

# Specyfikacja funkcjonalna programu WireWorld

Danuta Stawiarz, Katarzyna Stankiewicz

20 maja 2019 r.

## Spis treści

<b>1</b>	<b>Informacje ogólne</b>	<b>2</b>
<b>2</b>	<b>Schemat działania programu</b>	<b>2</b>
<b>3</b>	<b>Opis modułów</b>	<b>3</b>
3.1	Moduł Game . . . . .	3
3.2	Moduł Model . . . . .	4
3.3	Moduł Controller . . . . .	5
3.4	Moduł View . . . . .	6
<b>4</b>	<b>Zastosowane algorytmy</b>	<b>6</b>
4.1	Algorytm wyznaczania stanu komórki dla wariantu gry Life . . .	6
4.2	Algorytm wyznaczania stanu komórki dla wariantu gry Wireworld	6
<b>5</b>	<b>Testy</b>	<b>7</b>
5.1	Test klasy Cells . . . . .	7
5.2	Test klasy Board . . . . .	7
5.3	Test modułu Files . . . . .	7
5.4	Test modułu View . . . . .	8

---

## 1 Informacje ogólne

W projekcie zastosowany będzie wzorzec MVC (Model View Controller). Aby ułatwić pracę nad projektem oraz późniejsze modyfikacje, wprowadzony zostanie przejrzysty przydział poszczególnych plików do odpowiednich pakietów. W pakiecie models będą się znajdować klasy, które służą do wykonywania wszelkich operacji związanych z implementacją funkcjonalności automatu komórkowego. W pakiecie View znajdują się wszystkie pliki .fxml odpowiedzialne za wygląd automatu komórkowego. W pakiecie controllers będą zawarte kontrolery, służące do obsługi żądań użytkownika.

## 2 Schemat działania programu

Program reaguje bezpośrednio na komendy użytkownika.

1. Pobranie od użytkownika informacji o parametrach początkowych gry - wariantu gry, który zostanie przeprowadzony (Wireworld lub Life). Inicjacja obiektu klasy Wireworld lub Life (rozszerzenie abstrakcyjnej klasy Game).
2. Pobranie danych o wymiarach planszy lub ścieżki do pliku, na podstawie którego ma zostać utworzona rozgrywka oraz utworzenie obiektu typu Board oraz umieszczenie go w obiekcie klasy Wireworld lub Life.
3. Reakcje na bezpośrednie działania użytkownika:
  - Start - naciśnięcie tego przycisku uruchomi symulację, zostanie wywołana metoda Play(), która przeprowadza symulację,
  - Generations - w przypadku naciśnięcia którejś ze strzałek, zostaje wywołana metoda klasy Game - Play() lub Play\_back(), które generują wygląd kolejnych siatek. Wynik symulacji zostanie wyświetlony na ekranie aplikacji,
  - Board reset - nastąpi zresetowanie tablicy reprezentującej siatkę za pomocą funkcji klasy Board - Reset Board poprzez zamianę w obiekcie obiektu klasy Board w klasie Game na nowoutworzony obiekt tego typu,
  - Upload file - spowoduje wczytanie siatki ze stanami poszczególnych komórek poprzez utworzenie obiektu klasy Board\_Import i odczyt informacji z niego,
  - Random fill - spowoduje wywołanie metody klasy Board - random\_fill() i wypełnienie siatki losowymi wartościami,
  - Save - spowoduje utworzenie obiektu klasy Board\_Export i zapis stanu tablicy do pliku za pomocą funkcji tej klasy

---

## 3 Opis modułów

Program składa się z trzech głównych modułów odpowiedzialnych za prawidłowe działanie programu. Są to

- Model - odpowiada za przeprowadzanie kolejnych generacji, jak i całej symulacji
- Files - moduł odpowiadający za obsługę plików tekstowych, z których możliwy jest odczyt i zapis informacji dotyczących parametrów gry
- GUI - moduł odpowiadający za aspekt wizualny gry, a także reakcję na działania użytkownika (przekazuje informację o przypadkach naciśnięcia poszczególnych przycisków)

### 3.1 Moduł Game

Moduł ten odpowiada za poprawne przeprowadzenie symulacji. Zawiera klasy reprezentujące grę i zawierające w sobie obiekty klas będących składowymi innymi modułów. Klasy:

1. **Abstract Class Game ()** - klasa ta stanowi podstawę dla klas dziedziczących po niej - Wireworld oraz Life, związane jest to z wariantem gry, który wybierze użytkownik. Klasa ta zawiera obiekt klasy Board, który reprezentuje siatkę komórek. Główne metody tej klasy to:
  - public set\_Cell (int stan) - umożliwia zmianę stanu wybranej komórki w siatce
  - public get\_Board () - zwraca obiekt klasy Board reprezentujący siatkę,
  - public set\_Board ( Board) - pobiera obiekt klasy Board i zapisuje w zmiennej board,
  - public update (Board) - pobiera obiekt klasy Board i nadpisuje go w miejscu zmiennej board,
  - public play() - metoda ta jest nadpisywana w klasach dziedziczących po klasie Game, odpowiada za przeprowadzenie symulacji w kierunku chronologicznym (generacja kolejnych stanów siatki) zgodnie z zasadami danego wariantu gry,
  - public play\_back()- metoda ta jest nadpisywana w klasach dziedziczących po klasie Game, odpowiada za przeprowadzenie symulacji w kierunku odwrotnym (generacja poprzednich stanów siatki) zgodnie z zasadami danego wariantu gry.
2. **public class Wireworld extends Game** - klasa ta jest pochodną klasy Game, następuje w niej przeciążenie metody play oraz play\_back. Zostają one dostosowane do zasad gry.

- 
3. **public class Wireworld extends Game** - klasa ta jest pochodną klasy Game, następuje w niej przeciążenie metody play oraz play\_back. Zostają one dostosowane do zasad gry.

### 3.2 Moduł Model

Moduł ten zawiera klasy będące składowymi klasy Game. Umożliwiają one przeprowadzenie szczegółowej symulacji. Najważniejsze klasy tego modułu to:

1. **public class Cells** - klasa reprezentująca pojedynczą komórkę. Główne zmienne zawarte w tej klasie to:
  - private double x, private double y - zmienne określające położenie komórki w siatce,
  - private double size - wielkość komórki,
  - private int state - określa stan komórki, w zależności od wariantu gry komórka może znajdować się w dwóch lub czterech różnych stanach,
  - private int aliveNextCells - określa ilość żywych sąsiadów w najbliższym otoczeniu komórki zgodnie z zasadą sąsiedztwa Moore'a,

Klasa zawiera następujące metody:

- public double getX() - zwraca wartość zmiennej **x**,
- public void setX(double) ustawia wartość zmiennej **x** na wartość podaną jako argument wywołania metody,
- public double getY() - zwraca wartość zmiennej **y**,
- public void setY(double) - ustawia wartość zmiennej **y** na wartość podaną jako argument wywołania metody,
- public double getSize() - zwraca wartość zmiennej **size**,
- public int getState(int) - zwraca wartość zmiennej **state**,
- public void setState(int) - ustawia stan komórki na wartość podaną przy wywołaniu metody,
- public void setaliveNextCells(int) - ustawia wartość zmiennej **aliveNextCells** na wartość podaną przy wywołaniu metody

Klasa Cell posiada również kilka konstruktorów, są to:

- public Cell() - przy braku argumentów, wartość zmiennej **state** będzie wynosiła 0,
  - public Cell (Cell cell) - przy tworzeniu obiektu jako argument zostaje podany obiekt klasy Cell - zmienna **state** otrzymuje taką wartość, jak wartość zmiennej **state** obiektu
2. **public class Board** - klasa ta reprezentuje siatkę komórek i jest jedną z najważniejszych klas. Zawiera następujące zmienne:

- 
- private int columns - zmienna oznaczająca liczbę kolumn w siatce,
  - private int rows - zmienna oznaczająca liczbę wierszy w siatce,
  - Cell[] cells - tablica obiektów klasy Cells reprezentująca siatkę komórek

Klasa ta posiada następujące główne metody:

- public void neighbours\_count() - metoda zliczająca sąsiadów o stanie określonym jako "żywa" dla każdej komórki w siatce i przypisująca obliczoną wartość do zmiennej `aliveNextCells` w obiektach klasy Cells (metoda dla wariantu gry Life).
- public void neighbours2\_count() - metoda zliczająca sąsiadów o stanie określonym jako Głowa Elektronu dla każdej komórki w siatce i przypisująca obliczoną wartość do zmiennej `aliveNextCells` w obiektach klasy Cells (metoda dla wariantu Wireworld).
- public Cell getCell (int,int) - zwraca obiekt klasy Cell umieszczony w siatce pod wskazanymi w wywołaniu współrzędnymi,
- public void setCell (int,int,int) - przypisuje pod podane współrzędne komórkę o określonym stanie, metoda jest przeciążana,
- public void setCell(Cells) - przypisuje podaną w argumencie wywołania komórkę pod adres w tablicy.

Klasa ta posiada także kilka konstruktorów:

- public Board() - pusty konstruktor, zostaje w nim utworzona pusta tablica obiektów Cells, o wymiarach domyślnych,
- public Board(Board) - konstruktor, w którym nowopowstały obiekt przejmuje parametry obiektu podanego w argumenach wywołania.

### 3.3 Moduł Controller

Moduł ten odpowiada za obsługę przez program plików tekstowych. Użytkownik ma możliwość otworzenia symulacji przez wczytanie jej parametrów do pliku, a także jej późniejszy zapis. Wymaganiom tym odpowiadają dwie klasy:

1. **Board\_export** - klasa ta odpowiada za zapis pliku danych dotyczących siatki komórek. Korzysta z biblioteki `java.io.*`. Posiada zmienne:

- `FileReader file` - plik, dp którego znak po znaku zostaną zapisane dane

Metodą tej klasy jest:

- public void Save (Board) - przekazany do metody obiekt typu Board, zostaje zapisany do pliku, poprzez zapis poszczególnych danych dotyczących wymiarów siatki, położenia i stanów poszczególnych komórek

- 
2. **Board\_import** - klasa ta odpowiada za odczyt z pliku danych dotyczących siatki komórek. Korzysta z biblioteki `java.io.*`. Posiada zmienne:

- `FileReader file` - plik, z którego znak po znaku zostaną odczytane dane

Metodą tej klasy jest:

- `public Board Read ()` - w metodzie tej następuje odczytanie poszczególnych danych z pliku. Zwraca obiekt klasy `Board`.

### 3.4 Moduł View

## 4 Zastosowane algorytmy

W programie zostały zastosowane następujące algorytmy:

### 4.1 Algorytm wyznaczania stanu komórki dla wariantu gry Life

1. Wywołanie metody klasy `Board` dla konkretnej komórki w celu zliczenia jej żywych sąsiadów.
2. Określenie stanu komórki na podstawie ilości jej żywych sąsiadów:
  - Komórka martwa - jeżeli liczba żywych sąsiadów wynosi 3, stan komórki zostaje określony jako "żywa", w innym wypadku komórka pozostaje martwa;
  - Komórka żywa - jeżeli liczba żywych komórek wynosi 2 lub 3, stan nie zmienia się, w innym wypadku komórka pozostaje martwa.
3. Zapis nowej wartości stanu komórki w obiekcie klasy `Cells` pod zmienną `state`.

### 4.2 Algorytm wyznaczania stanu komórki dla wariantu gry Wireworld

1. Wywołanie metody klasy `Board` zliczającej sąsiednich komórek, których stan określany jest jako Głowa Elektronu.
2. Określenie stanu komórki na podstawie poprzedniego stanu i analizy stanów sąsiednich komórek.
  - Komórka pozostaje Pusta, jeśli była Pusta.
  - Komórka staje się Ogonem elektronu, jeśli była Głową elektronu.
  - Komórka staje się Przewodnikiem, jeśli była Ogonem elektronu.
  - Komórka staje się Głową elektronu tylko wtedy, gdy dokładnie 1 lub 2 sąsiadujące komórki są Głowami Elektronu.

- 
- Komórka staje się Przewodnikiem w każdym innym wypadku

3. Zapis nowej wartości stanu komórki w obiekcie klasy Cells pod zmienną `state`.

Dla obu algorytmów wykorzystane jest sąsiedztwo Moore'a.

## 5 Testy

### 5.1 Test klasy Cells

W celu przetestowania tej klasy należy utworzyć jej obiekt oraz poprzez wywołania kolejnych metod sprawdzić, czy jej zmienne są poprawnie inicjowane. Szerszych testów wymaga moduł zliczający żywych sąsiadów lub sąsiadów o określonym stanie w zależności od wariantu gry, w którym ma funkcjonować obiekt.

### 5.2 Test klasy Board

Test tego modułu zakłada zainicjowanie obiektu klasy Board oraz przetestowanie działania jego metod. Do przetestowania tego modułu potrzebny jest poprawnie działająca klasa Cells. Szczegółowych testów wymaga metoda `neighbours` oraz `neighbours2`.

### 5.3 Test modułu Files

Testy tego modułu polegają na odczycie i zapisie do plików txt. W celu przetestowania tego modułu niezbędny jest poprawnie działający moduł Board. Zapis do pliku w przypadku gry Life powinien wyglądać następująco:

```
5 5
1 0 1 0 0
0 0 0 0 0
1 0 0 1 0
0 1 1 1 1
0 0 0 0 0
```

Pierwsze dwie cyfry oznaczają tu wymiary tablicy, która jest zmienną w obiekcie klasy typu Board. Kolejne cyfry symbolizują stany poszczególnych komórek (obiektów klasy Cells), które zostają zainicjowane w tablicy w obiekcie klasy Board. Zera oznaczają stan komórki jako "martwa", a jedynki jako "żywa". Plik przedstawiający siatkę dla wariantu gry "Wireworld" powinien wyglądać następująco:

```
5 5
1 0 1 3 0
0 0 0 0 2
```

---

```
1 0 0 1 0
0 1 1 1 1
2 0 0 3 0
```

Pierwsze dwie cyfry oznaczają tu wymiary tablicy, która jest zmienną w obiekcie klasy typu Board. Kolejne cyfry symbolizują stany poszczególnych komórek (obiektów klasy Cells), które zostają zainicjowane w tablicy w obiekcie klasy Board. Zera oznaczają stan komórki jako "pusta", jedynki jako "głowa elektronu", dwójki jako "ogon elektronu", a trójki jako "przewodnik".

## 5.4 Test modułu View

Test polega na sprawdzeniu sfery wizualnej oraz reakcji programu na poszczególne działania użytkowników (przypadki naciśnięcia przynicku lub wpisania tekstu).