Programozói Dokumentáció

1. Általános leírás

A program egy parancsterminálban futó pasziánszmegvalósítás. A kártyákat színezett ASCII és UTF karakterekkel jeleníti meg. A játékszabályok a klasszikus, úgy nevezett „Klondike” pasziánsz szabályait követik (https://en.wikipedia.org/wiki/Klondike_(solitaire)#Rules), egyszerre egy kártya felfordításával és bármennyi újra keveréssel. A felhasználó a billentyűzeten található gombok és az egér segítségével játszhatja a játékot. A játék megkezdése előtt szükséges egy felhasználónevet megadni, ezen név alatt tartja számolja program a nyerések és megkezdett játékok számát, ebből számít pontszámot, melyet egy dicsőségtáblán meg is jelenít.

A program használatának a részletei a felhasználó dokumentációban találhatóak meg.

2. Futtatás és fordítás

A program a „**gcc -Wall -Werror Solitaire.c cards.c gamestate.c input.c leaderboard.c logging.c rendering.c -o Solitaire.exe -lncurses -DNCURSES_STATIC**” paranccsal fordítható le.

Fontos a hogy a projecthez használt ncurses könyvtár a MINGW-w64 csomagot használja, ennek a telepítésének a leírása az alábbi linken érhető el: https://stackoverflow.com/a/30071634

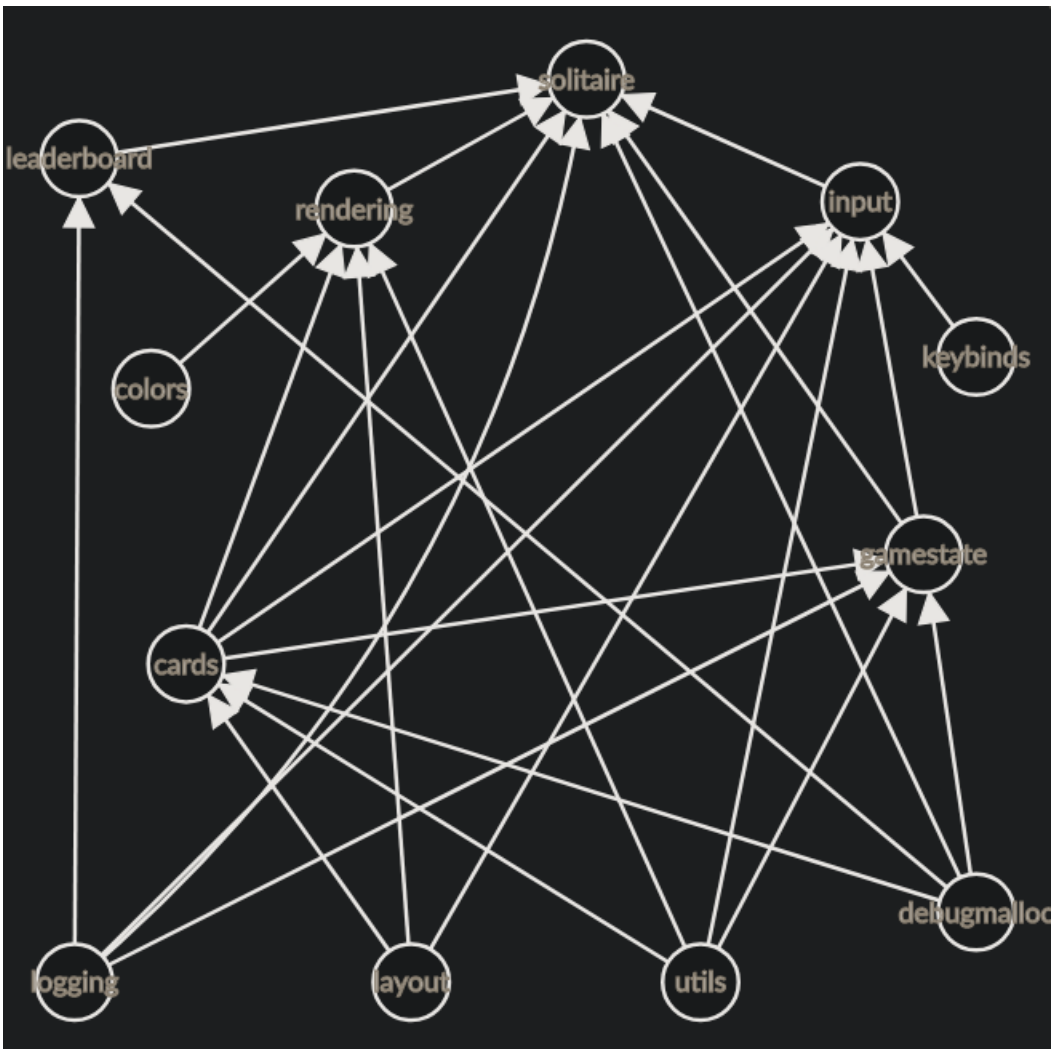
Ha MINGW-s gcc-vel fordítunk és szeretnénk azt, hogy a program, olyan számítógépeken is fusson, ami nem tartalmazza a libwinpthread-1.dll file akkor az alábbi paranccsal érdemes fordítani:

„**gcc -Wall -Werror -static -static-libgcc -static-libstdc++ Solitaire.c cards.c gamestate.c input.c leaderboard.c logging.c rendering.c -o Solitaire.exe -lncurses -DNCURSES_STATIC**”

A könyvtár útvonala a MINGW install esetében az ncurses/ncurses.h lesz, ahol a program a könyvtárat használja ezen az útvonalon éri el. Ez fontos mivel a program más install esetében nem biztos, hogy képes lesz megtalálni az ncurses könyvtárat.

3. Modulok

A programban használt modulokat az alábbi gráfon lehet megjeleníteni, a nyilak jelzik, hogy melyik modul include-ol mely másik modul(oka)t. Az egyes modulokat és az azokban használt függvényeket lejjebb részletesebben is kifejtem.



Utils.h

A gyakran, több modul által használt függvényeket tárolja és makrókat tárolja. Gyakorlatban egy MIN és egy MAX makrót tartalmaz. Lényegében evidens, hogy mire jók és sok más modul használja őket.

Layout.h

A kártyák méretét, illetve a terminálhoz és az egymáshoz viszonyított elhelyezkedését meghatározó konstansokat tárolja

Cards.c és Cards.h

A kártyákhoz kapcsolódó függvényeket és adattípusokat kezeli.

struct Card: Egy darab kártya adatait tárolja

int suit: A kártya színe: (1-4, avagy káró ♠ (fekete), szív ♥ (piros), treff ♣ (fekete), kőr ♦ (piros))

int rank: A kártya rangja (1-13, avagy Asz, 2, 3, 4, 5, 6, 7, 8, 9, 10, Jumbo, Qdáma, Király)

char CardSuit(Card card)*:

A kártya színéhez tartozó UTF karaktert adja vissza

char CardRank(Card card):

A kártya rangjához tartozó karaktert adja vissza (A, 2, 3, 4, 5, 6, 7, 8, 0, J, Q, K)

int CardColor(Card card):

A kártya szövegéhez tartozó színhez tartozó indexet adja vissza

struct CardPile: Egy az asztalon lévő kártyakupac adatait tárolja

Card cards*: A benne lévő kártyák dinamikusan foglalt tömbje

int size: A kupacban lévő kártyák száma

int capacity: a dinamikusan foglalt tömbbe maximum beleférő elemek száma

int maxDisplay: a kupac felső n darab megjelenítendő kártyája

int uncovered: az n-ik(0-ás kezdetű indexelés) kártya alatti indexű kártyák le lesznek fordítva

CardPile CreateCardPile(): Létrehoz egy kupac kártyát alapértelmezett tulajdonságokkal

void ResizePile(CardPile pile, int newCapacity)*: Átméretezi a megadott kupacot newCapacity kapcitásúra és megtartja az eddigi elemeit, a méretet csak akkor változik, ha az új kapacitás kisebb, mint az eddigi méret

void AddCard(CardPile pile, Card card)*: A kupac tetejére ráteszi a megadott kártyát és amennyiben szükséges extra helyet foglal(mindig dupláz)

void RemoveTopCard(CardPile pile)*: A pakli tetejéről törli a felső kártyát és amennyiben kapacitás több mint fele kihasználatlan megfelezi a pakli méretét.

void ShuffleCards(Card cards, int length)*: Véletlenszerűen megkever egy length hosszúságú kártyalistát

int DisplayedCardCount(CardPile pile): Megadja hány darab kártyát kell kirajzolni egy megadott kupac esetében, figyelembe veszi a maxDisplay-t és azt, hogy az üres kupac esetében is ki kell rajzolni az üres helyet

struct Table: A rows*columns asztalon lévő kártya kupacok adatait tárolja és az asztal dimenzióit

*CardPile** *piles*: Kupacmátrix

int rows: Az asztal sorainak a száma

int columns: Az asztal oszlopainak a száma

int RowHeight(Table table, int row): Egy asztal row-adik sora esetében meghatározza, hogy a képernyőn hány sor szükséges a kiíratásához

struct CardPosition: Egy adott asztalon lévő kártyához tartozó pozíció

int row: Hányadik sorban lévő kupacban található a kártya

int column: Hányadik oszlopban lévő kupacban található a kártya

int inPile: A kupac hányadik kártyája

#define INVALID_POSITION: Egy olyan pozíció, ami a táblán nem létezhet, akkor használjuk ha szeretnénk jelezni, hogy egy pozíció nem létezik

CardPile GetPile(Table table, CardPosition position): visszaadja a pozícióhoz tartozó kupacot

Card GetCard(Table table, CardPosition position): visszaadja a pozícióhoz tartozó kártyát

Gamestate.c és Gamestate.h

A gamestate.c-től függ az összes játékszabály és tartalmazza a játék menetét irányító függvényeket

Table SetupTable: Létrehozza az asztalt és az azon lévő kupacokat és gondoskodik arról, hogy a kártyák meg legyenek keverve

void FreeTable: Felszabadítja az asztal és az azon lévő kupacok számára foglalt memóriát

CardPositon DefaultPosition(Table table): Visszaadja a pozíciót ahol a kurzor a játék elején kezd

bool CanSelect(Table table, CardPosition position): Az adott kártya és az alatti lévő kártyák kiválaszthatók e: jelen esetben a feltétel az, hogy különböző színek csökkenő sorrendben egymást kövessék és ne legyenek lefordított oldallal

static bool DrawCard(Table table): Kártyát húz a húzópakliból

static bool ResetDrawPile(Table table): Visszarakja a dobópakli kártyáit a húzópakliba

bool TryInteract(): Visszaadja, hogy egérakttintásra vagy gomb lenyomására interakcióba lehet e lépni az adott paklival, jelen esetben ilyen interakció a kártya húzása és ha a húzópakli üres a dobópakli visszarakása a húzópakliba, amennyiben az interakció lehetséges végrehajtja

bool CanMove(Table table, CardPosition from, CardPosition to): Visszaadja hogy from pozícióban lévő kártya átmozgatható-e a to pozíció tetejére

void MoveCard(Table table, CardPosition from, CardPosition to): Átmozgatja a from pozícióban lévő kártyát és fölötte lévő kártyákat a to pozícióra és felfordítja a from pozíció alatti kártyát ha az le van fordítva

bool HasWon(Table table): Megállapítja, hogy egy játékállás meg van-e nyerve: minden kártya a felső négy gyűjtőpakliban található

Keybinds.h

A különböző felhasználói billentyűhöz bemenetekhez tartozó billentyű kódokat tárolja. Minden gombhoz tartozik egy alternatív, második lehetőség is.

int ConvertInput(int input): Átalakítja az alternatív bemeneteket a normál variánsra

Input.c és Input.h

void InitInput(): Beállítja az ncurses megfelelő beállításait az egér és billentyűzet bemenet beolvasásához

enum InputResult: Mit csináljon a játék az a bemenet beolvasása után

NOTHING: A képernyőn semmi sem változott, nem kell újra rajzolni

RERENDER: A képernyőn valami változott, újra kell rajzolni

EXIT: Kilépés a játékból a főmenübe

RESTART_GAME: A játék újraindítása, új keveréssel

WON: Játék megnyerése, kilépés a főmenübe

static CardPosition GetClickPosition(int x, int y): Megállapítja, hogy a terminál x, y karakteréhet melyik kártya tartozik és visszaadja a pozícióját

s*tatic InputResult KeyMoveInput(Table table, CardPosition* cursorPosition, int input)*: A megadott input alapján mozgatja a kurzort, az olyan paklikat ahova nem lehet kártyát rakni átugorja, a cursorPositiont megfelelően módosítja

static InputResult InteractInput(Table table, CardPosition cursorPosition, CardPosition* selectedPosition)*: Szóköz vagy egérkattintásra megpróbál a kurzor pozíción lévő kártyával interakcióra lépni, ha nem sikerül és ha van kiválasztott kártya megpróbálja a kurzor pozícióra mozgatni, ha nincs vagy

nem lehet megpróbálja a kurzor pozíciót kiválasztani, a cursor, selectedPositiont és az asztalt megfelelően módosítja

static InputResult MouseInput(Table table, CardPosition cursorPosition, CardPosition* selectedPositon)*: A kurzort a kattintott kártya helyére állítja és megpróbál interakcióba lépni az adott kártyával

InputResult HandleInput(Table table, CardPosition cursorPosition, CardPosition* selectedPosition)*: Beolvas a billentyűzetről vagy egérről egy bemenetetet és annak megfelelően módosítja az asztalt és kurzort és a kiválasztott pozíciót

Colors.h

A játék során használt színek konfigurációját tároló header. Az ncurses a színeket párokban kezeli, minden color_pair tartalmaz egy elő és egy háttérszínt. A színpárok indexét a háttér és az előtér szín indexéből határozom meg. A 0 az alap karakter és 1-4-ig a színek a kártyaspecifikus színeket jelöli. A háttérszínt a tíz többszöröseivel határozom meg 10 a háttérhez, 20 az üres kupachoz, 30 az alap kártyaháttérhez, 40 a kártyán található kurzorhoz, 50 a kiválasztott kártyához tartozó szín. A kettő összegéből meghatározható melyik színpárt kell beállítani.

void CreateColorPairs(): Létrehozza a játék során használt színpárokat

Rendering.c és Rendering.h

void InitRenderer(): Beállítja az ncurses megfelelő beállításait a képernyőre íráshoz és színek kezeléséhez

void PrintInColor(char str, int cardBackground)*: Kiír a jelenlegi kurzor pozícióra egy & által kódolt színezésű sztringet. A & jelet követi két karakter,

melyek nem kerülnek kiíratásra hanem meghatározzák, hogy az őket követő karakterek és a hozzájuk tartozó milyen színnel jelenjenek meg. Az első karakter ’b’ értéke az alapértelmezett háttérszínt jelenti, a ’c’ értéke a paraméterként megadott kártya háttérszít. A második karakter ’t’ értéke az alapértelmezett karakter színt jelöli, míg az ’1’-’4’-es értékek a hozzájuk tartozó kártyaszínt

void RenderPile(CardPile pile, int x, int y, int cursorIndex, int selectedIndex): Kirajzol egy kártyakupacot a képernyőre, melynek a bal felső sarka az x, y, koordinátákban leledzik. A cursorIndex-edik kártyáját megjelöli a kurzor színével, a selectedIndex-ediket pedig a kiválasztás színével, ha az indexek invalid pozícióban vannak, a kártyák nem kerülnek különleges kiszínezésre

void Render(Table table, CardPosition cursorPosition, CardPosition selectedPosition): Kirajzolja az asztalon lévő kártyákat a képernyőre, megadott helyen a kurzorral és kiválasztással, ha invalid pozícióban vannak nem jeleníti meg őket

Logging.c és Logging.h

#define LOG_NAME: A fájl neve, ahova a program a naplózást végzi

#define ENABLE_LOG: A naplózás be van e kapcsolva, lehet true vagy false

#define CRASH_LOGGING: Minden naplóbejegyzés után bezárja és újra megnyitja a file-t, akkor érdemes használni, ha a program összeomlik és emiatt elveszhet a naplózófájl. A játék a futása során naplózza a játék fontosabb eseményeit.

extern File logFile*: A megnyitott naplózófájlra mutató pointer

extern bool successfulOpen: A naplózófájl megnyitásának sikeressége

#define LOG(message): Ha sikeresen meg van nyitva a naplózófájl beleírja a message üzenetet

#define LOG_ARGS(message, ...): Ha sikeresen meg van nyitva a naplózófájl beleírja a message üzenetet az extra argumentumokkal formázva

void StartLog(): Megnyitja a naplózófájlt és nullázza a tartalmát

void UpdateLog(): Bezárja és megnyitja a naplózófájlt, ezáltal elmenti a tartalmát

void CloseLog(): Bezárja a naplózófájlt

Leaderboard.c és Leaderboard.h

A leaderboard.c és leaderboard.h lehetővé teszi, hogy a játék elején a játékos bejelentkezzen egy felhasználónév segítségével és számon tartsa a nyeréseinek és játszott játékainak számát. A játék kezdetekor megjeleníti a legnagyobb pontszámmal rendelkező játékosokat ezzel ösztönözve a felhasználót, hogy ~~több időt pazaroljon a játékra~~ a tőle telhető legjobbat nyújtsa. A játékosok nevét, nyereségeinek és megkezdett játékainak számát egy kétszeresen tárolt listában tárolja és pontszám szerint rendezetten tartja. Amint egy játékos megkezd vagy megnyer egy játékot a fájl azonnal frissül, a változás azonnali elmentésre kerül.

#define LEADERBOARD_NAME: A ranglistafájl neve

#define LEADERBOARD_LENGTH: A menüben megjelenített n legnagyobb pontszámú játékos

#define MIN_NAME_LENGTH: Minimum hány karakterből kell a felhasználónévnek állnia

#define MAX_NAME_LENGTH: Maximum hány karakterből állhat a felhasználónév

struct UserData: Egy felhasználó adatait tárolja el, mutatók a listában fölötte és alatta lévő felhasználókra

UserData prev*: Ranglistában előző (kisebb pontszámú) felhasználó

UserData next*: Ranglistában következő (nagyobb pontszámú) felhasználó

char name[MAX_NAME_LENGTH + 1]: A felhasználó neve

int games: Megkezdett játékok száma

int wins: Megnyert játékok száma

UserData* currentUser: A felhasználó, melyként a játékos jelenleg be van jelentkezve

UserData* leaderboard: A ranglista első (legmagasabb pontszámú) eleme

static void RemoveChar(char str, char ch)*: Eltávolítja minden darabot egy megadott karakterből egy sztringből

static int CalculateScore(UserData user): Kiszámolja egy felhasználó pontszámát a képlet: score = wins * winrate% = 100 * wins^2 / games

static void SwapWithNext(UserData user)*: Kicseréli egy felhasználó listában elfoglalt helyét a listában következő felhasználóval

static void AdjustUserOrder(UserData user)*: Addig cserélgeti egy felhasználót a szomszédaival, amíg nem teljesül az a feltétel, hogy a felhasználó előtt ne legyen nála nagyobb, mögötte ne legyen nála kisebb pontszámú felhasználó

static UserData AddNewUser(UserData* entry)*: A lista elejére beszúr egy megadott új felhasználót és utána a helyére rendezi, feltételezi, hogy a lista a beszúrás előtt rendezett

static void RegisterUser(char UserName[MAX_NAME_LENGTH + 1]): Ha az adott néven létezik játékos, jelenleg bejelentkezett játékosnak kiválasztja, ha nem akkor létrehoz egy új felhasználót és bejelentkezik az új felhasználóként

static void DisplayLeaderboard(): Megjeleníti a ranglistát az első *LEADERBOARD_LENGTH* elemmel

bool ReadUserName(): Beolvassa egy játékos felhasználónevét karakterenként, enter billentyű lenyomására, amennyiben a név megfelel a hosszbeli követelményeknek, bejelentkezik az adott néven, amennyiben szükséges létrehozza, ilyenkor true értékkel tér vissza, ha azonban a felhasználó escape karakterrel kilép, false értéket ad vissza

void LoadLeaderBoard(): Megpróbálja a ranglista fájlt beolvasni és a benne található játékosokat rendezetten a listához fűzni, ha olyan sorral találkozik, amit nem képes értelmezni azt átugorja

static void SaveLeaderBoard(): Elmenti a ranglista jelenlegi állapotát a ranglista fájlba

void FreeLeaderboard(): Felszabadítja a ranglista memóriáját

void RegisterGame(): Eggyel megnöveli a jelenleg bejelentkezett játékos által megkezdett játékok számát és elmenti a ranglista változását

void RegisterWin(): Eggyel megnöveli a jelenleg bejelentkezett játékos által megnyert játékok számát és elmenti a ranglista változását

Solitaire.c

A főprogram, meghívja az inicializáló függvényeket és tartalmazza a játék fő logikáját

void InitCurses(): Beállítja a karakterek UTF-8 kódolását és meghívja az ncurses-hez kapcsolódó inicializáló függvényeket

void EndCurses(): Kilép az ncurses terminálkezelési módjából visszaadja az irányítást az alap terminálkezelő függvényeknek, így a debugmalloc meg tudja jeleníteni a végső összegzését

void GameLoop(): Elindít egy pasziánszjátékot, akkor lép ki ha a játékos megnyeri a játékot vagy kilép a játékból. A játék végeztével felszabadítja a menet során használt memóriát.

int main(void): A függvény, ahol a program megkezdi a futását, itt inicializálja a szükséges dolgokat. Elindítja a bejelentkezést és ha ez sikeres a játékot, ezt egészen addig amíg a játékos ki nem lép a menüből. Utána felszabadítja a foglalt memóriát.

4. Végszó

Köszönöm, hogy elolvasta a dokumentumot/a végére tetszett tekerni.