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

Теория алгоритмов

Отчёт по лабораторной работе №1
«Реализация и анализ клеточного автомата»
вариант 11

Студент,
группы 5130201/20102 _____ Гаар В.С.

Преподаватель _____ Востров А.В.

«_____»_____ 2024 г.

Санкт-Петербург, 2024

Содержание

Введение

Данный отчёт содержит в себе описание реализации лабораторной работы №1 по дисциплине «Теория алгоритмов», направленной на изучение аспектов работы клеточных автоматов. Содержание лабораторной работы представлено ниже:

1. Реализовать двумерный клеточный автомат с окрестностью фон Неймана в соответствии с полученным номером $№ = 2424840_{10}$.
2. Реализовать торроидальные, нулевые, единичные граничные условия с возможностью выбора пользователем.
3. Пользователь определяет ширину поля и количество итераций.
4. Необходимо учесть возможность ввода различных начальных условий (как вручную, так и случайным образом) по выбору пользователя.
5. Отображение реализовать в консоли.
6. Проанализировать свой клеточный автомат в отчете (его поведение, паттерны, «сходимость» и т.д.).

1 Математическое описание

1.1 Клеточные автоматы

Клеточный автомат представляет собой двусторонне бесконечную ленту, каждая ячейка которой может находиться в некотором состоянии. Множество состояний Q, обозначим состояние ячейки i как $s[i]$. Изначально все ячейки находятся в состоянии $B \in Q$, кроме ячеек с номерами от 1 до n. Ячейка с номером i, где $1 \leq i \leq n$ находится в состоянии x_i, где x – входное слово (будем считать, что $\Sigma \subset Q, B \notin \Sigma$).

Правила работы клеточного автомата такие: задано число d и функция $f : Q^{2d+1} \to Q$. За один шаг все клетки меняют состояние по следующему правилу: новое состояние клетки i равно $f(s[i-d], s[i-d+1], \ldots, s[i+d-1], s[i+d])$. Если клетка с номером 0 переходит в состояние Y, то автомат допускает слово x.

Клеточным автоматом (КА) (англ. *cellular automaton*) A размерности d называется четверка $< Z^d, S, N, \delta >$, где

- S – конечное множество, элементы которого являются состояниями A.
- N – конечное упорядоченное подмножество Z^d, $N = \{n_j | n_j = (x_{1_j}, \ldots, x_{d_j}), j \in \{1 \ldots n\}\}$, называемое **окрестностью** (англ. *neighborhood*) A. В данном определении полагается, что клетка всегда принадлежит своей окрестности.
- $\delta : S_n \to S$ – функция перехода для A. [1]

Окрестность фон Неймана ячейки — совокупность ячеек в сетке (двумерном паркете, трёхмерном Евклидовом пространстве, разбитом на равновеликие кубы), имеющих общую сторону (грань) с данной ячейкой. [2]

Пример двумерной окрестности фон Неймана порядка 1 представлен на рисунке № 1.

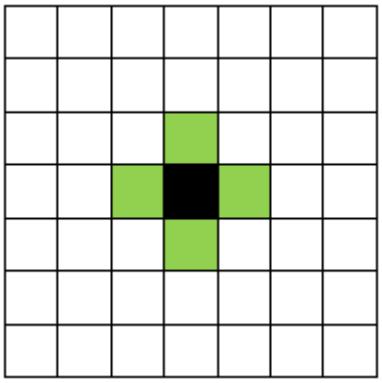

Рис. 1. Двумерная окрестность фон Неймана порядка 1

Двумерный клеточный автомат можно определить как множество конечных автоматов на плоскости, помеченных целочисленными координатами (i, j), каждый из которых может находиться в одном из состояний

$$\sigma_{i,j} : \sigma_{i,j} \in \Sigma \equiv \{0, 1, 2 \ldots k-1, k\}, \quad i, j \in \mathbb{N} \cup \{0\}.$$

Изменение состояний автоматов происходит согласно правилу перехода

$$\sigma_{i,j}(t+1) = \phi(\sigma_{k,l}(t) | (k,l) \in \mathcal{N}(i,j)), \quad i, j \in \mathbb{N} \cup \{0\},$$

где $\mathcal{N}(i,j)$ – некоторая окрестность точки (i,j). К примеру, окрестность фон Неймана определяется как

$$\mathcal{N}_N^1(i,j) = \{(k,l) : \ |i-k| + |j-l| \leq 1\}, \quad i,j \in \mathbb{N} \cup \{0\},$$

а окрестность Мура

$$\mathcal{N}_M^1(i,j) = \{(k,l) : \ |i-k| \leq 1, |j-l| \leq 1\}, \quad i,j \in \mathbb{N} \cup \{0\}.$$

Число всех возможных правил перехода определяется числом состояний σ и количеством соседей n и составляет

$$N_r = \sigma^{\sigma^n}[3]$$

Граничные условия Определяют, как будет выглядеть окрестность клеток на границах сетки. Основные типы:

- *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.

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

где N и M — размеры сетки.

1.2 Классификация клеточных автоматов

1.2.1 Классификация по типам поведения

Стивен Вольфрам в своей книге A New Kind of Science предложил 4 класса, на которые все клеточные автоматы могут быть разделены в зависимости от типа их эволюции. Классификация Вольфрама была первой попыткой классифицировать сами правила, а не типы поведения правил по отдельности. В порядке возрастания сложности классы выглядят следующим образом:

- 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, что доказано для Правила 110 и игры «Жизнь».

1.3 Тоталистичные клеточные автоматы

Существует специальный класс клеточных автоматов, называемых тоталистичными. На каждом шаге эволюции клеточного автомата значение клетки равно какому-либо целому числу (обычно выбираемого из конечного множества), а новое состояние клетки определяется суммой значений клеток-соседей и, возможно, предыдущим состоянием клетки. Если состояние клетки на новом шаге зависит от её предыдущего состояния, то такой клеточный автомат называется внешним тоталистичным. Игра Жизнь является примером внешнего тоталистического клеточного автомата с набором значений ячеек $0, 1$.

Термин тоталистичный происходит от английского *totalistic*. В свою очередь *total* может быть переведено как сумма, что и отражено в принципе действия этого типа автоматов, когда новое значение клетки зависит от суммы значений других клеток.

1.4 Представление функции с данным №

Номер функции был вычислен по следующему выражению: № = номер варианта * 11 * год рождения * день * месяц, где номер варианта = 11. Получилось значение 2424840_{10}, которое далее было представлено в двоичном виде и дополнено до 32 символов незначащими ведущими нулями. Далее использовался получившийся вектор-столбец

$$f(s_0, s_1, s_2, s_3, s_4) = (00000000001001010000000000001000)_2,$$

где переменные s_0, s_1, s_2, s_3, s_4 соответствуют следующим клеткам в окрестности фон Неймана, представленным на рисунке № 2:

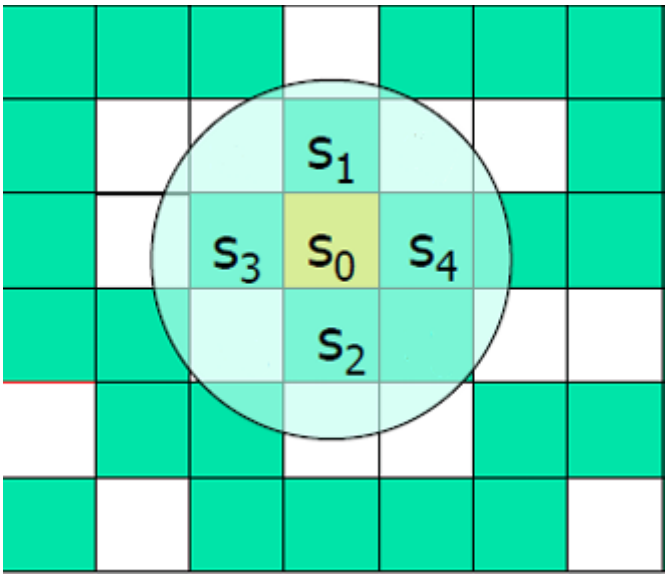

Рис. 2. Нумерация клеток в окрестности фон Неймана

2 Особенности реализации

Сам двумерный клеточный автомат реализован в виде класса `CellularAutomaton` со следующими характеристиками:

- **Поля**:
  - `std::mt19937 rng`: переменная, используемая для генерации случайных чисел;
  - `int width, height`: переменные для хранения ширины и высоты поля соответственно;
  - `std::string rule`: переменная, используемая для хранения вектора-столбца значений двоичной функции $f(s_0, s_1, s_2, s_3, s_4)$ в виде строки;
  - `std::string boundaryCondition`: переменная для хранения формата граничных условий - «zero», «one», «toroidal»;
  - `std::vector<std::vector<int» field`: контейнер для хранения текущего состояния клеточного поля;
  - `std::vector<std::vector<int» nextField`: контейнер для хранения следующего состояния клеточного поля, сдвинутого на один такт эволюции вперёд.
- **Методы**:
  - `void evolve()`: метод, выполняющий сдвиг клеточного поля на один такт эволюции вперёд;
  - `void display() const`: метод для отображения текущего состояния клеточного поля на консоль;
  - `void setInitialState()`: метод для задания исходного состояния клеточного поля пользователем;
  - `void generateInitialState()`: private-метод для случайной генерации исходного состояния клеточного поля;
  - `unsigned int countAliveNeighbors(int x, int y) const`: private-метод для подсчёта количества «живых» клеток, граничащих с клеткой с координатами (x, y);
  - `int getCell(int x, int y) const`: метод для получения значения клетки, расположенной по координатам (x, y) с учётом заданных граничных условий в поле boundaryCondition.

Код создания данного класса приведён на рисунке № 3.

```
1 class CellularAutomaton
2 {
3     std::mt19937 rng;
4 
5     int width, height;
6     std::string rule;
7     std::string boundaryCondition;
```

```
8       std::vector<std::vector<int>> field;
9       std::vector<std::vector<int>> nextField;
10
11  public:
12      CellularAutomaton(int width, int height, std::string rule, std::
            string boundaryCondition, bool randomInit = false);
13
14      void evolve();
15
16      void display() const;
17
18      void setInitialState();
19
20  private:
21      void generateInitialState();
22
23      unsigned int countAliveNeighbors(int x, int y) const;
24
25      int getCell(int x, int y) const;
26  };
```

Рис. 3. Код создания класса `CellularAutomaton`

2.1 Функция получения значения клетки

Функция принимает на вход два параметра – координаты клетки (x, y), значение которой Необходимо получить. Далее функция обрабатывает каждый из возможных вариантов граничных условий в поле `boundaryCondition` и возвращает соответствующее значение. Код реализации метода `getCell` представлен на рисунке № 4.

`Вход: координаты клетки x, y типа int.`

`Выход: значение клетки поля типа int в соответствии с её координатами и заданным граничным условием.`

```
1  int CellularAutomaton::getCell(int x, int y) const {
2    if (boundaryCondition == "toroidal") {
3      if (x < 0) {
4        return field[height - 1][y];
5      }
6      else if(x > height) {
7        return field[0][y];
8      }
9
10     if (y < 0) {
11       return field[x][width-1];
12     }
13     else if (y > width) {
14       return field[x][0];
15     }
16     return field[x%height][y%width];
17   }
```

```
18    else if (boundaryCondition == "zero") {
19      if (x < 0 || x >= height || y < 0 || y >= width) {
20        return 0;
21      }
22      return field[x][y];
23    }
24    else if (boundaryCondition == "one") {
25      if (x < 0 || x >= height || y < 0 || y >= width) {
26        return 1;
27      }
28      return field[x][y];
29    }
30    return 0;
31  }
```

Рис. 4. Код реализации метода `getCell`

2.2 Функция подсчёта количества живых клеток

Функция `countAliveNeighbors` подсчитывает состояния соседей клетки, используя метод `getCell` для получения их значений. Возвращает закодированное целое число, где состояния соседей закодированы битами. Каждый сосед представлен в виде отдельного бита:

- Сосед сверху s_1 – 8-й бит (самый старший, s1 « 3).
- Сосед снизу s_2 – 4-й бит.
- Сосед слева s_3 – 2-й бит.
- Сосед справа s_4 – 1-й бит (самый младший, s4).

Код реализации метода `countAliveNeighbors` представлен на рисунке № 5.

```
Вход:
x (тип int): координата клетки по оси X.
y (тип int): координата клетки по оси Y.

Выход:
Возвращает unsigned int: целое число, где состояния соседей закодированы в
4 битах.
```

```
1  unsigned int CellularAutomaton::countAliveNeighbors(int x, int y) const
       {
2    unsigned int s1 = getCell(x, y - 1);
3    unsigned int s2 = getCell(x, y + 1);
4    unsigned int s3 = getCell(x - 1, y);
5    unsigned int s4 = getCell(x + 1, y);
6
7    return (s1 << 3) + (s2 << 2) + (s3 << 1) + s4;
8  }
```

Рис. 5. Код реализации метода `countAliveNeighbors`

2.3 Функция перехода к следующему состоянию

Функция `evolve` вычисляет следующее состояние клеточного автомата. Она проходит через каждую клетку, подсчитывает состояния соседей, используя `countAliveNeighbors`, определяет новое состояние клетки на основе текущего состояния и правила, закодированного в строке `rule`, и сохраняет новое состояние в массиве `nextField`. Затем обновляет основное поле `field` значениями из `nextField`.

Код реализации метода `evolve` представлен на рисунке № 6.

```
Вход:
field (тип std::vector<std::vector<int>>)
Текущее состояние клеточного автомата, представленное в виде двумерного
массива.

width (тип int)
Ширина поля (количество столбцов).

height (тип int)
Высота поля (количество строк).

rule (тип std::string)
Правило перехода клеточного автомата, закодированное в строке.

nextField (тип std::vector<std::vector<int>>)
Временный массив для записи новых состояний клеток.

Выход:
Поле field обновляется до следующего поколения.
```

```
1  void CellularAutomaton::evolve() {
2    // Проходим по каждой клетке
3    for (int x = 0; x < height; x++) {
4      for (int y = 0; y < width; y++) {
5        // Подсчитываем количество живых соседей для клетки (x, y)
6        int aliveNeighbors = countAliveNeighbors(x, y);
7
8        // Текущий статус клетки
9        int currentState = field[x][y];
10
11       // Вычисляем индекс в правиле для данной клетки и соседей
12       int ruleIndex = (currentState << 4) | aliveNeighbors;
13       int newState = (rule[ruleIndex] == '1') ? 1 : 0;
14
15       // Обновляем nextField для данной клетки
16       nextField[x][y] = newState;
17     }
18   }
19
20   // Копируем nextField в field для следующего поколения
21   for (int x = 0; x < height; x++) {
22     for (int y = 0; y < width; y++) {
```

```
23          field[x][y] = nextField[x][y];
24      }
25    }
26  }
```

Рис. 6. Код реализации метода `evolve`

3 Результаты работы программы

На рисунке № 7 представлено создание клеточного автомата №1 с полем 2x2, ручным заданием начального состояния, единичными границами и количеством итераций, равным 4.

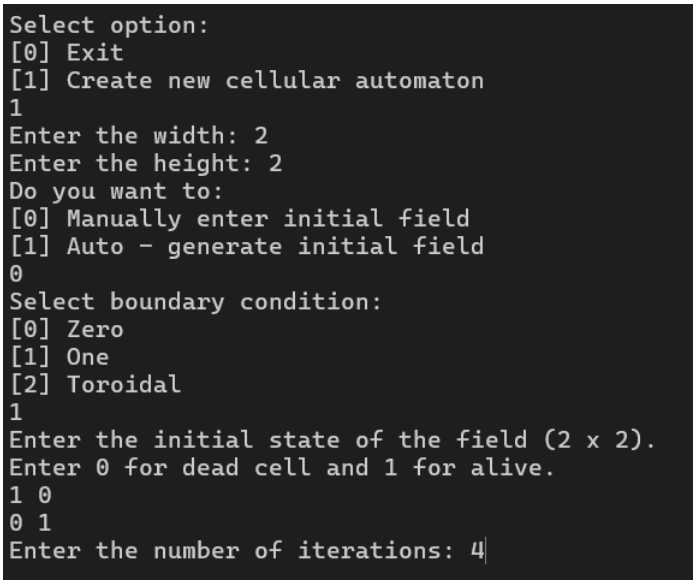

Рис. 7. Создание клеточного автомата №1

На рисунке № 8 представлены результаты эволюции клеточного автомата №1 за все 4 итерации.

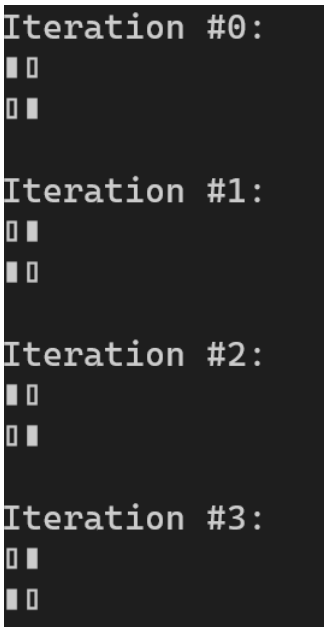

Рис. 8. Итерации эволюции клеточного автомата №1

На рисунке № 9 представлено создание клеточного автомата №2 с полем 2x2,

ручным заданием начального состояния, тороидальными границами и количеством итераций, равным 4.

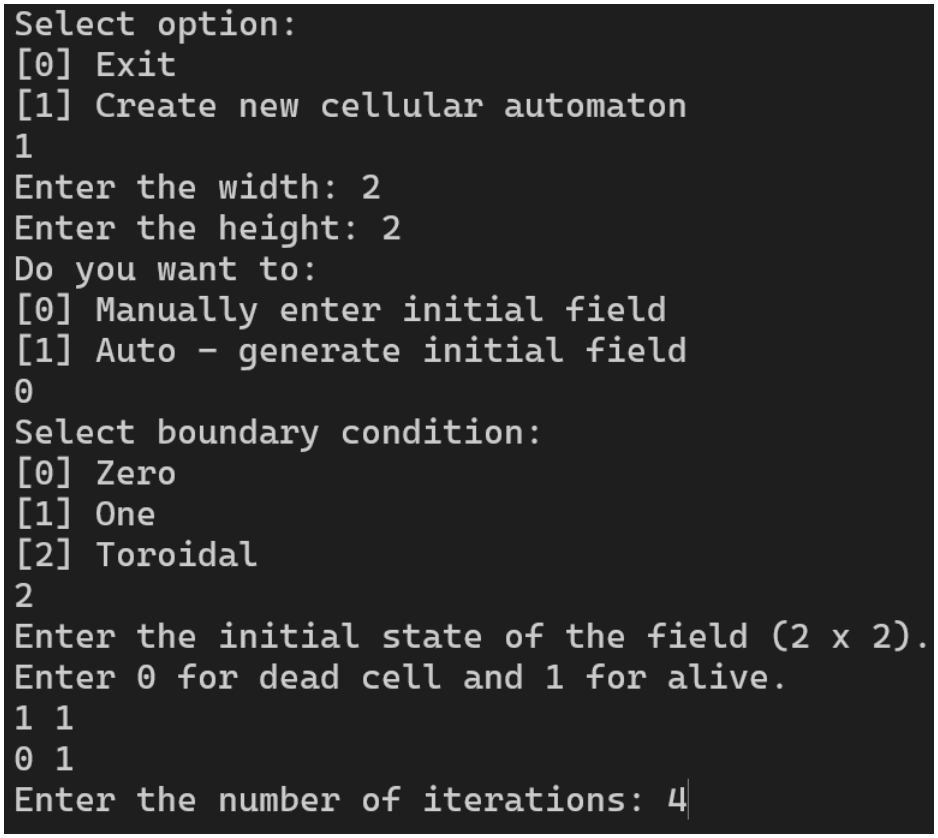

Рис. 9. Создание клеточного автомата №2

На рисунке № 10 представлены результаты эволюции клеточного автомата №2 за все 4 итерации.

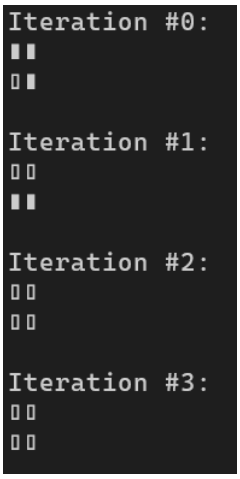

Рис. 10. Итерации эволюции клеточного автомата №2

На рисунке № 11 представлено создание клеточного автомата №3 с полем 20x20, автоматическим заданием начального состояния, нулевыми границами и количеством итераций, равным 4.

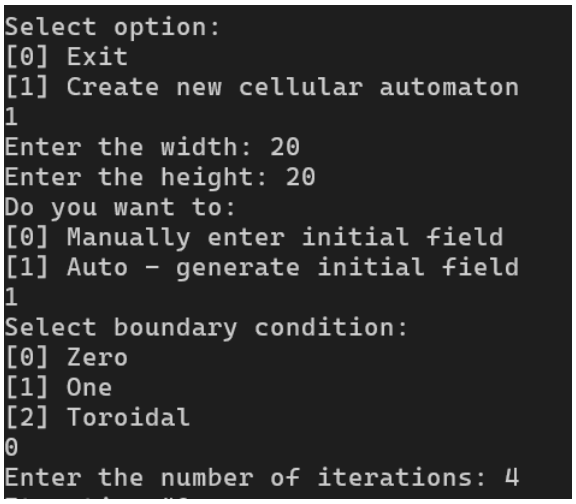

Рис. 11. Создание клеточного автомата №3

На рисунке № 12 представлены результаты эволюции клеточного автомата №3 за все 4 итерации.

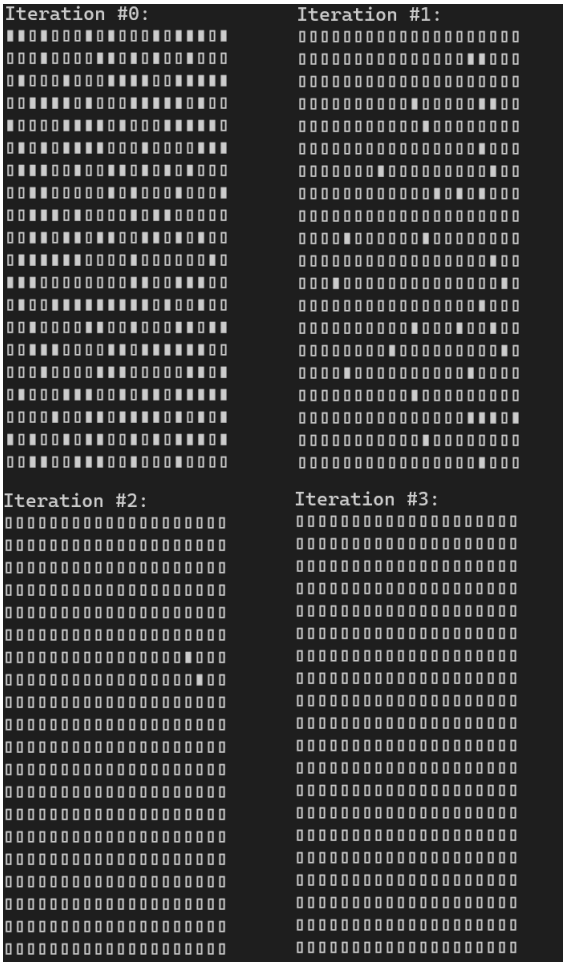

Рис. 12. Итерации эволюции клеточного автомата №3

На рисунке № 13 представлено создание клеточного автомата №4 с полем 5x5,

автоматическим заданием начального состояния, единичными границами и количеством итераций, равным 20.

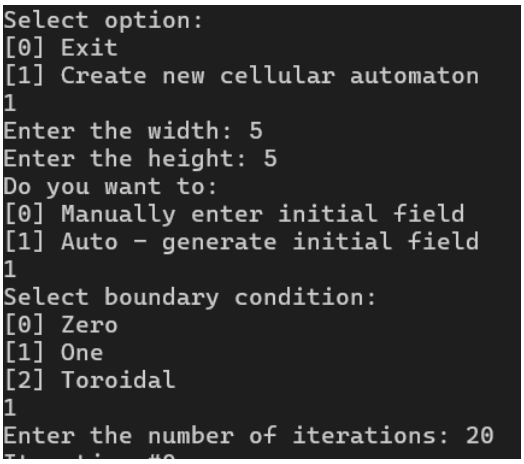

Рис. 13. Создание клеточного автомата №4

На рисунке № 14 представлены результаты эволюции клеточного автомата №4 за все 20 итераций.

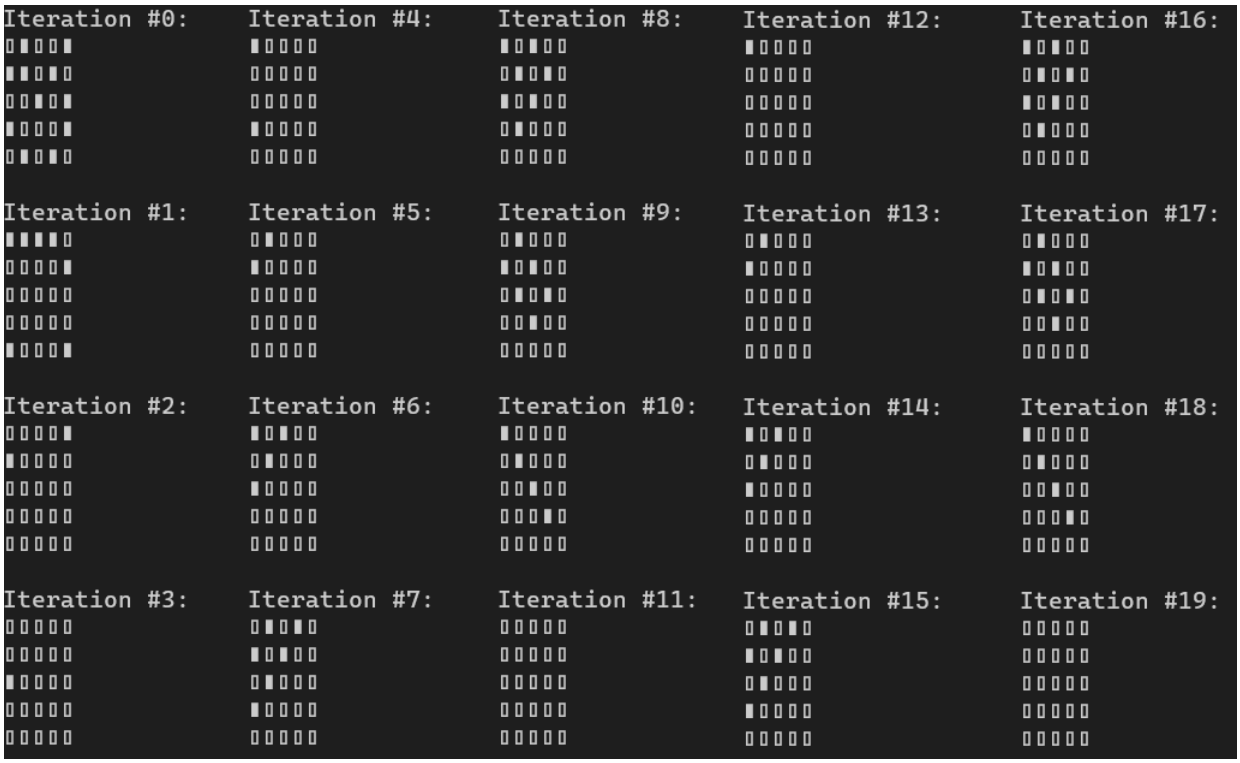

Рис. 14. Итерации эволюции клеточного автомата №4

4 Анализ клеточного автомата

В итоге получился клеточный автомат, реализующий правило 2424840_{10} с четырьмя соседями (окрестность фон Неймана). Граничные условия выбирает пользователь (торроидальные, нулевые, единичные). Начальные состояния генерируются случайным образом или задаются пользователем.

Для анализа использовались пять случайно сгенерированных полей размером 20x20. Автомат эволюционирует, обновляя каждую клетку на основе правила, учитывающего текущее состояние и число активных соседей.

Реализованный клеточный автомат с единичными границами и любым начальным состоянием:

- Демонстрирует периодический паттерн, устойчивый к изменениям начальных условий.
- Демонстрирует цикличность, которая возникает через 3–5 итераций (в зависимости от начального состояния).
- Полностью не затухает: количество живых клеток остаётся постоянным на каждом цикле.
- Появляется устойчивый рисунок в виде прямоугольника, перемещающегося из левого верхнего угла в правый нижний и постепенно уменьшающегося по высоте.
- Сходится к периодичности состояний (рис. 15, 16).
- Относится к классу 2 по классификации Вольфрама.
- Напоминает биологические циклы, такие как суточные ритмы или процессы восстановления.

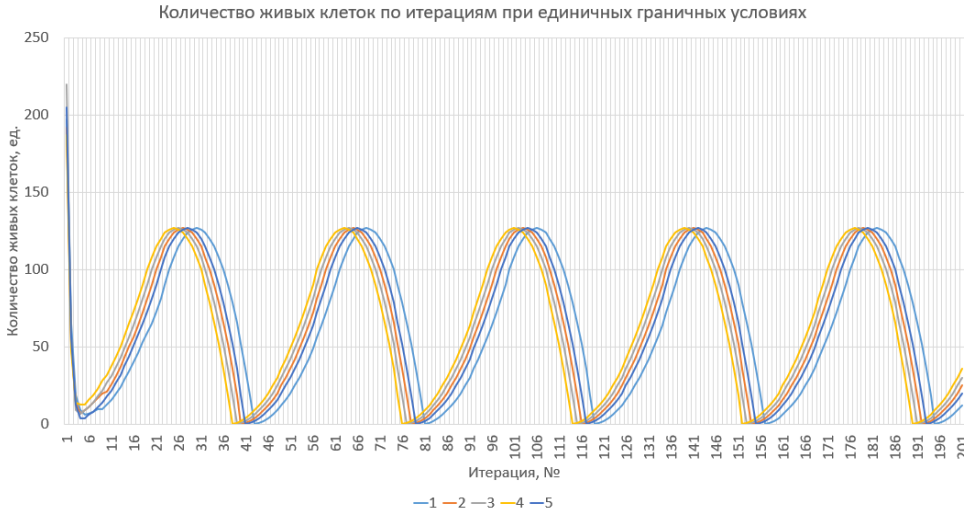

Рис. 15. График количества живых клеток по итерациям при единичных граничных условиях

Реализованный клеточный автомат с нулевыми и тороидальными границами и любым начальным состоянием:

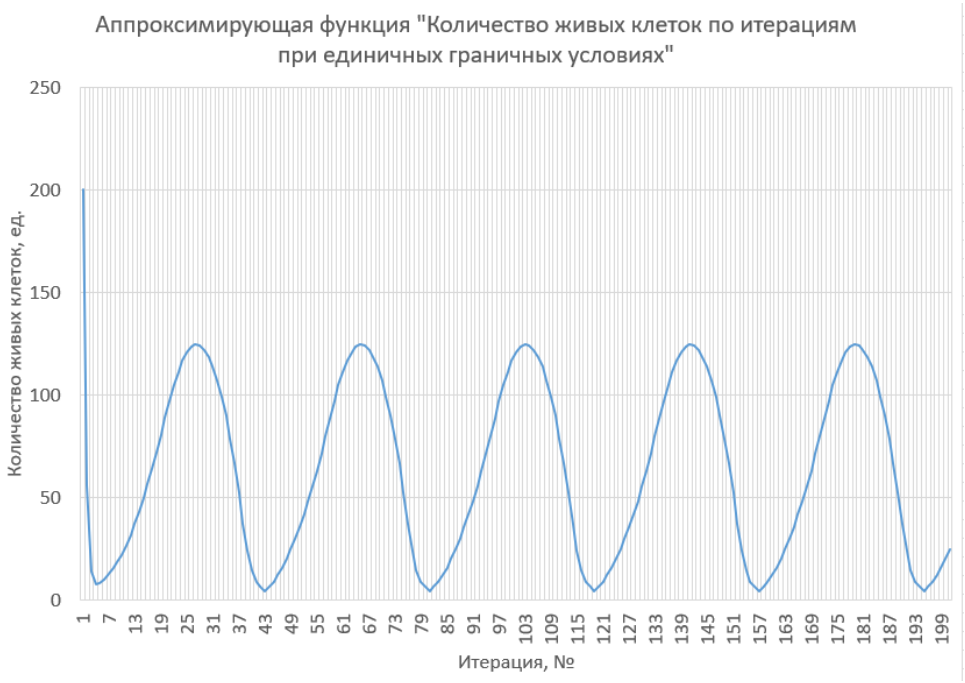

Рис. 16. График аппроксимирующей функции количества живых клеток по итерациям при единичных граничных условиях

- Демонстрирует хаотичный паттерн в первые 2–3 итерации.
- Полностью затухает к 3–4 итерации, независимо от начальных условий.
- Не демонстрирует устойчивых структур или цикличности.
- Отсутствуют локальные области порядка.
- Сходится к состоянию исчезновения всех живых клеток (рис. 17-20).
- Относится к классу 1 по классификации Вольфрама.
- Для полностью заполненного единичного поля автомат затухает быстрее – за одну итерацию.
- Напоминает диссипативный процесс, например, рассеивание тепла или угасание волн.

Рис. 17. График количества живых клеток по итерациям при нулевых граничных условиях

Рис. 18. График аппроксимирующей функции количества живых клеток по итерациям при нулевых граничных условиях

Рис. 19. График количества живых клеток по итерациям при тороидальных граничных условиях

Рис. 20. График аппроксимирующей функции количества живых клеток по итерациям при тороидальных граничных условиях

Заключение

В ходе выполнения лабораторной работы №1 были выполнены следующие задачи:

1. Реализован двумерный клеточный автомат с окрестностью фон Неймана в соответствии с полученным номером № = 2424840_{10}.
2. Реализованы торроидальные, нулевые, единичные граничные условия с возможностью выбора пользователем.
3. Пользователь определяет ширину поля и количество итераций.
4. Учтена возможность ввода различных начальных условий (как вручную, так и случайным образом) по выбору пользователя.
5. Отображение было реализовано в консоли.
6. Проанализирован клеточный автомат в отчете.

Из достоинств реализации можно отметить:

- Поддержка нескольких типов границ (торроидальные, нулевые, единичные).
- Обобщённость правила.

Из недостатков реализации можно отметить:

- Ограниченность правил: возможность задавать только одно фиксированное правило за запуск.
- Фиксированная окрестность: используется только окрестность фон Неймана.
- Отсутствие статистики.
- Ограниченная визуализация.

Проект располагает к масштабированию: имеется возможность добавить поддержку окрестности Мура, внедрить возможность задания правила в виде 32-битного двоичного числа через пользовательский интерфейс, внедрить сбор статистики, добавить графическую визуализацию.

Работа была выполнена в среде разработки Microsoft Visual Studio 2022 (v143). Использовался стандарт языка ISO C++ 14 и компилятор MSVC версии 14.32.31326. Была выбрана платформа решений x64.

Исходный код всего проекта находится в репозитории GitHub (`https://github.com/vlgUseless/CellularAutomaton`).

Полученные знания могут быть и будут испольованы в работе над последующими проектами и заданиями.

Список источников

[1] ИТМО. Линейный клеточный автомат, эквивалентность МТ [Электронный ресурс] URL: https://neerc.ifmo.ru/wiki/index.php?title=Линейный_клеточный_автомат,_эквивалентность_МТ (дата обращения 20.11.2024).

[2] ИТМО. Модели клеточных автоматов [Электронный ресурс] URL: https://neerc.ifmo.ru/wiki/index.php?title=Модели_клеточных_автоматов (дата обращения 20.11.2024).

[3] A.G.Hoekstra, J.Kroc, P.Sloot. Simulating complex systems by cellular automata. Springer, 2010. ISBN 978-3-642-12202-6