Programozói dokumentáció

Istvánfi Csenge
MQQHER

Az amőba játék menüjei és a játékpálya ugyanabban az ablakban jelennek meg SDL-esen, mely ablak mérete állandó.
A beállítható többféle nehézségi szint kétféle stratégia alapján működik, de mindkettőnek ugyanaz a logikája: megnézi, hogy minden meghatározott számú szomszédos mezőn mennyi van a keresett fajta bábúból, és ennek függvényében rakja le a gép a saját bábuját.

A program felépítése

A program több .c és .h fájlból áll. A kirajzolas modul tartalmazza az SDL-es függvényeket és amik hozzájuk kellenek. Például a gombok, menü kirajzolása, az X és a kör mezőmérethez igazított kirajzolása. Az összes szükséges struktúra egy strukturak nevű heather-ben van. A gép beállítható stratégiája, ami szerint játszik szintén külön van egy gep_strat nevű modulban. A fájlba mentés és onnan beolvasás, valamint az, hogy ki győz szintén különálló modulok. A main.c-ben pedig összefűzöm a modulokban lévő függvényeket, itt követhető végig a program menete.

Maga a program alapvetően két részre bontható: a menü(k)re, játékbeállításokra és magára a játékra. Más tekintetben a program felépítése pedig úgy bontható két fő részre, hogy az első felében vannak leírva a megjelenítéshez szükséges függvények, a második felében pedig van az eseményvezérlés (a mainben).

Modulok

DEBUGMALLOC: Az InfoC oldalról letöltött heather fájl, amivel a memóriakezelést ellenőriztem.

STRUKTURAK:

Ez csak egy heather fájl, nem tartozik hozzá .c fájl.

- *typedef enum Babu*
  { ures,
  kor,
  iksz,
  } Babu;
  Ennek a felsorolt típusnak a segítségével határozhatjuk meg az egyes mezők állapotát, azaz, hogy milyen bábu van rajta, vagy üres-e.

- *typedef struct RubrikaAdatok*
  {
  double balfelsoX;
  double balfelsoY;
  double jobbalsoX;
  double jobbalsoY;
  Babu babu;
  *struct Szomszedok *szomszedok;*

} RubrikaAdatok;
Ennek segítségével definiálhatjuk a rubrikákat (mezőket). A Babu típusú változó jelzi a mező pillanatnyi állapotát, a másik 4 változó pedig az egyes mezők koordinátáit tárolják. Erre akkor lesz szükség, amikor egy bábut el akarunk helyezni egy mezőben.

- *typedef struct Gombtarolas*
  {
  int szelesseg, magassag;
  *char *felirat;*
  int x,y;
  int azonosito;
  } Gombtarolas;
  Ez a programban szereplő gombok tulajdonságait foglalja egybe. Az azonosító segítségével a gombokat tömbbe rendezve egyszerűen tudjuk megkülönböztetni őket.

- *typedef struct Tabla*
  {
  int mags;
  int szels;
  *RubrikaAdatok **mezo;*
  } Tabla;
  Az aktuális tábla állás paramétereit gyűjti össze, tehát a pálya szélességét, magasságát és a mezők állapotát.

- *typedef struct ListaElem*
  {
  *RubrikaAdatok *adat;*
  *struct ListaElem *kov;*
  } ListaElem;
  A gép stratégiájában használt láncolt listák létrehozásához szükséges.

- *enum Strat*
  {
  konnyu,
  kozepes,
  nehez
  };
  A gép stratégiájának beállításához kell.

- *enum Kikezd*
  {
  gep,
  jatekos
  };
  A játékbeállításoknál a kezdő játékos beállításához kell.

KIRAJZOLAS:

Itt vannak összegyűjtve a megjelenítéshez szükséges kirajzoló függvények. Mindegyik void típusú.

Az alábbiak a menüknél használt kirajzoló függvények:

- *fomenu(SDL_Renderer *renderer, int *melyik, Gombtarolas *gombtomb)* :
  A főmenü két gombját rajzolja ki (ezért van szükség a gombtomb-re paraméterként). Minden gombhoz tartozik egy téglalap és egy felirat is. Attól függően, hogy éppen melyiken állunk, annak a téglalapnak színe kék lesz, a másiké piros (erre van a for ciklus a függvényben).
- *beallitas_menugomb(SDL_Renderer *renderer,int *menulepteto, Gombtarolas *gombtomb)* :
  Az előzőhöz hasonlóan működik azzal a különbséggel, hogy itt nem 2, hanem 4 gomb közül színez át egyet kékre (éppen amelyik beállítási ponton vagyunk).
- *magszel (SDL_Renderer *renderer,int *mags, int *szels, Gombtarolas *gombtomb)* :
  Ebben a függvényben a pálya szélességének és magasságának beállításának a kezelése van. Mivel az SDL függvények közül én a stringRGBA-t használtam, ahhoz, hogy sztringekből számokat csináljak, karaktertömböket és sprintf( ) függvényeket használtam, így a tömböket a kívánt (int) formátumban tudtam átadni a függvényeknek.
- *kikezdi (SDL_Renderer *renderer, int *kikezd, Gombtarolas *gombtomb)* :
  A függvény vagy a „gep” vagy a „jatekos” feliratot jeleníti meg attól függően, hogy épp melyiken állunk.
- *gep_strat (SDL_Renderer *renderer, int *strat, Gombtarolas *gombtomb)* :
  Ez az előzőhőz hasonló szerepet tölt be, csak itt a több opció miatt switch – case szerkezettel döntjük el, hogy melyik felirat (konnyu/kozepes/nehez) íródik ki.

A játékpályához szükséges és azzal egyszerre látható elemek függvényei:

- *palyarajz (SDL_Renderer *renderer, int* mezooldal, int *szels, int *mags)* :
  Ez a függvény rajzolja ki a pályát alkotó vonalakat a mezooldal változóban tárolt optimális mező oldalhosszal számolva.
- *kor_rajz(SDL_Renderer *renderer,Tabla *tabla,int *mezooldal)* :
  Piros kört rajzol egy mezőbe.
- *iksz_rajz(SDL_Renderer *renderer,Tabla *tabla)* :
  Egy kék X-et rajzol egy mezőbe. (Az X-et két vonallal hozza létre amik [a sarkoktól kis távolsággal bennebbről] a mező átlói.)
- *mentes(SDL_Renderer *renderer, Gombtarolas *gombtomb*) :
  A játék állásának elmentésére szolgáló gomb rajza.
- *uj_jatek_kezd(SDL_Renderer *renderer, Gombtarolas *gombtomb*) :
  Az új játé kezdéséhez szükséges gomb kirajzolása.
- *dontetlen_kiir(SDL_Renderer *renderer, int SZELESSEG, int MAGASSAG)* :
  Ha a játék döntetlennel végződik, megjelenik ez a rajz. (Egy piros téglalap, körülötte vastag kék vonalakkal, közepén felirattal, hogy döntetlen.)

GEP_STRAT:

*geprak(int *strat, Tabla *tabla)* :
void típusú függvény, így nincs visszatérési értéke. Ebben van szétválasztva, hogy a 3 különféle nehézségi szinten melyik másik függvényt hívja meg. A könnyű szinten az epito függvényt, nehéz szinten az akadalyozot, közepesen pedig 50% eséllyel az egyiket vagy a másikat.

A gép játékstratégiáját a geprak függvény irányítja, ami az alábbi további függvényekből épül fel.

A geprak függvényben használt függvények

Az alább felsoroltak közül mind void típus

- *vizszintes/fuggoleges/ek/eny_tamad (Tabla * tabla, ListaElem **tamado_mezok, int *szam)* (ez 4 függvény):
  A tabla segítségével tudjuk, hogy milyen magas és milyen széles az aktuális pályánk, így tudja a függvény, hogy meddig kell vizsgáljon a for ciklussal, amivel végigmegy a mezőkön, amik állapota szintén a tabla-val van megadva, hogy melyikben milyen bábu van, vagy üres-e. Számnyi darab szomszédos mezőt vizsgálunk (mindegyik függvény egy-egy irányban), hogy ha szam-1 darab X van benne, akkor, ha az a maradék egy üres, akkor az kerüljön a tamado_mezok listára, mint egy olyan mező amire potenciálisan bábut rakhat, ha támado/építő stratégiát alkalmazunk. Ezt úgy tudjuk elképzelni, mintha mindig számnyi méretű ablakot csúsztatgatnánk végig a pályán és abban számolnánk meg az X-eket.
- *vizszintes/fuggoleges/ek/eny_vedekez (Tabla *tabla, ListaElem **veszelyes_mezo)* (ez is 4 függvény):
  Ezek is hasonlóak az előző 4-hez, csak itt nincs a paraméterlistában a szam, ehelyett mindig 4 darab szomszédos mezőt vizsgál egy vonalban. További különbség, hogy itt nem az X-ek, hanem a körök számát figyeli, és ha ezekből 3 van és a 4. üres, akkor a veszelyes_mezo listára kerül fel ez a 4. (üres) mező, azaz azok a mezők, amik erre a listára kerülnek, potenciálisan olyanok, amikre bábut rakhat a gép, ha védekező/akadályozó stratégiát alkalmazunk.
- Az *epito_tamado (Tabla * tabla, ListaElem **tamado_mezok,int *szam)* és a *ved_akadalyoz (Tabla *tabla, ListaElem **veszelyes_mezo)* függvények az előző két alpontban leírt függvényeket fogja 2 csokorba (stratégiának megfelelően), azért, hogy könnyebb legyen meghívni őket egy másik függvénybe.
- *epito (Tabla *tabla)*:
  A tabla paraméterre azért van szükség (gyakorlatilag mindenhol, ahol használom ez a szerepe), mert ebben a struktúrában lévő szélesség, magasság és mezők állapotának segítségével tudunk csak végigmenni a pályán minden mezőt megvizsgálva.
  Ez a függvény először az epito_tamado függvényt hívja meg egy while ciklusban először szam=5-re, ami addig egy, amíg a tamado_mezok lista NULL-al egyenlő. Minden alkalommal amikor a ciklus lefut, az epito_tamado függvény hívása után a számot csökkentjük 1-el. Ha a tamado_mezok üres, akkor random lerak egy X-et, egyébként a lista elemei közül választ ki random egyet, és oda rakja le a bábuját. A while ciklus leállási feltétele biztosítja, hogy a gép mindig a neki legkedvezőbb helyekre tudja lerakni a bábuját. A függvény elején létre kellett hozni a tamado_mezok

listát, amit a végén fel is szabadítunk, tehát ez a lista minden báburakásnál létrejön és fel is szabadul.

- *akadalyozo (Tabla *tabla)*:
  Ez meghívja a ved_akadalyoz, és az epito_tamado függvényt is (ez utóbbit szam=5-re), ezért itt a veszelyes_mezo és a tamado_mezok listát is létre kell hozni. Ha a veszelyes_mezok != NULL, akkor ha a tamado_mezok == NULL, akkor random lerak egyet a veszelyes_mezo elemei közül, ha viszont a tamado_mezok != NULL, akkor a tamado_mezok elemei közül rak le egyet random. (Ezzel biztosítjuk, hogy ha a gép egy lépésre van a győzelemtől, akkor ne a játékost próbálja meg megakadályozni, hanem nyerjen) Egyébként (ha a veszelyes_mezo lista üres, meghívja az epito függvényt. A végén mindkét listát felszabadítjuk.

Ebben a modulban vannak a listakezelő függvények is, mivel ezek szorosan kapcsolódnak a gép stratégiájához. Három ilyen listákhoz szükséges függvényt használok:
egyet, ami a lista elejére beszúr egy elemet (ListaElem * típusú),
egy felszabadító függvényt, ami felszabadítja a paraméterként megadott lista dinamikusan lefoglalt területét (void típusú)
és egyet, ami a lista hosszát mondja meg (ez ott hasznos, amikor az egyik lista elemei közül akarok random kiválasztani egyet; int típusú).

KINYER:

Minden itt szereplő függvény void típusú.
A nyert_e függvény fogja egybe a többi ebben a modulban lévő függvényt.

- *gyoze_vsz(Tabla *tabla, bool *kor_nyert, bool *x_nyert, bool *babutrak)*:
  Az epito_tamado és a ved_akadalyoz függvényekben lévő alfüggvényekhez hasonlóan működik ez is, csak itt 5 db mezőt vizsgálunk egy vonalban. Egy for ciklusban nézzük, hogy egy ilyen ötös „ablakban” hány darab kör és X van. Ha odb==5 (tehát van 5db kör egy vonalban) akkor a kor_nyertr változó legyen true, ha pedig xdb==5 (tehát X-ekből gyűlt össze egy vonalban 5 darab egymás mellett), akkor az x_nyert legyen true. Mindkét esetben a babutrak pedig false lesz, hogy ne tudjunk bábut elhelyezni a pályán ezután. Ha nem győzött senki, nem történik semmi.
  (A gyoze_fg, gyoze_eny és gyoze_ek függvények ugyanezt csinálják csak más irányokat vizsgálnak.)
- *nyert_e(Tabla *tabla, bool *kor_nyert, bool *x_nyert, bool *babutrak, bool *dontetlen)* :
  meghívja a négy irányra a vizsgálófüggvényeket, amik eldöntik, hogy nyert-e valaki. Ha senki sem nyert és nincs üres mező a pályán, a dontetlen nevű bool változó legyen true.

MENTES:

Itt vannak a játék állásának és beállításainak fájlba mentéséhez és fájlból beolvasásához szükséges függvények leírásai.

- *jatek_ment (char const *fajlnev,Tabla *tabla, int strat, int kijon)* :
  Mivel ahhoz, hogy majd vissza tudjuk tölteni a játékállást, a menüben megváltoztatott

beállításokat (kivétel az, hogy ki kezd) kell elmentenünk, ezért vannak ezek paraméterként átadva. A kijon paraméter pedig azért kell, mert addig lehet bábut lerakni, a pályára kattintani, amíg van üres mező. És mivel minden lépéskor a kijön 1-el nő, így addig van üres mező, amíg a kijon kisebb mint a mezők száma (a kijon 0-tól indul).
Ahhoz, hogy mentsünk, meg kell nyitni egy fájlt. Ha ez nem sikerül, hibaüzenetet kapunk. Egyébként egyesével átírjuk a kellő adatokat a fájlba. A pálya állásának elmentéséhez for ciklusokkal végignézzük az összes mezőt. Ha üres akkor 0-t, ha kör van benne 1-es, ha pedig X, akkor 2-es számjegy íródik be a fájlba (az enum típus miatt). A függvény végén be is kell zárni a fájlt.
Void típusú.

- *jatek_betolt(char const *fajlnev, int *szels, int *mags,int *strat,int *kijon)* :
  Beolvasni ugyanazokat az adatokat kell mint, amiket elmentettünk, a paraméterlistából viszont itt hiányzik kétdimenziós mezo tömb, mert ennek a függvényen belül kell memóriát foglalni (mintha csak most állítottuk volna be a pálya méretét), amit meg tudunk tenni a szels (szélesség) és a mags (magasság) ismeretében. Itt is az első lépés a fájlt megnyitni, ha ez nem sikerül, hibaüzenetet kapunk. Egyesével beolvassuk a magasságot, szélességet, stratégiát és a kijon értékét. Az uj mezők lefoglalása után pedig for ciklusokkal végigmegyünk az új tömbön és az elmentett helyekre lerakja a megfelelő bábukat. A beolvasás végeztével be kell zárni a fájlt.
  A függvény visszatérési értéke az uj kétdimenziós tömb. (A függvény RubrikaAdatok** típusú.)

MAIN:

A main.c-ben fűzöm össze a modulokat, itt lehet végigkövetni a program működését.
Egy-egy int const globális változóban tárolom a megjelenítő ablak méretét.
Ide került még az SDL kezeléséhez szükséges (hibaüzeneteket is küld, ha nem működik valami az SDL-el kapcsolatban) függvényen kívül még a:

- *gombfeltoltes(Gombtarolas *gombtomb,int x, int y, int szelesseg, int magassag, int azonosito, char *felirat)* :
  Ennek segítségével adhatjuk meg egyszerűen majd a gombok tulajdonságait, amik ennek a függvénynek a paraméterei. A gombtomb (Gombtarolas típusú) azonosito-adik elemének különféle tulajdonságait inicializáljuk itt.
  Void típusú.
- *oldalhossz (int szels, int mags, int teljszel, int teljmag)* :
  Double típusú függvény.
  A teljszel és a teljmag változók az ablaknak azt a nagyságú részét jelzik, amin ki akarjuk majd rajzolni a játékpályát. Ezeket úgy kaptam, hogy az ablak teljes méretéből még levontam a gomboknak szánt területet a szélességből és egy kis margót minden oldalról, hogy semmiképp se legyen a pálya széle egyenlő az ablak szélével.
  A paraméterek ismeretében kiszámolja a függvény, hogy mekkora legyen egy mező oldala majd, ha azt szeretnénk, hogy négyzet alakúak legyenek, semmiképp ne lógjanak ki a képernyőről (ami egyben azt is jelenti, hogy ezzel együtt a lehető

legnagyobbak legyenek). Visszatérési értéke a teljszel/szels, ha a telszel/szels kisebb mint a teljmag/mags. Ellenkező esetben a visszatérési érték teljmag/mags.

A main függvény két fő részre bontható: az első a kirajzolás, a második a vezérlés.

Kirajzolás

Létrehozzuk a megjelenítő ablakot (amiről minden alkalommal, amikor másik menüre vagy a játéktérre váltunk töröljük az addig rajta levő dolgokat). Hogy éppen mit kell megjeleníteni logikai változók segítségével dönti el a program. Először a menük majd a játékpályához tartozó elemek rajzolódnak ki. Ha valaki nyert (vagy éppen döntetlen lett) az ezeket jelző függvények hívódnak meg. Ilyenkor egy hosszúkás téglalapba kiírja, hogy ki nyert.

Vezérlés

Billentyűkkel vezérlés

A jobbra, balra, fel és le nyilakat, valamint az entert használja a program.
Ezeket a menük vezérlésére használom: a nyilakat a menüpontok közti váltásra, az enterrel pedig, ha az adott menüben a beállításokat menteni akarom, azaz tovább akarok lépni a következő ablakképre. Itt a kirajzolás függvényeket irányító bool-okat állítom mindig a megfelelő értékre. Itt tudtam megadni (feltételként) azt is, hogy mekkora legyen a maximális mérete a pályának. Mikor a játékpályát jelenítem meg (az ehhez vezető enter lenyomásával egy időben) meghívom az oldalhossz függvényt és definiálom a tabla.mezo-t ami egy két dimenziós tömb. Itt már mivel ismerem az oldalhosszt, ki tudom számolni az egyes mezők sarkainak koordinátáit, amik majd az egérrel vezérlésnél kellenek ahhoz, hogy a bábuka elhelyezhessük.
Ha a korábbi játék betöltését válasszuk az első menüben, akkor az enterrel egyszerre a jatek_betolt függvény hívódik meg és újra definiáljuk a mezők koordinátáit az elmentett adatoknak megfelelően.

Egérrel vezérlés

Itt három féle dolog történhet, ha az egér bal gombját lenyomjuk attól függően, hogy az ablak mely koordinátáin történt a kattintás.

1. Ha a pályán kattintok, akkor az adott koordinátahatároknak megfelelő mezőn egy bábu helyeződik el annak függvényében, hogy éppen ki jön. Egy kattintással mindig két bábu helyeződik el: egy kör és egy X, de minden bábu lerakása után megvizsgáljuk, hogy nyert-e valaki (a nyert_e függvénnyel) és ha nem, akkor a kijon változót növeljük, így biztosítva, hogy ne kerülhessen a pályára kétszer egymás után ugyanolyan bábu. Valamint itt hívódik meg a geprak függvény is, amikor a gép jön. (Ha a gép kezd, először az enter lenyomásakor a pálya kirajzolódásával egyszerre hívódik meg először a gep_rak.)
2. Ha a játék mentésének gombjára kattintok, akkor a jatek_ment függvény hívódik meg.
3. Ha pedig az új játék kezdésének gombjára, akkor a tábla mezőit felszabadítom, hogy ne legyen memóriaszivárgás és az értékeket úgy állítom be, hogy a fomenu függvénnyel úgy tudjak kirajzolni, mintha akkor nyitottam volna meg a programot.

Egyéb vezérlés: a programból kilépni az ablak feletti sarokban lévő X-el lehet.

Program vége:

A dinamikusan foglalt memóriaterületeket (tabla.mezo[ ] és tabla.mezo) mindenképpen felszabadítom amikor kilépek a programból.

Egyéb:

A program futtatásához szükség van az InfoC-ről letölthető SDL könyvtárra.

A programban szerepel néhány kódrészlet, amiket az InfoC honlapról vettem át.
(https://infoc.eet.bme.hu/)