

# Specyfikacja implementacyjna – Gra w życie

Krzysztof Dąbrowski i Jakub Bogusz

12 maja 2019

# Spis treści

<b>1</b>	<b>Opis klas</b>	<b>2</b>
1.1	Package „Controllers” . . . . .	2
1.1.1	CellularAutomatonController . . . . .	2
1.1.2	GameOfLifeController . . . . .	4
1.2	Package „Models” . . . . .	4
1.2.1	CellularAutomaton . . . . .	5

# Rozdział 1

## Opis klas

### 1.1 Package „Controllers”

Package składający się z klas mających na celu połączenie graficznego interfejsu użytkownika z logiką działania automatów komórkowych. Będzie zawierać 3 klasy, jedną ogólną „CellularAutomatonController”, łączącą w sobie cechy wspólne obsługi interfejsu obydwu automatów, oraz z dwóch klas dziedziczących z poprzedniej, zawierających elementy różne dla GameOfLife i WireWorld.

#### 1.1.1 CellularAutomatonController

##### Pola

##### Pola chronione:

- `protected Canvas canvas` - płótno na którym rysowana będzie plansza,
- `protected Slider zoomSlider` - suwak reprezentujący przybliżenie planszy ,
- `protected Slider speedSlider` - suwak reprezentujący prędkość wyświetlania kolejnych generacji w trybie automatycznym,
- `protected ToggleButton autoRunToggleButton` - przycisk włączający i wyłączający tryb automatyczny,
- `protected Button nextGeneration` - przycisk służący do stworzenia i wyświetlenia kolejnej generacji,
- `protected Button previousGeneration` - przycisk służący do wyświetlenia poprzedniej generacji,
- `protected Spinner<Integer> widthSpinner` - pole reprezentujące szerokość generowanej planszy,

- `protected Spinner<Integer> heightSpinner` - pole reprezentujące wysokość generowanej planszy,
- `protected Button RandomButton` - przycisk służący do wygenerowania i wyświetlenia losowej planszy początkowej,
- `protected Button EmptyButton` - przycisk służący do wygenerowania i wyświetlenia pustej planszy początkowej,
- `protected Button saveButton` - przycisk służący do zapisania aktualnego stanu planszy,
- `protected Button loadButton` - przycisk służący do wczytania planszy,
- `protected MenuButton menuButton` - przycisk służący do zapisania części planszy lub narysowania i zapisania wzoru,
- `protected Label generationLabel` - napis reprezentujący numer aktualnie wyświetlanej generacji,
- `protected CellularAutomatonView cellularAutmatonView` - obiekt odpowiedzialny za narysowanie planszy.

#### Pola prywatne:

- `private Boolean running` - zmienna typu prawda/fałsz, określająca czy tryb automatyczny jest włączony,
- `private Thread t` - wątek w którym generowane i wyświetlane są kolejne pokolenia w trybie automatycznym,
- `private long delay` - odstęp czasowy między wyświetlaniem kolejnych generacji w trybie automatycznym.

#### Metody

##### Metody publiczne:

- `public Controller(Slider speedSlider, Canvas canvas, Slider zoomSlider, ToggleButton autoRunToggleButton, Button previousGenerationButton, Button nextGenerationButton, Spinner widthSpinner, Spinner heightSpinner, Button randomButton, Button emptyButton, Button saveButton, Button loadButton, Label generationNumberLabel)` - metoda odpowiedzialna za zainicjowanie wszystkich zmiennych, połączenie elementów graficznym z odpowiednimi metodami,
- `public void setCanvas` - metoda dostępowa pozwalająca ustawić wartość pola `canvas` funkcjom spoza tego pakietu.

### Metody chronione:

- `protected void shrinkSlider()` - metoda odpowiedzialna za dopasowanie maksymalnej wartości suwaka przybliżenia, tak aby wielkość wyświetlanego obrazu mieściła się w maksymalnym rozmiarze płótna,
- `protected void enableButtons()` - metoda odpowiedzialna za aktywowanie przycisków, które przy starcie programu były nieaktywne ze względu na brak funkcjonalności,
- `protected generationNumberChanged(ObservableValue<? extends Number> observable, Number oldValue, Number newValue)` - metoda odpowiedzialna za aktywowanie i dezaktywowanie przycisku `previousGeneration`, gdy wyświetlenie poprzedniej generacji jest nie możliwe.

### Metody prywatne:

- `private createThread()` - metoda odpowiedzialna za stworzenie nowego wątku `t`,
- `private zoomSliderChanged(ObservableValue<? extends Number> observable, Number oldValue, Number newValue)` - metoda odpowiedzialna za zmianę rozmiaru rysowanych komórek, na podstawie wartości suwaka przybliżenia,
- `private speedSliderChanged(ObservableValue<? extends Number> observable, Number oldValue, Number newValue)` - metoda odpowiedzialna za zmianę prędkości generowania i wyświetlania kolejnych generacji, na podstawie wartości suwaka prędkości,
- `private void nextGeneration(Event event)` - metoda odpowiedzialna za przekazanie informacji do modelu automatu komórkowego, o tym że należy wygenerować następne pokolenie,
- `private void previousGeneration(Event event)` - metoda odpowiedzialna za przekazanie informacji do modelu automatu komórkowego, o tym że należy wygenerować poprzednie pokolenie,
- `private play()` - metoda odpowiedzialna za uruchomienie tryb automatycznego.

### 1.1.2 GameOfLifeController

## 1.2 Package „Models”

Package składający się z klas reprezentujących odpowiednie automaty komórkowe, odpowiedzialnych za przechowywanie ich zasad, przeprowadzanie symulacji i generowanie kolejnych pokoleń. Będzie on zawierać 3 klasy, jedną ogólną `CellularAutomaton`, łączącą w sobie cechy wspólne wszystkich automatów

komórkowych oraz 2 klasy dziedziczące z poprzedniej, opisujące działanie konkretnych automatów (`GameOfLife` oraz `WireWorld`).

### 1.2.1 CellularAutomaton

```
public abstract class CellularAutomaton<T extends Enum>
```

Klasa abstrakcyjna reprezentująca dowolny automat komórkowy. Typ `T` jest typem wyliczeniowym możliwych stanów komórki automatu.

#### Konstruktory:

- `public CellularAutomaton(int width, int height)` – Tworzy automat o podanym rozmiarze z komórkami o domyślnym stanie.

#### Pola chronione:

- `protected final int width` – Liczba kolumn planszy automatu,
- `protected final int height` – Liczba wierszy planszy automatu,
- `protected T[] cells` – Tablica wszystkich komórek automatu,
- `protected static Random random` – Zmienna używana do losowania stanów.

#### Pola prywatne:

- `private List<T[]> history` – Lista poprzednich stanów automatu. Umożliwia przejście od poprzedniego stanu,
- `private IntegerProperty currentGeneration` – Liczba reprezentująca number aktualnego pokolenia automatu.

#### Metody publiczne:

- `public abstract T[] getPossibleCellValues()` – Zwraca tablicę możliwych stanów komórki automatu,
- `public void setCells(T[] cells)` – Ustawia wszystkie komórki automatu na nowe wartości,
- `public T getCell(int row, int column)` – Zwraca wartość konkretnej komórki,
- `public int getCellCount()` – Zwraca ilość komórek w automacie,
- `public void nextGeneration()` – Przeprowadza automat do następnego stanu,

- `public void previousGeneration()` – Przeprowadza automat do poprzedniego stanu
- `public void clear()` – Ustawia stan wszystkich komórek na domyślny,
- `public abstract void randomize()` – Ustawia stan wszystkich komórek na losowy.

#### **Metody chronione:**

- `protected abstract T[] generateNextGeneration()` – Zwraca stany komórek następnej generacji,
- `protected abstract T getDefaultState()` – Zwraca domyślny stan dla danego automatu.

#### **Metody prywatne:**

- `private void clearHistory()` – Ustawia aktualny stan automatu jako jedyny stan w historii.