

Requisito 1

Prueba: Límites de tablero I

Creando la prueba:

```
public class TestTicTacToe {
    TicTacToe ticTacToe;

    @BeforeEach
    public void setUp() {
        ticTacToe = new TicTacToe();
    }

    @Test
    public void givenTablero3x3WhenJuegasFueraTEjeXThenRuntimeException() {
        assertThrows(RuntimeException.class, () -> {
            ticTacToe.jugar(5, 2);
        });
    }
}
```

Resultado:

```
java: cannot find symbol
  symbol:   method jugar(int,int)
  location: variable ticTacToe of type produccion.TicTacToe
```

Falla porque jugar aún no se ha creado.

Creando el método jugar:

```
public class TicTacToe {
    public void jugar(int x, int y) {

    }
}
```

Resultado:

```
org.opentest4j.AssertionFailedError: Expected java.lang.RuntimeException to be
thrown, but nothing was thrown.

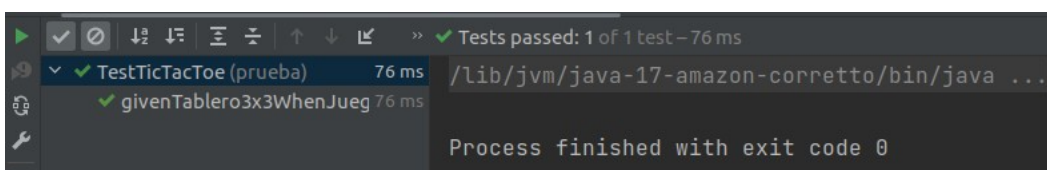
    at org.junit.jupiter.api.Assertions.assertThrows (Assertions.java:3082)
    at
prueba.TestTicTacToe.givenTablero3x3WhenJuegasFueraTableroThenRuntimeException (T
estTicTacToe.java:19)
```

Falla porque solo se creó el método pero esta vacío.

Implementando el método jugar:

```
public class TicTacToe {
    public void jugar(int x, int y) {
        if (x < 1 || x > 3) {
            throw new RuntimeException();
        }
    }
}
```

Al ejecutar la prueba se obtiene:



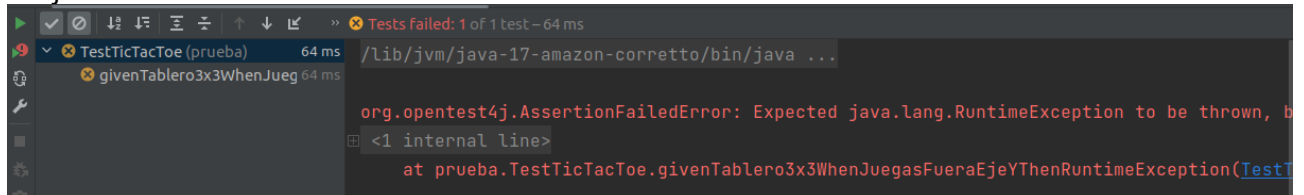
Pasó la prueba

Prueba: límites del tablero II

Se añade la prueba para jugadas fuera del eje y:

```
@Test
public void givenTablero3x3WhenJuegasFueraEjeYThenRuntimeException() {
    assertThrows(RuntimeException.class, () -> {
        ticTacToe.jugar(2, 5);
    });
}
```

Al ejecutar se obtiene:

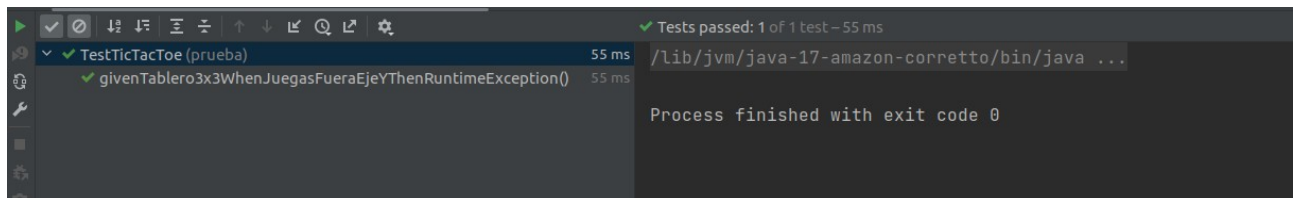


La prueba falla porque el método jugar solo comprueba límites en el eje x.

Módicando el método jugar:

```
public class TicTacToe {
    public void jugar(int x, int y) {
        if (x < 1 || x > 3 || y < 1 || y > 3) {
            throw new RuntimeException();
        }
    }
}
```

Probando nuevamente:



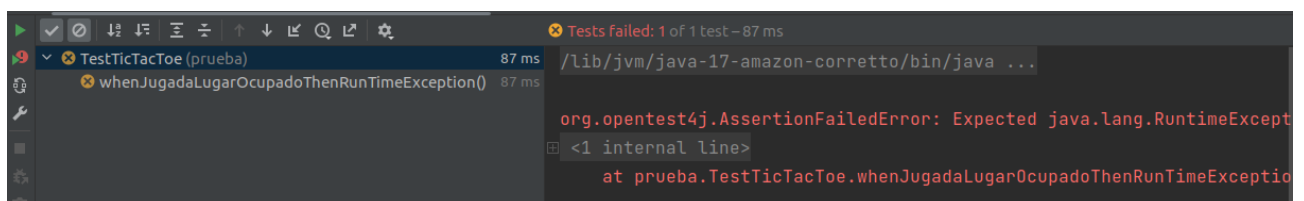
Esta vez pasa la prueba.

Prueba - lugar ocupado

Escribiendo la prueba:

```
@Test
public void whenJugadaLugarOcupadoThenRunTimeException() {
    ticTacToe.jugar(1,1);
    assertThrows(RuntimeException.class, () -> {
        ticTacToe.jugar(1, 1);
    });
}
```

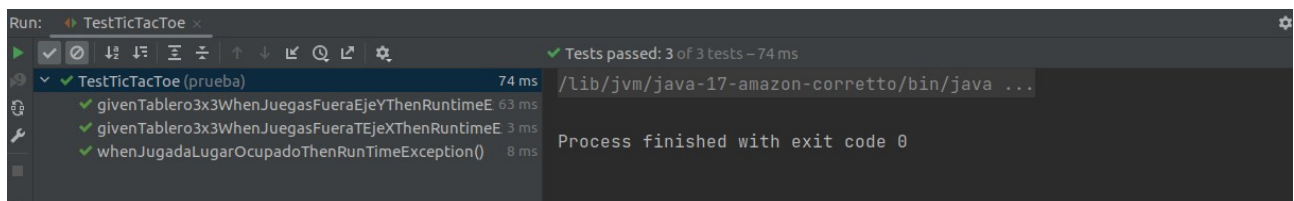
Resultado:



Implementando el código:

```
public class TicTacToe {
    boolean[][] casillaOcupada = new boolean[3][3];
    public void jugar(int x, int y) {
        if (x < 1 || x > 3 || y < 1 || y > 3) {
            throw new RuntimeException();
        }
        if(casillaOcupada[x-1][y-1]){
            throw new RuntimeException();
        }
        casillaOcupada[x-1][y-1] = true;
    }
}
```

Probando nuevamente:



La prueba pasa.

Refactorizando:

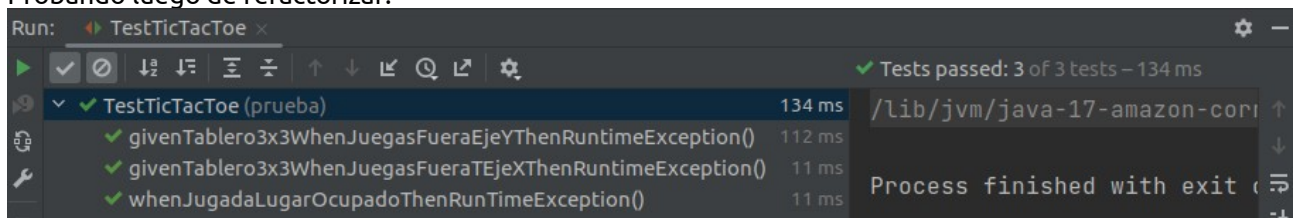
```
package produccion;

public class TicTacToe {
    boolean[][] casillaOcupada = new boolean[3][3];
    public void jugar(int x, int y) {
        verificarJugadaDentroTablero(x,y);
        verificarCasillaOcupada(x, y);
    }

    public void verificarJugadaDentroTablero(int x, int y) {
        if (x < 1 || x > 3 || y < 1 || y > 3) {
            throw new RuntimeException();
        }
    }

    public void verificarCasillaOcupada(int x, int y) {
        if(casillaOcupada[x-1][y-1]){
            throw new RuntimeException();
        }
        casillaOcupada[x-1][y-1] = true;
    }
}
```

Probando luego de refactorizar:



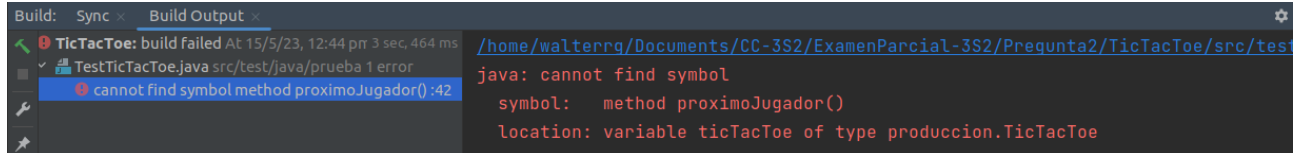
Requisito 2: agregar soporte para dos jugadores

Prueba – X juega primero

Prueba:

```
@Test
public void whenIniciaJuegoThenXJuegaPrimero() {
    assertEquals("X", ticTacToe.proximoJugador());
}
```

Resultado:



Build: Sync x Build Output x

TicTacToe: build failed At 15/5/23, 12:44 pm 3 sec, 464 ms

TestTicTacToe.java src/test/java/prueba 1 error

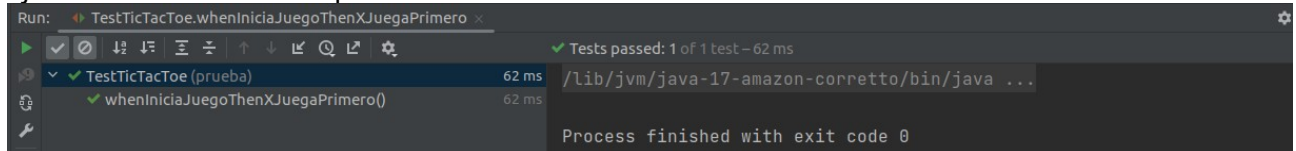
cannot find symbol method proximoJugador():42

java: cannot find symbol
symbol: method proximoJugador()
location: variable ticTacToe of type produccion.TicTacToe

Escribiendo el método:

```
public String proximoJugador() {
    return "X";
}
```

Ejecutando nuevamente la prueba:



Run: TestTicTacToe.whenIniciaJuegoThenXJuegaPrimero x

Tests passed: 1 of 1 test – 62 ms

TestTicTacToe (prueba) 62 ms /lib/jvm/java-17-amazon-corretto/bin/java ...

whenIniciaJuegoThenXJuegaPrimero() 62 ms

Process finished with exit code 0

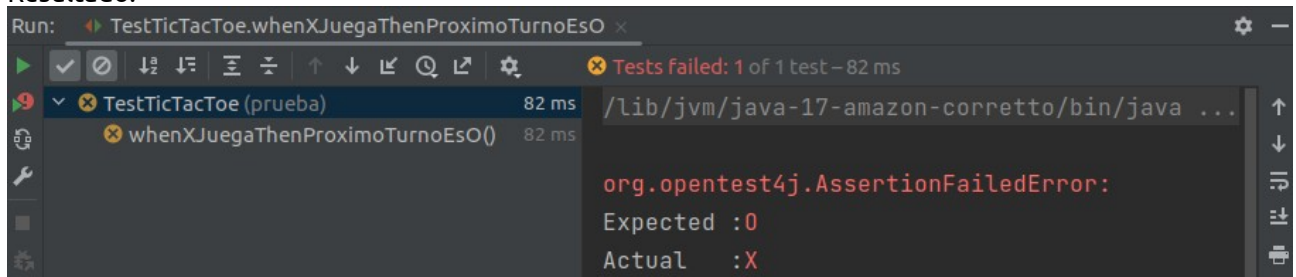
La prueba pasa.

Prueba: O juego justo después de X

Prueba:

```
@Test
public void whenXJuegaThenProximoTurnoEsO() {
    ticTacToe.jugar(1,1); // juega X
    assertEquals("O", ticTacToe.proximoJugador());
}
```

Resultado:



Run: TestTicTacToe.whenXJuegaThenProximoTurnoEsO x

Tests failed: 1 of 1 test – 82 ms

TestTicTacToe (prueba) 82 ms /lib/jvm/java-17-amazon-corretto/bin/java ...

whenXJuegaThenProximoTurnoEsO() 82 ms

org.opentest4j.AssertionFailedError:
Expected :0
Actual :X

Falla porque no cambió de jugador.

Implementación:

```
public class TicTacToe {
    private String proximoJugador = "X";
    boolean[][] casillaOcupada = new boolean[3][3];
    public void jugar(int x, int y) {
        verificarJugadaDentroTablero(x,y);
        verificarCasillaOcupada(x, y);
        if(proximoJugador.equals("X")){
            proximoJugador = "O";
        } else {
            proximoJugador = "X";
        }
    }
}
```

```

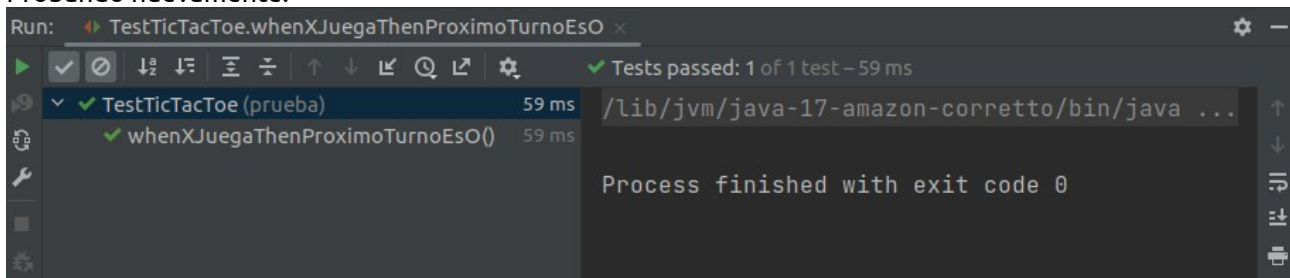
    public void verificarJugadaDentroTablero(int x, int y) {
        if (x < 1 || x > 3 || y < 1 || y > 3) {
            throw new RuntimeException();
        }
    }

    public void verificarCasillaOcupada(int x, int y) {
        if(casillaOcupada[x-1][y-1]){
            throw new RuntimeException();
        }
        casillaOcupada[x-1][y-1] = true;
    }

    public String proximoJugador() {
        return proximoJugador;
    }
}

```

Probando nuevamente:



Pasa la prueba

Prueba: X juega justo después de O

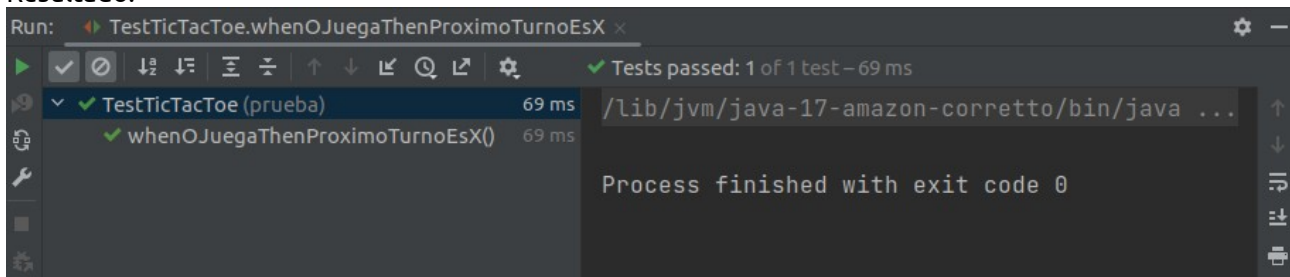
Prueba:

```

@Test
public void whenOJuegaThenProximoTurnoEsX() {
    ticTacToe.jugar(1,1); // juega X
    ticTacToe.jugar(2,2); // juega O
    assertEquals("X", ticTacToe.proximoJugador());
}

```

Resultado:



La prueba pasa sin necesidad de escribir nuevo código, por lo tanto se desecha la prueba.

Requisito 3: agregar condiciones ganadoras

Prueba: por defecto no hay ganador

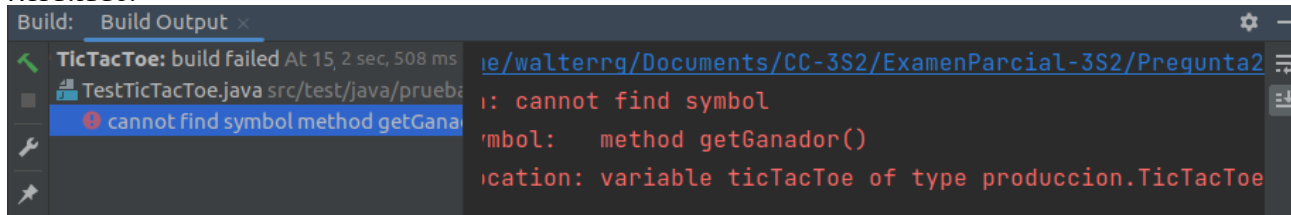
Prueba de que si no hay condición ganadora no hay aganador:

```

@Test
public void whenNoHay3EnLineaThenNoHayGanador() {
    ticTacToe.jugar(1, 1);
    assertEquals("No hay ganador", ticTacToe.getGanador());
}

```

Resultado:



```
Build: Build Output x
TicTacToe: build failed At 15, 2 sec, 508 ms
TestTicTacToe.java src/test/java/prueba
cannot find symbol
symbol: method getGanador()
location: variable ticTacToe of type produccion.TicTacToe
```

Falla porque aún no se creó el método getGanador.

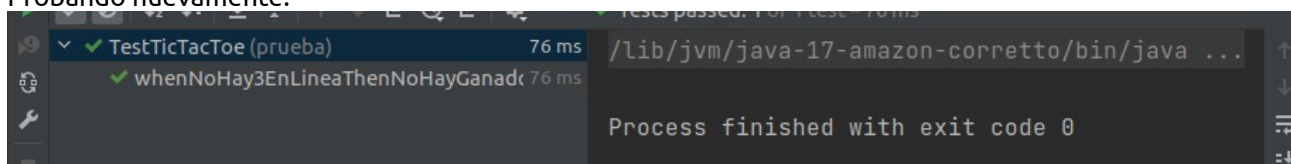
Implementación:

Se añade a la clase TicTacToe:

```
private String ganador = "No hay ganador";

public String getGanador() {
    return ganador;
}
```

Probando nuevamente:



```
TestTicTacToe (prueba) 76 ms
whenNoHay3EnLineaThenNoHayGanador 76 ms
Process finished with exit code 0
```

La prueba pasa

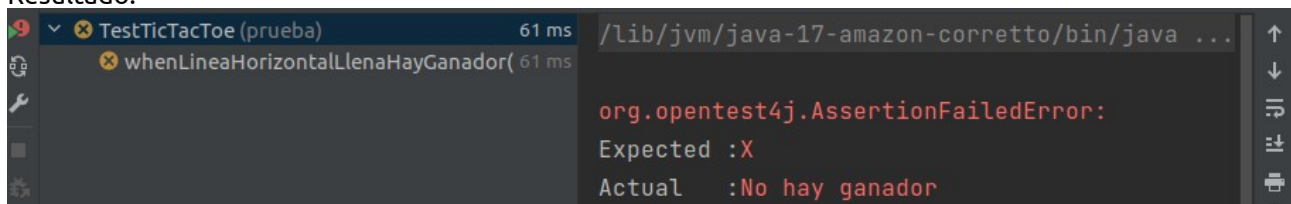
Prueba – condición ganadora I

Prueba para ganar con línea horizontal:

```
@Test
public void whenLineaHorizontalLlenaHayGanador() {
    ticTacToe.jugar(1, 1);
    ticTacToe.jugar(2, 1);
    ticTacToe.jugar(1, 2);
    ticTacToe.jugar(2, 2);
    ticTacToe.jugar(1, 3);
    assertEquals("X", ticTacToe.getGanador());
}
```

Se llena la primera fila de "X"

Resultado:



```
TestTicTacToe (prueba) 61 ms
whenLineaHorizontalLlenaHayGanador( 61 ms
org.opentest4j.AssertionFailedError:
Expected :X
Actual   :No hay ganador
```

Falla porque aún no se implementa

Implementación:

```
public class TicTacToe {
    private String proximoJugador = "X";
    private String ganador = "No hay ganador";
    boolean[][] casillaOcupada = new boolean[3][3];
    String[][] tablero = new String[3][3];

    public void jugar(int x, int y) {
        verificarJugadaDentroTablero(x, y);
        verificarCasillaOcupada(x, y);
        tablero[x - 1][y - 1] = proximoJugador();
        if ((tablero[x-1][0] == tablero[x-1][1]) && (tablero[x-1][1] ==
        tablero[x-1][2])) {
            ganador = proximoJugador();
        }
        if (proximoJugador.equals("X")) {
            proximoJugador = "O";
        }
    }
}
```

```

    } else {
        proximoJugador = "X";
    }
}

public String getGanador() {
    return ganador;
}

public void verificarJugadaDentroTablero(int x, int y) {
    if (x < 1 || x > 3 || y < 1 || y > 3) {
        throw new RuntimeException();
    }
}

public void verificarCasillaOcupada(int x, int y) {
    if (casillaOcupada[x - 1][y - 1]) {
        throw new RuntimeException();
    }
    casillaOcupada[x - 1][y - 1] = true;
}

public String proximoJugador() {
    return proximoJugador;
}

```

Resultado de la prueba:

```

TestTicTacToe (prueba) 52 ms
  whenLineaHorizontalLlenaHayGanador( 52 ms
    Process finished with exit code 0

```

La prueba pasa

Refactorización:

```

package produccion;

public class TicTacToe {
    private String proximoJugador = "X";
    String jugadorActual = "";
    private String ganador = "No hay ganador";
    boolean[][] casillaOcupada = new boolean[3][3];
    String[][] tablero = new String[3][3];

    public void jugar(int x, int y) {
        jugadorActual = proximoJugador();
        verificarJugadaDentroTablero(x, y);
        verificarCasillaOcupada(x, y);
        realizarJugada(x, y);
        establecerGanador(x, y);
        cambiarTurno();
    }

    public void establecerGanador(int x, int y) {
        if ((tablero[x - 1][0] == tablero[x - 1][1]) && (tablero[x - 1][1] ==
tablero[x - 1][2])) {
            ganador = jugadorActual;
        }
    }

    public void realizarJugada(int x, int y){
        tablero[x - 1][y - 1] = jugadorActual;
    }

    public void cambiarTurno() {
        if (proximoJugador.equals("X")) {
            proximoJugador = "O";
        } else {
            proximoJugador = "X";
        }
    }
}

```

```

    }

    public String getGanador() {
        return ganador;
    }

    public void verificarJugadaDentroTablero(int x, int y) {
        if (x < 1 || x > 3 || y < 1 || y > 3) {
            throw new RuntimeException("Jugada fuera del tablero");
        }
    }

    public void verificarCasillaOcupada(int x, int y) {
        if (casillaOcupada[x - 1][y - 1]) {
            throw new RuntimeException("Casilla está ocupada");
        }
        casillaOcupada[x - 1][y - 1] = true;
    }

    public String proximoJugador() {
        return proximoJugador;
    }
}

```

Realizando pruebas luego de refactorizar:

```

TestTicTacToe (prueba) 92 ms
  ✓ whenLineaHorizontalLlenaHayGanador() 75 ms
  ✓ whenXJuegaThenProximoTurnoEsO() 2 ms
  ✓ givenTablero3x3WhenJuegasFueraEjeYTL 4 ms
  ✓ whenIniciaJuegoThenXJuegaPrimero() 2 ms
  ✓ givenTablero3x3WhenJuegasFueraTEjeX 3 ms
  ✓ whenJugadaLugarOcupadoThenRunTime 4 ms
  ✓ whenNoHay3EnLineaThenNoHayGanador 2 ms

```

Process finished with exit code 0

Las pruebas pasan

Prueba – condición ganadora II

Prueba para ganar con línea vertical:

```

@Test
public void whenLineaVerticalLlenaHayGanador() {
    ticTacToe.jugar(3, 3);
    ticTacToe.jugar(1, 2);
    ticTacToe.jugar(1, 1);
    ticTacToe.jugar(2, 2);
    ticTacToe.jugar(1, 3);
    ticTacToe.jugar(3, 2);
    assertEquals("O", ticTacToe.getGanador());
}

```

Resultado de la prueba:

```

TestTicTacToe (prueba) 68 ms
  ✗ whenLineaVerticalLlenaHayGanador() 68 ms

```

org.opentest4j.AssertionFailedError:
Expected :0
Actual :No hay ganador

Falla porque solo se implementó para el caso horizontal

Implementación:

```


public void establecerGanador(int x, int y) {
    if ((tablero[x - 1][0] == tablero[x - 1][1]) && (tablero[x - 1][1] ==
    tablero[x - 1][2])) {
        ganador = jugadorActual;
    }
    if ((tablero[0][y - 1] == tablero[1][y - 1]) && (tablero[1][y - 1] ==
    tablero[2][y - 1])) {
        ganador = jugadorActual;
    }
}

```



```
}  
}
```

Realizando la prueba nuevamente:



The screenshot shows the IDE's test runner interface. The top bar indicates the test suite 'TestTicTacToe (prueba)' passed with a green checkmark and took 53 ms. Below it, the specific test 'whenLineaVerticalLlenaHayGanador()' also passed with a green checkmark and took 53 ms. The output pane on the right shows the command path and the message 'Process finished with exit code 0'.

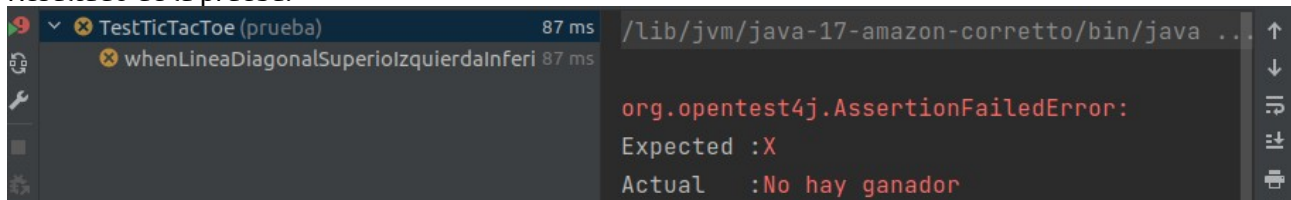
La prueba pasa

Prueba – condición ganadora III

Prueba para ganar con línea diagonal desde la parte superior izquierda a la parte inferior derecha:

```
@Test  
public void whenLineaDiagonalSuperioIzquierdaInferiorDerechaLlenaHayGanador() {  
    ticTacToe.jugar(1, 1);  
    ticTacToe.jugar(1, 2);  
    ticTacToe.jugar(2, 2);  
    ticTacToe.jugar(2, 3);  
    ticTacToe.jugar(3, 3);  
    assertEquals("X", ticTacToe.getGanador());  
}
```

Resultado de la prueba:



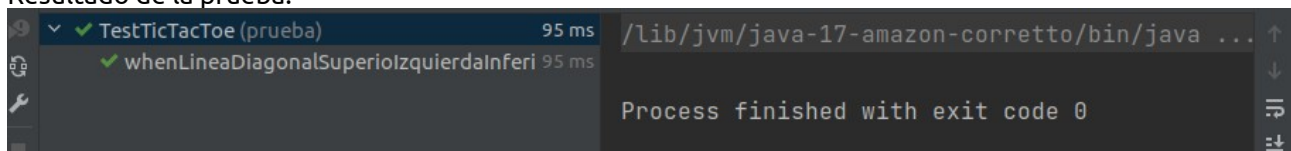
The screenshot shows the IDE's test runner interface. The top bar indicates the test suite 'TestTicTacToe (prueba)' failed with a red X and took 87 ms. Below it, the specific test 'whenLineaDiagonalSuperioIzquierdaInferiorDerechaLlenaHayGanador()' also failed with a red X and took 87 ms. The output pane on the right shows the error message: 'org.opentest4j.AssertionFailedError: Expected :X Actual :No hay ganador'.

Prueba falla porque solo se implementó casos horizontal y vertical

Implementación:

```
public void establecerGanador(int x, int y) {  
    if ((tablero[x - 1][0] == tablero[x - 1][1]) && (tablero[x - 1][1] ==  
    tablero[x - 1][2])) {  
        ganador = jugadorActual;  
    }  
    if ((tablero[0][y - 1] == tablero[1][y - 1]) && (tablero[1][y - 1] ==  
    tablero[2][y - 1])) {  
        ganador = jugadorActual;  
    }  
    if ((tablero[0][0] == tablero[1][1]) && (tablero[1][1] == tablero[2][2])  
        && tablero[1][1] != null) {  
        ganador = jugadorActual;  
    }  
}
```

Resultado de la prueba:



The screenshot shows the IDE's test runner interface. The top bar indicates the test suite 'TestTicTacToe (prueba)' passed with a green checkmark and took 95 ms. Below it, the specific test 'whenLineaDiagonalSuperioIzquierdaInferiorDerechaLlenaHayGanador()' also passed with a green checkmark and took 95 ms. The output pane on the right shows the message 'Process finished with exit code 0'.

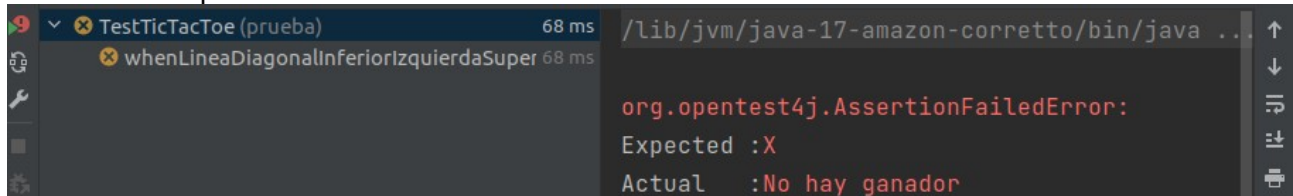
Prueba pasa

Prueba – condición ganadora IV

Prueba para ganar con línea diagonal desde la parte inferior izquierda a la parte superior derecha:

```
@Test
public void whenLineaDiagonalInferiorIzquierdaSuperiorDerechaLlenaHayGanador() {
    ticTacToe.jugar(1, 3);
    ticTacToe.jugar(1, 2);
    ticTacToe.jugar(2, 2);
    ticTacToe.jugar(2, 3);
    ticTacToe.jugar(3, 1);
    assertEquals("X", ticTacToe.getGanador());
}
```

Resultado de la prueba:



```
TestTicTacToe (prueba) 68 ms
  whenLineaDiagonalInferiorIzquierdaSuperiorDerechaLlenaHayGanador 68 ms

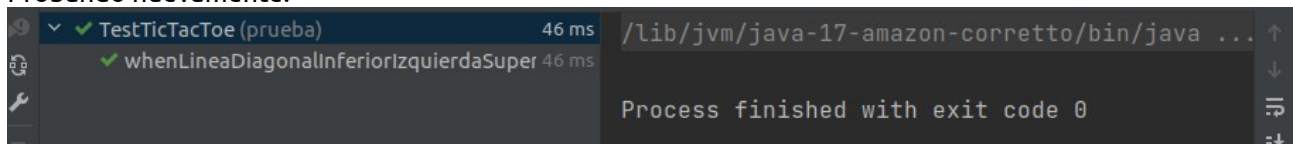
org.opentest4j.AssertionFailedError:
Expected :X
Actual   :No hay ganador
```

Prueba falla

Implementación:

```
public void establecerGanador(int x, int y) {
    if ((tablero[x - 1][0] == tablero[x - 1][1]) && (tablero[x - 1][1] ==
    tablero[x - 1][2])) {
        ganador = jugadorActual;
    }
    if ((tablero[0][y - 1] == tablero[1][y - 1]) && (tablero[1][y - 1] ==
    tablero[2][y - 1])) {
        ganador = jugadorActual;
    }
    if ((tablero[0][0] == tablero[1][1]) && (tablero[1][1] == tablero[2][2])
        && tablero[1][1] != null) {
        ganador = jugadorActual;
    }
    if ((tablero[0][2] == tablero[1][1]) && (tablero[1][1] == tablero[2][0])
        && tablero[1][1] != null) {
        ganador = jugadorActual;
    }
}
```

Probando nuevamente:



```
TestTicTacToe (prueba) 46 ms
  whenLineaDiagonalInferiorIzquierdaSuperiorDerechaLlenaHayGanador 46 ms

Process finished with exit code 0
```

La prueba pasa:

Refactorización:

```
package produccion;

public class TicTacToe {
    private String proximoJugador = "X";
    private String jugadorActual = "";
    private String ganador = "No hay ganador";
    boolean[][] casillaOcupada = new boolean[3][3];
    private String[][] tablero = new String[3][3];

    public void jugar(int x, int y) {
        jugadorActual = proximoJugador();
        verificarJugadaDentroTablero(x, y);
        verificarCasillaOcupada(x, y);
        realizarJugada(x, y);
        establecerGanador(x, y);
        cambiarTurno();
    }
}
```

```

    public void establecerGanador(int x, int y) {
        // Verifica si gana con línea horizontal:
        if ((tablero[x - 1][0] == tablero[x - 1][1]) && (tablero[x - 1][1] ==
tablero[x - 1][2])) {
            ganador = jugadorActual;
        }
        // Verifica si gana con línea vertical:
        if ((tablero[0][y - 1] == tablero[1][y - 1]) && (tablero[1][y - 1] ==
tablero[2][y - 1])) {
            ganador = jugadorActual;
        }
        // Verifica si gana con línea diagonal:
        // Diagonal superior izquierda a inferior derecha:
        if ((tablero[0][0] == tablero[1][1]) && (tablero[1][1] == tablero[2][2])
&& tablero[1][1] != null) {
            ganador = jugadorActual;
        }
        // Diagonal inferior izquierda a superior derecha
        if ((tablero[0][2] == tablero[1][1]) && (tablero[1][1] == tablero[2][0])
&& tablero[1][1] != null) {
            ganador = jugadorActual;
        }
    }

    public void realizarJugada(int x, int y) {
        tablero[x - 1][y - 1] = jugadorActual;
    }

    public void cambiarTurno() {
        if (proximoJugador.equals("X")) {
            proximoJugador = "O";
        } else {
            proximoJugador = "X";
        }
    }

    public String getGanador() {
        return ganador;
    }

    public void verificarJugadaDentroTablero(int x, int y) {
        if (x < 1 || x > 3 || y < 1 || y > 3) {
            throw new RuntimeException("Jugada fuera del tablero");
        }
    }

    public void verificarCasillaOcupada(int x, int y) {
        if (casillaOcupada[x - 1][y - 1]) {
            throw new RuntimeException("Casilla está ocupada");
        }
        casillaOcupada[x - 1][y - 1] = true;
    }

    public String proximoJugador() {
        return proximoJugador;
    }
}

```

```
✓ TestTicTacToe (prueba) 107 ms
  ✓ whenLineaDiagonalSuperiorIzquierdaInferior 55 ms
  ✓ whenLineaHorizontalLlenaHayGanador() 2 ms
  ✓ whenXJuegaThenProximoTurnoEsO() 3 ms
  ✓ givenTablero3x3WhenJuegasFueraEjeYThen 6 ms
  ✓ whenLineaVerticalLlenaHayGanador() 11 ms
  ✓ whenIniciaJuegoThenXJuegaPrimero() 6 ms
  ✓ whenLineaDiagonalInferiorIzquierdaSuperior 12 ms
  ✓ givenTablero3x3WhenJuegasFueraTEjeXThen 4 ms
  ✓ whenJugadaLugarOcupadoThenRunTimeError 5 ms
  ✓ whenNoHay3EnLineaThenNoHayGanador() 3 ms
```

/lib/jvm/java-17-amazon-corretto/bin/java ...

Process finished with exit code 0

Requisito 4: condiciones de empate

```
@Test
public void whenNoQuedanCasillasYNoHayGanadorThenEmpate() {
    ticTacToe.jugar(1, 1);
    ticTacToe.jugar(2, 2);
    ticTacToe.jugar(1, 3);
    ticTacToe.jugar(1, 2);
    ticTacToe.jugar(3, 2);
    ticTacToe.jugar(2, 3);
    ticTacToe.jugar(2, 1);
    ticTacToe.jugar(3, 1);
    ticTacToe.jugar(3, 3);

    assertTrue(ticTacToe.esEmpate());
}
```

TicTacToe: build failed At 15, 1 sec, 787 ms
TestTicTacToe.java src/test/java/prueb...
cannot find symbol method esEmpate

```
public boolean esEmpate() {
    boolean hayCasillasVacias = false;
    for(int i=0;i<3;i++){
        for(int j=0;j<3;j++){
            if(tablero[i][j] == null) {
                hayCasillasVacias = true;
            }
        }
    }
    if(!hayCasillasVacias && ganador == "No hay ganador") {
        return true;
    }
    return false;
}
```

```
✓ TestTicTacToe (prueba) 50 ms /lib/jvm/java-17-amazon-corretto/bin/java ...
  ✓ whenNoQuedanCasillasYNOHayGanadorTt 50 ms

Process finished with exit code 0
```

Prueba pasa

Refactorizando:

```
package produccion;

public class TicTacToe {
    private String proximoJugador = "X";
    private String jugadorActual = "";
    private String ganador = "No hay ganador";
    boolean[][] casillaOcupada = new boolean[3][3];
    private String[][] tablero = new String[3][3];

    public void jugar(int x, int y) {
        jugadorActual = proximoJugador();
        verificarJugadaDentroTablero(x, y);
        verificarCasillaOcupada(x, y);
        realizarJugada(x, y);
        if(!esGanador(x, y) && !esEmpate()){
            cambiarTurno();
        }
    }

    public boolean esGanador(int x, int y) {
        // Verifica si gana con línea horizontal:
        if ((tablero[x - 1][0] == tablero[x - 1][1]) && (tablero[x - 1][1] ==
tablero[x - 1][2])) {
            ganador = jugadorActual;
            return true;
        }
        // Verifica si gana con línea vertical:
        if ((tablero[0][y - 1] == tablero[1][y - 1]) && (tablero[1][y - 1] ==
tablero[2][y - 1])) {
            ganador = jugadorActual;
            return true;
        }
        // Verifica si gana con línea diagonal:
        // Diagonal superior izquierda a inferior derecha:
        if ((tablero[0][0] == tablero[1][1]) && (tablero[1][1] == tablero[2][2])
&& tablero[1][1] != null) {
            ganador = jugadorActual;
            return true;
        }
        // Diagonal inferior izquierda a superior derecha
        if ((tablero[0][2] == tablero[1][1]) && (tablero[1][1] == tablero[2][0])
&& tablero[1][1] != null) {
            ganador = jugadorActual;
            return true;
        }
        return false;
    }

    public void realizarJugada(int x, int y) {
        tablero[x - 1][y - 1] = jugadorActual;
    }

    public void cambiarTurno() {
        if (proximoJugador.equals("X")) {
            proximoJugador = "O";
        } else {
            proximoJugador = "X";
        }
    }

    public String getGanador() {
        return ganador;
    }

    public void verificarJugadaDentroTablero(int x, int y) {
        if (x < 1 || x > 3 || y < 1 || y > 3) {
            throw new RuntimeException("Jugada fuera del tablero");
        }
    }

    public void verificarCasillaOcupada(int x, int y) {
```

```

        if (casillaOcupada[x - 1][y - 1]) {
            throw new RuntimeException("Casilla está ocupada");
        }
        casillaOcupada[x - 1][y - 1] = true;
    }

    public String proximoJugador() {
        return proximoJugador;
    }

    public boolean esEmpate() {
        if (!hayCasillasVacias() && ganador == "No hay ganador") {
            return true;
        }
        return false;
    }

    boolean hayCasillasVacias() {
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                if (tablero[i][j] == null) {
                    return true;
                }
            }
        }
        return false;
    }
}

```



Ejecutando nuevamente las pruebas:

Las pruebas pasan

Cobertura de código

Ejecutando las pruebas usando JaCoco se obtienen los resultados de cobertura de código:

| Coverage: TestTicTacToe x | | | | |
|---------------------------|------------|--------------|--------------|-------------|
| | | | | |
| Element ^ | Class, % | Method, % | Line, % | Branch, % |
| v production | 100% (1/1) | 100% (10/10) | 100% (49/49) | 93% (43/46) |
| TicTacToe | 100% (1/1) | 100% (10/10) | 100% (49/49) | 93% (43/46) |

| Element ^ | Class, % | Method, % | Line, % | Branch, % |
|--|------------|--------------|--------------|-------------|
| ▼  production | 100% (1/1) | 100% (10/10) | 100% (49/49) | 93% (43/46) |
|  TicTacToe | 100% (1/1) | 100% (10/10) | 100% (49/49) | 93% (43/46) |