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 tabla 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.

```
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 lényeges függvényei

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

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.

Egyéb függvények és definiált típusok

void poz(int x, int y, Mezo *m) - Itt a Mezo tipusu m pointerhez hozzárendeli a megfelelő x és y koordinátákat.

enum {MERET: 52} - Ez ahhoz kell, hogy mikor kiolvassuk a képből a bábukat, megtudjuk, mekkora is egy ilyen bábu, ez a MERET, a képben levő bábu méretét jelöli.