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**

**ОТЧЁТ №6**

**Дисциплина: Многоагентное моделирование**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. А. Иванов

Направление подготовки: 02.03.03 Математическое обеспечение и администрирование информационных систем

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А. А. Миков

Краснодар

2024

**Цель работы**

Составить программу, которая будет моделировать «Эпидемию»: на карте создаются здоровые агенты, через время некоторые из них заболевают, превращаются в зомби и начинают заражать других агентов.

**Описание задачи**

***Эпидемия***

**Контекст**

Есть n агентов, случайно расположенных в квадрате 100х100 (это значит, что у каждого i агента есть координаты позиции на плоскости (x\_i, y\_i)). Каждый агент имеет определённое состояние: здоровый, заражённый, зомби, выздоровевший.

У каждого агента есть некоторое направление и скорость движения (v\_xi, v\_yi), 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 длительности t\_init среди них случайным образом m агентов становятся заражёнными, и у них начинается инкубационный период. Во время этого периода скорость агента снижена по сравнению со стандартной.

По истечению инкубационного периода, заражённый агент становится зомби. Целью зомби является заражение здоровых агентов. В свою очередь у здоровых агентов целью является не быть заражёнными.

На каждой итерации цикла, зомби с 1% шансом может стать выздоровевшим. Выздоровевший агент не является целью для зомби, однако если он попадёт в радиус их действия, то с 25% шансом он обращается снова, миную стадию заражения.

Все агенты зрячие, причём видят они сектор окружности, расположенный согласно вектору их движения, имеющий угол alpha 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).

**Характеристики состояний**

**Здоровый**

Скорость (сколько может пройти за итерацию): v\_h – при обычном движении; v\_h \* 1.25 – при бегстве от зомби.

Сектор видимости: угол – alpha (случайная для каждого агента величина в диапазоне от 90 до 150 градусов); радиус – r\_h.

Имеет 2 состояния: случайное блуждание в области; бегство от зомби.

Если агент видит зомби в своём секторе, то сектор делится надвое: если зомби в правом секторе, то его направление движения меняется влево так, чтобы указанный зомби (или несколько) больше не попадал в сектор его видимости; если зомби в левом секторе, то его направление движения меняется вправо так, чтобы указанный зомби (или несколько) больше не попадал в сектор его видимости; если зомби есть в обоих секторах, то агент поворачивается на 180 градусов и движется в обратном направлении.

Не избегает заражённых.

**Заражённый**

Скорость снижена по сравнению со стандартной на 10%. Сектор видимости аналогичен здоровому. Не имеет цели и не является целью для зомби.

Инкубационный период: случайное значение из диапазона [t\_inc\_min, t\_inc\_max] (согласно равномерному распределению).

**Зомби**

Скорость снижена по сравнению со стандартной на 15%.

Сектор видимости: угол – alpha \* 0.75 (то есть уменьшается на 25% по сравнению со стандартной видимостью конкретного агента); радиус – r \* 1.1 (радиус обзора увеличен на 10% по сравнению со стандартной видимостью конкретного агента).

Сектор действия (сектор, при попадании в которой здоровый агент становится заражённым), по всем характеристикам меньше сектора видимости на 7%.

Когда зомби видит здорового агента, он начинает двигаться в его сторону. При попадании в зону видимости нескольких здоровых агентов, движение осуществляется к ближайшему из них.

На каждой итерации с небольшим шансом может стать выздоровевшим.

**Выздоровевший**

Имеет стандартные характеристики (как у здорового). Не является целью для зомби, не убегает от них. В случае, если агент попадает в зону действия зомби, он с 25% вероятностью снова становится зомби.

**Процесс моделирования**

В нулевой момент времени в области равномерно распределяются агенты, все они имеют статус здоровый. Они передвигаются случайным образом, с некоторой заданной скоростью в случайном направлении (на каждом шаге к их позиции добавляется определённый вектор, так, чтобы их перемещение было по направлению движения, но величина смещения была не больше значения скорости). Если при смещении на следующем шаге, позиция агента окажется за пределами области, то он должен изменить своё направление движения согласно принципу отражения. Движение осуществляется на протяжении t\_move итераций (случайное значение из диапазона [t\_move\_min, t\_move\_max]), после чего генерируется новое направление и время движения.

По истечении времени t\_init m агентов становятся заражёнными, для каждого агента генерируется своё время инкубационного периода из диапазона [t\_inc\_min, t\_inc\_max] (согласно равномерному распределению).

Когда инкубационный период у агента заканчивается, он становится зомби, его целью становится преследование и заражение здоровых, здоровые же, в свою очередь, могут его идентифицировать и спасаться.

Моделирование заканчивается, когда в области не остаётся агентов в состоянии здоровый, или по истечению времени моделирования T.

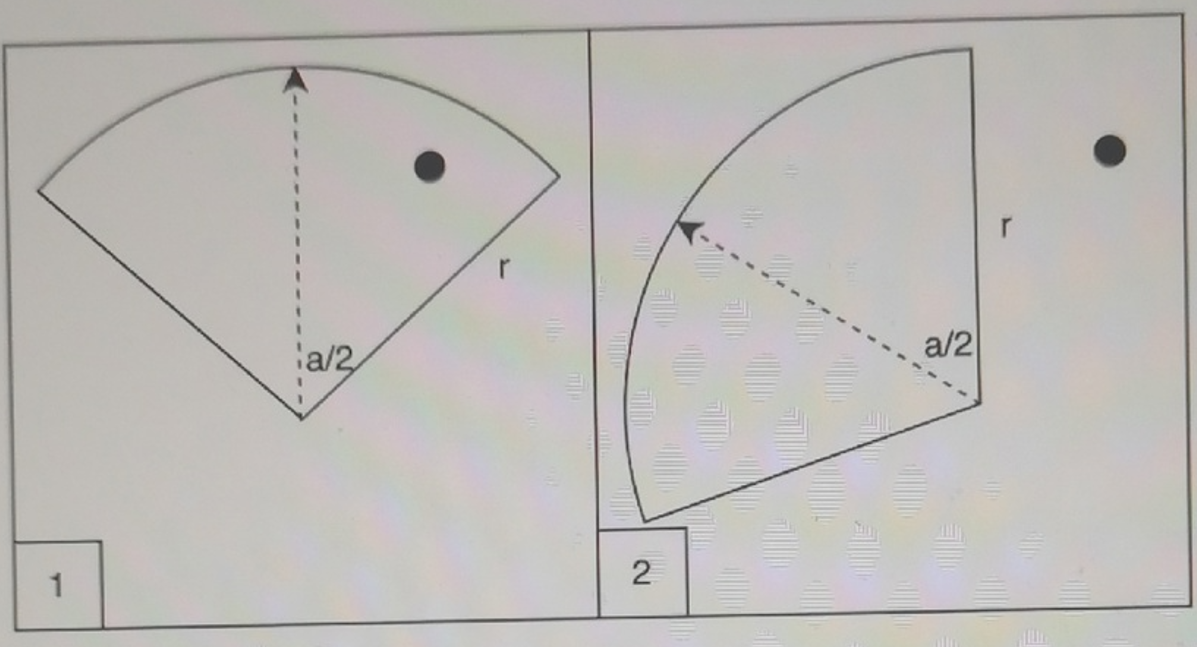
**Задача**

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

Рекомендации и ремарки

1. Для реализации изменяемого поведения агентов можно использовать паттерн проектирования «Машина состояний».

2. Пример изменения направления движения у здоровых агентов при попадании зомби в правый сектор видимости:



3. Функции для работы с геометрией и математикой в целом можно вынести в отдельный класс/файл. Разумно добавить набор тестов для каждой функции, чтобы быстро убедиться в правильности её работы.

4. Пример организации цикла в программе:

**Описание решения**

Для написания программы был выбран язык Python ввиду его простой реализации ООП.

В ходе решения будет использоваться объектно-ориентированный подход, поскольку данную задачу можно разбить на классы для упрощения логики работы. Кроме того, будет использована фабричная функция для динамического изменения классов объектов.

Всего было создано 7 файлов: task.docx (описание задачи), main.py (главный файл, отвечающий за симуляции), Agent.py (базовый класс агентов, также Здоровый агент), InfectedAgent.py (наследник Агента, имеет изменённую логику для заражённого агента), ZombieAgent.py (наследник Агента, имеет изменённую логику для зомби-агента), RecoveredAgent.py (наследник Агента, имеет изменённую логику для выздоровевшего агента), agent\_factory.py (фабричная функция, используется для устранения цикличных импортов классов).

Описание файлов:

Main.py: данный файл отвечает за создание списка всех агентов, строит анимированную симуляцию агентов, а также запускает циклично прогонки для пар значений, которые позже будут отображены в таблице;

Agent.py: базовый класс агентов, включает всю основную логику (работа с направлением и скоростью через тригонометрические функции, подсчёт скорости для корректного векторного перемещения, построение сектора видимости, понимание попадание агентов в сектор видимости (сравнивается площадь треугольника сектора видимости с суммой трёх площадей треугольников, полученных путём нахождения агента в зоне видимости) и логика перемещения); также является классом Здорового Агента;

InfectedAgent.py: класс-наследник Агента, имеет изменённую логику перемещения (медленнее перемещается) и имеет инкубационный период, по окончанию которого становится Зомби;

ZombieAgent.py: класс-наследник Агента, имеет изменённую логику для преследования здоровых агентов и обращения их в зомби, а также реализована вероятность стать выздоровевшим агентом;

RecoveredAgent.py: класс-наследник Агента, имеет изменённую логику перемещения (не убегает от зомби и не преследуется зомби), а также имеет шанс снова стать зомби при попадании в область действия зомби;

Agent\_factory.py: фабричный метод для изменения классов объектов Зомби и Выздоровевшего агента без прямого импорта последними друг друга.

**Код программы**

import random as rnd

import math as mt

class Agent:

    def \_\_init\_\_(*self*, *id*: int, *x*: float, *y*: float) -> None:

        """

        Метод-конструктор класса Агент

        Поля:

            id (int): Id агента

            x (float): Координата агента по X

            y (float): Координата агента по Y

            vision\_radius (int): Радиус обзора агента

            vision\_angle (int): Угол обзора агента

            speed (int): Общая скорость агента

            speed\_x (float): Скорость агента по X

            speed\_y (float): Скорость агента по Y

            direction (int): Направление движения агента (в градусах (0 - вертикаль по Y+))

            overview (list): Координаты точек конуса видимости агента относительно направления движения

            t\_move (int): Время, в течении которого агент перемещается

        Args:

            id (int): Id агента

            x (float): Координата агента по X

            y (float): Координата агента по Y

        Returns:

            None

        """

*self*.id = *id*

*self*.x = *x*

*self*.y = *y*

*self*.vision\_radius = rnd.randint(10, 15)

*self*.vision\_angle = rnd.randint(90, 150)

*self*.speed = 0

*self*.speed\_x = 0

*self*.speed\_y = 0

*self*.direction = 0

*self*.overview = []

*self*.t\_move = 0

    def info(*self*) -> None:

        """

        Метод, который выводит основную информацию об агенте

        Args:

            None

        Returns:

            None

        """

        print(f"id = {*self*.id} x = {*self*.x} y = {*self*.y} vision\_radius = {*self*.vision\_radius} vision\_angle = {*self*.vision\_angle} speed = {*self*.speed} speed\_x = {*self*.speed\_x} speed\_y = {*self*.speed\_y} direction = {*self*.direction} overview = {*self*.overview} t\_move = {*self*.t\_move}")

    def speed\_calculation(*self*) -> None:

        """

        Метод, который рассчитывает координатные скорости в зависимости от направления движения агента

        Args:

            None

        Returns:

            None

        """

        angle\_radian = mt.radians(*self*.direction)

*#по оси X выбран sin, потому что 0 градусов - вертикаль Y+, а не горизонталь X+*

*self*.speed\_x = *self*.speed \* mt.sin(angle\_radian)

*self*.speed\_y = *self*.speed \* mt.cos(angle\_radian)

    def overview\_calculation(*self*) -> None:

        """

        Метод, который вычисляет точки конуса видимости агента в зависимости от его положения и направления движения

        Args:

            None

        Returns:

            None

        """

        cone\_points = [(*self*.x, *self*.y)]            *#начальная точка сектора - позиция агента*

        half\_angle = *self*.vision\_angle / 2          *#половина угла обзора*

        angle\_center = mt.radians(*self*.direction)   *#угол направления агента в радианах*

*#вычисление крайних точек сектора видимости*

        for angle\_offset in [-half\_angle, half\_angle]:

            angle = angle\_center + mt.radians(angle\_offset)

            cone\_x = *self*.x + *self*.vision\_radius \* mt.sin(angle)

            cone\_y = *self*.y + *self*.vision\_radius \* mt.cos(angle)

            cone\_points.append((cone\_x, cone\_y))

*self*.overview = cone\_points

    def is\_in\_vision\_cone(*self*, *other\_agent*) -> bool:

        """

        Метод, который проверяет, попадает ли другой агент в область видимости текущего агента

        Args:

            other\_agent (Agent): проверяемый агент

        Returns:

            bool: True, если агент находится в секторе видимости, иначе False

        """

*self*.overview\_calculation()

        a, b, c = *self*.overview   *#координаты вершин треугольника конуса видимости*

*#функция для вычисления площади треугольника по координатам*

        def triangle\_area(*p1*, *p2*, *p3*):

            return abs((*p1*[0] \* (*p2*[1] - *p3*[1]) + *p2*[0] \* (*p3*[1] - *p1*[1]) + *p3*[0] \* (*p1*[1] - *p2*[1])) / 2.0)

        area\_ABC = triangle\_area(a, b, c)   *#полная площадь треугольника ABC*

*# Площадь треугольников, образованных точкой other и вершинами треугольника ABC*

        area\_ABP = triangle\_area(a, b, (*other\_agent*.x, *other\_agent*.y))

        area\_BCP = triangle\_area(b, c, (*other\_agent*.x, *other\_agent*.y))

        area\_CAP = triangle\_area(c, a, (*other\_agent*.x, *other\_agent*.y))

*#проверка: other\_agent находится внутри треугольника, если сумма площадей совпадает с полной площадью*

        return abs(area\_ABC - (area\_ABP + area\_BCP + area\_CAP)) < 1e-5   *#допустимая погрешность*

    def initial\_movement(*self*) -> None:

        """

        Метод, который инициализирует случайные параметры для начала перемещения агента

        Args:

            None

        Returns:

            None

        """

*self*.speed = rnd.randint(3, 7)

*self*.direction = rnd.randint(0, 360)

*self*.t\_move = rnd.randint(25, 50)

*self*.speed\_calculation()

*self*.overview\_calculation()

    def move\_iteration(*self*) -> None:

        """

        Метод, который выполняет итерацию перемещения агента исходя из параметров перемещения

        Args:

            None

        Returns:

            None

        """

*self*.x += *self*.speed\_x

*self*.y += *self*.speed\_y

*#если агент пытается пересечь границу по X - направление меняется как 360-текущее и пересчитывается скорость по X*

*#если агент путается пересечь границу по Y - направление меняется как 180-текущее и пересчитывается скорость по Y*

*#решено делать так, а не менять знак соответствующей скорости для того, чтобы корректно изменялось направление*

*# движение агента*

        if *self*.x < 0 or *self*.x > 100:

*self*.direction = 360 - *self*.direction

*self*.speed\_calculation()

*self*.x += *self*.speed\_x \* 2

        if *self*.y < 0 or *self*.y > 100:

*self*.direction = 180 - *self*.direction

*self*.speed\_calculation()

*self*.y += *self*.speed\_y \* 2

*self*.overview\_calculation()

    def who\_does\_see(*self*, *list\_of\_agents*: list) -> list:

        """

        Метод, который перебирает список всех агентов и ищет тех, кого видит агент

        Args:

            list\_of\_agents (list): Список всех агентов

        Returns:

            visible\_agents (list): Список id видимых агентов

        """

        visible\_agents = []

        for i in range(len(*list\_of\_agents*)):

            if *list\_of\_agents*[i].id != *self*.id:

                if *self*.is\_in\_vision\_cone(*list\_of\_agents*[i]):

                    visible\_agents.append(*list\_of\_agents*[i].id)

            else:

                continue

        return visible\_agents

    def calculate\_escape\_direction(*self*, *zombies\_left*, *zombies\_right*):

        """

        Метод для вычисления нового направления, чтобы избежать зомби

        Args:

            zombies\_left (bool): Есть ли зомби в левом секторе

            zombies\_right (bool): Есть ли зомби в правом секторе

        Returns:

            int: Новое направление агента

        """

        if *zombies\_left* and *zombies\_right*:

*#зомби с обеих сторон - поворот на 180 градусов*

            return (*self*.direction + 180) % 360

        elif *zombies\_right*:

*#зомби справа - поворот влево*

            return (*self*.direction - 90) % 360

        elif *zombies\_left*:

*#зомби слева - поворот вправо*

            return (*self*.direction + 90) % 360

        return *self*.direction   *#если нет зомби, направление не меняется*

    def move(*self*, *list\_of\_agents*: list) -> None:

        """

        Метод, который управляет перемещением агента.

        Если в поле зрения есть зомби, агент меняет направление.

        Args:

            list\_of\_agents (list): Список всех агентов

        Returns:

            None

        """

*#проверка на видимых агентов*

        visible\_agents = *self*.who\_does\_see(*list\_of\_agents*)

        zombies\_left = zombies\_right = False

        for agent\_id in visible\_agents:

            target\_agent = next((a for a in *list\_of\_agents* if a.id == agent\_id), None)

            if target\_agent.\_\_class\_\_.\_\_name\_\_ == "ZombieAgent":  *#проверка на зомби без импорта класса Зомби*

*#определяем угол до зомби*

                angle\_to\_zombie = mt.degrees(mt.atan2(target\_agent.y - *self*.y, target\_agent.x - *self*.x))

                angle\_to\_zombie = (angle\_to\_zombie + 360) % 360   *#приводим к диапазону [0, 360]*

*#определяем левый и правый пределы сектора обзора*

                left\_bound = (*self*.direction - *self*.vision\_angle / 2 + 360) % 360

                right\_bound = (*self*.direction + *self*.vision\_angle / 2) % 360

*#проверка положения зомби относительно центральной линии обзора*

                if left\_bound < right\_bound:

                    if left\_bound <= angle\_to\_zombie < *self*.direction:

                        zombies\_left = True

                    elif *self*.direction <= angle\_to\_zombie < right\_bound:

                        zombies\_right = True

                else:

*#если сектор пересекает границу 360 градусов*

                    if left\_bound <= angle\_to\_zombie < 360 or 0 <= angle\_to\_zombie < *self*.direction:

                        zombies\_left = True

                    elif *self*.direction <= angle\_to\_zombie < 360 or 0 <= angle\_to\_zombie < right\_bound:

                        zombies\_right = True

*#если есть зомби в поле зрения, меняем направление*

        if zombies\_left or zombies\_right:

            new\_direction = *self*.calculate\_escape\_direction(zombies\_left, zombies\_right)

*#print(f"Agent id={self.id} меняет направление на {new\_direction} чтобы убежать от зомби")*

*self*.direction = new\_direction

*self*.speed\_calculation()   *#пересчёт скоростей в новом направлении*

*#движение агента*

        if *self*.t\_move == 0:

*self*.initial\_movement()

*self*.move\_iteration()

*self*.t\_move -= 1

from Agent import \*

from ZombieAgent import \*

class InfectedAgent(Agent):

    def \_\_init\_\_(*self*, *id*: int, *x*: float, *y*: float, *incubation\_period*: int) -> None:

        """

        Метод-конструктор класса Заражённый агент, наследуется от класса Агент

        Поля:

            id (int): Id агента

            x (float): Координата агента по X

            y (float): Координата агента по Y

            vision\_radius (int): Радиус обзора агента

            vision\_angle (int): Угол обзора агента

            speed (int): Общая скорость агента

            speed\_x (float): Скорость агента по X

            speed\_y (float): Скорость агента по Y

            direction (int): Направление движения агента (в градусах (0 - вертикаль по Y+))

            overview (list): Координаты точек конуса видимости агента относительно направления движения

            t\_move (int): Время, в течении которого агент перемещается

            incubation\_period (int): Инкубационные период заражённого агента

        Args:

            id (int): Id заражённого агента

            x (float): Координата заражённого агента по X

            y (float): Координата заражённого агента по Y

            incubation\_period (int): Инкубационный период заражённого агента

        Returns:

            None

        """

        super().\_\_init\_\_(*id*, *x*, *y*)

*self*.incubation\_period = *incubation\_period*

    def move\_iteration(*self*) -> None:

        """

        Метод, который выполняет итерацию перемещения агента исходя из параметров перемещения

        Args:

            None

        Returns:

            None

        """

*#скорость заражённого агента = 90% скорости обычного агента (здорового)*

*self*.x += *self*.speed\_x \* 0.9

*self*.y += *self*.speed\_y \* 0.9

*#если агент пытается пересечь границу по X - направление меняется как 360-текущее и пересчитывается скорость по X*

*#если агент путается пересечь границу по Y - направление меняется как 180-текущее и пересчитывается скорость по Y*

*#решено делать так, а не менять знак соответствующей скорости для того, чтобы корректно изменялось направление*

*# движение агента*

        if *self*.x < 0 or *self*.x > 100:

*self*.direction = 360 - *self*.direction

*self*.speed\_calculation()

*self*.x += *self*.speed\_x \* 2 \* 0.9

        if *self*.y < 0 or *self*.y > 100:

*self*.direction = 180 - *self*.direction

*self*.speed\_calculation()

*self*.y += *self*.speed\_y \* 2 \* 0.9

*self*.overview\_calculation()

        if *self*.incubation\_period > 0:

*self*.incubation\_period -= 1

    def move(*self*, *list\_of\_agents*: list) -> None:

        """

        Метод, который управляет перемещением агента

        Args:

            list\_of\_agents (list): Список всех агентов

        Returns:

            None

        """

        if *self*.t\_move == 0:

*self*.initial\_movement()

*self*.move\_iteration()

        if *self*.incubation\_period <= 0:

*list\_of\_agents*[*list\_of\_agents*.index(*self*)] = ZombieAgent(

*self*.id, *self*.x, *self*.y, *self*.vision\_radius, *self*.vision\_angle)

        visible\_agents = *self*.who\_does\_see(*list\_of\_agents*)

        if len(visible\_agents) == 0:

*#print(f"Агент id = {self.id} никого не видит")*

            pass

        else:

*#print(f"Агент id = {self.id} видит агентов {visible\_agents}")*

            pass

*self*.t\_move -= 1

import math as mt

import random as rnd

from agent\_factory import replace\_agent

from Agent import Agent

class ZombieAgent(Agent):

    def \_\_init\_\_(*self*, *id*: int, *x*: float, *y*: float, *vision\_radius*: int, *vision\_angle*: int) -> None:

        """

        Метод-конструктор класса Зомби, наследуется от класса Агент

        Поля:

            id (int): Id агента

            x (float): Координата агента по X

            y (float): Координата агента по Y

            vision\_radius (int): Радиус обзора агента

            vision\_angle (int): Угол обзора агента

            speed (int): Общая скорость агента

            speed\_x (float): Скорость агента по X

            speed\_y (float): Скорость агента по Y

            direction (int): Направление движения агента (в градусах (0 - вертикаль по Y+))

            overview (list): Координаты точек конуса видимости агента относительно направления движения

            t\_move (int): Время, в течении которого агент перемещается

            action\_radius (int): Радиус действия зомби

            action\_angle (int): Угол действия зомби

            action\_overview (list): Координаты точек конуса сектора действия зомби относительно направления движения

            target\_agent (Agent): Цель преследования зомби

        Args:

            id (int): Id агента

            x (float): Координата агента по X

            y (float): Координата агента по Y

            vision\_radius (int): Радиус обзора агента

            vision\_angle (int): Угол обзора агента

        Returns:

            None

        """

        super().\_\_init\_\_(*id*, *x*, *y*)

*self*.vision\_radius = *vision\_radius* \* 1.1

*self*.vision\_angle = *vision\_angle* \* 0.75

*self*.action\_radius = *self*.vision\_radius \* 0.93

*self*.action\_angle = *self*.vision\_angle \* 0.93

*self*.action\_overview = []

*self*.target\_agent = None

    def overview\_calculation(*self*) -> None:

        """

        Метод, который вычисляет сектор действия зомби

        """

        super().overview\_calculation()

*#вычисляем сектор действия на основе action\_radius и action\_angle*

        action\_cone\_points = [(*self*.x, *self*.y)]   *#начальная точка сектора действия - позиция зомби*

        half\_action\_angle = *self*.action\_angle / 2

        angle\_center = mt.radians(*self*.direction)

        for angle\_offset in [-half\_action\_angle, half\_action\_angle]:

            angle = angle\_center + mt.radians(angle\_offset)

            action\_x = *self*.x + *self*.action\_radius \* mt.sin(angle)

            action\_y = *self*.y + *self*.action\_radius \* mt.cos(angle)

            action\_cone\_points.append((action\_x, action\_y))

*self*.action\_overview = action\_cone\_points   *#координаты для сектора действия*

    def is\_in\_action\_cone(*self*, *other\_agent*) -> bool:

        """

        Проверяет, находится ли другой агент в секторе действия зомби

        Args:

            other\_agent (Agent): проверяемый агент

        Returns:

            bool: True, если агент находится в секторе действия, иначе False

        """

*self*.overview\_calculation()

        a, b, c = *self*.action\_overview  *#координаты вершин треугольника сектора действия*

*#вычисление площади треугольников для проверки нахождения внутри сектора действия*

        def triangle\_area(*p1*, *p2*, *p3*):

            return abs((*p1*[0] \* (*p2*[1] - *p3*[1]) + *p2*[0] \* (*p3*[1] - *p1*[1]) + *p3*[0] \* (*p1*[1] - *p2*[1])) / 2.0)

        area\_ABC = triangle\_area(a, b, c)

        area\_ABP = triangle\_area(a, b, (*other\_agent*.x, *other\_agent*.y))

        area\_BCP = triangle\_area(b, c, (*other\_agent*.x, *other\_agent*.y))

        area\_CAP = triangle\_area(c, a, (*other\_agent*.x, *other\_agent*.y))

        return abs(area\_ABC - (area\_ABP + area\_BCP + area\_CAP)) < 1e-5

    def move\_iteration(*self*) -> None:

        """

        Метод, который выполняет итерацию перемещения агента исходя из параметров перемещения

        Args:

            None

        Returns:

            None

        """

        if *self*.target\_agent:

*#если есть цель, продолжаем движение к ней*

            angle\_to\_target = mt.degrees(mt.atan2(*self*.target\_agent.y - *self*.y, *self*.target\_agent.x - *self*.x))

*self*.direction = angle\_to\_target

*self*.speed\_calculation()   *#обновляем скорости в новом направлении*

*self*.x += *self*.speed\_x \* 0.85

*self*.y += *self*.speed\_y \* 0.85

        else:

*#скорость заражённого агента = 85% скорости обычного агента (здорового)*

*self*.x += *self*.speed\_x \* 0.85

*self*.y += *self*.speed\_y \* 0.85

*#если агент пытается пересечь границу по X - направление меняется как 360-текущее и пересчитывается скорость по X*

*#если агент путается пересечь границу по Y - направление меняется как 180-текущее и пересчитывается скорость по Y*

*#решено делать так, а не менять знак соответствующей скорости для того, чтобы корректно изменялось направление*

*# движение агента*

            if *self*.x < 0 or *self*.x > 100:

*self*.direction = 360 - *self*.direction

*self*.speed\_calculation()

*self*.x += *self*.speed\_x \* 2 \* 0.85

            if *self*.y < 0 or *self*.y > 100:

*self*.direction = 180 - *self*.direction

*self*.speed\_calculation()

*self*.y += *self*.speed\_y \* 2 \* 0.85

*self*.overview\_calculation()

    def move(*self*, *list\_of\_agents*: list) -> None:

        """

        Метод, который управляет перемещением зомби

        Args:

            list\_of\_agents (list): Список всех агентов

        Returns:

            None

        """

        if rnd.random() < 0.01:

*#print(f"Зомби id={self.id} выздоровел!")*

            replace\_agent(*list\_of\_agents*, *self*, "recovered")  *#выздоровление зомби*

            return  *#прекращаем выполнение, так как агент стал выздоровевшим*

        visible\_agents = *self*.who\_does\_see(*list\_of\_agents*)

*#проверка видимых агентов и выбор цели*

        if *self*.target\_agent == None:

            for agent\_id in visible\_agents:

                target\_agent = next((a for a in *list\_of\_agents* if a.id == agent\_id), None)

                if type(target\_agent) == type(Agent(0, 0, 0)):

*self*.target\_agent = target\_agent

                    break

*#проверка сектора действия и превращение цели в зомби*

        if *self*.target\_agent and *self*.is\_in\_action\_cone(*self*.target\_agent):

*#print(f"Агент id = {self.target\_agent.id} превращается в зомби!")*

            for i, agent in enumerate(*list\_of\_agents*):

                if agent.id == *self*.target\_agent.id:

*list\_of\_agents*[i] = ZombieAgent(*self*.target\_agent.id, *self*.target\_agent.x, *self*.target\_agent.y, *self*.target\_agent.vision\_radius, *self*.target\_agent.vision\_angle)

                    break

*self*.target\_agent = None

*#движение зомби*

        if *self*.t\_move == 0:

*self*.initial\_movement()

*self*.move\_iteration()

*self*.t\_move -= 1

from Agent import Agent

from agent\_factory import replace\_agent

import random as rnd

class RecoveredAgent(Agent):

    def \_\_init\_\_(*self*, *id*: int, *x*: float, *y*: float) -> None:

        """

        Конструктор класса Выздоровевший агент.

        Args:

            id (int): ID агента

            x (float): Начальная координата по X

            y (float): Начальная координата по Y

        """

        super().\_\_init\_\_(*id*, *x*, *y*)

    def move(*self*, *list\_of\_agents*: list) -> None:

        """

        Метод, который управляет перемещением выздоровевшего агента.

        Args:

            list\_of\_agents (list): Список всех агентов

        Returns:

            None

        """

*# Проверка на возможность повторного заражения*

        for agent\_id in *self*.who\_does\_see(*list\_of\_agents*):

            target\_agent = next((a for a in *list\_of\_agents* if a.id == agent\_id), None)

            if target\_agent and target\_agent.\_\_class\_\_.\_\_name\_\_ == "ZombieAgent" and target\_agent.is\_in\_action\_cone(*self*):

                if rnd.random() < 0.25:  *# Шанс 25% снова стать зомби*

*#print(f"Выздоровевший агент id={self.id} снова стал зомби!")*

                    replace\_agent(*list\_of\_agents*, *self*, "zombie")  *# Превращаем обратно в зомби*

                    return  *# Прекращаем выполнение, так как агент снова стал зомби*

*# Если не заразился, движется как обычный агент*

        if *self*.t\_move == 0:

*self*.initial\_movement()

*self*.move\_iteration()

*self*.t\_move -= 1

def replace\_agent(*agent\_list*, *agent*, *new\_type*):

    """

    Заменяет агента в списке на новый тип агента.

    Args:

        agent\_list (list): Список всех агентов

        agent (Agent): Агент, который должен быть заменён

        new\_type (str): Тип нового агента ('zombie' или 'recovered')

    """

    index = *agent\_list*.index(*agent*)

    if *new\_type* == "zombie":

        from ZombieAgent import ZombieAgent

*agent\_list*[index] = ZombieAgent(*agent*.id, *agent*.x, *agent*.y, *agent*.vision\_radius, *agent*.vision\_angle)

    elif *new\_type* == "recovered":

        from RecoveredAgent import RecoveredAgent

*agent\_list*[index] = RecoveredAgent(*agent*.id, *agent*.x, *agent*.y)

import matplotlib.pyplot as plt

from matplotlib.animation import FuncAnimation

import random as rnd

import math as mt

from Agent import \*

from InfectedAgent import \*

from ZombieAgent import \*

from RecoveredAgent import \*

def create\_all\_agents(*n*: int) -> list:

    """

    Функция, которая создаёт n агентов в случайных точках зоны 100x100

    Args:

        n (int): Кол-во агентов

    Returns:

        list\_of\_agents (list): Список всех агентов

    """

    list\_of\_agents = []

    for i in range(*n*):

        coords = [rnd.uniform(1, 99) for \_ in range(2)]

        agent = Agent(i + 1, coords[0], coords[1])

*#agent.info()*

        agent.initial\_movement()

        agent.overview\_calculation()

        list\_of\_agents.append(agent)

    return list\_of\_agents

def anim\_multiple\_agents(*list\_of\_agents*: list, *m*: int, *t\_inc\_min*: int, *t\_inc\_max*: int) -> None:

    """

    Функция, которая анимирует перемещение всех агентов с визуализацией их зон видимости

    и включает этап заражения и инкубации в одной анимации

    Args:

        list\_of\_agents (list): Список агентов

        m (int): Количество агентов, которые станут заражёнными

        t\_inc\_min (int): Минимальное время инкубации

        t\_inc\_max (int): Максимальное время инкубации

    Returns:

        None

    """

*#параметры анимации*

    walk\_frames = 50   *#количество кадров для "прогулки"*

    infection\_start = walk\_frames + 1   *#номер кадра, с которого начинается заражение*

*#настройка графика*

    fig, ax = plt.subplots()

    ax.set\_xlim(0, 100)

    ax.set\_ylim(0, 100)

*#создаём точку и конус видимости для каждого агента*

    points = []

    vision\_cones = []

    typing\_list = [Agent(0, 0, 0), InfectedAgent(0, 0, 0, 0), ZombieAgent(0, 0, 0, 0, 0), RecoveredAgent(0, 0, 0)]

    for agent in *list\_of\_agents*:

        if type(agent) == type(typing\_list[0]):

            color = "blue"

        elif type(agent) == type(typing\_list[1]):

            color = "green"

        elif type(agent) == type(typing\_list[2]):

            color = "red"

        point, = ax.plot([], [], *marker*="o", *markersize*=7, *color*=color)   *#точка-агент*

        vision\_cone = plt.Polygon([(agent.x, agent.y), (agent.x, agent.y), (agent.x, agent.y)], *color*='lightblue', *alpha*=0.3)   *#конус видимости агента*

        ax.add\_patch(vision\_cone)

        points.append(point)

        vision\_cones.append(vision\_cone)

*#инициализация анимации*

    def init():

        for point, vision\_cone, agent in zip(points, vision\_cones, *list\_of\_agents*):

            point.set\_data(agent.x, agent.y)

            vision\_cone.set\_xy(agent.overview)

        return points + vision\_cones

*#функция для обновления каждого кадра анимации*

    def update(*frame*):

*#этап 1: первая "прогулка" агентов*

        if *frame* <= walk\_frames:

            for i, agent in enumerate(*list\_of\_agents*):

                agent.move(*list\_of\_agents*)

                points[i].set\_data(agent.x, agent.y)

                vision\_cones[i].set\_xy(agent.overview)

*#этап 2: заражение определённых агентов*

        elif *frame* == infection\_start:

            for i in range(*m*):

                agent = *list\_of\_agents*[i]

                infected\_agent = InfectedAgent(agent.id, agent.x, agent.y, rnd.randint(*t\_inc\_min*, *t\_inc\_max*))

*list\_of\_agents*[i] = infected\_agent  *#заменяем на заражённого агента*

                points[i].set\_color("green")  *#обновляем цвет для заражённого*

*#этап 3: основная симуляция заражения и инкубации*

        else:

            for i, agent in enumerate(*list\_of\_agents*):

                agent.move(*list\_of\_agents*)   *#обновляем положение агента*

                points[i].set\_data(agent.x, agent.y)

                vision\_cones[i].set\_xy(agent.overview)

                if type(agent) == type(typing\_list[2]):

                    points[i].set\_color("red")

                if type(agent) == type(typing\_list[3]):

                    points[i].set\_color("yellow")

        return points + vision\_cones

*# Создание анимации*

    anim = FuncAnimation(fig, update, *frames*=150, *init\_func*=init, *blit*=True, *repeat*=False)

    plt.show()

def all\_simulations(*pair\_values*: list):

    """

    Функция, которая выполняет по 1000 симуляций для каждой пары значений

    Args:

        pair\_values (list): Список кортежей (пар) значений n и m

    Returns:

        None

    """

    list\_all\_times = []

    for pair in *pair\_values*:

        n = pair[0]

        m = pair[1]

        num\_of\_iterations = 0

        for i in range(1000):

            list\_of\_agents = create\_all\_agents(n)

            for \_ in range(25):

                for agent in list\_of\_agents:

                    agent.move(list\_of\_agents)

            for j in range(m):

                agent = list\_of\_agents[j]

                infected\_agent = InfectedAgent(agent.id, agent.x, agent.y, rnd.randint(t\_inc\_min, t\_inc\_max))

                list\_of\_agents[j] = infected\_agent

            num\_it = 0

            for j in range(1000):

                print(f"Итерация {j} симуляции {i} пары {pair}")

                for agent in list\_of\_agents:

                    agent.move(list\_of\_agents)

                num\_it += 1

                count\_zombie = 0

                for agent in list\_of\_agents:

                    if isinstance(agent, ZombieAgent):

                        count\_zombie += 1

                if count\_zombie == len(list\_of\_agents):

                    break

            num\_of\_iterations += num\_it

        list\_all\_times.append(num\_of\_iterations / 5)

        print(list\_all\_times)

        if i % 10 == 0:

            print(f"Прошло {i} симуляций для пары {pair}")

if \_\_name\_\_ == "\_\_main\_\_":

*#основные переменные*

    n = 10   *#кол-во агентов*

    m = 5   *#кол-во агентов, которые станут заражёнными*

    t\_inc\_min = 10   *#минимальное время инкубации*

    t\_inc\_max = 25   *#максимальное время инкубации*

    pair\_values = [(10, 5), (100, 10), (100, 25), (500, 50)]

*#создание списка всех агентов*

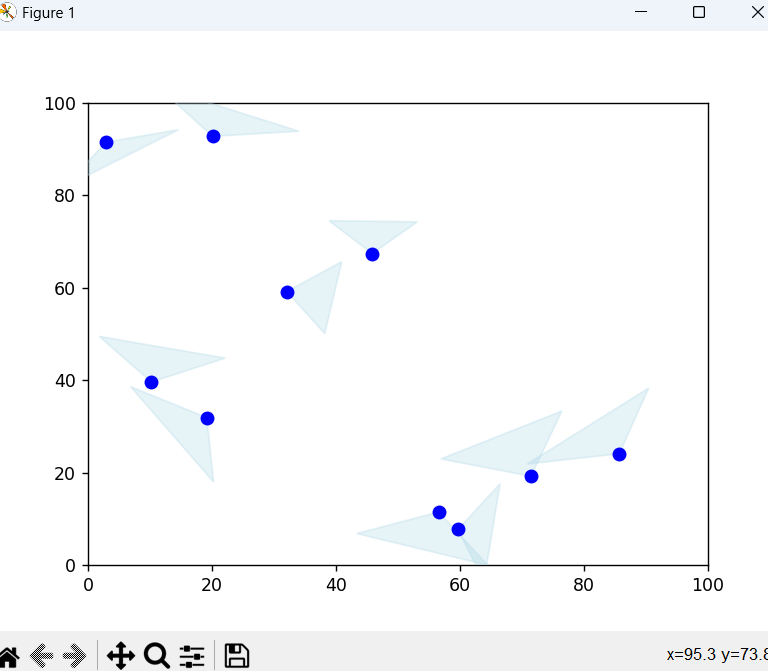
    list\_of\_agents = create\_all\_agents(n)

*#основная симуляция*

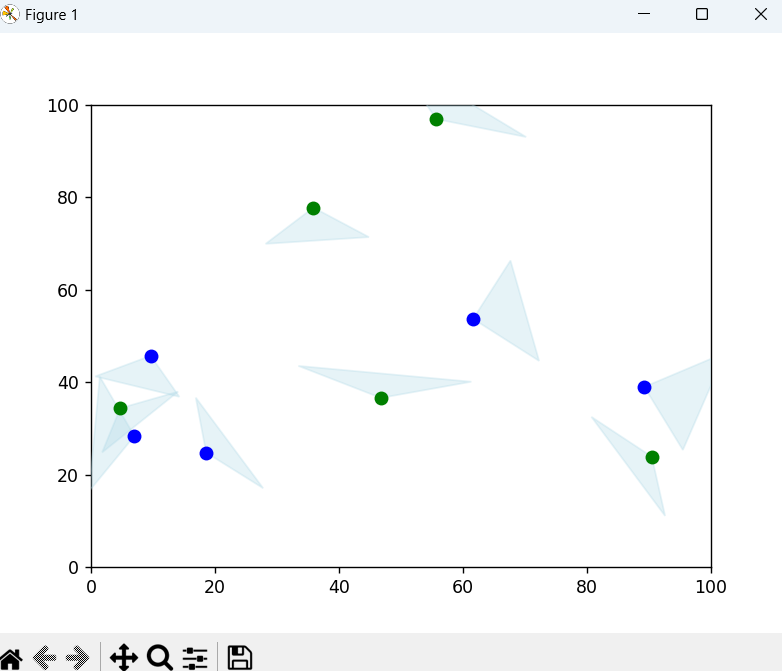
*#anim\_multiple\_agents(list\_of\_agents, m, t\_inc\_min, t\_inc\_max)*

    all\_simulations(pair\_values)

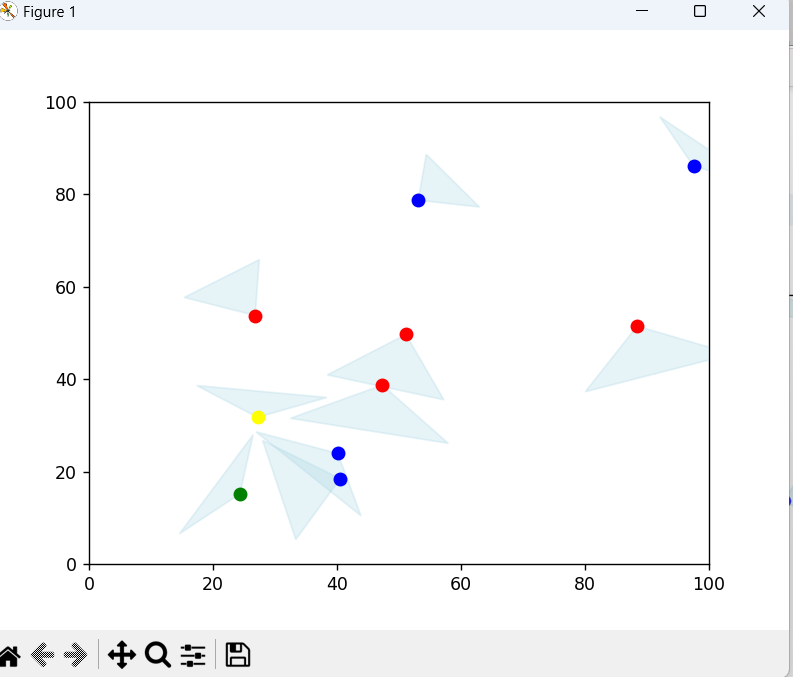
**Анимация симумляции (кадры)**

****

**Начальный этап, только здоровые агенты**

****

**Промежуточный этап, появление заражённых (зелёные)**

****

**Основная симуляция (красные – зомби, жёлтые - выздоровевшие)**

**Результирующая таблица**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во агентов и заражённых | 10, 5 | 100, 10 | 100, 25 | 500, 50 |
| Среднее время окончания симуляции | 416.6 | 500.0 | 477.0 | 340.0 |

Исходя из таблицы можно сделать вывод, что слишком большое кол-во агентов на карте приводит к ускорению эпидемии, в то время как относительно небольшое кол-во агентов не приводит к скачку скорости заражения.