WireWorld - Sprawozdanie

Michał Jereczek

10 czerwca 2014

Spis treści

1 Specyfikacja funkcjonalna

1.1 Opis programu

Głównym zadaniem programu Wireworld jest umożliwienie użytkownikowi symulacji automatu komórkowego działającego na zasadach "Wireworld" Briana Silvermana. Użytkownik powinien być w stanie ustalić początkową generację komórek i na jej podstawie móc prześledzić kolejne generacje za pośrednictwem graficznego interfejsu użytkownika.

1.2 Wymagania funkcjonalne

Program musi pozwalać na dwa tryby wprowadzania (przekazywania) danych odnośnie początkowej konfiguracji komórek. Pierwszym jest wczytywanie z tekstu, drugim tworzenie konfiguracji z wykorzystaniem graficznego interfejsu użytkownika. Program powinien pozwalać na łączenie tych metod - wczytanie początkowej konfiguracji z pliku, i późniejsze jej modyfikowanie przed generowaniem. Po ustaleniu początkowej konfiguracji program ma za zadanie przedstawić jej przebieg w postaci interaktywnej animacji (użytkownik powinien mieć możliwość przemieszczania się pomiędzy poszczególnymi generacjami, bądź włączenia automatycznej "animacji").

1.2.1 Struktura pliku tekstowego

W Wireworld wyróżniamy podane poniżej stany komórek:

- EMPTY - pusta komórka
- CONDUCTOR - przewodnik
- TAIL - ogon elektrony
- HEAD - głowa elektronu

  oraz struktury:
- AND - bramka logiczna AND
- OR - bramka logiczna OR
- DIODE - Dioda
- ELECTRON - elektron
- CLOCKS - mały generator

- CLOCKB - duży generator

Natomiast w Life:

- DEAD - martwa komórka
- ALIFE - żywa komórka

Plik tekstowy powinien zostać zapisany z rozszerzeniem .automaton i składać się co najmniej z dwóch linijek: pierwsza informująca o szerokości i wysokości siatki, następna o regułach automatu, gdzie pierwszy wyraz określa zasady (Wireworld albo Life), natomiast drugi o rodzaju granic (normal lub wrapped). Następnie powinny być w dokumencie zdefiniowane co linijkę kolejne punkty bądź struktury komórek w taki sposób, że:

- Strukturę lub stan komórek reprezentuje jeden wyraz i 2 uporządkowane liczby: NAZWA struktury(stanu), położenie X i Y (w przypadku struktury jesto to punkt zaczepienia komórki najbardziej wysuniętej w górny-lewy róg tej struktury.)

Przykład:

40 30
Wireworld wrapped
AND 2 2
TAIL 2 18
CONDUCTOR 40 18
HEAD 2 19

informuje o siatce rozmiarów 40x30 będącej interpretacją Wireworld z "zawiniętymi" granicami. posiadającej strukturę AND zahaczoną w pkt. 2/2, ogon elektronu w pkt 2/18, przewodnik w pkt. 40/18, głowę elektronu w pkt. 2/19.

1.2.2 Opcje interfejsu graficznego

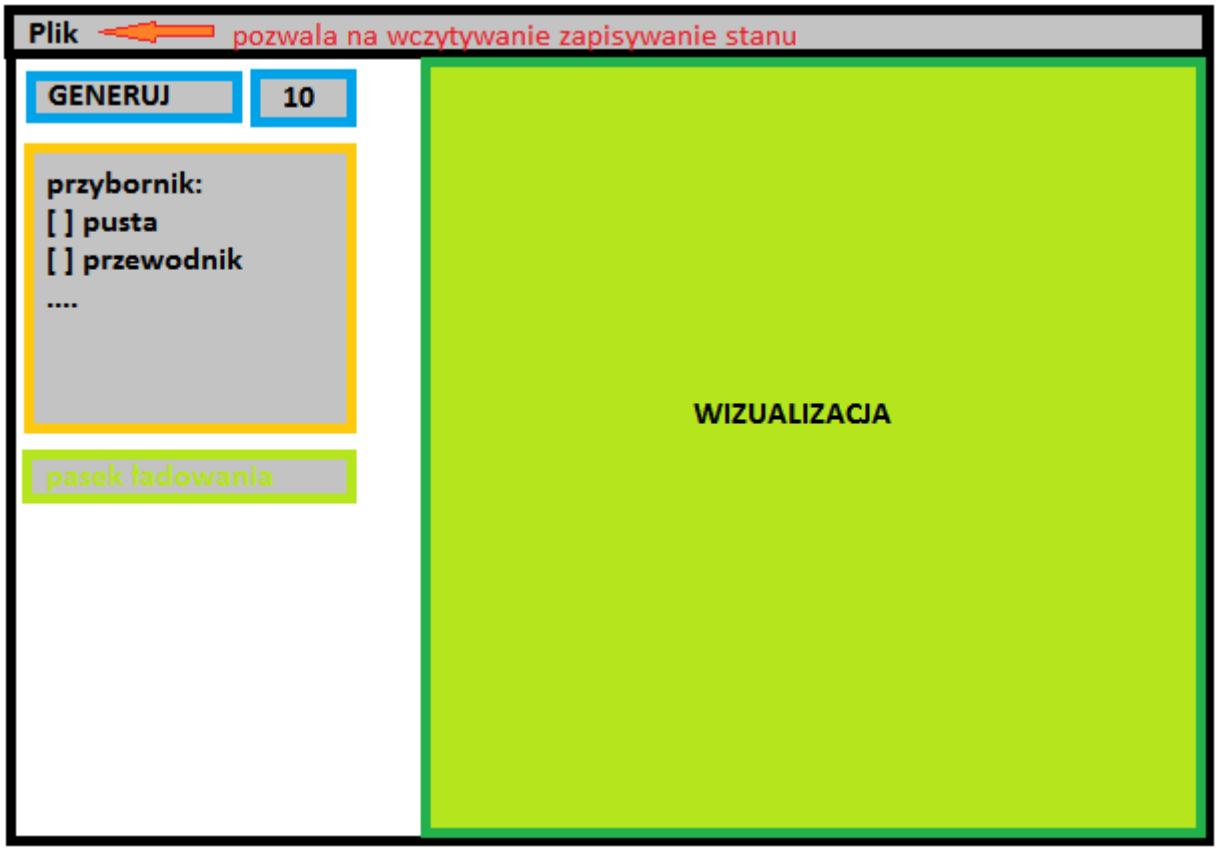
Interfejs graficzny powinien pozwalać na zdefiniowanie szeregu opcji z jakimi zostanie przeprowadzona symulacja. Powinien również zapewnić konkretne możliwości w czasie samego procesu generowania. Do opcji (możliwości) tych należą:

- wczytywanie konfiguracji z pliku
- projektowanie konfiguracji początkowej z użyciem predefiniowanych struktur, bądź pojedynczych komórek.

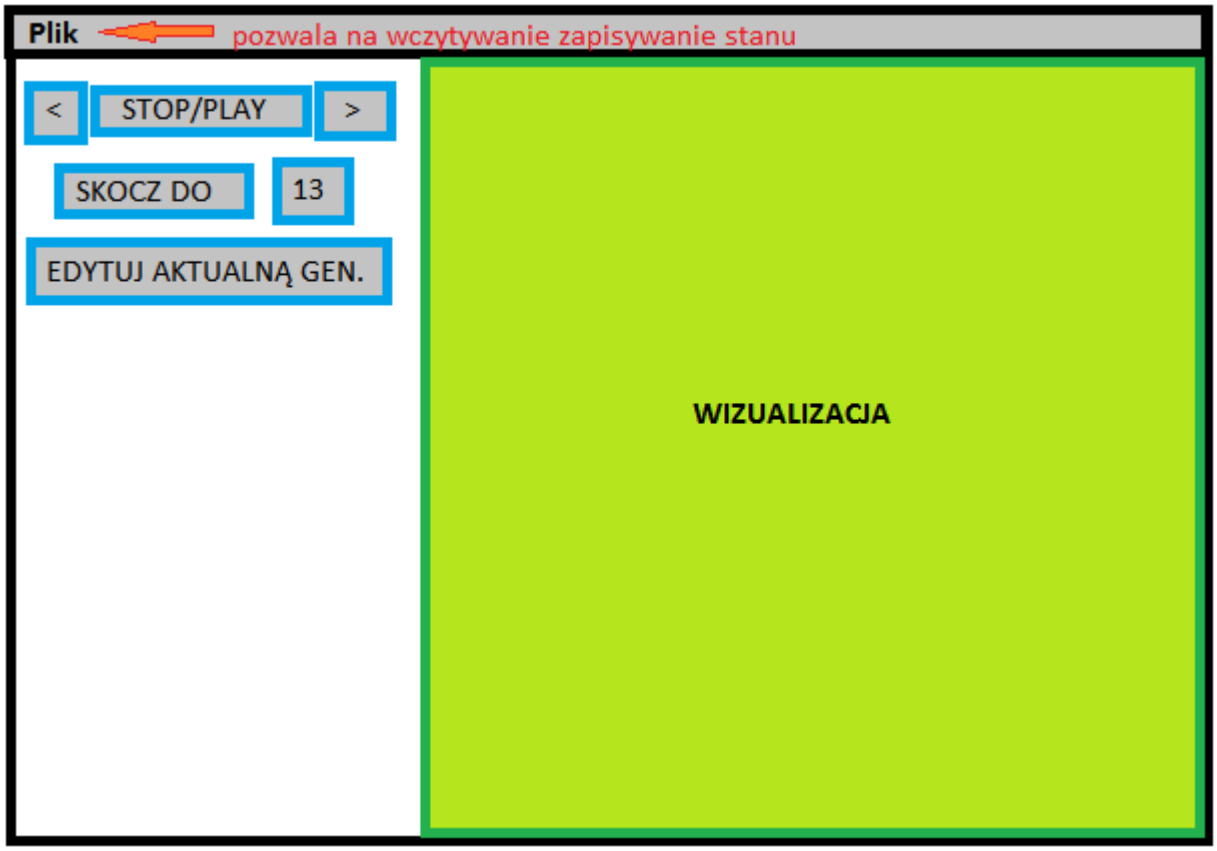
- ustalenie liczby generacji do przeprowadzenia
- zapisanie aktualnego stanu siatki do pliku tekstowego
- przeskakiwanie do konkretnej generacji
- cofanie/przewijanie do przodu generacji
- automatyczne przewijanie kolejnych generacji (animacja)

1.2.3 Szkic poglądowy interfejsu graficznego

Tryb edycji.



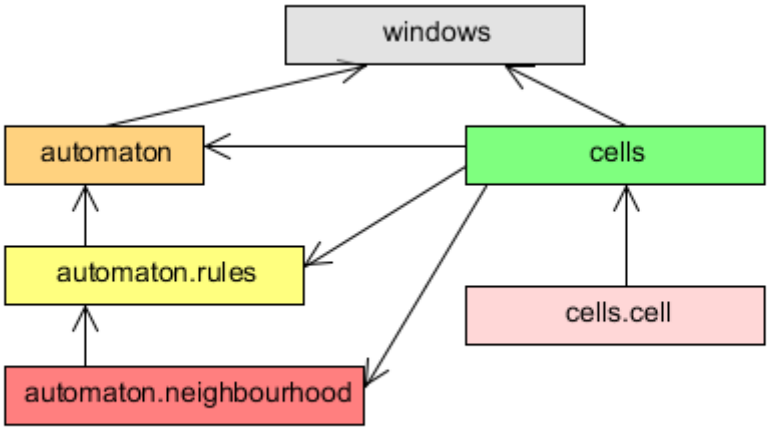
Tryb odtwarzania.



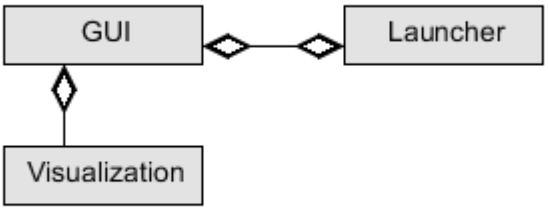
2 Specyfikacja implementacyjna

2.1 Diagram klas

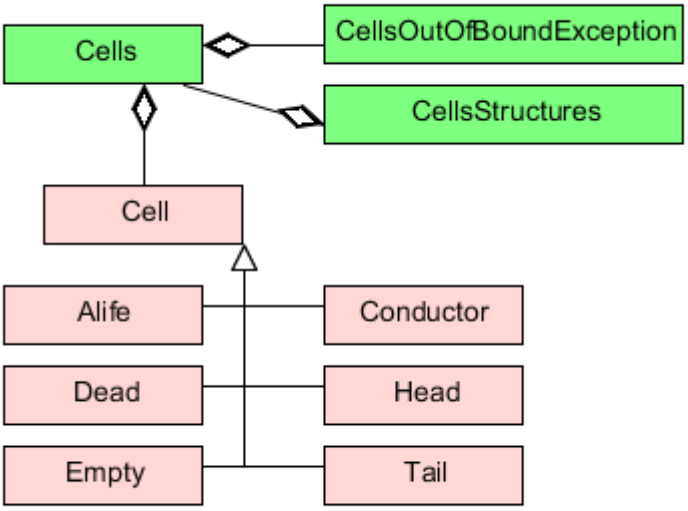
2.1.1 Podział na paczki



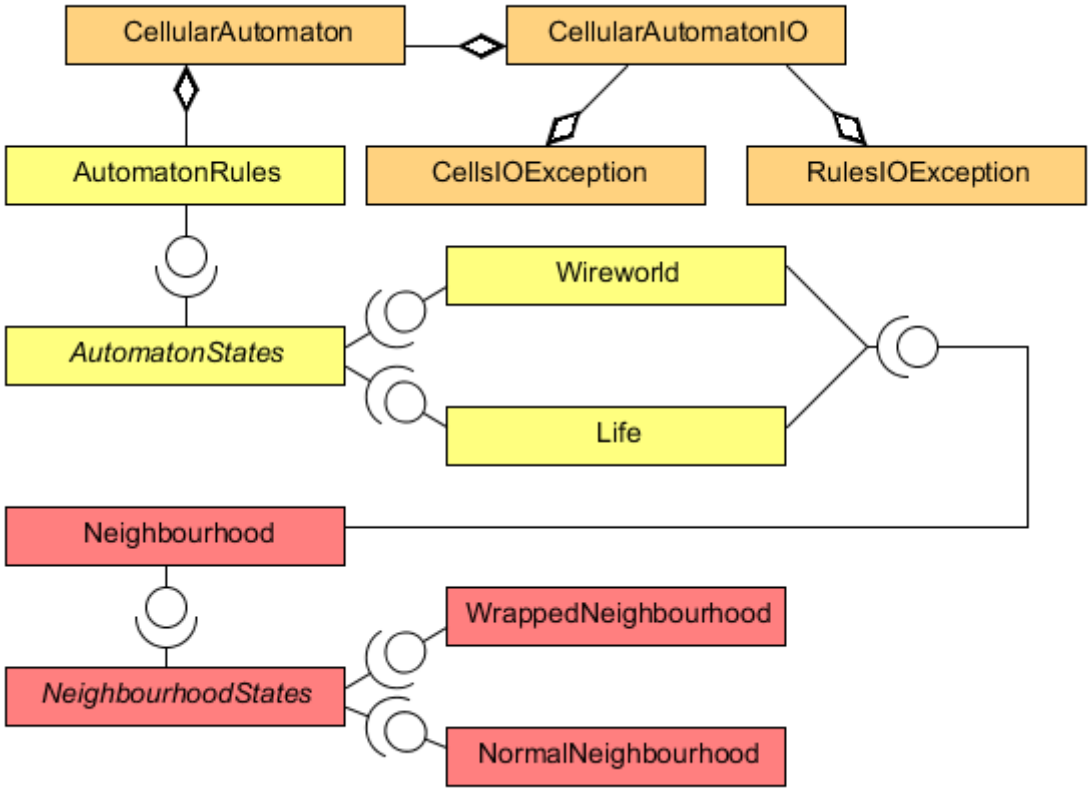
2.1.2 windows



2.1.3 cells

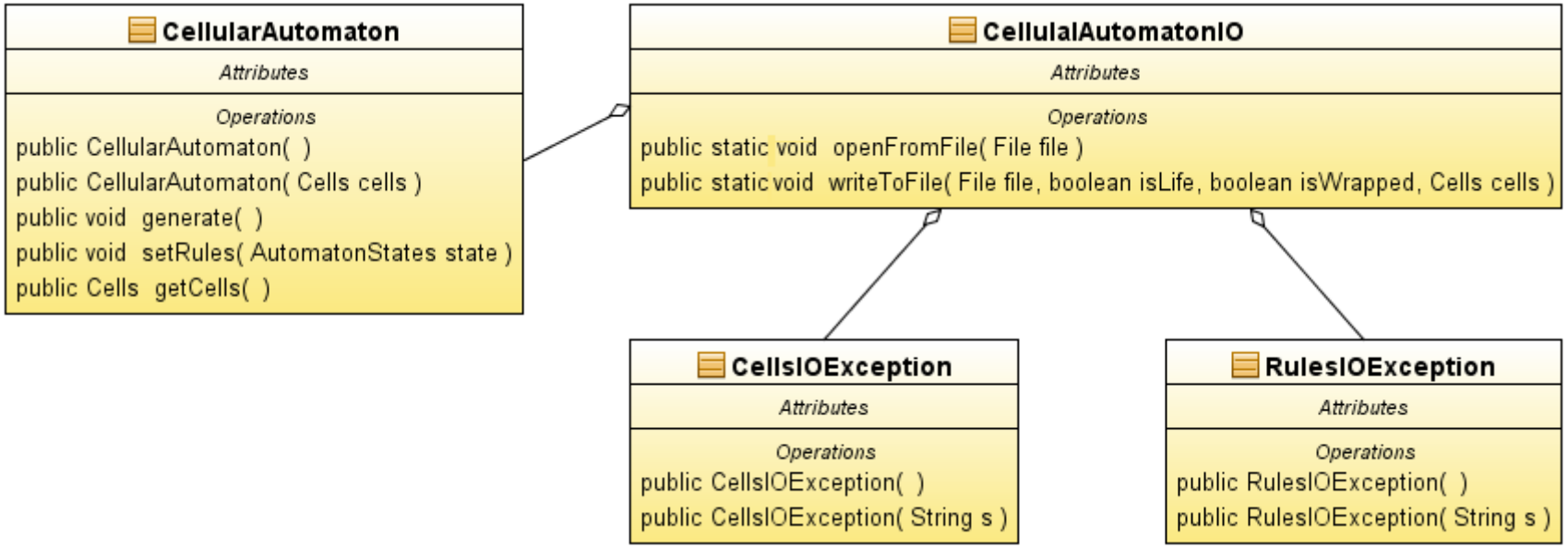


2.1.4 automaton

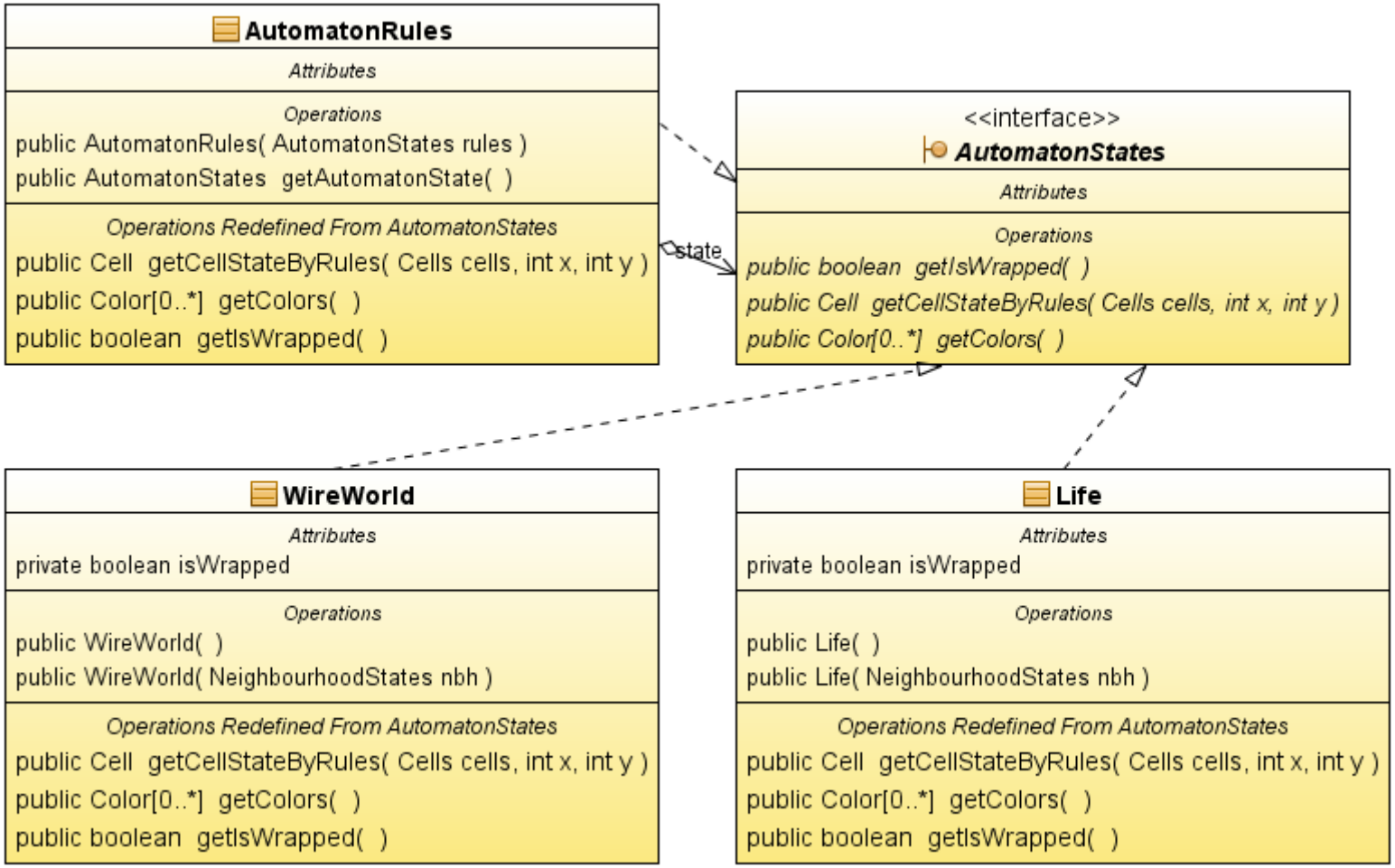


2.2 Szczegółowy diagram klas

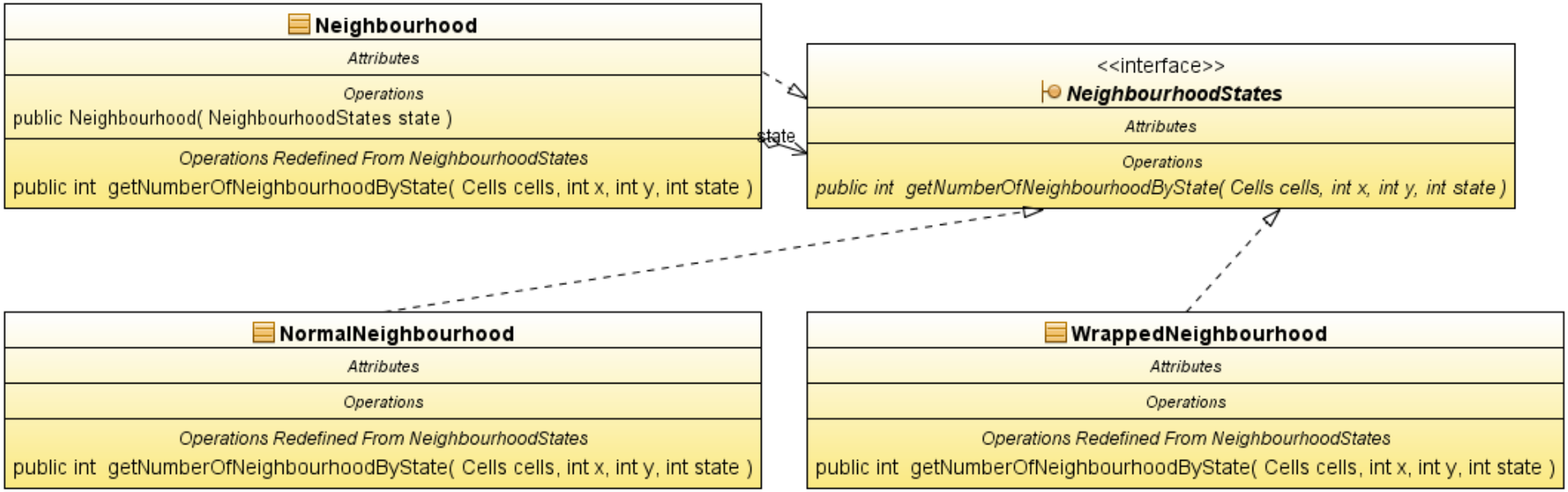
2.2.1 automaton



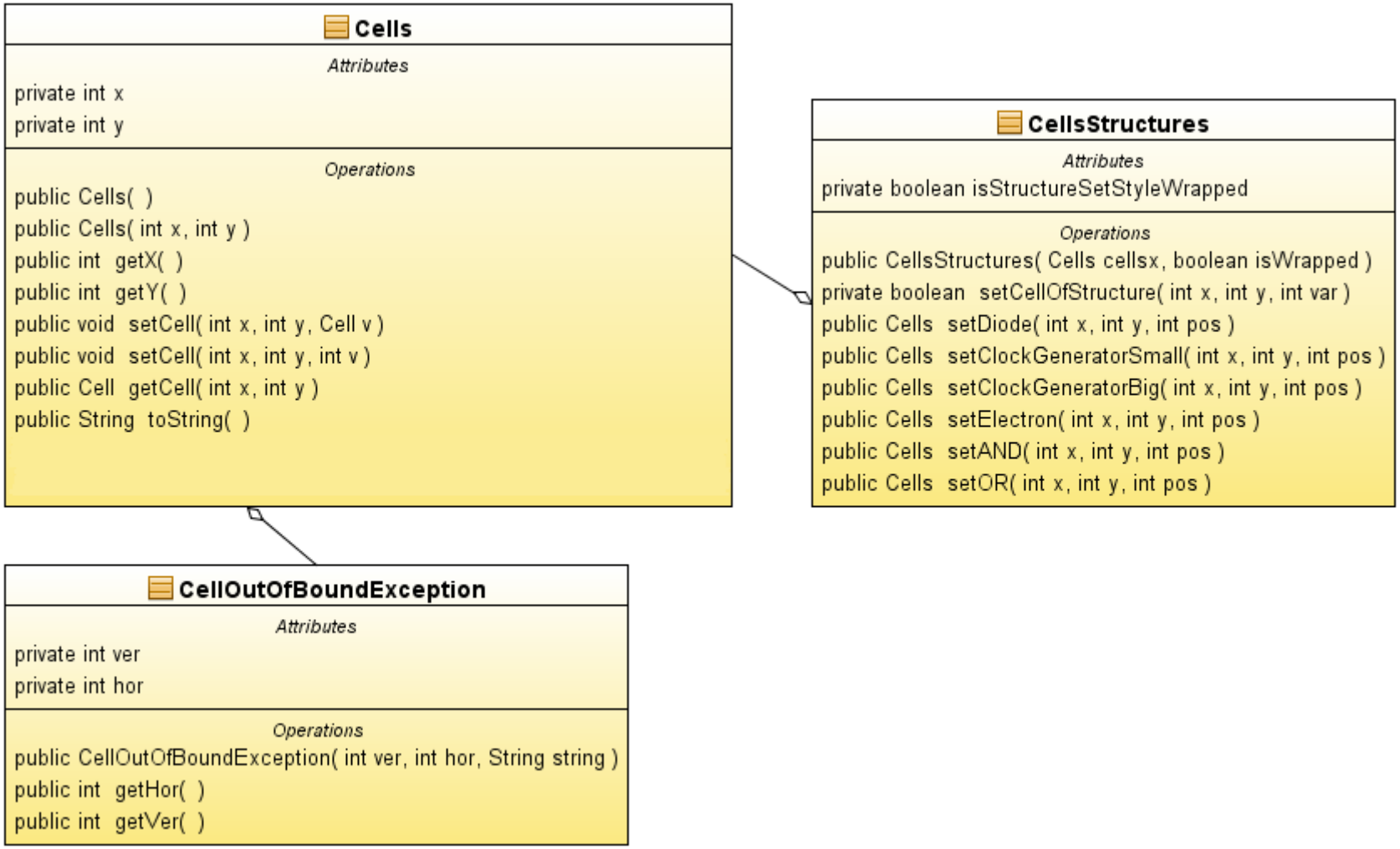
2.2.2 automaton.rules



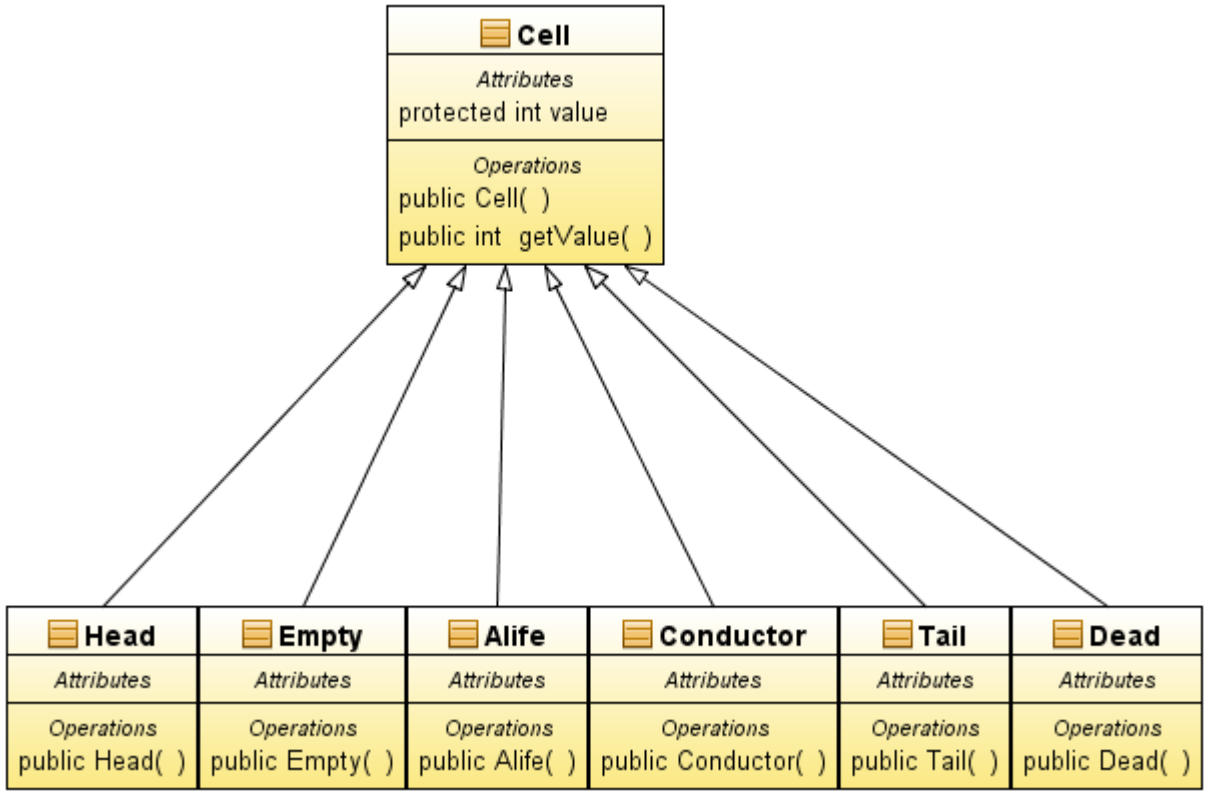
2.2.3 automaton.neighbourhood



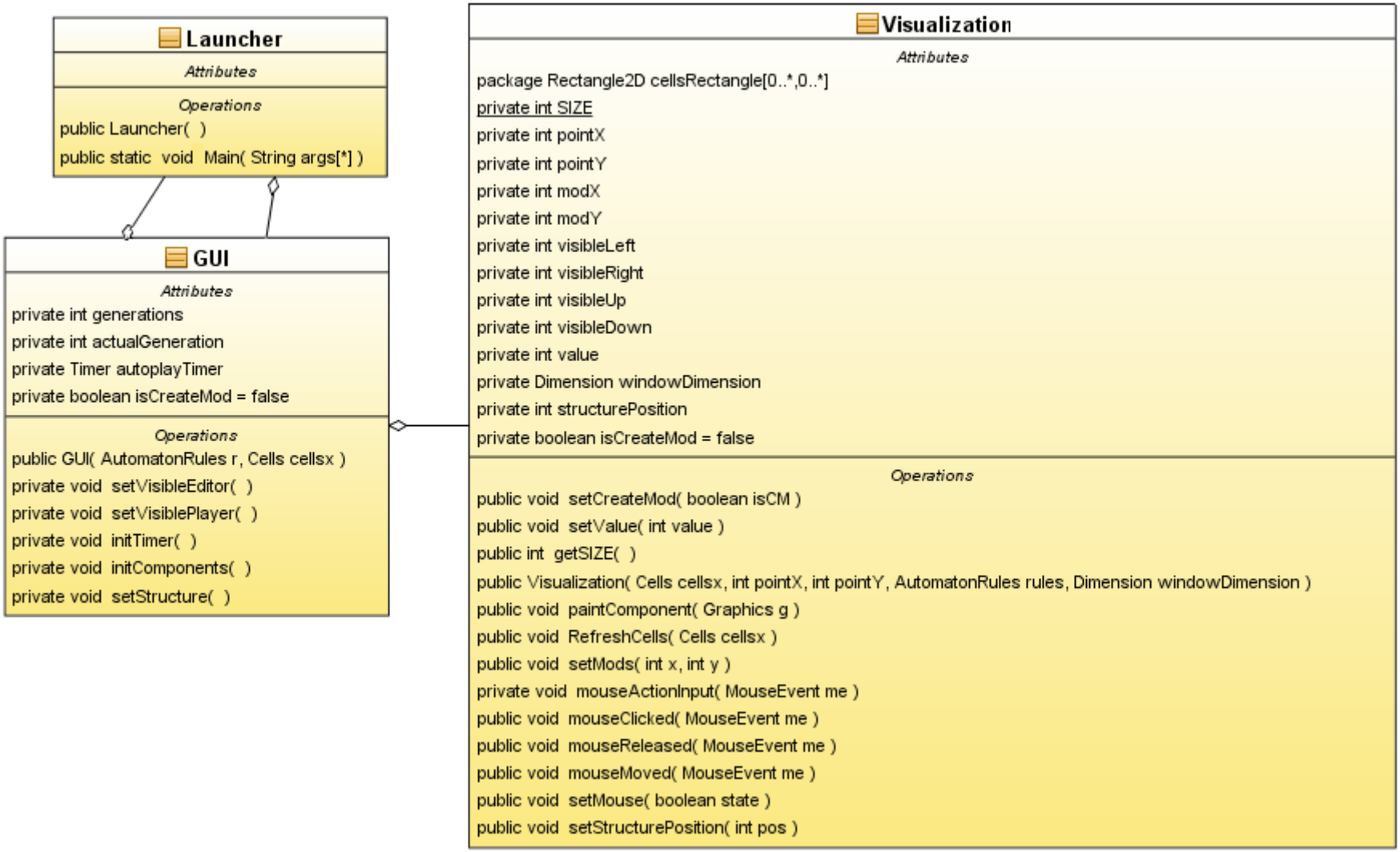
2.2.4 cells


Cells
Attributes
private int x
private int y
Operations
public Cells( )
public Cells(int x, int y)
public int getX( )
public int getY( )
public void setCell(int x, int y, Cell v)
public void setCell(int x, int y, int v)
public Cell getCell(int x, int y)
public String toString( )
CellsStructures
Attributes
private boolean isStructureSetStyleWrapped
Operations
public CellsStructures(Cells cellsx, boolean isWrapped)
private boolean setCellOfStructure(int x, int y, int var)
public Cells setDiode(int x, int y, int pos)
public Cells setClockGeneratorSmall(int x, int y, int pos)
public Cells setClockGeneratorBig(int x, int y, int pos)
public Cells setElectron(int x, int y, int pos)
public Cells setAND(int x, int y, int pos)
public Cells setOR(int x, int y, int pos)
CellOutOfBoundException
Attributes
private int ver
private int hor
Operations
public CellOutOfBoundException(int ver, int hor, String string)
public int getHor( )
public int getVer( )

2.2.5 cells.cell



2.2.6 windows


Launcher
Attributes
Operations
public Launcher( )
public static void Main(String args[*])
GUI
Attributes
private int generations
private int actualGeneration
private Timer autoplayTimer
private boolean isCreateMod = false
Operations
public GUI(AutomatonRules r, Cells cellsx)
private void setVisibleEditor( )
private void setVisiblePlayer( )
private void initTimer( )
private void initComponents( )
private void setStructure( )
Visualization
Attributes
package Rectangle2D cellsRectangle[0..*,0..*]
private int SIZE
private int pointX
private int pointY
private int modX
private int modY
private int visibleLeft
private int visibleRight
private int visibleUp
private int visibleDown
private int value
private Dimension windowDimension
private int structurePosition
private boolean isCreateMod = false
Operations
public void setCreateMod(boolean isCM)
public void setValue(int value)
public int getSIZE( )
public Visualization(Cells cellsx, int pointX, int pointY, AutomatonRules rules, Dimension windowDimension)
public void paintComponent(Graphics g)
public void RefreshCells(Cells cellsx)
public void setMods(int x, int y)
private void mouseActionInput(MouseEvent me)
public void mouseClicked(MouseEvent me)
public void mouseReleased(MouseEvent me)
public void mouseMoved(MouseEvent me)
public void setMouse(boolean state)
public void setStructurePosition(int pos)

2.3 Opis Klas

2.4 windows

Paczka ta zawiera klasy odpowiadające za wyświetlanie oraz sterowanie automatem. W sprawozdaniu zostały pominięte opisy metod generowanych automatycznie dla interfejsu graficznego np. jButtonGenerate, ich nazwy są na tyle proste, że można z łatwością domyśleć się ich zastosowania na podstawie samego użytkowania kontrolek w interfejsie.

Chciałem ponadto zwrócić uwagę na nieużycie wzorca projektowego Obserwator w projekcie, mimo, że była taka prośba od prowadzącego. Związane jest to z założeniami (specyfikacją funkcjonalną), która została sporządzona przed informacją o potrzebie użycia wyżej wymienionego wzorca. Z założeń wynika, że program powinien pozwalać na dowolne przeskakiwanie pomiędzy poszczególnymi generacjami, użycie obserwatora było by uciążliwe dla użytkownika np. przy hipotetycznej sytuacji, gdzie aktualnie wyświetlana jest generacja nr. 650, a użytkownik chciałby wrócić, zobaczyć 644, przy użyciu obserwatora program musiałby wczytać jeszcze raz generację początkową i następnie przeprowadzić 644 generacji (mimo, że zrobił to już wcześniej tworząc 650 generacji), z punktu użytkownika skutkowałoby to chwilą oczekiwania na dostanie wyników, co byłoby uciążliwe (np. gdyby użytkownik skakał do tyłu o jedną generację, wada ta była by nieznośna).

2.4.1 Launcher

Graficzny interfejs otwierany przed samym użyciem GUI aby w prosty sposób określić zasady na jakich chcemy działać w trakcie użytkowania automatu.

- public Launcher() - Konstruktor wyświetlający nowe okno launcher'a.
- public static void Main() - Wywołuje launcher przy uruchomieniu programu.

2.4.2 GUI

Klasa odpowiadająca za komunikację na poziomie użytkownik-program poprzez graficzny interfejs użytkownika.

- public GUI() - Konstruktor wyświetlający nowe okno sterowania i zarządzania automatem komórkowym.
- public setVisibleEditor() - Ukrywa odtwarzacz generacji, odkrywa edytor siatki.

- public setVisiblePlayer() - Odkrywa odtwarzacz generacji, zakrywa edytor siatki.

- private void initTimer() - Inicjuje timer potrzebny do "autoodtwarzania" animacji generacji.

- private void setStructure() - Dodatkowa metoda wspomagająca używanie rozwijanej listy struktur w GUI.

2.4.3 Visualisation

Klasa odpowiadająca za wizualizowanie w postaci graficznej stanu siatki komórkowej.

- public void setValue(int value) - ustawia wartość(strukturę) jaka będzie wprowadzana do siatki w edytorze (w GUI mamy przybornik pozwalający na wybieranie struktur np. AND, OR).

- public int getSIZE() - zwraca szerokośc pojedynczej komórki.

- public Visualization(Cells cellsx, int pointX, int pointY, AutomatonRules rules, Dimension windowDimension) - Konstruktor służący za zainicjowanie wyświetlania siatki. Do działania potrzebuje dostać parametry: cellsx - siatka do wizualizowania, pointX i pointY - punkt zaczepienia animacji względem nadrzędnej jFrame (w tym wypadku GUI), d - rozmiary nadrzędnego jFrame (długość i szerokość), rules - reguły w których zawarte są informacje o kolorach dla poszczególnych wartości komórek.

- public void paintComponent(Graphics g) - Odpowiada za rysowanie figur.

- public void RefreshCells(Cells cellsx) - Na podstawie nowo zadanej siatki odświeża obraz.

- public void setMods(int x, int y) - Ustawia modyfikatory wyświetlanej siatki (efekt przesuwania obrazu przy użyciu jScrollów).

- public void mouseActionInput(MouseEvent me) - Akcja dla użycia myszki (kliknięcia) pozwalająca edytować siatkę.

- public void setMouse(boolean state) - Ustawia możliwość edytowania (nasłuchiwania) myszki. Należy wywołać np. gdy wychodzimy z trybu edytora w programie.

- public void setStructurePosition(int pos) - Struktury mogą mieć różne pozycje (obrót o 90 stopni), metoda pozwala przekazać informację o żądanej pozycji/obrocie dla wprowadzanych struktur.

2.5 automaton

W paczce tej zawarte są klasy odpowiadające za działanie automatu w sensie tworzenia kolejnych generacji. Zastosowany został tutaj wzorzec projektowy STAN, co pozwala na płynną zmianę zasad działania automatu.

2.5.1 CellularAutomaton

Klasa generująca kolejne generacje dla zadanych konfiguracji komórek (dostarczonych w klasie Cells).

- public CellularAutomaton() - Konstruktor który tworzy automat z pustą siatką 10x10.
- public CellularAutomaton(Cells cells) - Konstruuje z użyciem predefiniowanej siatki.
- public void generate() - Generuje kolejną generację.
- public void setRules(AutomatonStates state) - Ustanawia zasady działania automatu. Argumentem będzie np. new WireWorld().
- public Cells getCells() - Zwraca aktualną generację.

2.5.2 AutomatonRules

Interfejs z podstawowymi metodami, które są niezbędne dla funkcjonowania CellularAutomaton. Działa na zasadzie wzorca projektowego STAN, co pozwala na proste przełączanie się (dobieranie) zasad. Przykładowymi klasami implementującymi ten interfejs są WireWorld i Life.

- public AutomatonRules(AutomatonStates rules) - Konstruktor, któremu należy podać zasady.
- public AutomatonStates getAutomatonState() - Zwraca zasady.
- public int getCellStateByRules(Cells cells, int x, int y) - Funkcja zwracająca przewidywany stan komórki (x,y) w następnej generacji.
- public Color[] getColors() - Zwraca kolory (wykorzystywanie przez Visualisation do rysowania konkretnych barw dla konkretnych wartości).

- public boolean getIsWrapped() - Zwraca informację, czy zasady uwzględniają zawijaną siatkę.

2.5.3 Neighbourhood

Interfejs (opcjonalny) określający sposób obliczania ilości sąsiadów dla danej komórki. Wykorzystywany jest przez klasy WireWorld i Life (jednakże należy zwrócić uwagę na to, że mogą istnieć zasady, dla których sąsiedztwo nie jest potrzebne, dlatego interfejs ten jest opcjonalnie podpięty pod wyżej wspomniane klasy.) Przykładowymi klasami spełniającymi jego założenia są WrappedNeighbourhood i NormalNeighbourhood.

- public Neighbourhood(NeighbourhoodStates state) - Konstruktor otrzymujący informację o implementacji sąsiedztwa, argumentem będzie np. new NormalNeighbourhood().

- public int getNumberOfNeighbourhoodByState(Cells cells, int x, int y, int state) - Otrzymujemy liczbę sąsiadów o zadanej wartości wobec ustalonej komórki w siatce.

2.5.4 CellularAutomatonIO

Odpowiada za odczytywanie pliku tekstowego z konfiguracją komórek oraz jej zapis (konfiguracji do pliku) w prostej postaci.

- public static void OpenFromFile(File file) - odczytuje informację o siatce i zasadach a potem przekazuje je do GUI.

- public static void WriteToFile(File file, boolean isLife, boolean isWrapped, Cells cells) - Zapisuje informację o zasadach, stanie siatki i zawinięciu do pliku.

2.5.5 CellsIOException

- CellsIOException() - Konstruktor wyjątku dotyczącego błędnego formatu wczytywanego pliku pod względem stanów komórek.

- CellsIOException(String s) - Jak wyżej, tylko przekazuje wiadomość.

2.5.6 RulesIOException

- CellsIOException() - Konstruktor wyjątku dotyczącego błędnego formatu wczytywanego pliku pod względem wczytywania konfiguracji zasad.

- CellsIOException(String s) - Jak wyżej, tylko przekazuje wiadomość.

2.6 cells

Paczka ta zawiera podstawowe klasy działające na strukturze siatki komórkowej.

2.6.1 Cells

Klasa służąca jako kontener dla siatki komórkowej pozwalająca na wykonywanie na niej podstawowych operacji np. zmiana, dodanie, sprawdzenie stanu komórki.

- public Cells() - Tworzy pustą siatkę 10x10.
- public Cells(int x, int y) - Tworzy pustą siatkę x/y.
- public int getX() - Zwraca liczbę kolumn siatki.
- public int getY() - Zwraca liczbę wierszy siatki
- public void setCell(int x, int y, int v) - Ustawia wartość dla konkretnej komórki.
- public Cell getCell(int x, int y) - Zwraca komórkę.
- public String toString() - Zwraca tekstową reprezentację siatki.

2.6.2 CellsStructures

Klasa przekazująca metody pozwalające na ustawianie konkretnych struktur (np. bramki AND) na siatce komórkowej.

- public CellsStructures(Cells cellsx, boolean isWrapped) - Konstruktor w którym przekazujemy informację o siatce, na której działamy oraz, czy jest ona zawijana.
- private boolean setCellOfStructure(int x, int y, int var) - funkcja bliźniacza do setCell z cells, tylko z tą różnicą, że nie wyrzuca wyjątków lecz je obsługuje w celu wprowadzenia poprawnie struktury. Poniższe metody zwracają siatkę z wprowadzonymi strukturami takimi jak w nazwie. Pos określa kąt obrotu. 0 - 0, 1 - 90, 2 - 180, 3 - 270.
- public Cells setDiode(int x, int y, int pos) ustawia diodę.

- public Cells setClockGeneratorSmall(int x, int y, int pos) - ustawia mały generator elektronów.
- public Cells setClockGeneratorBig(int x, int y, int pos) - ustawia duży generator elektronów.
- public Cells setAND(int x, int y, int pos) - ustawia bramkę logiczną AND.
- public Cells setOR(int x, int y, int pos) - ustawia bramkę logiczną OR.

2.6.3 Cell

Klasa reprezentująca pojedynczą komórkę w siatce, dziedziczą po niej klasy: Alife, Dead, Empty, Conductor, Head oraz Tail.

- public Cell() - konstruktor (w klasach dziedziczonych przyjmuje ich nazwę) ustawiający wartość komórki.
- public int getValue() - zwraca liczbową reprezentację stanu komórki.

2.6.4 CellsOutOfBoundException

Klasa odpowiadająca za informację o wyjątkach dotyczących problemów z wychodzeniem poza zakres siatki komórkowej.

- public CellOutOfBoundException(int ver, int hor, String string) - Konstruktor wyjątku dotyczącego wyjścia poza ramy siatki przy np. użyciu setCells poza jej obszarem.
- public int getHor() - Zwraca informację o horyzontalnym wyjściu poza siatkę (np. -1 oznacza wyjście poza siatkę o jeden punkt w górę, dodatnie wartości informują o wyjściu poza dolną granicę).
- public int getVer() - Podobnie jak getHor(), tylko tyle, że zwraca informację o wertykalnym przekroczeniu granic. Ujemne liczby - poza lewą granicę, dodatnie - prawą.

3 Testy

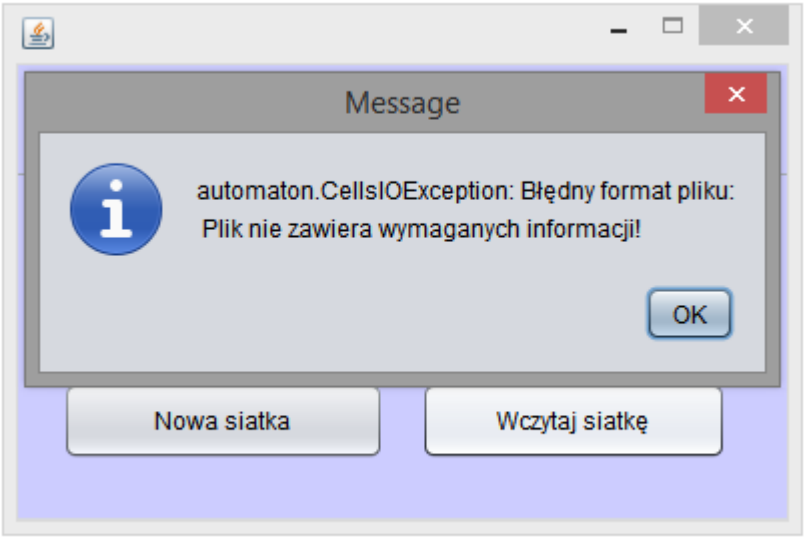
3.1 informacje

Wszystkie pliki wymagane do testów są zawarte w folderze testy w odpowiednich podfolderach. Aby przeprowadzić test wystarczy uruchomić aplikację i w launcherze wczytać odpowiedni plik z danymi.

3.2 Testy IO

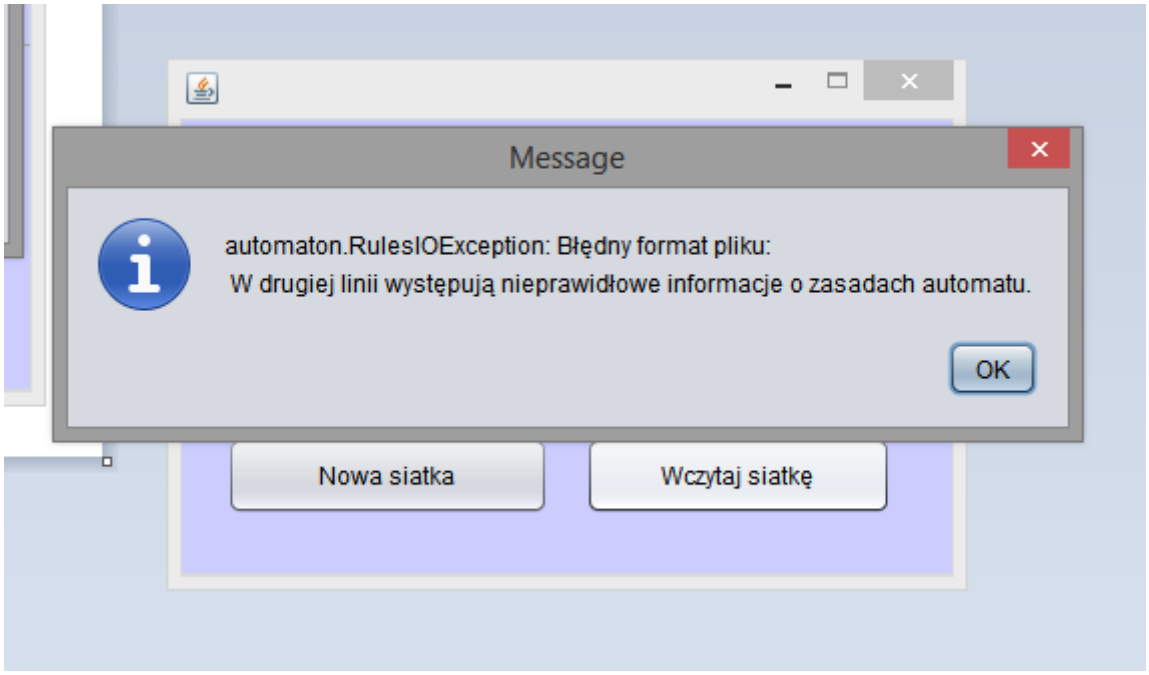
3.2.1 Nieodpowiednia ilość linii w pliku

- Plik: brak_linii.automat
- Wyniki:



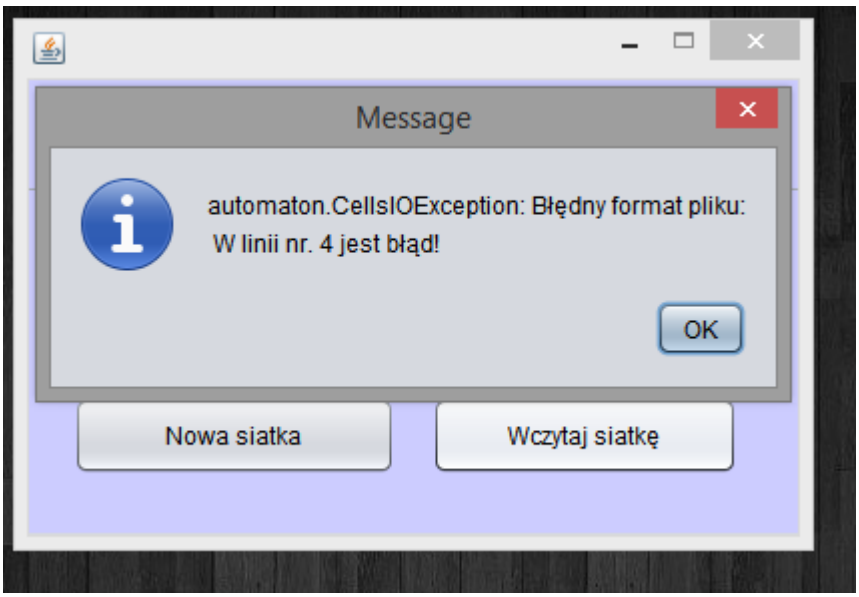
3.2.2 Literówki w pliku

- Plik: literówki.automat
- Wyniki:



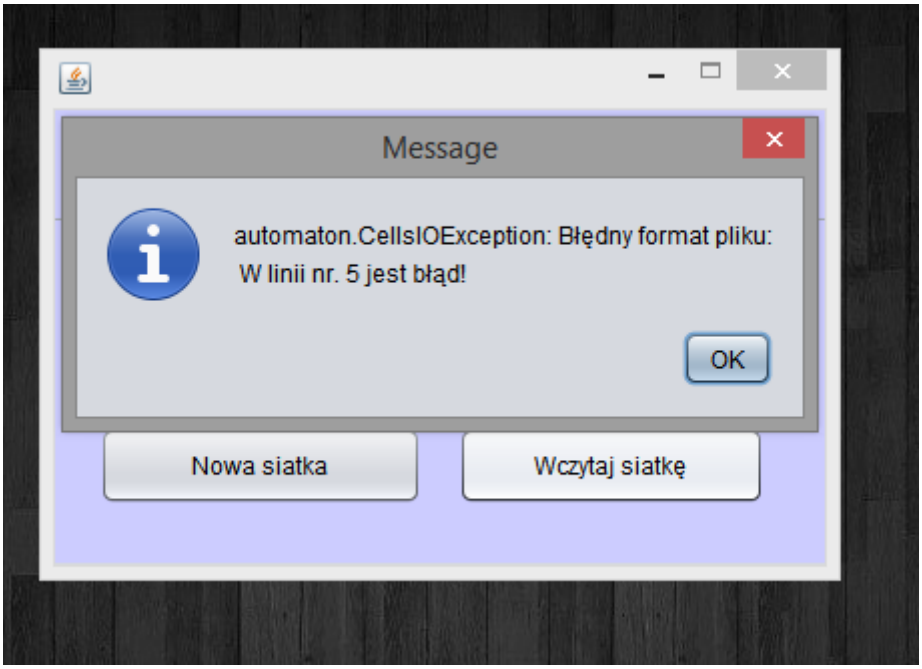
3.2.3 Niepoprawne dane stanów.

- Plik: niepełne_dane_stanów.automat
- Wyniki:



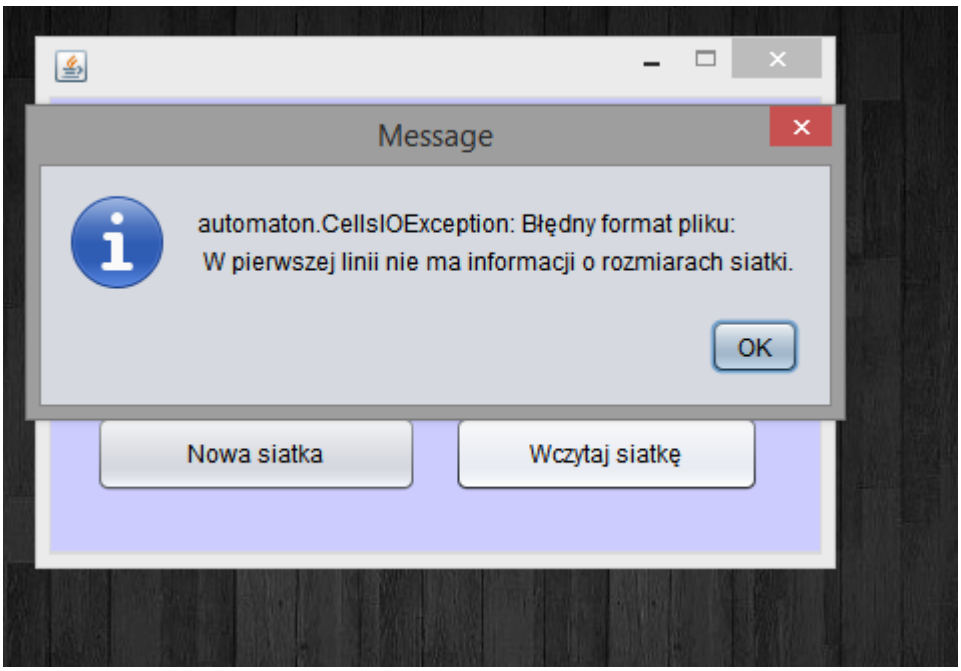
3.2.4 Niepoprawne dane struktur.

- Plik: niepełne_dane_struktur.automat
- Wyniki:



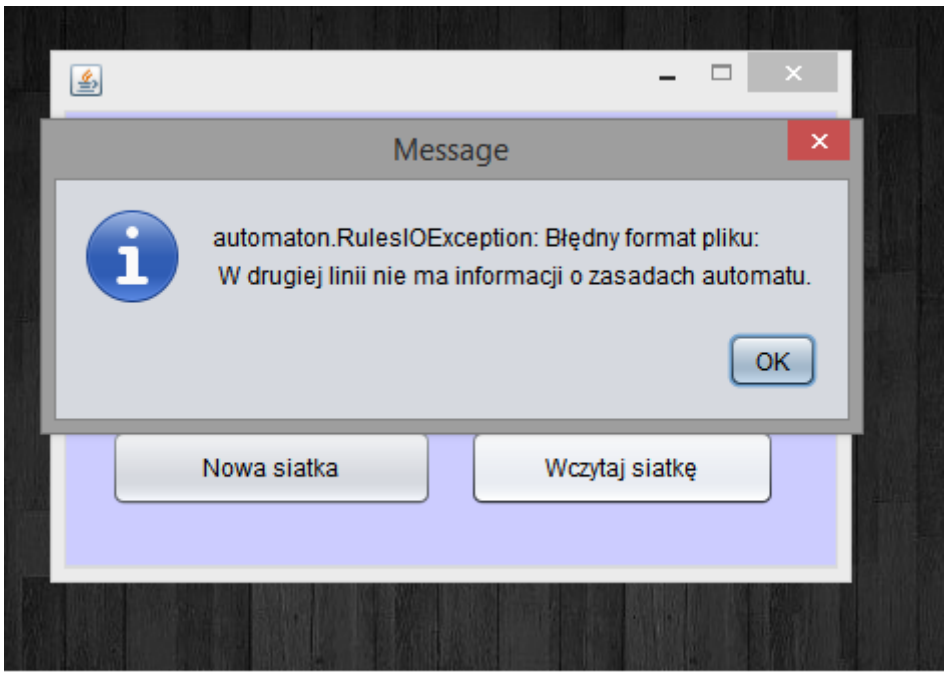
3.2.5 Niepoprawne dane wymiarów.

- Plik: niepełne_dane_wymiarów.automat
- Wyniki:



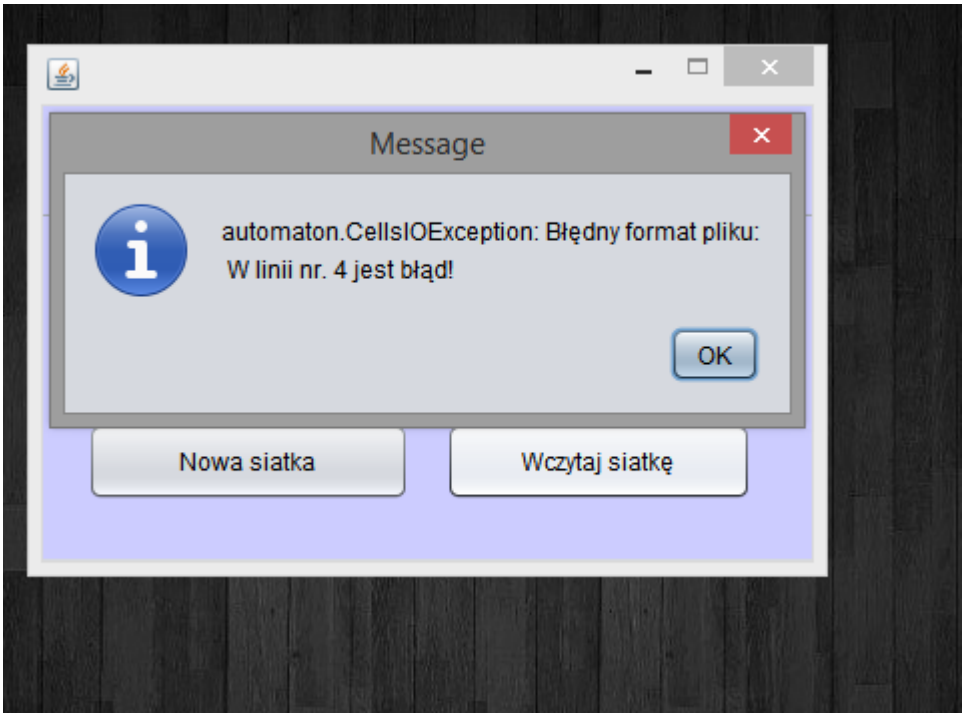
3.2.6 Niepoprawne dane zasad

- Plik: niepełne_dane_zasad.automat
- Wyniki:



3.2.7 Próba wyjścia poza granicę siatki.

- Plik: wychodzenie_poza_granice.automat
- Wyniki:

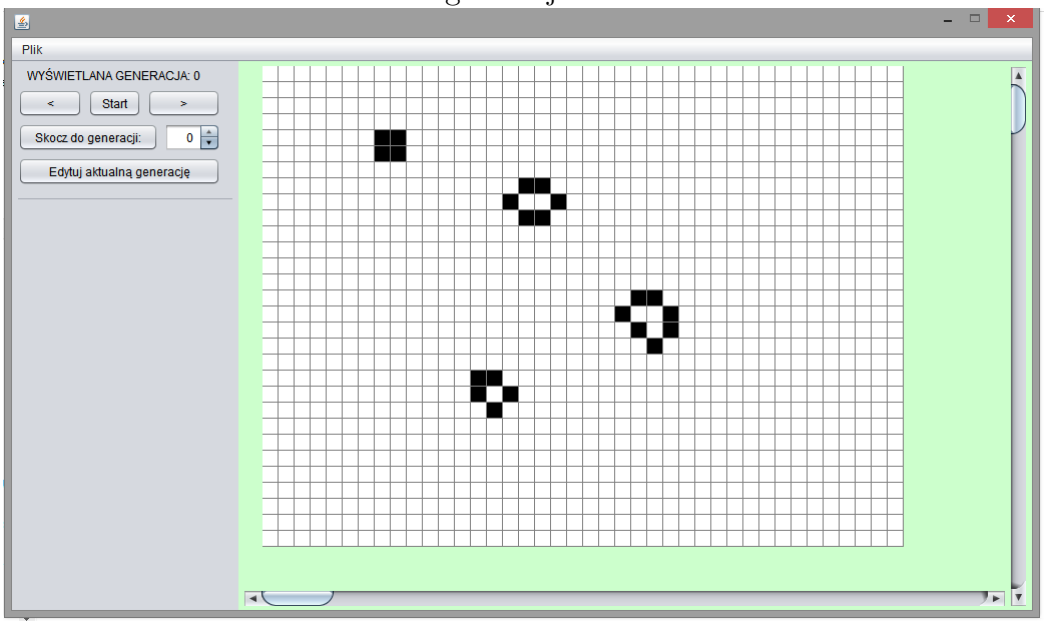


3.3 Testy Life

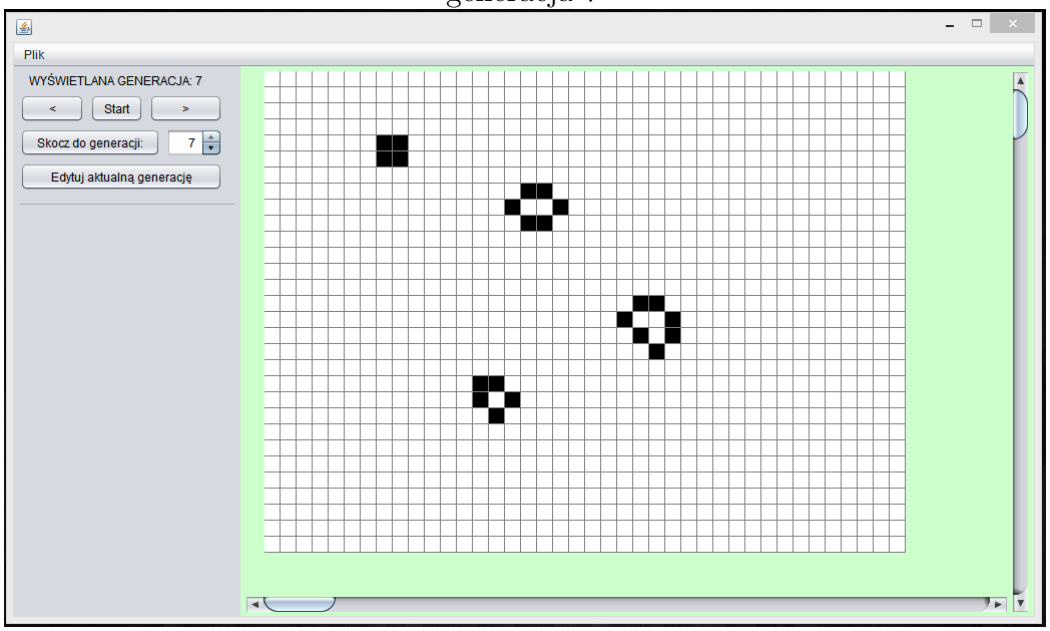
3.3.1 Struktury stałe - niezmienne.

- Plik: niezmiennicze.automat
- Wyniki:

generacja 0



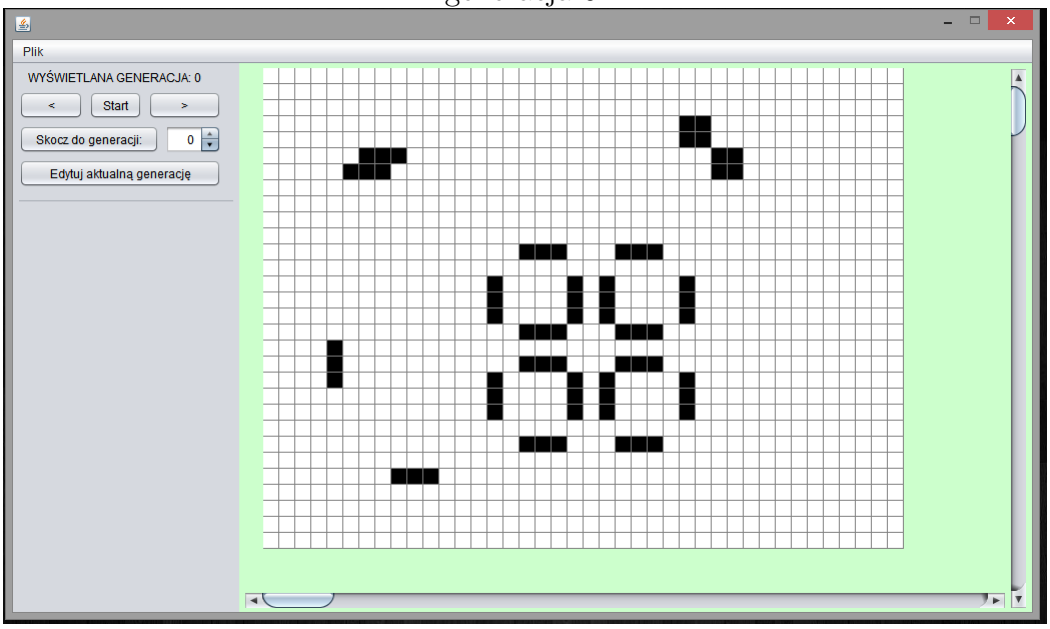
generacja 7



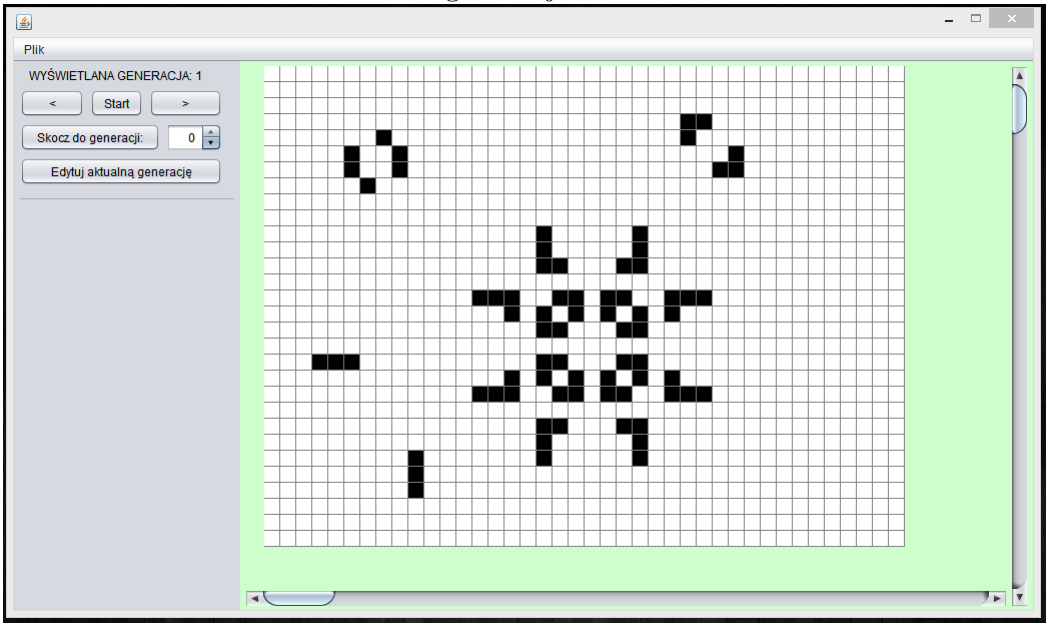
3.3.2 Oscylatory

- Plik: oscylatory.automat
- Wyniki:

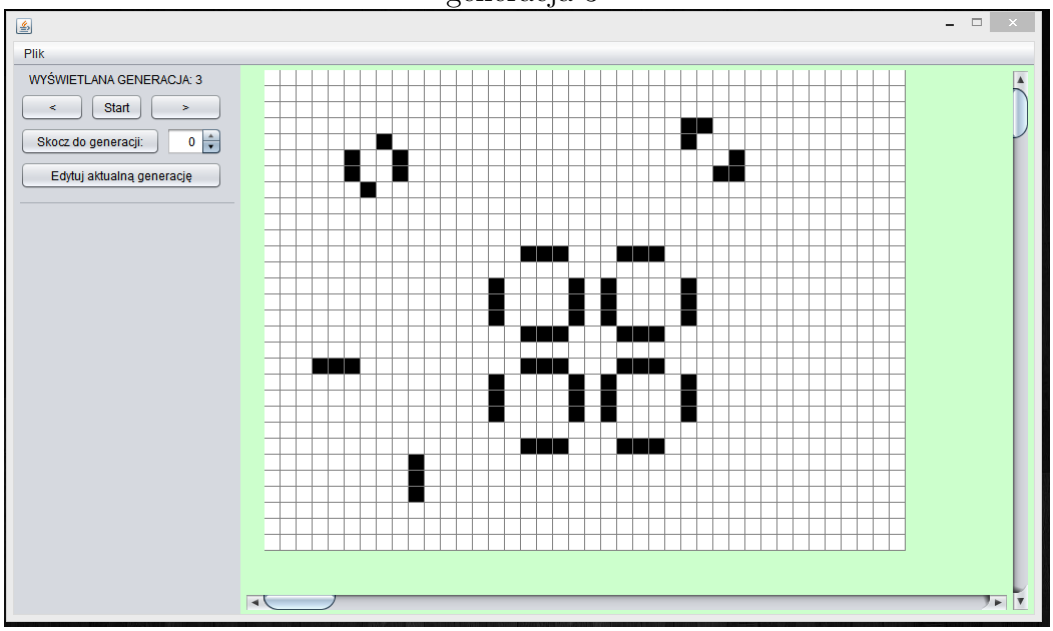
generacja 0



generacja 1



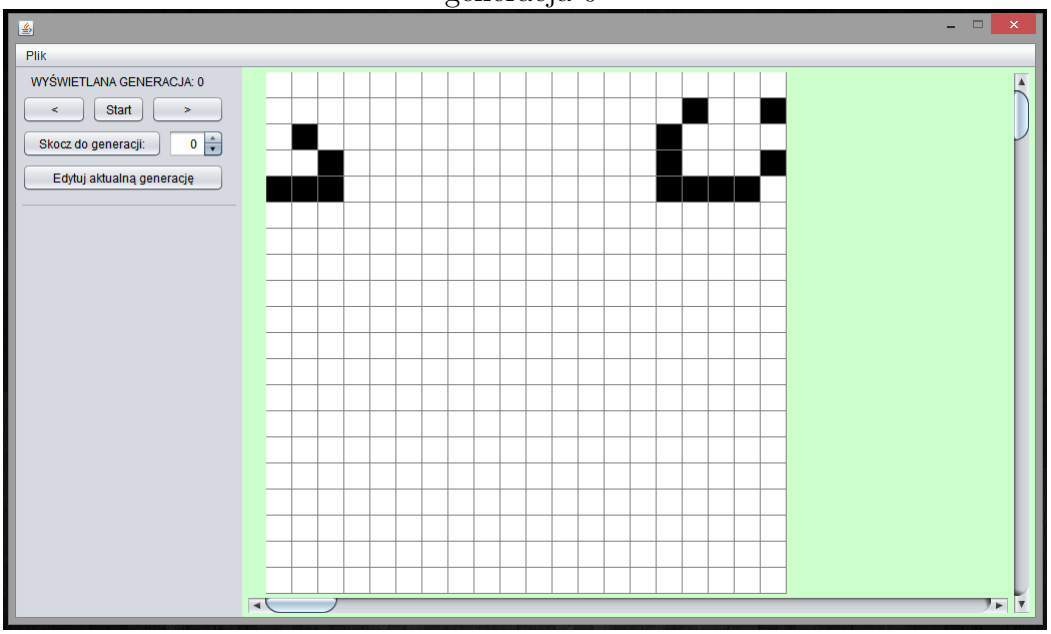
generacja 3


Plik
WYŚWIETLANA GENERACJA: 3
<
Start
>
Skocz do generacji:
0
Edytuj aktualną generację

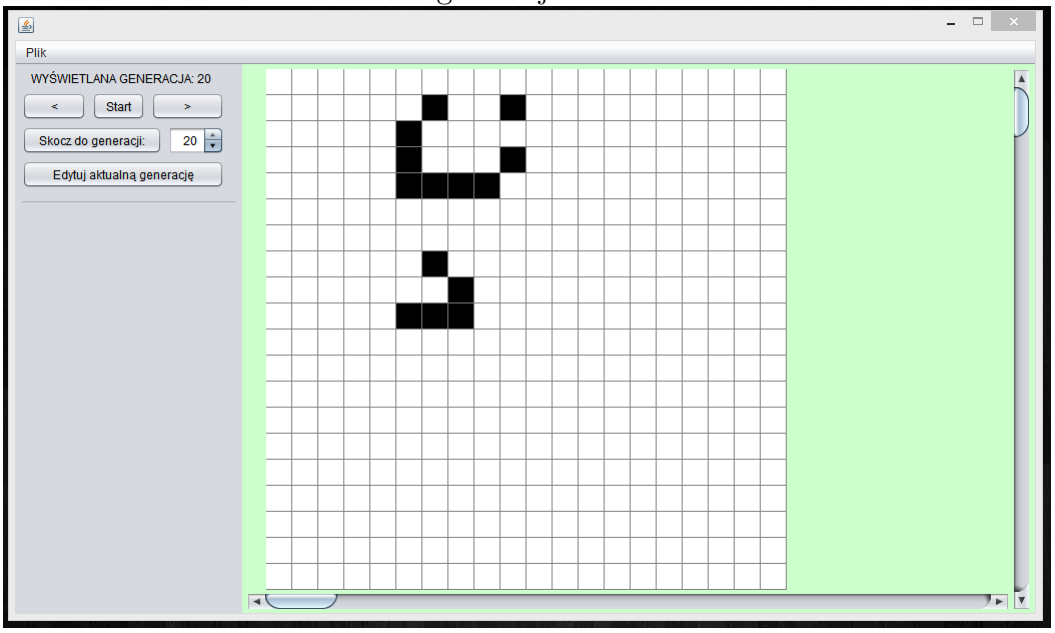
3.3.3 Statki przy normalnych granicach

- Plik: statki_granice_normalne.automat
- Wyniki:

generacja 0



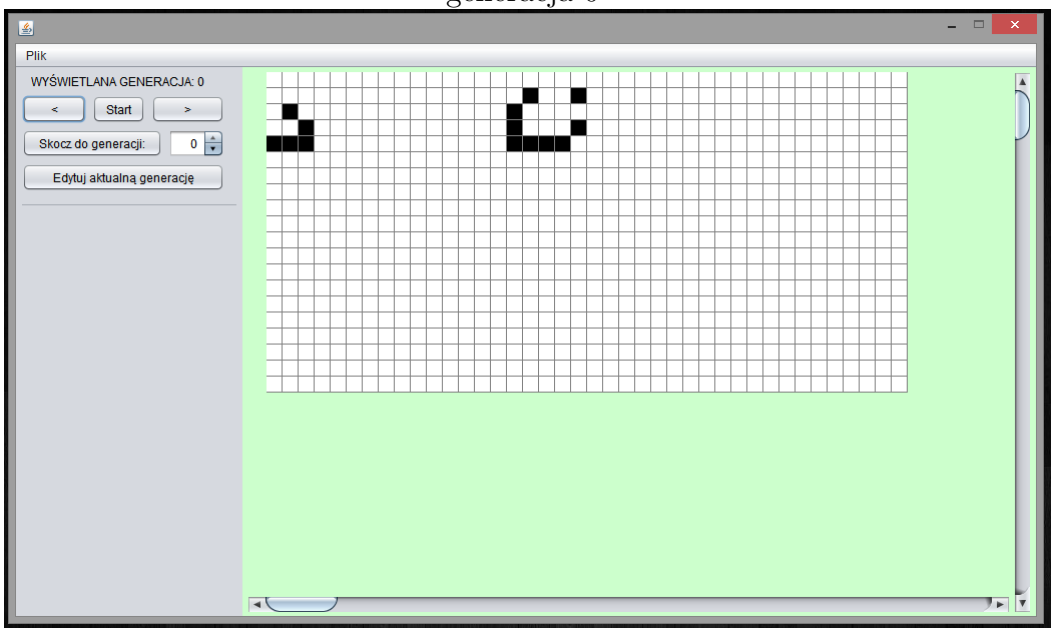
generacja 20



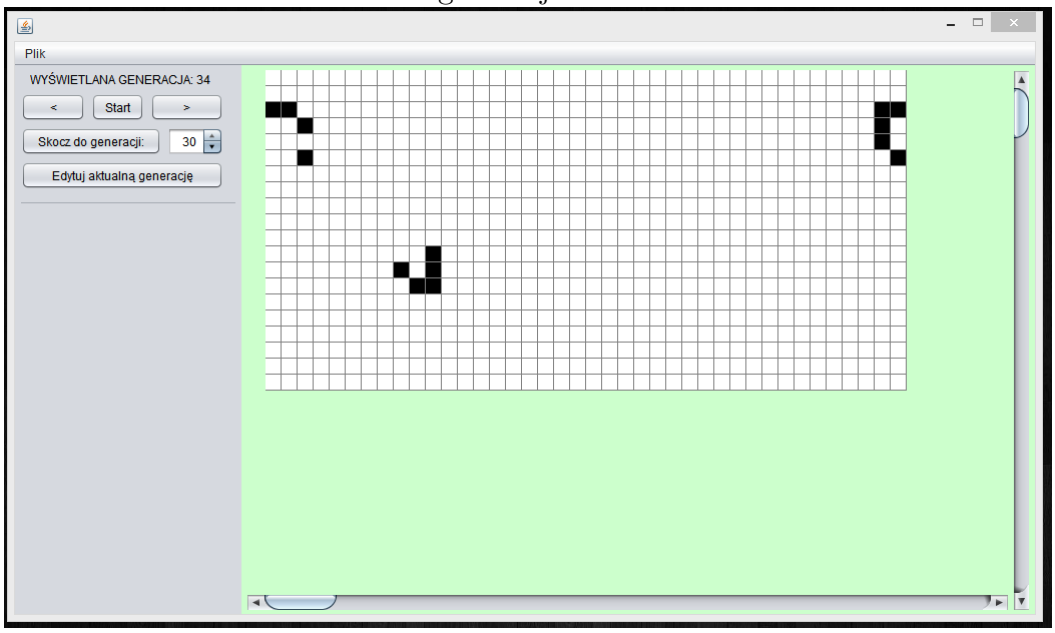
3.3.4 Statki przy zawijanych granicach

- Plik: statki_granice_zawijane.automat
- Wyniki:

generacja 0



generacja 34

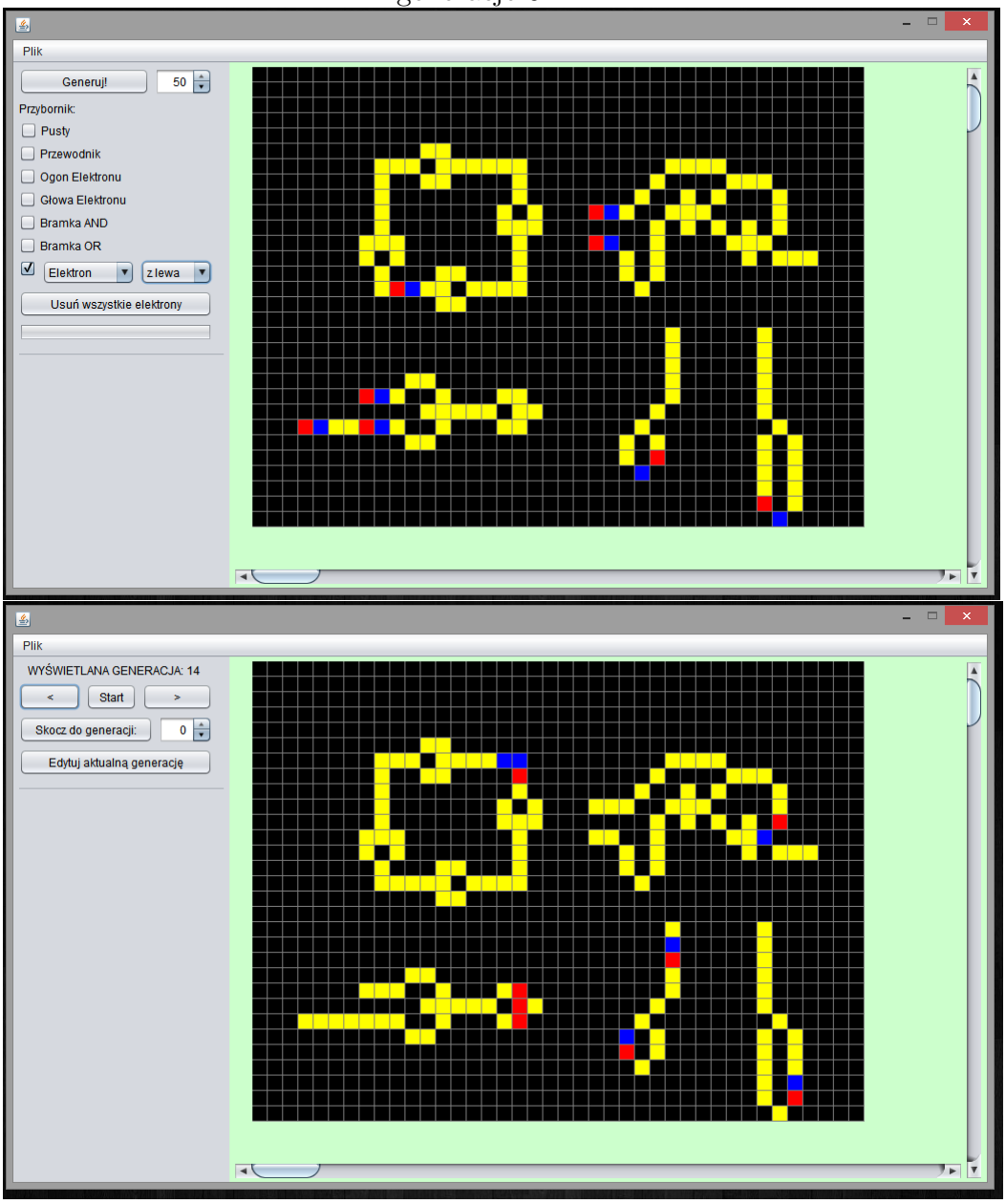


3.4 Testy Wireworld

3.4.1 Wszystkie struktury - oddzielone od siebie

- Plik: struktury_oddzielnie.automat
- Wyniki:

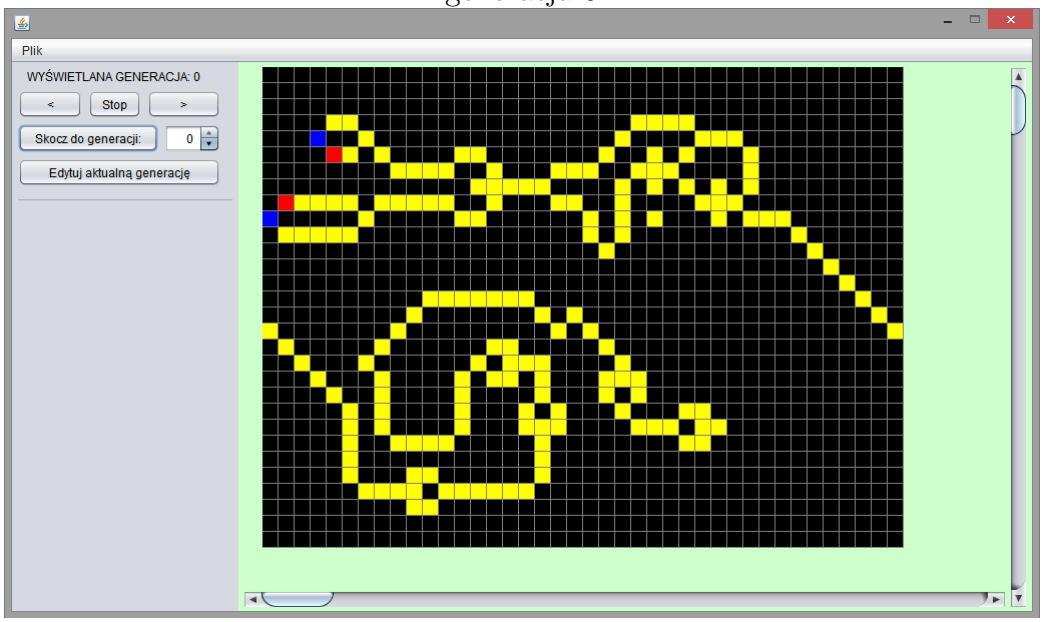
generacje 0 i 14



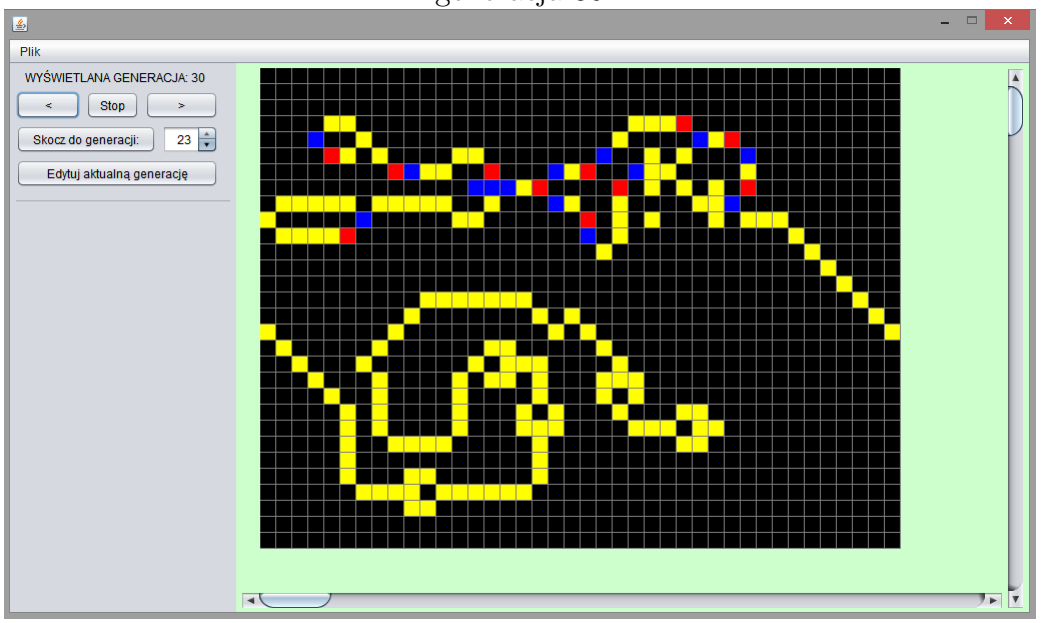
3.4.2 Połączone struktury przy zawijanych granicach

- Plik: wszystkie_struktury_zawijane_granice.automat
- Wyniki:

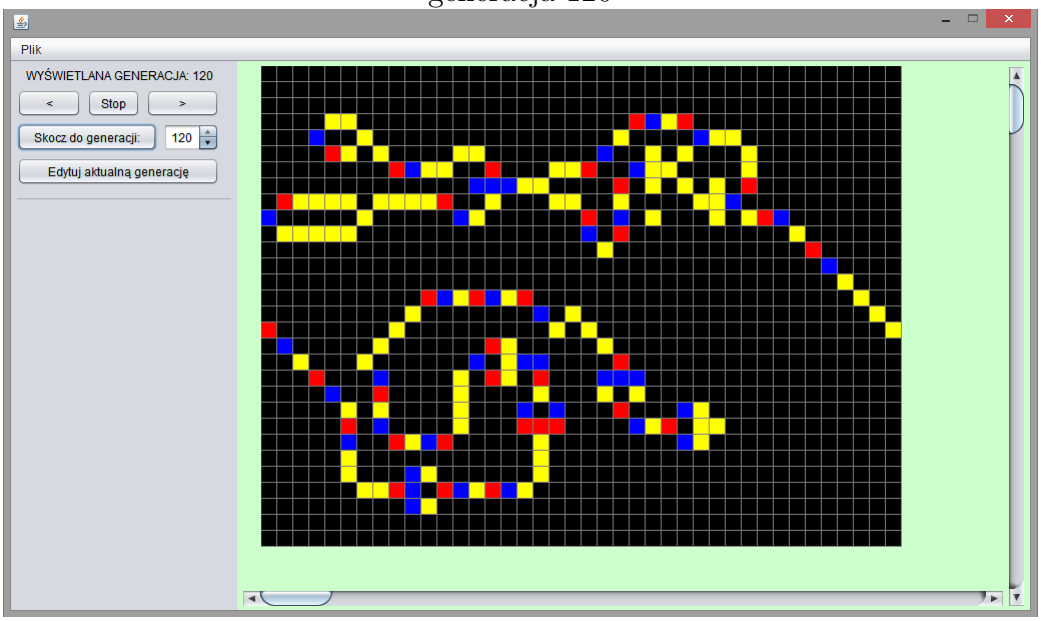
generacja 0



generacja 30



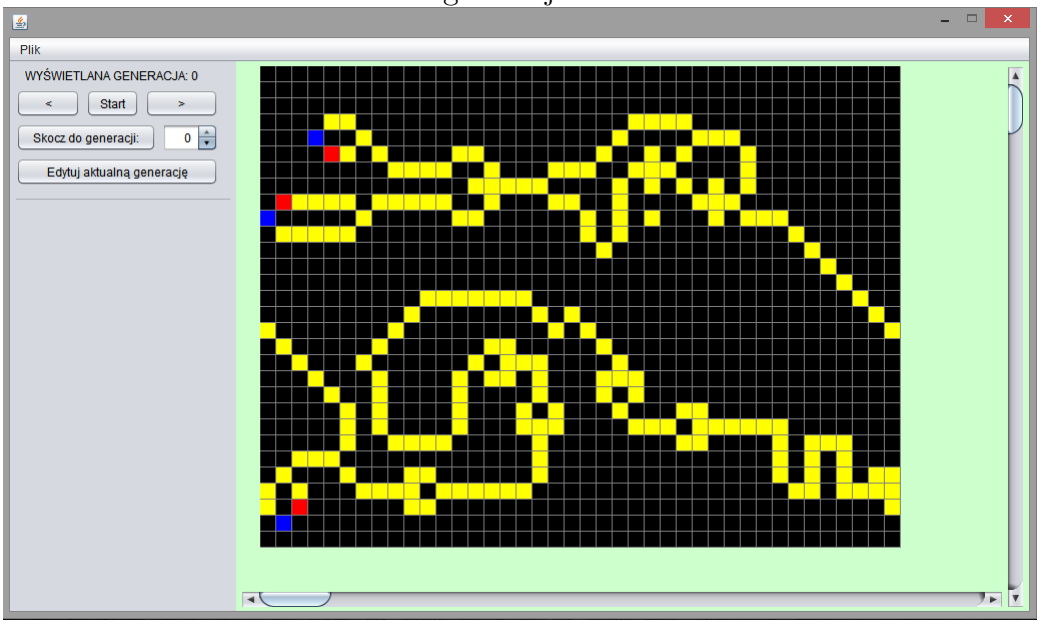
generacja 120



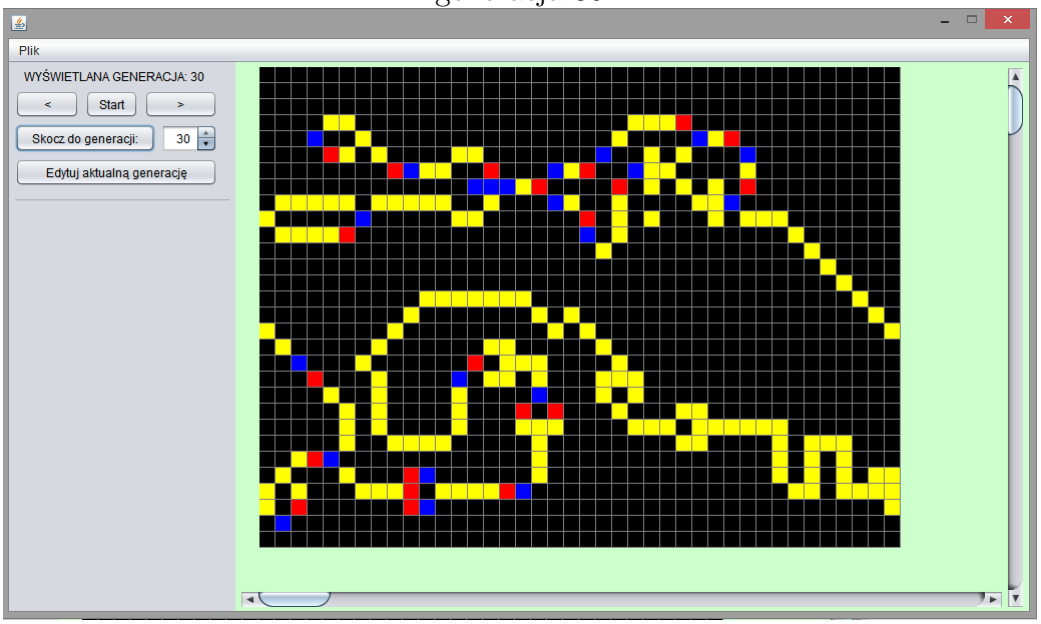
3.4.3 Połączone struktury przy normalnych granicach

- Plik: wszystkie_struktury_normalne_granice.automat
- Wyniki:

generacja 0



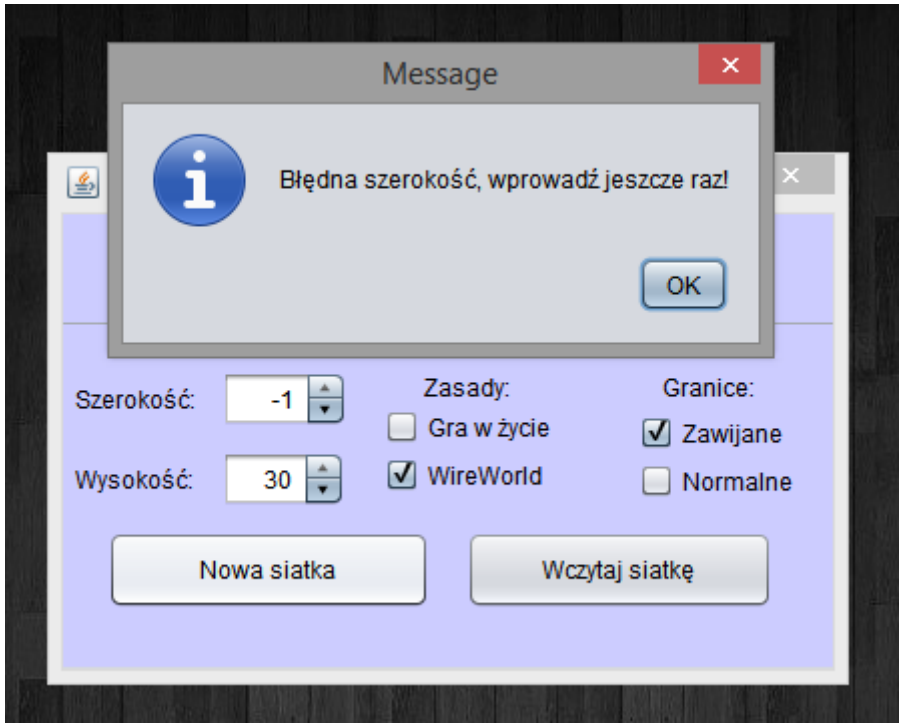
generacja 30

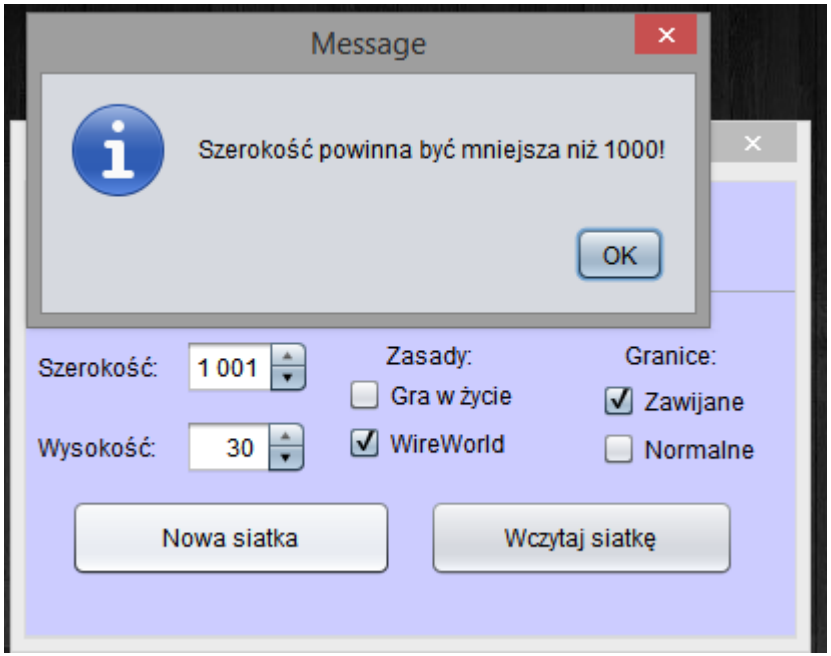


3.5 Testy Przeciążania

3.5.1 Rozmiary siatki

- Informacja: Test miał na celu sprawdzenie zachowania aplikacji, gdy użytkownik podaje nielogiczne wymiary siatki.
- Wyniki:

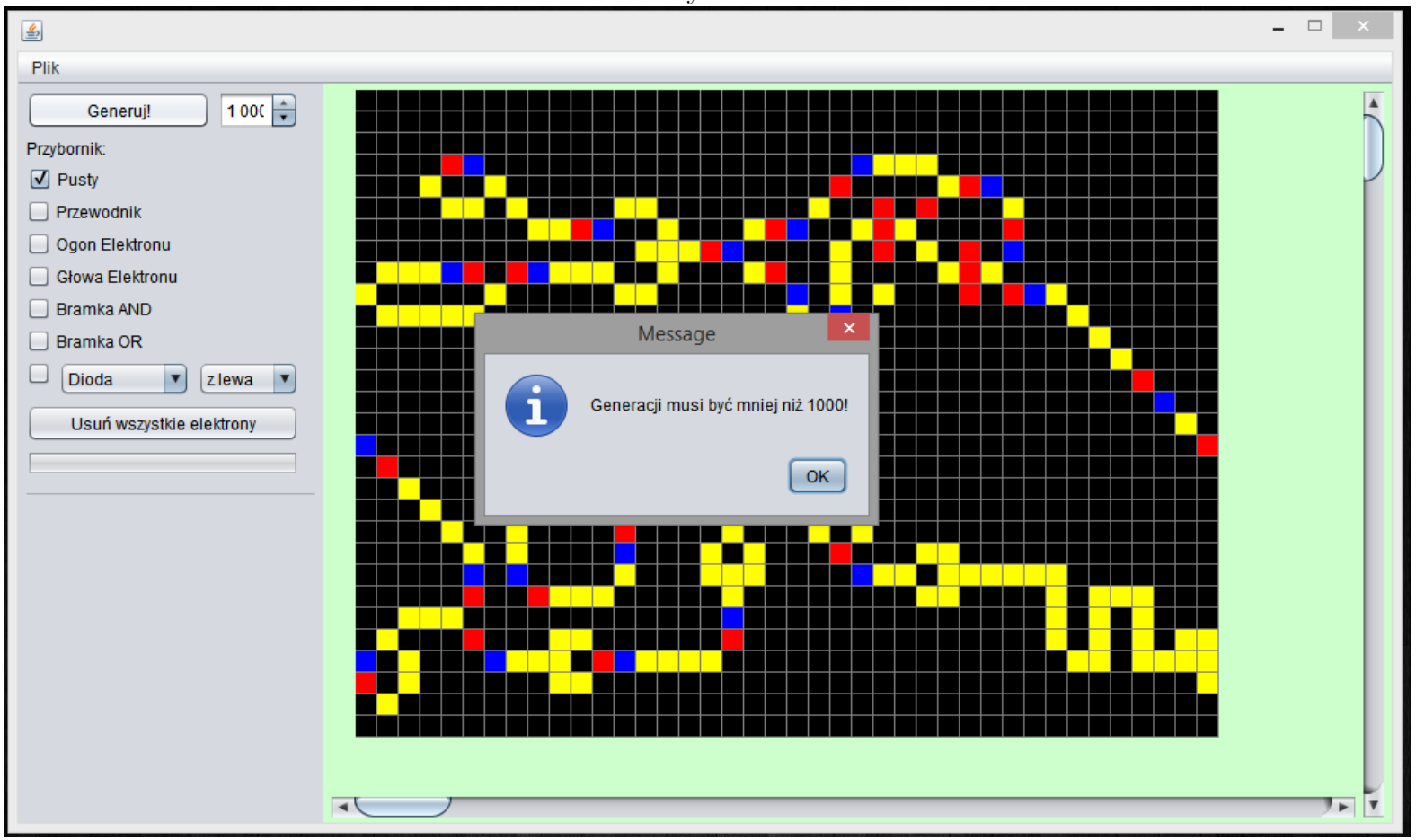


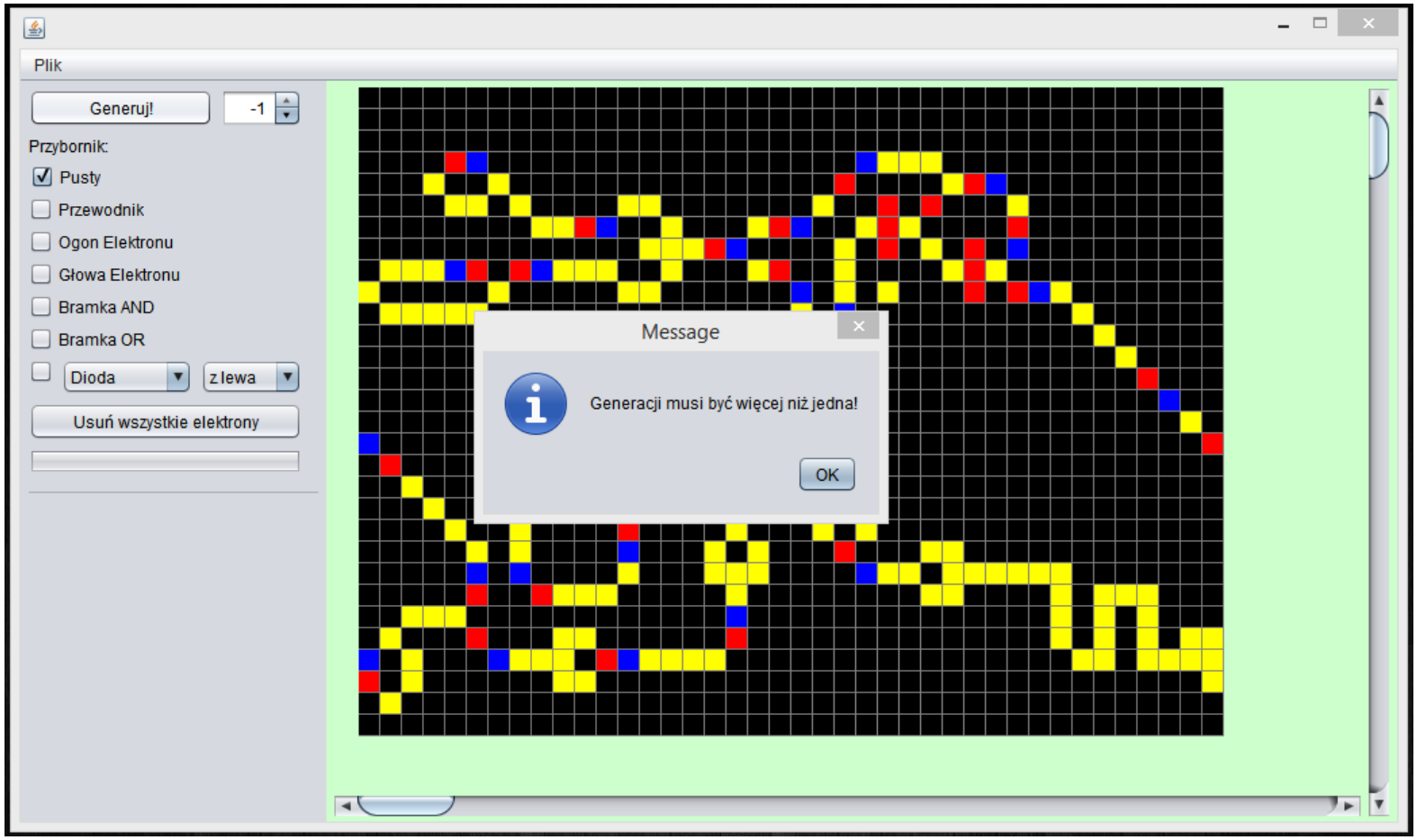


3.5.2 Ilość generacji

- Informacja: Test miał na celu sprawdzenie zachowania aplikacji, gdy użytkownik podaje nielogiczną liczbę generacji do przeprowadzenia.

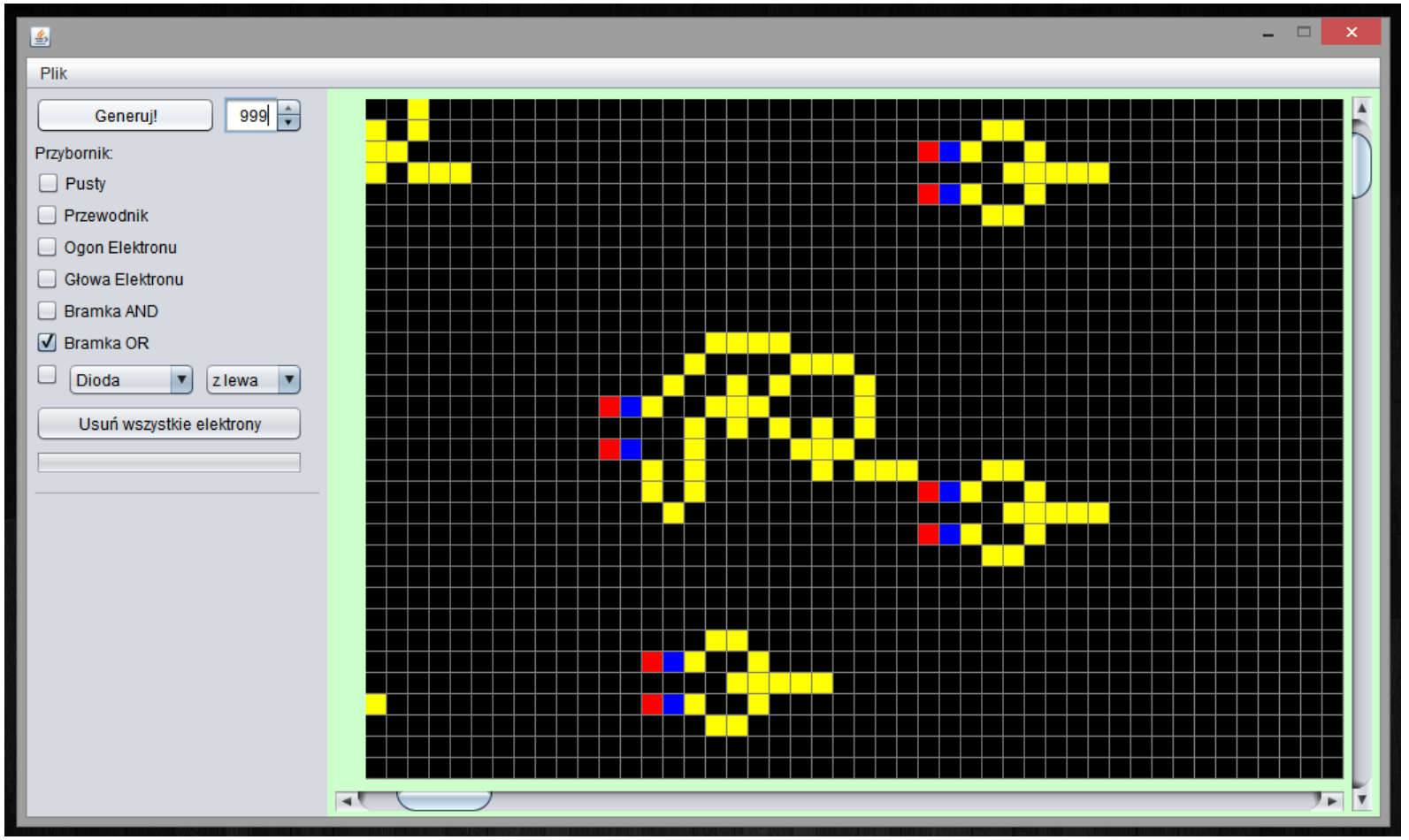
Wyniki:

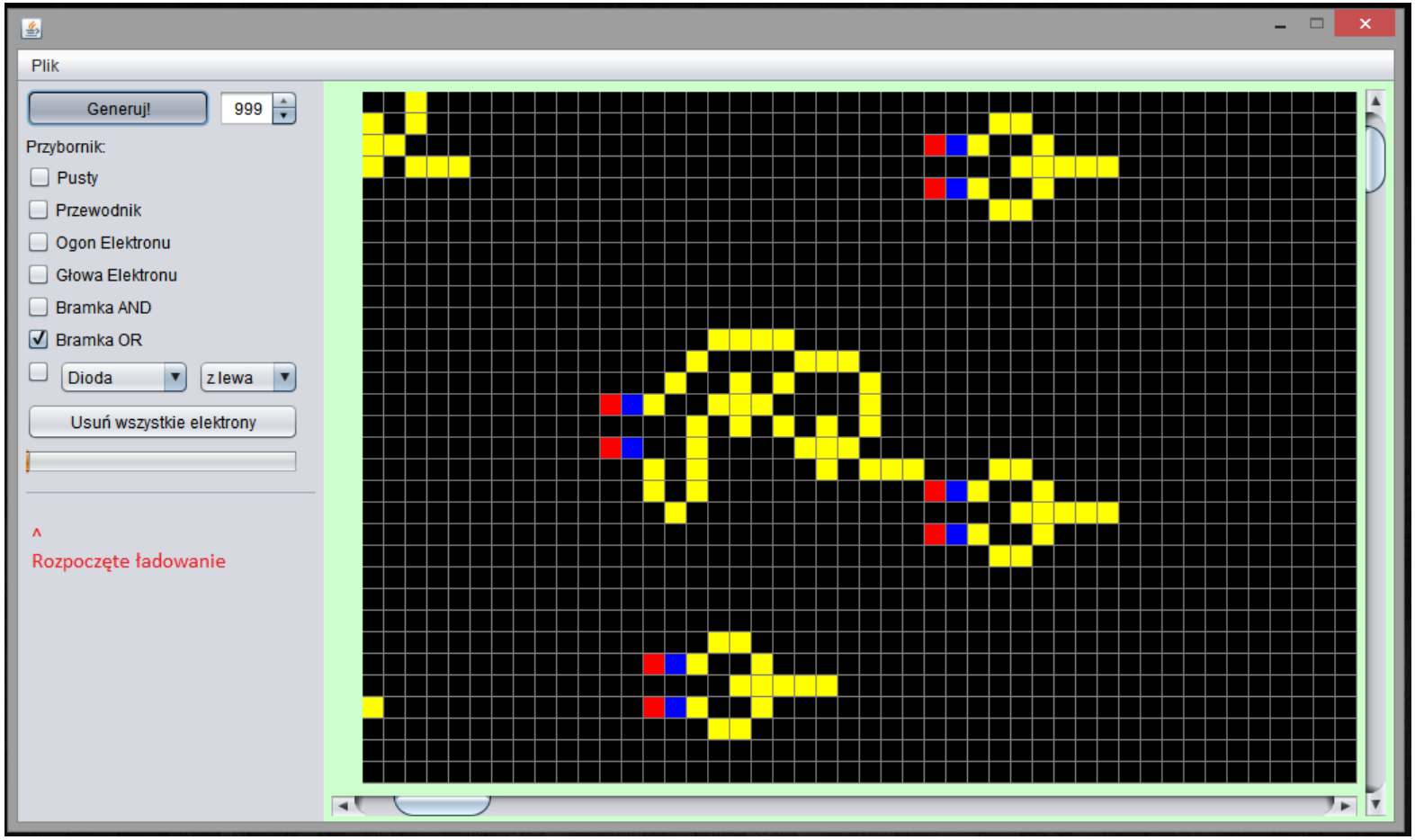

Plik
Generuj!
1 000
Przybornik:
Pusty
Przewodnik
Ogon Elektronu
Głowa Elektronu
Bramka AND
Bramka OR
Dioda
z lewa
Usuń wszystkie elektrony
Message
Generacji musi być mniej niż 1000!
OK

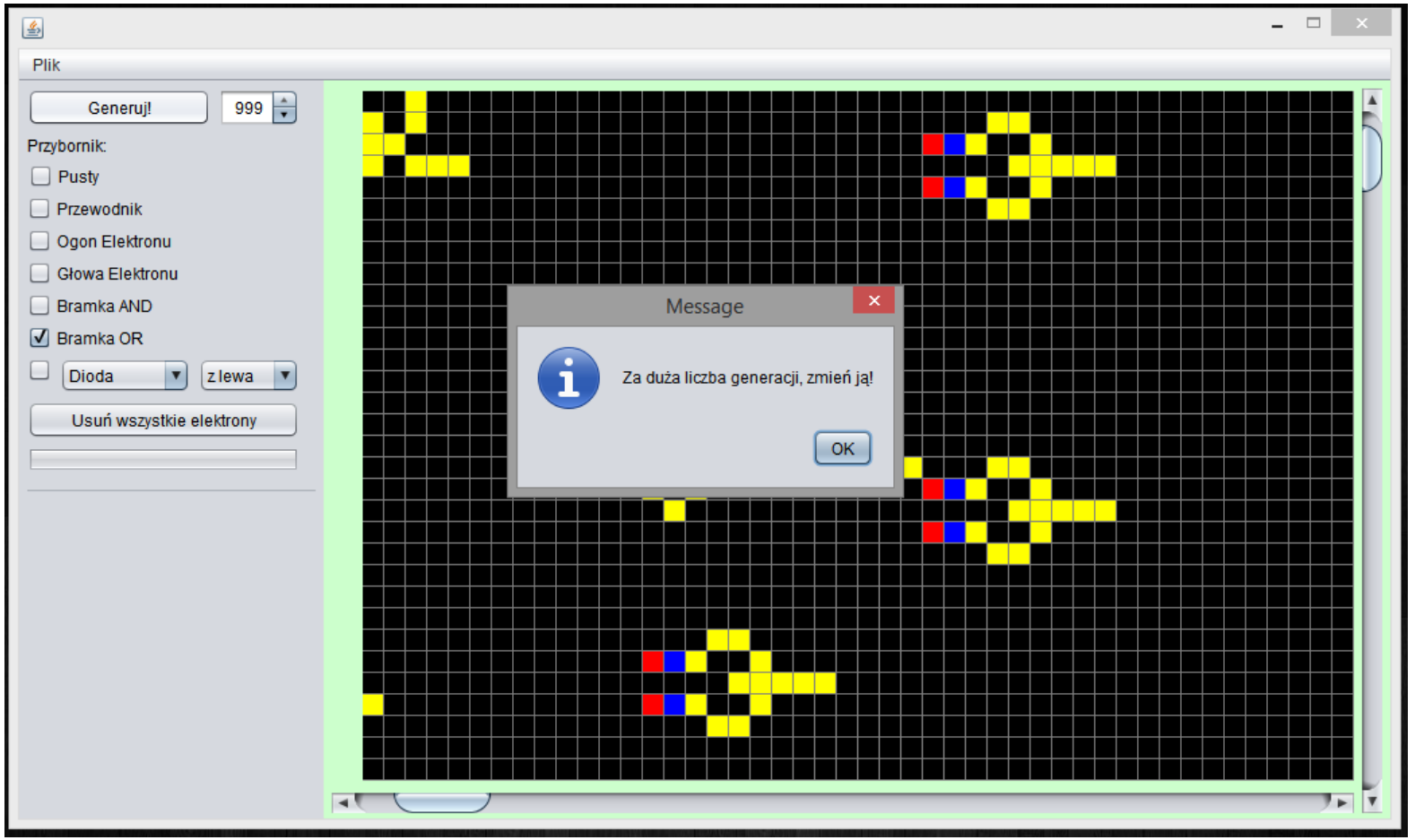

Plik
Generuj!
-1
Przybornik:
Pusty
Przewodnik
Ogon Elektronu
Głowa Elektronu
Bramka AND
Bramka OR
Dioda
z lewa
Usuń wszystkie elektrony
Message
Generacji musi być więcej niż jedna!
OK

3.5.3 Ilość generacji - zasoby sprzętowe

- Informacja: Test mający na celu sprawdzenie "wytrzymałości" programu. Najbardziej brutalnym sprzętowo scenariuszem jest stworzenie siatki o rozmiarach 999x999 i wygenerowanie 999 generacji.
- Plik: wytrzymałość.automat
- Podsumowanie: Jak widać na załączonych niżej zdjęciach, program próbuje wygenerować określoną liczbę generacji (widać to po pasku stanu - ładowania), jednak gdy dochodzi do momenty, gdy brakuje pamięci podręcznej program wysyła odpowiedni komunikat i zwalnia pamięć, prosząc użytkownika o podanie innej ilości do wygenerowania.


Plik
Generuj!
999
Przybornik:
Pusty
Przewodnik
Ogon Elektronu
Głowa Elektronu
Bramka AND
Bramka OR
Dioda
z lewa
Usuń wszystkie elektrony

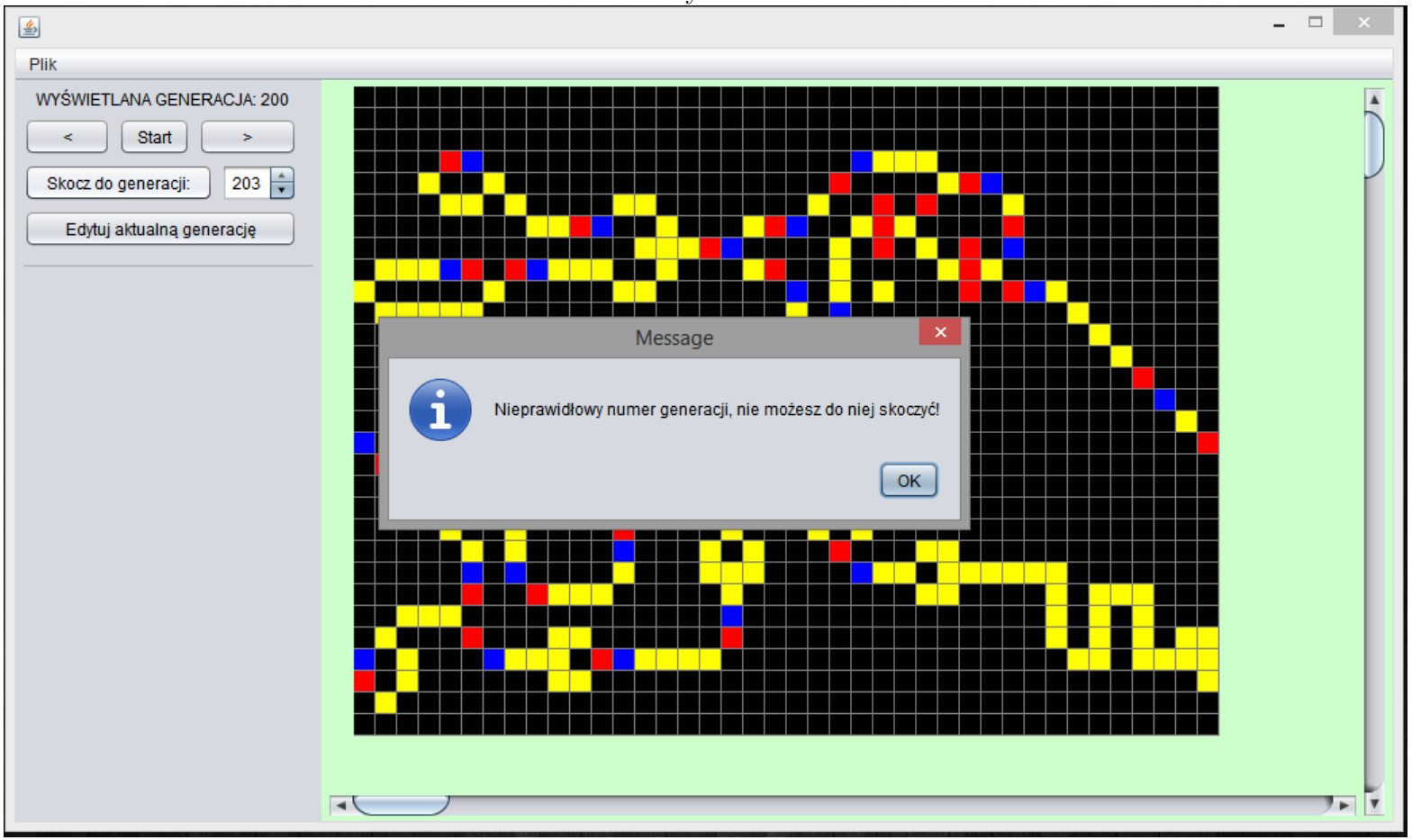

Plik
Generuj!
999
Przybornik:
Pusty
Przewodnik
Ogon Elektronu
Głowa Elektronu
Bramka AND
Bramka OR
Dioda
z lewa
Usuń wszystkie elektrony
^
Rozpoczęte ładowanie

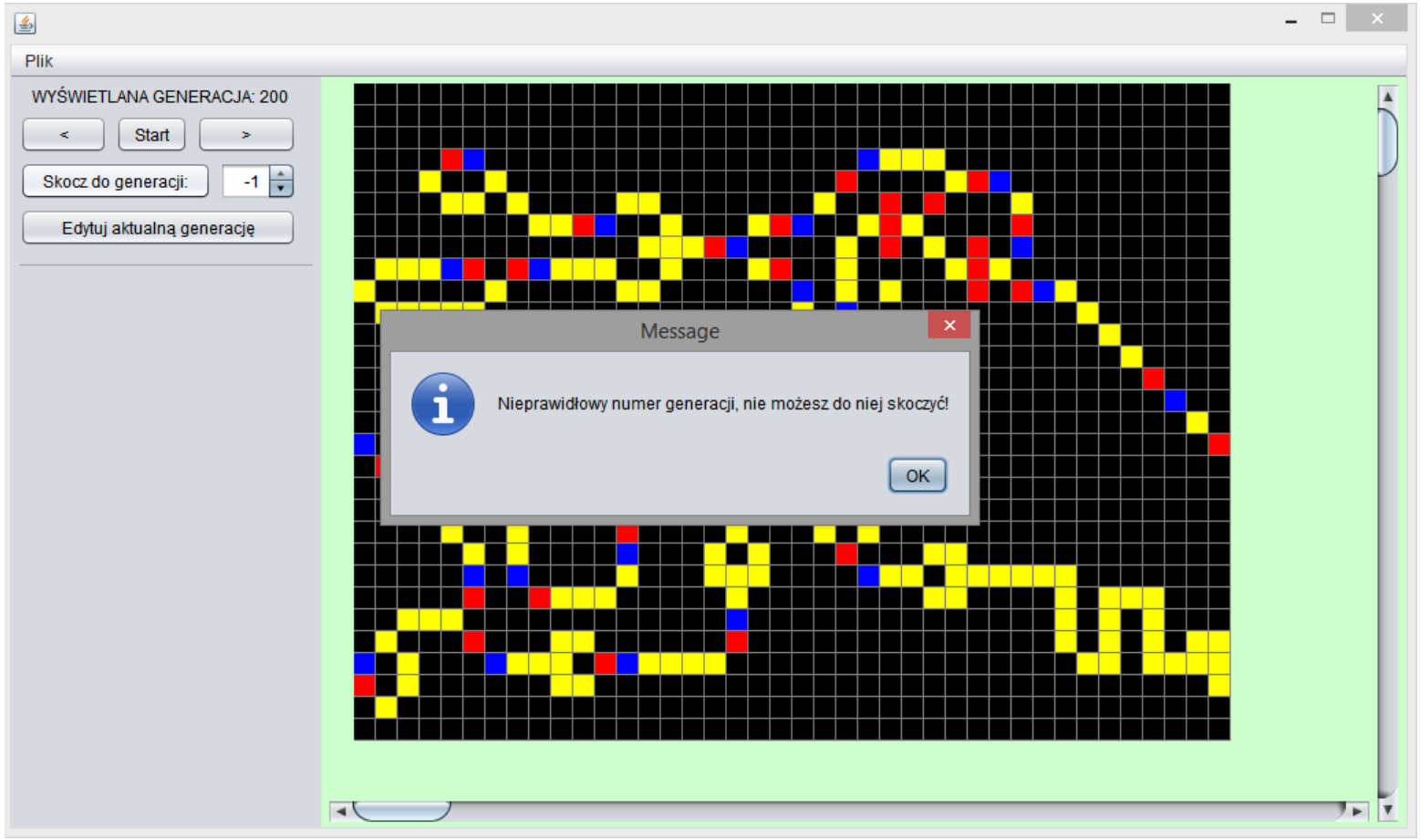

Plik
Generuj!
999
Przybornik:
Pusty
Przewodnik
Ogon Elektronu
Głowa Elektronu
Bramka AND
Bramka OR
Dioda
z lewa
Usuń wszystkie elektrony
Message
Za duża liczba generacji, zmień ją!
OK

3.5.4 Skakanie do określonej generacji

- Informacja: Test miał na celu sprawdzenie zachowania aplikacji, gdy użytkownik podaje nielogiczną generację, do której chciałby przeskoczyć.

Wyniki:


Plik
WYŚWIETLANA GENERACJA: 200
<
Start
>
Skocz do generacji:
203
Edytuj aktualną generację
Message
Nieprawidłowy numer generacji, nie możesz do niej skoczyć!
OK


Plik
WYŚWIETLANA GENERACJA: 200
<
Start
>
Skocz do generacji:
-1
Edytuj aktualną generację
Message
Nieprawidłowy numer generacji, nie możesz do niej skoczyć!
OK

4 Podsumowanie

Modyfikowanie programu dla innego automatu nie będzie już jednak takie proste jak w poprzednim projekcie, wymagane będzie tutaj napisanie własnego IO, dodanie dziedziczonych komórek (Cell) oraz dodanie kolejnego Stanu zasad (Rules). Niestety GUI nie jest przystosowane do łatwego wprowadzania modyfikacji, dlatego polecam stworzenie nowego na potrzeby innych automatów.

Prosty przykład nowego automatu zawiera się w dodatkowej paczce easyautomatonexample, do samego automatu można dostać się poprzez Launcher, wpisując "-404" w polu szerokości i klikając następnie w przycisk "Nowa Siatka".

Duże problemy w projekcie sprawiła mi szczególnie nieznajomość Swinga oraz parę nieprzemyślanych do końca decyzji projektowych. Podchodząc do tematu ponownie, program napisałbym zapewne w całkiem inny sposób, dzięki nabytemu doświadczeniu.