**Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования**

**"** **Уфимский университет науки и технологий "**

**Кафедра** высокопроизводительных вычислений и дифференциальных уравнений

**Дисциплина:** Математическое моделирование.

**Отчет по лабораторной работе № 2**

**Тема:** «Имитационное моделирование сложных систем с помощью клеточных автоматов»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Группа ПМ-457 | Фамилия И.О. | Подпись | Дата | Оценка |
| Студент | Садыков Р.А. |  |  |  |
| Принял | Лукащук С.Ю. |  |  |  |

**Уфа 2025**

**Цель работы:** получить навык имитационного моделирования сложных динамических систем с использованием клеточных автоматов на примере моделей биологических систем.

**Задание**

**Задача I. Клеточный автомат «Жизнь»**

Выполнить программную реализацию клеточного автомата, функционирующего в соответствии со следующими правилами:

1) клетка может находиться в двух состояниях - пассивном и активном;

2) в качестве окрестности рассматривается восемь соседних клеток;

3) если в окрестности пассивной клетки две активных, то данная клетка также становится активной ("рождается");

4) если в окрестности активной клетки три или более активных клеток, то она становится пассивной ("умирает").

Реализовать алгоритм на клеточном пространстве 32х32 ячеек. Начальное распределение активных и пассивных клеток – случайное, подчиняющееся равномерному закону распределения. Также подобрать начальные распределения, соответствующие стационарным и циклическим структурам (по три примера каждой структуры).

**Задача II. Клеточный автомат «Нейронная сеть»**

Данный автомат имитирует явления в однородной двумерной нейронной сети, состоящей из возбудимых элементов и функционирует по следующим правилам:

1) клетка может находиться в трех состояниях: покоя, активном и состоянии восстановления;

2) в качестве окрестности рассматриваются восемь соседних клеток;

3) переход в состояние активности зависит от некоторого параметра, называемого уровнем активатора. В возбужденном состоянии клетки уровень активатора равен 1. В других состояниях он распадается на А % за такт;

4) если клетка была в покое и общее количество активатора в восьми соседних и в данной клетке превысило порог активации П, то клетка возбуждается на Т тактов;

5) через Т тактов возбужденная клетка переходит в состояние восстановления на В тактов, а затем в переходит в состояние покоя.

Реализовать алгоритм при следующих параметрах: клеточное пространство 256х256 ячеек, А = 30%, П = 3, Т = 5, В = 8. Начальное распределение состояния клеток задано плоским фронтом. Также имеется периодический источник возбуждения (3х3 клетки) с периодом 15 тактов. Выявить характер взаимодействия между собой различных фронтов возбуждения.

**Задача III. Клеточный автомат «Организмы - питательная среда»**

Клеточный автомат моделирует взаимодействие одноклеточных организмов с питательной средой и функционирует по следующим правилам:

1) клеточное пространство образует поле NxN клеток;

2) окрестность клетки составляют восемь соседних клеток;

3) каждой клетке соответствует значение P степени питательности раствора (энергоемкости), которое может изменяться от 0 до Pmax;

4) прирост ΔP питательности (энергоемкости) раствора клетки за такт времени выполняется следующим образом: ΔP=0 при P=Pmax и ΔP=r при P

5) общий запас энергии питательного раствора определяется суммарной питательностью (энергией) всех клеток и не может быть более Pmax;

6) клетка может быть свободной или содержать не более одного одноклеточного или другого живого организма;

7) отдельная особь одноклеточного черпает энергию из питательного раствора клетки, в которой она находится, снижая его питательность и повышая свой запас энергии на Δp за такт;

8) максимально возможное количество энергии, запасаемое одноклеточным, не превышает p1;

9) на свои нужды отдельная особь затрачивает Δe энергии за такт;

10) особь всегда старается перейти на соседнюю свободную клетку, выбирая направление перехода случайным образом;

11) время жизни отдельной особи составляет L тактов;

12) если время жизни особи превысило продолжительность жизни для данных организмов или запас энергии снизился до нуля, то особь умирает;

13) начиная с возраста T тактов особь считается зрелой и может производить себе подобных, затрачивая Δr энергии при каждом делении дополнительно. При этом старая особь переходит на свободную соседнюю клетку, а новая остается в старой. Если свободных клеток в окрестности нет, то деления не происходит;

14) начальное распределение особей по клеточному пространству подчинено равномерному закону распределения. Начальное число особей составляет А% максимально возможного, равного N 2

Реализовать алгоритм при следующих параметрах: клеточное пространство 256х256 ячеек, Pmax=10, r=1, А = 30%, L=15, T=3, Δp=5, p1=35, Δe=2, Δr=3. Выявить характерные зависимости в поведении колонии одноклеточных.

**Задача IV. Модифицированный клеточный автомат «Организмы - питательная среда»**

Выполнить модификацию алгоритма из задачи III, заменив правило 10 на следующее: особь всегда старается перейти на соседнюю свободную клетку с наибольшим уровнем энергоемкости. Если ячейки в окрестности, имеют меньший запас энергии, то особь остается в прежней клетке.

**Ход работы**

Клеточный автомат «Жизнь»

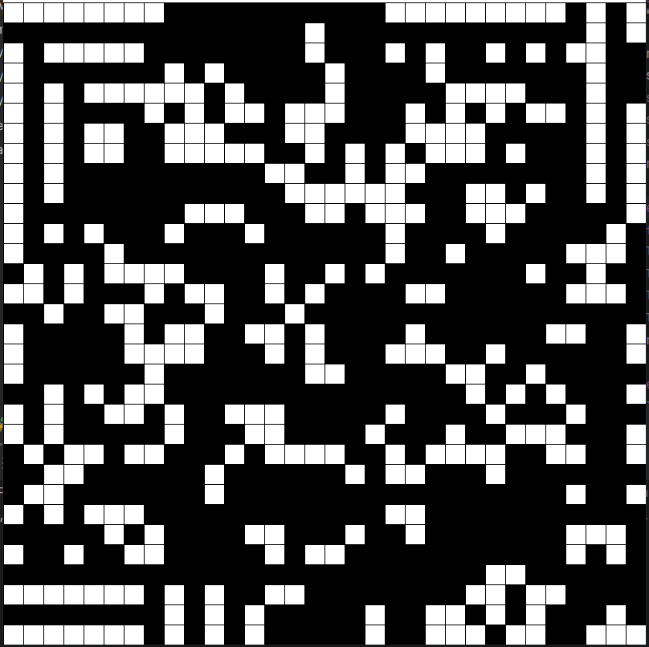
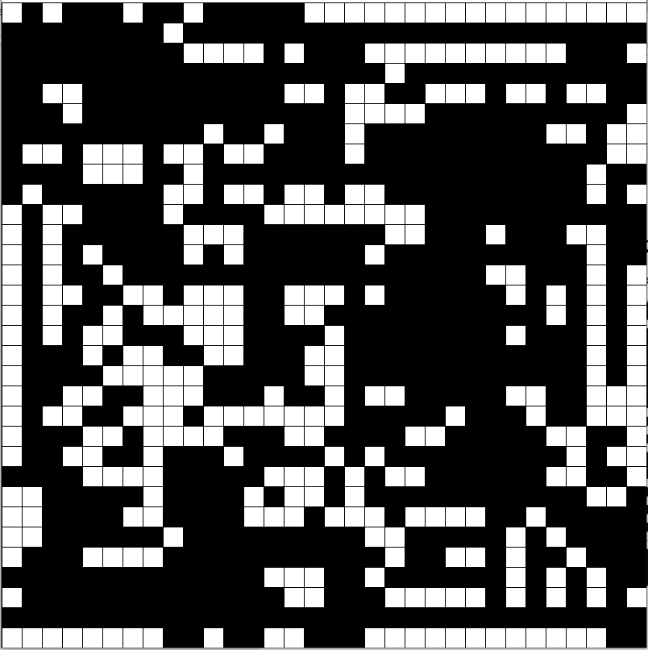
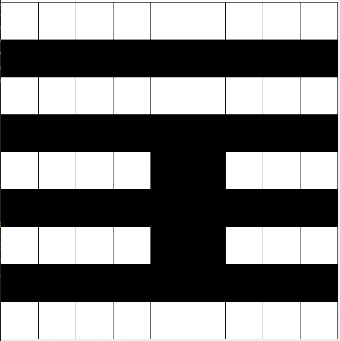
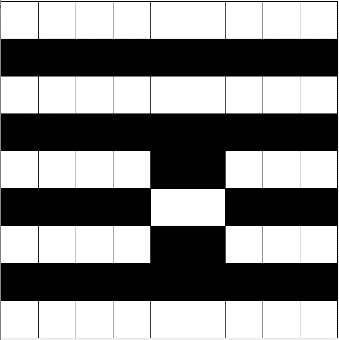
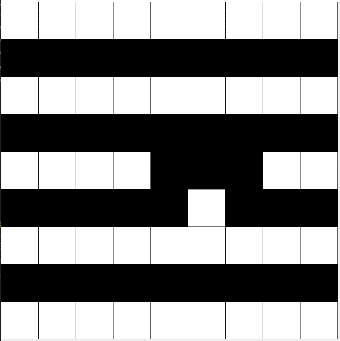
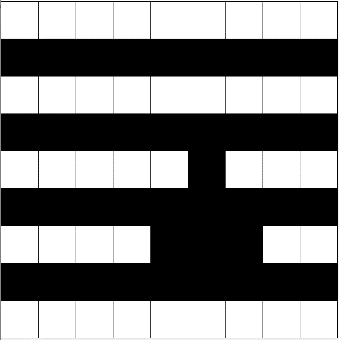
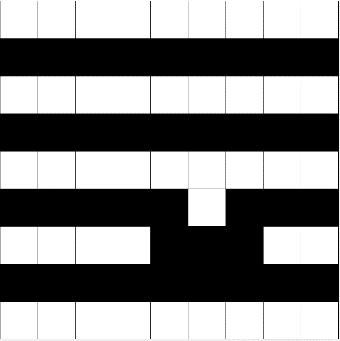
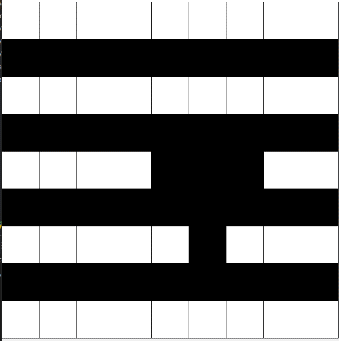
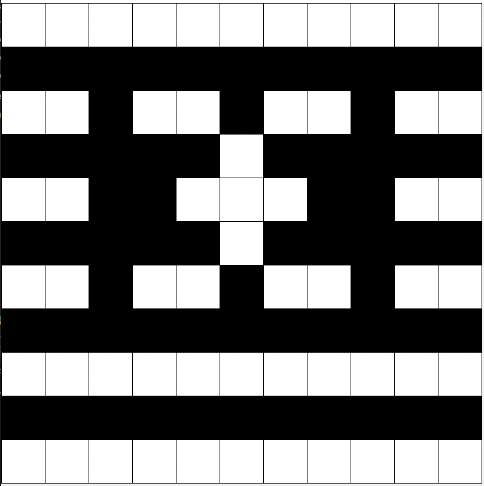
 

Рисунок 1 – Пример реализации клеточного автомата на разных итерациях

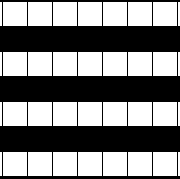
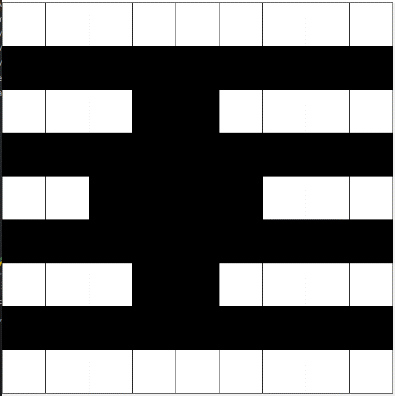
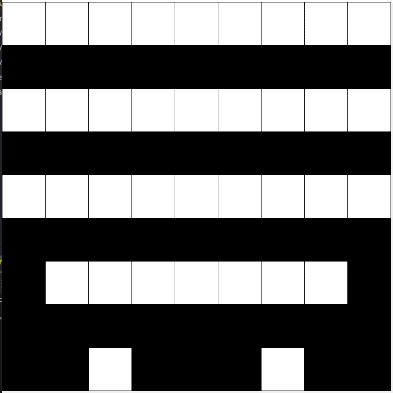
Подберем такие начальные распределения, которые соответствуют циклическим структурам:



Аналогично для стационарных структур:

Клеточный автомат «Нейронная сеть»:

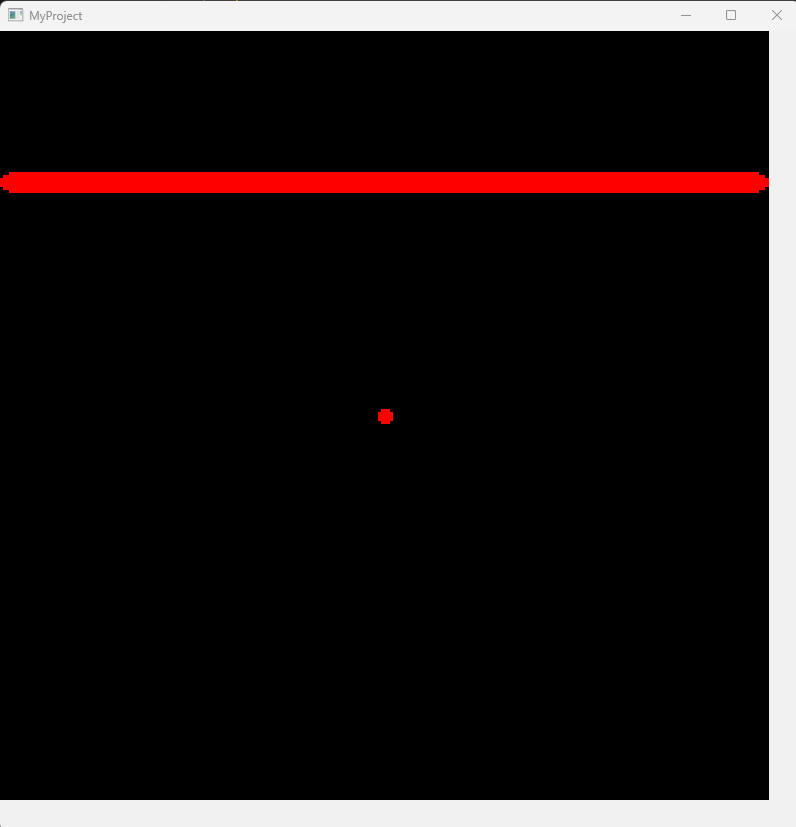
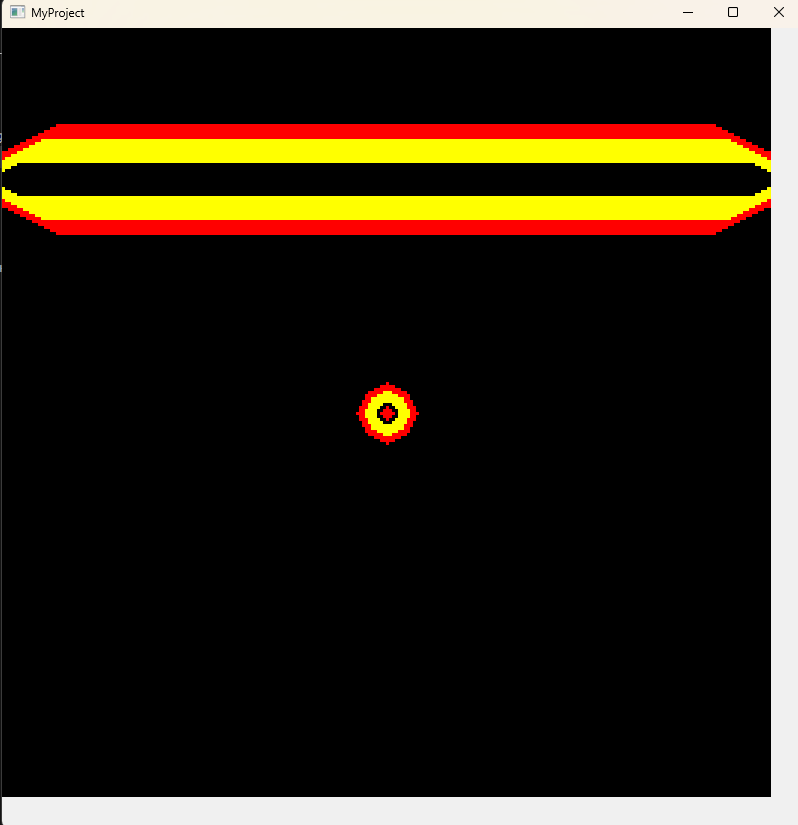


Рисунок 2 – Начальное распределение



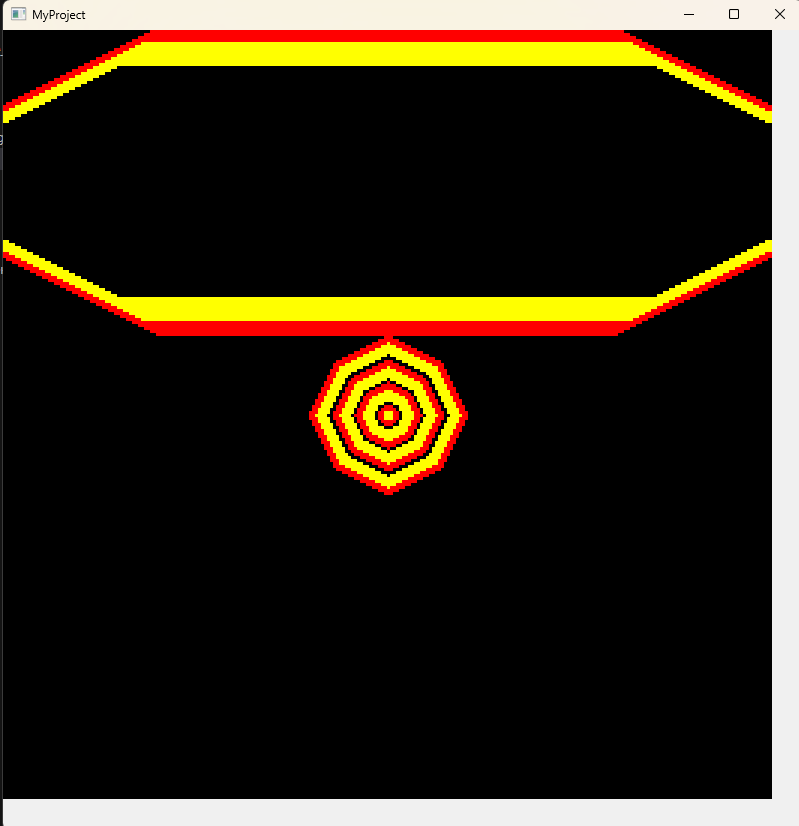






Рисунок 2 – ход работы программы за 150 тактов

По рисункам 2-6 видно, что при соединении волны продолжают движение вместе.

Клеточный автомат «Организмы – питательная среда»:

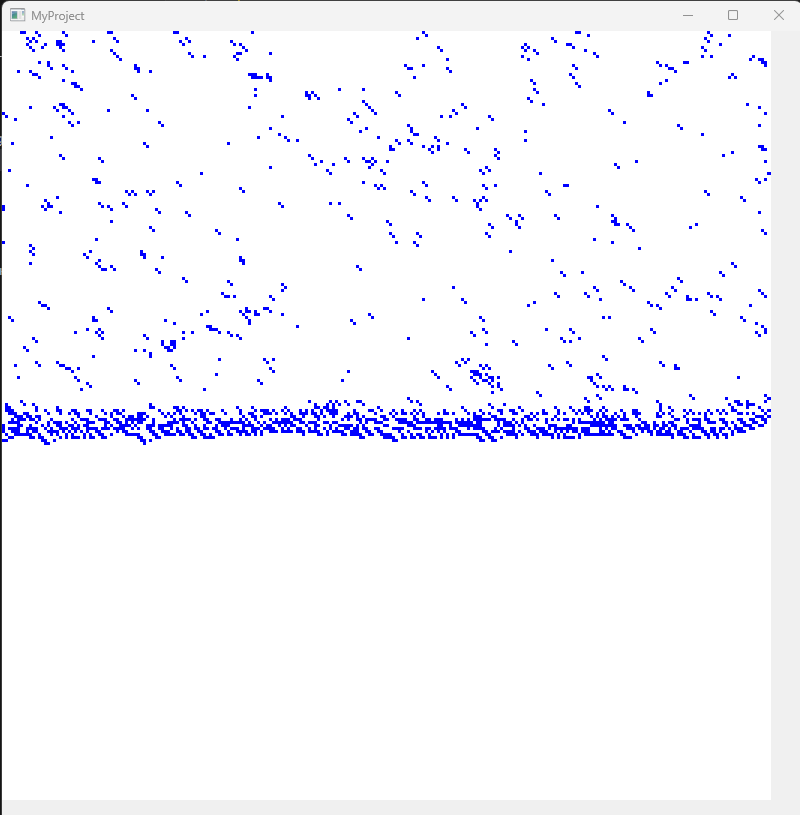
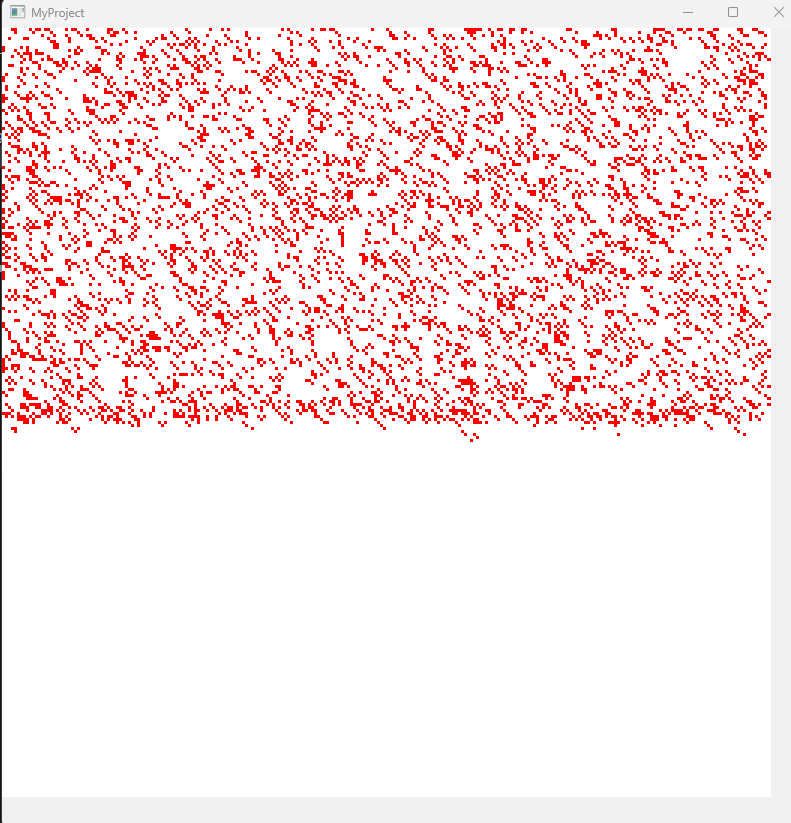


Рисунок 4 – Начальное распределение клеточного автомата

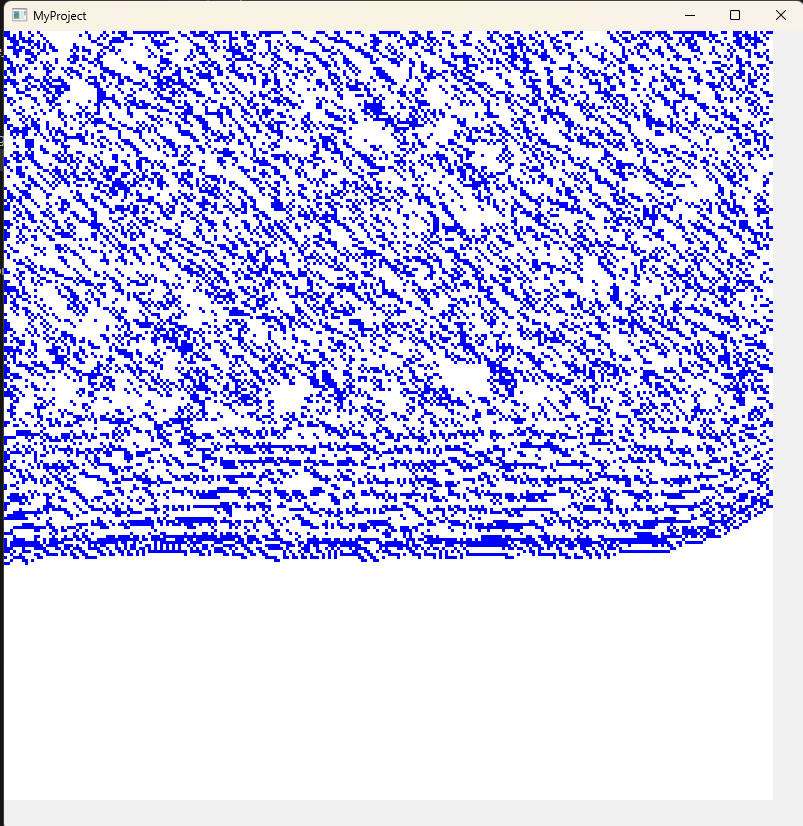
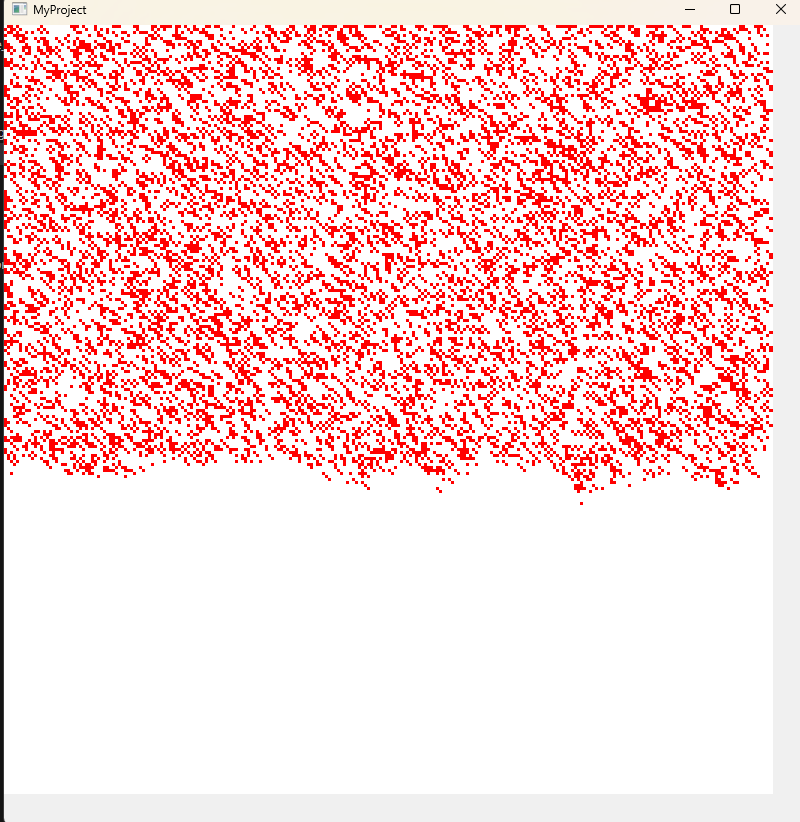


Рисунок 5 – Результат работы на 10 такте

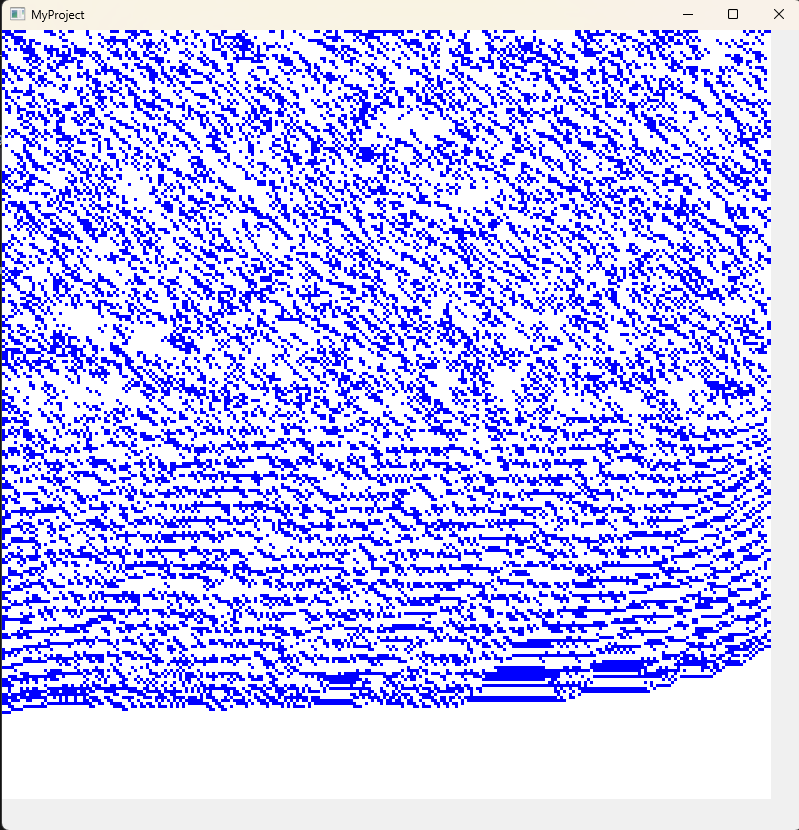
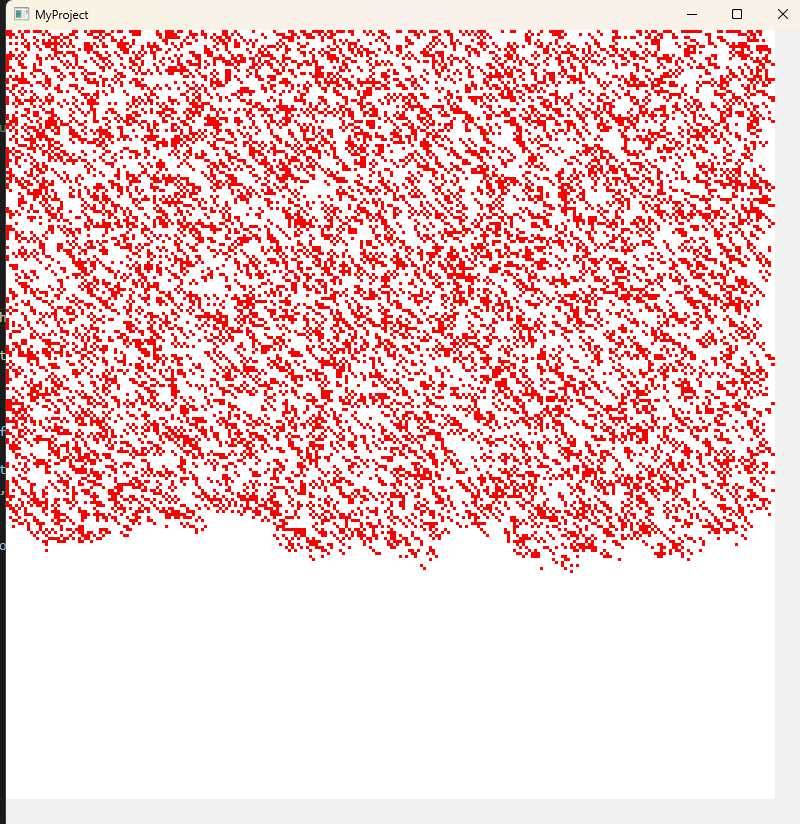


Рисунок 6 – Результат работы на 30 такте

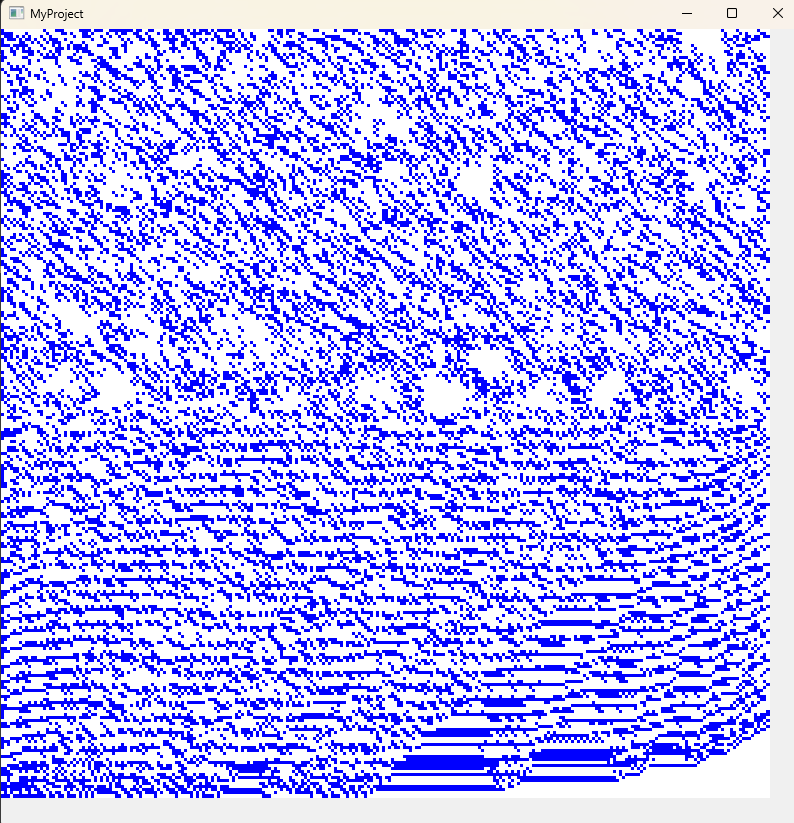
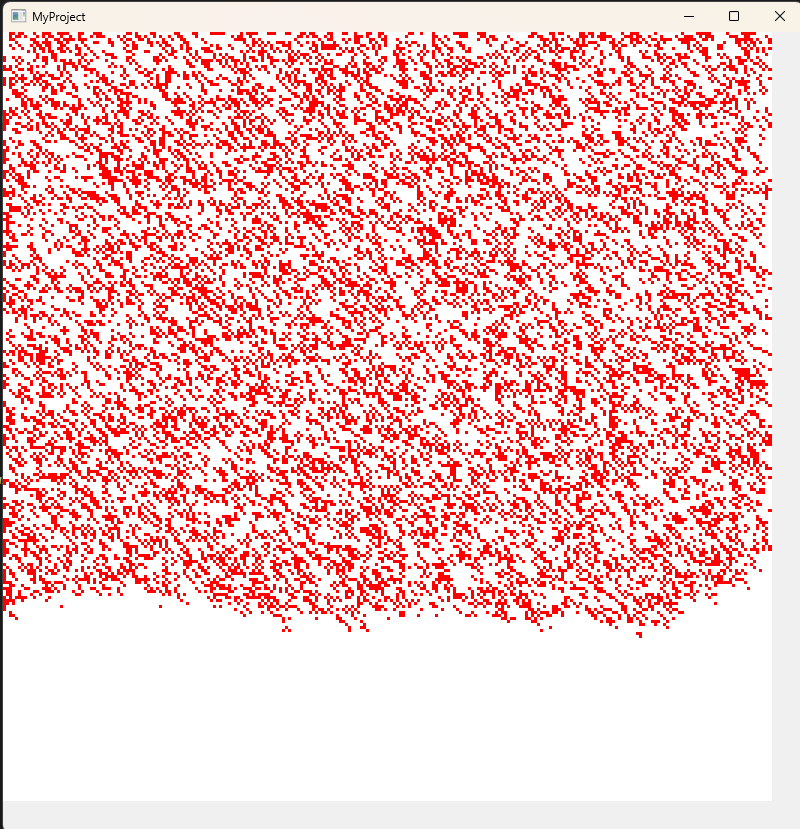


Рисунок 7 – Результат работы на 100 такте

В модифицированном алгоритме клеточного автомата «Организм – питательная среда» при начальном распределении в верхней половине поля, виден явный фронт движения частиц и слегка более высокая скорость в сторону пустого пространства, в то время как не модифицированный алгоритм продолжает движение частиц более случайно.

**Вывод**

В ходе лабораторной работы были написаны вычислительные программы для реализации алгоритмов «Жизнь», «Нейронная сеть» и «Организмы – питательная среда» для клеточных автоматов.

В клеточном автомате «Жизнь» задания движение ячеек полностью хаотичное и структурированности не наблюдается. В клеточном автомате «Нейронная сеть» волны возбуждения при столкновении продолжают движение вместе, а генератор образует круговые волны. В клеточном автомате «Организмы – питательная среда» до модификации с умным переходом, клетки располагались и передвигались хаотично. После модификации наблюдается четкий фронт движения частиц.

**Приложение**

var size = 32;

var currentState = {};

var currentQnt = {};

var nextQnt = {};

var nextState = {};

var states = Array('alive', 'dead');

jQuery.fn.exists = function(){ return this.length > 0; }

function task(t)

{

window.setTimeout(function(){

console.log("timeout start for t="+t);

for (var i = 0; i <= size+1; i++) {

var tmp = [];

var tmpQnt = [];

for (var j = 0; j <= size+1; j++) {

if ((i==0) || (j==0) || (i==size+1) || (j==size+1))

{

tmp[j] = states[1];

}

else

{

var qnt = 0;

for (var iOff = -1; iOff <= 1; iOff++) {

for (var jOff = -1; jOff <= 1; jOff++) {

if (iOff == 0 && jOff == 0)

{

//pass

}

else

{

if (currentState[i + iOff][j + jOff] == states[0])

{

qnt++;

}

}

}

}

tmpQnt[j] = qnt;

if (qnt == 2)

{

tmp[j] = states[0];

}

else if (qnt >= 3)

{

tmp[j] = states[1];

}

else

{

tmp[j] = currentState[i][j];

}

}

}

nextState[i] = tmp;

currentQnt[i] = tmpQnt;

}

currentState = nextState;

console.log("timeout ends for t="+t);

console.log("Updating for t="+t);

$('.el').each(function(index){

var h = $(this).data("h");

var v = $(this).data("v");

//$(this).text(currentQnt[h][v]);

$(this).removeClass("alive").removeClass("dead").addClass(currentState[h][v]);

})

}, 200\*t);

}

$(document).ready(function(){

for (var i = 0; i <= size+1; i++) {

var tmp = [];

var tmpQ = [];

for (var j = 0; j <= size+1; j++) {

tmp[j] = states[1];

tmpQ[j] = 0;

if (i==1 && j==2)

tmp[j] = states[0];

if (i==1 && j==3)

tmp[j] = states[0];

if (i==2 && j==3)

tmp[j] = states[0];

//tmp[j] = states[Math.floor(Math.random() \* states.length)];

if ((i==0) || (j==0) || (i==size+1) || (j==size+1))

{

tmp[j] = states[1];

}

else

{

$('.wrap').append('<div data-h="'+i+'" data-v="'+j+'" class="el '+tmp[j] +'"></div>');

}

}

currentState[i] = tmp;

currentQnt[i] = tmpQ;

}

T = 10;

for (var t = 0; t<=T; t++) {

console.log("t = " + t);

//task(t);

}

})

var size = 128;

var sizePX = 640;

var currentState = {};

var currentA = {};

var currentT = {};

var states = Array('rest', 'active', 'recovery');

var dA = 0.3; // клетка распадается на dА за такт

var threshold = 3; // пороговое значение для активации

var T = 5; // период возбуждения

var B = 8; // период восстановления

var delay = 100; // пауза между кадрами при аниамации

/\*

Каждая клетка:

1) возбуждается при 3 вокруг на 5 тактов, А=1

2) восстановление на 8 тактов, А -= .33

3) покой

currentTakt = [0 : 15]:

[0] - покой

[1 : 8] - восстановление

[9 : 13] - возбуждение

\*/

function task(t){

window.setTimeout(function(){

console.log("timeout start for t="+t);

if (t%15 == 0)

{

generator(currentState, currentA, currentT);

}

var nextState = {};

var nextA = {};

var nextT = {};

for (var i = 0; i <= size+1; i++) {

var tmpS = [];

var tmpA = [];

var tmpT = [];

for (var j = 0; j <= size+1; j++) {

if ((i==0) || (j==0) || (i==size+1) || (j==size+1))

{

tmpS[j] = states[0];

tmpA[j] = 0;

tmpT[j] = 0;

}

else

{

if (currentState[i][j] == states[0])

{

var sum = 0;

for (var iOff = 1; iOff >= -1; iOff--) {

for (var jOff = 1; jOff >= -1; jOff--) {

if (currentState[i + iOff][j + jOff] == states[1])

{

sum += 1;

}

}

}

if (sum >= threshold)

{

//console.log("t=" + t + " [" + i + "; " + j + "] => "+sum);

tmpS[j] = states[1];

tmpA[j] = 1;

tmpT[j] = 15;

}

else

{

tmpA[j] = currentA[i][j] - dA;

(tmpA[j] <= 0) ? tmpA[j]=0 : tmpA[j]-=0;

tmpT[j] = currentT[i][j] - 1;

(tmpT[j] <= 0) ? tmpT[j]=0 : tmpT[j]-=0;

}

}

else

{

tmpA[j] = currentA[i][j] - dA;

(tmpA[j] <= 0) ? tmpA[j]=0 : tmpA[j]-=0;

tmpT[j] = currentT[i][j] - 1;

(tmpT[j] <= 0) ? tmpT[j]=0 : tmpT[j]-=0;

}

if (tmpT[j] >= 9) {

tmpS[j] = states[1];

tmpA[j] = 1;

} else if (tmpT[j] >= 1) {

tmpS[j] = states[2];

} else {

tmpS[j] = states[0];

}

}

}

nextState[i] = tmpS;

nextA[i] = tmpA;

nextT[i] = tmpT;

}

currentState = nextState;

currentA = nextA;

currentT = nextT;

update(currentState);

//console.log(takt[18][16]);

}, delay \*t);

}

function draw(state) {

for (var i = 1; i <= size; i++) {

for (var j = 1; j <= size; j++) {

$('.wrap').append('<div data-h="'+i+'" data-v="'+j+'" class="el '+state[i][j] +'"></div>');

}

}

}

function update(state) {

$('.el').each(function(index){

var h = $(this).data("h");

var v = $(this).data("v");

$(this).removeClass("rest").removeClass("active").removeClass("recovery").addClass(currentState[h][v]);

})

}

function generator(currentState, currentA, currentT) {

for (var i = size/2-1; i <= size/2+1; i++) {

for (var j = size/2-1; j <= size/2+1; j++) {

currentState[i][j] = states[1];

currentA[i][j] = 1.0;

currentT[i][j] = 15;

}

}

}

$(document).ready(function(){

for (var i = 0; i <= size+1; i++) {

var tmpS = [];

var tmpA = [];

var tmpT = [];

for (var j = 0; j <= size+1; j++) {

tmpS[j] = states[0];

tmpA[j] = 0;

tmpT[j] = 0;

}

currentState[i] = tmpS;

currentA[i] = tmpA;

currentT[i] = tmpT;

}

for (var i = size/2-1; i <= size/2+1; i++) {

for (var j = size/2-1; j <= size/2+1; j++) {

currentState[i][j] = states[1];

currentA[i][j] = 1.0;

currentT[i][j] = 15;

}

}

for (var i = 0; i <= size+1; i++) {

currentState[5][i] = states[1];

currentA[5][i] = 1.0;

currentT[5][i] = 15;

currentState[size-20][i] = states[1];

currentA[size-20][i] = 1.0;

currentT[size-20][i] = 15;

}

draw(currentState);

$('.wrap').css("width", sizePX+"px");

$('.wrap').css("height", sizePX+"px");

$('.el').css("width", "calc("+sizePX+"px/"+size+")");

$('.el').css("height", "calc("+sizePX+"px/"+size+")");

T = 25;

for (var t = 0; t<=T; t++) {

//console.log("t = " + t);

//task(t);

}

})

var size = 64;

var sizePX = 640;

var currentP = {}; // source

var currentE = {}; // energy

var currentT = {}; // lifetime in each point

var currentState = {};

var lifeTime = 15;

var bornTime = 3;

var Pmax = 10;

var Emax = 35;

var dP = 1; // energy by 1 takt

var dE = 2; // energy for living by 1 takt

var dR = 3; // energy for born

var sumMaxP = size\*size\*Pmax;

var startCount = 0.3 \* size \* size;

var states = Array('empty', 'notempty');

var delay = 500; // пауза между кадрами при аниамации

function task(t){

window.setTimeout(function(){

console.log("timeout start for t="+t);

for (var i = 0; i <= size+1; i++) {

for (var j = 0; j <= size+1; j++) {

// check if boundary then make all zero

if ((i==0) || (j==0) || (i==size+1) || (j==size+1))

{

currentP[i][j] = 0;

currentE[i][j] = 0;

currentT[i][j] = 0;

currentState[i][j] = states[0];

}

else

{

// if SQ is empty

if (currentState[i][j] == states[0])

{

currentE[i][j] = 0;

currentT[i][j] = 0;

// add P to this

currentP[i][j] += dP;

(currentP[i][j] >= Pmax) ? currentP[i][j] = Pmax : currentP[i][j]-=0;

}

// if SQ is NOT empty

else

{

// eat from P to E

if (currentP[i][j] >= dE)

{

currentP[i][j] -= dE;

currentE[i][j] += dE;

(currentE[i][j] >= Emax) ? currentE[i][j]=Emax : currentE[i][j]-=0;

}

// take energy from organsim just for living

currentE[i][j] -= dE;

currentT[i][j] += 1;

// if there's not enough energy - DIE

if (currentE[i][j] <= 0 || currentT[i][j] > lifeTime)

{

currentState[i][j] = states[0];

currentE[i][j]=0;

currentT[i][j]=16;

}

}

}

}

}

// MOVE

var offsetRnd = [-1, 0, 1];

for (var i = 1; i <= size; i++) {

for (var j = 1; j <= size; j++) {

if (currentState[i][j] == states[1])

{

var offset = [];

offset[0] = Math.floor(Math.random() \* 3)-1;

offset[1] = Math.floor(Math.random() \* 3)-1;

while (currentState[i+offset[0]][j+offset[1]] == states[1] )

{

offset[0] = Math.floor(Math.random() \* 3)-1;

offset[1] = Math.floor(Math.random() \* 3)-1;

}

//console.log("(" + i + ", " + j + ") => " + currentP[i][j] + ", " + currentE[i][j] + ", " + currentT[i][j]);

currentState[i+offset[0]][j+offset[1]] = states[1];

//tmpState[i][j] = states[0];

currentState[i][j] = states[0];

currentE[i+offset[0]][j+offset[1]] = currentE[i][j];

currentE[i][j] = 0;

currentT[i+offset[0]][j+offset[1]] = currentT[i][j];

currentT[i][j] = 0;

}

}

}

// BORN

for (var i = 1; i <= size; i++) {

for (var j = 1; j <= size; j++) {

if (currentState[i][j] == states[1] && currentT[i][j] > 3 && currentE[i][j] > dR)

{

var offset = [];

offset[0] = Math.floor(Math.random() \* 3)-1;

offset[1] = Math.floor(Math.random() \* 3)-1;

while (currentState[i+offset[0]][j+offset[1]] == states[1] )

{

offset[0] = offsetRnd[Math.floor(Math.random() \* offsetRnd.length)];

offset[1] = offsetRnd[Math.floor(Math.random() \* offsetRnd.length)];

}

currentState[i][j] = states[1];

currentState[i+offset[0]][j+offset[1]] = states[1];

currentE[i][j] -= dR;

currentE[i+offset[0]][j+offset[1]] = currentE[i][j];

currentT[i+offset[0]][j+offset[1]] = 0;

}

}

}

update(currentState);

}, delay \*t);

}

function pseudoRandom() {

var notRandomNumbers = [0, 0, 0, 0, 0, 0, 0, 1, 1, 1];

var idx = Math.floor(Math.random() \* notRandomNumbers.length);

return notRandomNumbers[idx];

}

function draw(state) {

for (var i = 1; i <= size; i++) {

for (var j = 1; j <= size; j++) {

$('.wrap').append('<div data-h="'+i+'" data-v="'+j+'" class="el '+state[i][j] +'"></div>');

}

}

}

function update(state) {

$('.el').each(function(index){

var h = $(this).data("h");

var v = $(this).data("v");

if (currentState[h][v] == states[0])

{

$(this).removeClass("empty").removeClass("notempty").addClass(currentState[h][v]);

}

else

{

$(this).removeClass("empty").removeClass("notempty").addClass(currentState[h][v]);

}

})

}

$(document).ready(function(){

var count = 0;

for (var i = 0; i <= size+1; i++) {

var tmpP = [];

var tmpE = [];

var tmpT = [];

var tmpState = [];

for (var j = 0; j <= size+1; j++) {

tmpP[j] = Pmax/2;

tmpE[j] = 25;

tmpT[j] = 0;

tmpState[j] = states[pseudoRandom()];

if (tmpState[j] == states[1])

count++;

}

currentState[i] = tmpState;

currentP[i] = tmpP;

currentE[i] = tmpE;

currentT[i] = tmpT;

}

console.log("N = " + count + "; A = " + Math.floor((count/size/size)\*100) + "%");

draw(currentState);

$('.wrap').css("width", sizePX+"px");

$('.wrap').css("height", sizePX+"px");

$('.el').css("width", "calc("+sizePX+"px/"+size+")");

$('.el').css("height", "calc("+sizePX+"px/"+size+")");

T = 10;

for (var t = 0; t<=T; t++) {

//console.log("t = " + t);

//task(t);

}

})

var size = 64;

var sizePX = 640;

var currentP = {}; // source

var currentE = {}; // energy

var currentT = {}; // lifetime in each point

var currentState = {};

var lifeTime = 15;

var bornTime = 3;

var Pmax = 10;

var Emax = 35;

var dP = 1; // energy by 1 takt

var dE = 2; // energy for living by 1 takt

var dR = 3; // energy for born

var sumMaxP = size\*size\*Pmax;

var startCount = 0.3 \* size \* size;

var states = Array('empty', 'notempty');

var delay = 500; // пауза между кадрами при аниамации

function task(t){

window.setTimeout(function(){

console.log("timeout start for t="+t);

for (var i = 0; i <= size+1; i++) {

for (var j = 0; j <= size+1; j++) {

// check if boundary then make all zero

if ((i==0) || (j==0) || (i==size+1) || (j==size+1))

{

currentP[i][j] = 0;

currentE[i][j] = 0;

currentT[i][j] = 0;

currentState[i][j] = states[0];

}

else

{

// if SQ is empty

if (currentState[i][j] == states[0])

{

currentE[i][j] = 0;

currentT[i][j] = 0;

// add P to this

currentP[i][j] += dP;

(currentP[i][j] >= Pmax) ? currentP[i][j] = Pmax : currentP[i][j]-=0;

}

// if SQ is NOT empty

else

{

// eat from P to E

if (currentP[i][j] >= dE)

{

currentP[i][j] -= dE;

currentE[i][j] += dE;

(currentE[i][j] >= Emax) ? currentE[i][j]=Emax : currentE[i][j]-=0;

}

// take energy from organsim just for living

currentE[i][j] -= dE;

currentT[i][j] += 1;

// if there's not enough energy - DIE

if (currentE[i][j] <= 0 || currentT[i][j] > lifeTime)

{

currentState[i][j] = states[0];

currentE[i][j]=0;

currentT[i][j]=16;

}

}

}

}

}

// MOVE

for (var i = 1; i <= size; i++) {

for (var j = 1; j <= size; j++) {

if (currentState[i][j] == states[1])

{

var maxP = 0;

var maxI = 0;

var maxJ = 0;

for (var iOff = 1; iOff >= -1; iOff--) {

for (var jOff = 1; jOff >= -1; jOff--) {

if (currentP[i + iOff][j + jOff] >= maxP)

{

maxP = currentP[i + iOff][j + jOff];

maxI = iOff;

maxJ = jOff;

}

}

}

currentState[i+maxI][j+maxJ] = states[1];

//tmpState[i][j] = states[0];

currentState[i][j] = states[0];

currentE[i+maxI][j+maxJ] = currentE[i][j];

currentE[i][j] = 0;

currentT[i+maxI][j+maxJ] = currentT[i][j];

currentT[i][j] = 0;

}

}

}

// BORN

for (var i = 1; i <= size; i++) {

for (var j = 1; j <= size; j++) {

if (currentState[i][j] == states[1] && currentT[i][j] > 3 && currentE[i][j] > dR)

{

var offset = [];

offset[0] = Math.floor(Math.random() \* 3)-1;

offset[1] = Math.floor(Math.random() \* 3)-1;

while (currentState[i+offset[0]][j+offset[1]] == states[1] )

{

offset[0] = offsetRnd[Math.floor(Math.random() \* offsetRnd.length)];

offset[1] = offsetRnd[Math.floor(Math.random() \* offsetRnd.length)];

}

currentState[i][j] = states[1];

currentState[i+offset[0]][j+offset[1]] = states[1];

currentE[i][j] -= dR;

currentE[i+offset[0]][j+offset[1]] = currentE[i][j];

currentT[i+offset[0]][j+offset[1]] = 0;

}

}

}

update(currentState, currentP);

}, delay \*t);

}

function pseudoRandom() {

var notRandomNumbers = [0, 0, 0, 0, 0, 1, 1, 1, 1, 1];

var idx = Math.floor(Math.random() \* notRandomNumbers.length);

return notRandomNumbers[idx];

}

function draw(state) {

for (var i = 1; i <= size; i++) {

for (var j = 1; j <= size; j++) {

$('.wrap').append('<div data-h="'+i+'" data-v="'+j+'" class="el '+state[i][j] +'"></div>');

}

}

}

function update(state, P) {

$('.el').each(function(index){

var h = $(this).data("h");

var v = $(this).data("v");

if (currentState[h][v] == states[0])

{

$(this).removeClass("empty").removeClass("notempty").addClass(currentState[h][v]).css("opacity", 1);

}

else

{

$(this).removeClass("empty").removeClass("notempty").addClass(currentState[h][v]).css("opacity", 1);

}

})

}

$(document).ready(function(){

var count = 0;

for (var i = 0; i <= size+1; i++) {

var tmpP = [];

var tmpE = [];

var tmpT = [];

var tmpState = [];

for (var j = 0; j <= size+1; j++) {

tmpP[j] = Math.floor(Math.random() \* Pmax);

tmpE[j] = 25;

tmpT[j] = 0;

tmpState[j] = states[pseudoRandom()];

if (tmpState[j] == states[1])

count++;

}

currentState[i] = tmpState;

currentP[i] = tmpP;

currentE[i] = tmpE;

currentT[i] = tmpT;

}

for (var i = 0; i <= size+1; i++) {

currentP[i][0] = 0;

currentP[i][size+1] = 0;

currentP[0][i] = 0;

currentP[size+1][i] = 0;

}

console.log("N = " + count + "; A = " + Math.floor((count/size/size)\*100) + "%");

draw(currentState);

$('.wrap').css("width", sizePX+"px");

$('.wrap').css("height", sizePX+"px");

$('.el').css("width", "calc("+sizePX+"px/"+size+")");

$('.el').css("height", "calc("+sizePX+"px/"+size+")");

T = 0;

for (var t = 0; t<=T; t++) {

//console.log("t = " + t);

task(t);

}

})