Министерство образования и науки РФ

Санкт-Петербургский Политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа искусственного интеллекта

Направление 02.03.01 «Математика и компьютерные науки»

Дисциплина «Теория алгоритмов»

Отчёт по лабораторной работе

«Реализация клеточного автомата»

Студент: Башарина Е.А., гр. 3530201/90101

Преподаватель: Востров А.В.

Санкт-Петербург — 2022

Содержание

Введение

Клеточный автомат – это дискретная модель, представляющая собой сетку произвольной размерности, каждая клетка которой может принимать одно из конечного множества состояний. В каждый момент времени каждая клетка может перейти из одного состояния в другое в соответствии с заранее заданным правилом, обычно одинаковым для всех клеток сетки. Клеточные автоматы моделируют процессы, разворачивающиеся в дискретном пространстве и в дискретном времени.

В этом отчете содержится описание реализации клеточного автомата, клетки которого могут изменять своё состояние в соответствии с любой функцией, которую введёт пользователь. Кроме того, отчёт содержит анализ полученного автомата и примеры работы программы.

1 Постановка задачи

Цель лабораторной работы: реализовать двумерный клеточный автомат с окрестностью фон Неймана в соответствии с заданным вариантом (оптимальная реализация - по любому номеру). При этом рассматриваются *тороидальные* граничные условия.
При этом пользователь должен определять следующие параметры:

1. способ ввода начальных условий (заполнения поля): ручной либо автоматический;
2. размеры поля.

После реализации необходимо проанализировать полученный клеточный автомат в отчете.

Номер варианта был посчитан в соответствии с заданием как **число дня рождения** [в двоичном виде] % **месяц рождения** [в двоичном виде] % **год рождения** [в двоичном виде] (% означает операцию конкатенации):

$$\text{Число дня рождения: } 29_{10} = 11101_2$$

$$\text{Месяц рождения: } 5_{10} = 101_2$$

$$\text{Год рождения: } 2002_{10} = 11111010010_2$$

$$\text{Номер варианта: } 11101101111110100102 = 487378_{10}$$

2 Математическое описание

2.1 Клеточный автомат

Клеточный автомат — это пятерка $A =< C, N, Q, q^0, f >$, где

- C — множество клеток;
- $N : C \to C^l$ — отображение множества клеток во множество наборов клеток размера l, определяющее соседей каждой клетки. Таким образом, $N(c)$ — локальная окрестность клетки c;
- Q — конечное множество состояний, в которых могут находиться клетки автомата;
- $q^0 : C \to Q$ — начальное состояние автомата (задается начальное состояние каждой клетки автомата);
- $f : Q^l \to Q$ — набор правил для вычисления нового состояния клетки (которое зависит от текущих состояний всех ее l соседей)[3].

Клеточный автомат имеет состояние, определяемое как **набор** (вектор или матрица) состояний компонентных автоматов. Вход на каждом шаге клеточного автомата — **состояние его соседей**. На следующем шаге — новое состояние каждого автомата определяется как функция его текущего состояния и текущих состояний его соседей[3].

2.2 Классификация клеточных автоматов

В книге «A new kind of science» Стивен Вольфрам предложил 4 класса, на которые могут быть разделены все клеточные автоматы в зависимости от типа их поведения. Классы Вольфрама описываются следующим образом:

- **1 класс** — после некоторого количества шагов система стабилизируется. Все клетки поля переходят в одинаковое состояние;
- **2 класс** — состояния могут быть различными, однако клетки в данном случае образуют наборы статичных или периодических простых структур;
- **3 класс** — образуются сложные структуры, характер взаимодействия которых во многих случаях выглядит хаотичным;
- **4 класс** — сложные структуры, очень сложное, порой непредсказуемое (хаотичное) взаимодействие[1].

Кроме того, различают **одномерные** и **двумерные** клеточные автоматы.

2.3 Двумерный клеточный автомат

Клетки **двумерных клеточных автоматов** располагаются на плоскости в точках с целочисленными координатами, то есть образуют прямоугольную решетку[2].

Существенным отличием двумерных клеточных автоматов по сравнению с одномерными является то, что в двумерном случае возможно несколько различных типов **локальных окрестностей**. В данном случае рассматривается окрестность *фон Неймана* (рис. 1), которая включает в себя набор из четырёх клеток, окружающих данную только по вертикали и горизонтали (без учёта диагональных).

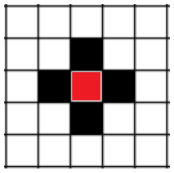

Рис. 1: Окрестность фон Неймана

Функция перехода каждой клетки в новое состояние для **окрестности фон Неймана** определяется следующим образом:

$$F : (s_0, s_1, s_2, s_3, s_4) \to \begin{cases} 0, & f(s_0, s_1, s_2, s_3, s_4) = \text{true} \\ 1, & f(s_0, s_1, s_2, s_3, s_4) = \text{false}. \end{cases}$$

Также для клеточного автомата необходимо ввести **граничные условия** для тех клеток, которые находятся по краям поля. Существуют два способа их введения [2]:

- **граничные условия по умолчанию** — в этом случае несуществующие соседи всегда имеют одно заданное значение;
- **тороидальное граничное условие** — несуществующие соседи присоединяются с обратной стороны поля.

В описываемой реализации использованы тороидальные начальные условия.
Тороидальное граничное условие определяется следующим образом. Пусть $s_0 = a_{i,j}$, то есть находится в i-ой строке и j-ом столбце клеточного автомата, а всего он имеет n строк и m столбцов. Тогда соседи определяются следующим образом.

$$s_1 = \begin{cases} a_{i-1,j}, & \text{если } 0 < i \leq n \\ a_{n,j}, & \text{если } i = 0, \end{cases}$$

$$s_2 = \begin{cases} a_{i+1,j}, & \text{если } 0 \leq i < n \\ a_{0,j}, & \text{если } i = n, \end{cases}$$

$$s_3 = \begin{cases} a_{i,j-1}, & \text{если } 0 < j \leq m \\ a_{i,m}, & \text{если } j = 0, \end{cases}$$

$$s_4 = \begin{cases} a_{i,j+1}, & \text{если } 0 \leq i < m \\ a_{i,0}, & \text{если } i = m. \end{cases}$$

2.4 Функция 487378_{10}

В данной работе в качестве функции, которая управляет работа автомата, задана двоичная функция 487378_{10}. Ее таблица истинности приведена ниже.

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	0
0	1	1	0	0	0
0	1	1	0	1	1
0	1	1	1	0	1
0	1	1	1	1	1
1	0	0	0	0	0
1	0	0	0	1	1
1	0	0	1	0	1
1	0	0	1	1	0
1	0	1	0	0	1
1	0	1	0	1	1
1	0	1	1	0	1
1	0	1	1	1	1
1	1	0	0	0	1
1	1	0	0	1	1
1	1	0	1	0	0
1	1	0	1	1	1
1	1	1	0	0	0
1	1	1	0	1	0
1	1	1	1	0	1
1	1	1	1	1	0

3 Реализация

Для реализации задания были написаны два класса: **Rule**, реализующий правило изменения клеток, и **Game**, визуализирующий изменение состояния поля со временем. Кроме того, в файле main.py реализовано взаимодействие с пользователем и создание объектов этих классов.

3.1 main.py

В файле main.py пользователю предлагается определить следующие параметры:

1. десятичное число rule number - правило изменения клеток автомата во времени;
2. два целых числа m и n - размеры поля;
3. способ заполнения поля - ручной либо автоматический.

Создаются объекты классов Rule и Game, после чего из последнего запускается метод run.

```
from front import Game
from back import Rule

print("Enter rule number: ")
rule = Rule(int(input()))
# rule = Rule(487378)
print("Enter field size: ")
m, n = [int(i) for i in input().split()]
print("Choose way to fill the field:\n1)manually\n2)random")
ans = int(input())
game = Game(m, n, rule)
if ans != 1 and ans != 2:
    print("Couldn't recognise your answer, assuming random generation")
if ans == 1:
    game.run(True)
else:
    game.run(False)
```

3.2 Реализация правила

В конструкторе класса Rule десятичное число, поданное на вход, конвертируется в двоичный вид и сохраняется в поле self.rule.

```
class Rule:
    def __init__(self, rule):
        self.rule = self.dec_to_bin(rule)
```

Кроме того, класс содержит три метода:

1. dec_to_bin
   Вход: число n
   Выход: список двоичных разрядов (0 и 1) - число n в двоичном представлении
   Этот метод предназначен для перевода числа из десятичного вида в двоичный.

```
    def dec_to_bin(self, n):
            number = []
            while n != 0 and len(number) <= 32:
                number.append(n % 2)
                n //= 2
            while len(number) != 32:
                number.append(0)
            number.reverse()
            return number
```

2. bin_to_dec
   Вход: список двоичных разрядов (0 и 1)
   Выход: число n
   Этот метод предназначен для перевода числа из двоичного вида в десятичный.

```
    def bin_to_dec(self, number):
        number.reverse()
        n = 0
        for i in range(len(number)):
            n += number[i]*pow(2, i)
        return n
```

3. calculate_res
   Вход: список из пяти чисел, имеющих значение 0 или 1 - (клетка, сосед сверху, сосед снизу, сосвед слева, сосед справа)
   Выход: True либо False
   Этот метод вычисляет, в какое состояние перейдёт клетка base (первый аргумент из списка) при определённом окружении (последние четыре аргумента из списка). Список, поданный на вход, представляется как двоичное число, десятичное представление которого является порядковым номером бита в заранее заданном правиле (self.rule). Метод возвращает результат сравнения этого бита с единицей.

```
    def calculate_res(self, number): # number: base, up, down, left, right
        return self.rule[self.bin_to_dec(number)] == 1
```

3.3 Визуализация

3.3.1 Класс Game

Класс Game содержит заданные константы цветов и числа кадров в секунду. В конструкторе этого класса создаётся поле заданной размерности, а также инициализируется правило, в соответствии с которым клетки автомата будут изменять своё состояние.

```
class Game:
    BLACK = (0, 0, 0)
    WHITE = (255, 255, 255)
    GREEN = (0, 255, 0)
    FPS = 100

    def __init__(self, w, h, r):
        pygame.init()
```

```
        self.width = w*20
        self.height = h*20
        self.rule = r
        self.screen = pygame.display.set_mode((self.width, self.height))
        pygame.display.set_caption("Lab1Alg")
        self.clock = pygame.time.Clock()
        self.cells = [[0 for i in range(self.width)] for i in range(self.height)]
```

3.3.2 Метод run. Цикл начальной отрисовки поля

Вход: значение man (True или False) - параметр, определяющий способ заполнения поля (ручной либо автоматический соответственно).
Выход: метод не возвращает никакого значения.
В первом цикле метода run поле заполняется сеткой заданного размера, после чего, в зависимости от значения параметра man, поле заполняется либо вручную пользователем, либо автоматически (каждая клетка поля принимает рандомное значение 0 или 1, закрашиваясь белым или чёрным цветом соответственно). Сигналом окончания ввода в обоих случаях является нажатие клавиши Enter.

```
    def run(self, man):
        # Заполняем экран белым цветом
        self.screen.fill(self.WHITE)
        # Рисуем сетку
        for i in range(0, self.screen.get_height() // 20):
            pygame.draw.line(self.screen, self.BLACK, (0, i * 20),\
(self.screen.get_width(), i * 20))
        for j in range(0, self.screen.get_width() // 20):
            pygame.draw.line(self.screen, self.BLACK, (j * 20, 0),\
(j * 20, self.screen.get_height()))
        pygame.display.update()
        running = True
        if man:
            while running:
                self.clock.tick(self.FPS)
                for event in pygame.event.get():
                    if event.type == pygame.QUIT:
                        quit()
                    elif event.type == pygame.MOUSEBUTTONUP:
                        pos = pygame.mouse.get_pos()
                        self.cells[pos[1]//20][pos[0]//20] = 1
                        pygame.draw.rect(self.screen, self.BLACK,\
(pos[0]//20 * 20, pos[1]//20 * 20, 20, 20))
                        pygame.display.flip()
                    elif event.type == pygame.KEYDOWN:
                        if event.key == pygame.K_RETURN:
                            running = False
        else:
            for i in range(self.height):
                for j in range(self.width):
                    self.cells[i][j] = random.randint(0, 1)
                    if self.cells[i][j] == 1:
                        pygame.draw.rect(self.screen, self.BLACK,\
(j * 20, i * 20, 20, 20))
            pygame.display.flip()
```

```
        while running:
            self.clock.tick(self.FPS)
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    quit()
                elif event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_RETURN:
                        running = False

        # number: base, up, down, left, right
```

3.3.3 Метод run. Цикл работы клеточного автомата

Второй цикл метода run содержит реализацию работы клеточного автомата. Для каждой клетки заполненного поля определяются её соседи в окрестности фон Неймана: клетки сверху, снизу, справа и слева. В случае, если клетка находится на границе поля, её соседом считается клетка с обратной стороны. После этого для пятерки (клетка, сосед сверху, сосед снизу, сосед справа, сосед слева) вызывается метод calculate_res поля rule, описанный выше. В зависимости от результата, клетка перекрашивается в чёрный либо белый цвет. Этот процесс происходит бесконечно, до тех пор, пока пользователь не прекратит работу программы клавишей «выход».

```
        while True:
            self.clock.tick(self.FPS)
            newcells = copy.deepcopy(self.cells)
            for i in range(self.height//20):
                for j in range(self.width//20):
                    base = (i, j)
                    up = (i-1, j) if i != 0 else (self.height-1, j)
                    down = (i+1, j) if i != self.height-1 else (0, j)
                    left = (i, j-1) if j != 0 else (i, self.width-1)
                    right = (i, j+1) if j != self.width-1 else (i, 0)
                    cond = self.rule.calculate_res([
                        self.cells[base[0]][base[1]],
                        self.cells[up[0]][up[1]],
                        self.cells[down[0]][down[1]],
                        self.cells[left[0]][left[1]],
                        self.cells[right[0]][right[1]]
                    ])
                    if cond:
                        pygame.draw.rect(self.screen, self.BLACK, \
(j * 20, i * 20, 20, 20))
                        newcells[i][j] = 1
                    if not cond and self.cells[i][j] == 1:
                        pygame.draw.rect(self.screen, self.WHITE, \
(j * 20 + 1, i * 20 + 1, 19, 19))
                        newcells[i][j] = 0
                    pygame.display.flip()
            self.cells = newcells.copy()

            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    quit()
```

4 Результат работы программы

При запуске программы пользователю предлагается через консоль ввести правило и размеры поля, а также выбрать способ его заполнения (рис. 2). При выборе ручного заполнения появляется пустое поле, в котором можно отмечать необходимые клетки (рис. 3). При выборе автоматического заполнения появляется поле, в котором рандомным образом расставлены чёрные клетки (рис. 5). После нажатия клавиши Enter и в том, и в другом случае автомат начинает свою работу, в ходе которое клетки умирают либо восстанавливаются (рис. 4, 6).

```
Enter rule number:
487378
Enter field size:
25 25
Choose way to fill the field:
1)manually
2)random
1
```

Рис. 2: Пользовательский ввод в консоль

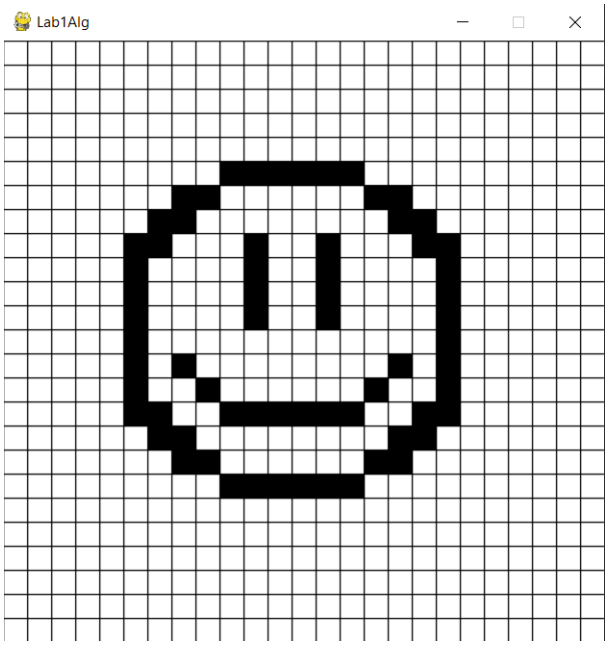

Рис. 3: Ручное заполнение поля

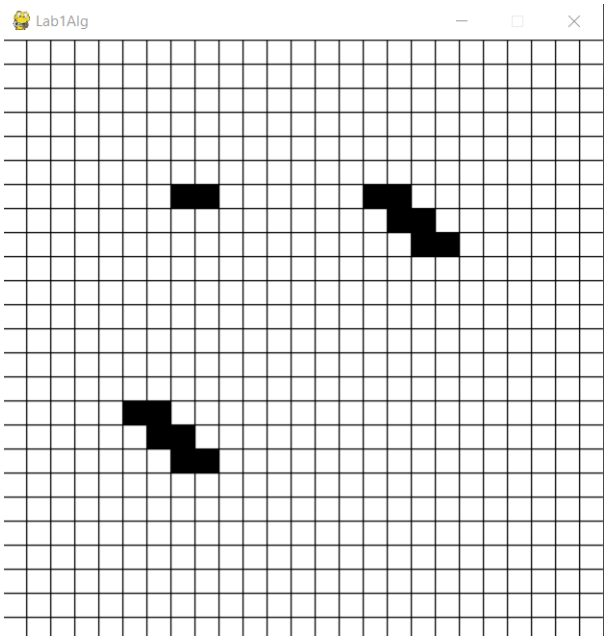

Рис. 4: Результат работы автомата при ручном заполнении

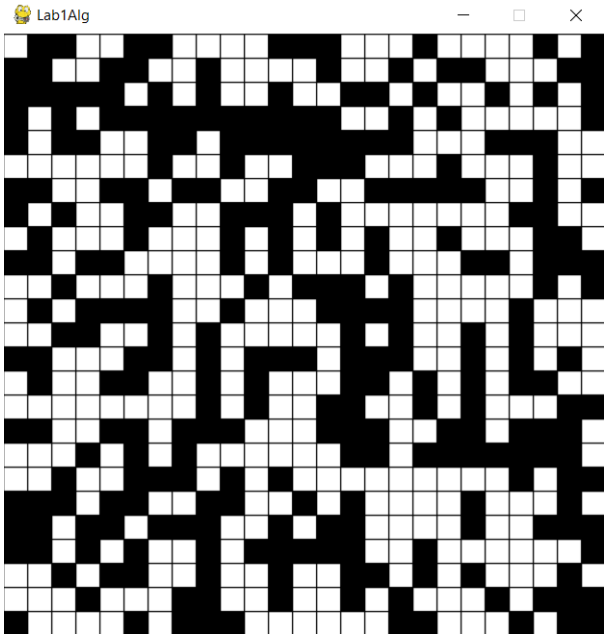

Рис. 5: Автоматическое заполнение поля

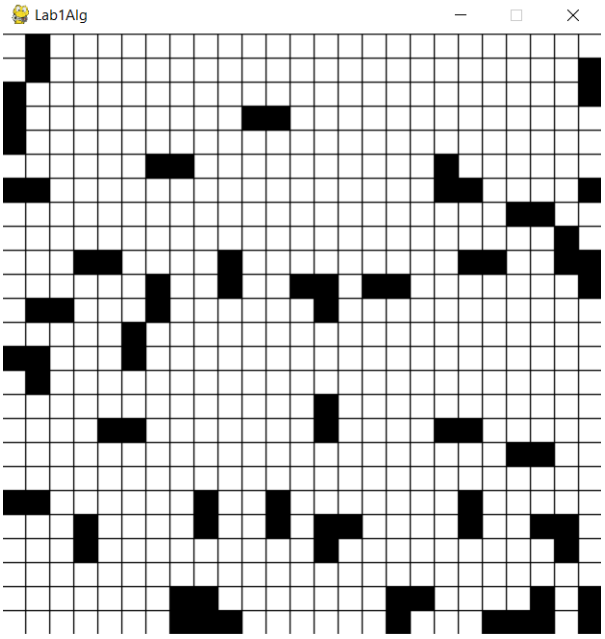

Рис. 6: Результат работы автомата при автоматическом заполнении

5 Анализ клеточного автомата

5.1 Стабильные сочетания

Некоторые комбинации клеток при заданном правиле оказываются незменяемыми. К ним относятся:

1. Полностью «мёртвое» поле

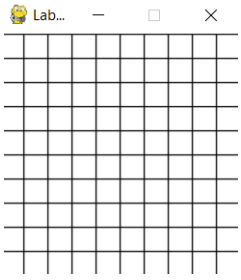

Рис. 7: Мёртвое поле

2. Любые парные «живые» клетки (по вертикали и горизонтали)

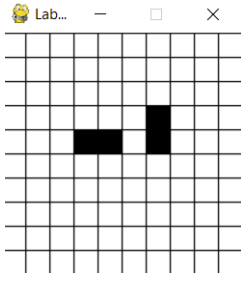

Рис. 8: Соседние пары клеток

3. Любые углы, образованные тройкой «живых» клеток, кроме правого нижнего

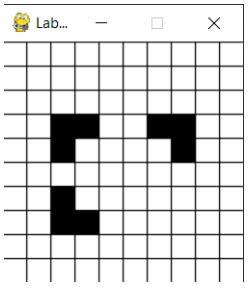

Рис. 9: Углы

4. Четвёрки «живых» клеток, образующие букву «Т», ориентированную наверх, вниз или налево

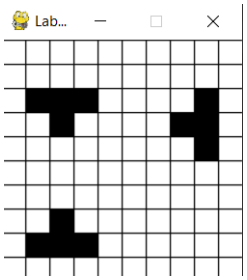

Рис. 10: Буквы «T»

5.2 Сочетания, «умирающие» целиком

В некоторых случаях все клетки комбинации переходят из состояния «1» в состояние «0». Ниже перечислены такие сочетания:

1. Полностью «живое» поле

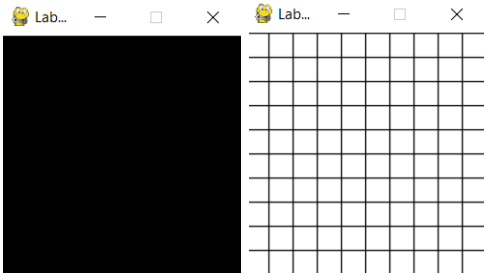

Рис. 11: Живое поле

2. Одиночные «живые» клетки

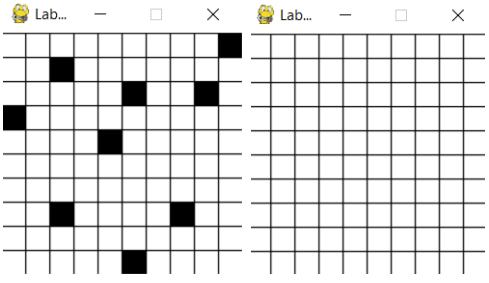

Рис. 12: Одиночные клетки

3. Любые линии из более чем двух «живых» клеток

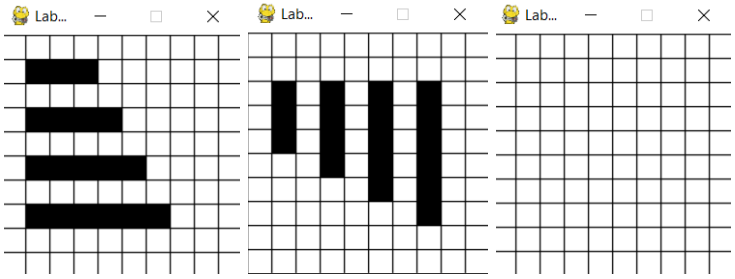

Рис. 13: Линии

4. Тройка клеток, образующая правый нижний угол

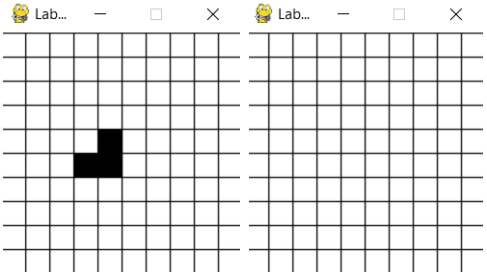

Рис. 14: Правый нижний угол

5. Четвёрка «живых» клеток, образующая букву «Т», ориентированную направо

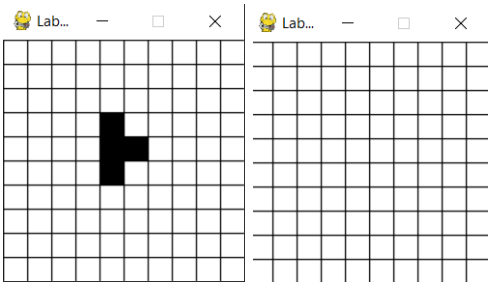

Рис. 15: Правая буква «Т»

5.3 Сочетания, при которых клетки «оживает»

Существует три комбинации, при которых клетка переходит из состояния «0» в состояние «1»:

1. Правый треугольник - «оживляет» центральную клетку

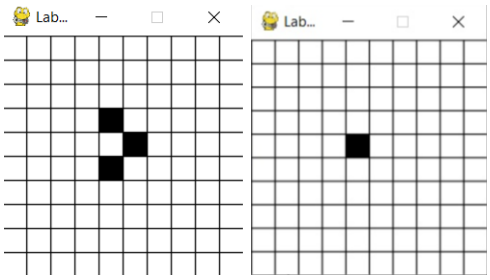

Рис. 16: Правый треугольник

2. Левый треугольник - «оживляет» центральную клетку

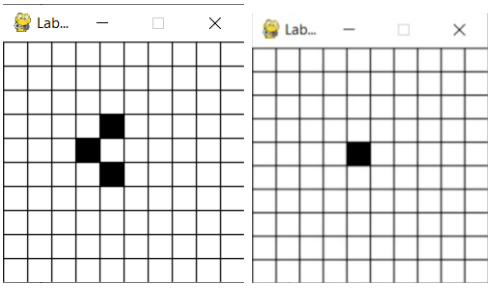

Рис. 17: Левый треугольник

3. Крест - «оживляет» центральную клетку

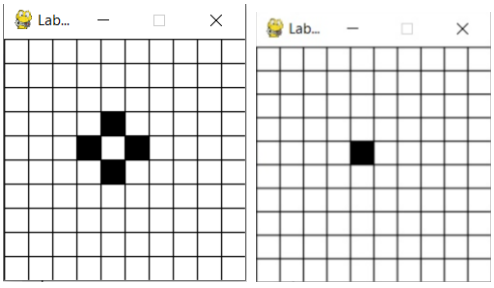

Рис. 18: Крест

5.4 Циклы

Ряд комбинаций вызывает простые циклические изменения состояний автомата. Ниже перечисленные некоторые из них:

1. Чередование верхней и нижней клеток

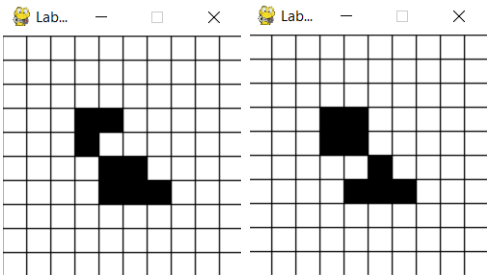

Рис. 19: Циклическая комбинация №1

2. Мигание центральной клетки

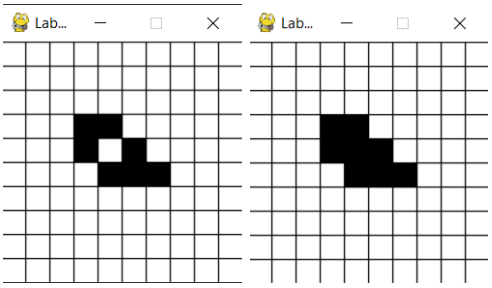

Рис. 20: Циклическая комбинация №2

3. Мигание левой клетки

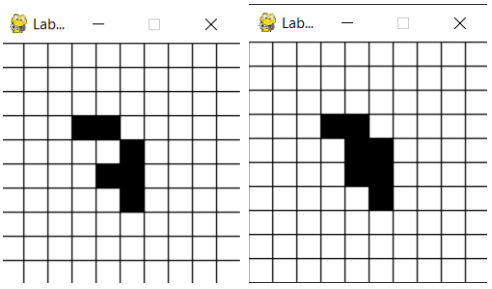

Рис. 21: Циклическая комбинация №3

4. Чередование левой и правой клеток

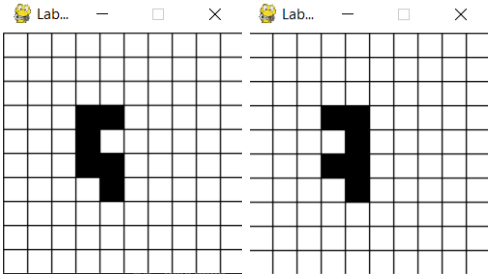

Рис. 22: Циклическая комбинация №4

Таким образом, в большинстве случаев автомат приходит к стабильному состоянию со статичной структурой, однако при некоторых комбинациях живых клеток возможны простые периодические структуры (циклы). Поэтому автомат с заданным правилом 487378_{10} можно отнести ко **второму классу** по Вольфраму.

Заключение

В результате данной работы было написано приложение, реализующее клеточный автомат с окрестностью фон Неймана и тороидальными граничными условиями. Реализованный клеточный автомат с функцией 487378_{10} был проанализирован, после чего были выявлены регулярные структуры - как статичные, так и периодические, что позволило определить его принадлежность ко второму классу по классификации Стивена Вольфрама.

Достоинства программы

1. Графическая визуализация клеточного автомата.
2. Возможность ввода произвольной десятичноё функции, двоичное представление которой не превышает размер 32 бита.

Недостатки программы

1. Отсутствует возможность перезапуска автомата после прихода к статичному состоянию - для этого требуется перезапуск всей программы.
2. При проверке состояния клеток в каждый момент времени применяются вложенные циклы, что не является эффективной реализацией: при большом размере поля программа будет работать долго.

Список литературы

[1] Вольфрам Стивен A New Kind of Science. - 2-е изд. - Лондон: Wolfram Media, 2002. - 1197 с.

[2] Востров А.В. Курс лекций по теории алгоритмов. https://tema.spbstu.ru/t_algorithm/ (Дата обращения: 19.06.2022)

[3] Ершов Н.М. Клеточные автоматы. - 1-е изд. - М.: 2011. - 16 с.