Gra w życie Johna Conwaya - projekt 1

Języki i metody programowania

Jakub Matłacz 307371, Szymon Kuś 307415

data utworzenia: 17 marca 2020
data ostatniej zmiany: 30 marca 2020

1 Wstęp

Zagadnieniem, nad którym miano się pochylić był automat komórkowy, Gra w życie (Game of Life) Johna Conwaya. To świetny przykład tego, jak z kilku prostych zasad może wynikać stosunkowo ogromna złożoność struktur powstałych na ich podstawie. Widzi się analogie między tym niepozornym automatem a rzeczywistością, w której według kwantowej teorii pola (Quantum Field Theory), wszelkie cuda otaczającego nas świata wynikają z superpozycji prostych pól.

2 Zasady gry

Proszę wyobrazić sobie nieskończoną siatkę złożoną z prostych, kwadratowych komórek, których jedyną właściwością jest ich stan: zapalony lub zgaszony. Komórka zgaszona – martwa, zapalona – żywa. Każda komórka ma 8 sąsiadów. Komórki mogą zmieniać swój stan według następujących zasad:

1. Martwa komórka o 3 żywych sąsiadach zapali się w następnym cyklu.
2. Żywa komórka o 2 lub 3 sąsiadach pozostanie żywa w następnym cyklu.
3. Żywa komórka o liczbie sąsiadów różnej od 2 lub 3 zgasi się w następnym cyklu.

Przez stan automatu rozumiemy informację o stanie każdej symulowanej komórki w danym skwantowanym czasie.

3 Opis implementacji

3.1 Opis teoretyczny

Dla uproszczenia obliczeń przyjmujemy skończoną siatkę do gry, gdyż symulacja nieskończonej planszy jest technicznie niemożliwa.

Stworzono tablicę dwuwymiarową o wymiarach r+2 rzędów i c+2 kolumn, gdzie r oraz c określają symulowany obszar, natomiast powiększenie tych wartości o 2 służy zrobieniu miejsca dla „ramki” (tzw. padding), która otacza ten obszar – jest to udogodnienie ułatwiające wdrożenie pomysłu.

Każda komórka jest strukturą zawierającą trzy elementy: stan obecny, stan w przyszłym cyklu oraz informację czy dana komórka jest, czy nie jest „ramką”:

```
typedef struct cell
{
    char state;
    char next_state;
    char is_padding;
}cell_t;
```

Dla oszczędności pamięci użyto typu *char*. Przy dużych rozmiarach symulowanej przestrzeni ma to niemały efekt.

Na podstawie stanu obecnego określamy stan następny automatu. Przechodzimy przez każdą komórkę i sprawdzamy kolejno warunki na narodziny, śmierć i przetrwanie. Kolejność ma tutaj znaczenie i powinna zostać zachowana.

1. Narodziny - do stanu w przyszłym cyklu zapisana jest informacja o zapaleniu komórki.
2. Śmierć – do stanu w przyszłym cyklu zapisana jest informacja o zgaszeniu komórki.
3. Przetrwanie - do stanu w przyszłym cyklu zapisana jest informacja o zapaleniu komórki (lub jeśli ktoś woli, skopiowaniu stanu obecnego – niezmienianiu go).

Tak przykładowo zaimplentowane zostało obsługiwanie przetrwania komórki:

```
int survival(int r, int c, cell_t space[r][c], int i, int j)
{
    if((space[i]+j)->state == ON)
    {
        int friends = count_friends(r,c,space,i,j);
        if( friends == 2 || friends == 3)
        {
            (space[i]+j)->next_state = ON;
            return 1;
        }
    }
    return 0;
}
```

Kluczowe jest w tym momencie efektywne zliczanie żywych sąsiadów, ponieważ jest wykonywane dla każdej komórki w każdym cyklu, tj. funkcja *count_friends*. Jest to funkcja, która na podstawie pozycji danej komórki (opisanej za pomocą dwóch indeksów) wykonuje to zadanie z wykorzystaniem wskaźników dla większej szybkości.

Pod koniec każdego cyklu stanowi obecnemu przypisywany jest stan w przyszłym cyklu.

3.2 Struktura programu

Program podzielono na następujące moduły:

- main.c - funkcja *main*
- service.h - plik biblioteczny zawierający makra, definicję struktury *cell_t* i prototypy funkcji
- prepare_space.c - funkcja inicjująca planszę
- update_space.c - zapisanie nowego stanu komórki do aktualnego i zwolnienie miejsca na kolejny
- load_save.c - wczytywanie i zapisywanie stanu symulacji przy pomocy plików *.life*
- print_png.c - zapisywanie stanu symulacji do pliku *.png*
- print_space.c - wyświetlanie stanu symulacji w konsoli
- life_cycle.c - obliczanie następnego stanu komórki
- count_friends.c - zliczanie sąsiadów komórki
- toys.c - funkcje generujące kilka prostych struktur, używane w przypadku niepodania stanu początkowego .life i używane tylko przez programistę
- Makefile - plik make do automatycznej kompilacji programu podzielonego na wiele modułów
- example.life gliders.life - przykładowe stany początkowe automatu

3.3 Sposób uruchamiania

Program kompilujemy poleceniem *make*. Wywołanie programu odbywa się według następującego schematu:

```
./exec plik_startowy.life liczba_generacji szybkosc_wyswietlania custom_name
    [numer_generacji1] [numer_generacji2] ...
```

1. Pierwszy argument to ścieżka do pliku z rozszerzeniem *.life*, który stanowi zapis początkowej generacji.
2. Drugi argument to liczba generacji do wykonania.
3. Trzeci argument określa szybkość wyświetlania. Przy podaniu 0 przebieg symulacji nie jest wyświetlany w konsoli.
4. Czwarty argument to nazwa folderu w którym zostaną zapisane wszystkie pliki *custom_ name*.
5. Jeśli podany jest piąty lub więcej argumentów, wygenerowane zostaną pliki dla podanych generacji (domyślnie generują się dla wszystkich). Można podać *-1* jako piąty argument, aby program nie generował żadnych plików.

Można pominąć argumenty, ale tylko zaczynając od prawej strony. Spowoduje to ustawienie wartości domyślnych. Możliwe jest na przykład wykonanie:

```
./exec example.life 240 10
```

ale nie wykonanie:

```
./exec 240 10 moja_nazwa 1 2 4
```

gdyż program zinterpretuje *240* jako nazwę pliku startowego, *10* jako liczbę generacji, *moja_ nazwa* jako szybkość wyświetlania itd., co nie da pożądanego efektu.

3.4 Funkcjonalności programu

Program udostępnia szereg funkcjonalności:

1. Zapisywanie kolejnych stanów automatu do plików *.png* - należy zwrócić uwagę, że zapalone komórki oznaczone są kolorem białym, a zgaszone czarnym, przeciwnie do powszechnej konwencji.
2. Zapisywanie kolejnych stanów automatu do plików *.life*, czyli zwykłego pliku tekstowego z opisem wymiarów oraz zerojedynkowym opisem stanu. Możliwe jest dzięki temu łatwe ponowne uruchomienie programu z danym stanem początkowym opisanym w tymże pliku.
3. Przedstawienie kolejnych stanów automatu w formie animacji w terminalu.

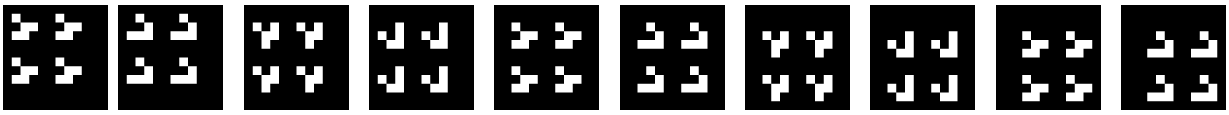
3.5 Struktury danych

Stan planszy przechowywany jest w statycznej tablicy *space*, która tworzona jest na początku działania programu. Zdecydowano się na takie rozwiązanie, aby uprościć działanie programu - nie jest wymagana dynamiczna tablica, gdyż wymiary planszy nie zmieniają się w trakcie symulacji. Struktura tablicy pozwala też na szybkie odwoływanie się do sąsiadów komórki przy użyciu wskaźników.

4 Testy

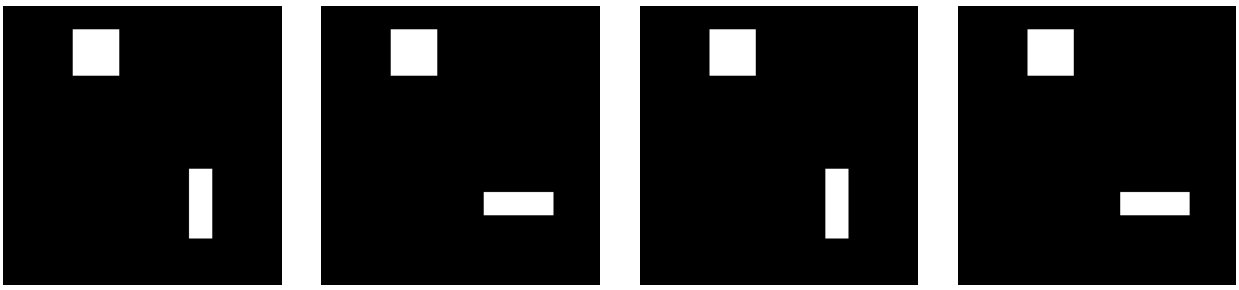
4.1 Wywołanie bez argumentów

Wywołanie programu bez argumentów skutkuje przejściem 100 generacji symulacji, oto pierwsze 10 wygenerowanych plików png:



gen1.png gen2.png gen3.png gen4.png gen5.png gen6.png gen7.png gen8.png gen9.png gen10.png

Od 49. generacji układ stabilizuje się i zaczyna oscylować z okresem 2 (pojawiają się 2 struktury: "Block" i "Blinker", z czego "Block" jest stałą strukturą).



gen49.png gen50.png gen51.png gen52.png

4.2 Błędne argumenty

W przypadku podania błędnego pierwszego argumentu (ścieżka do nieistniejącego pliku) program zachowuje się identycznie, jakby tego argumentu nie było. Przetestowano m.in.:

```
./exec 1234
./exec iofasjdosajdasoi
./exec "dsu#$@!34d$"
./exec chcemiecfajnaplansze
```

W przypadku podania ścieżki, która nie jest prawidłowym plikiem life, program nie uruchamia się lub generuje pustą planszę. Przykładowe przetestowane wywołania:

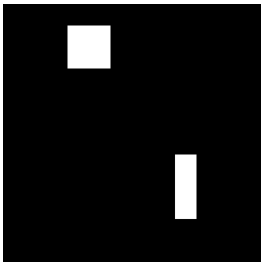
```
./exec main.c
./exec *
./exec ..
```

4.3 Wczytywanie z pliku

Przetestowano również wczytywanie generacji z pliku. Można chociażby wykorzystać generację numer 30 z wywołania bez argumentów i sprawdzić, czy 20 generacji później zobaczymy identyczną parę struktur Block i Blinker:

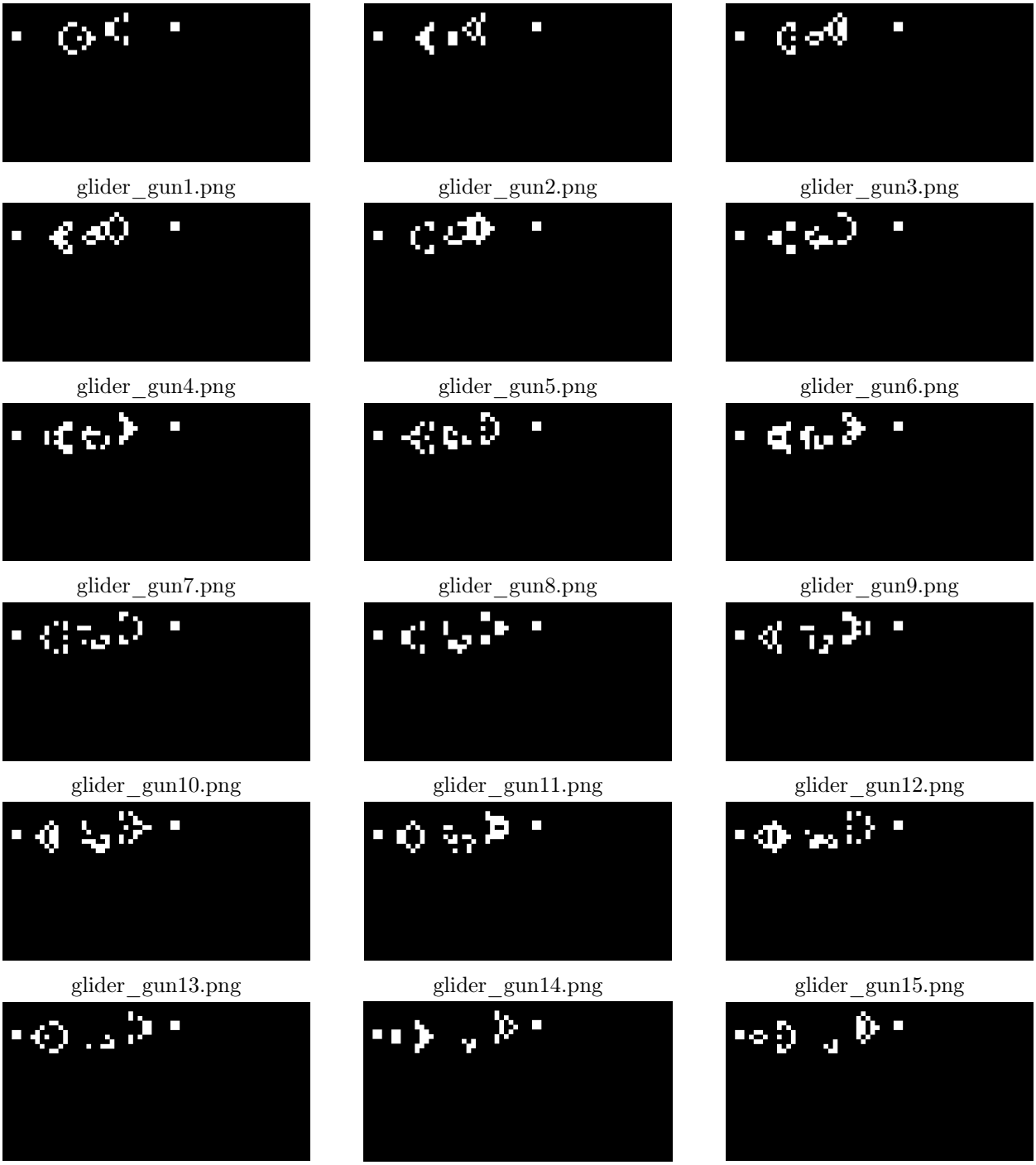
```
./exec
./exec output_states/gen30.life 20 0 input_test 20
```

Otrzymujemy następujący plik w folderze *input_ test*:



Rysunek 1: input_ test20.png

4.4 Przykładowa struktura



glider_gun1.png glider_gun2.png glider_gun3.png
glider_gun4.png glider_gun5.png glider_gun6.png
glider_gun7.png glider_gun8.png glider_gun9.png
glider_gun10.png glider_gun11.png glider_gun12.png
glider_gun13.png glider_gun14.png glider_gun15.png
glider_gun16.png glider_gun17.png glider_gun18.png

Rysunek 2: Glider gun, stan startowy znajduje się w pliku *example.life*

4.5 Wygenerowany plik *.life*

Oto jak prezentuje się zapis stanu do pliku o nazwie glider_gun_80.life dla następującego wywołania programu.

```
./exec example.life 100 0 glider_gun_ 80
```

```
COLUMNS = 64
ROWS = 32

00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000010000000000000000000000000000000000000000
00000000000000000000000000111000000000000000000000000000000000000000
00000000001100000000000001101010001100000000000000000000000000000000
00100000001001000000000001110100100110000000000000000000000000000000
01000110000000100000000000110101000000000000000000000000000000000000
01000001000000100000000000011110000000000000000000000000000000000000
00111110000000100000010000010000000000000000000000000000000000000000
00000000001001000000001000000000000000000000000000000000000000000000
00000000001100000000111000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000010000000000000000000000000000000000000000
00000000000000000000000001010000000000000000000000000000000000000000
00000000000000000000000011000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000010000000000000000000000000000000000000000
00000000000000000000000000100000000000000000000000000000000000000000
00000000000000000000000001110000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
```

4.6 Animacja w konsoli

Przetestowano prezentację i działanie animacji w konsoli Unix dla wywołania ./exec gliders.life 10 5. Poniżej widoczna jest kopia znaków wyświetlonych w stanie numer 10 automatu. Znak '#' oznacza komórkę zapaloną, a znak '_' komórkę zgaszoną.

```
_ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _
_ _ # _ # _ _ # _ # _ _
_ _ _ # # _ _ _ # # _ _
_ _ _ # _ _ _ _ # _ _ _
_ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _
_ _ # _ # _ _ # _ # _ _
_ _ _ # # _ _ _ # # _ _
_ _ _ # _ _ _ _ # _ _ _
_ _ _ _ _ _ _ _ _ _ _ _
```

5 Podsumowanie

Udało nam się udostępnić wszystkie oczekiwane funkcjonalności zaimplementowane w języku C z wygodnym podziałem na moduły.