Raport końcowy implementacji automatu komórkowego *Life*

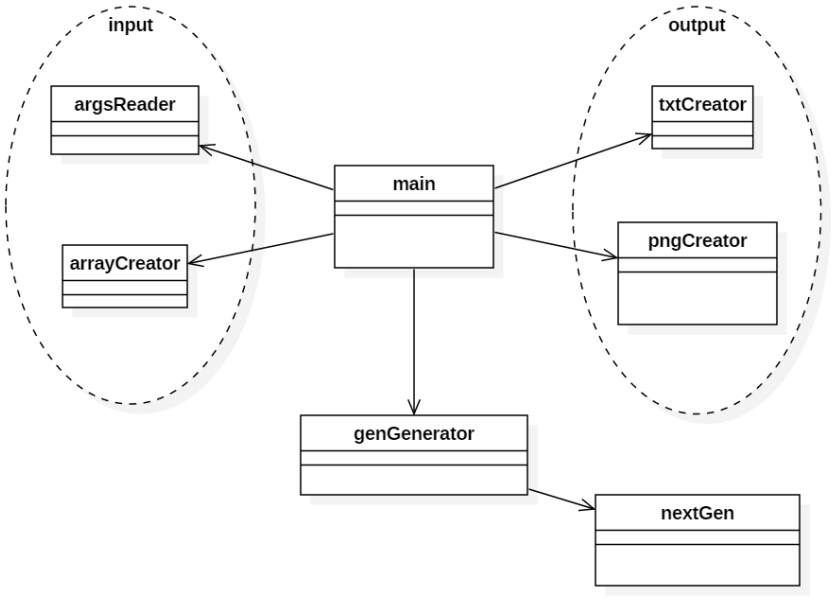
Krzysztof Maciejewski
Hubert Kunikowski

9 czerwca 2019

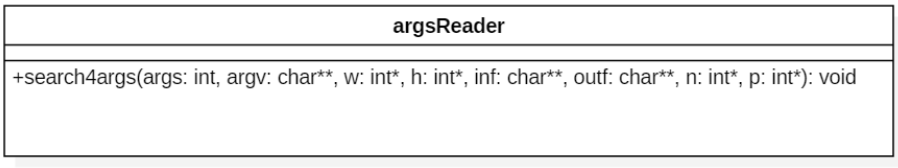
Spis treści

1 Ostateczny projekt modułów

Program składa się łącznie z 7 modułów.

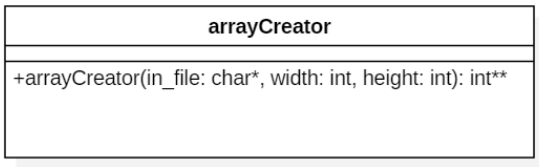


1. **Moduł argsReader**



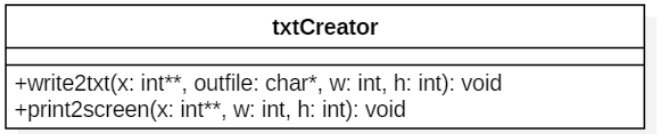
Moduł *argsReader* służy do odczytywania argumentów wywołania programu. Zawiera jedną funkcję *search4args*, która przyjmuje wskaźniki na zmienne oraz argumenty wywołania programu i ustawia wartości zmiennych zgodnie z podanymi argumentami wywołania. Plik nagłówkowy *argsReader.h* zawiera również wartości domyślne parametrów takich jak wysokość czy szerokość planszy.

2. **Moduł arrayCreator**



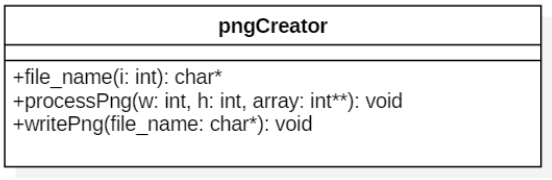
Moduł ten odpowiada za stworzenie tablicy przechowującej daną generację. Funkcja *arrayCreator* tworzy dwuwymiarową tablicę o określonych wymiarach na podstawie danych z pliku, którego nazwę podano jako argument.

3. **Moduł txtCreator**



Na moduł *txtCreator* składają się dwie funkcje. Funkcja *write2txt* pobiera tablicę przechowującą generację i zapisuje generację w postaci współrzędnych żywych komórek do pliku TXT. Druga funkcja, *print2screen* wyświetla generację w formie ACII art na ekran. Nie została ona użyta w głownym programie, natomiast skorzystano z niej przy testach.

4. **Moduł pngCreator**



Moduł ten wykorzystuje bibliotekę *libpng*. Plik nagłówkowy zawiera zmienne potrzebne do działania funkcji tej biblioteki, między innymi tablice przechowujące wartości odpowiadające odpowiednim kolorom.

Funkcja *processPng*, na podstawie tablicy przechowującej generację, uzupełnia tablice kolorów.

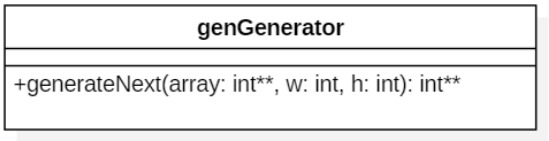
Funkcja *writePng* wykorzystuje funkcje biblioteki *libpng* do stworzenia pliku PNG.

Funkcja *file_name* służy do generowania nazw plików PNG dla kolejnych generacji (*genA.png, genB.png*, itd.).

Dodatkowo, plik nagłówkowy przechowuje informację o domyślnej szerokości wyświetlanej komórki w pikselach.

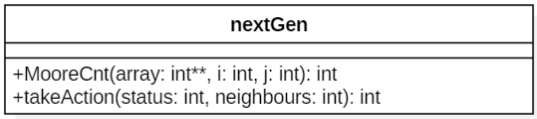
```
#define CELL_SIZE 16
```

5. **Moduł genGenerator**



Funkcja *generateNext* przyjmuje tablicę z bieżącą generacją i zwraca tablicę z generacją potomną. Do utworzenia nowej generacji używa funkcji z modułu *nextGen*.

6. **Moduł nextGen**



Funkcja *MooreCount* zlicza ile jest żywych sąsiadów komórki o współrzędnych i i j 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).

2 Opis modyfikacji

Program udało się napisać w dużej mierze zgodnie ze specyfikacją implementacyjną. Większe zmiany dotknęły dwa moduły:

1. **argsReader**

   Początkowo, moduł ten miał zawierać dwie funkcje. Jedna miała odczytywać argumenty typu *int* i zwracać ich wartość, druga zaś obsługiwała argumenty typu *char**.

   Zrezygnowano z tego rozwiązania z uwagi na jego niepraktyczność. Łatwiej jest przekazać adresy zmiennych do funkcji, gdyż pozwala to ustawić wartości ich wszystkich przy jednym wywołaniu. Rozwiązanie ze zwracaniem wartości wymaga wielokrotnego wywołania funkcji i rozbicia jej na poszczególne typy zwracanych zmiennych.

2. **pngCreator**

   Domyślnie, moduł ten miał zawierać dwie dodatkowe funkcje: *addCell* oraz *BlankMap*, odpowiedzialne za dodawanie do tablicy kolorów odpowiednio pojedynczej komórki oraz domyślnej planszy (czarna ramka, białe tło).

   Z pierwszej z tych funkcji tych zrezygnowano, ponieważ dodanie pojedynczej komórki do tablicy było operacją na tyle prostą, iż nie wymagało to stosowania osobnej funkcji.

   Drugą funkcję odrzucono, gdyż prościej było od razu przerzucić całą generację do tablicy kolorów, aniżeli rozbijać to na operację tworzenia domyślnej planszy i dodawania na nią żywych komórek.

   Dodana do modułu została także funkcja *file_name* odpowiedzialna za generowania nazw plików PNG, gdyż zwyczajnie zapomniano jej uwzględnić w specyfikacji implementacyjnej.

Warto również zaznaczyć, iż tablica przechowująca generację, oprócz przechowywania wartości 0 i 1 symbolizujących stan komórek, przechowuje również ramkę (pod postacią cyfry 2). Rozwiązanie to ułatwia działanie funkcji *MooreCnt* zliczającej sąsiadów oraz generowanie plików PNG.

Domyślne wartości wysokości i szerokości zmieniono na 10. Większe plansze są rzadko przydatne i niepotrzebnie zajmują pamięć.

Maksymalna liczba wygenerowanych plików PNG to 26 (liczba liter alfabetu). Rozwiązanie to pomaga przy generowaniu nazw kolejnych plików.

3 Prezentacja działania

1. **Wywołanie nr. 1**

   Najprostsze wywołanie programu. Podajemy tylko plik wejściowy, reszta domyślna.

   Plik wejściowy w postaci:

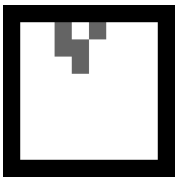
   ```
   1       3
   2       3
   2       4
   3       4
   1       5
   ```

   Wywołanie:

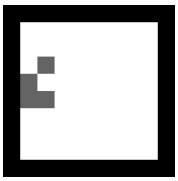
   ```
   root@kristoph:~/life/LIFE# ./life --f plik1
   Nie podano pliku do zapisu ostatniej generacji - plik nie zostanie utworzony
   root@kristoph:~/life/LIFE#
   ```

   Wygenerowane pliki PNG (pierwsza i ostatnia generacja):

   genA.png

   genB.png

2. **Wywołanie nr. 2** Tym razem lepiej sprawdzimy działania programu przy nieprawidłowych danych wejściowych. Podano wartość PNG większą niż liczbę generacji, natomiast podane wartości wysokości oraz szerokości są nieprawidłowe.

   Plik wejściowy w postaci:

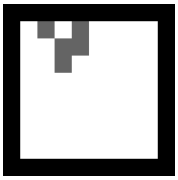
   ```
   1       4
   2       4
   2       3
   3       3
   1       2
   ```

   Wywołanie:

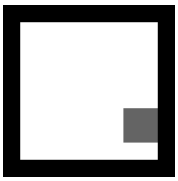
```
root@kristoph:~/life/LIFE# ./life --f plik1.txt --n 30 --p 45 --w --h a
Nie podano pliku do zapisu ostatniej generacji - plik nie zostanie utworzony
Podano nieprawidlowa szerokosc - przyjmuje wartosc domyslna szerokosci: 10
Podano nieprawidlowa wysokosci - przyjmuje wartosc domyslna wysokosci: 10
Nie mozna wygenerowac wiecej plików PNG niz generacji - przyjmuje liczbe PNG rowna liczbie generacji: 30
Maksymalna liczba plików PNG to 26! - przyjmuje wartosc domyslna liczby PNG: 2
root@kristoph:~/life/LIFE#
```

Wygenerowane pliki PNG (pierwsza i ostatnia generacja):

genA.png



genB.png



3. Wywołanie nr. 3

Tym razem wygenerujemy 5 plików PNG dla 5 generacji, zmienimy szerokość oraz wysokość planszy oraz stworzymy plik tekstowy ze współrzędnymi komórek ostatniej generacji.

Plik wejściowy ten sam co w wywołaniu nr. 2.

Wywołanie:

```
root@kristoph:~/life/LIFE# ./life --f plik1.txt --n 5 --p 5 --w 16 --h 9 --o plik2.txt
root@kristoph:~/life/LIFE#
```

Wygenerowane pliki PNG (kolejne 5 generacji):

genA.png

genB.png

genC.png

genD.png

genE.png

Wygenerowany plik TXT:

plik2.txt

```
2       5
3       3
3       5
4       4
4       5
```

4 Podsumowanie testów

4.1 Testy modułu argsReader : 2/2

Testy tego modułu polegały na wywołaniu programu testującego o nazwie *TEST_argsReader.c* z określonymi parametrami wywołania, a następnie porównaniu ich z wyświetlonym i na ekranie parametrami , które zostały pobrane przez funkcję *search4args*.

Pierwszy test:

```
kunikowh@jimp:~/LIFE/LIFE/LIFE$ cat TEST_arrayCreator
1 1       2 2     3 3      4 4
5 5       5 1     4 2      2 4
          1 5
kunikowh@jimp:~/LIFE/LIFE/LIFE$ ./a.out

 Wczytano tablice:
2 2 2 2 2 2 2 2 2 2
2 1 0 0 0 1 0 0 0 2
2 0 1 0 1 0 0 0 0 2
2 0 0 1 0 0 0 0 0 2
2 0 1 0 1 0 0 0 0 2
2 1 0 0 0 1 0 0 0 2
2 0 0 0 0 0 0 0 0 2
2 0 0 0 0 0 0 0 0 2
2 0 0 0 0 0 0 0 0 2
2 2 2 2 2 2 2 2 2 2
```

Drugi test (podanie nieprawdiłowych danych):

```
Nie podano pliku do zapisu ostatniej generacji - plik nie zostanie
 utworzony
Nie mozna wygenerowac wiecej plików PNG niz generacji - przyjmuje
liczbe PNG rowna liczbie generacji: 12
Blad - nie podano pliku do odczytu pierwszej generacji
```

4.2 Testy modułu arrayCreator : 2/2

Testem modułu arrayCreator było wywołanie programu *TEST_arrayCreator.c* , który wczytuje dane znajdujące się w pliku *TEST_arrayCreator.txt* do tablicy, a następnie w celu weryfikacji, wypisanie danych na standardowe wyjście . Jeśli funkcja działała poprawnie na ekranie powinniśmy ujrzeć tablicę planszę z żywymi komórkami układającymi się w znak X o wymiarach 5x5.

Pierwszy test:

```
kunikowh@jimp:~/LIFE/LIFE/LIFE$ cat TEST_arrayCreator
1 1       2 2     3 3      4 4
5 5       5 1     4 2      2 4
          1 5
kunikowh@jimp:~/LIFE/LIFE/LIFE$ ./a.out

 Wczytano tablice:
2 2 2 2 2 2 2 2 2 2
2 1 0 0 0 1 0 0 0 2
2 0 1 0 1 0 0 0 0 2
2 0 0 1 0 0 0 0 0 2
2 0 1 0 1 0 0 0 0 2
2 1 0 0 0 1 0 0 0 2
2 0 0 0 0 0 0 0 0 2
2 0 0 0 0 0 0 0 0 2
2 0 0 0 0 0 0 0 0 2
2 2 2 2 2 2 2 2 2 2
```

Drugi test(nieprawidłowe dane wejściowe):

```
Blad - wspolrzedne punktow w pliku nie zgadzaja sie z wymiarami tablicy
```

4.3 Testy modułu txtCreator : 1/1

Test modułu *txtCreator.c* polegał na stworzeniu tablicy, traktowanej jako tablica żywych komórek, o wymiarach 10x10, w której komórki układają się w znak x. Następnie przy użyciu funkcji *print2screen* i *write2txt* tablica ta jest wyświetlana na ekranie oraz jako generacja zapisywana jest do pliku o nazwie *TEST_txtCreator.txt* . Poprzez porównanie wyświetlonej tablicy ze współrzędnymi punktów stwierdzono prawidłowe działanie funkcji.

```
2 2 2 2 2 2 2 2 2 2
2 1 0 0 0 0 0 0 1 2
2 0 1 0 0 0 0 1 0 2
2 0 0 1 0 0 1 0 0 2
2 0 0 0 1 1 0 0 0 2
2 0 0 0 1 1 0 0 0 2
2 0 0 1 0 0 1 0 0 2
2 0 1 0 0 0 0 1 0 2
2 1 0 0 0 0 0 0 1 2
2 2 2 2 2 2 2 2 2 2

Efektem dzialania funkcji write2txt jest plik TEST_txtCreator.txt
kunikowh@jimp:~/LIFE/LIFE/LIFE$ cat TEST_txtCreator
1       1
1       8
2       2
2       7
3       3
3       6
4       4
4       5
5       4
5       5
6       3
6       6
7       2
7       7
8       1
8       8
```

4.4 Testy modułu pngCreator : 1/1

Program testujący moduł *pngCreator* wykorzystuje moduł *arrayCreator* i plik *TEST_arrayCreator.txt* zawierający dane w celu utworzenia tablicy. W wyniku wywołania programu testującego *TEST_pngCreator* powstał plik *genB.png* zawierający graficzną prezentację generacji z pliku *TEST_arrayCreator.txt*.
Żywe komórki w tablicy układały się w znak X, a więc plik genB.png zawierał również X .