Sakk programozói dokumentáció (C3FBCD)

Megvalósítás

Az első lépésben a program a futási módot kéri be a felhasználótól, amely bemenetet felhasználva egy switch segítségével kiválasztódik a megfelelő kódrészlet.

Érvénytelen bemenet esetén a goto paranccsal a kód elejére ugrik.

Érvényes esetből három lehetőség van (1,2,3), ezek esetén vagy új játék indul, előző játék betöltődik visszanézésre, vagy előző játék végállapota töltődik be, amit folytatni lehet.

Mindegyik egy fájlnév bekérésével kezdődik, amit vagy beolvasáshoz, vagy kiíráshoz használ a program, esetleg mindkettőhöz.

A legfontosabb része a programnak a lépések kezelése, amit a figurák típusától függően szétválasztva kezelek. Ebből hat féle van, amelyek minegyikét külön függvény kezeli. Ezeket a függvényeket szintén egy switch fogja össze és a bejövő figura típusa szerint hívja meg a megfelelő függvényt.

A nehéz része a játéknak a sok esetkezelés, ami rendkívül bonyolulttá teszi a programot egy csomó elágazással.

Egy ilyen a sakk figyelése, amit úgy oldottam meg, hogy az adott színű király helyéről megnéztem hova tudna lépni/ütni, ha más típusú figura lenne, és megnéztem, hogy azokon a helyeken áll-e olyan típusú ellenséges figura, ha igen akkor sakkban van a király.

A program egyik alapköve a keresés függvény, ami egyik színnél megnézi, hogy az megadott helyen áll-e figura.

A játék alapvetően kattintásokkal működik, a figurára kattintva választjuk ki, és a megjelölt helyekre kattintva választhatjuk ki a lépést helyét.

Adatszerkezet

Az adatszerkezetnek két fő eleme van, az első bábuk listája amiből kettő van, egy a fehérnek, egy pedig a feketének, mindkettőben 16 bábu struktúra van elmentve, minden bábunak el van mentve a színe, x és y koordinátája és típusa(p-gyalog(pawn), r-bástya(rook), b-futó(bishop), h-huszár(horse/knight), q-királynő(queen), k-király(king)). A második a lépések tárolására készített láncolt lista, egy lista elem két x és két y koordinátát tárol, ezeken kívül pedig egy karaktert, ami, ha egy gyalog felfejlődött, azzal hogy az utolsó sorba ért, megmutatja mivé fejlődött.

```
typedef struct Babu{
    int x;
    int y;
    char c;
    bool f;
}Babu;
typedef enum Piece {
```

```
typedef struct Lepes{
    int honnan1;
    int honnan2;
    int hova1;
    int hova2;
    char pro;
    struct Lepes *kov;
}Lepes;
```

Mivel a bábuk száma a játék során csak csökkenhet, ezért a tárolásra használt mód nem számít túl sokat, mivel ez a szám olyan alacsony, hogy technikailag nem változtat semmit a program gyorsaságán.

A lépések tárolására használt lista mindig csak a végére fűzéssel változik ezért ez a legoptimálisabb módszer a tárolásra.

Főbb függvények:

void uj_lepes(Lepes **eleje, int x1, int y1, int x2, int y2, char pro);

Eltárolja egy új listaelembe az aktuális lépést. Az első paramétere a láncolt lista elemére mutató pointer címe, amit a legelső lépés mentésekor változtatunk. Ezután két x-y pár következik, hogy honnan lépett a bábu és hova. Az utolsó érték gyalog fejlesztésnél szamít, ilyenkor bábú típust jelöl, egyébként egy space karakter.

void felszabadit(Lepes *eleje);

A láncolt lista felszabadításához írt függvény. A lista elejét kapja meg bemenetnek.

(A programban nincs használatban (ki van kommentelve), mert a fájlba kiírás alatt fel is szabadítom a listát.)

void babuk_letrehozas(Babu *feher, Babu *fekete);

A program elején használom. Létrehozza a 16 fehér, és 16 fekete figurát.

A két szín dinamikusan lefoglalt helyére mutató pointert kapja meg.

Babu *keres(Babu *eleje, int x, int y, int db);

Az egyik szín listájában megkeresi a beadott kordinátán álló bábut. Visszatérési értéke a bábura mutató pointer, ha megtalálta, ha nem, akkor NULL pointert ad vissza.

Babu* lehetsegeslepesek(Babu *elem, Babu *feher, Babu *fekete, int* db);

Egy paraméterként kapott bábuhoz visszaad egy dinamikus listát, a lehetséges lépésekkel. Ezek a lehetséges lépések nem veszik figyelembe, ha a király sakkban van. Azt a problémát máshogy oldottam meg. A paraméterek a bábura mutató pointer, a két figuralista eleje, és egy db érték címe, amit kint használunk, hogy tudjuk hány eleme van a listának.

Babu *gyalog_lepes(Babu *elem, Babu *feher, Babu *fekete, int* db);

A lehetséges gyaloglépéseket kezeli.

Babu *lo_lepes(Babu *elem, Babu *feher, Babu *fekete, int* db);

A lehetséges lólépéseket kezeli.

Babu *futo_lepes(Babu *elem, Babu *feher, Babu *fekete, int* db);

A lehetséges futólépéseket kezeli.

Babu *bastya_lepes(Babu *elem, Babu *feher, Babu *fekete, int* db);

A lehetséges bástyalépéseket kezeli.

Babu *kiralyno_lepes(Babu *elem, Babu *feher, Babu *fekete, int* db);

A lehetséges királynőlépéseket kezeli.

Babu *kiraly_lepes(Babu *elem, Babu *feher, Babu *fekete, int* db);

A lehetséges királylépéseket kezeli.

bool utes(Babu *elem, Babu *feher, Babu *fekete);

Egy bábu lépése után hívom meg, és megnézi, hogy van-e a másik színnek azon a mezőn figurája, ha van akkor „leüti”, azaz a játéktéren kívülre helyezi. A bemenő adatok a figura, ami lépett és a fehér és fekete bábuk listája

void shortcastle(Babu *elem, Babu* feher, Babu* fekete);

Speciális királylépéseket kezel, a rövid sáncolást. Ha a király épp sáncol akkor hívom meg és a bástyát áthelyezi a megfelelő helyre.

void longcastle(Babu *elem, Babu *feher, Babu *fekete);

A rövid sáncoláshoz hasonlóan, ez a függvény, a másik irányú sáncolást kezeli, szintén a bástyát helyezi át.

char promotion(SDL_Renderer *renderer, SDL_Texture *pieces, Babu* elem);

A gyalogok felfejlesztésére való. Az alsó 100 pixeles sávba kirajzolja a lehetséges fejlődéseket, ezekre kattintva lehet kiválasztani melyik tisztre szeretnéd változtatni a gyalogot. (Nem működik tökéletesen, először a választott tisztre kell kattintani, de csak akkor rakja a táblára az egységet, ha egy másik lehetséges tisztcserére kattintasz.)

bool sakk(bool vilagos, Babu *feher, Babu *fekete);

Egy állásban megnézi, hogy a fekete vagy a fehér király sakkban van-e. (bool értéktől függően az egyiket)

bool matt_e(bool kivansoron, Babu *feher, Babu *fekete);

A szerint, hogy ki van soron megnézi van-e értelmes lépés a pozícióban, ha van false értéket ad, ha nincs true-t.

void babu_rajz(SDL_Renderer *renderer, SDL_Texture *pieces, Babu *babuk);

Kirajzolja a bábukat a táblára a babu_rajzol függvény segítségével.

void babu_rajzol(SDL_Renderer *renderer, SDL_Texture *babuk, Piece melyik, int x, int y);

Kirajzol egy bábut a pieces.xls fájlból kivágott képrészlet segítségével.

Forrásfájlok

Négy c fájlban van a program, amiket három header fájl köt össze, az első a main.c, amiben a main függvényen kívül nincs más függvény.

A második a fuggvenyek.c ami az összes fontosabb függvényt tárolja.

A harmadik a lepesek.c, ami a lehetséges lépések megkereséséhez szükséges függvényeket tartalmazza.

A negyedikben a grafikus függvények vannak.

Sakk felhasználói dokumentáció (C3FBCD)

A sakk program a sakkjáték szabályait betartva egy virtuális játékot kezel.

A program a terminálból indul, ahol egy kis menüből választható három opció, mindegyik egy-egy számmal jelölve. A nem megfelelő karakter bevitelét a program kezeli, ilyenkor új karaktert kell beírni.

Minden opció után meg kell adni egy fájl nevet, ez az annak a fájlnak a neve, amibe a program írni fogja a lépéseket, vagy annak a neve, amiből beolvassa a lépéseket, a folytatás opciónál innen olvas és ide ír. Ezzel, hogy meg lehet adni a fájl nevét később könnyebb megtalálni a fájlokat.

A fájlok nevét a beolvasásnál kiterjesztéssel együtt kell beírni (azaz nem elég „lepesek”-t írni, hanem „lepesek.txt”-t kell írni, természetesen csak akkor ha szöveg állományt olvasunk be).

Az új játék opciónál egy kezdőpozícióból kezdődő játékot lehet játszani. Egy figurára kattintva lehet kiválasztani melyikkel szeretnél lépni. majd a megjelölt helyek közül lehet választani, hogy hova lépjen a bábu. (Figyelem! Ha nem sikerül lépni így, akkor lehet, hogy sakkban vagy.)

Az új játék funkcióban játszott játékot félbe lehet hagyni, egyszerűen csak be kell zárni az ablakot. Ezt később lehet folytatni.

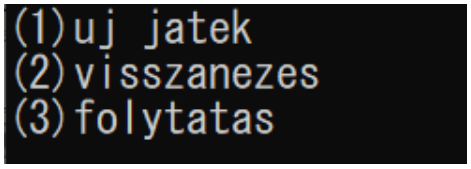
A játék észreveszi a mattot, ilyenkor kiírja a győztes színt.

A visszajátszás funkció betölt egy játékot, amit a képernyőre kattintással lehet léptetni. Az utolsó lépés utáni kattintás bezárja az ablakot.

A folytatás funkció egy félbehagyott játékot folytat, a félbehagyott pozíciótól.

Mindegyik funkció automatikusan elmenti a játékot a megadott nevű fájlba.

A programból az ablak bezárásával lehet kilépni.



menü

játéktábla