Programozói dokumentáció

Játékszabályok és a szimuláció menete

A játékban egy populációt szimulálunk.
A szabályok egyszerűek:

- Adott egy valamilyen méretű négyzethálóra felosztott sík amelyekben a négyzetek sejteket jelölnek.
- A sejteknek 2 egyszerű állapotuk lehet: vagy halottak vagy éppen élnek. A kezdő állapotuk adott.
- Minden sejt a következő lépésben amelynek 3x3-as környezetében, ha:
  - több mint 3 élő sejt van, túlnépesedés miatt meghal.
  - 3 élő sejt van, feléled vagyis ha élő volt élő marad, ha halott volt akkor élő lesz.
  - 2 élő sejt van, akkor változatlan marad az állapota.
  - 2-nél kevesebb élő sejt van, meghal.

Adatszerkezetek

A program fő célja hogy egy sík 2 dimenziós tér keretein belül sejteket szimuláljon. Mivel az elvárások szerint a játéktér fix méretű, tehát a szimuláció alatt nem változik, a céljaink elérésére, és a sejtek adatainak tárolására teljesen tökéletesen megfelel egy átlagos két dimenziós statikus tömb amelyet dinamikusan tárolunk hogy tetszőleges méretű lehessen. A választható méretek miatt szükségünk lesz tehát egy szerkezetre amely tárolja a sejteket és a játéktér méreteit. Az egyszerűség kedvéért és azért hogy ne keljen minden sornak `malloc`-al külön külön memóriát foglalni, a

tömb **1 dimenziós** lesz (emiatt a tömb mérete `width * height`).
A mi esetünkben ez a `struct GameState`:

```
typedef struct GameState {
    int width;
    int height;
    bool* cells;
} GameState;
```

> ✎ Note
>
> Mivel a sejtekről csak egy 1 bites információt tárolunk (hogy éppen él vagy sem), teljesen elegendő egy "1 bites" boolean tömbben tárolnunk.

Továbbá szükségünk lesz egy adatszerkezetre amellyel koordinátákat tárolunk, hogy megkönnyítsük a függvények közötti kommunikációt, és hogy egyszerűbb legyen a koordinátákkal dolgozni.
A mi esetünkben ez a `struct Vector`:

```
typedef struct Vector {
    int x;
    int y;
} Vector;
```

Függvények

A fönti adatszerkezetek mellé szükségünk lesz segéd függvényekre:

```
int getCellIndex(Vector coords, int width);
```

Átalakít egy `struct Vector`-t arra az indexre ami megfeleltethető az adott koordináták helyén lévő sejtnek. Röviden: Átmenetet képez `Vector` és a

`GameState`-ben tárolt `bool* cells` között.

> **Important**
>
> Az egyszerűség kedvéért egyenlőre egy 1 dimenziós tömbben tárlojuk a sejteket, emiatt van szükség erre a függvényre, hogy átalakítsunk egy 2 dimenziós koordinátát egy indexre

```
GameState* createNewState(int width, int height);
```

Dinamikus memória kezeléssel létrehoz egy `GameState` -et a megadott hosszúság és szélesség szerint majd visszaad egy pointert ami erre mutat. Fontos hogy alapból nem nullázza ki az összes sejt adatát, hanem memória szemét van még a tömbben.

```
void destroyGameState(GameState* state);
```

A `createNewState` el létrehozott `GameState` dinamikusan lefoglalt memóriáját felszabadítja.

```
void clearCells(GameState* game);
```

Kinullázza egy `GameState` cell tömbjét.

```
void randomizeCells(GameState* game);
```

Random `bool`-okkal feltölti a `GameState` cell tömbjét.

> **Info**
>
> A random függvény használata előtt meg kell hívni az `srand`-ot:
>
> ```
> srand(time(NULL));
> ```
>
> Ez teljes mértékben a függvényt felhasználóra van bízva

További a játékot irányító függvények:

```
int countAliveNeighbours(Vector* pos, GameState* game);
```

Egy `GameState`-ben megszámolja a `pos` koordinátán lévő sejt 3x3 as környezetében az élő sejteket.

```
GameState* calculateNextState(GameState* game);
```

Egy adott `GameState`-nek visszaadja következő iterációját, a fenti szabályokat alkalmazva, egy új dinamikusan foglalt `GameState`-ben.

> **Important**
>
> A memóriaszivárgást elkerülendő fontos hogy amikor új `GameState` et hozunk létre, a régit a `destroyGameState` el töröljük

További instrukciók, megjegyzések:

A függvények és az adatstruktúrák külön modul-t kaptak, ezek elérése egyszerűen a `conwayGame.h` header file include-olásával megoldható. Az adatszerkezetek a header fileban kaptak helyet a függvények implementációja kommentekkel együtt pedig a c source code fileban vannak.

A `main` függvény mellett a `main.c` fileban található még egy render függvény is.

```
void renderState(GameState* game);
```

Ez a függvény `printf`-el a konzolon megjeleníti az adott jázékállást

Megjegyzés:

Úgy gondolom ez a függvény nem a játék moduljába való mivel igazából a `GameState` -en kívül semmi köze nincs a játékhoz. Nem a játék menetével hanem annak megjelenítésével foglalkozik, ebből kifolyólag szerintem teljesen saját modulja kéne legyen. Mivel 1 db függvényről van szó ami közel sem a végleges függvény ezért egyenlőre nem hoztam létre külön modult neki.

> **Todo**
>
> A jövőben a megjelenítéssel foglalkozó logika egy game engine modulban lesz, amely ugyanúgy a konzolon fogja a játékot megjeleníteni. Ezen felül a game engine fog foglalkozni a felhasználóval való kommunikációval és az adatok bekérésével. Ezek szintúgy nem kapcsolódnak a játékhoz, hanem generikus logika amit más programban ugyanígy fel lehet használni

További változtatások a jövőben:

> **Todo**
>
> A fájlból való beolvasás, a játékállás mentése és a menüvezérlés implementálása

> **Todo**
>
> Paraméter kezelés és vezérlés
> `gameoflife.exe <param>`

> **Todo**

Általános hibakezelés implementálása és üzenetek a felhasználó felé
Pl.: `malloc`-nál jelenleg nem vizsgálunk rá hogy sikerült e a memória foglalás

Todo

Általános optimalizálás ha van rá mód

Bug

A megjelenítésnél jelenleg minél nagyobb pályát választunk annál látványosabban villog a kép a karakterek kiírásának lassúságából adódóan

Bug

A pálya széléhez érve a sejtek furán viselkednek. A pálya széle úgy viselkedik mint egy keret halott sejtekből.

Question

A pálya széle hogy funkcionáljon? Jó hogy olyan mint egy keret halott sejtekből? Ha nem akkor hogyan máshogy?

Ábrázolás

Esetleg van-e jobb módja a sejtek ábrázolásának. Jobb karakter?
Pl.: `#` és `.`

Felhasználói dokumentáció

Egyszerű a felhasználás:

- Indítsuk el a programot valamilyen terminálban pl.: `cmd`
- A programot az `ESC` gomb lenyomásával megállíthatjuk