Kovács Dávid Ákos: SAKK - Félkész dokumentáció

A programom 5+1 részre van tagolva:

- Különböző typedefek, színkódok: `piece.h`
- FEN átalakító felhasználhatóvá: `fentocb.c`
- Legális lépések megkeresésének első fázisa: `pseudolegal.c`
- Sakktábla kiírása: `boardprt.c`
- Menü: `menu.c`
- Illetve a `main.c`, ahol jelenleg a legális lépések megkeresésének második fázisa, és maga a játékfolyamat zajlik.

`piece.h`:

- Itt találhatóak a használt színodok, amikkel például kiszínezem a sakktábla mezőit. ANSI escape kódokat használtam. Továbbá itt találhatóak a főbb adatszerkezetek:
    - A `piece` struktúra tartalmazza egy mezőn álló bábú színét, típusát, illetve hogy éppen legálisan lehetne-e vele lépni, vagy sem. (Ez alapállapotban mindig "illegális". Csak akkor változhat, ha lépünk, utánna újból alapállapotba áll vissza.):

```
typedef enum {white, black, null} piece_color;

typedef enum {empty, pawn, knight, bishop, rook, queen, king}
piece_type;

typedef enum {illegal, legal} is_legal;

typedef struct {
piece_color color;
piece_type type;
is_legal legality;
} piece;
```

    - További struktúrák:
        - `fen_data`: mely a `fentocb.h`-nál hasznos, ebbe a strukúrába tárolódik az összes adat, amit egy FEN kódból kinyerhetünk.
        - `moving`: Itt tároljuk a felhasználótól bekért lépést: honnan-hova szeretne lépni.
        - `temp_moving`: Az előzőhöz hasonló, azonban egy olyan lépés amit a program tesz meg ideiglenesen, annak érdekében, hogy kiderítse, legális lenne-e.

```
typedef struct fen {
bool color;
bool w_k_c;
bool w_q_c;
bool b_k_c;
bool b_q_c;
int first_enp_row;
int first_enp_col;
} fen_data;

typedef struct moving {
int from_row;
int from_col;
int to_row;
int to_col;
} moving;

typedef struct temp_moving {
int temp_from_row;
int temp_from_col;
int temp_to_row;
int temp_to_col;
} temp_moving;
```

`fentocb.c`:

- Ez a program lefordít egy kapott FEN kódot, majd betölti a `fen_data` struktúrába azt.
    - FEN kódra egy példa (alapállás):

      ```
      "rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1"
      ```

    - A `void fen_to_chessboard(char fen[100],piece board[8][8],fen_data *data)` először kinullázza a `fen_data`-t, majd a kapott FEN kódon végigolvas: Ha talál "/" jelet, az addig olvasott string-et tekinti egy tokennek. Ha betűt talál tokenen belül elemenként haladva, tudja hogy ott egy bábu áll, és betölti a `board` megfelelő sor- és oszlopába azt. Azonban ha számot talál, tudja hogy ki kell hagynia annyi számnyi mezőt, amekkora számot kapott.

      Ha talál `" "`-t, akkor tudja hogy beolvasott minden bábut, most jönnek az extra adatok, amiket a `fen_data`-ba olvas be: melyik fél van lépésen `"w"`, illetve ki- és merre tud rosálni. `"KQkq"`. ebből 4 féle lehetőség lehet igaz, vagy hamis. felenként kettő: tud-e sáncolni balra, vagy jobbra?

      Ezen felül ha van éppen en passant-ra lehetőség, azt jelzi a FEN kód egy mezővel, ahová érkezhetne az ellenfél gyalogja, ha leütnék azt en passant. Jelen esetben nincs ilyen mező: `"-"`.

      A további számokat nem veszem figyelembe.

`pseudolegal.c`:

- Ez a kód keresi meg az összes legális lépését egy adott bábunak. Lényegében meghatározza, hogy hova léphetne egy bábu. Minden bábunak kell külön szabály meghatározás. Erre egy példa:

```
void psl_bishop(piece board[8][8], int from_row, int from_col) {
    psl_diagonal(board, from_row, from_col);
}
```

Mivel a futó átlósan tud csak haladni így a függvény hívja az átlósan haladás szabályát:

```
void psl_diagonal(piece board[8][8], int from_row, int from_col) {
    int directions[4][2] = {{1, 1}, {1, -1}, {-1, 1}, {-1, -1}};

    for (int i = 0; i < 4; i++) {
        int row = from_row + directions[i][0];
        int col = from_col + directions[i][1];

        while (row >= 0 && row < 8 && col >= 0 && col < 8) {
            if (board[row][col].type == empty) {
                board[row][col].legality = legal;
            } else if (board[from_row][from_col].color != board[row]
[col].color) {
                // ha a celmezoben egy ellenseges babu all, azt leutheted
                board[row][col].legality = legal;
                break; // leutottel egy ellenseges babut, ne keressen tovabb
szabad mezoket
            } else {
                break; // nem lephetsz ra a sajat babudra
            }

            row += directions[i][0];
            col += directions[i][1];
        }
    }
}
```

Ha például a vezérrel szeretnénk lépni, neki hívni kell az átlós, és a horizontális szabályt.

Kivételt képez a király, és a gyalog. Rájuk külön szabályok vonatkoznak a specifikációban leírtak szerint.

`boardprt.c`:

- Kirajzolja a sakktáblát a terminálba. Elősször cleareli a terminált, hogy úgy tűnjön, mintha az előző tábla frissülne. Itt jön képbe hogy milyen platformon játszunk:

```
void clear_terminal() {
  #ifdef _WIN32
  system("cls");  // For Windows
  #else
  system("clear");  // For Linux/macOS
  #endif
}
```

A nyomtatás lényegi része:

```
for (int row = 0; row < 8; row++) {
      for (int col = 0; col < 8; col++) {
          chessboard[row][col].legality = illegal;
          if ((row + col) % 2 == 0) {
              printf("%s",ANSI_WHITE);
          } else {
              printf("%s",ANSI_BLACK);
          }

          print_piece(chessboard[row][col].color, chessboard[row]
[col].type);

          if (col == 7) {
              printf("%s%d ",ANSI_RESET,notation);
              notation--;
          }

      }
      printf("\n");
  }
```

Itt a `print_piece()` tartalmazza az UTF-8 kódolású sakk karaktereket, és a megfelelőt nyomtatja ki.

`menu.c`:

- Innen indul a program, köszönti a felhasználót, és választási lehetőséget nyújt a specifikációnak megfelelően. Ha a játékos az 1. opciót választja, elindul a `main.c`-ben játék az alapállásból: `game()`. Ha azonban a második opciót választja, bekér a felhasználótól egy FEN kódot, majd `fentocb.h` szerint, és így indul a `game()`. Harmadik opció, azaz játék betöltése még nincs kész - így még nincs fájlkezelés a programban. 4. a kilépés, itt leáll a program.

```
int menu1() {
  printf(ANSI_YELLOW"Kovács Dávid Ákos: SAKK\n\n");
  printf(" "ANSI_WHITE"1 "ANSI_RESET". Játék indítása\n");
  printf(" "ANSI_BLACK"2 "ANSI_RESET". Játék indítása speciális
állástól\n");
  printf(" "ANSI_WHITE"3 "ANSI_RESET". Játszma betöltése\n");
  printf(" "ANSI_BLACK"4 "ANSI_RESET". Kilépés\n\n");
  printf(ANSI_RESET"Válassz " ANSI_WHITE"1 "ANSI_RESET" és "ANSI_BLACK"4
"ANSI_RESET" között (pl.: \"2\"): \n\n");
  int option, c;
  bool valid = false;
  do {
      if (scanf("%d", &option) == 1) {
          if (option > 0 && option < 5) {
              valid = true;
          } else {
              printf(ANSI_RED"HIBA: Hibás formátum.\n"ANSI_RESET);
          }
      } else {
          while (c = getchar(), c != '\n' && c != EOF);
          printf(ANSI_RED"HIBA: Hibás formátum.\n"ANSI_RESET);
      }
  } while (!valid);
  return option;
}
```

- A `game()`:
    - Ez köti össze a fő függvényeket. Beolvassa hogy hova szeretne lépni a felhasználó, (gondosan ellenőrizve a formátum helyességét) kideríti hogy legális-e a lépés, ha nem, újra bekér a felhasználótól egy lépést. Továbbá sötét-és világos játékos között váltakozik. Ez a függvény gondoskodik a sakk-ról is, illetve egyéb felfedett sakkok miatti illegálisnak talált lépésekről is. Ez a függvény nagyon nagy és összetett, így nem végleges.