SPECYFIKACJA FUNKCJONALNA "Gra w życie" DLA JĘZYKA PROGROMOWANIA C

Wykonali:
1. Ivan Prakapets

Spis treści

1 Opis Ogólny **2**

1.1 Nazwa programu 2

1.2 Poruszany problem 2

2 Opis funkcjonalności **3**

2.1 Możliwości programu 3

2.2 Jak korzystać z programu? 3

2.3 Uruchomienie programu 4

3 Format danych i struktura katalogów **5**

3.1 Struktura katalogów 5

3.2 Dane wejściowe 5

3.3 Dane wyjściowe 5

4 Obsługa sytuacji błędnych **6**

5 Testowanie **7**

1 Opis Ogólny

1.1 Nazwa programu

Nazwa programu: `gra_w_zycie`.

1.2 Poruszany problem

Zbudowanie programu automatu komórkowego w języku C. Będzie on przedstawiał działanie automatu na podstawie zasad Gry w życie z angielskiego "Game of life" autorstwa Johna Conwaya.

Gra w życie składa się z komórek martwych i żywych. Stan wszystkich komórek nazywamy genereacją. Komórki mogą zmieniać swój stan w zależności od sąsiedztwa, dokładnie od liczby żywych sąsiadów i obecnego stanu.

Zestaw zasad przy tworzeniu nowej generacji jest następujący:

- martwa komórka, która ma dokładnie 3 żywych sąsiadów, staje się żywa w następnej jednostce czasu (rodzi się),
- żywa komórka z 2 albo 3 żywymi sąsiadami pozostaje nadal żywa; przy innej liczbie sąsiadów umiera (z "samotności" albo "zatłoczenia").

Są dwa rodzaje sąsiedztw Moore'a i von Neumanna. W sąsiedztwie Moore'a mamy 8 przylegających komórek (znajdujących się: na południu, na południowym-zachodzie, na zachodzie, na północnym-zachodzie, na północy, na północnym-wschodzie, na wschodzie i na południowym-wschodzie) oraz w sąsiedztwie von Neumanna 4 przylegających komórek (na południu, zachodzie, północy i wschodzie).

2 Opis funkcjonalności

2.1 Możliwości programu

Program będzie zawierał następujące możliwości:

1. Wczytywanie danych wejściowych zadanych jako argumenty.
2. Wczytywanie pierwszej generacji z pliku tekstowego, składającego się z zer i jedynek, gdzie zero oznacza komórkę martwą, a jedynka komórkę żywą.
3. Wczytywanie pierwszej generacji z pliku graficznego, gdzie piksel czarny oznacza komórkę żywą, a piksel biały oznacza komórkę martwą.
4. Użytkownik ma możliwość wyboru algorytmu rozwiązania (Moore'a lub von Neumanna).
5. Użytkownik ma możliwość wyboru ilości symulacji kolejnych generacji komórek.
6. Zapisanie stanów do plików graficznych PNG oraz tworzenie pliku GIF przedstawijącego zmiany w kolejnych generacjach.
7. Sprawdzanie poprawności pliku wejściowego i podanych argumentów.
8. Obsługa różnych błędnych danych.

2.2 Jak korzystać z programu?

Program nie ma interfejsu graficznego. Z tego powodu skompilowany program musi być uruchomiony z konsoli. Wszystkie ustawienia pro-

gramu podawane są jako kolejne argumenty.

Lista argumentów do wywołania programu:

1. Ścieżka do pliku z danymi opisującymi pierwszą generację. Program wczytuje formaty: `*.txt, *.png.`
2. Liczbę symulowanych generacji.
3. Początek nazwy plików wyjściowych.
4. Tryb uruchomienia (`-m` - Moore'a, `-n` - von Neumanna).

2.3 Uruchomienie programu

Przykład wywołania programu:

```
./gra_w_zycie obrazek.png out 100 -m
```

3 Format danych i struktura katalogów

3.1 Struktura katalogów

Gra w życie będzie zawierała kilka katalogów, w katalogie głównym będzie plik wywołania programu. Podkatalog ''gra_w_zycie'' będzie zawierał pliki źródłowe i nagłówkowe. Podkatalog ''dane'' będzie zawierał pliki *.txt oraz *.png. Podkatalog ''wynik'' będzie zawierał obrazy w formacie PNG i GIF oraz pliki tekstowe z wybraną generacją.

3.2 Dane wejściowe

Program Gra w życie otrzymuje dane wejściowe. Najpierw plik tekstowy *.txt lub plik graficzny *.png. Plik tekstowy składa się z zer i jedynek, gdzie zero oznacza komórkę martwą, a jedynka komórkę żywą. Plik graficzny składa się z pikseli, gdzie piksel biały oznacza komórkę martwą, a piskel czarny oznacza komórkę żywą.

3.3 Dane wyjściowe

W wyniku wywołania programu, pierwsza generacja zostanie wczytana z pliku *.png lub *.txt. Zostaną wygenerowane pliki w formacie PNG zawieracjące generacje symulacji, o nazwach, out_[nr-generacji].png oraz plik GIF o nazwie out.gif.

4 Obsługa sytuacji błędnych

Program będzie obsługiwał błędne dane z odpowiednimi komunikatami:

* *Nieprawidłowa liczba argumentów.*
* *Niepoprawny argument.*
* *Plik wejściowy nie istnieje.*
* *Plik wejściowy zawiera niepoprawne dane.*
* *Tryb nie został wybrany.*

5 Testowanie

Do przetestowania kodu będę używał kompilator ISO C99 z poleceniem `-Wall`, a także będę używał takiego programu jak `''Valgrind''` dla sprawdzania wycieków pamięci. Dla stworzenia plików wejściowych PNG będę używał programu `''GNU Image Manipulation Program''`.