

Sprawozdanie końcowe

Michał Sut

1 Ostateczna implementacja projektu

1.1 Diagram klas

Zaimplementowane klasy oraz powiązania między nimi różnią się od tych wcześniej zaplanowanych. Spowodowane jest to po części brakiem doświadczenia w programowaniu obiektowym, a po części tym, że podczas tworzenia kodu pewne funkcjonalności okazały się być niepotrzebne lub zbyt skomplikowane. Ostateczny diagram klas wygląda następująco:



Rysunek 1: Diagram klas

1.2 Opis klas

1.2.1 WireWorld

Klasa startowa, zawiera metodę **main**. Tworzone są w niej okno główne, okno ustawień, wszystkie panele, obiekt klasy PanelsControl, który pośredniczy między panelami, oraz obiekt reprezentujący

cy ustawienia. Z klasy MyDimensions pobierane są wymiary okien i paneli.

1.2.2 SettingsWindow

Okno ustawień, zawiera dwa panele. Pierwszy z opcjami koloru, drugi z wyborem wielkości komórek.

1.2.3 WireWorldMainFrame

Główne okno programu. Zawiera 4 panele.

1.2.4 MyDimensions

Klasa zawierająca jedynie zmienne statyczne, określające wymiary poszczególnych elementów grafiki.

1.2.5 Settings

Obiekt tej klasy przechowuje aktualne ustawienia programu oraz udostępnia je innym obiektom.

1.2.6 PanelsControl

Obiekt tej klasy jest pewnego rodzaju "nadzorcą" i koordynatorem wszystkich paneli. Za jego pośrednictwem zdarzenia w jednym panelu mają efekty w innym, bez ich wzajemnej wiedzy o sobie.

1.2.7 Generator

Klasa, której obiekt odpowiedzialny jest za przeprowadzenie generacji. Posiada metody rozpoczynające i kończące generowanie kolejnych stanów planszy, a także metodę, która pozwala w dowolnym momencie "przeskoczyć" o jedną generację do przodu. Wiąże się z klasą Rules, w której zdefiniowane są zasady automatu komórkowego oraz sposób liczenia sąsiedztwa.

1.2.8 Rules

Obiekt tej klasy zawiera metodę, która dla podanej tablicy z informacjami o planszy oraz współrzędnych punktu, zwraca liczbę oznaczającą stan wskazanej komórki. Druga metoda, prywatna, służy do wyliczania komórek sąsiadujących.

1.2.9 ControlledPanel (interfejs)

Interfejs wszystkich paneli zarządzanych przez PanelsControl. Służy do tego, aby wymusić na każdym panelu posiadanie metody, która powiąże dany obiekt z PanelsControl.

1.2.10 BoardPanel

Panel, na którym rysowana jest plansza. Wiąże się z klasami: IO, FileData oraz Board, aby odpowiednio: zapisać swój stan do pliku, odczytać dane z pliku, lub narysować element w miejscu kliknięcia.

1.2.11 LeftButtonPanel

Panel przycisków z lewej strony okna. Odpowiada głównie za to, by przekazać "nadzorcy", a więc obiektowi klasy PanelsControl, informację o tym, jakie działanie ma zostać wykonane (inne dla każdego przycisku).

1.2.12 BottomButtonPanel

Panel z przyciskami wyboru wstawianej struktury. Informuje "nadzorcę" o tym, jaki element ma zostać wstawiony.

1.2.13 OptionsPanel

Panel w lewym dolnym rogu głównego okna. Przyciski w nim zawarte odpowiadają za ustawienie liczby generacji, resetowanie planszy, zapisanie generacji do pliku oraz wyjście z programu.

1.2.14 ColorSettingsPanel

Panel wchodzący w skład okna ustawień. Zawiera rozwijane listy kolorów dla każdego możliwego stanu komórki. Łączy się z obiektem klasy Settings, aby przekazać informację o wybranym kolorze.

1.2.15 OthersSettingsPanel

Panel, w którym miały znaleźć się pozostałe ustawienia, jednak ostatecznie zawarte zostało jedno - wybór wielkości komórki, a tym samym rozdzielczości planszy. Łączy się z obiektem przechowującym ustawienia, aby odczytywać i zapisywać wybraną wielkość komórki.

1.2.16 IO

Klasa zawierająca dwie publiczne metody statyczne pozwalające na zapisywanie/wczytywanie danych do/z pliku.

1.2.17 FileData

Klasa, której obiekty przechowują informacje pobrane z pliku. Zawiera listę elementów, którą udostępnia klasie Board w celu stworzenia planszy.

1.2.18 Board

Obiekt klasy Board przechowuje informacje o stanie wszystkich komórek na planszy. Wykonywane jest to za pomocą dwuwymiarowej tablicy liczb całkowitych. Zawiera metodę, która pozwala wstawić element do tablicy zarówno z pliku, jak i po kliknięciu w panel planszy.

1.2.19 Element

Ogólna klasa reprezentująca element. Każdy element ma nazwę, pozycję, oraz tablicę dwuwymiarową, w której przechowuje informacje o swoich komórkach.

1.2.20 EmptyCell, Wire, ElectronHead, ElectronTail, DiodeN, DiodeR, OrGate, XorGate, AndGate, NotGate

Klasy elementów jakie możemy wstawić na planszę. Dziedziczą po klasie Element. Każdy ma inaczej wypełnioną tablicę dwuwymiarową, tworzoną w konstruktorze.

2 Opis zmian

W tej części opisane są zmiany jakie zaszły zarówno w diagramie klas, jak i w funkcjonalności, ich przyczyny i skutki.

2.1 Klasy i powiązania między nimi

W tej kwestii zaszły największe zmiany. Dopiero pisanie projektu pokazało jakie klasy rzeczywiście będą potrzebne i jakie zastosować między nimi relacje. Liczba klas jest znacznie większa niż planowano, a powiązania między nimi są zapewniają odpowiednie działanie programu.

2.2 Wygląd okna

Wygląd okna zmienił się w bardzo małym stopniu względem tego, co zostało pokazane w specyfikacji funkcjonalnej. Doszedł jedynie panel z przyciskami resetu, ustawienia liczby generacji, zapisania do pliku i wyjścia z programu. Zmieniło się ustawienie przycisków wyboru wstawianych elementów.

2.3 Obsługa programu

Sposób obsługi aplikacji pozostał taki sam jak opisano w specyfikacji funkcjonalnej. Dane można wprowadzać z pliku bądź ręcznie.

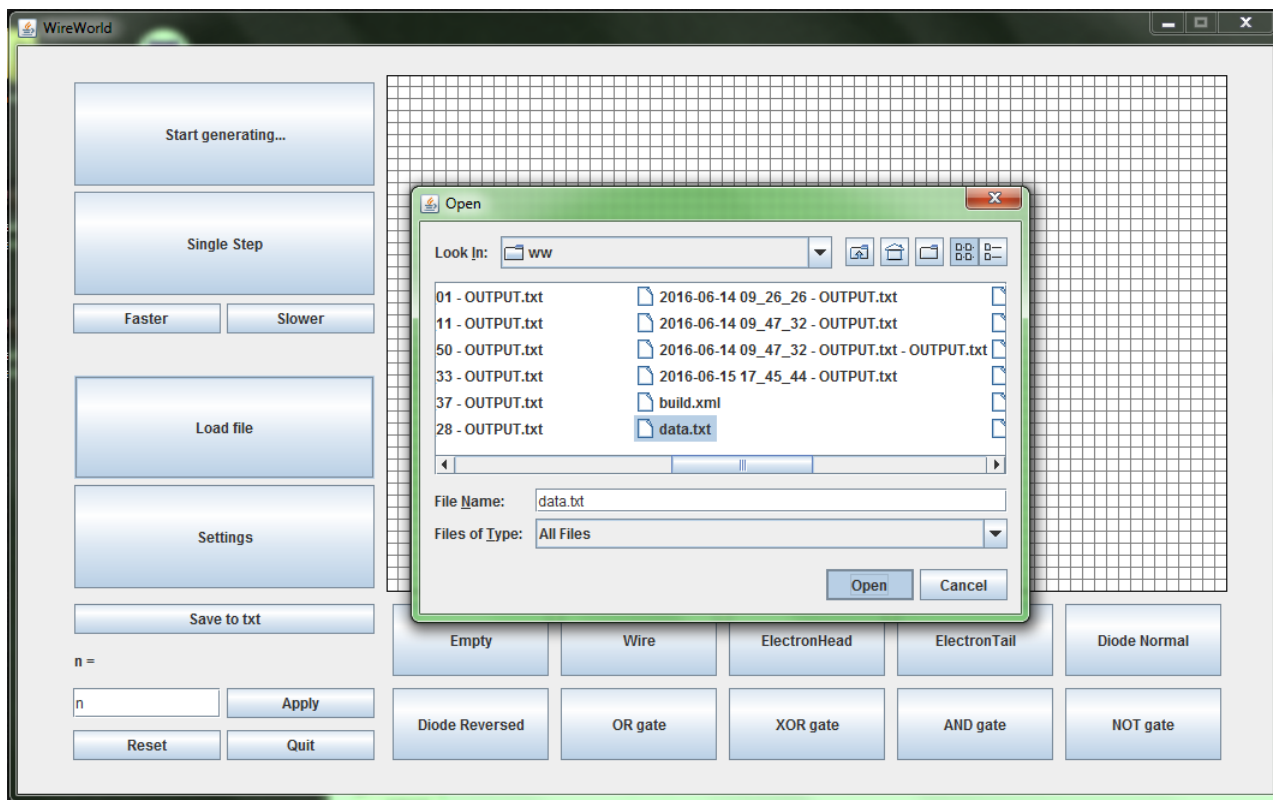
2.4 Dostępna funkcjonalność

Program posiada wiele dodatkowych, niewymaganych funkcji. Jednakże z powodu mocno ograniczonego czasu, nie udało się zaimplementować wszystkich wcześniej zaplanowanych funkcji. Część z nich została umyślnie niezaimplementowana.

3 Działanie programu

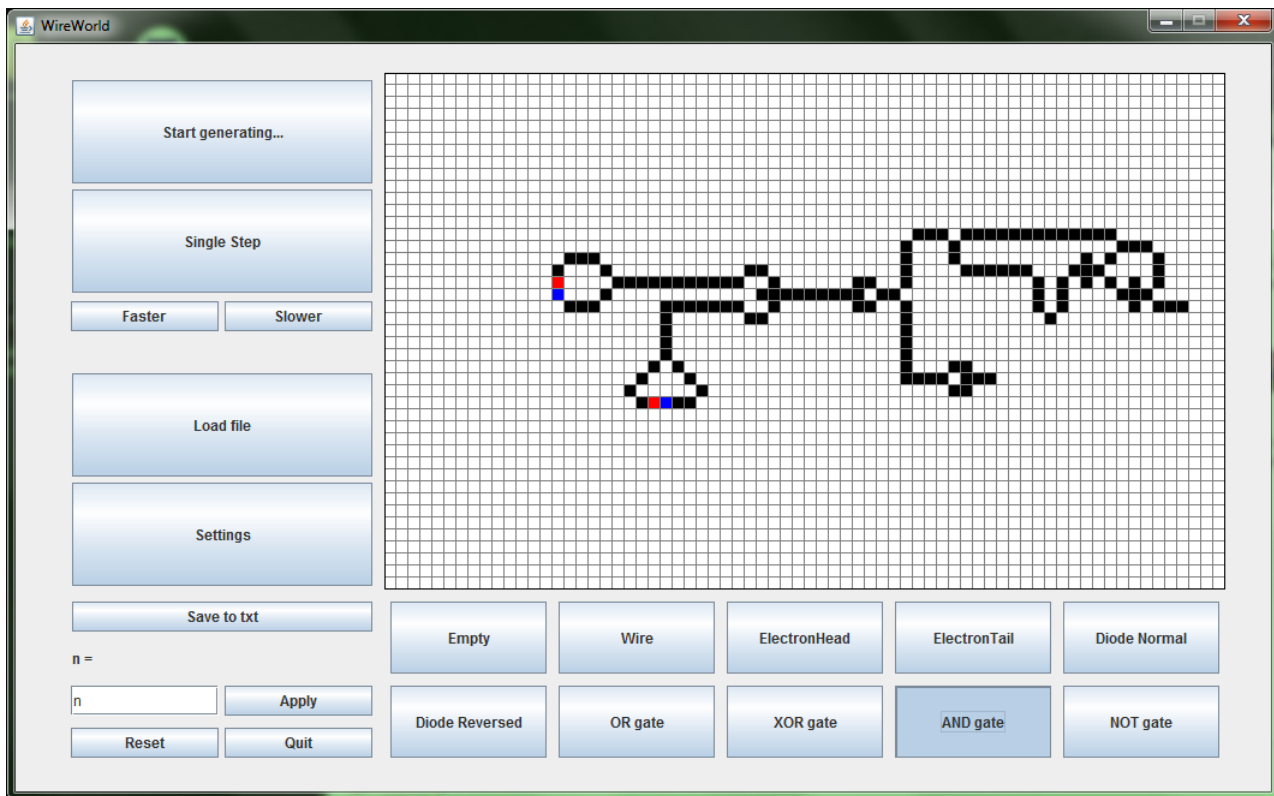
Przykładowe użycie aplikacji:

1. Wybór pliku z danymi:



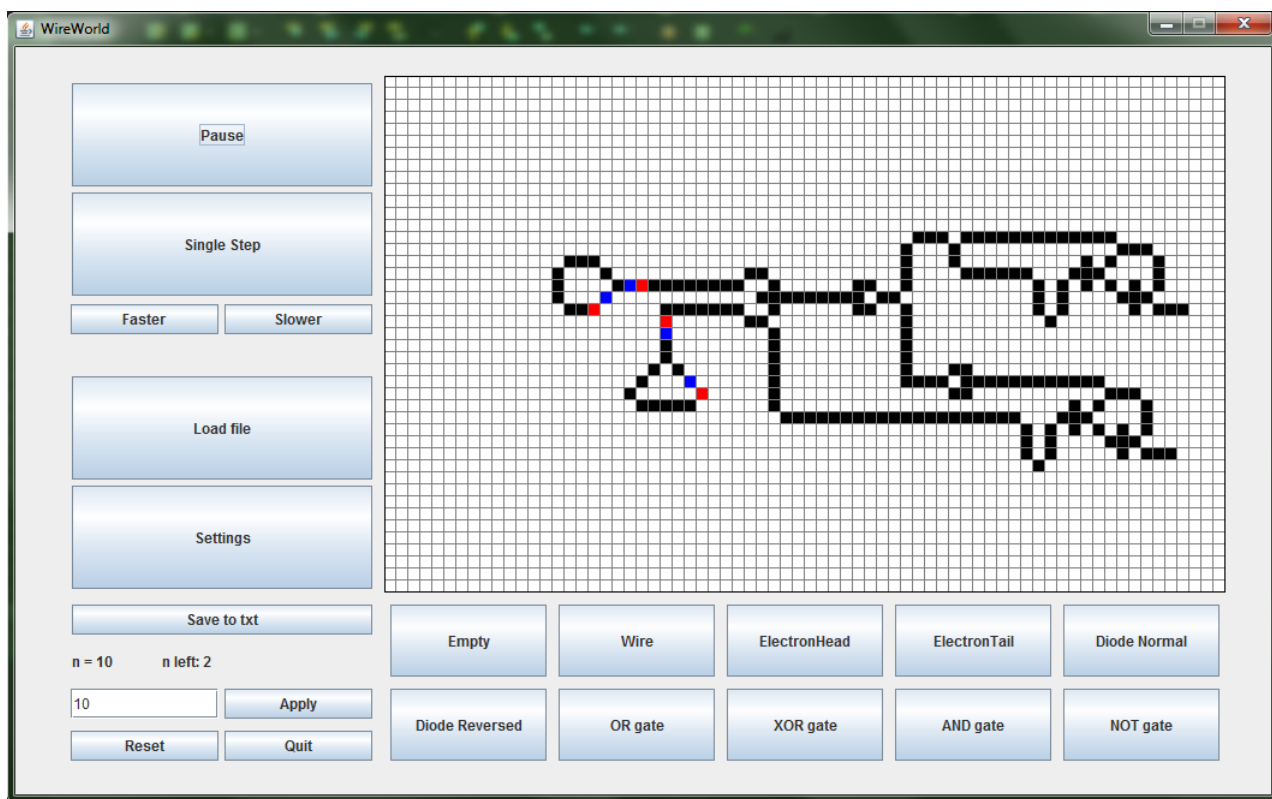
Rysunek 2: Wybór pliku

2. Wstawienie innych elementów za pomocą przycisków:



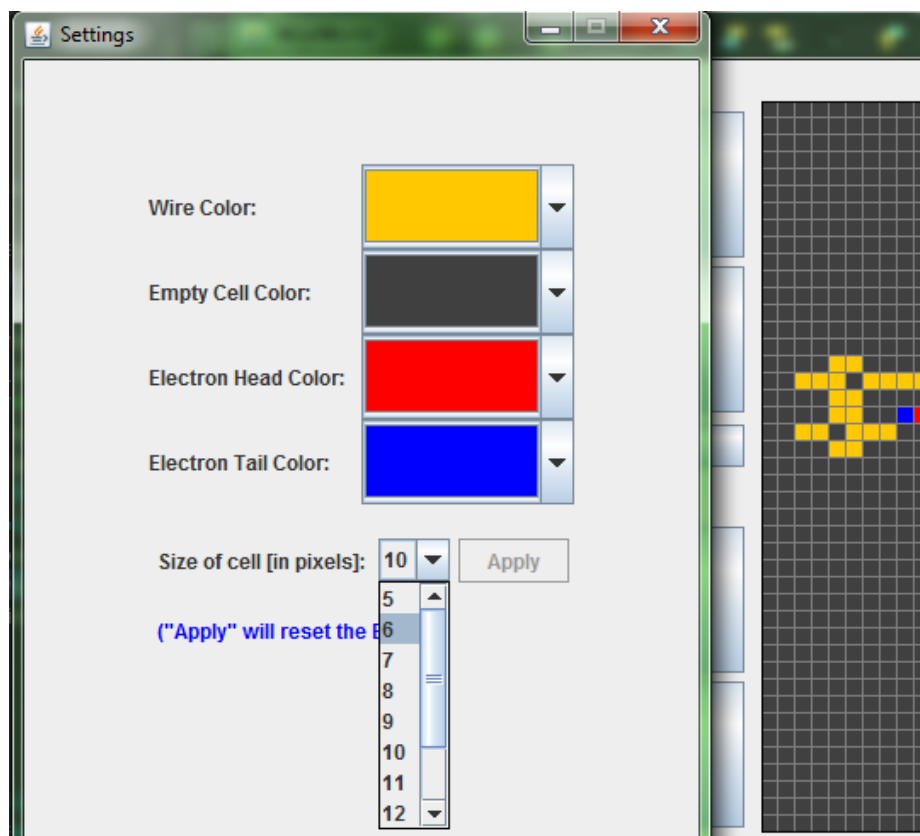
Rysunek 3: Wstawianie elementów za pomocą interfejsu graficznego

3. Przeprowadzenie określonej liczby generacji (pod przyciskiem "Save to Txt" widoczna jest liczba generacji pozostałych do przeprowadzenia):



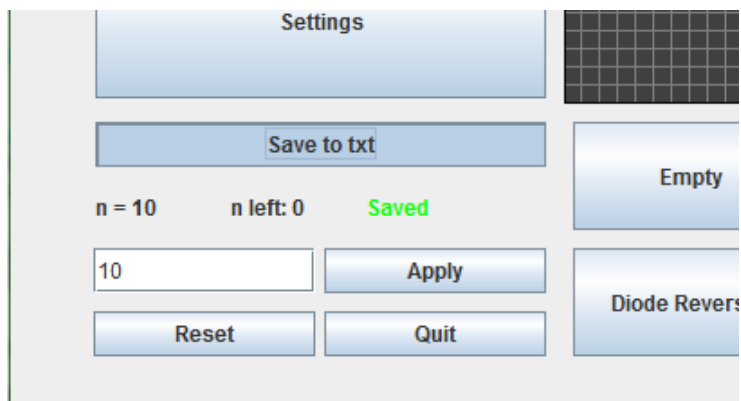
Rysunek 4: Aplikacja w trakcie przeprowadzania symulacji automatu komórkowego

4. Zmiana ustawień programu:



Rysunek 5: Zmiana kolorów oraz wielkości komórek

5. Zapisanie do pliku tekstowego:

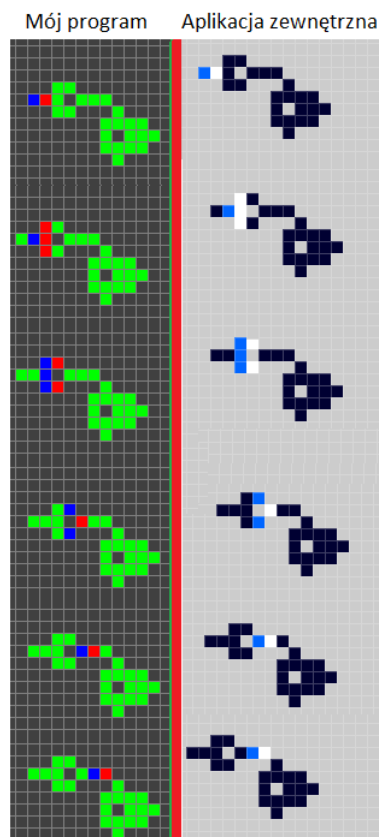


Rysunek 6: Zapisanie do pliku

4 Testowanie

4.1 Sprawdzenie, czy aplikacja poprawnie realizuje zasady automatu komórkowego "WireWorld".

Test ten będzie polegał na porównaniu trzech kolejnych stanów planszy z mojego programu z trzema kolejnymi etapami w programie zewnętrznym ([WireWorld - online](#)) Generacja początkowa będzie taka sama w obu przypadkach:



Rysunek 7: Porównanie działania dwóch niezależnych programów

4.2 Błędne dane

Dane uznaje się za błędne jeśli:

- Nazwa struktury jest nierozpoznawalna. Znane nazwy to: *empty*, *wire*, *electronhead*, *electrontail*, *diodenormal*, *diodereversed*, *orgate*, *xorgate*, *andgate*, *notgate*.
- Liczba określająca współrzędną jest ujemna, bądź nie jest liczbą całkowitą.
- W jednym wierszu pliku znajdują się mniej niż 3 elementy (nazwa, x, y).
- Współrzędne punktu wykraczają poza obszar planszy.

W celu przeprowadzenia testów przygotowałem kilka plików tekstowych, każdy zawierał inny rodzaj niepoprawnych danych. Wyniki prezentują się następująco:

Błędna nazwa struktury w linii nr. 2	Komunikat „Wrong data format in line 2”
Ujemna liczba w wierszu 1	Komunikat „Wrong data format in line 1”
Mniej niż 3 elementy w wierszu 3	Komunikat „Wrong data format in line 3”
Współrzędne punktu wykraczają poza obszar planszy	Komunikat „Not enough space to insert element”

W pierwszych 3 przypadkach, program wyświetlał komunikat o błędzie oraz dodatkowo przez chwilę pojawiała się informacja „Opening failed” pod przyciskiem „Save to txt”. Do planszy nie były wczytywane żadne dane. W ostatnim przypadku, gdy współrzędne wychodzą poza zakres planszy, wyświetla się odpowiedni komunikat, jednakże wszystkie elementy, których współrzędne były poprawne zostają umieszczone na planszy.

4.3 GUI

Test interfejsu graficznego opierał się na sprawdzeniu czy wszystkie przyciski działają, czy spełniają poprawnie swoją rolę, oraz czy kilkukrotne kliknięcie tego samego przycisku nie spowoduje błędu. Program w tym aspekcie nie wykazał nieprawidłowości. Wszystkie przyciski działają. Pomimo tego, zauważyłem pewną cechę, która powinna zostać poprawiona. Mianowicie, przycisk „Save to txt” wyświetla zbyt słabo widoczną informację o tym, czy zapis się powiódł.

4.4 Testy klas odpowiedzialnych za wczytywanie i przechowywanie danych

Przetestowane zostaną klasy `FileData`, `Board` oraz `Settings`, a `IO` oraz `Element` (oraz jej klasy dziedziczące) zostaną naturalnie sprawdzone razem z testami klasy `FileData`.

4.4.1 FileData, IO, Element

Test polega na wczytaniu danych z pliku, a następnie wypisaniu danych jakie przechowuje obiekt typu `FileData` i porównanie ich plikiem. Zawartość zostanie wypisana na konsolę przy użyciu metody `toString()`

Poprawne dane w pliku:

```
wire 0 1
wire 0 2
electronhead 0 3
orgate 1 5
```


Wynik w konsoli:

```
FileData has succesfully read 4 elements:  
wire( 0,1 ) - Dimension: 1 x 1  
wire( 0,2 ) - Dimension: 1 x 1  
electronhead( 0,3 ) - Dimension: 1 x 1  
orgate( 1,5 ) - Dimension: 3 x 3
```

Błędne dane w pliku:

```
wire 0 1  
wire 0 2  
electronhead -1 3  
gateor 1 5
```

Wynik w konsoli:

```
Wrong data format in line 3!
```

Poprawność działania klasy IO możemy rozpoznać po tym, że odrzucony został plik z błędnymi danymi, a poprawny wczytany. Klasa Element działa poprawnie, ponieważ oprócz nazwy i współrzędnych elementów na wyjściu otrzymujemy także wymiary danej struktury, a to właśnie informacje uzyskane z obiektów klas dziedziczących. Poprawność FileData wynika z tego, że obiekt poprawnie wypisał elementy, które przechowywał.

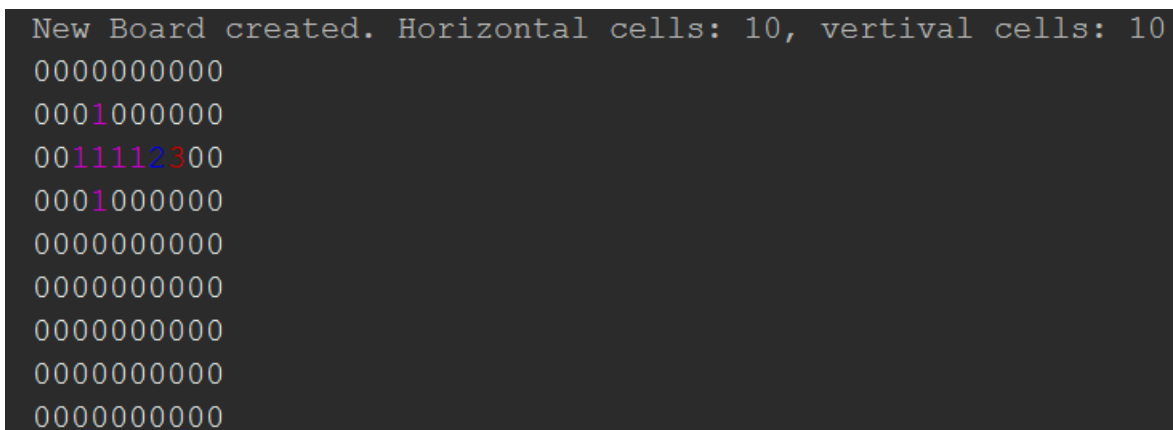
4.4.2 Board

Obiektowi klasy Board zostanie przekazany obiekt FileData, aby sprawdzić, czy dane zostaną odpowiednio przekształcone na tablicę dwuwymiarową liczb całkowitych.

Dane w pliku:

```
orgate 2 2  
wire 5 2  
electronhead 7 2  
electrontail 6 2
```

Wyniki z konsoli (W postaci zrzutu ekranu, ponieważ widać różne kolory dla różnych elementów):



```
New Board created. Horizontal cells: 10, vertival cells: 10  
0000000000  
0001000000  
0011112300  
0001000000  
0000000000  
0000000000  
0000000000  
0000000000  
0000000000  
0000000000
```

Rysunek 8: Wynik testu klasy Board

Można łatwo sprawdzić, że dane zostały prawidłowo przekształcone.

4.5 Settings

Działanie tej klasy najłatwiej sprawdzić zmieniając ustawienia w trakcie działania programu. Kolory można zmieniać na bieżąco, nawet podczas uruchomionej automatycznej generacji. Wszystkie kolory zmieniają się poprawnie. Zmiana szerokości komórki wymusza zresetowanie programu. Możemy dokonać tego zarówno w stanie gdy generator jest wyłączony jak również w trakcie odtworzenia (spowoduje to zatrzymanie generatora). Każda wielkość komórki dostępna z rozwijanej listy jest prawidłowo wyświetlana, jednakże, dla rozmiarów, które nie są dzielnikami liczb 700 i 430 (wymiary planszy w pikselach) powstanie niewielka wolna przestrzeń po prawej stronie i na dole planszy.

4.6 Generator + Rules

Te dwie klasy, które odpowiadają za przeprowadzenie generacji zostały już pośrednio sprawdzone w teście na realizację zasad WireWorld. Poprawność przeprowadzonych wtedy generacji świadczy o prawidłowym działaniu obiektów klas Generator i Rules.

5 Podsumowanie błędów

Realizacja zasad WireWorld	Brak błędów
Błędne dane	Brak błędów
GUI	Niewielki błąd - słabo widoczny komunikat o zapisie do pliku
Wczytywanie i przechowywanie danych	Brak błędów
Klasa Settings	Komórki dopasowują się do planszy, tylko gdy ich wymiary są dzielnikami szerokości i wysokości tej planszy.
Generator + Rules	Brak błędów