Politechnika Warszawska
Wydział Elektryczny

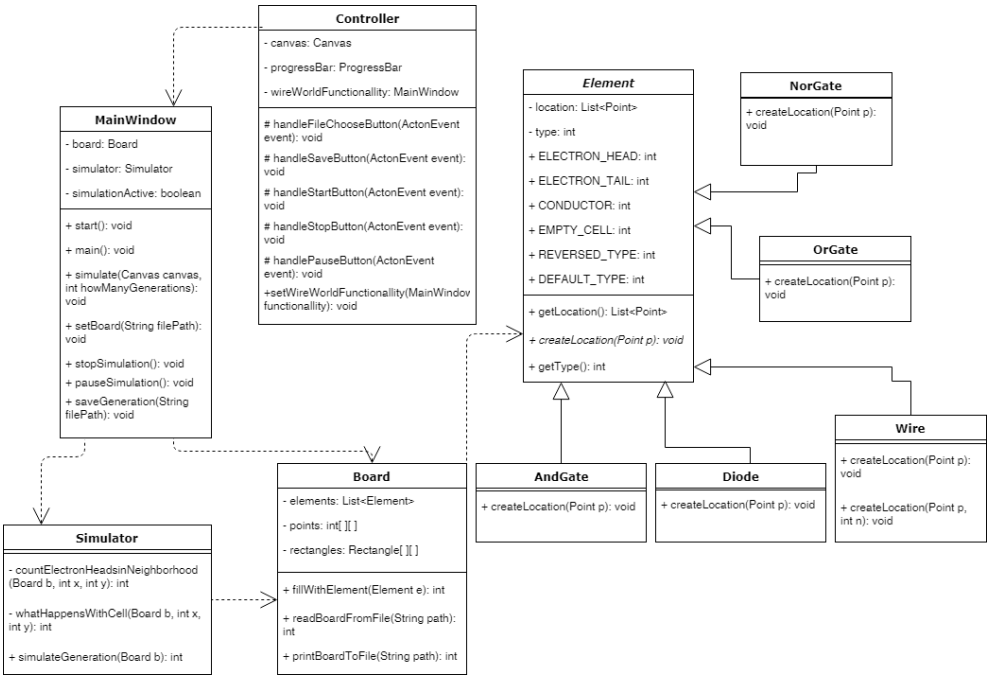
Specyfikacja implementacyjna symulatora automatu komórkowego *Wireworld*

Autorzy:
J. Korczakowski, nr albumu 291079
B. Suchocki, nr albumu 291111
Grupa projektowa nr 11

9 maja 2018

Spis treści

1 Diagram klas



Rysunek 1: Diagram klas

2 Opis klas

2.1 Simulator

Odpowiada za symulację kolejnych generacji komórek.
Pola: `brak`.
Metody:

1. `private int countElectronHeadsInNeighbourhood(Board b, int x, int y)`.
   - Argumenty:
     - `b` - obiekt klasy `Board` reprezentujący bieżący stan generacji komórek,
     - `x,y` - współrzędne komórki, dla której zliczana jest ilość sąsiadujących z nią `głów elektronu`.
   - Wartość zwracana:
     - Ilość `głów elektronu` sąsiadujących z daną komórką.
   - Działanie:

- Funkcja zlicza ilość `głów elektronu` w sąsiedztwie komórki o podanym położeniu według zasad sąsiedztwa Moore'a.

2. `private int whatHappensWithCell(Board b, int x, int y).`

   - Argumenty:
     - `b` - obiekt klasy `Board` reprezentujący bieżący stan generacji komórek,
     - `x,y` - współrzędne komórki, dla której ustalany jest nowy stan.
   - Wartość zwracana:
     - Liczba (stała) określająca nowy stan analizowanej komórki.
   - Działanie:
     - Funkcja, korzystając z funkcji `countElectronHeadsInNeighbourhood` określa czy, w następnej generacji, będzie `głową elektronu`, `ogonem elektronu`, `przewodnikiem` czy `pustą komórką`. Dla każdego z takich stanów zwraca liczbę całkowitą (stałą) go symbolizującą. Odpowiednio: `ELECTRON_HEAD`, `ELECTRON_TAIL`, `CONDUCTOR`, `EMPTY_CELL`.

3. `public int simulateGeneration(Board b).`

   - Argumenty:
     - `b` - obiekt klasy `Board` reprezentujący bieżący stan generacji komórek,
   - Wartość zwracana:
     - Liczba całkowita okrelająca czy symulacja zakończyła się powodzeniem (0 - jeśli tak, 1 - jeśli nie).
   - Działanie:
     - Metoda symuluje przejście komórek do następnej generacji. Symulacja wykonywana jest poprzez iteracyjne sprawdzanie poprzedniego stanu analizowanej komórki oraz zliczanie (przy użyciu funkcji `countElectronHeadsInNeighbourhood`) `głów elektronu` w jej sąsiedztwie . Na podstawie tej analizy (przy użyciu metody `whatHappensWithCell` nastąpi określenie, do jakiego stanu przejdzie dana komórka. W wyniku działania metody, zmodyfikowany zostanie obiekt b przekazany jako argument jej wywołania. Po zakończeniu działania metody, będzie on przechowywać stan aktualnej generacji (po symulacji).

2.2 Controller

Odpowiada za odpowiadanie na akcje użytkownika (np. wciśnięcie przycisku).

Pola:

- `private Canvas canvas` - graficzna reprezentacja planszy,
- `private ProgressBar progressBar` pasek wizualizujący postęp symulacji,
- `private MainWindow wireWorldFunctionallity` - obiekt potrzebny do używania funkcjonalności automatu WireWorld w odpowiedzi na akcje użytkownika.

Metody:

1. `public void setWireWorldFunctionallity(MainWindow functionallity)`
   - Argumenty:
     - `functionallity` - obiekt klasy MainWindow umożliwiający korzystanie z funkcjonalności automatu WireWorld w odpowiedzi na akcje użytkownika.
   - Wartość zwracana:
     - Void
   - Działanie:
     - Metoda ustawia pole wireWorldFunctionallity na argument wywołania.
2. `protected void handleFileChooseButton(ActionEvent event).`
   - Argumenty:
     - `event` - obiekt klasy `ActionEvent` reprezentujący zdarzenie naciśnięcia przycisku.
   - Wartość zwracana:
     - Void
   - Działanie:
     - Metoda, w odpowiedzi na naciśnięcie przycisku wyboru pliku weściowego przez użytkownika, wyświetla dialog, w którym użytkownik może wybrać plik wejściowy z odpowiedniej lokalizacji. Po wyborze, ścieżka do pliku wświetlona zostaje w+polu poniżej przycisku i wywołana zostaje metoda `setBoard` obiektu klasy MainWindow (jako argument wywołania, przekazywana jest lokalizacja pliku wejściowego).

3. `protected void handleStartButton(ActionEvent event).`

    - Argumenty:
        - `event` - obiekt klasy `ActionEvent` reprezentujący zdarzenie naciśnięcia przycisku.
    - Wartość zwracana:
        - Void
    - Działanie:
        - Metoda, w odpowiedzi na naciśnięcie przycisku startu przez użytkownika, wywołuję metodę `simulate` obiektu klasy MainWindow.

4. `protected void handleStopButton(ActionEvent event).`

    - Argumenty:
        - `event` - obiekt klasy `ActionEvent` reprezentujący zdarzenie naciśnięcia przycisku.
    - Wartość zwracana:
        - Void
    - Działanie:
        - Metoda, w odpowiedzi na naciśnięcie przycisku stopu przez użytkownika, wywołuje metodę `stopSimulation` obiektu klasy MainWindow.

5. `protected void handlePauseButton(ActionEvent event).`

    - Argumenty:
        - `event` - obiekt klasy `ActionEvent` reprezentujący zdarzenie naciśnięcia przycisku.
    - Wartość zwracana:
        - Void
    - Działanie:
        - Metoda, w odpowiedzi na naciśnięcie przycisku pauzy przez użytkownika, wywołuje metodę `pauseSimulation` obiektu klasy MainWindow.

6. `protected void handleSaveButton(ActionEvent event).`

    - Argumenty:
        - `event` - obiekt klasy `ActionEvent` reprezentujący zdarzenie naciśnięcia przycisku.

- Wartość zwracana:
  - Void
- Działanie:
  - Metoda, w odpowiedzi na naciśnięcie przycisku zapisu przez użytkownika, wyświetla dialog umożliwiający wybranie ścieżki do pliku wyjściowego oraz uruchamia metodę `saveGeneration` obiektu klasy MainWindow, przekazując lokalizację pliku jako argument wywołania.

2.3 MainWindow

Jest modułem realizującym wszystkie funkcjonalności automatu komórkowego `WireWorld`.

Pola:

- `private Board board` - obiekt przechowujący stan kolejnych generacji,
- `private Simulator simulator` obiekt umożliwiający symulację kolejnych generacji,
- `private boolean simulationActive` - zmienna służąca monitorowaniu czy wątek przeprowadzający kolejne symulacje powinien zakończyć działanie.

Metody:

1. `public void start()`.
   - Argumenty: `brak`.
   - Wartość zwracana:
     - Void
   - Działanie:
     - Metoda inicjalizuje pola klasy oraz wyświetla graficzny interfejs użytkownika.
2. `public static void main(String[] args)`.
   - Argumenty:
     - `args` - ewentualne argumenty wywołania programu (jeśli program uruchomiony został z wiersza poleceń).
   - Wartość zwracana:
     - Void

- Działanie:
  - Metoda uruchamia metodę `start`.

3. `public void simulate(Canvas canvas, int howManyGenerations).`

   - Argumenty:
     - `canvas` - obiekt klasy `Canvas` reprezentujący graficzny wygląd planszy,
     - `howManyGenerations` - liczba całkowita określające liczbę symulacji do przeprowadzenia.
   - Wartość zwracana:
     - Void
   - Działanie:
     - Metoda przeprowadza kolejne symulacje przejścia komórek do następnej generacji (korzystając z metody `simulateGeneration` obiektu klasy Simulator. Każdą kolejną generację wyświetla na graficznej planszy.

4. `public void setBoard(String filePath).`

   - Argumenty:
     - `filePath` - lokalizacja pliku wejściowego.
   - Wartość zwracana:
     - Void
   - Działanie:
     - Metoda ustawia zawartość pola `board` na podstawie pliku wejściowego, korzystając z metody `readBoardFromFile` klasy Board.

5. `public void stopSimulation().`

   - Argumenty: `brak`.
   - Wartość zwracana:
     - Void
   - Działanie:
     - Metoda kończy wcześniej rozpoczętą symulację ustawiając wartość zmiennej `simulationActive` na `false` oraz przywraca początkowy stan generacji.

6. `public void pauseSimulation().`

- Argumenty: `brak`.
- Wartość zwracana:
  - Void
- Działanie:
  - Metoda zatrzymuje wcześniej rozpoczętą symulację ustawiając wartość zmiennej `simulationActive` na `false` .

7. `public void saveGeneration(String filePath).`

   - Argumenty:
     - `filePath` - lokalizacja pliku wyjściowego, do którego zapisany będzie stan bieżącej generacji.
   - Wartość zwracana:
     - Void
   - Działanie:
     - Metoda zapisuje (korzystając z metody `printBoardToFile` obiektu klasy Board) stan bieżącej generacji do pliku, którego lokalizacją jest argument wywołania metody.

2.4 Element

Klasa abstrakcyjna opisująca pojedynczy element obwodu *Wireworld*. Dziedziczą po niej klasy, które opisują poszczególne typy obwodu.

Pola:

- `private List<Point> location` - zawiera listę punktów zajmowanych przez daną strukturę.
- `private int type` - zawiera informację czy dana struktura jest odwrócona, czy znajduje się w domyślnym położeniu
- `public static final int ELECTRON_HEAD` - zawiera stałą oznaczającą głowę elektronu.
- `public static final int ELECTRON_TAIL` - zawiera stałą oznaczającą ogon elektronu.
- `public static final int CONDUCTOR` - zawiera stałą oznaczającą przewodnik.
- `public static final int EMPTY_CELL` - zawiera stałą oznaczającą pustą komórkę.
- `public static final int REVERSED_TYPE` - zawiera stałą oznaczającą odwrócone położenie.

- `public static final int DEDAULT_TYPE` - zawiera stałą oznaczającą domyślne położenie.

Metody:

1. `public List<Point> getLocation()`.
    - Argumenty:
    - Wartość zwracana:
        - Lista zawierająca punkty(`List<Point>`).
    - Działanie:
        - Jest to metoda dostępowa zwracająca listę punktów, które zajmuje dany element.
2. `public abstract void createLocation( Point p )`.
    - Argumenty:
        - `p` - Punkt początkowy elementu.
    - Wartość zwracana:
        - `Void`
    - Działanie:
3. `public int getType()`.
    - Argumenty:
    - Wartość zwracana:
        - Typ elementu.
    - Działanie:
        - Jest to metoda dostępowa zwracająca typ elementu.

2.5 AndGate

Klasa dziedzicząca po klasie `Element`. Opisuje strukturę odpowiadającą bramce logicznej AND.

1. `public void createLocation( Point p )`.
    - Argumenty:
        - `p` - Punkt początkowy elementu.
    - Wartość zwracana:
        - `Void`
    - Działanie:
        - Metoda po otrzymaniu punktu początkowego wypełnia pole `location` punktami, które zajmuje bramka logiczna AND.

2.6 OrGate

Klasa dziedzicząca po klasie `Element`. Opisuje strukturę odpowiadającą bramce logicznej OR.

1. `public void createLocation( Point p )`.
    - Argumenty:
        - `p` - Punkt początkowy elementu.
    - Wartość zwracana:
        - `Void`
    - Działanie:
        - Metoda po otrzymaniu punktu początkowego wypełnia pole `location` punktami, które zajmuje bramka logiczna OR.

2.7 NorGate

Klasa dziedzicząca po klasie `Element`. Opisuje strukturę odpowiadającą bramce logicznej NOR.

1. `public void createLocation( Point p )`.
    - Argumenty:
        - `p` - Punkt początkowy elementu.
    - Wartość zwracana:
        - `Void`
    - Działanie:
        - Metoda po otrzymaniu punktu początkowego wypełnia pole `location` punktami, które zajmuje bramka logiczna NOR.

2.8 Diode

Klasa dziedzicząca po klasie `Element`. Opisuje strukturę odpowiadającą diodzie.

1. `public void createLocation( Point p )`.
    - Argumenty:
        - `p` - Punkt początkowy elementu.
    - Wartość zwracana:
        - `Void`
    - Działanie:
        - Metoda po otrzymaniu punktu początkowego wypełnia pole `location` punktami, które zajmuje dioda.

2.9 Wire

Klasa dziedzicząca po klasie `Element`. Opisuje strukturę odpowiadającą kablowi.

1. `public void createLocation( Point p )`.
    - Argumenty:
        - `p` - Punkt początkowy elementu.
    - Wartość zwracana:
        - `Void`
    - Działanie:
        - Metoda po otrzymaniu punktu początkowego wypełnia pole `location` punktami, które zajmuje kabel o długości domyślnej 2.
2. `public void createLocation( Point p, int n )`.
    - Argumenty:
        - `p` - Punkt początkowy elementu.
        - `n` - Długość kabla.
    - Wartość zwracana:
        - `Void`
    - Działanie:
        - Metoda po otrzymaniu punktu początkowego wypełnia pole `location` punktami, które zajmuje kabel o podanej długości `n`.

2.10 Board

Klasa obsługująca planszę. Umożliwia odczytanie planszy z pliku, zapis do pliku i wypełnienie planszy elementami.

Pola:

1. `private List<Elements> elements` - lista zawierająca wszystkie elementy z podanego pliku.
2. `private int [][] points` - dwuwymiarowa tablica zawierająca stany poszczególnych komórek wyrażone za pomocą liczb całkowitych.
3. `private Rectangle [][] rectangles` - dwuwymiarowa tablica kwadratów zawierająca graficzną reprezentację stanów komórek.

Metody:

1. `public int readBoardFromFile(String path)`.
    - Argumenty:
        - `path` - ścieżka do pliku z którego chcemy odczytać elementy.
    - Wartość zwracana:
        - Wartość liczbowa odpowiadająca sukcesowi lub porażce podczas wczytywania z pliku.
    - Działanie:
        - Metoda odczytuje z wybranego przez użytkownika z pliku po kolei elementy obwodu. W przypadku błędnego elementu metoda go pominie. Metoda podczas wczytywania uzupełnia listę elementów `elements`.
2. `public int fillWithElement`.
    - Argumenty:
        - `e` - element, który ma zostać wpisany do tablic `points` i `rectangles`.
    - Wartość zwracana:
        - Wartość liczbowa odpowiadająca sukcesowi lub porażce podczas wypełniania tablic.
    - Działanie:
        - Metoda wypełnia tablice `points` i `rectangles` punktami danego elementu.
3. `public int printBoardToFile(String path)`.
    - Argumenty:
        - `path` - ścieżka pliku, do którego ma być zapisana lista elementów.
    - Wartość zwracana:
        - Wartość liczbowa odpowiadająca sukcesowi lub porażce podczas zapisywania.
    - Działanie:
        - Metoda tworzy na podstawie tablicy `points` listę elementów i zapisuje ją do wybranego pliku.

3 Testy

Podczas implementacji projektu wykonamy testy jednostkowe dla modułów `Board` oraz `Simulator`. Klasy testujące nazwiemy BoardTest oraz SimulatorTest, nazwy metod testujących będą wskazywać jednoznacznie jaką funkcjonalność/przypadek poddamy sprawdzeniu. Do wykonania testów jednostkowych użyjemy bibliotek `JUnit`, `AssertJ` oraz `Mockito`. Pozostałe moduły przetestujemy uruchamiając program i obserwując jego zachowanie.

4 Ekrany programu

Ekran naszego programu został przedstawiony w specyfikacji funkcjonalnej w rozdziale 4.3 Ekrany działania programu.

5 Strona techniczna projektu

Wersja języka

W naszym projekcie będziemy korzystać z języka Java w wersji 8. Chcemy używać wersji 8, ponieważ do budowy GUI użyjemy technologii JavaFX, która została w wersji 8 ulepszona i poprawiona.

Używany system operacyjny

Projekt będzie tworzony w systemie Windows.

Dodatkowe narzędzia

Potrzebne biblioteki (`JUnit`, `AssertJ`, `Mockito`) dołączymy do programu używając narzędzia `Maven`.

Wersjonowanie

Podczas tworzenia projektu będziemy go wersjonować zaczynając od `0.1`, a gotowa wersja naszego programu będzie miała numer `1.0`. Podczas pracy z Gitem będziemy tworzyć nowy branch za każdym razem gdy będziemy chcieli wprowadzić nową funkcjonalność.