Specyfikacja implementacyjna programu WireWorld *Life*

Krzysztof Maciejewski
Hubert Kunikowski

9 czerwca 2019

Spis treści

1 Diagram modułów **2**

2 Opis modułów **2**

3 Opis przepływu sterowania **4**

4 Opis głównych algorytmów **5**

5 Testy **5**

1 Diagram modułów

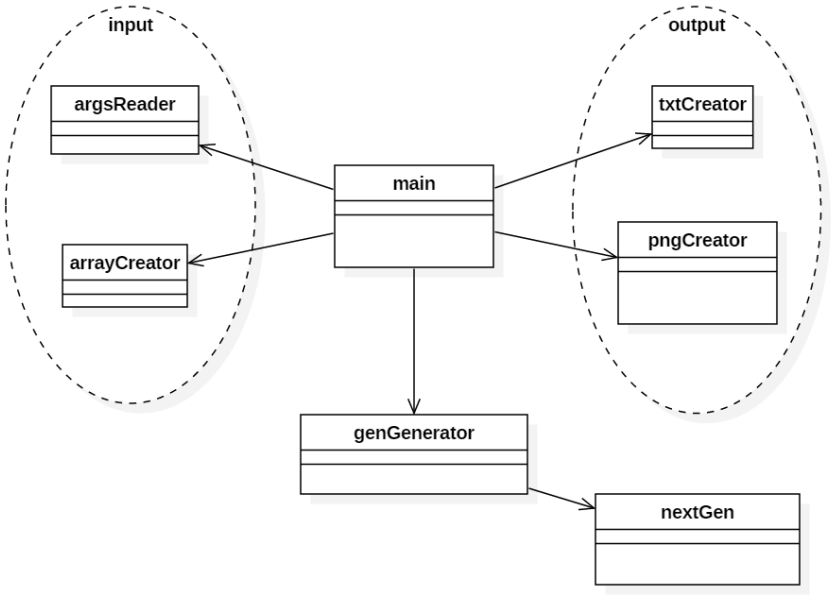
Program składa się łącznie z 7 modułów.

Input programu to moduły:

- *argsReader* - sczytuje argumenty wywołania
- *arrayCreator* - tworzy tablicę odpowiadającą generacji początkowej na podstawie pliku z danymi.

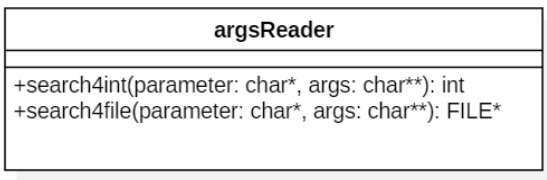
Output programu to moduły:

- *txtCreator* - tworzy plik TXT dla danej generacji komórek
- *pngCreator* - tworzy plik PNG dla danej generacji komórek



2 Opis modułów

1. **Moduł argsReader**



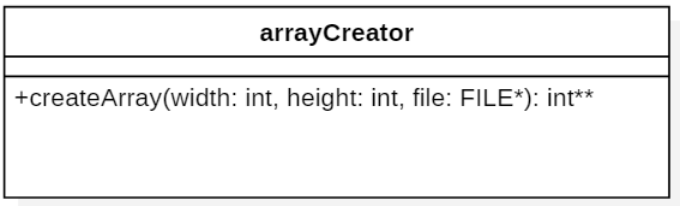
Moduł *argsReader* służy do odczytywania argumentów wywołania programu. Zawiera dwie funkcje: jedna służy do odczytywania plików z argumentów wywołania, a druga do odczytywania wartości typu *int*. Obie funkcje pobierają dwuwymiarową tablicę znaków, czyli argumenty wywołania, oraz, który określa, którego argumentu szukamy. Przykładowo:

```
int width = search4int("-w", args);
```

znajdzie nam wysokość wśród argumentów wywołania.

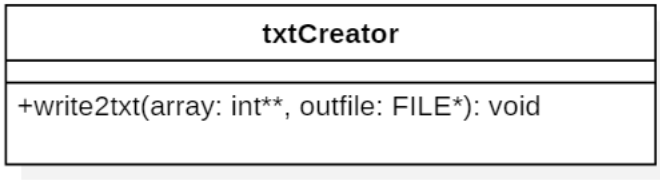
Jeżeli funkcje nie znajdą argumentu, zwrócą wartości domyślne.

2. **Moduł arrayCreator**



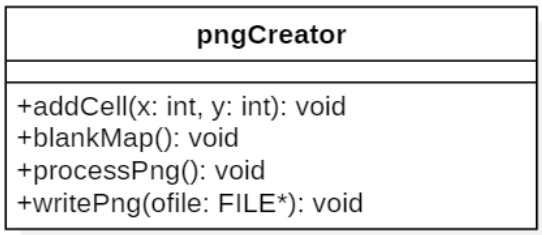
Moduł *arrayCreator* odpowiada za stworzenie tablicy przechowującej daną generację. Funkcja *createArray* tworzy dwuwymiarową tablicę na podstawie współrzędnych komórek zawartych w pliku wejściowym. Tablica będzie składała się z wartości 0 (komórki martwe) i 1 (komórki żywe). Jeżeli przy odczycie z pliku napotka nieprawidłowe dane, zwróci *NULL*.

3. **Moduł txtCreator**



Moduł *txtCreator* będzie służył w programie do zapisu stanu ostatniej generacji komórek do pliku TXT w postaci współrzędnych żywych komórek. Realizuje to funkcja *write2txt*, która jako argumenty przyjmuje tablicę przechowującą generację oraz plik do zapisu.

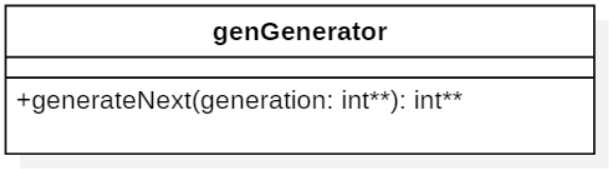
4. **Moduł pngCreator**



Moduł *pngCreator* służy do generowania plików PNG dla danej generacji komórek. Funkcja *processPng* służy do stworzenia tablicy określającej kolory danych pól. Określonym polom zostaną przypisane odpowiednie kolory przy pomocy funkcji *blankMap* oraz *addCell*.

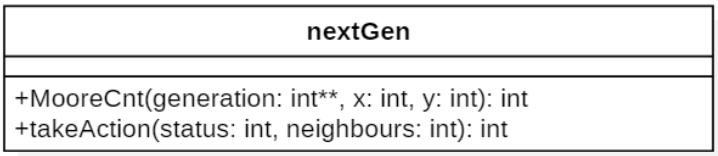
Funkcja *writePng* wykorzysta bibliotekę *libpng* do wygenerowania pliku PNG na podstawie tablicy kolorów.

5. **Moduł genGenerator**



Moduł *genGenerator* odpowiada za tworzenie kolejnych generacji komórek. Jako argument przyjmuję tablicę przechowującą aktualną generację i zwraca nową tablicę z nową generacją.

6. **Moduł nextGen**



Moduł *nextGen* określa na jakich warunkach tworzone są następne generacje.

Funkcja *MooreCount* zlicza ile jest żywych sąsiadów komórki o współrzędnych x i y i zwarca tę wartość.

Funkcja *takeAction* przyjmuje jako argumenty status danej komórki (żywa lub martwa) oraz liczbę jej żywych sąsiadów i zwraca status tej komórki dla następnej generacji (0 lub 1).

Moduł ten można łatwo wymienić, co pozwala na manipulację działaniem automatu komórkowego (np. zmiana sąsiedztwa).

3 Opis przepływu sterowania

1. Wczytanie argumentów wywołania
2. Utworzenie tablicy na podstawie pliku wejściowego
3. Ewentualne wygenerowanie pliku PNG
4. Rozpoczęcie generowania kolejnych generacji
   - 4.1 Iteracja po komórkach tablicy
     - 4.1.1 Ewentualna zmiana stanu komórek
   - 4.2 Ewentualne wygenerowanie pliku PNG
   - 4.3 Sprawdzenie czy powinniśmy wygenerować kolejną generację
5. Ewentualne wygenerowanie pliku TXT
6. Zakończenie działania programu

4 Opis głównych algorytmów

Zmiana stanu komórek

Najważniejszym algorytmem w programie jest konwersja komórek żywych na martwe i vice versa, która pozwala na generowanie kolejnych generacji. Konwersja ta odbywa się dwuetapowo - najpierw korzystamy z funkcji *MooreCnt*. Sprawdza ona sąsiedztwo Moore'a danego punktu. Każda komórka tablicy w sąsiedztwie zawierająca 1 jest podliczana i w rezultacie zwracana jest suma żywych komórek w sąsiedztwie. Następnie użyta jest funkcja *takeAction*. Pobiera ona status danej komórki oraz liczbę jej żywych sąsiadów. Zgodnie z ustalonymi zasadami gry, funkcja zwraca nowy status tej komórki - 0 lub 1. W ten sposób iterujemy po wszystkich komórkach tablicy, generując przy tym nową tablicę, będącą kolejną generacją.

5 Testy

1. **Test modułu *argsReader***

   W ramach testu tego modułu utworzymy prosty program main. Będzie on wywoływał funkcje modułu *argsReader* dla kolejnych parametrów, a następnie wypisywał je na ekran w określonej kolejność. Wypisane wartości zostaną porównane z podanymi argumentami.

   Przykładowe test:

   Argumenty wywołania:

   ```
   ./a.out -f plik1.txt -w 30 -n 5 -o plik2.txt
   ```

   Oczekiwane wyniki:

   ```
   Nazwa pliku wejściowego: plik1
   Szerokość planszy: 30
   Wysokość planszy: 30
   Liczba przeprowadzonych generacji: 5
   Nazwa wyjściowego pliku tekstowego: plik2.txt
   Liczba wygenerowanych plików PNG: 2
   ```

   Sprawdzane będzie również zachowanie działania funkcji w przypadku podania błędnej wartości argumentu oraz nie podania żadnego argumentu.

2. **Testy modułu *arrayCreator***

   Testy tego modułu będą polegały na utworzeniu prostego main'a, w którym otworzymy plik do odczytu o konkretnej nazwie (np.: "plik"). Następnie wywołamy funkcję *createArray*, której przekażemy ten plik oraz ustalone wartości szerokości i wysokości planszy. Na końcu programu main utworzymy pętlę wypisującą tablicę na stdin. Prócz tego utworzymy wspominany plik tekstowy z określonymi danymi. Dane podane w pliku tekstowym oraz wyświetlane na ekran będą porównywane.

   Przykładowy test:

   Zawartość pliku tekstowego:

```
1 1
1 3
2 3
2 4
3 3
4 2
4 3
```

Wywołanie funkcji *createArray*:

```
tab = arrayCreator(4, 4, fopen("plik.txt","r");
```

Pożądany wynik na stdout:

```
1 0 1 0
0 0 1 1
0 0 1 0
0 1 1 0
```

Sprawdzane będzie również zachowanie funkcji *createArray* dla błędnych argumentów w pliku.

3. **Test modułu *txtCreator***

   Kiedy testy modułu *arrayCreator* przebiegną pomyślnie, wykorzystamy kod tego testu do przetestowania modułu *txtCreator*. Funkcji *write2txt* przekażemy jako argument stworzoną wcześniej tablicę dwuwymiarową. Następnie sprawdzimy czy zawartość pliku wejściowego i wyjściowego są równoważne.

   Przykładowy test:

   Wejściowy plik:

```
1     1
1  3
2     3
2   4
3  3
4  2
4 3
```

   Oczekiwany wyjściowy plik:

```
1 1
1 3
2 3
2 4
3 3
4 2
4 3
```

4. **Test modułu *pngCreator***

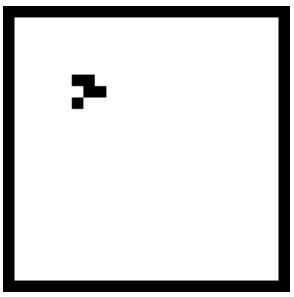
   Test modułu *pngCreator* również przeprowadzimy w oparciu o test modułu *arrayCreator*. W tym przypadku wykorzystamy wczytaną tablicę, aby wygenerować na jej podstawie plik PNG. Następnie sprawdzimy czy współrzędne komórek na obrazku zgadzają się z tymi podanymi w pliku wejściowym.

   Przykładowy test:

   Plik wejściowy:

   ```
   6 6
   6 7
   7 7
   7 8
   8 6
   ```

   Oczekiwany plik wyjściowy:

5. **Test modułu *nextGen***

   Do testów modułu *nextGen* zdefiniujemy sobie dwuwymiarową tablicę typu *int*. Następnie, przy pomocy funkcji *MooreCnt* zliczymy sąsiadów komórek o poszczególnych indeksach i wyświetlimy tę liczbę na ekran. Obok ilości sąsiadów każdej komórki wyświetlimy także wartość komórki o tym indeksie w następnej generacji, używając do tego funkcji *takeAction*. W ten sposób sprawdzimy czy generowane są poprawne wartości.

   Przykładowy test:

   Dwuwymiarowa tablica typu *int*:

   ```
   int tab[3][3] = {{1,1,0},{0,1,1},{0,0,1}};
   ```

   Wywołanie funkcji:

   ```
   int a = MooreCnt(tab, 1, 1);
   int b = takeAction(tab[1][1], a);
   printf("%d - %d\n", a, b);
   ```

   Oczekiwany wynik:

   ```
   4 - 0
   ```

6. **Test modułu *genGenerator***

   Do przetestowania funkcji *generateNext* z modułu *genGeneration* stworzymy dwuwymiarową tablicę typu *int*, do której wpiszemy dane o generacji i wypiszemy tę tablicę na ekran. Następnie wywołamy funkcję *genNext* dla tej tablicy, która utworzy generację potomną. Nową generację również wypiszemy na ekran, co pozwoli na wygodne sprawdzenie poprawności wyników.

   Przykładowy test:

   Dwuwymiarowa tablica typu *int*:

   ```
   int tab[3][3] = {{1,1,0},{0,1,1},{0,0,1}};
   ```

   Wywołanie funkcji:

   ```
   tab = generateNext(tab);
   ```

   Oczekiwane wyniki:

   ```
   Generacja 1:
   1 1 0
   0 1 1
   0 0 1

   Generacja 2:
   1 1 1
   1 0 1
   0 1 1
   ```