

# Specyfikacja implementacyjna – Gra w życie

Krzysztof Dąbrowski i Jakub Bogusz

13 marca 2019

## Spis treści

<b>1</b>	<b>Podział na moduły</b>	<b>1</b>
1.1	GameOfLife . . . . .	2
1.1.1	Funkcje . . . . .	2
1.2	ArgumentsParser . . . . .	2
1.2.1	Funkcje . . . . .	2
1.2.2	Struktury . . . . .	2
1.3	BoardHandler . . . . .	3
1.3.1	Struktury . . . . .	3
1.3.2	Funkcje . . . . .	4
1.4	Simulator . . . . .	4
1.4.1	Funkcje . . . . .	4
1.5	Loader . . . . .	5
1.5.1	Funkcje . . . . .	5
<b>2</b>	<b>Biblioteki zewnętrzne</b>	<b>5</b>
2.1	zlib.h . . . . .	5
2.2	libpng.h . . . . .	6

## 1 Podział na moduły

Program będzie podzielony na współdziałające moduły. Pozwoli to na łatwiejszą modyfikację programu oraz dodawanie nowych funkcjonalności.

### DIAGRAM MODÓŁÓW

#### Spis modułów

- GameOfLife
- ArgumentsParser
- BoardHandler
- Simulator

- Save
- Loader
- GraphicsGenerator

## 1.1 GameOfLife

Główny moduł kontrolujący przepływ sterowania i danych między pozostałymi modułami.

### 1.1.1 Funkcje

`int main(int argc, char** args)` – Punkt startowy programu. Z niej wywoływane będą kolejne funkcje. Przyjmować będzie 2 argumenty – argumenty wsadowe programu:

`int argc` – ilość argumentów,

`char** args` – tablica napisów – faktycznych argumentów wywołania programu.

## 1.2 ArgumentsParser

Moduł odpowiadający za interpretację podanych wsadowo argumentów programu, konwersji ich oraz zapisu do utworzonej w tym celu struktury.

### 1.2.1 Funkcje

`params parseArgs(int argc, char** argv)` – będzie przetwarzać argumenty wsadowe programu podane w postaci flag na strukturę zawierającą ustawienia symulacji. Przyjmować będzie 2 argumenty – argumenty wsadowe programu:

`int argc` – ilość argumentów,

`char** args` – tablica napisów, będących faktycznymi argumentami wywołania programu.

Zwracać będzie strukturę zawierającą ustawienia symulacji.

### 1.2.2 Struktury

Struktura zawierająca ustawienia symulacji:

```
typedef struct{
    int help;
```

```

char* file;
char* output_dest;
char* type;
int number_of_generations;
int step;
int delay;
}params;

```

**help** – informuje o tym czy ma być wyświetlana pomoc,  
**file** – informuje o tym jaka jest ścieżka do pliku wejściowego,  
**output\_dest** – informuje o tym jaka jest ścieżka dla pliku/plików wyjściowych,  
**type** – informuje o tym jaki jest typ zwracanych wyników,  
**number\_of\_generations** – informuje o tym ile pokoleń komórek zostanie wygenerowane,  
**step** – informuje o tym co ile pokoleń zapisywane będą dane wyjściowe,  
**delay** – informuje w jakich odstępach czasu mają się wyświetlać kolejne generacje komórek.

## 1.3 BoardHandler

Moduł definiujący strukturę planszy – *Board* oraz podstawowe funkcje związane z tą strukturą.

### 1.3.1 Struktury

Typ wyliczeniowy zawierający możliwe stany pojedynczej komórki.

```

typedef enum{
    DEAD = 0,
    ALIVE = 1
}CellState;

```

DEAD – Komórka jest martwa

ALIVE – Komórka jest żywa

Struktura przechowująca stan pojedynczego pokolenia.

```

typedef struct{
    int sizeX;
    int sizeY;
    CellState *cells;
}Board;

```

sizeX – Szerokość planszy

sizeY – Wysokość planszy

cells – Tablica zawierająca stany wszystkich komórek

### 1.3.2 Funkcje

**Board\* createRandomBoard(int x, int y)** – Tworzy planszę o podanej wielkości z komórkami o losowych stanach. Zwracany jest wskaźnik na nowy obiekt **Board** utworzony na stercie.

**int x** – szerokość planszy

**int y** – wysokość planszy

**void disposeBoard(Board \*board)** – Zwalnia dynamicznie alokowane pola struktury **Board**. Powinna zostać wywołana przed zwolnieniem dowolnej zmiennej typu **Board**. Po zastosowaniu tej metody zmienna wskazana przez argument **board** nie nadaje się już do użytku.

**Board \*board** – Adres planszy, która będzie zwalniana

**char\* boardToString(Board \*board)** – Generuje ciąg znaków reprezentujący stan wskazanej planszy. Zwrócony napis należy po wykożystaniu zwolnić.

**Board \*board** – Adres planszy, której reprezentacja ma zostać wygenerowana

**char\* serializeBoard(Board \*board)** – Generuje ciąg znaków reprezentujący wskazaną strukturę **Board**. Zwrócony napis zawiera wszystkie niezbędne informacje do rekonstrukcji struktury na jego podstawie. Zwrócony napis należy po wykożystaniu zwolnić.

**Board \*board** – Adres planszy, której reprezentacja ma zostać wygenerowana

## 1.4 Simulator

Moduł odpowiadający za przeprowadzenie właściwej symulacji zgodnie z regułami gry w życie.

### 1.4.1 Funkcje

**board simulate(board b, params p)** – będzie przeprowadzać symulację całej gry w życie:

**board b** – struktura zawierająca stan planszy,

**params p** – struktura zawierająca ustawienia symulacji.

Zwracać będzie strukturę zawierającą końcowy stan planszy.

`board nextGen(board b)` – będzie generować planszę odpowiadającą następnemu pokoleniu komórek. Jako argument będzie przyjmować:

`board b` – struktura zawierająca stan planszy,

Zwracać będzie strukturę zawierającą stan planszy w następnym pokoleniu.

## 1.5 Loader

Moduł odpowiedzialny za wczytanie planszy początkowej z pliku tekstowego i zapisanie jej do struktury.

### 1.5.1 Funkcje

`board load(char* path)` – będzie przetwarzać plik wejściowy i zapisywać go do struktury. Jako argument będzie przyjmować:

`char* path` – ścieżka do pliku wejściowego,

Zwracać będzie strukturę opisującą początkowy stan planszy.

`int* getSize(char* path)` – będzie wczytywać rozmiar planszy zapisanej w pliku wejściowym:

`char* path` – ścieżka do pliku wejściowego,

Zwracać będzie dwuelementowy wektor zawierający rozmiary planszy początkowej.

## 2 Biblioteki zewnętrzne

Program będzie korzystał z bibliotek nie będących częścią standardu języka C. Umożliw to dostarczenie szerszej funkcjonalności.

### 2.1 zlib.h

Biblioteka zawierająca darmową implementację algorytmów kompresji i dekompresji *deflate*. Jest ona wykorzystywana wewnętrznie przez bibliotekę `libpng.h`.

## 2.2 libpng.h

Biblioteka ta umożliwia tworzenie oraz odczytywanie plików graficznych takich jak *png* czy *gif*.