Programowanie dla fizyków - projekt
Graficzne przedstawienie rozprzestrzeniania się wirusa Covid-19

Gabriela Białoskórska, Mikołaj Knysak, Ignacy Tekieli

Czerwiec 2020

1 Cel pracy

Celem niniejszej pracy jest ukazanie rozprzestrzeniania się wirusa Covid-19 w Polsce w sposób graficzny, tak aby zwiększać świadomość społeczną dotyczącą podejmowanych na terenie kraju działań prewencyjnych. Utworzyliśmy ku temu dwa programy napisane w języku Python, jednak opierające się na odmiennych modelach: SEIRD oraz Monte-Carlo. Umożliwi nam to również porównanie wspomnianych metod pod kątem ich dokładności oraz wiarygodności.

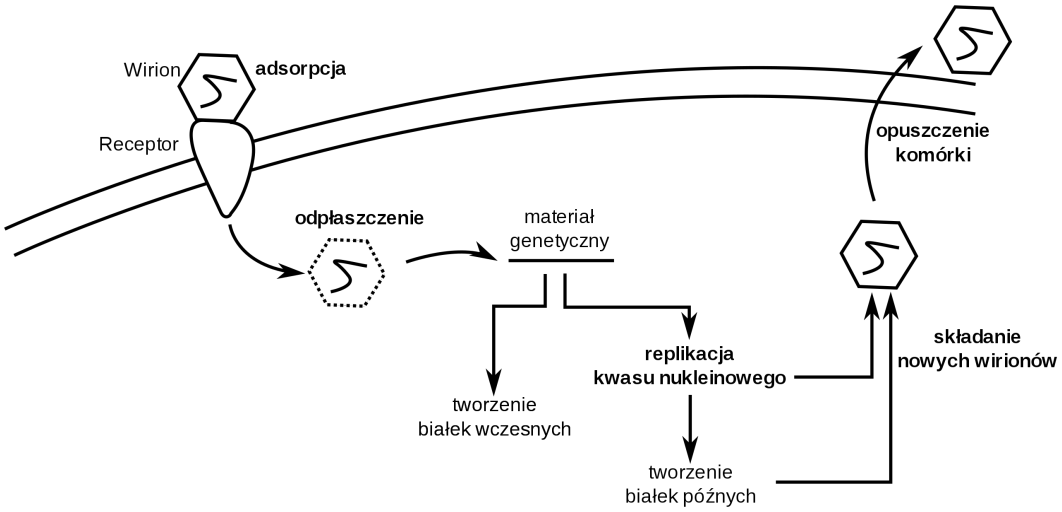
2 Wstęp teoretyczny

2.1 Wirusy

Wirusy, choć towarzyszą ludzkości od zarania dziejów, pozostają dla nas wciąż bytami enigmatycznymi. Przede wszystkim stają się bytami wyłącznie w kontakcie z komórkami nosiciela – poza nim nie są w stanie się rozmnażać i po określonym czasie inaktywują się. W związku z tym nie są one klasyfikowane jako organizmy żywe, choć w skład ich budowy również wchodzą białka (aminokwasy) – nie jest to jednak budowa komórkowa.

Wirusy atakują organizm ludzki poprzez wszczepienie swojego kwasu nukleinowego w DNA komórki nosiciela. Wówczas zaczyna ona wytwarzać białka oraz powielać materiał wirusa (DNA lub RNA) czyli elementy składowe nowych wirionów, zdolnych do infekowania kolejnych komórek.

Choroby wirusowe są grupą schorzeń, przeciwko której wciąż nie dysponujemy lekami. Leczenie ma zazwyczaj charakter objawowy, ale terapia zwalczająca wirusy nie istnieje. Organizm może jedynie nabyć odporność na dany typ wirusa poprzez wejście z nim w bezpośredni kontakt – na tej właśnie zasadzie działają szczepienia. Istnieje jednak wiele wirusów, przeciwko którym nie posiadamy szczepionek – jednym z nich jest Covid-19.



Rysunek 1: Schematyczne przedstawienie ataku wirusa na organizm żywy

Na ten moment wspomniany koronawirus jest określany mianem pandemii. Oznacza to, że występuje w tym samym czasie w kilku krajach znajdujących się na tym samym kontynencie lub różnych kontynentach (w przeciwieństwie do epidemii, która występuje w określonym czasie na danym terenie). Rozwojowi pandemii sprzyja jego wysoka zaraźliwość, brak nabytej odporności wśród ludzi oraz możliwość jego bezobjawowego przechodzenia.

2.2 Modelowanie komputerowe

W dzisiejszych czasach głównym narzędziem walki z nieznanymi dotąd wirusami (zatem takimi, przeciwko którym nie dysponujemy szczepionką) jest wcześniejsze określanie ich możliwego przebiegu w taki sposób, aby można było podjąć działania prewencyjne. Technologie którymi obecnie dysponujemy umożliwiają nam dokładny opis procesu rozprzestrzeniania się epidemii, jednak ich podwalin należy szukać już w XVIII wieku. Wówczas pierwszy opis matematyczny podobnego zjawiska zaprezentował na Akademii Francuskiej fizyk Daniel Bernoulli. Był on oparty na równaniach różniczkowych, w których współczynniki charakteryzowały właściwości choroby zakaźnej.

Rozwój technologii informacyjnej umożliwił wykonywanie skomplikowanych obliczeń numerycznych i badań symulacyjnych, co przyczyniło się do znacznego postępu w dziedzinie modelowania rozprzestrzeniania się epidemii. Dzięki nim możemy w przybliżeniu określać liczbę nowych zachorowań, zasięg, oraz liczbę osób zmarłych w kolejnych dniach, co pozwala oszacować ilość koniecznych do walki z epidemią środków medycznych, a także opracować scenariusz walki z nią, nim będziemy dysponować skuteczną szczepionką.

Poniżej przedstawiamy dwa typy modeli, które wykorzystaliśmy do stworzenia projektu.

Model SEIRD

Model SEIRD *(ang. Susceptible-Infected-Exposed-Recovered-Dead)* wykorzystuje w swoim działaniu układ równań różniczkowych, reprezentujący pięć grup: osoby podatne na zarażenie (susceptible), osoby zarażone (infected), osoby narażone na zachorowanie (exposed), osoby ozdrowiałe (recovered) oraz osoby zmarłe (dead). Układ ten wyraża się w następujący sposób:

$$\frac{dS}{dt} = \frac{\beta}{N} SI \tag{1}$$

$$\frac{dE}{dt} = \frac{\beta}{N} SI - \alpha E \tag{2}$$

$$\frac{dI}{dt} = \alpha E - (\gamma_R + \gamma_D) I \tag{3}$$

$$\frac{dR}{dt} = \gamma_R I \tag{4}$$

$$\frac{dD}{dt} = \gamma_D I \tag{5}$$

gdzie:

N – całkowita populacja
β – współczynnik zachorowań
γ – parametr określający osoby zainfekowane, które nabyły odporność
α – czas inkubacji

Jak można zauważyć, założenia dotyczące rozprzestrzeniania się i okresu inkubacji wchodzą w parametry powyższego modelu.

Model Monte-Carlo

Działanie metody Monte-Carlo opiera się na modelowaniu zmiennych pseudolosowych – losowanie dokonywane jest zgodnie ze znanym rozkładem. Struktura algorytmu obliczeń nie jest skomplikowana, wobec czego model ten stosuje się w przypadkach, dla których wystarcza znajomość wyniku z niewielką dokładnością. Dokładność ta zależy od liczby sprawdzeń i jakości stosowanego generatora liczb pseudolosowych.

Zazwyczaj przy pomocy metody Monte-Carlo tworzy się program realizujący jedno zdarzenie losowe, powtarzane następnie N-razy, aby eksperyment był niezależny od poprzednich. Uzyskane wyniki uśrednia się, stąd alternatywna nazwa „metoda prób statystycznych"

3 Opis projektu

3.1 Model SEIRD

Pracę nad powyższym modelem rozpoczęliśmy od opisania układu równań różniczkowych:

```
def deriv(y, t, n, beta, gamma, delta, alpha, rho):
    S, E, I, R, D = y

    dSdt = -beta(t) * S * I / n
    dEdt = beta(t) * S * I / n - delta * E
    dIdt = delta * E - (1 - alpha) * gamma * I - alpha * rho * I
    dRdt = (1 - alpha) * gamma * I
    dDdt = alpha * rho * I

    return dSdt, dEdt, dIdt, dRdt, dDdt
```

Następnie zainicjowaliśmy parametry wirusa oraz funkcję zmieniającą je w zależności od czasu:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint

TOTAL_POPULATION = 38000000
S0, E0, I0, R0, D0 = (TOTAL_POPULATION - 100, 100, 0, 0, 0)
INCUBATION_PERIOD, INFECTION_DURATION, LETHALITY = (4.7, 5.8, 0.034)
DEATH_DELAY, LOCKDOWN_DELAY = (9, 14)


def R_0(t):
    return 6.1 if t < LOCKDOWN_DELAY else 1.2


def beta(t):
    return R_0(t) * gamma
```

Część kodu odpowiedzialna za zamianę parametrów wirusa na symbole wykorzystane w równaniach różniczkowych modelu SEIRD oraz rozwiązanie wspomnianych równań wygląda następująco:

```
if __name__ == '__main__':
    y0 = S0, E0, I0, R0, D0
    t = np.linspace(0, 300)
    n = TOTAL_POPULATION
    alpha = LETHALITY
    gamma = 1 / INFECTION_DURATION
    delta = 1 / INCUBATION_PERIOD
    rho = 1 / DEATH_DELAY

    result = odeint(deriv, y0, t, args=(n, beta, gamma, delta, alpha, rho))
    S, E, I, R, D = result.T

    plotseird(t, S, E, I, R, D, LOCKDOWN_DELAY)
```

Tworzenie wykresu:

```
def plotseird(t, S, E, I, R, D, L):
    _, ax = plt.subplots(1, 1, figsize=(10, 4))
    plt.title("Lockdown after {} days".format(L))

    # ax.plot(t, S, 'b', alpha=0.7, linewidth=2, label='Susceptible')
    ax.plot(t, E, 'y', alpha=0.7, linewidth=2, label='Exposed')
    ax.plot(t, I, 'r', alpha=0.7, linewidth=2, label='Infected')
    ax.plot(t, R, 'g', alpha=0.7, linewidth=2, label='Recovered')
    ax.plot(t, D, 'k', alpha=0.7, linewidth=2, label='Dead')

    ax.set_xlabel('Time (days)')

    legend = ax.legend(borderpad=2.0)
    legend.get_frame().set_alpha(0.5)
    plt.show()
```

3.2 Model Monte-Carlo

Na wstępie zainicjowaliśmy parametry symulacji:

```
# Left click to next frame, right click to plot a graph

import tkinter as tk
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt


INITIAL_POPULATION_DENSITY, INITIAL_INFECTED_DENSITY = (0.75, 0.0005)
SPREAD_DIRECT_PROB, SPREAD_DIAGONAL_PROB = (0.86, 0.88)
RESTORATION_RATE, LETHALITY = (0.08, 0.006)
NEW_INFECTION_PROB, NEW_HEALTHY_PROB = (0.0001, 0.0001)
HEALTHY, INFECTED, RECOVERED, DEAD, EMPTY = ('green', 'red', 'pink',
                                             'black', 'white')
```

W poniższym fragmencie kodu znajduje się klasa okna z symulacją, metoda umożliwiająca stworzenie pojedynczej komórki w całości siatki oraz metoda przekształcająca cały obszar obrazu w siatkę drobnych komórek, reprezentujących popuację:

```python
class PandemicWindow:
    def __init__(self, master, canvas_dimensions=(960, 960),
                 box_count=(120, 120)):
        self.canvas_width, self.canvas_height = canvas_dimensions
        self.box_count = box_count
        self.box_dimensions = (canvas_dimensions[0] // box_count[0],
                               canvas_dimensions[1] // box_count[1])
        self.current_frame = 1
        self.data = {
            'Frame': [0],
            'Healthy': [0],
            'Infected': [0],
            'Recovered': [0],
            'Dead': [0],
            'Total': [0]
            }

        self.master = master
        self.frame = tk.Frame(self.master)
        self.canvas = tk.Canvas(self.master, width=self.canvas_width,
                                height=self.canvas_height)

        self.canvas.bind('<Button-1>', self.next_frame)
        self.canvas.bind('<Button-2>', self.plot_data)
        self.canvas.pack()
        self.frame.pack()

        self.grid = self.init_grid()
        self.data_update()

    def init_box(self, coords, color):
        """ Create rectangle on canvas. """
        x, y = coords
        width, height = self.box_dimensions
        x2 = x + width
        y2 = y + height

        box = self.canvas.create_rectangle(x, y, x2, y2, fill=color)
        return box

    def init_grid(self):
        """ Turn canvas into grid of boxes. """
        horizontal_count, vertical_count = self.box_count
        width, height = self.box_dimensions

        grid = [[None for _ in range(vertical_count)]
                for _ in range(horizontal_count)]

        for x in range(horizontal_count):
            for y in range(vertical_count):
                coords = (x * width, y * height)
                if np.random.rand() <= INITIAL_POPULATION_DENSITY:
                    if np.random.rand() <= INITIAL_INFECTED_DENSITY:
                        grid[x][y] = self.init_box(coords, INFECTED)
                    else:
                        grid[x][y] = self.init_box(coords, HEALTHY)
                else:
                    grid[x][y] = self.init_box(coords, EMPTY)
        return grid
```

Metoda Monte-Carlo umieszcza na siatce zdrowych ludzi oraz kilku pierwszych zarażonych z użyciem liczb pseudolosowych oraz parametrów początkowej gęstości zaludnienia i szany wystąpienia zarażonej osoby na początku symulacji.

Zapisywanie w konsoli stanu symulacji w każdej pojedynczej klatce wygląda następująco:

```
def get_status(self):
    """ Return status of the grid. """
    horizontal_count, vertical_count = self.box_count
    healthy = infected = recovered = dead = 0

    for x in range(horizontal_count):
        for y in range(vertical_count):
            status = self.canvas.itemcget(self.grid[x][y], 'fill')

            if status == HEALTHY:
                healthy += 1
            elif status == INFECTED:
                infected += 1
            elif status == RECOVERED:
                recovered += 1
            elif status == DEAD:
                dead += 1
    return healthy, infected, recovered, dead

def data_update(self):
    h, i, r, d = self.get_status()
    t = i + r + d

    self.data['Frame'].append(self.current_frame)
    self.data['Healthy'].append(h)
    self.data['Infected'].append(i)
    self.data['Recovered'].append(r)
    self.data['Dead'].append(d)
    self.data['Total'].append(t)

    print('{} Healthy, {} infected, {} reovered and {} dead '
          'in frame {}, {} cases in total'
          .format(h, i, r, d, self.current_frame, t))
```

Generowanie kolejnych klatek symulacji jest wywoływane lewym przyciskiem myszy. Zmienia stan każdej komórki biorąc pod uwagę szansę na zarażenie, komórki znajdujące się dookoła oraz szansę na wyzdrowienie lub śmierć komórki zarażonej.

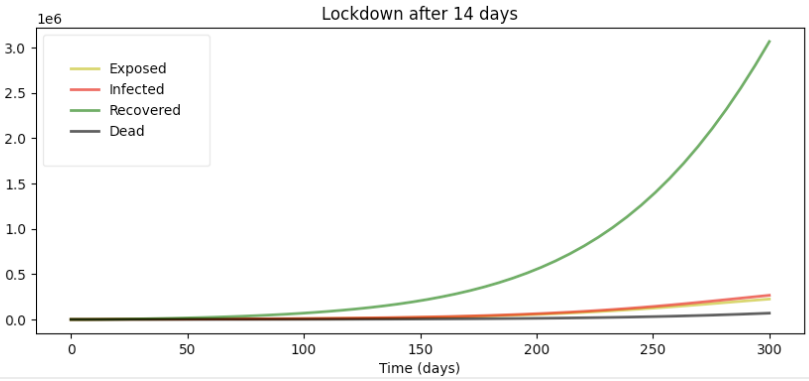
```
def next_frame(self, _):
    """ Generate next frame of the simulation. """
    self.current_frame += 1

    horizontal_count, vertical_count = self.box_count
    new_grid = [[None for _ in range(vertical_count)]
                for _ in range(horizontal_count)]

    for x in range(horizontal_count):
        for y in range(vertical_count):
            current_box = self.grid[x][y]
            def status(box): return self.canvas.itemcget(box, 'fill')

            # This is dumb. There has to be a better way of doing this.
            if status(current_box) == HEALTHY:
                if (x > 0 and y > 0 and
                        status(self.grid[x - 1][y - 1]) == INFECTED and
                        np.random.rand() <= SPREAD_DIAGONAL_PROB):
                    new_grid[x][y] = INFECTED
                elif (y > 0 and status(self.grid[x][y - 1]) == INFECTED and
                        np.random.rand() <= SPREAD_DIRECT_PROB):
                    new_grid[x][y] = INFECTED
                elif (y > 0 and x < horizontal_count - 1 and
                        status(self.grid[x + 1][y - 1]) == INFECTED and
                        np.random.rand() <= SPREAD_DIAGONAL_PROB):
                    new_grid[x][y] = INFECTED
                elif (x > 0 and status(self.grid[x - 1][y]) == INFECTED and
                        np.random.rand() <= SPREAD_DIRECT_PROB):
                    new_grid[x][y] = INFECTED
                elif (x < horizontal_count - 1 and
                        status(self.grid[x + 1][y]) == INFECTED and
                        np.random.rand() <= SPREAD_DIRECT_PROB):
                    new_grid[x][y] = INFECTED
                elif (x > 0 and y < vertical_count - 1 and
                        status(self.grid[x - 1][y + 1]) == INFECTED and
                        np.random.rand() <= SPREAD_DIAGONAL_PROB):
                    new_grid[x][y] = INFECTED
                elif (y < vertical_count - 1 and
                        status(self.grid[x][y + 1]) == INFECTED and
                        np.random.rand() <= SPREAD_DIRECT_PROB):
                    new_grid[x][y] = INFECTED
                elif (x < horizontal_count - 1 and
                      y < vertical_count - 1 and
                      status(self.grid[x + 1][y + 1]) == INFECTED and
                      np.random.rand() <= SPREAD_DIAGONAL_PROB):
                    new_grid[x][y] = INFECTED
                elif np.random.rand() <= NEW_INFECTION_PROB:
                    new_grid[x][y] = INFECTED
            elif status(current_box) == INFECTED:
                if np.random.rand() <= LETHALITY:
                    new_grid[x][y] = DEAD
                elif np.random.rand() <= RESTORATION_RATE:
                    new_grid[x][y] = RECOVERED
            elif np.random.rand() <= NEW_HEALTHY_PROB:
                new_grid[x][y] = HEALTHY

    for x in range(horizontal_count):
        for y in range(vertical_count):
            self.canvas.itemconfig(self.grid[x][y], fill=new_grid[x][y])

    self.data_update()
```

Rysowanie wykresu stanu symulacji do klatki obecnej po naciśnięciu prawego przycisku myszy oraz inicjacja i rozpoczęcie działania programu wygląda następująco:

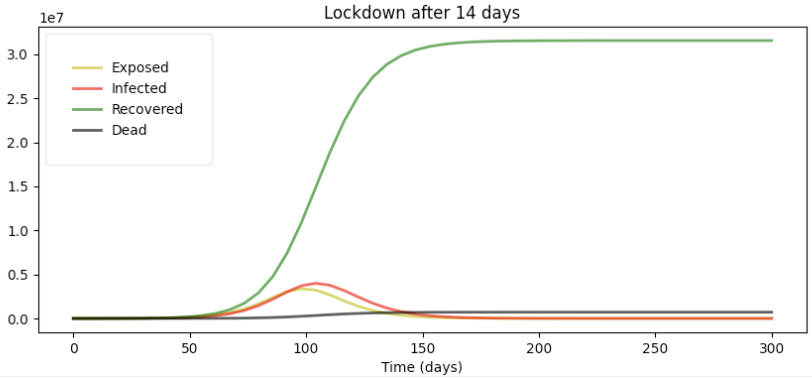
```
    def plot_data(self, _):
        df = pd.DataFrame(self.data)

        _, ax = plt.subplots(1, 1, figsize=(10, 4))
        df.plot(kind='line', x='Frame', y='Healthy', color='green', ax=ax)
        df.plot(kind='line', x='Frame', y='Infected', color='red', ax=ax)
        df.plot(kind='line', x='Frame', y='Recovered', color='pink', ax=ax)
        df.plot(kind='line', x='Frame', y='Dead', color='black', ax=ax)
        df.plot(kind='line', x='Frame', y='Total', color='blue', ax=ax)
        plt.show()


if __name__ == '__main__':
    root = tk.Tk()
    forest = PandemicWindow(root)
    root.mainloop()
```

4 Wnioski

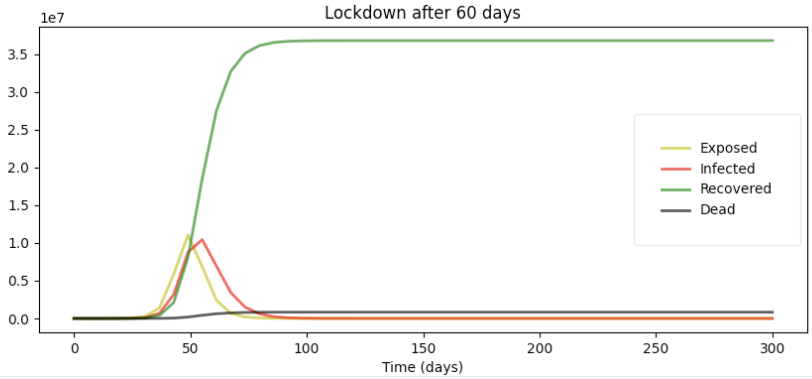
Program oparty na modelu SEIRD jest znacznie bardziej obrazowy i szczegółowy. Przede wszystkim otrzymane wykresy odpowiadają rzeczywistemu rozprzestrzenianiu się epidemii koronawirusa. Przy odpowiednich warunkach początkowych (danych, którymi dysponujemy), jesteśmy w stanie określić punkt w którym znajdujemy się obecnie na krzywej zachorowań:



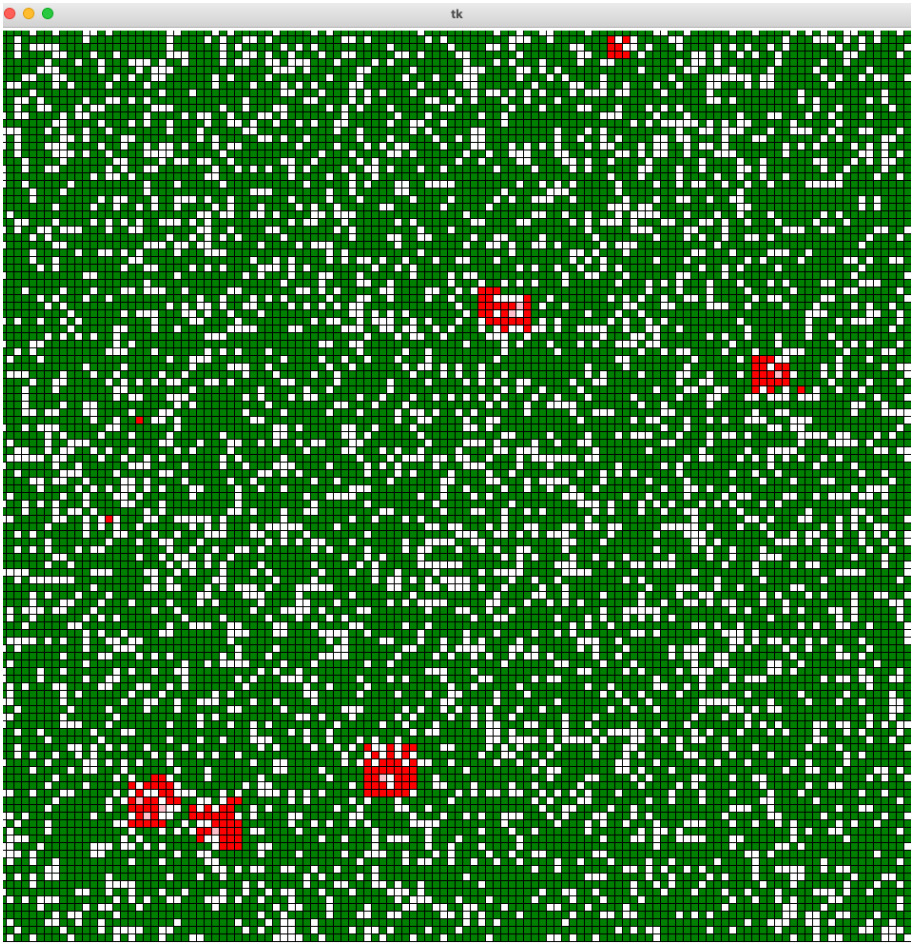
Możemy także przewidzieć dalszy przebieg krzywej w przypadku mniej skutecznych metod profilaktycznych:

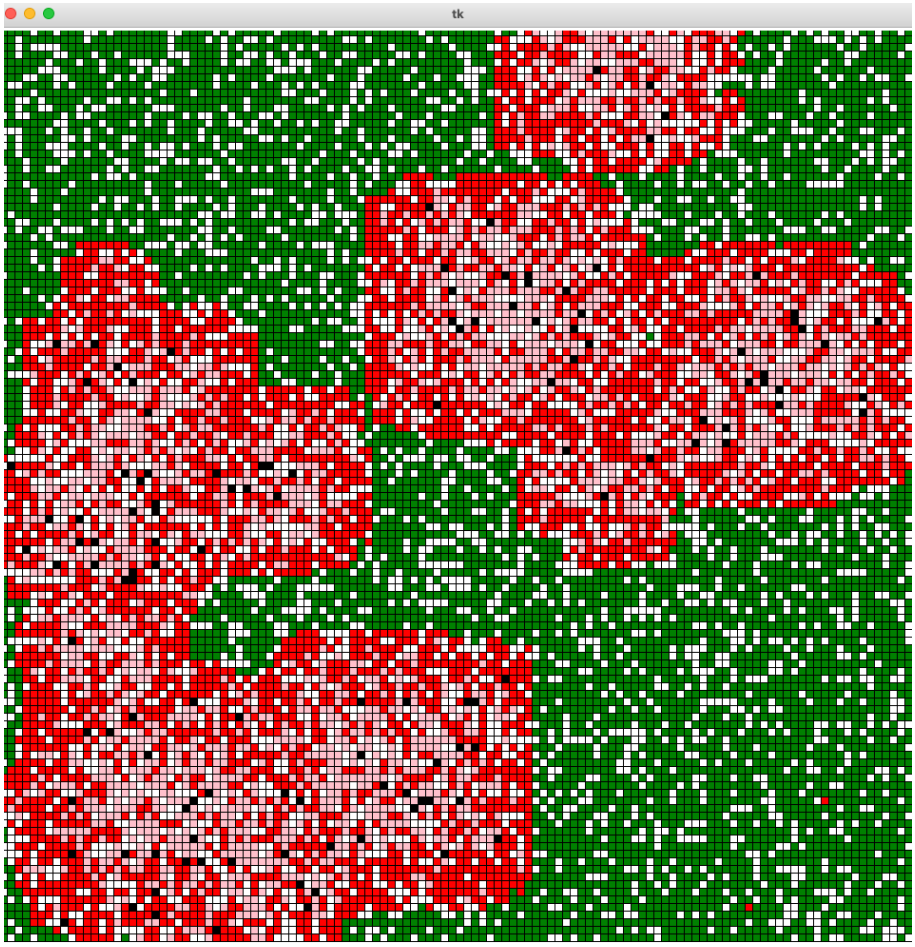


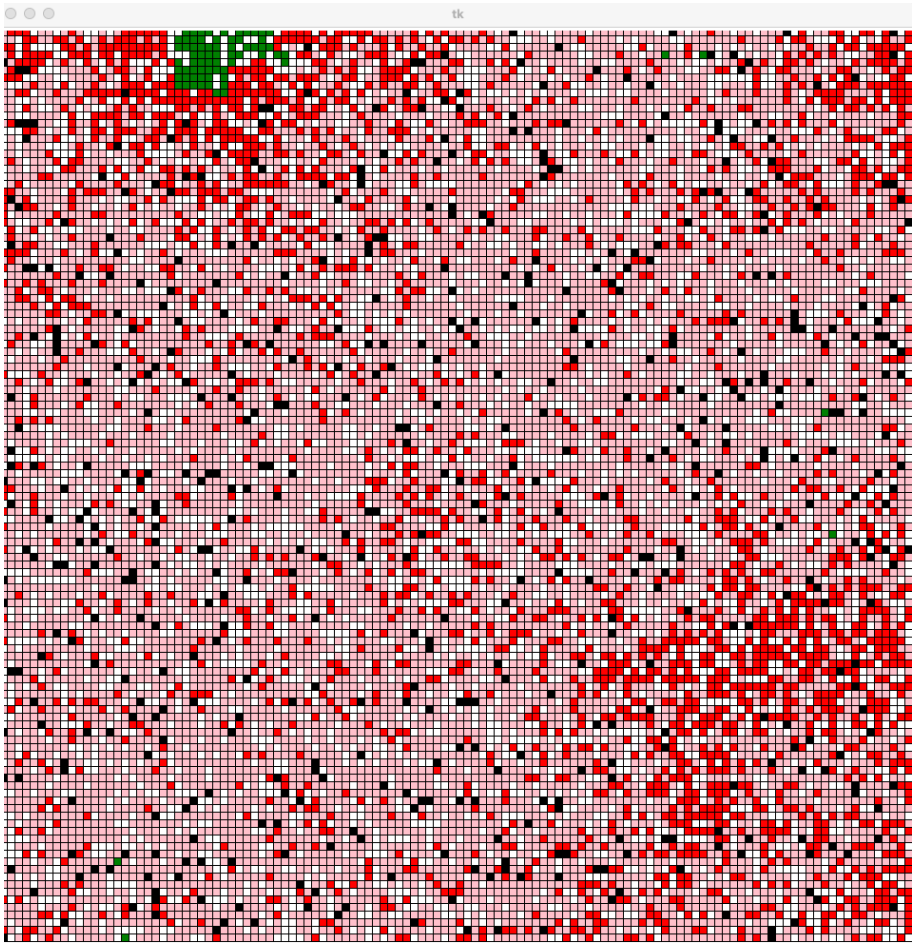
Lub przypadek całkowitego braku zachowania środków bezpieczeństwa i ochrony zdrowia:



Model Monte-Carlo umożliwia nam klarowne przedstawienie rozprzestrzeniania się wirusa, jednak w znacznie bardziej ogólny sposób, przypominający nieco popularną na komputerach stacjonarnych grę „Saper". Z perspektywy graficznej jest on bardziej popularnonaukowy od swojego poprzednika, co świetnie sprawdzi się w celu szerzenia świadomości społeczeństwa dotyczącej sposobu rozprzestrzeniania się wirusa. Pod kątem naukowym wypada znacznie gorzej, ponieważ na jego podstawie możemy uzyskać jedynie podstawowe informacje, które nie niosą za sobą żadnych nowatorskich obserwacji.

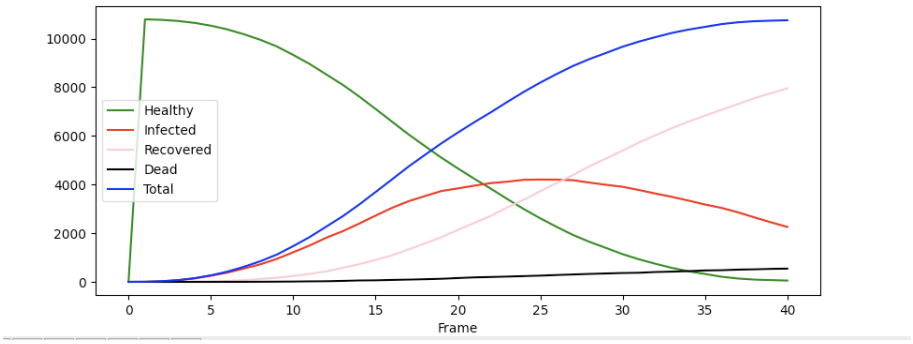

tk


tk


tk

```
10790 Healthy, 4 infected, 0 reovered and 0 dead in frame 1, 4 cases in total
10769 Healthy, 25 infected, 0 reovered and 0 dead in frame 2, 25 cases in total
10723 Healthy, 68 infected, 3 reovered and 0 dead in frame 3, 71 cases in total
10645 Healthy, 142 infected, 8 reovered and 0 dead in frame 4, 150 cases in total
10528 Healthy, 252 infected, 16 reovered and 0 dead in frame 5, 268 cases in total
10375 Healthy, 380 infected, 40 reovered and 1 dead in frame 6, 421 cases in total
10178 Healthy, 547 infected, 69 reovered and 2 dead in frame 7, 618 cases in total
9949 Healthy, 721 infected, 121 reovered and 5 dead in frame 8, 847 cases in total
9676 Healthy, 943 infected, 169 reovered and 8 dead in frame 9, 1120 cases in total
9326 Healthy, 1216 infected, 241 reovered and 13 dead in frame 10, 1470 cases in total
8954 Healthy, 1496 infected, 325 reovered and 21 dead in frame 11, 1842 cases in total
8528 Healthy, 1813 infected, 429 reovered and 26 dead in frame 12, 2268 cases in total
8102 Healthy, 2080 infected, 574 reovered and 40 dead in frame 13, 2694 cases in total
7620 Healthy, 2394 infected, 724 reovered and 58 dead in frame 14, 3176 cases in total
7105 Healthy, 2723 infected, 905 reovered and 64 dead in frame 15, 3692 cases in total
6584 Healthy, 3041 infected, 1092 reovered and 80 dead in frame 16, 4213 cases in total
6061 Healthy, 3313 infected, 1331 reovered and 93 dead in frame 17, 4737 cases in total
5577 Healthy, 3527 infected, 1585 reovered and 109 dead in frame 18, 5221 cases in total
5102 Healthy, 3734 infected, 1835 reovered and 127 dead in frame 19, 5696 cases in total
4660 Healthy, 3841 infected, 2140 reovered and 157 dead in frame 20, 6138 cases in total
4235 Healthy, 3956 infected, 2422 reovered and 185 dead in frame 21, 6563 cases in total
3829 Healthy, 4057 infected, 2713 reovered and 200 dead in frame 22, 6970 cases in total
3403 Healthy, 4118 infected, 3061 reovered and 218 dead in frame 23, 7397 cases in total
2992 Healthy, 4194 infected, 3377 reovered and 237 dead in frame 24, 7808 cases in total
2611 Healthy, 4205 infected, 3728 reovered and 257 dead in frame 25, 8190 cases in total
2257 Healthy, 4202 infected, 4060 reovered and 282 dead in frame 26, 8544 cases in total
1923 Healthy, 4177 infected, 4396 reovered and 305 dead in frame 27, 8878 cases in total
1641 Healthy, 4078 infected, 4755 reovered and 327 dead in frame 28, 9160 cases in total
1392 Healthy, 3989 infected, 5075 reovered and 346 dead in frame 29, 9410 cases in total
1138 Healthy, 3907 infected, 5394 reovered and 364 dead in frame 30, 9665 cases in total
926 Healthy, 3773 infected, 5729 reovered and 375 dead in frame 31, 9877 cases in total
743 Healthy, 3628 infected, 6027 reovered and 405 dead in frame 32, 10060 cases in total
572 Healthy, 3492 infected, 6319 reovered and 421 dead in frame 33, 10232 cases in total
436 Healthy, 3340 infected, 6586 reovered and 442 dead in frame 34, 10368 cases in total
325 Healthy, 3179 infected, 6831 reovered and 469 dead in frame 35, 10479 cases in total
212 Healthy, 3040 infected, 7074 reovered and 479 dead in frame 36, 10593 cases in total
138 Healthy, 2858 infected, 7307 reovered and 503 dead in frame 37, 10668 cases in total
93 Healthy, 2650 infected, 7548 reovered and 515 dead in frame 38, 10713 cases in total
72 Healthy, 2448 infected, 7752 reovered and 534 dead in frame 39, 10734 cases in total
54 Healthy, 2257 infected, 7951 reovered and 544 dead in frame 40, 10752 cases in total
```

Przewaga metody SEIRD nad Monte-Carlo wynika z ilości uwzględnionych w niej parametrów. W drugim przypadku znacznie ciężej jest je dopasować.



5 Podsumowanie

Wykonany projekt ukazuje w dwóch różnych sposobach graficznych rozprzestrzenianie się epidemii. Oba wykorzystane modele spełniają swoją rolę, choć w założeniu są skierowane do dwóch różnych grup odbiorców, w zależności od ich oczekiwań. Zgodnie z opisanymi wnioskami, metodyka zastosowana w SEIRD jest znacznie bardziej dokładna, Monte-Carlo charakteryzuje się natomiast przystępniejszym, bardziej oczywistym wyglądem.

Warto wspomnieć, że powyższe sposoby sprawdzą się także w przypadku modelowania zachowań innych wirusów lub zjawisk przyrodniczych np. pożarów. Często stosowane są również w bardziej abstrakcyjnych projektach, np. grach komputerowych dotyczących pandemii zombie.

Ich cechą wspólną jest niewątpliwie prosta konstrukcja, która znacznie ogranicza możliwość popełnionego przy tworzeniu programów błędu. Sprawia to również, że prawidłowo wykonane działają bez zarzutu przy zaimplementowanych danych początkowych, co można zaobserwować we wnioskach.

6 Bibliografia

Materiały wykorzystane w tworzeniu wstępu teoretycznego:

https://en.wikipedia.org/wiki/Virus

https://www.fuw.edu.pl/~jarekz/MODELOWANIE/modele_epidemii.pdf

http://kft.umcs.lublin.pl/baran/epk/modelowanie/sobol.pdf

Materiały wykorzystane w trakcie opracowywania modeli:

https://arxiv.org/pdf/2003.09909.pdf?fbclid=IwAR0h5Fn-NZsVK6xdWksZ_tjlp1Tyanevdt6sOfYG_XlTYY2v-AQSxnWlk_Q

https://en.wikipedia.org/wiki/Percolation_theory

Wzorowaliśmy się także na przesłanych przez Pana plikach "perc.py" oraz "MCPlot.py"

Ilustracja przedstawiająca schemat ataku wirusa na organizm żywy pochodzi ze strony:

https://pl.wikipedia.org/wiki/Wirusy

Ilustracja wykorzystana w podsumowaniu, przedstawiająca rozgrywkę w grze Plague Inc., pochodzi ze strony:

https://apps.apple.com/pl/app/plague-inc/id525818839?l=pl#?platform=ipad

Pozostałe, wykorzystane screenshoty pochodzą z naszych programów.