



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

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, generateNextStepsm 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.

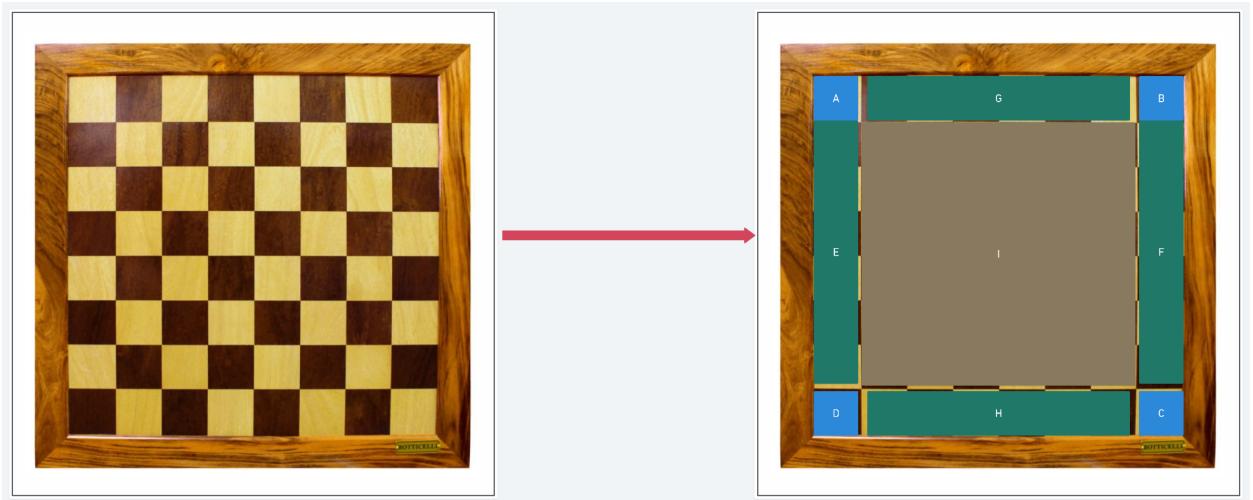


Figura 2.5: Tabuleiro vazio e um tabuleiro com os cenários marcados

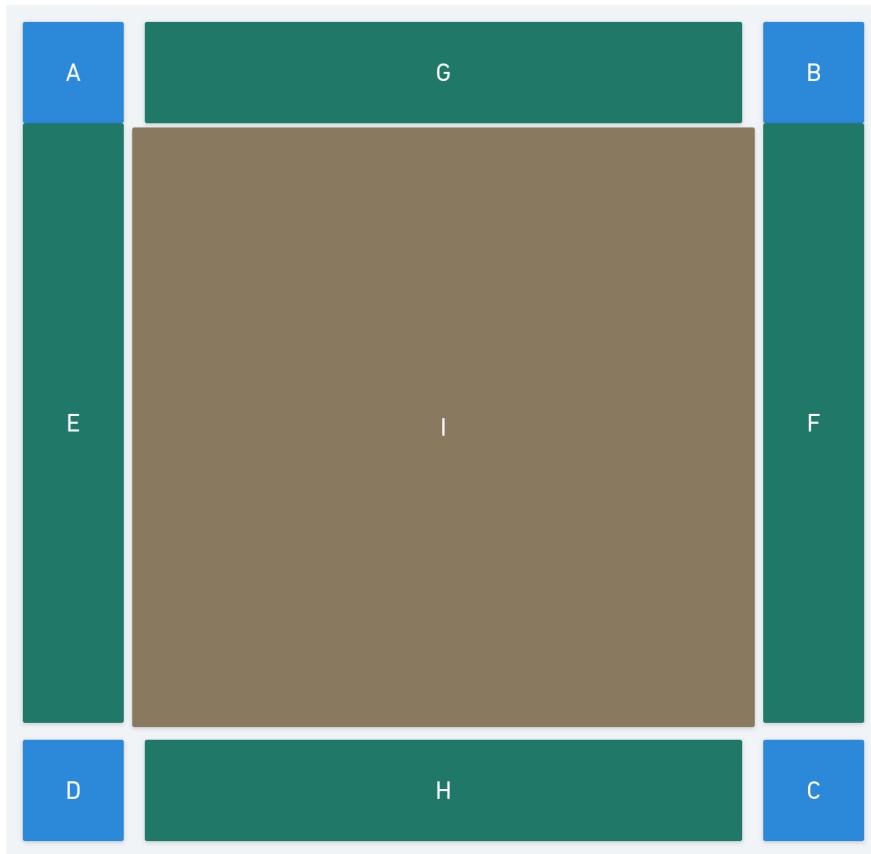


Figura 2.6: Destaque para os cenários com o intuito de melhorar a visualização

- 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

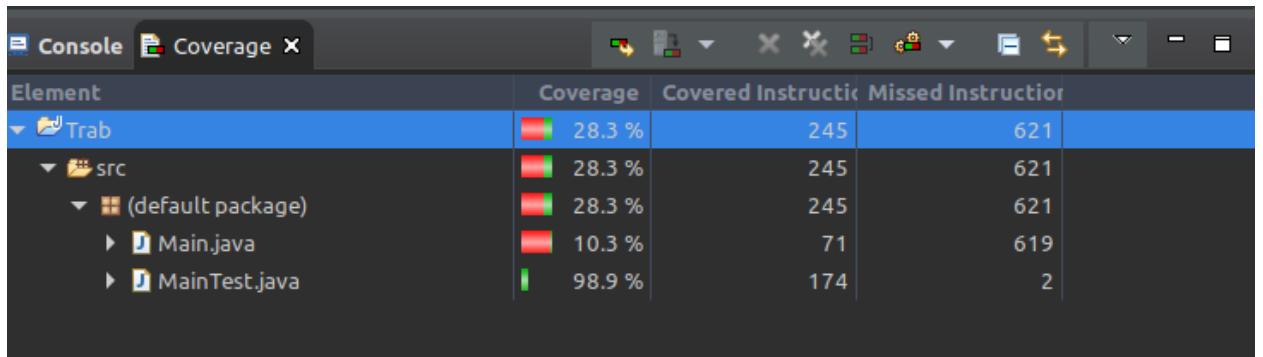


Figura 3.0 - Cobertura de código com eclemma

3.2) Baduino

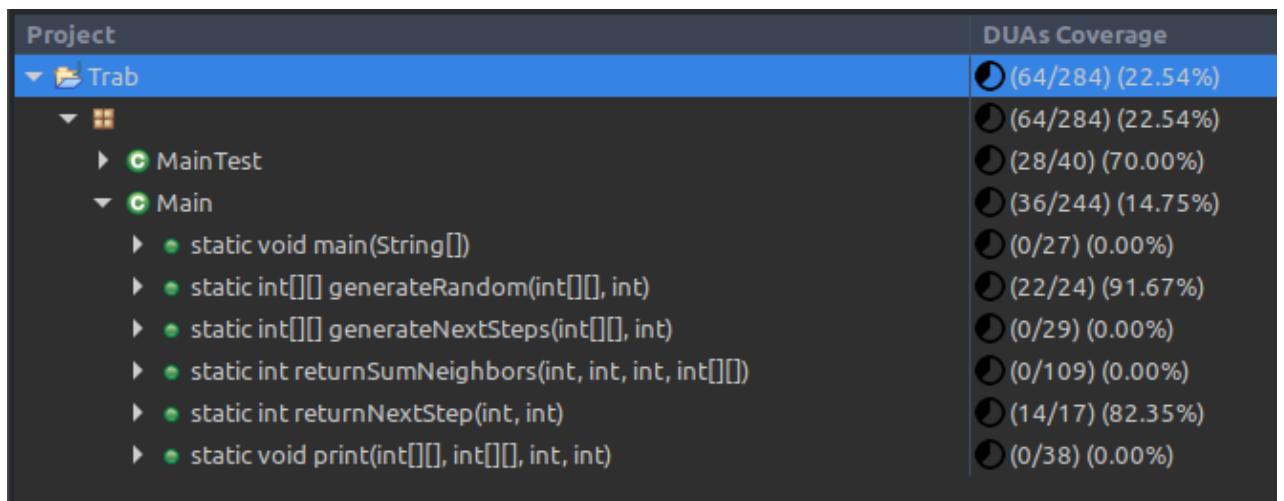


Figura 4.1: Cobertura geral dos métodos da classe Main

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)	✓ true
(37, (37, 43), j)	✗ false
(37, (37, 38), j)	✓ true
(37, (39, j))	✓ true
(37, (37, j))	✓ true
(38, (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, (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, 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

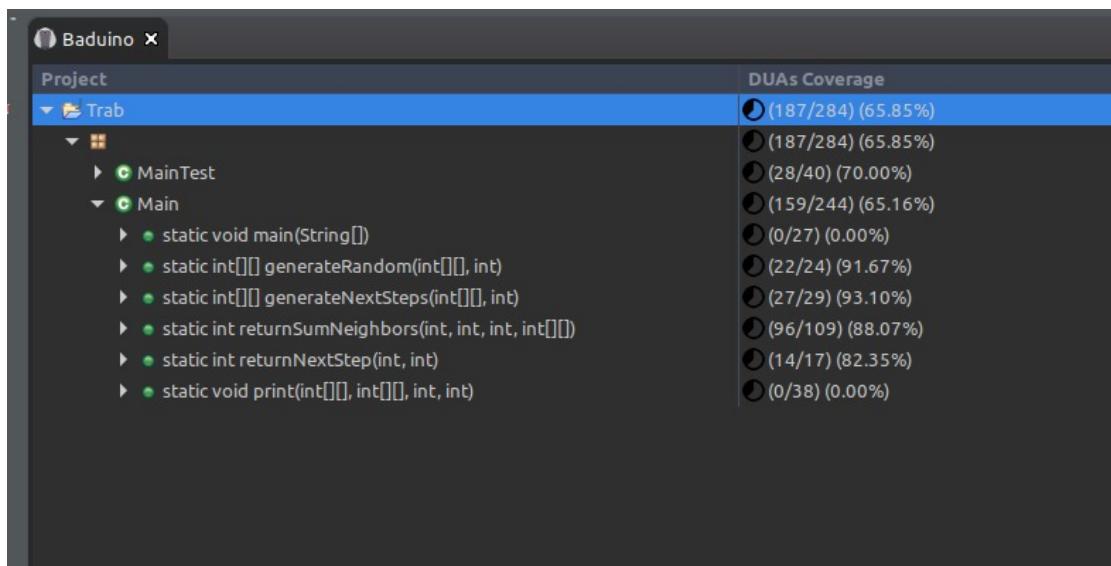


Figura 4.1: Cobertura geral dos métodos da classe Main

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)

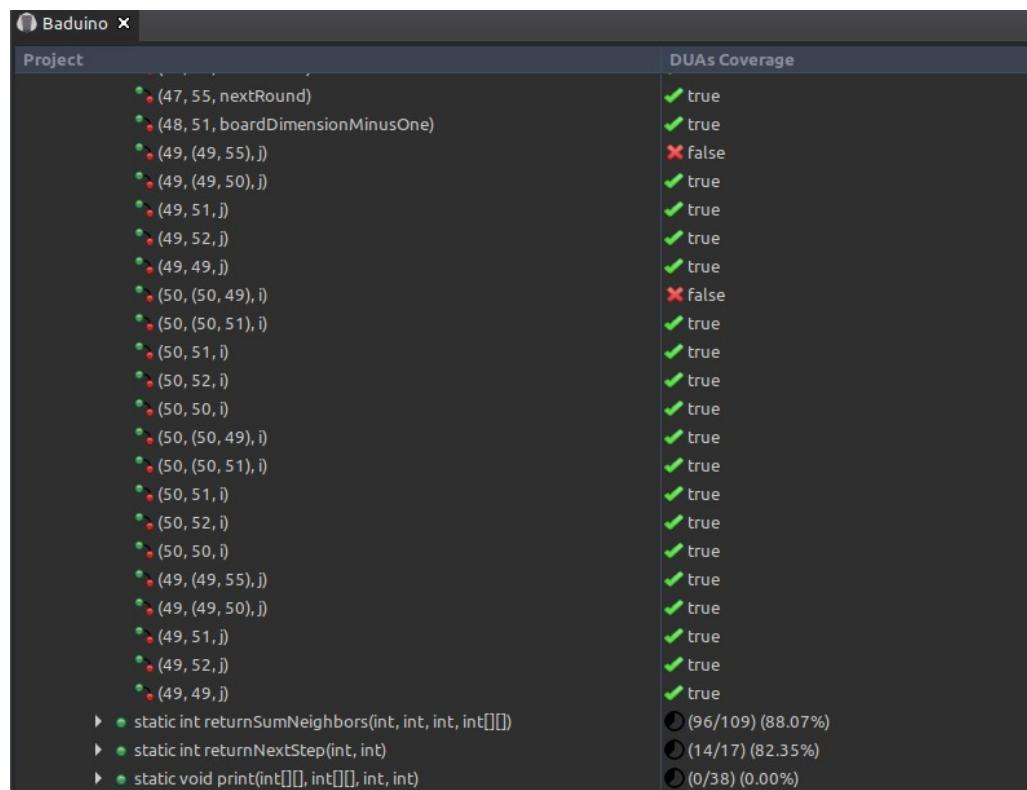


Figura 4.3.2: Cobertura do método generateNextSteps(int[][], int)

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[][])

▼ • 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 ferramentas JUnit para implementação dos casos de testes definidos e das ferramentas Eclemma e Baduíno 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 Baduíno, foram realizados no total 17 testes considerando os casos de testes gerados. Todos os testes realizados passaram e tiveram 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

6) Teste Baseado em Defeito: Teste de Mutação

a) Avaliação da Qualidade do Conjunto de Casos de Teste Funcional e Estrutural

Através dos casos de teste gerados nas Partes II-A e II-B, foi executada a ferramenta de teste de mutação PITest, aplicando todos os operadores de mutação (Full Mutation). Os resultados encontrados, foram:

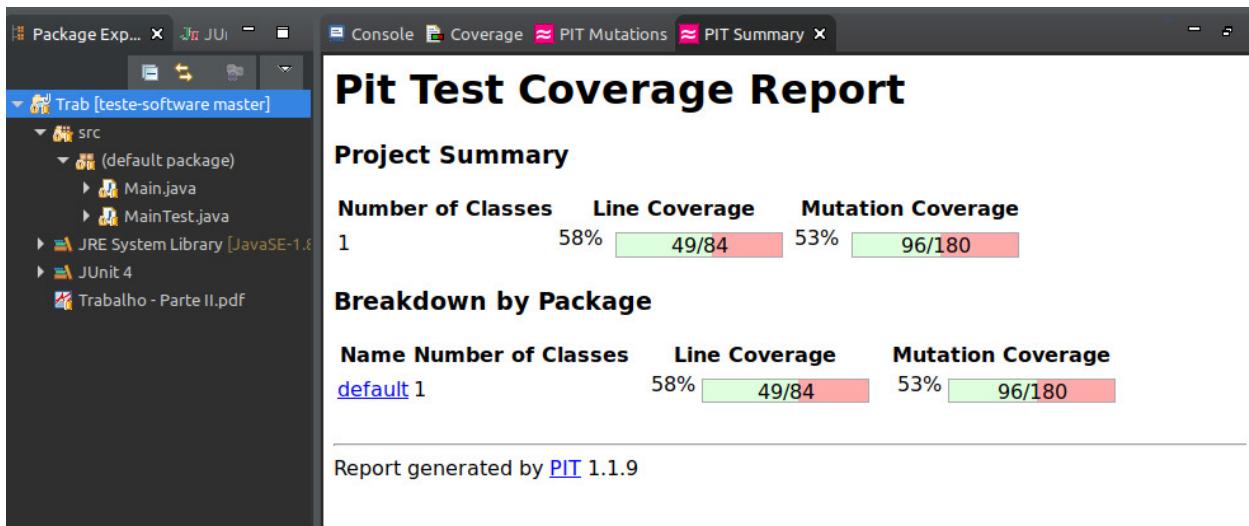


Figura 6.1 - Coverage Report

The screenshot shows the 'PIT Mutations' tab of a software interface. The title bar includes tabs for 'Console', 'Coverage', 'PIT Mutations', and 'PIT Summary'. The main area displays a tree structure under the heading 'SURVIVED (55)'. The 'Trab' package is expanded, showing the '(default package)' node which contains the 'Main (55)' class. Below 'Main (55)', a list of 55 mutations is displayed, each preceded by a small red bomb icon. The mutations are categorized into two main types: negated conditionals (labeled 37 through 50) and integer arithmetic operations (labeled 61 through 65). The list is as follows:

- 37: negated conditional
- 38: negated conditional
- 39: negated conditional
- 49: negated conditional
- 50: negated conditional
- 61: Replaced integer addition with subtraction
- 61: Replaced integer subtraction with addition
- 61: Replaced integer subtraction with addition
- 62: Replaced integer addition with subtraction
- 62: Replaced integer addition with subtraction
- 62: Replaced integer subtraction with addition
- 62: Replaced integer subtraction with addition
- 63: Replaced integer subtraction with addition
- 63: Replaced integer subtraction with addition
- 63: Replaced integer subtraction with addition
- 65: Replaced integer addition with subtraction
- 65: Replaced integer addition with subtraction

Figura 6.2.1 - Mutantes que sobreviveram

This screenshot continues the list of surviving mutants from Figure 6.2.1. The interface is identical, showing the 'PIT Mutations' tab with the 'SURVIVED (55)' list. The list continues from where it left off, starting with mutation 65 and ending at 81. The mutations are all of the 'Replaced integer addition with subtraction' type.

- 65: Replaced integer addition with subtraction
- 65: Replaced integer addition with subtraction
- 65: Replaced integer addition with subtraction
- 66: Replaced integer subtraction with addition
- 66: Replaced integer subtraction with addition
- 68: Replaced integer addition with subtraction
- 68: Replaced integer subtraction with addition
- 68: Replaced integer subtraction with addition
- 69: Replaced integer addition with subtraction
- 69: Replaced integer addition with subtraction
- 71: Replaced integer addition with subtraction
- 71: Replaced integer addition with subtraction
- 71: Replaced integer subtraction with addition
- 71: Replaced integer subtraction with addition
- 72: Replaced integer addition with subtraction
- 72: Replaced integer addition with subtraction
- 74: Replaced integer addition with subtraction
- 74: Replaced integer subtraction with addition
- 74: Replaced integer subtraction with addition
- 75: Replaced integer addition with subtraction
- 75: Replaced integer addition with subtraction
- 77: Replaced integer addition with subtraction
- 79: Replaced integer addition with subtraction
- 81: Replaced integer addition with subtraction

Figura 6.2.2 - Mutantes que sobreviveram cont.

The screenshot shows the PIT Mutations interface with the following details:

- Console**: Shows the command-line interface.
- Coverage**: Shows code coverage information.
- PIT Mutations**: The active tab, showing a list of mutations. The first mutation (id 81) is highlighted in blue.
 - 81: Replaced integer addition with subtraction
 - 83: Replaced integer addition with subtraction
 - 94: changed conditional boundary
 - 94: negated conditional
 - 96: changed conditional boundary
 - 98: negated conditional
 - 98: negated conditional
 - 101: negated conditional
 - 101: replaced return of integer sized value with ($x == 0 ? 1 : 0$)
- KILLED (96)**: A section showing mutations that did not survive.
 - Trab (96)
 - (default package) (96)
 - Main (96)
 - 37: Changed increment from 1 to -1
 - 37: changed conditional boundary
 - 38: Changed increment from 1 to -1
 - 38: changed conditional boundary
 - 43: mutated return of Object value for Main::generateRandom to (if ($x != null$) null else throw new RuntimeException)
 - 48: Replaced integer subtraction with addition
 - 49: Changed increment from 1 to -1
 - 49: changed conditional boundary
 - 50: Changed increment from 1 to -1
 - 50: changed conditional boundary
 - 55: mutated return of Object value for Main::generateNextSteps to (if ($x != null$) null else throw new RuntimeException)
 - 60: negated conditional
 - 60: negated conditional

Figura 6.2.3 - Mutantes que sobreviveram cont. e Mutantes mortos

The screenshot shows the PIT Mutations interface with the following details:

- Console**: Shows the command-line interface.
- Coverage**: Shows code coverage information.
- PIT Mutations**: The active tab, showing a long list of killed mutations.
 - 60: negated conditional
 - 60: negated conditional
 - 60: negated conditional
 - 61: Replaced integer addition with subtraction
 - 64: changed conditional boundary
 - 64: changed conditional boundary
 - 64: negated conditional
 - 64: negated conditional
 - 64: negated conditional
 - 65: Replaced integer addition with subtraction
 - 66: Replaced integer addition with subtraction
 - 67: changed conditional boundary
 - 67: changed conditional boundary
 - 67: negated conditional
 - 67: negated conditional
 - 67: negated conditional
 - 68: Replaced integer addition with subtraction
 - 68: Replaced integer addition with subtraction
 - 68: Replaced integer subtraction with addition
 - 68: Replaced integer subtraction with addition

Figura 6.2.4 - Mutantes mortos cont.

```
68: Replaced integer subtraction with addition
69: Replaced integer subtraction with addition
70: changed conditional boundary
70: changed conditional boundary
70: negated conditional
70: negated conditional
70: negated conditional
71: Replaced integer addition with subtraction
72: Replaced integer addition with subtraction
73: changed conditional boundary
73: changed conditional boundary
73: negated conditional
73: negated conditional
73: negated conditional
74: Replaced integer addition with subtraction
74: Replaced integer addition with subtraction
74: Replaced integer subtraction with addition
74: Replaced integer subtraction with addition
75: Replaced integer subtraction with addition
76: negated conditional
76: negated conditional
77: Replaced integer addition with subtraction
77: Replaced integer addition with subtraction
```

Figura 6.2.5 - Mutantes mortos cont.

```
77: Replaced integer addition with subtraction
78: negated conditional
78: negated conditional
79: Replaced integer addition with subtraction
79: Replaced integer addition with subtraction
79: Replaced integer addition with subtraction
79: Replaced integer subtraction with addition
79: Replaced integer subtraction with addition
80: negated conditional
80: negated conditional
81: Replaced integer addition with subtraction
81: Replaced integer addition with subtraction
81: Replaced integer addition with subtraction
81: Replaced integer subtraction with addition
81: Replaced integer subtraction with addition
82: negated conditional
82: negated conditional
83: Replaced integer addition with subtraction
83: Replaced integer subtraction with addition
86: replaced return of integer sized value with (x == 0 ? 1 : 0)
```

Figura 6.2.6 - Mutantes mortos cont.

The screenshot shows the PIT Mutations interface with the following details:

- Console tab is active.
- Coverage tab is present.
- PIT Mutations tab is active.
- PIT Summary tab is present.
- Current mutation count: 85: Replaced integer subtraction with addition
- Selected mutation: 86: replaced return of integer sized value with (x == 0 ? 1 : 0)
- Other mutations listed:
 - 92: negated conditional
 - 93: replaced return of integer sized value with (x == 0 ? 1 : 0)
 - 94: negated conditional
 - 95: replaced return of integer sized value with (x == 0 ? 1 : 0)
 - 96: negated conditional
 - 96: negated conditional
 - 97: replaced return of integer sized value with (x == 0 ? 1 : 0)
 - 99: replaced return of integer sized value with (x == 0 ? 1 : 0)
- NO_COVERAGE section (29 mutations):
 - Trab (29)
 - (default package) (29)
 - Main (29)
 - 7: removed call to java/io/PrintStream::println
 - 8: removed call to java/io/PrintStream::println
 - 9: removed call to java/io/PrintStream::println
 - 10: removed call to java/io/PrintStream::println
 - 11: removed call to java/io/PrintStream::println
 - 18: negated conditional
 - 19: removed call to Main::print
 - 20: Changed increment from 1 to -1
 - 22: removed call to java/io/PrintStream::println
 - 23: removed call to java/io/PrintStream::println
 - 24: removed call to java/io/PrintStream::println
 - 25: removed call to java/io/PrintStream::println
 - 27: negated conditional

Figura 6.2.7 - Mutantes mortos cont. e No Coverage

The screenshot shows the PIT Mutations interface with the following details:

- Console tab is active.
- Coverage tab is present.
- PIT Mutations tab is active.
- PIT Summary tab is present.
- Current mutation count: 10: removed call to java/io/PrintStream::println
- Selected mutation: 27: negated conditional
- Other mutations listed:
 - 11: removed call to java/io/PrintStream::println
 - 18: negated conditional
 - 19: removed call to Main::print
 - 20: Changed increment from 1 to -1
 - 22: removed call to java/io/PrintStream::println
 - 23: removed call to java/io/PrintStream::println
 - 24: removed call to java/io/PrintStream::println
 - 25: removed call to java/io/PrintStream::println
 - 27: negated conditional
 - 27: negated conditional
 - 106: removed call to java/io/PrintStream::println
 - 107: removed call to java/io/PrintStream::println
 - 108: Changed increment from 1 to -1
 - 108: changed conditional boundary
 - 108: negated conditional
 - 109: Changed increment from 1 to -1
 - 109: changed conditional boundary
 - 109: negated conditional
 - 110: removed call to java/io/PrintStream::print
 - 112: removed call to java/io/PrintStream::print
 - 113: Changed increment from 1 to -1
 - 113: changed conditional boundary
 - 113: negated conditional
 - 114: removed call to java/io/PrintStream::print
 - 116: removed call to java/io/PrintStream::println

Figura 6.2.8 -No Coverage cont.

b) Aplicação dos testes de mutação

- i) Para realizar a remoção dos mutantes vivos, foi necessário a edição de alguns casos de teste, principalmente, os que estavam relacionados ao posicionamento do elemento e o total de vizinhos (returnSumNeighbors). Com isso, foi necessário realizar a adição de um novo vizinho e ficar metrificando o resultado atual dos vizinhos para que todos os mutantes fossem eliminados. Nas imagens a seguir é possível visualizar a implementação dos novos casos de teste, estes responsáveis pela remoção de quase 30 mutações.

```
@Test
public void elementoEhDoisZero(){
    assertEquals(0, main.returnSumNeighbors(2, 0, 2,result));
    result[0][0] = 1;
    assertEquals(0, main.returnSumNeighbors(2, 0, 2,result));
    result[0][1] = 1;
    assertEquals(1, main.returnSumNeighbors(2, 0, 2,result));
    result[0][2] = 1;
    assertEquals(1, main.returnSumNeighbors(2, 0, 2,result));
    result[1][0] = 1;
    assertEquals(1, main.returnSumNeighbors(2, 0, 2,result));
    result[1][1] = 1;
    assertEquals(2, main.returnSumNeighbors(2, 0, 2,result));
    result[1][2] = 1;
    assertEquals(3, main.returnSumNeighbors(2, 0, 2,result));
    result[2][0] = 1;
    assertEquals(3, main.returnSumNeighbors(2, 0, 2,result));
    result[2][1] = 1;
    assertEquals(3, main.returnSumNeighbors(2, 0, 2,result));
    result[2][2] = 1;
    assertEquals(3, main.returnSumNeighbors(2, 0, 2,result));
}
```

Figura 6.3.1 - Novos Casos de Testes.

```

@Test
public void elementoEhUmDois(){
    assertEquals(0, main.returnSumNeighbors(1, 2, 2,result));
    result[0][0] = 0;
    assertEquals(0, main.returnSumNeighbors(1, 2, 2,result));
    result[0][1] = 0;
    assertEquals(0, main.returnSumNeighbors(1, 2, 2,result));
    result[0][2] = 0;
    assertEquals(0, main.returnSumNeighbors(1, 2, 2,result));
    result[1][0] = 1;
    assertEquals(1, main.returnSumNeighbors(1, 2, 2,result));
    result[1][1] = 1;
    assertEquals(2, main.returnSumNeighbors(1, 2, 2,result));
    result[1][2] = 1;
    assertEquals(3, main.returnSumNeighbors(1, 2, 2,result));
    result[2][0] = 1;
    assertEquals(4, main.returnSumNeighbors(1, 2, 2,result));
    result[2][1] = 0;
    assertEquals(4, main.returnSumNeighbors(1, 2, 2,result));
    result[2][2] = 1;
    assertEquals(5, main.returnSumNeighbors(1, 2, 2,result));
}

```

Figura 6.3.2 - Novos Casos de Testes.

```

@Test
public void elementoEhUmZero(){
    assertEquals(0, main.returnSumNeighbors(1, 0, 2, result));
    result[0][0] = 1;
    assertEquals(1, main.returnSumNeighbors(1, 0, 2, result));
    result[0][1] = 1;
    assertEquals(1, main.returnSumNeighbors(1, 0, 2, result));
    result[0][2] = 1;
    assertEquals(2, main.returnSumNeighbors(1, 0, 2, result));
    result[1][0] = 1;
    assertEquals(3, main.returnSumNeighbors(1, 0, 2, result));
    result[1][1] = 1;
    assertEquals(4, main.returnSumNeighbors(1, 0, 2, result));
    result[1][2] = 1;
    assertEquals(5, main.returnSumNeighbors(1, 0, 2, result));
    result[2][0] = 1;
    assertEquals(5, main.returnSumNeighbors(1, 0, 2, result));
    result[2][1] = 1;
    assertEquals(5, main.returnSumNeighbors(1, 0, 2, result));
    result[2][2] = 1;
    assertEquals(5, main.returnSumNeighbors(1, 0, 2, result));
}

```

Figura 6.3.3 - Novos Casos de Testes.

```

@Test
public void elementoZeroDois(){
    assertEquals(0, main.returnSumNeighbors(0, 2, 2,result ));
    result[0][0] = 1;
    assertEquals(0, main.returnSumNeighbors(0, 2, 2,result ));
    result[0][1] = 1;
    assertEquals(0, main.returnSumNeighbors(0,2, 2,result ));
    result[0][2] = 1;
    assertEquals(0, main.returnSumNeighbors(0, 2, 2,result ));
    result[1][0] = 1;
    assertEquals(1, main.returnSumNeighbors(0, 2, 2,result ));
    result[1][1] = 1;
    assertEquals(2, main.returnSumNeighbors(0, 2, 2,result ));
    result[1][2] = 1;
    assertEquals(2, main.returnSumNeighbors(0, 2, 2,result ));
    result[2][0] = 1;
    assertEquals(2, main.returnSumNeighbors(0, 2, 2,result ));
    result[2][1] = 1;
    assertEquals(3, main.returnSumNeighbors(0, 2, 2,result ));
    result[2][2] = 1;
    assertEquals(3, main.returnSumNeighbors(0, 2, 2,result ));
}

```

Figura 6.3.4 - Novos Casos de Testes.

```

@Test
public void elementoEhCentralizado(){
    assertEquals(0, main.returnSumNeighbors(1, 1, 2,result ));
    result[0][0] = 1;
    assertEquals(1, main.returnSumNeighbors(1, 1, 2,result ));
    result[0][1] = 1;
    assertEquals(2, main.returnSumNeighbors(1, 1, 2,result ));
    result[0][2] = 1;
    assertEquals(3, main.returnSumNeighbors(1, 1, 2,result ));
    result[1][0] = 1;
    assertEquals(4, main.returnSumNeighbors(1, 1, 2,result ));
    result[1][1] = 1;
    assertEquals(4, main.returnSumNeighbors(1, 1, 2,result ));
    result[1][2] = 1;
    assertEquals(5, main.returnSumNeighbors(1, 1, 2,result ));
    result[2][0] = 1;
    assertEquals(6, main.returnSumNeighbors(1, 1, 2,result ));
    result[2][1] = 1;
    assertEquals(7, main.returnSumNeighbors(1, 1, 2,result ));
    result[2][2] = 1;
    assertEquals(8, main.returnSumNeighbors(1, 1, 2,result ));
}

```

Figura 6.3.5 - Novos Casos de Testes.

```

@Test
public void elementoEhZeroUm(){
    assertEquals(0, main.returnSumNeighbors(0, 1,2, result));
    result[0][0] = 1;
    assertEquals(1, main.returnSumNeighbors(0, 1,2, result));
    result[0][1] = 1;
    assertEquals(2, main.returnSumNeighbors(0, 1,2, result));
    result[0][2] = 0;
    assertEquals(2, main.returnSumNeighbors(0, 1,2, result));
    result[1][0] = 0;
    assertEquals(2, main.returnSumNeighbors(0, 1,2, result));
    result[1][1] = 1;
    assertEquals(3, main.returnSumNeighbors(0, 1,2, result));
    result[1][2] = 0;
    assertEquals(3, main.returnSumNeighbors(0, 1,2, result));
    result[2][0] = 1;
    assertEquals(4, main.returnSumNeighbors(0, 1,2, result));
    result[2][1] = 1;
    assertEquals(5, main.returnSumNeighbors(0, 1,2, result));
    result[2][2] = 0;
    assertEquals(5, main.returnSumNeighbors(0, 1,2, result));
}

```

Figura 6.3.6 - Novos Casos de Testes.

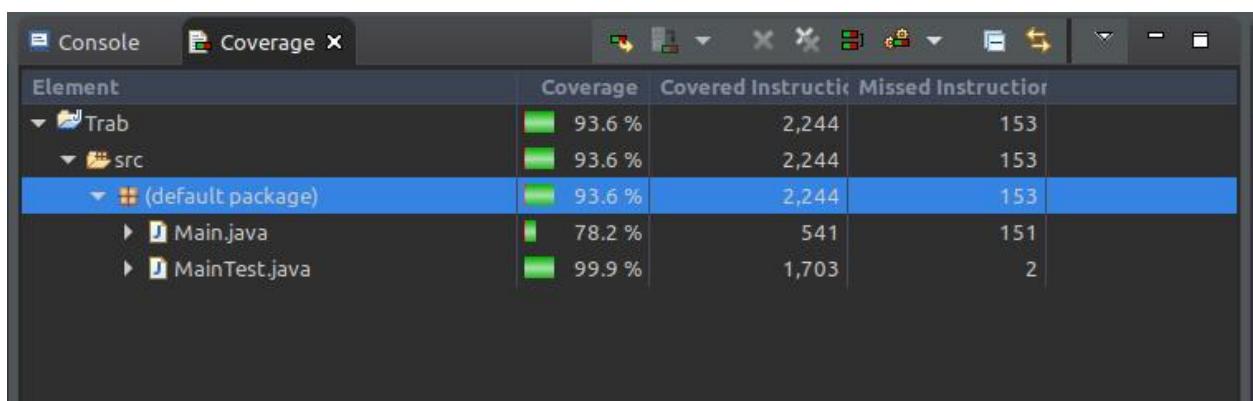
```

@Test
public void elementoEhZeroZero(){
    assertEquals(0, main.returnSumNeighbors(0, 0, 2, result));
    result[0][0] = 1;
    assertEquals(0, main.returnSumNeighbors(0, 0, 2, result));
    result[0][1] = 1;
    assertEquals(1, main.returnSumNeighbors(0, 0, 2, result));
    result[0][2] = 1;
    assertEquals(1, main.returnSumNeighbors(0, 0, 2, result));
    result[1][0] = 1;
    assertEquals(2, main.returnSumNeighbors(0, 0, 2, result));
    result[1][1] = 1;
    assertEquals(3, main.returnSumNeighbors(0, 0, 2, result));
    result[1][2] = 1;
    assertEquals(3, main.returnSumNeighbors(0, 0, 2, result));
    result[2][0] = 1;
    assertEquals(3, main.returnSumNeighbors(0, 0, 2, result));
    result[2][1] = 1;
    assertEquals(3, main.returnSumNeighbors(0, 0, 2, result));
    result[2][2] = 1;
    assertEquals(3, main.returnSumNeighbors(0, 0, 2, result));
}

```

Figura 6.3.7 - Novos Casos de Testes.

- ii) Ao final do programa, foi implementado 22 testes, com uma cobertura de 78,2%. Ao analisar apenas o valor da cobertura do código da a entender que o programa não está totalmente testado, porém visualizando o código toda a parte que não recebe cobertura é a parte relacionada a impressão das informações ao usuário.
Foi adicionado uma imagem para representar a cobertura final do código, pois, na parte 2 foi adicionado uma cobertura antes de realizarmos os ajustes no projeto, ou seja, a cobertura da primeira bateria de testes implementados



- iii) Com relação aos mutantes equivalentes, é possível encontrar o mutante da linha 123, pois, como é uma função randomizada que gera um valor booleano verdadeiro ou falso, mesmo que o valor seja modificado, o programa vai funcionar da mesma maneira.

```

123; replaced return of integer sized value with (x==0?1:0)
1 KILLED (142)
  Trab (142)
    (default package) (142)
      Main (142)
        * 36: Changed increment from 1 to -1
        * 36: changed conditional boundary
        * 37: Changed increment from 1 to -1
        * 37: changed conditional boundary
        * 41: mutated return of Object value for Main::generateRandom to (if(x!=null) null else throw new RuntimeException())
104
105  public static void print(int[][] actual, int[][] previous, int acumulate, int boardDimension) {
106      System.out.println("\n" + acumulate + " Iteração");
107      System.out.println("\t Anterior \t\t || \t\t Atual");
108      for (int j = 0; j < boardDimension; j++) {
109          for (int i = 0; i < boardDimension; i++) {
110              System.out.print("[ " + previous[j][i] + " ]");
111          }
112          System.out.print("\t || \t");
113          for (int i = 0; i < boardDimension; i++) {
114              System.out.print("[ " + actual[j][i] + " ]");
115          }
116          System.out.println("");
117      }
118  }
119
120  public static int returnRandomInt() {
121      Random random = new Random();
122      boolean result = random.nextBoolean();
123      return result == true ? 1 : 0;
124  }

```

Figura 6.4 - Mutante Equivalente.

- iv) Com a implementação de novos teste, para solucionar o problema dos mutantes vivos, foi descoberto uma falha no programa que não foi encontrada com os testes anteriores. Sendo assim, algumas posições do tabuleiro não estava sendo cobertas pelo algoritmo, ou seja, quando o elemento contém oito vizinhos e todos estivessem vivos o programa estava retornando um valor igual à 5.

Na imagem abaixo é possível visualizar o ajuste realizado no programa:

```

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 - 1] + actualRound[j][i - 1] + actualRound[j + 1][i - 1]
            + actualRound[j - 1][i] + actualRound[j + 1][i] + actualRound[j - 1][i + 1] + actualRound[j][i + 1]
            + actualRound[j + 1][i + 1];
    }
}

```

Figura 6.5 - Função ajustada para resolver o problema encontrado

- v) Na imagem a seguir é possível visualizar o relatório com os testes de mutações, após os ajustes realizados:

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage	Mutation Coverage
1	60% <div style="width: 60%; background-color: #90EE90; border: 1px solid black; display: inline-block;">52/87</div>	79% <div style="width: 79%; background-color: #FFB6C1; border: 1px solid black; display: inline-block;">142/180</div>

Breakdown by Package

Name	Number of Classes	Line Coverage	Mutation Coverage
default	1	60% <div style="width: 60%; background-color: #90EE90; border: 1px solid black; display: inline-block;">52/87</div>	79% <div style="width: 79%; background-color: #FFB6C1; border: 1px solid black; display: inline-block;">142/180</div>

Figura 6.6 - Relatório final de Cobertura do PITest

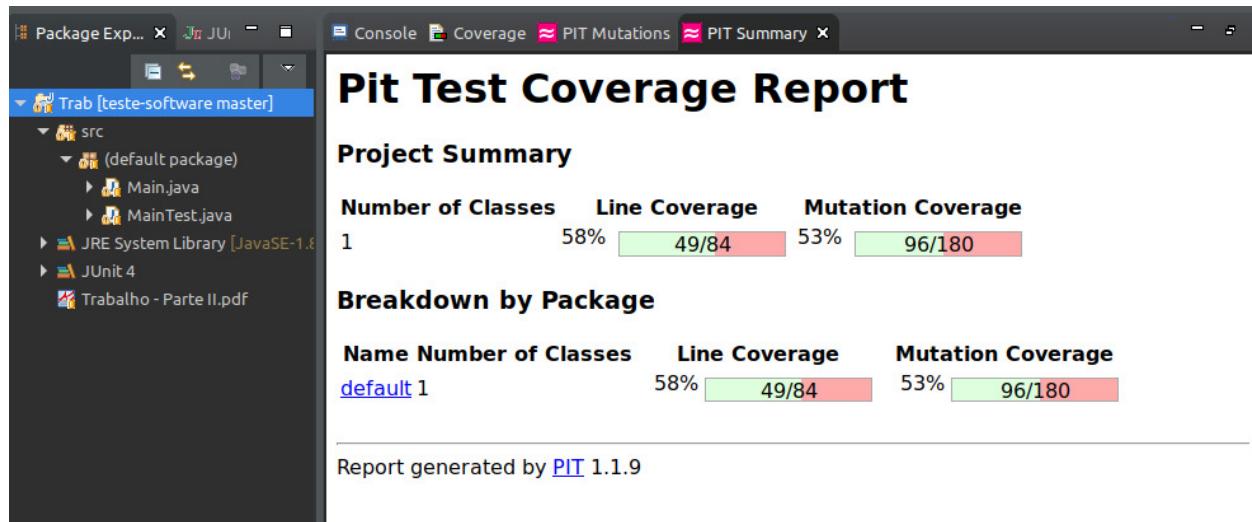


Figura 6.7 - Relatório inicial de Cobertura do PITest

Ao analisar a cobertura de código é possível afirmar que as principais linhas do programa estão sendo cobertas, porém os 40% das linhas restantes têm relação a impressão das informações para que o usuário tenha um feedback do andamento do programa.

c) Continuação do relatório de execução dos testes

Dado a primeira aplicação da ferramenta PITest, considerando a métrica de Escore de Mutação e Número de Mutantes sobreviventes, obtemos a seguinte avaliação da qualidade do conjunto de testes:

Critérios	Inicial	Pós-ajustes
Escore de Mutação	53%	79%
Mutantes Vivos	55	9

Ao rodar o teste de mutação inicial não foi encontrado nenhum erro no código, mas ao iniciar o processo de resolver os problemas dos mutantes sobreviventes e aumentar o escore de mutação, foi encontrado um erro na implementação do valor centralizado, ou seja, qualquer valor que não esteja na extremidade. A correção foi realizada e aplicada na etapa III-B..

Após a execução da etapa III-B, podemos concluir que houve um aumento de 26% na cobertura, resultado das correções e adições de novos casos de teste, realizadas após a primeira execução do PITest.

The screenshot shows a tree view of mutation analysis results. The root node is 'SURVIVED (9)'. Under it, there is a folder 'Trab (9)' which contains a folder '(default package) (9)' and a file 'Main (9)'. The 'Main (9)' file has 123 mutations listed, with the last one highlighted in orange. The mutations are categorized as follows:

- 36: negated conditional
- 37: negated conditional
- 50: negated conditional
- 51: negated conditional
- 94: changed conditional boundary
- 96: negated conditional
- 101: replaced return of integer sized value with $(x == 0 ? 1 : 0)$
- 123: negated conditional
- 123: replaced return of integer sized value with $(x == 0 ? 1 : 0)$

Below 'SURVIVED (9)', there is a folder 'KILLED (142)' which contains a folder 'Trab (142)' and a folder '(default package) (142)'. The '(default package) (142)' folder contains a file 'Main (142)' with 62 mutations listed:

- * 36: Changed increment from 1 to -1
- * 36: changed conditional boundary
- * 37: Changed increment from 1 to -1
- * 37: changed conditional boundary
- * 44: mutated return of Object value for Main::generateRandom to ($\text{if } (x != \text{null}) \text{ null else throw new RuntimeException()}$)
- * 49: Replaced integer subtraction with addition
- * 50: Changed increment from 1 to -1
- * 50: changed conditional boundary
- * 51: Changed increment from 1 to -1
- * 51: changed conditional boundary
- * 56: mutated return of Object value for Main::generateNextSteps to ($\text{if } (x != \text{null}) \text{ null else throw new RuntimeException()}$)
- * 61: negated conditional
- * 61: negated conditional
- * 61: negated conditional
- * 62: Replaced integer addition with subtraction

Figura 6.7 - Mutantes Gerados

```
* 62: Replaced integer subtraction with addition
* 62: Replaced integer subtraction with addition
* 62: Replaced integer subtraction with addition
* 63: Replaced integer addition with subtraction
* 63: Replaced integer addition with subtraction
* 63: Replaced integer subtraction with addition
* 63: Replaced integer subtraction with addition
* 64: Replaced integer addition with subtraction
* 64: Replaced integer addition with subtraction
* 65: changed conditional boundary
* 65: changed conditional boundary
* 65: negated conditional
* 65: negated conditional
* 65: negated conditional
* 66: Replaced integer addition with subtraction
* 67: Replaced integer addition with subtraction
* 67: Replaced integer subtraction with addition
* 67: Replaced integer subtraction with addition
* 68: changed conditional boundary
* 68: changed conditional boundary
* 68: negated conditional
* 68: negated conditional
* 68: negated conditional
* 69: Replaced integer addition with subtraction
* 69: Replaced integer subtraction with addition
```

Figura 6.7.1 - Mutantes Gerados

- 70: Replaced integer subtraction with addition
- 71: changed conditional boundary
- 71: changed conditional boundary
- 71: negated conditional
- 71: negated conditional
- 71: negated conditional
- 72: Replaced integer addition with subtraction
- 72: Replaced integer subtraction with addition
- 72: Replaced integer subtraction with addition
- 73: Replaced integer addition with subtraction
- 73: Replaced integer addition with subtraction
- 73: Replaced integer addition with subtraction
- 74: changed conditional boundary
- 74: changed conditional boundary
- 74: negated conditional
- 74: negated conditional
- 74: negated conditional
- 75: Replaced integer addition with subtraction
- 75: Replaced integer subtraction with addition
- 76: Replaced integer addition with subtraction
- 76: Replaced integer addition with subtraction
- 76: Replaced integer subtraction with addition
- 77: negated conditional
- 77: negated conditional
- 78: Replaced integer addition with subtraction

Figura 6.7.2 - Mutantes Gerados

- 79: negated conditional
- 79: negated conditional
- 80: Replaced integer addition with subtraction
- 80: Replaced integer subtraction with addition
- 80: Replaced integer subtraction with addition
- 81: negated conditional
- 81: negated conditional
- 82: Replaced integer addition with subtraction
- 82: Replaced integer subtraction with addition
- 82: Replaced integer subtraction with addition
- 83: negated conditional
- 83: negated conditional
- 84: Replaced integer addition with subtraction
- 84: Replaced integer addition with subtraction
- 84: Replaced integer subtraction with addition
- 87: replaced return of integer sized value with ($x == 0 ? 1 : 0$)
- 92: changed conditional boundary
- 92: negated conditional
- 92: negated conditional
- 93: replaced return of integer sized value with ($x == 0 ? 1 : 0$)
- 94: negated conditional
- 94: negated conditional
- 95: replaced return of integer sized value with ($x == 0 ? 1 : 0$)
- 96: negated conditional
- 97: replaced return of integer sized value with ($x == 0 ? 1 : 0$)
- 98: negated conditional
- 99: replaced return of integer sized value with ($x == 0 ? 1 : 0$)

Figura 6.7.3 - Mutantes Gerados

```
▼ NO_COVERAGE (29)
  ▼ Trab (29)
    ▼ (default package) (29)
      ▼ Main (29)
        ⚡ 7: removed call to java/io/PrintStream::println
        ⚡ 8: removed call to java/io/PrintStream::println
        ⚡ 9: removed call to java/io/PrintStream::println
        ⚡ 10: removed call to java/io/PrintStream::println
        ⚡ 11: removed call to java/io/PrintStream::println
        ⚡ 18: negated conditional
        ⚡ 19: removed call to Main::print
        ⚡ 20: Changed increment from 1 to -1
        ⚡ 22: removed call to java/io/PrintStream::println
        ⚡ 23: removed call to java/io/PrintStream::println
        ⚡ 24: removed call to java/io/PrintStream::println
        ⚡ 25: removed call to java/io/PrintStream::println
        ⚡ 27: negated conditional
        ⚡ 27: negated conditional
        ⚡ 106: removed call to java/io/PrintStream::println
        ⚡ 107: removed call to java/io/PrintStream::println
        ⚡ 108: Changed increment from 1 to -1
        ⚡ 108: changed conditional boundary
        ⚡ 108: negated conditional
        ⚡ 109: Changed increment from 1 to -1
        ⚡ 109: changed conditional boundary
        ⚡ 109: negated conditional
        ⚡ 110: removed call to java/io/PrintStream::print
        ⚡ 112: removed call to java/io/PrintStream::print
        ⚡ 113: Changed increment from 1 to -1
        ⚡ 113: changed conditional boundary
        ⚡ 113: negated conditional
        ⚡ 114: removed call to java/io/PrintStream::print
        ⚡ 116: removed call to java/io/PrintStream::println
```

Figura 6.7.4 - Mutantes Gerados

Para responder à pergunta apresentada ao final dos requisitos do trabalho, é necessário analisar a definição do teste de mutação e, para eliminar os mutantes é necessário também que o valor errado seja encontrado em alguns dos testes como um caminho para encontrar a solução. Sendo assim, é possível afirmar que caso um mutante fosse morto e o resultado em questão seria um caminho correto, os testes implementados pelo desenvolvedor foi feito de uma forma incorreta. Com isso, para conseguir solucionar o problema, será necessário refatorar o código em questão, focando em ajustar o problema na implementação do teste.