

Specyfikacja implementacyjna programu lifeGameEmulator

Ciszewski Jakub, Rahachevich Aliaksandra

9 Marzec 2021

Spis treści

1	Informacje ogólne	3
2	Opis modułów	3
2.1	fileOperation	3
2.1.1	table* readFromFile(FILE* file)	3
2.1.2	void saveToFile(table* Table, char* outFileName, int to- Txt, int toPicture)	3
2.2	solver	4
2.2.1	int solveIteration(table* Table, int typeOfProximity, int typeOfArea)	4
2.3	proximity	4
2.3.1	int numberOfNeighbours(table* Table, int x, int y, int typeOfProximity, int typeOfArea)	4
2.4	tableOperation	5
2.4.1	table* initTable(int columns, int rows)	5
2.4.2	void destroyTable(table* Table)	5
2.4.3	int compareTable(table* Table1, table* Table2)	5
2.5	main	6
3	Testowanie	6
3.1	Test poprawności działania algorytmu	6
3.2	Test poprawności wczytywania i zapisywania danych	6
3.3	Test poprawności walidowania argumentów wywołania	6
3.4	Test działania całego programu	6
4	Diagram modułów	7

1 Informacje ogólne

Zakłada się, że program lifeGameEmulator:

- Zostanie napisany w języku C z użyciem modułów biblioteki standardowej (np. `stdio.h`, `stdlib.h`, `string.h`,...) oraz samodzielnie napisanych funkcji.
- Tworzony jest do działania w systemie z rodziny Linux w trybie konsolowym

2 Opis modułów

Zakłada się 5 modułów aplikacji:

2.1 fileOperation

Moduł odpowiedzialny za operacje na plikach.
Udostępnia on następujące funkcje:

2.1.1 `table* readFromFile(FILE* file)`

Funkcja odpowiedzialna jest za walidację danych znajdujących się w pliku, oraz stworzeniu na ich podstawie planszy.

Funkcja przyjmuje argument:

- ***file*** typu `FILE*`. Wskaźnik na plik zawierający strukturę wejściową.

Funkcja zwraca wskaźnik na zainicjalizowaną strukturę typu `table*`. W przypadku, gdy dane zawarte w pliku nie pozwalają na poprawną inicjalizację zwracany jest wskaźnik `null`.

2.1.2 `void saveToFile(table* Table, char* outFileName, int toTxt, int toPicture)`

Funkcja odpowiedzialna jest za zapis planszy do pliku. Plansza może zostać zapisana do pliku tekstowego (W formacie jaki akceptuje program jako plik wejściowy) oraz formacie graficznym. Funkcja sprawdza, czy plik o danej nazwie nie istnieje. W takim przypadku do nazwy pliku dopisywana jest liczba.

Funkcja przyjmuje argumenty:

- ***Table*** typu `table*`. Wskaźnik na strukturę, która ma zostać zapisana do pliku.
- ***outFileName*** typu `char*`. Nazwa jaką ma przyjąć zapisany plik.

- ***toText*** typu int. Określa czy struktura ma zostać zapisana do pliku tekstowego (wartość 1), czy nie ma być zapisana w tym formacie (wartość 0).
- ***toPicture*** typu int. Określa czy struktura ma zostać zapisana do pliku graficznego (wartość 1), czy nie ma być zapisana w tym formacie (wartość 0).

Funkcja nie zwraca żadnej wartości.

2.2 solver

Moduł odpowiedzialny za przeprowadzenie pojedynczej iteracji gry. Udostępnia on funkcje:

2.2.1 int solveIteration(table* Table, int typeOfProximity, int typeOfArea)

Funkcja przeprowadzająca pojedynczą iterację programu.

Funkcja przyjmuje argumenty:

- ***Table*** typu table*. Wskaźnik na strukturę, na której ma zostać wykonana pojedyncza iteracja gry.
- ***typeOfProximity*** typu int. Liczba określająca rodzaj sąsiedztwa jakie ma zostać zastosowane. Sąsiedztwo Moore-a (wartość 0) czy von Neumanna (wartość 1).
- ***typeOfArea*** typu int. Liczba określająca czy plansza ma zostać "zakończona" (wartość 0), czy być rozpatrywana jako płaska (wartość 1).

Funkcja zwraca liczbę 0 w przypadku, gdy w wyniku zajęcia iteracji nie doszło do żadnej zmiany względem planszy wejściowej. W przeciwnym wypadku zwracana jest wartość 1.

2.3 proximity

Moduł odpowiedzialny za obliczanie elementów żywych sąsiadujących z daną komórką.

Udostępnia on funkcje:

2.3.1 int numberOfNeighbours(table* Table, int x, int y, int typeOfProximity, int typeOfArea)

Funkcja zwraca ilość żywych elementów sąsiadujących z elementem według typu sąsiedztwa i typu planszy.

Funkcja przyjmuje argumenty:

- ***Table*** typu table*. Wskaźnik na strukturę, na której znajduje się element dla którego określone jest sąsiedztwo.

- ***x*** typu int. Liczba określająca kolumnę, w której znajduje się element.
- ***y*** typu int. Liczba określająca wiersz w którym znajduje się element.
- ***typeOfProximity*** typu int. Liczba określająca rodzaj sąsiedztwa jakie ma zostać zastosowane. Sąsiedztwo Moore-a(wartość 0) czy von Neumanna(wartość 1).
- ***typeOfArea*** typu int. Liczba określająca czy plansza ma zostać "zaokrąglona"(wartość 0), czy być rozpatrywana jako płaska(wartość 1).

Funkcja zwraca liczbę określającą ilość żywych sąsiadów, jaką posiada element o danych współrzędnych.

2.4 tableOperation

Moduł odpowiedzialny za operacje na planszy. Znajduje się w nim definicja struktury table.

Udostępnia on funkcje:

2.4.1 table* initTable(int columns, int rows)

Funkcja inicjalizuje planszę o podanej liczbie kolumn i wierszy i zwraca wskaźnik na tę strukturę.

Funkcja przyjmuje argumenty:

- ***columns*** typu int. Liczba oznaczająca ilość kolumn jaka ma zostać zainicjalizowana.
- ***rows*** typu int. Liczba oznaczająca ilość kolumn jaka ma zasotać zainicjalizowana.

Funkcja zwraca wskaźnik table* na zainicjalizowaną strukturę.

2.4.2 void destroyTable(table* Table)

Funkcja wymazuje z pamięci planszę.

Funkcja przyjmuje argumenty:

- ***Table*** typu table*.Wskaźnik na strukturę, która ma zostać wymazana z pamięci.

Funkcja nie zwraca żadnej wartości.

2.4.3 int compareTable(table* Table1, table* Table2)

Funkcja porównująca dwie struktury typu table. Funkcja przyjmuje argumenty:

- ***Table1*** i ***Table2*** typu table*. Wskaźniki na struktury, które mają zostać porównane.

Funkcja zwraca 1, gdy struktury mają różną zawartość. W innym przypadku zwracane jest 0.

2.5 main

Moduł odpowiedzialny za walidację danych oraz główną logikę programu. Moduł nie udostępnia żadnych funkcji.

3 Testowanie

Program zostanie przetestowany pod czterema różnymi względami. W przypadku wykrycia jakichkolwiek nieprawidłowości, część programu za nie odpowiedzialna zostanie skierowana do poprawy. Testy przeprowadzane są w sposób automatyczny, poprzez napisane programy testujące, które temu służą.

3.1 Test poprawności działania algorytmu

Algorytm rozwiązywania zagadnienia gry w życie zostanie przetestowany pod względem poprawności działania. W tym celu przygotowane zostaną plansze gry o losowym rozmiarze i rozkładzie komórek żywych i martwych. Wyniki testów zostaną zebrane i przeanalizowane.

3.2 Test poprawności wczytywania i zapisywania danych

Test zostanie przeprowadzony poprzez wczytywanie przygotowanych plików zawierających struktury opisane poprawnie oraz błędnie, oraz ich zapis do plików o formacie tekstowym i graficznym. Pliki zostaną przeanalizowane biorąc pod uwagę ich zawartość.

3.3 Test poprawności walidowania argumentów wywołania

Program zostanie sprawdzony pod względem odczytu argumentów wywołania oraz ich wartości. Sprawdzona zostanie poprawność wybierania odpowiednich wartości dla podanych argumentów oraz wyświetlanie komunikatów błędów.

3.4 Test działania całego programu

Sprawdzone zostaje zachowanie się programu podczas jego użytkowania. Sprawdzona zostanie poprawność działania programu względem wywoływanych argumentów oraz otrzymywanych wyników.

4 Diagram modułów

Diagram przedstawiający moduły wraz z funkcjami, które są przez nie wywoływane:

