Sokoban
Programozói dokumentáció

Program felépítése

- main.c – program kezdőpontja, ez hívja meg a menüt, játékot és pályaszerkesztőt
- menu.c – főmenü kirajzolása a kijelzőre
- play.c – játék játszása
- edit.c – szintszerkesztő
- file.c – fájlkezelés, szintekhez láncolt lista építése
- input.c – szövegbeviteli mező kezelése, előugró ablakok kezelése
- tiles.c – pályák mezőinek kezelése és ezek kijelzőre rajzolása
- coordinates.h – koordinátákat tartalmazó struktúra

Adatstruktúrák

A menu.c, play.c és editor.c mindegyike rendelkezik egy saját adatstruktúrával az aktuális állapot tárolásához. Ezekben a struktúrákban van eltárolva az adott programrész állapota, és a függvények nagy része kizárólag ezt az állapotot tartalmazó struktúrát kapják meg. Mindhárom adatstruktúra tartalmazza az SDL-hez szükséges pointereket.
A betöltött szinteket láncolt listában tárolja a program, amelyet a fájl betöltésekor generál. Ez a struktúra kétirányú láncolást tesz lehetővé, ami a szintek közötti könnyű lépkedést teszi lehetővé.

```
typedef struct Level
{
	Coordinates size;
	TileState *tiles;
	char *name;
	struct Level *prev;
	struct Level *next;
} Level;
```

A size-ban van eltárolva a szint mérete, a tiles-ban vannak a szint egyes mezőinek állapota. Ez azért csak egy sima lista, és nem egy 2D-s tömb, mivel így sokkal egyszerűbb a tárolása, és csak egy kicsivel bonyolultabb a hozzáférés (minden sor utolsó eleme után következik a következő sor eleje elválasztó nélkül, ezért mindenképpen kell tudni a pálya méretét). A name-ben van eltárolva a szint neve egy dinamikusan foglalt memóriaterületen. A prev és next az előző, illetve következő pályára mutatnak.

Fájlok betöltése és kiírása

A sokobanhoz használt fájlok valamennyire szabványosítva vannak, de így is van többféle variánsa. Például az egyiknél a pálya előtt van a neve, másiknál utána, van ahol vannak nem teljes sorok stb. Ezeket a programnak tudnia kell kezelnie. A fájlok beolvasásához az alábbi struktúrát használja a program

```
typedef struct LoadState
{
	char line[64];
	int lineLength;
	char name[64];
	TileState level[209];
	int rowIndex;
	int maxLength;
} LoadState;
```

Itt a program soronként olvassa be és dolgozza fel a fájlokat. Ha egy sok elején pontosvessző van, akkor azt azonnal beírja a szint nevének, ha több ilyen sor is van, akkor mindig felülírja az aktuális nevet, azaz mindig a legutolsó név marad meg
Amikor egy sor elején érvényes pálya karaktert talál, akkor azt lekezdi betölteni a level megfelelő helyére. Azt a pálya méretének tudata nélkül úgy csinálja, hogy minden sort 19 hosszúnak (maximális támogatott hossz) vesz, és úgy tölti ki a változót. Emellett számon tartja, hogy jelenleg melyik sorba ír, és hogy menni volt az eddigi leghosszabb sor. Ez majd a szint láncolt listához való hozzáadásakor lesz fontos.
A szint beolvasás akkor ér véget, amikor egy üres sort olvas a program, vagy a fájl végére ért. Ilyenkor a program a memóriát foglal a leendő szint elemnek, majd megfelelően elkezdi kitölteni. A szint mérete a rowIndex-ből és a maxLength-ből következik, ennek megfelelő mennyiségű memóriát foglal a program, majd bemásolja oda a pályát. A szint nevének is pontosan annyi memóriát foglal, amennyire szükség van. Ezek után beszúrja az újonnan készített szintet a láncolt lista végére

Szövegbeviteli mező és az ékezetes karakterek

Az ékezetes karakterek kezelése meglepően bonyolult. C-ben egy ékezetes karakter az 2 különálló karakterből áll, amely kettőből áll össze maga az ékezetes karakter. A szövegbeviteli mezőnél ez kimondottan fontos volt, mivel a szint nevét is ezzel lehet megadni.
A szövegbeviteli mező egy memóriában tárolt stringet módisít a bemeneteknek megfelelően, és mindig ezt írja ki a kijelzőre. Az ékezetes betűk miatt minden billentyűt külön kell kezelni. Egy billentyű lenyomásakor (vagy a kijelzőn lévő billentyűre kattintáskor) a program a bevitt stringhez egyszerűen hozzácsatolja az új karaktert. Szerencsére itt minden működik a ékezetes karakterekkel. A törlés sajnos már nem ilyen egyszerű, viszont ki lehet használni, hogy az összes magyar ékezetes karakternek mindkét tagja egyenként érvénytelen betű, azaz nincs benne a hagyományos ASCII táblában. A törlés függvény egyszerűen ezt nézi: ha az utolsó két karakter kódja „érvénytelen”, akkor mindkettőt törli (azaz töröl egy ékezetes betűt), és ha nem, akkor csak egyet töröl.

Programban használt főbb függvények

SDL_Renderer *initSDL()
SDL inicializálása. Visszatér SDL_Renderer-rel ha sikeres volt és NULL-lal ha sikertelen.

static TileState getTileState(Level *level, int x, int y)
A szinten egy mező állapotának lekérdezése. Ez a függvény számolja át a koordinátákat indexszé. Túlindexelés esetén érvénytelen állapotot ad vissza

static void setTileState(Level *level, int x, int y, TileState state)
A szinten egy mező állapotának beállítása. Ez a függvény számolja át a koordinátákat indexszé. Túlindexelés esetén nem csinál semmit

static void render(PlayState *state)
static void render(EditState *state)
static void renderMenu(MenuState *state)
static void renderInput(InputState *state)
Különböző állapotok kijelzőre rendrerlése. Mindegyik megkapja a state-ben a program jelenlegi állapotát (amit renderelnie kell) és a rendereléshes szükséges SDL-es pointereket

static bool fillState(PlayState *state, Level *level)
Szint betöltése a játékállapotba. Szint visszaállításához szükséges. Memóriát foglal a szintnek

static void freeState(PlayState *state)
Előző függvény által foglalt memóriát szabadítja fel.

static void nextLevel(PlayState *state)
static void prevLevel(PlayState *state)
static void nextLevel(EditState *state)
static void prevLevel(EditState *state)

Előző és következő szintekre ugrás (játék é szerkesztés módban). Mindig a jelenlegi állapotba tölti be az új szintet, ezért nincs visszatérési értéke.

`static bool checkLevels(Level *level)`
Szintek érvényességének vizsgálata. A játszás függvény futtatja le játék elkezdése előtt.

`static bool processMovement(int dir, PlayState *state)`
`static bool processMovement(int dir, MenuState *state)`
Mozgás feldolgozása megadott irány szerint. Ez a függvény tologatja a ládát is és teszteli, hogy a lépés érvényes-e. Visszaérési értéke igaz, ha a pályát újra ki kell rajzolni a kijelzőre (azaz változott benne valami).

`static bool handleEvent(SDL_Event event, PlayState *state)`
`static bool handleEvent(SDL_Event event, MenuState *state)`
`static bool handleEvent(SDL_Event event, EditState *state)`
`static bool handleEvent(SDL_Event event, InputState *state)`
SDL események kezelése. Itt történik a billentyű lenyomások tesztelése és az egér kattintások feldolgozása.

`int mainMenu(SDL_Renderer *renderer, SDL_Texture *tiles, TTF_Font *font, char *filename)`
Főmenü kirajzolása és futtatása. Visszatérési értéke 0, ha bezárták az ablakot, 1, ha játék, 2, ha szintszertkesztőt választott a játékos.

`int playLevel(SDL_Renderer *renderer, SDL_Texture *tiles, TTF_Font *font, char *filename)`
Pályák betöltése és játszása. Visszatérési értéke 0, ha bezárták az ablakot, 1, ha sikeresen véget ért a játék.

`int editLevel(SDL_Renderer *renderer, SDL_Texture *tiles, TTF_Font *font, char *filename)`
Szintszerkesztő megnyitása. Visszatérési értéke 0, ha bezárták az ablakot, 1, ha sikeresen véget ért a szint szerkesztése.

`static void addLevel(EditState *state, bool dir)`
Szintszerkesztés közben új szint hozzáadása a megadott irányba.

`static void deleteCurrent(EditState *state)`
Szintszerkesztés közben jelenlegi szint törlése.

`LoadLevelResult loadLevel(char *filename)`
Szint betöltése fájlból. A visszatérési struktúra tartalmazza a betöltés eredményességét és a láncolt listára mutató pointert (amennyiben eredményes volt a betöltés).

`static int addLevel(LoadState *state, Level **first, Level **current)`
Szint betöltése közben új szint hozzáadása az éppen épülő láncolt listához.

`void unloadLevel(Level *level)`
Szintek felszabadítása.

`bool saveLevel(Level *level, char *filename)`
Szintek elmentése az adott fájlba. Nem szabadítja fel a memóriát.

`int textInput(SDL_Renderer *renderer, SDL_Texture *tiles, TTF_Font *font, char *promptText, char *enteredText, int maxLength)`
Szöveges bemenet kérése a felhasználótól. Kozvetlenül a megadott memóriacímre írja a szöveget. Visszatérési értéke 0, ha bezárták az ablakot, 1, ha sikeres volt a bevitel, 2, ha sikertelen.

`int alertBox(SDL_Renderer *renderer, SDL_Texture *tiles, TTF_Font *font, char *promptText)`
Értesítés mutatása a felhasználónak. Visszatérési értéke 0, ha bezárták az ablakot, 1 ha sikeres volt az előugró ablak.

`int dialogBox(SDL_Renderer *renderer, SDL_Texture *tiles, TTF_Font *font, char *promptText)`
Eldöntendő kérdés feltevése a felhasználónak. Visszatérési értéke 0, ha bezárták az ablakot, 1, ha a felhasználó igennel válaszolt, 2, ha nem.

```
void renderTile(SDL_Renderer *renderer, SDL_Texture *tiles, Tile tile, int x, int y)
```
Játékmező kirajzolása az alábbi koordinátákra.

```
void renderTiles(SDL_Renderer *renderer, SDL_Texture *tiles, Tile tile, int x1, int y1, int x2, int y2)
```
Játékmezők kirajzolása az alábbi koordinátákkal közrefogott téglalapba

```
void renderFont(SDL_Renderer *renderer, TTF_Font *font, SDL_Color color, char *text, int x, int y,
bool centeredX, bool centeredY)
```
Szöveg kirajzolása az alábbi koordinátákra.

```
bool clickTile(int tileX, int tileY, int clickX, int clickY)
```
Mezőre kattintás tesztelése.