

BPC-PC2M

Závěrečný projekt

Game of Life

<https://github.com/krysar/Game-of-life>

Obsah

1	Game of Life	1
2	Základní popis programu	1
3	Ovládání	2
4	Formát vstupního souboru	2
5	Herní pole	2
6	Formát výpisu	3
7	Vývoj herního pole	5
8	Kompatibilita	6

1 Game of Life

Hra života (anglicky *Game of Life*) je dvoustavový celulární automat, který má chováním připomínat společenství živých organismů. Odehrává se na 2D matici buněk.

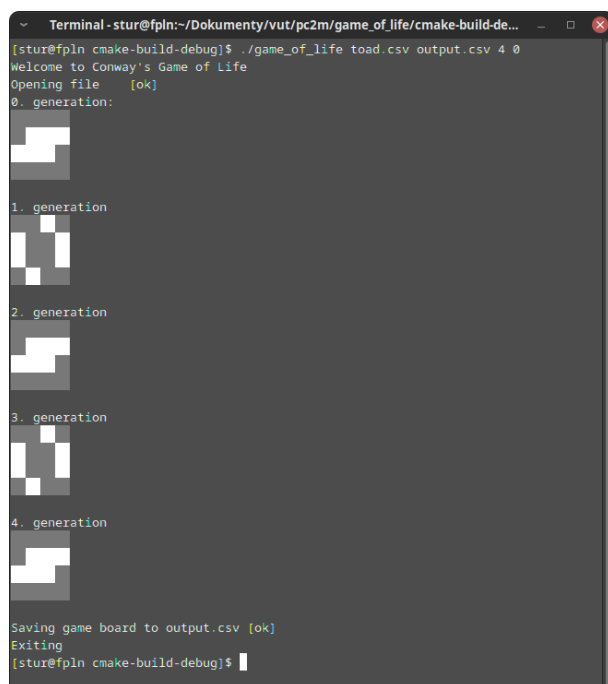
Buňka může nabývat dvou stavů - je buď mrtvá nebo živá. Její stav v následující generaci je určen stavem jí samotné a jejího Moorova okolí (tj. buněk, se kterými sousedí hranou nebo vrcholem).

Hra života je definována pravidly sestavenými britským matematikem Johnem Conwayem, jež jsou označovány jako S23/B3:

1. Každá živá buňka s méně než dvěma živými sousedy zemře.
2. Každá živá buňka se dvěma nebo třemi živými sousedy zůstává žít.
3. Každá živá buňka s více než třemi živými sousedy zemře.
4. Každá mrtvá buňka s právě třemi živými sousedy oživne.

2 Základní popis programu

Program načítá výchozí konfiguraci z csv souboru (viz kapitola 3), nechá proběhnout vývoj herního pole po stanovený počet generací a výsledné herní pole uloží do csv souboru ve stejném formátu, jako je vstup. Průběžný vývoj herního pole je vypisován do konzole jedním ze dvou způsobů.



```
Terminal - stur@fpln:~/Dokumenty/vut/pc2m/game_of_life/cmake-build-de...
[stur@fpln cmake-build-debug]$ ./game_of_life toad.csv output.csv 4 0
Welcome to Conway's Game of Life
Opening file [ok]
0. generation:
  █
  █
  █
  █
  █

1. generation
  █
  █
  █
  █
  █

2. generation
  █
  █
  █
  █
  █

3. generation
  █
  █
  █
  █
  █

4. generation
  █
  █
  █
  █
  █

Saving game board to output.csv [ok]
Exiting
[stur@fpln cmake-build-debug]$
```

Obrázek 1: Příklad běhu programu pro 4 generace oscilátoru typu Toad (ropucha)

3 Ovládání

Veškeré parametry jsou programu předávány při spuštění pomocí argumentů příkazového řádku. Syntaxe je následovná:

```
> název_programu [INPUT FILE] [OUTPUT FILE] [GENERATION COUNT] [PRINTING TYPE]
```

Popis parametrů:

<code>název_programu</code>	Název spouštěného programu, např. <code>game_of_life.exe</code>
<code>[INPUT FILE]</code>	Vstupní csv soubor
<code>[OUTPUT FILE]</code>	Výstupní csv soubor
<code>[GENERATION COUNT]</code>	Počet generací, po který má hra probíhat. Počáteční je nultá.
<code>[PRINTING TYPE]</code>	Formát výpisu do konzole (viz kapitola 5)

Příklad:

```
> game_of_life.exe toad.csv output.csv 30 1
```

Vzhledem ke způsobu předávání parametrů programu a absenci jakékoliv *zarážky* zabráňující ukončení programu po jeho doběhnutí je nutno program spouštět přímo z prostředí příkazové řádky, nikoliv tedy například rozkliknutím `.exe` souboru ve správci souborů.

4 Formát vstupního souboru

Vstupem je csv soubor, ten definuje počáteční konfiguraci a také velikost herního pole. Jde o tabulku jedniček a nul, jednička je buňka živá, nula buňka mrtvá. Povolený oddělovač sloupců je středník (;) a čárka (.). Velikost herního pole je definována počtem řádků a sloupců tabulky, pole musí být čtvercové (tj. počet sloupců se musí rovnat počtu řádků).

```
0;0;0;0
0;1;1;1
1;1;1;0
0;0;0;0
```

Listing 1: Příklad vstupního csv souboru pro oscilátor typu Toad

Velikost pole je omezena na 255 řádků/sloupců.

5 Herní pole

Jak již bylo zmíněno, jde o matici dvoustavových prvků. V programu je implementováno jako dynamicky alokované 2D pole (pointer na pointer) booleanů (datový typ `bool` z hlavičkového souboru `stdbool.h`). Jeho alokace probíhá poněkud komplikovaněji v rámci funkce `read_csv`.

Před počátkem čtení je pole alokováno s velikostí 1x1. Při každém načtení oddělovače sloupců (viz předchozí kapitola) dojde k realokaci s inkrementovaným počtem sloupců, při každém načtení nového řádku pak k realokaci s inkrementovaným počtem řádků. Průběžně probíhá kontrola symetričnosti pole.

Zdrojový kód zde není pro jeho délku začleněn, funkce se nachází v souboru `gol_csv.c`.

6 Formát výpisu

Výpis probíhá do *stdout*, což je standardně konzole. Pro správné zobrazení (především u výstupního formátu 1) je nutno zajistit, aby počet řádků konzole odpovídal velikosti herního pole a počet sloupců jeho dvojnásobku.

Posledním vstupním parametrem programu je `[PRINTING TYPE]` definující druh výpisu, povolené hodnoty jsou 0 a 1.

6.1 Formát 0

Tento formát vypíše všechny generace herního pole pod sebe do konzole.

6.2 Formát 1

Po každém výpisu herního pole do konzole je zavolán systémový příkaz na její vyčištění (pro Windows *cls*, pro Linux *clear*), díky čemuž je možno relativně názorně sledovat pohyb buněk v poli. Omezením je především rychlost výpisu, u většího herního pole lze narazit na problikávání. Výpis taktéž není nijak bržděn, proto má tenhle formát výpisu smysl jen pro velký počet generací.

Níže je zdrojový kód funkce sloužící k výpisu do konzole, zavolání správného systémového příkazu pro danou platformu je zajištěno podmíněným překladem.

```

1 void print_board(bool **field, uint8_t row_count, uint8_t col_count, uint8_t print_type) {
2     #ifdef WIN32
3         if(print_type == 1)
4             system("cls");
5         else if(print_type != 0) {
6             printf("Error: undefined printing format\n");
7             exit(ERR_UNDEFINED_PRINTING);
8         }
9     #elifdef __linux
10        if(print_type == 1)
11            system("clear");
12        else if(print_type != 0) {
13            printf("Error: undefined printing format\n");
14            exit(ERR_UNDEFINED_PRINTING);
15        }
16    #endif
17    for(register uint8_t i = 0; i < row_count; ++i) {
18        for(register uint8_t j = 0; j < col_count; ++j) {
19            if(field[i][j])
20                printf("\u2588\u2588"); // Full block
21            else
22                printf("\u2591\u2591"); // Light shade
23        }
24        printf("\n");
25    }
26    printf("\n");
27 }

```

Listing 2: Funkce pro výpis herního pole

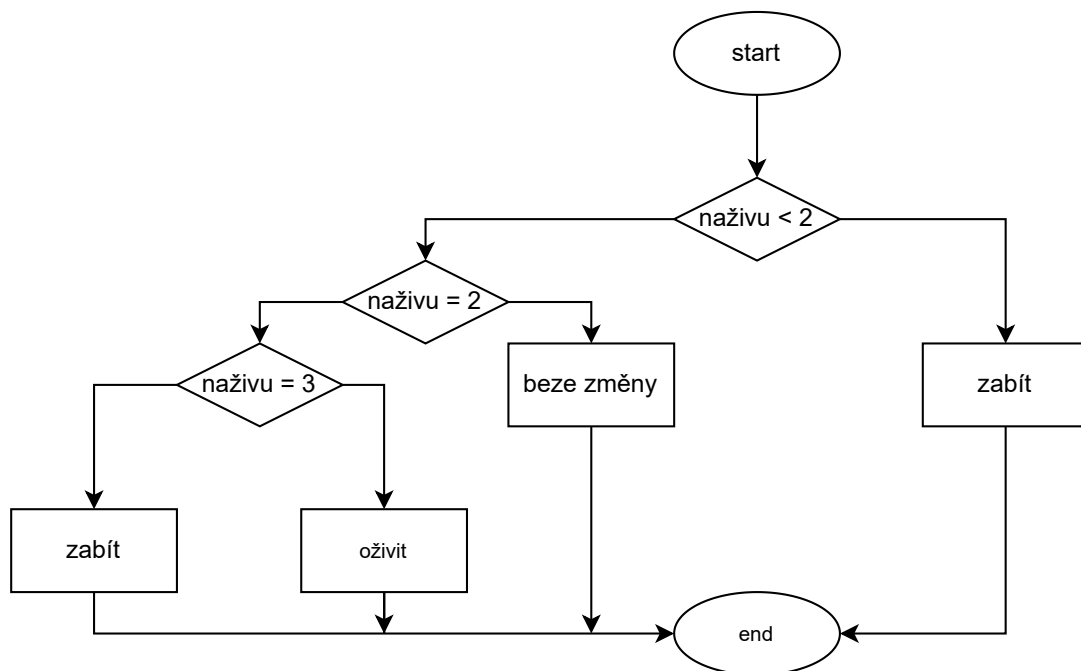
Povšimněte si použitých znaků pro pro buňky. Pro živou buňku jsou použity dva znaky *full block*, pro mrtvou dva znaky *light shade*. Nejde o znaky z ASCII, nýbrž z Unicode. Pod Linuxem (a také jinými moderními UN*X systémy) to není problém, standardně se již léta používá znaková sada UTF-8. Pod Windows ovšem nikoliv, je proto ji třeba už na začátku běhu programu manuálně přepnout. Toho jsem docílil podmíněným překladem na začátku funkce main:

```
1 int main(int argc, char *argv[]) {  
2     // On Windows UTF-8 isn't default charset but we're using Unicode characters  
3     #ifdef _WIN32  
4         system("CHCP 65001");  
5     #endif  
6     ...  
}
```

Listing 3: Přepínání znakové sady ve Windows

7 Vývoj herního pole

Základní algoritmus pro výpočet následující generace je poměrně jednoduchý a je objasněn v první kapitole. Taktéž lze vyjádřit tímto vývojovým diagramem:



Obrázek 2: Diagram vývoje herního pole

V rámci programu je algoritmus (s mírným zjednodušením) implementován ve funkci `update_board` závislé na funkci `get_num_neighbours` sloužící ke zjištění počtu živých buněk v Moorově okolí zkoumané buňky.

8 Kompatibilita

Program je plně kompatibilní s linuxovými distribucemi používající znakovou sadu UTF-8 a se systémem Windows. Částečná kompatibilita je s ostatními UN*X systémy (např. Mac OS), zde není bez doplnění podmíněného překladu možno použít *formát výpisu 1*.

Pro korektní výpis herního pole do konzole je taktéž vhodné, aby počet řádků konzole byl roven velikosti pole a počet sloupců jeho dvojnásobku (čehož lze docílit zmenšením fontu).