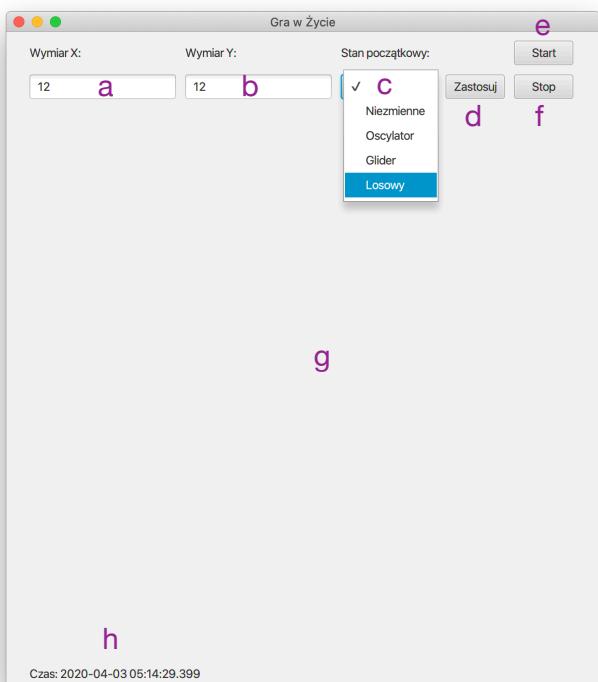


# Sprawozdanie

## 1. Wykonanie - interfejs aplikacji

Ćwiczenie wykonano w Javie, do stworzenia GUI użyto JavaFX.

Interfejs aplikacji, przedstawiony na rys. 1., zawiera następujące elementy:



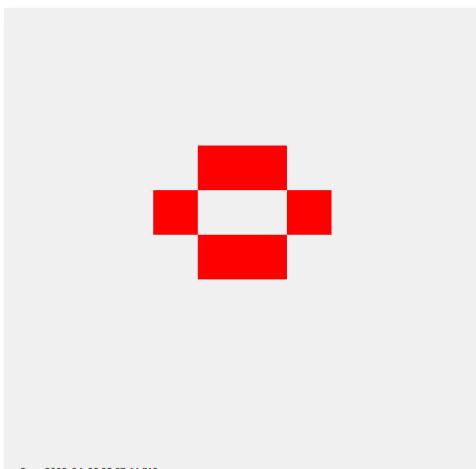
Rys. 1. Interfejs

- Pole tekstowe, umożliwiające użytkownikowi wybór szerokości siatki;
- Pole tekstowe do wyboru wysokości siatki;
- Pole wyboru z listą dostępnych stanów początkowych (pusta siatka, niezmienny, oscylator, glider, losowy);
- Przycisk „Zastosuj”, wywołujący utworzenie siatki o podanych wymiarach o wybranym stanie początkowym;
- Przycisk „Start”, uruchamiający 100 kroków gry w życie (stan początkowy - aktualny stan siatki);
- Przycisk „Stop”, zatrzymujący symulację;
- Płótno, na którym wyświetlane są wyniki w formie animacji; komórki żywe reprezentują kwadraty koloru czerwonego. Kliknięcie w komórkę powoduje zmianę jej stanu. Rozmiar komórek dostosowany jest do rozmiaru siatki, największy możliwy rozmiar to 600x600.
- Etykieta „Czas”, wyświetlająca czas wydrukowania aktualnego obrazu na płótnie (reprezentuje kolejność wyników).

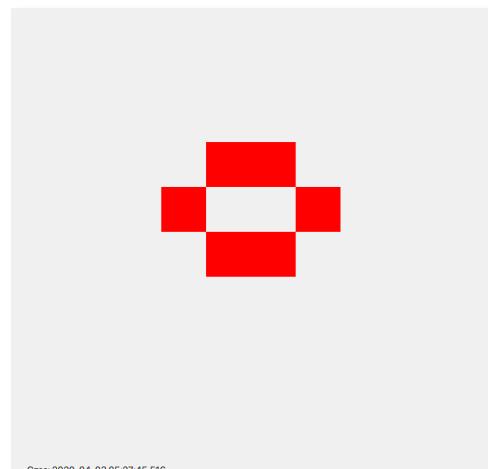
## 2. Wyniki

Wyniki działania aplikacji przedstawiają wydruki na rysunkach 2.- 6. Użyto parametrów:

- Stan początkowy: niezmienny, rozmiar siatki: 10x10, rys. 2. a) i b) przedstawiają kolejne kroki czasowe:

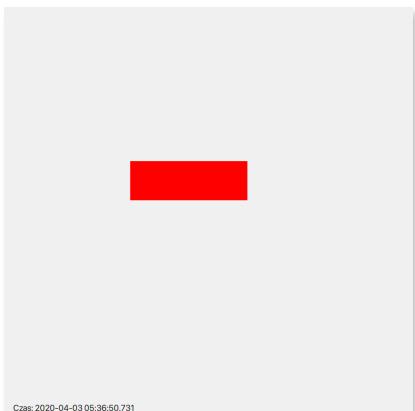


Rys. 2. a) Stan początkowy - niezmienny, 1. krok

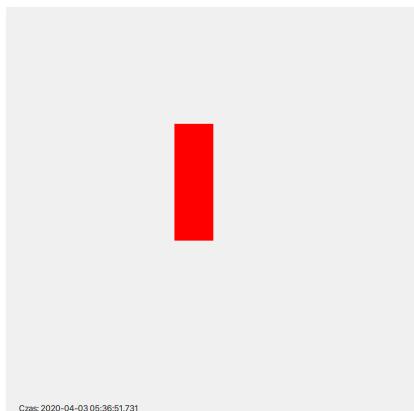


Rys. 2. b) Stan początkowy - niezmienny, 2. krok

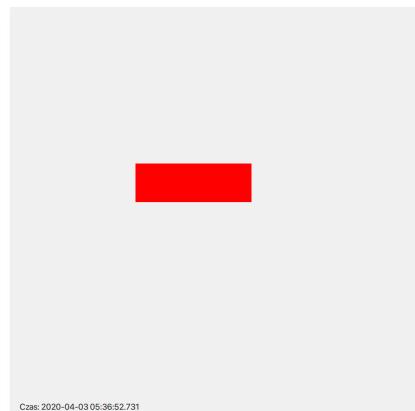
- b) Stan początkowy: oscylator, rozmiar siatki: 10x10, rys. 3. a) - c) przedstawiają kolejne kroki czasowe:



Czas: 2020-04-03 05:36:50.731



Czas: 2020-04-03 05:36:51.731



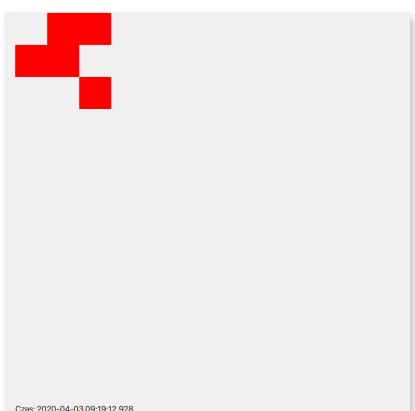
Czas: 2020-04-03 05:36:52.731

Rys. 3. a) Stan początkowy - oscylator, 1. krok

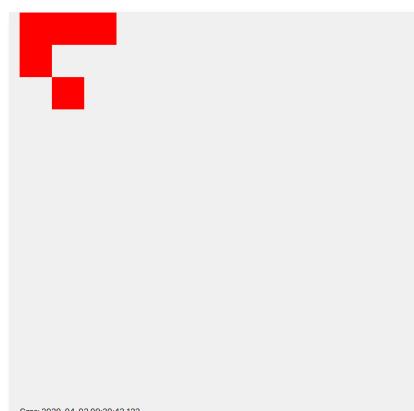
Rys. 3. b) Stan początkowy - oscylator, 2. krok

Rys. 3. c) Stan początkowy - oscylator, 3. krok

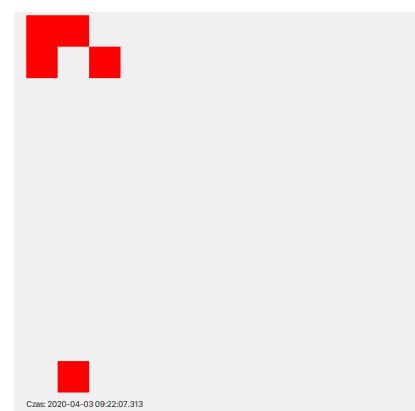
- c) Stan początkowy: glider, rozmiar siatki: 12x12, rys. 4. a) - e) przedstawiają kolejne kroki czasowe:



Czas: 2020-04-03 09:19:12.928



Czas: 2020-04-03 09:20:43.123



Czas: 2020-04-03 09:22:07.313

Rys. 4. a) Stan początkowy - glider, 1. krok

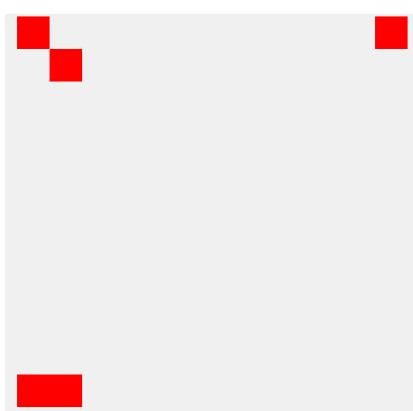
Rys. 4. b) Stan początkowy - glider, 2. krok

Rys. 4. c) Stan początkowy - glider, 3. krok



Czas: 2020-04-03 09:23:22.339

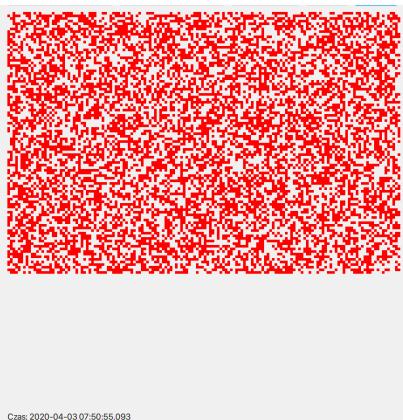
Rys. 4. d) Stan początkowy - glider, 4. krok



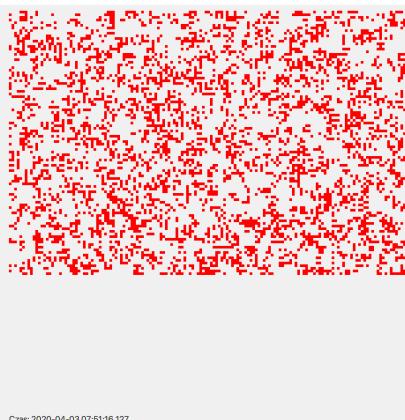
Czas: 2020-04-03 09:24:43.149

Rys. 4. e) Stan początkowy - glider, 5. krok

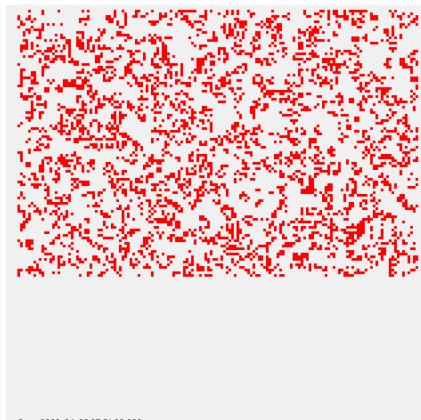
- d) Stan początkowy: losowy, rozmiar siatki: 150x100, rys. 5. a) - c) przedstawiają kolejne kroki czasowe:



Rys. 5. a) Stan początkowy - losowy, 1. krok

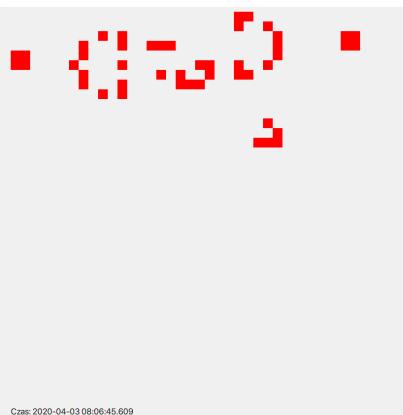


Rys. 5. b) Stan początkowy - losowy, 2. krok

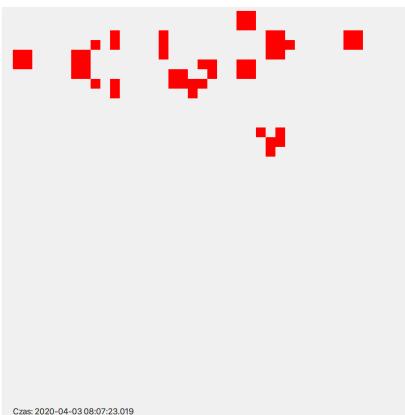


Rys. 5. c) Stan początkowy - losowy, 3. krok

- e) Stan początkowy: ustawiony ręcznie, rozmiar siatki: 40x40, rys. 6. a) - c) przedstawiają kolejne kroki czasowe, rys. 6. d) przedstawia modyfikację siatki w trakcie symulacji, rys. 6. e) i f) przedstawiają kolejne kroki symulacji po wprowadzeniu modyfikacji:



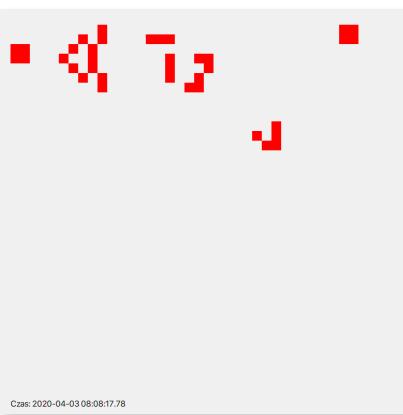
Rys. 6. a) Stan początkowy - ustawiony ręcznie, 1. krok



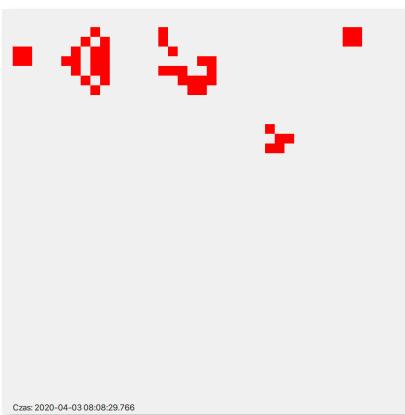
Rys. 6. b) Stan początkowy - ustawiony ręcznie, 2. krok



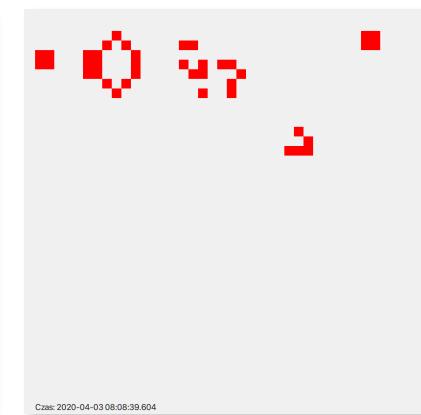
Rys. 6. c) Stan początkowy - ustawiony ręcznie, 3. krok



Rys. 6. d) Modyfikacja siatki - 1. krok



Rys. 6. e) Po modyfikacji siatki - 2. krok



Rys. 6. f) Po modyfikacji siatki - 3. krok

### 3. Spełnione wymagania

- a) Użytkownik może przez interfejs ustawić rozmiar siatki (maksymalny rozmiar - 600x600);
- b) Zdefiniowano trzy początkowe struktury - jedną niezmienną (pokazana w 2. a), jeden oscylator (2. b) i jeden glider (2. c);
- c) Zastosowano periodyczny warunek brzegowy - zaprezentowany w punkcie 2. c);
- d) Stan komórek można modyfikować w trakcie symulacji, jak pokazano w punkcie 2. e). Można, ale nie trzeba, w tym celu zatrzymać symulację przyciskiem „Stop”;
- e) Kroki symulacji są kolejno wizualizowane na jednym płótnie w odstępie około jednej sekundy.

### 4. Kod

Komórkę reprezentuje zmienna typu *boolean*, żywym komórkom przypisana jest wartość *true*, martwym - *false*. Automat jest reprezentowany w kodzie dwuwymiarową tablicą *grid*.

Obliczanie liczby sąsiadów prezentuje Listing 1. Funkcja *getNeighbours()* przyjmuje tablicę i położenie komórki, oraz sprawdza stan każdego sąsiada komórki w promieniu 1. Jeżeli sprawdzana komórka jest żywa, zmienna zwieracana *neighbours* jest inkrementowana. Kod oznaczony czerwoną ramką realizuje periodyczny warunek brzegowy - gdy w podanym położeniu nie ma żadnej komórki, wskazuje położenie po przeciwej stronie siatki.

Wizualizację kolejnych kroków symulacji w formie animacji zrealizowano z wykorzystaniem klasy *Timeline*. Tworzenie jednej klatki prezentuje Listing 2. Do tablicy pomocniczej zapisywane są wartości *grid* z poprzedniego kroku, następnie na jej podstawie liczona jest liczba sąsiadów komórek, podczas gdy do *grid* zapisywane są zmiany stanów komórek. Zmodyfikowana siatka jest rysowana na płótnie, a klatka jest dodawana do animacji.

Jest przewidzianych sto klatek animacji, jednak można ją zakończyć w dowolnym momencie przyciskiem „Stop” lub uruchomić kolejne sto klatek przyciskiem „Start”.

```
private int getNeighbours(boolean[][] grid, int y, int x){  
    int neighbours=0;  
    int a, b;  
    for(int i = -1; i<=1; i++){  
        for(int j = -1; j<=1; j++){  
            b=y+i;  
            if(b== -1)  
                b=grid.length-1;  
            else if(b==grid.length)  
                b=0;  
            a=x+j;  
            if(a== -1)  
                a=grid[0].length-1;  
            else if(a==grid[0].length)  
                a=0;  
            if(grid[b][a]&& (i!=0 || j!=0))  
                neighbours++;  
        }  
    }  
    return neighbours;  
}
```

```
KeyFrame kf = new KeyFrame(Duration.seconds(k+1), actionEvent2 -> {  
    for (int i = 0; i < width; i++) {  
        for (int j = 0; j < height; j++) {  
            temp[j][i] = gd[j][i];  
        }  
    }  
    for (int i = 0; i < width ; i++) {  
        for (int j = 0; j < height ; j++) {  
            int nb = getNeighbours(temp, j, i);  
            if (temp[j][i]) {  
                if (nb < 2 || nb > 3)  
                    gd[j][i] = false;  
                else if (nb == 3)  
                    gd[j][i] = true;  
            }  
        }  
    }  
    draw(gc, czas, gd, width, height);  
});  
play.getKeyFrames().add(kf);
```

Listing 1. Obliczanie liczby sąsiadów

Listing 2. Tworzenie jednej klatki animacji