授業名:プログラミング演習 2(Python)-2020

レポート名:ウイルス感染シミュレーション

提出日:2020/07/07

所属・学年:情報科学部・２年

学籍番号:19K1142

氏名:リ イーセイ

1. 課題

自分なりに設計したシミュレーションプログラムを作成する。ウイルス感染のシミュレーションを作成した。このシミュレーションでは、人の活動を「外出」、「帰宅」、「治療」の三つの状態に分けた。健康状態を「健康」、「ウイルス潜伏」、「発症」、「完治」の四つにした。健康状態により、人の色を変えた。プログラムで以下の条件を設定した。人は外出し、規定された時間に帰宅する。外で周りの 3x3 マスに感染者がいたら、感染されたか否かを計算する。感染者は発症したらすぐ病院に行く。病院に行った感染者の健康状態は、完治の確率によって計算する。また、画面の左側に現在の感染状況を表した。

2. 課題の目的

今年のコロナウイルス感染は世界中に拡大している。人の密集地域に行く危険性を表せるシミュレーションが必要と考え、ウイルス感染シミュレーションを作った。この課題に通じ、50 人以上の人を導入でき、人の健康と活動ステータスを条件にしたがって変えた。また、人の健康状態を色で表すことに加え、全体の感染状況を文字で表すことができた。

3. 方法

3.1.プログラムの構造

人の動きを示すフローチャットは以下である。

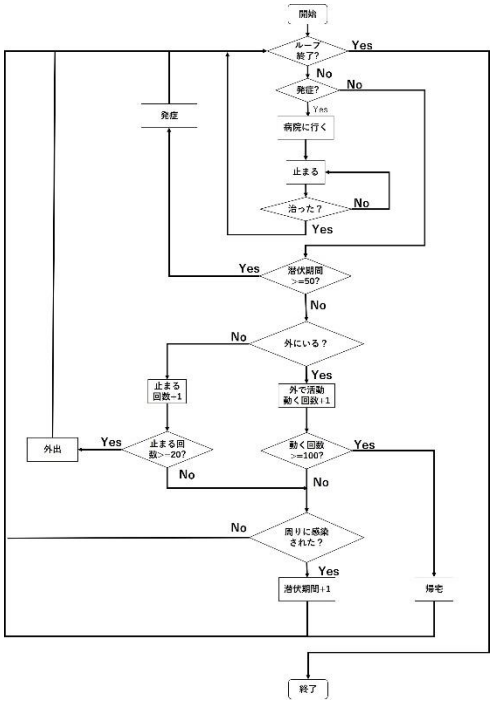


図 1. 人の動きのフローチャット

図 1 により、人はループ内で健康状態、外で動く回数、家に止まる回数と治るか否かによって活動の軌跡が変わる。ループ内、まず発症したかを確認し、発症したらすぐ病院に行く。病院で治らない場合、止まる。治ったらまた健康の人と同じく運動をする。発症していない人の居場所を確認し、外にいると動く回数を増す。家にいると止まる回数を増す。動く回数または止まる回数が一定になると、人の運動状態が変わる。そして、感染されたかを確認し、感染された場合は潜伏期間を増す。以上のプログラムをループする。ループが 1000 回に達すると、プログラムを終了する。

3.2. 健康状態

ステータス：「Health」、「Carry」、「Sick」、「Recover」

① 周りの感染者を見つかる

擬似コード：

```
SEARCH_VIRUS (self):
    to_check=八方向を表すリスト
    if 「Health」である
        for dir in to_check:
            for n=1 to 3:
                x,y=self.x+n*dir[0],self.y+n*dir[1]
                    if (x,y)座標に感染者がいる　かつ　感染された
                                「Carry」になる
```

健康の人の周りに感染者を見つかる。感染者がいると、健康の人が感染されるか否かを判定する。

② 感染される

擬似コード：

```
CHECK_DIFFERECT_LIST (self,x,y,people):
    for p in people:
        if p が家にいない
            if p の座標は(x,y)
                if p は「Carry」
                    r は-1 から 1 まで任意の浮動小数点数
```

```
                    if r の絶対値は 0.1 以下
                        True を返す
                else if p は「Sick」
                    r は-1 から 1 まで任意の浮動小数点数
                    if r の絶対値は 0.1 以下
                        True を返す
    False を返す
```

True を返す場合、この人は感染される。False の場合は感染されない。また、周りの感染者を潜伏期と発症に分け、他人を感染する確率は違うとする。

③ 潜伏期が最大値になると発症する

move 関数にステータスが「Carry」であるかどうかを判定し、True の場合、人が動くと 1 を増す。20 になると「Sick」の状態に切り替える。

④ 病院に行く

病院ではない場所に「Sick」になると、gohospital は True になり、病院の入口に向かって運動する。病院に入ると gohospital が False になり、athospital が True になる。「Recover」になるまで動かない。

⑤ 治療する

擬似コード：

```
RECOVER(self):
    self.gohospital=False
    r=-1 から 1 まで任意の浮動小数点数
    if r の絶対値は 0.1 より小さい
        ステータスが「Recover」になる
        Athospital=False
        y 方向の速度が-1 になる
```

ここでは、random.uniform で治ったかを決める。治る確率は 10%と設定する。また、元の x、y 方向の速度は 0 であるため、治った場合は y 方向速度を-1 と設定し、病院から出るようにしなければならない。

3.3.活動状態

パラメータ: 「play_outside」、「gohome」、「gohospital」、「athome」、「athospital」、「time_to_out」

「play_outside」と「time_to_out」は int の値である。他は True と False で表す。

① 外で活動する

play_outside が 100 になるまで、外で活動する。一回の動きで play_outside に 1 を足す。play_outside が 100 以上になったら、gohome が True になり、time_to_out を 0 にリセットする。家の入り口を向って運動する。

② 家に泊まる

家に入ると gohome が False になり、athome が True になる。全体の一回のループをしたら、time_to_out に 1 を足す。time_to_out が 20 になると athome が False になる。y 方向速度が-1 になる。

③ 病院に行く

gohospital は健康状態によって変わる。3.2 の③で説明した。

3.4. 全体の感染状況を表す

① 各状態の人を数える

擬似コード：

```
COUNT_PEOPLE(self):
    「Sick」「Health」「Carry」「Recover」の人は 0 とする
    すべての人の状態を確認し、カウントする
```

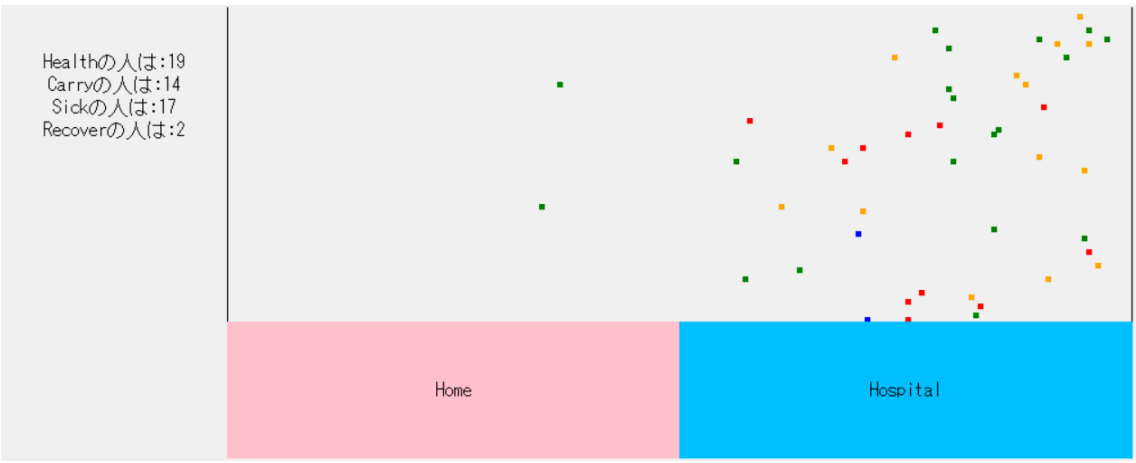
② Tag を書く

（図 2: Tag の表示）

図 2 のように Tag を左に置けばよい。

4. 結果

実行結果



（図 3: 実行結果）

図 3 で、緑の点は健康な人、オレンジ色の点は潜伏期の人、赤い点は発症した人、青い点は治った人を表す。図の下は Home と Hospital の場所である。左で全体の感染状況を表す。

プログラムは問題なく動いた。

5. 考察

このプログラムで、人の活動と健康状態を表すことができた。健康状態により、人の色を変えた。人は特定の条件で、特定の座標に移動することができた。また、画面の左側に現在の感染状況を表した。

このプログラムについて、人の活動状態のステータスは劣っていると思う。最初は arrival と location の二つのステータスを設定し、それぞれは行き先と現在の場所を表す。しかし、実行した結果、現在のプログラムよりかなり遅くなった。現在のプログラムは速いが、複雑であり、状態の判定を間違える可能性がある。現時点では、問題なく

動けるが、少数の発症した人が病院で治ってなく、外で動いていることがある。この問題に対し、良い解決案はまだできていない。

また、より現実的になるため、外で人の密集地を作ってみたいと考えている。人は外で密集地に集まる傾向があると設定する。

付録

① フローチャット

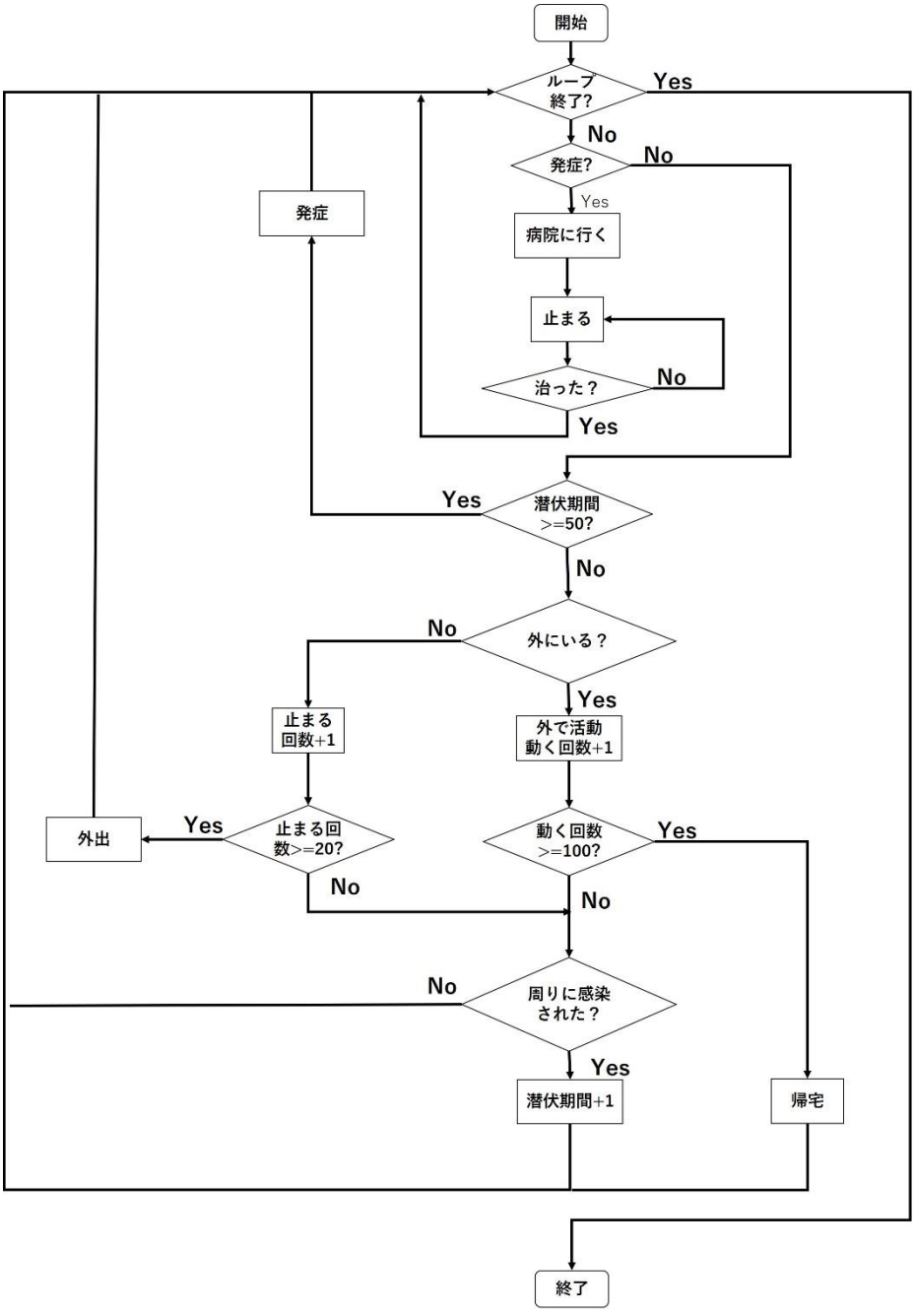


図 1. 人の動きのフローチャット

② プログラム

```
from tkinter import *
import time,random
from dataclasses import dataclass,field
import numpy as np


FIELD_X,FIELD_Y=250,100
GROUND_X,GROUND_Y=50,0
GROUND_W,GROUND_H=200,70
HOME_X,HOME_Y=GROUND_X,GROUND_Y+GROUND_H
HOME_W,HOME_H=(FIELD_X-GROUND_X)/2,FIELD_Y-GROUND_H
DOOR_X,DOOR_Y=(FIELD_X-HOME_X)/2,GROUND_H
HOSPITAL_X,HOSPITAL_Y=HOME_X+HOME_W,HOME_Y
HOSPITAL_W,HOSPITAL_H=(FIELD_X-GROUND_X)/2,FIELD_Y-GROUND_H
HOSPITAL_DOOR_X,HOSPITAL_DOOR_Y=(200,GROUND_H)
DURATION=0.02
u=4

tk=Tk()
tk.attributes('-topmost',True)
canvas=Canvas(tk,width=FIELD_X*u, height=FIELD_Y*u)
canvas.pack()

class Person:
    def
__init__(self,x=10,y=10,dx=1,dy=1,w=1,h=1,play_outside=0,gohome=False,gohospital
=False,athome=False,athospital=False,time_to_out=0,virus="Health",latency=0,world=
None,color='green'):
        self.x,self.y=x,y
        self.dx,self.dy=dx,dy
        self.w,self.h=w,h
        self.play_outside=play_outside
        self.gohome = gohome
        self.gohospital=gohospital
```

```
        self.athome=athome
        self.athospital = athospital
        self.time_to_out=time_to_out
        self.virus=virus
        self.latency=latency
        self.world=world
        self.id=canvas.create_rectangle(self.x*u, self.y*u, (self.x + self.w)*u, (self.y +
self.h)*u,
                                        outline=color,fill=color)

    def __str__(self):
        return f'({self.x},{self.y})'

    def redraw(self):
        canvas.coords(self.id, self.x*u, self.y*u,
                      (self.x + self.w)*u, (self.y + self.h)*u)

    def check_wall(self):
        if self.x + self.w > GROUND_X+GROUND_W-1 or self.x < GROUND_X+1:
            self.dx = -self.dx
        if self.y + self.h > GROUND_Y+GROUND_H or self.y < GROUND_Y+1:
            self.dy = -self.dy

    def virus_state(self):
        if self.virus=="Carry":
            c = "orange"
            canvas.itemconfigure(self.id, fill=c)
            canvas.itemconfigure(self.id, outline=c)
        if self.virus=="Carry" and self.latency==50:
            self.virus="Sick"
            c = "red"
            canvas.itemconfigure(self.id, fill=c)
            canvas.itemconfigure(self.id, outline=c)
```

```
        if self.virus=="Recover":
            c = "blue"
            canvas.itemconfigure(self.id, fill=c)
            canvas.itemconfigure(self.id, outline=c)

    def exploring(self):
        self.change_dir(-1,0.1)

    def change_dir(self, r0, p):
        self.dirs = [(1, 1), (0, 1), (-1, 1), (-1, 0), (-1, -1), (0, -1), (1, -1), (1, 0)]
        ind = self.dirs.index((self.dx, self.dy))
        r = random.uniform(r0, 1)
        newInd = (ind + int(np.sign(r))) % 8 if abs(r) < p else ind
        self.dx, self.dy = self.dirs[newInd]

    def go_back_home_state(self):
        if self.play_outside>=100:
            self.gohome=True
            self.time_to_out=0
        if self.gohome==True and (self.x,self.y)==(DOOR_X,DOOR_Y):
            self.play_outside=0
            self.gohome=False
            self.athome=True

    def go_out_state(self,y1):
        if self.y<y1:
            self.athome = False
        if self.time_to_out>=20:
            self.athome = False
            self.time_to_out=0
            self.dy=-1
```

```
def go_back(self,x1,y1):
    x2,y2=(self.x,self.y)
    if x1-x2<0:
        self.dx=-1
    elif x1-x2==0:
        self.dx=0
    else:
        self.dx=1

    if y1-y2<0:
        self.dy=-1
    elif y1-y2==0:
        self.dy=0
    else:
        self.dy=1

def search_virus(self):
    to_check=[(1,1),(0,1),(-1,1),(-1,0),(-1,-1),(0,-1),(1,-1),(1,0)]
    if self.virus=="Health":
        for d in to_check:
            for n in range(3):
                x=self.x+n*d[0]
                y=self.y+n*d[1]
                if self.world.check_virus(x,y):
                    self.virus="Carry"

def go_to_hosipital_state(self):
    if self.virus=="Sick" and self.athospital==False and self.gohospital==False:
        self.gohome=False
        self.gohospital=True
```

```
            if self.gohospital==True and
           (self.x,self.y)==(HOSPITAL_DOOR_X,HOSPITAL_DOOR_Y):
                self.play_outside=0
                self.gohospital = False
                self.gohome=False
                self.athospital=True
            if self.virus == "Recover" and self.athospital == True:
                self.athospital=False
                self.gohospital=False

    def recover(self):
            self.gohospital = False
            r = random.uniform(-1, 1)
            if abs(r) < 0.1:
                self.virus="Recover"
                self.athospital = False
                self.dy=-1

class HybridPeople(Person):
    def move(self):
            if self.gohome==False and self.play_outside<=100 and self.athome==False and
            self.gohospital==False:
                self.exploring()
                self.play_outside+=1
            elif self.athome==True and self.gohome==False:
                self.stay()
                self.time_to_out+=1
            elif self.athospital==True and self.virus=="Sick":
                self.stay()
            elif self.athome==False and self.gohome==True and self.gohospital==False:
                self.go_back(DOOR_X,DOOR_Y)
            elif self.athospital==False and self.gohospital==True and self.virus=="Sick":
                self.go_back(HOSPITAL_DOOR_X,HOSPITAL_DOOR_Y)
```

```
            else:
                print("what")
                self.exploring()
            if self.virus=="Carry":
                self.latency+=1
            self.x += self.dx
            self.y += self.dy


class Home:
    def __init__(self,x=HOME_X,y=HOME_Y,w=HOME_W,
                 h=HOME_H,world=None,color='pink'):
        self.x,self.y=x,y
        self.w,self.h=w,h
        self.world=world
        self.id=canvas.create_rectangle(self.x*u, self.y*u, (self.x + self.w)*u,
                                        (self.y + self.h)*u,outline=color,fill=color)

class Hospital:
    def __init__(self,x=HOSPITAL_X,y=HOSPITAL_Y,w=HOSPITAL_W,
                 h=HOSPITAL_H,world=None,color='deepskyblue'):
        self.x,self.y=x,y
        self.w,self.h=w,h
        self.world=world
        self.id=canvas.create_rectangle(self.x*u, self.y*u, (self.x + self.w)*u,
                                        (self.y + self.h)*u,outline=color,fill=color)

class Tag:
    def __init__(self,id=None,state=''):
        self.id=id
        self.state=state
```

```
@dataclass
class World:
    people:list=field(default_factory=list)
    home: Home = None
    hospital: Hospital=None
    tag_health: Tag=None
    tag_carry: Tag=None
    tag_sick: Tag=None
    tag_recover: Tag = None
    health: int=0
    carry: int=0
    sick: int=0
    recover: int = 0

    def draw_wall(self,x,y,w,h,u):
        canvas.create_rectangle(x * u, y * u, (x + w) * u, (y + h) * u)

    def set_people(self,n,x,y):
        for a in range(n):
            r = random.randint(-50, 50)
            p=HybridPeople(x=x+r,y=y,world=self)
            self.people.append(p)

    def set_sick_people(self,n,x,y):
        for a in range(n):
            p=HybridPeople(x=x,y=y,virus="Carry",world=self,color='red')
            self.people.append(p)

    def set_home(self):
        self.home=Home()
        canvas.create_text(100*u, 85*u, text="Home", font=('FixedSys', 2))

    def set_hospital(self):
        self.hospital=Hospital()
        canvas.create_text(200 * u, 85 * u, text="Hospital", font=('FixedSys', 2))
```

```
def count_people(self):
    self.sick =0
    self.health=0
    self.carry=0
    self.recover=0
    for p in self.people:
        if p.virus=="Sick":
            self.sick+=1
        elif p.virus=="Carry":
            self.carry+=1
        elif p.virus=="Health":
            self.health+=1
        else:
            self.recover+=1

def set_four_tag(self):
    self.tag_health=self.which_tag("Health",self.health,100,50)
    self.tag_carry = self.which_tag("Carry", self.carry, 100, 70)
    self.tag_sick = self.which_tag("Sick", self.sick, 100, 90)
    self.tag_recover = self.which_tag("Recover", self.recover, 100, 110)
    tk.update()

def which_tag(self,state,n,x,y):
    id = canvas.create_text(x, y, text=f"{state}の人は:{n}", font=('FixedSys', 2))
    return Tag(id,state)

def delete_tag(self):
    canvas.delete(self.tag_health.id)
    canvas.delete(self.tag_carry.id)
    canvas.delete(self.tag_sick.id)
    canvas.delete(self.tag_recover.id)
    tk.update()
```

```
    def animation_step(self,people):
        for h in people:
            h.go_back_home_state()
            h.go_to_hosipital_state()
            h.go_out_state(DOOR_Y)
            h.move()
            h.redraw()
            tk.update()
            h.virus_state()
            if h.athome==False and h.athospital==False:
                h.search_virus()
            if h.athome==False and h.gohome==False:
                h.check_wall()
            if h.athospital==True:
                h.recover()
        self.count_people()
        self.delete_tag()
        self.set_four_tag()
        tk.update()
        time.sleep(DURATION)

    def step(self):
        self.animation_step(self.people)

    def start(self,n_steps):
        self.set_people(50,200,50)
        self.set_sick_people(2, 200, 50)
        self.sick = 0
        self.health = 0
        self.set_four_tag()
        self.draw_wall(GROUND_X,GROUND_Y,GROUND_W,GROUND_H,u)
        self.set_home()
        self.set_hospital()
        for x in range(n_steps):
            self.step()
```

```
    def check_virus(self,x,y):
        if self.check_different_list(x,y,self.people):
            return True
        return False

    def check_different_list(self,x,y,people):
        for p in people:
            if p.athome==False:
                if (p.x,p.y)==(x,y):
                    if p.virus=="Carry":
                        r = random.uniform(-1, 1)
                        if abs(r) < 0.1:
                            return True
                    elif p.virus=="Sick":
                        r = random.uniform(-1, 1)
                        if abs(r) < 0.2:
                            return True
        return False

World().start(1000)

tk.mainloop()
```

(19k1142-R01.py)