МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ
ПЕДАГОГИЧЕСКИЙ УНИВЕРСИТЕТ им. А. И. ГЕРЦЕНА»

институт информационных технологий и технологического образования
кафедра информационных технологий и электронного обучения

Основная профессиональная образовательная программа
Направление подготовки 09.03.01 Информатика и вычислительная техника
Направленность (профиль) «Технологии разработки программного обеспечения»
форма обучения – очная

Курсовая работа

по дисциплине «Технологии компьютерного моделирования»

Компьютерное моделирование распространения вируса в популяции на основе модели клеточных автоматов

Обучающегося 2 курса
Величко Арсения Александровича

_____

Руководитель:
к.п.н, доцент
_____ Гончарова С. В.

«_____» _____ 2022 г.

Санкт-Петербург
2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Компьютерное моделирование — это наука о моделировании объектов и процессов реального мира посредством математического аппарата и ресурсов электронной вычислительной машины. Математический аппарат используется для составления математической модели некоторого явления реального мира. С помощью математического аппарата составляется математическая модель — формализованное описание некоторого явления на языке математики, позволяющее делать предположения о протекании и исходе исследуемого процесса без необходимости проведения натурных экспериментов. Таким образом, выводится некоторая закономерность, с помощью которой может быть предсказан исход схожего эксперимента.

Рабочую математическую модель можно впоследствии описать на языке алгоритмов и составить компьютерную программу для моделирования того или иного явления реального мира с помощью средств ЭВМ. Аппарат для реализации математических моделей в виде компьютерных программ предоставляет дисциплина «Компьютерное моделирование».

Таким образом может быть смоделировано любое , поддающееся описанию и математической формализации, явление реального мира, будь то некоторый физический процесс, статистически предсказуемое поведение некоторых экономических агентов или химическая реакция. Так, например, классическим предметом компьютерного моделирования являются физические процессы. С применением средств компьютерного моделирования может быть рассчитана траектория полета снаряда или аэродинамические свойства крыла самолета.

Целью данной курсовой работы является исследование технологий компьютерного моделирования клеточных автоматов и разработка компьютерной программы для моделирования распространения параметрируемого вируса среди

популяции параметрируемых бактерий. Для достижения поставленной цели потребуется выполнить следующие задачи:

1. Исследовать модель «клеточный автомат»;

2. Исследовать математическую игру «Жизнь» Джона Конвея;

3. Разработать модель симулятора распространения вируса в популяции на основе клеточных автоматов;

4. На основе полученной модели реализовать компьютерную программу для симуляции распространения вируса в популяции бактерий.

В ходе работы был произведен анализ проблем решения задач моделирования поведения масс живых существ и разработана программная реализация клеточного автомата.

Рисунок 1 - Применение компьютерного моделирования в медицине. Трехмерная модель зубов пациента

1. ИССЛЕДОВАНИЕ МОДЕЛИ «КЛЕТОЧНЫЙ АВТОМАТ»

1.1. Обзор модели «Клеточный автомат»

Примером достаточно простой модели поведения живых организмов можно назвать клеточный автомат. Клеточный автомат — дискретная модель, изучаемая в математике, теории вычислимости, физике, теоретической биологии и микромеханике. Основой является пространство из прилегающих друг к другу клеток (ячеек), образующих решетку. Каждая клетка может находиться в одном из конечного множества состояний (например, 1 и 0). Решётка может быть любой размерности, бесконечной или конечной, для решётки с конечными размерами часто предусматривается закольцованность при достижении предела (границы). Для каждой клетки определено множество клеток, называемых окрестностью. Например, окрестность фон Неймана ранга 2 включает все клетки на расстоянии не более 2 от текущей. Устанавливаются правила перехода клеток из одного состояния в другое. Обычно правила перехода одинаковы для всех клеток. Один шаг автомата подразумевает обход всех клеток и на основе данных о текущем состоянии клетки и её окрестности определение нового состояния клетки, которое будет у неё при следующем шаге. Перед стартом автомата оговаривается начальное состояние клеток, которое может устанавливаться целенаправленно или случайным образом[1].

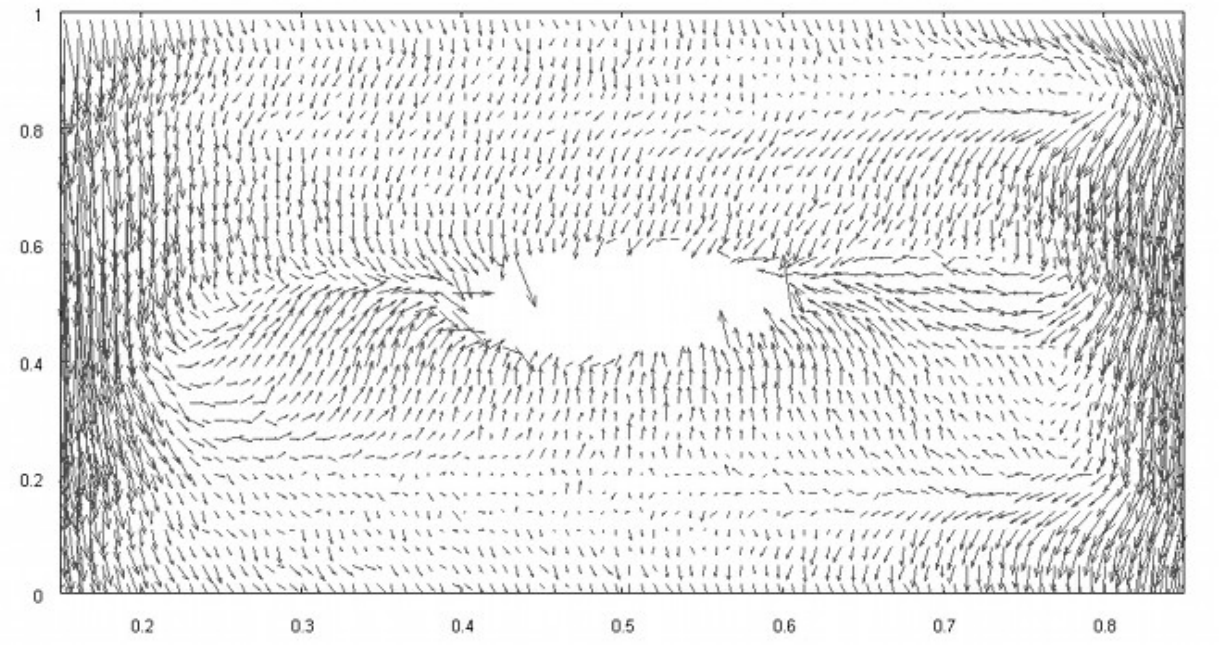

Рисунок 2 - Клеточный автомат для моделирования физики жидкостей

Клеточный автомат предлагает способ смоделировать поведение системы, состоящей их многочисленных объектов в некотором пространстве. Ключевым здесь является возможность представления одного объекта как обладателя одного из дискретных, конечных состояний. Таким образом, становится возможным составление математической модели через описание состояний, которые может принимать объект, и набора правил, согласно которым объект приобретает то или иное состояние.

1.2. Обзор математической игры «Жизнь» на основе модели «Клеточный автомат»

Рассмотрим один из наиболее известных клеточных автоматов — игру «Жизнь», предложенную английским математиком Джоном Конвеем в 1970 году. Суть этой «игры» можно описать следующим набором правил[2]:

1. Игра происходит на ограниченном или неограниченном клетчатом поле (плоскости);
2. Каждая клетка на поле имеет по 8 окружающих ее клеток-соседей;
3. В любой момент времени клетка принимает одно из двух состояний: клетка либо «жива» (заполнена) — 1, либо «мертва» (пуста) — 0;
4. В начале «игры» некоторые клетки заполняются. Далее на каждой итерации мертвая клетка, имеющая 3 живых соседа, становится живой, а живая клетка, имеющая менее двух или более трех живых соседей погибает;
5. Игра продолжается до тех пор, пока не будет выполнено одно из трех условий: либо все клетки погибнут, либо создастся устойчивая конфигурация (состояние поля перестанет меняться вместе с итерациями), либо сложится периодическая конфигурация (состояния поля начнут циклически воспроизводиться).

Рисунок 3 - Математическая игра "Жизнь"

Отличительная особенность игры «Жизнь» состоит в том, что достаточно сложной представляется задача предсказать конечное состояние всей системы, не просчитывая состояния ее отдельных частей. В этом и состоит сходство клеточных автоматов с системами реального мира. Так, например, в колонии бактерий отдельная особь не имеет информации обо всей колонии в целом. Она располагает лишь информацией о своем собственном состоянии и состоянии ближайших соседей. Именно этим определяются дальнейшие изменения ее состояния. Аналогию также можно провести со стаями птиц или косяками рыб.[3]

1.3. Разработка модели распространения параметрируемого вируса среди популяции параметрируемых бактерий

Как можно видеть из примера, математическая модель этой игры — лишь набор правил и состояний, которые в ней могут принимать клетки. Используя тот же подход, можно составить математическую модель и для некоторых явлений реального мира. Такая математическая модель может быть положена в основу компьютерной программы, при применении к ней методов компьютерного моделирования.

С опорой на принципы работы игры «Жизнь» Конвея, составим следующую модель:

1. Действие происходит на некоторой ограниченной плоскости (поле) размером $m \times n$, где m — ширина поля, а n - высота. Существа не могут выходить за пределы поля;

2. На поле находится некоторая колония существ (бактерий). Количество существ на поле p постоянно и не меняется в ходе симуляции;

3. Каждое существо имеет некоторое значение угла φ — направления, в котором оно повернуто. С каждой итерацией существо меняет направление на случайную величину в пределах v градусов в большую или меньшую сторону и сдвигается на некоторое расстояние l. Так достигается некоторое подобие броуновского движения существ по полю;

4. В ситуации, когда следующее передвижение существа в направлении φ невозможно, поскольку приведет к выходу существа за границы поля, существо поворачивается на 60° по часовой стрелке до тех пор, пока передвижение не станет возможно;

5. В любой отдельно взятый момент времени существо имеет одно из четырех состояний: 0 — существо мертво, 1 — существо живо и здорово, 2 — существо

заражено вирусом и может заражать им других существ, 3 — существо выздоровело и приобрело иммунитет к вирусу, то есть не может ни заразиться само, ни передать вирус другому существу;

6. В начале симуляции p существ помещаются в случайных точках поля и имеют случайные направления φ. i существ заражены вирусом (имеют состояние 2), остальные $p - i$ существ имеют состояние 1;

7. Существо в состоянии 1 (здорово), находящееся на расстоянии, не меньшем, чем радиус заражения r от существа в состоянии 2 (переносчика вируса) заражается вирусом (приобретает состояние 2) с вероятностью передачи c;

8. Существо сохраняет состояние 2 в течении t итераций (тиков);

9. По истечении t итераций, существо погибает (приобретает состояние 0) от вируса с вероятностью mt, либо выздоравливает с вероятностью $1 - mt$ и приобретает к вирусу иммунитет (переходит в состояние 3);

10. Погибшее от вируса существо (имеет состояние 0) перестает перемещаться по полю и не может ни быть заражено, ни передать вирус другому существу;

11. Симуляция продолжается до тех пор, пока на поле либо не останется живых существ, либо дальнейшая передача вируса станет невозможна (на после не останется существ в состоянии 2).

Состояния существ и возможные варианты их изменения представлены на рисунке 4.

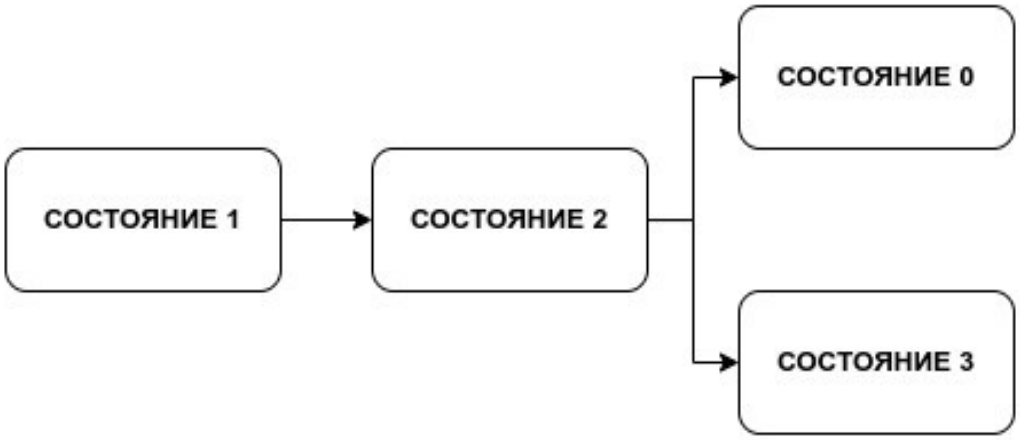

Рисунок 4 - Переход существа между состояниями

1.4. Проблемы компьютерного моделирования. Проблемы моделирования поведения живых существ.

Важно понимать, что ни одна модель не может дать абсолютно достоверных результатов. Это связано с тем, что объекты реального мира подвергаются влиянию неограниченно большого количества факторов, которые не могут быть в полной мере отражены в математической модели. Из этого следует, что ни одна модель не может дать абсолютно достоверно точных результатов, однако может предоставить достаточно точный для исследователя результат. То, какая точность результата будет достаточной, зависит от задач и целей исследователя. Так или иначе, для абсолютного большинства исследователей благоприятным исходом будет получение модели, дающей максимально возможную точность при имеющихся ресурсах.

Таким образом, технологии компьютерного моделирования могут быть применены для симуляции поведения масс некоторых живых существ. Так, может быть предсказана миграция животных или выживаемость колонии бактерий. Математические модели таких явлений основаны на предположении о том, что живые существа, в основной своей массе, действуют рационально. Это означает, что их поведение в значительной мере может быть предсказано, а значит такой процесс поддается формализованному описанию.

На основании этих данных в последствии выводится гипотеза о том, что в той или иной ситуации, либо под влиянием некоторых факторов, рассматриваемые живые существа, вероятнее всего, поведут себя определенным образом. Если такая гипотеза состоятельна, что может быть подтверждено, например, статистически в ходе экспериментов и наблюдений, гипотеза может быть положена в основу математической модели явления.

Как было сказано ранее, модель не может в полной мере описать все факторы реального мира, воздействующие на предмет исследования. По этой

причине, ко всем результатам моделирования следует относиться с некоторой долей скепсиса, не принимая их за абсолютно достоверное предсказание. Особенно это справедливо для моделей, предсказывающих поведение живых существ, поскольку даже существа, не способные к высшей нервной деятельности, могут действовать в разных ситуациях непредсказуемо, поэтому их поведение в таких условиях следует воспринимать как некоторый элемент случайности, неизвестности в математической модели.

2. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ СИМУЛЯТОРА РАСПРОСТРАНЕНИЯ ВИРУСА В ПОПУЛЯЦИИ БАКТЕРИЙ

По окончании разработки математической модели была начата разработка компьютерной программы для реализации симуляции. Симуляция параметрируема. Параметры симуляции могут быть заданы путём изменения значений переменных внутри кода компьютерной программы. Названия переменных и соответствующих им параметров математической модели, а также их смысл, приведены в таблице 1.

Таблица 1 — Изменяемые параметры программы

Параметр в мат. Модели	**Обозначение в программе**	**Смысл**
n	FIELD_HEIGHT	Высота игрового поля
m	FIELD_WIDTH	Ширина игрового поля
p	CREATURE_COUNT	Стартовое количество существ
i	INFECTED_INIT_COUNT	Стартовое количество зараженных существ
l	TICK_MOVE	Перемещение существа за один ход
v	FACING_VARIATION	Модуль максимального изменения направления за один ход

Параметр в мат. Модели	Обозначение в программе	Смысл
c	infectiousness	Вероятность передачи вируса существу (заразность вируса)
mt	lethality	Вероятность летального исхода болезни (летальность вируса)
r	transmission_radius	Максимальная дальность передачи вируса
t	lifetime	Продолжительность течения болезни

В качестве языка для разработки компьютерной программы был выбран язык программирования «Python 3»[4], для отображения игрового поля была использована библиотека «Pygame»[5], для вывода сообщений журнала событий (логов) использовалась библиотека «Loguru»[6]. Весь исходный программный код был опубликован в публичном репозитории[7] на платформе «Github»[8].

Для разработки программы был выбран объектно ориентированный подход. Данный подход позволяет связывать (инкапсулировать) данные (переменные) и операции, производимые над ними (функции) в единые сущности (классы) и создавать множество независимых экземпляров одного класса (объектов). Данный подход был выбран, поскольку наиболее точно отражает суть модели клеточных автоматов — независимость сущностей, существующих в рамках игрового поля.

Были описаны классы вируса и существа, а также несколько служебных классов для хранения таких данных, как информация о местоположении существа, статистика игрового поля. Каждый класс обладает своим набором

атрибутов (полей) и функций, использующих данные этих атрибутов (методов класса). Поля и методы разделены на приватные (используемые только внутри самого класса) и публичные (предназначенные для использования за пределами класса).

Класс вируса является простым классом-хранилищем данных и имеет лишь набор публичных полей для хранения данных о конкретном объекте вируса. Объект (экземпляр класса) вируса хранит в себе данные о названии вируса, заразности вируса, его летальности, максимальном расстоянии передачи и продолжительности течения болезни.

Класс существа имеет публичные поля для хранения имени существа, текущего цвета существа (зависит от состояния), логического значения, указывающего на то, живо ли существо, объекта позиции существа, остатка продолжительности болезни и объекта вируса (в случае, если существо заражено вирусом), а также множества вирусов, к которым существо приобрело иммунитет. Существо также обладает двумя полями, значения которых вычисляются в момент обращения (свойствами). Оба этих поля логические, их значения указывают на то, является ли существо переносчиком вируса в данный момент и на то, является ли существо реципиентом вируса, то есть может ли оно быть заражено.

Класс существа имеет два публичных метода. Один из них отвечает за заражение существа вирусом, а второй — за обновление состояния существа (вызывается для всех существ на каждой итерации). Помимо публичных методов, существо имеет набор приватных методов, вызываемых методом обновления состояния. Каждый из этих приватных методов выполняет свою задачу: перемещение существа на новую позицию, обновление статуса болезни и так далее.

Диаграмма всех описанных выше классов представлена на рисунке 5.

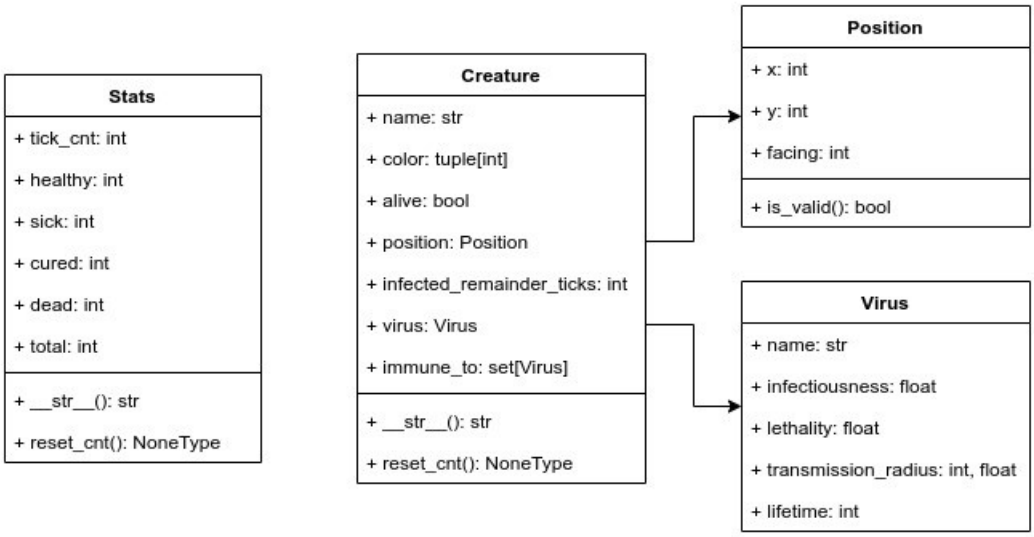

Рисунок 5 - Диаграмма классов

Таким образом, на верхнем уровне программа взаимодействует лишь с набором объектов существ, вызывая на каждой итерации их публичные методы для обновления состояния и, при необходимости, передачи им вируса. Преимущество данной реализации состоит в том, что объекты существ не зависят друг от друга и могут рассматриваться в отрыве от игрового поля и остальных существ на нем, что дает широкие возможности для изменения и расширения кода программы, введения новых сценариев работы и правил, динамического изменения параметров симуляции. То же справедливо и для вирусов: новые объекты вирусов могут быть быстро добавлены при необходимости, например, симуляции распространения нескольких видов вируса в популяции.

Логика верхнего уровня компьютерной программы — это, в данном случае, отраженные в программном коде некоторые правила, описанные в математической модели. К ним относятся процесс передачи вируса, логика

визуального представления состояния игрового поля. Действия, выполняемые программой при старте симуляции, будем называть этапом инициализации программы.

Этап инициализации программы включает в себя несколько необходимых для подготовки к началу симуляции действий. Перед началом симуляции программа запрашивает у операционной системы ЭВМ создание графического окна для вывода изображения поля, инициализирует (создает экземпляры классов) существ в необходимом количестве, инициализирует объект для сохранения статистики. На этом этапе создаются объекты существ в количестве, заданном параметрами симуляции. Все создаваемые существа имеют состояние 1. Существа размещаются в случайных точках игрового поля. Указанное в параметрах симуляции количество существ, случайно выбранных из общего множества существ, заражается вирусом.

По завершении инициализации программы, создания игрового поля и существ на нем, заражения существ вирусом, начинается сама симуляция. Симуляция представляет из себя недетерминированный циклический вычислительный процесс, то есть цикл. Количество итераций цикла заранее неизвестно и зависит от хода симуляции и ее продолжительности. Симуляция (главный цикл) прерывается либо при выполнении условий, описанных в математической модели симуляции, либо по запросу пользователя программы. Каждая итерация главного цикла — тик — влечет за собой обновление состояния игрового поля и всех существ на нём.

На каждой итерации симуляции для каждого существа на поле вызывается его публичный метод обновления состояния. В результате этого вызова существо перемещается, то есть приобретает новое положение, может изменить свое состояние. Для каждого зараженного существа производится проверка всех остальных незараженных существ на поле: для каждого существа вычисляется его

удаление от данного и, в случае, если удаление существа не превышает максимальной дальности передачи вируса, вызывается публичный метод существа для заражения его вирусом. Каждый тик обновляется статистика симуляции.

Состояние игрового поля представляется графически в отдельном окне средствами библиотеки «Pygame». Размер окна измеряется в пикселях и соответствует размеру игрового поля. Окно симуляции обновляется каждый тик. Для каждого существа на поле в месте, соответствующем положению существа, рисуется круг (точка). Круг окрашивается одним из следующих цветов в зависимости от состояния существа, которому он соответствует: черный — для существ в состоянии 0 (существо мертво), синий — для существ в состоянии 1 (существо живо и здорово), красный — для существ в состоянии 2 (существо заражено вирусом и может заражать им других существ), зеленый — для существ в состоянии 3 (существо выздоровело и приобрело иммунитет к вирусу). Для каждого существа в состоянии 2 также рисуется окружность радиусом, равным максимальной дальности передачи вируса с центром в месте расположения существа. Таким образом, окружность обозначает зону, попав в которую, существо в состоянии 1 может быть заражено. В левом верхнем углу окна симуляции отображается статистика: номер тика, количество существ в состояниях 0, 1, 2 и 3, общее количество существ на поле.

По окончании каждого тика программа ждет в течении 10 мс прежде чем перейти к следующей итерации. Это сделано для того, чтобы существа не перемещались по полю слишком быстро и за ними было удобнее наблюдать. По окончании симуляции, программа ждет в течении 5 секунд прежде чем запустить новую симуляцию. Выполнение программы, таким образом, продолжается до тех пор, пока она не будет остановлена пользователем.

Логика работы программы на верхнем уровне может быть описана блок-схемой, представленной на рисунке 6.

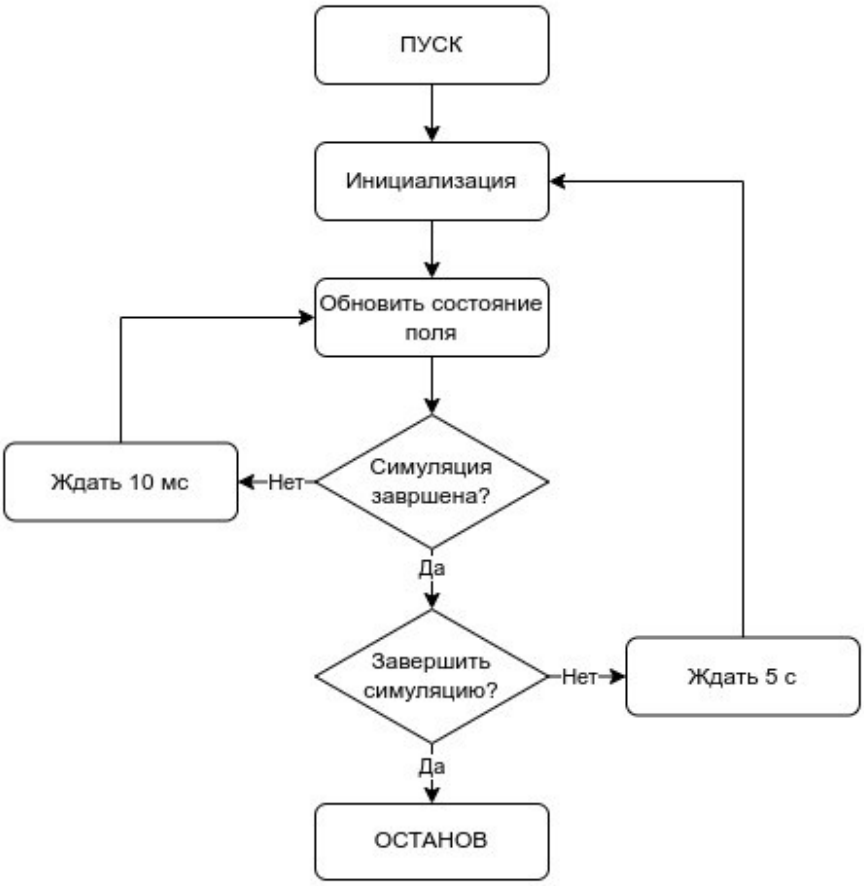

Рисунок 6 - Обобщенная блок-схема программы

Пример окна запущенной компьютерной программы представлен на рисунках 7 и 8.

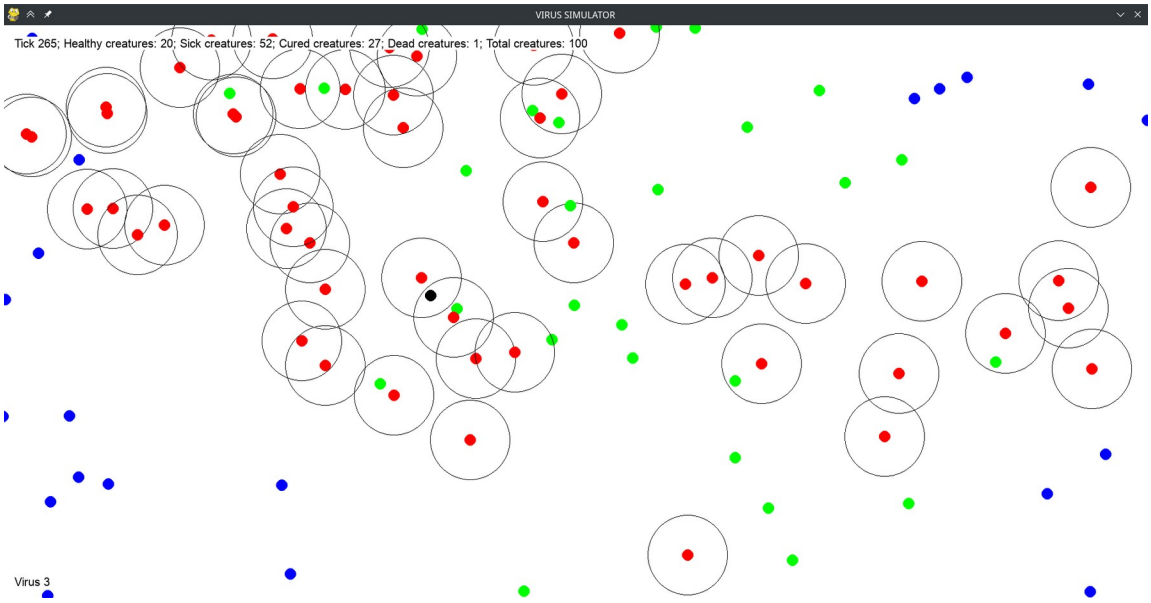

Рисунок 7 - Пример работы программы

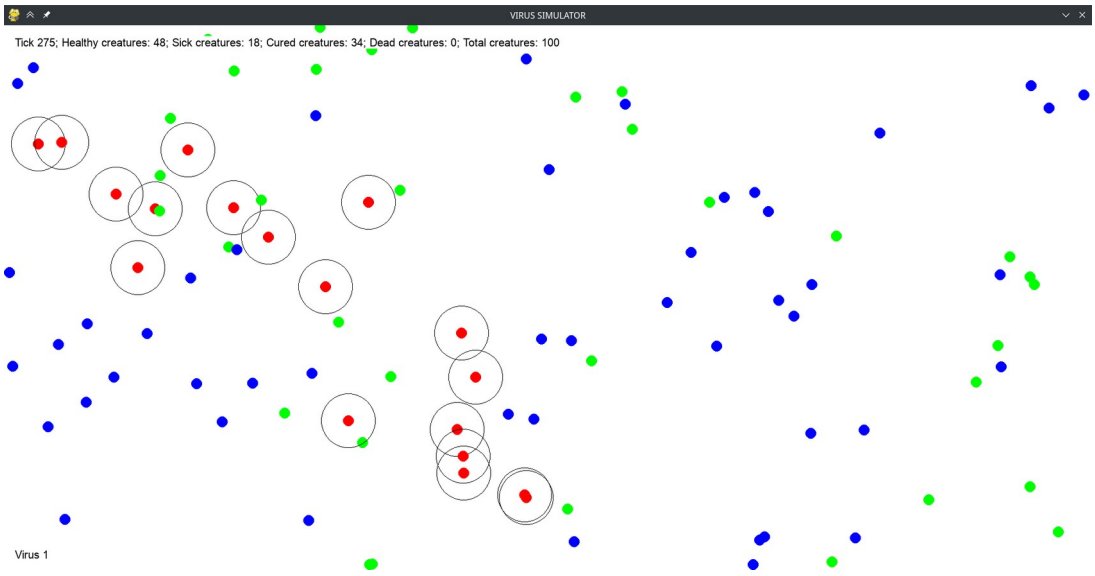

Рисунок 8 - Пример работы программы

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной курсовой работы были изучены методы компьютерного моделирования, способы применения средств и технологий компьютерного моделирования для моделирования объектов и явлений реального мира. Была проанализирована модель клеточного автомата и ее частное проявление — математическая игра «Жизнь» Джона Конвея. На основании модели клеточного автомата и принципов работы математической игры «Жизнь» была предложена реализация клеточного автомата для симуляции распространения параметрируемого вируса в популяции параметрируемых бактерий. Для предложенного клеточного автомата была составлена математическая модель и описана программная реализация. На основании предложенной модели была реализована компьютерная программа средствами языка программирования «Python» и некоторых сторонних библиотек.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Тоффоли Т., Марголус Н. Машины клеточных автоматов, М.: «Мир», 1991. ISBN 5-03-001619-8

2. Andrew Adamatzky. Game of Life Cellular Automata. — Springer-Verlag London, 2010. — ISBN 978-1-84996-216-2, 978-1-4471-6154-7, 978-1-84996-217-9. — doi:10.1007/978-1-84996-217-9.

3. Миллер, П. Роевой интеллект: Муравьи, пчелы и птицы способны многому нас научить// National Geographic Россия. — 2007. — № 8. — С. 88—107.

4. Сайт языка программирования Python: [электронный ресурс]. Режим доступа: https://www.python.org (дата обращения 01.05.2022)

5. Сайт библиотеки Pygame для языка программирования Python: [электронный ресурс]. Режим доступа: https://www.pygame.org/wiki/about (дата обращения 01.05.2022)

6. Сайт библиотеки Loguru для языка программирования Python: [электронный ресурс]. Режим доступа: https://www.pygame.org/wiki/about (дата обращения 01.05.2022)

7. Репозиторий системы контроля версий Git, содержащий исходные коды компьютерной программы: [электронный ресурс]. Режим доступа: https://github.com/arseniiarsenii/virus-simulator-2022 (дата обращения 01.05.2022)

8. Платформа Github для публикации репозиториев системы контроля версий Git: [электронный ресурс]. Режим доступа: https://github.com (дата обращения 01.05.2022)

Исходный код. Файл "creature.py"

```
import math
import typing as tp
from dataclasses import dataclass, field
from random import randint

from loguru import logger

from parameters import (
    COLOR_BLACK,
    COLOR_BLUE,
    COLOR_GREEN,
    COLOR_RED,
    CREATURE_COUNT,
    Color,
    FACING_VARIATION,
    INFECTED_INIT_COUNT,
    TICK_MOVE,
)
from position import Position
from utils import choose_with_probability
from virus import Virus


@dataclass
class Creature:
    """A creature descriptor object"""

    name: tp.Union[str, int]
    color: Color = COLOR_BLUE
    alive: bool = True
    position: Position = field(
        default_factory=lambda: Position()
    )
    infected_remainder_ticks: int = 0
    virus: tp.Optional[Virus] = None
    immune_to: tp.Set[Virus] = field(default_factory=set)

    @property
    def is_sick(self) -> bool:
        return self.virus is not None
```

```
@property
def is_recipient(self) -> bool:
    if self.is_sick or not self.alive or self.immune_to:
        return False
    return True

def tick_update(self) -> None:
    if not self.alive:
        return

    self._move()
    self._update_virus_status()

def _move(self) -> None:
    def get_new_position(facing: int) -> Position:
        return Position(
            facing=facing
            + randint(
                -FACING_VARIATION, FACING_VARIATION
            ),
            x=int(
                self.position.x
                + math.cos(math.radians(facing))
                * TICK_MOVE
            ),
            y=int(
                self.position.y
                + math.sin(math.radians(facing))
                * TICK_MOVE
            ),
        )

    facing = self.position.facing
    position = get_new_position(facing)

    while not position.is_valid:
        facing = (facing + 60) % 360
        position = get_new_position(facing)

    self.position = position
```

```
        def _update_virus_status(self) -> None:
            if not self.is_sick:
                return

            assert self.virus is not None

            if self.infected_remainder_ticks > 0:
                self.infected_remainder_ticks -= 1

            if self.infected_remainder_ticks == 0:
                lethal_outcome: bool = choose_with_probability(
                    self.virus.lethality
                )

                if lethal_outcome:
                    self._die()
                else:
                    self._cure()

        def _die(self) -> None:
            assert self.virus is not None
            self.alive = False
            self.color = COLOR_BLACK
            logger.info(
                f"{self.name} has died from {self.virus.name}"
            )
            self.virus = None

        def _cure(self) -> None:
            assert self.virus is not None
            self.immune_to.add(self.virus)
            self.color = COLOR_GREEN
            logger.info(
                f"{self.name} has cured from {self.virus.name} and became immune
to it"
            )
            self.virus = None

        def infect(self, virus: Virus) -> None:
            if self.is_sick or virus in self.immune_to:
                return

            if choose_with_probability(virus.infectiousness):
                self.infected_remainder_ticks = virus.lifetime
                self.color = COLOR_RED
```

```
            self.virus = virus
            logger.info(
                f"{self.name} has been infected with {virus.name}"
            )

def prepare_creatures(virus: Virus) -> tp.List[Creature]:
    creatures = [
        Creature(name=f"Creature {n}")
        for n in range(1, CREATURE_COUNT + 1)
    ]

    for i in range(INFECTED_INIT_COUNT):
        creatures[i].infect(virus)

    return creatures
```

Исходный код. Файл "main.py"

```
import sys
import time
from itertools import cycle

import pygame
from loguru import logger

import utils
from creature import prepare_creatures
from parameters import (
    COLOR_BLACK,
    COLOR_WHITE,
    CREATURE_COUNT,
    FIELD_HEIGHT,
    FIELD_WIDTH,
    VIRUSES,
)
from stats import Stats

pygame.init()
SCREEN = pygame.display.set_mode(
    (FIELD_WIDTH, FIELD_HEIGHT)
)
pygame.display.set_caption("VIRUS SIMULATOR")
```

```
viruses = cycle(VIRUSES)
virus = next(viruses)
CREATURES = prepare_creatures(virus)
running = True
STATS = Stats()

while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()

    if not running:
        continue

    SCREEN.fill(COLOR_WHITE)
    STATS.tick_cnt += 1

    for i in range(CREATURE_COUNT):
        CREATURES[i].tick_update()
        creature = CREATURES[i]

        if creature.is_sick:
            assert creature.virus is not None

            pygame.draw.circle(
                SCREEN,
                COLOR_BLACK,
                (creature.position.x, creature.position.y),
                creature.virus.transmission_radius,
                1,
            )

            for j in filter(
                lambda x: CREATURES[x].is_recipient,
                range(CREATURE_COUNT),
            ):
                distance = utils.get_distance(
                    creature.position, CREATURES[j].position
                )

                if (
                    distance
                    <= creature.virus.transmission_radius
```

```
                and CREATURES[j].alive
            ):
                CREATURES[j].infect(creature.virus)

        STATS.sick += 1
    elif not creature.alive:
        STATS.dead += 1
    elif creature.immune_to:
        STATS.cured += 1
    else:
        STATS.healthy += 1

    pygame.draw.circle(
        SCREEN,
        creature.color,
        (creature.position.x, creature.position.y),
        10,
    )

font_size = 20
font = pygame.font.SysFont("Arial", font_size)
stats = font.render(
    str(STATS), True, COLOR_BLACK, COLOR_WHITE
)
SCREEN.blit(stats, (20, 20))
virus_name = font.render(
    virus.name, True, COLOR_BLACK, COLOR_WHITE
)
SCREEN.blit(
    virus_name, (20, FIELD_HEIGHT - 20 - font_size)
)
pygame.display.update()

if STATS.sick == 0:
    logger.info(
        "No sick creatures left. Collective immunity achieved"
    )
    running = False

if STATS.dead == CREATURE_COUNT:
    logger.info("All creatures have died")
    running = False

if not running:
```

```
            virus = next(viruses)
            CREATURES = prepare_creatures(virus)
            running = True
            STATS = Stats()
            time.sleep(5)

        STATS.reset_cnt()

        time.sleep(0.01)
```

Исходный код. Файл "parameters.py"

```
import typing as tp

import virus

# Configuration
FIELD_HEIGHT = 1000
FIELD_WIDTH = 2000
CREATURE_COUNT = 100
INFECTED_INIT_COUNT = 3
TICK_MOVE = 5
FACING_VARIATION = 30
VIRUSES = [virus.virus_1, virus.virus_2, virus.virus_3]

# Colors
Color = tp.Tuple[int, int, int]
COLOR_BLACK: Color = (0, 0, 0)
COLOR_RED: Color = (255, 0, 0)
COLOR_GREEN: Color = (0, 255, 0)
COLOR_BLUE: Color = (0, 0, 255)
COLOR_WHITE: Color = (255, 255, 255)
```

Исходный код. Файл "position.py"

```
from dataclasses import dataclass, field
from random import randint

from parameters import FIELD_HEIGHT, FIELD_WIDTH
```

```
@dataclass(frozen=True)
class Position:
    """A position descriptor object"""
    x: int = field(
        default_factory=lambda: randint(0, FIELD_WIDTH)
    )
    y: int = field(
        default_factory=lambda: randint(0, FIELD_HEIGHT)
    )
    facing: int = field(
        default_factory=lambda: randint(0, 360)
    )

    @property
    def is_valid(self) -> bool:
        return (
            0 <= self.x <= FIELD_WIDTH
            and 0 <= self.y <= FIELD_HEIGHT
        )
```

Исходный код. Файл "stats.py"

```
from dataclasses import dataclass


@dataclass
class Stats:
    tick_cnt: int = 0
    healthy: int = 0
    sick: int = 0
    cured: int = 0
    dead: int = 0

    def total(self) -> int:
        return sum(
            (self.healthy, self.sick, self.cured, self.dead)
        )

    def __str__(self) -> str:
        return "; ".join(
            [
```

```
                f"Tick {self.tick_cnt}",
                f"Healthy creatures: {self.healthy}",
                f"Sick creatures: {self.sick}",
                f"Cured creatures: {self.cured}",
                f"Dead creatures: {self.dead}",
                f"Total creatures: {self.total}",
            ]
        )

    def reset_cnt(self) -> None:
        self.healthy = 0
        self.sick = 0
        self.cured = 0
        self.dead = 0
```

Исходный код. Файл "utils.py"

```
import math
import typing as tp
from random import randint

from position import Position


def choose_with_probability(
    probability: tp.Union[int, float]
) -> bool:
    """Chose with certain probability"""
    if 0 <= probability <= 1:
        probability = int(probability * 100)

    return randint(0, 100) <= probability


def get_distance(pos_1: Position, pos_2: Position) -> float:
    """Get distance between two given points`"""
    x_dist = abs(pos_1.x - pos_2.x)
    y_dist = abs(pos_1.y - pos_2.y)
    return math.sqrt(x_dist**2 + y_dist**2)
```

Исходный код. Файл "virus.py"

```python
import typing as tp
from dataclasses import dataclass


@dataclass(frozen=True)
class Virus:
    """A virus descriptor object"""

    name: str
    infectiousness: float
    lethality: float
    transmission_radius: tp.Union[int, float]
    lifetime: int


virus_1 = Virus(
    name="Virus 1",
    infectiousness=0.9,
    lethality=0.01,
    transmission_radius=50,
    lifetime=100,
)

virus_2 = Virus(
    name="Virus 2",
    infectiousness=0.95,
    lethality=0.02,
    transmission_radius=120,
    lifetime=140,
)

virus_3 = Virus(
    name="Virus 3",
    infectiousness=0.95,
    lethality=0.12,
    transmission_radius=70,
    lifetime=170,
)
```