Projet IN104 (2025) - Programmation d'une IA pour le jeu d'échecs en C

Encadrant : Tom Boumba

Fiche 8 : Interface UCI

On dispose à présent d'un moteur d'échecs permettant d'estimer le meilleur coup dans une position donnée. Nous allons maintenant implémenter le protocole de communication UCI (Universal Communication Interface), afin d'affronter notre moteur et de le mesurer à d'autres IA au moyen d'un programme disposant d'une interface graphique comme Arena Chess GUI.

Une description complète du protocole est disponible ici. Dans sa forme la plus simple, un moteur fonctionnant avec le protocole UCI doit être capable de lire et d'exécuter 6 commandes reçues sur son flux d'entrée :

- La commande `uci`, à laquelle il doit répondre sur son flux de sortie :

  ```
  id name <nom_moteur>
  id author <nom_auteurs>
  uciok
  ```

  afin d'affirmer au programme qui l'a appelé qu'il fonctionne bien avec le protocole UCI.

- La commande `isready`, à laquelle le moteur doit répondre `readyok` sur son flux de sortie afin de signifier qu'il est prêt à recevoir d'autres commandes.

- La commande `ucinewgame`, qui doit faire initialiser au moteur son échiquier à la position de départ.

- La commande `position startpos moves <liste_de_coups>`, ou `position fen <pos_fen> moves <liste_de_coups>`, qui doit faire initialiser au moteur l'état actuel de son échiquier à partir de ses arguments. L'échiquier est initialisé à l'état donné par la chaîne fen fournie en entrée, ou à la position initiale, puis les coups de la liste fournie en argument sont exécutés séquentiellement.

- La commande `go depth <profondeur>` ou `go wtime <temps_restant_blancs> btime <temps_restant_noirs>`, qui doit faire estimer au moteur le meilleur coup possible à jouer dans la position actuelle, avec une contrainte de profondeur maximale de parcours de l'arbre du jeu, ou en prenant en compte le

temps de calcul total restant pour chacun des joueurs (une implémentation classique consiste par exemple à faire répartir uniformément au moteur le temps dont il dispose sur un nombre de coups fixé à l'avance). Une fois son calcul terminé, le moteur doit écrire sur son flux de sortie le coup qu'il choisit de jouer.

- La commande `stop`, qui doit faire arrêter son calcul en cours au moteur et le faire renvoyer le meilleur coup qu'il a trouvé jusqu'ici.
- La commande `quit`, qui a le même effet que la commande `stop` si un calcul est en cours, mais qui termine aussi l'exécution du moteur.

Implémentation

Dans le fichier `uci.c` :

- Implémenter la fonction `void parse_position(char *instruction)`, qui exécute une commande `position` fournie en entrée par la chaîne de caractères `instruction`. Pour lire la séquence de coups fournie en entrée après le mot clé `moves`, vous pourrez utiliser la fonction `int get_coup_str(char *coup_gui)`, qui renvoie le coup légal associé à la chaîne de caractères fournie en entrée s'il existe et 0 sinon. Indications : Pour reconnaître une sous-chaîne au début d'une chaîne de caractères donnée, vous pouvez utiliser la condition `if (!strncmp(chaine, "<sous_chaine>", <longueur_de_la_sous_chaine>))`. Pour déplacer le pointeur du début de la chaîne de caractères au début de la première occurence d'une sous-chaîne donnée, vous pouvez utiliser l'instruction `chaine = strstr(chaine, "<sous_chaine>");`. Le pointeur `chaine` prendra la valeur `NULL` si la sous-chaîne est introuvable dans la chaîne.
- La fonction `void parse_go(char *instruction)` est déjà implémentée et permet d'exécuter une commande `go`. Cependant, pour que la recherche du meilleur coup puisse s'interrompre dans le cas où une commande `stop` ou `quit` a été reçue, ou dans le cas où le temps de recherche est épuisé, il faut que la variable globale `stopped` soit mise à 1. La fonction `int communicate()` vous permettra de mettre à jour régulièrement la valeur de la variable `stopped`. Concrètement, dans le fichier `negamax.h`, définissez la constante `UCI` à 1, puis dans le fichier `negamax.c`, incrémentez la variable `nb_noeuds_parcourus` à chaque nouvel appel de la fonction `negamax`. Si le flag `UCI` est à 1, vous pourrez alors mettre à jour la variable `stopped` à l'aide de la fonction `communicate`, par exemple tous les 2048 noeuds parcourus.
- Implémenter la boucle principale de l'interface UCI, i.e la fonction `void main_uci()`. Cette fonction vérifiera le flux d'entrée du moteur en permanence afin de vérifier l'arrivée des commandes UCI et de les exécuter. Le contenu du flux d'entrée pourra par exemple être récupéré dans une variable locale `char input[2000]` à l'aide de la fonction `fgets`. Un retour chariot seul (`\n`) en entrée sera ignoré.

Vous utiliserez les constantes `nom_moteur` et `nom_auteur` dont vous éditerez la valeur dans le fichier `uci.h` pour l'exécution de la commande `uci`.

– Dans un premier temps, vous pourrez tester votre interface uci en ligne de commande. Ensuite, vous pourrez installer et tester votre moteur d'échecs au sein d'un programme disposant d'une interface graphique comme Arena Chess GUI :

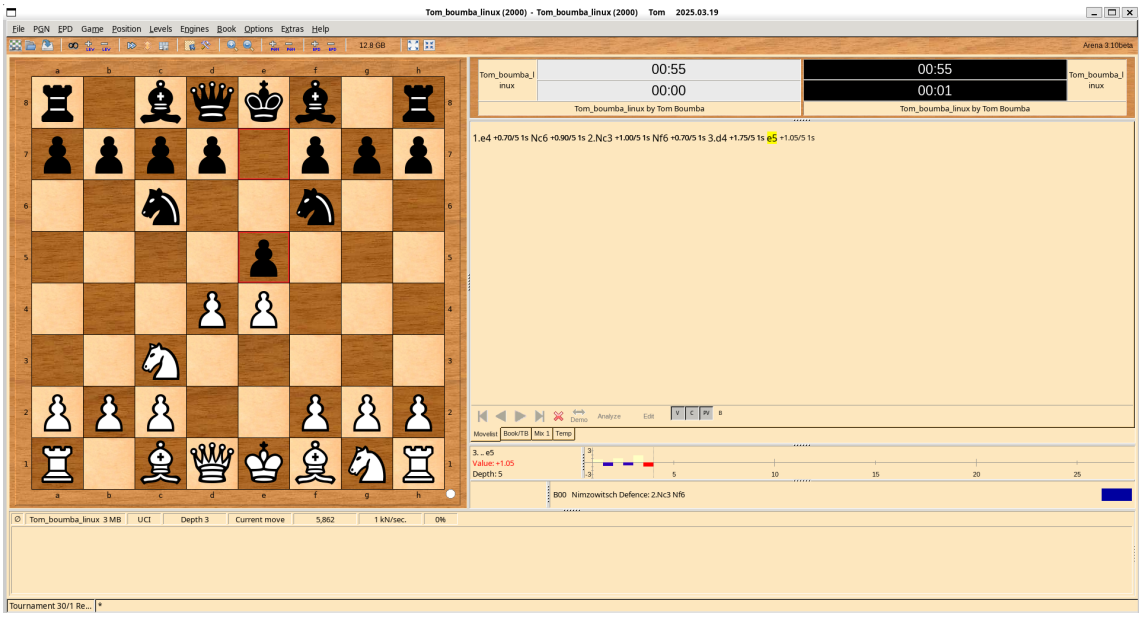

Figure 1: Test d'exécution du moteur d'échecs au sein du programme Arena Chess GUI. Pour installer votre moteur d'échecs, cliquez simplement sur : `Engines > Install New Engine` et sélectionnez l'exécutable binaire de votre moteur d'échecs correspondant au système sur laquelle s'exécute Arena Chess GUI. Pour le débogage, vous pourrez utiliser la fenêtre `Engines > Log - Window` (F4).