САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕРТА ВЕЛИКОГО

Институт компьютерных наук и кибербезопасности

Высшая школа технологий искусственного интеллекта

Направление 02.03.01 Математика и компьютерные науки

Отчет по лабораторной работе №1

«Двухмерный клеточный автомат»

по дисциплине «Теория Алгоритмов»

Выполнил:

Салимли А. _____

Преподаватель:

Востров А.В. _____

«___»_____ 20__г.

Санкт-Петербург - 2024 г.

Содержание

Введение

Данный отчет содержит описание лабораторной работы №1 по дисциплине «Теория алгоритмов».

Задача: Реализовать двухмерный клеточный автомат по варианту, с выбранными граничными условиями. Должна присутствовать визуализация работы автомата, либо консольным выводом либо с использованием графических интерфейсов. В программе должны присутствовать следующие функции:

1. Возможность генерации случайного состояния
2. Возможность ручной настройки состояния
3. Возможность задания размера поля
4. Возможность задания количества итераций

Вариант: №17

Дата рождения (для получения вектора функции): 08.04.2003

Итоговое число: 1089632

Итоговый вектор: 00000000000100001010000001100000

Граничные условия: Единичные

1 Математическое описание

1.1 Клеточный автомат

Клеточный автомат - это регулярное структура динамических объектов (клеток), функционирующих синхронно. Клеточный автоматы моделируют процессы разворачивающиеся в дискретном пространстве и в дискретном времени.

Клеточный автомат имеет состояние определяемое как набор (вектор или матрица) состояний из компонентных автоматов. Каждый автомат функционирует как автомат Мура, то есть это автомат без выхода, выходом является его состояние. Вход на каждом шаге клеточного автомата это состояние его соседей на следующем шаге новое состояние каждого автомата определяется как функция его текущего состояния и текущих состояние его соседей.

Обычно правило изменения состояния всех автоматов (кроме крайних) идентичны , множества состояний каждого автомата конечно. Изменение состояния системы во всех клетках происходит синхронно состояние клеток меняются одновременно на каждом такте.

1.1.1 Описание функции и тип клеточного автомата

Для варианта под номером 17. Алгоритмом создается число, в DEC формате, которое затем нужно перевести в BIN формат. После чего построить таблицу истинности для пяти переменных. Созданное число для варианта 17 - 17 x 2003 x 4 x 8 =1089615 (dec). 1089615 = 0000 0000 0001 0000 1010 0000 0110 0000 (bin). Переменные таблицы истинности являются значениями клеток по окрестности Фон-Неймана. Полученная таблица истинности приведена в рисунке 1. По полученному вектору бинарной функции от пяти переменных: 1 - клетка живая, 0 - клетка мертвая. Формула для получения состояния на i+1 итерации:

$$F_{+1} = (\overline{s_0}s_1\overline{s_2}s_3s_4) \vee (\overline{s_0}s_1\overline{s_2}s_3\overline{s_4}) \vee (\overline{s_0}s_1\overline{s_2}s_3s_4) \vee (\overline{s_0}s_1s_2s_3s_4) \vee$$
$$(s_0\overline{s_1}s_2s_3s_4) \vee (s_0\overline{s_1}\overline{s_2}s_3s_4) \vee (s_0\overline{s_1}s_2\overline{s_3}\overline{s_4}) \vee (s_0\overline{s_1}s_2s_3s_4) \vee (s_0s_1\overline{s_2}s_3s_4) \vee (s_0s_1s_2\overline{s_3}\overline{s_4})$$

где, F_{+1} - состояние клетки на i+1 итерации.

s_0,s_1,s_2,s_3,s_4 - состояния клеток из окрестности Фон-Неймана на текущей итерации. 0 - центр, 1 - вверх, 2 - низ, 3 - лево, 4 - право.

s0	s1	s2	s3	s4	F
0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	1	0	0
0	0	0	1	1	0
0	0	1	0	0	0
0	0	1	0	1	0
0	0	1	1	0	0
0	0	1	1	1	0
0	1	0	0	0	0
0	1	0	0	1	0
0	1	0	1	0	0
0	1	0	1	1	1
0	1	1	0	0	0
0	1	1	0	1	0
0	1	1	1	0	0
1	1	1	1	1	0
1	0	0	0	0	1
1	0	0	0	1	0
1	0	0	1	0	1
1	0	0	1	1	0
1	0	1	0	0	0
1	0	1	0	1	0
1	0	1	1	0	0
1	0	1	1	1	0
1	1	0	0	0	0
1	1	0	0	1	1
1	1	0	1	0	1
1	1	0	1	1	0
1	1	1	0	0	0
1	1	1	0	1	0
1	1	1	1	0	0
1	1	1	1	1	0

Рис.1 Таблица истинности

Окрестность фон Неймана порядка *r* - множество клеток, манхэттенское расстояние до которых от данной клетки не превышает r. Окрестность Фон-Неймана порядка *r* имеет форму ромба и включает в себя $C_{4,r+1}$ клеток, где

$C_{4,r+1} = n^2 + (n-1)^2$ - *n-е* центрированное квадратное число.

В данной работе рассматривается окрестность размера $r = 1$ в двумерном пространстве.

Окрестность Фон-Неймана порядка $r = 1$ (рисунок 2).

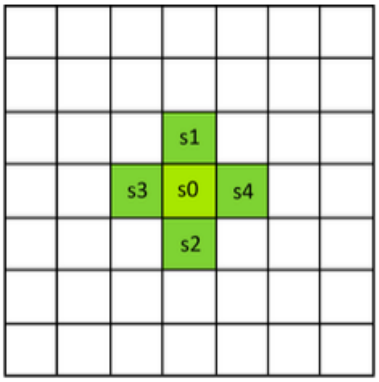

Рис. 2 Окрестность фон Неймана

Для получения информации о состоянии клеток на следующий итерации берутся значения выделенные зеленым цветом в таблицы истинности. То есть те которые равны 1. Пример:

Для строки №14 = (01011 | 1) будет верно:

- Центральная клетка была мертва
- Клетка сверху была жива
- Клетка снизу была мертва
- Клетка слева была жива
- Клетка справа была жива

Это одно из условий жизни клетки перечисленных в функции. Всего условий 32 (2^5), из них, 5 - при которых на $i+1$, клетка жива и 28 - при которых на $i+1$ клетка мертва.

Для разрешения крайних случаев, было выбрано **единичное** граничное условие. При этом, если клетка находится на границе то мы берем сторону за единицу на $i+1$ итерации.

Пример: Для клетки с координатами $(0,0)$:

- Если клетка была мертва: $s_0 = 0$, $s_1 = 1$(на границе), $s_2 = 0/1$, $s_3 = 1$(на границе), $s_4 = 0/1$
- Если клетка была жива: $s_0 = 1$, $s_1 = 1$(на границе), $s_2 = 0/1$, $s_3 = 1$(на границе), $s_4 = 0/1$

Это можно задать формулой, где *C* - состояние клетки в (i, j) координате:

$$C(i \pm 1, j \pm 1) = \begin{cases} C(i \pm 1, j \pm 1), & \text{если сосед внутри границы} \\ 1, & \text{если сосед за границей} \end{cases}$$

2 Особенности реализации

Реализация программы, выбрана структурой проекта Maven и JavaFX, в структуре которой есть 4 основных файла отвечающие за работу программы:

- module-info.java // За модули экспортируемые в остальные файлы и библиотеки
- HelloApplication.java // Основной (Main) файл проекта отвечает за старт и выход программы
- HelloController.java // Файл где прописана логика работы и взаимодействие с UI
- hello-view.fxml // Файл структурно описывающий UI дизайн программы

Файл HelloApplication - содержит взаимодействие экспортируемых библиотек и модулей с файлов HelloController.java и module-info.java. При работе с UI, использовалась утилита SceneBuilder, который взаимодействует с файлом hello-view.fxml, который в свою очередь создает дизайн кнопок, сетки, сцены, полей. После чего в SceneBuilder присваиваются по названиям функции из HelloController.java. В итоге HelloApplication собирает все библиотеки запускает объект сцены в котором собрана вся логика остальных файлов.

2.1 Структура клеточного автомата

Как было сказано ранее, HelloController - класс отвечающий за логику работы автомата и работу интерфейса. Все TextField - поля которые должен заполнить пользователь. Заполнение идет по gridSize - размер сетки, iterations - итерации, AnchorPane - сцена в которой будут все элементы взаимодействия с пользователем, Rectangle cells - клетки в сетке, truthTable - таблица истинности. В классе HelloApplication.java - start - запуск программы, exit - выход.

2.1.1 Поля класса

```
public class HelloController {
@FXML
private boolean isRunning;
@FXML
private TextField tGridW;
@FXML
private TextField Titer;
@FXML
private Label checking;
@FXML
private AnchorPane gridPane;
private Rectangle[][] cells;
private int gridSize;
private int iterations;
private boolean isGridInitialized;
private Map<String, Boolean> truthTable = new HashMap<>();
private Random random = new Random();
@FXML
private boolean isFirstRun = true;
}
```

2.1.2 Конструктор

JVM - неявно создает конструктор по умолчанию.

2.1.3 Функция случайной генерации

Вход: клеточный автомат в состоянии S
Выход: клеточный автомат в случайно сгенериррованном состоянии.
Функция создаёт сетку с случайными состояниями ячеек, если жива - клетка темно-синяя, если нет светло-серая.

```
    private void createGrid(boolean randomFill) {
        gridPane.getChildren().clear();
        cells = new Rectangle[gridSize][gridSize];
        isGridInitialized = true;
        double cellSize = Math.min(gridPane.getWidth() / gridSize, gridPane.getHeight() / gridSize);

        for (int i = 0; i < gridSize; i++) {
            for (int j = 0; j < gridSize; j++) {
                Rectangle cell = new Rectangle(cellSize, cellSize);
                cell.setFill(randomFill && random.nextBoolean() ? Color.DARKBLUE : Color.LIGHTGRAY);
                cell.setStroke(Color.WHITE);
                cell.setOnMouseClicked(event -> toggleCellState(cell));

                cells[i][j] = cell;
                gridPane.getChildren().add(cell);
                cell.setLayoutX(i * cellSize);
                cell.setLayoutY(j * cellSize);
            }
        }
        gridPane.widthProperty().addListener((obs, oldVal, newVal) -> resizeCells());
        gridPane.heightProperty().addListener((obs, oldVal, newVal) -> resizeCells());
    }
```

2.1.4 Функция итерирования

Вход: клеточный автомат в состоянии S(i).

Выход: клеточный автомат в состоянии S(i+1), где i - итерация.

Функция обновляет состояние ячеек по заданным правилам.

```
private void applyRules() {
    boolean[][] newStates = new boolean[gridSize][gridSize];
    for (int x = 0; x < gridSize; x++) {
        for (int y = 0; y < gridSize; y++) {
            String neighborsState = getNeighborsState(x, y);
            newStates[x][y] = truthTable.getOrDefault(neighborsState, false);
        }
    }
    for (int i = 0; i < gridSize; i++) {
        for (int j = 0; j < gridSize; j++) {
            cells[i][j].setFill(newStates[i][j] ? Color.DARKBLUE : Color.LIGHTGRAY);
        }
    }
}
```

2.1.5 Функция очистки

Вход: клеточный автомат в состоянии S

Выход: очищенный клеточный автомат (все клетки мертвы/0)

Функция сбрасывает все ячейки к неактивному состоянию то есть к 0 (светло-серый цвет).

```
@FXML
public void pressedClear(ActionEvent event) {
    isRunning = false;
    if (cells != null) {
        for (int i = 0; i < gridSize; i++) {
            for (int j = 0; j < gridSize; j++) {
                cells[i][j].setFill(Color.LIGHTGRAY);
            }
```

```
        }
        checking.setText("Сетка очищена");
    }
}
```

2.1.6 Визуализация

Визуализация разработана в функции createGrid, этот метод создает прямоугольники (Rectangle) для каждой ячейки сетки и добавляет их в AnchorPane (gridPane) для отображения в графическом интерфейсе. Метод также обрабатывает размеры ячеек, чтобы они корректно отображались в зависимости от размеров окна.

```
private void createGrid(boolean randomFill) {
    gridPane.getChildren().clear();
    cells = new Rectangle[gridSize][gridSize];
    isGridInitialized = true;
    double cellSize = Math.min(gridPane.getWidth() / gridSize, gridPane.getHeight() / gridSize)
    for (int i = 0; i < gridSize; i++) {
        for (int j = 0; j < gridSize; j++) {
            Rectangle cell = new Rectangle(cellSize, cellSize);
            cell.setFill(randomFill && random.nextBoolean() ? Color.DARKBLUE : Color.LIGHTGRAY);
            cell.setStroke(Color.WHITE);
            cell.setOnMouseClicked(event -> toggleCellState(cell));
            cells[i][j] = cell;
            gridPane.getChildren().add(cell);
            cell.setLayoutX(i * cellSize);
            cell.setLayoutY(j * cellSize);
        }
    }
    gridPane.widthProperty().addListener((obs, oldVal, newVal) -> resizeCells());
    gridPane.heightProperty().addListener((obs, oldVal, newVal) -> resizeCells());
}
```

2.2 Функции нажатия кнопок

Для взаимодействия между пользователем и компьютером, были реализованы кнопки: «Запуск», «Сгенерировать», «Очистить», «Выйти», и два поля: «Ширина» и «Кол-во итераций».

- При первом запуске программы, пользователю необходимо ввести размер сетки в поле: «Ширина», при первом нажатии «Запуск», генерируется сетка размером N x N, где N - введенная ширина. Второе нажатие по кнопке «Запуск» - проводится первая итерация, далее каждое нажатие приведет к $i+1, i+2, \ldots i+n$ - итерациям.
- При записи в поле «Кол-во итераций» - проводится поочередно с задержкой в 0.5 секунды, от 1 до M итераций, где M - введенное пользователем количество итераций.
- При нажатии на кнопку «Сгенерировать» - новая сетка заполняется случайным образов живыми клетками.
- При нажатии кнопки «Очистить» - сетка полностью очищается (все клетки мертвые).
- При нажатии кнопки «Выйти» - пользователя спрашивают о подтверждении выхода из программы.

Кнопка запуска:

Вход: Нажатие на кнопку, значение ширины, количество итераций

Выход: клеточный автомат в состоянии S(i) с размером gridSize x gridSize.

Кнопка начинает выполнение заданного числа итераций и построение сетки при ее изменении

```
@FXML
public void pressedRUN(ActionEvent event) {
    try {
        int newGridSize = Integer.parseInt(tGridW.getText().trim());
        if(newGridSize > 200){
            checking.setText("⚠️Пожалейте компьютер⚠️");
        }
        else if (newGridSize <= 0) {
            checking.setText("Ширина сетки должна быть > 0 😡");
            return;
        }
        if (gridSize != newGridSize || !isGridInitialized) {
            gridSize = newGridSize;
            createGrid(false);
            isFirstRun = true;
        }
        isRunning = true;
        if (isFirstRun) {
            if (Titer.getText().isEmpty()) {
                checking.setText("Сетка сгенерирована 🥳");
                isRunning = false;
            } else {
                iterations = Integer.parseInt(Titer.getText().trim());
                if (iterations == 0 || iterations < 0) {
                    checking.setText("Итерации должны быть > 0❗");
                    isRunning = false;
                    return;
                }
                performIterationsWithDelay(iterations);
            }
            isFirstRun = false;
        } else {
            if (Titer.getText().isEmpty()) {
                iterations = 1;
                performIterationsWithDelay(iterations);
            } else {
                iterations = Integer.parseInt(Titer.getText().trim());
                if (iterations == 0) {
                    checking.setText("Итерации должны быть > 0❗");
                    isRunning = false;
                    return;
                }
                performIterationsWithDelay(iterations);
            }
        }
    } catch (NumberFormatException e) {
        checking.setText("Введите целые числа для шир. и итер.❗");
    }
}
```

Кнопка случайной генерации:

Вход: нажатие на кнопку, значение размера сетки

Выход: клеточный автомат в случайно сгенеррированном состоянии с размером gridSize x gridSize.

Кнопка создаёт сетку с случайным заполнением ячеек, с проверкой на ввод задания размера.

```
@FXML
public void pressedGenerate(ActionEvent actionEvent) {
    try {
        gridSize = Integer.parseInt(tGridW.getText().trim());
        createGrid(true); // Случайное заполнение сетки
        checking.setText("Результат: ");
        System.out.println("Случайная сетка создана");
    } catch (NumberFormatException e) {
        checking.setText("Введите ширину сетки!");
```

```
            System.out.println("Ошибка ввода: ширина должна быть целым числом!");
        }
    }
```

Кнопка очистки:

Вход: нажатие на кнопку

Выход: клеточный автомат в начальном состоянии (все клетки мертвы/0)

Кнопка очищает сетку, возвращая все ячейки в начальное состояние.

```
@FXML
public void pressedClear(ActionEvent event) {
    isRunning = false;
    if (cells != null) {
        for (int i = 0; i < gridSize; i++) {
            for (int j = 0; j < gridSize; j++) {
                cells[i][j].setFill(Color.LIGHTGRAY);
            }
        }
        checking.setText("Сетка очищена");
    }
}
```

Кнопка выхода:

Вход: нажатие на кнопку

Выход: Alert уведомление

Кнопка завершает работу приложения с подтверждением.

```
public void exit(Stage stage) {
    Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
    alert.setTitle("Стойте✋");
    alert.setHeaderText("⚠️Вы нажали на выйти!⚠️");
    alert.setContentText("Вы уверены что хотите выйти?🤨");

    if (alert.showAndWait().get() == ButtonType.OK) {
        System.out.println("Удачи!");
        stage.close();
    }
}
```

3 Результаты работы

На рисунках 3-10 показаны снимки экрана, работы программы клеточного автомата с размером сетки 20x20:

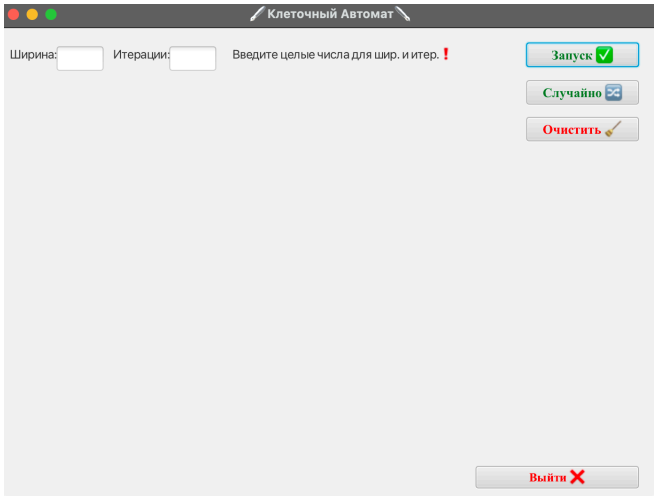

Рис.3 Попытка запуска без указания полей

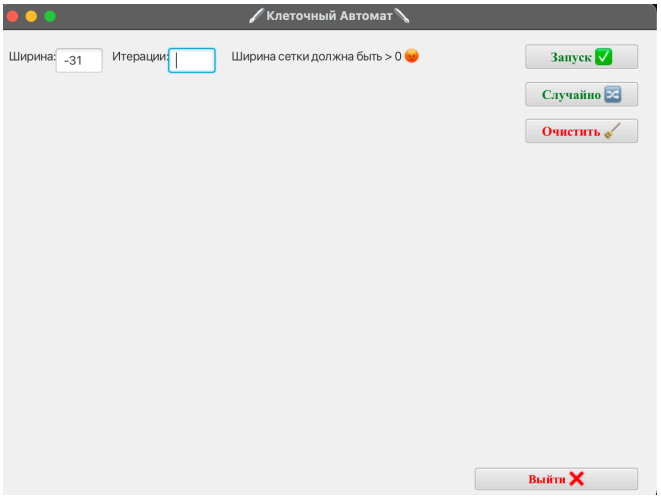

Рис. 4 Попытка записать неверные значения

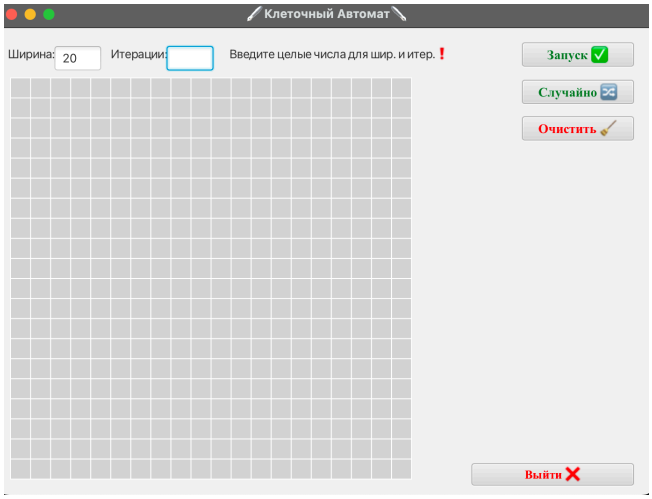

Рис.5 Сетка 20x20

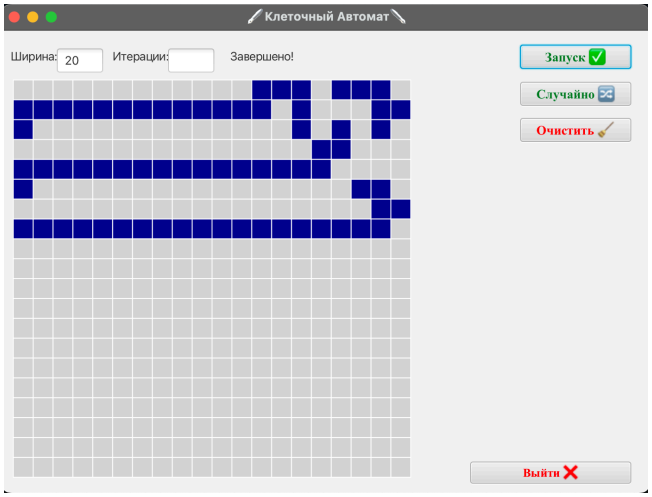

Рис.6 *i+8*(-ая) итерация

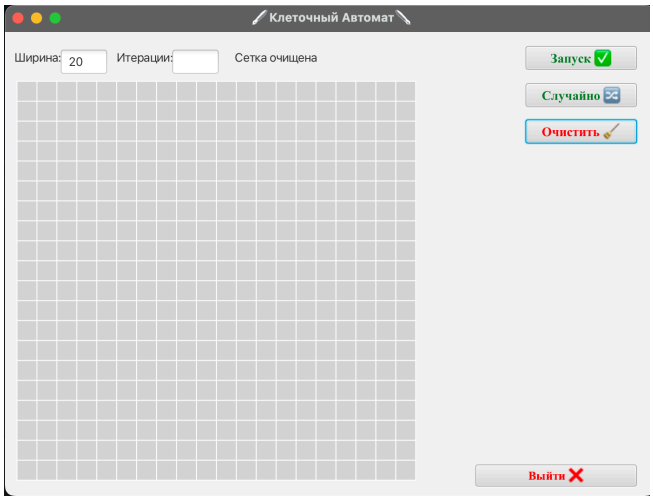

Рис. 7 Очистка сетки

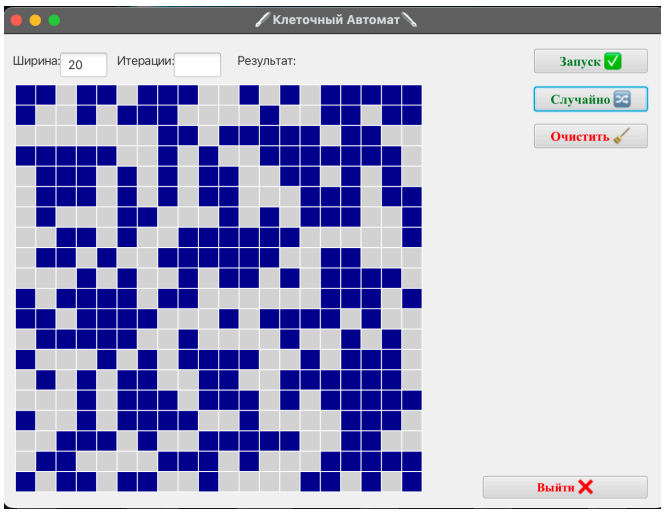

Рис.8 Случайно сгенерированная сетка

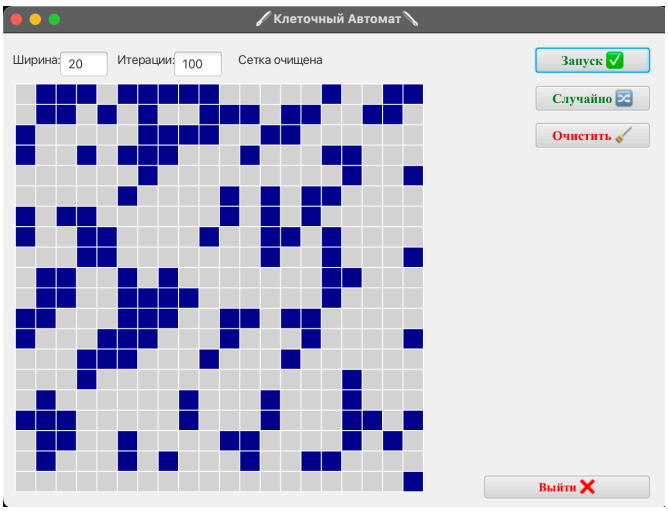

Рис. 9 Запуск 100 поочередных итераций

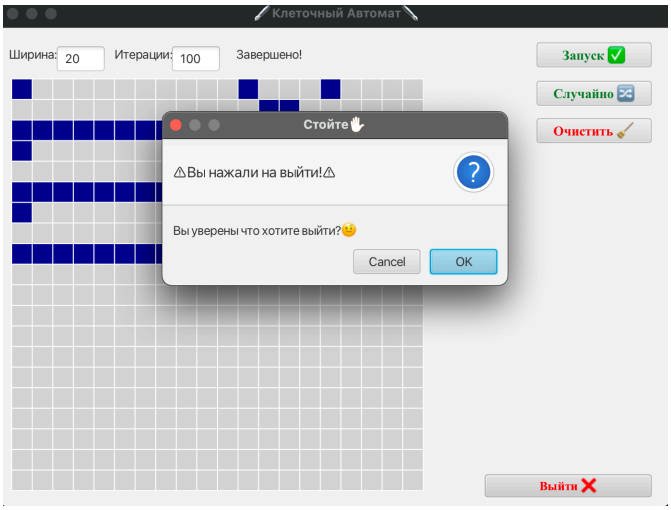

Рис. 10 Нажатие кнопки «Выйти»

4 Анализ работы клеточного автомата

Проанализировав работу автомата с единичными граничными условиями и с указанным ранее вектором булевой функции, можно заметить, что он относится к **квази-стабилизирующемуся** классу. Этот класс автоматов имеет особенность перехода к статическому или устойчивому состоянию после определенного количества итераций. А его тип можно отнести к **изменяемому гомогенному**.

Выбор класса стабилизирующихся автоматов (1-ый класс), объясняется тем, что по началу можно заметить структуру отличающуюся от последующих (сначала виден треугольник, который переходит в структуру «кусочков» которая затем, зацикливается с похожими структурами), гоммогенным, является так как для всех клеток действует одновременно действуют правила, изменяемый так как структура изменчива.

На приведенных рисунках 11-17 сетка размером 100 x 100 где на первых итерациях виден треугольник который в последствии превращается в повторяющиеся структуры идущие вниз по сетке. На рисунке 18 показан график сходимости. На графике видно, резкое уменьшение клеток так, как наш треугольник уменьшается затем периодические колебания показывают повторяющиеся структуры. Таким образов автомат **в квази-стабильном состоянии** (не сходится).

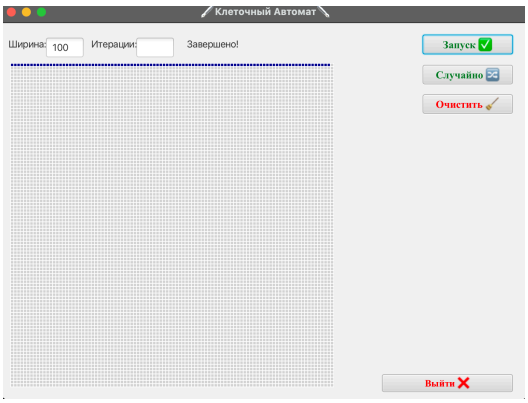

Рис. 11 Первая итерация

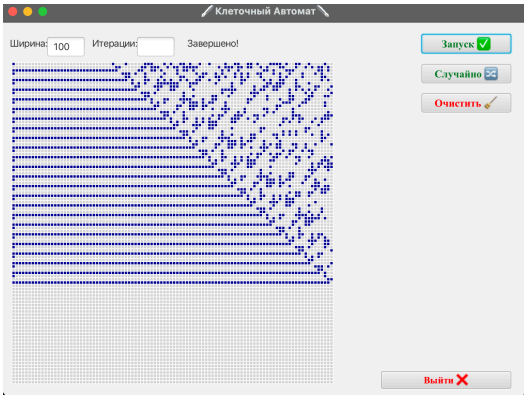

Рис. 13 Приблизительно 38 итераций, проявляется треугольник

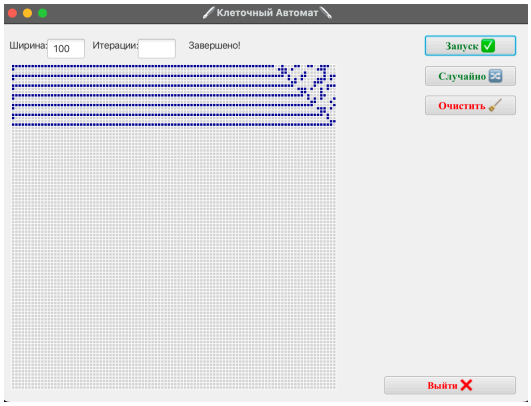

Рис. 12 *i + 15* итераций

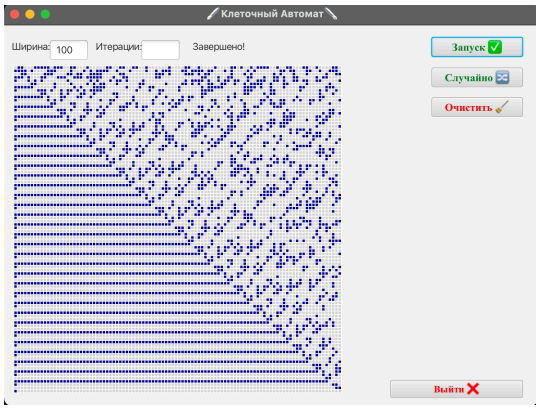

Рис. 14 Отчетливо виден треугольник, за гипотенузой, видны повторяющиеся структуры

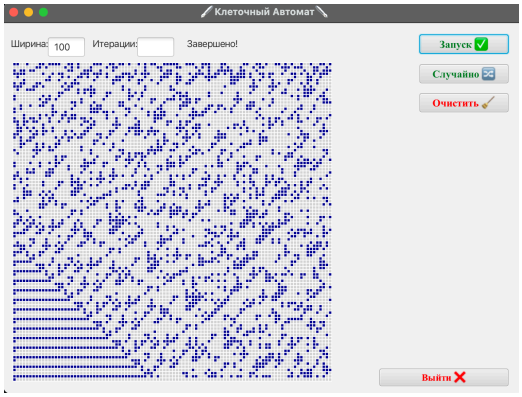

Рис. 15 Треугольник убывает, структуры идут вниз по автомату

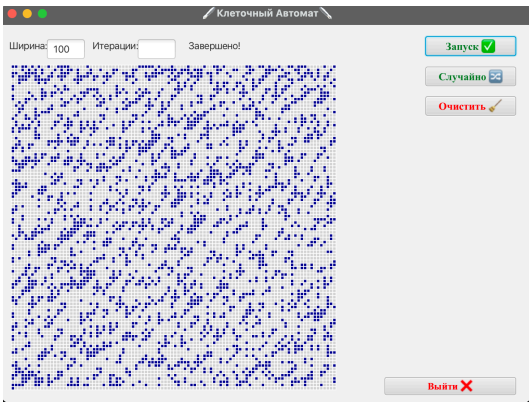

Рис. 16 Структуры повторяются

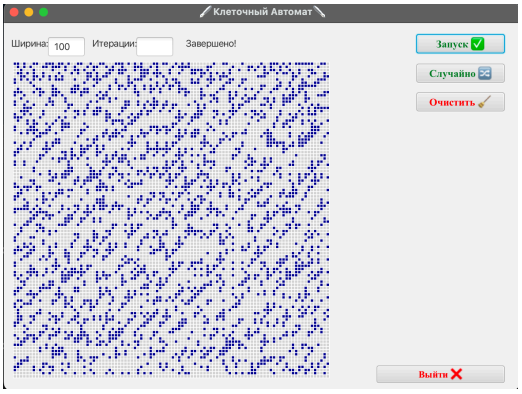

Рис. 17 Структуры повторяются

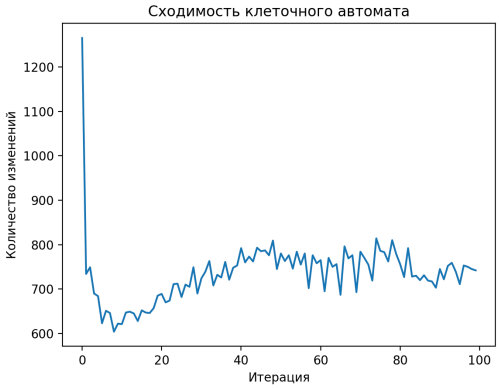

Рис.18 График сходимости

Заключение

В результате работы был реализован и визуализирован двумерный клеточный автомат, на котором можно:

- Задавать стартовое состояние
- Задавать размер сетки
- Задавать количество итераций
- Случайно генерировать стартовое состояние
- Очищать сетку

Код программы был написан на языке программирования Java 22.0.2 (OpenJDK), в интегрируемой среде разработки IntelliJ IDEA (CE). Для разработки UI, использовалась структура проекта Maven и JavaFX.

Достоинства:

- Кроссплатформенность.
- Визуализация работы.
- Контроль пользовательского ввода.
- Адаптация к размеру окон экрана.

Недостатки:

- Отсутствие JUnit - тестов, для полной проверки.
- При сетке размеров > 500, из-за большого количества вычислений, программа работает медленнее (в том числе, так как работает garbage collector).

Масштабируемость:

- В реализованную программу можно добавить: Выбор граничных условий.
- Возможна реализация окрестности Мура.

Автомат имеет **единичные граничные** условия, принадлежит к типу - **гомогенный и изменяемый**, класс - **квази-стабилизирующийся** автомат, **не сходимый**.

Список литературы

[1] Ф.А.Новиков. Дискретная математика для программистов. СПб: Питер Пресс, 2009г. 364с.

[2] Эриксон Дж. Алгоритмы. - 2023. -313 с.

[3] Стивен Вольфрам - Наука нового типа. - Wolfram Media, 2002, - 1197 с.