Specyfikacja implementacyjna programu lifeGameEmulator

Ciszewski Jakub, Rahachevich Aliaksandra

9 Marzec 2021

Spis treści

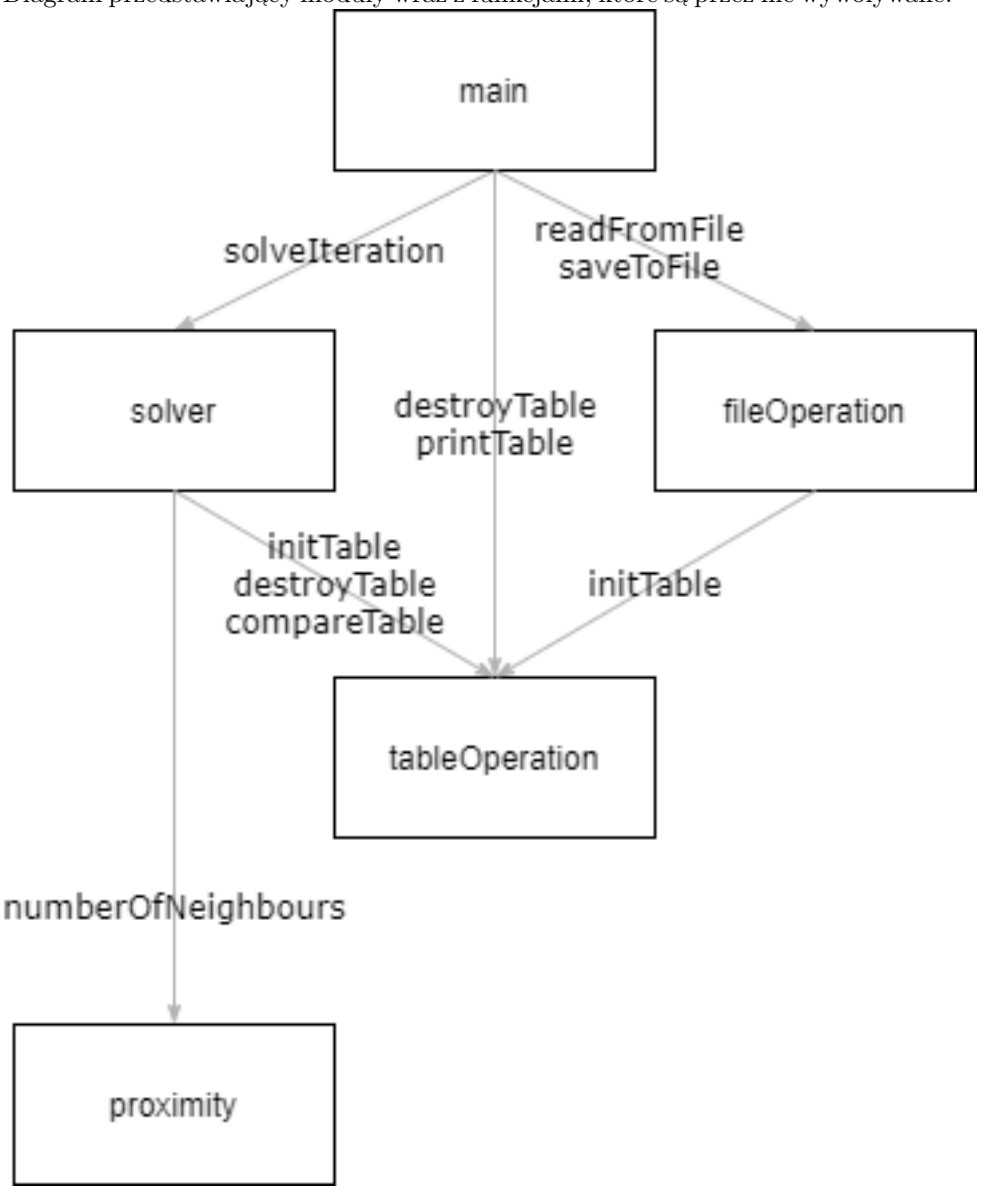
1 Informacje ogólne

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 Diagram modułów

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



3 Opis modułów

Zakłada się 5 modułów aplikacji:

3.1 fileOperation

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

3.1.1 table* readFromFile(FILE* file)

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

Funkcja przyjmuje argument:

- ***file*** typu FILE*. Wskaźnik na plik zawierający strukturę wejsciową.

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

3.1.2 void saveToFile(table* gameTable,char* outFileName, int toTxt, int toPicture)

Funkcja odowiedzialna 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:

- ***gameTable*** typu table*. Wskaźnik na strukturę, która ma zostać zapisana do pliku.
- ***outFileName*** typu char*. Nazwa jaką ma przyjąc zapisany plik.
- ***toTxt*** 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.

3.2 solver

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

3.2.1 int solveIteration(table* gameTable, int typeOfProximity, int typeOfArea)

Funkcja przeprowadzająca pojedynczą iteracje programu.
Funkcja przyjmuje argumenty:

- ***gameTable*** 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 "zaokrąglona"(wartość 0), czy być rozpatrywana jako płaska(wartość 1).

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

3.3 proximity

Moduł odpowiedzialny za obliczanie elementów żywych sąsiadujących z daną komórką.
Udostępnia on funkcje:

3.3.1 int numberOfNeighbours(table* gameTable, 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:

- ***gameTable*** typu table*. Wskaźnik na strukturę, na której znajduje się element dla którego określane jest sąsiedztwo.
- ***x*** typu int. Liczba określająca kolumne, 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.

3.4 tableOperation

Moduł odpowiedzialny za operacje na planszy. Znajduje się w nim definicja struktury table.
Struktura table posiada trzy wartości.

- colums typu int. Liczba określająca ilość kolumn.
- rows typu int. Liczba określająca ilość wierszy.
- board typu char**. Wskaźnik na wskaźnik na pojedynczy element znajdujący się na planszy.

Moduł udostępnia funkcje:

3.4.1 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.

3.4.2 void destroyTable(table* gameTable)

Funkcja wymazuje z pamięci planszę.
Funkcja przyjmuje argumenty:

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

Funkcja nie zwraca żadnej wartości.

3.4.3 int compareTable(table* gameTable1, table* gameTable2)

Funkcja porównująca dwie struktury typu table. Funkcja przyjmuje argumenty:

- ***gameTable1*** i ***gameTable2*** typu table*. Wskaźniki na struktury, które mają zostać porównane.

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

3.4.4 void printTable(table* gameTable)

Funkcja odpowiedzialna jest za wyświetlanie planszy w konsoli. Funkcja przyjmuje argumenty:

- ***gameTable*** typu table*. Wskaźnik na strukturę planszy, która ma zostać wyświetlona.

Funkcja nie zwraca żadnej wartości.

3.5 main

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

4 Testowanie

Program zostanie przetestowany pod pięcioma różnymi względami. W przypadku wykrycia jakichkolwiek nieprawidłowości, część programu za nie odpowiedzalna zostanie skierowana do poprawy. Testy przeprowadzone zostaną w sposób automatyczny.

4.1 Test poprawności działania algorytmu

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

4.2 Test poprawności wczytywania danych

Test zostanie przeprowadzony poprzez wczytywanie przygotowanych plików zawierających struktury opisane poprawnie oraz błędnie. Pliki zostaną przeanalizowane biorąc pod uwagę ich zawartość.

4.3 Test poprawności zapisywania danych

Zostanie sprawdzona poprawność zapisywania wyników do plików o formacie tekstowym i graficznym. Wyniki testów będą zebrane i przeanalizowane.

4.4 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 błedów.

4.5 Test działania całego programu

Sprawdzone zostanie zachowanie programu podczas jego użytkowania. Sprawdzona zostanie poprawność działania programu, względem wywoływanych argumentów, a także otrzymywanych wyników.