Specyfikacja funkcjonalna projektu "Gra w życie"

Krzysztof Kalata, Łukasz Laskowski

7 marca 2019

Spis treści

1 Opis ogólny

1.1 Nazwa programu

Program będzie nazywać się “GameOfLife”.

1.2 Poruszany problem

Program nasz będzie swojego rodzaju symulacją życia komórek. Każda komórka znajdować się może w jednym z dwóch stanów - może być żywa lub martwa. Jako że umieszczone są w prostokątnym układzie współrzędnych, każda komórka niepołożona przy brzegu posiada dokładnie 8 sąsiadów. Komórka żywa pozostaje, taką wyłącznie posiadając 2 lub 3 żywe komórki. W każdym innym przypadku komórka umiera z samotności lub zatłoczenia.
Ważną kwestią jest także to, że sąsiedztwo możemy definiować na dwa sposoby – wliczając wszystkie 8 komórek (Moore’a) lub licząc tylko te bezpośrednio stykające się krawędziami z obserwowaną komórką (von Neumanna). Domyślnie używamy pełnego, 8-komórkowego sąsiedztwa. Jeśli komórka była martwa,
ale posiadała dokładnie dwóch lub trzech żywych sąsiadów w jednej iteracji, w kolejnej jej miejsce zajmuje komórka nowonarodzona.

2 Opis funkcjonalności

2.1 Jak korzystać z programu?

Program należy uruchomić z poziomu konsoli, podając jako argumenty ścieżkę do pliku wejściowego i ścieżkę do katalogu wyjściowego. Możliwe jest także załączenie jednej z flag, opisanych punkt niżej.

2.2 Uruchomienie programu

Uruchomienie programu zaczyna się od podania jego nazwy oraz dwóch ścieżek do katalogów - do pliku wejściowego oraz do katalogu, gdzie program ma generować pliki wyjściowe. Następnie użytkownik może (nie jest to obowiązkowe) podać flagi, które odpowiednio zmodyfikują działanie programu. Możliwe flagi:

- ‘-n’ – załączenie tej flagi modyfikuję liczbę wygenerowanych kroków. Definiujemy ich ilość jako liczbę, podaną po fladze. Podając po fladze wartość “-1”, decydujemy się na działanie programu aż do osiągnięcia stanu ustalonego. W przypadku niepodania flagi domyślnie tworzone jest 50 generacji. Generowanie zostanie przerwane w momencie osiągnięcia stanu ustalonego, nawet jeżeli nie została wykonana jeszcze zadeklarowana liczba iteracji.

- ‘-sbs’ – załączenie flagi ‘step by step’ wymusza uruchomienie programu w trybie, w którym decyzję na temat zapisu danej generacji podejmuje użytkownik natychmiast po jej wygenerowaniu. W przypadku niepodania flagi zapisane zostaną wszystkie pojedyncze generacje.

Zatem przykładowe wywołanie programu mogłoby wyglądać tak:
./GameOfLife /home/in.txt /home/out -n 100 -sbs

2.3 Możliwości programu

- Wczytanie siatki początkowej z pliku wejściowego.
- Wygenerowanie zadanej liczby generacji wraz z zapisem każdego kroku w taki sposób, aby mógł być on wczytany jako początek nowej generacji.
- Uruchomienie programu w trybie *step by step*, umożliwiającego zapisywanie na dysk tylko wybranych generacji.
- Obsługa błędów.

3 Format danych i struktura plików

3.1 Struktura katalogów

Opisane poniżej katalogi to domyślna wersja ułożenia plików (w wypadku niezałączenia flag modyfikujących ścieżki wejścia i wyjścia). Cały program zostanie umieszczony w katalogu o nazwie "Game Of Life". W nim znajdują się pliki nagłówkowe (.h) oraz ich implementacje (.c).
Równolegle do plików z kodem znajdują się katalogi 'in' oraz 'out', w których znajdziemy odpowiednio pliki wejściowe (testowe, wygenerowane przez generator.c) oraz wyjściowe.
Każda zapisywana generacja tworzy obraz png oraz odpowiadający jej plik tekstowy, a nazwą jest data i czas jego wygenerowania, którego format odpowiada plikom wejściowym, aby móc uruchomić program od konkretnego kroku.

3.2 Przechowywanie danych w programie

Dane z pliku tekstowego wczytywane są do uprzednio stworzonej tablicy dwuwymiarowej o wymiarach o 2 większej niż zostało to wskazane w pliku wejściowym. Jest to robione w taki sposób, aby móc traktować pierwsze i ostatnie wiersza oraz kolumny jako brzegi, które nie będą widoczne na ostatecznie wygenerowanym obrazie png. Wiersze te oraz kolumny umożliwią aktualizowanie siatki bez konieczności wyodrębniania komórek, znajdujących się przy granicach siatki. Następnie, podczas symulacji, tworzona jest kopia siatki, aby móc bez zakłóceń przeprowadzić przejście do następnej iteracji. Do konwertera trafia jedna z tych tablic, gdzie jest przetwarzana na obraz.

3.3 Dane wejściowe

Zakładamy, że dane wejściowe będą w postaci pliku tekstowego, którego dwie pierwsze liczby określą wymiary (odpowiednio wysokość i szerokość) siatki, a po nich nastąpi ciąg zer i jedynek, jako wypisane stany komórek w kolejnych wierszach. Oczywiście, ustalamy, że cyfra '0' oznaczać będzie komórkę martwą, '1' - żywą.
Dodatkowym założeniem będzie też zabezpieczenie przed dwoma możliwymi błędnymi przypadkami – gdy w pliku wejściowym jest za mało lub za dużo liczb. W pierwszej sytuacji program dopisze w nieokreślone miejsca same zera, w drugim natomiast przeczytane zostanie tyle liczb, ile założono, podając wymiary siatki.

3.4 Dane wyjściowe

Program, po przeanalizowaniu danych wejściowych, generuje dwa typy wyników - w postaci plików png oraz txt. Każdy krok wytwarza po jednym pliku obu rozszerzeń, oba te pliki są wzajemnie równoważne. Plik png jest opisem stanu siatki komórek po iteracji, plik txt jest tym samym opisem, lecz w takiej formie, aby móc podać go jako plik wejściowy przy kolejnym wywołaniu programu. Oba typy plików zapisywane są do katalogu, do którego ścieżkę podano jako drugi argument wywołania programu.

4 Scenariusz działania programu

4.1 Scenariusz ogólny

1. uruchomienie programu
2. analiza argumentów
3. wczytanie pliku wejściowego
4. generacja zadanej liczby obrazów (i odpowiadającym im plików txt)
5. zakończenie działania programu

4.2 Scenariusz szczegółowy

1. uruchomienie programu
2. przeanalizowanie załączonych przy uruchomieniu programu flag
3. sprawdzenie poprawności pliku wejściowego oraz ewentualne obsłużenie błędów
4. pobranie danych z pliku wejściowego
5. przeprowadzenie wybranej ilości iteracji i zapisywanie każdego kroku jako plik png oraz odpowiadający mu plik txt
6. w przypadku załączenia flagi '-sbs', możliwy wybór konkretnych wyników iteracji do zapisania na dysku
7. sprawdzenie różnic między stanem aktualnym oraz poprzednim – w przypadku ich braku, program kończy działanie, wypisując odpowiedni komunikat na *stdout*
8. zamknięcie uchwytów do plików i zakończenie działania programu

5 Testowanie

5.1 Ogólny przebieg

Testy przeprowadzimy na trzech poziomach:

- korzystając z generatora siatek losowych, napisanego przez nas, który umieścimy w katalogu ‘in’
- ręcznie tworząc testy przypadków brzegowych, czyli potencjalnie niebezpiecznych, błędogennych
- przetestowanie programu na paru wygenerowanych losowo testach, mających na celu sprawdzenie kulturę zarządzania pamięcią (np programem *Valgrind*)