

Akademia Ekonomiczno-Humanistyczna

Algorytm sztuczne życie

Dmytro Martsynenko

2021

Wstępny opis projektu

Zadaniem projektu było dokonanie wstrzyknięcia dużej liczby bakterii do komórek świata, co odbywa się przy *zakażeniu* w rzeczywistości, dlatego dalej będziemy nazywać *wstrzyknięcie bakterii zakażeniem*. Po zakażeniu komórek świata należy podać некоторą ilość pełzacz, co jest bardzo podobne do *podawania leczenia*. Dokonywać takiego leczenia należy przez некоторy czas, tak żeby zobaczyć efekt, ponieważ algorytm pobiera wyniki przy domyślnych ustawieniach *co 10 taktów*. Zadaniem opcjonalnym jest implementacja świata w postaci graficznej, z symulacją w czasie rzeczywistym. Pozwala to lepiej zrozumieć zachowanie modelu *Drapieżniki i ofiary* w praktyce. W celu implementacji zakażenia w symulowanym świecie, należy dokonać wyboru punktu wstrzyknięcia. Najlepiej to zrobić przy pomocy losowania dwóch liczb odpowiadającym pozycji losowanego punktu.

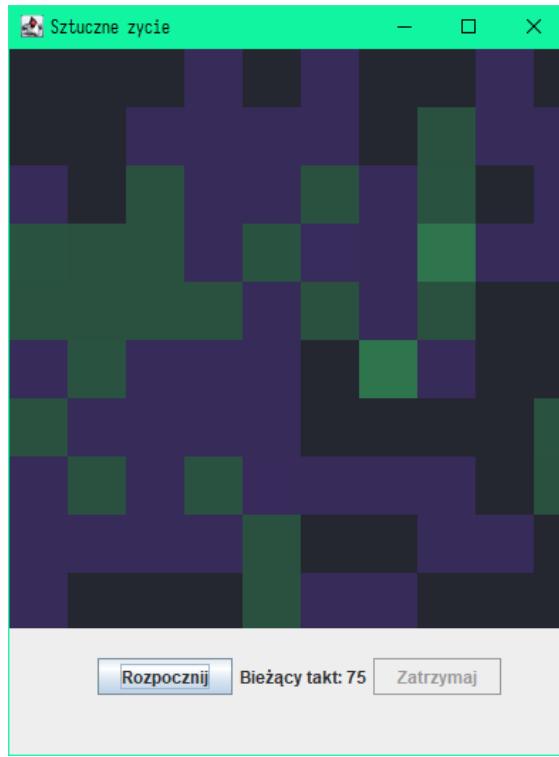
Pojawiły się dodatkowe parametry, niezbędne do ustawienia parametrów zakażenia:

- Łączna liczba bakterii wykorzystanych przy zakażeniu - **INJECTED_BACT_NUM**
- Od którego takty zaczyna się wstrzyknięcie - **START_INJECTING_AT_TACT**
- Długość wstrzyknięcia - **INJECT_FOR_TACTS**

Także są parametry *leczenia*:

- Łączna liczba pełzacz do leczenia - **INJECTED_CREEPERS_NUM**
- Przez ile taktów od początku zakażenia zacząć leczenie - **INJECT_CREEPERS_OFFSET**

Końcowa wersja projektu przedstawiona w postaci graficznej, która da możliwość zobaczyć proces zakażenia, a następnie działanie procesu leczenia.



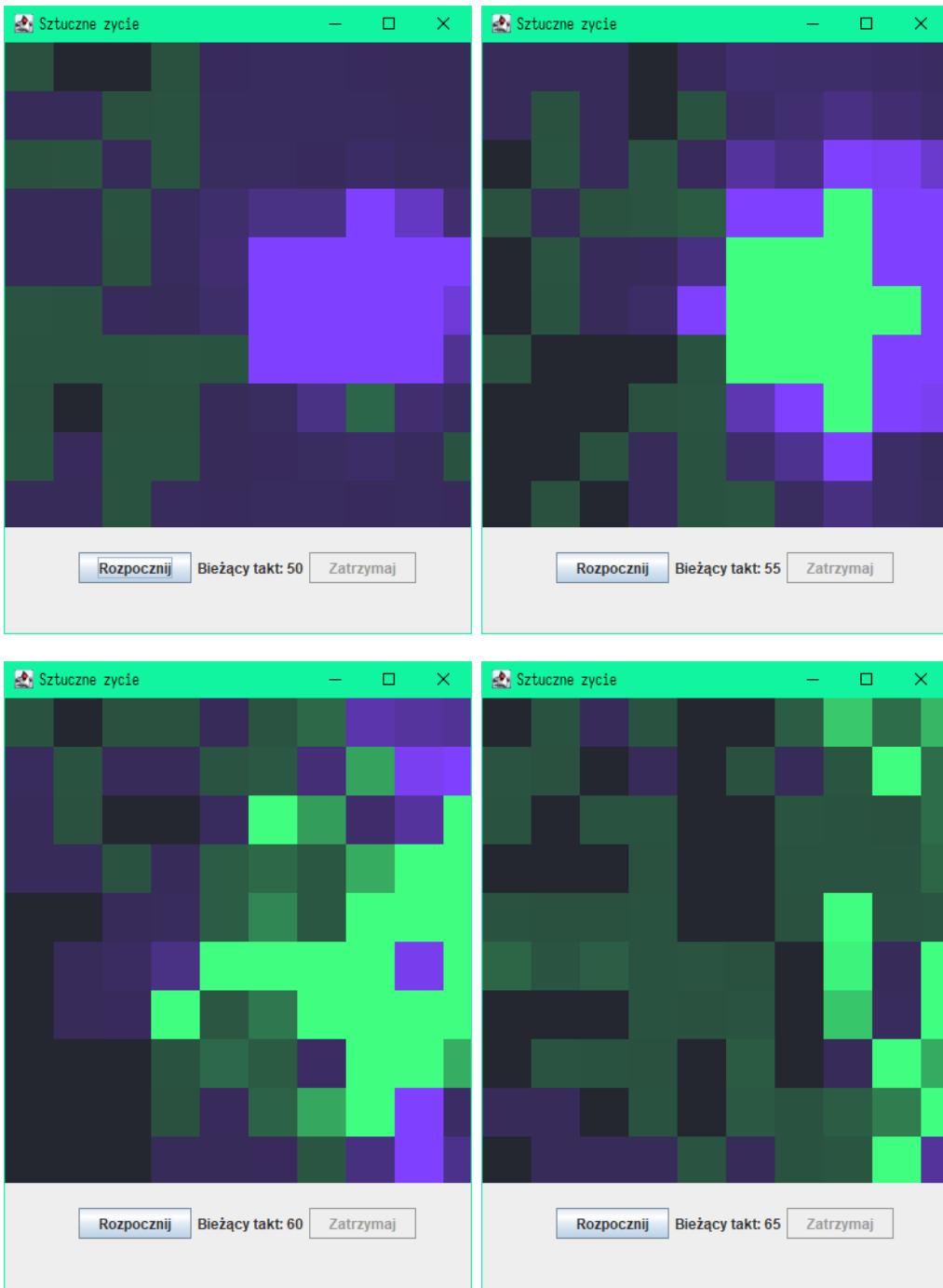
Rysunek 1: Wygląd okna symulacji

Na *Rysunku 1* przedstawiono wygląd okna symulacji po uruchomieniu programu. Fioletowym kolorem zaznaczono komórki świata, w których liczba bakterii jest *większa od liczby półzaczęty*, a zielonym - *równa lub mniejsza*.

W ciemniejszych komórkach liczba organizmów stanowi zero.

Żeby proces zakażenia oraz leczenia było łatwo zobaczyć, został dodany parametr **MAX_INTENSITY_COUNT**, który wyznacza maksymalną intensywność koloru dla komórki podając pewną liczbę organizmów.

Działa to w taki sposób, że dzielimy znaczenie alpha ustawionego koloru dla komórki przez **MAX_INTENSITY_COUNT**. Należy zwrócić uwagę, że na *Rysunku 1* alpha jest znacznie mniejsza od 1.



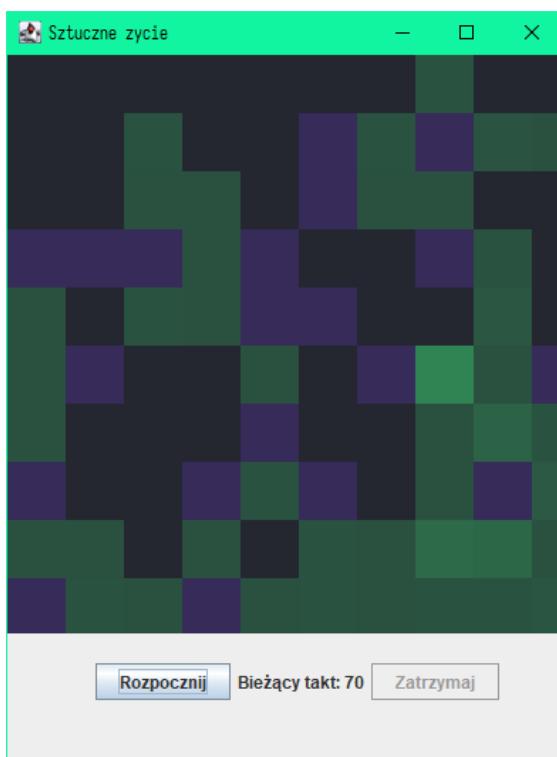
Rysunek 2: Przykład uruchomienia symulacji (takty 50-65)

W danym przykładzie parametr **START_INJECTING_AT_TACT** ustawiono na **40**. Ale ponieważ zakażenie dopiero zaczyna się w tym takcie, zobaczymy efekt chociażby po kilku taktach, a w danym przypadku w takcie 45, bo ustawiliśmy parametr **VIEW_NUM_TACT** na **5**, co znaczy, że aktualizujemy stan symulacji co 5 taktów.

Najbardziej znaczącymi taktami w danej symulacji są takty 50-65.

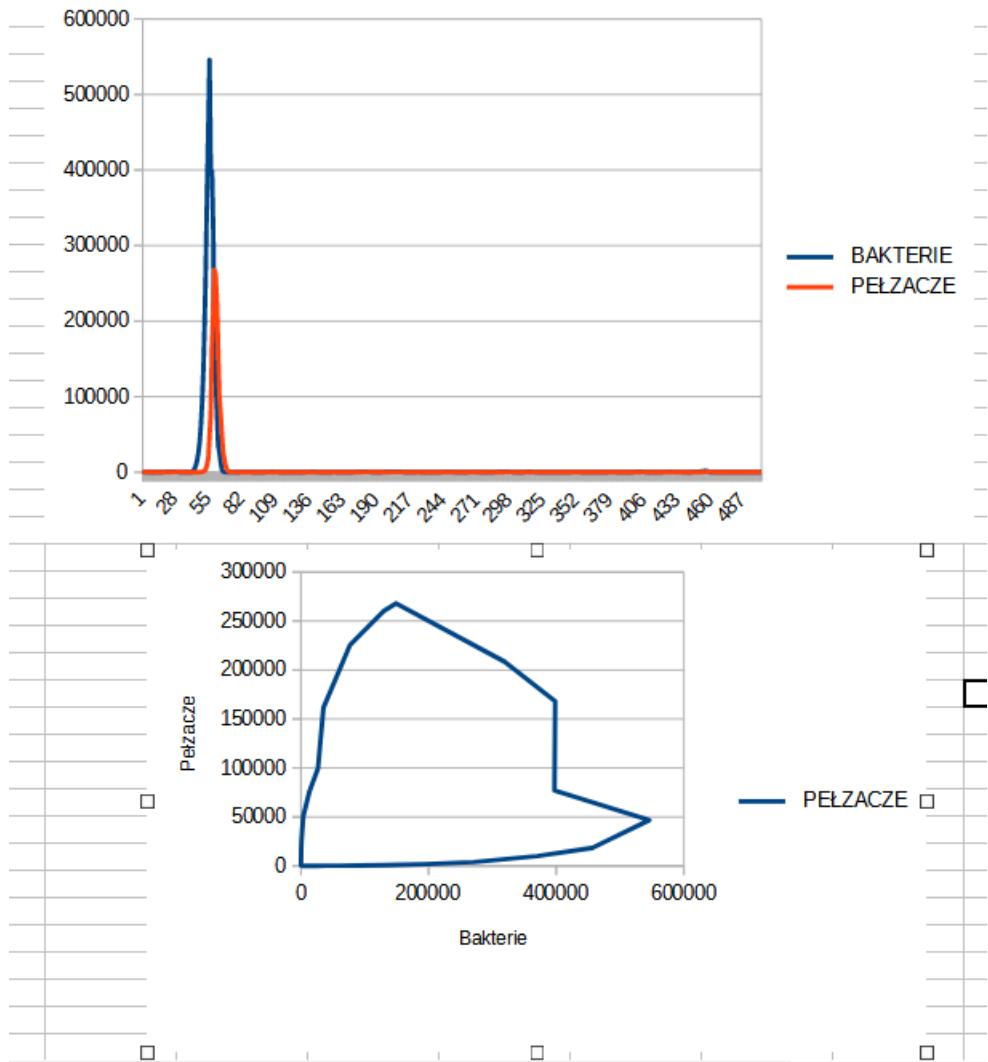
- W takcie 50 rozszerza się obszar zakażenia, który na początku symulacji znajduje się w jednym punkcie.

- W taktie 55 gwałtownie rośnie liczba pełzaczy, które wykonują funkcje obronną symulowanego organizmu.
- W taktie 60 widać delokalizację całego procesu, zajmuje on prawie cały obszar świata, przewagę w liczbie trzymają pełzacze.
- W taktie 65 liczba obu organizmów zaczyna się zmniejszać się, leczenie już się zakończyło.



Rysunek 3: Koniec procesów zakażenia oraz leczenia

Na *Rysunku 3* zobaczymy stan świata w *takcie 70*. Niczym się nie różni on od stanu stabilnego.



Rysunek 4: Wykres uruchomionej na *Rysunku 2* symulacji

Parametry uruchomienia powyższego przykładu:

PARAMETRY URUCHOMIENIA

Całkowita liczba taktów

500

Wyświetlaj wyniki co

5 taktów

Początkowa liczba pełzaczys

500

Początkowa liczba bakterii

500

Liczba energii do urodzenia pełzaczys

3

Zapas energii nowourodzonego pełzaczys

2

Rezerwa energii zostawiana po urodz. pełzaczys

2

Max. pełzaczys w 1 takcie

4

Max. bakterii do zjedz. przez pełzacza w 1 takcie

13

Współczynnik rozmnażania bakterii

0,50

Współczynnik rozprzestrzeniania bakterii

0,60

Początek wstrzyknięcia w

40 takcie

Długość wstrzyknięcia

25 takty/-ów

Wstrzyknięcie pełzaczys przez

10 takty/taktów

Liczba bakterii do wstrzyknięcia

32000

Liczba pełzaczys do wstrzyknięcia

4000

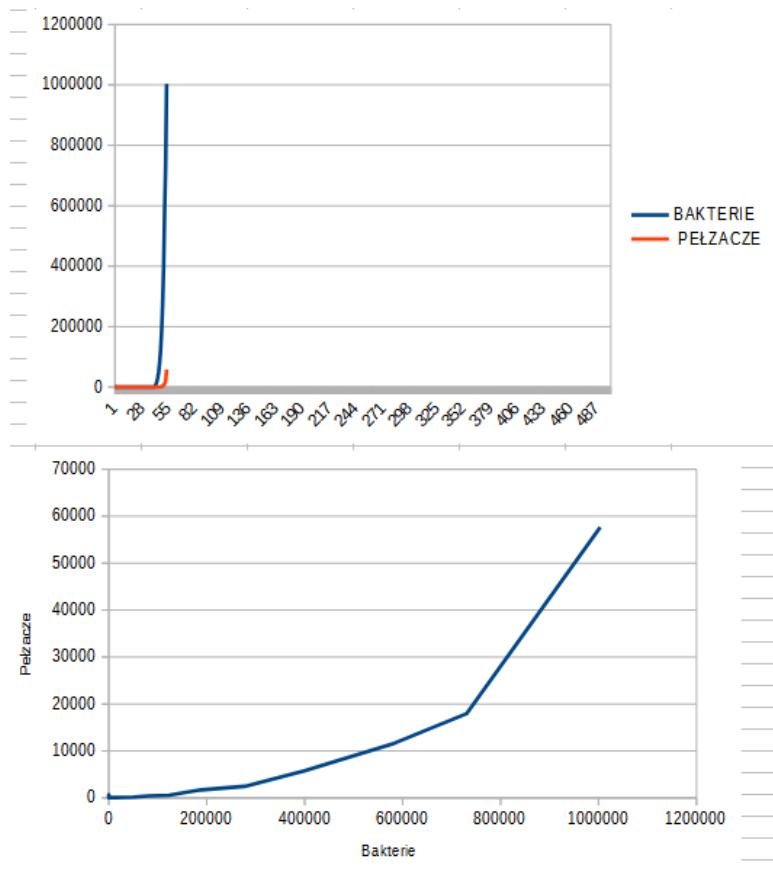
Przeprowadzone badania

Przedstawione badanie przeprowadzone są na podstawie wyników z 7 uruchomień.

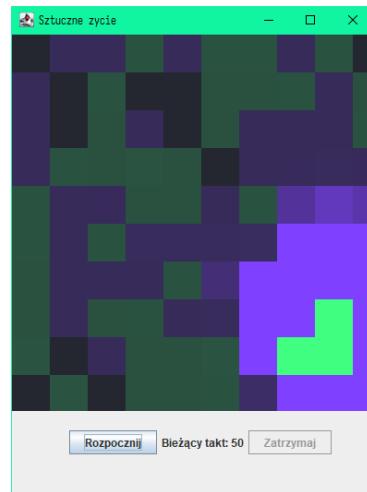
Wyniki uruchomień można podzielić na dwie grupy: **pierwsza z nich** to uruchomienia z dość krótkim *okresem wstrzyknięcia*.

Najważniejsze początkowe parametry uruchomień tej grupy są następujące:

Początkowa liczba pęzaczyc
500
Początkowa liczba bakterii
500
Początek wstrzyknięcia w
40 taktie
Długość wstrzyknięcia
15 takty/-ów
Wstrzyknięcie pęzaczyc przez
10 takty/taktów
Liczba bakterii do wstrzyknięcia
64000
Liczba pęzaczyc do wstrzyknięcia
8000

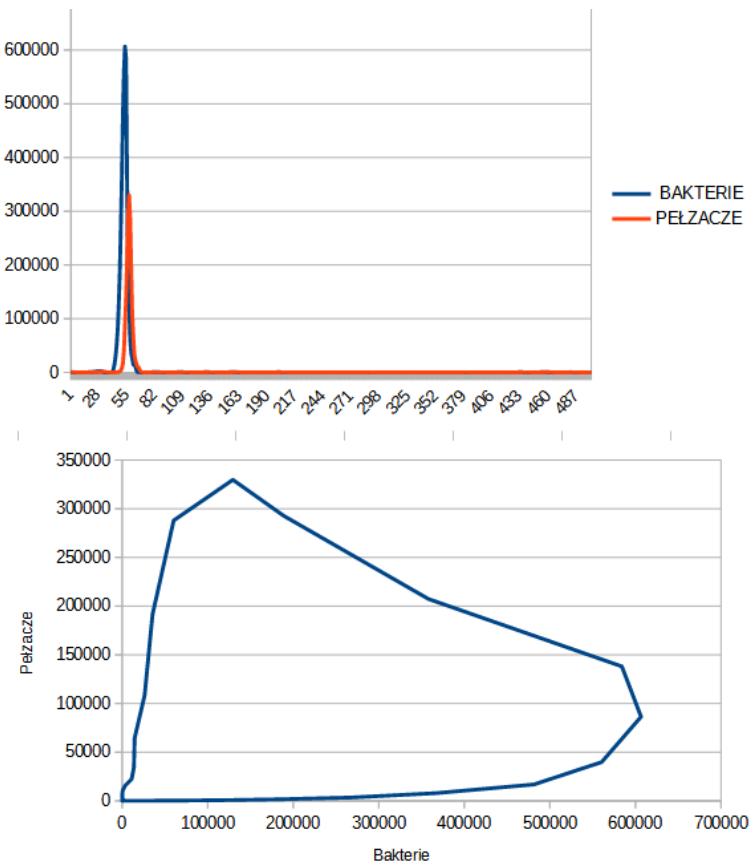


Z powyższego wykresu widać, że przy podaniu dość dużej ilości bakterii w krótkim okresie, niezwłocznie przekracza ona liczbę *1 million*, która jest maksymalnie dopuszczalną liczbą bakterii w badanym świecie.

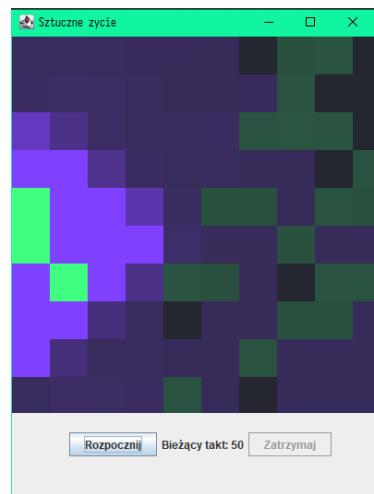


2D reprezentacja pokazuje, że w 50-tym takcie mamy dość mały obszar zakażenia w porównaniu do bieżącej liczby bakterii.

Nie zmieniając parametrów uruchomimy jeszcze raz, żeby zobaczyć czy będzie liczba bakterii przekraczać *max. 1 million* nadal.



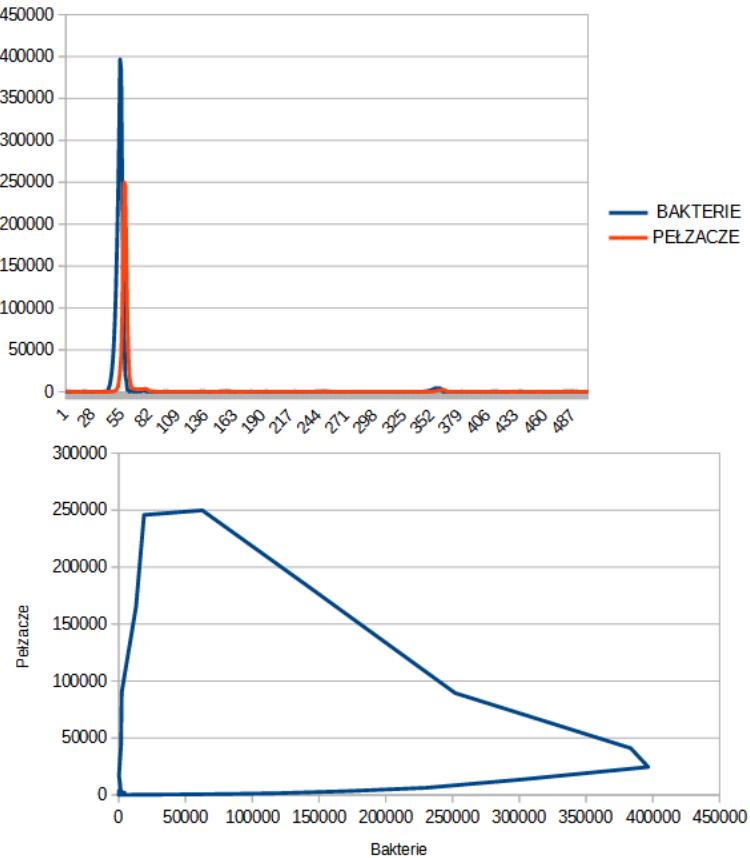
Po ponownym uruchomieniu programu, zobaczymy, że liczba bakterii już sięga tylko *ok. 600 tysięcy*, ale liczba jest nadal dość dużą, co zobaczymy w 50 takcie 2D reprezentacji.



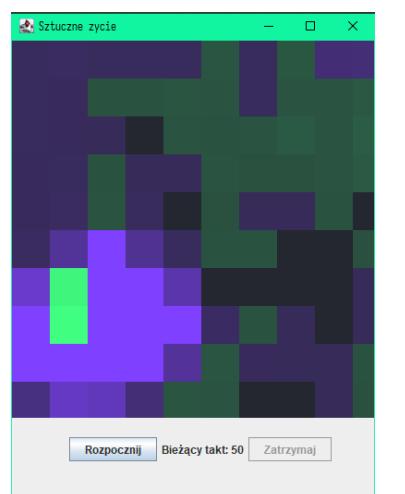
Następnie, uruchomimy program z nieco innymi parametrami, a właśnie:

```
-- Długość wstrzyknięcia
-----  
35 taktów/-ów
```

W tym uruchomieniu zwiększymy długość wstrzynienia o *20 taktów*.



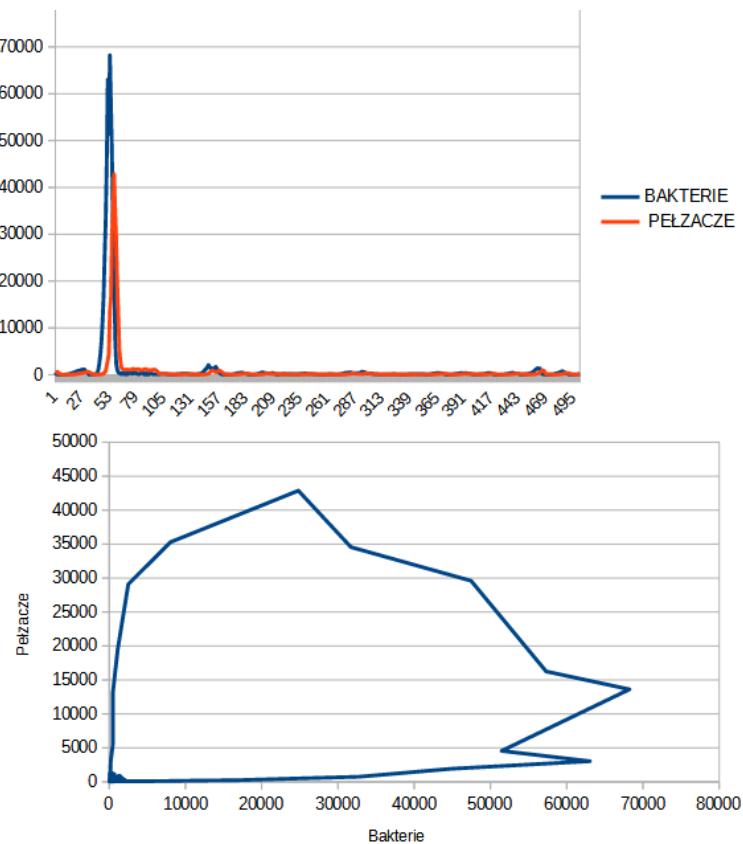
Wykres danej symulacji prawie niczym się nie różni od pierwszego, tylko możemy zobaczyć słabą aktywność po spadie liczby obu organizmów (nadal są aktywne, ponieważ rozmieszczone są w różnych komórkach, pełzaczki nie mogą zjeść bakterii w sąsiednich komórkach).



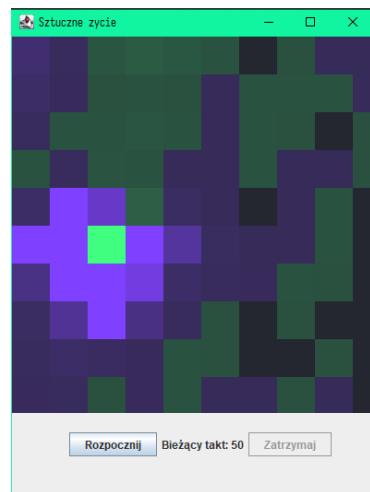
Obszar zakażenia jest nieco mniejszy, co mówi o słabszej aktywności bakterii (nie zmieniamy współczynnika rozmnażania bakterii).

W kolejnym uruchomieniu jeszcze raz zwiększymy *długość wstrzyknięcia*, a także zmniejszymy *liczbę do wstrzyknięcia* obu organizmów, żeby otrzymać bardziej wyraźne wyniki:

Długość wstrzyknięcia
55 takty/-ów
Liczba bakterii do wstrzyknięcia
32000
Liczba pełzaczego do wstrzyknięcia
4000



Po przeprowadzeniu symulacji, widzimy, że jest podobny skok liczby bakterii od **40** do **75** taktu, ale jeżeli zwrócimy uwagę, to aktywność obu organizmów nadal trwa (takty 79-105), i tylko po takcie 105 zaczyna spadać.

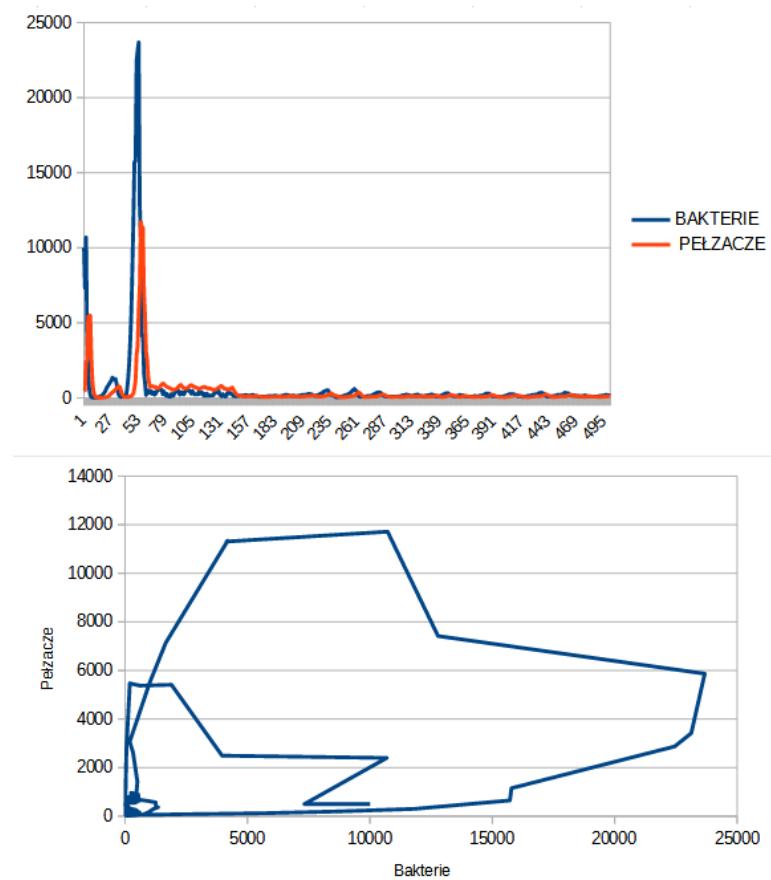


Obszar zakażenia jeszcze się zminiejszył, ponieważ wstrzyknięcia dokonano w dłuższym okresie.

W następnej grupie się znajdują wyniki uruchomień z bardzo długim okresem wstrzyknięcia:
Początkowe parametry dla danej grupy to:

```
-- Początkowa liczba pełzaczyc  
-----  
500  
-- Początkowa liczba bakterii  
-----  
10000  
-- Początek wstrzyknięcia w  
-----  
40 taktie  
-- Długość wstrzyknięcia  
-----  
100 takty/-ów  
-- Wstrzyknięcie pełzaczyc przez  
-----  
10 takty/taktów  
-- Liczba bakterii do wstrzyknięcia  
-----  
32000  
-- Liczba pełzaczyc do wstrzyknięcia  
-----  
4000
```

Długość wszstrzyknięcia stanowi 100 taktów, co o 6,6 razy więcej niż w początkowym stanie poprzedniej grupy symulacji.

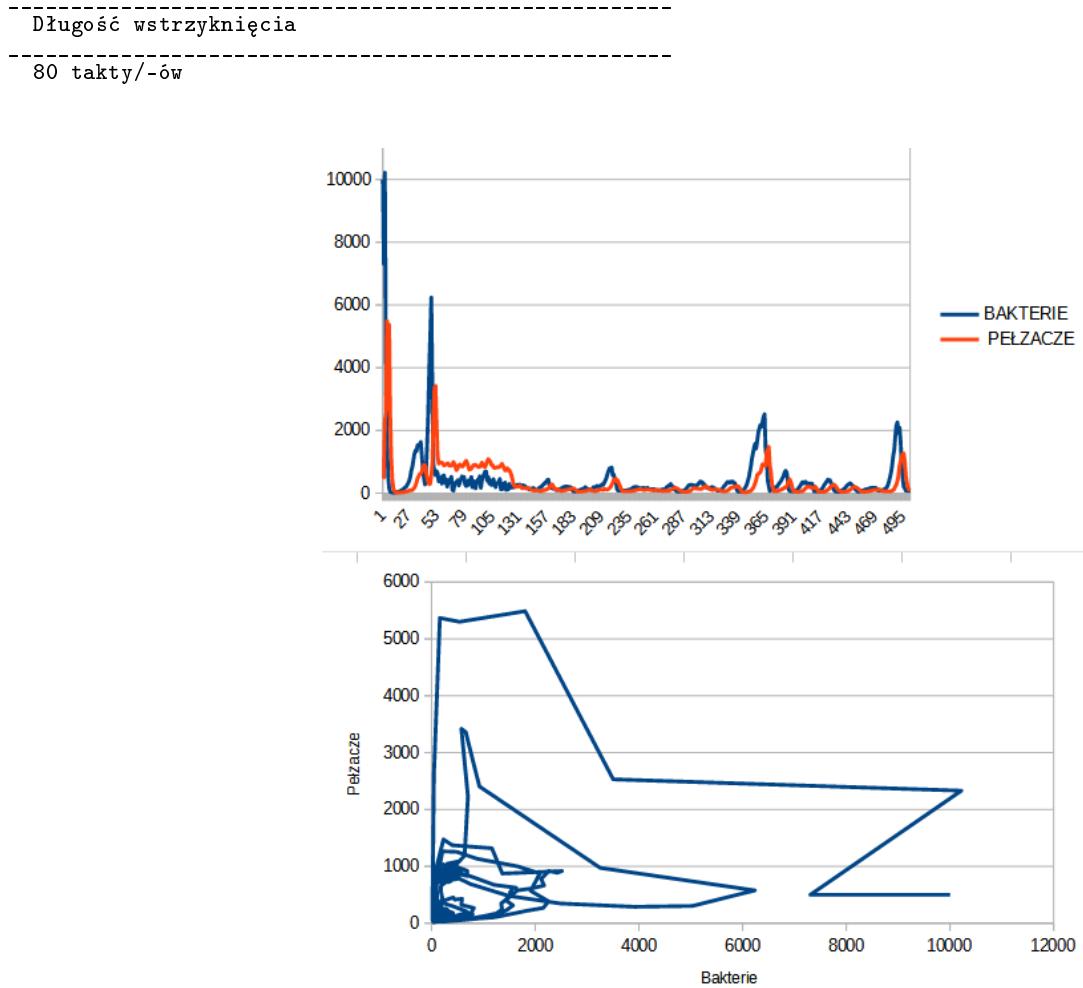


Wykres zupełnie się różni od poprzedniej grupy. Maksymalna liczba bakterii jest o wiele mniejsza - ok 25 tysięcy. Już wyraźnie widać końcową aktywność bakterii oraz pełzaczy, po spadzie w okresie od **79** do **157** taktu.

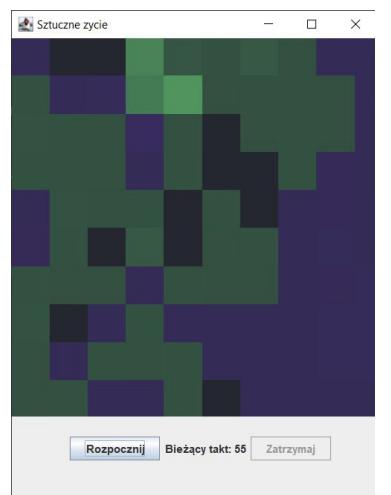


Wynik uruchomienia symulacji w wyglądzie graficznej reprezentacji pokazuje, że obszar zakażenia jest bardzo mały, nie przekraczający 6 komorek w promieniu, zakażenie jest zlokalizowane.

Następnie zmniejszymy długość wstrzyknięcia o 20 taktów

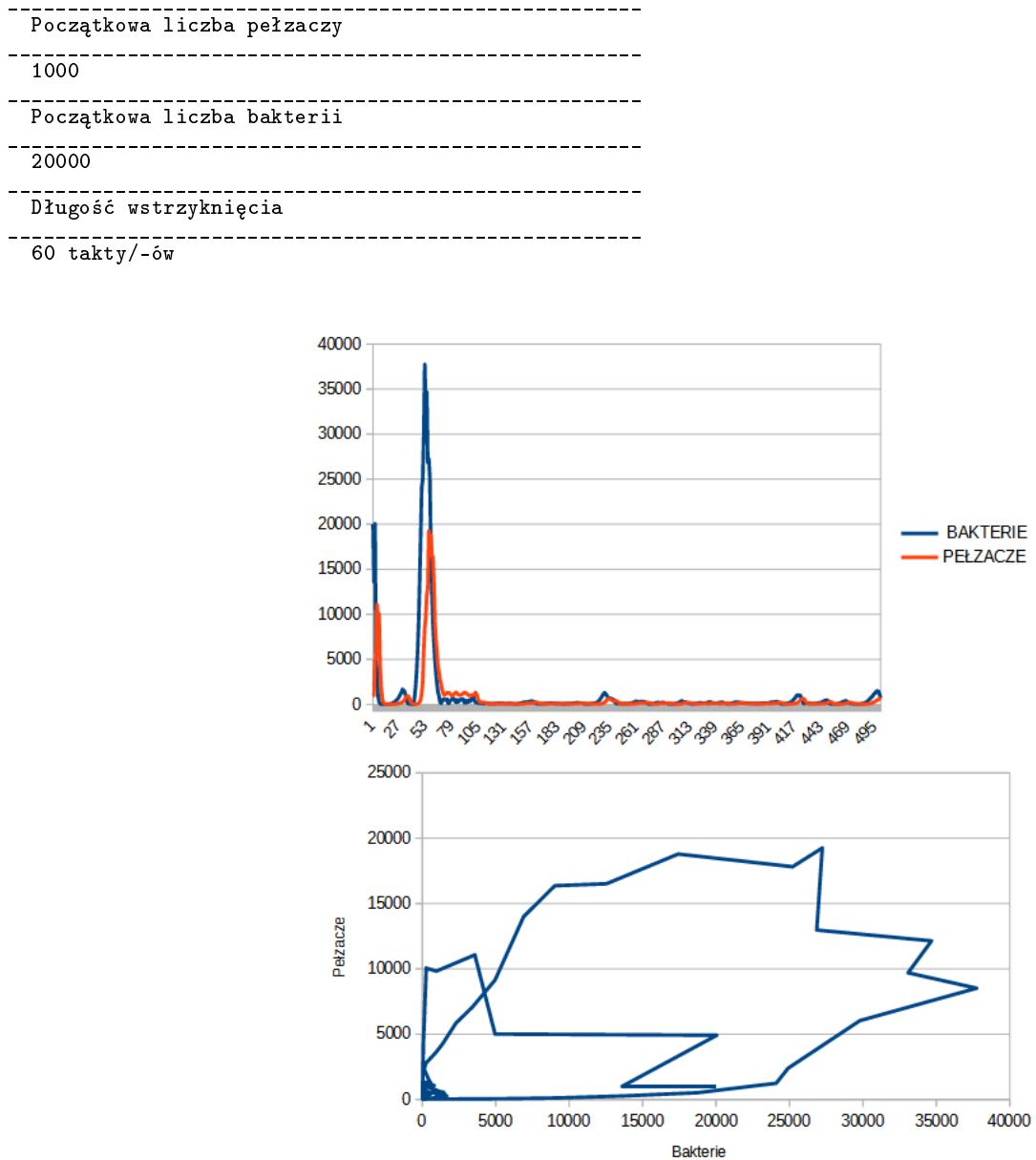


Otrzymaliśmy wykres, w którym wsztrzyknięcie nie powoduje żadnych gwałtownych zmian, ale nadal widać końcową aktywność, charakterystyczną właśnie dla wstrzyknięcia.



W graficznej reprezentacji prawie nie widzimy momentu zakażenia, świat jest stabilny.

W ostatnim uruchomieniu jeszcze zmniejszymy *okres wstrzyknięcia*, oraz zwiększymy *początkową liczbę bakterii* oraz *pełzaczów*



Jak widać, właśnie w skutek zwiększenia początkowej liczby organizmów otrzymujemy bardziej wyraźny wynik zakażenia.



Pomimo tego, mamy dwa bardzo małe obszary zakażenia. Znaczy to, że aktywność bakterii jest na bardzo niskim poziomie, ponieważ, jeżeli zobaczymy na wykresie, *liczba pełzacy* prawie od razu po wstrzyknięciu wzrosła, dlatego nie zobaczymy dużego obszaru zakażenia w graficznej reprezentacji.

Wniosek

Zwiększąc długość wsztrzyknięcia, nie zmieniając liczby bakterii, otrzymujemy bardziej łagodne zakażenie, które się nadaje do szczepionki, a zmniejszając otrzymujemy niszczący wpływ na badane komórki. Także podbiór parametrów jest przydatny dla podtrzymywania kolonii bakterii oraz pełzaczy w badanym środowisku.

Podsumowując otrzymane wyniki, z pewnością możemy stwierdzić, że przy pomocy parametrów możemy dość realistycznie zamodelować rzeczywiste procesy, w tym *zakażenie* organizmu oraz jego *leczenie*. Wykonanie tego projektu pozwala zrozumieć jak wykorzystać język **Java** oraz narzędzie do *automatyzacji* budowania projektów **Maven** dla realizacji zadań.

Kod projektu

ArtLife.java

```
package com.aeh;
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.*;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.LinkedList;
import java.util.ListIterator;
class Init {
    static final int INJECTED_BACT_NUM = 32000; //Liczba bakterii do wstrzygnięcia
    static final int INJECTED_CREEPERS_NUM = 4000; //Liczba pęzacz do wstrzygnięcia
    static final int START_INJECTING_AT_TACT = 40; //Rozpoczęcie wstrzygnięcia w taktie
    static final int INJECT_FOR_TACTS = 25; //Długość wstrzygnięcia w taktach
    static final int INJECT_CREEPERS_OFFSET = 10; //Podawanie leczenia przez ... taktów
    static final int SIZE_WORLD = 10; //rozmiar świata - NIE ZMIENIAĆ
    static final int NUM_TACT = 500; //całkowita liczba taktów
    static final int VEW_NUM_TACT = 5; //co ile taktów wyświetla wyniki
    static final int START_NUM_CREEPERS = 500; //początkowa liczba pęzacz
    static final int START_NUM_BACT = 500; //początkowa liczba bakterii
    static final int CREEPER_ENERGY_PRO_LIFE = 3; //ilość energii potrzebna do urodzenia nowego pęzacza
    static final int CREEPER_INITIAL_ENERGY = 2; //zasób energii nowo urodzonego pęzacza
    static final int CREEPER_ENERGY_RESERVE = 2; //rezerwa energii zostawiana podczas rodzenia nowego pęzacza
    //potrzebna do przetrwania, gdy jest mało pożywienia
    static final int MAX_CREEPER_NUM_BORN_PER_TACK = 4; //maksymalna liczba pęzacz rodzonych przez jednego
    //pezacza w jednym taktie.
    // Liczba pęzacz, które mogą się urodzić w jednym taktie jest ograniczona przez ilość energii pęzacza po
    // odjęciu CREEPER_ENERGY_RESERVE. Pezacz nie może zgromadzić zbyt dużo energii, ponieważ gdy tylko
    // przekracza poziom energii równy CREEPER_ENERGY_PRO_LIFE + CREEPER_ENERGY_RESERVE w następnym taktie
    // rodzi co najmniej jednego pęzacza i jego poziom energii jest zmniejszany o CREEPER_ENERGY_PRO_LIFE
    // na każdego urodzonego pęzacza.
    static final int MAX_BACT_EATEN_BY_CREEPER = 13; //Maksymalna liczba bakterii zjadanych przez pęzacza
    //w jednym taktie
    static final double BACT_MULTIPLICATION_RATE = 0.5; //współczynnik rozmnażania bakterii - tyle nowych bakterii
    //powstaje z jednej bakterii w każdym taktie.
    //MOŻE PRZYJMUWAĆ WARTOŚCI UŁAMKOWE.
    static final double BACT_SPREAD_RATE = 0.6; //współczynnik rozprzestrzeniania nowo urodzonych bakterii.
    //DOPUSZCZALNY ZAKRES: od 0 do 1
    //Np. przy wsp. = 0.7, 70% zostaje w komórce,
    //w której się urodziła, a 30% przenosi się
    //do sąsiednich losowo wybranych
    static final int BACT_NUM_LIMIT = 1000000; //graniczna liczba bakterii dla całego świata.
    //Po przekroczeniu tej liczby komórki umierają/koniec symulacji.
    static final int TACT_TIME_DURATION = 425; //długość symulacji w milisekundach
    static final int MAX_INTENSITY_COUNT = 1000; //maksymalna intensywność koloru przy liczbie
    static final boolean ENABLE_LOGGING = true; //zapisywanie wyników uruchomienia do pliku
    //lista modyfikatorów pozycji - w niej określamy, które miejsca (względem obecnego położenia)
    //mają być sprawdzane/uwzględniane przy przemieszczaniu się pęzacz lub bakterii z obecnego położenia
    static ArrayList<LocationModifier> initializedLocationModifiersList() {
        ArrayList<LocationModifier> locationModifiers = new ArrayList<>(4);
        locationModifiers.add(new LocationModifier(-1, 0));
        locationModifiers.add(new LocationModifier(1, 0));
        locationModifiers.add(new LocationModifier(0, -1));
        locationModifiers.add(new LocationModifier(0, 1));
        //możliwość sprawdzania dodatkowo komórek w narożnikach
        //locationModifiers.add(new LocationModifier(-1, -1));
        //locationModifiers.add(new LocationModifier(1, 1));
        //locationModifiers.add(new LocationModifier(-1, 1));
        //locationModifiers.add(new LocationModifier(1, -1));
        return locationModifiers;
    }
    public static PrintStream outStream;
}
class LocationModifier {
    int modifyX, modifyY;
```

```

    LocationModifier(int x, int y) {
        modifyX = x;
        modifyY = y;
    }
}

public class ArtLife {
    private static ArrayList<Integer> totallyCreepers = new ArrayList<>();
    private static ArrayList<Integer> totallyBacteria = new ArrayList<>();
    //powyżej kolekcje pomocnicze do zapamiętania liczby bakterii
    //i pełzacy w każdym takcie, celem późniejszego wyświetlenia
    private static void bacteriaTest(World w) {
        //wyświetla konsolowo liczbę bakterii w danym takcie
        //w układzie tablicy
        for (int i = 0; i < Init.SIZE_WORLD; i++) {
            for (int j = 0; j < Init.SIZE_WORLD; j++)
                Init.outStream.format("%4d", w.board[i][j].getBactNum());
            Init.outStream.print("\n");
        }
    }
    private static void creepersTest(World w) {
        //wyświetla konsolowo liczbę pełzacy w danym takcie
        //w układzie tablicy
        for (int i = 0; i < Init.SIZE_WORLD; i++) {
            for (int j = 0; j < Init.SIZE_WORLD; j++)
                Init.outStream.format("%4d", w.board[i][j].getCreepersNum());
            Init.outStream.print("\n");
        }
    }
    private static void mainTest(World w) {
        //wyświetla konsolowo w układzie tablicy
        //liczbę bakterii i pełzacy w danym takcie
        for (int i = 0; i < Init.SIZE_WORLD; i++) {
            for (int j = 0; j < Init.SIZE_WORLD; j++)
                Init.outStream.format("%8d%8d", w.board[i][j].getBactNum(),
                    w.board[i][j].getCreepersNum());
            Init.outStream.println();
        }
    }
    private static int totalNum(World w, String what) {
        //zwraca sumaryczną liczbę bakterii, lub pełzacy, w całym świecie
        int sum = 0;
        for (int i = 0; i < Init.SIZE_WORLD; i++)
            for (int j = 0; j < Init.SIZE_WORLD; j++)
                switch (what) {
                    case "BACTERIA":
                        sum += w.board[i][j].getBactNum();
                        break;
                    case "CREEPERS":
                        sum += w.board[i][j].getCreepersNum();
                }
        return sum;
    }
    private static void addNewBornOrganismsToMainWorldCellules(World mainWorld, World tempWorld) {
        for (int i = 0; i < Init.SIZE_WORLD; i++) {
            for (int j = 0; j < Init.SIZE_WORLD; j++) {
                mainWorld.board[i][j].addBactNum(tempWorld.board[i][j].getBactNum());
                mainWorld.board[i][j].creepers.addAll(tempWorld.board[i][j].creepers);
            }
        }
    }
    private static void printParams() {
        Init.outStream.print(" PARAMETRY URUCHOMIENIA\n");
        Init.outStream.println("-----");
        Init.outStream.println(" Całkowita liczba taktów ");
        Init.outStream.println("-----");
        Init.outStream.format(" %d", Init.NUM_TACT);
        Init.outStream.println("\n");
        Init.outStream.println("-----");
        Init.outStream.println(" Wyświetlaj wyniki co ");
        Init.outStream.println("-----");
        Init.outStream.format(" %d taktów", Init.VEW_NUM_TACT);
        Init.outStream.println("\n");
        Init.outStream.println("-----");
        Init.outStream.println(" Początkowa liczba pełzacy ");
        Init.outStream.println("-----");
        Init.outStream.format(" %d", Init.START_NUM_CREEPERS);
        Init.outStream.println("\n");
    }
}

```

```

Init.outStream.println("-----");
Init.outStream.println(" Początkowa liczba bakterii ");
Init.outStream.println("-----");
Init.outStream.format(" %d", Init.START_NUM_BACT);
Init.outStream.println("\n");
Init.outStream.println("-----");
Init.outStream.println(" Liczba energii do urodzenia pełzacza ");
Init.outStream.println("-----");
Init.outStream.format(" %d", Init.CREEPER_ENERGY_PRO_LIFE);
Init.outStream.println("\n");
Init.outStream.println("-----");
Init.outStream.println(" Zapas energii nowourodzonego pełzacza ");
Init.outStream.println("-----");
Init.outStream.format(" %d", Init.CREEPER_INITIAL_ENERGY);
Init.outStream.println("\n");
Init.outStream.println("-----");
Init.outStream.println(" Rezerwa energii zostawiana po urodz. pełzacza ");
Init.outStream.println("-----");
Init.outStream.format(" %d", Init.CREEPER_ENERGY_RESERVE);
Init.outStream.println("\n");
Init.outStream.println("-----");
Init.outStream.println(" Max. pełzacy w 1 taktie ");
Init.outStream.println("-----");
Init.outStream.format(" %d", Init.MAX_CREEPER_NUM_BORN_PER_TACK);
Init.outStream.println("\n");
Init.outStream.println("-----");
Init.outStream.println(" Współczynnik rozmnażania bakterii ");
Init.outStream.println("-----");
Init.outStream.format(" %.2f", Init.BACT_MULTIPLICATION_RATE);
Init.outStream.println("\n");
Init.outStream.println("-----");
Init.outStream.println(" Współczynnik rozprzestrzeniania bakterii ");
Init.outStream.println("-----");
Init.outStream.format(" %.2f", Init.BACT_SPREAD_RATE);
Init.outStream.println("\n");
Init.outStream.println("-----");
Init.outStream.println(" Początek wstrzyknięcia w ");
Init.outStream.println("-----");
Init.outStream.format(" %d taktie", Init.START_INJECTING_AT_TACT);
Init.outStream.println("\n");
Init.outStream.println("-----");
Init.outStream.println(" Długość wstrzyknięcia ");
Init.outStream.println("-----");
Init.outStream.format(" %d takty/-ów", Init.INJECT_FOR_TACTS);
Init.outStream.println("\n");
Init.outStream.println("-----");
Init.outStream.println(" Wstrzyknięcie pełzacy przez ");
Init.outStream.println("-----");
Init.outStream.format(" %d takty/taktów", Init.INJECT_CREEPERS_OFFSET);
Init.outStream.println("\n");
Init.outStream.println("-----");
Init.outStream.println(" Liczba bakterii do wstrzyknięcia ");
Init.outStream.println("-----");
Init.outStream.format(" %d", Init.INJECTED_BACT_NUM);
Init.outStream.println("\n");
Init.outStream.println("-----");
Init.outStream.println(" Liczba pełzacy do wstrzyknięcia ");
Init.outStream.println("-----");
Init.outStream.format(" %d", Init.INJECTED_CREEPERS_NUM);
Init.outStream.println("\n\n");
}

public static void main(String[] args) {
    try {
        Init.outStream = new PrintStream(System.out, true, "UTF-8");
    }
    catch (UnsupportedEncodingException e){
        Init.outStream = System.out;
        Init.outStream.println("Nieobsługiwane kodowanie.");
    }
    if (Init.ENABLE_LOGGING) {
        File logDir = new File("log/");
        if (!logDir.exists()) {

```

```

        logDir.mkdir();
    }
    SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
    Date date = new Date();
    String timestamp = formatter.format(date).replaceAll("[:-]", "").replaceAll("[ ]", "_");
    Init.outStream.print("Zapisywanie do pliku log/" + timestamp + ".txt");
    try {
        OutputStream printStream = new FileOutputStream("log/" + timestamp + ".txt", true);
        Init.outStream = new PrintStream(printStream, true, "UTF-8");
    } catch (IOException e) {
        Init.outStream.println("Nie mogę zapisać logi");
        e.printStackTrace();
    }
}
printParams();
World mainWorld = new World();
World tempWorld; // dodatkowy świat potrzebny czasowo w trakcie creepersAndBacteriaAction
                // do przechowywania nowo urodzonych bakterii (wszystkich) i pełzaczów (tylko tych
                // które przemieszczają się do sąsiednich komórek), aby nie zwiększały populacji
                // w niewylosowanych jeszcze komórkach.
                // Po zakończeniu akcji pełzaczów i bakterii dla wszystkich komórek,
                // bakterie i pełzacz z tempWorld są dodawane do odpowiednich komórek mainWorld.
                // Przed każdym creepersAndBacteriaAction wykonywanym dla wszystkich komórek mainWorld,
                // tworzony jest nowy, pusty tempWorld.
//-----
//kod do testowania - nie używany w standardowej symulacji
//-----
//        for (int i = 0; i < Init.SIZE_WORLD; i++) {
//            for (int j=0; j < Init.SIZE_WORLD; j++) {
//                mainWorld.setBacteriaNumAtPosition(5, i, j);
//            }
//        }
//        mainWorld.setOneCreeperAtPosition(5, 5);
//-----
//kod do testowania - nie używany w standardowej symulacji
//-----
mainWorld.sowBacteriaes(Init.START_NUM_BACT);
mainWorld.sowCreepers(Init.START_NUM_CREEPERS);
int numTact = 0, num, totalBactNum;
Init.outStream.println(" KOMÓRKI ŚWIATA");
Init.outStream.println("-----");
Init.outStream.println("Stan początkowy");
mainTest(mainWorld);
totallyCreepers.add(totalNum(mainWorld, "CREEPERS"));
totallyBacteria.add(totalNum(mainWorld, "BACTERIA"));
LinkedList<Cellule[] []> worldHistory = new LinkedList<Cellule[] []>();
worldHistory.add(mainWorld.cloneBoard());
Point injectionPoint = new Point((int) Math.round((Init.SIZE_WORLD-1)*Math.random()),
        (int) Math.round((Init.SIZE_WORLD-1)*Math.random()));
int liczbaBakterii = Init.INJECTED_BACT_NUM;
int liczbaPelzacz = Init.INJECTED_CREEPERS_NUM / Init.INJECT_FOR_TACTS;
boolean creeperOffsetOK = Init.INJECT_CREEPERS_OFFSET > 0;
boolean prematureEndOfSimulation = false;
while (numTact < Init.NUM_TACT && !prematureEndOfSimulation) {
    num = 0;
    while (num < Init.VEW_NUM_TACT && !prematureEndOfSimulation) {
        //Wstrzygnięcie bakterii
        if (Init.START_INJECTING_AT_TACT <= numTact
            && numTact < Init.START_INJECTING_AT_TACT+Init.INJECT_FOR_TACTS) {
            int liczba = liczbaBakterii/((Init.START_INJECTING_AT_TACT+Init.INJECT_FOR_TACTS) - numTact);
            mainWorld.board[injectionPoint.x][injectionPoint.y].addBactNum(liczba);
            liczbaBakterii-=liczba;
        }
        if (creeperOffsetOK) {
            //Wstrzygnięcie pełzaczów
            if ((Init.START_INJECTING_AT_TACT + Init.INJECT_CREEPERS_OFFSET <= numTact
                && numTact < Init.START_INJECTING_AT_TACT+Init.INJECT_FOR_TACTS + Init.INJECT_CREEPERS_OFFSET)
                for (int i = 0; i < liczbaPelzacz; i++) {
                    mainWorld.board[injectionPoint.x][injectionPoint.y].addCreeper(
                        new Creeper(injectionPoint.x, injectionPoint.y));
                }
        }
    }
    tempWorld = new World();
    mainWorld.creepersAndBacteriaAction (mainWorld, tempWorld);
    addNewBornOrganismsToMainWorldCellules (mainWorld, tempWorld);
    totallyCreepers.add(totalNum(mainWorld, "CREEPERS"));
}

```

```

        totalBactNum = totalNum(mainWorld, "BACTERIA");
        totallyBacteria.add(totalBactNum);
        if (totalBactNum > Init.BACT_NUM_LIMIT) prematureEndOfSimulation = true;
        num++;
        numTact++;
    }
    worldHistory.add(mainWorld.cloneBoard());
    Init.outStream.println("-----");
    Init.outStream.println("Przebieg " + numTact);
    mainTest(mainWorld);
}
Init.outStream.println();
Init.outStream.println(" LICZBA ORGANIZMÓW");
Init.outStream.println("-----");
Init.outStream.println(" BAKTERIE");
for (int i = 0; i < totallyCreepers.size(); i++) {
    Init.outStream.format("%8d%n", totallyBacteria.get(i));
    //Init.outStream.println(i + " " + totallyCreepers.get(i) + " " + totallyBacteria.get(i));
}
Init.outStream.println();
Init.outStream.println("-----");
Init.outStream.println(" PEŁZACZE");
for (int i = 0; i < totallyCreepers.size(); i++) {
    Init.outStream.format("%8d%n", totallyCreepers.get(i));
    //Init.outStream.println(i + " " + totallyCreepers.get(i) + " " + totallyBacteria.get(i));
}
if (prematureEndOfSimulation) {
    Init.outStream.println(
        "Sumaryczna liczba bakterii przekroczyła "
        + Init.BACT_NUM_LIMIT
        + " - komórki umierają/koniec symulacji.");
}
Init.outStream.println();
MainWindow mainWindow = new MainWindow(worldHistory);
}
}

class TimerTick implements ActionListener {
    private MainWindow parent;
    private int lastId;
    TimerTick(MainWindow parent, int lastId) {
        this.parent = parent;
        this.lastId = lastId;
    }
    @Override
    public void actionPerformed(ActionEvent evt) {
        if (!this.parent.jestKoniec()) {
            this.parent.nastepny();
        } else {
            this.parent.stopTimer();
        }
    }
}

class WorldGraphics extends JPanel {
    private Cellule[][] board;
    private int rozmiarKomorki;
    public WorldGraphics(Cellule[][] board, int rozmiarKomorki) {
        this.board = board;
        this.rozmiarKomorki = rozmiarKomorki;
    }
    public void updateState(Cellule[][] board, int rozmiarKomorki) {
        this.board = board;
        this.rozmiarKomorki = rozmiarKomorki;
        this.repaint();
    }
    public void updateState(Cellule[][] board) {
        this.updateState(board, this.rozmiarKomorki);
    }
    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        setBackground(new Color(36,38,48));
        for (int i = 0; i < Init.SIZE_WORLD; i++) {
            for (int j = 0; j < Init.SIZE_WORLD; j++) {
                Color cellColor = blendColors(getBactColor(this.board[j][i]),
                    getCreepColor(this.board[j][i]));
                g.setColor(cellColor);
                g.fillRect(i*rozmiarKomorki, j*rozmiarKomorki, rozmiarKomorki, rozmiarKomorki);
            }
        }
    }
}

```

```

        }
    }
}

private Color getBactColor(Cellule cell) {
    float alpha = (float)(cell.getBactNum()) / (float)(Init.MAX_INTENSITY_COUNT);
    if (alpha > 0) {
        alpha += (float) 0.2;
        if (alpha > 1) alpha = 1;
    }
    return new Color((float) 0.5, (float) 0.25, 1, alpha);
}

private Color getCreepColor(Cellule cell) {
    float alpha = (float)(cell.getCreepersNum()) / (float)(Init.MAX_INTENSITY_COUNT);
    if (alpha > 0) {
        alpha += (float) 0.2;
        if (alpha > 1) alpha = 1;
    }
    return new Color((float) 0.25, 1, (float) 0.5, alpha);
}

private Color blendColors(Color a, Color b) {
    int[] aCompArray = { a.getRed(), a.getGreen(), a.getBlue(), a.getAlpha() };
    int[] bCompArray = { b.getRed(), b.getGreen(), b.getBlue(), b.getAlpha() };
    int[] blendResult = new int[4];
    for (int i = 0; i < blendResult.length; i++) {
        blendResult[i] = aCompArray[3] > bCompArray[3] ? aCompArray[i] : bCompArray[i];
    }
    return new Color(blendResult[0], blendResult[1], blendResult[2], blendResult[3]);
}

class MainWindow extends JFrame implements ActionListener {
    private WorldGraphics worldGraphics;
    private JLabel label;
    private JButton controlButton1, controlButton2;
    private int windowSize;
    private LinkedList<Cellule[][]> board;
    ListIterator<Cellule[][]> iterator;
    private Cellule[][] biezacy;
    private Timer timer;
    public MainWindow(LinkedList<Cellule[][]> board) {
        this(board, 408);
    }
    public MainWindow(LinkedList<Cellule[][]> board, int windowSize) {
        super("Sztuczne życie");
        this.board = board;
        this.initSnapshot();
        this.windowSize = windowSize;
        JPanel controlPanel = new JPanel();
        this.controlButton1 = new JButton("Rozpocznij");
        this.controlButton2 = new JButton("Zatrzymaj");
        this.controlButton2.setEnabled(false);
        label = new JLabel("Stan poczatkowy");
        controlButton1.addActionListener(this);
        controlButton2.addActionListener(this);
        controlButton1.setActionCommand("START");
        controlButton2.setActionCommand("STOP");
        this.setBackground(new Color(36,38,38));
        controlPanel.add(controlButton1); controlPanel.add(label);
        controlPanel.add(controlButton2);
        this.setLayout(null);
        this.updateState();
        this.add(controlPanel);
        controlPanel.setBounds(0, windowSize + 16, windowSize, windowSize-256);
        setSize(this.windowSize, this.windowSize+128);
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                dispose();
            }
        });
        this.timer = new Timer(Init.TACT_TIME_DURATION, new TimerTick(this, this.board.size() - 1));
        this.timer.setInitialDelay(50);
        setVisible(true);
    }
    int getCurrentId() {
        if (this.board != null) {
            return board.indexOf(this.biezacy);
        }
        return -1;
    }
}

```

```

    }
    private void initSnapshot() {
        this.iterator = this.board.listIterator();
        if (this.iterator.hasNext()) {
            this.biezacy = this.iterator.next();
        }
    }
    public boolean jestPoczatek() {
        return this.getCurrentId() == 0;
    }
    public boolean jestKoniec() {
        return this.getCurrentId() == this.board.size() - 1;
    }
    private void poczatek() {
        while (this.iterator.hasPrevious()) {
            this.biezacy = this.iterator.previous();
        }
    }
    public void startTimer() {
        if (!this.jestPoczatek()) {
            this.poczatek();
        }
        if (this.timer != null) {
            this.timer.start();
        }
        this.controlButton1.setEnabled(false);
        this.controlButton2.setEnabled(true);
    }
    public void stopTimer() {
        if (this.timer.isRunning()) {
            this.timer.stop();
            this.controlButton1.setEnabled(true);
            this.controlButton2.setEnabled(false);
        }
    }
    public void nastepny() {
        if (this.iterator.hasNext()) {
            this.biezacy = this.iterator.next();
            this.updateState();
        }
    }
    private void updateState() {
        if (worldGraphics != null) {
            worldGraphics.updateState(this.biezacy);
        } else {
            this.worldGraphics = new WorldGraphics(this.biezacy, 41);
            this.add(worldGraphics);
            worldGraphics.setBounds(0, 0, this.windowSize, this.windowSize);
        }
        int currentId = getCurrentId();
        if (!jestPoczatek()) {
            int tactNum = currentId*Init.VIEW_NUM_TACT;
            this.label.setText("Bieżący takt: " + tactNum);
        } else {
            this.label.setText("Rozpocznij symulację");
        }
    }
    public void actionPerformed(ActionEvent e) {
        String command = e.getActionCommand();
        switch (command) {
            case "START":
                this.startTimer();
                break;
            case "STOP":
                this.stopTimer();
                break;
        }
    }
}
}

```