Tabela 1 – Classes de Equivalência

Primeiro, criaremos a tabela de classes de equivalência, dividindo as condições de entrada do jogo em classes válidas e inválidas.

Condição de Entrada	**Classes de Equivalência Válidas**	**Classes de Equivalência Inválidas**
Estado da célula	Célula está dentro dos limites (0 <= linha < 6 e 0 <= coluna < 6)	Linha ou coluna fora do intervalo (linha < 0 ou linha >= 6 ou coluna < 0 ou coluna >= 6)
Célula viva/morta	A célula pode estar viva (valor 1) ou morta (valor 0)	Valor diferente de 0 ou 1
Número de vizinhos vivos	De 0 a 8 vizinhos vivos, dentro dos limites da grade	O número de vizinhos é maior que 8 ou menor que 0
Definir Estado da Célula	Linha e coluna válidos, célula sendo definida como "viva" ou "morta"	Linha ou coluna inválida, ou valor do estado da célula inválido
Regras do Jogo (Próxima Geração)	A célula segue as regras: sobrevive com 2 ou 3 vizinhos, morre com menos de 2 ou mais de 3 vizinhos	Regra aplicada de maneira incorreta

---

Tabela 2 – Casos de Teste

Agora, usaremos as classes de equivalência e definiremos os casos de teste.

ID	**Condições de Entrada**	**Saída Esperada**	**Classes de Equivalência Exercitadas**	**Saída Obtida**
1	Definir célula (0, 0) como viva	A célula (0, 0) está viva	Estado de célula (válida), definição de célula viva (válida)	A célula (0, 0) está viva
2	Definir célula (0, 0) fora dos limites (linha = -1)	Erro, célula fora dos limites	Definir estado da célula com linha inválida (inválida)	Erro, célula fora dos limites
3	Contar vizinhos vivos de uma célula (1, 1) com 3 vizinhos vivos	Resultado 3 vizinhos vivos	Número de vizinhos (válido)	3 vizinhos vivos
4	Contar vizinhos vivos de uma célula (5, 5) com 0 vizinhos vivos	Resultado 0 vizinhos vivos	Número de vizinhos (válido)	0 vizinhos vivos

ID	Condições de Entrada	Saída Esperada	Classes de Equivalência Exercitadas	Saída Obtida
5	Proxima geração: célula (2, 2) morre por solidão	Célula (2, 2) estará morta	Regras do jogo (válida), célula com menos de 2 vizinhos vivos (inválida)	A célula (2, 2) morreu por solidão
6	Proxima geração: célula (2, 2) revive com 3 vizinhos	Célula (2, 2) estará viva	Regras do jogo (válida), célula com 3 vizinhos vivos (válida)	A célula (2, 2) revive com 3 vizinhos
7	Teste limite: célula (0, 0) com 3 vizinhos	Célula (0, 0) permanece viva	Definir célula e aplicar regras com vizinhos (válido), célula viva (válida)	A célula (0, 0) permanece viva
8	Teste limite: célula (5, 5) com 2 vizinhos vivos	Célula (5, 5) morre por solidão (menos de 2 vizinhos)	Definir célula e aplicar regras com vizinhos (válido), célula morta (válida)	A célula (5, 5) morreu por solidão

```
import jogoDaVida.JogoDaVida;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

/*
public class JogoDaVidaTest {

   // Testa se é possível definir o estado de uma célula corretamente para "viva"
   @Test
   public void testDefinirEstadoCelula() {
      JogoDaVida jogo = new JogoDaVida();
      // Define a célula (0, 0) como viva
      jogo.definirEstadoCelula(0, 0, true);
      // Verifica se a célula (0, 0) está viva após ser definida
      assertTrue(jogo.obterEstadoCelula(0, 0));
   }

   // Testa o método de contagem de vizinhos vivos ao redor de uma célula
   @Test
   public void testContarVizinhosVivos() {
      JogoDaVida jogo = new JogoDaVida();
      // Define células ao redor da célula (2, 2)
      jogo.definirEstadoCelula(1, 1, true);
      jogo.definirEstadoCelula(1, 2, true);
```

```
    jogo.definirEstadoCelula(2, 1, true);
    // Conta os vizinhos vivos da célula (2, 2)
    assertEquals(3, jogo.contarVizinhosVivos(2, 2));  // A célula (2, 2) deve ter 3 vizinhos vivos
}

// Testa se uma célula morre por solidão quando tem menos de 2 vizinhos vivos
@Test
public void testCélulaMorrePorSolidão() {
    JogoDaVida jogo = new JogoDaVida();
    // Define a célula (2, 2) como viva
    jogo.definirEstadoCelula(2, 2, true);
    jogo.definirEstadoCelula(1, 1, true); // Apenas uma célula viva (no caso, (1,1))
    // Avança para a próxima geração
    jogo.proximaGeracao();
    // A célula (2, 2) deve morrer por solidão
    assertFalse(jogo.obterEstadoCelula(2, 2));  // Espera-se que (2, 2) esteja morta
}

// Testa se uma célula revive quando tem exatamente 3 vizinhos vivos
@Test
public void testCélulaReviveCom3Vizinhos() {
    JogoDaVida jogo = new JogoDaVida();
    // Define três células vivas ao redor da célula (2, 2)
    jogo.definirEstadoCelula(1, 1, true);
    jogo.definirEstadoCelula(1, 2, true);
    jogo.definirEstadoCelula(2, 1, true);
    // Avança para a próxima geração
    jogo.proximaGeracao();
    // A célula (2, 2) deve reviver com 3 vizinhos vivos
    assertTrue(jogo.obterEstadoCelula(2, 2));  // Espera-se que (2, 2) esteja viva
}

// Testa as condições de limite do tabuleiro e a transição correta de células
@Test
public void testLimitesTabuleiro() {
    JogoDaVida jogo = new JogoDaVida();

    // Configura o tabuleiro com algumas células vivas ao redor de (2, 2)
    jogo.definirEstadoCelula(1, 1, true);
    jogo.definirEstadoCelula(1, 2, true);
    jogo.definirEstadoCelula(2, 1, true);

    // Verifica o estado inicial da célula (2, 2) antes de rodar a próxima geração
    assertFalse(jogo.obterEstadoCelula(2, 2));  // Antes da geração, (2, 2) deve estar morta

    // Avança para a próxima geração
    jogo.proximaGeracao();
```

```
        // Após a próxima geração, a célula (2, 2) deve se tornar viva
        assertTrue(jogo.obterEstadoCelula(2, 2));  // Após a geração, espera-se que (2, 2) esteja
viva
    }

    // Testa a evolução geral do tabuleiro após aplicar uma geração
    @Test
    public void testEvolucaoTabuleiro() {
        JogoDaVida jogo = new JogoDaVida();
        // Define as células (1, 1), (1, 2), e (2, 1) como vivas
        jogo.definirEstadoCelula(1, 1, true);
        jogo.definirEstadoCelula(1, 2, true);
        jogo.definirEstadoCelula(2, 1, true);
        // Avança para a próxima geração
        jogo.proximaGeracao();
        // Verifica a evolução da célula (2, 0) (deve ficar morta) e (2, 2) (deve ficar viva)
        assertFalse(jogo.obterEstadoCelula(2, 0));  // (2, 0) deve ficar morta
        assertTrue(jogo.obterEstadoCelula(2, 2));  // (2, 2) deve ficar viva
    }
}
*/

import jogoDaVida.JogoDaVida;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class JogoDaVidaTest {

    // Caso de Teste 1: Definir Estado de Célula Válido
    @Test
    public void testDefinirEstadoCelula() {
        JogoDaVida jogo = new JogoDaVida();
        jogo.definirEstadoCelula(0, 0, true);
        assertTrue(jogo.obterEstadoCelula(0, 0));
    }

    @Test
    public void testDefinirEstadoCelulaInvalidaLinha() {
        JogoDaVida jogo = new JogoDaVida();
        // Garantindo que as células fora dos limites não afetam o tabuleiro
        jogo.definirEstadoCelula(-1, 0, true); // Linha fora do limite
        jogo.definirEstadoCelula(0, 0, true);  // Definindo uma célula válida
        assertFalse(jogo.obterEstadoCelula(-1, 0)); // Nenhuma alteração deve ocorrer
        assertTrue(jogo.obterEstadoCelula(0, 0));  // A célula válida ainda deve estar "true"
    }

    // Caso de Teste 3: Contar Vizinhos Vivos - Válido com 3 vizinhos vivos
    @Test
```

```
    public void testContarVizinhosVivos() {
        JogoDaVida jogo = new JogoDaVida();
        jogo.definirEstadoCelula(1, 1, true);
        jogo.definirEstadoCelula(1, 2, true);
        jogo.definirEstadoCelula(2, 1, true);
        assertEquals(3, jogo.contarVizinhosVivos(2, 2));
    }

    // Caso de Teste 4: Contar Vizinhos Vivos - Válido com 0 vizinhos vivos
    @Test
    public void testContarVizinhosVivosZero() {
        JogoDaVida jogo = new JogoDaVida();
        jogo.definirEstadoCelula(5, 5, true);
        assertEquals(0, jogo.contarVizinhosVivos(5, 5)); // Nenhum vizinho
    }

    // Caso de Teste 5: Célula Morre por Solidão - Menos de 2 Vizinhos
    @Test
    public void testCelulaMorrePorSolidao() {
        JogoDaVida jogo = new JogoDaVida();
        jogo.definirEstadoCelula(2, 2, true); // Célula isolada
        jogo.definirEstadoCelula(1, 1, true); // Apenas 1 vizinho
        jogo.proximaGeracao();
        assertFalse(jogo.obterEstadoCelula(2, 2)); // A célula deve morrer
    }

    // Caso de Teste 6: Célula Revive com 3 Vizinhos Vivos
    @Test
    public void testCelulaReviveCom3Vizinhos() {
        JogoDaVida jogo = new JogoDaVida();
        jogo.definirEstadoCelula(1, 1, true);
        jogo.definirEstadoCelula(1, 2, true);
        jogo.definirEstadoCelula(2, 1, true);
        jogo.proximaGeracao();
        assertTrue(jogo.obterEstadoCelula(2, 2)); // A célula deve reviver
    }

    // Caso de Teste 7: Limite de Coordenadas - Testar com uma célula no topo-esquerdo
    @Test
    public void testLimiteCélulaTopoEsquerdo() {
        JogoDaVida jogo = new JogoDaVida();
        // Supondo que o tabuleiro tenha tamanho fixo, como 10x10
        jogo.definirEstadoCelula(0, 0, true); // Definindo no topo esquerdo
        jogo.proximaGeracao();
        assertFalse(jogo.obterEstadoCelula(0, 0)); // Verificando o estado da célula após a
próxima geração
    }
```

```
    // Caso de Teste 8: Limite de Coordenadas - Testar com uma célula no fundo-direita
    @Test
    public void testLimiteCélulaFundoDireita() {
        JogoDaVida jogo = new JogoDaVida();
        jogo.definirEstadoCelula(5, 5, true); // Definindo no fundo direito (assumindo um
tamanho 6x6)
        jogo.proximaGeracao();
        assertFalse(jogo.obterEstadoCelula(5, 5)); // Estado permanece correto após evolução
    }


    // Caso de Teste 9: Célula Não Morre por Superpopulação
    @Test
    public void testCelulaNaoMorrePorSuperpopulacao() {
        JogoDaVida jogo = new JogoDaVida();
        jogo.definirEstadoCelula(2, 2, true);
        jogo.definirEstadoCelula(1, 2, true);
        jogo.definirEstadoCelula(3, 2, true);
        jogo.proximaGeracao();
        assertTrue(jogo.obterEstadoCelula(2, 2)); // A célula deve sobreviver
    }

    // Caso de Teste 10: Valor Inválido para Estado da Célula
    @Test
    public void testValorInvalidoEstadoCelula() {
        JogoDaVida jogo = new JogoDaVida();
        jogo.definirEstadoCelula(2, 2, true); // Estado válido
        jogo.definirEstadoCelula(5, 5, false); // Valor inválido, fora das opções (0 ou 1)
        assertFalse(jogo.obterEstadoCelula(5, 5)); // Não deve ser alterado
    }

    // Caso de Teste 11: Testar Geração em Tabuleiro Preenchido (Múltiplos Vizinhos)
    @Test
    public void testEvolucaoTabuleiro() {
        JogoDaVida jogo = new JogoDaVida();
        jogo.definirEstadoCelula(1, 1, true);
        jogo.definirEstadoCelula(1, 2, true);
        jogo.definirEstadoCelula(2, 1, true);
        jogo.proximaGeracao();
        assertFalse(jogo.obterEstadoCelula(2, 0)); // Deveria morrer
        assertTrue(jogo.obterEstadoCelula(2, 2)); // Deve permanecer viva
    }

}
```