

***SPRAWOZDANIE KOŃCOWE***  
***"Gra w życie" DLA JĘZYKA***  
***PROGRAMOWANIA C***

Wykonali:

1. Ivan Prakapets

## Spis treści

<b>1</b>	<b>Wprowadzone zmiany</b>	<b>2</b>
1.1	Moduł read_and_write_png . . . . .	2
1.2	Moduł read_txt . . . . .	2
1.3	Moduł generacja . . . . .	3
1.4	Moduł sąsiedstwa . . . . .	3
<b>2</b>	<b>Dodanie modułów</b>	<b>4</b>
2.1	Moduł reguła_generacji . . . . .	4
2.2	Moduł symulator . . . . .	4
2.3	Moduł test_game . . . . .	5
<b>3</b>	<b>Działania i wyniki programu</b>	<b>6</b>
3.1	Kompilacja i uruchomienie . . . . .	6
3.1.1	Kompilacja . . . . .	6
3.1.2	Uruchomienie . . . . .	6
3.2	Wyniki programu . . . . .	7
3.2.1	Działania programu dla plików .png . . . . .	7
3.2.2	Działania programu dla plików .txt . . . . .	9
<b>4</b>	<b>Wnioski</b>	<b>11</b>

## 1 Wprowadzone zmiany

W implementacji programu zaszły następujące zmiany względem złożień specyfikacji implementacyjnej:

### 1.1 Moduł `read_and_write_png`

Moduł `read_and_write_png` składał się z trzech funkcji:

```
* void read_png_file ( char * )  
  
* void write_png_file ( char * , int )  
  
* void process_file ( int ** )
```

Została dodana funkcja:

```
* void process_file_for_txt ( int ** )
```

Funkcja `process_file_for_txt`. Celem tej funkcji jest sprawdzanie pliku `txt` oraz wdrażenie funkcji `read_png_file` do gry.

### 1.2 Moduł `read_txt`

Moduł `read_txt` nie został zmieniony.

### 1.3 Moduł generacja

Moduł generacja składał się z dwóch funkcji:

```
* void print_gen_to_file ( int ** )
```

```
* void print_gen ( int ** )
```

Została usunięta funkcja:

```
* void print_gen ( int ** )
```

Celem usunięcia tej funkcji jest taka, że więcej jej nie potrzebowali.

### 1.4 Moduł sasiadstwa

Moduł sasiadstwa składał się z dwóch funkcji z jednym imieniem

```
* int count_neighbours ( int , int , int ** )
```

Została wprowadzona zmiana nazwy funkcji i dodana funkcja:

```
* int count_neighbours_Moore_a ( int , int , int ** )
```

```
* int count_neighbours_Neumann ( int , int , int ** )
```

To jest zrobiono dla ułatwienia programu, dlatego że musimy zrobić taki samy program w dwa różnych sposoby.

## 2 Dodanie modułów

W implementacji programu zaszły następujące nowe moduły względem złożenia specyfikacji implementacyjnej:

### 2.1 Moduł `regula_generacji`

Moduł `regula_generacji` składa się z trzech funkcji:

```
* int** create_next_gen_Moore_a ( int ** )  
  
* int** create_next_gen_Neumann ( int ** )  
  
* void free_matrix ( int ** )
```

1. Funkcja `create_next_gen_Moore_a`. Celem tej funkcji jest stworzenie następującej generacji dla sąsiedztwa `Moore_a`.
2. Funkcja `create_next_gen_Neumann`. Celem tej funkcji jest stworzenie następującej generacji dla sąsiedztwa `Neumann`.
3. Funkcja `free_matrix`. Celem tej funkcji jest usunięcie wycieków pamięci (oczyszczenie tablicy `matrix`).

### 2.2 Moduł `symulator`

Moduł `symulator` składa się z 5 funkcji:

```
* void symulator_Moore_a ( int , int ** , char* )  
  
* void symulator_Neumann ( int , int ** , char* )
```

```
* void simulator_Moore_a_txt ( int , int ** , char* )  
  
* void simulator_Neumann_txt ( int , int ** , char* )  
  
* void usage ( )
```

1. Funkcja `simulator_Moore_a`. Celem tej funkcji jest symulacja dla sąsiedztwa `Moore_a`.
2. Funkcja `simulator_Neumann`. Celem tej funkcji jest symulacja dla sąsiedztwa `Neumann`.
3. Funkcja `simulator_Moore_a_txt`. Celem tej funkcji jest symulacja dla sąsiedztwa `Moore_` w formacie `.txt`.
4. Funkcja `simulator_Neumann_txt`. Celem tej funkcji jest symulacja dla sąsiedztwa `Neumann` w formacie `.txt`.
5. Funkcja `usage`. Celem tej funkcji jest wyświetlenie informacji jak uruchomić i korzystać z programu.

## 2.3 Moduł `test_game`

Moduł `test_game` składa się z jednej testującej funkcji:

```
* int main ( int , char * )
```

która sprawdza poprawne wczytywanie z pliku `.txt` i z pliku `.png` również wypisuje błędy.

## 3 Działania i wyniki programu

Po napisaniu koda są takie wyniki:

### 3.1 Kompilacja i uruchomienie

#### 3.1.1 Kompilacja

Ekran działania programu:

```
ivan@ivan:~/Ivan/C/Ostateczna_wersja_projektu/projekt_c$ make
cc -o game main.c generacja.c read_and_write_png.c read_txt.c regula_generacji.c sasiadstwa.c symulator.c -std=c99 -l png
ivan@ivan:~/Ivan/C/Ostateczna_wersja_projektu/projekt_c$ ./game
Zeby uruchomic program prosze wprowadzić 5 argumentów

Usage:
    Pierwszy argument:
    0 - To zaczyna się generacja za pomocą metody Moore'a
    1 - To zaczyna się generacja za pomocą metody Neumanna

    Drugi argument:
    2 - Czyta .png plik
    3 - Czyta .txt plik

    Trzeci argument:
    Wprowadź plik wejściowy .txt lub .png znajdujący w folderze dane w zależności od drugiego argumentu

    Czwarty argument:
    Wprowadź nazwę pliku wyjściowego

    Piąty argument:
    Wprowadź liczbę generacji

    Przykładowe wywołanie programu:
    ./game 0 2 dane/input.png out 200

    Oczyszczenie stworzonych plików,obrazków oraz folderu:
make clean
ivan@ivan:~/Ivan/C/Ostateczna_wersja_projektu/projekt_c$
```

Rysunek 1: Ekran programu

#### 3.1.2 Uruchomienie

Przykładowa komenda dla uruchomienia:

```
ivan@ivan:~/Ivan/C/Ostateczna_wersja_projektu/projekt_c$ ./game 0 2 dane/input.png output 10
.Plik o nazwie dane/input.png został wczytany
Tworzenie folderu wynik...
Plik gif o nazwie out.gif został stworzony z pomocą metody Moore'a
Zeby popatrzyc wyniki, prosze wejsc w katalog wynik
ivan@ivan:~/Ivan/C/Ostateczna_wersja_projektu/projekt_c$
```

Rysunek 2: Przykład komendy

## 3.2 Wyniki programu

### 3.2.1 Działania programu dla plików .png

Plik wejściowego dane/input.png ma postać:



Rysunek 3: Plik wejściowy input.png



Plik końcowy numer 116 `wynik/output_116.png` ma postać:



Rysunek 4: Plik końcowy `output_116.png`

Plik wejściowy ma postać:

[illegible]

Rysunek 5: Plik wejściowy

[illegible]

Rysunek 6: Plik końcowy wynik\_116.txt

## 4 Wnioski

Podczas implementacji programu zaszło kilka zmian względem założeń ze specyfikacji implementacyjnej. Wynikały one głównie z lepszego rozeznania problemu, a także z zauważania oddzielnych i niezależnych funkcjonalności w większych funkcjach. Zostały też przeprowadzone zmiany kosmetyczne dotyczące nazw modułów i wyglądu programu, mające na celu wyeliminowanie nieścisłości kodu oraz poprawę jego czytelności. Wówczas podstawowe i najważniejsze założenia pochodzące ze specyfikacji implementacyjnej zostały zgodne z pierwotnymi ustaleniami, co potwierdza dobrze przemyślaną koncepcję systemu zawartą w specyfikacji funkcjonalnej oraz implementacyjnej.