Életjáték

előzetes dokumentáció

A program használata

A program indításakor megjelenik maga a játéktér. Itt bal klikkel lehet cellákat élőre rajzolni, és jobb klikkel halottra. Ha a felhasználó nyomva tartja az egeret, akkor folyamatosan rajzol. A program további vezérlése billentyűkkel történik egyelőre, amíg a menü nincs kész.

A jelenlegi vezérlőbillentyűk:

- C: játéktér összes cellájának halottra állítása (törlés)
- Return (enter): szimuláció léptetése 1-el (kézi léptetés)
- Space: automatikus léptetés ki/bekapcsolása
- Fel/le nyíl: automatikus léptetés sebességének növelése/csökkentése

A játéknak 3 állapota van: nem csinál semmit/üres mód, rajzolás mód és automatikus léptetés mód. Egyszerre csak 1 állapotban lehet. Például nem lehet auto. léptetés közben rajzolni vagy játékteret törölni. Kézi léptetést pedig csak üres módban lehet végezni.

A program felépítése

main.c : program kindulási pontját és az időzítőket tartalmazza.

typedefs.h : típusdefiníciók (struccok, enumok)

display.c/display.h : megjelenítéssel kapcsolatos függvények

gamelogic.c/gamelogic.h : játéklogikával, szimulációval kapcsolatos függvények

Typedefs

- CellState enum

  Egy cella lehetséges állapotai: **DEAD**, **LIVE**

- GameField struct

  Struct a játéktér (cellák) tárolására

  - **cells, newCells** kétdimenziós tömbök, a newCells tömbbe kerül kiszámításra a következő iteráció, majd a 2 tömb pointere helyet cserél
  - **sizeX, sizeY** a cellák mennyisége a 2 irányban

- GridParams struct

  Struct a megjelenítéshez szükséges paraméterek tárolására

- SimData struct

  Struct a szimulációhoz szükséges adatok mozgatására

Display

* Inicializálja az SDL-t és létrehozza a renderert és egy ablakot.

* @param width Ablak szélessége (pixel).

* @param height Ablak magassága (pixel).

* @param title Ablak neve.

* @param pwindow A létrehozott ablak pointere milyen memóriacímre legyen írva.

* @param prenderer A létrehozott renderer pointere milyen memóriacímre legyen írva.

void sdl_init(int width, int height, char *title, SDL_Window **pwindow, SDL_Renderer **prenderer);

* Létrehozza a grid paramétereket tartalmazó structot dinamikus memóriában.

* @param width A játéktér szélessége pixelben.

* @param height A játéktér magassága pixelben.

* @param cellsX Cellák száma vízszintes irányban.

* @param cellsY Cellák száma függőleges irányban.

* @param padding A játéktér margójának szélessége.

* @param deadColor A halott cellák színe.

* @param liveColor Az élő cellák színe.

* @param borderColor A szegély színe.

* @param bgColor A háttérszín, a margónak a színe.

* @return A létrehozott struct példány pointere, a hívó kötelessége felszabadítani a free_grid_params() függvény hívásával.

GridParams *create_grid_params(short width, short height, short cellsX, short cellsY, short padding, Uint32 deadColor, Uint32 liveColor, Uint32 borderColor, Uint32 bgColor);

* Felszabadítja a memóriából a megadott grid paraméterek példányt.

* @param params A grid paraméterek példány.

void free_grid_params(GridParams *params);

* Kitölti az összes pixelt az ablakban a 0,0 saroktól a megadott width-1,height-1 sarokig a megadott színnel.

* A renderert szükséges renderelésre meghívni a függvény visszatérte után.

* @param renderer A renderer.

* @param width A szélesség, amekkora területet szeretnénk átszínezni.

* @param height A magasság, amekkora területet szeretnénk átszínezni.

* @param bgcolor A színezéshez használt szín.

void clear_background(SDL_Renderer *renderer, short width, short height, SDL_Color *bgcolor);

* Egy négyzetrácsot rajzol a rendererbe a megadott grid paraméterekkel.

* A renderert szükséges renderelésre meghívni a függvény visszatérte után.

* @param renderer A renderer.

* @param params A grid paraméterek, amikkel a négyzetrács pozícionálása és színezése történik.

void draw_grid(SDL_Renderer *renderer, GridParams *params);

* Kirajzolja a játéktér élő és halott celláit a megadott játéktér és grid paraméterek alapján.

* A renderert szükséges renderelésre meghívni a függvény visszatérte után.

* @param renderer A renderer.

* @param params A grid paraméterek.

* @param field A játéktér.

void draw_cells(SDL_Renderer *renderer, GridParams *params, GameField *field);

Gamelogic

* A játéktér összes celláját halottra állítja.

* @param field A játéktér.

void clear_cells(GameField *field);

* Dinamikusan létrehoz egy 2D-s CellState tömböt.

* @param sizeX A tömb vízszintes mérete.

* @param sizeY A tömb függőleges mérete.

* @return A létrehozott tömb pointere.

CellState **create_2D_array(short sizeX, short sizeY);

* Létrehoz egy megadott méretű játékteret.

* @param sizeX A játéktér hány cella széles legyen.

* @param sizeY A játéktér hány cella magas legyen.

* @return A létrehozott játéktér példány pointere, a hívó kötelessége felszabadítani a free_field() függvény hívásával.

GameField* create_field(short sizeX, short sizeY);

* Felszabadítja a memóriából a megadott játékteret.

* @param field A játéktér.

void free_field(GameField *field);

* Kiszámítja, hogy a megadott x és y képernyőkoordinátákon melyik cella van,

* és megváltoztatja annak állapotát a megadott állapotra.

* @param field A játéktér.

* @param params A grid paraméterek.

* @param x A változtatni kívánt cella x képernyőkoordinátája.

* @param y A változtatni kívánt cella y képernyőkoordinátája.

* @param state A változtatni kívánt cella új állapota.

void change_cell(GameField *field, GridParams *params, int x, int y, CellState state);

```
 * A játéktér egy cellájának lekérdezése átfordulással,
 * vagyis ha -1-et kérdezünk, akkor átugrik a pálya másik végére
 * és az lesz a lekérdezett cella, másik irányú túlindexelésnél szintúgy.
 * @param field A játéktér.
 * @param x A kérdezett cella x indexe. Lehet -1 vagy játéktér mérete értékű is.
 * @param y A kérdezett cella y indexe. Lehet -1 vagy játéktér mérete értékű is.
 * @return A lekérdezett cellaérték.
CellState get_cell(GameField *field, int x, int y);

 * Megszámolja, a megadott indexű cellának mennyi élő szomszédja van a 8-ból.
 * @param field A játéktér.
 * @param x A kérdezett cella x indexe.
 * @param y A kérdezett cella y indexe.
 * @return Élő szomszédok száma. (min 0, max 8)
int get_neighbor_count(GameField *field, int x, int y);

 * A játéktér következő iterációját kiszámolja a newCells tömbben,
 * majd megcseréli a cells és newCells tömböket.
 * @param field A játéktér.
void evolve(GameField *field);
```