Kovács Dávid Ákos: SAKK - dokumentáció

A programom 8+1 részre van tagolva:

- Különböző typedefek, színkódok: `piece.h`
- FEN átalakító felhasználhatóvá: `fen_code_to_chessboard_converter.c`
- Tábla - FEN átalakító: `chessboardtofen.c`
- Láncolt listához tartozó fájl: `fen_linked_list.c`
- Menti, vagy betölti az állást: `save_load.h`
- 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;
```

Továbbá a láncolt lista struktúrája is itt található, amik a lépések mentésére szolgálnak.

```
typedef struct fen_node {
char data[100];
struct fen_node* next; //lehet benne előre
struct fen_node* prev; //és hátrafele is lépkedni
} fen_node;
```

`fen_code_to_chessboard_converter.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 `fen_to_chessboard()` 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.

`chessboardtofen.c`:

- Hasonló az előzőhöz, csak éppen visszafele, ez kell a tábla megjegyzéséhez. Később ezt adja hozzá a láncolt listához, a játékmenet közben. Egy teljes állást ment el, különböző komplikációk elkerülése végett a szimpla lépésmentéssel szemben.

`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, huszár 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.

`fen_linked_list.c`:

- Itt vannak a láncolt listához szügséges függvények, mint a hozzáadás, nyomtatás, memória felszabadítás, végére járás, előző elemhez jutás, következő elemhez jutás.

- Hozzáadásra kód:

```
void fenlist_append(fen_node** head_reference,const char *new_board) {
    fen_node* newfen_node = (fen_node*)malloc(sizeof(fen_node));
    if (newfen_node == NULL) {
        printf("Nem sikerült memóriát foglalni.\n");
        return;
    }
    strcpy(newfen_node->data,new_board);
    newfen_node->next = NULL;

    if (*head_reference == NULL) {
        newfen_node->prev = NULL;
        *head_reference = newfen_node;
        return;
    }
    fen_node* temp = *head_reference;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newfen_node;
    newfen_node->prev = temp;
}
```

`save_load.c`:

- Itt történik a nézet módba váltás, a mentés, vagy a betöltés.
- Mentés:

```
void savings(fen_node *head) { //mentjük a játszmát
    FILE *file_pointer;
    file_pointer = fopen("saves.txt", "w");

    if (file_pointer != NULL) {
        while (head != NULL) {
            fprintf(file_pointer, "%s ", head->data);
            head = head->next;
        }
        fprintf(file_pointer, "\n");
        fclose(file_pointer);
    } else {
        printf("Hiba történt a fájl megnyitása során.\n");
    }
}
```

- Betöltés:

```
void file_load(fen_node **fen_list) { //betöltjük a mentést ha már volt
előző
    FILE *file_pointer;
    file_pointer = fopen("saves.txt", "r");
    if (file_pointer != NULL) {
        char string[100];
        while (fscanf(file_pointer, "%s", string) == 1) {
            fenlist_append(&*fen_list,string);
        }
        fenlist_print(*fen_list);
        fclose(file_pointer);
    } else {
        printf("Hiba történt a fájl megnyitása során.\n");
    }
}
```

- Nézet mód részlete:

```
if (action == 1) {
    if (count < size) {
        travel_node = move_to_next(travel_node);
        count++;
    }
}
if (action == 2) {
    if (count != 0) {
        travel_node = move_to_previous(travel_node);
        count--;
    }
}
if (action == 3) {
    nezet_mode = false;
    clear_terminal();
}
```

`menu.c`:

- `menu1()`:
    - Ide jut a program bármikor, mikor úgy döntünk hogy belépünk a menübe. Először ez indításnál történik meg. 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 a játék betöltése, ennél az opciónál

egyből bekerül a felhasználó egy nézet (`nezet()`) módba, ahol megtekintheti az előzőleg elmentett játszmát. Ha ilyen még nincs, marad a menüben. 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;
}
```

- `nezet_menu()`:
  - Ide kerül a játékos, ha épp megtekinteni akar egy játszmát. Ez lehet a saját éppen folyó játszmája, vagy a már elmentett, betöltött játszma.

```
printf("\nMerre szeretnél lépni?\n1 Előre\n2 Hátra\n3 Kilépés\n");
int option, c;
bool valid = false;
do {
    if (scanf("%d", &option) == 1) {
        if (option > 0 && option < 4) {
            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);
switch (option) {
    case 1:
        return 1;
    case 2:
        return 2;
    case 3:
        return 3;
    default: return -1;
}
```

A visszatérések után a `save_load.c`-ben a `nezet()` gondoskodik a lépésekről, vagy épp a kilépésről.

- `promotion_menu()`:
  - Ha a játékos promótál egy gyalogról egy tisztre, itt tudja kiválasztani melyikre szeretne.
- `ending_menu()`:
  - Végetért a játék. tt lehet kiválasztani, hogy mentsen-e a játék (`save_load.c`), kilépjen, vagy folytassa a játékot előrről.

`main.c`:

- A `start()`:
    - Itt kezdődik a játék. Addig indul újra, amíg a játékos ki nem lép abból. Itt inicializálódik a láncolt lista a játék mentéséhez, innen irányít át a program a fentebb említett `menu.c`-be.
- A `game()`:
    - 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 a `moving()` függvény segítségével. Figyeli az 50 lépés szabályt, minden lépés után hozzáadja a láncolt listához az állást a `fen_linked_list.c`-nél lerítak alapján. Ha végetér a játék, innen egy játék végetért menübe vezet, a `menu.c`-ben leírtakhoz.