



## Trabalho de Teste de Software - ( Parte II )

João Pedro Sequeto Nascimento - 207176022  
Josué de Oliveira Delgado Heringer - 201876023  
Marcelo Gonçalves de Souza Costa - 201776016

## 1) Especificação do Programa: Jogo da Vida

O jogo da vida consiste em um tabuleiro plano de dimensões 6x6, em que cada posição possui um valor: 1 - corresponde a uma célula viva e 0 - corresponde a uma célula morta. O jogo começa com uma configuração inicial, gerada aleatoriamente. A partir desta configuração cada passo, uma nova geração é obtida, de acordo com as seguintes regras

- Qualquer célula viva com menos de dois vizinhos vivos morre de solidão
- Qualquer célula viva com mais de três vizinhos morre de superpopulação
- Qualquer célula morta com exatamente três vizinhos vivos se torna uma célula viva
- Qualquer célula com dois vizinhos vivos continua no mesmo estado para a próxima geração.

O jogo não tem fim, assim, o usuário pode a cada passo escolher uma nova geração ou finalizar o jogo. A cada passo é mostrado ao usuário a geração anterior e a geração atual.

Considerando que a entrada do problema corresponda aos seguintes elementos (célula X, vizinho 1, vizinho 2, vizinho 3, vizinho 4, vizinho 5, vizinho 6, vizinho 7, vizinho 8).

## 2) Explicação do Código Fonte

O código foi dividido em 5 funções diferentes (print, returnNextStep, returnSumNeighbors, generateNextSteps, generateRandom ).

- Print => função responsável por imprimir a geração atual do jogo da vida e a geração anterior.

```

public static void print(int[][] actual, int[][] previous, int acumulate, int boardDimension) {
    System.out.println("\n" + acumulate + "ª Iteração");
    System.out.println("\t Anterior \t \t || \t Atual");
    for (int j = 0; j < boardDimension; j++) {
        for (int i = 0; i < boardDimension; i++) {
            System.out.print("[ " + previous[j][i] + " ]");
        }
        System.out.print("\t || \t");
        for (int i = 0; i < boardDimension; i++) {
            System.out.print("[ " + actual[j][i] + " ]");
        }
        System.out.println("");
    }
}

```

Figura 2.1: Código Fonte da função print

- GenerateRandom => função utilizada para gerar a primeira geração do programa do jogo da vida. Foi necessário a importação da lib Random para que fosse gerado o valor booleano.

```

public static int[][] generateRandom(int[][] actualRound, int boardDimension){
    Random random = new Random();
    for (int j = 0; j < boardDimension; j++) {
        for (int i = 0; i < boardDimension; i++) {
            actualRound[j][i] = random.nextBoolean() == true ? 1 : 0;
        }
    }
    return actualRound;
}

```

Figura 2.2: Código Fonte da função generateRandom

- GenerateNextStep => função responsável por gerar a próxima geração do jogo da vida, sendo assim a função necessita utilizar outras duas funções auxiliares ( returnSumNeighbors e returnNextStep ). Sendo assim, é passado como parâmetro a geração atual e o tamanho do tabuleiro ( por ser um tabuleiro 6x6 esse valor corresponde ao número 6 ).

```

public static int[][] generateNextSteps(int[][] actualRound, int boardDimension) {
    int[][] nextRound = new int[boardDimension][boardDimension];
    int boardDimensionMinusOne = boardDimension - 1;
    for (int j = 0; j < boardDimension; j++) {
        for (int i = 0; i < boardDimension; i++) {
            int sumNeighbors = returnSumNeighbors(i,j,boardDimensionMinusOne, actualRound);
            nextRound[j][i] = returnNextStep(sumNeighbors, actualRound[j][i]);
        }
    }
    return nextRound;
}

```

Figura 2.3: Código Fonte da função generateNextSteps

- ReturnNextStep => a função responsável por executar as retrições propostas pelo jogo, visto que ela recebe como parâmetro o total de vizinhos que o elemento que está sendo iterado contem e o valor que ele tem na geração atual. Sendo assim, é realizado a validação se o elemento tem 2 vizinhos, está vivo e com menos de dois vizinhos, está vivo e com mais de três vizinhos, está morto com exatamente 3 vizinhos e caso nenhum dos casos seja atendido, deve ser gerado um valor randomizado.

```

public static int returnNextStep(int totalNeighbors, int element) {
    Random random = new Random();
    if (totalNeighbors == 2) {
        return element;
    } else if (element == 1 && totalNeighbors < 2) {
        return 0;
    } else if (element == 1 && totalNeighbors > 3) {
        return 0;
    } else if (element == 0 && totalNeighbors == 3) {
        return 1;
    } else {
        return random.nextBoolean() == true ? 1 : 0;
    }
}

```

Figura 2.4: Código Fonte da função returnNextStep

- ReturnSumNeighbors => Função responsável por contar o total de vizinhos de cada elemento. Sendo assim, foi dividido em diversos cenários, para que fosse possível acessar os elementos dentro de um array. Na imagem, temos um tabuleiro onde é possível visualizar diferentes letras e marcações, ou seja, cada letra e marcação corresponde a um possível cenário no programa. O programa é composto por 9 cenários.

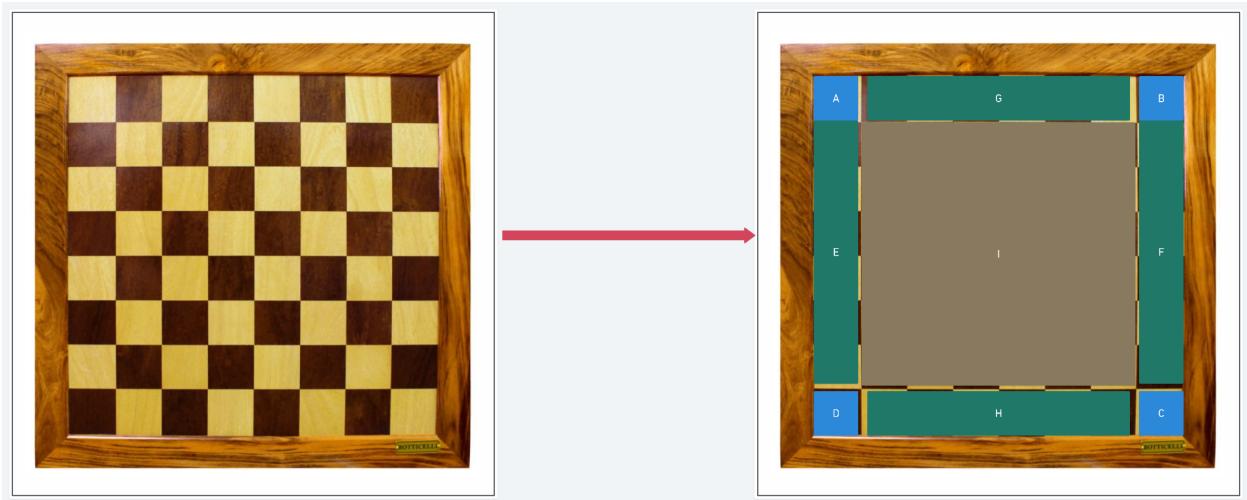


Figura 2.5: Tabuleiro vazio e um tabuleiro com os cenários marcados

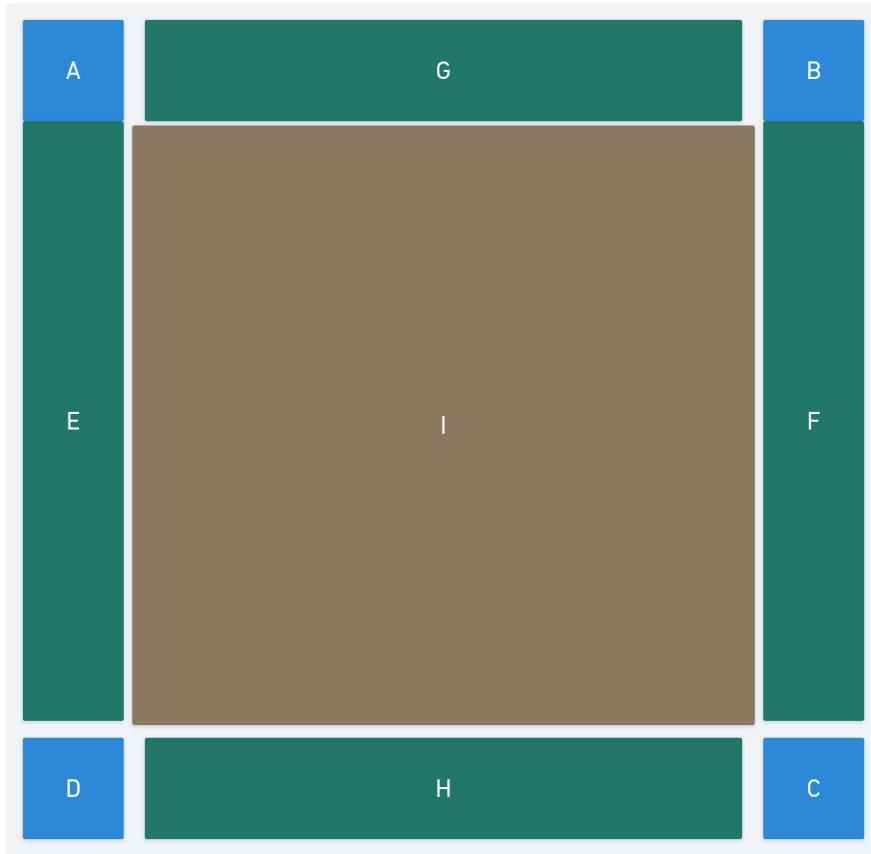


Figura 2.6: Destaque para os cenários com o intuito de melhorar a visualização

- Os cenários A ( 0,0 ), B ( 0,5 ), C ( 5,5 ) e D(5,0) tem um funcionamento muito semelhante, porém por ser um array, é necessário separá-los em casos distintos, pois, ao analisar os vizinhos do A devemos buscar o primeiro a direta, porém no caso do B essa posição é inválida e o programa pararia de funcionar. Sendo assim, nesses casos temos que pegar os 3 vizinhos e

realizar a soma ( Isso é possível pelo fato de que o array é composto por 0 e 1 ).

- Os cenários G (0, valores de 1 até 5), E (valores de 1 até 5 , 0), F (valores de 1 até 5,5) e H ( 5, valores de 1 até 5 ). Nesse caso, temos um total de 5 vizinhos em cada e será necessário somar todos para encontrar o total de vizinhos de cada elemento.
- O Cenários I é o cenário onde temos 8 vizinhos em um único elemento, ele é composto por qualquer valor que esteja posicionado fora das extremidades, tanto na vertical quanto horizontal, ou seja, (1 até 4, 1 até 4 ).

```
public static int returnSumNeighbors(int i, int j, int boardDimensionMinusOne, int[][][] actualRound){  
    int sumNeighbors = 0;  
    if (j != 0 && j != boardDimensionMinusOne && i != 0 && i != boardDimensionMinusOne) {  
        sumNeighbors = actualRound[j - 1][i] + actualRound[j - 1][i + 1] + actualRound[j][i + 1]  
                    + actualRound[j + 1][i + 1] + actualRound[j + 1][i] + actualRound[j + 1][i - 1]  
                    + actualRound[j][i - 1] + actualRound[j - 1][i - 1];  
    } else if (i > 0 && i < boardDimensionMinusOne && j == 0) {  
        sumNeighbors = actualRound[j][i + 1] + actualRound[j + 1][i + 1] + actualRound[j + 1][i]  
                    + actualRound[j + 1][i - 1] + actualRound[j][i - 1];  
    } else if (i > 0 && i < boardDimensionMinusOne && j == boardDimensionMinusOne) {  
        sumNeighbors = actualRound[j][i - 1] + actualRound[j - 1][i - 1] + actualRound[j - 1][i]  
                    + actualRound[j - 1][i + 1] + actualRound[j][i + 1];  
    } else if (j > 0 && j < boardDimensionMinusOne && i == 0) {  
        sumNeighbors = actualRound[j - 1][i] + actualRound[j - 1][i + 1] + actualRound[j][i + 1]  
                    + actualRound[j + 1][i + 1] + actualRound[j + 1][i];  
    } else if (j > 0 && j < boardDimensionMinusOne && i == boardDimensionMinusOne) {  
        sumNeighbors = actualRound[j - 1][i] + actualRound[j - 1][i - 1] + actualRound[j][i - 1]  
                    + actualRound[j + 1][i - 1] + actualRound[j + 1][i];  
    } else if (j == 0 && i == 0) {  
        sumNeighbors = actualRound[j][i + 1] + actualRound[j + 1][i + 1] + actualRound[j + 1][i];  
    } else if (j == 0 && i == boardDimensionMinusOne) {  
        sumNeighbors = actualRound[j][i - 1] + actualRound[j + 1][i - 1] + actualRound[j + 1][i];  
    } else if (j == boardDimensionMinusOne && i == 0) {  
        sumNeighbors = actualRound[j - 1][i] + actualRound[j - 1][i + 1] + actualRound[j][i + 1];  
    } else if (j == boardDimensionMinusOne && i == boardDimensionMinusOne) {  
        sumNeighbors = actualRound[j][i - 1] + actualRound[j - 1][i - 1] + actualRound[j - 1][i];  
    }  
  
    return sumNeighbors;  
}
```

Figura 2.7: Código Fonte da função returnSumNeighbors

### 3) EclEmma e Baduino - Parte II - A

#### 3.1) EclEmma

Element	Coverage	Covered Instructions	Missed Instructions
Trab	28.3 %	245	621
src	28.3 %	245	621
(default package)	28.3 %	245	621
Main.java	10.3 %	71	619
MainTest.java	98.9 %	174	2

Figura 3.0 - Cobertura de código com eclEmma

#### 3.2) Baduino

Project	DUAs Coverage
Trab	(64/284) (22.54%)
Main	(64/284) (22.54%)
Main	(28/40) (70.00%)
MainTest	(36/244) (14.75%)
main(String[])	(0/27) (0.00%)
generateRandom(int[][], int)	(22/24) (91.67%)
generateNextSteps(int[][], int)	(0/29) (0.00%)
returnSumNeighbors(int, int, int, int[][])	(0/109) (0.00%)
returnNextStep(int, int)	(14/17) (82.35%)
print(int[][], int[][], int, int)	(0/38) (0.00%)

Figura 4.1: Cobertura geral dos métodos da classe Main

Project	DUAs Coverage
static int[][] generateRandom(int[], int)	(22/24) (91.67%)
(36, 39, actualRound)	✓ true
(36, 43, actualRound)	✓ true
(36, (37, 43), boardDimension)	✓ true
(36, (37, 38), boardDimension)	✓ true
(36, (38, 37), boardDimension)	✓ true
(36, (38, 39), boardDimension)	✓ true
(36, (39, 39), random)	✓ true
(36, (39, 39), random)	✓ true
(37, (37, 43), j)	✗ false
(37, (37, 38), j)	✓ true
(37, (39, j))	✓ true
(37, (37, j))	✓ true
(38, (38, 37), i)	✗ false
(38, (38, 39), i)	✓ true
(38, (39, i))	✓ true
(38, (38, i))	✓ true
(38, (38, 37), i)	✓ true
(38, (38, 39), i)	✓ true
(38, (39, i))	✓ true
(38, (38, i))	✓ true
(38, (37, 43), j)	✓ true
(37, (37, 38), j)	✓ true
(37, (39, j))	✓ true
(37, (37, j))	✓ true

Figura 3.2 - Cobertura do método generateRandom(int[], int)

▼ static int[][] generateNextSteps(int[], int)	(0/29) (0.00%)
(47, 51, actualRound)	✗ false
(47, 52, actualRound)	✗ false
(47, (49, 55), boardDimension)	✗ false
(47, (49, 50), boardDimension)	✗ false
(47, (50, 49), boardDimension)	✗ false
(47, (50, 51), boardDimension)	✗ false
(47, 52, nextRound)	✗ false
(47, 55, nextRound)	✗ false
(48, 51, boardDimensionMinusOne)	✗ false
(49, (49, 55), j)	✗ false
(49, (49, 50), j)	✗ false
(49, 51, j)	✗ false
(49, 52, j)	✗ false
(49, 49, j)	✗ false
(50, (50, 49), i)	✗ false
(50, (50, 51), i)	✗ false
(50, 51, i)	✗ false
(50, 52, i)	✗ false
(50, 50, i)	✗ false
(50, (50, 49), i)	✗ false
(50, (50, 51), i)	✗ false
(50, 51, i)	✗ false
(50, 52, i)	✗ false
(50, 50, i)	✗ false

Figura 3.3.1: Cobertura do método generateNextSteps(int[], int)

Project	DUAs Coverage
• (47, 55, nextRound)	✗ false
• (48, 51, boardDimensionMinusOne)	✗ false
• (49, (49, 55), j)	✗ false
• (49, (49, 50), j)	✗ false
• (49, 51, j)	✗ false
• (49, 52, j)	✗ false
• (49, 49, j)	✗ false
• (50, (50, 49), i)	✗ false
• (50, (50, 51), i)	✗ false
• (50, 51, i)	✗ false
• (50, 52, i)	✗ false
• (50, 50, i)	✗ false
• (50, (50, 49), i)	✗ false
• (50, (50, 51), i)	✗ false
• (50, 51, i)	✗ false
• (50, 52, i)	✗ false
• (50, 50, i)	✗ false
• (49, (49, 55), j)	✗ false
• (49, (49, 50), j)	✗ false
• (49, 51, j)	✗ false
• (49, 52, j)	✗ false
• (49, 49, j)	✗ false
► • static int returnSumNeighbors(int, int, int, int[][])	⌚ (0/109) (0.00%)
► • static int returnNextStep(int, int)	⌚ (14/17) (82.35%)
► • static void print(int[], int[], int, int)	⌚ (0/38) (0.00%)

Figura 3.3.2: Cobertura do método generateNextSteps(int[][], int)

Project	DUAs Coverage
▼ • static int returnSumNeighbors(int, int, int, int[][])	⌚ (0/109) (0.00%)
• (59, (64, 64), i)	✗ false
• (59, (64, 67), i)	✗ false
• (59, (67, 67), i)	✗ false
• (59, (67, 70), i)	✗ false
• (59, (82, 83), i)	✗ false
• (59, (82, 86), i)	✗ false
• (59, 83, i)	✗ false
• (59, (80, 81), i)	✗ false
• (59, (80, 82), i)	✗ false
• (59, 81, i)	✗ false
• (59, (78, 79), i)	✗ false
• (59, (78, 80), i)	✗ false
• (59, 79, i)	✗ false
• (59, (76, 77), i)	✗ false
• (59, (76, 78), i)	✗ false
• (59, 77, i)	✗ false
• (59, (73, 74), i)	✗ false
• (59, (73, 76), i)	✗ false
• (59, 74, i)	✗ false
• (59, (70, 71), i)	✗ false
• (59, (70, 73), i)	✗ false
• (59, 71, i)	✗ false
• (59, (67, 67), i)	✗ false
• (59, (67, 70), i)	✗ false

Figura 3.4.1: Cobertura do método returnSumNeighbors(int, int, int, int[][])

Project	DUAs Coverage
• (59, (67, 70), i)	✗ false
• (59, 68, i)	✗ false
• (59, (64, 64), i)	✗ false
• (59, (64, 67), i)	✗ false
• (59, 65, i)	✗ false
• (59, (60, 60), i)	✗ false
• (59, (60, 64), i)	✗ false
• (59, (60, 61), i)	✗ false
• (59, (60, 64), i)	✗ false
• (59, 61, i)	✗ false
• (59, (60, 60), j)	✗ false
• (59, (60, 64), j)	✗ false
• (59, (70, 70), j)	✗ false
• (59, (70, 73), j)	✗ false
• (59, (73, 73), j)	✗ false
• (59, (73, 76), j)	✗ false
• (59, (76, 76), j)	✗ false
• (59, (76, 78), j)	✗ false
• (59, (78, 78), j)	✗ false
• (59, (78, 80), j)	✗ false
• (59, (80, 80), j)	✗ false
• (59, (80, 82), j)	✗ false
• (59, (82, 82), j)	✗ false
• (59, (82, 86), j)	✗ false
• (59, 83, j)	✗ false

Figura 3.4.2: Cobertura do método `returnSumNeighbors(int, int, int, int[][])`

Project	DUAs Coverage
• (59, 83, j)	✗ false
• (59, 81, j)	✗ false
• (59, 79, j)	✗ false
• (59, 77, j)	✗ false
• (59, (73, 73), j)	✗ false
• (59, (73, 76), j)	✗ false
• (59, 74, j)	✗ false
• (59, (70, 70), j)	✗ false
• (59, (70, 73), j)	✗ false
• (59, 71, j)	✗ false
• (59, (67, 68), j)	✗ false
• (59, (67, 70), j)	✗ false
• (59, 68, j)	✗ false
• (59, (64, 65), j)	✗ false
• (59, (64, 67), j)	✗ false
• (59, 65, j)	✗ false
• (59, (60, 60), j)	✗ false
• (59, (60, 64), j)	✗ false
• (59, 61, j)	✗ false
• (59, (80, 80), boardDimensionMinusOne)	✗ false
• (59, (80, 82), boardDimensionMinusOne)	✗ false
• (59, (82, 82), boardDimensionMinusOne)	✗ false
• (59, (82, 86), boardDimensionMinusOne)	✗ false
• (59, (82, 83), boardDimensionMinusOne)	✗ false
• (59, (82, 86), boardDimensionMinusOne)	✗ false

Figura 3.4.3: Cobertura do método `returnSumNeighbors(int, int, int, int[][])`

Project	DUAs Coverage
• (59, (82, 86), boardDimensionMinusOne)	✗ false
• (59, (78, 79), boardDimensionMinusOne)	✗ false
• (59, (78, 80), boardDimensionMinusOne)	✗ false
• (59, (73, 73), boardDimensionMinusOne)	✗ false
• (59, (73, 76), boardDimensionMinusOne)	✗ false
• (59, (73, 74), boardDimensionMinusOne)	✗ false
• (59, (73, 76), boardDimensionMinusOne)	✗ false
• (59, (70, 70), boardDimensionMinusOne)	✗ false
• (59, (70, 73), boardDimensionMinusOne)	✗ false
• (59, (67, 67), boardDimensionMinusOne)	✗ false
• (59, (67, 70), boardDimensionMinusOne)	✗ false
• (59, (67, 68), boardDimensionMinusOne)	✗ false
• (59, (67, 70), boardDimensionMinusOne)	✗ false
• (59, (64, 64), boardDimensionMinusOne)	✗ false
• (59, (64, 67), boardDimensionMinusOne)	✗ false
• (59, (60, 60), boardDimensionMinusOne)	✗ false
• (59, (60, 64), boardDimensionMinusOne)	✗ false
• (59, (60, 61), boardDimensionMinusOne)	✗ false
• (59, (60, 64), boardDimensionMinusOne)	✗ false
• (59, 83, actualRound)	✗ false
• (59, 81, actualRound)	✗ false
• (59, 79, actualRound)	✗ false
• (59, 77, actualRound)	✗ false
• (59, 74, actualRound)	✗ false
• (59, 71, actualRound)	✗ false

Figura 3.4.4: Cobertura do método returnSumNeighbors(int, int, int, int[])

Project	DUAs Coverage
• (59, (60, 60), boardDimensionMinusOne)	✗ false
• (59, (60, 64), boardDimensionMinusOne)	✗ false
• (59, (60, 61), boardDimensionMinusOne)	✗ false
• (59, (60, 64), boardDimensionMinusOne)	✗ false
• (59, 83, actualRound)	✗ false
• (59, 81, actualRound)	✗ false
• (59, 79, actualRound)	✗ false
• (59, 77, actualRound)	✗ false
• (59, 74, actualRound)	✗ false
• (59, 71, actualRound)	✗ false
• (59, 68, actualRound)	✗ false
• (59, 65, actualRound)	✗ false
• (59, 61, actualRound)	✗ false
• (59, 86, sumNeighbors)	✗ false
• (61, 86, sumNeighbors)	✗ false
• (65, 86, sumNeighbors)	✗ false
• (68, 86, sumNeighbors)	✗ false
• (71, 86, sumNeighbors)	✗ false
• (74, 86, sumNeighbors)	✗ false
• (77, 86, sumNeighbors)	✗ false
• (79, 86, sumNeighbors)	✗ false
• (81, 86, sumNeighbors)	✗ false
• (83, 86, sumNeighbors)	✗ false
► static int returnNextStep(int, int)	(14/17) (82.35%)
► static void print(int[], int[], int, int)	(0/38) (0.00%)

Figura 3.4.5: Cobertura do método `returnSumNeighbors(int, int, int, int[])`

Project	DUAs Coverage
► MainTest	(28/40) (70.00%)
▼ Main	
► static void main(String[])	(36/244) (14.75%)
► static int[] generateRandom(int[], int)	(0/27) (0.00%)
► static int[] generateNextSteps(int[], int)	(22/24) (91.67%)
► static int returnSumNeighbors(int, int, int, int[])	(0/29) (0.00%)
► static int returnNextStep(int, int)	(0/109) (0.00%)
► (91, (92, 93), totalNeighbors)	(14/17) (82.35%)
► (91, (92, 94), totalNeighbors)	✓ true
► (91, (92, 99), totalNeighbors)	✓ true
► (91, (98, 101), totalNeighbors)	✓ true
► (91, (96, 97), totalNeighbors)	✓ true
► (91, (96, 98), totalNeighbors)	✗ false
► (91, (94, 95), totalNeighbors)	✓ true
► (91, (94, 96), totalNeighbors)	✓ true
► (91, (94, 94), element)	✓ true
► (91, (94, 96), element)	✓ true
► (91, (96, 98), element)	✓ true
► (91, (98, 98), element)	✓ true
► (91, (98, 101), element)	✗ false
► (91, 93, element)	✓ true
► (91, (101, 101), random)	✓ true
► (91, (101, 101), random)	✗ false
► static void print(int[], int[], int, int)	(0/38) (0.00%)

Figura 3.5: Cobertura do método `returnNextStep(int, int)`

## 4) EclEmma e Baduino - Parte II- B

### 4.1) EclEmma

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
Trab	84.4 %	841	156	997
src	84.4 %	841	156	997
(default package)	84.4 %	841	156	997
Main.java	78.0 %	538	152	690
MainTest.java	98.7 %	303	4	307

Figura 4.0 - Cobertura de código com eclEmma

### 4.2) Baduino

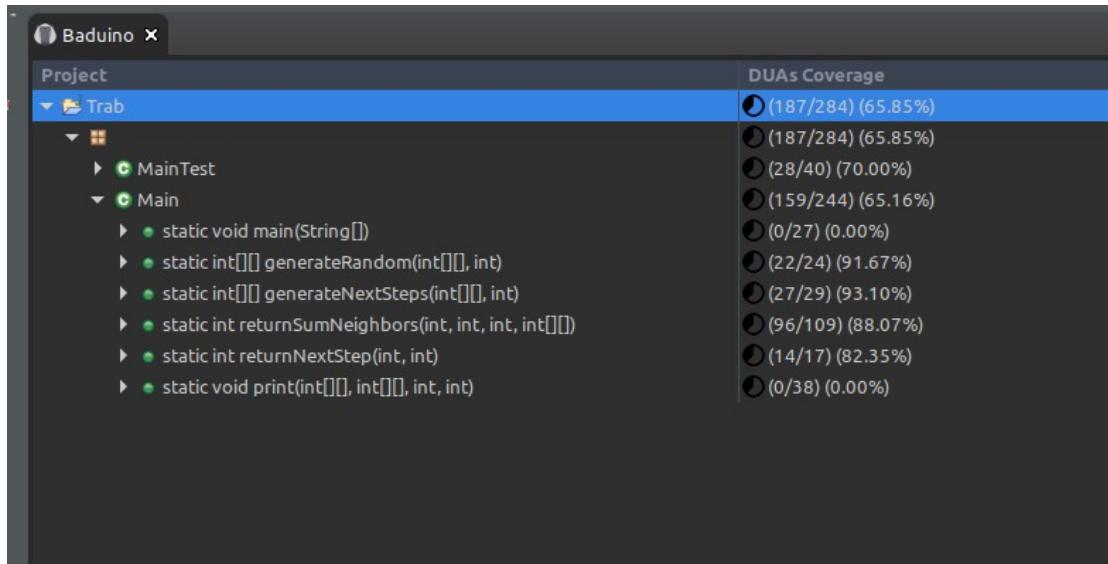


Figura 4.1: Cobertura geral dos métodos da classe Main

Project	DUAs Coverage
static int[][] generateRandom(int[], int)	(22/24) (91.67%)
• (36, 39, actualRound)	✓ true
• (36, 43, actualRound)	✓ true
• (36, (37, 43), boardDimension)	✓ true
• (36, (37, 38), boardDimension)	✓ true
• (36, (38, 37), boardDimension)	✓ true
• (36, (38, 39), boardDimension)	✓ true
• (36, (39, 39), random)	✓ true
• (36, (39, 39), random)	✗ false
• (37, (37, 43), j)	✓ true
• (37, (37, 38), j)	✓ true
• (37, 39, j)	✓ true
• (37, 37, j)	✓ true
• (38, (38, 37), i)	✗ false
• (38, (38, 39), i)	✓ true
• (38, 39, i)	✓ true
• (38, 38, i)	✓ true
• (38, (38, 37), i)	✓ true
• (38, (38, 39), i)	✓ true
• (38, 39, i)	✓ true
• (38, 38, i)	✓ true
• (37, (37, 43), j)	✓ true
• (37, (37, 38), j)	✓ true
• (37, 39, j)	✓ true
• (37, 37, j)	✓ true

Figura 4.2: Cobertura do método generateRandom(int[], int)

Project	DUAs Coverage
static int[][] generateNextSteps(int[], int)	(27/29) (93.10%)
• (47, 51, actualRound)	✓ true
• (47, 52, actualRound)	✓ true
• (47, (49, 55), boardDimension)	✓ true
• (47, (49, 50), boardDimension)	✓ true
• (47, (50, 49), boardDimension)	✓ true
• (47, (50, 51), boardDimension)	✓ true
• (47, 52, nextRound)	✓ true
• (47, 55, nextRound)	✓ true
• (48, 51, boardDimensionMinusOne)	✓ true
• (49, (49, 55), j)	✗ false
• (49, (49, 50), j)	✓ true
• (49, 51, j)	✓ true
• (49, 52, j)	✓ true
• (49, 49, j)	✓ true
• (50, (50, 49), i)	✗ false
• (50, (50, 51), i)	✓ true
• (50, 51, i)	✓ true
• (50, 52, i)	✓ true
• (50, 50, i)	✓ true
• (50, (50, 49), i)	✓ true
• (50, (50, 51), i)	✓ true
• (50, 51, i)	✓ true
• (50, 52, i)	✓ true
• (50, 50, i)	✓ true

Figura 4.3.1: Cobertura do método generateNextSteps(int[], int)

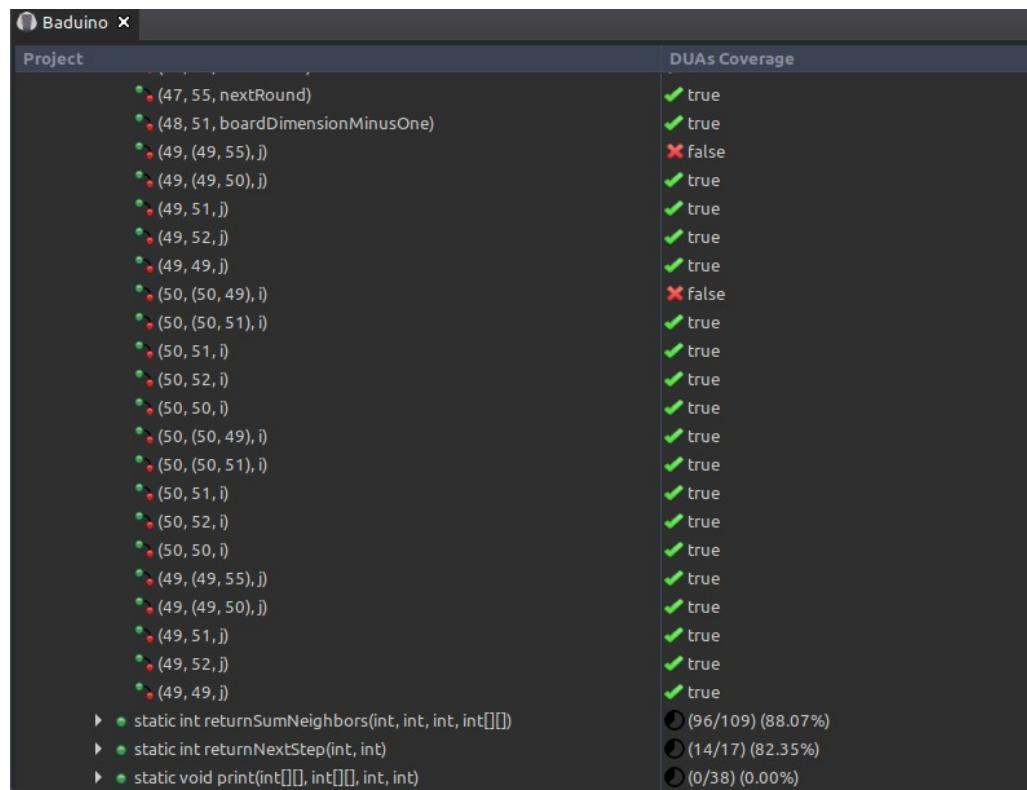


Figura 4.3.2: Cobertura do método generateNextSteps(int[], int)

Project	DUAs Coverage
static int returnSumNeighbors(int, int, int, int[][])	(96/109) (88.07%)
(59, (64, 64), i)	✓ true
(59, (64, 67), i)	✓ true
(59, (67, 67), i)	✓ true
(59, (67, 70), i)	✓ true
(59, (82, 83), i)	✓ true
(59, (82, 86), i)	✗ false
(59, 83, i)	✓ true
(59, (80, 81), i)	✓ true
(59, (80, 82), i)	✓ true
(59, 81, i)	✓ true
(59, (78, 79), i)	✓ true
(59, (78, 80), i)	✗ false
(59, 79, i)	✓ true
(59, (76, 77), i)	✓ true
(59, (76, 78), i)	✓ true
(59, 77, i)	✓ true
(59, (73, 74), i)	✓ true
(59, (73, 76), i)	✗ false
(59, 74, i)	✓ true
(59, (70, 71), i)	✓ true
(59, (70, 73), i)	✓ true
(59, 71, i)	✓ true
(59, (67, 67), i)	✓ true
(59, (67, 70), i)	✓ true

Figura 4.4.1: Cobertura do método `returnSumNeighbors(int, int, int, int[][])`

Project	DUAs Coverage
• (59, (67, 70), i)	✓ true
• (59, 68, i)	✓ true
• (59, (64, 64), i)	✓ true
• (59, (64, 67), i)	✓ true
• (59, 65, i)	✓ true
• (59, (60, 60), i)	✓ true
• (59, (60, 64), i)	✓ true
• (59, (60, 61), i)	✓ true
• (59, (60, 64), i)	✓ true
• (59, 61, i)	✓ true
• (59, (60, 60), j)	✓ true
• (59, (60, 64), j)	✓ true
• (59, (70, 70), j)	✓ true
• (59, (70, 73), j)	✓ true
• (59, (73, 73), j)	✓ true
• (59, (73, 76), j)	✓ true
• (59, (76, 76), j)	✓ true
• (59, (76, 78), j)	✓ true
• (59, (78, 78), j)	✓ true
• (59, (78, 80), j)	✓ true
• (59, (80, 80), j)	✓ true
• (59, (80, 82), j)	✗ false
• (59, (82, 82), j)	✓ true
• (59, (82, 86), j)	✗ false
• (59, 83, j)	✓ true

Figura 4.4.2: Cobertura do método `returnSumNeighbors(int, int, int, int[])`

Project	DUAs Coverage
• (59, 83, j)	✓ true
• (59, 81, j)	✓ true
• (59, 79, j)	✓ true
• (59, 77, j)	✓ true
• (59, (73, 73), j)	✓ true
• (59, (73, 76), j)	✓ true
• (59, 74, j)	✓ true
• (59, (70, 70), j)	✓ true
• (59, (70, 73), j)	✓ true
• (59, 71, j)	✓ true
• (59, (67, 68), j)	✓ true
• (59, (67, 70), j)	✗ false
• (59, 68, j)	✓ true
• (59, (64, 65), j)	✓ true
• (59, (64, 67), j)	✓ true
• (59, 65, j)	✓ true
• (59, (60, 60), j)	✓ true
• (59, (60, 64), j)	✓ true
• (59, 61, j)	✓ true
• (59, (80, 80), boardDimensionMinusOne)	✓ true
• (59, (80, 82), boardDimensionMinusOne)	✗ false
• (59, (82, 82), boardDimensionMinusOne)	✓ true
• (59, (82, 86), boardDimensionMinusOne)	✗ false
• (59, (82, 83), boardDimensionMinusOne)	✓ true
• (59, (82, 86), boardDimensionMinusOne)	✗ false

Figura 4.4.3: Cobertura do método returnSumNeighbors(int, int, int, int[])

Project	DUAs Coverage
• (59, (82, 86), boardDimensionMinusOne)	✗ false
• (59, (78, 79), boardDimensionMinusOne)	✓ true
• (59, (78, 80), boardDimensionMinusOne)	✗ false
• (59, (73, 73), boardDimensionMinusOne)	✓ true
• (59, (73, 76), boardDimensionMinusOne)	✓ true
• (59, (73, 74), boardDimensionMinusOne)	✓ true
• (59, (73, 76), boardDimensionMinusOne)	✗ false
• (59, (70, 70), boardDimensionMinusOne)	✓ true
• (59, (70, 73), boardDimensionMinusOne)	✓ true
• (59, (67, 67), boardDimensionMinusOne)	✓ true
• (59, (67, 70), boardDimensionMinusOne)	✓ true
• (59, (67, 68), boardDimensionMinusOne)	✓ true
• (59, (67, 70), boardDimensionMinusOne)	✗ false
• (59, (64, 64), boardDimensionMinusOne)	✓ true
• (59, (64, 67), boardDimensionMinusOne)	✓ true
• (59, (60, 60), boardDimensionMinusOne)	✓ true
• (59, (60, 64), boardDimensionMinusOne)	✓ true
• (59, (60, 61), boardDimensionMinusOne)	✓ true
• (59, (60, 64), boardDimensionMinusOne)	✓ true
• (59, 83, actualRound)	✓ true
• (59, 81, actualRound)	✓ true
• (59, 79, actualRound)	✓ true
• (59, 77, actualRound)	✓ true
• (59, 74, actualRound)	✓ true
• (59, 71, actualRound)	✓ true

Figura 4.4.4: Cobertura do método returnSumNeighbors(int, int, int, int[])

Project	DUAs Coverage
• (59, (60, 60), boardDimensionMinusOne)	✓ true
• (59, (60, 64), boardDimensionMinusOne)	✓ true
• (59, (60, 61), boardDimensionMinusOne)	✓ true
• (59, (60, 64), boardDimensionMinusOne)	✓ true
• (59, 83, actualRound)	✓ true
• (59, 81, actualRound)	✓ true
• (59, 79, actualRound)	✓ true
• (59, 77, actualRound)	✓ true
• (59, 74, actualRound)	✓ true
• (59, 71, actualRound)	✓ true
• (59, 68, actualRound)	✓ true
• (59, 65, actualRound)	✓ true
• (59, 61, actualRound)	✓ true
• (59, 86, sumNeighbors)	✗ false
• (61, 86, sumNeighbors)	✓ true
• (65, 86, sumNeighbors)	✓ true
• (68, 86, sumNeighbors)	✓ true
• (71, 86, sumNeighbors)	✓ true
• (74, 86, sumNeighbors)	✓ true
• (77, 86, sumNeighbors)	✓ true
• (79, 86, sumNeighbors)	✓ true
• (81, 86, sumNeighbors)	✓ true
• (83, 86, sumNeighbors)	✓ true
▶ • static int returnNextStep(int, int)	(14/17) (82.35%)
▶ • static void print(int[], int[], int, int)	(0/38) (0.00%)

Figura 4.4.5: Cobertura do método returnSumNeighbors(int, int, int[], int)

Project	DUAs Coverage
▼ • static int returnNextStep(int, int)	(14/17) (82.35%)
• (91, (92, 93), totalNeighbors)	✓ true
• (91, (92, 94), totalNeighbors)	✓ true
• (91, (98, 99), totalNeighbors)	✓ true
• (91, (98, 101), totalNeighbors)	✓ true
• (91, (96, 97), totalNeighbors)	✓ true
• (91, (96, 98), totalNeighbors)	✗ false
• (91, (94, 95), totalNeighbors)	✓ true
• (91, (94, 96), totalNeighbors)	✓ true
• (91, (94, 94), element)	✓ true
• (91, (94, 96), element)	✓ true
• (91, (96, 96), element)	✓ true
• (91, (96, 98), element)	✓ true
• (91, (98, 98), element)	✓ true
• (91, (98, 101), element)	✗ false
• (91, 93, element)	✓ true
• (91, (101, 101), random)	✓ true
• (91, (101, 101), random)	✗ false
▶ • static void print(int[], int[], int, int)	(0/38) (0.00%)

Figura 4.5: Cobertura do método returnNextStep(int, int[])

#### **4.3) Considerações Finais sobre a Parte II A e B**

Podemos observar que no primeiro conjunto de testes (*TestSet-Func*), gerados pelas técnicas de Particionamento em Classes de Equivalência e Análise de Valor Limite, a cobertura de código beirou os 10 e 15% com as ferramentas Eclemma e Baduino, respectivamente. Os testes não revelaram defeitos no código, mas não poderíamos atribuir o fato em questão ao código em questão e afirmar, naquele momento, que o código não possuia erros, já que a cobertura era baixa. Portanto, expandimos o conjunto de testes gerando o conjunto *TestSet-Estr*.

Após a criação de novos testes, para a segunda execução, obtemos os resultados de 78 e ~65% para a cobertura de código utilizando Eclemma e Baduino, respectivamente. Podemos perceber um aumento expressivo na porcentagem quando comparamos as execuções, o que nos leva a crer que a utilização das ferramentas para medição de cobertura nos ajudou a nortear nosso processo de teste do software em questão, buscando casos de teste para uma maior cobertura e consequentemente, alcançar maior qualidade no processo de teste. Por fim foi concluído que o software funcionava como esperado, com uma maior certeza do que anteriormente, graças ao aumento na porcentagem de cobertura.

### **5) Análise sobre o Relatório e as Técnicas Empregadas.**

Como demonstrado no relatório, a utilização da ferramenta JUnit para implementação dos casos de testes definidos e das ferramentas Eclemma e Baduino para análise de cobertura foram fundamentais para testar com eficiência o código, aplicando os critérios de teste estrutural baseados em fluxo de dados e controle. Como demonstrado no Log gerado pela ferramenta Baduino, foram realizados no total 17 testes considerando os casos de testes gerados. Todos os testes realizados passaram e teve uma cobertura de aproximadamente 80%.

```
[DEBUG] 31-01-2022 21:50:12 TestsRunner:28 - Loading classes
[DEBUG] 31-01-2022 21:50:12 TestsRunner:42 - Finished loading classes
[DEBUG] 31-01-2022 21:50:12 TestsRunner:59 - Finished loading test classes
[DEBUG] 31-01-2022 21:50:12 TestsRunner:60 - Testing with JUnit
[INFO ] 31-01-2022 21:50:12 TestsRunner:64 - Executed 17 tests in 0.007
seconds. 0 tests failed.
```

```
[ INFO ] 31-01-2022 21:50:12 TestsRunner:67 - All tests passed.
```

Com relação a implementação dos testes antes das funções, a implementação do código foi afetada significativamente pelo desenvolvimento dos testes funcionais, pois o desenvolvimento foi realizado com conhecimento prévio das possíveis falhas, sendo assim mais produtivo e assertivo o desenvolvimento do código