les premières séances.

Echiquier

Un échiquier est formé de deux jeux et du joueur courant. Il s'agit donc d'une structure C qui contient:

- le jeu de pièces blanches
- le jeu de pièces noires
- l'indice du joueur courant (0 pour joueur pièces blanches, 1 pour joueur pièces noires).



Note: Attention, un échiquier n'est **pas** un tableau 2D contenant des pièces ou du vide!
Il s'agirait d'une autre structure de données possible! L'algorithmique de parcours d'un jeu donné serait différent.

Vous compléterez les fonctions de ce niveau d'abstraction lors de ce projet dans les premières séances.

Echiquier déplacement:

Echiquier déplacement est une abstraction qui permet ou non de valider le déplacement d'une pièce de l'échiquier en fonction des règles du jeu d'échec.
C'est en quelque sorte la partie *intelligence* du jeu.
Notez qu'il ne s'agit pas de la modélisation d'une entité physique telle qu'une pièce du jeu.

Contrairement aux structures de stockage précédentes, cette abstraction est supposée être de haut-niveau. Elle n'a pas à interagir directement avec le tableau statique contenant les pièces, et doit pouvoir être indépendante du type de stockage.

Elle doit offrir par contre une interface simple telle que: déplacement de la pièce située aux coordonnées (x0,y0) vers (x1,y1).
Dans le cas où le déplacement est permis par les règles du jeu, celui-ci peut être effectué.
Dans le cas où le déplacement est non permis, une structure d'erreur spécifique est mise à jour permettant au joueur de comprendre en quoi ce coup est invalide.

Cette structure d'erreur contient:

- Le type d'erreur sous forme d'énumération (ex. Absence de déplacement, pièce bloquante, …)
- Les coordonnées qui sont potentiellement à l'origine de l'erreur (tel que les coordonnées d'une pièce bloquant un déplacement).

Vous compléterez la totalité des fonctions de ce niveau d'abstraction lors de ce projet dans les dernières séances.

API echec:

L'API est l'interface de plus haut niveau fournie par une librairie. C'est cette interface qui est utilisée lorsqu'un développeur veut utiliser une librairie donnée afin de l'interfacer avec son propre code.
Dans notre cas, l'API représente le niveau d'abstraction de l'intention (ou la main) de l'utilisateur vis à vis du jeu d'échec.

Dans notre cas, l'API propose les fonctions de:

- déplacement de pièce d'une coordonnée à une autre,
- initialisation du jeu d'échec,
- sauvegarde de l'état du jeu dans un fichier,
- lecture de l'état du jeu à partir d'un fichier.

Toutes les variables donnée en tant qu'argument des fonctions de l'API doivent être génériques (ex. Coordonnées x et y).
Elle ne doivent pas dépendre du type de la structure de stockage par exemple.

Dans notre cas, l'API propose également une fonction qui lance une communication interactive avec l'utilisateur en ligne de commande.

Vous compléterez la gestion globale de déplacement d'une pièce dans l'API lors de ce projet.

Entree sortie :

Entree sortie est un niveau d'abstraction permettant de faire le lien entre l'échiquier existant en RAM et un état d'échiquier stocké sur le disque dur.
Il propose une interface de lecture et d'écriture dans des fichiers du disque.

Il s'agit principalement de concentrer la partie spécifique à l'écriture sur le disque dans une abstraction spécifique afin d'offrir une interface de sauvegarde et de chargement simple pour les autres partie du programme.

De même, il s'agit d'une structure qui ne doit pas dépendre du type de stockage du jeu d'échec en interne. On considère que les appels de cette abstraction sont de haut-niveau.

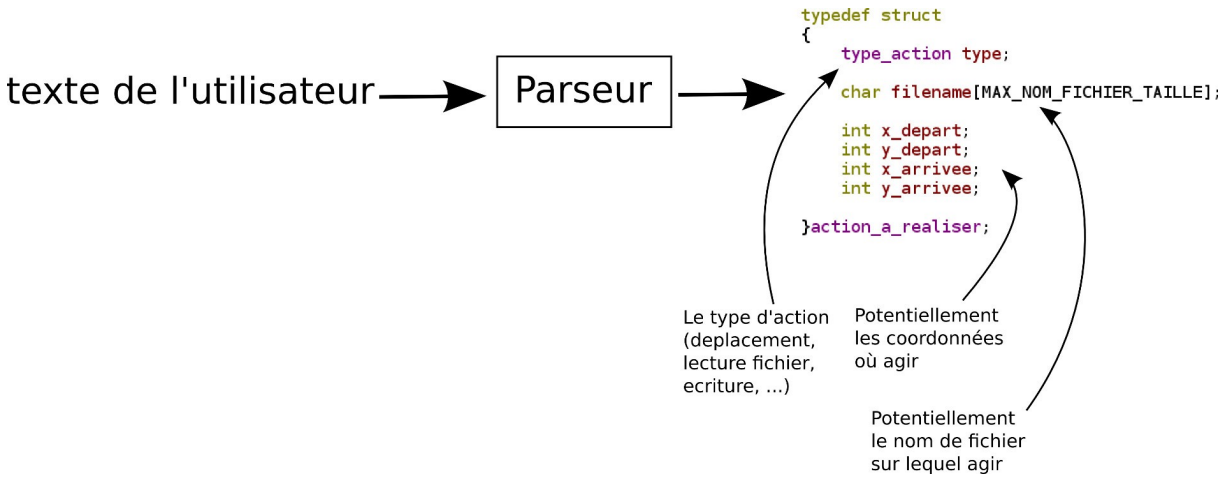
Vous écrirez le code de sauvegarde de l'état de l'échiquier ainsi que de lecture et d'écriture de l'historique dans un fichier lors de ce projet.

Parseur mode interactif:

Tout comme entree_sortie, ce niveau d'abstraction vient principalement décharger l'API de la problématique de la validation et de l'interprétation d'une chaîne de caractère donnée par l'utilisateur lors du dialogue en ligne de commande.

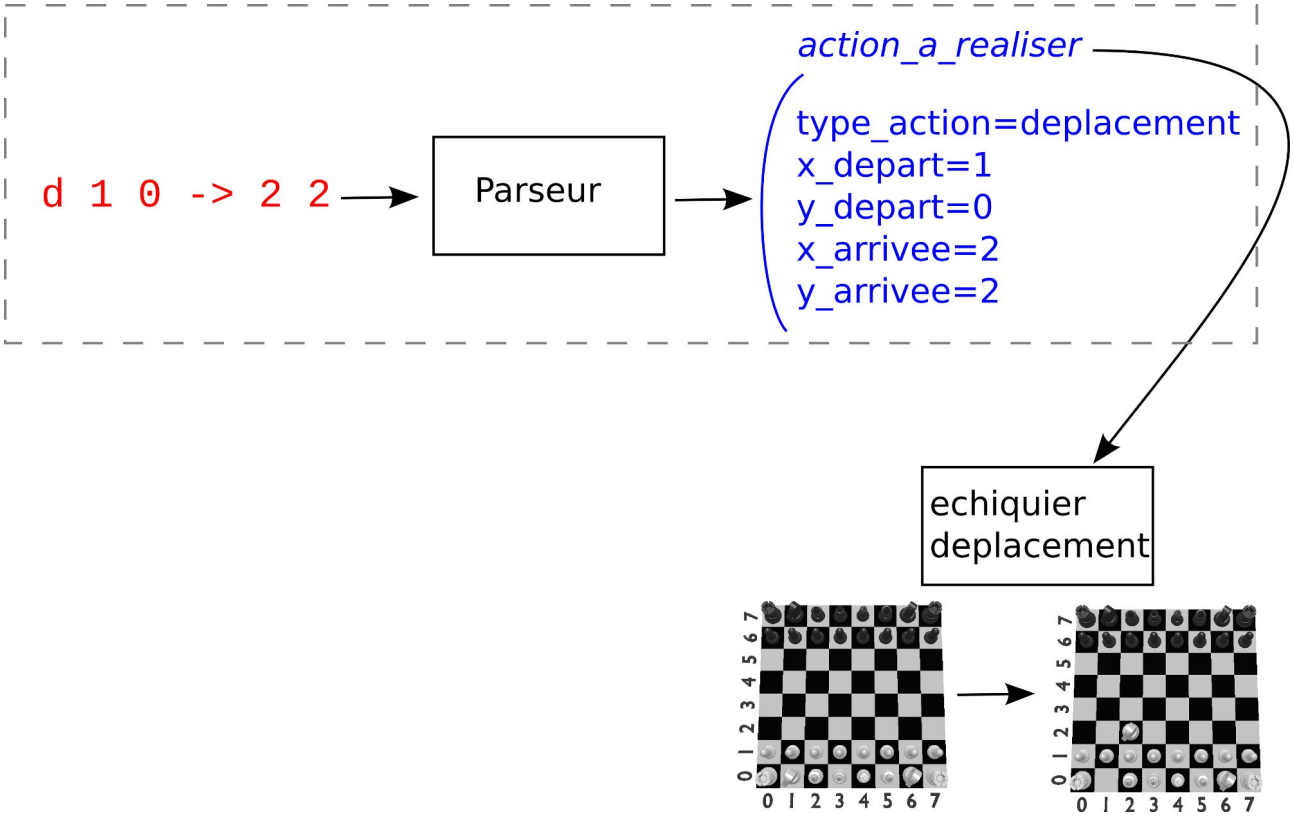
Le parseur vient tout d'abord analyser si la chaîne de caractère est valide au niveau de sa syntaxe.
Ensuite, il vient interpréter ce message.
Enfin, il renvoie une structure contenant les informations de ce message.

Note: Parseur est un terme commun d'informatique désignant l'analyse syntaxique d'un texte.
(ex. Parseur xml, parseur de ligne de commande, etc).



Exemple:

Entrée utilisateur =
`> d 1 0 -> 2 2`



Le parseur est déjà entièrement écrit lors de ce projet.

Diagramme des niveaux d'abstractions:

Le diagramme suivant résume l'organisation (simplifiée) des différents niveaux d'abstractions utiles au début du projet ainsi que leurs relations.