Systemy operacyjne 2 Projekt - Wizualizacja przyrostu ilości przypadków koronawirusa w poszczególnych krajach.

Tomasz Dyśko, 241321
Aleksandra Ziobrowska, 241129
Prowadzący - dr inż. Dominik Żelazny

Kwiecień 2020

1 Wstęp

Sprawozdanie z projektu na przedmiot Systemy operacyjne 2. W ramach projektu wykonaliśmy program który symuluje zarażanie się wirusem przez kontakt. Odbywa się to na zadanej planszy i z ustalonym w kodzie prawdopodobieństwem. Program wykorzystuje wiele wątków które reprezentowane są przez osoby - implementacje klasy **Person**. Wspólnym elementem są 'kafelki' symbolizujące miejsce w przestrzeni - klasa **Tile**.

2 Implementacja

Mogłyby się pojawić wyścigi w przypadku, gdy dwie osoby chciałyby stanąć na tym samym kafelku, zostało to zabezpieczone poprzez zastosowanie semaphorów.

2.1 Klasa Tile

Klasa ta może być używana przez wiele osób (czyli wątków) więc wymagana jest kontrola zasobów. Aby to osiągnąć użyliśmy klasy **Semaphore** z pakietu:
import java.util.concurrent.Semaphore; Zawiera ona metody do ustawiania osoby na kafelku czyli na sobie, zwalniania kafelka, oraz sprawdzania czy jest możliwe postawienie osoby na kafelku.

Poniżej przedstawiony jest kod klasy **Tile**, użycie semaphora jest widoczne np. w funkcji **setVisitor(Person visitor)**.

```
package model;

import javax.swing.*;
import java.awt.*;
import java.util.concurrent.Semaphore;

public class Tile extends JPanel {

    private static final Color HEALTHY_COLOR = Color.GREEN;
    private static final Color INFECTED_COLOR = Color.RED;
    private static final Color NO_PERSON_COLOR = Color.WHITE;

    private Person visitor;
    private JLabel text;

    private Semaphore semaphore;

    public Tile(Person visitor) {
        this.semaphore = new Semaphore(1, true);
        this.visitor = visitor;
        initUI();
    }

    public Tile() {
```

```
        this.semaphore = new Semaphore(1, true);
        this.visitor = null;
        initUI();
    }

    private void initUI() {
        this.setBorder(BorderFactory.createLineBorder(Color.black));
        this.setBackground(NO_PERSON_COLOR);

        this.text = new JLabel("");
        if (this.visitor != null) {
            this.text.setText(this.visitor.getPersonName());
        }

        this.add(this.text);
    }

    public boolean isFree() {
        return visitor == null;
    }

    public void free() {
        this.visitor = null;
        this.text.setText("");
        this.setBackground(NO_PERSON_COLOR);
        this.semaphore.release();
    }

    public Person getVisitor() {
        return visitor;
    }

    public void setVisitor(Person visitor) {
        try {
            this.semaphore.acquire();
            this.visitor = visitor;
            this.text.setText(visitor.getPersonName());
            if (visitor.getStatus().equals(HealthStatus.HEALTHY)) {
                this.setBackground(HEALTHY_COLOR);
            } else if (visitor.getStatus().equals(HealthStatus.INFECTED)) {
                this.setBackground(INFECTED_COLOR);
            }
        }
        catch (InterruptedException ex) {
            ex.printStackTrace();
```

```
        }
    }
}
}
```

2.2 Klasa Person

Klasa **Person** reprezentuje osobę która przemieszcza się w przestrzeni czyli w naszej aplikacji po kafelkach. Klasa ta dziedziczy po klasie Thread i startuje nowy wątek w nadpisanej metodzie **run()**.

```
@Override
    public void run() {
        while (true) {
            try {
                Thread.sleep(this.sleepTime);
                tryToMove();
                this.country.updateSummary();
            }
            catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
```

Osoba losowo porusza się w pionie lub poziomie, dlatego w jej polach jest obiekt klasy **Random**. Oto pozostałe pola klasy:

```
    private static final int[] MOVE_DIRECTION = {-1, 1};

    private String personName;
    private HealthStatus status;
    private int positionX;
    private int positionY;
    private Country country;
    private int sleepTime;
    private Random random;
```

Metoda **tryToMove()** odpowiedzialna za poruszanie po kafelkach:

```
    private void tryToMove() {
        int moveX = MOVE_DIRECTION[random.nextInt(2)];
        int moveY = MOVE_DIRECTION[random.nextInt(2)];

        int newX = (positionX + moveX) % country.getCountryWidth();
        if (newX < 0)
            newX = 0;

        int newY = (positionY + moveY) % country.getCountryHeight();
```

```
        if (newY < 0)
            newY = 0;

        if (this.country.getTiles()[newX][newY].isFree()) {
            this.country.movePerson(this, newX, newY);
            this.positionX = newX;
            this.positionY = newY;
        }
    }
```

2.3 Klasa Country

Klasa ta jest jednocześnie rozszerzeniem klasy JPanel i elementem łączącym klasy **Tale** i **Person**. Odpowiada za inicjalizację obiektów, a więc także rozpoczęcie działania wątków. Dodatkowo udostępnia API do poruszania osobami:

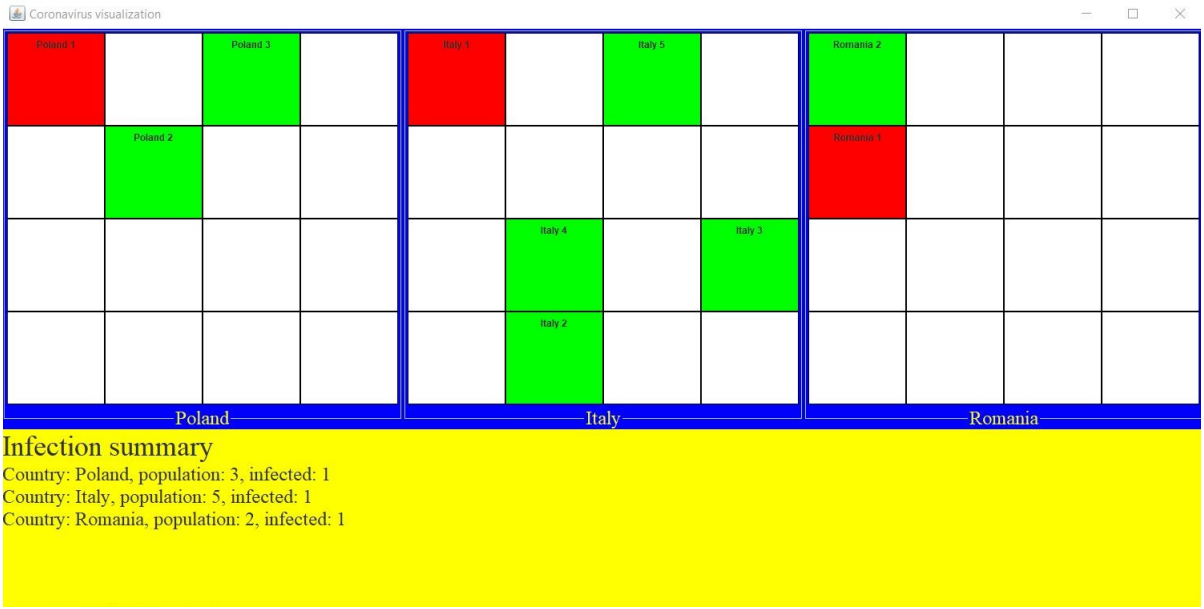
```
 private List<Person> preparePeople(int populationNumber) {
        List<Person> people = new ArrayList<>();
        for (int i = 0; i < populationNumber; i++) {
            int initialX = r.nextInt(this.tiles.length);
            int initialY = r.nextInt(this.tiles[0].length);

            while (!tiles[initialX][initialY].isFree()) {
                initialX = r.nextInt(this.tiles.length);
                initialY = r.nextInt(this.tiles[0].length);
            }

            Person p = new Person(this.name + "␣" + (i + 1), initialX,
                initialY, this);
            if (i == 0)
                p.setStatus(HealthStatus.INFECTED);
            tiles[initialX][initialY].setVisitor(p);
            people.add(p);
        }
        return people;
    }

public int getInfectedNumber() {
        int i = 0;
        for (Person p : this.people) {
            if (p.getStatus().equals(HealthStatus.INFECTED))
                i++;
        }
        return i;
    }
```

```
public void updateSummary() {
        this.summaryPanel.updateSummary(this.name, this.people.size(),
            getInfectedNumber());
    }

    public void movePerson(Person p, int destinationX, int destinationY) {
        this.tiles[p.getPositionX()][p.getPositionY()].free();

        if (isInfectedPersonNearby(destinationX, destinationY)) {
            int infectionProbability = r.nextInt(100);
            if (infectionProbability > INFECTION_PROBABILITY_CLOSE) {
                p.setStatus(HealthStatus.INFECTED);
            }
        }

        this.tiles[destinationX][destinationY].setVisitor(p);
    }
```

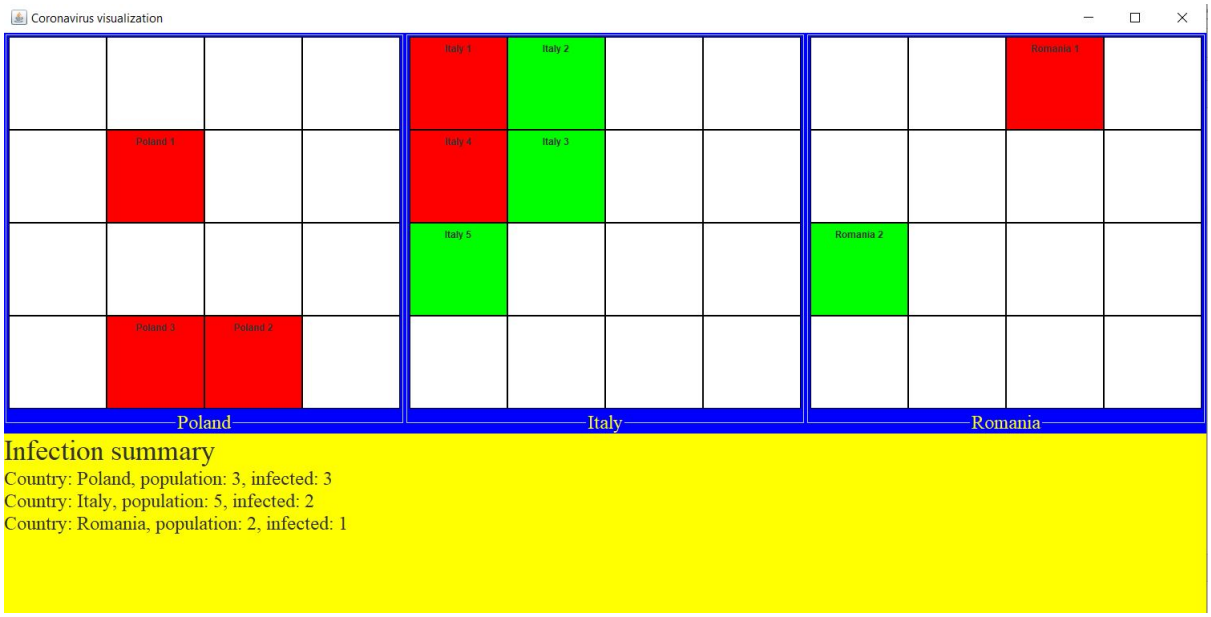
2.4 Klasa SummaryPanel

Jest to klasa odpowiedzialna za podsumowanie widoczne pod mapami.

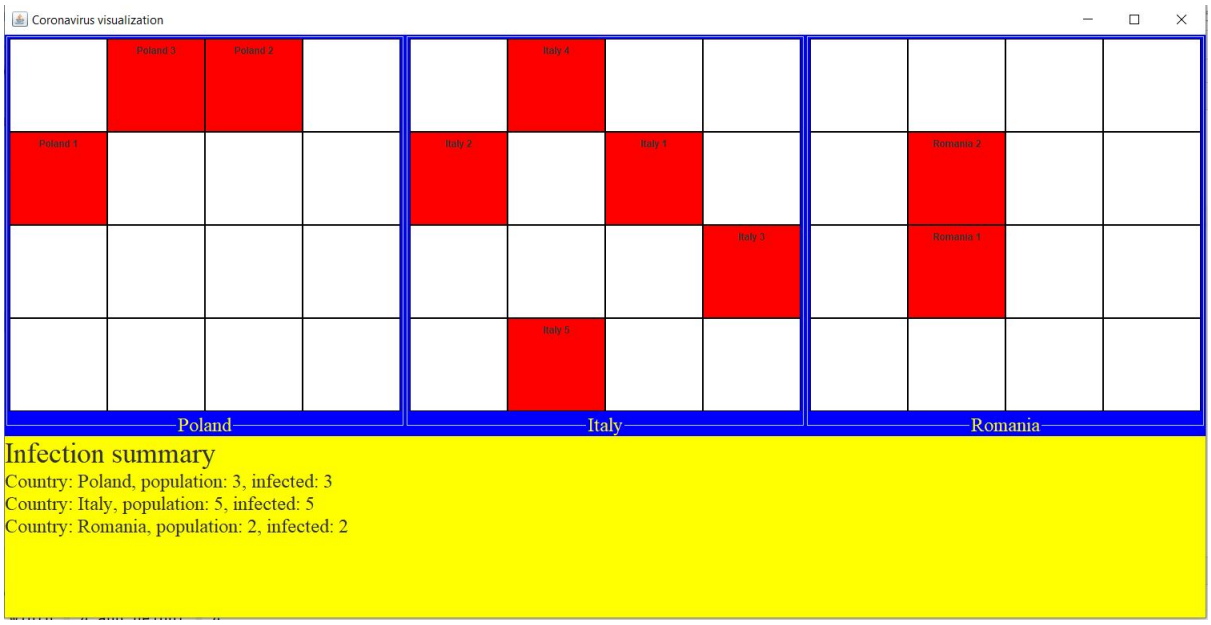
3 Wygląd aplikacji



Rysunek 1: Screen prezentujący wstępny wygląd map



Rysunek 2: Screen pokazujący mapę po kilku wykonanych ruchach



Rysunek 3: Screen mapy po kolejnym okresie czasu

4 Wnioski i podsumowanie

Wykonując projekt nauczyliśmy się bezkonfliktowego zarządzania zasobami i procesami w wielowątkowym programie w języku JAVA. Aby to osiągnąć niezbędne były klasy **Thread** oraz **Semaphore**. Znając takie rozwiązania byliśmy w stanie dość łatwo zamodelować jakiś proces ze świata rzeczywistego. Jak wiadomo w prawdziwym życiu bardzo trudno jest znaleźć problem, którego modelowanie nie będzie wymagało wielu wątków. Język JAVA udostępnia do tego bardzo dobre narzędzia jednocześnie dość łatwo jest graficznie pokazać aktualny status. Zrobiliśmy to z użyciem klasy **JPanel**.