Министерство науки и высшего образования РФ

**НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

Кафедра Вычислительной техники

Новосибирск 2022

|  |  |
| --- | --- |
| Факультет: АВТФ  Группа: АВТ-042  Студент: Антонов А.Р. | Преподаватель:  Дубков И.С. |

КУРСОВАЯ РАБОТА

ПО ДИСЦИПЛИНЕ «Технологии и методы программирования»

*Тема: Программа «***Conway's Game of Life***»*

Оглавление

[Оглавление 2](#_Toc6176)

[Описание 3](#_Toc28028)

[Структурное описание разработки 7](#_Toc24766)

[Функциональное описание разработки 7](#_Toc26666)

*[Задание размеров поля и клеток в нем](#_Toc15554)* [7](#_Toc15554)

*[Настройка спецификации swing](#_Toc4881)* [7](#_Toc4881)

*[Настройка меню](#_Toc5581)* [7](#_Toc5581)

*[Настройка доски игры](#_Toc14030)* [7](#_Toc14030)

*[Разыгровка](#_Toc27035)* [7](#_Toc27035)

*[Программирование действий с панели управления а так же добавление информации.](#_Toc7032)* [7](#_Toc7032)

*[Array list под память а так же увеличение размера](#_Toc17416)* [7](#_Toc17416)

*[Обработка координат клика с мыши](#_Toc30990)* [8](#_Toc30990)

*[Перезагрузка игрового окна](#_Toc21065)* [8](#_Toc21065)

*[Очистка всего поля игры](#_Toc4193)* [8](#_Toc4193)

*[Рандомная генерация клеток на игровом поле](#_Toc8365)* [8](#_Toc8365)

*[Прорисовка клеток по координатам мыши](#_Toc22589)* [8](#_Toc22589)

*[Установка правильного размера игрового поля](#_Toc24485)* [8](#_Toc24485)

*[Обработка движения мыши как выбор нескольких рядом лежащих точек](#_Toc18647)* [8](#_Toc18647)

*[Основа игры, правила жизни и смерти точек](#_Toc12781)* [8](#_Toc12781)

[Описание пользовательского интерфейса 8](#_Toc24537)

[Руководство пользователя 12](#_Toc21761)

[Руководство программиста 12](#_Toc2419)

[Програма требует openjdk-18 для корректной работы. 12](#_Toc15948)

[Заключение 12](#_Toc16226)

[Список источников 19](#_Toc23214)

**Описание**

**Игра «Жизнь»** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA" \o "Английский язык) *Conway's Game of Life*) — [клеточный автомат](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B5%D1%82%D0%BE%D1%87%D0%BD%D1%8B%D0%B9_%D0%B0%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82" \o "Клеточный автомат), придуманный английским математиком [Джоном Конвеем](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D0%B2%D0%B5%D0%B9,_%D0%94%D0%B6%D0%BE%D0%BD_%D0%A5%D0%BE%D1%80%D1%82%D0%BE%D0%BD" \o "Конвей, Джон Хортон) в [1970 году](https://ru.wikipedia.org/wiki/1970_%D0%B3%D0%BE%D0%B4" \o "1970 год).

**Правила**

Место действия игры — размеченная на клетки плоскость, которая может быть безграничной, ограниченной, или замкнутой.

* Каждая клетка на этой поверхности [имеет восемь соседей](https://ru.wikipedia.org/wiki/%D0%9E%D0%BA%D1%80%D0%B5%D1%81%D1%82%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%D0%9C%D1%83%D1%80%D0%B0" \o "Окрестность Мура), окружающих её, и может находиться в двух состояниях: быть «живой» (заполненной) или «мёртвой» (пустой).
* Распределение живых клеток в начале игры называется первым поколением. Каждое следующее поколение рассчитывается на основе предыдущего по таким правилам:
  + - в пустой (мёртвой) клетке, с которой соседствуют три живые клетки, зарождается жизнь;
    - если у живой клетки есть две или три живые соседки, то эта клетка продолжает жить; в противном случае (если живых соседей меньше двух или больше трёх) клетка умирает («от одиночества» или «от перенаселённости»).
* Игра прекращается, если
  + - на поле не останется ни одной «живой» клетки;
    - конфигурация на очередном шаге в точности (без сдвигов и поворотов) повторит себя же на одном из более ранних шагов (складывается периодическая конфигурация)
    - при очередном шаге ни одна из клеток не меняет своего состояния (предыдущее правило действует на один шаг назад, складывается стабильная конфигурация)

Игрок [не принимает активного участия в игре](https://ru.wikipedia.org/wiki/Zero_Player_Game" \o "Zero Player Game). Он лишь расставляет или генерирует начальную конфигурацию «живых» клеток, которые затем изменяются согласно правилам. Несмотря на простоту правил, в игре может возникать огромное разнообразие форм.

**Происхождение**

Джон Конвей заинтересовался проблемой, предложенной в [1940-х](https://ru.wikipedia.org/wiki/1940-%D0%B5" \o "1940-е) годах известным математиком [Джоном фон Нейманом](https://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D0%B9%D0%BC%D0%B0%D0%BD,_%D0%94%D0%B6%D0%BE%D0%BD_%D1%84%D0%BE%D0%BD" \o "Нейман, Джон фон), который пытался создать гипотетическую машину, которая может воспроизводить сама себя. Джону фон Нейману удалось создать математическую модель такой машины с очень сложными правилами. Конвей попытался упростить идеи, предложенные Нейманом, и в конце концов ему удалось создать правила, которые стали правилами игры «Жизнь».

Впервые описание этой игры было опубликовано в [октябрьском](https://ru.wikipedia.org/wiki/%D0%9E%D0%BA%D1%82%D1%8F%D0%B1%D1%80%D1%8C" \o "Октябрь) ([1970 год](https://ru.wikipedia.org/wiki/1970_%D0%B3%D0%BE%D0%B4" \o "1970 год)) выпуске журнала *[Scientific American](https://ru.wikipedia.org/wiki/Scientific_American" \o "Scientific American)*, в рубрике «Математические игры» [Мартина Гарднера](https://ru.wikipedia.org/wiki/%D0%93%D0%B0%D1%80%D0%B4%D0%BD%D0%B5%D1%80,_%D0%9C%D0%B0%D1%80%D1%82%D0%B8%D0%BD" \o "Гарднер, Мартин) (*Martin Gardner*).

Компьютерная реализация

В компьютерных реализациях игры поле ограничено и, как правило, замкнуто — верхняя граница поля «соединена» с нижней, а левая граница — с правой, что представляет собой эмуляцию поверхности [тора](https://ru.wikipedia.org/wiki/%D0%A2%D0%BE%D1%80_(%D0%BF%D0%BE%D0%B2%D0%B5%D1%80%D1%85%D0%BD%D0%BE%D1%81%D1%82%D1%8C)" \o "Тор (поверхность)), но на экране поле всегда отображается в виде равномерной сетки.

Простейший алгоритм «смены поколения» последовательно просматривает все клетки решётки, для каждой подсчитывает соседей, определяя судьбу клетки в новом поколении (не изменится, умрёт, родится). Такой алгоритм использует два двумерных массива — для текущего и для следующего поколений.

Более быстрый алгоритм делает первый проход по всем клеткам, но при этом составляет список клеток для просмотра в последующем поколении. Клетки, которые через поколение принципиально не могут измениться, в список не вносятся. Например, если какая-либо клетка и все её соседи не изменились при текущем обсчёте нового поколения, то эта клетка не изменится и при следующем проходе.

**Влияние на развитие наук**

Хотя игра состоит всего из двух простых правил, тем не менее она более сорока лет привлекает внимание учёных. Игра «Жизнь» и её модификации повлияли (в ряде случаев взаимно) на многие разделы таких точных наук, как [математика](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0" \o "Математика), [информатика](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0" \o "Информатика) и [физика](https://ru.wikipedia.org/wiki/%D0%A4%D0%B8%D0%B7%D0%B8%D0%BA%D0%B0" \o "Физика). Это, в частности:

* [Теория автоматов](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B0%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82%D0%BE%D0%B2" \o "Теория автоматов),
* [Теория алгоритмов](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D0%BE%D0%B2" \o "Теория алгоритмов),
* [Теория игр](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B8%D0%B3%D1%80" \o "Теория игр) и [математическое программирование](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5" \o "Математическое программирование),
* [Алгебра](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%B5%D0%B1%D1%80%D0%B0" \o "Алгебра) и [теория чисел](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D1%87%D0%B8%D1%81%D0%B5%D0%BB" \o "Теория чисел),
* [Теория вероятностей](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B2%D0%B5%D1%80%D0%BE%D1%8F%D1%82%D0%BD%D0%BE%D1%81%D1%82%D0%B5%D0%B9" \o "Теория вероятностей) и [математическая статистика](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%81%D1%82%D0%B0%D1%82%D0%B8%D1%81%D1%82%D0%B8%D0%BA%D0%B0" \o "Математическая статистика),
* [Комбинаторика](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%B1%D0%B8%D0%BD%D0%B0%D1%82%D0%BE%D1%80%D0%B8%D0%BA%D0%B0" \o "Комбинаторика) и [теория графов](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2" \o "Теория графов),
* [Фрактальная геометрия](https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B0%D0%BA%D1%82%D0%B0%D0%BB" \o "Фрактал),
* [Вычислительная математика](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0" \o "Вычислительная математика),
* [Теория принятия решений](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%BF%D1%80%D0%B8%D0%BD%D1%8F%D1%82%D0%B8%D1%8F_%D1%80%D0%B5%D1%88%D0%B5%D0%BD%D0%B8%D0%B9" \o "Теория принятия решений),
* [Математическое моделирование](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5" \o "Математическое моделирование).

Кроме того, многие закономерности, обнаруженные в игре, имеют свои аналогии в других, подчас совершенно «нематематических» дисциплинах. Вот список наук, теории которых имеют интересные точки соприкосновения с феноменами «Жизни»:

* [Кибернетика](https://ru.wikipedia.org/wiki/%D0%9A%D0%B8%D0%B1%D0%B5%D1%80%D0%BD%D0%B5%D1%82%D0%B8%D0%BA%D0%B0" \o "Кибернетика). Сама игра является удачной попыткой Конвея доказать существование простых самовоспроизводящихся систем, а также появление некоего «разума» у самовоспроизводящихся систем.
* [Биология](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D1%8F" \o "Биология). Внешнее сходство с развитием популяций примитивных организмов впечатляет.
* [Бактериология](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%BA%D1%82%D0%B5%D1%80%D0%B8%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D1%8F" \o "Бактериология). Некоторые интересные вариации игры с дополнительными условиями могут с точностью повторить размножение бактерий, которые с случайной вероятностью могут мутировать (по условию модификации).
* [Физиология](https://ru.wikipedia.org/wiki/%D0%A4%D0%B8%D0%B7%D0%B8%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D1%8F" \o "Физиология). Рождение и смерть клеток подобны процессу возникновения и исчезновения нейронных импульсов.
* [Астрономия](https://ru.wikipedia.org/wiki/%D0%90%D1%81%D1%82%D1%80%D0%BE%D0%BD%D0%BE%D0%BC%D0%B8%D1%8F" \o "Астрономия). Эволюции некоторых сложных колоний удивительным образом схематично повторяют этапы развития [спиралевидных галактик](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D0%BE%D0%B5%D0%BD%D0%B8%D0%B5_%D0%B3%D0%B0%D0%BB%D0%B0%D0%BA%D1%82%D0%B8%D0%BA" \o "Строение галактик)
* [Физика твёрдого тела](https://ru.wikipedia.org/wiki/%D0%A4%D0%B8%D0%B7%D0%B8%D0%BA%D0%B0_%D1%82%D0%B2%D1%91%D1%80%D0%B4%D0%BE%D0%B3%D0%BE_%D1%82%D0%B5%D0%BB%D0%B0" \o "Физика твёрдого тела). Теория автоматов вообще и игра «Жизнь» в частности используются для анализа «явлений переноса» —  [диффузии](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%84%D1%84%D1%83%D0%B7%D0%B8%D1%8F" \o "Диффузия), [вязкости](https://ru.wikipedia.org/wiki/%D0%92%D1%8F%D0%B7%D0%BA%D0%BE%D1%81%D1%82%D1%8C" \o "Вязкость) и [теплопроводности](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BF%D0%BB%D0%BE%D0%BF%D1%80%D0%BE%D0%B2%D0%BE%D0%B4%D0%BD%D0%BE%D1%81%D1%82%D1%8C" \o "Теплопроводность).
* [Квантовая физика](https://ru.wikipedia.org/wiki/%D0%9A%D0%B2%D0%B0%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D1%8F_%D1%84%D0%B8%D0%B7%D0%B8%D0%BA%D0%B0" \o "Квантовая физика). Поведение «жизненных» ячеек (рождение новых и взаимное уничтожение) во многом напоминают процессы, происходящие при столкновении [элементарных частиц](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BC%D0%B5%D0%BD%D1%82%D0%B0%D1%80%D0%BD%D1%8B%D0%B5_%D1%87%D0%B0%D1%81%D1%82%D0%B8%D1%86%D1%8B" \o "Элементарные частицы).
* [Наномеханика](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D0%BD%D0%BE%D1%82%D0%B5%D1%85%D0%BD%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D1%8F" \o "Нанотехнология). Стационарные и пульсирующие колонии являются показательным примером простейших устройств, созданных на основе нанотехнологий.
* [Электротехника](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D1%82%D0%B5%D1%85%D0%BD%D0%B8%D0%BA%D0%B0" \o "Электротехника). Правила игры используются для моделирования самовосстанавливающихся [электрических цепей](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%86%D0%B5%D0%BF%D1%8C" \o "Электрическая цепь).
* [Химия](https://ru.wikipedia.org/wiki/%D0%A5%D0%B8%D0%BC%D0%B8%D1%8F" \o "Химия). Конфигурации, подобные строящимся в игре, возникают во время химических реакций на поверхности; в частности, в опытах М. С. Шакаевой возникают движущиеся молекулярные конструкции, аналогичные «жизненному» планеру. Также предпринимаются попытки объяснить периодические [химические реакции](https://ru.wikipedia.org/wiki/%D0%A5%D0%B8%D0%BC%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B5_%D1%80%D0%B5%D0%B0%D0%BA%D1%86%D0%B8%D0%B8" \o "Химические реакции) с помощью многомерных клеточных автоматов. Самоорганизацией элементарных частиц также занимается [супрамолекулярная химия](https://ru.wikipedia.org/wiki/%D0%A1%D1%83%D0%BF%D1%80%D0%B0%D0%BC%D0%BE%D0%BB%D0%B5%D0%BA%D1%83%D0%BB%D1%8F%D1%80%D0%BD%D0%B0%D1%8F_%D1%85%D0%B8%D0%BC%D0%B8%D1%8F" \o "Супрамолекулярная химия).

Возможно, эта игра связана и с другими научными явлениями, в том числе и с теми, о которых современной науке пока неизвестно. Также возможно, что не открытые на сегодня законы природы и общества станут более понятными благодаря «Жизни» и её модификациям.

**Реализовать**

Задание 42 Моделирующая игра «Жизнь»

Напишите программу, моделирующую колонию Жизни. Исходными данными служит начальное расположение животных (заданное пользователем или получаемое случайно - реализовать оба случая), а в качестве результата нужно получить вид сверху в графическом режиме всех поколений колонии.

**Структурное описание разработки**

Описание классов: спецификации данных, методов

Программа состоит из 1 класса.

* ConwaysGameOfLife

В нем содержится полная реализация поставленной задачи.

Программа написана при помощи Javax swing (**Swing** — [библиотека](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)" \o "Библиотека (программирование)) для создания [графического интерфейса](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8F" \o "Графический интерфейс пользователя) для программ на языке [Java](https://ru.wikipedia.org/wiki/Java" \o "Java). Swing был разработан компанией [Sun Microsystems](https://ru.wikipedia.org/wiki/Sun_Microsystems" \o "Sun Microsystems). Он содержит ряд графических [компонентов](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BC%D0%B5%D0%BD%D1%82_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81%D0%B0" \o "Элемент интерфейса) ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA" \o "Английский язык) *Swing widgets*), таких как кнопки, поля ввода, таблицы и т. д.)

**Функциональное описание разработки**

*Задание размеров поля и клеток в нем*

public class ConwaysGameOfLife extends JFrame implements ActionListener

*Настройка спецификации swing*

public static void main(String[] args) {  
 JFrame game = new ConwaysGameOfLife();

*Настройка меню*

public ConwaysGameOfLife() {

*Настройка доски игры*

gb\_gameBoard = new GameBoard();  
add(gb\_gameBoard);

*Разыгровка*

public void setGameBeingPlayed(boolean isBeingPlayed) {

*Программирование действий с панели управления а так же добавление информации.*

public void actionPerformed

*Array list под память а так же увеличение размера*

private ArrayList<Point>

updateArraySize

*Обработка координат клика с мыши*

public void addPoint

*Перезагрузка игрового окна*

public void removePoint

*Очистка всего поля игры*

public void resetBoard

*Рандомная генерация клеток на игровом поле*

public void randomlyFillBoard

*Прорисовка клеток по координатам мыши*

public void paintComponent

*Установка правильного размера игрового поля*

public void componentResized

*Обработка движения мыши как выбор нескольких рядом лежащих точек*

public void mouseDragged

*Основа игры, правила жизни и смерти точек*

public void run

**Описание пользовательского интерфейса**

На рис. 1 представлен стартовый экран приложения. Он содержит кнопки:

* File – переход к вкладке, в которой осуществляется управление скоростью игры, а так же выход из приложения.
* Game – открытие окна запуска игры с рандомной  
   генерацией клеток, запуска, паузы, очистки окна.
* Help – открытие диалогового окна с краткой информацией об игре и кнопкой ссылкой на исходник проекта в github.

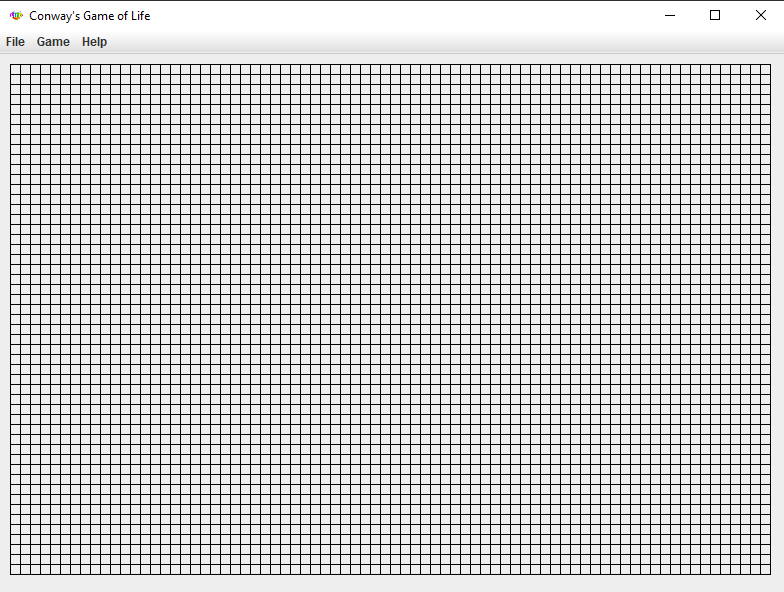
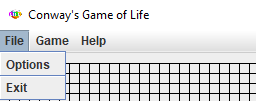
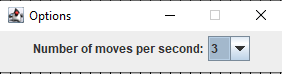
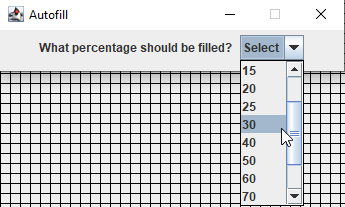


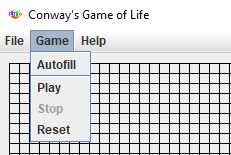
Рис. 1 – Стартовый экран приложения

После нажатия на кнопку File

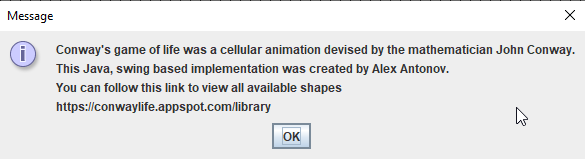
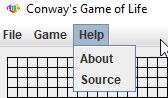


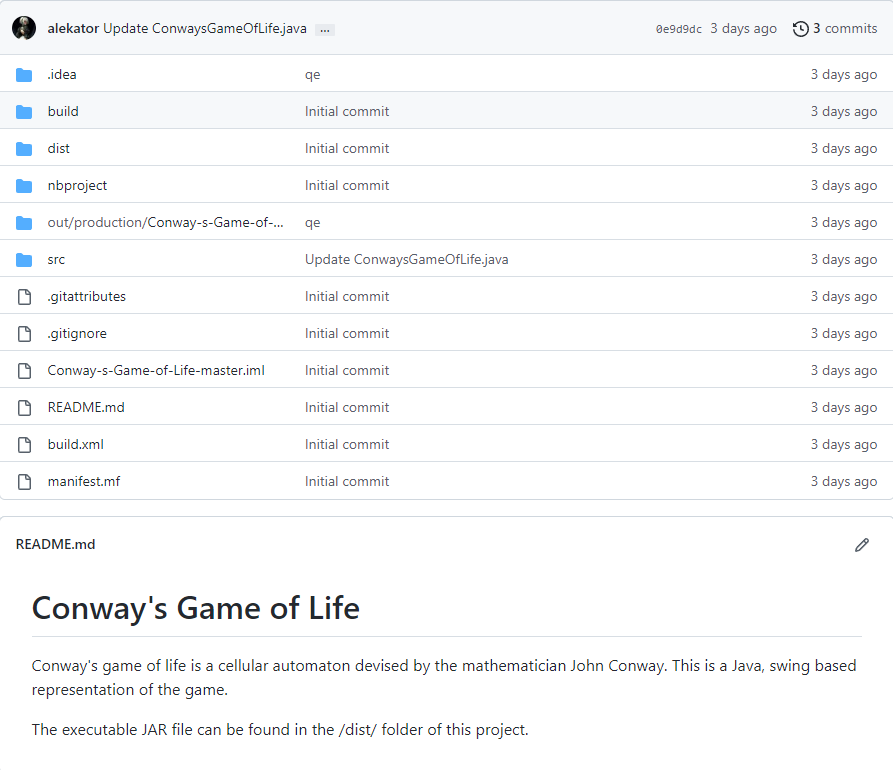


После нажатия на кнопку Game, Autofill (рандомная генерация клеток)



После нажатия на кнопку Help, About - выводит информацию об игре, Sours - перебрасывает нас на Github





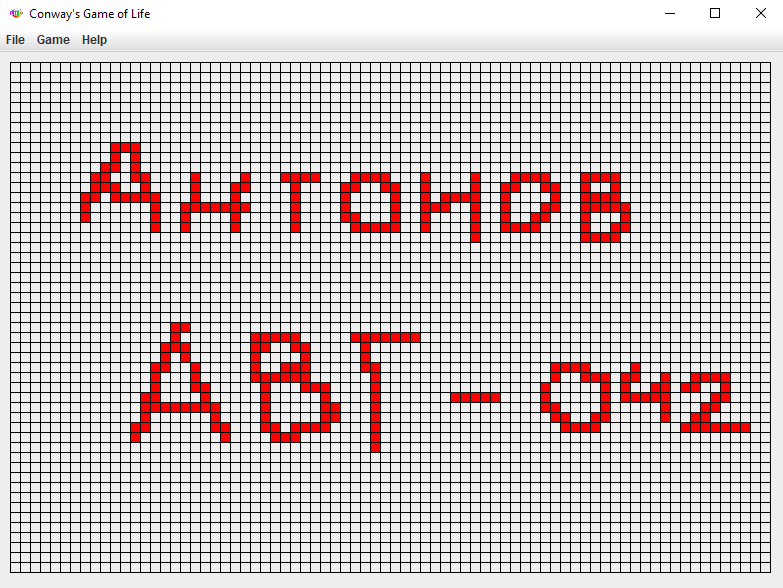


Рис. 2 - задание начальной симуляции клеток

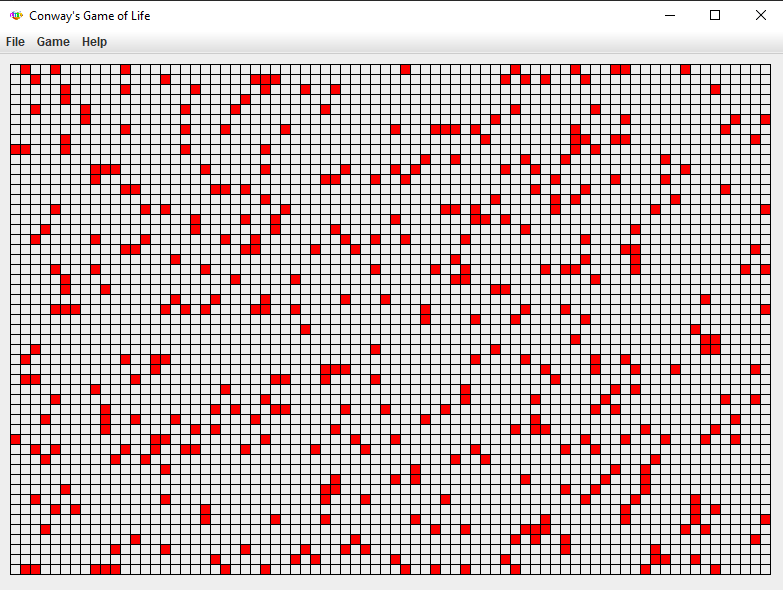


Рис. 3 - Рандомная стартовая генерация

**Руководство пользователя**

После запуска приложения перед вами появится стартовый экран приложения, представленный на Рис. 1.

При помощи мыши вы можете создать любую начальную генерацию клеток Рис 2. и после этого начать симуляцию через панель Game - Play..

Если же вы хотите использовать рандомную симуляцию, Через панель Game - Autofill Рис.3. выберите количество клеток и запускайте через Play.

Для паузы используйте Stop, Reset для очистки игрового поля.

Воспользуйтесь вкладкой Help если вы так и не поняли что нужно делать...

**Руководство программиста**

Програма требует openjdk-18 для корректной работы.

**Заключение**

Была реализована дна из сложнейших игр - клеточный автомат *Conway's Game of Life,* важность которой сложно переоценить, эта игра описывает огромное множество вещей в нашей вселенной, от поведения мельчайших молекул до величественных космических тел.

**Приложение**

import java.awt.\*;  
import java.awt.event.\*;  
import java.net.URI;  
import java.util.ArrayList;  
import java.util.ConcurrentModificationException;  
import javax.swing.\*;  
  
*/\*\*  
 \* Conway's game of life is a cellular automaton devised by the  
 \* mathematician John Conway.  
 \*/*public class ConwaysGameOfLife extends JFrame implements ActionListener {  
 private static final Dimension *DEFAULT\_WINDOW\_SIZE* = new Dimension(800, 600);  
 private static final Dimension *MINIMUM\_WINDOW\_SIZE* = new Dimension(400, 400);  
 private static final int *BLOCK\_SIZE* = 10;  
  
 private JMenuBar mb\_menu;  
 private JMenu m\_file, m\_game, m\_help;  
 private JMenuItem mi\_file\_options, mi\_file\_exit;  
 private JMenuItem mi\_game\_autofill, mi\_game\_play, mi\_game\_stop, mi\_game\_reset;  
 private JMenuItem mi\_help\_about, mi\_help\_source;  
 private int i\_movesPerSecond = 3;  
 private GameBoard gb\_gameBoard;  
 private Thread game;  
   
 public static void main(String[] args) {  
 // Setup the swing specifics  
 JFrame game = new ConwaysGameOfLife();  
 game.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 game.setTitle("Conway's Game of Life");  
 game.setIconImage(new ImageIcon(ConwaysGameOfLife.class.getResource("/images/logo.png")).getImage());  
 game.setSize(*DEFAULT\_WINDOW\_SIZE*);  
 game.setMinimumSize(*MINIMUM\_WINDOW\_SIZE*);  
 game.setLocation((Toolkit.*getDefaultToolkit*().getScreenSize().width - game.getWidth())/2,   
 (Toolkit.*getDefaultToolkit*().getScreenSize().height - game.getHeight())/2);  
 game.setVisible(true);  
 }  
   
 public ConwaysGameOfLife() {  
 // Setup menu  
 mb\_menu = new JMenuBar();  
 setJMenuBar(mb\_menu);  
 m\_file = new JMenu("File");  
 mb\_menu.add(m\_file);  
 m\_game = new JMenu("Game");  
 mb\_menu.add(m\_game);  
 m\_help = new JMenu("Help");  
 mb\_menu.add(m\_help);  
 mi\_file\_options = new JMenuItem("Options");  
 mi\_file\_options.addActionListener(this);  
 mi\_file\_exit = new JMenuItem("Exit");  
 mi\_file\_exit.addActionListener(this);  
 m\_file.add(mi\_file\_options);  
 m\_file.add(new JSeparator());  
 m\_file.add(mi\_file\_exit);  
 mi\_game\_autofill = new JMenuItem("Autofill");  
 mi\_game\_autofill.addActionListener(this);  
 mi\_game\_play = new JMenuItem("Play");  
 mi\_game\_play.addActionListener(this);  
 mi\_game\_stop = new JMenuItem("Stop");  
 mi\_game\_stop.setEnabled(false);  
 mi\_game\_stop.addActionListener(this);  
 mi\_game\_reset = new JMenuItem("Reset");  
 mi\_game\_reset.addActionListener(this);  
 m\_game.add(mi\_game\_autofill);  
 m\_game.add(new JSeparator());  
 m\_game.add(mi\_game\_play);  
 m\_game.add(mi\_game\_stop);  
 m\_game.add(mi\_game\_reset);  
 mi\_help\_about = new JMenuItem("About");  
 mi\_help\_about.addActionListener(this);  
 mi\_help\_source = new JMenuItem("Source");  
 mi\_help\_source.addActionListener(this);  
 m\_help.add(mi\_help\_about);  
 m\_help.add(mi\_help\_source);  
 // Setup game board  
 gb\_gameBoard = new GameBoard();  
 add(gb\_gameBoard);  
 }  
   
 public void setGameBeingPlayed(boolean isBeingPlayed) {  
 if (isBeingPlayed) {  
 mi\_game\_play.setEnabled(false);  
 mi\_game\_stop.setEnabled(true);  
 game = new Thread(gb\_gameBoard);  
 game.start();  
 } else {  
 mi\_game\_play.setEnabled(true);  
 mi\_game\_stop.setEnabled(false);  
 game.interrupt();  
 }  
 }  
   
 @Override  
 public void actionPerformed(ActionEvent ae) {  
 if (ae.getSource().equals(mi\_file\_exit)) {  
 // Exit the game  
 System.*exit*(0);  
 } else if (ae.getSource().equals(mi\_file\_options)) {  
 // Put up an options panel to change the number of moves per second  
 final JFrame f\_options = new JFrame();  
 f\_options.setTitle("Options");  
 f\_options.setSize(300,80);  
 f\_options.setLocation((Toolkit.*getDefaultToolkit*().getScreenSize().width - f\_options.getWidth())/2,   
 (Toolkit.*getDefaultToolkit*().getScreenSize().height - f\_options.getHeight())/2);  
 f\_options.setResizable(false);  
 JPanel p\_options = new JPanel();  
 p\_options.setOpaque(false);  
 f\_options.add(p\_options);  
 p\_options.add(new JLabel("Number of moves per second:"));  
 Integer[] secondOptions = {1,2,3,4,5,10,15,20};  
 final JComboBox cb\_seconds = new JComboBox(secondOptions);  
 p\_options.add(cb\_seconds);  
 cb\_seconds.setSelectedItem(i\_movesPerSecond);  
 cb\_seconds.addActionListener(new ActionListener(){  
 @Override  
 public void actionPerformed(ActionEvent ae) {  
 i\_movesPerSecond = (Integer)cb\_seconds.getSelectedItem();  
 f\_options.dispose();  
 }  
 });  
 f\_options.setVisible(true);  
 } else if (ae.getSource().equals(mi\_game\_autofill)) {  
 final JFrame f\_autoFill = new JFrame();  
 f\_autoFill.setTitle("Autofill");  
 f\_autoFill.setSize(360, 80);  
 f\_autoFill.setLocation((Toolkit.*getDefaultToolkit*().getScreenSize().width - f\_autoFill.getWidth())/2,   
 (Toolkit.*getDefaultToolkit*().getScreenSize().height - f\_autoFill.getHeight())/2);  
 f\_autoFill.setResizable(false);  
 JPanel p\_autoFill = new JPanel();  
 p\_autoFill.setOpaque(false);  
 f\_autoFill.add(p\_autoFill);  
 p\_autoFill.add(new JLabel("What percentage should be filled? "));  
 Object[] percentageOptions = {"Select",5,10,15,20,25,30,40,50,60,70,80,90,95};  
 final JComboBox cb\_percent = new JComboBox(percentageOptions);  
 p\_autoFill.add(cb\_percent);  
 cb\_percent.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 if (cb\_percent.getSelectedIndex() > 0) {  
 gb\_gameBoard.resetBoard();  
 gb\_gameBoard.randomlyFillBoard((Integer)cb\_percent.getSelectedItem());  
 f\_autoFill.dispose();  
 }  
 }  
 });  
 f\_autoFill.setVisible(true);  
 } else if (ae.getSource().equals(mi\_game\_reset)) {  
 gb\_gameBoard.resetBoard();  
 gb\_gameBoard.repaint();  
 } else if (ae.getSource().equals(mi\_game\_play)) {  
 setGameBeingPlayed(true);  
 } else if (ae.getSource().equals(mi\_game\_stop)) {  
 setGameBeingPlayed(false);  
 } else if (ae.getSource().equals(mi\_help\_source)) {  
 Desktop desktop = Desktop.*isDesktopSupported*() ? Desktop.*getDesktop*() : null;  
 try {  
 desktop.browse(new URI("https://github.com/alekator/Conway-s-Game-of-Life-master"));  
 } catch (Exception ex) {  
 JOptionPane.*showMessageDialog*(null, "Source is available on GitHub at:\nhttps://github.com/alekator/Conway-s-Game-of-Life-master", "Source", JOptionPane.*INFORMATION\_MESSAGE*);  
 }  
 } else if (ae.getSource().equals(mi\_help\_about)) {  
 JOptionPane.*showMessageDialog*(null, "Conway's game of life was a cellular animation devised by the mathematician John Conway.\nThis Java, swing based implementation was created by Alex Antonov.\nYou can follow this link to view all available shapes\nhttps://conwaylife.appspot.com/library");  
 }  
 }  
   
 private class GameBoard extends JPanel implements ComponentListener, MouseListener, MouseMotionListener, Runnable {  
 private Dimension d\_gameBoardSize = null;  
 private ArrayList<Point> point = new ArrayList<Point>(0);  
   
 public GameBoard() {  
 // Add resizing listener  
 addComponentListener(this);  
 addMouseListener(this);  
 addMouseMotionListener(this);  
 }  
   
 private void updateArraySize() {  
 ArrayList<Point> removeList = new ArrayList<Point>(0);  
 for (Point current : point) {  
 if ((current.x > d\_gameBoardSize.width-1) || (current.y > d\_gameBoardSize.height-1)) {  
 removeList.add(current);  
 }  
 }  
 point.removeAll(removeList);  
 repaint();  
 }  
   
 public void addPoint(int x, int y) {  
 if (!point.contains(new Point(x,y))) {  
 point.add(new Point(x,y));  
 }   
 repaint();  
 }  
   
 public void addPoint(MouseEvent me) {  
 int x = me.getPoint().x/*BLOCK\_SIZE*-1;  
 int y = me.getPoint().y/*BLOCK\_SIZE*-1;  
 if ((x >= 0) && (x < d\_gameBoardSize.width) && (y >= 0) && (y < d\_gameBoardSize.height)) {  
 addPoint(x,y);  
 }  
 }  
 /\* public void delPoint(int x, int y) {  
 point.remove(new Point(x,y));  
  
 }  
 public void delPoint(MouseEvent me) {  
 int x = me.getPoint().x/BLOCK\_SIZE-1;  
 int y = me.getPoint().y/BLOCK\_SIZE-1;  
 if ((x >= 0) && (x < d\_gameBoardSize.width) && (y >= 0) && (y < d\_gameBoardSize.height)) {  
 delPoint(x,y);  
 }  
 }\*/  
   
 public void removePoint(int x, int y) {  
 point.remove(new Point(x,y));  
 }  
   
 public void resetBoard() {  
 point.clear();  
 }  
   
 public void randomlyFillBoard(int percent) {  
 for (int i=0; i<d\_gameBoardSize.width; i++) {  
 for (int j=0; j<d\_gameBoardSize.height; j++) {  
 if (Math.*random*()\*100 < percent) {  
 addPoint(i,j);  
 }  
 }  
 }  
 }  
   
 @Override  
 public void paintComponent(Graphics g) {  
 super.paintComponent(g);  
 try {  
 for (Point newPoint : point) {  
 // Draw new point  
 g.setColor(Color.*red*);  
 g.fillRect(*BLOCK\_SIZE* + (*BLOCK\_SIZE*\*newPoint.x), *BLOCK\_SIZE* + (*BLOCK\_SIZE*\*newPoint.y), *BLOCK\_SIZE*, *BLOCK\_SIZE*);  
 }  
 } catch (ConcurrentModificationException cme) {}  
 // Setup grid  
 g.setColor(Color.*BLACK*);  
 for (int i=0; i<=d\_gameBoardSize.width; i++) {  
 g.drawLine(((i\**BLOCK\_SIZE*)+*BLOCK\_SIZE*), *BLOCK\_SIZE*, (i\**BLOCK\_SIZE*)+*BLOCK\_SIZE*, *BLOCK\_SIZE* + (*BLOCK\_SIZE*\*d\_gameBoardSize.height));  
 }  
 for (int i=0; i<=d\_gameBoardSize.height; i++) {  
 g.drawLine(*BLOCK\_SIZE*, ((i\**BLOCK\_SIZE*)+*BLOCK\_SIZE*), *BLOCK\_SIZE*\*(d\_gameBoardSize.width+1), ((i\**BLOCK\_SIZE*)+*BLOCK\_SIZE*));  
 }  
 }  
  
 @Override  
 public void componentResized(ComponentEvent e) {  
 // Setup the game board size with proper boundries  
 d\_gameBoardSize = new Dimension(getWidth()/*BLOCK\_SIZE*-2, getHeight()/*BLOCK\_SIZE*-2);  
 updateArraySize();  
 }  
 @Override  
 public void componentMoved(ComponentEvent e) {}  
 @Override  
 public void componentShown(ComponentEvent e) {}  
 @Override  
 public void componentHidden(ComponentEvent e) {}  
 @Override  
 public void mouseClicked(MouseEvent e) {}  
 @Override  
 public void mousePressed(MouseEvent e) {}  
 @Override  
 public void mouseReleased(MouseEvent e) {  
 // Mouse was released (user clicked)  
 addPoint(e);  
 }  
 @Override  
 public void mouseEntered(MouseEvent e) {}  
  
 @Override  
 public void mouseExited(MouseEvent e) {}  
  
 @Override  
 public void mouseDragged(MouseEvent e) {  
 // Mouse is being dragged, user wants multiple selections  
 addPoint(e);  
 }  
 @Override  
 public void mouseMoved(MouseEvent e) {}  
  
 @Override  
 public void run() {  
 boolean[][] gameBoard = new boolean[d\_gameBoardSize.width+2][d\_gameBoardSize.height+2];  
 for (Point current : point) {  
 gameBoard[current.x+1][current.y+1] = true;  
 }  
 ArrayList<Point> survivingCells = new ArrayList<Point>(0);  
 // Iterate through the array, follow game of life rules  
 for (int i=1; i<gameBoard.length-1; i++) {  
 for (int j=1; j<gameBoard[0].length-1; j++) {  
 int surrounding = 0;  
 if (gameBoard[i-1][j-1]) { surrounding++; }  
 if (gameBoard[i-1][j]) { surrounding++; }  
 if (gameBoard[i-1][j+1]) { surrounding++; }  
 if (gameBoard[i][j-1]) { surrounding++; }  
 if (gameBoard[i][j+1]) { surrounding++; }  
 if (gameBoard[i+1][j-1]) { surrounding++; }  
 if (gameBoard[i+1][j]) { surrounding++; }  
 if (gameBoard[i+1][j+1]) { surrounding++; }  
 if (gameBoard[i][j]) {  
 // Cell is alive, Can the cell live? (2-3)  
 if ((surrounding == 2) || (surrounding == 3)) {  
 survivingCells.add(new Point(i-1,j-1));  
 }   
 } else {  
 // Cell is dead, will the cell be given birth? (3)  
 if (surrounding == 3) {  
 survivingCells.add(new Point(i-1,j-1));  
 }  
 }  
 }  
 }  
 resetBoard();  
 point.addAll(survivingCells);  
 repaint();  
 try {  
 Thread.*sleep*(1000/i\_movesPerSecond);  
 run();  
 } catch (InterruptedException ex) {}  
 }  
 }  
}

Список источников

1. *Andrew Adamatzky.* Game of Life Cellular Automata. — Springer-Verlag London, 2010.
2. *Хэйес Б.* Клеточный автомат создает модель мира и мир вокруг себя. // [В мире науки](https://ru.wikipedia.org/wiki/%D0%92_%D0%BC%D0%B8%D1%80%D0%B5_%D0%BD%D0%B0%D1%83%D0%BA%D0%B8" \o "В мире науки), 1984, № 5, с.97-104
3. <https://conwaylife.appspot.com/library>
4. <https://hr-vector.com/java/swing-graficheskij-interfejs>