

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ  
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»  
(МОСКОВСКИЙ ПОЛИТЕХ)

Факультет информационных технологий  
Кафедра «Инфокогнитивные технологии»

**КУРСОВОЙ ПРОЕКТ**

на тему: *«Основы объектно-ориентированного программирования.  
Реализация игры-симулятора «Жизнь»*

Направление подготовки 09.03.03 «Прикладная информатика»  
Профиль «Корпоративные информационные системы»

**Выполнил:**

студент группы 231-361

Ешуков Степан Алексеевич

14.03.2024 \_\_\_\_\_

(подпись)

Москва 2024

## Введение

Игра «Жизнь», придуманная английским математиком Джоном Конвеем в 1970 г., – это клеточный автомат, представляющий собой дискретную динамическую систему [1]. Его используют для моделирования процессов, происходящих при зарождении, развитии и гибели колонии живых организмов.

Действие игры происходит на бесконечной плоскости, разделенной на клетки. Каждая клетка имеет восемь соседей и может находиться в одном из двух состояний – живом или мертвом. Основная идея игры состоит в том, чтобы, начав с некоторой конфигурации живых клеток, проследить за ее эволюцией.

Начальная конфигурация живых клеток определяется игроком. Каждое следующее поколение колонии рассчитывается на основе предыдущего по следующим правилам («генетическим законам»):

- 1) в пустой (мертвой) клетке зарождается жизнь, если с ней соседствуют три живые клетки;
- 2) живая клетка продолжает жить, если с ней соседствуют две или три живые клетки;
- 3) живая клетка умирает от одиночества, если у нее меньше двух живых соседей;
- 4) живая клетка умирает от перенаселенности, если у нее больше трех.

Игра прекращается, если:

- 1) на поле не останется ни одной живой клетки;
- 2) конфигурация на очередном шаге в точности (без сдвигов и поворотов) повторит себя же на одном из более ранних шагов;
- 3) при очередном шаге ни одна из клеток не меняет своего состояния.

Часто исходные конфигурации клеток переходят в колебательный режим или обретают устойчивые состояния. Примером периодически повторяющейся фигуры со смещением может служить «планер» (glider), его особенностью является способность спонтанно возникать и перемещаться за пределы видимого пространства (рис. 1).

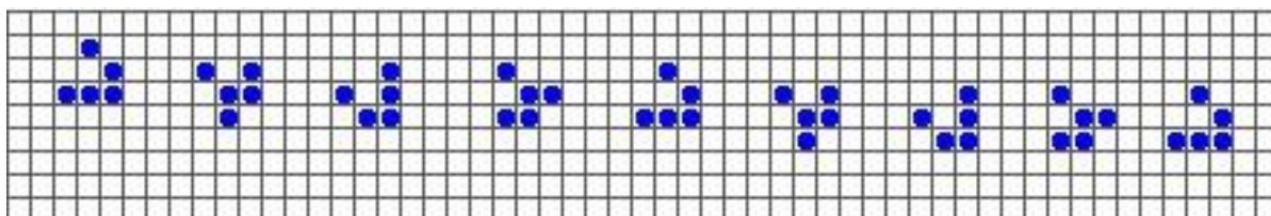


Рисунок 1 — Эволюция объекта «планер»

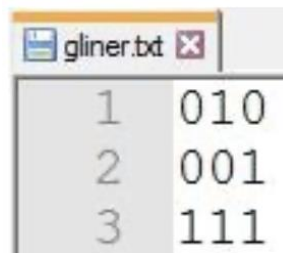
## 1. Постановка задачи

Целью настоящей работы является изучение основных свойств и преимуществ объектно-ориентированного программирования посредством реализации симулятора игры «Жизнь» на языке программирования Java [2].

Требования к программному решению:

- 1) Управление приложением должно осуществляться вводом команд из консоли.
- 2) Состояние игрового поля должно демонстрироваться в окне, созданном с помощью графической библиотеки Swing.
- 3) Размерность поля (число строк, число столбцов) должна задаваться пользователем.
- 4) Верхнюю границу поля следует считать продолжением нижней, а левую границу – продолжением правой.
- 5) Исходные колонии клеток должны задаваться текстовыми файлами - (пример представлен на рисунке 2).

- 6) Пользователю должна быть предоставлена возможность указать путь к файлу с описанием фигуры (или колонии) и ее местоположение на поле в виде координат левого верхнего угла минимального прямоугольника, ограничивающего фигуру.
- 7) Структура кода должна отвечать парадигме объектно-ориентированного программирования (рекомендуется как минимум создание классов Field и Cell).
- 8) Программа не должна содержать графические элементы управления и обрабатывать события.



1	010
2	001
3	111

Рисунок 2 – Пример описания конфигурации живых клеток

## 2 Проектирование и разработка приложения

1. Класс Main вызывает конструктор класса Field.
2. Класс Field отвечает за получение данных от пользователя , в нем метод CheckNeig осуществляет проверку на количество соседей и последующее изменение статуса клеток в зависимости от соседей, проверку закончена ли игра или нет. Метод isArrayEmpty поводит проверку пустой ли массив или нет. В дальнейшем от этого зависит будет ли заполнение нулями. Метод CheckGame проверяет наступил ли конец игры – есть ли живые клетки и меняется ли положение клеток.  
Объект класса имеет атрибуты :  
str и stlb (количество строк и столбцов ).
3. Класс Cell предназначен для отображения всех клеток в окно вывода , за это отвечает метод drawCells.
4. Класс ReadFail в нем метод ReadArray переносит в массив типа char данные из файла.

При запуске программы , класс Main вызывает конструктор класса Field который запрашивает у пользователя необходимую информацию для начала игры (Листинг 1) :

- количество строк и столбцов
- координаты (x,y) левого верхнего угла минимального прямоугольника, ограничивающего фигуру.
- путь к файлу с описанием фигуры

Идет проверка был ли это первый вход в цикл , если это так то массивы заполняются нулями. За проверку на первое вхождение отвечает метод isEmpty. При появлении игрового поля в окне графической библиотеки происходит процесс прорисовки заданной фигуры из файла , полученные данные переводятся в массив данных методом ReadArray (Листинг - 3) и пользователь может выбрать продолжить эволюцию нажав "e" или закончить игру нажав "q" в следствии чего программа будет завершена (Листинг - 4). Выбрав процесс Эволюции которая в дальнейшем изменятся по ранее описанным правилам. Клетки со статусом - мертвая ("0") остаются в таком же состоянии , либо изменяют статус на живая ("1"). У клеток со статусом - живая , эволюция проходит также , статус сохраняется или меняется на противоположный в зависимости от статуса клеток вокруг. Метод CheckNeighbors осуществляет подсчет живых соседей у каждой клетки и в зависимости от их количества записывает в новый массив состояние клетки ('0' или '1') , (Листинг - 5). После изменения статусов клеток изображение в игровом поле обновляется с помощью метода drawCells (Листинг - 6). И пользователь снова может выбрать дальнейшие действия программы продолжить эволюцию или завершить игру. Игра заканчивается если не осталось живых клеток или объект стал статичным - все клетки остаются на своих местах за данную функцию отвечает метод CheckGame (Листинг — 7).

Далее представлены листинги ключевых частей кода:

### Листинг 1 – ввод требуемых данных пользователем

```
Scanner sc = new Scanner(System.in);
System.out.println("Введите желаемое количество строк");
str = sc.nextInt();
System.out.println("Введите желаемое количество столбцов");
stlb = sc.nextInt();
System.out.println("Введите координату по X левого верхнего угла минимального
прямоугольника \n" +
    "ограничивающего фигуру (координатная ось X направлена вправо)");
int x = sc.nextInt();
System.out.println("Введите координату по Y левого верхнего угла минимального
прямоугольника \n" +
    "ограничивающего фигуру (координатная ось Y направлена вниз)");
int y = sc.nextInt();
System.out.println("Введите путь к файлу с описанием фигуры : ");
sc = new Scanner(System.in);
String file = sc.nextLine();
Readfail Fail = new Readfail();
Fail.ReadArray(file);
```

### Листинг2 – проверка первый ли раз вход в цикл (заполнен ли массив)

```
public boolean isArrayEmpty(char[][] array) {
    for (int i = 0; i < array.length; i++) {
        for (int j = 0; j < array[i].length; j++) {
            if (array[i][j] != '\0') {
                return false;
            }
        }
    }
    return true;
}
```

### Листинг 3 – Заполнение массива данными из файла введенными ранее пользователем

```
public void ReadArray(String filename) {
    try {
        FileReader fileReader = new FileReader(filename);
        BufferedReader bufferedReader = new BufferedReader(fileReader);
        List<String> lines = new ArrayList<>();
        String line;
        while ((line = bufferedReader.readLine()) != null) {
            lines.add(line);
        }
        int numRows = lines.size();
        int numCols = lines.get(0).length();
        charArray = new char[numRows][numCols];
        for (int i = 0; i < numRows; i++) {
            for (int j = 0; j < numCols; j++) {
                charArray[i][j] = lines.get(i).charAt(j);
            }
        }
        bufferedReader.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

### Листинг 4 – Запрос пользователю продолжить игру или завершить ее

```
while (true) {
    System.out.println("Нажмите е , чтобы произошла эволюция");
    System.out.println("Нажмите q , чтобы завершить игру");
    sc = new Scanner(System.in);
    String line = sc.nextLine();
    switch (line) {
        case "e":
            if (isArrayEmpty(FutureArray)) { //первый запуск - пустой
                МАССИВ
                // Массив не заполнен, заполняем значениями '0'
                for (int i = 0; i < str; i++) {
                    for (int j = 0; j < stlb; j++) {
                        FutureArray[i][j] = '0';
                    }
                }
                for (int i = 0; i < str; i++) {
                    for (int j = 0; j < stlb; j++) {
                        newArray[i][j] = '0';
                    }
                }
            }
        }
    }
}
```

```

        }
        System.out.println("Массивы были заполнены значениями
'0'.");

        //комбинирование двух массивов : из файла и newArray
        for (int i = 0; i < (Fail.charArray.length); i++) {
            for (int j = 0; j < (Fail.charArray[i].length); j++)
            {
                FutureArray[i + x][j + y] =
Fail.charArray[i][j];
            }
        }
        System.out.println("Комбинация двух массивов ");
        Cell.drawCells(FutureArray);
        continue;
    }
    for (int i = 0; i < str; i++) { // два массива становятся
одинаковыми
        for (int j = 0; j < stlb; j++) {
            newArray[i][j] = FutureArray[i][j];
        }
    }
    Cell.drawCells(FutureArray); // прорисовка
    CheckNeig(newArray, FutureArray); // подсчет соседей
    if (CheckGame(newArray, FutureArray)) break; // проверка на
конец игры
        break;
    case "q":
        System.out.println("Игра окончена");
        break;
    }
    if (line.equals("q")) break;
}

```

## Листинг 5 – Метод для обновления графического изображения

```

public static JFrame frame = new JFrame("Cells Field");
public static void drawCells(char[][] array) {
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JPanel panel = new JPanel() {
        protected void paintComponent(Graphics g) {
            super.paintComponent(g);
            int cellSize = 20; // Размер клетки
            int startX = 0; // Начальная координата X вывода клеток
            int startY = 0; // Начальная координата Y вывода клеток
            for (int i = 0; i < array.length; i++) {
                for (int j = 0; j < array[i].length; j++) {
                    if (array[i][j] == '1') {
                        g.setColor(Color.BLUE);
                        g.fillRect(startX + j * cellSize, startY + i *
cellSize, cellSize, cellSize);
                    } else if (array[i][j] == '0') {

```



```

        g.setColor(Color.WHITE);
        g.drawRect(startX + j * cellSize, startY + i *
cellSize, cellSize, cellSize);
    }
}
}
};
frame.add(panel);
frame.setSize(300, 300);
frame.setVisible(true);
}

```

**Листинг 6 – Метод определяющий состояние клетки в следующем ходу, а также определяющий количество живых клеток вокруг**

```

public static void CheckNeig(char[][] array, char[][] Farray) {
    // проверка соседей для каждой клетки
    for (int i = 0; i < str; i++) {
        for (int j = 0; j < stlb; j++) {
            int count = 0;
            // проверяем всех соседей клетки
            for (int x = -1; x <= 1; x++) {
                for (int y = -1; y <= 1; y++) {
                    // игнорируем саму клетку
                    if (x == 0 && y == 0) continue;
                    // учитываем краевые условия
                    int neighborX = (i + x + stlb) % stlb;
                    int neighborY = (j + y + str) % str;
                    // проверяем соседей на наличие живых клеток
                    if (array[neighborX][neighborY] == '1') count++;
                }
            }
            if (count == 3) Farray[i][j] = '1';
            else if (count != 2) Farray[i][j] = '0';
        }
    }
}
}

```

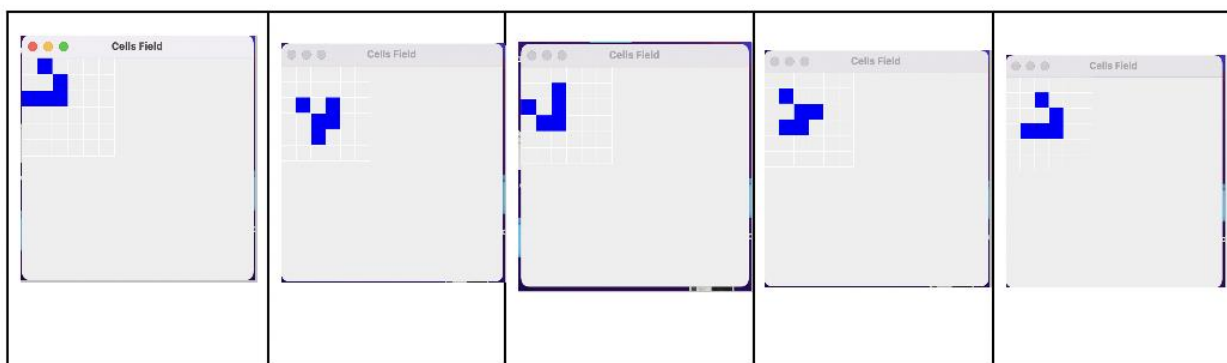
## Листинг 7 – Метод проверка наступил конец игры или нет

```
public static boolean CheckGame(char[][] array, char[][] Farray) {
    int kol1 = 0;
    int kol2 = 0;
    for (int i = 0; i < str; i++) { //проверка на статику
        for (int j = 0; j < stlb; j++) {
            if (Farray[i][j] != array[i][j]) kol1+=1;
        }
    }
    if (kol1==0) {
        System.out.println("GAME OVER !!!");
        System.exit(0);
        return true;
    }
    for (int i = 0; i < str; i++) { // проверка есть ли живые клетки
        for (int j = 0; j < stlb; j++) {
            if (Farray[i][j] != '0') kol2+=1;
        }
    }
    if (kol2==0){
        System.out.println("GAME OVER !!!");
        System.exit(0);
        return true;
    }
    else return false;
}
```

### 3 Тестирование и сценарии работы в приложении

Для тестирования программы были использованы начальные колонии типа «Планер» и «Восьмерка».

Эволюция объекта «Планер» представлена на Рисунке 3. После 4 созданий новых поколений объект «Планер» сместился вниз на одну клетку и вправо на одну клетку. Эволюция объекта «Восьмерка» представлена на Рисунке 4. Изначально объект похож на цифру 8, цикл эволюции состоит из 8 этапов создания новых поколений. Начальный объект после полного цикла



остается на месте.

Рисунок 3 – эволюция объекта «Планер»

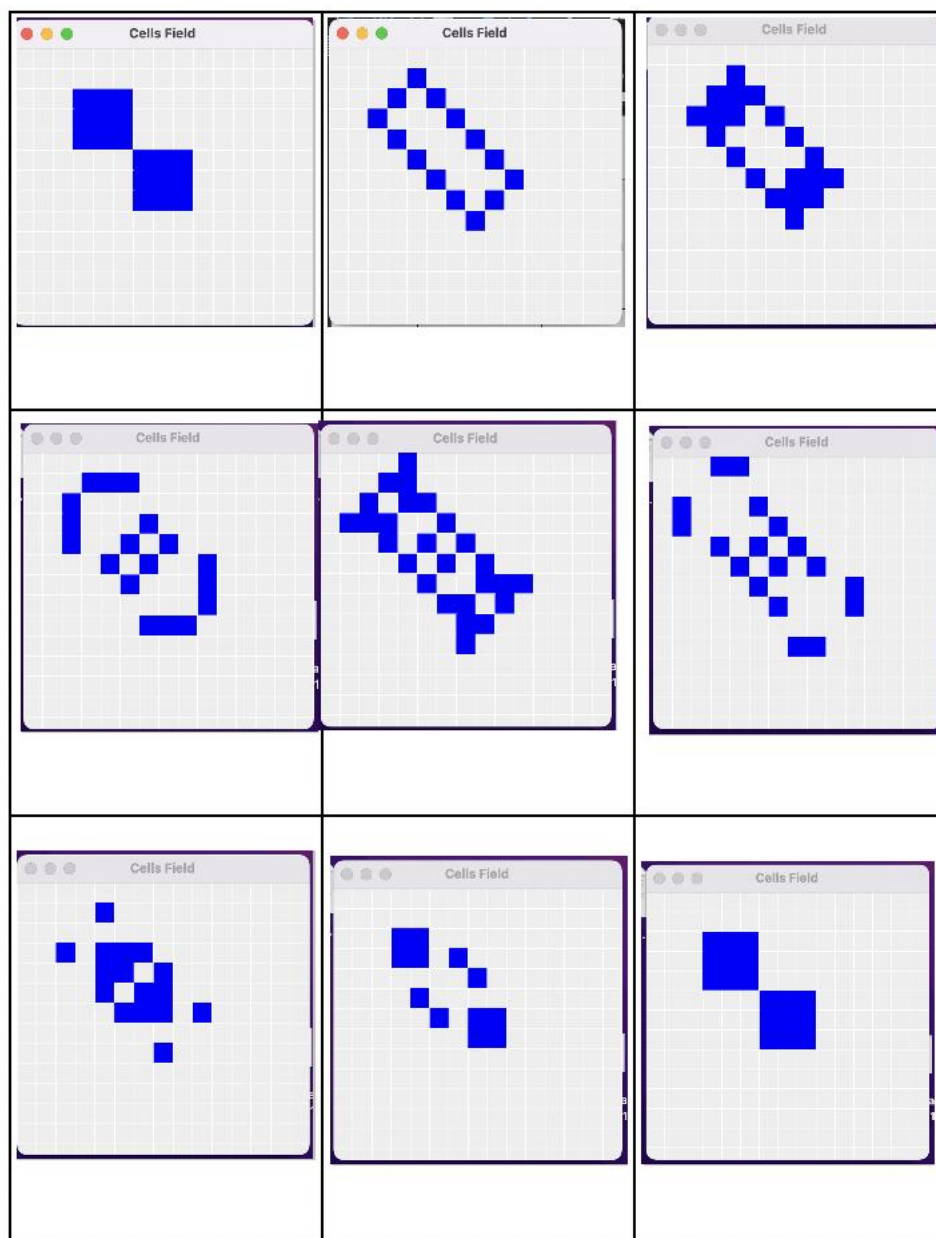


Рисунок 4 - эволюция объекта «Восьмерка»

## Заключение

В рамках курсового проекта по дисциплине «Основы программирования» было разработано отвечающее парадигме объектно-ориентированного программирования программа , которая моделирует эволюционные изменения колонии живых организмов и может быть использовано как основа для решения различных прикладных задач . Данный проект позволяет

пользователю самостоятельно разрабатывать начальные положения клеток ,  
изменять доступные параметры и производить анализ полученных данных .

### **Список литературы и интернет-ресурсов**

1. Клумова И. Игра «Жизнь» [Электронный ресурс]. URL: [http://kvant.mccme.ru/1974/09/igra\\_zhizn.htm](http://kvant.mccme.ru/1974/09/igra_zhizn.htm) (дата обращения: 01.12.2022).

2. Java Tutorials Learning Paths [Электронный ресурс]. URL: <https://docs.oracle.com/javase/tutorial/tutorialLearningPaths.html> (дата обращения: 01.12.2022).