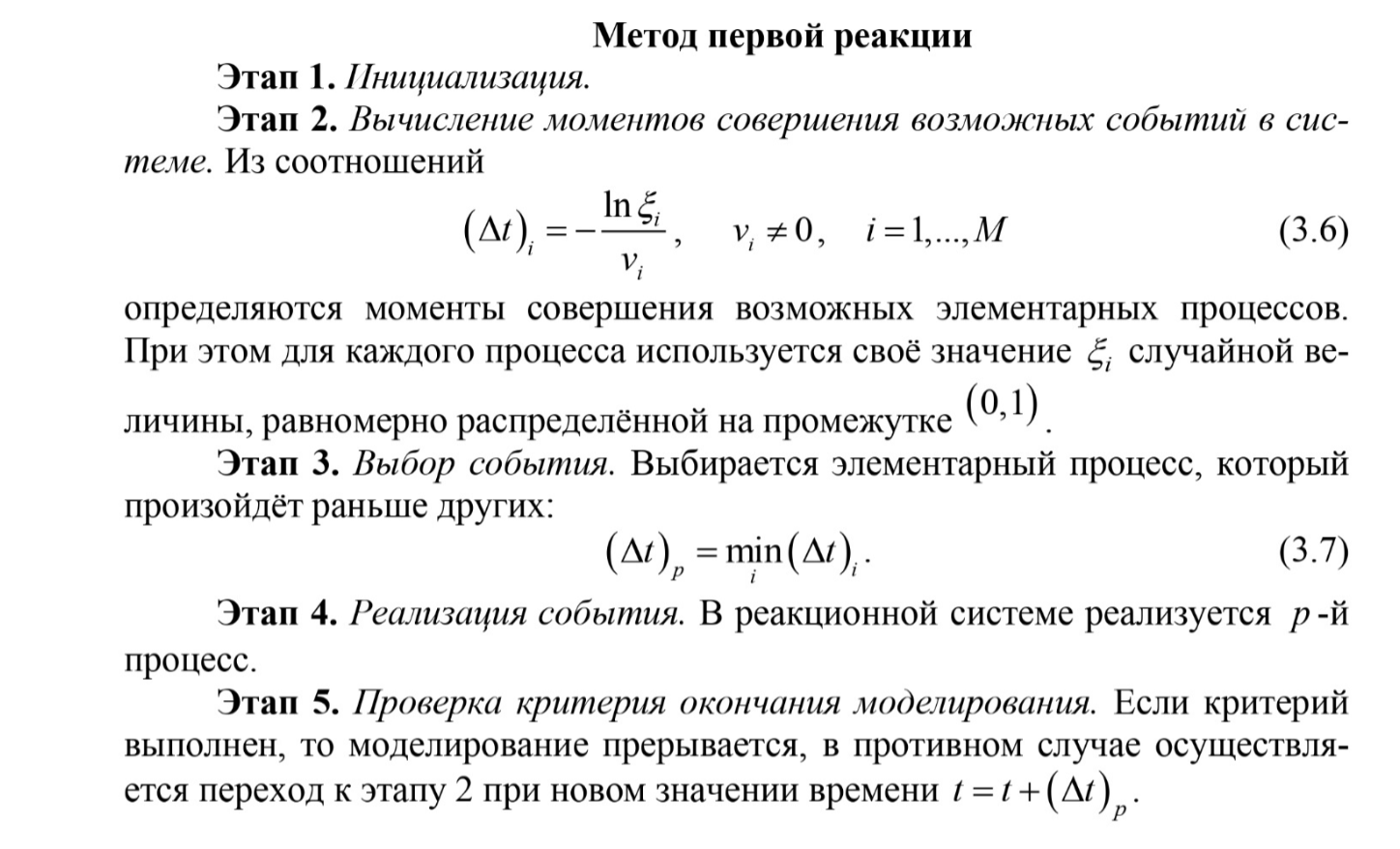
**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ 5**

Суть метода первой реакции заключается в определении моментов со- вершения всех возможных элементарных процессов в системе; далее реали- зуется то событие, которое должно произойти первым.



Метод первой реакции не сложен, но достаточно трудоёмок, поскольку при каждом изменении состояния фрагмента приходится многократно генерировать значения действительной случайной величины, равномерно распределённой на промежутке (0,1), и вычислять значения натуральных логарифмов, что требует много счётного времени.

**Код**

**Генерируем данные**

#include <iostream>

#include <vector>

#include <cstdlib>

#include <ctime>

#include <algorithm>

#include <chrono>

#include <thread>

#include <fstream>

#include <cmath>

const int GRID\_SIZE = 20; // Размер сетки

const int SLEEP\_TIME = 70;

const std::string FILENAME = "output.txt";

const std::string LIVING\_CELL = "O ";

const std::string DEAD\_CELL = ". ";

const std::string LIVING\_CELL\_CMD = "\u2591";

const std::string DEAD\_CELL\_CMD = "\u2588";

const std::string RESET\_COLOUR = "\033[0m"; // Сброс цвета

const std::string LIGHT\_GREEN = "\033[1;32m"; // Светло-зеленый

enum class Event {

MOV = 1,

DEATH = 2,

DIV = 3

};

// Функция для вывода текущего состояния сетки

void print\_grid(const std::vector<std::vector<int>>& grid, std::ofstream& output) {

for (int i = 0; i < GRID\_SIZE; ++i) {

for (int j = 0; j < GRID\_SIZE; ++j) {

output << (grid[i][j] ? LIVING\_CELL : DEAD\_CELL);

}

output << '\n';

}

output << "\n\n";

}

void print\_grid(const std::vector<std::vector<int>>& grid) {

for (int i = 0; i < GRID\_SIZE; ++i) {

for (int j = 0; j < GRID\_SIZE; ++j) {

if (grid[i][j] == 1)

std::cout << LIVING\_CELL\_CMD;

else

std::cout << LIGHT\_GREEN << DEAD\_CELL\_CMD << RESET\_COLOUR;

}

std::cout << '\n';

}

}

// Функция для получения списка соседей клетки

std::vector<std::pair<int, int>> get\_neighbors(int x, int y) {

std::vector<std::pair<int, int>> neighbors = {

{x - 1, y}, {x + 1, y}, {x, y - 1}, {x, y + 1}

};

// Фильтрация соседей, чтобы они были в пределах сетки

neighbors.erase(std::remove\_if(neighbors.begin(), neighbors.end(),

[](std::pair<int, int> p) {

return p.first < 0 || p.first >= GRID\_SIZE || p.second < 0 || p.second >= GRID\_SIZE;

}), neighbors.end());

return neighbors;

}

// ф-ция для генерации времени по методу первой реакции

double generate\_time(double v){

double e = static\_cast<double>(rand()) / RAND\_MAX;

return - std::log(e) / v;

}

// Функция для обработки событий

std::vector<std::vector<int>> process\_event(std::vector<std::vector<int>>& grid) {

std::vector<std::vector<int>> new\_grid = grid;

for (int i = 0; i < GRID\_SIZE; ++i) {

for (int j = 0; j < GRID\_SIZE; ++j) {

if (grid[i][j] == 1) {

auto neighbors = get\_neighbors(i, j);

std::vector<std::pair<int, int>> free\_neighbors;

int live\_neighbors = 0;

double t\_mov = generate\_time(static\_cast<double>(static\_cast<int>(Event::MOV)));

double t\_death = generate\_time(static\_cast<double>(static\_cast<int>(Event::DEATH)));

double t\_div = generate\_time(static\_cast<double>(static\_cast<int>(Event::DIV)));

double min\_time = std::min({t\_mov, t\_death, t\_div});

for (const auto& neighbor : neighbors) {

if (grid[neighbor.first][neighbor.second] == 1)

++live\_neighbors;

else

free\_neighbors.push\_back(neighbor);

}

if (min\_time == t\_mov) {

if (!free\_neighbors.empty()) {

auto direction = free\_neighbors[rand() % free\_neighbors.size()];

new\_grid[i][j] = 0;

new\_grid[direction.first][direction.second] = 1;

}

}

else if (min\_time == t\_death && live\_neighbors > 3) {

new\_grid[i][j] = 0;

}

else if (min\_time == t\_div && live\_neighbors > 0) {

if (!free\_neighbors.empty()) {

auto direction = free\_neighbors[rand() % free\_neighbors.size()];

//if (new\_grid[direction.first][direction.second] == 0)

new\_grid[direction.first][direction.second] = 1;

}

}

}

}

}

return new\_grid;

}

// Проверка, есть ли живые клетки на поле

bool has\_alive\_cells(const std::vector<std::vector<int>>& grid) {

for (int i = 0; i < GRID\_SIZE; ++i) {

for (int j = 0; j < GRID\_SIZE; ++j) {

if (grid[i][j] == 1) {

return true;

}

}

}

return false;

}

void choose\_action(int ch){

// Создаём сетку 19x20 и размещаем начальные клетки

std::vector<std::vector<int>> grid(GRID\_SIZE, std::vector<int>(GRID\_SIZE, 0));

std::vector<std::pair<int, int>> initial\_positions = {{5, 5}, {15, 5}, {5, 15}, {15, 15}};

for (const auto& pos : initial\_positions) {

grid[pos.first][pos.second] = 1; // 1 означает, что клетка жива

}

// Основной цикл

unsigned long int steps = 0;

switch(ch){

case 1:

for (int i = 0; i < GRID\_SIZE \* 100; ++i) {

if (!has\_alive\_cells(grid)) {

std::cout << "All the cells died out after " << steps << " steps.\n";

break;

}

std::cout << "\x1b[2J\nStep " << steps << ":\n";

print\_grid(grid); // Вывод текущего состояния сетки

grid = process\_event(grid); // Обработка событий для всех клеток

++steps;

std::this\_thread::sleep\_for(std::chrono::milliseconds(SLEEP\_TIME));

//std::cout << "Press Enter to continue...\n";

//std::cin.get(); // Ожидание для продолжения шагов

}

break;

case 2:{

std::ofstream output(FILENAME);//создаем поток вывода в файл

if (!output.is\_open()){//открываем поток и если он не открылся выводим ошибку и убиваем прогу

std::cerr << "Error in opening file " << FILENAME << '\n';

exit(0);

}

for (int i = 0; i < GRID\_SIZE \* 100; ++i) {

if (!has\_alive\_cells(grid)) {

std::cout << "All the cells died out after " << steps << " steps.\n";

break;

}

output << "Step " << steps << ":\n";

print\_grid(grid, output); // Вывод текущего состояния сетки

grid = process\_event(grid); // Обработка событий для всех клеток

++steps;

}

output.close();

std::cout << "Data has been saved to " << FILENAME <<'\n';

break;

}

case 3:

while (has\_alive\_cells(grid)){

std::cout << "Step " << steps << ":\n";

print\_grid(grid);

grid = process\_event(grid);

++steps;

//std::this\_thread::sleep\_for(std::chrono::milliseconds(SLEEP\_TIME));

//std::cout << "Press Enter to continue...\n";

//std::cin.get(); // Ожидание для продолжения шагов

}

std::cout << "All the cells died out after " << steps << " steps.\n";

break;

}

}

int main() {

srand(time(0));

std::cout << "To print all data to terminal, enter 1, to print data to file, enter 2(these both variants are executed until";

std::cout << "the max number of iterations is reached);\nIf you want the program work until all cells die, enter 3, data is displayed to terminal)\n";

int ch;

bool right\_choice = false;

while(std::cin >> ch){

switch(ch) {

case 1:

case 2:

case 3:

right\_choice = true;

break;

default:

std::cout << "\nWrong input, press 1, 2 or 3\n";

}

if (right\_choice) break;

}

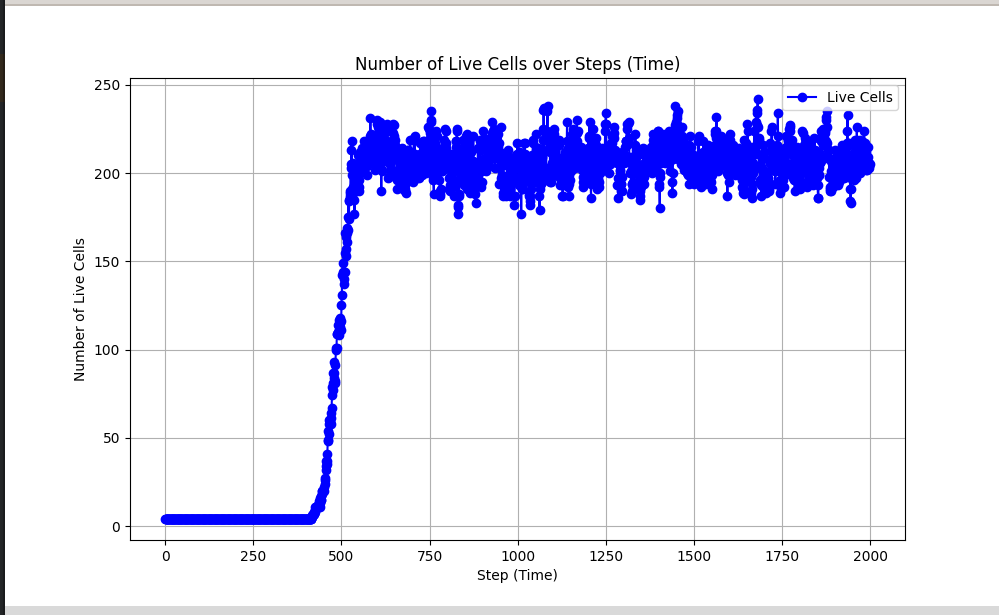
choose\_action(ch);

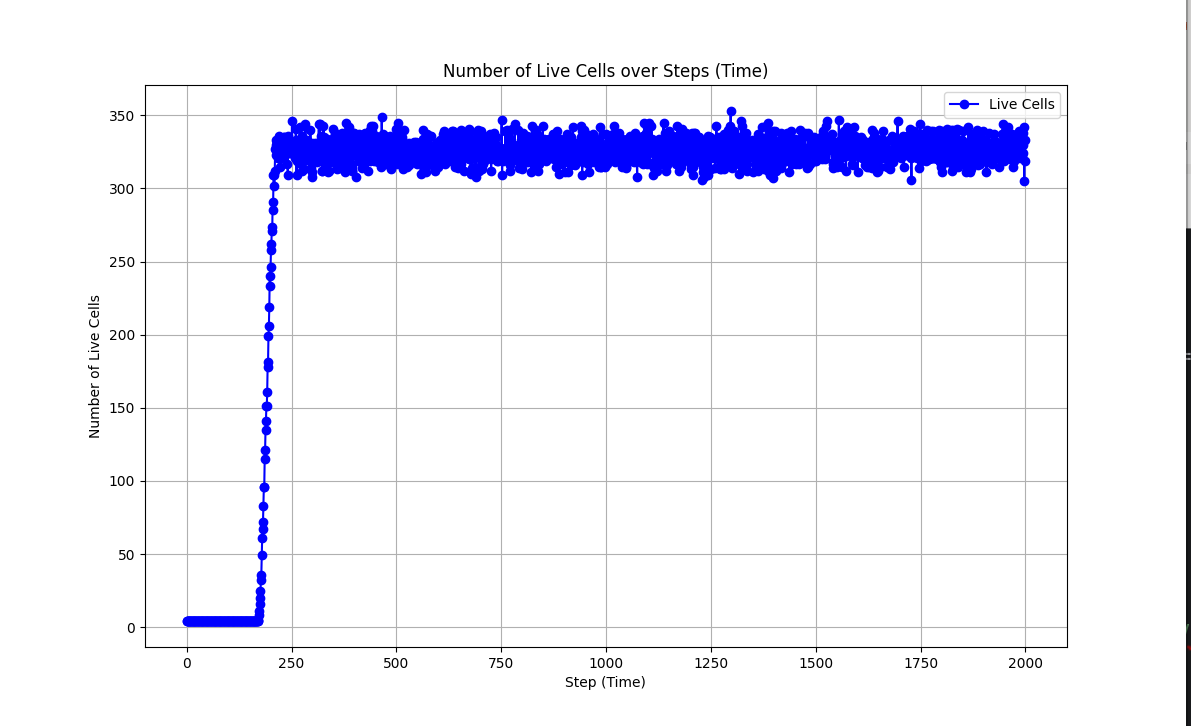
return 0;

}

**Строим график**

import matplotlib.pyplot as plt  
  
# Функция для парсинга файла  
def parse\_file(filename):  
 steps = []  
 live\_cells = []  
  
 with open(filename, 'r') as file:  
 step = 0  
 alive\_count = 0  
 in\_step = False  
  
 for line in file:  
 if line.startswith("Step"): # Определяем начало нового шага  
 if in\_step: # Если это не первый шаг, сохраняем данные предыдущего шага  
 steps.append(step)  
 live\_cells.append(alive\_count)  
  
 step = int(line.split()[1].strip(':')) # Получаем номер шага  
 alive\_count = 0 # Обнуляем счётчик живых клеток для нового шага  
 in\_step = True  
 elif '.' in line or 'O' in line: # Считываем состояние клеток в строке  
 alive\_count += line.count('O') # Считаем живые клетки ('O')  
  
 # Добавляем последний шаг в данные  
 if in\_step:  
 steps.append(step)  
 live\_cells.append(alive\_count)  
  
 return steps, live\_cells  
  
# Функция для построения графика  
def plot\_data(steps, live\_cells):  
 plt.figure(figsize=(10, 6))  
 plt.plot(steps, live\_cells, marker='o', color='b', linestyle='-', label='Live Cells')  
 plt.title('Number of Live Cells over Steps (Time)')  
 plt.xlabel('Step (Time)')  
 plt.ylabel('Number of Live Cells')  
 plt.grid(True)  
 plt.legend()  
 plt.show()  
  
# Основная функция  
if \_\_name\_\_ == '\_\_main\_\_':  
 filename = '/home/buzzismaloy/University\_shit/7th-term/monte-karlo/fifth-lab/output.txt' # Укажите путь к вашему файлу  
 steps, live\_cells = parse\_file(filename)  
 plot\_data(steps, live\_cells)

Рис.1. Зависимость количества живых клеток от времени(шаг)

  
Рис.2. Зависимость количества живых клеток от времени(шаг)

**Выводы**

В результате данной работы был реализован метод первой реакции. Программа в

реальном времени показывает эволюцию системы заданую простыми правилами.