The Game of Life

Cybulski, Krzysztof
Rybicki, Damian

JIMP 2 2017

Spis treści

1 Opis ogólny

1.1 Nazwa programu

The Game of Life

1.2 Poruszany problem

Gra w życie została wymyślona w roku 1970 przez brytyjskiego matematyka Johna Conwaya. Jest jednym z pierwszych i najbardziej znanych przykładów automatu komórkowego.

Gra toczy się na prostokątnej planszy n x m podzielonej na kwadratowe komórki. Każda komórka ma określoną ilość „sąsiadów" (zwykle jest to osiem lub cztery), czyli komórki przylegające do niej bokami i rogami. Każda komórka może znajdować się w jednym z dwóch stanów: może być albo „żywa" (włączona), albo „martwa" (wyłączona). Stany komórek zmieniają się w pewnych jednostkach czasu. Stan wszystkich komórek w pewnej jednostce czasu jest używany do obliczenia stanu wszystkich komórek w następnej jednostce. Po obliczeniu wszystkie komórki zmieniają swój stan dokładnie w tym samym momencie. Stan komórki zależy tylko od liczby jej żywych sąsiadów.

2 Budowa programu

2.1 Struktura katalogów

```
/
├── obj/
│   └── *.o
├── src/
│   ├── *.c
│   ├── game/
│   ├── parser/
│   ├── utils/
│   └── test/
├── resources/
│   ├── snaps/
│   ├── structures/
│   ├── rules/
│   └── welcome/
├── docs/
│   ├── doc.tex
│   └── doc.pdf
├── main
└── Makefile
```

2.2 Algorytm

Algorytm zakłada, że plansza składa się z pól oznaczonych liczbami całkowitymi reprezentującymi ilość sąsiadów oraz informację o tym czy dana komórka jest żywa czy nie. Korzystając z tej informacji oraz listy komórek, które powinny się zmienić w następnej generacji algorytm wykonuje następujące kroki:

1. Zmieniamy typ (żywa/martwa) każdej komórki z listy aktywnych komórek (funkcja *invert*)
2. W zależności czy dana komórka zmarła czy się urodziła zwiększami lub zmiejszamy sumę sąsiadów wszystkich sąsiadujących komórek.
   (a) Jeśli nowa wartość u sąsiada zawiera się w wartościach oznaczających śmierć bądź urodzenie dodajemy tą komórkę do listy nowych aktywnych komórek.
3. Zastępujemy listę aktywnych komórek nową listą

Takie podejście sprawdza się świetnie w przypadku dużych plansz, ponieważ ilość operacji nie zależy od wielości planszy a jedynie od ilości komórek, które zmieniają się w następnej generacji. W przypadku gdy cała plansza jest wypełniona zmieniającymi się strukturami może okazać się, że program będzie działał wolniej od sprawdzającego po kolei całą planszę. Głównym tego powodem jest konieczność sprawdzenia czy komórka dodana do listy aktywnych komórek nie powtarza się.

2.3 Przechowywanie danych

Struktura zawierająca mapę oraz udostępniane funkcje

```
typedef struct Map {
    char *name;
    int width;
    int height;
    int *cells;
} *map_t;

map_t alloc_map(char*, int, int);
int invert(map_t, int cell_index);
int increment(map_t, int cell_index, int active_cell_value);
```

2.3.1 Zasady

Struktura zawierająca zasady oraz udostępniane funkcje

```
typedef struct Rules {
        char *name;
    int *live;
        int live_n;
        int *born;
        int born_n;
        int (*neighbours)[2];
        int neighbours_amount;
} *rules_t;
```

2.3.2 Gra

Struktura zawierająca aktualną grę oraz udostępniane funkcje

```
typedef struct Game {
        map_t map;
        rules_t rules;
    int age;
        int *actives;
        int actives_amount;
} *game_t;

game_t start(rules_t, map_t);
int recalculate(game_t);
int step(game_t);
int place(game_t, int *new_active_cells_indexes, int n);
```

2.4 Generowanie plików graficznych

Do generowania plików .png ze stanem planszy wykorzystujemy bibliotekę standardową libpng. Plik ten powstaje poprzez nadanie "żywej komórceczy pikselowi o odpowiednich współrzędnych białej barwy.

2.5 Parser poleceń

Program pozwala na interakcję z użytkownikiem za pomocą dostępnej linii poleceń. System pozwala na łatwe dodawanie nowych poleceń za pomocą funkcji *int register_cmd(parser_t parser, char *name, void *help, int (*cmd)(char **, game_t));* , która przyjmuje nazwę polecenie, instrukcję pomocy oraz wskaźnik do wywoływanej funkcji w programie. Specjalnym poleceniem jest *help*, wyświetlające listę zarejestrowanych poleceń oraz ich instrukcje pomocy.

3 Dane wejściowe

3.1 Uruchamianie programu

Program można uruchomić z domyślnymi ustawieniami na systemie Linux za pomocą polecenia *./main* . Program jest uruchamiany ze standardowymi opcjami. Plansza 20x20, zasady Conway-Moor.

3.2 Flagi

Program pozwala na ustawienie pierwszej mapy, zasad czy rozmiaru planszy. Służą do tego odpowiednie flagi. Wprowadzona jest restrykcja na rozmiar mapy, MIN SIZE = 2, MAX SIZE = 999. Warto zaznaczyć, że nie należy łączyć flagi służącej do ładowania mapy oraz ustawienia rozmiaru planszy.

Przykłady:

./main -m spaceship -r conway_neumann - załaduje mapę ze śtatkiem kosmicznym"używając zasad gry według Conwaya i sąsiedztwa Neumanna.

./main –rules highlife_moor –width 20 –height 30 - załaduje mapę o rozmiarze 20x30 używając zasad gry Highlife oraz sąsiedztwa Moore'a.

3.3 Komendy

show_rules	Pokazuje dostepne zasady gry
show_maps	Pokazuje mapy, ktore mozemy wybrac
set_rules	Ustawia <nazwa> zasady
place	Zmienia stan komorki
set_size	Ustawia plansze <wysokosc> and <szerokosc>
next	Przechodzi do kolejnej generacji
n	Skrot od komendy next
play	Pokazuje <ilosc> generacji z <opoznienie> milisekundy
gif	Tworzy animacje z snapow o nazwie <nazwa>
random	Tworzy losowa mape <procent> zageszczenia
snap	Zapisuje biezaca generacje do <nazwa pliku>
clean	Czysci mape
save	Zapisuje mape do <nazwa pliku>
load	Laduje mape <nazwa pliku>
exit	Wylacza program

3.4 Własne zasady gry

Użytkownik może w bardzo łatwy sposób zmodyfikować lub stworzyć własne zasady gry. Mowa tutaj o zmianie ilości sąsiadów dla których komórka umiera się lub rodzi oraz zdefiniowaniu nowego systemu sąsiedztwa (na przykład zaimplementowanie sąsiedztwa Von Neumanna o większym promieniu). W tym celu wystarczy stworzyć plik w katalogu *resources/rules/* z następującą zawartością:

```
name: <nazwa>
live_n: <ilosc wartosci live>
born_n: <ilosc wartosci born>
neighbours_n: <ilosc sasiadow>
live: <ilosc sasiadow dla ktorych komorka pozostaje
zywa (np. 2 3)>
born: <ilosc sasiadow dla ktorych komorka sie rodzi
```

```
(np. 3)>
neighbours: <wzgledne wspolrzedne kolejnych sasiadow
(nizej przyklad)>
-1 -1
0 -1
1 -1
-1 0
1 0
-1 1
0 -1
1 -1
```

4 Dane wyjściowe

4.1 Plansza

Użytkownik może zapisać bieżącą generację planszy to pliku. Służy do tego komenda save <nazwa pliku>.

4.2 Graficzna interpretacja

Do projektu został dołączony generator plików .png. Są dwa rodzaje wykorzystania, możemy użyć komendy snap, aby zapisać bieżącą generację. Drugim sposobem jest wywołanie funkcji play <ilość generacji, które zobaczymy> <opóźnienie konsoli> <nazwa pliku>, pliki będą się zapisywać w postaci <nazwa pliku>-bieżąca_generacja.png

5 Testowanie

Projekt zawiera bardzo prostą implementację testów jednostkowych, pozwalającą na sprawdzenie czy dana funkcja zwraca przewidywaną wartość. Testy można uruchomić za pomocą wprowadzenia "test"jako pierwszego argumentu przy uruchamianiu programu: *./main test* .

Program został przetestowany w różnych, również skrajnych warunkach i nie zostały stwierdzone żadne błędy. Główne środowisko w jakim testowany był program to Linux Debian

6 Dokumentacja

Powyższa dokumentacja została przygotowana za pomocą oprogramowania *Latex*. Pliki dokumentacji znajdują się w folderze doc/. Możliwe jest skompilowanie pliku .tex do .pdf za pomocą polecenia "make doc".