BlitzBolt - a simple Chess game (written only in C!)

Introduction

BlitzBolt is a command-line-based chess game written in C language (only!), designed for players who want to enjoy a quick game of chess on their computer. It is a complex yet fun game that you can play against an AI or with another player.

The game features a standard 8x8 chess board, with pieces that move according to the standard chess rules. The environment where all the action is happening (sadly enough) is the console interface where you are currently running the game. The game also features a basic AI opponent, which provides a (somewhat) good challenge for beginners and casual players.

Usage

To create the program, use the command 'make' (or any equivalent for this Ubuntu command). Upon opening the executable, you can choose if you want to play versus another player (option 1) or play against our *(Stockfish's baby)* AI.

After that, an implementation of a chess table will be shown to `STDOUT`. The player is able to choose what color to play and is even able to save or load an unfinished game.

Commands

All commands are given from the command line, each of them being on a single, continuous line. The possible commands are:

- `START_GAME [AI]` -- This command initiates a new PvP game with the default chess positions. It starts with white's turn and alternates with each move. The optional argument AI will instead initiate a new PvE game, giving the player a prompt to choose which color to play as (WHITE / BLACK). If a game is already in progress, the player will be given the choice to either save the current game or discard it, the default option being to discard it (y / N).

- `SAVE <path_to_save_file>` -- This command saves the current game to a file given by its path (will create a new file if the given file doesn't exist).
- `LOAD <path_to_save_file>` -- This command loads the game saved in the file given by the path.
- `EXIT` -- This command exits the game and closes the application. If a game is in progress, the player will be prompted with the choice to either save the game or not. Otherwise, the application simply closes.
- `MOVE <move>` -- This command will move a given piece to a given position. The format for the move is as follows: [start_position]-[end_position].

  Example:

  `MOVE c4-d2`

  The program will first check if the move is valid and that it won't result in a check for the current player and then executes the move.
- `PASS` -- This command will skip the current player's turn. Unorthodox, but useful when you want to create specific scenarios.
- `RESIGN` -- This command will result in the game ending by forfeit of the current player. The game will be discarded.
- `HINT` -- This command will give the player a "good" suggestion for a move he could do.

Implementation

All pieces are dynamically allocated in memory and stored as a structure:

```c
typedef struct piece {
  char color;  // piece's color
  char type;   // piece's type

  // Pointer to a function that checks if the piece can move to the given
  // position and returns TRUE if it can. Otherwise, returns FALSE
  bool (*move)(piece ***, uint8_t[2], uint8_t[2]);
} piece;
```

Chess AI Implementation

Chess is a complex game with many possible moves and positions. When searching for the best move in a given position, it's important to use an efficient search algorithm that can explore the game tree as quickly as possible.

For the moment, we are unsure of what algorithm(s) and implementation(s) are going to be used in the final release. Stay tuned for updates! :)

Team Composition and Tasks

This C-programmed chess game will be made by two students, each having multiple tasks regarding this project. In short, here is a (changeable overtime) TODO list for them:

- Munteanu Eugen:
  - Responsible for `SAVE/LOAD` commands;
  - Documentation, preparation for the presentation of the project;
  - (Also) responsible for `EXIT/RESIGN/HELP` commands;
- Lazar Cristian-Stefan:
  - Responsible for Chess AI Implementation (choice of algorithm, efficiency, performance etc.);
  - Check(mate) detection on the 8x8 board;
  - Responsible for HINT command;
- Both:
  - Founding of the project base (way of storing the chess board, structures created, method of getting commands from `STDIN`);
  - Final checks for bugs and coding style issues, also debugging;
  - Contributing to *README.md*;
  - *(having at least a small understanding of what the other one did)*

Contact

The project will be often updated on the following **GitHub link:** https://github.com/EugenM03/simple--ai--chess

It is a private repository for the moment. If you want access to its content, don't hesitate to contact us on Teams!

Final Words

Whether you're a seasoned chess player or a beginner looking to learn chess (*current population for Chess: everyone*), BlitzBolt is the perfect way to get your fix. So why not give it a try today?