Love letter kártyajáték Programozói és Felhasználói dokumentáció

Felhasználói dokumentáció

A program célja

A love letter egy pörgős 2-től 4-személyes kártyajáték. A program célja az alapjáték virtuális környezetbe való átültetése volt.

Bemenetek

A játék során a játékosok a számokra hagyatkozva tudják bevinni a programba a döntéseiket.

Kimenetek

A program egészében a consol-ablakban, szöveges formátumban fut le.
A könnyebb átláthatóságot segítik különböző karakterekből kirajzolt vonalak, rubrikák.

Irányítás

A játék mindig szöveges formában kínál fel opciókat, amikhez számokat (pld.: 1-igen, 0-nem), társít.
A játékosnak a választásuk véglegesítését mindig enterrel kell megtenniük, ügyelve arra, hogy a szabályoknak megfelelően játszanak, és ne hibás karaktereket vigyenek be, ugyanis a játék jelenlegi verziója nem képes minden téves bevitelt a helyén kezelni.
Előfordul a programban olyan szituáció is, ahol a játékosokkártyák titokban maradásának érdekében üres képernyőt láthatunk. Ilyenkor bármilyen karakter bevitelével tovább tudunk haladni a programban.

Programozói dokumentáció

Projekt felépítése, fájlok tartalm

A projekt alaphelyzetében 8 fájlból áll. Ez 4 forráskódot és 4 header fájlt jelent. Ezen felül saját mappájába egy futási opcióként egy txt fájl generálódik, de ez nem szükséges a projekt indításához, pusztán a játékállás tárolására szolgál (neve: allas.txt).

A main.c-ben található a fő program, funkciói: menü rendszer kezelése, játékoskör lebonyolítása, a játék állas mentese es előhívása, győztes kihirdetésé.

A paklileterehoz.c és a hozzá tartozó azonos nevű header egy 16 lapból álló pakli legenerálásáért, és a lapok első kiosztásáért felel.

A kartyaleirasok.c és azonos nevű .h Alapvetően szövegeket tárol. Itt talalható a 8 különböző leírása és a játékszabály is. Mindezek kiírása is innen fut, emellett itt kapott helyet pár consolt kezelő függvény is.

Az interakciok.c a main utáni legnagyobb forráskód. A hozzátartozó headerrel együtt azért felelnek, hogy a mainben kiválasztott és kijátszott lapok helyesen interaktáljanak egymással. Ide tartozik tehát a kártyák eldobása, egymáshoz hasonlítása, kicserélése, megtippelése, és még hasonló funkciók.

A debugmalloc.h-t memóriaszivárgás ellenőrzésére használom.

Fejlesztői környezet

A program c nyelvű. Random számok és logikai változók kezeléséhez meghívtam a stdlib.h, time.h, stdbool.h könyvtárakat. Ezen és a stdio.h-n felül nem igényel semmilyen más könyvtárat.

Adatzerkezetek

A játék egy program elején dinamikusan lefoglalt kétdimenziós, egész változókat tároló tömbben kezeli a kártyapaklit. A félkész leadásnál még egy struktúra alapú dinamikus adattárolót használtam, viszont az átláthatósága és bővítési lehetőségei miatt váltottam.

A kartya nevű tömb első dimenziója tartalmazza 0-15 ig a kártyák indexeit. Ez minden kártyánál egyedi, ennek rengeteg előnye van, mind léptetéses ciklusoknál, pl kereséseknél, de random kártyaosztásnál is.

A második dimenzió kettő értéket vehet fel: 0; így eltárolva a specifikus kártya értékét, és 1; ebben az esetben pedig a kártya viszonyát, hol létét jelöli meg a felvett érték. Ennek működését különböző kódok használatával lehet elérni. Például a 0.indexű,1-es azaz helyet jelölő tárolt érték lehet 0. Ebben az esetben a kartya[0][1] még a kiosztatlan pakliban tartózkodik.

> Az általam használt jelölések sorra a következők: 0: pakliban, 1-4: 1-4.játékosnál, 10,20,30,40: 1.,2.,3.,4. játékos előtt, 88: kivéve a pakliból, 99: kiesett/dobott lapok.

Ezzel a tárolási módszerrel végtelenül egyszerű megmondani, hogy még van-e szabad kártya a pakliba; elég összeszorozni a helyszíneket, 0-s eredménynél következtethetünk arra, hogy van még legalább egy kártyánk, ami a pakliban van.
Így sokszor intervallumokra tudok szűrni, ha csak egy bizonyos tulajdonság érdekel, pld: játékosoknál lévő kártyák->ertek>0 && ertek<=4;
Ezek, és ehhez hasonló praktikus és átlátható előnyök miatt választottam ezt a megoldást.

A függvények

Ezeket a függvényeket tartom a legfontosabbnak kiemelni, de rengeteg széljegyzet van írva a forráskódokba is, így nem jelenthet problémát a forráskód átlátása, lényegének megértése.

A main függvény

Fontosabb kiemelendő szerepei: dinamikus kétdimenziós tömb mallocos foglalása és feloldása. menürendszer kezelése; ennél főleg a „goto” függvényre hagyatkoztam, bár fejtörést okozott, hogy egy-egy menüelem hová is ugrasszon. Itt különösen kell vigyázni nehogy átugorjunk egy fontos deklarációt. Itt vannak kezelve a fájlok is, de mivel a program magának hozza létre a játék állapotmentését, így félelem nélkül hagyatkozhattam a beolvasásnál az fscanf függvényekre.
Itt fut le minden játékos köre is, de valójában ez csak egy vázprogram, a komolyabb számításokat és

rendezéseket mind külön függvénybe raktam. A következő játékos kijelölését egy if else logika végzi, ami átugorja az esetlegesen már kiesett köztes játékosokat. Vegul pedig itt talalhato a jatek lezarasa is, amit egy for ciklussal megoldott maximum számítás.

Az interakciók függvény

Ez a függvény felel a kártyák kijátszásáért, és mivel minden kártyatípusnak saját szabályai vannak, ez valójában 8 elkülönülő if eset 8 külön megoldandó problémával, bár azért vannak hasonlóságok a kártyák közt.
Ilyen például az, hogy a Handmaid kártya megvéd mindenféle interakciótól egy egész körig, így ja egy olyan játékost választanánk aki előtt esetleg még érvényesen lebeg ez a lap, a játék figyelmeztet erre, erre szolgál a handmaid függvény ami igaz hamis értéket ad vissza az alapján, hogy a neki címzett emberünknek van e ilyen védelme érvényben.
A másik gyakran felbukkanó feladat a kijátszott kártya eldobása, ezt mind a 8 esetben implementáltam a kartya[i][1] = 99; sorral.

Paklikever

Maradékos osztással egy 0 és 15 kozotti számot kapunk, ami tokéletes hogy egy random indexű kártyához hozzárendeljünk egy játékost. Próbálkozzunk addig amíg nem találunk szabad helyet, és ismételjük meg ezt a játékosszámszor. Ennyi a lényege a pakli első kiosztásának.

Pakligeneral

Létrehozza a 0-15 ös indexű 1-8 ig terjedő értékű kártyákat, és mindet belerakja a pakliba(0).
Ennél a résznél is remekül jött hogy számokkal dolgoztam pl nevek helyett, így páros és páratlan esetekre osztva a dolgot, gyorsan legenerálható egy csomó kártyapár egy bizonyos intervallumon belül.

Ujkartya

Először a már az adatszerkezetek fejezetnél említett példához hasonlóan megbizonyosodik a függvény, hogy van még mit kiosztani, majd a pakligeneralból megismert módon egy random lapot kap a kiválasztott játékos.