Sakk - Programozói dokumentáció

A tábla felépítése és a bábuk

Egy hagyományos sakk tábla 64 darab fekete-fehér négyzetből áll. A táblára 6 darab különféle bábut helyezünk előre meghatározott hagyományos sorrendben. Minden bábunak van egy színe, illetve egy típusa. Ez határozza meg a helyét.

Adatszerkezetek választása

Több nagyon fontos építőkövei vannak a programnak, a bábuk, aminek egy típusa és egy színe van, a másik pedig maga a tábla, aminek mezői vannak, pontosabban 64 darab.

Egy bábu strukturanak ilyen a felépítése:

```
typedef struct Babu{
    Tipus tipus;
    Babuszin szin;
}Babu;
```

A típus, illetve a szín két felsorolt adattípus, amik egyesével, így néznek ki:

```
typedef enum Tipus{
    kiraly,vezer,bastya,futo,lo,paraszt,nincs
}Tipus;
```

```
typedef enum Babuszin{
    feher,fekete,nincs2
}Babuszin;
```

Mindkét esetben szükséges volt egy nincs-et is felsorolni, mert úgy lett feltöltve a tábla, hogy ahol a kezdő pozícióban nincsenek bábuk, oda egy ilyen került, a későbbi egyszerűbb adatkezelés miatt.

A második nagyobb építőkő, az a tábla, itt a struktúra viszonylag egyszerűen van kialakítva, csupán egy darab sort tartalmaz, ami annál lényegesebb, hogy ez egy Mezo típusu elem, ami egy dupla pointer. Ez azért volt szükséges, mert egy, a táblán lévő mezőnek két darab koordinátája van, egy x és egy y, ezért volt szükséges egy ilyet definiálni:

```
typedef struct Palya{
    Mezo **palya;
}Palya;
```

Egy mező struktúra, pedig egy Babu típusu valtozóval és egy x,y koordinátával rendelkezik:

```
typedef struct Mezo{
    Babu babu;
    int x,y;
}Mezo;
```

Egy másik nem említett, de igencsak hasznos részlet, ami egy összetett adatszerkezetet igényelt, ami nem más mint a megjelenítés a képernyőre, egy kirajzol struktúra:

```
typedef struct rajzolo{
    SDL_Texture *texture;
    SDL_Window *window;
    SDL_Renderer *renderer;
    Palya *p;
}rajzolo;
```

Rendelkezik egy pályával, hogy elérje a táblát, illetve több beépített SDL könyvtári adattípusokkal.

A program működését vezérlő fő függvények

A tábla lényeges függvényei

Palya *Field() - Ebben a függvényben történik nagyrészt a dinamikus memória kezelés, itt foglalja le az adatokat a memóriában, továbbá beállítódik a mezőknek az x,y koordinátái a táblán, és itt hívodik meg a kezdes() függvény is. Visszatérési értéke egy Palya adatszerkezet.

void kezdes(Palya *p) - Egy Palya pointer-t vár a paraméterébe. Itt történik meg a bábuknak a pozícionálása a táblán. Itt dől el, hogy egy bábuból, hogy lesz fekete vagy fehér, illetve, hogy a egy mező, hogy lesz foglalt vagy nem foglalt. Nem szabad megfeledkezni arról sem, hogy milyen sorrendben leszenk fent a bábuk a táblán, itt ez a feladat is sorra kerül.

```c
void kezdes(Palya *p){
    for(int i = 0;i<8;i++){
       for(int j = 0;j<8;j++){ //Hozzafuzzuk a kulonbozo mezokhoz a szint illetve
           if(j>1 && j<6){
            p->palya[i][j].babu.tipus = nincs;
            p->palya[i][j].babu.szin = nincs2;
           }else{
               if(j<2){
                p->palya[i][j].babu.szin = fekete;
                if(j==1){
                    p->palya[i][j].babu.tipus = paraszt;
                }
           }
            if(j>5){
                p->palya[i][j].babu.szin = feher;
            if(j==6){
                 p->palya[i][j].babu.tipus = paraszt;
               }
           }
           }
           if(j==0 || j==7){
                p->palya[0][j].babu.tipus = bastya;
                p->palya[1][j].babu.tipus = lo;
                p->palya[2][j].babu.tipus = futo;
                p->palya[3][j].babu.tipus = vezer;
                p->palya[4][j].babu.tipus = kiraly;
                p->palya[5][j].babu.tipus = futo;
                p->palya[6][j].babu.tipus = lo;
                p->palya[7][j].babu.tipus = bastya;
           }
       }
    }
}
```

void felszabadit(Palya *p) - Ez a függvény felelős a lefoglalt adamennyiség felszabadításáért. Paraméterbe egy Palya adatszerkezet kerül, így fordított sorrendben fel lehet szabadítani a Field() függvényben lefoglalt területet. Azért visszafelé, mert ez egy pointer és ha csak magában azt írnánk ebbe a függvénybe, hogy free(p), akkor a többi területre mutató pointerek memóriaszivárgást okoznának.

A megjelenit.c

void sdl_init(rajzolo *r, int szeles,int magas) - Itt történik minden, ami ahhoz kell, hogy vizuálisan megjeleníthető legyen maga a program. Egy rajzolo pointer-t vár paraméterben, illetve két darab számot, az alapján, hogy mekkorára kívánjuk növelni az megnyíló ablak méretét. Ez egy alapfüggvény, ha az SDL könyvtárat szeretnénk használni.

void indul() - Lényegében ez a függvény a legfontosabb, mert ez hívódik meg a main.c-ben, itt inicializálunk egy rajzolo *r nevu pointert, amit meghivunk a megjelenites(rajzolo *r) függvényben. Még ugyanitt meghívjuk a két felszabadító 3 függvényünket is a lefoglalt memória miatt, hogy felszabadítsuk őket, ismételten, hogy megelőzzük a memória szivárgást.

void megjelenites(rajzolo *r) - A vizuális megjelenítés végett, itt rajzolja ki a bábukat, illetve a mezőket a táblára a függvény egy bizonyos feltétel alapján

baburajz(rajzolo *r,Babuszin sz,Tipus t,int i,int j) - Itt egy bizonyos szín és egy típus alapján kiválasztja a bábuk képének forrásfájljából a megfelelő színű és típusú bábut, majd a cél koordinátára másolja

azt. Ez abból a szempontból előnyös, hogy nem kell 12 darab képet tárolni egy mappában, hanem elég egyet, amiben benne van mint a 12 darab bábú.

```
void megjelenites(rajzolo *r){
    for(int i = 0;i<8;i++){ //Vegig megyunk a 8 soron és oszlopon
        for(int j=0;j<8;j++){
            if((i+j)%2==0){ //Megnezzuk, hogyha a i+j osszege oszthato-e kettovel maradek
                boxRGBA(r->renderer,100+100*i,100+100*j,100*i,100*j,221,230,238,255); //E
                baburajz(r,r->p->palya[i][j].babu.szin,r->p->palya[i][j].babu.tipus,i,j);
            }else{
                boxRGBA(r->renderer,100+100*i,100+100*j,100*i,100*j,123,155,176,255);
                baburajz(r,r->p->palya[i][j].babu.szin,r->p->palya[i][j].babu.tipus,i,j);
            }
        }
    }
    SDL_RenderPresent(r->renderer);
}


void baburajz(rajzolo *r,Babuszin sz,Tipus t,int i,int j){
    if(t != nincs){
        SDL_Rect src = {(t%6)*62+10, (sz%2)*60+10, MERET, MERET}; //A koncepcio az, hogy a
        SDL_Rect dest = {100*i,100*j,MERET+40,MERET+40};
        SDL_RenderCopy(r->renderer,r->texture,&src,&dest);
    }
}
```

void szabadit(rajzolo *r) - Koncepció ugyanaz, mint a felszabadit()-nál, itt is felszabadítjuk a memóriát, de előbb használjuk az SDL beépített függvényeket, ahhoz, hogy megsemmisítsük a renderer-t, a texture-t, illetve a window-t, majd ezekután szabadítjuk fel a lefoglalt memóriát.

Itt történik miden amit a játéknak lényegében csinálnia kell, a szabályok betartásától a végjátékig minden.

Függvények

Void felirat (rajzolo *r,int x,int y,char *szoveg,int bMeret,int red,int g,int b,char *srcFile) - Lényegében egy tetszőleges szöveget ír ki a képernyőre, tetszőleges színben és betűtípussal

Void indul() - Volt már szó erről a függvényről de térjünk vissza rá, mert itt történnek a lényeges függvény hívások és az események.
Mindennek a while ciklus az atyja, hiszen ő dönti el, hogy a programnak vége lesz, vagy nem. A kattintásokat itt értelmezi a program és teszi őket megvalósíthatóvá. Csak a bal egérgomb használatát nézi. Két fő részből áll a bal egérgomb, amikor kijelölte a bábut és mikor egy megadott helyre szeretné mozgatni azt. Mivel a sakk szabályai azt követelik meg, hogy egy fehér lépés után egy fekete jöjjön, ezért be lett vezetve egy db nevű int változó, akinek megváltoztatjuk az értékét minden második kattintásnál, ha a kattintás helyes volt. Ez bitléptetéssel lett megoldva.
Többféle kattintás esetén több a lehetőségek száma is. Jól mutatja ez azt, hogy a második kattintás során megnézi a program, hogy a megadott hely ahová szeretni mi is, illetve, hogy mivel szeretne odalépni, különleges eset, amikor egy paraszt beér az ellenfél kezdősorába és átalakul, jelen esetben vezérré.
Mivel egy kép fájl tartalmazta az össze bábut, ezért ha lépünk egyet, akkor megváltozik annak a mezőnek a típusa és a színe, ahova és ahonnan léptünk, ha a lépés helyes volt.
Négy lényeges változó van még ebben a függvényben, az xkatt,ykatt és az m.x,m.y változók. Ezekközül a két első tárolja az első kattintás helyét, második kettő pedig a második kattintás helyét és az első kattintásnál megvizsgáljuk, hogy a kattintott hely az nem üres, mert ha igen akkor ne csináljon semmit.

A függvény elején létrehoztunk egy Mezo m típusú változót, amiben eltároljuk annak a mezőnek az értékét, amire először kattintott a játékos

Ha sorban haladunk, akkor az első függvény a **void castle(int szin,rajzolo *r,int kezdoX,int kezdoY,int mostX,int mostY,bool rovid,FILE *fp)** - Ezt hívjuk meg lentebb a lephet függvényben a király lépésénél, de arról később. A függvény lényegi funkciója, sáncolt lépést lehessen végrehajtani, ha a király nem mozdult még meg és az egyik bástya, amelyik oldalra akar sáncolni, az sem mozdult még el.

bool ideOda(int elozoX,int elozoY,int mostX,int mostY,Palya *p,int szin2) - Az ideOda függvény megnézi, hogy érvényes-e a lépés fel- vagy lefele. Mivel egyik koordinátája kötött, ezért csak az egyiket kell csökkenteni vagy növelni és közben figyelni kell, hogyha a kezdőpont és a végpont koordinátái között van-e valami, ha van akkor a visszatérési értéke hamis, ha nincs akkor igaz. A szin2 paraméter azt vizsgálja, hogy ahova szeretett volna lépni az illető, az nem-e a bábunak a saját színe.

bool keresztbe(int elozoX,int elozoY,int mostX,int mostY,Palya *p,int szin2) - Lényegében ugyanazt csinálja, mint az ideOda függvény, csak átlósan és annyi különbséggel, hogy meredekséget számol a koordinátákból és az alapján dönti el a függvény, hogy igazat vagy hamisat adjon vissza, nyílván ez még függ az előző függvényben említett dolgoktól is.

bool lovacska(int elozoX,int elozoY,int mostX,int mostY,Palya *p,int szin2) - Ahogy a neve is mutatja, ez a lólépést vizsgálja meg, olyan módon, hogy van 8 lehetséges lépése a lónak, ezt két tömbben, egy X és egy Y nevű int tömbben tároljuk, és végig megyünk ezeken a tömbökön, és ha a kezdőpont koordinátáit összeadjuk a tömbben található számokkal és ha ezek összege egyenlő a végpont koordinátáival, akkor a lépés jó és return igaz, ha nem jó akkor hamisat fog visszadni

bool sakk(Palya *p,int mostX,int mostY,int szin,int szin2) - Itt végig megyünk az egész pályán és ha az ellenfél bábúnak van bármilyen lépése a saját királyunkra, akkor igazat ad vissza, ha nincs akkor hamisat.Így tudjuk megelőzni azt, hogyha a király sakkban van és vele akarunk lépni, akkor ne tudjunk újra sakkba lépni.

bool lephet(int elozoX,int elozoY,int mostX,int mostY,Tipus tBabu,Babuszin szBabu,Palya *p, rajzolo *r,FILE *fp) - Lényegében itt nézzük meg, hogy az első kattintás során milyen bábura kattintott a játékos, majd ha lépni szeretne vele, akkor megnézzük, hogy a célpont az a bábu lépései közé tartozik, ha benne van, akkor odaléphet és igazat ad vissza, ha nincs benne, akkor hamisat és nem lép oda.