Specyfikacja implementacyjna – Gra w życie

Krzysztof Dąbrowski i Jakub Bogusz

14 maja 2019

Spis treści

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 ogólna "CellularAutomatonController", łączącą w sobie cechy wspólne obsługi interfejsu obydwu automatów, oraz z dwóch klas dziedziczących z poprzedniej, zawierających elementu różne dla GameOfLife i WireWorld.

1.1.1 CellularAutomatonController

Klasa łącząca w sobie wspólne cechy kontrolerów obydwu automatów skończonych - opisująca powtarzające się elementy graficznego interfejsu użytkownika i ich funkcjonalności.

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 cellularAutomatonView` - 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

Konstruktory:

- `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,

Metody publiczne:

- `public void setCanvas` - metoda dostępowa pozwalająca ustawić wartość pola `canvas` funkcjom spoza tego pakietu.

Metody chronione:

- `protected void shrinkSlider()` - metoda odpowiedzialna za dopasowanie maksymalnej wartość 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

Klasa opisująca części kontrolera specyficzne dla automatu GameOfLife.

Pola

Pola chronione:

- `private RadioButton aliveRadioButton` - checkbox oznaczający, że rysowane komórki będą żywe,
- `private RadioButton deadRadioButton` - checkbox oznaczający, że rysowane komórki będą martwe.

Pola prywatne:

- `private Map<GameOfLife.CellStates, Paint> coloring` - mapa przyporządkowująca stanowi automatu kolor (czarny - martwa komórka, biały - żywa komórka).

Funkcje

Konstruktory:

- `public GameOfLifeController(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, RadioButton aliveRadioButton, RadioButton deadRadioButton)` - inicjuje wszystkie zmienne, łączy elementy graficzne z odpowiednimi funkcjami.

Metody publiczne:

- `public Map<GameOfLife.CellStates, Paint> getColoring()` - metoda dostępowa pozwalająca ustawić wartość pola `coloring` funkcjom spoza tego pakietu.

Metody prywatne:

- `private void randomizeBoard(Event event)` - metoda odpowiedzialna za stworzenie modelu GameOfLife z losowo wygenerowaną planszą początkową,
- `private void clearBoard(Event event)` - metoda odpowiedzialna za stworzenie modelu GameOfLife z pustą planszą początkową,
- `private void canvasClicked(Event event)` metoda odpowiedzialna za przekazanie informacji do modelu automatu komórkowego, o tym że należy na podstawie zaznaczonych checkbox'ów należy zmienić stan odpowiedniej komórki oraz ją narysować,
- `private GameOfLife.CellStates getSelectedState()` - metoda odpowiedzialna za pobranie aktualnie wybranego stanu na podstawie zaznaczonych zaznaczonych checkbox'ów.

1.1.3 WireWorldController

Klasa opisująca części kontrolera specyficzne dla automatu WireWorld.

Pola

Pola chronione:

- `private RadioButton emptyRadioButton` - checkbox oznaczający, że rysowane komórki będą puste,
- `private RadioButton headRadioButton` - checkbox oznaczający, że rysowane komórki będą głowami elektronu,
- `private RadioButton tailRadioButton` - checkbox oznaczający, że rysowane komórki będą ogonami elektronu,
- `private RadioButton conductorRadioButton` - checkbox oznaczający, że rysowane komórki będą przewodnikami,
- `private Button powerOff` - przycisk zamieniający wszystkie komórki będące głowami lub ogonami elektronu na przewodniki.

Pola prywatne:

- `private Map<WireWorld.CellStates, Paint> coloring` - mapa przyporządkowująca stanowi automatu kolor (czerwony - głowa elektronu, niebieski - ogon elektronu, żółty - przewodnik, czarny - pusta komórka).

Funkcje

Konstruktory:

- `public WireWorld(Button powerOff, 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, RadioButton emptyRadioButton, RadioButton tailRadioButton, , RadioButton headRadioButton, , RadioButton conductorRadioButton)` - inicjuje wszystkie zmienne, łączy elementy graficzne z odpowiednimi funkcjami.

Metody publiczne:

- `public Map<WireWorld.CellStates, Paint> getColoring()` - metoda dostępowa pozwalająca ustawić wartość pola `coloring` funkcjom spoza tego pakietu.

Metody prywatne:

- `private void randomizeBoard(Event event)` - metoda odpowiedzialna za stworzenie modelu WireWorld z losowo wygenerowaną planszą początkową,
- `private void clearBoard(Event event)` - metoda odpowiedzialna za stworzenie modelu WireWorld z pustą planszą początkową,
- `private void canvasClicked(Event event)` metoda odpowiedzialna za przekazanie informacji do modelu automatu komórkowego, o tym że należy na podstawie zaznaczonych checkbox'ów należy zmienić stan odpowiedniej komórki oraz ją narysować,
- `private GameOfLife.CellStates getSelectedState()` - metoda odpowiedzialna za pobranie aktualnie wybranego stanu na podstawie zaznaczonych zaznaczonych checkbox'ów,
- `private void powerOff(Event event)` - metoda odpowiedzialna za przekazanie informacji do modelu automatu komórkowego o tym, że należy zamienić wszystkie komórki będące głowami i ogonami elektronu na przewodniki.

1.2 Pakiet „Models"

Pakiet 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.

1.2.2 GameOfLife

`public class GameOfLife extends CellularAutomaton<GameOfLife.CellStates>`
Klasa jest konkretną implementacja automatu komórkowego *Game of Life*.

Klasy wewnętrzne:
Typ wyliczeniowy reprezentujący możliwe stany komórki.

```
public enum CellStates {
    DEAD,
    ALIVE;

    public static CellStates randomState() - Zwraca losowy stan komórki.
}
```

Konstruktory:

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

Metody prywatne:

- `private int[] countAliveNeighbors()` – Oblicza ilość żywych sąsiadów dla każdej komórki automatu,
- private int countAliveNeighbors(final int cellX, final int cellY) – Oblicza ilość żywych sąsiadów dla konkretnej komórki automatu.

Klasa implementuje poniższe metody abstrakcyjne z klasy bazowe:

- `CellStates[] getPossibleCellValues()`,
- `protected CellStates[] generateNextGeneration()`,
- `public void randomize()`,
- `protected CellStates getDefaultState()`.

Szczegółowy opis każdej z metod znajduje się w klasie bazowej CellularAutomaton.

1.2.3 WireWorld

`public class WireWorld extends CellularAutomaton<WireWorld.CellStates>`
Klasa jest konkretną implementacja automatu komórkowego *WireWorld*.

Klasy wewnętrzne:
Typ wyliczeniowy reprezentujący możliwe stany komórki.

```
public enum CellStates {
    EMPTY,
    HEAD,
    TAIL,
    CONDUCTOR;

    public static CellStates randomState() - Zwraca losowy stan komórki.
}
```

Konstruktory:

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

Metody publiczne:

- `public void killElectrons()` – Zamienia wszystkie głowy elektronów i ogony elektronów na przewodniki.

Metody prywatne:

- `private int countHeads(final int cellX, final int cellY)` – Oblicza ilość sąsiadów konkretnej komórki będących głową elektronu,
- `private int[] countHeads()` – Dla każdej komórki oblicza ilość sąsiadów będących głową elektronu.

Klasa implementuje poniższe metody abstrakcyjne z klasy bazowe:

- `CellStates[] getPossibleCellValues()`,
- `protected CellStates[] generateNextGeneration()`,
- `public void randomize()`,
- `protected CellStates getDefaultState()`.

Szczegółowy opis każdej z metod znajduje się w klasie bazowej CellularAutomaton.

Rozdział 2

Projekt systemu

2.1 Digram klas

2.2 Wzorce projektowe

2.3 Przepływ sterowania

2.4 Zastosowane algorytmy

Rozdział 3

Testy

Rozdział 4

Biblioteki