GCC255 - REO 4: Relatório de Ferramentas 1

Mateus Carvalho Gonçalves - 201810245
Otávio de Lima Soares - 201811022
Pedro Antônio de Souza - 201810557

Parte 1:

Para o particionamento em classes de equivalência foram consideradas as possíveis entradas das computações, tanto as entradas que são geradas pelo sistema quanto às interações com usuário. Dessa forma, as variáveis que controlam o fluxo do programa são: a condição atual da célula (viva ou morta), o número de vizinhos vivos, e a variável de controle para continuar ou parar as iterações do tabuleiro.

Condição de entrada	Classes válidas	Classes inválidas
Condição da célula (C)	0 <= C <= 1	C < 0 E C > 1
Número de vizinhos vivos (N)[1]	0 <= N < 2 (C recebe 0)	N < 0 E N > 8
	N == 2 (C mantém)	
	N == 3 (C recebe 1)	
	3 < N <= 8 (C recebe 0)	
Continuar? (F)[2]	‘s’ E ‘n’	Entradas diferentes de ‘s’ e ‘n’

[1] Valores apresentados são para células que não estão nas bordas do tabuleiro. Valores válidos para bordas variam de 0 <= N <= 3 para células nas quinas do tabuleiro, e 0 <= N <= 5 para células nas bordas que não são quinas.
[2] Valores são case insensitive.

Apesar das especificações não apontarem valores inválidos, eles podem ser inferidos pelos valores das classes válidas.

Enquanto as classes de equivalência mostram as cada condição de entrada, a análise do valor limite busca testar valores limites quando há intervalos nas classes válidas. Nesse caso, C e N possuem intervalos e serão as condições com valores limites.

Para testar a condição F, serão considerados dois valores válidos (um de cada tipo - ‘s’ / ‘n’) e um valor inválido qualquer.

Valores limite de C: {0, 1}; e os valores vizinhos são: {-1, 2}
Valores limite de N: {0, 2, 3, 8}; e os valores vizinhos são: {-1, 1, 4, 7, 9}.

T = {
(<C=-1, N=3>, erro - C inválido),
(<C=2, N=0>, erro - C inválido),

(<C=0, N=-1>, erro - N inválido),
(<C=1, N=0>, 0),
(<C=0, N=0>, 0),
(<C=1, N=1>, 0),
(<C=1, N=2>, 1),
(<C=0, N=2>, 0),
(<C=1, N=3>, 1),
(<C=0, N=3>, 1),
(<C=1, N=4>, 0),
(<C=1, N=7>, 0),
(<C=1, N=8>, 0),
(<C=0, N=8>, 0),
(<C=1, N=9>, erro - N inválido),
(<F=S>, continua execução),
(<F=N>, encerra execução),
(<F=Nop>, pede nova entrada de F ao usuário)
}

Uma vez que a computação do programa depende dos valores de C e N (ignorando o trecho para F que já foi explicado acima) e pela percepção de é importante testar casos ue tanto mantém o valor de C quanto mudam-no para um dado valor de N, os casos de teste foram desenvolvidos da seguinte maneira:

- Para valores inválidos apenas um caso de teste é feito;
- Para cada valor limite de N foram feitos dois casos, um para cada valor de C;
- Para os vizinhos válidos dos valores limite de N apenas o caso que troca o valor de C é testado.

Casos de teste para os vizinhos válidos de N só foram feitos por causa da especificação do método AVL. Todas as classes de equivalência destes elementos já estão sendo verificadas por outros valores, logo, eles poderiam ser descartados.

Por fim, é importante frisar que os testes foram desenvolvidos a partir das computações de uma célula da matriz, não da matriz inteira

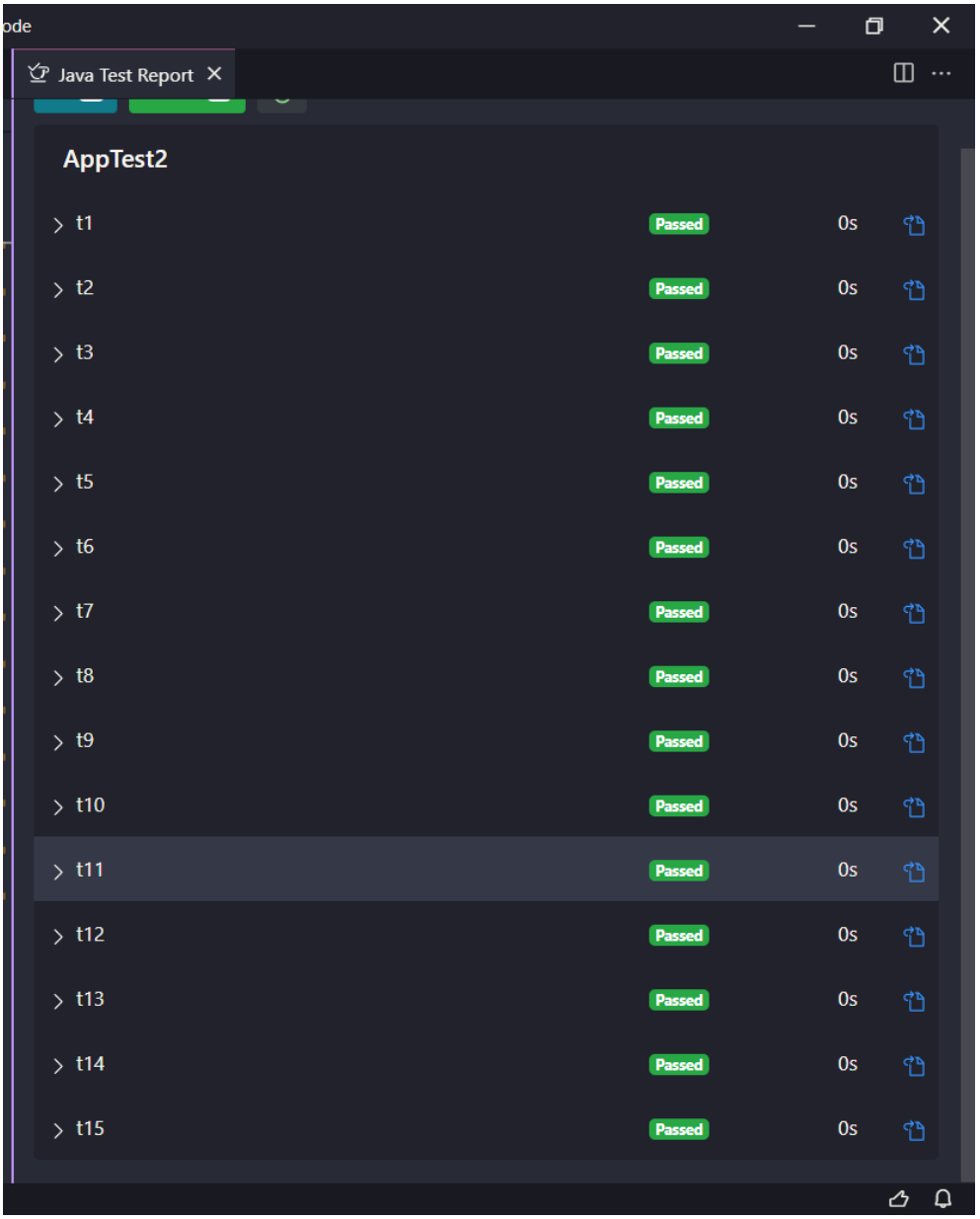
Parte 2:

Os arquivos do programa, dos testes e um arquivo HTML com o relatório do EclEmma estão presentes no arquivo compactado enviado.

Parte 3 e 4:

Os testes funcionais com JUnit tiveram taxa de sucesso de 100%, mostrado na figura abaixo.

Já os testes estruturais com EclEmma tiveram uma cobertura de 32% para os vértices e 31% para as arestas. Isso se deve ao fato de que os testes funcionais foram implementados para serem executados apenas com o método getNextState() - que faz chamadas para alguns outros métodos também. Por isso, somente o método getNextState() e alguns construtores, além do arquivo de teste do JUnit, receberam 100% de cobertura pelo EclEmma.



Para melhorar a taxa de cobertura atribuída pelo EclEmma, deve-se criar casos de teste que avaliem os outros trechos e funções do código desenvolvido, de modo a cobrir mais arestas e vértices do GFC do programa.

Com base nos dados descritos acima, o grupo avaliou que o JUnit é uma ferramenta bastante eficiente para a construção de testes funcionais e unitários, já que possui uma série de ferramentas auxiliares que garantem abrangência e qualidade na construção dos testes. Por sua vez, o EclEmma se mostra uma ferramenta com potencial poderoso, porém depende de numerosos e robustos casos de teste para ter uma boa cobertura. Para o grupo, isso significa que, em alguns casos, o uso dessa ferramenta pode ter um baixo custo-benefício em relação ao esforço e tempo gasto para escrita e execução dos testes, como em um software pequeno como o produto deste relatório. Para programas críticos como de Internet Banking, por exemplo, essa abordagem pode ser uma boa alternativa.