Dokumentacja końcowa programu WireWorld

Danuta Stawiarz, Katarzyna Stankiewicz

20 maja 2019 r.

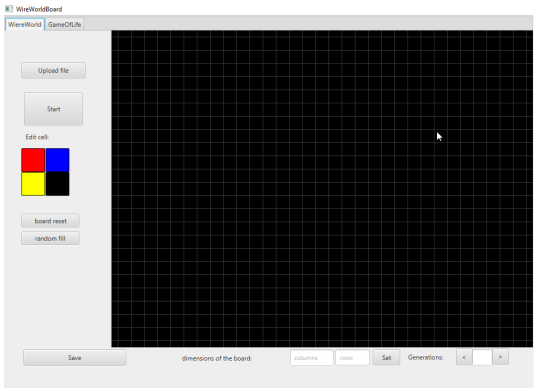
Spis treści

1 Informacje ogólne

W projekcie zastosowany będzie wzorzec MVC (Model View Controller). W projekcie wprowadzony został przejrzysty przydział poszczególnych plików do odpowiednich pakietów. W pakiecie models znajdują się 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 oraz pliki odpowiedzialne za wyświetlanie interfejsu. W pakiecie controllers będą zawarte kontrolery, służące do obsługi żądań użytkownika.

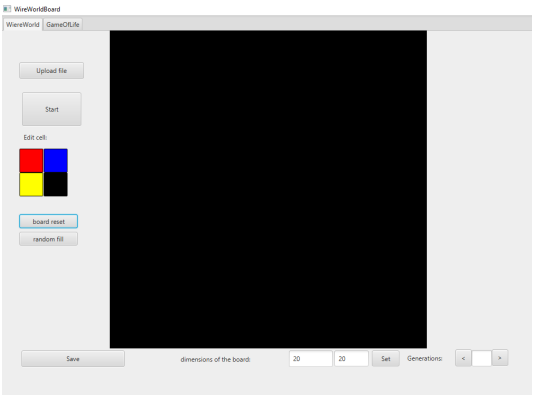
1.1 Działanie programu

1. **Wariant Wireworld**



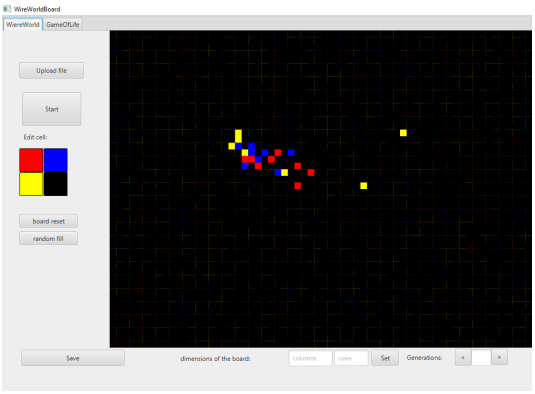
Rysunek 1: Ekran startowy aplikacji

W celu zmiany wymiarów planszy użytkownik wprowadza oczekiwane wymiary oraz naciska "Set"



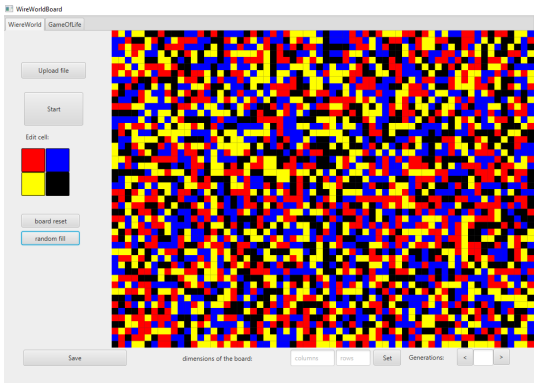
Rysunek 2: Zmiana wymiarów planszy przez użytkownia

W celu wypełnienia planszy według własnych oczekiwań użytkownik wybiera kolor z palety dostępnej po lewej stronie planszy oraz wybiera komórkę na planszy.



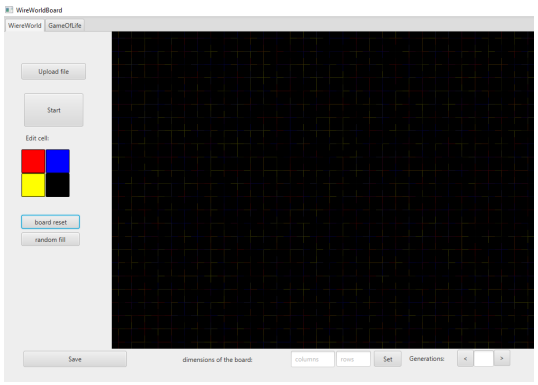
Rysunek 3: Wypełnienie planszy

W celu wybrania losowego ustawienia komórek użytkownik naciska "Random Fill".



Rysunek 4: Losowe wypełnienie planszy

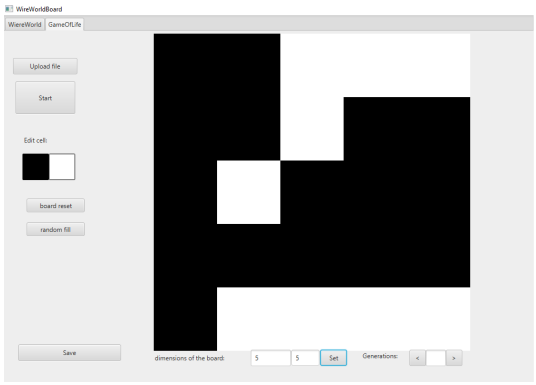
W celu zresetowania planszy użytkownik naciska "Reset Board"



Rysunek 5: Resetowanie planszy

2. **Wariant Game of Life**
   Zasady obsługi nie różnią się od obsługi wariantu gry Wireworld.

Rysunek 6: Ekran startowy



Rysunek 7: Wypełnianie planszy przez użytkownia

2 Schemat działania programu

Program reaguje bezpośrednio na komendy użytkownika.

3 Diagram klas

1. Uruchomienie aplikacji - ekran startowy. Fragment przedstawiający siatkę komórek jest czarny - komórki domyślnie są martwe lub puste w zależności od wariantu gry

2. Reakcje na bezpośrednie działania użytkownika:

   - Start - naciśnięcie tego przycisku uruchomi symulację,

- Generations - w przypadku naciśnięcia którejś ze strzałek nastąpi przeprowadzenie generacji. Wynik symulacji zostanie wyświetlony na ekranie aplikacji,
- Board reset - nastąpi zresetowanie tablicy reprezentującej siatkę- na ekranie pojawi się czarna tablica.
- Upload file - spowoduje wczytanie siatki ze stanami poszczególnych komórek i wyświetlenie jej na ekranie
- Random fill - spowoduje wypełnienie siatki looswymi wartościami,
- Save - spowoduje zapis stanu tablicy do pliku
- Set - wprowadzenie w pola odpowiednich liczb ustawi wybrany rozmiar siatki komórek
- Edit - po kliknięciu wybraną komórkę na planszy, użytkownik zmienia jej stan oraz reprezentację na planszy (kolor). W celu wybrania koloru użytkownik wybiera go z palety dostępnej na ekranie aplikacji.

4 Zmiany w stosunku do specyfikacji funkcjonalnej oraz implementacyjnej

W zakładanym wyglądzie oraz aspekcie funkcjinalnym programu nie zaszły znaczące zmiany.Zmianie uległ kod projektu. Moduły uległy znacznemu rozszerzeniu, w celu ułatwienia pracy nad projektem i zachowaniu spójności, zmianie uległy nazwy klas. Projekt został podzielony zgodnie z modelem Model-View-Controller. W module Model, poza klasami, których istnienie przewidywała specyfikacja implementacyjna pojawiły się klasy reprezentujące tablicę komórek (Cellls) oraz klasy dziedziczące po niej. W module View znalazły się dodatkowe klasy odpowiadające za wygląd aplikacji, jest to głównie klasa Board, która pierwotnie miała być elementem modułu Model. Pośrednio jej rolę przejęła klasa Cells, co umożliwiło rozdzielenie dwóch modułów.

5 Opis modułów i klas

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

- Models - komputerowa reprezentacja gry, odpowiada za przeprowadzanie kolejnych generacji, jak i całej symulacji
- Controllers - odpowiada za odbiór, przetworzenie oraz analizę danych wejściowych od użytkownika, po przetworzeniu danych zmienia stan modelu oraz odświeża widok
- View - moduł odpowiadający za apsekt wizualny gry, a także reakcję na działana użytkownika (przekazuje informację o przypadkach naciśnięcia poszczególnych przycisków)

5.1 Moduł Model

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

1. **Abstract Class Game extends Observable()** - klasa ta stanowi podstawę dla klas dziedziczących po niej - Wireworld oraz Life, związane jest to z wariantem gry, który wybierze użytkownik. Klasa ta dziedziczy po klasie Observable. Elementami tej klasy są:

   - protected Cells cells - jest to obiekt klasy, która reprezentuje wszystkie komórki zawarte na planszy (siatkę komórek)

   Główne metody tej klasy to:

   - Metody abstrakcyjne:
     - public void initCellsBoard() - metoda przypisująca dane do obiektu cellsBoard, który jest elementem klasy cells, która to stanowi element klasy Game
     - public abstract void play() - przeprowadza symulację generacji zgodnie z zasadami określonymiprzez wariant gry
     - public abstract void readStates(int []) - metoda odczytująca stany poszczególnych komórek na podstawie ich reprezentacji w tablicy typu int oraz odzwierciedlająca stany komórek w jednym z obiektów tej klasy - CellsBoard
     - public abstract void readStatesFromCells() - metoda odczytująca stany komórek z tablicy obiektów typu CellState oraz odzwierciedlająca je w obiekcie tej klasy o nazwie cellls
     - public abstract void setCellsBoard(Cell.State[]) - metoda tworząca tablicę obiektów typu Cell będącej zmienną tej klasy, na podstawie przekazanej tablicy typu Cell.State[]
     - public abstract class play() - metoda odpowiadająca za przeprowadzenie symulacji generacji komórek, a więc za właściwą funckjonalność programu. W klasach dziedziczących jest implementowana w zależności od zasad gry.
   - Metody typu "get" i "set":
     - public Cells getCells ()
     - public void setCells (Cells cells)

   Klasy dziedziczące po klasie Game reprezentują poszczególne warianty gry, które różnią się miiędzy sobą głównie wartościami zmiennych reprezentującymi stan komórek oraz metodami odpowiadającymi za przeprowadzenie symulacji (metoda play() oraz metody, z których korzysta). Klasy dziedziczące po klasie Game to:

(a) **public class GameOfLife extends Game** - klasa ta jest pochodną klasy Game.Elementami tej klasy są:

- public golCells Cells,

Klasa zawiera następujące metody:

- public GameOfLife (golCells cellls) - konstruktor klasy
- public void countAliveNextCells() - metoda charakterystyczna dla tego wariantu gry, zlicza ilość obiektów typu Cell o stanie "Alive" w najbliższym sąsiedztwie wybranego obiektu na siatce obiektów typu Cell
- Override public void setCellsBoard(Cell.State[ ] states)
- Override public void readStates(int[ ] intStates)
- Override public void readStatesFromCells()
- Override public void play()

(b) **public class Wireworld extends Game** Elementami tej klasy są:

- protected Cells cells
- public void initCellsBoard() - metoda przypisująca dane do obiektu cellsBoard, który jest elementem klasy cells, która to stanowi element klasy Game;
- public abstract void play() - przeprowadza symulację generacji zgodnie z zasadami określonymi przez wariant gry;
- public abstract void readStates(int []) -
- public abstract void readStatesFromCells()
- public abstract void setCellsBoard(Cell.State[])
- public Cells getCelsl ()
- public void setCelsl (Cells cells)

2. **public abstract class Cell** - klasa reprezentująca pojedynczą komórkę w grze, a więc najmniejszy podstawowy obiekt bedący składowymi takich klas jak Cells oraz Game. W dalszej częsci opisu obiekty tej klasy będą nazywane komórkami. Zawiera elementy:

   - private int x - współrzędna określająca położenie x komórki na planszy,
   - private int y - współrzędna określająca położenie y komórki na planszy,
   - private double size - określa rozmiar komórki,
   - private State state - określa stan komórki,
   - public enum State - zawarte są tu wszystkie możliwe wartości stanu komórek dla obu wariantów gry

   Główne metody tej klasy to metody typu "get" oraz "set". Są to:

- public void seState(State state)
- public State getState()
- public void setSize(double)
- public double getSize()
- public double getX()
- public double getY()
- public void setX(double)
- public void setY(double)

Klasy dziedziczące po klasie Cell:

(a) **public class golCell extends Cell** - klasa dziedzicząca po klasie Cell, poza elementami właściwymi dla tej klasy, zawiera także zmienne:

- protected int aliveNextCells - zmienna charakterystyczna dla tego wariantu gry, odzwierciedla ilość obiektów o stanie "Alive" w siatce obiektów typu Cell dla wybranego obiektu

Inne metody tej klasy:

- public State checkState() - sprawdza wartość zmiennej State, na podstawie uzyskanych danych

(b) **public class wwCell extends Cell**- klasa dziedzicząca po klasie Cell, poza elementami właściwymi dla tej klasy, zawiera także zmienne:

- protected int headsNext

3. **public abstract class Cells** klasa abstrakcyjna reprezentująca siatkę komórek gry, zależenie od jej wariantu. Zawiera elementy:

- protected int rows - wymiar x siatki komórek,
- protected int columns - wymiar y siatki komórek,
- protected double cellSize - wielkość pojedynczej komórki w siatce (zmienna wraz ze zmiennymi wymiarami siatki),
- protected Cell.State[] cellsStates - tablica typu Cell.State reprezentująca stany poszczególnych komórek w siatce
- protected int numberOfCells- ilość komórek w siatce,
- protected Cell[] cellsBoard - tablica obiektów typu Cell, a więc reprezentująca siatkę

Główne metody tej klasy to metody typu "get" oraz "set":

- public int getRows (),
- public void setRows(int),

- public int getColumns(),
- public void setColumns(int),
- public double getCellSize(),
- public void setCellSize(double),
- public Cell.State[] getCellsStates(),
- public void setCellsStates(Cell.State[]),
- public int getNumberOfCells(),
- public void setNumberOfCells(int),
- public Cell[] getCellsBoard(),
- public Cell getCellsBoard(int),
- public void setCellsBoard(Cell[]),

Klasy dziedziczące po tej klasie:

(a) **g**olCells - klasa dla wariantu gry Game of Life,

(b) **w**wCells - klasa dla wariantu gry Wireworld.

4. **public abstract class Observabler** - klasa zawiera metody:
   - public void subscribe(Observatorr),
   - public void notifyObservators().
5. **public interface Observator**

5.2 Moduł Controller

Klasa ta odpowiada za właściwą komunikację pomiędzy interfejsem użytkownika (moduł View) a właściwym działaniem programu (Moduł Model). Główną składową tej klasy jest metoda **Initialize()** , w której następuje inicjalizacja obiektów reprezentujących obie rozgrywki gry oraz ich składowych. Metoda ta wywołuje zarządza działaniami programu na podstawie działań użytkownika - odbiera bodźce i na ich podstawie wywołuje metody niezbędne do przeprowadzenia prawidłowej reakcji programu.

5.3 Moduł View

1. **public abstract class Board** - klasa abstrakcyjna implementuje interfejs Observator. Zawiera metody umożliwiające zauważanie, przekazanie do programu i analizę działań użytkownik oraz poźniejsze zmiany w interfejsie aplikacji. Klasa zawiera zmienne:
   - protected Game game - obecność tego obiektu tej klasy umożliwia przekazywanie oraz modyfikację danych między modułami View a Model,

- protected Canvas canvas - obiekt odpowiadający stricte za wygląd planszy,
- private int clickedX - współrzędna x miejsca na planszy, w którym wykryto działanie użytkownika,
- private int clickedY współrzędna y miejsca na planszy, w którym wykryto działanie użytkownika,
- private boolean selectionActive,
- protected Color selectedColor- zmienna określająca kolor.

Metody klasy:

- metody:
  - public Board(Gam, Canvas) - konstruktor klasy
  - public void setDimension(TextField, TextField) - metoda obsługująca działanie użytkownika polegające na wyborze wymiarów siatki,
  - public void cleanCanvas(Canvas) - metoda obsługująca działanie użytkownika polegaące na żądaniu wyczyszczenia planszy (ustawienie płótna na "czarne"),
  - public void drawSingeCell(Canvas, int, int , Cell.State) - metoda zmieniająca wygląd pola planszy, które reprezentuje konkretną komórkę,
  - public void blackFill(Canvas) - wypełnienie płótna czarnym kolorem,
  - public void setPromptForDimensions(TextField, TextField)-
  - public void setCell(Cell.State, int, int , boolean) - metoda zmieniająca zmienną "state" komórki na podstawie koloru, jaki reprezentuje ją na planszy widpcznje dla użytkownika,
  - public void boardClicked(MouseEvent) - metoda obsługująca kompleksowo przypadek zarejestrowania działania użytkownika w obrębie planszy (użytkownik ma możliwość zmiany stanu komórki przez wybranie miejsca, w którym sie znajduje)
  - public void onUpdate() - metoda uaktualniająca obiekt typu game, na podstawie zmian dokonanych w zmiennych "state" obiektów typu Cell (komórek)
- mettody typu "get" i "set" :
  - public void setCanvas(Canvas) ,
  - public Canvas getCanvas(),
  - public Color getSelectedColor(),
  - public void setSelectedColor(Color).
- metody abstrakcyjne:

- public abstract void draw(Cell.State[]) - metoda "rysująca" planszę na podstawie przekazanej tablicy typu Cell.State reprezentującej stany poszczególnych komórek,
- public abstract void randomFill(Game game) - metoda odpowiadająca za wypełnienie pól planszy losowymi kolorami reprezentującymi stany komórek,
- public abstract Cell.State pickState() - metoda zmieniająca wartość zmiennej "state" obiektu klasy Cell (stan komórki) na podstawie reprezentowanego przez nią na planszy koloru)

Klasy dziedziczące po tej klasie to:

(a) **public class GameofLifeBoard** - klasa odpowiadająca wariantowi gry "Game of Life"

(b) **public class WireworldBoard** - klasa odpowiadająca wariantowi gry "Wireworld"

Klasy te nadpisują metody abstrakcyjne klasy Board głównie w zależności od ilości możliwych stanów komórek oraz ich reprezentacji w interfejsie użytkownika.

2. **Game.fxml** - dokument opisujący wygląd Graficznego Interfejsu Użytkownika.

6 Zastosowane algorytmy

W programie zostały zastosowane następujące algorytmy:

6.1 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`.

6.2 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.

7 Testy

7.1 Test modułu Models

7.2 Test modułu View

7.3 Testy zapisu do pliku i odczytu z pliku