Szokoban - Programozói dokumentáció

Készítette: Szenes Mártons

Tartalom

Program felépítése

A program két fő részből áll, a menürendszerből (menu.c) és a játékból (game.c). Az indításkor először a menürendszer nyílik meg, és onnan tud a felhasználó elnavigálni az *Új játék* almenübe, majd a játékos kiválasztása után elindul a játék. A játék befejeztével visszalép a program a menübe, és innen tud a felhasználó más almenükbe átlépni vagy kilépni a programból.

A menü

A program indítása után a `main` beállítja a karaterkódolást, és meghívja a `void menu_MainScreen() {...}` -t. Ez az eljárás futtatja ciklikusan a menüt, amíg ki nem lép a felhasználó a programból.
A menü állapotait egy `enum State {...}` -ben tárolja a program, mivel véges számú állapota lehet a menünek, és ezáltal könnyű azonosítani az egyes menüpontokat.

Menü állapotai (State)

A menüben való navigálás során a következő állapotok léphetnek fel:

- `mainMenu` : A program indításákor ez az alapállot, a főmenüt jelenti.
- `newPlayer` : A főmenü első menüpontja, ez az állapot az új játékos hozzáadását jelenti.
- `chosePlayer` : A főmenü második menüpontja, ez az állapot a játékosválasztást jelenti.
- `rankList` : A főmenü harmadik menüpontja, a dicsőséglistát jelenti.

- `exitApp` : A főmenü negyedik menüpontja, kilépés a programból.
- `deletePlayer` : A `chosePlayer` állapotból léphet ebbe a program. A játékos törlését jelenti.
- `editPlayer` : A `chosePlayer` állapotból léphet ebbe a program. A játékosnév szerkesztését jelenti.
- `game` : A `chosePlayer` állapotból léphet ebbe a program. A játék futtatását jelenti.
- `exitGame` : A `game` állapotból léphet ebbe a program. A játékból való kilépést jelenti.
- `winGame` : A `game` állapotból léphet ebbe a program. Az összes szint teljesítését jelenti.

```
typedef enum State {
    mainMenu,
    newPlayer,
    chosePlayer,
    rankList,
    exitApp,
    deletePlayer,
    editPlayer,
    game,
    exitGame,
    winGame
} State;
```

Menü működése (MainScreen)

A `void menu_MainScreen()` eljárás pszedudókóddal leírva:

```
Eljárás Menü():
    Változók inicializálása
    Főcím kiiratása
    Szintek mappájának beolvasás

    Ciklus amíg menü fut
        Lenyomott billentyű kiértékelése
        Ha játékos kiválasztva, akkor
            Játék indítás
        Képernyőre írás menüpont
        Ha fut a menü, akkor
            Billentyűnyomásra vár
    Ciklus vége

    Játékosok adatainak mentése
    Lefoglalt memóriaterületek felszabadítása
Eljárás vége
```

Játékos (Player) struktúra

A fő adatstruktúra a menüben a `Player` struktúra. Ebben tárolja a program az egyes játékosok adatait: név, szint, statisztika, következő játékosra mutató pointer.

- `name` : A játékos nevét tárolja. Hossza a `datatypes.h` fájlban található makró szerint határozott meg (`#define nameLenght 20`). Ez azt jelenti, hogy a képernyőn 20 db karakter fog maximum megjelenni. Mivel ékezetes karaktereket is tartalmazhat a név (á, é, í, ó, ö, ő, ú, ü, ű), amik 2 byte-on tárolódnak, ezért a 2-szeresét vesszük és +1 byte-ot a lezáró nullának, így jön ki a hossza.
- `numOfCompletedLevels` : A játékos szintjét tárolja, hogy hágy szintet teljesített már.
- `*levelStats` : Egy `Statistics` típusú láncolt lista első elemére mutató pointer. Ebben tárolja el a program a játékos által megtett lépések számát az egyes szinteken.
- `*next` : A következő `Player` struktúrára mutató pointer a láncolt listában.

```
typedef struct player {
    char name[nameLenght*2+1];
    int numOfCompletedLevels;
    struct Statistics *levelStats;
    struct Player *next;
} Player;
```

Szint statisztika (Statistics) struktúra

A játékosok dicsőséglistájához elengedhetetlen számon tartani, hogy mely játékos, hány lépéssel tudta teljesíteni az egyes szeinteket. Ezt a tulajdonságot egy `Statistics` struktúrában tárolja a program.

- `stepCount` : A megtett lépések száma a szinten
- `*next` : A következő `Statistics` struktúrára mutató pointer a láncolt listában.

```
typedef struct statistic{
    int stepCount;
    struct Statistics *next;
} Statistics;
```

A játék

A játékot a `bool game_Init()` fügvénnyel lehet meghívni a `game.c` fájlon kívülről. Ez hívja meg benne a `bool game_StartGame()` függvényt, ami a játékot elindítja és futtatja ciklikusan. Erre a két függvényre azért van szükség, hogy a program moduláris lehessen. Tehát ha például más játékot szeretnénk leprogramozni, amiben hasonlóan több játékos lehet és a szintek egymás után következnek, akkor ugyanazokkal a paraméterekkel meg lehet hívni a `game_Init()` függvényt, ami majd a másik játék függvényeit hívja meg.

Játékmenet működése

A `bool game_StartGame()` függvény pszeudókóddal leírva:

```
Függvény Játék(játékos, szint): visza logikai
    Változók inicializálása
    Szint beolvasása fájlból
    Szint kiírása a képernyőre

    Ciklus amíg nincs a szint teljesítve
        Billentyűnyomásra vár
        Lenyomott billentyű kiértékelése:
            Ha Esc, akkor
                Kilépés
            Különben ha kurzor billentyű, akkor
                Ha lehetséges a lépés, akkor
                    Lépés lebonyolítása
            Különben ha V, akkor
                Előző lépés visszavonása
            Különben ha R, akkor
                Szint újrakezdése
        Elágazás vége
    Ciklus vége
    Lefoglalt memóriaterületek felszabadítása

    Ha szint teljesítve, akkor
        Játékos szintje növelése 1-el
        Lépések száma hozzáadása a játékos statisztikájához
        vissza igaz
    Különben
        vissza hamis
    Elágazás vége
Függvény vége
```

A pálya

A játékban a pályát, vagyis az adott szint mezőinek elrendezését a program a `levels` mappából olvassa be. Minden pálya külön `.xsb` fájlban van eltárolva. Így a `levels` mappához tetszés szerint lehet pályákat hozzáadni és elvenni. Fontos megjegyezni, hogy a program a pályákat fájlnevek szerint 'abc' rendben fogja beolvasni és eltárolni. Tehát fájlnév szerint növekvő sorrendben fognak következni egymás után a szintek. Így könnyű besorolni az egyes szinteket nehézségük szerint.
A pályákat két fő változó írja le:

```
CellType **map = NULL;
Size mapSize;
```

A `**map` egy kétdimenziós dinamikus tömb (mátrix), aminek minden eleme a pályán egy-egy mező, amiben `CellType` típussal kódolja a program a mező értékeit.
A `mapSize` egy `Size` struktúrában tárolja el a pálya méreteit.

Cellatípusok (CellType)

Egy pálya beolvasásakor az `.xsb` fájlt dekódolja a program, és a dekódolt értékeket a `**map` -ben tárolja el. Az egyes cellák a következő értékeket vehetik fel:

- `null` : Érvénytelen cellatípus, a fájl beolvasáskor rossz bemeneti karakter esetén.
- `EMPTY` : Üres cella, ami vagy a játéktéren kívül van, vagy a falakon belül, amin tud mozogni a játékos bábuja.
- `WALL` : A fal a játéktér határoló karatere. Erre nem léphet a játékos, nem tudja elmozdítani
- `TARGET` : A dobozok célmezője. Ezekkel jelölt cellákra kell a játékosnak tolnia a dobozokat. Erre léphet a játékos, de nem tudja elmozdítani.
- `PLAYER` : A játékbábu, ha üres mezőn áll.
- `PLAYERONTARGET` : A játékosbábú, ha célmezőn áll.
- `BOX` : A doboz, elmozdíthatja a játékos, de nem léphet rá erre a cellára.
- `BOXONTARGET` : A doboz, ha célmezőn van. A játékos el tudja mozdítni, de nem léphet rá.

```
typedef enum celltype {
    null,
    EMPTY,
    WALL,
    TARGET,
    PLAYER,
    PLAYERONTARGET,
    BOX,
    BOXONTARGET
} CellType;
```

.XSB fájl

A szokoban játékoknál a legtöbbet használt pályaleíró kiterjesztés a `.xsb` fájlformátum. Ez egy egyszerű szöveges dokumentum igazából, amiben minden karakter egy-egy cellát jelöl a pályán. A program ezt olvassa be és alakítja át `CellType` típusra a kezelhetőség miatt.
Az `.xsb` fájl által tartalmazható karakterek:

Karakter	Magyarázat	`CellType`
(Space)	Üres cella	`TARGET`
`#`	Fal	`WALL`
`.`	Célmező	`TARGET`
`@`	Játékos	`PLAYER`
`+`	Játékos egy célmezőn	`PLAYERONTARGET`
`$`	Doboz	`BOX`
`*`	Doboz egy célmezőn	`BOXONTARGET`

Példa.xsb

```
###
#.###
#*$ #
# @ #
#####
```

Méret (Size)

A pálya beolvasásnál először meghatározza a program, hogy mekkora pályára lesz szüksége a cellák eltárolásához. Így a pályának a szélessége az egyes sorokból a leghosszabb karakterszámú lesz, a magassága pedig a beolvasott nem üres sorok száma.
Ezt a `mapSize` változóban tárolja a program:

- `width` : A pálya teljes szélessége
- `height` : A pálya teljes magassága

```
typedef struct size{
    int width;
    int height;
} Size;
```

Pozíció (Point)

A `Point` struktúra egy kétdimenziós térben egy pont `x` és `y` koordinátáit tárolja el. Ezt a struktúrát sok helyen használja a program. A képernyőre való kiíráshoz, a dobozok (`*boxPositions`), a játékos poziciójának (`playerPosition`) vagy a lépések koordinátájának eltárolásához (`*playerMovesListHead`) is ez ad egyszerű kezelhetőséget.

```
typedef struct point{
    int x, y;
} Point;
```

Lépés (Move)

A játékos minden egyes lépését egy `Move` struktúrában tárolja el a program, és ezeket egy láncolt listába teszi ami veremként működik. Ez a lista a `Move *playerMovesListHead` . Ez azért szükséges, hogy lehessen visszavonni lépéseket. Ilyenkor a verem tetején lévő lépést visszacsinálja a program és törli azt a listából.
Egy lépést a következő tulajdonságok határoznak meg:

- `from` : A játékosbábu eredeti koordinátája, ahonnan ellép a játékos.
- `to` : A lépés irányában a következő cella koordinátája, ahova lép a játékos.
- `boxPushed` : Igaz, ha a lépés során eltolt egy dobozt a játékos, hamis, ha nem tolt el dobozt. Erre azért van szükség, hogy a lépés visszavonásakor tudja a program, hogy egy dobozt is kell-e visszamozdítani az előző pozíciójára vagy sem.
- `*next` : A következő `Move` struktútára mutató pointer a veremben.

```
typedef struct move{
    Point from;
    Point to;
    bool boxPushed;
    struct Move *next;
} Move;
```

Segédkönyvtárak

Debugmalloc

Készítők: *Czirkos Zoltán*, *Szekeres Dániel* · 2021.08.24.

A Debugmalloc egy varázs-malloc(), amely képes kilistázni a felszabadítatlan területeket, és ezzel megkönnyíti a hibakeresést. Bizonyos keretek között a túlindexelést is tudja ellenőrizni.

Forrás: INFOC - Debugmalloc, memóriakezelés

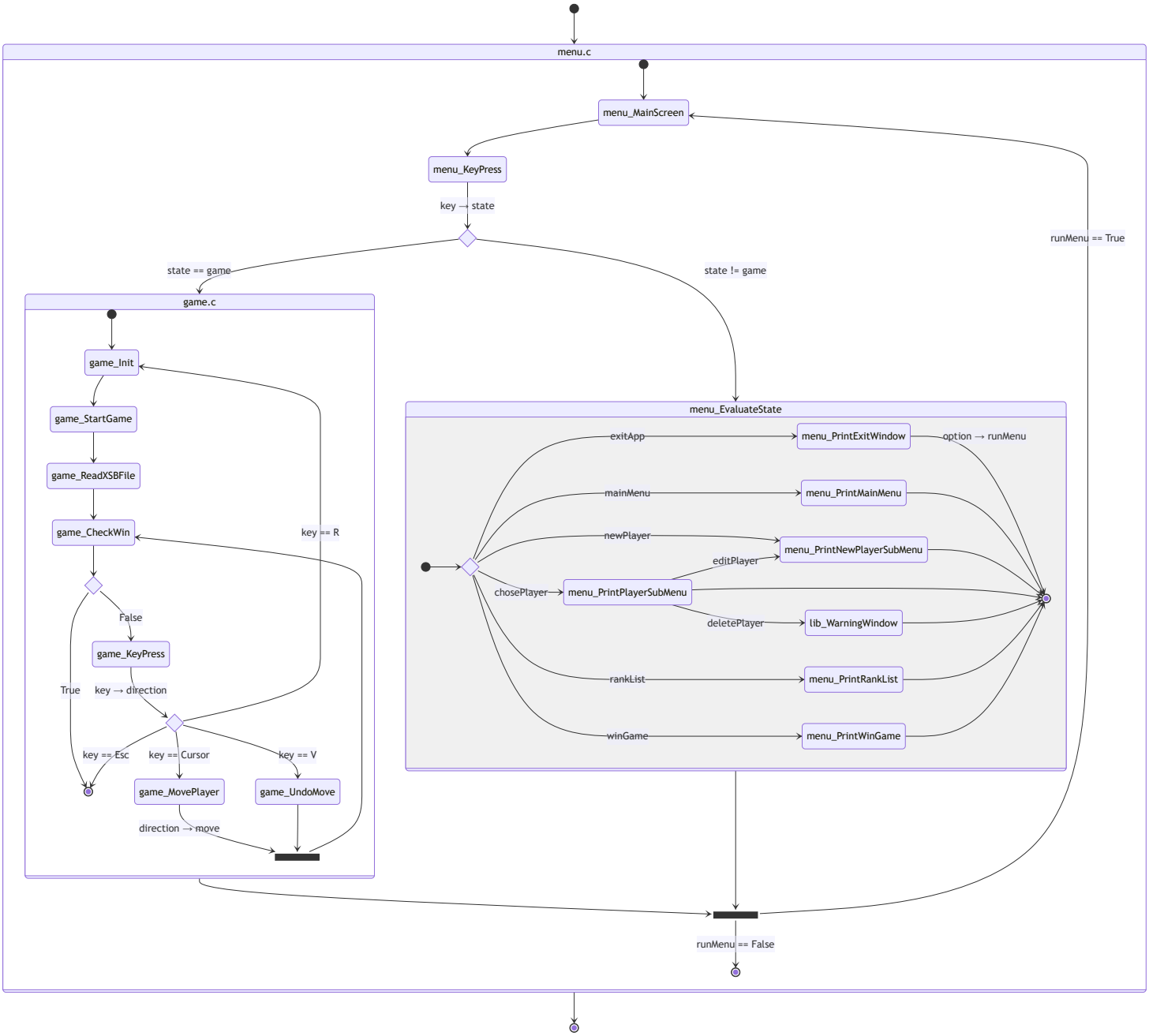
Econio

Készítő: *Czirkos Zoltán* · 2022.09.21.

A nagy házi feladatban használható, szöveges és grafikus megjelenítést segítő függvénykönyvtárak.

Forrás: INFOC - Szöveges és grafikus megjelenítés

Függvények rendszere



Függvénydokumentáció

Menu.h

`static void menu_PrintTitle()`

Kiírítja a főcímet a képernyő tetejére nagy betűkkel több sorosan.

`static void menu_PrintNavControls()`

Kiírja a képernyő aljára az aktuális menüpot nevigációs lehetőségeit.

`static void menu_PrintExitWindow(Point p)`

Kiír a képernyére egy ablakot, amiben megkérdezi a felhasználót, hogy biztos ki akar e lépni, Igen/Nem válaszlehetőségekkel.
Paraméterek:

- `Point` — `p` — A kiíráshoz legfelső középső pont a képernyőn.

`static void menu_PrintMainMenu(Point p)`

Kiírja a képernyőre a főmenü menüpontjait, és kijelöli az aktuálisan kiválasztott menüpontot más színnel.
Paraméterek:

- `Point` — `p` — A kiíráshoz legfelső középső pont a képernyőn.

`static void menu_PrintNewPlayerSubMenu(Player **playerListHead, int *numOfPlayers, Point p)`

Kiírja a képernyőre az új játékos felvételéhez szükséges mezőt, és a bemenet után hozzáadja a játéklistához az új játékost.
Paraméterek:

- `Player**` — `playerListHead` — A játékosok adatait tartalmazó láncolt lista. (Cím szerint)
- `int*` — `numOfPlayers` — A játékosok darabszáma. (Cím szerint)
- `Point` — `p` — A kiíráshoz legfelső középső pont a képernyőn.

`static void menu_PrintPlayerSubMenu(Player **playerListHead, Player **currentPlayer, int *numOfPlayers, Point p )`

Kiírja a képernyőre a játékoslistát, és kijelöli az aktuálisan kiválasztott játékost más színnel.
Paraméterek:

- `Player**` — `playerListHead` — A játékosok adatait tartalmazó láncolt lista. (Cím szerint)
- `Player**` — `currentPlayer` — A kiválasztott játékos struktúrájára mutató pointer, playerListHead elem. (Cím szerint)
- `int*` — `numOfPlayers` — A játékosok darabszáma. (Cím szerint)
- `Point` — `p` — A kiíráshoz legfelső középső pont a képernyőn.

`static void menu_PrintRankList(Player **playerListHead, int *numOfPlayers, Point p)`

Kiírja a képernyőre a dicsőséglistát táblázatosan.
Paraméterek:

- `Player**` — `playerListHead` — A játékosok adatait tartalmazó láncolt lista. (Cím szerint)
- `int*` — `numOfPlayers` — A játékosok darabszáma. (Cím szerint)
- `Point` — `p` — A kiíráshoz legfelső középső pont a képernyőn.

`static void menu_PrintWinGame(Point p)`

Kiírja a képernyőre, hogy teljesítette a szinteket a játékos.
Paraméterek:

- `Point` — `p` — A kiíráshoz legfelső középső pont a képernyőn.

`static void menu_ResetMenuVars()`

Visszaállítja a kezdőértékeket a menüben, ha menüpontváltás volt.

`static void menu_KeyPress(Player *currentPlayer, Player **playerListHead, int *numOfPlayers, char *levelList[], int nu`

Kiértékeli a felhasználó által lenyomott billentyőt, és megváltoztatja a program állapotát aszerint.
Paraméterek:

- `Player*` — `currentPlayer` — A kiválasztott játékos struktúrájára mutató pointer, playerListHead elem.
- `Player**` — `playerListHead` — A játékosok adatait tartalmazó láncolt lista. (Cím szerint)
- `*char[]` — `numOfPlayers` — A játékosok darabszáma. (Cím szerint)
- `int` — `levelList` — A szintek fájlnevét tároló dinamikus string tömb.

`static void menu_EvaluateState(Player **playerListHead, int *numOfPlayers, Player **currentPlayer, Point p, int *lines`

Az aktuális állapotnak megfelelően végrehajtja a szükséges utasításokat, és kiértékeli a bemeneteket.
Paraméterek:

- `Player**` — `playerListHead` — A játékosok adatait tartalmazó láncolt lista. (Cím szerint)
- `int*` — `numOfPlayers` — A játékosok darabszáma. (Cím szerint)
- `Player**` — `currentPlayer` — A kiválasztott játékos struktúrájára mutató pointer, playerListHead elem.
- `Point` — `p` — A kiíráshoz legfelső középső pont a képernyőn.
- `int*` — `linesPrinted` — Az előző menüpontba a képernyőre írt sorok.

`void menu_MainScreen()`

A főmenüt futtató függvény. Egyszer hivandó meg a mainben.

Player.h

`void player_ReadTxtFile(Player **playerListHead, int *numOfPlayers)`

Beolvassa a playerDataPath-ban megadott fájlt, és elátrolja a playerListHead láncolt listában
Paraméterek:

- `Player**` — `playerListHead` — A játékosok adatait tartalmazó láncolt lista (Cím szerint)
- `int*` — `numOfPlayers` — A játékosok darabszáma (Cím szerint)

`void player_WriteTxtFile(Player *playerListHead)`

Kiírja fájlba a playeListHead-ben tárolt játékosok adatait: név;szintek;lépészámok..
Paraméterek:

- `Player*` — `playerListHead` — sA játékosok adatait tartalmazó láncolt lista (Cím szerint)
- `int` — `numOfPlayers` — A játékosok darabszáma (Cím szerint)

`Player *player_MakePlayer(char name[], int numOfLevels, Statistics *statsListHead)`

Létrehoz egy Player struktúrára mutató pointert a paraméterként kapott értékekből, hogy aztán Lístába lehessen fűzni.
Paraméterek:

- `char[]` — `name` — A játékos neve (max 20 karakter)
- `int` — `numOfLevels` — A játékos által teljesített szintek száma
- `Statistics` — `statsListHead` — A játékos lépésstatisztikájának láncolt listája

Visszatér: `Player*` — Player struktúrára mutató pointer a kapott adatokkal

`void player_FreePlayerList(Player **playerListHead)`

Felszabadítja a az egész láncolt listának foglalt memóriát
Paraméterek:

- `Player**` — `playerListHead` — A játékosok adatait tartalmazó láncolt lista (Cím szerint)

`void player_FreePlayerNode(Player **playerNode)`

Felszabadítja egy elem lefoglalt memóriáját a listából
Paraméterek:

- `Player**` — `playerNode` — Egy Player struktúrára mutató pointer a láncolt listából (Cím szerint)

`void player_AddPlayerInOrder(Player *newPlayer, Player **playerListHead, int *numOfPlayers)`

Beszúrja a játékoslistába au új játékost a nevének a hossza szerint növekvő sorrendben
Paraméterek:

- `Player*` — `newPlayer` — Új játékos struktúrájára mutató pointer
- `Player**` — `playerListHead` — A játékosok adatait tartalmazó láncolt lista (Cím szerint)
- `int*` — `numOfPlayers` — A játékosok darabszáma (Cím szerint)

`bool player_RemovePlayer(Player *removablePlayer, Player **playerListHead, int *numOfPlayers)`

Törli a paramterként kapott játékost a listából
Paraméterek:

- `Player*` — `removablePlayer` — A törlendő játékos struktúrájára mutató pointer
- `Player**` — `playerListHead` — A játékosok adatait tartalmazó láncolt lista (Cím szerint)
- `int*` — `numOfPlayers` — A játékosok darabszáma (Cím szerint)

Visszatér: `bool` — Igaz, ha sikeres a törlés a listából; Hamis, ha nem sikerült törölni a játékost

`Player* player_GetSelectedPlayer(Player *playerListHead, int selectedPlayer)`

Megkeresi a listában a selectedPlayer-edik elemet
Paraméterek:

- `Player*` — `playerListHead` — A játékosok adatait tartalmazó láncolt lista
- `int` — `selectedPlayer` — A játékos sorszáma / indexe a listában

Visszatér: `Player*` — A keresett játékos struktúrájára mutató pointer, ha megtalálta, különben NULL pointer

`int player_GetIndexOfPlayer(Player *playerListHead, char name[])`

Megkeresi a listában a játékos nevét, és visszaadja a sorszámát / indexét a listában
Paraméterek:

- `Player*` — `playerListHead` — A játékosok adatait tartalmazó láncolt lista
- `char[]` — `name` — A keresett játékos neve

Visszatér: `int` — A keresett játékos indexe, ha megtalálta, különben -1

`void player_PrintPlayerList(Player *playerList, int selectedPlayerIndex, Point p)`

Kiírja a képernyőre a játékoslistát (nevüket és szintjüket) egymás alá, és kiemeli az aktuálisan kiválasztott játékost
Paraméterek:

- `Player*` — `playerList` — A játékosok adatait tartalmazó láncolt lista
- `int` — `selectedPlayerIndex` — Az aktuálisan kiválasztott játékos sorszáma / indexe
- `Point` — `p` — A kiíráshoz legfelső középső pont a képernyőn

Statistics.h

`void stats_AddLevelStatistics(int stepCount, Statistics **statsListHead)`

Beszúrja a paraméterként kapott stepCount értéket a statsListHead láncolt lista végére
Paraméterek:

- `int` — `stepCount` — A szinten megtett lépések száma
- `Statistics**` — `statsListHead` — A lépések számát tároló láncolt lista (Cím szerint)

`void stats_FreeStatisticsList(Statistics **statsListHead)`

Felszabadítja a az egész láncolt listának foglalt memóriát
Paraméterek:

- `Statistics**` — `statsListHead` — A lépések számát tároló láncolt lista (Cím szerint)

`void stats_FreeStatNode(Statistics **statNode)`

Felszabadítja egy elem lefoglalt memóriáját a listából

Paraméterek:

- `Statistics**` — `statNode` — Egy Statistics struktúrára mutató pointer a láncolt listából (Cím szerint)

Game.h

`bool game_Init(Player *player, char **levelList)`

Ezzel kell meghívni a játékot. Inicializálja a játékhoz szükséges elemeket.

Paraméterek:

- `Player*` — `player` — Az aktuális játékos adatait tartalmazza. (Cím szerint)
- `char[][]` — `levelList` — A pályák fájlneveit tartalmazó string tömb.

Visszatér: `bool` — Igaz, ha a játékos teljesítette a szintet; Hamis, ha a játékos kilépett a játékból.

`static bool game_KeyPress(CellType **map, Size mapSize, int *numOfMoves, Player *player, Point *playerPosition, Point`

Kiértékeli a felhasználó által lenyomott billentyőt, és megváltoztatja a játék vagy a játékosbábu helyzetét aszerint.

Paraméterek:

- `CellType**` — `map` — A pályát leíró 2D-s dinamikus mátrix.
- `Size` — `mapSize` — A pálya szélességét és magasságát leíró struktúra.
- `int*` — `numOfMoves` — A megtett lépések száma a szinten. (Cím szeirnt)
- `Player*` — `player` — Az aktuális játékos adatait tartalmazza. (Cím szerint)
- `Point*` — `playerPosition` — A játékos kezdő koordinátája a pályán (map-en). (Cím szerint)
- `Point*` — `boxPositions` — A dobozok koordinátáinak dinamikus tömbje. (Cím szerint)
- `Move**` — `movesListHead` — A játékos lépéseit eltároló láncolt lista. (Cím szerint)

Visszatér: `bool` — Igaz, ha újraindítja a játékos a szinten; Különben hamis

`static bool game_StartGame(Player *player, char levelName[])`

Ez a függvény indítja el és futtatja ciklikusan a játékot.

Paraméterek:

- `Player*` — `player` — Az aktuális játékos adatait tartalmazza. (Cím szerint)
- `char[]` — `levelName` — A betöltendő pálya fájlneve.

Visszatér: `bool` — Igaz, ha a játékos teljesítette a szintet; Hamis, ha a játékos kilépett a játékból.

`static bool game_CheckWin(CellType **map, Size mapSize)`

Ellenőrzi, hogy a játékos teljesítette-e a szintet, vagyis, hogy minden doboz a helyére került-e.

Paraméterek:

- `CellType**` — `map` — A pályát leíró 2D-s dinamikus mátrix.
- `Size` — `mapSize` — A pálya szélességét és magasságát leíró struktúra.

Visszatér: `bool` — Igaz, ha minden doboz a helyére került; Hamis, ha van egy doboz is, ami nincs a helyén.

`static bool game_MovePlayer(CellType ***map, Point *currentPosition, Point **boxPositions, Point direction, Move **mov`

A játékos elmozdulását tesztelő függvény a direction irányba. Ha lehetséges a lépés, vagy ha doboz van a lépés irányában akkor a dobozt eltolja, és igazat ad vissza; Hamis, ha nem lehetséges a lépés.

Paraméterek:

- `CellType***` — `map` — A pályát leíró 2D-s dinamikus mátrix. (Cím szerint)
- `Point*` — `currentPosition` — A játékos aktuális koordinátája a pályán (map-en). (Cím szerint)
- `Point**` — `boxPositions` — A dobozok koordinátáinak dinamikus tömbje. (Cím szerint)

- `Point` — `direction` — A játékos elmozdulásvektora.
- `Move**` — `movesListHead` — A játékos lépéseit eltároló láncolt lista. (Cím szerint)

Visszatér: `bool` — Igaz, ha el tud mozdulni a játékos az adott irányba; Hamis, ha nem lehetséges a lépés.

`static bool game_UndoMove(CellType ***map, Point *currentPosition, Point **boxPositions, Move **moveListHead)`

Visszavonja a játékos előző lépést. Egészen addig fut le sikeresen, amíg a moveList-ben volt elem, azaz meglépett lépés.
Paraméterek:

- `Celltype**` — `map` — A pályát leíró 2D-s dinamikus mátrix (Cím szerint)
- `Point*` — `currentPosition` — A játékos aktuális koordinátája a pályán (map-en) (Cím szerint)
- `Point**` — `boxPositions` — A dobozok koordinátáinak dinamikus tömbje (Cím szerint)
- `Move**` — `moveListHead` — A játékos lépéseit eltároló láncolt lista (Cím szerint)

Visszatér: `bool` — Igaz, ha sikeresen visszavonta a lépést; Hamis, ha nincs több visszavonható lépés

`static void game_ReadXSBFile(char filename[], CellType ***map, Size *mapSize, Point *playerPosition, Point **boxPositi`

Beolvassa a kapott fájlnévben lévő pályát és eltárolja a map mátrixban
Paraméterek:

- `char[]` — `filename` — A pálya fájlneve (Bemenet)
- `CellType***` — `map` — A pályát leíró 2D-s dinamikus mátrix (Cím szerint, kimenet)
- `Size*` — `mapSize` — A pálya szélességét és magasságát leíró struktúra (Cím szerint, kimenet)
- `Point*` — `playerPosition` — A játékos kezdő koordinátája a pályán (map-en) (Cím szerint, kimenet)
- `Point**` — `boxPositions` — A dobozok koordinátáinak dinamikus tömbje (Cím szerint, kimenet)
- `int*` — `boxCount` — A dobozok koordinátáit tároló dinamikus tömb elemszáma, dobozok száma a pályán (Cím szerint, kimenet)

`static CellType game_ConvertInputCharToCellType(char character)`

A beolvasott fájl egy karakterét értelmezi és átalakítja cellType értékké
Paraméterek:

- `char` — `character` — char A beolvasott fájl egy karaktere

Visszatér: `CellType` — A kapott karakter értelmezett CellType értékekké alakított értéke

`static void game_PrintStyledMap(CellType **map, Size mapSize)`

Kiiratja a pályát a képernyőre színesen
Paraméterek:

- `CellType**` — `map` — A pályát leíró 2D-s dinamikus mátrix
- `Size` — `mapSize` — A pálya szélességét és magasságát leíró struktúra

`static void game_PrintPosition(CellType **map, Point pos)`

Egy kapott koordinátán lévő mezőt írja ki a képernyőre színesen és a megfelelő definiált karaterrel
Paraméterek:

- `CellType**` — `map` — A pályát leíró 2D-s dinamikus mátrix
- `Point` — `pos` — A kiirandó karakter koordinátája

`static void game_PrintStatsAndNav(Size mapSize, int numOfSteps, int level)`

Kiírja a képernyőre a játéktér mellé az aktuális szintet és a lépések számát, vagy a tutorial pályánál a bevezető instrukciókat
Paraméterek:

- `Size` — `mapSize` — A pálya szélességét és magasságát leíró struktúra
- `int` — `numOfSteps` — A szinten megtett lépések száma
- `int` — `level` — Az aktuális szint száma

`static void game_AllocateMemoryToMap(CellType ***map, Size *mapSize)`

Memóriát foglal a pályát tároló 2D-s dinamikus mátrixnak (map-nek)
Paraméterek:

- `CellType***` — `map` — A pályát leíró 2D-s dinamikus mátrix (Cím szerint)
- `Size*` — `mapSize` — A pálya szélességét és magasságát leíró struktúra

`static void game_AllocateDynamicArray(Point **newArray, int lenght)`

Memóriát foglal egy 1D-s dinamikus tömbnek
Paraméterek:

- `Point**` — `newArray` — Az új 1D-s dinamikus tömb címe (Cím szerint)
- `int` — `lenght` — A létrehozandó dinamikus tömb hossza, elemszáma

`static void game_FreeAllocatedMemoryFromMap(CellType ***map)`

Felszabadítja a pályát tároló 2D-s dinamikus mátrix lefoglalt memóriáját, ha volt lefoglalva
Paraméterek:

- `Celltype***` — `map` — A pályát leíró 2D-s dinamikus mátrix (Cím szerint)

`static void game_FreeDynamicArray(Point **dynamicArray)`

Felszabadítja egy 1D-s dinamikus tömbnek lefoglalt memóriáját, ha volt lefoglalva
Paraméterek:

- `Point**` — `dynamicArray` — 1D-s dinamikus tömb címe (Cím szerint)

Level.h

`void level_ReadDirectory(char directory[], char **levelList[], int *numOfFiles)`

Beolvassa a megadott mappából a fájlneveket, és eltárolja egy dinamikusan foglalt tömbben
Paraméterek:

- `char[]` — `directory` — A mappa elérési útvonala
- `*char[][]` — `levelList` — A dinamikusan foglalt string tömb (Cím szerint)
- `int*` — `numOfFiles` — A Beolvasott fájlnevek száma (Cím szerint)

`void level_FreeLevelList(char **levelList[], int *numOfLevels)`

Felszabadítja a dinamikusan foglalt fájlnevek string tömbjét
Paraméterek:

- `*char[][]` — `levelList` — A dinamikusan foglalt string tömb (Cím szerint)
- `int*` — `numOfLevels` — A Beolvasott fájlnevek száma (Cím szerint)

Lib.h

`void lib_WarningWindow(const char Message[], Point p, bool *displayFirst, int option, EconioColor baseColor, EconioCol`

Kiír a képernyőre egy figyelmeztető ablakot a megadott Message üzenettel, Igen/Nem válaszlehetőségekkel
Paraméterek:

- `char[]` — `Message` — Az üzenet
- `Point` — `p` — képernyő közepének koordinátája
- `bool*` — `displayFirst` — Először megy-e be a ciklus a menüpontba tulajdonság (Cím szerint)
- `int` — `option` — A kiválasztott opció

- `EconioColor` — `baseColor` — Alap betűszíne az ablaknak
- `EconioColor` — `accentForeColor` — Kijelölt opció betűszíne
- `EconioColor` — `accentBgColor` — Kijelölt opció háttérszíne

`void lib_ClearScrBellow()`

Letörli a cím alatt lévő területet a képernyőről

`void lib_ClearScreenSection(int x1, int y1, int x2, int y2, EconioColor bgColor)`

Letörli a képernyőt megadott koordinátákon belül a kapott bgColor színnel.
Paraméterek:

- `int` — `x1` — Bal felső sarok x koordinátája
- `int` — `y1` — Bal felső sarok y koordinátája
- `int` — `x2` — Jobb alsó sarok x koordinátája
- `int` — `y2` — Jobb alsó sarok y koordinátája
- `EconioColor` — `bgColor` — Törlendő terület háttérszíne

`void lib_printError(const char errormessage[])`

Kiírja képernyőre a hibaüzenetet
Paraméterek:

- `char[]` — `errormessage` — Hibaüzenet

`//void print(char const str[], int x, int y)`

Kiírja a képernyőre a kapott szöveget a megadott kezdő koordinátákra
Paraméterek:

- `char[]` — `str` — Kiírandó szöveg
- `int` — `x` — x koordináta a képernyőn
- `int` — `y` — y koordniáta a képernyőn

`void printfc(char const str[], int x, int y, EconioColor foreColor)`

Kiírja a képernyőre a kapott szöveget a megadott kezdő koordinátákra a megadott betűszínnel
Paraméterek:

- `char[]` — `str` — Kiírandó szöveg
- `int` — `x` — x koordináta a képernyőn
- `int` — `y` — y koordniáta a képernyőn
- `EconioColor` — `foreColor` — A szöveg színe

`void printfbc(char const str[], int x, int y, EconioColor foreColor, EconioColor bgColor)`

Kiírja a képernyőre a kapott szöveget a megadott kezdő koordinátákra a megadott betűszínnel és háttérszínnel
Paraméterek:

- `char[]` — `str` — Kiírandó szöveg
- `int` — `x` — x koordináta a képernyőn
- `int` — `y` — y koordniáta a képernyőn
- `EconioColor` — `foreColor` — A szöveg színe
- `EconioColor` — `bgColor` — A szöveg háttérszíne

`Point addPoints(Point a, Point b)`

Két pont koordinátáit összeadó függvény
Paraméterek:

- `Point` — `a` — Egyik koordináta
- `Point` — `b` — Másik koordináta

Visszatér: `Point` — Koordináták összege

`Point subPoints(Point a, Point b)`

Két pont koordinátáinak különbségét kiszámoló függvény
Paraméterek:

- `Point` — `a` — Egyik koordináta
- `Point` — `b` — Másik koordniáta

Visszatér: `Point` — Koordináták különbsége

`bool comparePoints(Point a, Point b)`

Két pontot hasonlít össze, hogy egyenlőek-e
Paraméterek:

- `Point` — `a` — Egyik koordináta
- `Point` — `b` — Másik koordináta

Visszatér: `bool` — Egyenlőek e a paraméterként kapott koordináták

`int utf8_strlen(const char str[])`

Megszámolja, hogy a kapott string hány utf8 karakterből áll, hány krakter íródik ki a képernyőre
Paraméterek:

- `char[]` — `str` — Karaktertömb, string (Bemenet)

Visszatér: `int` — A string hossza megjelenített karakterszámban

`int stringlenghtMax(const char str[], int max)`

Megszámolja, hogy hány byte-on tárolódik a max karakterszámú string
Paraméterek:

- `char[]` — `str` — Karaktertömb, string (Bemenet)
- `int` — `max` — Megjelenítendő karakterek száma

Visszatér: `int` — Megjelenítendő string max karakterű byte hossza

`bool isBlankString(const char* str)`

Megnézi a függvény, hogy a string csak üres karaktereket tartalmaz-e (szóköz, \n, \t)
Paraméterek:

- `char[]` — `str` — Karaktertömb, string (Bemenet)

Visszatér: `bool` — Csak üres karaktereket tartalmaz-e a string

Move.h

`Move* move_CreateMove(Point stepfrom, Point stepTo, bool boxPushed)`

Létrehoz egy Move struktúrára mutató pointert a paraméterként kapott értékekből, hogy aztán Lístába lehessen fűzni.
Paraméterek:

- `Point` — `stepfrom` — A legutolsó pozíció koordinátája
- `Point` — `stepTo` — A következő pozíció koordinátája
- `bool` — `boxPushed` — Igaz, ha eltolt doboz; Hamis, ha csak a játékos mozdult el

Visszatér: `Move*` — Move strúktúrára mutató pointer a kapott paraméterekkel

`void move_AddMoveToList(Move *newMove, Move **moveListHead)`

Beszúrja a paraméterként kapott newMove elemet a láncolt lista (moveListHead) elejére

Paraméterek:

- `Move*` — `newMove` — Új elmozdulást tároló struktúrára mutató pointer
- `Move**` — `moveListHead` — Az elmozdulásokat tároló láncolt lista (Verem/Stack)

`Move move_RemoveMoveFromList(Move **moveListHead)`

Eltávolítja az első elemet a láncolt listából (Veremből/Stack)

Paraméterek:

- `Move**` — `moveListHead` — Az elmozdulásokat tároló láncolt lista (Verem/Stack) (Cím szerint)

Visszatér: `Move` — Visszaadja az eltávolított listaelem struktúráját

`void move_FreeMoveList(Move **moveListHead)`

Felszabadítja a az egész láncolt listának foglalt memóriát

Paraméterek:

- `Move**` — `moveListHead` — Az elmozdulásokat tároló láncolt lista (Verem/Stack) (Cím szerint)

`void move_FreeNode(Move **moveNode)`

Felszabadítja egy elem lefoglalt memóriáját a listából

Paraméterek:

- `Move**` — `moveNode` — Egy Move struktúrára mutató pointer a láncolt listából (Cím szerint)