МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение
высшего образования «Санкт-Петербургский политехнический
университет Петра Великого»

Институт компьютерных наук и кибербезопасности

Высшая школа технологий искусственного интеллекта

Направление: 02.03.01 «Математика и компьютерные науки»

Отчет по лабораторной работе №1
«Реализация двумерного клеточного автомата»
по дисциплине «Теория алгоритмов»
Вариант 25

Студент,
группы 5130201/20102 _____ Тищенко А. А.

Преподаватель _____ Востров А. В.

«_____»_____ 2024г.

Санкт-Петербург, 2024

Содержание

Введение

Данная лабораторная работа по дисциплине «Теория алгоритмов» заключается в разработке консольного приложения на языке C++, которое позволяет создавать и запускать двумерные клеточные автоматы с различными настройками и функцией переходов, заданной вариантом лабораторной работы. Также в этом отчёте к лабораторной работе проводится анализ заданного клеточного автомата.

1 Постановка задачи

Лабораторная работа заключалась в следующем:

- Реализовать двумерный клеточный автомат с окрестностью фон Неймана в соответствии с полученным номером (№109063350), который задаёт функцию перехода автомата.
- Граничные условия: тороидальные, нулевые, единичные.
- Обеспечить возможность пользователю задавать ширину поля и количество итераций.
- Учесть возможность ввода различных начальных условий (как вручную, так и случайным образом) по выбору пользователя.
- Реализовать автомат в консольном или графическом интерфейсе.
- Проанализировать поведение клеточного автомата, выявить паттерны, оценить "сходимость"и другие характеристики в отчете.

2 Математическое описание

2.1 Клеточные автоматы

Клеточные автоматы [1] – это регулярная структура динамических объектов (клеток), функционирующих синхронно. Клеточные автоматы моделируют процессы, разворачивающиеся в дискретном пространстве и в дискретном времени. Клеточный автомат имеет состояние, определяемое как набор (вектор или матрица) состояний компонентных автоматов. Каждый автомат функционирует как автомат Мура, т.е. это автомат без выхода, выходом является его состояние. Вход на каждом шаге клеточного автомата – состояния его соседей. На следующем шаге (в следующем такте) новое состояние каждого автомата определяется как функция его текущего состояния и текущих состояний его соседей. Обычно правила изменения состояний всех автоматов (кроме крайних) идентичны (однородность системы), множество состояний каждого автомата – конечно. Изменение состояния системы во всех клетках происходит синхронно: состояния клеток меняются одновременно, в каждом такте.

2.2 Двумерные клеточные автоматы

Двумерные клеточные автоматы являются расширением одномерных клеточных автоматов, описанных ранее. В одномерных автоматах состояние каждой клетки определяется её собственным состоянием и состояниями соседних клеток на линии. В двумерных клеточных автоматах клетки расположены на двумерной сетке, и их состояние зависит от собственного состояния и состояний соседних клеток в двух измерениях.

Основные элементы клеточного автомата:

1. **Сетка клеток:** Двумерный массив клеток $C(i, j)$, где $i \in \{0, 1, \ldots, N-1\}$ и $j \in \{0, 1, \ldots, M-1\}$ обозначают индексы клетки в сетке, N и M — размеры сетки.

2. **Состояния:** Каждая клетка может находиться в одном из $k \in \mathbb{N}$ возможных состояний: $S = \{s_0, s_1, \ldots, s_{k-1}\}$. В данной лабораторной работе рассматриваются автоматы с двумя возможными состояниями «0» («мёртвые» клетки) и «1» («живые» клетки).

3. **Тип окрестности:** Для клетки $C(i, j)$ задаётся множество соседей $\mathcal{N}(i, j)$. Окрестность фон Неймана (см. Рис. 1) задаётся следующим образом (включает 4 соседа и саму клетку):

$$\mathcal{N}_N(i, j) = (C(i, j), C(i-1, j), C(i+1, j), C(i, j-1), C(i, j+1)).$$

4. **Функция перехода:** Функция f определяет состояние клетки $C(i, j)$ на следующем шаге:

$$C'(i, j) = f\left(\mathcal{N}(i, j)\right),$$

где $C'(i, j)$ — состояние клетки после обновления.

5. **Граничные условия:** Определяют, как будет выглядеть окрестность клеток на границах сетки. В этой работе рассматриваются несколько вариантов граничных условий:

   - *Нулевые граничные условия:* За пределами сетки клетки считаются находящимися в нулевом состоянии.

$$S(i,j) = \begin{cases} 0, & \text{если } i < 0 \text{ или } i \geq N \text{ или } j < 0 \text{ или } j \geq M, \\ C(i,j), & \text{иначе.} \end{cases}$$

   - *Единичные граничные условия:* За пределами сетки клетки считаются находящимися в единичном состоянии.

$$S(i,j) = \begin{cases} 1, & \text{если } i < 0 \text{ или } i \geq N \text{ или } j < 0 \text{ или } j \geq M, \\ C(i,j), & \text{иначе.} \end{cases}$$

   - *Тороидальные граничные условия:* Сетка рассматривается как тор, то есть клетки на одной границе соединены с противоположной границей.

$$S(i,j) = \begin{cases} C((i+N) \bmod N, (j+M) \bmod M), & \text{если } i < 0 \text{ или } i \geq N \\ & \text{или } j < 0 \text{ или } j \geq M, \\ C(i,j), & \text{иначе.} \end{cases}$$

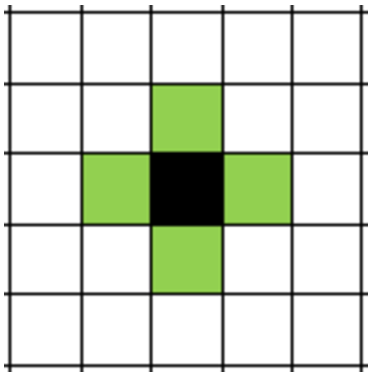

Рис. 1. Двумерная окрестность фон Неймана.

2.3 Функция перехода

Рассматриваемому варианту лабораторной работы соответствует №109063350. В двоичном виде, после добавления ведущих нулей до 32 разрядов, это число выглядит следующим образом:

$$00000110100000000010110010110110_2$$

Битовое представление этого числа в обратном порядке задаёт вектор значений для булевой функции переходов от пяти переменных ($2^5 = 32$). Каждая переменная соответствует состоянию клетки из окресности фон Неймана в определённом порядке, который указан на Рис. 2. В таблице 1 представлена таблица истинности для функции переходов.

Таблица 1. Таблица истинности для функции переходов.

s_0	s_1	s_2	s_3	s_4	$f(s_0, s_1, s_2, s_3, s_4)$
0	0	0	0	0	0
0	0	0	0	1	1
0	0	0	1	0	1
0	0	0	1	1	0
0	0	1	0	0	1
0	0	1	0	1	1
0	0	1	1	0	0
0	0	1	1	1	1
0	1	0	0	0	0
0	1	0	0	1	0
0	1	0	1	0	1
0	1	0	1	1	1
0	1	1	0	0	0
0	1	1	0	1	1
0	1	1	1	0	0
0	1	1	1	1	0
1	0	0	0	0	0
1	0	0	0	1	0
1	0	0	1	0	0
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	1
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

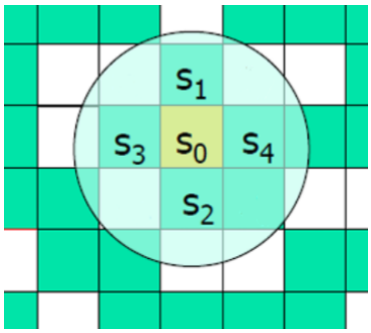

Рис. 2. Окрестность фон Неймана с метками для соседей.

2.4 Паттерны, сходимость и классификация клеточных автоматов

Паттерны – это устойчивые структуры, которые формируются в процессе эволюции клеточного автомата. В зависимости от правил, эти структуры могут быть статичными (не изменяются с течением времени), циклическими (повторяются через несколько итераций), или распространяющимися (разрастаются в пространстве).

Сходимость – это поведение клеточного автомата, при котором его состояние стабилизируется спустя некоторое количество итераций. Это может быть достижение статического состояния, циклического паттерна или полное угасание активности (все клетки становятся "мёртвыми").

Стивен Вольфрам в своей книге A New Kind of Science [2] предложил 4 класса, на которые все клеточные автоматы могут быть разделены в зависимости от типа их эволюции:

- Класс 1: Результатом эволюции начальных условий является быстрый переход к гомогенной стабильности. Любые негомогенные конструкции быстро исчезают.
- Класс 2: Результатом эволюции начальных условий является быстрый переход в неизменяемое негомогенное состояние либо возникновение циклической последовательности. Большинство структур начальных условий быстро исчезает, но некоторые остаются. Локальные изменения в начальных условиях оказывают локальный характер на дальнейший ход эволюции системы.
- Класс 3: Результатом эволюции почти всех начальных условий являются псевдо-случайные, хаотические последовательности. Любые стабильные структуры, которые возникают почти сразу же уничтожаются окружающим их шумом. Локальные изменения в начальных условиях оказывают неопределяемое влияние на ход эволюции системы.
- Класс 4: Результатом эволюции являются структуры, которые взаимодействуют сложным образом с формированием локальных, устойчивых структур. В результате эволюции могут получаться некоторые последовательности Класса 2, описанного выше. Локальные изменения в начальных условиях оказывают неопределяемое влияние на ход эволюции системы. Некоторые клеточные автоматы этого класса обладают свойством универсальности по Тьюрингу.

3 Особенности реализации

3.1 Класс CellularAutomaton

В листинге 1 представлен код объявления класса `CellularAutomaton`.

Класс содержит следующие атрибуты:

- `static const unsigned int m_functionValues` – хранит число, соответствующее номеру варианта лабораторной работы, его битовое представление соответствует функции переходов двумерного клеточного автомата;
- `int m_fieldWidth` – ширина поля клеточного автомата;
- `int m_fieldHeight` – высота поля клеточного автомата;
- `std::vector<std::vector<int> > m_field` – двумерный вектор, представляющий текущее состояние клеточного автомата;
- `std::vector<std::vector<int> > m_fieldNextState` – двумерный вектор для хранения следующего состояния клеточного автомата;
- `BoundaryCondition m_boundaryCondition` – граничные условия, задаются значением из перечисления `BoundaryCondition`.

Перечисление `BoundaryCondition` имеет три возможных значения:

- `BOUNDARY_ONES` – единичные граничные условия;
- `BOUNDARY_ZEROS` – нулевые граничные условия;
- `BOUNDARY_TOROIDAL` – тороидальные граничные условия.

Методы класса описываются в последующих разделах.

Листинг 1. Код объявления класса CellularAutomaton.

```
1  enum BoundaryCondition {
2      BOUNDARY_ONES,
3      BOUNDARY_ZEROS,
4      BOUNDARY_TOROIDAL
5  };
6
7  class CellularAutomaton
8  {
9      static const unsigned int m_functionValues = 25 * 11 * 2003 * 18 * 11;
10
11     int m_fieldWidth, m_fieldHeight;
12     std::vector<std::vector<int>> m_field;
13     std::vector<std::vector<int>> m_fieldNextState;
14     BoundaryCondition m_boundaryCondition;
15
16     void initializeRandom();
17     void initializeManual();
18
```

```
19      int getCellState(int x, int y) const;
20      int getNeighborhoodIndex(int x, int y) const;
21  public:
22      CellularAutomaton(int width, int height, bool fillWithRandom, BoundaryCondition
          boundaryCondition);
23
24      void update();
25      void displayField() const;
26  };
```

3.1.1 Конструктор CellularAutomaton

Конструктор класса CellularAutomaton, код которого представлен в листиге 2, принимает следующие параметры:

- `int width` – ширина поля клеточного автомата;
- `int height` – высота поля клеточного автомата;
- `bool fillWithRandom` – флаг, определяющий, инициализировать ли поле случайными значениями;
- `BoundaryCondition boundaryCondition` – условие на границе поля.

Конструктор создает объект `CellularAutomaton` с заданными размерами и граничными условиями. Если `fillWithRandom` равно `true`, то поле инициализируется случайным образом с помощью метода `initializeRandom`, иначе пользователь может вручную ввести начальное состояние с помощью метода `initializeManual`.

Листинг 2. Код конструктора CellularAutomaton.

```
1  CellularAutomaton::CellularAutomaton(int width, int height, bool fillWithRandom,
       BoundaryCondition boundaryCondition)
2      : m_fieldWidth(width), m_fieldHeight(height), m_boundaryCondition(boundaryCondition)
3  {
4      m_field.resize(m_fieldHeight, std::vector<int>(m_fieldWidth, 0));
5      m_fieldNextState.resize(m_fieldHeight, std::vector<int>(m_fieldWidth, 0));
6
7      if (fillWithRandom) initializeRandom();
8      else initializeManual();
9  }
```

3.1.2 Метод initializeRandom

В листинге 3 представлен код метода `initializeRandom`. Метод явно ничего не принимает и не возвращает, а работает только с атрибутами класса `CellularAutomaton`. В результате его работы атрибут `m_field` заполняется нулями и единицами случайным образом, для этого используется равномерное распределение.

Листинг 3. Код метода initializeRandom.

```
1  void CellularAutomaton::initializeRandom()
```

```
2  {
3      std::random_device rd;
4      std::mt19937 gen(rd());
5      std::uniform_int_distribution<> dis(0, 1);
6
7      for (int y = 0; y < m_fieldHeight; ++y)
8      {
9          for (int x = 0; x < m_fieldWidth; ++x)
10         {
11             m_field[y][x] = dis(gen);
12         }
13     }
14 }
```

3.1.3 Метод initializeManual

В листинге 4 представлен код метода `initializeManual`. Метод не принимает аргументов и ничего не возвращает. Он позволяет пользователю вручную ввести начальное состояние поля клеточного автомата. Для каждой клетки запрашивается значение 0 или 1, которые сохраняются в атрибуте `m_field`.

Листинг 4. Код метода initializeManual.

```
1  void CellularAutomaton::initializeManual()
2  {
3      std::cout << "Введите␣начальное␣состояние␣поля␣(" << m_fieldWidth << "␣x␣" <<
         m_fieldHeight << ").\n";
4      std::cout << "Введите␣0␣или␣1␣для␣каждой␣клетки.\n";
5
6      for (int y = 0; y < m_fieldHeight; ++y)
7      {
8          for (int x = 0; x < m_fieldWidth; ++x)
9          {
10             std::cout << "Клетка␣(" << x << ",␣" << y << "):␣";
11             int cellValue = inputNumber(0, 1);
12
13             m_field[y][x] = cellValue;
14         }
15     }
16 }
```

3.1.4 Метод getCellState

В листинге 5 представлен код метода `getCellState`. Метод принимает на вход координаты клетки `int x` и `int y`. Возвращает `int` – состояние клетки по указанным координатам. Если координаты выходят за пределы поля, то поведение определяется граничными условиями, заданными атрибутом `m_boundaryCondition`.

Листинг 5. Код метода getCellState.

```
1  int CellularAutomaton::getCellState(int x, int y) const
2  {
3      if (x < 0 || x >= m_fieldWidth || y < 0 || y >= m_fieldHeight)
```

```
4     {
5         switch (m_boundaryCondition)
6         {
7         case BOUNDARY_ONES:
8             return 1;
9         case BOUNDARY_ZEROS:
10            return 0;
11        case BOUNDARY_TOROIDAL:
12            x = (x + m_fieldWidth) % m_fieldWidth;
13            y = (y + m_fieldHeight) % m_fieldHeight;
14            return m_field[y][x];
15        default:
16            return 0;
17        }
18    }
19    return m_field[y][x];
20 }
```

3.1.5 Метод getNeighborhoodIndex

В листинге 6 представлен код метода `getNeighborhoodIndex`. Метод принимает на вход координаты клетки `int x` и `int y`. Возвращает `int` – индекс конфигурации окрестности фон Неймана клетки. Индекс вычисляется на основе состояний центральной клетки и её четырёх соседей (сверху, снизу, слева и справа), где каждый сосед соответствует одному биту в индексе. Этот индекс используется для определения следующего состояния клетки по функции переходов в методе `update`.

Листинг 6. Код метода getNeighborhoodIndex.

```
1  int CellularAutomaton::getNeighborhoodIndex(int x, int y) const
2  {
3      int s0 = getCellState(x, y);
4      int s1 = getCellState(x, y − 1);
5      int s2 = getCellState(x, y + 1);
6      int s3 = getCellState(x − 1, y);
7      int s4 = getCellState(x + 1, y);
8
9      int index = (s0 << 4) | (s1 << 3) | (s2 << 2) | (s3 << 1) | s4;
10
11     return index;
12 }
```

3.1.6 Метод update

В листинге 7 представлен код метода `update`. Метод не принимает аргументов и ничего не возвращает. Он обновляет состояние клеточного автомата на следующий временной шаг. Для каждой клетки вычисляется новое состояние на основе текущего состояния и функции переходов, значения которой хранятся в битах числа `m_functionValues`. После вычисления новое состояние сохраняется в `m_fieldNextState`, а затем происходит обмен содержимого `m_field` и `m_fieldNextState`.

Листинг 7. Код метода update.

```
void CellularAutomaton::update()
{
    for (int y = 0; y < m_fieldHeight; ++y)
    {
        for (int x = 0; x < m_fieldWidth; ++x)
        {
            int neighborhood = getNeighborhoodIndex(x, y);
            m_fieldNextState[y][x] = (m_functionValues >> neighborhood) & 1;
        }
    }

    m_field.swap(m_fieldNextState);
}
```

3.1.7 Метод displayField

В листинге 8 представлен код метода `displayField`. Метод не принимает аргументов и ничего не возвращает. Он выводит текущее состояние поля клеточного автомата в консоль, отображая каждую клетку как '0' или '1'.

Листинг 8. Код метода displayField.

```
void CellularAutomaton::displayField() const
{
    for (const auto& row : m_field)
    {
        for (const auto& cell : row)
        {
            std::cout << (cell ? '1' : '0') << ' ';
        }
        std::cout << '\n';
    }
}
```

3.2 Функция main

В функции main, код которой представлен в листинге 9, содержится бесконечный цикл, который обрабатывает пользовательский ввод. На каждой итерации цикла, пользователю предлагается выбрать конфигурацию конечного двумерного автомата и количество итераций. Затем внутри функции создаётся объект класса `CellularAutomaton`, с заданными пользователем параметрами, и у него вызывается метод `update` согласно указанному количеству итераций. Функция `main` возвращает целое число, которое является кодом завершения программы - 0, если программа выполнилась успешно, и ненулевое значение, если произошла какая-либо ошибка.

Листинг 9. Код функции main.

```
int main()
{
    setlocale(LC_ALL, "Russian");
```

```
4
5       while (true) {
6           clear();
7
8           cout << "Выберите␣граничные␣условия:\n"
9               "Единичные␣(0)\n"
10              "Нулевые␣(1)\n"
11              "Тороидальные␣(2)\n"
12              "Завершить␣работу␣(3)\n\n";
13          int actionId = inputNumber(0, 3);
14
15          clear();
16
17          if (actionId == 3) {
18              cout << "Выйти␣из␣программы?␣(yes/no)\n";
19              if (userApprove()) return 0;
20              continue;
21          }
22
23          BoundaryCondition boundaryCondition = static_cast<BoundaryCondition>(actionId);
24
25          cout << "Укажите␣ширину␣поля␣(min␣1):␣";
26          int fieldWidth = inputNumber(1);
27
28          cout << "Укажите␣высоту␣поля␣(min␣1):␣";
29          int fieldHeight = inputNumber(1);
30
31          cout << "Укажите␣количество␣итераций␣(min␣1):␣";
32          int iterationsCount = inputNumber(1);
33
34          cout << "Заполнить␣поле␣случайными␣значениями?␣(yes/no)\n";
35          bool fillWithRandom = userApprove();
36
37          CellularAutomaton ca(fieldWidth, fieldHeight, fillWithRandom, boundaryCondition);
38          clear();
39
40          cout << "\Итерацияn␣0:\n";
41          ca.displayField();
42
43          for (int i = 0; i < iterationsCount; ++i)
44          {
45              cout << "\Итерацияn␣" << i + 1 << ":\n";
46              ca.update();
47              ca.displayField();
48          }
49
50          cout << "\Нажмитеn␣на␣enter,␣чтобы␣продолжить...";
51          waitForEnter();
52      }
53  }
```

3.3 Пользовательский ввод

Одним из требований к лабораторным работам являлась защита от некорректного пользовательского ввода. Для реализации такой защиты и большей читаемости кода все функции связанные с пользовательским вводом были вынесены в отдельный файл. Все функции проверяют данные вводимые пользователем и, если что-то не так, печатают информацию о неверном выводе.

- `int inputNumber(int minVal, int maxVal)` - принимает два числа, которые указывают диапозон возможных для ввода значений. Возвращает введённое пользователем число.
- `char* inputString(int maxLen)` - принимает число, максимальную длину строки, возвращает указатель на строку введённую пользователем. Не позволяет вводить строки больше указанной длины.
- `bool userApprove()` - возвращает true, если пользователь ввёл «yes» или «y» и false, если пользователь ввёл «no» или «n».
- `void waitForEnter()` - останавливает выполнение программы пока пользователь не нажмёт на клавишу «Enter».

4 Результаты работы программы

В данном разделе представлены скриншоты с примерами ввода-вывода, демонстрирующие работу программы и её основной функционал.

На Рис. 3 показано основное стартовое меню программы. Пользователю предоставляется возможность выбрать граничные условия или завершнить работу с приложением. В верхней строке выводится вектор значений функции переходов, соответствующий варианту лабораторной работы.

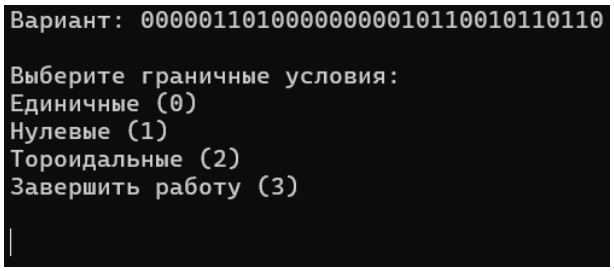

Рис. 3. Начальное меню приложения.

На Рис. 4 демонстрируется процесс задания ширины и высоты поля клеточного автомата, а также количества итераций. Программа также предлагает выбрать способ задания начальной конфигурации. В случае утвердительного ответа, поле заполняется нулями и единицами случайным образом. Если ответ отрицательный, то программа предлагает пользователю заполнить значения для всех полей вручную, этот процесс демонстрируется на Рис. 5.

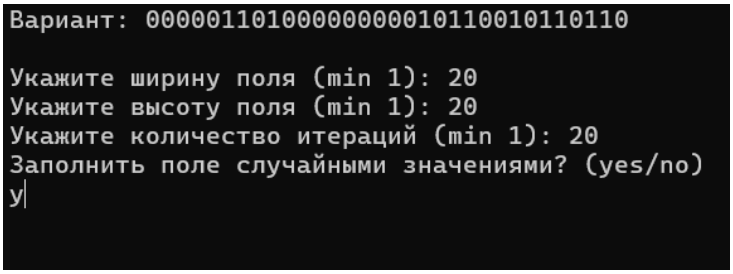

Рис. 4. Заполнение параметров поля автомата и количества итераций.

После заполнения значений клеток поля в консоль выводится последовательность состояний поля по итерациям (см. Рис. 6).

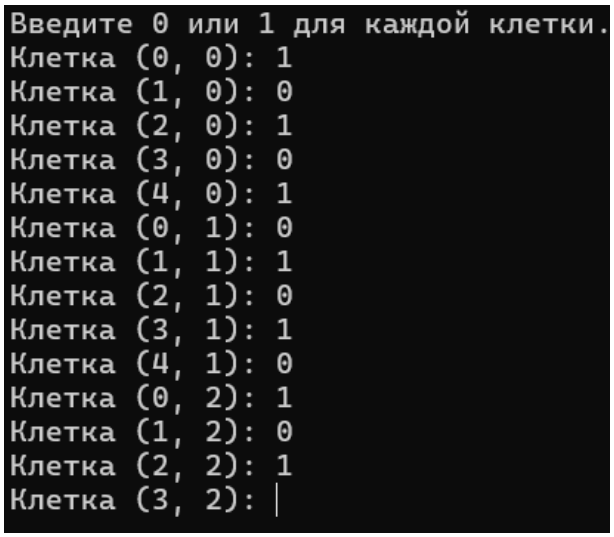

Рис. 5. Демонстрация процесса заполнения значений клеток поля вручную.

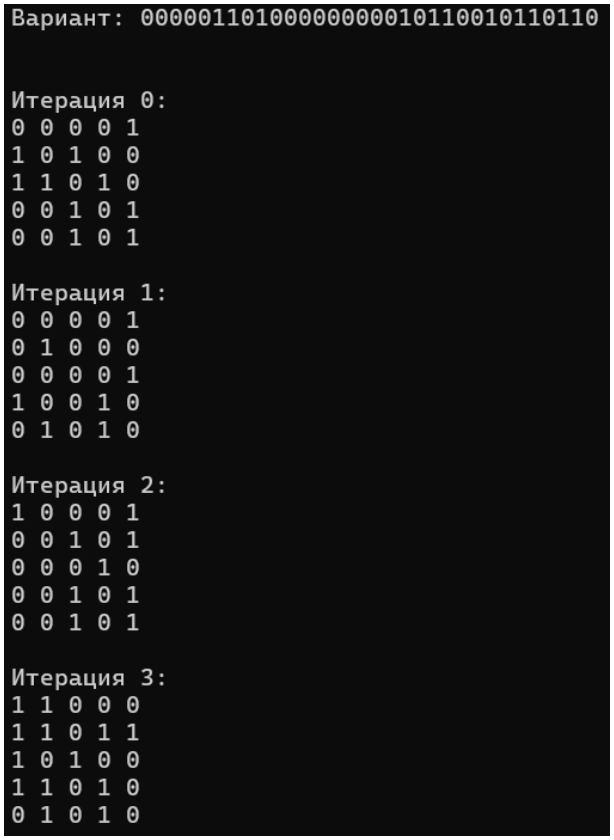

Рис. 6. Вывод состояний автомата по итерациям.

5 Анализ клеточного автомата

5.1 Паттерны

В ходе работы были исследованы паттерны клеточного автомата на поле размером 5×5 клеток в течение 25 итераций с различными начальными конфигурациями и граничными условиями. Краткие результаты для каждой комбинации граничных условий и начальных конфигураций представлены в таблице 2. Начальные конфигурации представлены на Рис. 7-11.

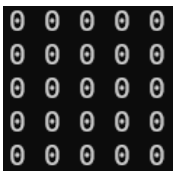

Рис. 7. Начальная конфигурация «Пустое поле».

Рис. 8. Начальная конфигурация «Горизонтальная палочка».

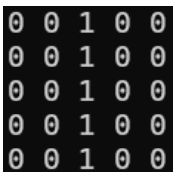

Рис. 9. Начальная конфигурация «Вертикальная палочка».

Рис. 10. Начальная конфигурация «Квадратик».

Как видно по таблице 2 в результате анализа удалось обнаружить несколько циклических паттернов, рассмотрим их подробнее.

Рис. 11. Начальная конфигурация «Единичное поле».

Таблица 2. Результаты анализа клеточного автомата при различных начальных конфигурациях и граничных условиях.

Исходная конфигурация	**Граничные условия**	**Результат**
Пустое поле	Единичные	Случайные паттерны
	Нулевые	Пустое поле
	Тороидальные	Пустое поле
Горизонтальная палочка	Единичные	Случайные паттерны
	Нулевые	Движение вверх и нулевое поле
	Тороидальные	Циклическое движение вверх
Вертикальная палочка	Единичные	Случайные паттерны
	Нулевые	Зацикливание
	Тороидальные	Зацикливание
Квадратик	Единичные	Случайные паттерны
	Нулевые	Зацикливание
	Тороидальные	Случайные паттерны
Единичное поле	Единичные	Случайные паттерны
	Нулевые	Зацикливание
	Тороидальные	Превратилось в нулевое поле

При задании начальной конфигурации «Горизонтальная палочка» (см. Рис. 8) и нулевых или тороидальных граничных условий палочка каждый такт времени сдвигается вверх по сетке автомата на одну строку. При нулевых граничных условиях палочка доходит до границы поля и исчезает (см. Рис 12). При тороидальных граничных условиях палочка доходит до границы, затем появляется снизу и снова движется вверх по сетке (см. Рис 13).

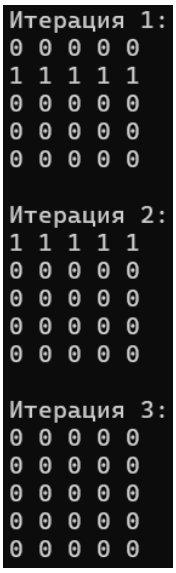

Рис. 12. Работа автомата при нулевых граничных условиях и начальной конфигурации «Горизонтальная палочка».

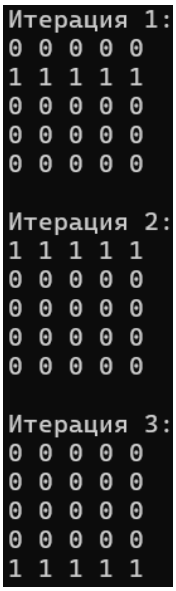

Рис. 13. Работа автомата при тороидальных граничных условиях и начальной конфигурации «Горизонтальная палочка».

При задании начальной конфигурации «Вертикальная палочка» (см. Рис. 14) и нулевых или тороидальных граничных условий палочка сначала раздваивается и расходится в стороны до границ сетки, потом обратно сдвигается на одну клетку к центру и зацикливается (см. Рис. 14).

При задании начальной конфигурации «Квадратик» (см. Рис. 10) или «Единичное поле» (см. Рис. 11) и нулевых граничных условий автомат зацикливается после 18-ой итерации (см. Рис. 15). При этом паттерны перед зацикливанием отличаются для разных начальных конфигураций и выглядят случайными.

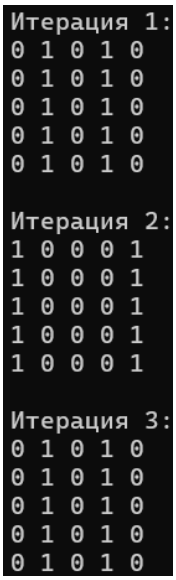

Рис. 14. Работа автомата при нулевых и тороидальных граничных условиях и начальной конфигурации «Вертикальная палочка».

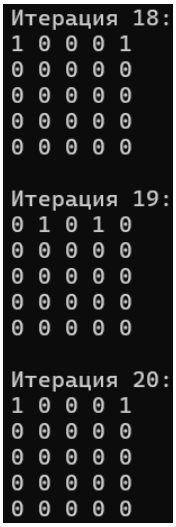

Рис. 15. Зацикливание автомата при нулевых граничных условиях и начальной конфигурации «Квадратик» или «Единичное поле».

Также автомат был несколько раз запущен на том же размере сетки (5 x 5), нулевых начальных условиях и случайных начальных конфигурациях для большего числа итераций ($n = 300$). Таким образом удалось обнаружить ещё несколько циклических паттернов, к которым сходится рассматриваемый автомат. На Рис. 16 представлен пример достаточно короткого циклического паттерна, а на Рис. 17 представлена конфигурация, которая повторяется снова каждые 8 итераций.

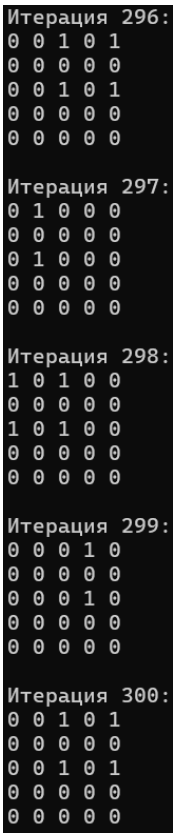

Рис. 16. Один из циклических паттернов рассматриваемого автомата.

Рис. 17. Пример конфигурации, повторяющейся каждые 8 итераций.

5.2 Сходимость

По таблице 2 видно, что автомат ведёт себя по разному, в зависимости от начальной конфигурации и граничных условий. В некоторых случаях автомат сходился к пустому полю, все клетки которого мертвы. В некоторых случаях он сходится к различным циклическим паттернам. А для нескольких конфигураций не удалось определить сходимость на заданном числе итераций.

5.3 Классификация

Результаты запуска автомата для пяти произвольных начальных конфигураций представлены на графике, который изображён на Рис. 18. На Рис. 19, изображён график среднего значения количества живых клеток для всех пяти экспериментов. По этому графику видно, что количество живых клеток стабилизируется после 400 итераций на уровне 70 клеток. На Рис. 20 изображён график аппроксимирующей

функции количества живых клеток от итераций. По нему также видно, что количество живых клеток сходится к, примерно, 70 клеткам.

Исходя из описания классов и анализа паттернов клеточного автомата, можно сделать вывод, что при различных начальных конфигурациях и граничных условиях, автомат с полем 5 на 5 проявляет свойства 1 (все ячейки становятся мёртвыми) и 2 (переход к циклическим паттернам) классов. Если же увеличить размер поля до 20 клеток и задать нулевые граничные условия, то автомат перестаёт проявлять признаки этих классов и, кажется, что его можно отнести к 3 классу, так как результатом почти всех начальных условий становятся псевдо-случайные хаотические последовательности.

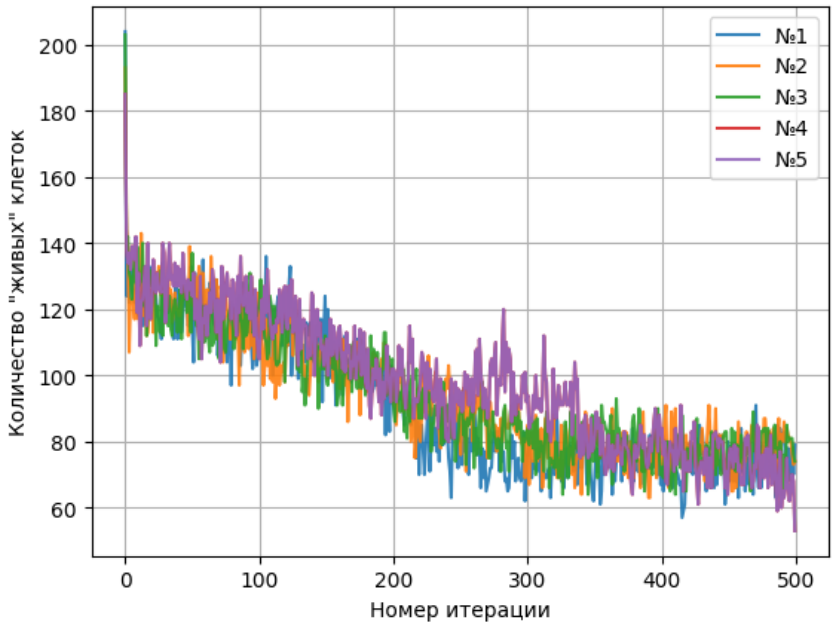

Рис. 18. График количества «живых» клеток относительно номера итерации при нулевых граничных условиях для сетки размером 20 x 20 для пяти произвольных начальных конфигураций.

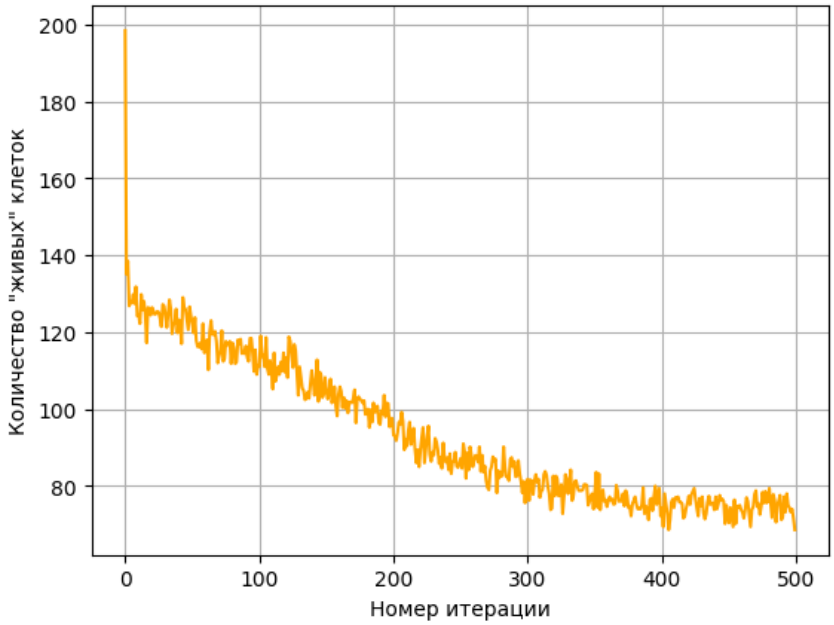

Рис. 19. График среднего количества «живых» клеток относительно номера итерации при нулевых граничных условиях для сетки размером 20 x 20 по всем экспериментам.

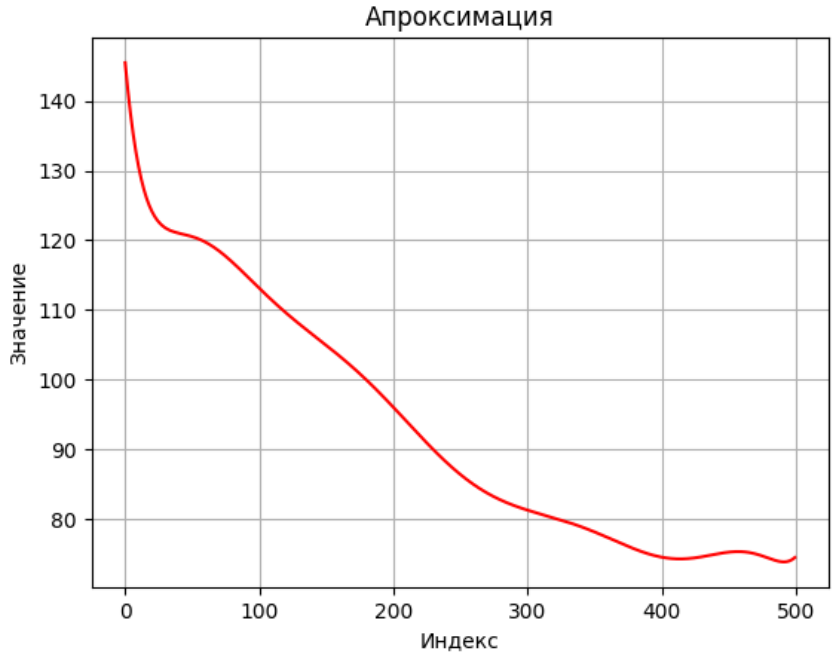

Рис. 20. График полиномиальной аппроксимирующей функции (n=10) количества «живых» клеток относительно номера итерации при нулевых граничных условиях для сетки размером 20 x 20.

Заключение

В ходе выполнения лабораторной работы был реализован двумерный клеточный автомат в консоли с окрестностью фон Неймана в соответствии с заданным номером 109063350. Двоичное представление этого числа, дополненное ведущими нулями, соответствует вектору значений функции переходов рассматриваемого клеточного автомата. Анализ паттернов был проведён для трёх вариантов граничных условий: единичные, нулевые и тороидальные. Было обнаружено как минимум 6 различных циклических паттернов для поля размером 5 на 5. Для поля размером 20 на 20 и нулевых начальных условий были построены графики количества «живых» клеток для различных начальных конфигураций. Также по результатам анализа, было определено, что при размере поля 5 на 5 рассматриваемый автомат проявляет признаки первого и второго классов по классификации Стивена Вольфрама, однако при большем размере поля в 20 на 20 клеток он больше подходит под определение третьего класса.

Из достоинств выполнения лабораторной работы можно выделить структурирование кода за счёт использования ООП. Вся логика работы клеточного автомата вынесена в отдельный класс `CellularAutomaton`, который может быть переиспользован в других программах. Также достоинством является то, что приложение позволяет пользователю выбирать граничные условия.

Во время разработки упор был сделан на упрощении и понятности кода, поэтому в качестве недостатка можно отметить не самую лучшую его эффективность, в особенности в использовании памяти. Поскольку каждая клетка может быть в одном из двух состояний можно было бы использовать битовые структуры для хранения поля, однако в работе для этих целей используются векторы целых чисел. Таким образом для хранения состояния каждой отдельной клетки выделяется по 32 бита, вместо одного необходимого.

Функционал программы достаточно несложно масштабировать. Например, сейчас функция переходов жёстко задаётся в классе `CellularAutomaton`, с помощью незначительных изменений кода можно было бы предоставить пользователю возможность задавать функцию перехода, посредством ввода вектора значений функции или натурального числа, двоичное представление которого соответствовало бы вектору значений функции переходов.

Работа выполнена в среде разработки Visual Studio 2022, стандарт ISO C++ 14, компилятор Microsoft (R) C/C++ версии 19.33.31629.

Список литературы

[1] Востров А. В, «Теория алгоритмов» URL: https://tema.spbstu.ru/algorithm/ (Дата обращения: 01.12.2024).

[2] Wolfram, Stephen «A New Kind of Science». — Wolfram Media (2002) — 1197 с.