МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ПРОФЕССИОНАЛЬНОГО 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

Лабораторная работа «Реализация клеточного автомата» по дисциплине «Теория алгоритмов»

Обучающийся: _____ Яшнова Дарья Михайловна
группа 5130201/20002

Руководитель: _____ Востров Алексей Владимирович

«___»_____ 2024г

Санкт-Петербург, 2024

Содержание

Введение

В рамках данной лабораторной работы необходимо реализовать двумерный клеточный автомат с окрестностью фон Неймана в соответствии с заданным вариантом.

Для этого было использовано следующее число: 26373501_{10}.

Также необходимо определиться, какие граничные условия будут использованы. Необходимо дать возможность пользователю самостоятельно определять ширину поля и количество итераций.

Необходимо предусмотреть возможность ввода различных начальных условий (как вручную, так и случайным образом) по выбору пользователя.

Реализовать программу можно как в консоли, так и в графическом интерфейсе.

В результате необходимо проанализировать поведение клеточного автомата в отчёте, включая паттерны, сходимость и другие характеристики.

1 Математическое описание

1.1 Клеточный автомат

Клеточный автомат — дискретная модель, изучаемая в математике, теории вычислимости, физике, теоретической биологии и микромеханике.

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. Обозначим множество состояний клеток $S = \{s_1, s_2, ..., s_n\}$.

$$\phi(c_0, c_1, c_2, c_3, c_4)$$

- функция, аргументами которой являются состояния соседей клетки, вычисляющая новое состояние для клетки.

$$new_s = \phi(c_0, c_1, c_2, c_3, c_4)$$

- новое состояние клетки.

Для корректной работы программы нужно, чтобы i и j не превышали 30.

Основой клеточного автомата является пространство из прилегающих друг к другу клеток (ячеек), образующих решётку. Каждая клетка может находиться в одном из конечного множества состояний (например, 1 и 0).

Для каждой клетки определено множество клеток, называемых окрестностью. Устанавливаются правила перехода клеток из одного состояния в другое. Обычно правила перехода одинаковы для всех клеток. В данной работе используется окрестность фон Неймана (рис.1).

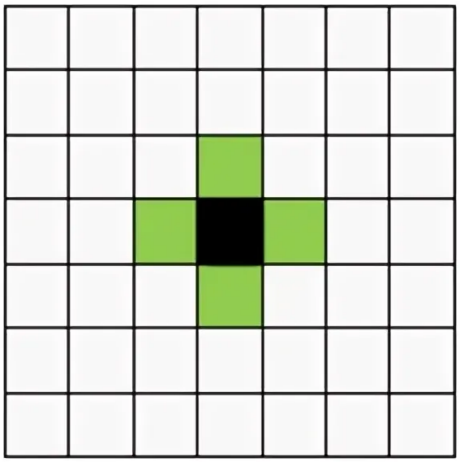

Рис. 1: Окрестность фон Неймана

Один шаг автомата подразумевает обход всех клеток и на основе данных о текущем состоянии клетки и её окрестности определение нового состояния клетки, которое будет у неё при следующем шаге.

Перед стартом автомата оговаривается начальное состояние клеток, которое может устанавливаться целенаправленно или случайным образом.

В данной работе функция представлена числом 11*19*21*3*2003 = 26373501_{10}. Так как нужна функция 5 переменных, то двоичная форма числа дополняется нулями в старших разрядах. Все переходы состояний представлены на рис.2.

	x0	x1	x2	x3	x4	f
0	0	0	0	0	0	0
1	0	0	0	0	1	0
2	0	0	0	1	0	0
3	0	0	0	1	1	0
4	0	0	1	0	0	0
5	0	0	1	0	1	0
6	0	0	1	1	0	0
7	0	0	1	1	1	1
8	0	1	0	0	0	1
9	0	1	0	0	1	0
10	0	1	0	1	0	0
11	0	1	0	1	1	1
12	0	1	1	0	0	0
13	0	1	1	0	1	0
14	0	1	1	1	0	1
15	0	1	1	1	1	0
16	1	0	0	0	0	0
17	1	0	0	0	1	1
18	1	0	0	1	0	1
19	1	0	0	1	1	0
20	1	0	1	0	0	1
21	1	0	1	0	1	1
22	1	0	1	1	0	0
23	1	0	1	1	1	1
24	1	1	0	0	0	0
25	1	1	0	0	1	1
26	1	1	0	1	0	1
27	1	1	0	1	1	1
28	1	1	1	0	0	1
29	1	1	1	0	1	1
30	1	1	1	1	0	0
31	1	1	1	1	1	1

Рис. 2: Таблица переходов состояний

В качестве граничных были выбраны тороидальные условия. Для клетки в i-той строке и j-том столбце клетки-соседи находятся по формулам:

$$cell_0 = field_i^j$$
$$cell_1 = field_i^{(j-1) \mod h}$$
$$cell_2 = field_{(i-1) \mod v}^j$$
$$cell_3 = field_i^{(j+1) \mod h}$$
$$cell_4 = field_{(i+1) \mod v}^j,$$

где h - размер поля по горизонтали, v - размер поля по вертикали, а $field_i^j$ - клетка в i-той строке и j-том столбце.

1.2 Классификация клеточных автоматов

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.

2 Особенности реализации

Клеточный автомат задан в программе числом func = 26373501_{10}. Для представления самого клеточного автомата используются двумерные списки. Реализация переходов состояний включает следующие функции:

- Функция upd_state - обновляет состояние клетки на основе состояния ближайших соседей и текущего состояния с помощью функции ith. Принимает текущее состояние клетки и ее соседей, возвращает обновленное состояние клетки.

```
def upd_state(state, n0, n1, n2, n3):
    num = state * 2 ** 4 + n3 * 2 ** 3 + n2 * 2 ** 2 + n1 * 2 + n0
    return ith(num)
```

- Функция ith - возвращает конкретное состояние на основании целого числа i и функции func, которая определяет правила обновления.

```
def ith(i):
    i = 32 - i - 1
    return (func & (1 << i)) >> i
```

- Функция updMx - обновляет всю матрицу состояний (матрицу клеточного автомата) применяя функцию upd_state к каждой клетке. На вход принимает матрицу состояний и возвращает обновленную матрицу состояний.

```
def updMx(mas):
    nMx = []
    for i in range(v_n):
        list = []
        for j in range(h_n):
            nstate = upd_state(mas[i][j],
                               mas[i][(j - 1) % h_n],
                               mas[(i - 1) % v_n][j],
                               mas[i][(j + 1) % h_n],
                               mas[(i + 1) % v_n][j])
            list.append(nstate)
        nMx.append(list)
    return nMx
```

- Обработка ввода пользователя:
  - Пользователю предлагается ввести количество итераций, вертикальный и горизонтальный размер поля.
  - Пользователь может выбрать случайную или вручную введенную начальную матрицу состояний.

```
print("Input number of iterations:")
it = 0
while it <= 0:
    try:
        it = int(input())
    except ValueError:
        print("Try again")
    if (it < 0):
        print("Try again")

print("Input vertical size of field:")
v_n = 0
while v_n <= 0:
```

```
    try:
        v_n = int(input())
    except ValueError:
        print("Try again")
    if (v_n < 0):
        print("Try again")

print("Input horizontal size of field:")
h_n = 0
while h_n <= 0:
    try:
        h_n = int(input())
    except ValueError:
        print("Try again")
    if (h_n < 0):
        print("Try again")

print("I can generate random matrix. If you want - press 0, if you
   need to enter matrix - press 1")

ch = 2
while ch != 0 and ch != 1:
    try:
        ch = int(input())
        if ch != 0 and ch != 1:
            print("Try again")
    except ValueError:
        print("Try again")
```

- Выбор начальной матрицы masBeg - создается либо случайная матрица, либо пользователь вводит ее сам.

```
 masBeg = []

if ch == 0:
    masBeg = np.random.randint(0,2,(v_n, h_n))
else:
    for i in range(v_n):
        arr = list(map(int, input().split()))
        res = all(((x == 0) or (x==1)) for x in arr)
        while(len(arr)!=h_n or not res):
            print("A lot of numbers or not 0 or 1, try again")
            arr = list(map(int, input().split()))
            res = all(((x == 0) or (x == 1)) for x in arr)

        masBeg.append(arr)
```

- Цикл итераций автомата:
  - Выполняется цикл на заданное пользователем количество итераций.
  - В каждой итерации обновляется текущая матрица состояний и выводится ее визуализация с помощью plt.spy.
  - Итерации выводятся с задержкой для наблюдения изменений.

```
    for i in range(it - 1):
    curMx = updMx(masBeg)
    print(i + 1)
    print(np.matrix(curMx))
    plt.ion()
    x = range(1, v_n+1)
    y = range(1, h_n + 1)
```

```
8       plt.spy(curMx, precision=0.1, markersize=20)
9       plt.xticks(np.arange(0, h_n, 1))
10      plt.yticks(np.arange(0, v_n, 1))
11
12      plt.grid(visible=True, linewidth=0.5, which='both')
13      print(i)
14      plt.show()
15      plt.pause(1.1)
16
17      plt.clf()
18      masBeg = curMx
```

- В качестве дополнительной опции реализовано воспроизведение музыки во время смены итераций автомата.

3 Результаты работы программы

На рис.3-6 представлены результаты вывода программы. Для каждого состояния выводится матрица и номер перехода. На рис.6 представлены различные варианты обработки ошибок ввода. На рис.7 представлена визуализация одного из переходов с помощью графика.

```
Input number of iterations:
10
Input vertical size of field:
10
Input horizontal size of field:
10
I can generate random matrix. If you want - press 0, if you need to enter matrix - press 1
0
Beginning state:
[[1 1 0 1 0 1 1 0 1 1]
 [1 0 1 1 1 0 1 1 1 1]
 [0 0 0 1 1 0 0 0 1 0]
 [1 1 0 0 0 1 1 0 0 1]
 [0 0 1 1 0 1 1 0 0 1]
 [0 0 1 1 0 0 0 1 1 0]
 [1 1 0 0 0 0 1 0 1 1]
 [0 1 0 1 1 1 1 1 1 1]
 [1 0 0 1 0 1 1 0 1 1]
 [1 1 1 0 1 1 1 1 1 1]]
```

Рис. 3: Пример ввода размеров поля, количества итераций и генерации матрицы

```
2
[[0 1 0 1 0 1 1 1 1 0]
 [1 1 1 1 1 1 1 1 1 1]
 [0 1 0 0 0 0 1 0 0 1]
 [1 0 0 1 0 1 1 0 0 0]
 [1 0 1 1 0 0 0 1 0 1]
 [1 0 0 0 1 0 1 1 1 1]
 [1 1 0 0 0 0 0 0 1 0]
 [0 1 0 1 1 1 1 1 1 1]
 [1 1 0 1 1 1 1 1 1 0]
 [1 1 1 1 1 1 1 1 1 1]]
3
[[1 1 0 1 0 0 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1]
 [0 1 0 0 0 1 1 0 0 1]
 [0 0 0 0 0 1 0 0 0 1]
 [1 0 1 0 0 0 1 0 0 1]
 [1 0 0 0 0 0 1 1 1 1]
 [0 1 0 1 0 1 0 1 1 0]
 [0 1 0 1 1 1 1 1 1 1]
 [1 1 0 0 1 1 1 1 1 0]
 [1 1 1 1 1 1 1 1 1 1]]
```

Рис. 4: Пример вывода матриц на каждой итерации

```
Input number of iterations:
10
Input vertical size of field:
4
Input horizontal size of field:
4
I can generate random matrix. If you want - press 0, if you need to enter matrix - press 1
1

A lot of numbers or not 0 or 1, try again
1 0 0 1
0 0 0 0
1 1 1 1
0 0 0 0
Beginning state:
[[1 0 0 1]
 [0 0 0 0]
 [1 1 1 1]
 [0 0 0 0]]
1
[[1 0 0 1]
 [0 1 1 0]
 [1 1 1 1]
 [0 0 0 0]]
```

Рис. 5: Пример ввода матрицы вручную

```
Input number of iterations:
r
Try again
-1
Try again
30
Input vertical size of field:
y
Try again
10
Input horizontal size of field:
10
I can generate random matrix. If you want - press 0, if you need to enter matrix - press 1
1
w w w w
Try again
```

Рис. 6: Пример обработки ошибок ввода

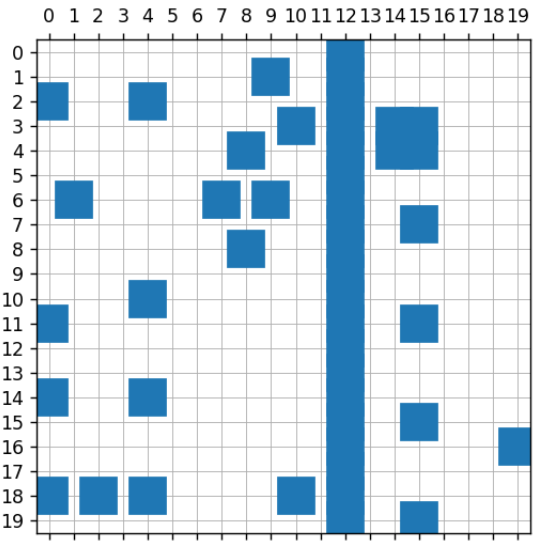

Рис. 7: Пример визуализации состояний

4 Анализ

В данной работе представлен автомат 4 класса, так как автомат порождает сложные, взаимодействующие между собой структуры, способные выживать длительное время, однако не достигает стабильности. Примеры получающихся переходов можно видеть на рис.8-12.

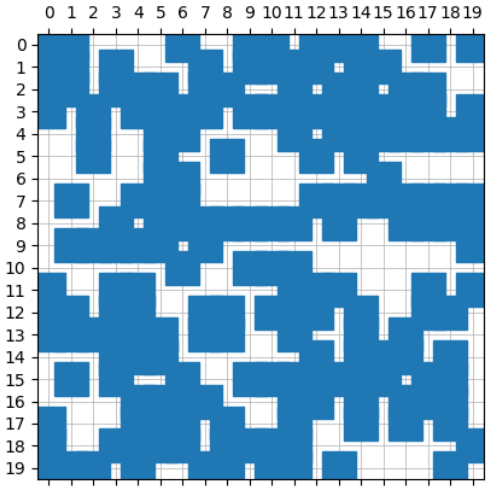

Рис. 8: Начальное состояние автомата 20*20 клеток

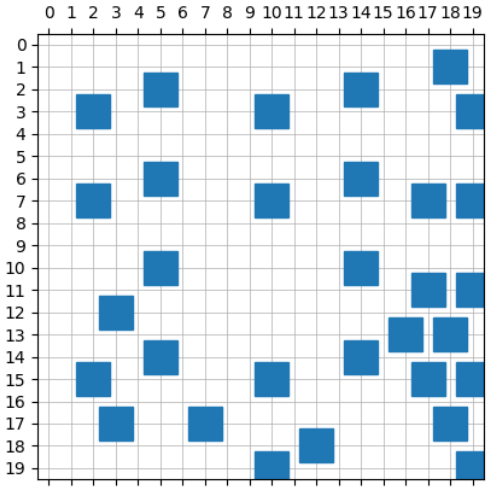

Рис. 9: 98-е состояние автомата 20*20 клеток

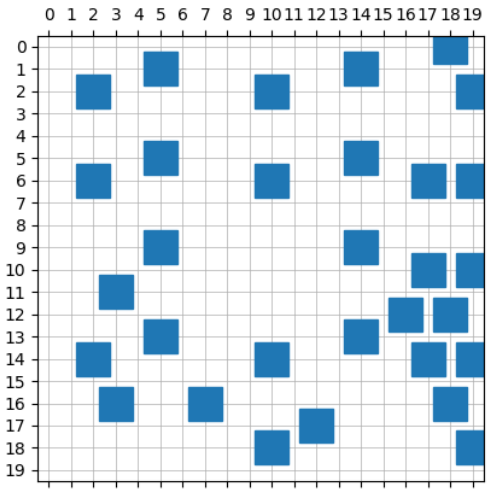

Рис. 10: 99-е состояние автомата 20*20 клеток

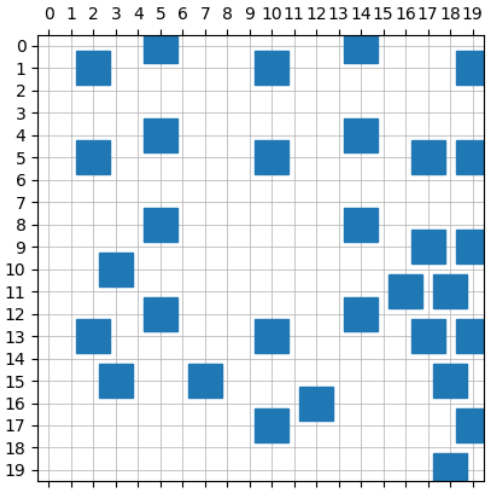

Рис. 11: 100-е состояние автомата 20*20 клеток

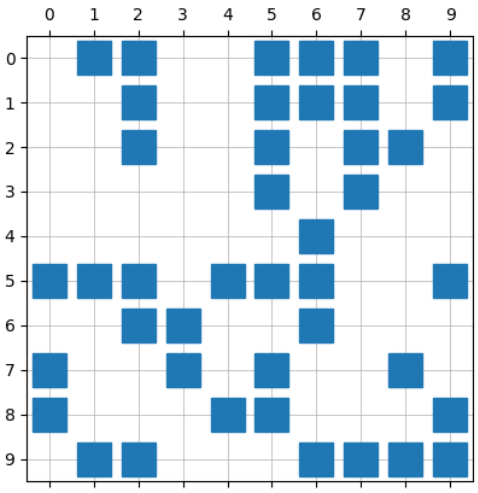

Рис. 12: Начальное состояние автомата 10*10 клеток

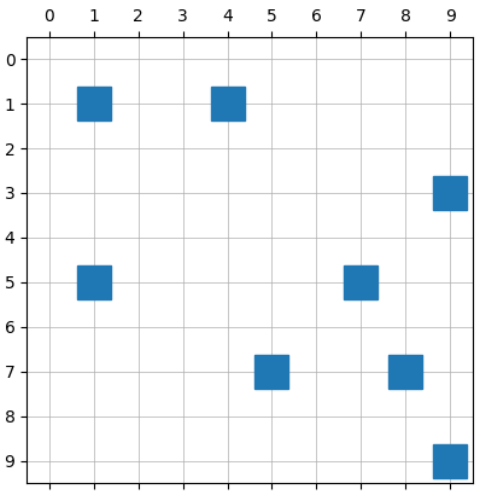

Рис. 13: 98-е состояние автомата 10*10 клеток

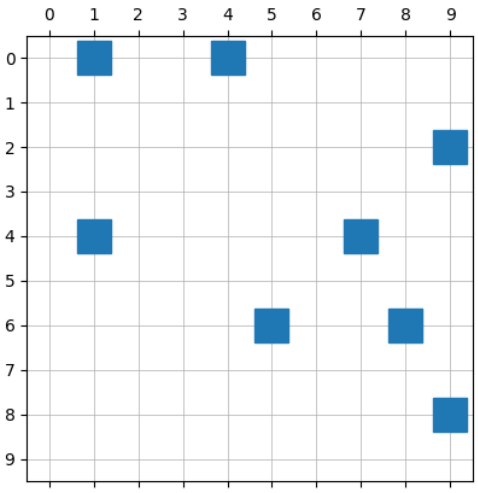

Рис. 14: 99-е состояние автомата 10*10 клеток

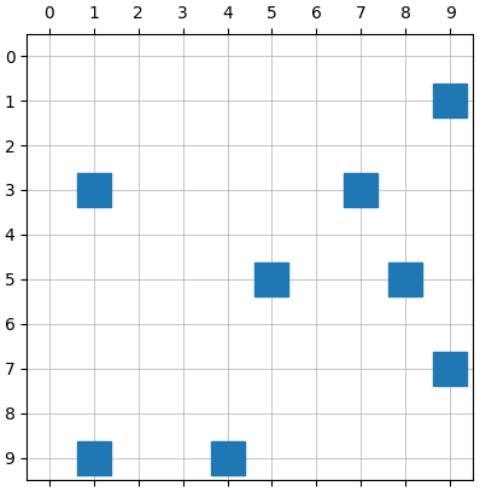

Рис. 15: 100-е состояние автомата 10*10 клеток

На рисунках представлены различные паттерны поведения автомата для некоторых начальных клеточных структур.

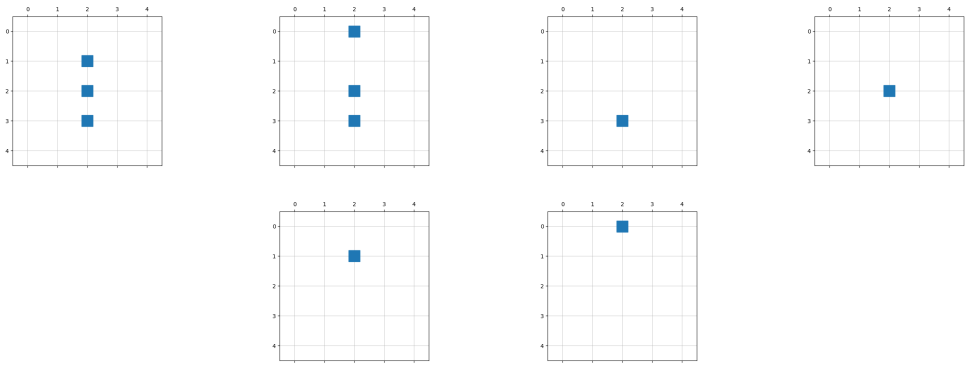

Рис. 16: Поведение автомата, если начальное состояние - прямоугольник 1*3

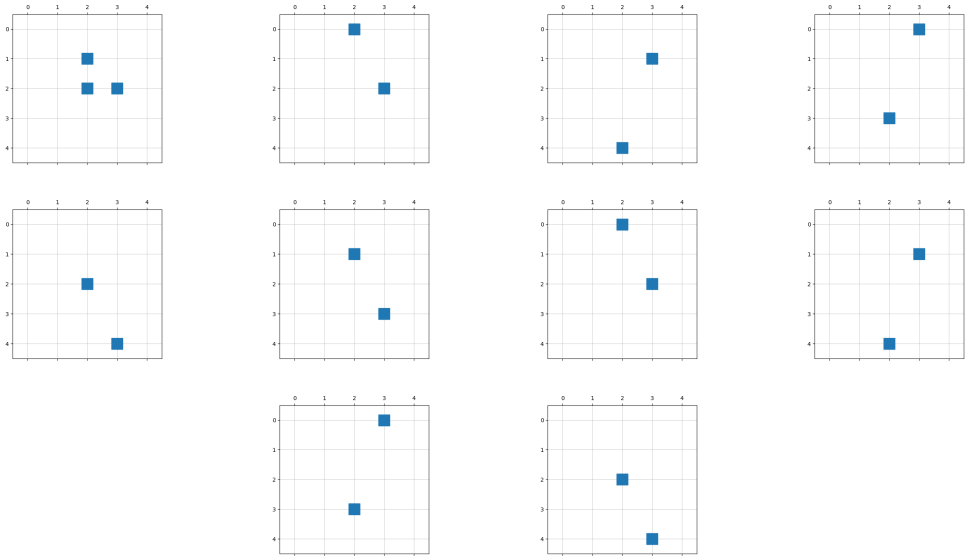

Рис. 17: Поведение автомата, если начальное состояние - уголок из трех клеток

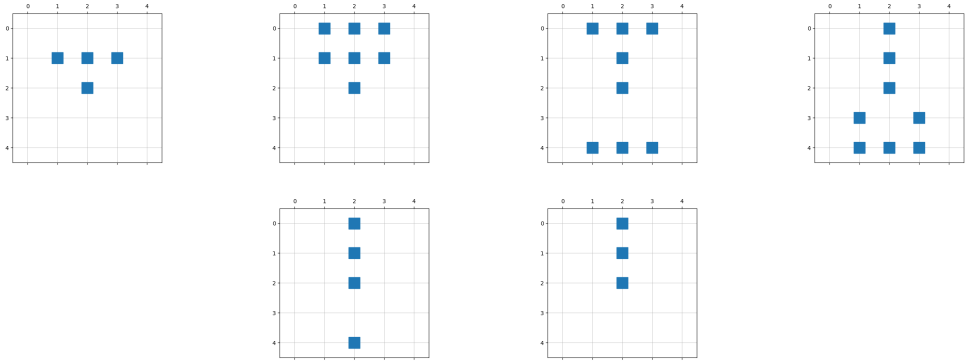

Рис. 18: Поведение автомата, если начальное состояние - фигура в форме буквы T из 4 клеток

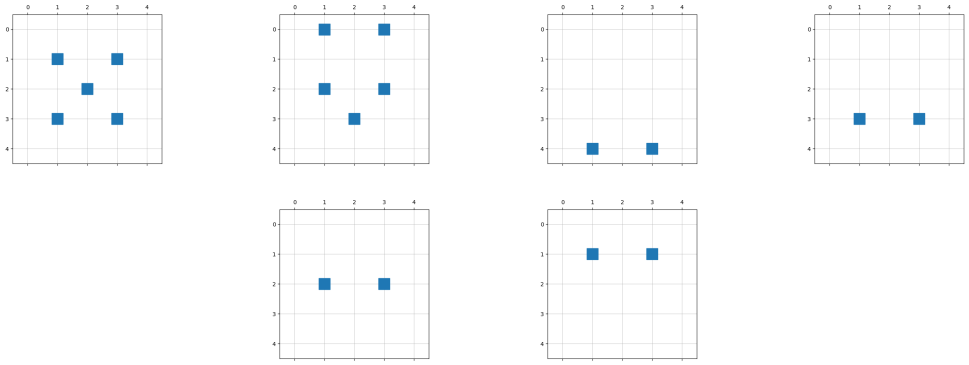

Рис. 19: Поведение автомата, если начальное состояние - фигура в форме буквы X из 5 клеток

В большинстве случаев после некоторых преобразований «живые» клетки двигаются вверх до конца поля, и так как условия тороидальные, те же самые клетки появляются снизу поля. На рис.16 представлена диаграмма зависимости количества живых клеток

от номера итерации. Можно видеть что примерно после 50 шага количество почти не изменяется. До 50 шага количество живых клеток уменьшается.

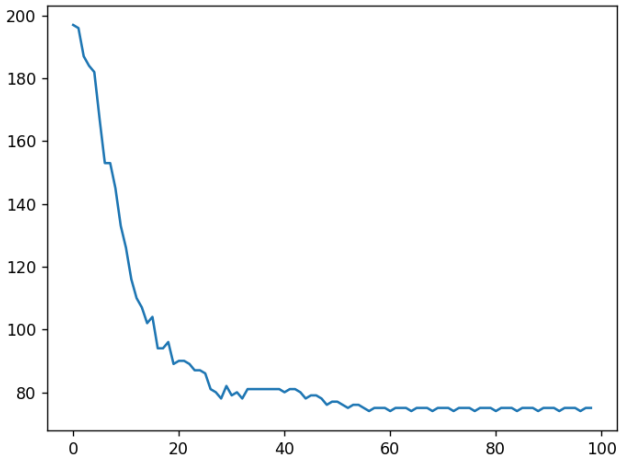

Рис. 20: Диаграмма количества живых клеток в зависимости от номера итерации

Заключение

В данной работе был реализован клеточный автомат с окрестностью Фон-Неймана. В выбранной модели каждая ячейка взаимодействует с пятью элементами: самой собой и четырьмя соседями по вертикали и горизонтали. Функция перехода задана числом 26373501_{10}. Размер поля вводится пользователем, что обеспечивает возможность адаптации модели под различные задачи. Начальные условия могут быть заданы вручную или сгенерированы случайно, что позволяет исследовать влияние стартовых конфигураций на эволюцию системы. Торроидальные (циклические) граничные условия означают, что противоположные края поля связаны между собой, формируя замкнутую поверхность. Это исключает влияние краевых эффектов и позволяет исследовать внутреннюю динамику автомата без искажений, связанных с границами. В рамках программы была реализована дополнительная функция — воспроизведение мелодии во время демонстрации работы автомата в графическом режиме.

Возможно масштабировать автомат для других функций, зависящих от не более, чем от 5 переменных, так как в реализации функция задана числом. Это является достоинством реализации. Также достоинством реализации является обновление матрицы с использованием битовых операций. Недостатком программы можно назвать невозможность перехода к окрестности Мура и невозможность изменять граничные условия.

Список литературы

[1] Диссертация на тему «Классификация поведения одномерных клеточных автоматов», Скаков П.С., Шалыто А.А.
URL:https://is.ifmo.ru/diploma-theses/_skakov_master.pdf
Дата обращения: 01.11.2024

[2] «New kind of science», S. Wolfram
URL:https://web.archive.org/web/20070508234748/http://www.wolframscience.com/
Дата обращения: 01.11.2024