Научное общество учащихся «Эврика»

Муниципальное бюджетное образовательное учреждение

«Лицей №87 им. Л. И. Новиковой»

Московского района г. Нижнего Новгорода

Применение генетических алгоритмов при симуляции эволюции

Выполнил: Нечаев Никита

Ученик 9б класса

Научный руководитель:

Желаннова О. В.

Учитель информатики

Нижний Новгород

2021г.

Содержание

1. Введение

[1.1 Пример задачи для генетического алгоритма 3](#_Toc65516240)

[1.2 Процесс решения 4](#_Toc65516241)

[1.3 Использование генетических алгоритмов 5](#_Toc65516242)

[2. Симуляция эволюции 7](#_Toc65516243)

[2.1 Среда разработки 7](#_Toc65516244)

[2.2 Базовые алгоритмы работы 7](#_Toc65516245)

[2.3 Двумерный массив 8](#_Toc65516246)

[2.4 Биологические начала 10](#_Toc65516247)

[2.5 Концепция бота 11](#_Toc65516248)

[3. Строение бота 14](#_Toc65516249)

[3.1 Механизмы питания и размножения 14](#_Toc65516250)

[3.2 Коэффициенты, влияющие на жизнь 18](#_Toc65516251)

[3.3 Правила перемещения 18](#_Toc65516252)

[3.4 Устройство стен 18](#_Toc65516253)

[4. Практика 20](#_Toc65516254)

[4.1 Рандомная генерация 24](#_Toc65516255)

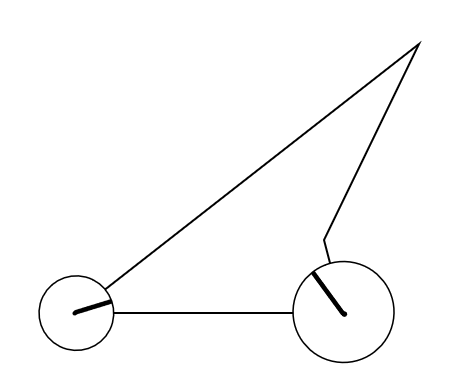
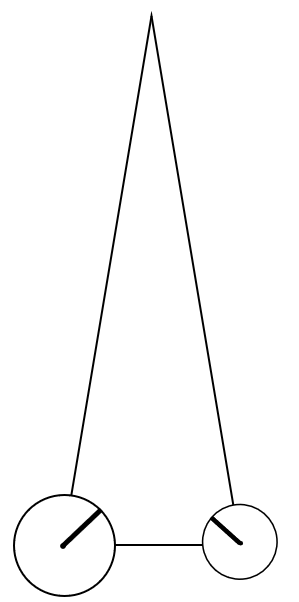
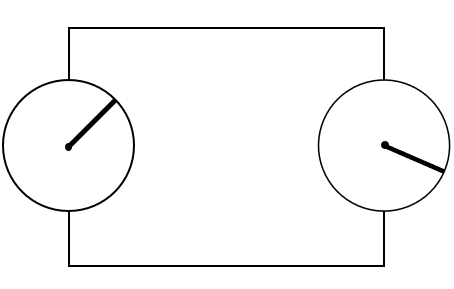
[4.2 Возможности применения 25](#_Toc65516256)

1. **Введение**

Для начала размышления о применениях такого метода решения задач, как генетический алгоритм, стоит разобраться с самим термином. Интернет-источники говорят нам, что генетический алгоритм это - эвристический алгоритм поиска, используемый для решения задач оптимизации и моделирования путём случайного подбора, комбинирования и вариации искомых параметров с использованием механизмов, аналогичных естественному отбору в природе. Но для более точного понимания этого определения и принципа его работы покажем работу генетического алгоритма при решении простой задачи.

## **Пример задачи для генетического алгоритма**

Допустим, компьютеру выдали задачу – спроектировать форму транспорта для того, чтобы он был максимально проходимым. Так же у этого транспорта должно быть 2 колеса и рама любой формы. Первым шагом в решении такой задачи будет принятие начального значения. Обычно, это нечто, очень отдаленно напоминающее решение задачи. Мы поступим так же и нарисуем 3 первых экземпляра:



*рис. 3*

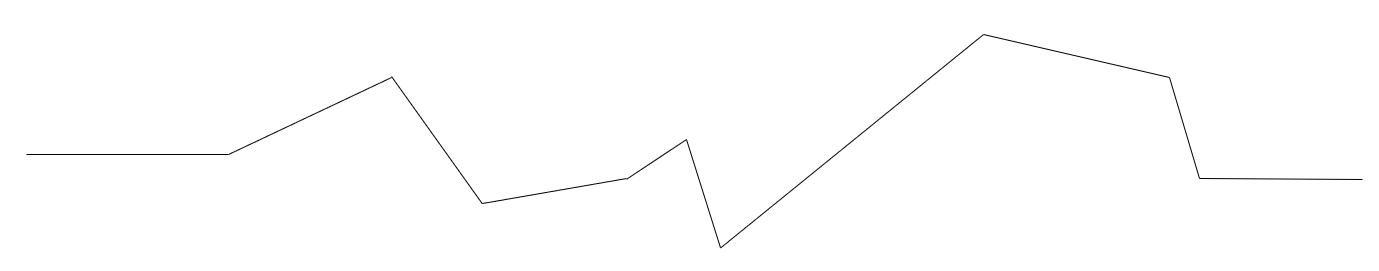
*рис. 2*

*рис. 1*

Для удобства, эти образцы будем называть ботами и обозначать их порядковые номера (1, 2 и 3 в соответствии с рисунком).

У каждого из этих ботов, есть свой набор параметров. С данными ботами мы можем указать 3 параметра: форму корпуса, положение колес на корпусе и размер колес

Следующее, что предстоит «пережить» этим ботам является симуляцией их испытаний *in silico –* на компьютере. Для этого, мы создаем некий мир, который населяют эти боты – представим его в виде условного рельефа местности:



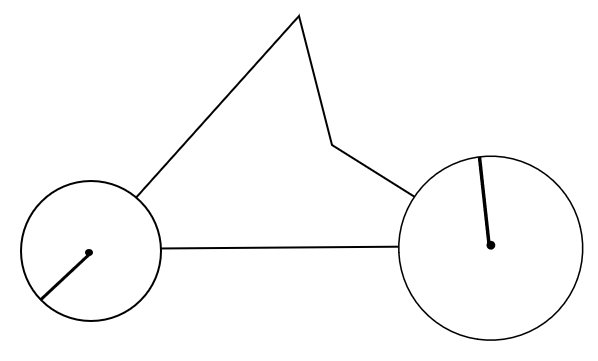
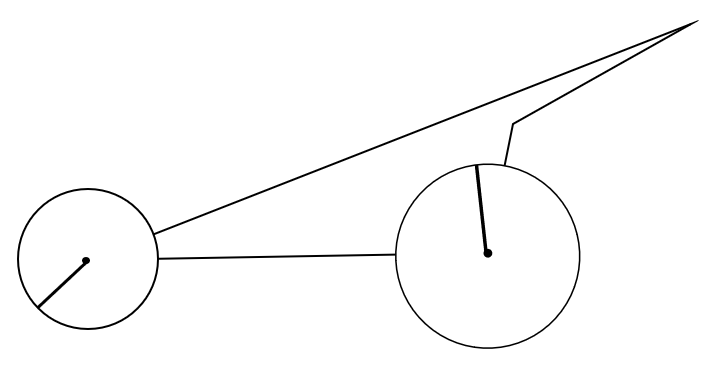
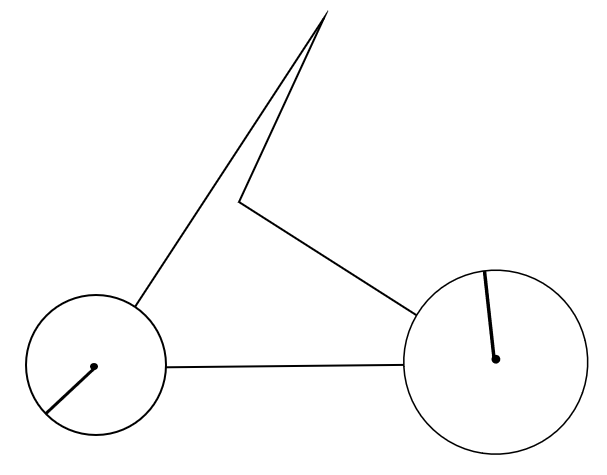
*рис. 4*

А бот, который пройдет всю трассу от начала до конца – будет решением задачи.

## **1.2 Процесс решения**

В нашем случае, каждая итерация представляет из себя гонку, в которой победителем считается тот, кто проехал дальше всех, а для бота-участника, заезд заканчивается, когда он больше не сможет двигаться дальше, то есть если бот застревает, то он заканчивает свой заезд. Рассмотрим первую итерацию на примере:

Все боты появляются в начальной точке – 0 у.е.[[1]](#footnote-1) и начинают двигаться. Первый бот не сможет ничего преодолеть, так как будет стоять на своем корпусе, впустую прокручивая колеса, следовательно, первый бот выбывает с результатом 0. Второй бот, в отличие от первого, двигаться будет и, допустим, проедет 3 у.е., так как вскоре упадет. Он заканчивает с результатом 3. Третий же бот, в свою очередь, будет двигаться гораздо дальше из-за более успешной конструкции. Допустим, что он пройдет 20 у.е. Затем компьютер вычислит победителя – бота под номером 3 и даст ему создать несколько своих копий, в нашем случае, их так же будет три. Каждая копия должна в целом должна иметь параметры, схожие с родительским ботом, но с некоторыми изменениями – мутациями, следовательно выглядеть они будут, в целом, одинаково, но с небольшими отличиями, которые играют роль в проходимости получившегося транспорта. Представим, как могли бы выглядеть получившиеся боты в настоящей программе:

*рис. 7*

*рис. 5*

*рис. 6*

Из этой выборки самым эффективным будет первый бот, который потом аналогичным образом сделает «потомство». С каждой итерацией мы будем получать все более совершенный транспорт, пока одна из машин не достигнет финиша. Этот бот будет базовым решением данной задачи. Его будет достаточно, чтобы пройти трассу, но можно сделать лучше. Тут начинается второй этап решения задачи – на улучшение результата. Этот этап нужен не во всех задачах, но в нашем случае его можно оставить.

Для второго этапа мы можем либо сохранить данную трассу со стартом и финишем заставлять ботов соревноваться на скорость. С каждой итерацией у нас будет получаться все более совершенный бот. Этот процесс может продлеваться бесконечно. В результате такой работы, программа всегда будет приближаться к идеальному решению по гиперболе, но никогда его не достигнет. Так же мы можем создать процедурную генерацию трассы, тогда, аналогично с предыдущим решением, программа будет всегда приближаться к идеальному боту, но он, в свою очередь, будет хорошо приспособлен к прохождению любых участков такого «внедорожья», в отличие от первого, который будет преодолевать конкретный участок трассы максимально быстро.

## **1.3 Использование генетических алгоритмов**

Задача, которую мы, рассмотрели, действительно существовала. Ее решали авторы сайта Genetic Cars, откуда был взят этот пример.

Таким образом, почти без участия человека, получается решение задачи с нетривиальным ответом. Такие способности генетического алгоритма на данный момент нашли некоторое применение в создании систем, симулирующих жизнь. Из-за того, что такой метод решения задач достаточно новый, необычный, то широкого применения на данный момент не получил. Сейчас генетические алгоритмы чаще всего используются в компьютерных играх, создавая персонажей, которые ведут себя как настоящие люди, или симулируя поведение большой толпы людей. Но круг задач, использующих генетический алгоритм, стабильно растет и в дальнейшем, генетические алгоритмы смогут применяться в областях инженерии, где присутствуют понятия идеальных коэффициентов, идеальной формы, идеального компромисса. Например, можно использовать генетический алгоритм при создании идеального автомобиля в реальной жизни. Такая программа, теоретически, может просчитать автомобиль с самой большой вместительностью, но хорошей аэродинамикой, такой автомобиль должен быть удобен в управлении, потреблять немного топлива, но быть достаточно мощной. Такие компромиссы сможет разрешить генетический алгоритм.

# **Симуляция эволюции**

## **2.1 Среда разработки**

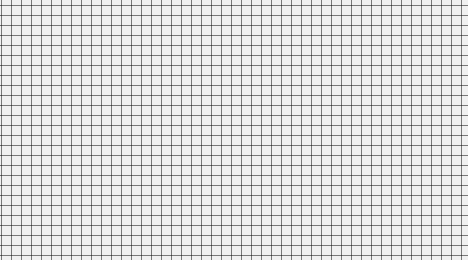
Теперь, после представления такого метода решения задачи, как генетический алгоритм, разберем его работу и применим в настоящей задаче. Создадим программу для симуляции важнейшего механизма биологии – эволюции.

Для создания этого механизма симуляции я использовал язык программирования С++ - один из популярнейших и многофункциональных языков, а для создания и отрисовки интерфейса – платформу Qt 5.15. В данной связке можно изготовить программу с любым функционалом, которую можно будет запустить почти на любой платформе. Но мы остановимся на программе, ориентированную на Windows 10.

## **2.2 Базовые алгоритмы работы**

Поскольку общий механизм генетических алгоритмов всегда схожий, то и программа симуляции эволюции будет похожа на программу создания идеального бота-транспорта из первой главы.

Ни один организм не может существовать без определенной среды обитания. Давайте ее создадим. Среда, в которой мы будем симулировать эволюционные процессы представляет из себя сетку, состоящую из ячеек, каждая из которых может быть заселена ботом, или оставаться пустой:

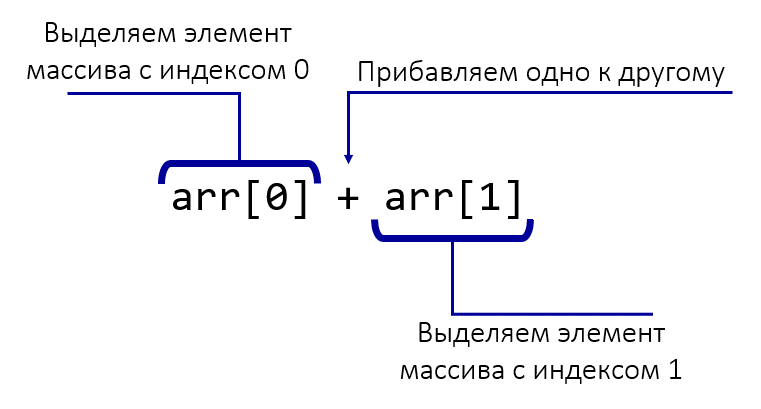


*рис. 7*

На программном уровне эта программа представляет из себя двумерный массив. Это понятие так же стоит разобрать.

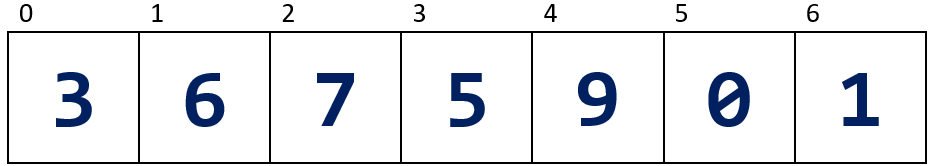
## **2.3 Двумерный массив**

Для разбора двумерного массива, сначала стоит определить, что входит в понятие об обычном массиве. Массив – это структура данных, представляющая из себя несколько переменных с присвоенным индексом. Например, представим такой массив – {5,10,4,2}. Это массив, состоящий из четырех переменных типа int[[2]](#footnote-2). У каждого элемента есть порядковый номер – индекс, по которому его можно вызвать. Счет индексов начинается с нуля, то есть у переменной 5 индексом будет 0, у 10 индекс 1 и так далее. Так же, каждому массиву должно быть присвоено имя. Назовем наш массив *arr* (сокр. от array - (англ. массив)). Теперь мы можем, например, прибавить один член массива к другому. В среде

рограммирования это выглядит так: 

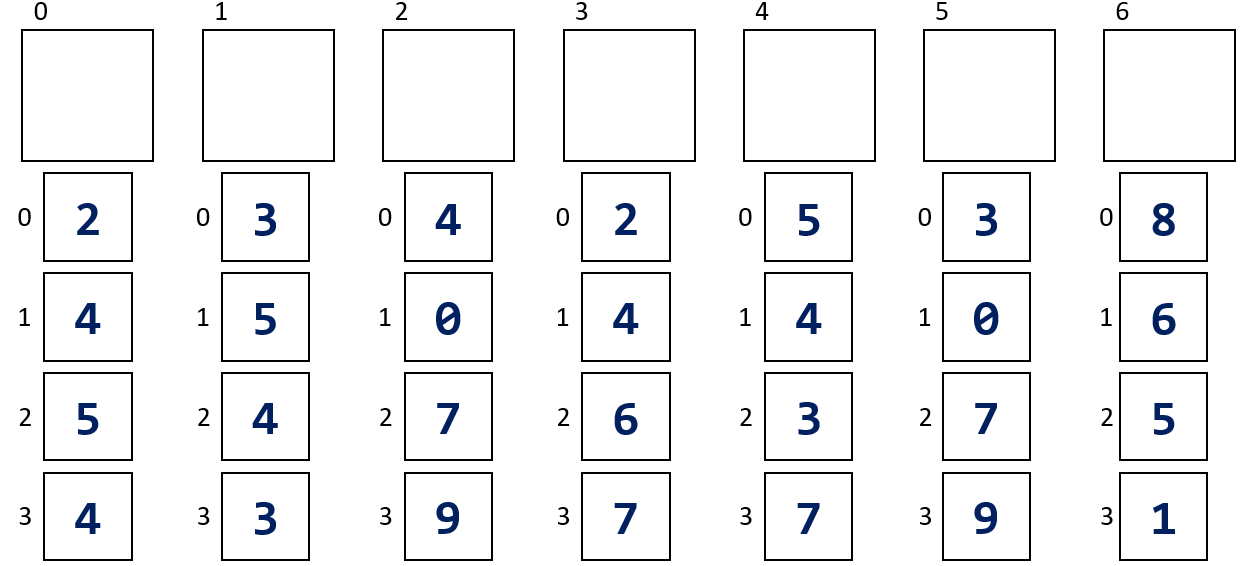
*рис. 8*

Поскольку у каждого элемента массива есть свои индекс и значение (которое может быть пустотой[[3]](#footnote-3)), то весь массив мы можем представить в виде клеточек, выстроенных в ряд. Для демонстрации, создадим новый массив под названием *arr1* и наполним его – arr1 = {3,6,7,5,9,0,1}. А теперь представим его в виде строки из клеток:



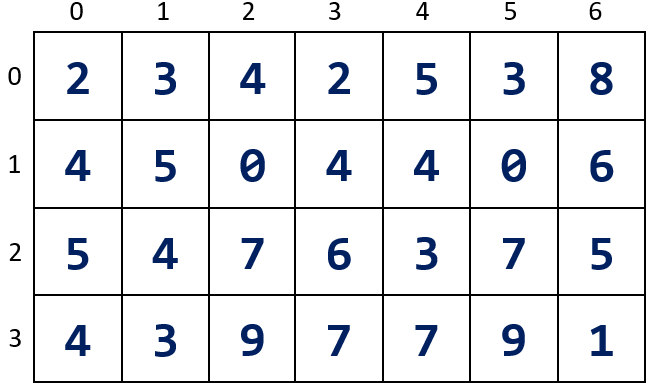
*рис. 9*

В такой иллюстрации четко видно каждый элемент, его индекс и значение. Но что, если вместо числа, как элемент массива, мы возьмем другой массив. Создадим такой массив и назовем его *2darr* уже по названию этого массива, понятно, что такие необычные массивы называют двумерными, но почему? Это будет четко выражено в иллюстрации:



*рис. 10*

На этом изображении мы видим первый массив, состоящий из семи элементов, каждый из которых представлен большим квадратом, а все содержимое элемента массива представлено ниже. То есть, каждый элемент массива из семи элементов, включает в себя массив из четырех элементов, каждый элемент которого, включает в себя некоторое число. Но такое изображения массива *2darr* не слишком удобно. Обычно, двумерные массивы изображают в виде таблицы. Мы поступим так же:



*рис. 11*

Такое изображение выглядит куда лаконичнее, но сразу встает вопрос о вызове одного из элементов такого массива. Ответ здесь довольно простой – к каждому элементу мы присваиваем 2 индекса. Получается, что мы получили некое подобие координат на плоскости, а поскольку мы указываем 2 индекса, то и плоскость у нас состоит из двух координат, то есть является двумерной. Индексы первого подмассива массива *2darr,* состоящего элемента мы можем сравнить с координатой , а индексы массивов, состоящих из четырех элементов, с координатами по . Отсюда и выходит название двумерного массива.

По аналогии мы можем составить трехмерный массив, который сможем изобразить в виде куба, а вызывать элементы из трехмерного массива будем через 3 индекса. С помощью компьютера, мы сможем создать даже четырехмерный и пятимерный массив, но очевидно, что в удобном графическом виде изобразить мы его не сможем.

## **2.4 Биологические начала**

Временно отстранившись от принципов работы программы, поговорим о связи данного проекта с реально существующим миром и биологией.

Наверное, всем известно, что на данный момент, главной общепринятой теорией появления жизни на земле является теория эволюции. По этой теории жизнь зародилась из примитивных капель первых белков – коацерватов, которые имели некоторые признаки жизни, но полноценными живыми организмами они не считались. Далее, в следствии химической эволюции, молекулы, которые были в составе коацерватов стали усложняться и дифференцироваться. В итоге получились предшественники современных молекул липидов, углеводов и белков – главного «строительного материала» природы. Так же появилась молекула РНК – предшественник молекулы ДНК – самого сложного, но и самого важного компонента всего ныне живущего. После этапа химической эволюции и рождения первых, простейших клеток, начался процесс биологической эволюции – клетки усложнялись, потом объединялись в многоклеточные организмы, в последствии чего, появилось все живое, что мы видим вокруг себя, включая нас самих.

Как уже было сказано, главным элементом всего этого процесса стала одна молекула – молекула ДНК, а точнее то, из чего она состоит, грубо говоря, это гены, о которых мы так часто слышим в повседневной жизни. Совокупность генов является геномом или генотипом организма, а в процессе эволюции, этот генотип подвергается изменениям в виде мутаций. Они бывают положительно влияющими и негативно влияющими, очевидно, что организм с негативной мутацией не проживет долго и не сможет дать потомство, в то время как организм с положительной мутацией передаст ее своим детям, сохранив эту мутацию во всем своем роде.

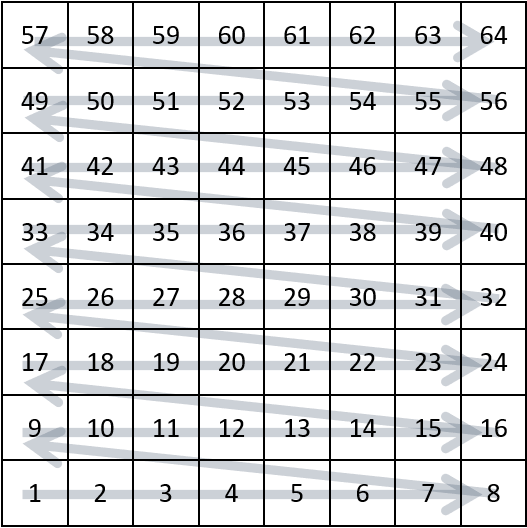
## **2.5 Концепция бота**

Поскольку, представленная ранее программа должна симулировать процесс эволюции, то совместим все вышеописанное, а именно базовые знания об эволюции и знание некоторых принципов работы компьютерных программ. Если мы хотим создать симуляцию эволюции, то обратимся к предыдущему разделу и вспомним, что для эволюции нам нужно подобие молекулы ДНК, генов и генома, которые содержит некий организм. Опишем его работу.

Аналогично примеру из главы 1, назовем наш организм ботом. Бот может находиться только в одной клетке из своего мира одновременно. Может передвигаться, питаться и размножаться, но принципы функционирования отдельного бота рассмотрим позже. Как уже было сказано, для эволюции такого организма нам нужен геном и гены. Представим один ген, как команду к некоторому действию. Каждая команда имеет свой номер:

* 0 – фотосинтез
* 1 – шаг вверх
* 2 – шаг вверх-вправо
* 3 – шаг вправо
* 4 – шаг вниз-вправо
* 5 – шаг вниз
* 6 – шаг вниз-влево
* 7 – шаг влево
* 8 – шаг вверх-влево
* 9 – съесть соседа
* 10 – поставить стену наверх
* 11 – поставить стену направо
* 12 – поставить стену вниз
* 13 – поставить стену налево
* 14 – сломать стену

Каждую из этих команд мы рассмотрим позже. Как и подобает настоящим генам, они хранятся в геноме бота. Геном, в свою очередь, представлен таблицей, состоящей из 64 клеток. Каждая клетка отвечает за один ген – одну команду. Визуально изобразить геном и последовательность его выполнения можно так:



*рис. 12*

На этой иллюстрации номерами и стрелками указан порядок чтения генома ботом.

Функционирование бота и взаимодействие его с окружающим миром происходит не постоянно, а каждый определенный промежуток времени. Все боты могут совершать один ход за определенный промежуток времени – тик. За один ход, бот может выполнить одну из своих команд генома. Поскольку ходы всех ботов выполняются одновременно, то никаких преимуществ у ботов, находящихся в разных местах виртуального мира, не будет.

На этом этапе таблица генома начинает напоминать двумерный массив. Он здесь идеально подошел. Для реализации генома бота был использован двумерный массив, в котором каждая строка этой таблицы обновляется каждое поколение, совершенствуя геном. Об этом механизме подробно рассказано в части 3 (о механизмах питания и размножения). Но чтобы бот создал потомство, а следовательно, несколько поколений, должны существовать механизмы размножения.

# **Строение бота**

Любое явление в симуляции должно коррелировать с этим же явлением из реальной жизни, в симуляции эволюции организмы должны быть схожи с настоящими организмами из реального мира. Наши боты не будут схожи с тем же человеком ни внешне, ни внутренним строением, ни инструментами общения, но бот схож с нами по раду других признаков – бот вынужден питаться, по возможности размножаться, он может двигаться и немного рассказывать о своем внутреннем состоянии. Начнем описание бота с базовой потребности любого живого организма – питание (дыхание мы исключаем за ненадобностью).

## **3.1 Механизмы питания и размножения**

Как было сказано ранее, питание и размножение – это одни из важнейших функций любого живого организма. Наши боты не исключение, и у них есть алгоритмы работы, отвечающие за обмен веществ с окружающей средой и за размножение.

Питание бота может осуществляться тремя способами: автотрофным, то есть бот сам производит питательные вещества. В этом случает бот будет фотосинтезировать и тратить один ход на питание фотосинтезом. И гетеротрофным, то есть бот поедает уже готовые питательные вещества. Гетеротрофное питание, в свою очередь делится еще на два типа: хищничество и питание падалью. Хищники поедают других живых ботов, а падальщики питаются останками умерших ботов.

При любом из приведенных вариантов питания, бот получает энергию. При значении энергии в 0 единиц, бот умирает, оставляя свой труп, а при накоплении 225 единиц энергии, бот делится, создавая свою копию с некоторыми мутациями в геноме. В целом этот принцип схож с принципом размножения виртуальных машин из первой главы, но механизм размножения будет рассмотрен чуть позже. Так же, для каждого типа питания, существует свое количество единиц энергии, получаемых при этом питании. При процессе фотосинтеза, бот получает количество энергии, соответствующее своему возрастному фактору (подробно рассмотрено в части «коэффициенты, влияющие на жизнь»). При поедании своего соседа, то есть при хищничестве бота, бот получает 30 единиц энергии, но для хищничества существуют свои ограничения. Для того, чтобы съесть своего соседа по клетке, бот не должен быть ни слишком молодым, ни слишком старым (возраст бота больше 200 тиков, но меньше 2500 тиков). Так же бот не сможет съесть своего сородича, будучи сытым, то есть если у бота уже больше 249 очков энергии (максимальное кол-во - 255), то возможность хищничества для него закрывается. Боты-боты падальщики сильно схожи с ботами-хищниками, как программно, так и на практике. Бот-падальщик так же получает 30 единиц энергии за поедание, но на него не действуют такие жесткие ограничения, как на хищников. Падальщики могут питаться в любом возрасте, а состояние сытости приходит к ним при полном заполнении энергией (255 единиц).

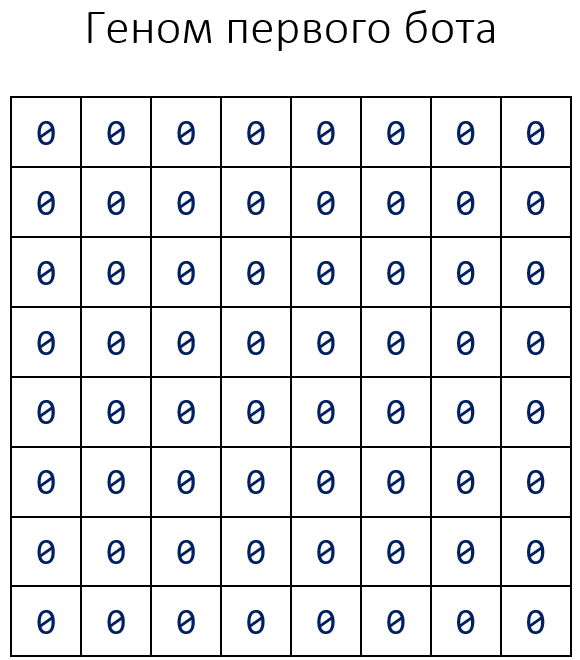
Так же для того, чтобы ярко показать рацион питания бота, он окрашивается в определенный цвет, зависящий от его рациона. Для фотосинтезирующих ботов это зеленый цвет, для хищников красный, а для падальщиков синий. А степень насыщенности или сытости бота показывает насыщенность его цвета, то есть, если у бота мало энергии, то он почти черный, а если он сытый, то его цвет становится ярким и насыщенным. Но что, если рацион бота смешан? Тогда смешивается и цвет окраски. При каждом «приеме пищи» бот меняет свой цвет. Вся энергия, которой владеет бот, делится в процентном соотношении на энергию, полученную фотосинтезом, хищничеством или собиранием падали. Выбор цвета для бота можно показать в виде цветовой палитры:



*рис. 13*

Эта палитра актуальна только при количестве энергии у бота, равное 255 единицам, при уменьшении количества энергии, палитра начнет темнеть.

На данный момент можно справедливо предположить, что главная цель бота – это получить 255 единиц энергии. Но что потом? А потом, после достижения бота жизни в комфортных условиях, бот начинает размножатся. При размножении, 100 единиц энергии материнского бота переходят дочернему боту. Так же, при размножении бота, меняется его геном. Аналогично примеру из первой главы, мутирует некоторый признак бота. В нашем случае мутирует геном, но не полностью, а частично. С каждым новым размножением, геном все усложняется. У самой первой клетки, геном состоял только из нулей, и если обратиться к предыдущей части, то можно понять, что первый бот умеет только фотосинтезировать. Его жизнь достаточно банальна. Он фотосинтезирует на протяжении нескольких тиков, а потом размножается, являя свету бота с немного искаженным геномом отца – меняется один ген – одна клетка из таблицы в 64 элемента. Причем, для изменения, существуют некоторые законы. Меняться может только нижняя строка таблицы, а при размножении бота с измененным геномом, эта строка дублируется вниз, вытесняя верхнюю строку таблицы и затем так же происходит процесс мутации. Этот процесс показан ниже:



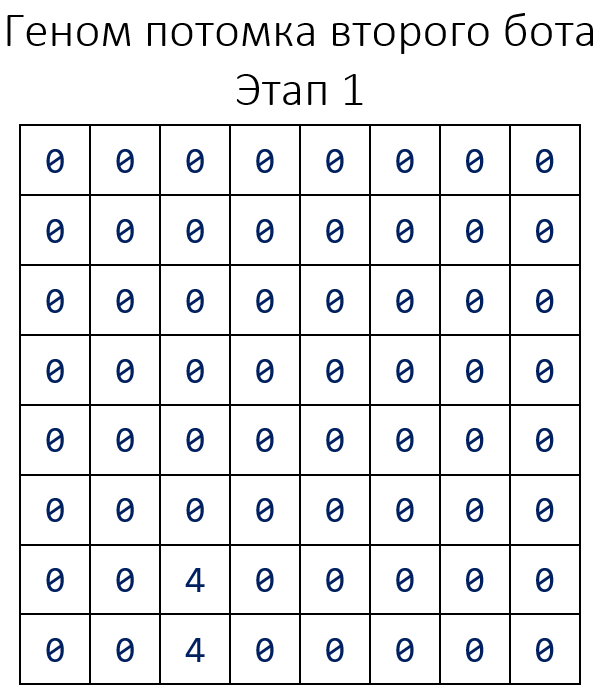
*рис. 14*

Далее, при размножении этого бота, у его потомка, мутирует нижняя строка:



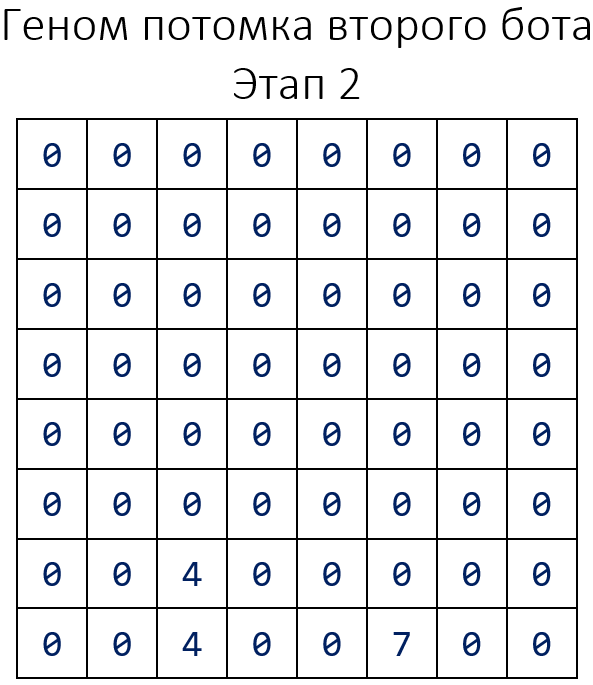
*рис. 15*

При размножении этого бота, у его потомка дублируется нижняя строка:



*рис. 16*

А затем, снова мутирует нижняя строка:



*рис. 17*

И таким способом, процесс искусственной эволюции может проходить бесконечно.

Но в таком режиме работы бота, он может жить бесконечно много. А жестко ограничивать жизнь бота неким числом неправильно. Поэтому в программу была введена система коэффициентов.

## **3.2 Коэффициенты, влияющие на жизнь**

Как было объявлено ранее, в программе присутствует система коэффициентов, которые влияют на существование ботов. Главным из них является коэффициент поглощения энергии, который отвечает за количество энергии, полученное в ходе питания. Начальное значение этого коэффициента = 2, то есть в процессе фотосинтеза, бот с возрастом 0 получает 2 единицы энергии. Но с каждым новым тиком, значение этого коэффициента увеличивается на 0,05 с условием, которое ограничивает бесконечный рост бота. Увеличение коэффициента питания будет проходить только пока бот не достиг возраста в 2000 тиков. После этого, коэффициент начнет уменьшаться на 0,025 за один тик. Из этих данных мы можем понять, что идеальным возрастом для жизни бота – это промежуток от 1000 тиков до 2000 тиков. В этот период бот получает до 100 единиц энергии за один тик, что является очень большим числом. После 2000 тиков, прожитых ботом, начинается процесс старения и количество единиц энергии, полученное за один тик, постепенно уменьшается. Это не дает ботам жить меньше, но дает больше возможностей для размножения ботам с более успешным геномом, ведь такие боты проживут дольше.

## **3.3 Правила перемещения**

Для того, чтобы боты были еще более схожими с реальными живыми существами, в программе присутствуют возможности перемещения. Как было указано в списке всех возможных для бота команд, бот может перемещаться во все соседствующие с ним клетки, включая диагональные. На одно перемещение бот тратит одну единицу энергии из-за чего не может перемещаться постоянно

## **3.4 Устройство стен**

Также для того, чтобы длительность жизни фотосинтезирующих ботов увеличилась, был добавлен механизм защиты. Они могут расставлять на поле стенки, которые впоследствии будут съедены хищниками, сохранив жизнь зеленому боту.

Стенки обозначаются коричневым цветом и занимают одну клетку на поле. Как было сказано выше, бот-хищник может съесть стенку, причем получит от стенки в два раза меньше энергии, чем получил бы от охоты на живого бота. Так же, любой бот по соответствующей команде генома может удалить встретившуюся ему стенку. Энергии при этом он не получит, поэтому, такое действие не считается за питание.

# **Практика**

## **Реализация кода**

Как было сказано выше, для симуляции эволюционных процессов, нужно создать систему, позволяющую управлять ботами в замкнутой системе. В данной работе для такой системы будет использоваться язык программирования C++ и фреймворк для создания приложений с графическим интерфейсом – Qt. В основе мира лежит двумерный массив из переменных, которые занимают боты. В коде данный мир представлен следующей строкой:



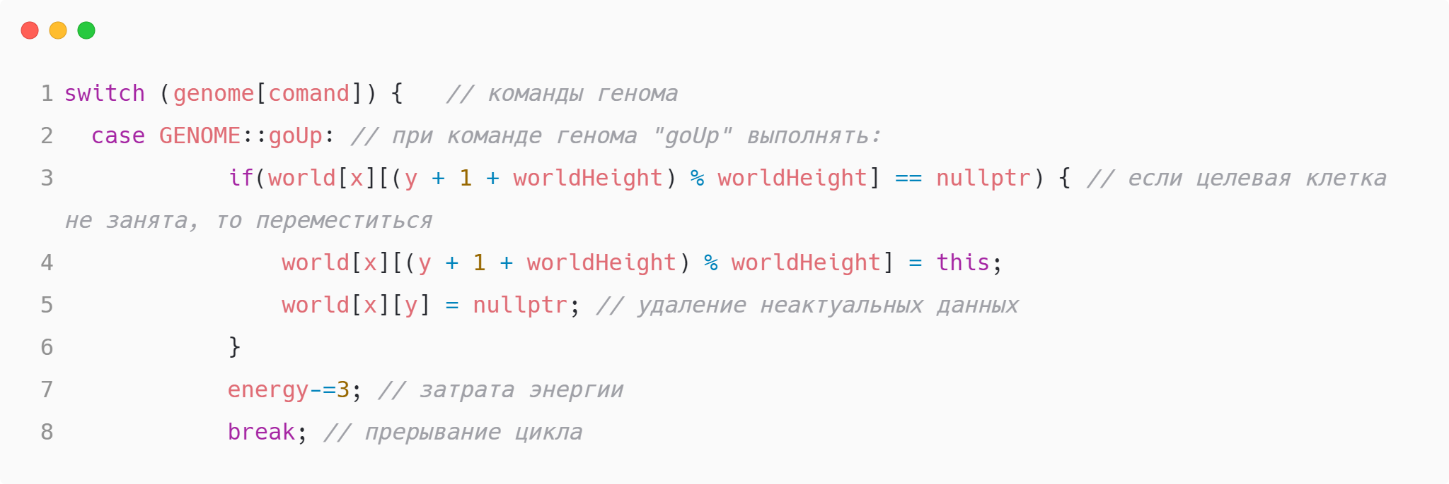
*рис. 18*

Она создает двумерный массив (его принцип работы представлен в главе 2.3) заполняет его ботами. В свое время бот программно представляет из себя класс, выглядящий так:



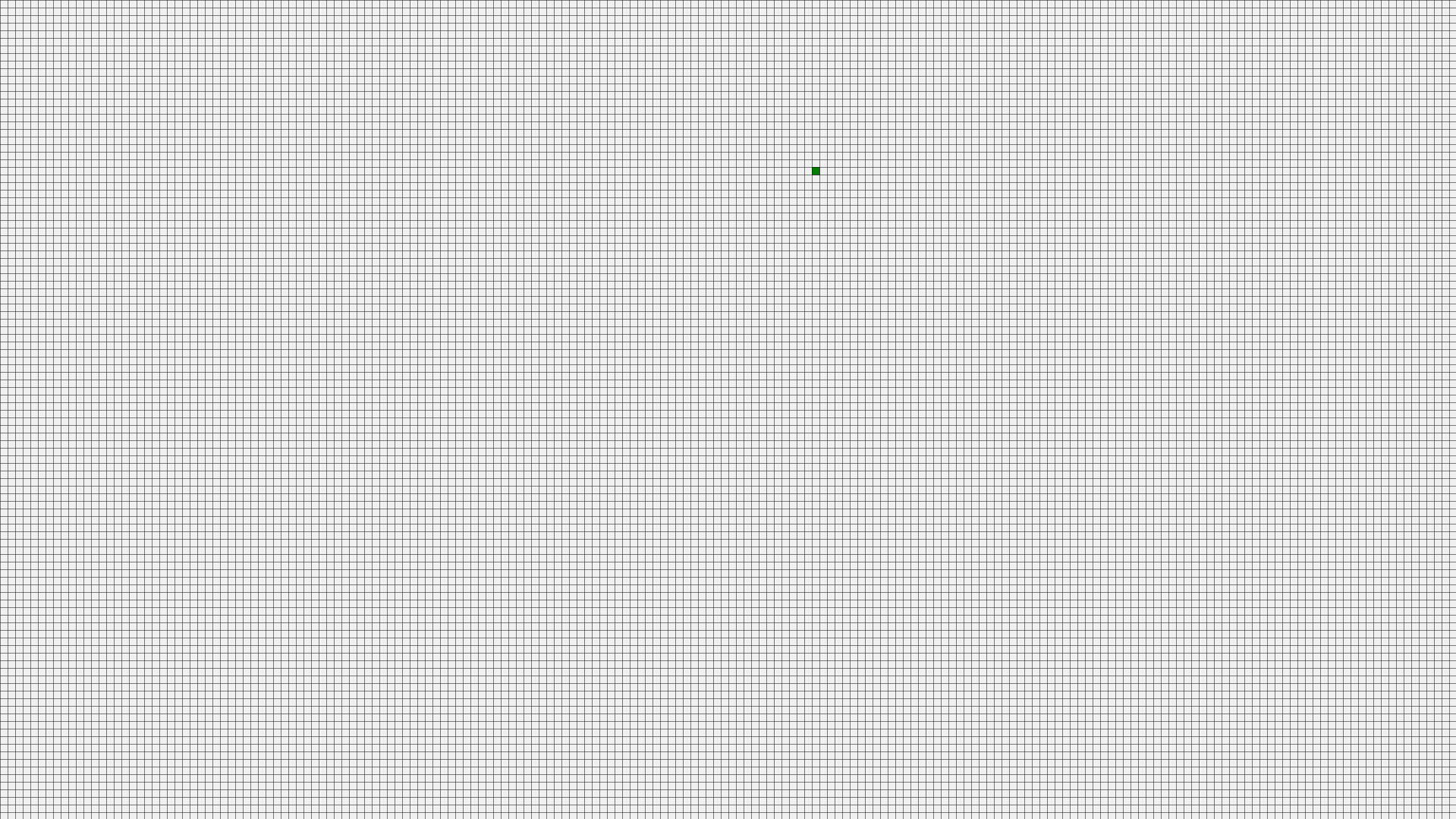
*рис. 19*

Но для того, чтобы данные боты смогли жить и перемещается для них реализованы команды. Разберем их на примере одной команды – команды движения:



После стольких слов, сказанных о теории, поговорим о практической части проекта, как было уже неоднократно сказано, она представляет из себя программу, симулирующую эволюцию некоторых ботов. Для наилучшего представления об этой программе, опишем ее работу с момента запуска.

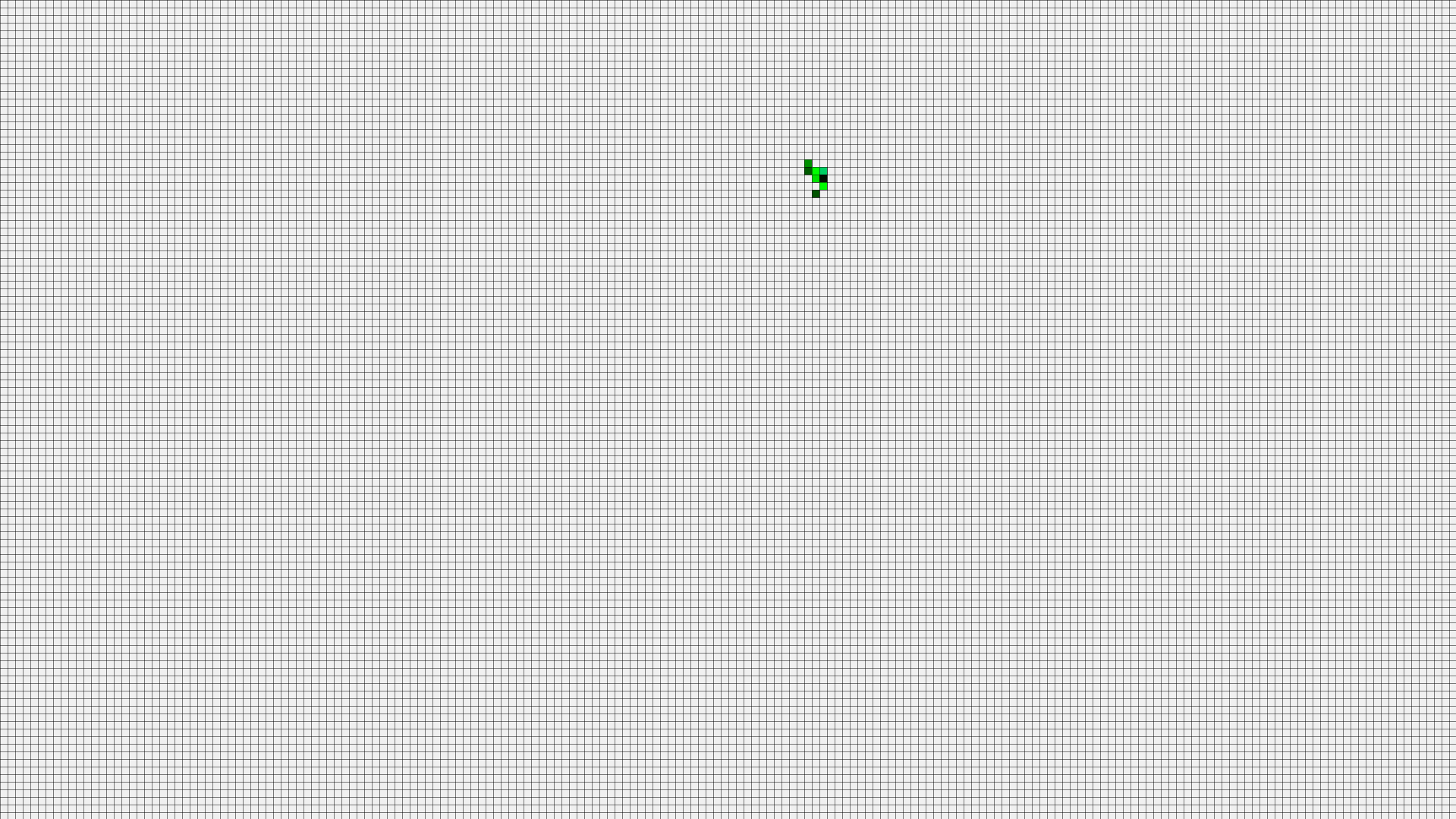
Запустив программу, мы увидим сетку из свободных клеток и первого бота, который находится в одной из них.



*рис. 18*

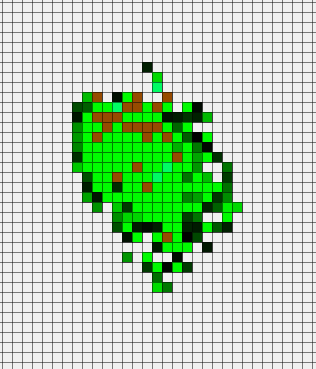
В самом начале, этот бот будет черного цвета, так как появляется он с минимальным запасом энергии (1 единица энергии). Далее, он постепенно зеленеет из-за того, что накапливает энергию с помощью фотосинтеза. Двигаться он не будет, так как весь его геном состоит только из команды о фотосинтезе. Подробно это было описано в главе о механизмах питания и размножения. В той же главе был описан принцип размножения, и мы знаем, что бот создает свою копию после накопления максимального количества энергии – 255 единиц.

Новоиспеченный бот будет помещен в соседнюю с родительским ботом клетку. В идеальных условиях, «направление» для размножения должно определяться случайным образом. В данной программе, ради оптимизации принцип рандомного определения клетки для размножения происходит только один раз. Если выбранная клетка уже занята, то выбирается клетка, находящаяся справа от бота-родителя. Если правая клетка тоже занята, то бот выбирает нижнюю клетку, если она не доступна, то бот выбирает клетку слева или сверху от себя аналогичным образом. Из-за такого механизма, в дальнейшем в поведении ботов появится одна особенность, которую мы рассмотрим позже.



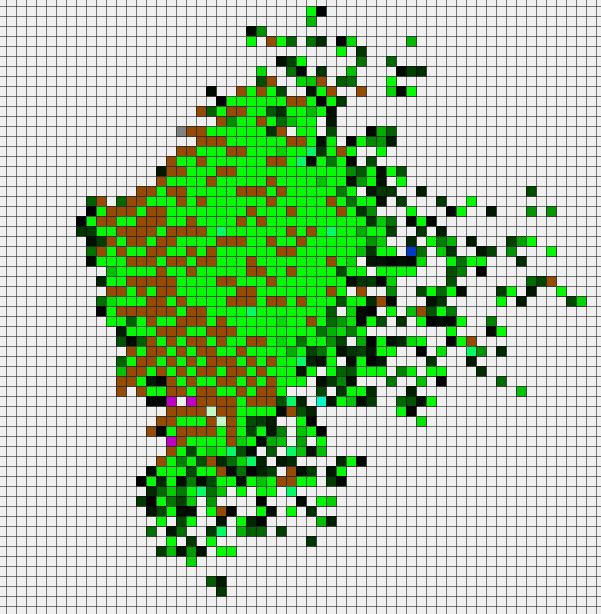
*рис. 19*

После того, как боты несколько раз размножились, мы можем наблюдать изменения в геноме. Они начинают перемещаться по полю. Перемещаются они не слишком активно, но это все равно остается заметным.



*рис. 20*

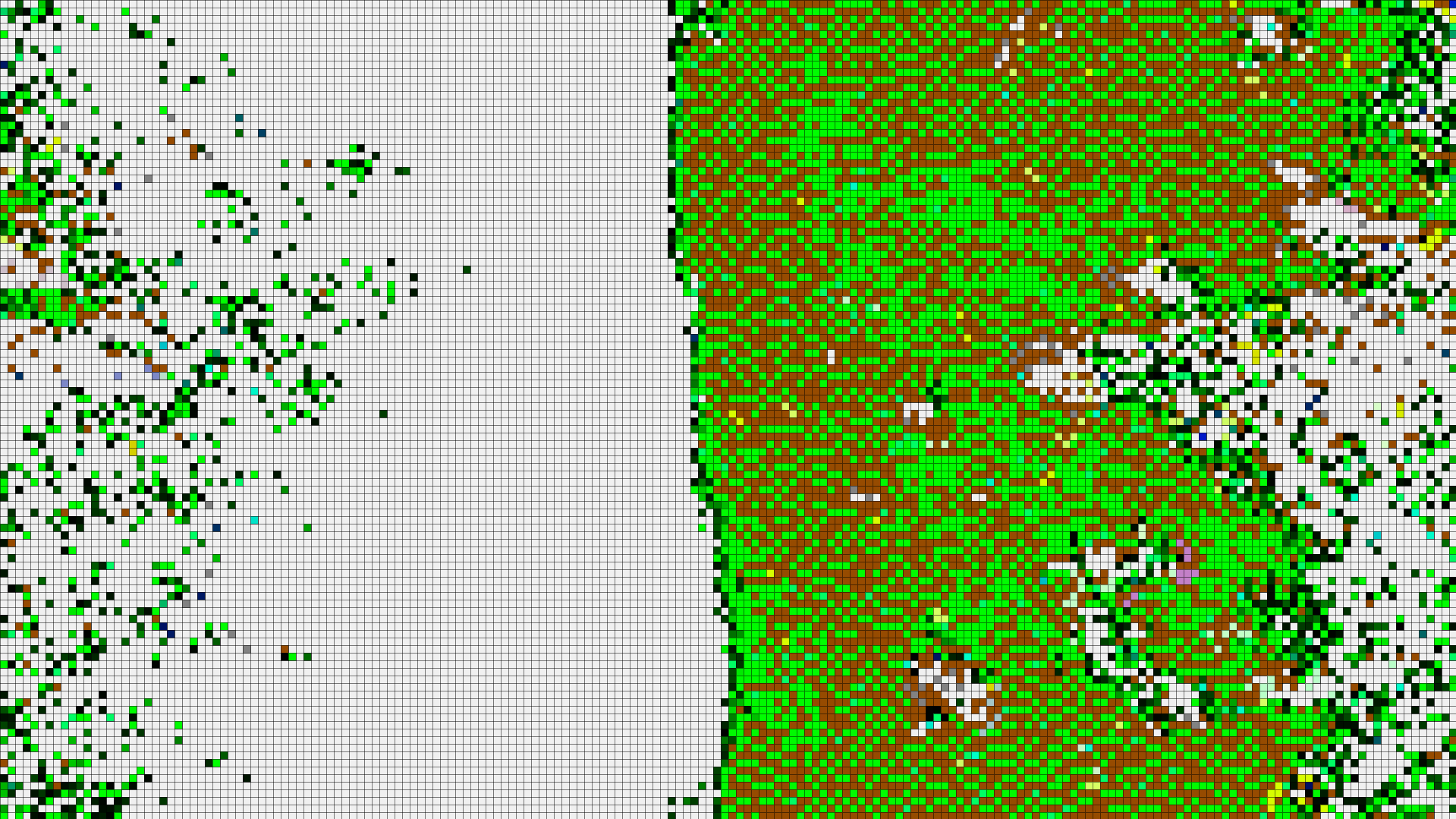
После еще нескольких итераций работы программы, начинают появляться стенки. На данный момент не появилось ни одного хищника, и колония фотосинтезирующих ботов быстро разрастается.



*рис. 21*

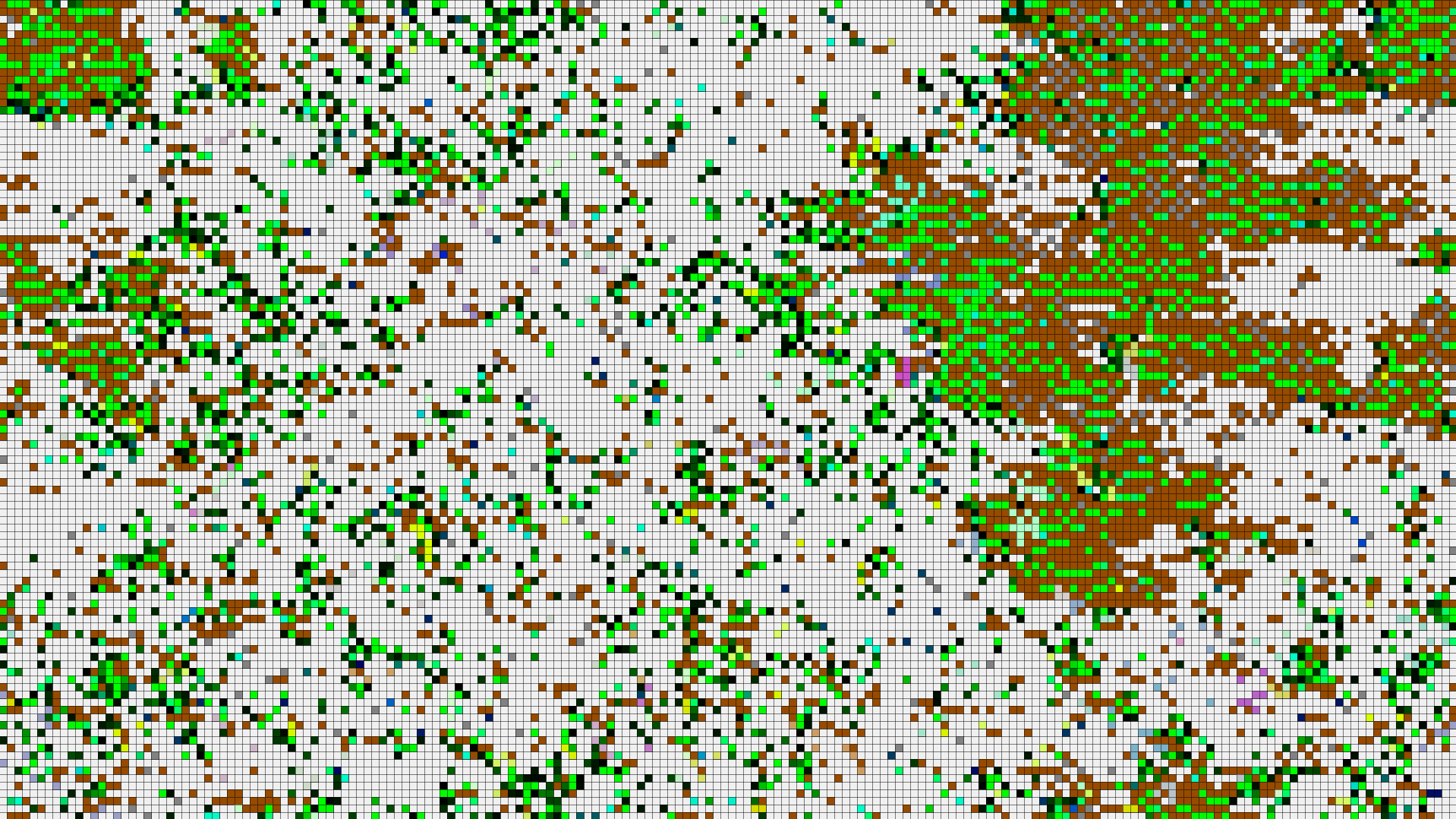
А на данном этапе развития системы можно наблюдать первых ботов не зеленого цвета. Мы видим синего бота, значит он является падальщиком, а поскольку его цвет ярко синий, то его основным источником питания является падаль, то есть уже мертвые боты. Одного такого мертвого бота мы можем увидеть в левом верхнем углу колонии. Он представлен серой клеткой.

Так же в данный момент в системе присутствуют боты со смешанным питанием, то есть они берут энергию из разных источников питания. На иллюстрации выше мы видим 2 вида таких ботов. Это бирюзовый бот и несколько розовых ботов. В соответствии с цветами ботов, мы можем установить их тип питания, на пример, розовые боты – это хищники и падальщики одновременно, а бирюзовый бот занимается как падалью, так и фотосинтезом.



*рис. 22*

Следующим переломным моментом в жизни системы можно назвать образование некой монолитной стены из ботов, питающихся фотосинтезом и стенок, которые они создают. Но эту стену начинают разрушать боты, подошедшие к ней сзади. Это может происходить из-за того, что весь мир можно представить разверткой тора. Это позволяет ботам, «прикоснувшись» к одному краю мира, сразу же переместиться на другой его край. Это работает как по вертикальной оси, так и по горизонтальной. Такая реализация позволяет проиллюстрировать условия шарообразной планеты.

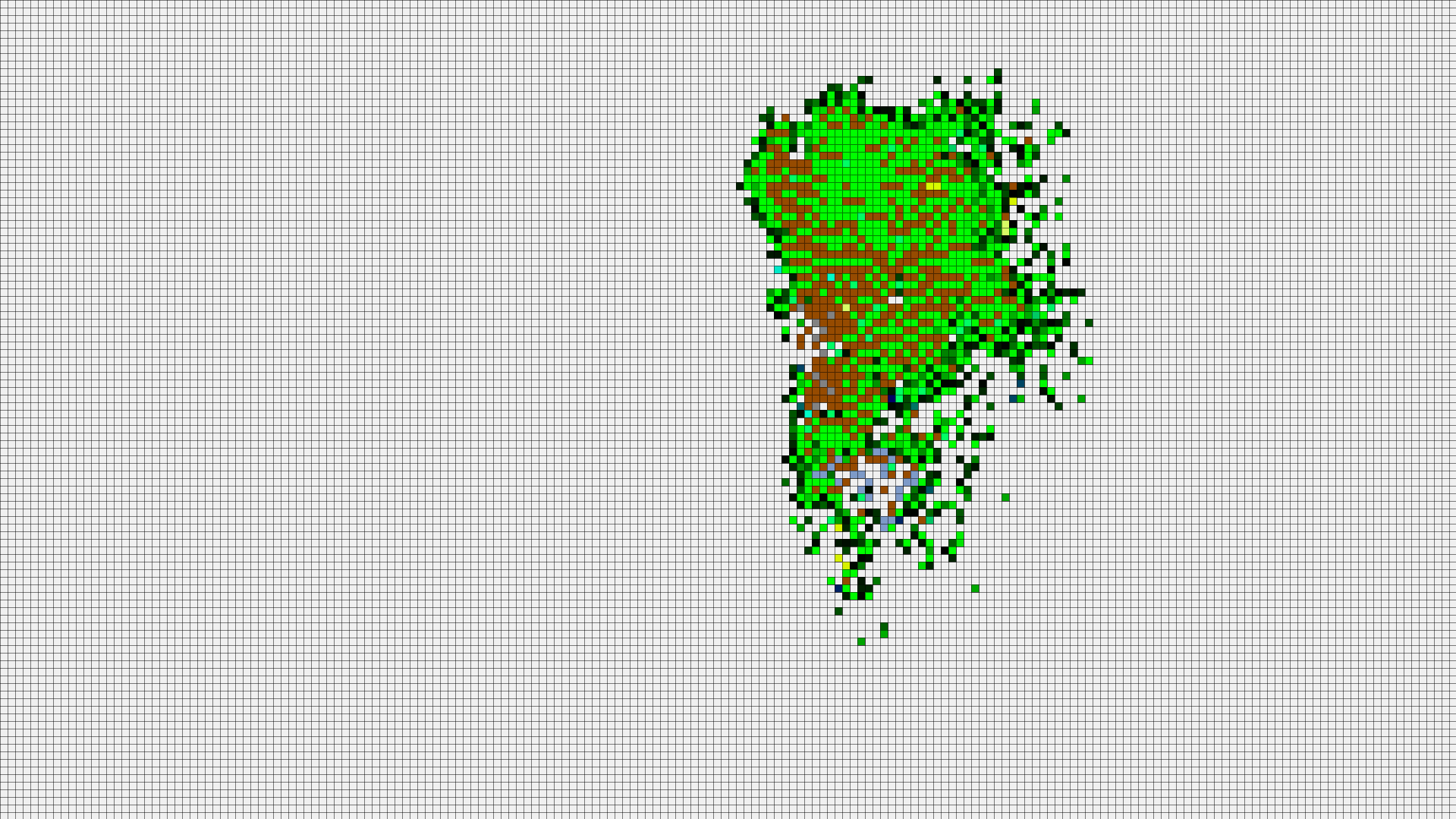
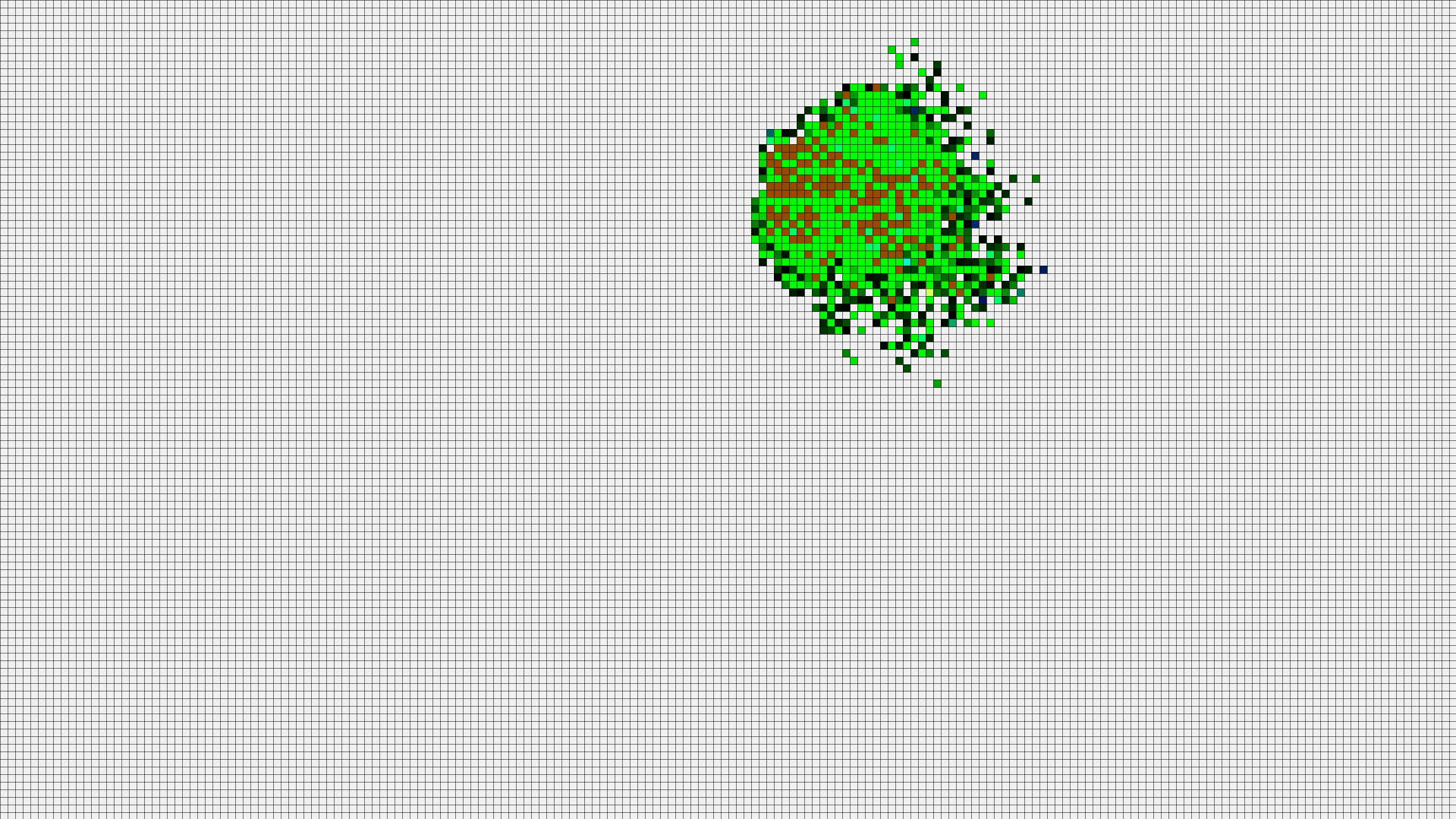
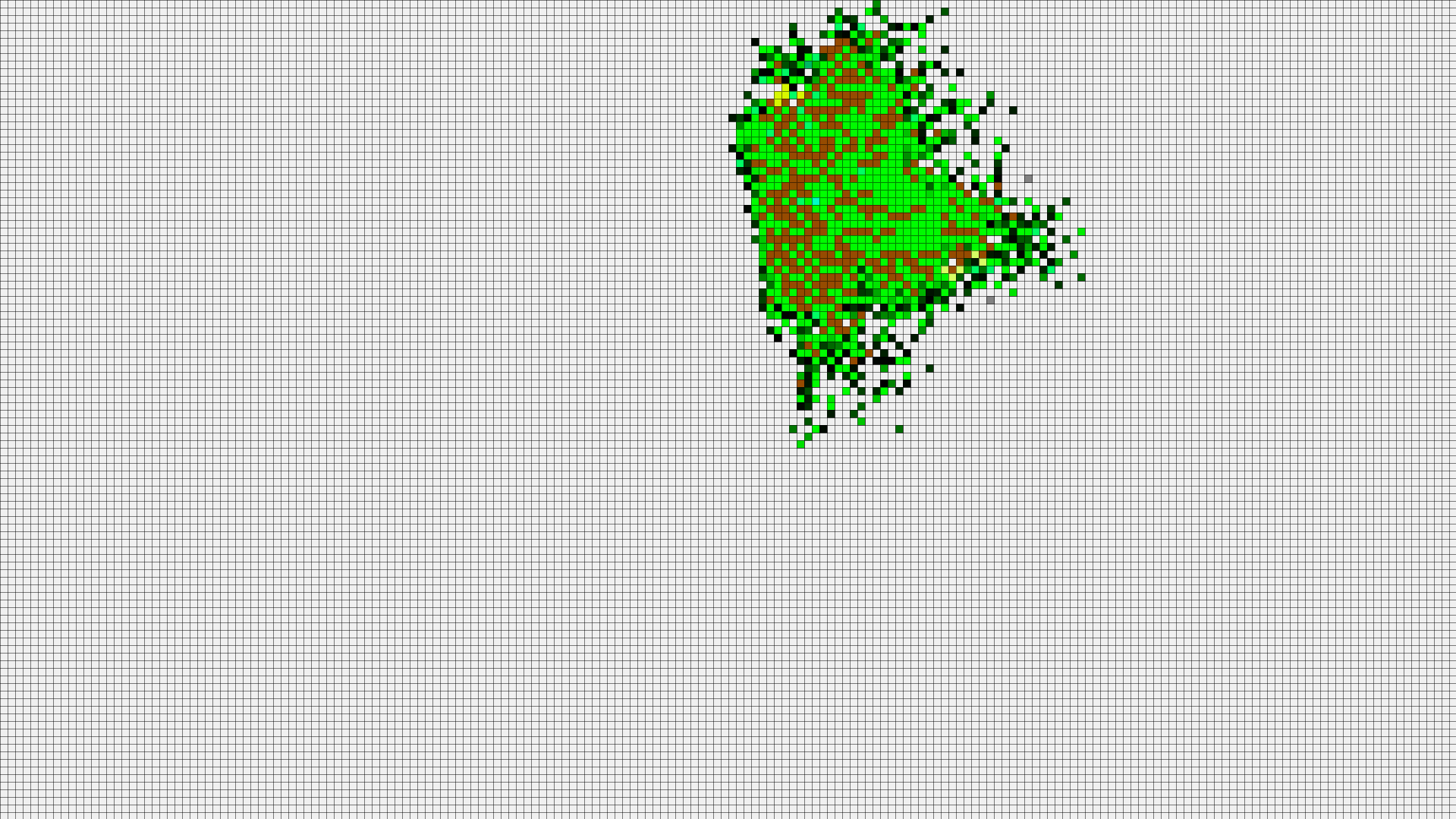


*рис. 23*

После разрушения большой стены, жизнь в программе становится спокойнее. Боты передвигаются не так активно, а разные типы питания ботов позволяют компенсировать друг друга и сохранять равновесие в системе.

## **4.3 Рандомная генерация**

Основным преимуществом как этой программы, так и многих других, основанных на методе решения с помощью генетического алгоритма, является случайная генерация мира, событий и прочих условий, меняющих систему. В нашем случае, таким объектом изменения является геном ботов. Каждый перезапуск программы не похож на предыдущий. Попробуем это наглядно показать. Ниже приведены изображения работы программы через 2000 тиков после старта.

*рис. 26*

*рис. 25*

*рис. 24*

На данных изображениях отчетливо видно, что образовавшиеся за 2000 тиков колонии чем-то схожи, но все равно, выглядят по-разному.

Такая особенность при решении задач обычно идет на пользу человеку, показывая вариативные пути решения и давая право выбора между решениями.

Так же, работа программы проиллюстрирована в видео ниже:

<https://youtu.be/TaJuLmNC_B8>

<https://youtu.be/OpWpkrmAsoY>

https://youtu.be/Eul2R4Ni1jo

https://youtu.be/uDV3u7vNhx4

## **4.4 Возможности применения**

Как уже было сказано в первой части, генетический алгоритм, как метод решения задач сможет широко применяться в жизни человека. Инженерные задачи на решение некоторых компромиссов могут легко поддаться решению через генетические алгоритмы. С помощью такого решения, человек сможет получить идеальные автомобили, ракеты и прочий транспорт, гаджеты, которые в идеальных пропорциях смогут сочетать комфортные размеры и наилучшую производительность. Все из перечисленных изобретений можно получить, создав систему с соревнующимися ботами. Производительность данного способа ниже, чем у способа решения задач привычными нам способами, но мощности компьютеров постоянно развиваются, позволяя ускорять работу любой системы, симулирующей жизнь.

**Источники**

**Книги:**

* Г. Саттер. Решение сложных задач на С++. Серия С++ In-Depth.
* Ч. Петцольд. Код. Тайный язык информатики / пер. с англ. О. Сивченко.
* М. Шлее. Qt 5.10. Профессиональное программирование на С++.
* С. Липпман, Ж. Лажойе. Язык программирования С++.
* А. Бхаргава. Грокаем алгоритмы. Пособие для программистов.
* Р. Шмид. Наглядная биотехнология и генетическая инженерия.

**Электронные ресурсы:**

* ru.wikipedia.org/wiki/
* <https://rednuht.org/genetic_cars_2/>
* <https://www.youtube.com/channel/UCP1JsJgeNs86oqLGnjfGo9Q>
* <http://qt-doc.ru>
* <https://habr.com/ru/post/86777/>
* <https://docs.microsoft.com/ru-ru/cpp/?view=msvc-160>
* [https://www.youtube.com/bioinforussia](https://vk.com/away.php?to=https%3A%2F%2Fwww.youtube.com%2Fbioinforussia&cc_key=)
* https://stepik.org/org/bioinf
* <https://stepik.org/course/2557/promo#toc>
* <https://basegroup.ru/community/articles/ga-math>
* <https://doc.qt.io>
* http://doc.crossplatform.ru/qt/4.8.x/html-qt/

1. Условные единицы измерения расстояния, актуальные только для этой задачи (задачи про идеальный транспорт) [↑](#footnote-ref-1)
2. Int – (сокр. от Integer - (англ. - целое число)) тип данных во многих языках программирования, представляющий из себя целое число, обычно занимающее 2 байта памяти. [↑](#footnote-ref-2)
3. В программировании понятие нуля и пустоты (null) сильно дифференцированы, ведь для компьютера разные числа одного типа данных, например, 0 и 3 достаточно схожи, и для хранения их обоих, выделяется один объем памяти. Для значения null ячейка памяти тоже выделяется, но остается незаполненной. [↑](#footnote-ref-3)