# Задание

**2. Модель клеточного автомата «игра Жизнь»**

а) классический вариант игры Жизнь: поле – двумерная матрица, замкнутая в тор;

состояние автомата – бинарное (0-мертв, 1-жив);

S(t) – исходное состояние;

N(t) – количество «живых» соседей.

Описание переходов: S(t)=1,

N(t)<2 –> S(t+1)=0;

N(t)<4 –> S(t+1)=1;

N(t)=4 –> S(t+1)=0;

S(t)=0,

N(t)=2 –> S(t+1)=1.

б) игра Жизнь на ограниченном 2-мерном поле (за границей поля все пространство имеет состояние 0).

в) игра Жизнь с возрастом: каждый автомат S(t)=1 имеет счетчик возраста. По достижении предельного возраста автомат переходит в состояние 0 независимо от внешних условий.

г) учет состояния 8 соседей, включая соседей "по диагонали", правила функционирования формируются самостоятельно.

д) любой из вышеперечисленных вариантов в Игры Жизнь, но на 3-мерном поле.

Внимание! Для всех вариантов Игры Жизнь выход из программы происходит в случае выполнения хотя бы одного из условий:

- система пришла в стационарное состояние;

- достигнуто заданное число итераций.

# Представление поля в программе

Во всех случаях будем представлять поле в виде булевого многомерного массива состояния (жив, мёртв) автомата (для случая «в» - массива структур из состояния и возраста автомата), первый индекс которого указывает на ширину, второй – на высоту, третий – на глубину (для 3-мерного поля) положения автомата в поле.

# Построение нового поля

Во всех случаях новое поле строится на основе данных о количестве соседей каждого автомата (для случая «в» - ещё и возраста).

Листинг кода для классического варианта

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Обновление поля

field[WIDTH][HEIGHT] - текущее поле.

Возвращает true, если поле изменило своё состояние, иначе false

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool newField(bool field[WIDTH][HEIGHT])

{

int countNeighbors = 0; //Количество живых соседей

bool notStatState = false; //Возвращаемое значение

bool newField[WIDTH][HEIGHT];

for(int h = 0; h < HEIGHT; h++)

for(int w = 0; w < WIDTH; w++)

{

countNeighbors = 0;

countNeighbors += field[w == 0 ? WIDTH-1 : (w-1)] [h];

countNeighbors += field[(w+1)%WIDTH] [h];

countNeighbors += field[w] [h == 0 ? HEIGHT - 1 : (h-1)];

countNeighbors += field[w] [(h+1)%HEIGHT];

newField[w][h] = newState(field[w][h], countNeighbors);

if(newField[w][h] != field[w][h])

notStatState = true;

}

for(int h = 0; h < HEIGHT; h++)

for(int w = 0; w < WIDTH; w++)

field[w][h] = newField[w][h];

return notStatState;

}

Листинг кода для классического варианта (продолжение)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Получение нового состояния автомата в зависимости от старого состояния и количества соседей.

oldState - старое состояние автомата

neighborCnt - количество живых соседей

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool newState(const bool oldState, const int neighborCnt)

{

if(oldState == true)

{

if(neighborCnt < 2) return false;

if(neighborCnt < 4) return true;

if(neighborCnt == 4) return false;

}

if(oldState == false)

{

if(neighborCnt == 2) return true;

}

return oldState;

}

# Контрольные примеры работы программы (обновление поля в классическом варианте)

Для наглядности, будем живую автомата обозначать закрашенным символом (219 в таблице ASCII), а мёртвую – точкой (в конечной реализации – пробелом).

Пример 1.

Возьмём поле большого размера (25х20) для проверки выполнения условий перехода задании (рис. 1). Левая часть – начальное состояние поля. Правая – состояние поля после первой итерации (однократного вызова рассмотренной выше функции newField). Здесь показаны все возможные положения клеток и их соседей (для каждого состояния – от 0 до 4 соседей).

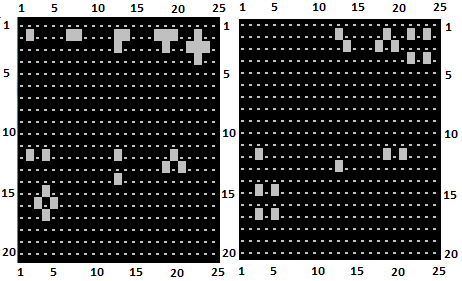


Рис.1. Результат работы однократного вызова функции newField для всех возможных состояний поля.

Пример 2.

Поле небольшого размера (6х14). Рассмотрим варианты для крайних случаев: несколько живых клеток у границы поля (рис. 2). Видим верный результат для клеток

(3, 1), (1, 4) и всех остальных.

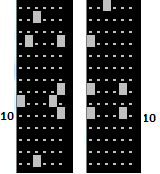


Рис.2. Результат работы однократного вызова функции newField для крайних случаев.

Пример 3.

Рассмотрим 2 случая для поля 1х2 (рис. 3). В данном случае каждый живой автомат (с состоянием 1) является своим левым и правым соседом.



Рис.3. Результат работы однократного вызова функции newField для поля 1х2.

Пример 4.

Случай для поля 3х2 (рис. 4).



Рис.4. Результат работы однократного вызова функции newField для поля 3х2.

Пример 5.

Два случая для поля 1х1 (рис. 5). Во втором автомат с состоянием 1 является своим левым, правым, верхним и нижним соседом.

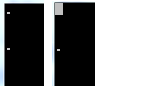


Рис.5. Результат работы однократного вызова функции newField для поля 1х1.

# Игра Жизнь на ограниченном 2-мерном поле

Данная реализация отличается лишь тем, что поле ограничено. Поэтому программный код отличается от предыдущего только функцией newField (здесь и далее ключевые изменения в коде будут выделены красным цветом).

Листинг кода для ограниченного поля.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Обновление поля

field[WIDTH][HEIGHT] - текущее поле.

Возвращает true, если поле изменило своё состояние, иначе false

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool newField(bool field[WIDTH][HEIGHT])

{

int countNeighbors = 0; //Количество живых соседей

bool notStatState = false; //Возвращаемое значение

bool newField[WIDTH][HEIGHT];

for(int h = 0; h < HEIGHT; h++)

for(int w = 0; w < WIDTH; w++)

{

countNeighbors = 0;

if(w > 0)

countNeighbors += field[w-1][h];

if(w < WIDTH-1)

countNeighbors += field[w+1][h];

if(h > 0)

countNeighbors += field[w][h-1];

if(h < HEIGHT-1)

countNeighbors += field[w][h+1];

newField[w][h] = newState(field[w][h], countNeighbors);

if(newField[w][h] != field[w][h])

notStatState = true;

}

for(int h = 0; h < HEIGHT; h++)

for(int w = 0; w < WIDTH; w++)

field[w][h] = newField[w][h];

return notStatState;

}

# Контрольные примеры работы программы (обновление ограниченного поля)

Рассмотрим примеры из классического варианта.

Пример 1.

Возьмём поле 25х25 из примера 1 для классического варианта (рис. 6). Обратим внимание, что ничего не изменилось.

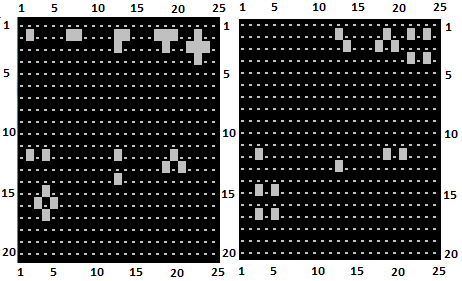


Рис.6. Результат работы однократного вызова функции newField для всех возможных состояний ограниченного поля.

Пример 2.

Рассмотрим несколько граничных случаев из примера 2 для классического варианта (рис. 7) для поля 6х14. Как видим, результат резко изменился:

клетки (автоматы) (3, 1), (1, 4) и другие ожившие в том примере остались мертвы.

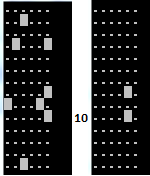


Рис 7. Результат работы однократного вызова функции newField для крайних случаев в ограниченном поле.

Пример 3.

Два примера с полем 1х2 (рис. 8). Теперь никакая клетка не является своим соседом и умирает.



Рис.8. Результат работы однократного вызова функции newField для ограниченного поля 1х2.

Пример 4.

Случай для поля 3х2 (рис. 9). По той же причине, что и в предыдущем примере, все живые клетки умерли.



Рис.9. Результат работы однократного вызова функции newField для ограниченного поля 3х2.

Пример 5.

Два случая для поля 1х1 (рис. 10). Во втором живая клетка теперь не является своим соседом, но результат остался прежним.



Рис.10. Результат работы однократного вызова функции newField для ограниченного поля 1х1.

# Игра Жизнь с возрастом

Данная реализация отличается от классической тем, что всякая живая клетка обладает возрастом, а по достижении предельного возраста умирает. Программный код отличается от классического описанием состояния каждого автомата с помощью структуры (жив/мёртв и возраст), и функцией обновления поля newField.

Листинг кода

typedef struct cell

{

bool currentState;

int age;

}cell;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Обновление поля

field[WIDTH][HEIGHT] - текущее поле.

Возвращает true, если поле изменило своё состояние, иначе false

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool newField(cell field[WIDTH][HEIGHT])

{

int countNeighbors = 0; //Количество живых соседей

bool notStatState = false; //Возвращаемое значение

cell newField[WIDTH][HEIGHT];

for(int h = 0; h < HEIGHT; h++)

for(int w = 0; w < WIDTH; w++)

{

countNeighbors = 0;

countNeighbors += field[w == 0 ? WIDTH-1 : (w-1)] [h] .currentState;

countNeighbors += field[(w+1)%WIDTH] [h] .currentState;

countNeighbors += field[w] [h == 0 ? HEIGHT - 1 : (h-1)].currentState;

countNeighbors += field[w] [(h+1)%HEIGHT] .currentState;

newField[w][h].currentState = newState(field[w][h].currentState, countNeighbors);

if(newField[w][h].currentState != field[w][h].currentState)

notStatState = true;

//Проверка на достижение максимального возраста

if(field[w][h].age >= MAX\_AGE)

newField[w][h].currentState = false;

//Обнуление возраста при смерти

if((newField[w][h].currentState == true) && (field[w][h].currentState == false))

field[w][h].age = 0;

//Увеличение возраста живой клетки

if((newField[w][h].currentState == true) && (field[w][h].currentState == true))

++field[w][h].age;

//Проверка на достижение максимального возраста

if(field[w][h].age >= MAX\_AGE)

newField[w][h].currentState = false;

}

for(int h = 0; h < HEIGHT; h++)

for(int w = 0; w < WIDTH; w++)

field[w][h].currentState = newField[w][h].currentState;

return notStatState;

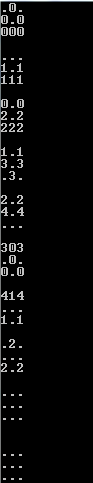
}

# Контрольные примеры работы программы (обновление поля игры «Жизнь» с возрастом)

Для наглядности, будем живую клетку обозначать цифрой с её возрастом (возраст начинается с нуля), а мёртвую – точкой. Поле в данном случае замкнуто в тор. За максимальный возраст возьмём число 5.

Пример 1.

Пример показывает выполнение основных функций программы: за одно обновление поля счётчик возраста каждого автомата увеличивается на 1, а при достижении предельного возраста, 5, автомат переходит в состояние 0 (не живой) вне зависимости от внешних условий.

  
  
Рис.11. Результат работы многократного вызова функции newField для поля 3х3.

Пример 2.

Наблюдаем аналогичную ситуацию.



Рис.12. Результат работы многократного вызова функции newField для поля 6х6.

# Игра Жизнь с учётом всех соседей.

Правила выбраны такие же, как в классическом варианте, с заменой

N(t) = 4 –> S(t+1)=0 на N(t) >= 4 –>S(t+1)=0, т. е. Автомат переходит из состояния 1 в состояние 0, если у него 4 или больше соседей.

В данной реализации код отличается функцией newField. (Добавлены проверки на наличие соседей по диагонали).

Листинг кода

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Обновление поля

field[WIDTH][HEIGHT] - текущее поле.

Возвращает true, если поле изменило своё состояние, иначе false

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool newField(bool field[WIDTH][HEIGHT])

{

int countNeighbors = 0; //Количество живых соседей

bool notStatState = false; //Возвращаемое значение

bool newField[WIDTH][HEIGHT];

for(int h = 0; h < HEIGHT; h++)

for(int w = 0; w < WIDTH; w++)

{

countNeighbors = 0;

countNeighbors += field[w == 0 ? WIDTH-1 : (w-1)] [h];

countNeighbors += field[(w+1)%WIDTH] [h];

countNeighbors += field[w] [h == 0 ? HEIGHT - 1 : (h-1)];

countNeighbors += field[w] [(h+1)%HEIGHT];

countNeighbors += field[w == 0 ? WIDTH-1 : (w-1)] [h == 0 ? HEIGHT - 1 : (h-1)];

countNeighbors += field[(w+1)%WIDTH] [(h+1)%HEIGHT];

countNeighbors += field[(w+1)%WIDTH] [h == 0 ? HEIGHT - 1 : (h-1)];

countNeighbors += field[w == 0 ? WIDTH-1 : (w-1)] [(h+1)%HEIGHT];

newField[w][h] = newState(field[w][h], countNeighbors);

if(newField[w][h] != field[w][h])

notStatState = true;

}

for(int h = 0; h < HEIGHT; h++)

for(int w = 0; w < WIDTH; w++)

field[w][h] = newField[w][h];

return notStatState;

}

Листинг кода (продолжение)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Получение нового состояния клетки в зависимости от старого состояния и количества соседей.

oldState - старое состояние клетки

neighborCnt - количество живых соседей

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool newState(const bool oldState, const int neighborCnt)

{

if(oldState == true)

{

if(neighborCnt < 2) return false;

if(neighborCnt < 4) return true;

if(neighborCnt >= 4) return false;

}

if(oldState == false)

{

if(neighborCnt == 2) return true;

}

return oldState;

}

# Контрольные примеры работы программы (обновление поля с учётом всех соседей)

Пример 1.

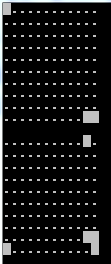


Рис.13. Результат работы однократного вызова функции newField для поля 12х10.

Пример 2.

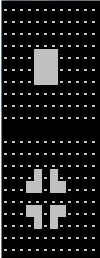


Рис.14. Результат работы однократного вызова функции newField для 9 автоматов с состоянием 1, расположенных в квадрате 3х3.

Пример 3.

Автоматы с состоянием 1 расположены в углах квадрата (рис. 15). Тогда для каждого углового автомата имеем 3 соседа, для каждого автомата, расположенного между угловыми автоматами 2 соседа, для каждого центрального — 1 сосед.



Рис.15. Результат работы однократного вызова функции newField для поля 4х4.

# Игра Жизнь на трёхмерном поле.

За основу возьмём вариант игры на ограниченном поле. Таким образом у каждого автомата будет не больше 6 соседей.

В данной реализации код отличается функцией newField. (Добавлены проверки для двух дополнительных соседей).

Листинг кода

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Обновление поля

field[WIDTH][HEIGHT][DEPTH] - текущее поле.

Возвращает true, если поле изменило своё состояние, иначе false

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool newField(bool field[WIDTH][HEIGHT][DEPTH])

{

int countNeighbors = 0; //Количество живых соседей

bool notStatState = false; //Возвращаемое значение

bool newField[WIDTH][HEIGHT][DEPTH];

for(int h = 0; h < HEIGHT; h++)

for(int w = 0; w < WIDTH; w++)

for(int d = 0; d < DEPTH; d++)

{

countNeighbors = 0;

if(w > 0)

countNeighbors += field[w-1][h][d];

if(w < WIDTH-1)

countNeighbors += field[w+1][h][d];

if(h > 0)

countNeighbors += field[w][h-1][d];

if(h < HEIGHT-1)

countNeighbors += field[w][h+1][d];

if(d > 0)

countNeighbors += field[w][h][d-1];

if(d < DEPTH-1)

countNeighbors += field[w][h][d+1];

newField[w][h][d] = newState(field[w][h][d], countNeighbors);

if(newField[w][h][d] != field[w][h][d])

notStatState = true;

}

for(int h = 0; h < HEIGHT; h++)

for(int w = 0; w < WIDTH; w++)

for(int d = 0; d < DEPTH; d++)

field[w][h][d] = newField[w][h][d];

return notStatState;

# }

# Контрольные примеры работы программы (игра на 3-мерном поле)

Выводить будем отдельно каждый слой (Layer).

Пример 1.

Проверка условия S(t)=0, N(t)=2 –> S(t+1)=1. Оно действительно выполняется.

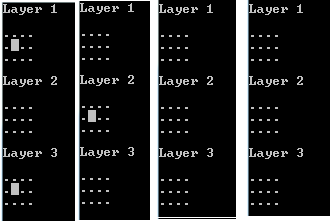


Рис.16. Результат работы многократного вызова функции newField для поля 4х3х3.

Пример 2.

Проверка условия S(t)=1, N(t)<2 –> S(t+1)=0.

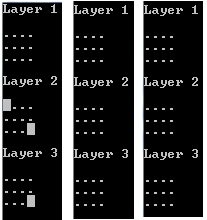


Рис.17. Результат работы многократного вызова функции newField для поля 4х3х3.

Пример 3.

Проверка условия N(t)=4 –> S(t+1)=0.

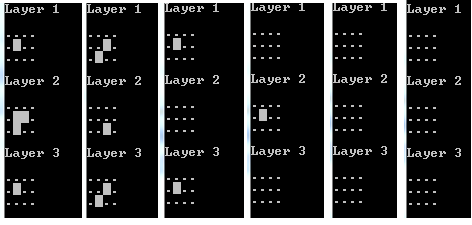


Рис.18. Результат работы многократного вызова функции newField для поля 4х3х3.

# Заключение

В ходе данной лабораторной была смоделирована игра «Жизнь» в различных реализациях: на замкнутом, ограниченном полях; с различными условиями: состояние автомата без возраста или с возрастом. Так же была смоделирована игра на трёхмерном поле.

Был проделан ряд тестов, подтверждающих корректное выполнение программы для заданных условий, а так же был разобран ряд критических случаев.