Aknakereső játék – NHF 4. Programozói dokumentáció

A játék folyamata

A program indulásakor kiírja a konzolba az eddigi legjobb eredményeket, vagyis a dicsőséglistát, továbbá itt lehetősége van a felhasználónak kiválasztani a nehézségi fokozatot. Ezután a felugró ablakban történik igazán a játékmenet: A győzelem után (megtalálta az összes aknamentes mezőt) a felhasználó kiléphet, visszatérve a konzolba begépelheti nevét egy .txt fájlban tárolódó dicsőséglistába, azonban ha veszít akkor a program automatikusan befejeződik.

Adatszerkezetek választása

A program három darab adatszerkezettel rendelkezik. Az egyik ezek közül magának a pályának az adatszerkezete, mely igazából a mezők számontartásáért felelős. Számon tartja, hogy egy mező zászlóval megjelölt-e, felfedték-e már, és azt, hogy mennyi akna van közvetlen környezetében vagy esetleg aknás mezőről beszélünk. Ezzel a mezőhöz tartozó számmal való dolgozás nagyon egyszerű ugyanis, ha aknás mezőről beszélünk, akkor -1 az értéke, hiszen normál esetben ilyen nem jöhet létre, szóval egyértelműen meg tudjuk állapítani, hogy egy mező aknás-e vagy sem. A mezők adatai egy kétdimenziós dinamikus tömbben helyezkednek el. A dinamikus tömb szélességét és magasságát kettővel nagyobbra foglaljuk le, hogy egy 0-val feltöltött keretet adjunk az egész tömbnek. Ez a keret lehetővé teszi, hogy könnyebben végezhessünk függvényekkel műveleteket. Nagy előnye, amikor megnézzük mind a 8 oldalra egy mező környezetét, akkor nem kell külön feltételeket szabjunk az ellenőrzésnek, hanem a tényleges játéktéren egyetlen egy függvénnyel meg tudjuk állapítani minden egyes mező „mennyi" értékét.

```
typedef struct Palya{
    bool felfedve;
    bool zaszlos;
    int mennyi;
}Palya;
```

A kezelendő dicsőséglista is egy külön struktúra, amely tartalmazza egy adott sorban szereplő becenevet, a játék teljesítéséhez szükséges időt és az előre meghatározott nehézségi fokozatot. A teljes dicsőséglista egydimenziós Dicsoseglista típusú tömbben helyezkedik el. Ezt a tömböt fogjuk később rendszerezni a toplista megjelenítéséhez.

```
typedef struct Dicsoseglista{
    char becenev[51];
    int ido;
    int nehezseg;
}Dicsoseglista;
```

Végül, de nem utolsó sorban a játékhoz szükséges képek struktúrája, amely tárolja, hogy mikor, milyen képet kell megjelenítenie az egyes függvényeknek.

```
typedef enum Ikon {
    zero, one, two, three, four, five, six, seven, eight,
    bomb, boom, facingdown, flagged, bombx
}Ikon;
```

A program moduljai és fő függvényei

- **jatekallas.c:** A matematikai műveletekért, a játékállás számontartásáért, maga az aknakereső játék funkcióinak és szabályainak létrehozásáért, továbbá a teljes dicsőséglista kezeléséért felelős.

Függvényei:

```
void inic(int *magassag, int *szelesseg, int *aknak, int *nehezseg)
Elvégzi az inicializálást.

void dicsoseglista_beleir(int nehezseg, int ido)
Ha megnyeri a felhasználó a játékot, akkor ez a függvény végzi el a fájlba való
írási folyamatot.

Dicsoseglista* dicsoseglista_foglal(void)
Lefoglalja a dicsőséglistának a dinamikus tömböt, hogy később rendezni tudjuk az
elemeit.

void dicsoseglista_rendez(Dicsoseglista *lista, int meret)
Közvetlen kiválasztásos módszerrel rendezni a dicsőséglistát.

void dicsoseglista_becenevek(Dicsoseglista *lista, int nehezseg, int meret)
Ez a függvény írja ki a képernyőre a legjobb x nevet egy adott nehézségi fokozaton.

void dicsoseglista_kiir(Dicsoseglista *lista, int meret)
A függvény az előre megadott nehézségi fokozatokra kiírja a legjobb időket.

void dicsoseglista_elkeszit(Dicsoseglista *lista)
Összefoglalja az előző függvényeket és elvégzi a rendezést, továbbá a kiírást.

void nehezseg_kivalaszt(int valasz, int *magassag, int *szelesseg, int *aknak)
A nehézségi szint kiválasztásáért felel.

Palya** dintomb_foglal(int magassag, int szelesseg)
Dinamikusan foglal egy kétdimenziós Palya típusú tömböt.

void dintomb_felszabadit(int magassag, int szelesseg, Palya **tomb)
Felszabadítja a dinamikus tömböt.

void dintomb_nullaz(int magassag, int szelesseg, Palya **tomb)
A Palya típusú tömb "mennyi" kezdeti értékét nullára állítja.

void dintomb_akna_letesz(int magassag, int szelesseg, int aknak, Palya **tomb)
Véletlenszerűen lehelyez aknákat a tömbben, olyan helyekre, ahol még nincs.

void dintomb_aknanelkul_megszamol(int ii, int jj, Palya **tomb)
Egy mezőre mind a 8 irányban megállapítja, hogy hány akna van körülötte.

void dintomb_aknanelkul_letesz(int magassag, int szelesseg, Palya **tomb)
Az összes mezőre elvégzi a szomszédos aknák megszámolását a
dintomb_aknanelkul_megszamol függvénnyel
```

- **megjelenites.c:** Az SDL platformfüggetlen multimédiás függvénykönyvtár kezelésére szolgál, ez a modul „szembesíti” igazán a felhasználót az igazi játékkal.
  Függvényei

```
Uint32 idozit(Uint32 ms, void *param){
    SDL_Event ev;
    ev.type = SDL_USEREVENT;
    SDL_PushEvent(&ev);
```

```
    return ms;
}
```
A struktúra az időzítés megvalósításához szükséges tulajdonságokat tartalmazza.

`void sdl_init( char const *felirat, int szelesseg, int magassag, SDL_Window **pwindow, SDL_Renderer **prenderer)`

Létrehozza az ablakot.

`void ikon_rajzol(SDL_Renderer *renderer, SDL_Texture *ikonok, Ikon melyik, int ikonx, int ikony)`

Kirajzol egy ikont; a forrás a betöltött png, a cél nevű képre rajzol.

`void szoveg_rajzol(TTF_Font *font, SDL_Surface *felirat, SDL_Texture *felirat_t, SDL_Rect hova, SDL_Renderer *renderer, SDL_Color fekete, int szelesseg, int magassag)`

A különböző szöveg rajzolásáért felelős függvény.

`void teglalap_rajzol(SDL_Renderer *renderer, int x, int y, int w, int h , int r , int g, int b)`

Mindenféle téglalapok rajzolásához szükséges függvény, ennek segítségével lehet véghez vinni a zászlók és az eltelt idő kirajzolását.

`void palya_rajzol(SDL_Renderer *renderer, SDL_Texture *ikonok, Ikon melyik, int szelesseg, int magassag)`

Kirajzolja a pályát kezdetben a lefedett ikonokból, pont annyi mezővel, amit a felhasználó megadott.

`void adatok_rajzol(SDL_Renderer *renderer, TTF_Font *font, char* szoveg, int szelesseg, int magassag)`

A függvény elvégzi a tájékoztató szövegek kiírását ablakban.

`void infoszoveg_elkeszit(SDL_Renderer *renderer, TTF_Font *font, int magassag)`

Az adatok_rajzol függvény segítségével kiírja a „Megmaradt aknak” és „Eltelt ido” szövegeket az ablakba.

`void zaszloszam_rajzol(TTF_Font *font, SDL_Renderer *renderer, int szelesseg, int magassag, int zaszlok)`

Az adatok_rajzol függvény segítségével kiírja az aktuális zászló / megmaradt aknaszámot.

`void ido_rajzol(TTF_Font *font, SDL_Renderer *renderer, int szelesseg, int magassag, int ido)`

Ismét az adatok_rajzol segédfüggvénnyel kiírja az eltelt perceket és másodperceket.

`void tisztit_ido(SDL_Renderer *renderer, int szelesseg, int magassag)`

Az időt számoló szöveg területét tisztítja le a teglalap_rajzol függvénnyel.

`void tisztit_zaszlo(SDL_Renderer *renderer, int szelesseg, int magassag)`

A zászlót számoló szöveg területét tisztítja le a teglalap_rajzol függvénnyel.

`void hatter(SDL_Renderer *renderer, int szelesseg, int magassag)`

A függvény egyszerűen csak megrajzolja az információs szövegek mögötti hátteret a játék kezdetén a teglalap_rajzol segédfüggvénnyel.

`void vesztett_rajzol(SDL_Renderer *renderer, SDL_Texture *ikonok, Ikon melyik, Palya **tomb, int szelesseg, int magassag)`

```
Felelős az összes akna kirajzolásáért, miután a felhasználó rányomott egyre. Ezen
felül a korábban zászlóval megjelölt aknákat x el jelöli.

void vegsoszoveg_elkeszit(SDL_Renderer *renderer, TTF_Font *font, int szelesseg,
int magassag, char* statusz, char* kilepes)
Kiírja győzelem / vereség utáni szöveget az adatok_rajzol függvénnyel.

void rekurziv(SDL_Renderer *renderer, SDL_Texture *ikonok, Palya **tomb, int ikonx,
int ikony, int szelesseg, int magassag)
Felderíti az aknamentes környezetű mezőket.

void jatekvege(SDL_Renderer *renderer, TTF_Font *font, int szelesseg, int magassag)
A vegsoszoveg_elkeszit függvény által kiírt szöveg mögé készít egy hátteret a
teglalap_rajzol függvénnyel.

void kilepesgomb(int szelesseg, int magassag, int xpoz, int ypoz)
Érzékeli, hogy a felhasználó rányom-e a kilépésgombra, miután bezárja az ablakot.
```

- **main.c:** Ez a modul hozza létre az ablakot, tölti be a képeket, a betűtípust, rajzolja meg a feliratot, és tartalmazza magát az eseményvezérelt hurkot.

A megfelelő függvények megalkotják a játékteret majd az eseményvezérelt hurok teszi „játszhatóvá” a programunkat.
Az eseményvezérelt hurok SDL_Event típusú eventek szerint vezérli a játékot. Különböző eventek lehetnek a bal vagy jobb egérgomb lenyomása, a kilépés vagy akár az idő múlása. A megfelelő függvények segítségével ezekre az eventekre reagál a program, majd elvégzi a megfelelő lépéseket.

A bal egérgomb lenyomása felfedi a még rejtett mezőt vagy mezőket, a jobb egérgombbal zászlót helyezhetünk, vehetünk fel, amit jelez a segítő szöveg is. Az idő múlását nyomon követhetjük másodpercenként, amelyet perc:másodperc formában írunk ki. A bombára nyomás vagy győzelem esetén felugrik egy szövegdoboz, mely segítségével ki is léphetünk.