МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «КубГУ»)

Факультет компьютерных технологий и прикладной математики

Кафедра информационных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 6
по дисциплине
«МНОГОАГЕНТНОЕ МОДЕЛИРОВАНИЕ»

Выполнил студент группы 45/2 _____ Т. Э. Айрапетов

Направление подготовки 02.03.03 Математическое обеспечение и администрирование информационных систем
Курс 4

Отчет принял доктор физико-математических наук,
профессор _____ А.И. Миков

Краснодар
2024 г.

Задание:

Есть n агентов, случайно расположенных в квадрате 100x100 (это значит, что у каждого i агента есть координаты позиции на плоскости (x_i, y_i)). Каждый агент имеет определенное состояние:

- Здоровый

- Зараженный

- Зомби

- Выздоровевший

У каждого агента есть некоторое направление и скорость движения (v_{xi}, vy_i), согласно которым он движется случайный период времени (t_{move}), по истечению которого для него генерируется новое направление движения.

В начале моделирования все агенты на поле являются здоровыми. Спустя некоторый период длительностью t_{init} среди них случайным образом m агентов становятся зараженными и у них начинается инкубационный период. Во время этого периода скорость агента снижена по сравнению со стандартной.

По истечению инкубационного периода, зараженный агент становится зомби. Целью зомби является заражение здоровых агентов. В свою очередь у здорового агента целью является не быть зараженным.

На каждой итерации цикла, зомби с 1% шансом может стать выздоровевшим. Выздоровевший агент не является целью для зомби, однако если он попадает в радиус их действия, с шансом он обращается снова, минуя стадию заражения.

Все агенты зрячие, причем видят они сектор окружности, расположенный согласно вектору их движения, имеющий угол a и радиус r. Соответственно они могут идентифицировать других агентов, если те попали в их сектор видимости, о других они не знают (ни их положения, ни их состояния).

Задача

Для нескольких разных наборов значений n и m (не менее 4 значений для каждой характеристики), составить таблицу, в которой будет указано среднее время обращения в зомби всей популяции (в области не остается

здоровых агентов). Для получения среднего времени для каждой конфигурации провести 1000 экспериментов.

Решение.

Для реализации изменчивого поведения агентов был создан статический класс State, хранящий коэффициенты значений угла обзора, радиуса обзора и скорости агентов для разных состояний.

Также был создан класс Agent, состоящий из следующих полей:

- *coords* - координаты агента (x_i, y_i);
- *t_inc* - время окончания инкубационного периода или -1;
- *state* - состояние агента (0, 1, 2, 3 - здоров, заражен, зомби, выздоровел);
- *ang* - угол обзора агента (случайное значение из диапазона [90, 150]);
- *v* - скорость агента;
- *r* - радиус обзора агента;
- *dir* - вектор направления агента.

В классе Agent также есть методы *look* и *move* для осмотра соседей из области видимости и для движения соответственно.

В программе также присутствует вспомогательная функция *rotate_vector*, реализующая поворот вектора направления агента на определенный угол.

Функция реализующая моделирование среды, *sim*, принимает на вход параметры моделирования:

- кол-во агентов *n*;
- кол-во зараженных *m*;
- максимальное время моделирования T .

В процессе моделирования на каждой итерации для каждого агента реализуется поведение, соответствующее состоянию агента. Моделирование останавливается в следующих случаях:

- все агенты являются зомби;

- никто из агентов не является зомби или зараженным;
- достигнуто максимальное время моделирования.

Случаи, когда никто из агентов не является зомби или зараженным, помечаются флагом *early_break*.

Моделирование было проведено для n = [20, 50, 100, 120] и m = [0.1, 0.3, 0.5, 0.8]. Количество зараженных m вводится как коэффициент зараженных относительно общего числа. Ниже представлены средние значения времени выполнения моделирования 1000 запусков для T = 100 (рис. 1).

n	m	t	early_break
20	2	48.2	0.0
20	6	57.1	0.0
20	10	63.7	0.0
20	16	84.2	0.0
50	5	56.4	0.0
50	15	83.6	0.0
50	25	100.0	0.0
50	40	100.0	0.0
100	10	77.3	0.0
100	30	100.0	0.0
100	50	100.0	0.0
100	80	100.0	0.0
120	12	84.3	0.0
120	36	100.0	0.0
120	60	100.0	0.0
120	96	100.0	0.0

Рисунок 1 - Таблица среднего времени обращения в зомби всех агентов

Как можно заметить, с увеличением общего числа агентов, время обращения в зомби всех агентов увеличивается. Можно предположить, что это связано с тем, что при большом числе агентов, являющихся зомби, вероятность выздоровления хотя бы одного из них становится довольно высокой.

Также можно обратить внимание на то, что флаг early_break на всех итерациях был равен нулю (хотя бы 1 агент являлся зомби или зараженным).

Немаловажными параметры при моделировании являются v и r. Так, при малых значениях r (<30), зомби в течение всего цикла моделирования не находили агентов, либо выздоравливали. Параметр v также стоит подбирать аккуратно, так как при достаточно больших значениях зомби будут «перескакивать» через свои «цели», а простые агенты будут слишком быстро убегать от зомби. При малых значениях v агенты будут слишком медленно перемещаться по карте, что также приводит к увеличению среднего времени моделирования. Экспериментально были взяты значения r=50, v=10.

Текст программы на языке Python:

```
import numpy as np
import matplotlib.pyplot as plt
def draw(agents, t):
    plt.clf()  # Очистка текущего графика
    plt.xlim(0, 100)
    plt.ylim(0, 100)
    for i in agents:
        plt.scatter(*i.coords, c='gyrb'[i.state])
        plt.quiver(*i.coords, *(i.dir), width=0.005)
        # print(*i.coords, *(i.dir*i.v), i.dir*i.v)
        # plt.text(*i.coords, i.dir)
    # plt.axis([0,100,0,100])
    plt.title(f'Состояние среды t={t}')



class State:
    healthy = 0
```

```
    infected = 1
    zombie = 2
    recovered = 3
    v_scaler = [1, 0.9, 0.85, 1]
    r_scaler = [1, 1, 1.1, 1]
    a_scaler = [1, 1, 0.75, 1]

t_inc_min = 1
t_inc_max = 10

class Agent:
    def __init__(self, id, v, r):
        self.id = id
        self.coords = np.random.randint(0, 101, 2)
        self.t_inc = -1
        self.state = State.healthy

        self.ang = np.random.randint(90, 151)
        self.v = v
        self.r = r

        self.dir = np.random.rand(2)*2-1 # направление движения
        self.dir /= np.linalg.norm(self.dir) # нормализация
        # self.a_h = self.a # а здорового агента

    def get_rad(self):
        return self.r*State.r_scaler[self.state]

    def get_ang(self):
        return self.ang*State.a_scaler[self.state]

    # Возвращаем угол и расстояние до другого агента
    def look(self, other):
        a, b = self.dir.copy(), other.coords-self.coords
        dot_product = np.dot(a, b)
```

```
        det = a[0] * b[1] - a[1] * b[0]
        angle = np.arctan2(det, dot_product)
        return np.degrees(angle), np.linalg.norm(b)

    def move(self, run=False):
        v = self.v*State.v_scaler[self.state] # факт. скорость
        self.coords = self.coords + self.dir*v*[1,1.25][run]
        if not 0<=self.coords[0]<101:
            self.dir[0] *= -1
            self.coords[0] = 100-(self.coords[0]%100)
        if not 0<=self.coords[1]<101:
            self.dir[1] *= -1
            self.coords[1] = 100-(self.coords[1]%100)

    def __repr__(self):
        return f"Agent{self.id} ({self.x}, {self.y})"

    def __str__(self):
        return self.__repr__()

def rotate_vector(vector, angle_degrees):
    angle_radians = np.radians(angle_degrees)

    rotation_matrix = np.array([
        [np.cos(angle_radians), -np.sin(angle_radians)],
        [np.sin(angle_radians), np.cos(angle_radians)]
    ])
    rotated_vector = rotation_matrix.dot(vector)
    return rotated_vector / np.linalg.norm(rotated_vector)

def sim(n, m, T, visualize=False, log=False):
    r = 50
    v = 10
    agents = []
    for i in range(n):
```

```
    agents.append(Agent(i, v, r))


t_init = np.random.randint(1, 20)
m_agents = np.random.choice(n, m, replace=False)
if log:
    print(m_agents)
for t in range(t_init):
    for a in agents:
        a.move()


for i in m_agents:
    agents[i].t_inc = t + np.random.randint(t_inc_min, t_inc_max)
    agents[i].state = State.infected

def is_all_zombies():
    temp = sum([i.state==State.zombie for i in agents])
    # print(temp)
    return temp  == len(agents)
def no_zombies():
    temp = sum([(i.state in [State.infected, State.zombie]) for i in agents])
    # print(temp)
    return temp == 0


t = t_init
early_break = 0
while (t < T) and (not is_all_zombies()):
    t += 1
    temp = no_zombies()
    # print(temp)
    if temp:
        early_break = 1
        break
    for a in agents:
        # Прошел инкубационный период
        if a.t_inc != -1 and a.t_inc == t:
            a.state = State.zombie
```

```
        a.t_inc = -1
        if log:
            print(f'{t}: Agent{a.id} стал зомби')

    # Вероятность выздороветь 1%
    if a.state == State.zombie:
        if np.random.choice(2, p=[0.99, 0.01]):
            a.state = State.recovered
            if log:
                print(f'{t}: Agent{a.id} выздоровел')

    # Бежит ли агент
    run = False
    temp = []

    # Собираем агентов из области видимости
    for b in agents:
        if b.id != a.id:
            ang, dist = a.look(b)
            if abs(ang) <= a.get_ang()/2 and dist <= a.get_rad():
                temp.append((b, ang, dist))

    match a.state:
        # Здоровый агент убегает если рядом зомби
        case State.healthy:
            right, left = False, False
            for b,ang,dist in temp:
                if b.state == State.zombie:
                    if ang <= 0:
                        right = True
                    else:
                        left = True
            if right or left:
                run = True
                if right and left:
                    a.dir *= -1
                elif right:
```

```
                a.dir = rotate_vector(a.dir, a.ang/2)
            else:
                a.dir = rotate_vector(a.dir, -a.ang/2)
    # Зомби заражает агента из области действия
    # или бежит за ближайшим агентом из области видимости
    case State.zombie:
        r_min = -1
        an_min = -1

        for b,ang,dist in temp:
            if b.state in [State.recovered, State.healthy]:
                if dist <= a.get_rad()*0.93 and abs(ang) <= a.get_ang()*0.93:
                    # Заражает агентов из области действия
                    if b.state == State.healthy:
                        b.state = State.infected
                        if log:
                            print(f'{t}: Agent{a.id} заболел')
                        b.t_inc = t + np.random.randint(t_inc_min, t_inc_max)
                    # С вероятностью 25% обращает в зомби выздоровевшего агента
                    elif np.random.choice(2, p=[0.75, 0.25]):
                        b.state = State.zombie
                        if log:
                            print(f'{t}: Agent{a.id} стал зомби')
                # Гонится за ближайшим здоровым агентом
                elif b.state == State.healthy:
                    if r_min == -1:
                        r_min = dist
                        an_min = ang
                    elif r_min > dist:
                        r_min = dist
                        an_min = ang
        if r_min != -1:
            a.dir = rotate_vector(a.dir, an_min)


a.move(run)
```

```
    if visualize:
        draw(agents, t)

 return t, early_break

ns = [20, 50, 100, 120]
ms = [0.1, 0.3, 0.5, 0.8]
T = 100
d = {'n':[], 'm':[], 't':[], 'early_break':[]}
for n in ns:
    print(n)
    for m in ms:
        s = 0
        brk = 0
        m_t = int(m*n)
        k = 1000
        for _ in range(k):
            a,b = sim(n, m_t, T)
            s+=a
            brk+=b
        d['n'].append(n)
        d['m'].append(m_t)
        d['t'].append(s/k)
        d['early_break'].append(brk/k)

import pandas as pd

print(pd.DataFrame(d))
```

Выводы. Как можно заметить в нашем случае, большое число зараженных агентов (или зомби) не гарантирует более быстрое завершение моделирования (хотя наивно это наиболее ожидаемый результат). Из этого можно сделать вывод, что для успешной реализации

сред взаимодействующих агентов и достижения желаемого результата, важным фактором является подбор параметров.