

Algorytm Sztuczne Życie

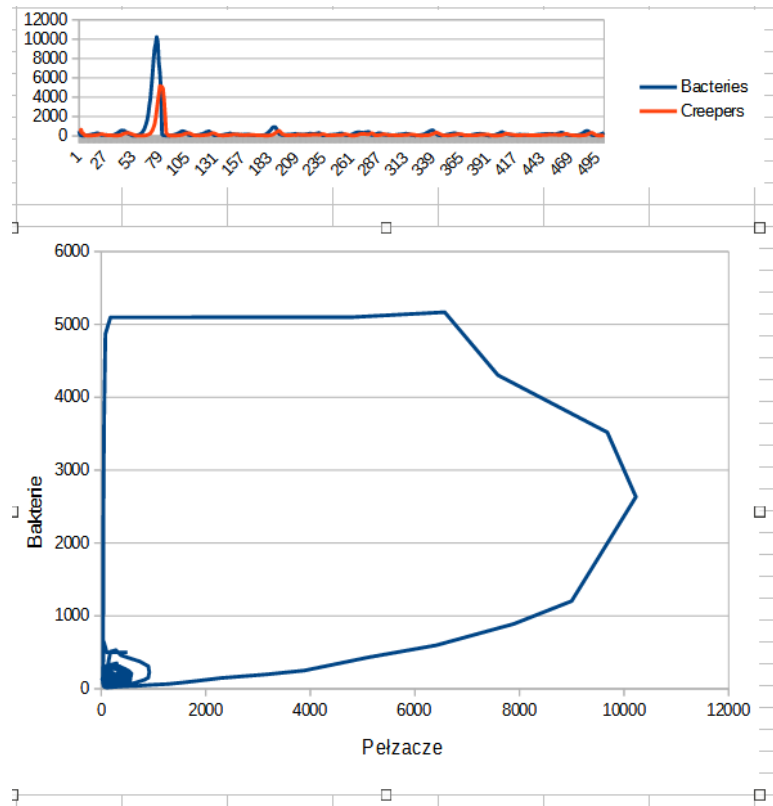
Dmytro Martsynenko

Sprawozdanie końcowe

Etap 1

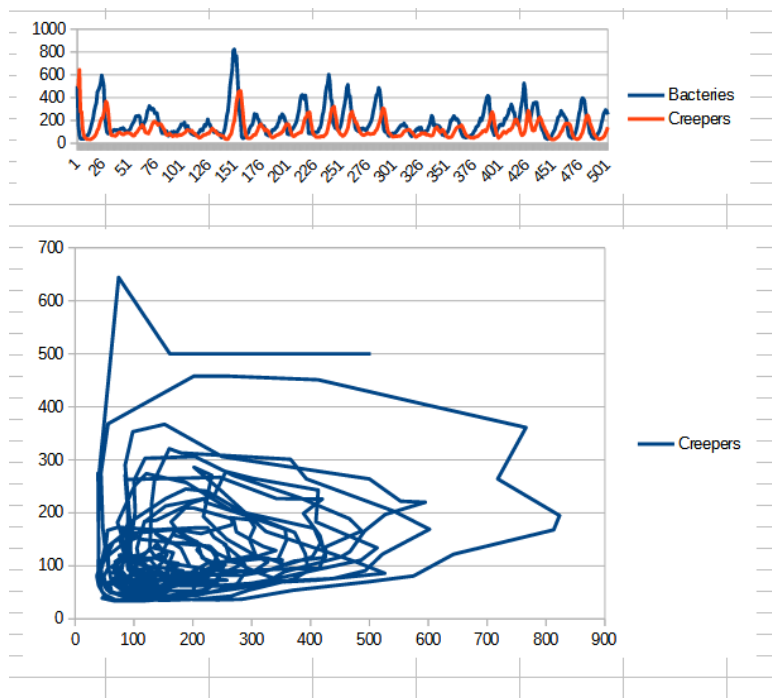
Celem pierwszego etapu jest zapoznanie się z przedstawionym algorytmem oraz badanie wpływu poszczególnych parametrów na dynamikę liczby bakterii oraz pelzaczy. Jaki stan systemu uważamy jako stabilny. Jest to taki stan, przy którym liczba bakterii nie przekracza 1 miliona, liczba każdego z organizmów nie spada do zera oraz wahania tych liczb są w przedziale kilkuset.

Wykresy



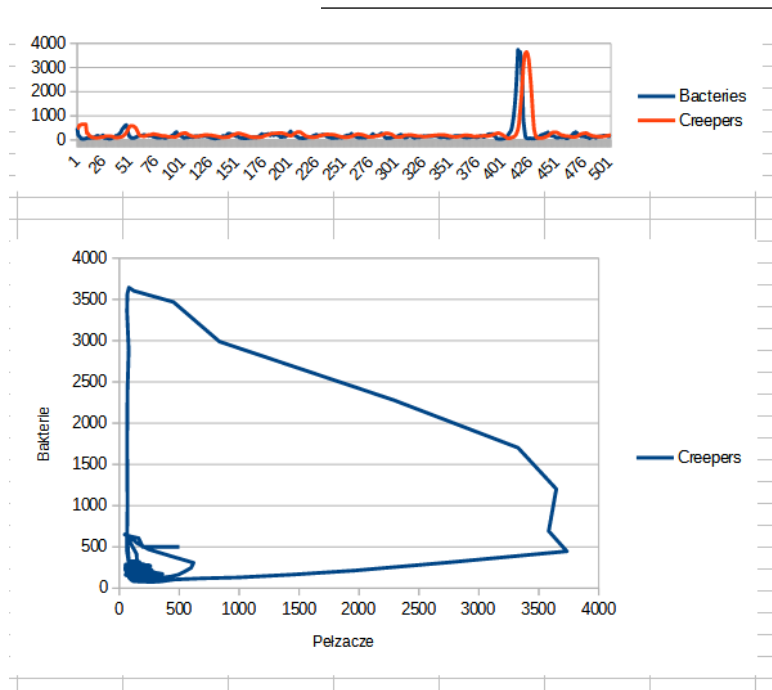
Wykres 1

Na pierwszym wykresie otrzymaliśmy skok liczby bakterii do ok. 10 tysięcy, ale został on zniwelowany przez pelzaczy, liczba który sięga ok. 5 tysięcy po kilkunastu taktów, dalej liczba obydwu organizmów trzyma się 300-500.



Wykres 2

Na drugim wykresie zobaczymy cykliczne wahania ze średnią amplitudą ok. 400 bakterii oraz 200 pelzaczy. Także można zobaczyć modulacje takich wahań z okresem 300 taktów.



Wykres 3

Na trzecim wykresie mamy skok liczby bakterii oraz pelzaczy do ok 4 tysięcy. Na początku są wahania 600-700 organizmów, od 76 taktu one maleją.

Wniosek

Zmniejszenie początkowej liczby organizmów nie gwarantuje mniejsze wahania w systemie, ponieważ mocno wpływają na dynamikę współczynniki rozmnażania i rozprzestrzeniania bakterii oraz, aktywność pelzaczy, która zależy od ograniczenia liczby urodzonych pelzaczy w takcie oraz od ograniczenia liczby zjedzonych bakterii przez pelzacza. Ponieważ reprezentacja bakterii oraz pelzaczy różni się, przeprowadzone badanie uwzględnia proces podbioru parametrów (znaleźć takie parametry dla *bakterii*, żeby były one równoważące parametrom dla *pelzaczy*).

Etap 2

Zadaniem drugiego etapu 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 niektórą ilość pelzaczy, co jest bardzo podobne do *podawania leczenia*. Dokonywać takiego leczenia należy przez niektóry 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.

Opis projektu

W celu implementacji zakażenia w symulowanym świecie, należy dokonać wyboru punktu wsztrzyknięcia. 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 pelzaczy do leczenia - **INJECTED_CREEPERS_NUM**
- Przez ile taktów od początku zakażenia zacząć leczenie - **INJECT_CREEPERS_OFFSET**

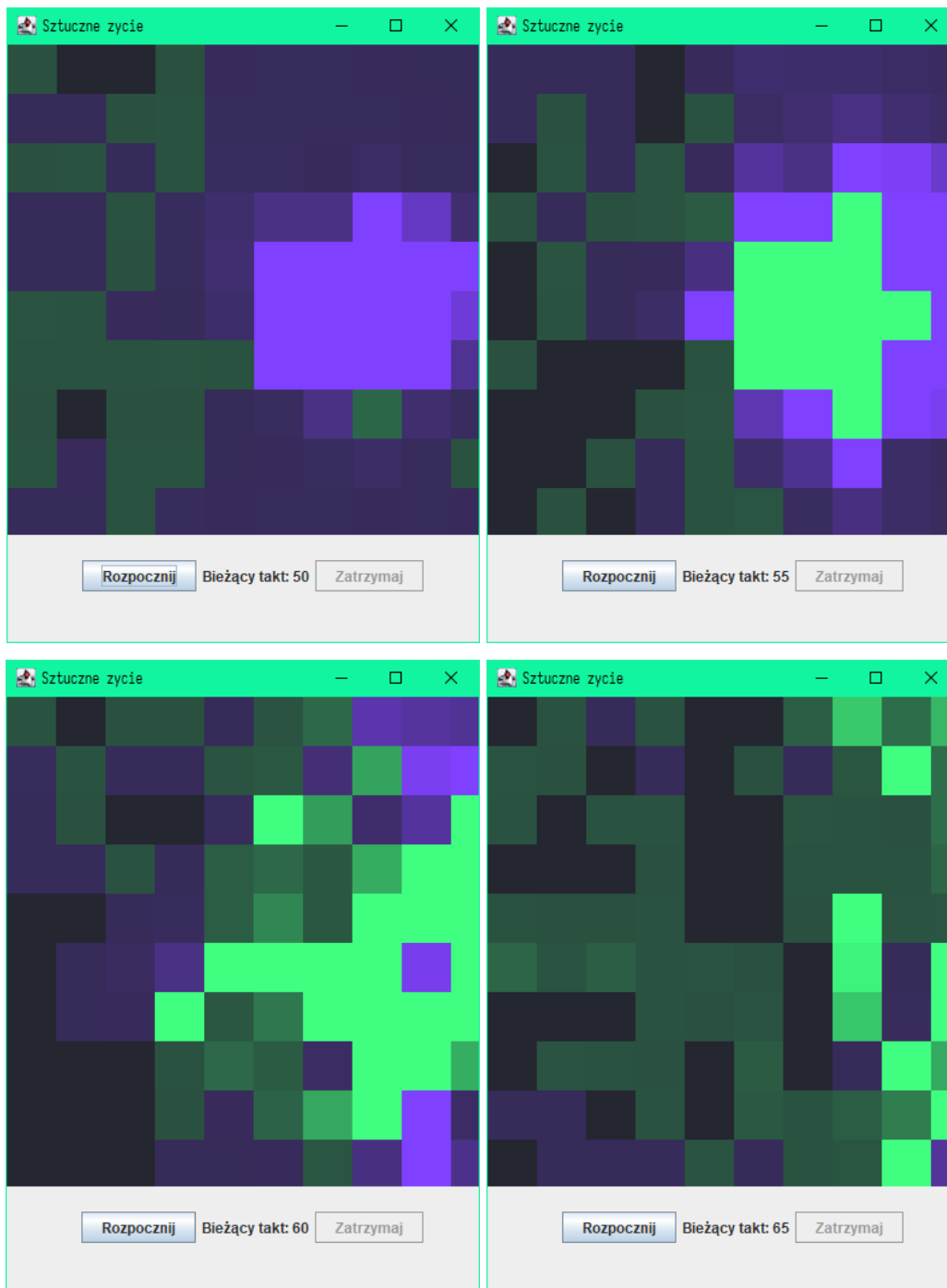
Końcowa wersja projektu przedstawiona w postaci graficznej, która da możliwość zobaczyć proces zakażenia, a następnie działanie procesu leczenia.



Rysunek 4: Wygląd okna symulacji

Na *Rysunku 4* przedstawiono wygląd okna symulacji po uruchomieniu programu. Fioletowym kolorem zaznaczono komórki świata, w których liczba bakterii jest *większa od liczby pływaczy*, 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 4* alpha jest znacznie mniejsza od 1.

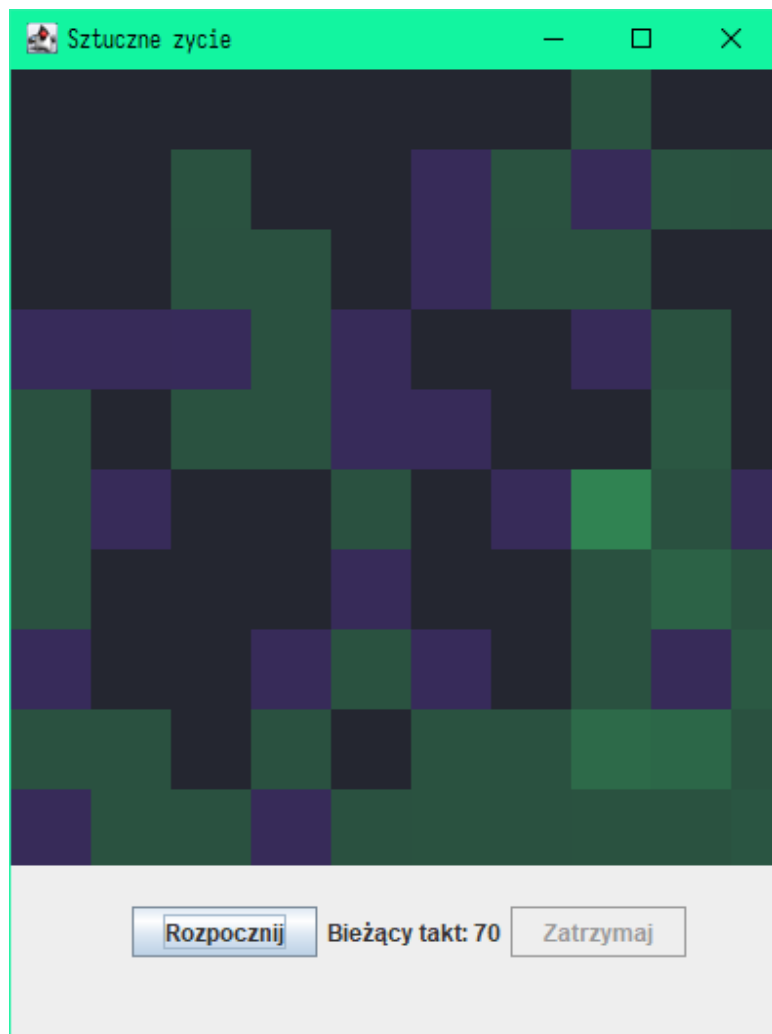


Rysunek 5: Przykład uruchomienia symulacji (takty 50-65)

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 **VEW_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 takcie 55 gwałtownie rośnie liczba pęłaczy, które wykonują funkcje obronną symulowanego organizmu. W takcie 60 widać delokalizację całego procesu,

zajmuje on prawie cały obszar świata, przewagę w liczbie trzymają pełzacze. W takcie 65 liczba obu organizmów zaczyna się zmniejszać się, leczenie już się zakończyło.



Rysunek 6: *Koniec procesów zakażenia oraz leczenia*

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

PARAMETRY URUCHOMIENIA

Całkowita liczba taktów

500

Wyświetlaj wyniki co

5 taktów

Początkowa liczba pełzaczy

500

| |
|---------------------------------------------------|
| Początkowa liczba bakterii |
| 500 |
| Liczba energii do urodzenia pełzacza |
| 3 |
| Zapas energii nowourodzonego pełzacza |
| 2 |
| Rezerwa energii zostawiana po urodz. pełzacza |
| 2 |
| Max. pełzaczy 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 pełzaczy przez |

10 takty/taktów

Liczba bakterii do wstrzyknięcia

32000

Liczba pełzaczy do wstrzyknięcia

4000

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 wstrzyknięcia
    static final int INJECTED_CREEPERS_NUM = 4000; //Liczba pełzaczy do wstrzyknięcia

    static final int START_INJECTING_AT_TACT = 40; //Rozpoczęcie wtrzyknięcia w takcie
    static final int INJECT_FOR_TACTS = 25; //Długość wstrzyknięcia 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 pełzaczy
    static final int START_NUM_BACT = 500; //początkowa liczba bakterii
    static final int CREEPER_ENERGY_PRO_LIFE = 3; //ilość energii potrzebna do urodzenia nowego pełzacza
    static final int CREEPER_INITIAL_ENERGY = 2; //zapas eneergii nowo urodzonego pełzacza
    static final int CREEPER_ENERGY_RESERVE = 2; //rezerwa energii zostawiana podczas rodzenia nowego p
        //potrzebna do przetrwania, gdy jest mało pożywienia
    static final int MAX_CREEPER_NUM_BORN_PER_TACK = 4; //maksymalna liczba pełzaczy rodzonych przez je
        //pełzacza w jednym takcie.

    // Liczba pełzaczy, które mogą się urodzić w jednym takcie jest ograniczona przez ilość energii pełz
    // odjęciu CREEPER_ENERGY_RESERVE. Pełzacz nie może zgromadzić zbyt dużo energii, ponieważ gdy tylk
    // przekracza poziom energii równy CREEPER_ENERGY_PRO_LIFE + CREEPER_ENERGY_RESERVE w następnym tak
    // rodzi co najmniej jednego pełzacza i jego poziom energii jest zmniejszany o CREEPER_ENERGY_PRO_L
    // na każdego urodzonego pełzacza.

    static final int MAX_BACT_EATEN_BY_CREEPER = 13; //Maksymalna liczba bakterii zjadanych przez pełzacz
        //w jednym takcie
    static final double BACT_MULTIPLICATION_RATE = 0.5; //współczynnik rozmnażania bakterii - tyle nowy
        //powstaje z jednej backterii w każdym takcie.
        //MOŻE PRZYJMOWAĆ WARTOŚCI UŁAMKOWE.
    static final double BACT_SPREAD_RATE = 0.6; //współczynnik rozprzestrzeniania nowo urodzonych bakte
```

```

//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 s

static final int TACT_TIME_DURATION = 425; //długość symulacji w millisekundach
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 pełzaczy 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 outputStream;
}

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 pełzaczy 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 pełzaczy 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 pełzaczy 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 pełzaczy, 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 pełzaczy ");
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. pełzaczy w 1 takcie ");
Init.outStream.println("-----");
Init.outStream.format(" %d", Init.MAX_CREEPER_NUM_BORN_PER_TACT);
Init.outStream.println("\n");
Init.outStream.println("-----");
Init.outStream.println(" Max. bakterii do zjedz. przez pełzacza w 1 takcie ");
Init.outStream.println("-----");
Init.outStream.format(" %d", Init.MAX_BACT_EATEN_BY_CREEPER);
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 takcie", 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 pełzaczy 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 pełzaczy 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 pełzaczy (tylko t

```

```

        // które przemieszczają się do sąsiednich komórek), aby nie zwiększały populac.
        // w niewylosowanych jeszcze komórkach.
        // Po zakończeniu akcji pełzaczy i bakterii dla wszystkich komórek,
        // bakterie i pełzacze z tempWorld są dodawane do odpowiednich komórek mainWorld.
        // Przed każdym creepersAndBacteriaAction wykonywanym dla wszystkich komórek m
        // 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.sowBacteries(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 liczbaPełzaczy = 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) {

        //Wstrzyknięcie 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) -

```

```

        mainWorld.board[injectionPoint.x][injectionPoint.y].addBactNum(liczba);
        liczbaBakterii-=liczba;
    }

    if (creeperOffsetOK) {
        //Wstrzyknięcie pełzaczy
        if (Init.START_INJECTING_AT_TACT + Init.INJECT_CREEPERS_OFFSET <= numTact
            && numTact < Init.START_INJECTING_AT_TACT+Init.INJECT_FOR_TACTS + Init.INJECT
                for (int i = 0; i < liczbaPelzaczy; i++) {
                    mainWorld.board[injectionPoint.x][injectionPoint.y].addCreeper(new Creeper(in
                }

        }
    }

    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 "
    + " - 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]), getCreeperColor(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 getCreeperColor(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 zycie");
    }
}

```

```

        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 początkowy");

        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 jestPoczątek() {
        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.VEW_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;
    }
}
```