Politechnika Warszawska
Wydział Elektryczny

Specyfikacja funkcjonalna symulatora automatu komórkowego *Wireworld*

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

25 kwietnia 2018

1 Opis ogólny

1.1 Nazwa programu

Nasz program będzie nazywał się `WireWorld`.

1.2 Poruszany problem

Zadaniem naszego programu będzie symulacja automatu komórkowego WireWorld wymyślonego przez Brian Silvermanna w 1987 roku. Kolejne symulacje polegają na zmianie stanów komórek ułożonych na prostokątnej planszy według okrelonych zasad. Każda komórka może znajdować się w jednym z czterech stanów: ogon elektronu (kolor żółty), głowa elektronu (kolor czerwony), przewodnik (kolor czarny), pusta komórka (kolor biały). Zasady symulacji:

- 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.

Dodatkowo, w naszym programie zakładamy, że plansza jest ograniczona i komórki poza nią są puste. Naszym zdaniem uprości to symulację i zwiększy jej czytelność.

1.3 Cel projektu

Celem tego projektu jest nauka i praktyka programowania w języku JAVA przy jednoczesnym zapoznaniu się z tematem automatu komórkowego WireWorld Briana Silvermanna.

2 Opis funkcjonalności

2.1 Uruchomienie i korzystanie z programu

Program uruchamiany jest poprzez dwukrotne kliknięcie na ikonę. Korzystanie z niego odbywa się za pomocą graficznego interfejsu użytkownika.

2.2 Możliwości programu:

- Wczytywanie plików wejściowych z danymi planszy i liczbą generacji
- Wizualizacja generacji na żywo.
- Możliwość wstrzymania i zakończenia generacji.
- Zapis obecnie wyświetlanej generacji do pliku możliwego do ponownego wczytania przez program.

3 Format danych i struktura plików

3.1 Pojęcia

Komórka – podstawowa jednostka automatu komórkowego.

Generacja – jeden cykl, w którym komórki zmieniają swój stan.

Stan komórki – komórka może znajdować się w czterech stanach: ogon elektronu, głowa elektronu, przewodnik lub pusta komórka.

Sąsiedzi komórki – komórki przylegające do niej bokami i rogami.

Bramka logiczna - element konstrukcyjny realizujący prostą funkcję logiczną.

Bramka logiczna NOR - bramka logiczna realizująca funkcję alternatywy wykluczającej.

Bramka logiczna AND - bramka logiczna realizująca funkcję koniunkcji.

Bramka logiczna OR - bramka logiczna realizująca funkcję alternatywy.

3.2 Struktura katalogów

Projekt będzie znajdował się w katalogu `WireWorldr`. Pliki z kodem źródłowym przechowywane będą w katalogach odpowiadającym ich zastosowaniu (np. pliki odpowiedzialne za graficzny interfejs użytkownika będą w katalogu `gui`). Katalogi te będą z kolei przechowywane w folderze `src`, w którym znajdzie się również katalog `data` zawierający przykładowe pliki wejściowe oraz katalog `test` zawierający pliki testowe dla poszczególnyc modułów programu. Wyniki działania naszego programu(plik wyjściowy) oraz pliki wynikowe z rozszerzeniem .class umieścimy w katalogu `bin`. W katalogi `External Libraries` znajdą się biblioteki, z których korzysta nasz program. Graficzne przedstawienie struktury katalogów w naszym projekcie:

```
WireWorld
+--- src
|    +--- gui
|    +--- board
|    +--- inne pakiety z plikami źródłowymi
|    +--- test
|    +--- data
+--- bin
+--- External Libraries
```

3.3 Przechowywanie danych w programie

Plansza przeznaczona do przechowywania stanów wszystkich komórek będzie reprezentowana przez klasę zawierającą dwuwymiarową tablicę liczb całkowitych, zmienne zapamiętującą jej wymiary (wysokość i szerokość) oraz metody ją obsługujące.

3.4 Dane wejściowe

Plik wejściowy zawiera w każdym wierszu definicję obiektu lub stanu pojedynczej komórki. Definicja powinna zostać podana w postaci:
`Nazwa_obiektu_lub_stanu_komórki:` X,Y
Gdzie:

`nazwa_obiektu_lub_stanu_komorki` – nazwa tworzonego obiektu lub nazwa stanu tworzonej komórki,

`ile_generacji` – współrzędne oznaczające położenie początka obiektu lub położenie komórki.

Dostępne nazwy obiektów/stanów komórek:

- Diode - dioda,
- OR - bramka logiczna OR,
- NOR - bramka logiczna NOR,
- AND - bramka logiczna AND,
- Wire - drut
- ElectronHead - głowa elektronu,
- ElectronTail - ogon elektronu,
- Blank - pusta komórka,
- Conductor - przewodnik.

W przypadku definiowania drutu można (opcjonalnie) podać jeszcze dodatkowe dwa parametry:

`N` - długość drutu (ilość pól planszy, które będzie zajmował), domśylnie ustawiany na: 2,

`orientation` - orientacja drutu określająca czy kolejne pola przez niego zajmowane będą w linii poziomej (`horizontal`) czy pionowej (`vertical`). Domyślnie parametr ustawiony jest jako: `horizontal`.

Przykład definicji drutu:

```
Wire: 0,3 3 vertical
```

W przypadku kilkukrotnego wpisania obiektu/stanu komórki na te same współrzędne, program weźmie pod uwagę ostatnią definicję.

3.5 Dane wyjściowe

Plik wyjściowy programu (opisujący stan wybranej generacji przy zastopowanej symulacji) sporządzony zostanie według tej samej struktury co plik wejściowy, dzięki czemu będzie mógł być potem użyty jako plik wejściowy.

4 Scenariusz działania programu

4.1 Scenariusz ogólny

1. Uruchomienie programu poprzez dwukrotne kliknięcie ikony.
2. Wyświetlenie interfejsu aplikacji wraz z pustą planszą.
3. Otwarcie wybranego przez użytkownika pliku wejściowego.
4. Generacja zadanej liczby generacji.
5. Zapisanie obecnie wyświetlanej generacji, jeżeli użytkownik kliknie przycisk `Zapisz obecną generację`.
6. Zakończenie generacji planszy. Użytkownik może wybrać następny plik do generacji lub zakończyć działanie programu.

4.2 Scenariusz szczegółowy

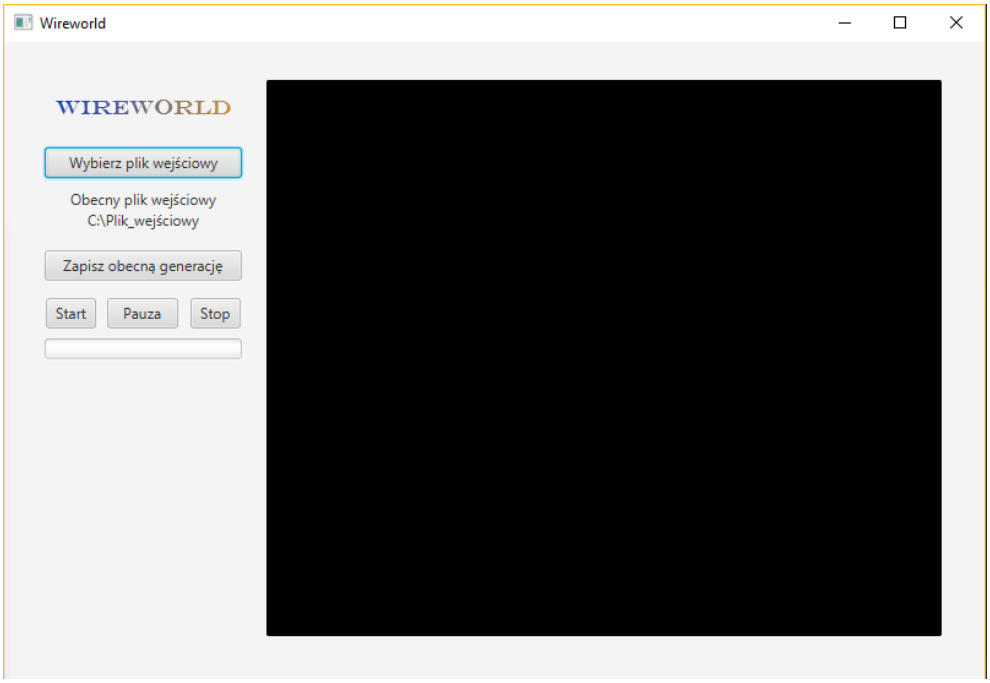
1. Uruchomienie programu poprzez dwukrotne kliknięcie ikony.
2. Wyświetlenie interfejsu aplikacji wraz z pustą planszą.
3. Użytkownik wybiera plik wejściowy poprzez kliknięcie na przycisk `Wybierz plik`, a następnie wskazanie na odpowiedni plik w drzewie katalogów. Następuje próba wczytania danych planszy i liczba generacji.

- Jeśli wybrany przez użytkownika plik nie jest możliwy do wczytania to zostanie wyświetlony komunikat o błędzie.
- Jeśli plik wejściowy zawiera błędne dane to zostanie wyświetlony komunikat o błędzie
- Jeśli liczba generacji nie została podana to program będzie generował planszę w nieskończoność, z wyjątkiem przypadku gdy użytkownik naciśnie przycisk `Stop`.

4. Wyświetlenie wczytanej z pliku planszy.
   - Po naciśnięciu przez użytkownika przycisku `Start` program rozpocznie wyświetlanie nowych generacji w odstępie 2 sekund. W czasie generacji zapełniany będzie pasek postępu znajdujący się po prawej stronie ekranu.
   - Jeśli użytkownik naciśnie przycisk `Pauza`, to program wstrzyma wyświetlanie nowych generacji i umożliwi zapis bieżącej generacji do pliku za pomocą naciśnięcia przycisku `Zapisz obecną generację`, a następnie wpisania nazwy pliku do zapisu. Wznowienie generowania odbywa się za pomocą przycisku `Start`
     - Jeśli nie będzie można zapisać generacji do pliku o nazwie podanej przez użytkownika to program wyświetli komunikat o błędzie i poprosi o ponowne wybranie pliku.
   - Jeśli użytkownik naciśnie przycisk `Stop` to program zakończy generację i wyświetli początkową planszę. Możliwe będzie wtedy rozpoczęcie generacji od nowa lub wczytanie innego pliku wejściowego.
5. Zakończenie generacji na skutek wypełnienia zadanej liczby.
6. Użytkownik może wczytać nowy plik wejściowy, rozpocząć generację od nowa lub zakończyć działanie programu.

4.3 Ekrany działania programu

Ekran aplikacji został zamieszczony poniżej(Rys 1). W lewym górnym rogu znajduję się tytuł programu. Poniżej znajduje się przycisk `Wybierz plik wejściowy` odpowiedzialny za wyświetlenie menu do wybrania pliku wejściowego. Poniżej wypisany jest aktualnie używany plik wejściowy. Poniżej znajduje się przycisk `Zapisz obecną generację` odpowiedzialny za wyświetlenie menu do wybrania pliku do zapisu. Poniżej znajdują się przyciski odpowiedzialne za sterowanie generowaniem. Poniżej znajduje się pasek progresu.



Rysunek 1: Ekran aplikacji

5 Testowanie

Działanie poszczególnych części programu przetestujemy przy użyciu biblioteki `AssertJ` oraz przykładowych plików wejściowych. Tylko część graficzną (GUI) przetestujemy ręcznie np. sprawdzając czy program wykona odpowiednią akcję po nacinięciu przycisku.