Specyfikacja funkcjonalna projektu *Wireworld* w języku Java

Chabik Jan (291060), Łuczak Mateusz (291088)

26 kwietnia 2018

Spis treści

1 Opis ogólny

1.1 Nazwa programu

Program został nazwany `Wireworld`. Nazwa nawiązuje do rozwiązywanego problemu, którym jest stworzenie emulatora *Wireworld* Briana Silvermana.

1.2 Poruszany problem

Poruszanym problemem będzie stworzenie emulatora *Wireworld* Briana Silvermana w języku Java.

Wireworld jest typem automatu komórkowego zaproponowanym przez Briana Silvermana w roku 1987. Jest to system często używany do symulacji elementów elektronicznych operujących na wartościach bitowo. Pomimo tego, iż sam sposób działania nie jest skomplikowany, jest możliwe zbudowanie funkcjonującego komputera.

Mapa dla danego automatu komórkowego reprezentowana jest przez cztery stany:

- Komórka pusta;
- Głowa elektronu;
- Ogon elektronu;
- Przewodnik.

Komórki są innego koloru, aby rozróżnić każdy ze stanów (przykładowo komórka pusta może być czarna, głowa elektronu niebieska, ogon czerwony, a przewodnik żółty). W jednostce czasowej, każdej komórce przypisywany jest stan na podstawie stanów jej sąsiadów. Rozwiązaniem będzie wizualizacja działającego automatu komórkowego oraz pliki wynikowe opisujące stan automatu.

1.3 Cel projektu

Celem projektu jest zapoznanie się przez studentów z problemem *Wireworld* Briana Silvermana oraz stworzenie implementacji w języku Java. Zadanie przewidziane zostało dla grup dwuosobowych. Ze względu na to, ważna jest umiejętność pracy w zespole.

2 Opis funkcjonalności

2.1 Korzystanie z programu

Program po uruchomieniu zostanie wywołany jako aplikacja okienkowa. Wszystkie opcje dostosowania symulacji będą dostępne w środowisku graficznym.

2.2 Uruchomienie programu

Uruchomienie programu następuje poprzez aktywację pliku wynikowego w środowisku IntelliJ Idea.

2.3 Możliwości programu

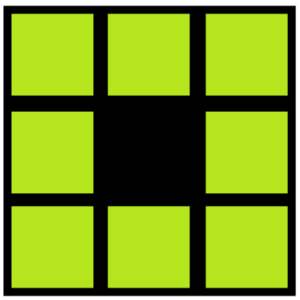
- Emulacja *Wireworld* Briana Silvermana;
- Odczyt danych z pliku tekstowego zawierającego informacje dotyczące mapy automatu komórkowego;
- Generowanie wybranej przez użytkownika ilości generacji automatu komórkowego;
- Symulacja automatu dla wybranego przez użytkownika sąsiedztwa (wybór spośród sąsiedztwa Moore'a i von Neumanna);
- Zapis danych wyjściowych do pliku tekstowego oraz pliku `.png`;
- Możliwość wybrania symulacji automatu w wybranych przez użytkownikach interwałach (np. $\frac{1}{4}s$) oraz pominięcia wybranej ilości generacji.

3 Format danych i struktura plików

3.1 Słownik pojęć

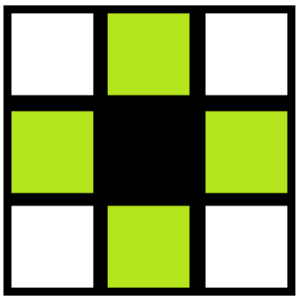
- **Plasza** - jest to prostokąt podzielony na pola (komórki), z których każda ma przypisaną wartość (0, 1, 2, 3) oraz odpowiedni kolor (czarny, niebieski, czerwony, żółty). Gdy komórka znajduje się stanie '0' mówimy, że jest pusta. W przypadku stanu '1', jest to głowa elektronu. Stanowi '2' przypisany jest ogon elektronu, a '3' przewodnik;
- **Sąsiedztwo Moore'a** - typ sąsiedztwa polegający na tym, iż rozpatrywane są wszystkie 8 komórek sąsiadujących z daną komórką (komórka na północ,

południe, wschód, zachód, północny-wschód, północny-zachód, południowy-zachód, południowy-wschód);



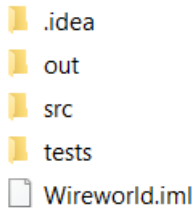
Rysunek 1: Sąsiedztwo Moore'a. Po środku znajduje się komórka (czarna), dla której rozpatrujemy sąsiadów. Komórki zielone opisują, które z nich są sąsiadami w opisywanym sąsiedztwie.

- **Sąsiedztwo Von Neumanna** - typ sąsiedztwa polegający na tym, iż rozpatrywane są 4 komórki sąsiadujące (komórka na północ, południe, wschód i zachód);



Rysunek 2: Sąsiedztwo von Neumanna. Po środku znajduje się komórka (czarna), dla której rozpatrujemy sąsiadów. Komórki zielone opisują, które z nich są sąsiadami w opisywanym sąsiedztwie.

3.2 Struktura katalogów



Rysunek 3: Struktura katalogów projektu

Struktura katalogów w projekcie będzie niemalże identyczna ze strukturą projektu stworzonego w IntelliJ Idea. Jedyną różnicą będzie dodatkowy katalog `tests`, w którym znajdować się będą klasy z testami jednostkowymi dla poszczególnych klas.

3.3 Przechowywanie danych w programie

- Informacje o opcjach zapisane będą w klasie statycznej;
- Każda macierz zapisana będzie w obiekcie Matrix;
- Obiekty Matrix trzymane będą w klasie głównej Main.

3.4 Dane wejściowe

Jako dane wejściowe służy plik w formacie `.txt`, w którym zapisana jest plansza.

Plik rozpoczyna się wymiarami planszy zapisanej za pomocą liczb naturalnych większych od 0. Następnie podawane są cyfry ‘0’, ‘1’, ‘2’, ‘3’ oznaczające stan komórki (pusta, głowa, ogon, przewodnik). Ilość cyfr powinna odpowiadać wielkości planszy. Przykładowo, jeżeli wymiary macierzy to `3x4`, to powinno być podanych 12 cyfr oznaczających stan komórek.

3.5 Dane wyjściowe

- Pliki `.png`, w których znajduje się graficzna interpretacja każdej kolejnej generacji;
- Pliki w formacie `.txt`, w których zapisane są kolejne generacje;
- Zwizualizowana generacja automatu w środowisku graficznym.

4 Scenariusz działania programu

4.1 Scenariusz ogólny

1. Uruchomienie programu;
2. Wybór pliku wejściowego przez użytkownika;
3. Wybór katalogu wyjściowego przez użytkownika;
4. Załadowanie pliku wejściowego;
5. Opcjonalnie dostosowanie przez użytkownika opcji tworzenia generacji;
6. Uruchomienie automatu;

4.2 Scenariusz szczegółowy

1. Uruchomienie programu:

   - Po uruchomieniu programu generuje się okno GUI;
   - W oknie GUI użytkownik zaznacza następujące opcje:
     - Wybór pliku wejściowego;
     - Wybór katalogu wyjściowego;
     - Sąsiedztwo;
     - Ilość generacji;
     - Zapis do pliku `.txt` i/lub `.png`

   Opcje wyboru pliku wejściowego i katalogu wyjściowego są obowiązkowe. Reszta jest wybrana automatycznie, jednakże może być modyfikowana.

2. Wybór pliku wejściowego przez użytkownika:

   - Użytkownik za pomocą przycisku pozwalającego na przeszukiwanie katalogów zaznacza plik typu `.txt`. Program zapisuje ścieżkę do pliku w postaci zmiennej typu `String`. W razie braku dostępu do pliku lub sytuacji gdy plik nie jest w formacie `.txt` wyświetlany jest odpowiedni komunikat.

3. Wybór katalogu wyjściowego przez użytkownika:

- Użytkownik za pomocą przycisku pozwalającego na przeszukiwanie katalogów zaznacza katalog, do którego zapisywane będą pliki `.txt` i `.png`. Program zapisuje ścieżkę do katalogu w postaci `String`. W razie niepoprawnego katalogu wyświetlany jest odpowiedni komunikat

4. Załadowanie pliku wejściowego:

   - Po przyciśnięciu przycisku `Load File`, program generuje macierz na podstawie pliku wejściowego. Wygenerowaną macierz pokazuje w interfejsie użytkownika. Sprawdzana jest poprawność pliku wejściowego.

5. Opcjonalne dostosowane opcji programu przez użytkownika:

   - Za pomocą chechboxów `TXT` i `PNG` zaznacza się preferencje co do generowania plików (automatycznie obie opcje są odznaczone);
   - Za pomocą dropbara `TimeInterval` wybiera się interwał czasowy `t`, w jakim na ekranie mają się pojawiać kolejne generacje (automatycznie wybrana jest 1 sekunda);
   - Za pomocą radiobuttonów wybiera się sąsiedztwo które ma stosować program (automatycznie wybrane jest sąsiedztwo `Moore'a`);
   - W okno tekstowe wpisujemy ilość generacji, które program ma wygenerować (automatycznie wpisane jest 5);

6. Uruchamianie automatu

   (a) Za pomocą przycisku `One Move`:
   - Blokowany jest dostęp do wszystkich przycisków;
   - Wykonywane jest pojedyncze przejście *(Opis znajduje się w rozdziale 4.3)*;
   - Odblokowywany jest dostęp do wszystkich przycisków.

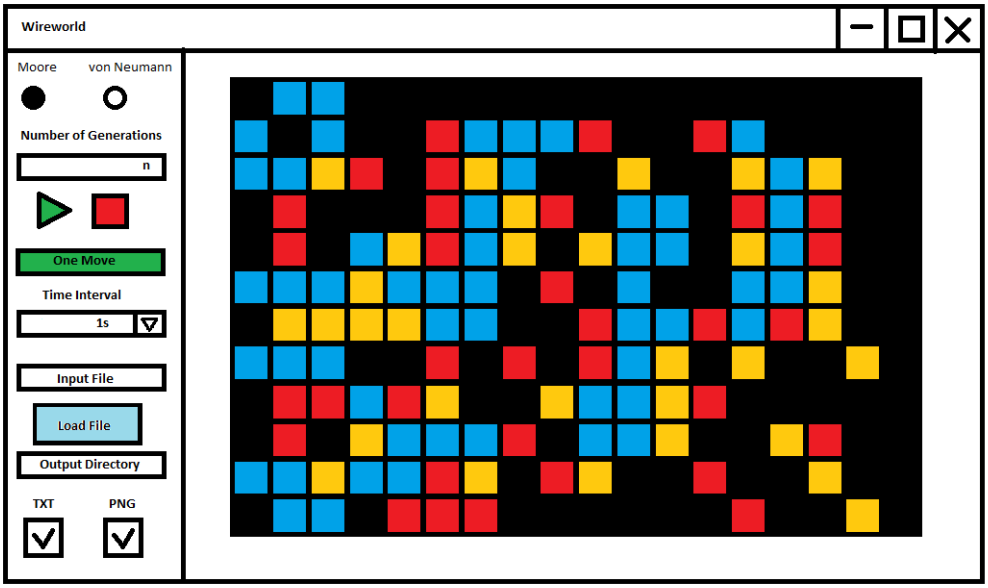
   (b) Za pomocą przycisku oznaczonego zielonym trójkątem (`START`):
   - Blokowany jest dostęp do wszystkich przycisków poza przyciskiem `STOP`;
   - Włącza się pętla wykonywana n-razy (n to liczba wpisana w pole jako `NumberOfGenerations`);
   - Każde pojedyncze przejście wykonuje się co `t` sekund;
   - Po każdym przejściu sprawdzane jest czy podczas działania programu został naciśnięty przycisk `STOP`. Jeśli tak, pętla zostaje przerwana i odblokowywany jest dostęp do pozostałych przycisków.

4.3 Opis pojedynczego przejścia

1. Sprawdzane jest czy w polu `Number Of Generations` zapisana jest poprawna wartość (dodatni `int` mniejszy od 10 000). W razie błędu na ekranie pojawia się odpowiedni komunikat;
2. Na podstawie opcji, utworzonej macierzy i wartości w polu `NumberOf Generations` program generuje kolejną macierz;
3. Program wyświetla kolejną macierz na pulpicie;
4. Na podstawie opcji program generuje pliki `.txt` i/lub `.png`;
5. Program odejmuje 1 od wartości wpisanej w pole tekstowe oznaczone jako `NumberOfGenerations`.

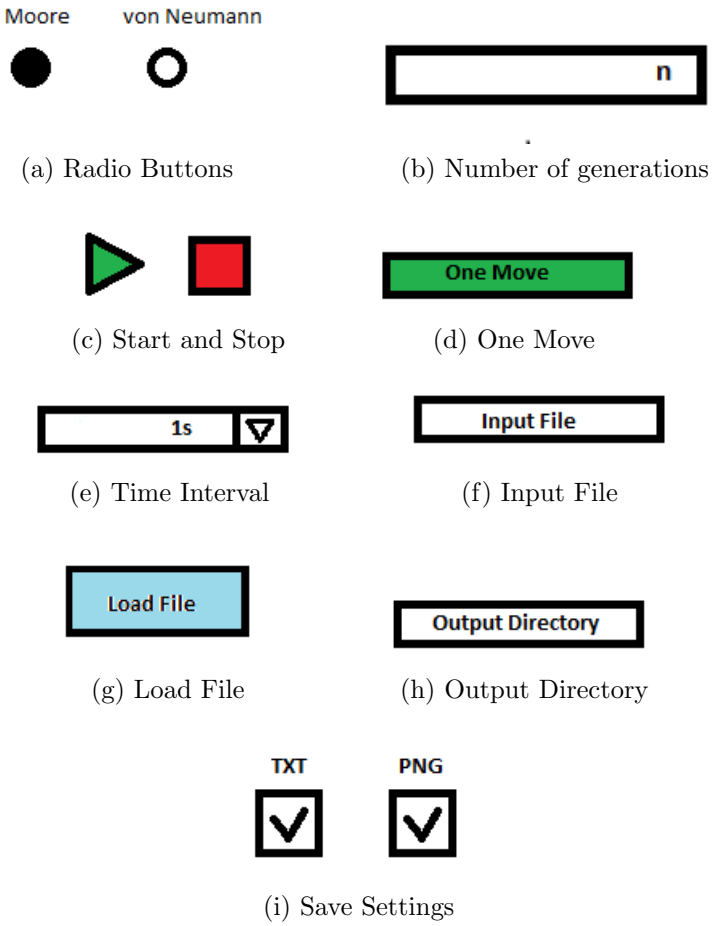
4.4 Opis środowiska graficznego programu

4.4.1 Projekt środowiska graficznego



Rysunek 4: Projekt środowiska graficznego

4.4.2 Opis guzików



(a) Radio Buttons

(b) Number of generations

(c) Start and Stop

(d) One Move

(e) Time Interval

(f) Input File

(g) Load File

(h) Output Directory

(i) Save Settings

Rysunek 5: Zestawienie wszystkich guzików

(a) Radio Buttons – grupa przycisków pozwalająca wybierać sąsiedztwo;

(b) Number of Generations – pole tekstowe, do którego należy wpisać ilość przejść automatu komórkowego (wartość musi być dodatnią liczbą całkowitą mniejszą od 10 000);

(c) Start and Stop – przyciski `START` i `STOP` pozwalające uruchomić i zatrzymać automat;

(d) One Move - przycisk, który sprawia, że automat wykonuje pojedyncze przejście;

(e) TimeInterval – przycisk typu Dropbar odpowiadający za interwał czasowy pomiędzy przejściami automatu komórkowego;

(f) Input File – przycisk typu FileChooser pozwalający wybrać plik wejściowy;

(g) Load File - przycisk z pomocą którego tworzona i wyświetlana w interfejsie jest macierz z pliku wejściowego;

(h) Output Directory – przycisk typu FileChooser, za pomocą którego wybieramy katalog, do którego generowane są pliki;

(i) Save Settings – 2 przyciski typu CheckBox pozwalające wybierać preferencje dotyczące zapisu generacji.

5 Testowanie

Testować działanie programu będziemy za pomocą narzędzi dostępnych w Frameworku JUnit 4. Dla każdej klasy stworzona zostanie klasa testująca, w której testować będziemy każdą z metod. Skorzystamy z metodyki testów regresywnych, czyli po dodaniu nowej funkcjonalności testować będziemy również poprzednio sprawdzone klasy w celu upewnienia się, czy program działa poprawnie.