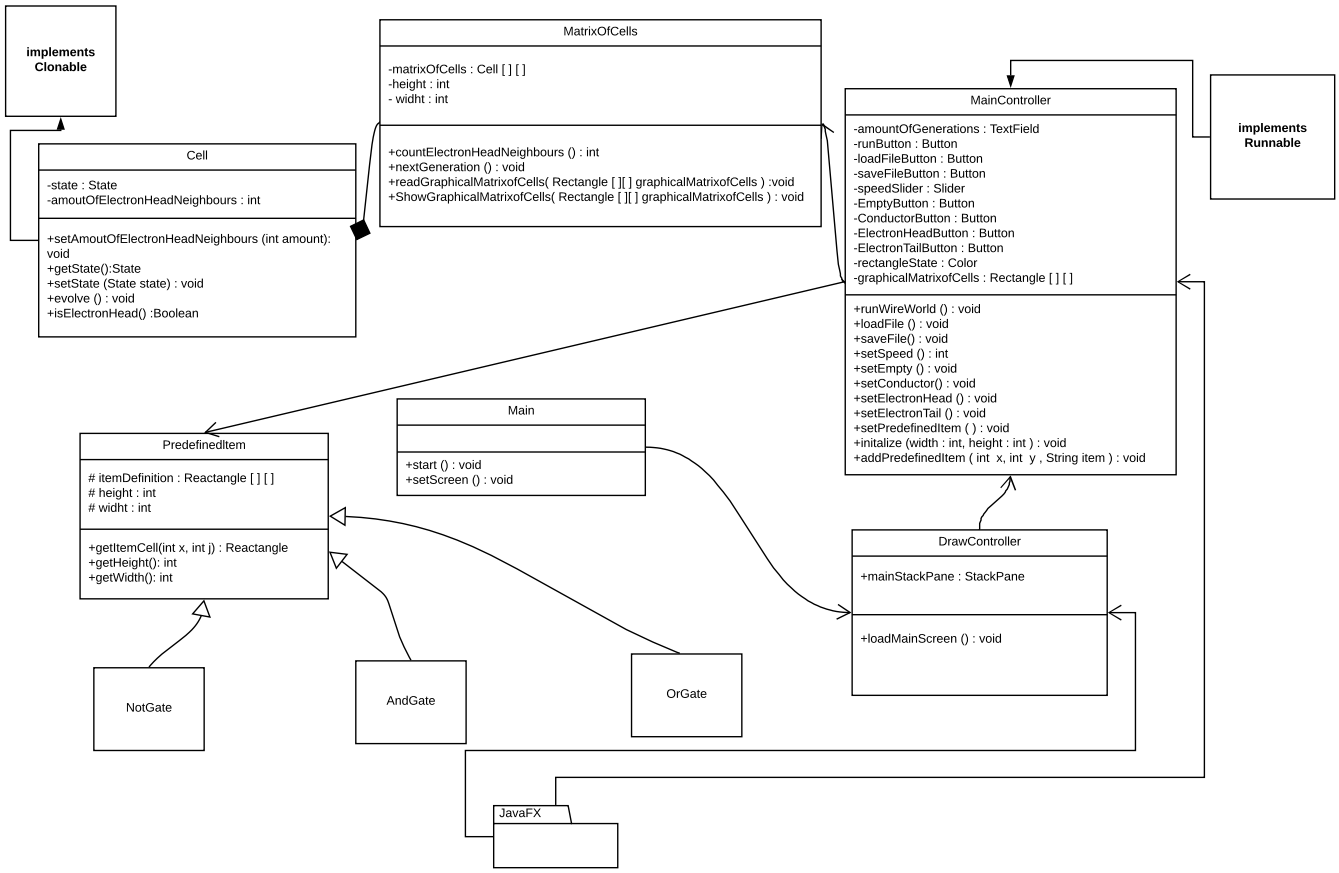
Politechnika Warszawska
Wydział Elektryczny
Kierunek Informatyka

Specyfikacja implementacyjna

Wykonali: Piotr Jeleniewicz, Aliaksandr Karolik
Warszawa, 19.03.2018

Spis treści

1 Diagram Klas



RYS.01 DIAGRAM KLAS

2 Opis metod i pól klas

2.1 Wstęp.

Większość klas zawiera metody typu set i get. Większość tych metod służy do ustawiania oraz pobierania wartości poszczególnych pól obiektów. Jeśli metoda wykonuje inną funkcję zostanie to opisane.

2.2 Klasa Cell.

Klasa reprezentująca komórkę, z których tworzona jest plansza do gry w WireWorld. Klasa Cell będzie zawierać w sobie typ danych typu wyliczeniowego State. Definicja tego typu ma następującą postać:

```
public static enum State {
EMPTY,
ELECTRON_HEAD,
ELECTRON_TAIL,
CONDUCTOR
}
```

Składa się z pól:

```
private State state;
```

Pole typu wyliczeniowego określające stan.

```
private int amountOfElectronHeadNeighbours;
```

Pole przedstawiające liczbę komórek sąsiadujących z daną komórką, będących w stanie ElectronHead

Będzie zawierać metody:

```
public void  evolve ();
```

Metoda przeprowadzająca zmianę stanu komórki w zależności od obecnego statu oraz obecnej ilości sąsiadów będących w stanie ElectronHead.

```
public Boolean isElectronHead ();
```

Metoda będzie sprawdzać czy komórka jest obecnie w stanie ElectronHead.

2.3 Klasa MatrixOfCells.

Klasa przedstawiająca planszę dla automatu komórkowego WireWorld. Klasa jest wykorzystywana do obsługi logiki systemu. Klasa MatrixOfCells zawiera następujące pola:

```
private Cell [] [] matrixOfCells;
```

Pole przedstawiające planszę automatu komórkowego WireWorld.

```
private int height;
private int widht;
```

Pola określające wymiary planszy.
Klasa będzie zawierać metody:

```
public int countElectronHeadNeighbours();
```

Metoda ta służy do wyznaczenia ilości sąsiadów typu ElectronHead dla wybranej komórki.

```
public void nextGeneration();
```

Metoda będzie przeprowadzać generowanie następnej generacji planszy zgodnie z regułami automatu komórkowego WireWorld.

```
public void readReactangles(Rectangles[][] rectangles);
```

Metoda przekształcająca graficzną reprezentację planszy na logiczną, reprezentowaną jako tablica obiektów typu Cell. Jako argument metoda readReactangles przyjmuje dwuwymiarową tablicę obiektów typu Rectangle.

```
public void showReactangles(Rectangles[][] rectangles);
```

Metoda przekształcająca logiczną reprezentację planszy na graficzną, reprezentowaną jako tablica obiektów typu Rectangle, bazując na tablicy matrixOfCells. Jako argument metoda readReactangles przyjmuje dwuwymiarową tablicę obiektów typu Rectangle.

2.4 Klasa MainController

Jest to klasa będąca kontrolerem głównego interfejsu systemu.

```
private TextField amountOfGenerations;
private Button runButton;
private Button loadFileButton;
private Button saveFileButton;
private Slider speedSlider;
private Button EmptyButton;
private Button ConductorButton;
private Button ElectronHeadButton;
private Button ElectronTailButton;
private Color  rectangleState;
```

Klasa MainController będzie zawierać w sobie pola odpowiadające przyciskom oraz innym elementom interfejsu graficznego.

```
private Rectangle[][] graphicalMatrixofCells;
```

Pole będące graficzną reprezentację planszy.
Metody dostępne w tej klasie:

```
public void WireWorld();
```

Metoda uruchamiająca symulację.

```
public void loadFile();
public void saveFile();
```

Metody służące do wczytywania oraz zapisywania plików PNG zawierających generację.

```
public setSpeed();
```

Metoda służy do ustalania prędkości przeprowadzania symulacji.

```
public void setEmpty();
public void setConductor();
public void setElectronHead();
public void setElectronTail();
```

Metody służące do wybierania typu komórek umieszczanych przez użytkownika na planszy.

```
public void addPredefinedItem(int x, int y, String item);
```

Metoda dodająca do planszy predefiniowany obiekt, którego nazwa jest zawarta w polu item. Lewy, górny róg obiektu zostanie umieszczony w punkcie o współrzędnych zawartych w zmiennych x oraz y, o ile na planszy będzię wystarczająco miejsca.

```
public void initalize(int width, int height);
```

Metoda inicjalizująca oraz ustalająca wymiary okna programu oraz planszę symulacji. Jako argumenty metoda przyjmuje wysokość oraz szerokość planszy.

2.5 Klasa DrawController

Ta klasa będzie kontrolerem sceny początkowej, a także będzie wczytywać główny interfejs systemu.

Będzie zawierać następujące pole:

```
public StackPane mainStackPane;
```

Pole służące do przechowywania obiektu typu StackPane zawierającego główny intefejs systemu.

W tej klasie będą zdefiniwane następujące metody:

```
public void loadMainScreen;
```

Metoda wczytująca plik FXML zawierający informację o głównym interfejsie systemu.

2.6 Klasa Main

Jest to główna klasa systemu, w której uruchamiany jest interfejs.

Klasa ta oprócz metody main zawiera 2 metody:

```
public void start;
```

Metoda inicjująca program.

```
public void setScreen;
```

Metoda służąca do ustawiania obecnie wyświetlanej sceny.

2.7 Klasa PredefinedItem

Jest to klasa nadrzędna, po której dziedziczą inne klasy będące reprezentacją gotowych elementów automatu komórkowego WireWorld takich jak bramki logiczne itp.

Klasa ta zawiera pola:

```
protected Reactangle[][] itemDefinition;
```

Pole będące tablicą dwuwymiarową która będzie reprezentacją gotowych elementów automatu komórkowego.

```
protected int height;
protected int width;
```

Pola definiujące rozmiary predefiniowanego elementu.

Występują 3 klasy dziedziczące po klasie PredefinedItem:

1. NotGate - reprezentacja negacji
2. AndGate - reprezentacja bramki logicznej AND
3. OrGate - reprezentacja bramki logicznej OR

Klasy te nie posiadają dodatkowych pól i metod. Posiadają jedynie konstruktory, które definiują reprezentowaną struktury w polu itemDefinition.

3 Przepływ sterowania.

Po uruchomieniu się programu metoda *start* z klasy main zainicjuje kontroler *DrawController*, który wywoła główny interfejs użytkownika umieszczając go na scenie. W interfejsie użytkownika nastąpi zdefiniowanie pierwszej generacji symulacji. Może się to stać na dwa sposoby:

1. Wybranie pliku PNG zawierającego definicję pierwszej generacji.
2. Poprez narysowanie pierwszej generacji na planszy wyświetlanej na interfejsie przez użytkownika.

W przypadku wczytania pierwszej generacji z pliku PNG, użytkownik za pomocą przycisku uruchomi eksplorator plików, w którym wskaże plik źródłowy. Następnie metoda *loadFile* przetworzy wybrany plik. Następnie na podstawie wymiarów pliku graficznego uruchomi metodę *initialize()*, która przygotuje planszę o odpowiednich wymiarach i wyświetli ją w interfejsie graficznym.

W drugim przypadku gdy użytkownik wpisuje najpierw wymiary planszy oraz zatwierdza je przyciskiem co powoduje wywołanie metody *initialize()*, która przygotuje planszę o odpowiednich wymiarach i wyświetli ją w interfejsie graficznym. Następnie użytkownik rysuje pierwszą generację na planszy, używając przycisków za pomocą których dostosowuje typ komórki, którą umieszcza na planszy. Przyciski te będą wywoływały metody:

1. *setEmpty()*;
2. *setElectronHead()*;
3. *setElectronTail()*;
4. *setConductor()*;

Po wybraniu przycisku uruchamiającego jedna z tych metod, użytkownik będzię umieszczał na planszy komórki wybranego typu. Dodatkowo będzie miał możliwość umieszczania na planszy gotowych elementów takich jak bramki logiczne. Po wybraniu z listy żądanego elemntu oraz określeniu jego położenia na planszy zostanie wywołana metoda *addPredefinedItem()*, która korzystając z definicji zdefiniowanego wcześniej elementu w klasach dziedziczących po *PredefinedItem* umieści go na planszy, sprawdzając wcześniej czy jest na niej wystarczająco miejsca aby taki element tam umieścić.

Następnie po wprowadzeniu ilości generacji do pola określającego liczbę generacji o nazwie *amountOfGenerations* i naciśnięciu przycisku uruchamiającego symulucję działania automatu komórkowe-

go WireWorld, zostanie wywołana metoda *runWireWorld()*. Przez cały czas działania symulacji użytkownik za pomocą suwaka speedSlider będzie mógł zmienić prędkość wyświetlania symulacji. Prędkość ta będzię ustawiana za pomocą metody *setSpeed()*.

W metodzie runWireWorld() zostanie wywołana metoda *readGraphicalMatrixOfCells()*, która przekształci graficzną reprezentację planszy zawartą w tablicy *grapicalMatrixOfCells* z klasy *MainController* na logiczną zawartą w tablicy *matrixOfCells* z klasy *MatrixOfCells*. Następnie dla logicznej tablicy *matrixOfCells* zostanie wyowłana maetoda *nextGeneration()*, która dla każdej komórki w tablicy wywoła metodę *evolve()* z klasy *Cell*. Metoda *evolve* zmieni stan komórki, korzystojąc zasad automatu komórkowego WireWorld. W przypadku, gdy komórka będzie w stanie *Conductor* zostanie dla niej dodatkowo wywołana metoda *countElectronHeadNeighbours()*, który wyliczy sąsiadów w stanie ElectronHead badanej komórki. Następnie za pomocą metody *ShowGraphicalMatrixOfCells* zostanie zaktualizowana graficzna plansza wyświetlana w interfejsie. Proces ten powtarza się w zależności od ilości generacji.

Gdy użytkownik naciśnie przycisk SaveToFile zostanie wywołana metoda *saveFile()*, która zapisze bieżącą generację do pliku PNG.

4 Testy jednostkowe.

4.1 Test klasy Cell

Test klasy Cell polega sprawdzeniu poprawności działania metod:

1. *evolve()*;
2. *isElectronHead()*;

Do sprawdzenia metody *evolve()* będą stworzone cztery obiekty typu *Cell*. Będą one reprezentować wszystkie możliwe typy komórek. Dla każdego obiektu będzie uruchomiona metoda *evolve()*. Test zostanie zaliczony w przypadku gdy:

1. Obiekt *Cell* przed użyciem metody *evolve()* miał stan *Empty* po użyciu metody obiekt będzie miał stan *Empty*.
2. Obiekt *Cell* przed użyciem metody *evolve()* miał stan *ElectronHead* po użyciu metody obiekt będzie miał stan *ElectronTail*.
3. Obiekt *Cell* przed użyciem metody *evolve()* miał stan *ElectronTail* po użyciu metody obiekt będzie miał stan *Conductor*.
4. Obiekt *Cell* przed użyciem metody *evolve()* miał stan *Conductor* oraz ilość sąsiadów w stanie *ElectronHead* będzie wynosić jeden albo dwa. Wtedy po użyciu metody obiekt będzie miał stan *ElectronHead*.

Do sprawdzenia metody *isElectronHead()* będą stworzone cztery obiekty typu *Cell*. Będą one reprezentować wszystkie możliwe typy komórek. Dla każdego obiektu zostanie uruchomiona metoda *isElectronHead()*. Test zostanie zaliczony w przypadku gdy:

1. Obiekt jest w stanie *Empty*. Metoda *isElectronHead()* zwróci *False*.
2. Obiekt jest w stanie *ElectronHead*. Metoda *isElectronHead()* zwróci *True*.
3. Obiekt jest w stanie *ElectronTail*. Metoda *isElectronHead()* zwróci *False*.
4. Obiekt jest w stanie *Conductor*. Metoda *isElectronHead()* zwróci *False*.

4.2 Test klasy MatrixOfCells

Do testu klasy *MatrixOfCells* zostaną przyjętę założenia:

1. C - symbolizuje komórkę typu *Conductor*,
2. E - symbolizuje komórkę typu *Empty*,
3. H - symbolizuje komórkę typu *ElectronHead*,
4. T - symbolizuje komórkę typu *ElectronTail*.

Przy wyżej wymienonych założeniach będą przeprowadzone następujące testy:

1. Test metody *countElectrionHeadNeighbours()* będzię polegał na zliczeniu liczby sąsiadów typu *ElectronHead* dla następujących reprezentacji plansz *matrixOfCells*:

   (a)
   ```
   C H T
   C T E
   E H E
   ```

   Dla komórki *Conductor* na pozycji (0, 0) oczekiwana wartość 1.
   Dla komórki *Conductor* na pozycji (1, 0) oczekiwana wartość 2.

(b)

```
E H T
H C E
E H C
```

Dla komórki *Conductor* na pozcji (1, 1) oczekiwana wartość 3.
Dla komórki *Conductor* na pozcji (2, 2) oczekiwana wartość 1.

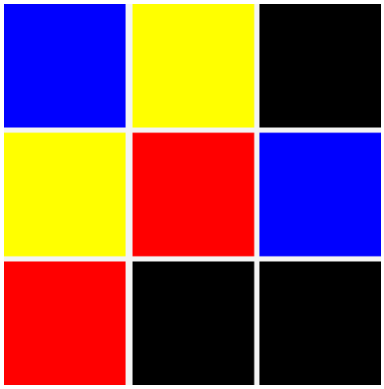
2. Test metody *nextGeneration()* będzie przeprowadzony na następnej generacji planszy:

```
E E E E E E E E E
E E E E C C E E E
C C T H C E C C C
E E E E C C E E E
```

Oczekiwany wynik jest następujący:

```
E E E E E E E E E
E E E E H C E E E
C C C T H E C C C
E E E E H C E E E
```

3. Test metody *readGraphicalMatrixofCells()* będzie przeprowadzony dla następującej graficznej reprezentacji planszy:



RYS.02 Graficzna reprezentacja testowej planszy

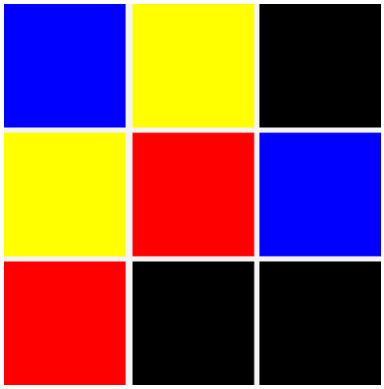
Oczekiwany wynik działania metody *readGraphicalMatrixofCells()* to uzyskanie planszy logicznej w postaci:

```
H C E
C T H
T E E
```

4. Test metody *showGraphicalMatrixofCells()* będzie przeprowadzony dla następującej logicznej reprezentacji planszy:

```
H C E
C T H
T E E
```

Oczekiwany wynik działania metody *showGraphicalMatrixofCells()* to uzyskanie planszy graficznej w postaci:

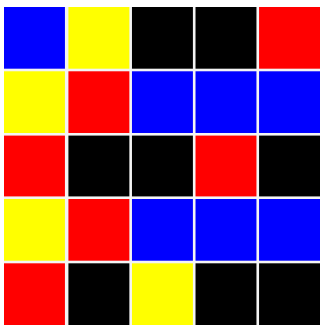


RYS.03 Graficzna reprezentacja oczekiwanej testowej planszy

4.3 Test Klasy MainController

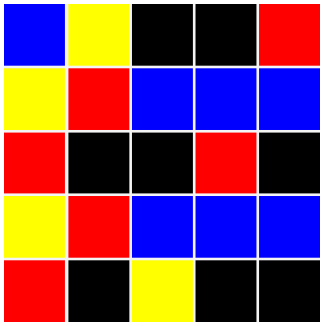
W klasie MainController będą przeprowadzone następujące testy:

1. Test metody *loadFile()* będzie polegał na wczytaniu obrazu PNG oraz sprawdzeniu czy wczytany obrazek został prawidłowo zinterpretowany, oraz czy wymiary planszy zostały prawidłowo ustalone:



RYS.04 Reprezentacja testowego pliku PNG

Oczekiwany widok planszy:

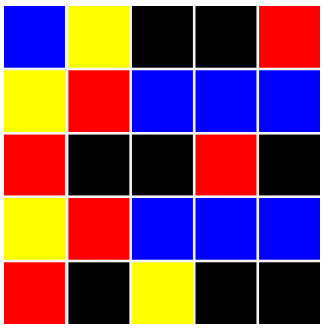


RYS.05 Oczekiwana sytuacja na planszy.

Wymiary planszy po wczytaniu obrazka mają postać:

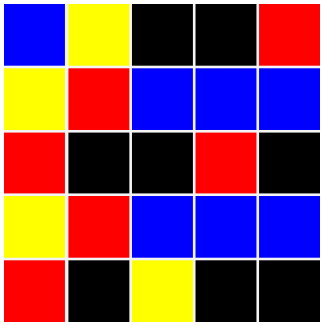
```
width: 5
height: 5
```

2. Test metody *saveFile()* będzię polegał na zapisaniu stanu obecnego na planszy do pliku PNG oraz sprawdzeniu czy zapisany obrazek został prawidłowo zinterpretowany:



RYS.06 Testowa sytuacja na planszy

Oczekiwana sytuacja na planszy:



RYS.07 Oczekiwany plik PNG

3. Testy metod *setEmpty*, *setConductor*, *setElectronHead*, *setElectrionTail* będą polegały na wybraniu odpowiedniego przycisku i sprawdzeniu czy po umieszczeniu elemntu na planszy, jest on tego samego typu co wybrany przycisk.

   (a) Metoda *setEmpty*: na planszy pojawi się element:

RYS.08 Reprezentacja elementu typu Empty.

   (b) Metoda *setConductor*: na planszy pojawi się element:

RYS.09 Reprezentacja elementu typu Conductor.

   (c) Metoda *setElectronTail*: na planszy pojawi się element:

RYS.10 Reprezentacja elementu typu ElectronTail.

   (d) Metoda *setElectronHead*: na planszy pojawi się element:

RYS.11 Reprezentacja elementu typu ElectronHead.