Specyfikacja funkcjonalna

1. Opis ogólny

   1.1. Nazwa programu
   Program nazywa się *Wireworld*.

   1.2. Poruszany problem
   Implementować będziemy WireWorld Briana Silvermana.
   Komórka może znajdować się w jednym z czterech stanów:
   Pusta,
   Głowa elektronu,
   Ogon elektrony,
   Przewodnik.
   Zwykle przymuje się następujące kolory stanów: biały, czerwony, żółty, czarny.
   Kolejne generacje budowane są z wykorzystaniem zestawu pięciu zasad:
   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.
   W WireWorld stosuje się sąsiedztwo Moore'a.

   1.3. Użytkownik docelowy
   Użytkownikiem docelowym naszego programu jest prowadzący zajęć laboratoryjnych.

2. Ogólna funkcjonalność

   2.1. Korzystanie z programu
   Program wykonany jest w formie aplikacji okienkowej. Znajduje się on w katalogu *automatkomorkowy* pod nazwą *Wireworld*. Występuje w formie pliku wykonywalnego o rozszerzeniu *jar*. W tym samym folderze znajduje się plik *dane.txt*. Zawiera on dane na temat stanów wybranych komórek.

2.2. Uruchamianie programu
Uruchomienie programu następuje przez klikniecie na plik „Wireworld". Po tym program spyta ile generacji planszy użytkownik chce wykonać, a następnie wyświetli je po kolei.

2.3. Możliwosci programu
Program poza pokazaniem poszczególnych generacji planszy zapisuje również te generacje do folderu „generacje". Przy każdym użyciu programu folder ten jest czyszczony i nowe generacje są zapisywane.

3. Format danych i struktura plików

3.1. Słownik
.png - nazwa pliku w postaci **gen-x.png**, gdzie x jest numerem generacji,

dane - umieszczone w jednym wierszu w pliku informacje o polach tabeli tworzącej nasz projekt, np. Diode: 0, 3, Normal w przypadku diod albo ElectronHead: 1, 3 w przypadku głów elektronu albo ElectronTali: 1, 9 w przypadku ogonów elektronu,

GUI - graphical user interface, czyli to, co aplikacja pokazuje użytkownikowi,

Macierz - tablica dwuwymiarowa o wymiarach X na Y, gdzie X to liczba wierszy, a Y to liczba kolumn,

3.2. Struktura katalogów
Plik **.jar** będzie przechowywany w katalogu **automat komórkowy** pod nazwą **Wireworld**. Dane znajdują się w tym samym katalogu. Dane wyjściowe przechowywane są w podkatalogu **generacje**.

3.3. Przechowywanie danych w programie
Nasz program znajduje się w repozytorium pod adresem *https* : *//github.com/Achreko/WireWorld − Java*. Jeżeli zaś chodzi o struktury danych w programie, to planszę, na której będziemy umieszczać poszczególne elementy zapiszemy w postaci *macierzy*.

Będziemy też mieli drugą *macierz*, w której znajdować się będzie przyszła generacja i to właśnie tam będziemy dokonywać zmian.

3.4. Dane wejściowe
Dane wejściowe znajdują się w pliku *dane.txt*. Zawiera on *dane*. Jeśli użytkownik chce zmienić *dane* lub wprowadzić nowe to musi edytować ten plik.

3.5. Dane wyjściowe
Dane wyjściowe będą w postaci plików **.png*, których ilość zależeć będzie od żądanej przez użytkownika ilości generacji.

4. Scenariusz działania programu

4.1. Scenariusz ogólny

4.1.1. Program wczytuje *dane* z pliku
4.1.2. Program pyta użytkownika o liczbę generacji.
4.1.3. Program analizuje stany komórek i transformuje je.
4.1.4. Program pokazuje wynik pojedynczej generacji.
4.1.5. Program zapisuje pojedynczą generację do pliku.
4.1.6. Program ponownie wykonuje operacje z punktów 3-5, aż nie wykona ilości generacji równej tej podanej w punkcie 2.
4.1.7. Program pyta czy użytkownik chce jeszcze raz wykonać proces (powrót do punktu 1) dla tych samych danych, czy zakończyć.

4.2. Scenariusz szczegółowy

4.2.1. Program wczytuje *dane* z pliku.
- Program sprawdza czy sposób podania *danych* jest poprawny, jeśli nie pokaże komunikat „błąd danych" i zakończy prace.

4.2.2. Program pyta użytkownika o liczbę generacji.
- Sprawdzane jest czy użytkownik podał wartość numeryczną większą od 0. Jeśli tak to przechodzi dalej, jeśli nie to wyświetla informację o podaniu złej wartości i pyta ponownie o liczbę generacji.

4.2.3. Program usuwa wszystkie pliki z katalogu „generacje".

- Jeśli katalog jest pusty to tylko idzie dalej, nic nie usuwa.

4.2.4. Program analizuje stany komórek i transformuje je.
- Operacja analizy przeprowadzana jest przy użyciu 2 tablic dwuwymiarowych. Jedna z nich służy do przechowywania nowych stanów, a druga do badania sąsiedztwa i ustalania tych nowych stanów.

4.2.5. Program pokazuje wynik pojedynczej generacji.

4.2.6. Program zapisuje pojedynczą generację do pliku w przypadku błędu tworzenia pliku pojawi się komunikat i program zostanie zamknięty.

4.2.7. Program ponownie wykonuje operacje z punktów 3-5, aż nie wykona ilości generacji równej tej podanej w punkcie 2.

4.2.8. Program pyta czy użytkownik chce jeszcze raz wykonać proces (powrót do punktu 1) dla tych samych danych czy zakończyć.

5. Testowanie

I Ogólny przebieg testowania
Do testów kodu użyjemy JUnit, a *GUI* będziemy testować na bieżąco tworząc aplikację.