МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

Кафедра математического моделирования

**КУРСОВАЯ РАБОТА**

**КЛЕТОЧНО-АВТОМАТНОЕ МОДЕЛИРОВАНИЕ ПРОЦЕССА ТЕЧЕНИЯ ЖИДКОСТИ ПРИ НАЛИЧИИ ПРЕПЯТСТВИЙ И ПРИМЕСИ**

Работу выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ М.А.Пузырёв

Направление подготовки 01.03.02 Прикладная математика и информатика

Направленность (профиль) Математическое моделирование и вычислительная математика: Математическое моделирование

Научный руководитель

канд. физ.-мат. наук, доц.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С.Е. Рубцов

Нормоконтролер

канд. физ.-мат. наук, доц.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С.Е. Рубцов

Краснодар

2023

РЕФЕРАТ

Курсовая работа 42 с., 10 рис., 3 источника, 1 прил.

МАТЕМАТИЧЕСКОЕ МОДЕЛИРОВАНИЕ, КЛЕТОЧНЫЕ АВТОМАТЫ, GAS-LATTICE МОДЕЛИРОВАНИЕ, МОДЕЛЬ FHP-I

Объектом работы является моделирование процесса течения жидкости при наличии препятствий и примесей с помощью клеточный автоматов.

Цель работы – исследование процесса течения жидкости при наличии препятствий и распространение примесей в сплошной среде с помощью клеточных автоматов.

Для программной реализации была выбрана среда разработки Visual Studio. В результате написана программа на языке С++, моделирующая процесс течения жидкости в плоскости с препятствиями и примесями; средствами библиотеки SFML этот процесс был визуализирован.

СОДЕРЖАНИЕ

Введение4

1 Клеточный автомат5

1.1 Примеры клеточных автоматов5

1.2 Клеточный автомат6

1.3 Клеточно-автоматная модель6

2 Модель FHP-I7

3 Программная реализация11

3.1 Постановка задачи13

3.2 Инструментарий11

3.3 Структура программы и описание алгоритма12

3.4 Результаты работы13

3.4.1 Течение жидкости13

3.4.1 Распространение примесей15

Заключение20

Список использованных источников21

Приложение22

# **ВВЕДЕНИЕ**

При решении целого ряда научных и практических задач возникает необходимость исследования процессов течения жидкости. В частности, при описании распространения загрязняющих веществ в водной среде используются различные типы математических моделей в зависимости от пространственных масштабов изучаемых явлений и целей исследования. Моделирование миграции загрязнителей позволяет решать проблемы блокирования распространения загрязняющих веществ, оценки локального состояния окружающей среды и прогнозирования последствий воздействия на экосистему.  
 Основным математическим инструментом описания указанных процессов в сплошной среде являются уравнения Навье-Стокса, решение которых производится, как правило, методами математической физики либо методами вычислительной математики. Альтернативным подходом к изучению пространственной динамики является так называемое клеточно-автоматное (КА) моделирование газовой динамики (Gas-Lattice). При этом соответствие Gas-Lattice моделей уравнениям Навье-Стокса строго математически доказано.

В данной курсовой работе будет рассмотрена программная реализация клеточного автомата, моделирующего процесс течения жидкости при наличии препятствий и примесей.

Цель данной работы является исследование процесса распространения примесей с течением времени.

Итог работы – визуализированный процесс течения и распространения.

**1 Клеточные автоматы**

* 1. **Примеры клеточных автоматов**

В настоящее время известно множество самых разнообразных клеточных автоматов. Самым известным примером клеточного автомата является игра «Жизнь». Она была реализована в 1970-м году британским математиком Джоном Конвеем по мотивам задачи, поставленной в 1940-м году другим великим математиком, Джоном фон Нейманом. Коротко задача формулировалась так: придумать машину, которая в рамках своей деятельности сможет воспроизводить саму себя (добиться полноты по Тьюрингу).

Игра представляла собой плоскость, заполненную квадратными ячейками. Каждая ячейка-клетка могла находиться всего в 2-х состояниях:

живой или мёртвой. Начальную конфигурацию клеток задаёт игрок. В классическом варианте игры все клетки подчиняются трём правилам:

* Живая клетка с 2-мя или 3-мя соседями выживает
* Клетка с другим количеством соседей умирает
* При наличии ровно 3-х живых соседей в клетке появляется жизнь

Всего три простых правила позволили игре «Жизнь» и её модификациям стать инструментом для моделирования самых разнообразных природных, математических и социальных процессов. К примеру, рождение и смерть клеток подобны процессу возникновения и исчезновения нейронных импульсов, эволюция клеточного автомата имеет внешнее сходство с развитием популяций примитивных организмов, например бактерий, а также игра «Жизнь» используется для анализа «явлений переноса» - диффузии, вязкости, теплопроводности.

Но самым главным свойством придуманного Конвеем клеточного автомата является самовоспроизведение, т.е. в игре «Жизнь» можно было воссоздать её же, что означает что она является полной по Тьюрингу.

**1.2 Клеточный автомат**

Клеточный автомат определяется множеством клеток, заполняющих дискретное пространство, расстояние между центрами клеток равно d. Каждой клетке поставлен в соответствие конечный автомат, входами которого являются выходы клеток, расположенных от него не далее чем на расстоянии r. Все конечные автоматы одинаковы, они определяются тремя значениями {Q, S, δ}, где Q – множество входных состояний, S – множество внутренних состояний, δ:Q × S → S – функция переходов, значение которой определяет новое внутреннее состояние элементарного автомата. По способам перехода клеток в новое состояние различают синхронные и асинхронные функции переходов, а также детерминированные и стохастические. Итеративный процесс переходов из состояния в состояние называется эволюцией КА.

**1.3 Клеточно-автоматная модель**

При моделировании потока субстанции частицам, характеризующим наличие в каждой точке рассматриваемого пространства единицы массы, придаётся единичный вектор скорости. Частицы движутся в направлении, заданном вектором скорости и меняют направление своего движения при столкновениях между собой или с препятствиями. Правила перемещения и столкновения частиц составлены таким образом, чтобы обеспечить выполнение законов сохранения массы, импульса и энергии. Набор разработанных правил составляет таблицу переходов элементарного конечного автомата, называемого клеткой. Состояние клетки определяет наличие или отсутствие составляющей скорости в данный момент в данной точке. Поведение каждой клетки определяется булевыми функциями, называемыми микроскопическими уравнениями.

**2 Модель FHP-I**

Классическая клеточно-автоматная модель потока жидкости (FHP-I) представляет собой тройку объектов (W, A, N), где W ={w1, w2, … , wi, …} - множество клеток, заданное их координатами в некотором — дискретном пространстве. Каждой клетке поставлен в соответствие конечный автомат А.

Для каждой клетки w ϵ W определено некоторое упорядоченное множество

N(w) = {Ni(w): N0(w) = w, Ni(w) ϵ W & d(w, Ni(w)) = 1, (*i* = 1,2,…,6), элементы которого находятся в отношении соседства с клеткой у и называются ее соседями. Каждая клетка является соседом сама себе.

Состояние клетки w ϵ W представлено вектором s(w) с булевыми компонентами s0(w), si(w),…, s6(w). На рисунке 1 изображена клетка w, векторы скорости сi находящихся в ней частиц и ее соседи Ni(w), *i* = 0,1,..., 6.

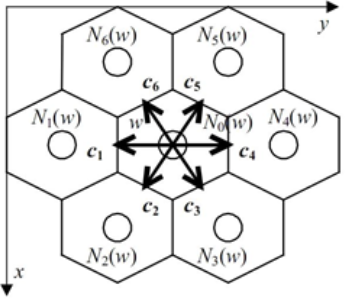


Рисунок 1 – Векторы скорости частиц. Нумерация соседей

Состояние клетки w в каждый момент дискретного времени t однозначно определяется набором находящихся в ней частиц. Вектор скорости i, каждой из них либо направлен в сторону одной из соседних клеток Ni(w) (при *i*=1,...,6) либо равен нулю (при *i* = 0).

На рисунке 2 изображён пример вектора состояния клетки.

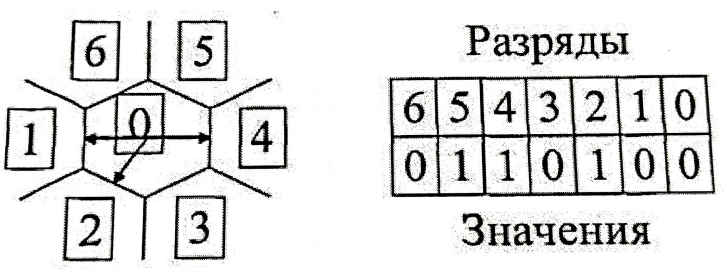


Рисунок 2 – Вектор состояния клетки

Множество состояний s(w) всех клеток w ϵ W в один и тот же момент времени *t* называется глобальным состоянием

σ(t) = {s(w1), s(w2),…, s(wi),…}

клеточного автомата.

Клетки wi ϵ W могут быть одного из трех типов: клетками среды

wср ϵ Wср, в которых выполняются законы сохранения массы и импульса, клетками стенок wст ϵ Wст, в которых выполняется закон сохранения массы, но может нарушаться закон сохранения импульса, клетками источника

wист ϵ Wист в которых могут нарушаться как закон сохранения массы, так и закон сохранения импульса. Множества клеток среды Wср, стенок Wст и источников Wист, попарно не пересекаются (Wср  Wст = Wср  Wист = Wст  Wист = ).

Объединение этих множеств совпадает с множеством всех клеток автомата (Wср Wст Wист = W). Поведение стенок и источников задает граничные условия клеточного автомата.

В модели FНР-I используется клеточный автомат с синхронным режимом функционирования. На каждом такте происходит смена состояний s(t) элементарных автоматов A во всех клетках w ϵ W на состояния

*s*(*t*+1) = *δ*(*s*(*t*)), где *δ*(*s*(*t*)) – функция переходов элементарного автомата A. Клеточный автомат при этом переходит из глобального состояния *σ*(*t*) в новое глобальное состояние *σ*(*t* + 1).

Каждый такт работы клеточного автомата выполняется в две фазы: сдвиг и столкновение. Функция переходов *δ* элементарного автомата A состоит, таким образом, из композиции функций *δ*1 (сдвиг) и

*δ*2 (столкновение): *δ*(*s*) = *δ*2(*δ*1(*s*)). При сдвиге масса и импульс частиц в клетке изменяются, в пределах всего клеточного автомата они сохраняются. В фазе столкновения происходит изменение направления движения частиц согласно некоторым правилам столкновения, не зависящим от состояний соседних клеток, т.е. *δ*2 зависит только от внутреннего состояния своего элементарного автомата. Функция *δ*2 в модели FHP-I содержит как вероятностные, так и детерминированные правила перехода.

В клетке среды функция *δ*2, выбирается такой, чтобы сохранялись масса m(w):

2(si(wср)) = i(wср)

и импульс (w):

2(i(wср)) = i(wср)i(wср)

частиц в клетке.

В клетках, являющихся стенками, частицы «отражаются» в обратном направлении, нарушая при этом закон сохранения импульса. Из-за того, что

количество частиц в клетке не меняется, условия сохранения массы

выполняются. Условие сохранения импульса может нарушаться, так как

меняются направления векторов скорости *c* частиц, но это допускается

граничными условиями. Такое поведение частиц в клетках-стенках моделирует условие нулевой скорости потока на границах препятствий.  
Каждая клетка-источник wист ϵ Wист генерирует частицы со всевозможными направлениями вектора скорости. Из клеток-источников можно создавать различные объекты. Например, установив их в пространстве в одну линию (как правило, у границы клеточного массива), можно получить источник равномерного потока частиц. Естественно, при генерации новых частиц ни масса *m*(wист), ни импульс *p*(wист) не сохраняются.

При моделировании потоков практический интерес представляют не столько значение параметров автомата на микроуровне, т.е. масса *m*(w) и скорость I (w) частиц в каждой клетке w ϵ W, сколько осреднённые значения их скоростей и концентраций по некоторой окрестности *Av*(w), которая включает все клетки wj ϵ *W*, удалённые от клетки w не более чем на некоторую величину *r*, называемую радиусом осреднения.

Осреднённая скорость вычисляется по формуле

(w) = ii ,

где k – количество клеток, попадающих в окрестность осреднения *Av*(w), ci – единичный вектор скорости, соответствующий *i*-му разряду вектора состояния *s*(w*j*), а si – значение *i*-го разряда вектора состояния *s*(w*j*) клетки w*j* ϵ *Av*(w).

Осреднённая концентрация частиц подсчитывается в той же окрестности *Av*(w) следующим образом:

(w) = i

**3 Программная реализация**

**3.1 Постановка задачи**

Моделирование при помощи гексагональных клеточных автоматов имеет много преимуществ, но также имеет и свои недостатки. Главным недостатком является сложность программной реализации. Однако многие физико-химические процессы очень удобно моделировать используя гексагональные сетки, например, для решения задач гидродинамики, газодинамики и диффузии.

Гидродинамика – раздел физики сплошных сред и гидроаэродинамики, который изучает движение идеальных и реальных жидкостей и газов, а также их силовое взаимодействие с твёрдыми телами. В данной работе рассматривается закрытая область с некоторыми начальными источниками жидкости, в области действует сила тяжести. В области могут встречаться различные препятствия, поэтому одной из задач является необходимость пронаблюдать распространение жидкости в пространстве с течением времени.

**3.2 Инструментарий**

Программа написана как консольное приложение Windows в среде разработки Visual Studio. В качестве языка программирования был выбран C++. Выбор ЯП не играл существенной роли в реализации программы, так как в программе не использовались особенности, которым владеет именно C++. Все функции и методы были написаны без использования сторонних расширений, библиотек или программного обеспечения.

Визуализация программы не требовала сложного графического представления, поэтому я выбрал библиотеку SFML, так как она требует большой производительности и довольно проста в обращении.

**3.3 Структура программы и описание алгоритма**

Структура программы выглядит следующим образом.

В блоке “Функции и процедуры” находятся все фукнции и процедуры для работы с вычислениями и графикой. Процедуры RenderWindow , Event и drawGrid позволяют инициализировать окно заданных размеров, а также проверять состояние окна на каждой иттерации и заполнять его новыми данными. Имеются 3 функции с различными формами препятствий: placeObstaclesBorders– задаёт границы мира по краям окна, placeObstacles1 - строит цельный прямоугольник, placeObstacles2 – строит отрезок шириной в 3 клетки, расположенный под углом.

Самыми важными для реализации программы являются функции: initializeGrid – создаёт пустую клеточную решётку, где каждая клетка представляет собой гексагон, updateCellsLiquid – представляет собой программную реализацию работы данного клеточного автомата, вызывается на каждой иттерации и выдаёт новые координаты для каждой клетки.

В основной программе происходит инициализация основных функций и процедур, а также добавление начальных условий источников воды, считывание количества предельных иттераций от пользователя и запуск самой визуализации.

Принцип работы программы следующий:

Пользователю предлагается ввести число максимально возможных иттераций - maxcount. После получения данных программа запускает цикл, где на каждом шагу, в том числе и первом, будет выводиться сетка клеточного автомата с новыми значениями клеток. После прохождения максимального количества иттераций программа останавливается и с задержкой закрывает окно вывода.

**3.4 Результаты работы**

**3.4.1 Течение жидкости**

После написания программы, приступаем к её тестированию. Как уже говорилось, программа не имеет интерфейса, поэтому для ввода и вывода данных будет использоваться консоль.

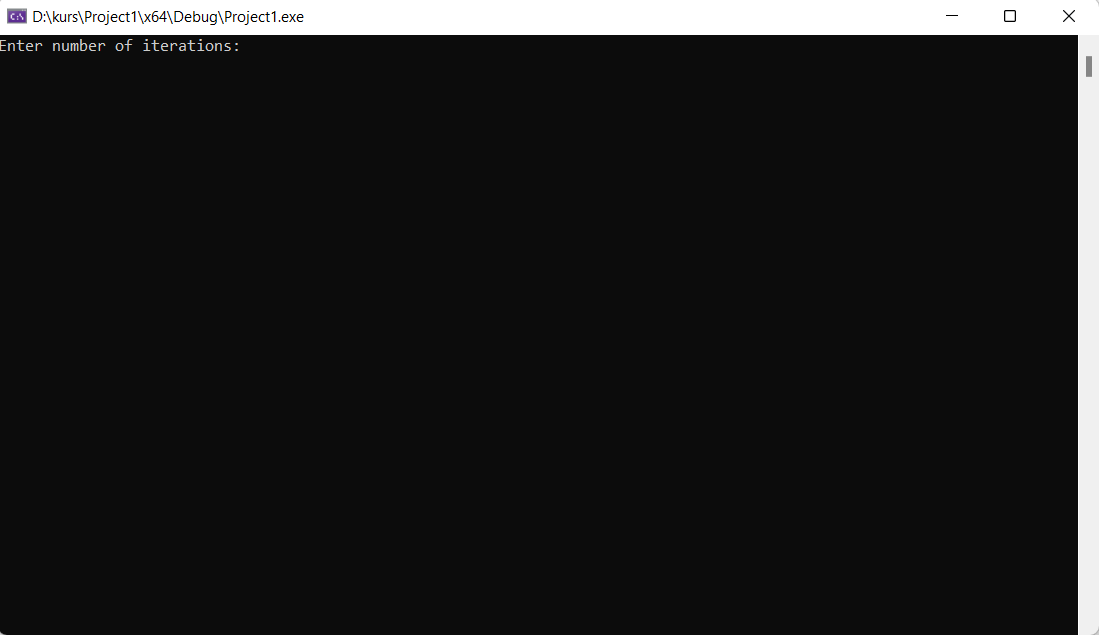


Рисунок 3 – Ввод данных пользователем

В процессе работы программы, на экране будет показываться текущее состояние клеточного автомата.

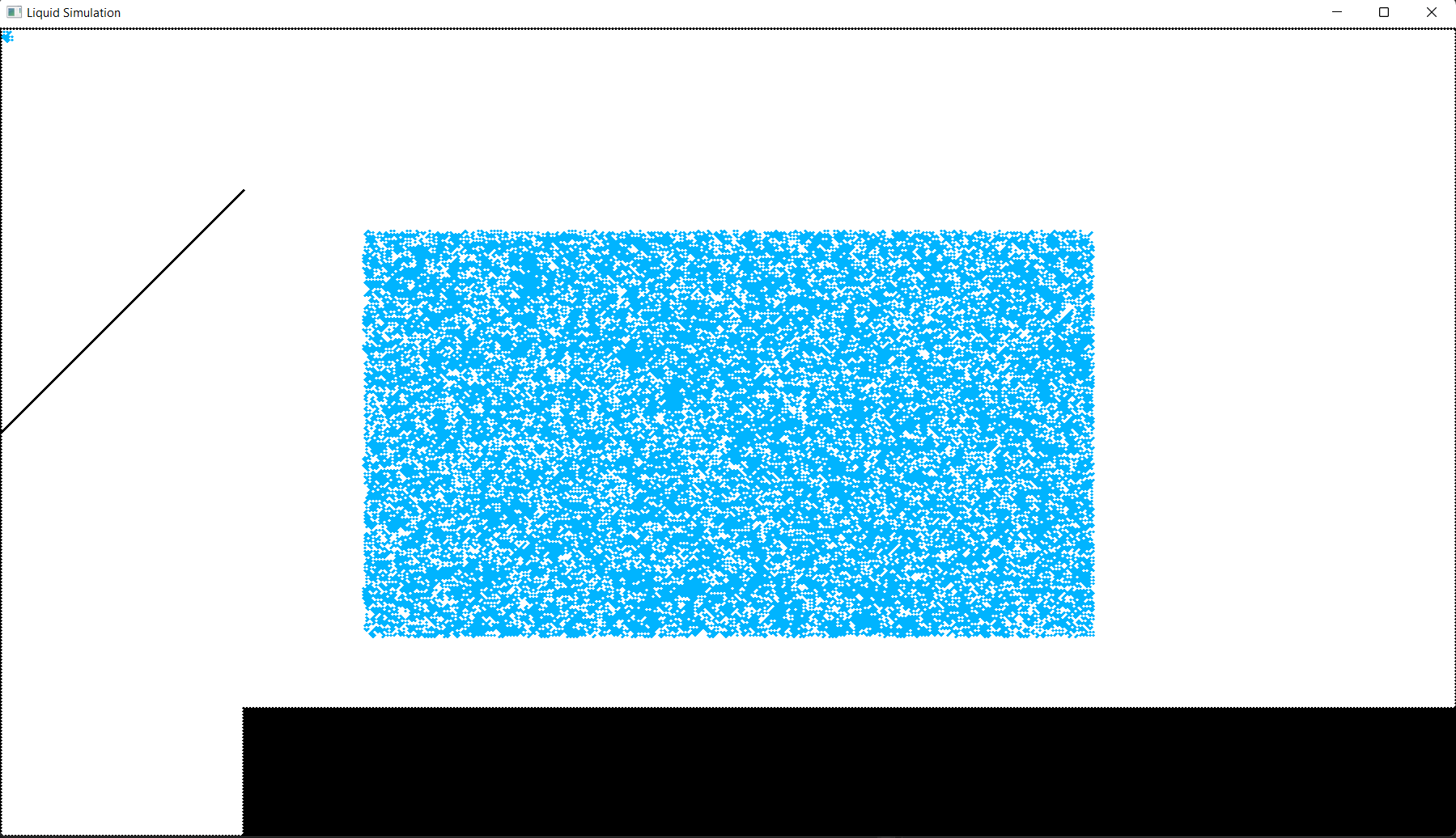


Рисунок 4 – вывод на 1 иттерации

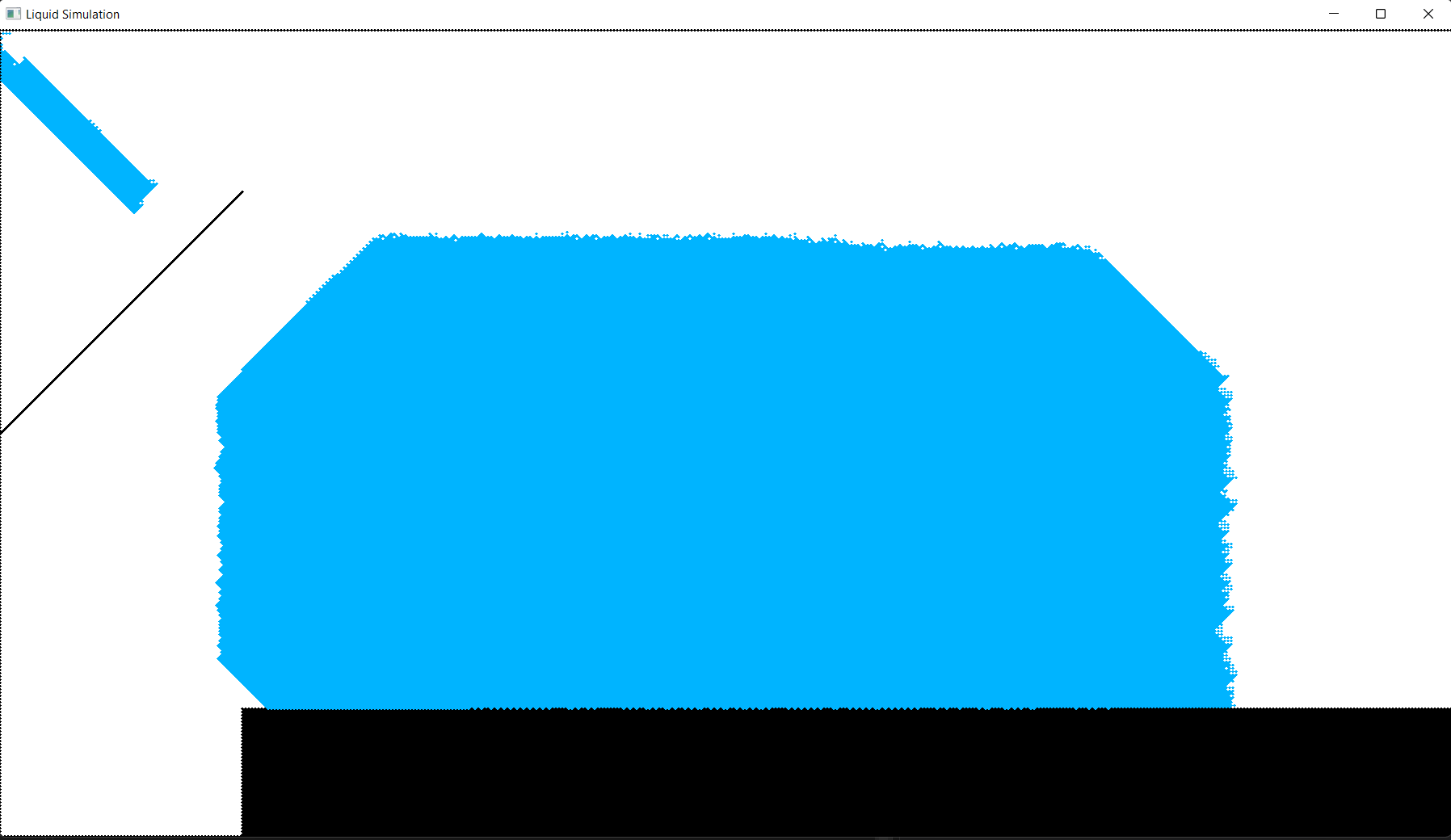


Рисунок 5 – вывод на 92 иттерации

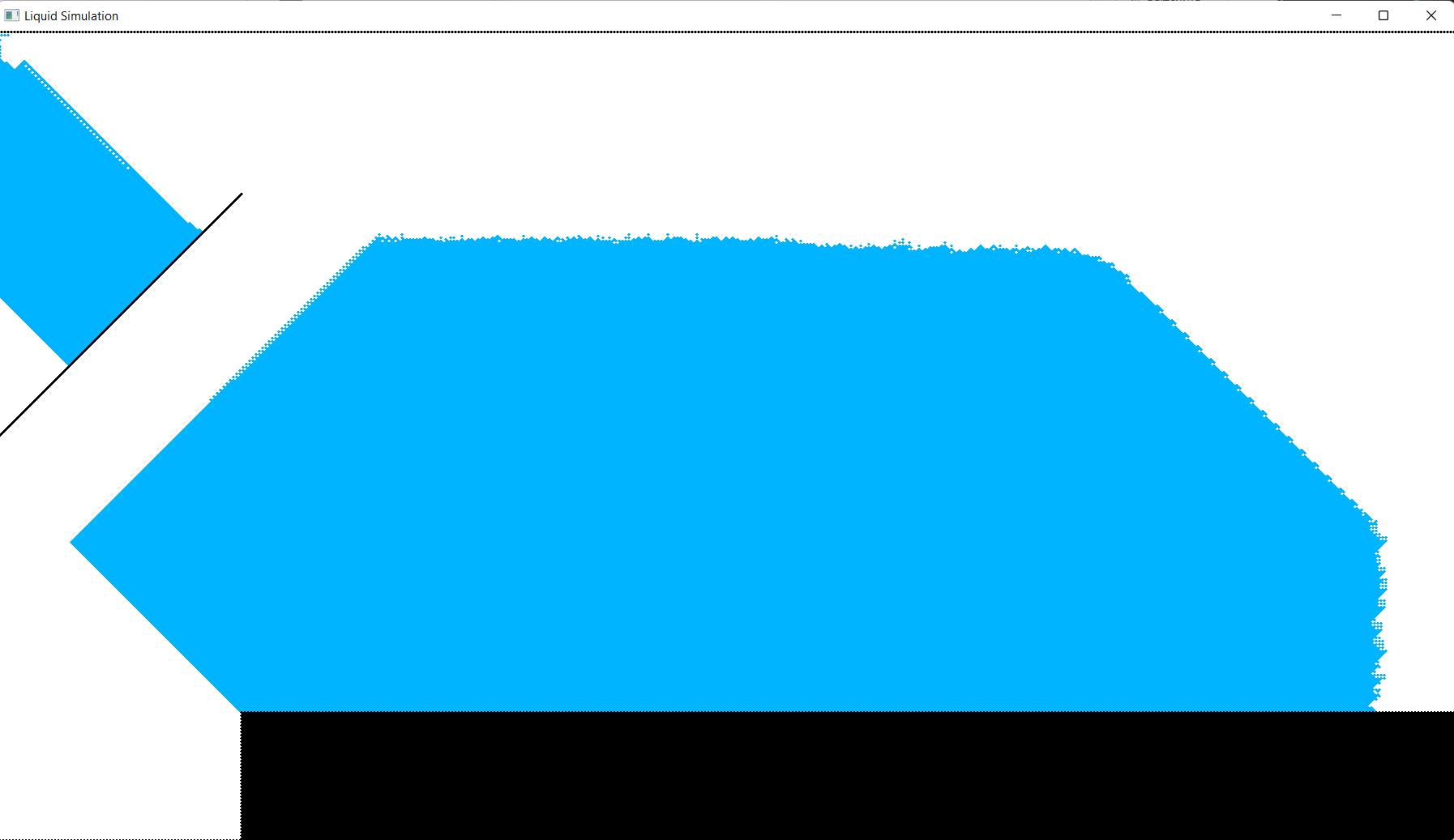


Рисунок 6 – вывод на 182 иттерации

На основе данного эксперимента можно судить о том, что автомат полностью справляется со своей задачей моделирования распространения жидкости в пространстве с препятствиями.

**3.4.2 Распространение примесей**

Для моделирования потока жидкости с примесями был введён новый идентификатор, позволяющий понять: это клетка основного потока или примесь. На частицы примеси действуют те же законы, что и на частицы воды, поэтому для программной реализации мы будем высчитывать данные для клеток жидкости и отдельно для клеток примесей.

На следующих рисунках будет показана эволюция клеточного автомата, моделирующего распространение “тяжёлой” примеси, поступающей в жидкость .

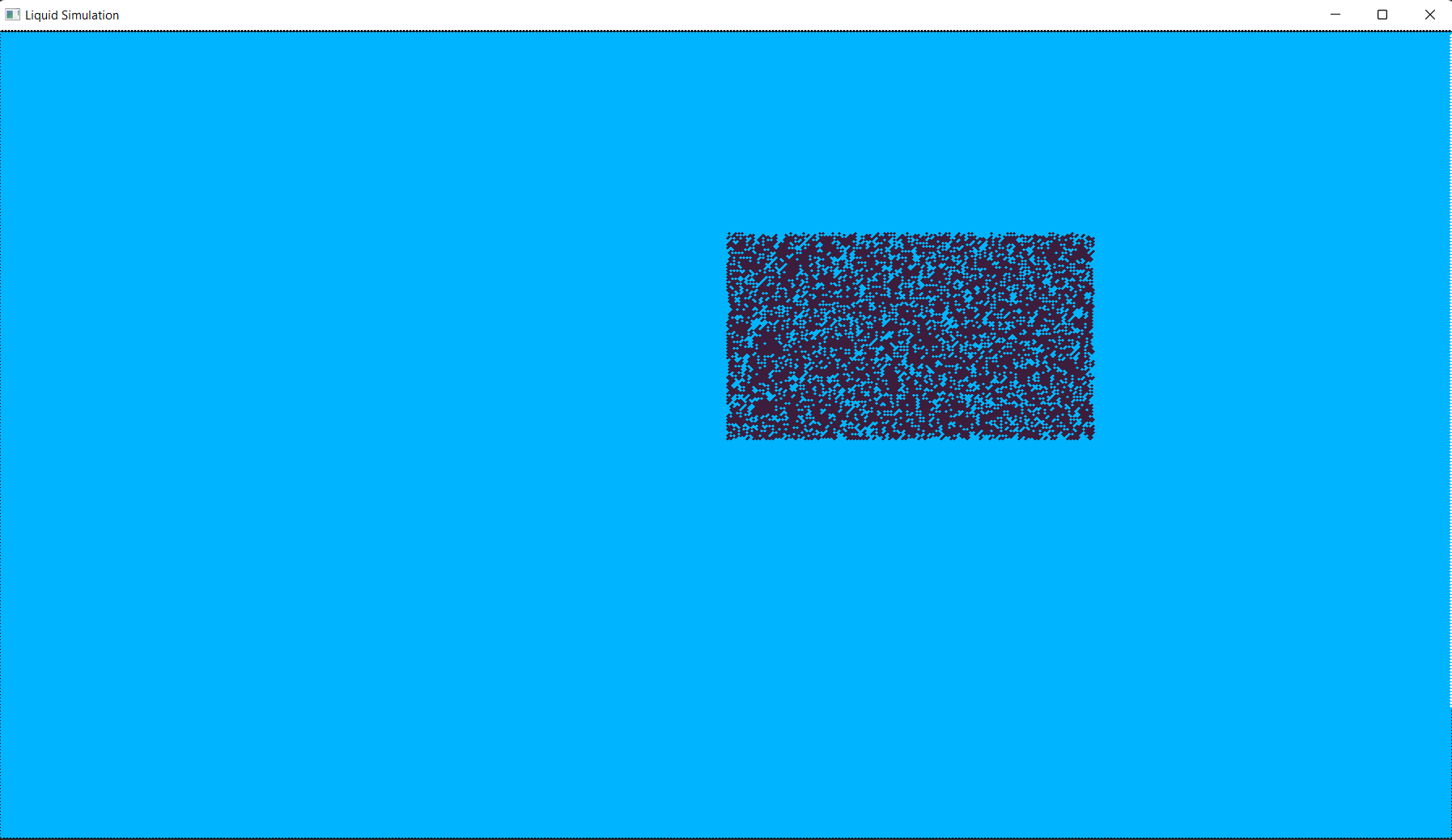


Рисунок 7 – 1 итерация

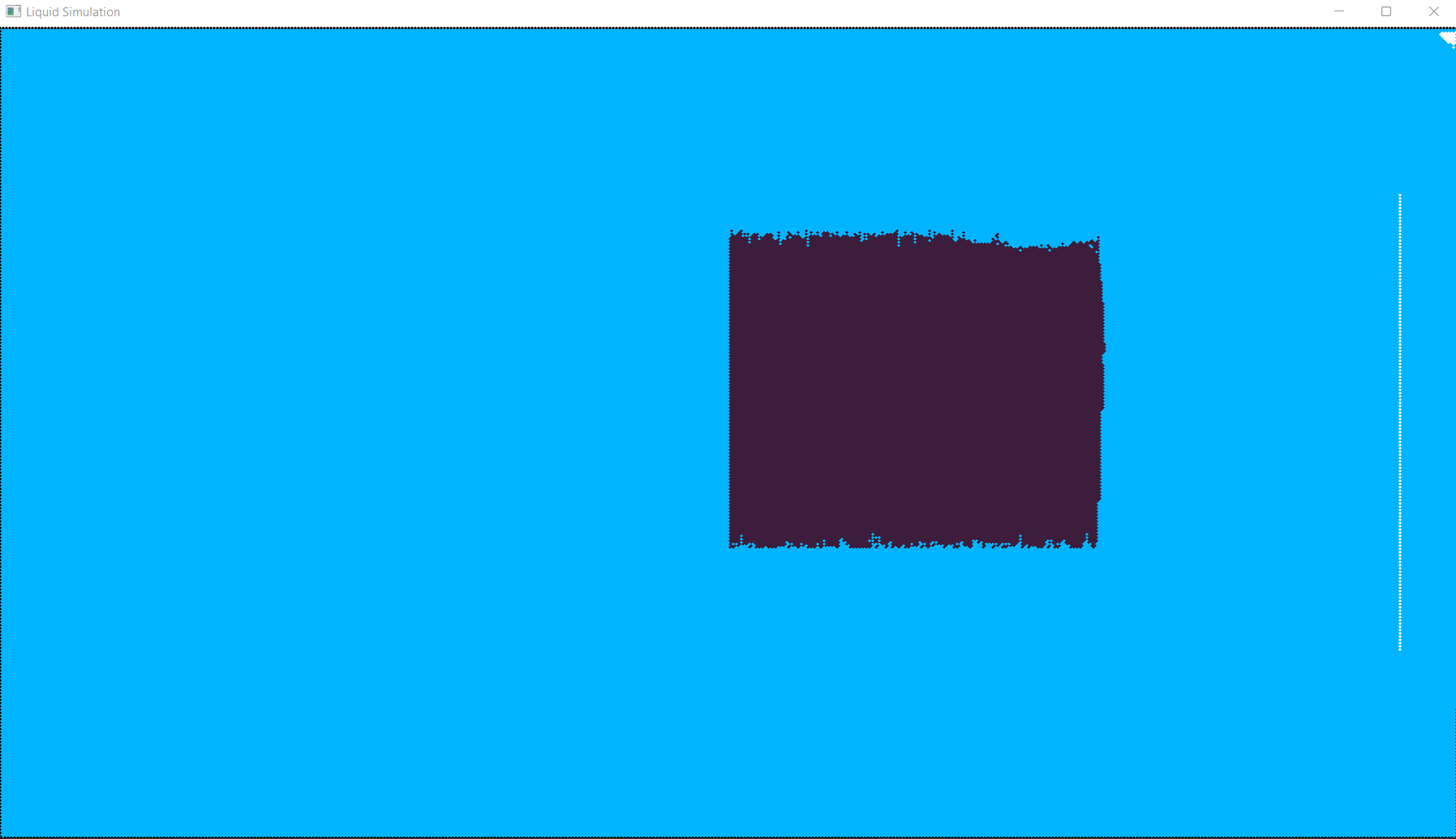


Рисунок 8 – 35 итерация



Рисунок 9 – 65 итерация



Рисунок 10 – 105 итерация

Как видим программа успешно смоделировала процесс распространения тяжёлой примеси в среде одной жидкости. Из рисунков

7-10 отчётливо видно как все клетки примеси под действием силы тяжести, обладая большим весом чем жидкость, устремились вниз, испытывая лишь слабое сопротивление от клеток жидкости. Очевидно, что клеточный автомат успешно справился со своей задачей моделирования процессов течения жидкости и распространения примеси.

**Заключение**

В курсовой работе были рассмотрены основные теоретические материалы о клеточных автоматах, FHP-I модели и реализация, с их помощью, процессов течения и распространения вещества в среде. В результате разработана программа, позволяющая пронаблюдать за распространением веществ в замкнутой среде и лишь одной силой тяжести в виде внешних источников сил. Тем самым достигнута поставленная цель курсовой работы.

В дальнейшей предполагается перейти к трёхмерному пространству и рассмотреть влияние сил взаимодействия двух веществ, а также некоторых других внешних сил.

**Список используемых источников**

1 Бандман, О.Л. Клеточно-автоматные модели пространственной динамики / О.Л. Бандман // Системная информатика, 2006. С. 59 – 113.

2 Тоффоли, Т. Т63 Машины клеточных автоматов. / Т. Тоффоли, Н. Марголус пер. с англ. - М.: Мир, 1991. – 280 с.

3 Павловская, Т.А. С/С++. Программирование на языке высокого уровня / Т.А. Павловска

**Приложение**

Код программы

#include <SFML/Graphics.hpp>

#include <vector>

#include <cstdlib>

#include <ctime>

#include <iostream>

#include <windows.h>

using namespace std;

//––––––––––––––––––––ГЛОБАЛЬНЫЕ КОНСТАНТЫ–––––––––––––––––––

const int CELL\_SIZE = 2; //РАЗМЕР КЛЕТКИ

const int GRID\_WIDTH = 900; //ШИРИНА РЕШЕТКИ (В КЛЕТКАХ)

const int GRID\_HEIGHT = 500; //ВЫСОТА РЕШЕТКИ (В КЛЕТКАХ)

const int WINDOW\_WIDTH = GRID\_WIDTH \* CELL\_SIZE; //ШИРИНА ОКНА ВЫВОДА

const int WINDOW\_HEIGHT = GRID\_HEIGHT \* CELL\_SIZE; //ВЫСОТА ОКНА ВЫВОДА

//СОСТОЯНИЯ КЛЕТОЧНОГО АВТОМАТА

struct Particle {

int liquid; // жидкость

int impurity; // примесь

};

enum CellType {

EMPTY,

OBSTACLE, // стенка

PARTICLE // частица

};

struct Cell {

CellType type;

Particle part;

sf::CircleShape shape;

};

vector<vector<Cell>> grid(GRID\_HEIGHT, vector<Cell>(GRID\_WIDTH));

//–––––––––––––––––––ПРОЦЕДУРЫ––––––––––––––––––––––

//СОЗДАНИЕ ПУСТОЙ КЛЕТОЧНОЙ РЕШЁТКИ

void initializeGrid() {

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

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

if ((i + j) % 2 == 0) {

grid[i][j].type = EMPTY;

grid[i][j].part.liquid = 0;

grid[i][j].part.impurity = 0;

grid[i][j].shape.setFillColor(sf::Color::White);

grid[i][j].shape.setRadius(CELL\_SIZE);

grid[i][j].shape.setPointCount(6);

grid[i][j].shape.setPosition(j \* CELL\_SIZE, i \* CELL\_SIZE);

}

}

}

}

//CОЗДАНИЕ ПОРИСТОГО МАТЕРИАЛА

void placeObstaclesBorders() {

// Границы поля

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

grid[i][0].type = OBSTACLE;

grid[i][0].shape.setFillColor(sf::Color::Black);

grid[i][GRID\_WIDTH - 1].type = OBSTACLE;

grid[i][GRID\_WIDTH - 1].shape.setFillColor(sf::Color::Black);

}

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

grid[0][j].type = OBSTACLE;

grid[0][j].shape.setFillColor(sf::Color::Black);

grid[GRID\_HEIGHT - 1][j].type = OBSTACLE;

grid[GRID\_HEIGHT - 1][j].shape.setFillColor(sf::Color::Black);

}

}

void placeObstacles1() {

for (int i = GRID\_HEIGHT - 80; i < GRID\_HEIGHT; ++i) {

for (int j = GRID\_WIDTH / 6; j < GRID\_WIDTH; ++j) {

//СТЕНКИ

grid[i][j].type = OBSTACLE;

grid[i][j].shape.setFillColor(sf::Color::Black);

}

}

}

void placeObstacles2() {

int k = 0;

for (int i = GRID\_HEIGHT / 2; i >= GRID\_HEIGHT / 5; --i) {

grid[i][k].type = OBSTACLE;

grid[i][k].shape.setFillColor(sf::Color::Black);

k++;

}

k = 0;

for (int i = GRID\_HEIGHT / 2 - 1; i >= GRID\_HEIGHT / 5 - 1; --i) {

grid[i][k].type = OBSTACLE;

grid[i][k].shape.setFillColor(sf::Color::Black);

k++;

}

k = 0;

for (int i = GRID\_HEIGHT / 2 + 1; i >= GRID\_HEIGHT / 5 + 1; --i) {

grid[i][k].type = OBSTACLE;

grid[i][k].shape.setFillColor(sf::Color::Black);

k++;

}

}

//ОБНОВЛЕНИЕ СОСТОЯНИЙ КЛЕТОК

void updateCellsLiquid() {

for (int i = GRID\_HEIGHT - 1; i >= 1; --i) {

for (int j = 1; j <= GRID\_WIDTH - 2; ++j) {

if (grid[i][j].part.liquid == 1 and (i>=3 && i<= GRID\_HEIGHT - 3) and (j>=2 && j<= GRID\_WIDTH - 3)) {

int s[6] = {0, 0, 0, 1, 0, 0}; int cnt = 1;

if (grid[i + 2][j].type == OBSTACLE or grid[i + 2][j].part.liquid == 1) {

s[0] = 0; s[3] = 0; cnt--;

}

if (grid[i + 2][j].type == EMPTY) {

s[0] = 0;

}

if ((grid[i + 1][j - 1].part.liquid == 1 or grid[i + 1][j - 1].type == OBSTACLE) and (grid[i - 1][j + 1].part.liquid == 1 or grid[i - 1][j + 1].type == OBSTACLE)) {

s[5] = 0; s[2] = 0;

}

else if (grid[i - 1][j + 1].type == EMPTY and (grid[i + 1][j - 1].part.liquid == 1 or grid[i + 1][j - 1].type == OBSTACLE)) {

s[5] = 1; cnt += 1;

}

else if((grid[i - 1][j + 1].type == OBSTACLE or grid[i - 1][j + 1].part.liquid == 1) and grid[i + 1][j - 1].type == EMPTY) {

s[2] = 1; cnt += 1;

}

if ((grid[i + 1][j + 1].part.liquid == 1 or grid[i + 1][j + 1].type == OBSTACLE) or (grid[i - 1][j - 1].part.liquid == 1 or grid[i - 1][j - 1].type == OBSTACLE)) {

s[1] = 0; s[4] = 0;

}

else if (grid[i - 1][j - 1].type == EMPTY and (grid[i + 1][j + 1].part.liquid == 1 or grid[i + 1][j + 1].type == OBSTACLE)) {

s[1] = 1; cnt += 1;

}

else if ((grid[i - 1][j - 1].type == OBSTACLE or grid[i - 1][j - 1].part.liquid == 1) and grid[i + 1][j + 1].type == EMPTY) {

s[4] = 1; cnt += 1;

}

if (cnt == 3) break;

else if (cnt == 1) { // если воздействует только 1 клетка

for (int l = 0; l < 6; l++) {

if (s[l] == 1) {

switch (l) {

/\*case 0:

if (grid[i + 2][j].type != OBSTACLE and grid[i + 2][j].part.liquid != 1) {

grid[i][j].type = EMPTY; grid[i][j].part.liquid = 0; grid[i][j].shape.setFillColor(sf::Color::White);

grid[i + 2][j].part.liquid = 1; grid[i + 2][j].type = PARTICLE; grid[i + 2][j].shape.setFillColor(sf::Color(0, 180, 255));

}\*/

case 1:

if (grid[i + 1][j - 1].type != OBSTACLE and grid[i + 1][j - 1].part.liquid != 1) {

grid[i][j].type = EMPTY; grid[i][j].part.liquid = 0; grid[i][j].shape.setFillColor(sf::Color::White);

grid[i + 1][j - 1].part.liquid = 1; grid[i + 1][j - 1].type = PARTICLE; grid[i + 1][j - 1].shape.setFillColor(sf::Color(0, 180, 255));

}

case 2:

if (grid[i + 1][j - 1].type != OBSTACLE and grid[i + 1][j - 1].part.liquid != 1) {

grid[i][j].type = EMPTY; grid[i][j].part.liquid = 0; grid[i][j].shape.setFillColor(sf::Color::White);

grid[i + 1][j - 1].part.liquid = 1; grid[i + 1][j - 1].type = PARTICLE; grid[i + 1][j - 1].shape.setFillColor(sf::Color(0, 180, 255));

}

case 3:

if (grid[i + 2][j].part.liquid != 1 and grid[i + 2][j].type != OBSTACLE) {

grid[i][j].type = EMPTY; grid[i][j].part.liquid = 0; grid[i][j].shape.setFillColor(sf::Color::White);

grid[i + 2][j].part.liquid = 1; grid[i + 2][j].type = PARTICLE; grid[i + 2][j].shape.setFillColor(sf::Color(0, 180, 255));

}

case 4:

if (grid[i + 1][j + 1].type != OBSTACLE and grid[i + 1][j + 1].part.liquid != 1) {

grid[i][j].type = EMPTY; grid[i][j].part.liquid = 0; grid[i][j].shape.setFillColor(sf::Color::White);

grid[i + 1][j + 1].part.liquid = 1; grid[i + 1][j + 1].type = PARTICLE; grid[i + 1][j + 1].shape.setFillColor(sf::Color(0, 180, 255));

}

case 5:

if (grid[i + 1][j + 1].type != OBSTACLE and grid[i + 1][j + 1].part.liquid != 1) {

grid[i][j].type = EMPTY; grid[i][j].part.liquid = 0; grid[i][j].shape.setFillColor(sf::Color::White);

grid[i + 1][j + 1].part.liquid = 1; grid[i + 1][j + 1].type = PARTICLE; grid[i + 1][j + 1].shape.setFillColor(sf::Color(0, 180, 255));

}

}

break;

}

}

}

else if (cnt == 2) { // если воздействует 2 клетки

int summ = 0;

int k = 0;

for (int l = 0; l < 6; l++) {

if (s[l] == 1) {

summ += l + 1;

if (k == 0) k == l + 1;

}

}

switch (summ / 2 + summ % 2) {

case 2: // клетка движется в направлении 5

if (grid[i + 1][j - 1].type != OBSTACLE and grid[i + 1][j - 1].part.liquid != 1) {

grid[i][j].type = EMPTY; grid[i][j].part.liquid = 0; grid[i][j].shape.setFillColor(sf::Color::White);

grid[i + 1][j - 1].part.liquid = 1; grid[i + 1][j - 1].type = PARTICLE; grid[i + 1][j - 1].shape.setFillColor(sf::Color(0, 180, 255));

}

case 3: // клетка движется в направлении 0

if (grid[i + 2][j].type != OBSTACLE and grid[i + 2][j].part.liquid != 1) {

grid[i][j].type = EMPTY; grid[i][j].part.liquid = 0; grid[i][j].shape.setFillColor(sf::Color::White);

grid[i + 2][j].part.liquid = 1; grid[i + 2][j].type = PARTICLE; grid[i + 2][j].shape.setFillColor(sf::Color(0, 180, 255));

}

case 5: // клетка движется в направлении 1

if (grid[i + 1][j + 1].type != OBSTACLE and grid[i + 1][j + 1].part.liquid != 1) {

grid[i][j].type = EMPTY; grid[i][j].part.liquid = 0; grid[i][j].shape.setFillColor(sf::Color::White);

grid[i + 1][j + 1].part.liquid = 1; grid[i + 1][j + 1].type = PARTICLE; grid[i + 1][j + 1].shape.setFillColor(sf::Color(0, 180, 255));

}

case 4:

if (k == 2) { // клетка движется в направлении 5

if (grid[i + 1][j - 1].type != OBSTACLE and grid[i + 1][j - 1].part.liquid != 1) {

grid[i][j].type = EMPTY; grid[i][j].part.liquid = 0; grid[i][j].shape.setFillColor(sf::Color::White);

grid[i + 1][j - 1].part.liquid = 1; grid[i + 1][j - 1].type = PARTICLE; grid[i + 1][j - 1].shape.setFillColor(sf::Color(0, 180, 255));

}

}

else { // клетка движется в направлении 1

if (grid[i + 1][j + 1].type != OBSTACLE and grid[i + 1][j + 1].part.liquid != 1) {

grid[i][j].type = EMPTY; grid[i][j].part.liquid = 0; grid[i][j].shape.setFillColor(sf::Color::White);

grid[i + 1][j + 1].part.liquid = 1; grid[i + 1][j + 1].type = PARTICLE; grid[i + 1][j + 1].shape.setFillColor(sf::Color(0, 180, 255));

}

}

case 6: // клетка движется в направлении 1

if (grid[i + 1][j + 1].type != OBSTACLE and grid[i + 1][j + 1].part.liquid != 1) {

grid[i][j].type = EMPTY; grid[i][j].part.liquid = 0; grid[i][j].shape.setFillColor(sf::Color::White);

grid[i + 1][j + 1].part.liquid = 1; grid[i + 1][j + 1].type = PARTICLE; grid[i + 1][j + 1].shape.setFillColor(sf::Color(0, 180, 255));

}

}

}

}

}

}

}

void updateCellsImpurity() {

for (int i = GRID\_HEIGHT - 1; i >= 1; --i) {

for (int j = 1; j <= GRID\_WIDTH - 2; ++j) {

if (grid[i][j].part.impurity == 1 and (i >= 3 && i <= GRID\_HEIGHT - 3) and (j >= 2 && j <= GRID\_WIDTH - 3)) {

int s[6] = { 0, 0, 0, 1, 0, 0 }; int cnt = 1;

if (grid[i + 2][j].type == OBSTACLE or grid[i + 2][j].part.impurity == 1) {

s[0] = 0; s[3] = 0; cnt--;

}

if (grid[i + 2][j].type == EMPTY) {

s[0] = 0;

}

if ((grid[i + 1][j - 1].part.impurity == 1 or grid[i + 1][j - 1].type == OBSTACLE) and (grid[i - 1][j + 1].part.impurity == 1 or grid[i - 1][j + 1].type == OBSTACLE)) {

s[5] = 0; s[2] = 0;

}

else if (grid[i - 1][j + 1].type == EMPTY and (grid[i + 1][j - 1].part.impurity == 1 or grid[i + 1][j - 1].type == OBSTACLE)) {

s[5] = 1; cnt += 1;

}

else if ((grid[i - 1][j + 1].type == OBSTACLE or grid[i - 1][j + 1].part.impurity == 1) and grid[i + 1][j - 1].type == EMPTY) {

s[2] = 1; cnt += 1;

}

if ((grid[i + 1][j + 1].part.impurity == 1 or grid[i + 1][j + 1].type == OBSTACLE) or (grid[i - 1][j - 1].part.impurity == 1 or grid[i - 1][j - 1].type == OBSTACLE)) {

s[1] = 0; s[4] = 0;

}

else if (grid[i - 1][j - 1].type == EMPTY and (grid[i + 1][j + 1].part.impurity == 1 or grid[i + 1][j + 1].type == OBSTACLE)) {

s[1] = 1; cnt += 1;

}

else if ((grid[i - 1][j - 1].type == OBSTACLE or grid[i - 1][j - 1].part.impurity == 1) and grid[i + 1][j + 1].type == EMPTY) {

s[4] = 1; cnt += 1;

}

if (cnt == 3) break;

else if (cnt == 1) { // если воздействует только 1 клетка

for (int l = 0; l < 6; l++) {

if (s[l] == 1) {

switch (l) {

/\*case 0:

if (grid[i + 2][j].type != OBSTACLE and grid[i + 2][j].part.liquid != 1) {

grid[i][j].type = EMPTY; grid[i][j].part.liquid = 0; grid[i][j].shape.setFillColor(sf::Color::White);

grid[i + 2][j].part.liquid = 1; grid[i + 2][j].type = PARTICLE; grid[i + 2][j].shape.setFillColor(sf::Color(0, 180, 255));

}\*/

case 1:

if (grid[i + 1][j - 1].type != OBSTACLE and grid[i + 1][j - 1].part.impurity != 1) {

grid[i][j].type = EMPTY; grid[i][j].part.impurity = 0; grid[i][j].shape.setFillColor(sf::Color(0, 180, 255));

grid[i + 1][j - 1].part.impurity = 1; grid[i + 1][j - 1].type = PARTICLE; grid[i + 1][j - 1].shape.setFillColor(sf::Color(60, 30, 60));

}

case 2:

if (grid[i + 1][j - 1].type != OBSTACLE and grid[i + 1][j - 1].part.impurity != 1) {

grid[i][j].type = EMPTY; grid[i][j].part.impurity = 0; grid[i][j].shape.setFillColor(sf::Color(0, 180, 255));

grid[i + 1][j - 1].part.impurity = 1; grid[i + 1][j - 1].type = PARTICLE; grid[i + 1][j - 1].shape.setFillColor(sf::Color(60, 30, 60));

}

case 3:

if (grid[i + 2][j].part.impurity != 1 and grid[i + 2][j].type != OBSTACLE) {

grid[i][j].type = EMPTY; grid[i][j].part.impurity = 0; grid[i][j].shape.setFillColor(sf::Color(0, 180, 255));

grid[i + 2][j].part.impurity = 1; grid[i + 2][j].type = PARTICLE; grid[i + 2][j].shape.setFillColor(sf::Color(60, 30, 60));

}

case 4:

if (grid[i + 1][j + 1].type != OBSTACLE and grid[i + 1][j + 1].part.impurity != 1) {

grid[i][j].type = EMPTY; grid[i][j].part.impurity = 0; grid[i][j].shape.setFillColor(sf::Color(0, 180, 255));

grid[i + 1][j + 1].part.impurity = 1; grid[i + 1][j + 1].type = PARTICLE; grid[i + 1][j + 1].shape.setFillColor(sf::Color(60, 30, 60));

}

case 5:

if (grid[i + 1][j + 1].type != OBSTACLE and grid[i + 1][j + 1].part.impurity != 1) {

grid[i][j].type = EMPTY; grid[i][j].part.impurity = 0; grid[i][j].shape.setFillColor(sf::Color(0, 180, 255));

grid[i + 1][j + 1].part.impurity = 1; grid[i + 1][j + 1].type = PARTICLE; grid[i + 1][j + 1].shape.setFillColor(sf::Color(60, 30, 60));

}

}

break;

}

}

}

else if (cnt == 2) { // если воздействует 2 клетки

int summ = 0;

int k = 0;

for (int l = 0; l < 6; l++) {

if (s[l] == 1) {

summ += l + 1;

if (k == 0) k == l + 1;

}

}

switch (summ / 2 + summ % 2) {

case 2: // клетка движется в направлении 5

if (grid[i + 1][j - 1].type != OBSTACLE and grid[i + 1][j - 1].part.impurity != 1) {

grid[i][j].type = EMPTY; grid[i][j].part.impurity = 0; grid[i][j].shape.setFillColor(sf::Color(0, 180, 255));

grid[i + 1][j - 1].part.impurity = 1; grid[i + 1][j - 1].type = PARTICLE; grid[i + 1][j - 1].shape.setFillColor(sf::Color(60, 30, 60));

}

case 3: // клетка движется в направлении 0

if (grid[i + 2][j].type != OBSTACLE and grid[i + 2][j].part.impurity != 1) {

grid[i][j].type = EMPTY; grid[i][j].part.impurity = 0; grid[i][j].shape.setFillColor(sf::Color(0, 180, 255));

grid[i + 2][j].part.impurity = 1; grid[i + 2][j].type = PARTICLE; grid[i + 2][j].shape.setFillColor(sf::Color(60, 30, 60));

}

case 5: // клетка движется в направлении 1

if (grid[i + 1][j + 1].type != OBSTACLE and grid[i + 1][j + 1].part.impurity != 1) {

grid[i][j].type = EMPTY; grid[i][j].part.impurity = 0; grid[i][j].shape.setFillColor(sf::Color(0, 180, 255));

grid[i + 1][j + 1].part.impurity = 1; grid[i + 1][j + 1].type = PARTICLE; grid[i + 1][j + 1].shape.setFillColor(sf::Color(60, 30, 60));

}

case 4:

if (k == 2) { // клетка движется в направлении 5

if (grid[i + 1][j - 1].type != OBSTACLE and grid[i + 1][j - 1].part.impurity != 1) {

grid[i][j].type = EMPTY; grid[i][j].part.impurity = 0; grid[i][j].shape.setFillColor(sf::Color(0, 180, 255));

grid[i + 1][j - 1].part.impurity = 1; grid[i + 1][j - 1].type = PARTICLE; grid[i + 1][j - 1].shape.setFillColor(sf::Color(60, 30, 60));

}

}

else { // клетка движется в направлении 1

if (grid[i + 1][j + 1].type != OBSTACLE and grid[i + 1][j + 1].part.impurity != 1) {

grid[i][j].type = EMPTY; grid[i][j].part.impurity = 0; grid[i][j].shape.setFillColor(sf::Color(0, 180, 255));

grid[i + 1][j + 1].part.impurity = 1; grid[i + 1][j + 1].type = PARTICLE; grid[i + 1][j + 1].shape.setFillColor(sf::Color(60, 30, 60));

}

}

case 6: // клетка движется в направлении 1

if (grid[i + 1][j + 1].type != OBSTACLE and grid[i + 1][j + 1].part.impurity != 1) {

grid[i][j].type = EMPTY; grid[i][j].part.impurity = 0; grid[i][j].shape.setFillColor(sf::Color(0, 180, 255));

grid[i + 1][j + 1].part.impurity = 1; grid[i + 1][j + 1].type = PARTICLE; grid[i + 1][j + 1].shape.setFillColor(sf::Color(60, 30, 60));

}

}

}

}

}

}

}

//ЗАПОЛНЕНИЕ ОКНА

void drawGrid(sf::RenderWindow& window) {

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

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

if((i+j)%2==0)

window.draw(grid[i][j].shape);

}

}

}

int main() {

sf::RenderWindow window(sf::VideoMode(WINDOW\_WIDTH, WINDOW\_HEIGHT), "Liquid Simulation");

int count = 0, maxcount = 0;

cout << "Enter number of iterations: \n";

cin >> maxcount;

initializeGrid();

placeObstaclesBorders();

placeObstacles1();

placeObstacles2();

//ДОБАВЛЕНИЕ ЖИДКОСТИ

srand(time(NULL));

/\*for (int i = GRID\_HEIGHT / 4; i < GRID\_HEIGHT \* 3 / 4; ++i) {

for (int j = GRID\_WIDTH / 4; j < GRID\_WIDTH\*3/4; ++j) {

if (rand() % 3 == 0 or rand() % 3 == 1) {

grid[i][j].type = PARTICLE;

grid[i][j].part.liquid = 1;

grid[i][j].shape.setFillColor(sf::Color(0, 180, 255));

}

}

}

for (int i = 2; i < 7; ++i) {

for (int j = 2; j < 7; ++j) {

grid[i][j].type = PARTICLE;

grid[i][j].part.liquid = 1;

grid[i][j].shape.setFillColor(sf::Color(0, 180, 255));

}

}\*/

for (int i = GRID\_HEIGHT - 2; i >= 1; --i) {

for (int j = 1; j < GRID\_WIDTH - 2; ++j) {

grid[i][j].type = PARTICLE;

grid[i][j].part.liquid = 1;

grid[i][j].shape.setFillColor(sf::Color(0, 180, 255));

}

}

for (int i = GRID\_HEIGHT / 2; i >= GRID\_HEIGHT / 4; --i) {

for (int j = GRID\_WIDTH / 2; j <= GRID\_WIDTH \* 3 / 4; ++j) {

if (rand() % 2 == 0) {

grid[i][j].type = PARTICLE;

grid[i][j].part.impurity = 1;

grid[i][j].shape.setFillColor(sf::Color(60, 30, 60));

}

}

}

while (window.isOpen()) {

sf::Event event;

while (window.pollEvent(event)) {

if (event.type == sf::Event::Closed) {

Sleep(20000);

window.close();

}

if (count > maxcount) window.close();

}

if (count == 1) {

Sleep(10000);

}

updateCellsLiquid();

updateCellsImpurity();

count++;

window.clear(sf::Color::White);

drawGrid(window);

window.display();

cout << count << endl;

}

return 0;

}

я, – СПб: Питер, 2003. – 461 с.