Szokoban - Programozói dokumentáció

Készítette: Szenes Mártons

Program felépítése

A program két fő részből áll, a menürendszerből és a játékból. Az indításkor először a menürendszer nyílik meg, és onnan tud a felhasználó tovább navigálni a játék elindításáig. A menü állapotait egy `enum State {...}`-ben tárolja, mivel véges számú állapota lehet a menünek, és ezáltal könnyű azonosítani az egyes menüpontokat. A menü aktuális állapotát egy `State` változó tárolja, amit az egyes menüpontok kiválasztásakor változtat a program. A menüt `void menu_MainScreen() {...}` a `main` hívja meg egyszer, a program indulásakor. Ez az eljárás futtatja ciklikusan a mneüt, amíg ki nem lép a felhasználó a programból.

A menü felépítése

```
void menu_MainScreen(){
    // Konstansok  és változók inicializálása
    ...
    // Főcím kiiratása
    menu_PrintTitle();
    // Szintek mappájának beolvasása
    level_ReadDirectory("./levels/", &levelList, &numOfLevels);
    econio_rawmode(); // Billentyűérzékelés bekapcsolása

    // == MENÜ FUTTATÁSA ==
    while(runMenu){
        p = (Point) {center, 9}; // Koordináta beállítása
        prevOption = option; // Előző kijelölés eltárolása;

        // Lenyomott billentyű kiértékelése
        menu_KeyPress(...);
        // Képernyőre írás választott mód szerint
        menu_EvaluateState(...);
        // Billentyűlenyomásra vár, ha fut a menü
        if (runMenu && !displayFirst) key = econio_getch();
    }
    // Játékosok adatainak mentése
    player_WriteTxtFile(...);
    // Játékosok listájának felszabadítása
    player_FreePlayerList(...); // Player lista felszabadítása
    // Lefoglalt levelLista felszabadítása
    level_FreeLevelList(...);
}
```

Adatstruktúrák

Enumerátorok

Mezőtípusok (CellType)

Mezők

```
/* A pálya egyes mezőinek lehetséges értékei.*/
typedef enum celltype { null, EMPTY, WALL, TARGET, PLAYER, PLAYERONTARGET, BOX,
BOXONTARGET } CellType;
```

Menü állapotai (State)

```
/* A menü lehetséges állapotértékei */
typedef enum State { mainMenu, newPlayer, chosePlayer, rankList, exitApp,
deletePlayer, editPlayer, game, exitGame, winGame } State;
```

Struktúrák

Pozíció (Point)

```
/* Egy koordinátát eltároló struktúra, mely láncolt listába fűzhető */
typedef struct position{
    int x, y;           // Koordinátái
    struct Point *next; // A következő pontra mutató pointer a láncolt listában
} Point;
```

Méret (Size)

```
/* A pálya méretét eltároló struktúra */
typedef struct size{
    int width;  // A pálya szélessége
    int height; // A pálya magassága
} Size;
```

Lépés (Move)

```
/* A játékos egy lépését tároló struktúra, mely láncolt listába fűzhető */
typedef struct move{
    Point from;         // A játékos által elhagyott mező koordinátája
    Point to;           // A játékos által meglépett mező koordinátája
    bool boxPushed;     // Logikai, eltolt-e a játékos a lépés során dobozt
    struct Move *next;  // a lépéseket tároló láncolt listában a következő elemre
mutató pointer
} Move;
```

Szint statisztika (Statistics)

```
/* A játékos egy szinten megtett lépéseinek számát tároló struktúra, mely láncolt
listába fűzhető */
typedef struct statistic{
    int stepCount;           // Egy szinten a játékos által megtett lépések száma
    struct Statistics *next; // A statisztika láncolt listában a következő elemre
mutató pointer
} Statistics;
```

Játékos (Player)

```
/* A játékos adatait eltároló struktúra, mely láncolt listába fűzhető */
typedef struct player {
    char name[nameLenght*2+1];      // Játékosnév: maximum nameLenght hosszú (*2+1
az ékezetes karakterek és a \0 miatt)
    int numOfCompletedLevels;       // Teljesített szintek száma
    struct Statistics *levelStats;  // A teljesített szinteken megtett lépések
száma láncolt listában
    struct Player *next;            // A játékos láncolt listában a következő
elemre mutató pointer
} Player;
```

!!! bug Ez egy bug
!!! attention
!!! deprecated
!!! warning

Algoritmusok

Player.h

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

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

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

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

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

- `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 Listába lehessen fűzni. **Parameters:**

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

Returns: `Player` — struktúrűra mutató pointer a kapott adatokkal

`void player_FreePlayerList(Player **playerListHead)`

Felszabadítja a az egész láncolt listának foglalt memóriát **Parameters:**

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

`static void player_FreePlayerNode(Player **playerNode)`

Felszabadítja egy elem lefoglalt memóriáját a listából **Parameters:**

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

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

Beszúrja a játékoslistának a végére az új játékos elemet **Parameters:**

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

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

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

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

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

Törli a paramterként kapott játékost a listából **Parameters:**

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

Returns: `Logikai:` — 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 **Parameters:**

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

Returns: `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 **Parameters:**

- `A` — `playerListHead` — játékosok adatait tartalmazó láncolt lista
- `A` — `name` — keresett játékos neve

Returns: `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 **Parameters:**

- `A` — `playerList` — játékosok adatait tartalmazó láncolt lista
- `Az` — `selectedPlayerIndex` — aktuálisan kiválasztott játékos sorszáma / indexe
- `A` — `p` — 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 **Parameters:**

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

```
void stats_FreeStatisticsList(Statistics **statsListHead)
```

Felszabadítja a az egész láncolt listának foglalt memóriát **Parameters:**

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

```
static void stats_FreeStatNode(Statistics **statNode)
```

Felszabadítja egy elem lefoglalt memóriáját a listából **Parameters:**

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

Kód szerekzete

Kód részeltes dokumentációja

> powershell: D:\Programozas\soko> & "C:\Users\Szenes Márton\node_modules.bin\docblox2md" .\SokobanDevDoc.md