Tóth Gergely György

Programozói dokumentáció

Rizikó

Környezet

Felhasznált külső könyvtár: SDL2

Szükséges fájlok:

adatkezeles.c és .h, jatekmenet.c és .h, kijelzo.c és .h, strukturak.h, jatekos.bin, szomszedsag.bin, prov.bin, LiberationSerif-Regular.ttf, terkep.jgp

Windows 10 operációs rendszeren futtatva.

Felépítés

A .c és .h fájlok a nevükhöz releváns függvényeket tartalmazó modulok.

A terkep.jpg egy olyan képfájl, aminek a bal oldala a játék menüje és kijelzője, rajta egyértelműen látszanak a gombok. Ezen gombok és szöveg kiírására alkalmas felületek képbeli koordinátái a main.c-ben találhatók koordináta tömbökben, a fő játék ciklus ezen koordináták alapján dönti el, hogy a kattintás aktiválja-e a gombhoz tartozó funkciót. A jobb oldalon egy térkép látható, ami csakis esztétikai szempontot lát el, fehéren hagyva a jobb oldalt a provinciák színes köre akkor is ki lesznek rajzolva.

A .ttf fájl az infoc-ről letöltött betűkészlet, az olvasható kiírást szolgáltatja.

A jatekos.bin fájl egy konverter programmal írt fájl, tartalmazza a benne szereplő Játékos struktúrájú adatok számát, és magukat az adatok ebben a sorrendben. A prov.bin a provinciákkal ugyan ez. Ezeket a fájlokat cserélve lehet a játékosokat, provinciákat változtatni, akár számban, akár valamilyen tulajdonságukban : pl játékos színe, bizonyos provincia kezdő katonái.

A szomszedsag.bin tartalmazz a a provinciák szomszédsági mátrixát, először a mérete aztán maga a mátrix, ami a mozgás során a „választott” provincia szomszédait hivatott azonosítani.

Tóth Gergely György

Adatszerkezetek

struct Koord { int x; int y; }
Koord;
Koordinátát tárol, a 2 dimenziós grafikus megjelenítés ezt megköveteli

struct Jatekos { char nev[30+1]; int id; int osszkatona; int kepezheto; int erosites, int r,g,b; struct Jatekos* kov; }
Jatekos;
A játékos struktúrája, tárolja a játékos
nevét (pl Ausztria, Oszmán birodalom), ami fix hosszú lehet, hogy férjen is ki a kijelzőre
egy *id*-t amivel azonosítani lehet az adott játékost (ez alapján találja meg a fa építő függvénycsomag az adott provincia birtokosát első felépítéskor),
az *összes katonájának* számát amit minden körben újraszámol hogy up-to-date legyen,
a játékos tartalékos katonáik akiket ki*képezhet*,
az *erősítés* a birtoklott provinciáinak száma, ennyivel nőnek a tartalékosai körönként,
három int értéket melyekből rgb kód képezhető (SDL_Color típussal hiba volt),
mutatót a következő játékos struktúrájára.

Egy láncolt listába történik a beolvasásuk, aminek a végének a következő pointerét az első elemhez rendeljük, és ezzel egy körbe zárt listát kapunk. Ez a kör alkotja a játék fix sorrendben történő játékosváltásait.

struct Prov { int id; char nev[20+1]; int katona; int kepzes; Koord kepzeskoord; Koord katonakoord; Koord pkoord; struct Prov *bal, *jobb; Jatekos* birtokos; }
Prov;
A provinciák struktúrája, fa csomópont.
id: Ez a fa felépítésének kulcsa, ez alapján rendezi a fát
nev: A provincia neve, max mérete van, hogy kiférjen, ez jelenik meg a kijelzőn
katona: az adott provincián állomásozó katonák száma
kepzes: általában 0, ha nem 0 akkor ott a játékos új katonákat képez ki, ez az érték hozzáadódik a katona-hoz majd nullázódik a kör végén
kepzeskoord: a grafikus megjelenítéshez kell, a képzett katonák számát a provincia körébe írja ki
katonakoord: előzővel analóg
pkoord: A provincia színes körének középpontja, ez alapján rajzolja azt ki
bal és *jobb* mutatók: A provinciákat egy önkiegyenlítő bináris fában tárolja a program, ezek ehhez kellenek
birtokos: a terület birtokosa, az ő színében jelenik meg a kör, a mozgás/harc két provincia birtokosainak egyenlőségén/különbségén múlik

struct Osszadat { Prov* gyok; Jatekos* aktiv; SDL_Renderer* renderer; TTF_Font* font; Koord* kezdo; Koord* vegzo; int szomszedsag; Prov* valasztott; Prov* hova; int lepesek; int mozgas; int mennyit; }**
Osszadat;
Ennek a struktúrának az az értelme, hogy a működési függvények konszolidált számú paraméterűek legyenek, és így minden fő információ egy struktúrában tárolva bármikor elérhetővé teszi azokat.
gyok: A provincia fa gyökere
aktiv: A jelenleg lépő játékos, egyben a kör lista számunkra megjegyzett pontja
renderer: Az SDL megjelenítője
font: A kiírásoknál használt betűtípus
kezdo: Ez egy koordináta tömb a kijelző dinamikus részeihez: a szöveg, amit megjelenít a releváns játékosról és választott provinciákról. Ez azon területek bal felső koordinátája.
vegzo: kezdo-vel analog, a jobb alsó koordináta. A kiíráshoz először „felül kell festeni" az előzőleg kiírt neveket, számokat, hogy az újak olvashatóak legyenek. Ezt teszi lehetővé ez a koordináta
szomszedsag: A provinciák szomszédsági mátrixa, egy 2 dimenziós tömb ami meghatározza melyik provinciáról melyikre lehet katonát mozgatni
valasztott: Ez a játékos által elsőnek megkattintott provincia, ami egyben a sajátja is
hova: Ez az előbb megkattintott provincia egyik szomszédja, birtokostól függetlenül
lepesek: Az adott körben lépő játékosnak még hány mozgatása van hátra.
mozgas: Egy 0-1 értékű változó ami a mozgás gomb állapotát mutatja
mennyit: A „választott" és „hova" provinciák között mozgó katonák száma

Tóth Gergely György

Függvények

Adatkezelés

Jatekos* Jatekos_beolvas();
Beolvassa a játékosok adatait tartalmazó fájlt, lefoglal nekik területet, beállítja körkörös láncolt listára. Visszatér a lista egy elemére mutató mutatóval.

void Jatekos_free(Jatekos* eleje);
Felszabadítja a körkörös listát.

int Szomszedsag_beolvas();**
Beolvassa a szomszédsági mátrixot, lefoglal neki területet és visszatér a lefoglalt mátrixxal. Infoc alapján

void Szomszedsag_free(int tomb);**
Felszabadítja a szomszédsági mátrixot. Infoc alapján

Jatekos* birtokoskeres(Jatekos* jatekos,int id);
Megkeresi azt a játékost, akinek az id-je egyezik a paraméter id-vel, majd visszatér egy erre a játékosra mutató mutatóval

Prov* Prov_fa_beolvas(Jatekos* jatekos);
Fájlból beolvassa a provincia fa elemeit, majd átadja őket a beszúró függvénynek. Visszatér a fa gyökerével

Prov* beszur(Prov* gyok,Prov* uj);
A fő fa építő függvény, beszúrja a provincia fa elemeit egyesével, majd a részfák súlya (magassága) alapján elforgatja a fa csúcspontjait, így az mindig kiegyensúlyozott marad (már amennyire az lehetséges az elemszám miatt)

Prov* bal_forgat(Prov* gyok);
A jobb részfa bal részfáját teszi a gyökér jobb részfájává, a jobb részfa bal részfáját pedig a gyökérré, majd a jobb részfa gyökerével tér vissza, effektíve balra forgatva a gyökerénél a fát

Prov* jobb_forgat(Prov* gyok);
A bal_forgat tükörképe

void Prov_fa_free(Prov* gyok);
Standard fa felszabadító függvény. Infoc alapján

int Balancefactor(Prov* gyok);
A jobb és bal részfákat hasonlítja össze magasság alapján, -1 0 1 értéknél még elfogadható a fa, ennél nagyobb értéknél forgatni kell

int height(Prov* gyok);
A fa magasságát határozza meg, ezzel a számmal tér vissza

Prov* Prov_keres(Prov* gyok, int id);
A fában keres megadott id-jű elemet amivel vissza is tér

void Jatekos_kieses (Osszadat* adat);
A játékosok listájából kifűzi és felszabadítja azt a játékost akinek nem maradt provinciája (az Erősítése a provinciái számától függ, ez használható jelzésként erre). Ha csak egy maradt akkor kiírja hogy ő győzött.

void Adat_inicializal(Osszadat* adat);
Beolvassa a kezdeti értékeket a játékos, fa, szomszédsági mátrixba és egyéb alap értékeket beállít

void Adat_free(Osszadat* adat);
A felszabadító függvényeket meghívja a lefoglalt memóriaterületekre

void Uj_jatek(Osszadat* adat,SDL_Event ev);
Felszabadítja a régi lefoglalt adatokat, újrainicializálja őket, kirajzolj mindent, és felülfesti a győzelmi üzenet helyét

Játékmenet

int Osszkat_szamol(int id,Prov* gyok);

Megszámolja az adott játékos id-hez kapcsolódó provinciák katonáit

int Erosites_szamol(Prov* gyok, int id);

Megszámolja hány provinciája van az adott id-jű játékosnak, ennyivel nő majd a képezhető katonáinak száma

void Kepzes_hozzaad (Osszadat* adat,Prov* gyok);
A kör végén a képzett katonák valódi katonák lesznek, a képzés lenullázódik, a katonák száma pedig nő

int Dobas_vedo(int katonak);

A védő dobásai harcnál, a katonák száma alapján többsször is dobhat, a legnagyobbat adja vissza

int Dobas_tamado(int katonak);
Analóg a védő dobással

void Prov_valaszt (Osszadat* adat,SDL_Event ev,Prov* gyok);
Ha a kattintás koordinátái a provincia színes körébe esnek és a provincia tulajdonosa van soron, akkor azt a provinciát kiválasztotta, ezt kiírja a kijelzőbe és beállítja az Összesített adatstruktúrában

void hova_valaszt (Osszadat* adat,SDL_Event ev,Prov* gyok);
Ha már van kiválasztva provincia akkor annak szomszédsági mátrixát megnézi beleesik-e az adott kattintás koordinátához tartozó provincia. Beállítja a hova-t és kiírja

void Valaszt_master (Osszadat* adat,SDL_Event ev);
Ez dönti el teljesül-e a hova_valaszt feltétele, ha nem akkor simán Prov_választ

void Kepzes_plusz(Osszadat* adat,SDL_Event ev);
Növeli a választott provincién a képzést eggyel, csökkenti a játékos képezhető katonáit eggyel

void Kepzes_minusz(Osszadat* adat,SDL_Event ev);
Analóg az előzővel

void Mozgas_gomb(Osszadat* adat,SDL_Event ev);
Átállítja a mozgás gombot, ami a hova_valaszt egyik feltétele, gyakorlatilag bit invertálást végez mert bool-t csak ezen az egy helyen használnék de nem akartam

void Mennyit_plusz(Osszadat* adat,SDL_Event ev);
Növeli a két provincia között mozgatandó katonák számát. Max értéke a választott provincia katonaszáma

void Mennyit_minusz(Osszadat* adat,SDL_Event ev);
Analóg az előzővel

void Korvege(Osszadat* adat,SDL_Event ev);
Elvégzi a képzéseket, lépteti a játékost

void Mozgas_confirm(Osszadat* adat,SDL_Event ev);
Két terület között katonák mozgatása, a forrás provincián levonjuk a mozgatni kívánt katonák számát, a célprovincia ha baráti egyszerűen hozzáadjuk és csökkentjük a maradék lépések számát, ha ellenséges a terület akkor harc, ahol dobásokkal eldöntjük ki veszít katonát. Egyesével veszítik a harcoló felek a katonákat, ha a védők vesztettek a támadó lesz az új birtokos. A játékosok kiesését is ellenőrizni kell ilyenkor.

Kijelző

void Prov_kirajzol(Osszadat* adat,Prov* gyok);
Kirajzolja a provinciák színes körét, rá a katonák számát, és a képzés számát

void Katona_rajzol_rajzolo(Osszadat* adat,int x, int y, char* szoveg);
A megadott koordinátákra kiírja a megadott szöveget. Ez a provinciákba beleírt számokat jelenti

void Kijelzo_kiir(Osszadat* adat,char* szoveg, int hanydik);
A megadott sorszámú kijelző területre kiírja a megadott szöveget.

void Mozgas_jelzo_rajzol(Osszadat* adat);
A mozgást jelző piros és zöld „forgalmi lámpákat” rajzolja ki, a mozgás on/off állapotától függően

void Kijelzo_master (Osszadat* adat);
Minden kijelző elemet kiír kivéve a győzelmet, azt a kiesést figyelő függvény végzi