ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

Кафедра вычислительных систем

КУРСОВАЯ РАБОТА

по дисциплине «Технологии разработки программного обеспечения»

на тему «Утвержденная преподавателем тема»

Выполнил:
ст. гр. ИП-015
Голоктионов С. В.

Проверил:
доц., к.ф.-м.н. Пудов С. Г.

Новосибирск, 2022

Оглавление

Введение и постановка задачи....3
Техническое задание....4
Описание выполненного проекта....5
Подсчет соседей....5
Реализация правила....5
Конструктор класса....5
Вывод....5
Обновление....5
Main....6
Скриншоты выполнения....6
Личный вклад в проект....10
Приложение. Текст программы....11

Введение и постановка задачи

Реализовать на языке C++ клеточный автомат по правилам Игра “Жизнь”. Продемонстрировать его работу. Сделать код с возможность изменений правил под нужды. Реализовать красивый интерфейс отображения.

Техническое задание

Игра «Жизнь» (англ. Conway's Game of Life) — клеточный автомат, придуманный английским математиком Джоном Конвеем в 1970 году.

Правила

- Место действия этой игры — «вселенная» — это размеченная на клетки поверхность или плоскость — безграничная, ограниченная, или замкнутая (в пределе — бесконечная плоскость).
- Каждая клетка на этой поверхности может находиться в двух состояниях: быть «живой» (заполненной) или быть «мёртвой» (пустой). Клетка имеет восемь соседей, окружающих её.
- Распределение живых клеток в начале игры называется первым поколением. Каждое следующее поколение рассчитывается на основе предыдущего по таким правилам:
    - в пустой (мёртвой) клетке, рядом с которой ровно три живые клетки, зарождается жизнь;
    - если у живой клетки есть две или три живые соседки, то эта клетка продолжает жить; в противном случае, если соседей меньше двух или больше трёх, клетка умирает («от одиночества» или «от перенаселённости»)
- Игра прекращается, если
    - на поле не останется ни одной «живой» клетки
    - конфигурация на очередном шаге в точности (без сдвигов и поворотов) повторит себя же на одном из более ранних шагов (складывается периодическая конфигурация)
    - при очередном шаге ни одна из клеток не меняет своего состояния (складывается стабильная конфигурация; предыдущее правило, вырожденное до одного шага назад)

Эти простые правила приводят к огромному разнообразию форм, которые могут возникнуть в игре.

Игрок не принимает прямого участия в игре, а лишь расставляет или генерирует начальную конфигурацию «живых» клеток, которые затем взаимодействуют согласно правилам уже без его участия (он является наблюдателем).

Описание выполненного проекта

Для оптимизации, создадим одинаковые по размеру векторы из векторов:

```
std::vector<std::vector<bool>> ArrayFirst;

std::vector<std::vector<bool>> ArraySecond;
```

Мы выделяем память сразу на два вектора и поочередно записываем сначала в первый, а после во второй. Для проверки, из кого мы будем записывать создаем `bool isFirst;`

Также в дальнейшем это можно использовать для реализации нескольких подсчетов.

Для красивого вывода мы даем возможность изменить закрашенную или не закрашенную клетку. Для этого создаем

- `char keyYES;`
- `char keyNO;`

И для вывода номера шага создаем `int Gen;`

Подсчет соседей

Подсчет соседей будет производится в

```
byte calc(const int& x, const int& y, const std::vector<std::vector<bool>>& Array)
```

Куда поступают два индекса и ссылка на первый или второй массив, что бы была возможность сразу записывать информацию.

Далее идет на выбор, делать ли быстрые проверки или более точные, чтобы не выйти за пределы массива и в результате получаем кол-во соседей.

Реализация правила

Для начала получаем статус клетки и сумму в зависимости от “старого” массива.

Если клетка закрашена, то если сумма соседей 2 или 3 закрашиваем, в иных случаях нет. Если клетка не закрашена, то если сумма соседей 3 клетка будет закрашена.

Конструктор класса

В конструкторе сделали генерацию случайного поля, позволяем указать размеры, шанс закраски и значения символов для живой и не живой клетки. Можно было бы сделать перегрузки, но это не главная цель работы.

Вывод

Для вывода есть две функции `void PrintPlus()` `void Print()`

Они отличаются исключительно оформлением, второй выводит только поле, а первый делает рамочку и выводит доп. параметры. При желании можно было бы добавить время подсчета поля.

Обновление

Эту функцию стоило бы реализовать мульти поточно, главная задача, на каждую клетку применить функцию правила.

Также есть проверка на наличие обновлений `bool isUpdated()`

Но, к сожалению, если появится даже простая мигалка скажет, что изменения были, что, по сути, правда.

Main

Указываем начальное значение srand(100); создаем класс и начинаем циклично очищать консоль и обновлять поле, пока есть обновления функцией описанной выше. При желании можно дополнительно ограничить цикл с помощью переменной i.

Скриншоты выполнения

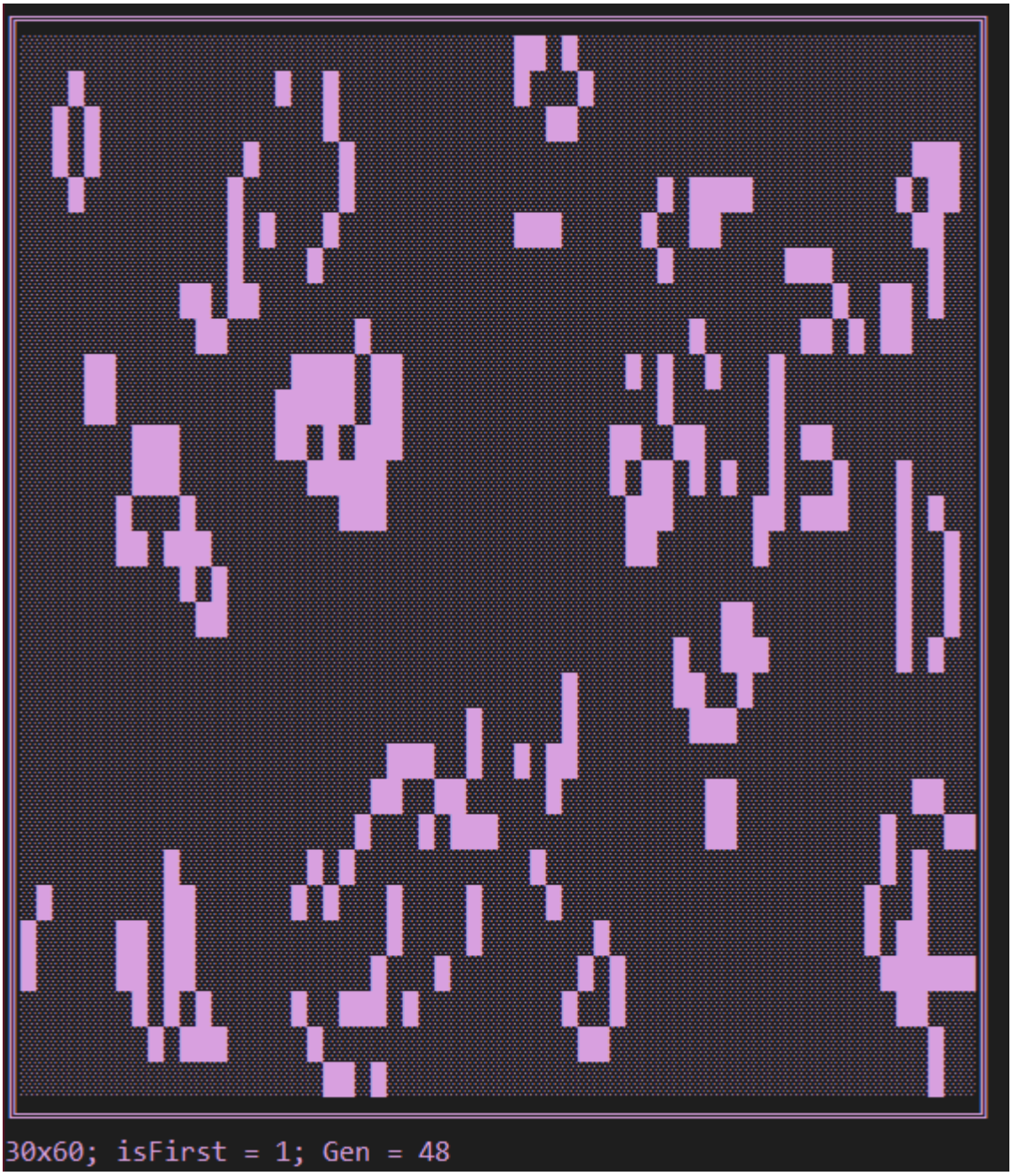

Рисунок 1- Результат работы

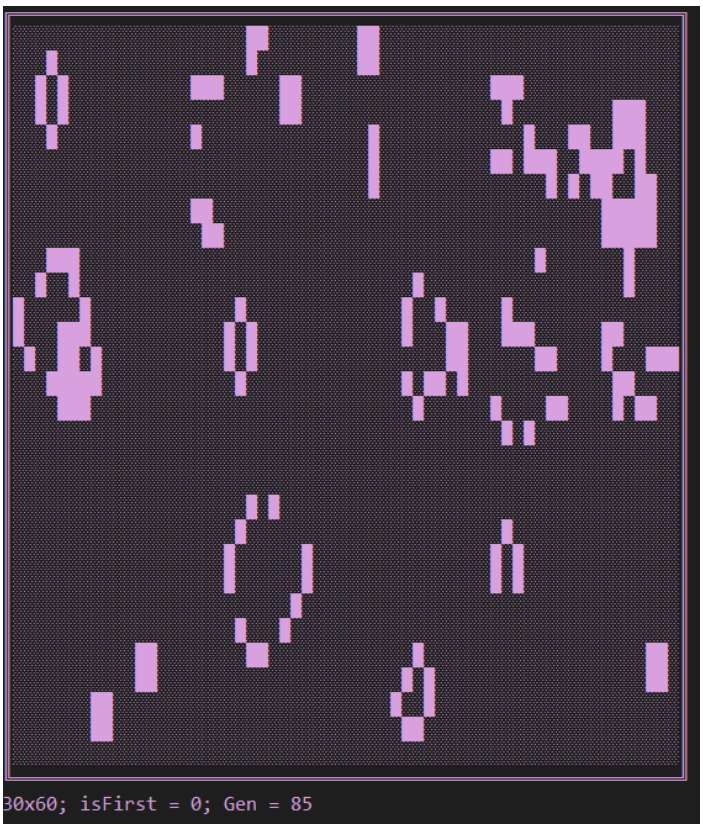

Рисунок 2- Результат работы

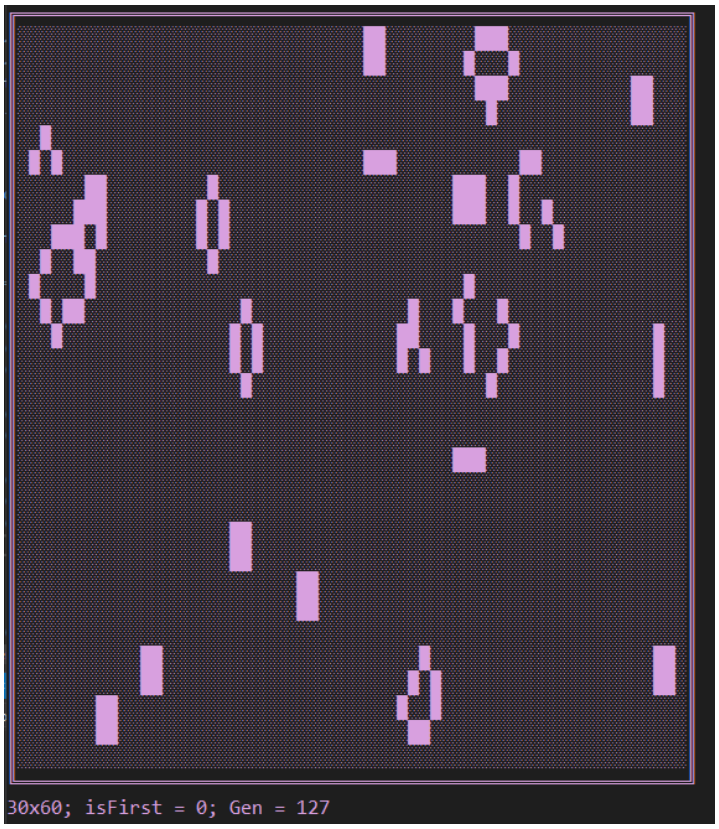

Рисунок 3- Результат работы

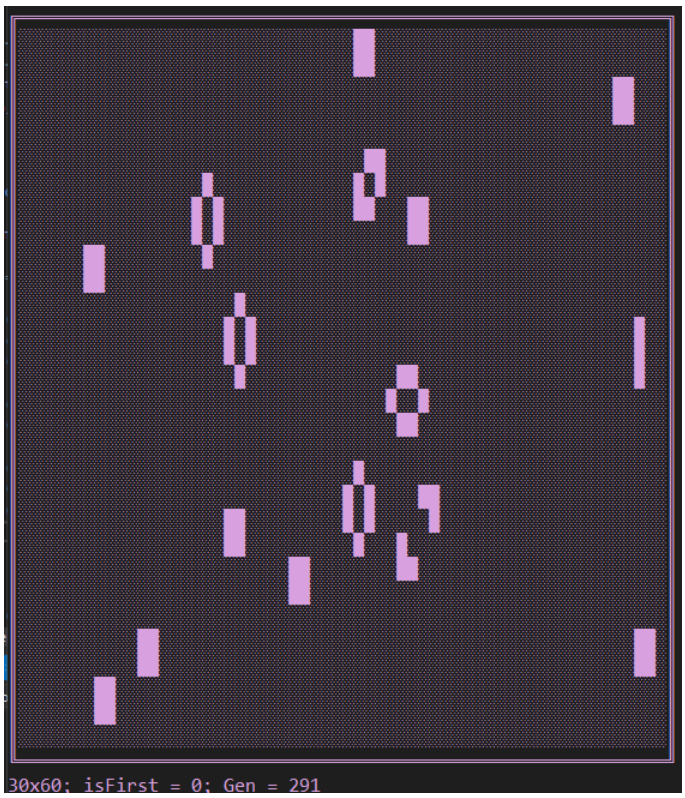

Рисунок 4- Результат работы

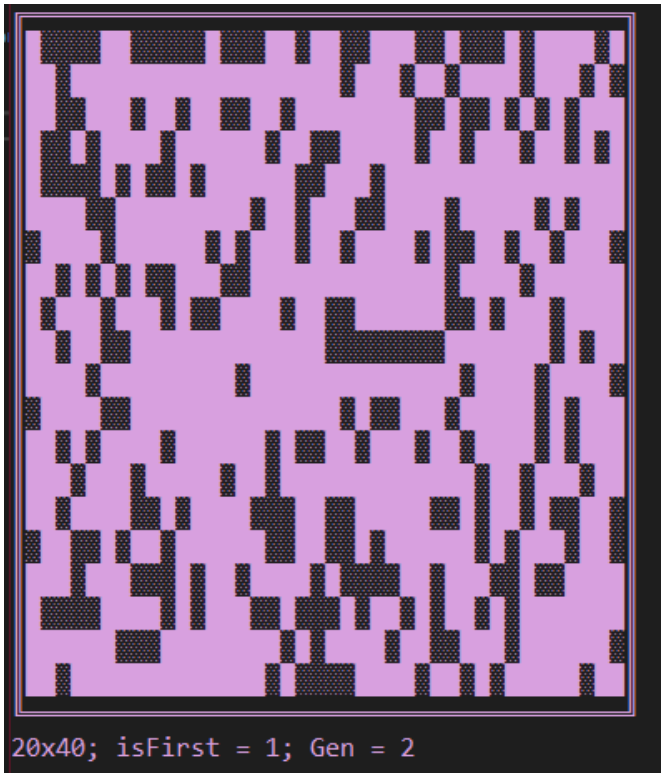

Рисунок 5- Результат работы

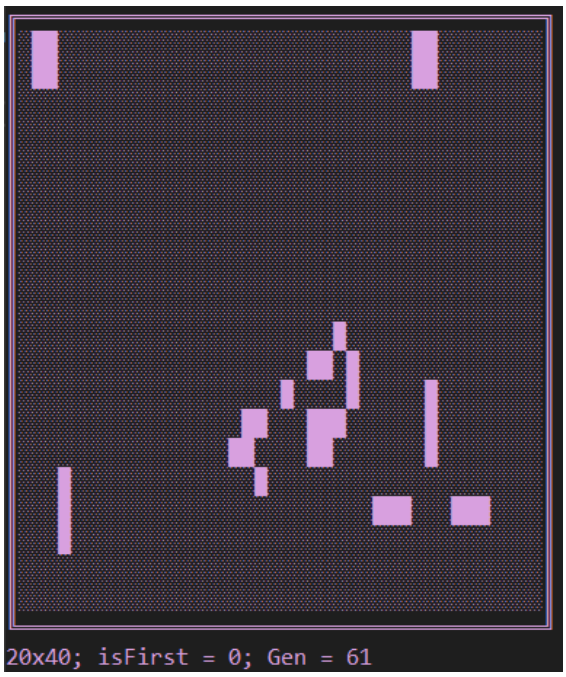

Рисунок 6- Результат работы

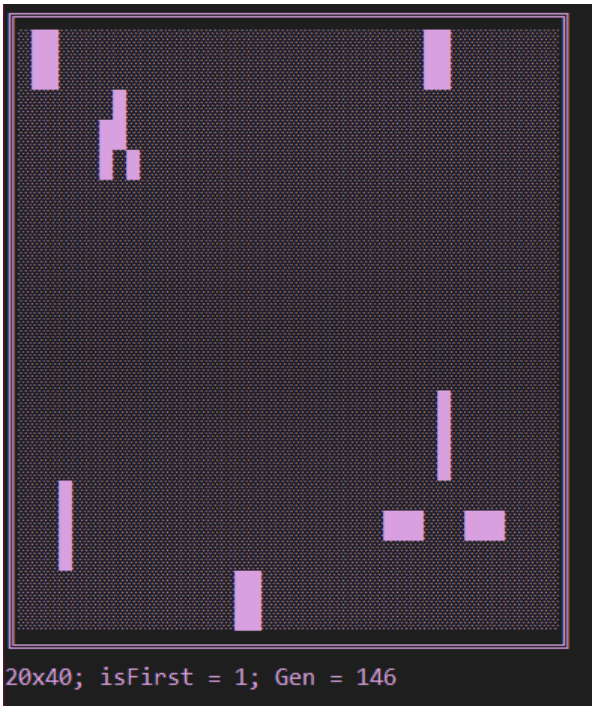

Рисунок 7- Результат работы

Личный вклад в проект

Оформил курсовую работу и тестировал код.

Приложение. Текст программы

```
#include <iostream>
#include <windows.h>
#include <vector>
#include <cstdlib>
#include <ctime>

using namespace std;

class GameLife {

private:
	std::vector<std::vector<bool> > ArrayFirst;
	std::vector<std::vector<bool> > ArraySecond;
	bool isFirst;
	char keyYES;
	char keyNO;
	int Gen;

	byte calc(const int& x, const int& y, const vector<vector<bool> >& Array)
	{
		byte buffer = 0;

		if (x >= 1 && y >= 1 && x < Array.size() - 1 && y < Array[0].size() - 1)
		{
			if (Array[x - 1][y - 1]) buffer++;
			if (Array[x][y - 1]) buffer++;
			if (Array[x + 1][y - 1]) buffer++;

			if (Array[x - 1][y]) buffer++;
			if (Array[x + 1][y]) buffer++;

			if (Array[x - 1][y + 1]) buffer++;
			if (Array[x][y + 1]) buffer++;
			if (Array[x + 1][y + 1]) buffer++;
		}
		else
		{
			if (y - 1 <= Array[0].size() && y - 1 >= 0)
			{
				if (x - 1 >= 0)
					if (Array[x - 1][y - 1]) buffer++;

				if (Array[x][y - 1]) buffer++;

				if (x + 1 < Array.size())
					if (Array[x + 1][y - 1]) buffer++;
			}

			if (y <= Array[0].size() && y >= 0)
			{
				if (x - 1 >= 0)
					if (Array[x - 1][y]) buffer++;

				if (x + 1 < Array.size())
					if (Array[x + 1][y]) buffer++;
```

```
                }

                if (y + 1 < Array[0].size() && y + 1 >= 0)
                {
                        if (x - 1 >= 0)
                                if (Array[x - 1][y + 1]) buffer++;

                        if (Array[x][y + 1]) buffer++;

                        if (x + 1 < Array.size())
                                if (Array[x + 1][y + 1]) buffer++;

                }
        }

        return buffer;
    }

    bool rule(const int& x, const int& y)
    {
        byte sum;
        bool status;

        if (isFirst)
        {
                status = ArraySecond[x][y];
                sum = calc(x, y, ArraySecond);

        }
        else
        {
                status = ArrayFirst[x][y];
                sum = calc(x, y, ArrayFirst);
        }

        if (status)
        {
                return (sum == 2 || sum == 3);
        }
        else
        {
                return (sum == 3);
        }
    }

public:
    GameLife()
    {
        int chance = 70; //1 - 100
        int sizeH = 20; //Height
        int sizeW = 40; //Width
        ArrayFirst.resize(sizeH);
        ArraySecond.resize(sizeH);
        isFirst = true;

        srand(static_cast<unsigned int>(time(0)));
        for (int i = 0; i < sizeH; i++)
        {
```

```
                ArrayFirst[i].resize(sizeW);
                ArraySecond[i].resize(sizeW);
                for (int j = 0; j < sizeW; j++)
                {
                        if ((100 - chance) < rand() % 100) ArrayFirst[i][j] = true;
                        else ArrayFirst[i][j] = false;

                        ArraySecond[i][j] = ArrayFirst[i][j];
                }
        }

        keyYES = (char)219;
        keyNO = (char)176;
        Gen = 0;
    }

    void PrintPlus()
    {
        if (!isFirst) ArrayFirst = ArraySecond;

        std::cout << (char)201;

        for (int j = 0; j < ArrayFirst[0].size(); j++)
            std::cout << (char)205;

        std::cout << (char)187;

        std::cout << std::endl;

        for (int i = 0; i < ArrayFirst.size();i++)
        {
            std::cout << (char)186;

            for (int j = 0; j < ArrayFirst[0].size(); j++)
                if (isFirst)
                {
                    if (ArrayFirst[i][j]) std::cout << keyYES;
                    else std::cout << keyNO;
                }
                else
                {
                    if (ArraySecond[i][j]) std::cout << keyYES;
                    else std::cout << keyNO;
                }
            std::cout << (char)186 << std::endl;
        }

        std::cout << (char)200;

        for (int j = 0; j < ArrayFirst[0].size(); j++)
            std::cout << (char)205;

        std::cout << (char)188;

        std::cout << endl << ArrayFirst.size() << "x" << ArrayFirst[0].size() << "; isFirst
= " << isFirst << "; Gen = " << Gen << endl;
    }
```

```
	void Print()
	{
		for (int i = 0; i < ArrayFirst.size(); i++)
		{
			for (int j = 0; j < ArrayFirst[0].size(); j++)
				if (isFirst)
				{
					if (ArrayFirst[i][j]) std::cout << keyYES;
					else std::cout << keyNO;
				}
				else
				{
					if (ArraySecond[i][j]) std::cout << keyYES;
					else std::cout << keyNO;
				}
			std::cout << std::endl;
		}
		std::cout << std::endl;
	}

	void Update()
	{
		Gen++;

		for (int i = 0; i < ArrayFirst.size(); i++)
			for (int j = 0; j < ArrayFirst[0].size(); j++)
			{
				if (isFirst) ArrayFirst[i][j] = rule(i, j);
				else ArraySecond[i][j] = rule(i, j);
			}

		isFirst = !isFirst;
	}

	bool isUpdated()
	{
		return!(ArrayFirst == ArraySecond);
	}
};
int main()
{
	srand(100);
	int i = 0;
	GameLife demo;
	demo.PrintPlus();

	bool play = true;
	while (play)
	{
		std::system("cls");
		demo.PrintPlus();
		demo.Update();

		if (i % 10 == 0) play = demo.isUpdated();
		Sleep(100);
		i++;
	}
}
```