

PREGUNTA 2

Para la pregunta 2, se procedió a reutilizar el código del TicTacToe.

Requisito1: colocación de piezas

Requisito2: agregar soporte para dos jugadores

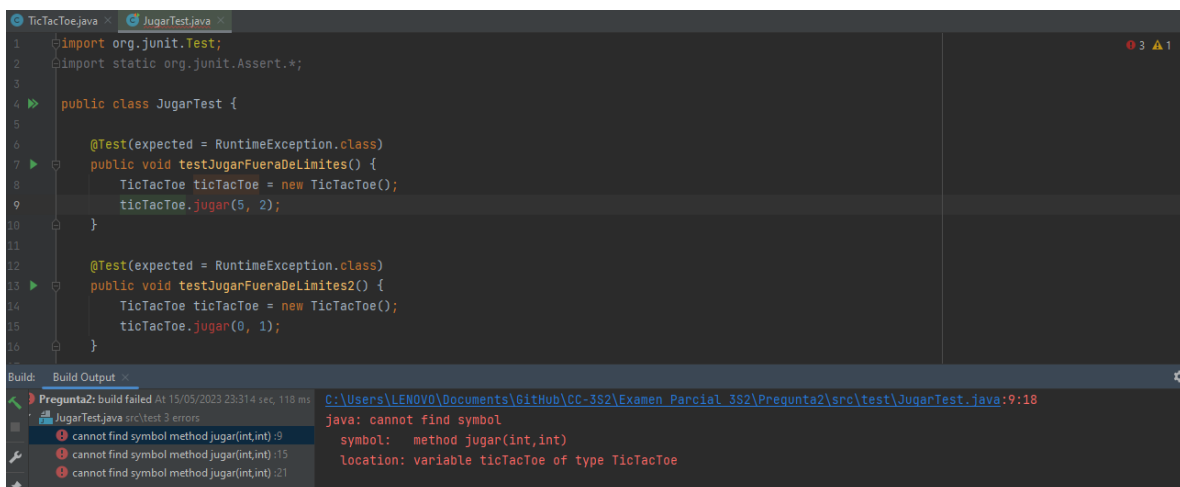
Requisito3: agregar condiciones ganadores

Requisito4: condiciones de empate

Requisito 1: colocación de piezas**Prueba: límites del tablero I**

Todo lo que tenemos que hacer es crear el método jugar y asegurarnos de que arroja RuntimeException cuando el argumento x es menor que 1 o mayor que 3 (el tablero es 3x3).

Debes ejecutar esta prueba tres veces. La primera vez, debería fallar porque el método jugar no existe.



```
1 import org.junit.Test;
2 import static org.junit.Assert.*;
3
4 public class JugarTest {
5
6     @Test(expected = RuntimeException.class)
7     public void testJugarFueraDeLmites() {
8         TicTacToe ticTacToe = new TicTacToe();
9         ticTacToe.jugar(5, 2);
10    }
11
12    @Test(expected = RuntimeException.class)
13    public void testJugarFueraDeLmites2() {
14        TicTacToe ticTacToe = new TicTacToe();
15        ticTacToe.jugar(0, 1);
16    }
17 }
```

Build: Build Output

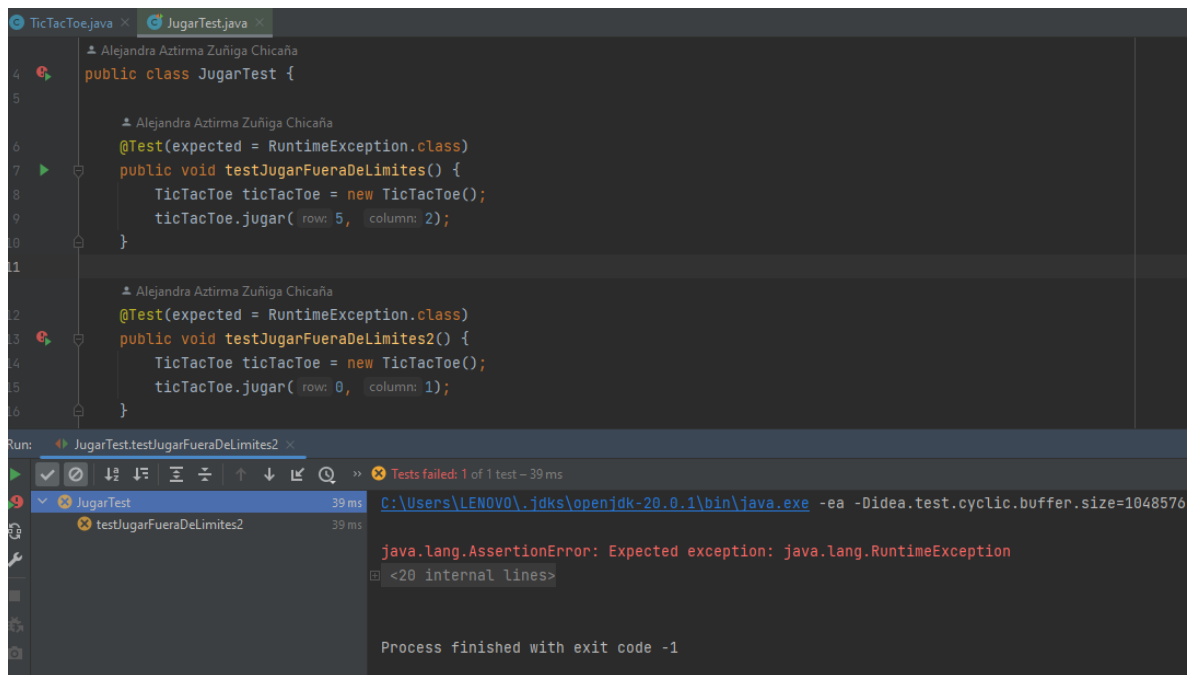
Pregunta2: build failed At 15/05/2023 23:314 sec, 118 ms

JugarTest.java src/Test 3 errors

- cannot find symbol method jugar(int,int) :9
- cannot find symbol method jugar(int,int) :15
- cannot find symbol method jugar(int,int) :21

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

Una vez que se agrega, debería fallar porque no se lanza RuntimeException.



```
4 public class JugarTest {
5
6     @Test(expected = RuntimeException.class)
7     public void testJugarFueraDeLimites() {
8         TicTacToe ticTacToe = new TicTacToe();
9         ticTacToe.jugar( row: 5, column: 2);
10    }
11
12    @Test(expected = RuntimeException.class)
13    public void testJugarFueraDeLimites2() {
14        TicTacToe ticTacToe = new TicTacToe();
15        ticTacToe.jugar( row: 0, column: 1);
16    }
17 }
```

Run: JugarTest.testJugarFueraDeLimites2

Tests failed: 1 of 1 test - 39 ms

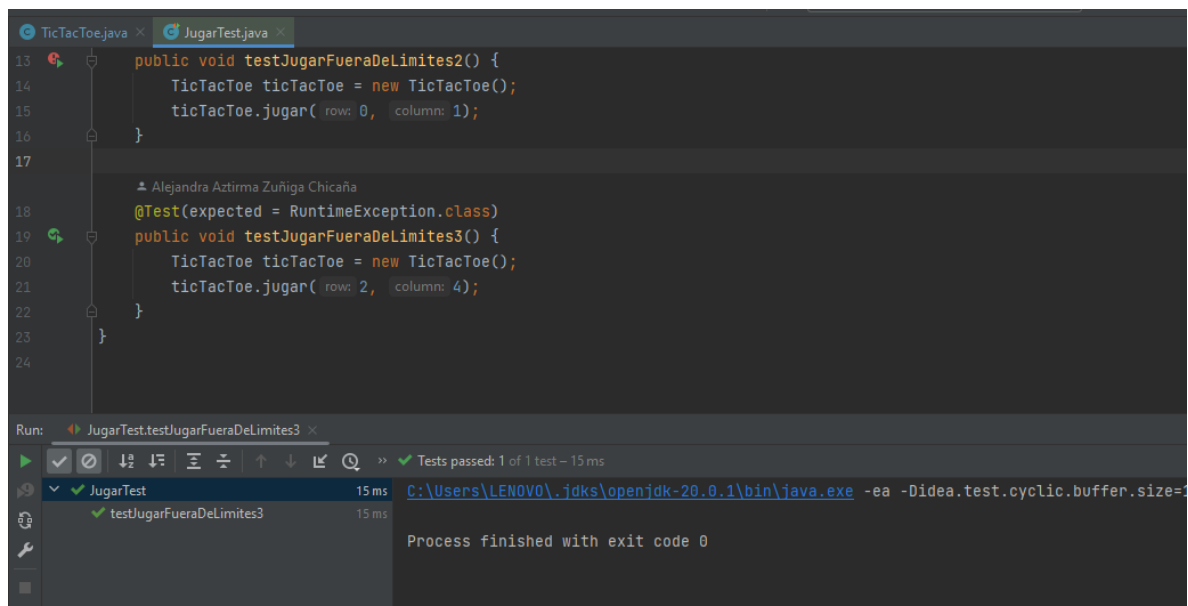
JugarTest 39 ms

testJugarFueraDeLimites2 39 ms

java.lang.AssertionError: Expected exception: java.lang.RuntimeException
<20 internal lines>

Process finished with exit code -1

La tercera vez, debería tener éxito porque el código que corresponde a esta prueba está completamente implementado.



```
13 public void testJugarFueraDeLimites2() {
14     TicTacToe ticTacToe = new TicTacToe();
15     ticTacToe.jugar( row: 0, column: 1);
16 }
17
18 @Test(expected = RuntimeException.class)
19 public void testJugarFueraDeLimites3() {
20     TicTacToe ticTacToe = new TicTacToe();
21     ticTacToe.jugar( row: 2, column: 4);
22 }
23 }
```

Run: JugarTest.testJugarFueraDeLimites3

Tests passed: 1 of 1 test - 15 ms

JugarTest 15 ms

testJugarFueraDeLimites3 15 ms

Process finished with exit code 0

Prueba: límite de tablero II

Realiza la implementación de esta especificación es casi la misma que la anterior.



```
15  @Test(expected = RuntimeException.class)
16  public void testJugarFueraDeLimitesY() {
17      TicTacToe ticTacToe = new TicTacToe();
18      ticTacToe.jugar( row: 1, column: 5);
19  }
20  }
```

Run: JugarTest.testJugarFueraDeLimitesY < X

Tests passed: 1 of 1 test - 8 ms

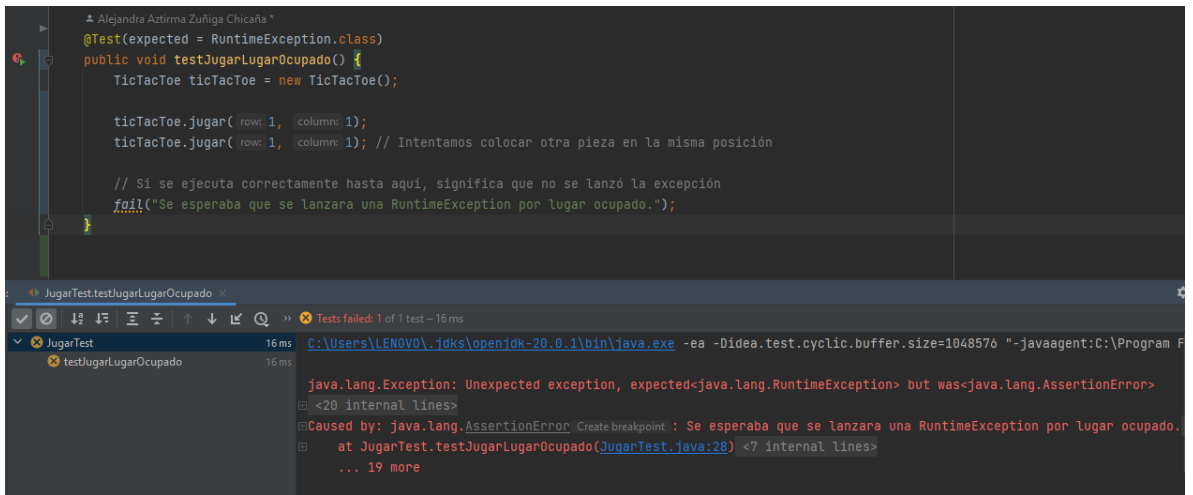
JugarTest 8 ms C:\Users\LENOVO\jdk\openjdk-20.0.1\bin\java.exe -ea -Didea.test.cyclic.buffer.size=1048576

testJugarFueraDeLimitesY 0 ms

Process finished with exit code 0

Prueba: lugar ocupado

Una vez establecidos los límites del tablero, ahora solo falta asegurar de que se pongan en lugares desocupados, se creó la prueba para esto y se complementó el método Jugar.



```
15  @Test(expected = RuntimeException.class)
16  public void testJugarLugarOcupado() {
17      TicTacToe ticTacToe = new TicTacToe();
18
19      ticTacToe.jugar( row: 1, column: 1);
20      ticTacToe.jugar( row: 1, column: 1); // Intentamos colocar otra pieza en la misma posición
21
22      // Si se ejecuta correctamente hasta aquí, significa que no se lanzó la excepción
23      fail("Se esperaba que se lanzara una RuntimeException por lugar ocupado.");
24  }
```

Run: JugarTest.testJugarLugarOcupado < X

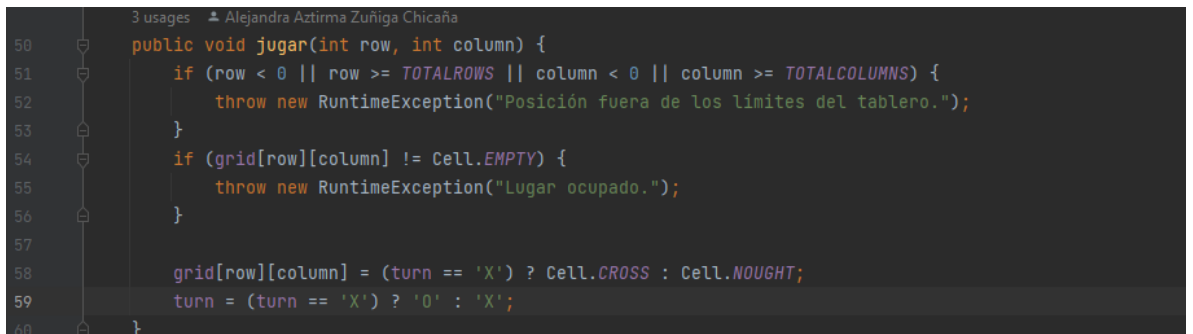
Tests failed: 1 of 1 test - 16 ms

JugarTest 16 ms C:\Users\LENOVO\jdk\openjdk-20.0.1\bin\java.exe -ea -Didea.test.cyclic.buffer.size=1048576 "-javaagent:C:\Program F

testJugarLugarOcupado 16 ms

java.lang.Exception: Unexpected exception, expected<java.lang.RuntimeException> but was<java.lang.AssertionError>
<20 internal lines>
Caused by: java.lang.AssertionError: Se esperaba que se lanzara una RuntimeException por lugar ocupado.
at JugarTest.testJugarLugarOcupado(JugarTest.java:20) <7 internal lines>
... 19 more

Se tiene esta salida debido a que aún no se complementó el método jugar, se procede a complementar y ahora si la prueba pasa.



```
3 usages Alejandro Azirma Zuñiga Chicaña
50 public void jugar(int row, int column) {
51     if (row < 0 || row >= TOTALROWS || column < 0 || column >= TOTALCOLUMNS) {
52         throw new RuntimeException("Posición fuera de los límites del tablero.");
53     }
54     if (grid[row][column] != Cell.EMPTY) {
55         throw new RuntimeException("Lugar ocupado.");
56     }
57
58     grid[row][column] = (turn == 'X') ? Cell.CROSS : Cell.NOUGHT;
59     turn = (turn == 'X') ? 'O' : 'X';
60 }
```

Prueba pasada una vez se implementa el código

```
11  Alejandro Azirma Zuñiga Chicaña
12  @Test(expected = RuntimeException.class)
13  public void testJugarLugarOcupado() {
14      TicTacToe ticTacToe = new TicTacToe();
15
16      ticTacToe.jugar( row: 1, column: 1);
17      ticTacToe.jugar( row: 1, column: 1); // Intentamos colocar otra pieza en la misma posición
18
19      // Si se ejecuta correctamente hasta aquí, significa que no se lanzó la excepción
20      fail("Se esperaba que se lanzara una RuntimeException por lugar ocupado.");
21  }
22
23  }
```

Run: JugarTest.testJugarLugarOcupado

Tests passed: 1 of 1 test - 14 ms

JugarTest 14 ms C:\Users\LENOVO\jdk\openjdk-20.0.1\bin\java.exe -ea -Didea.test.cyclic.buffer.size=1048576 "-javaagent:...

testJugarLugarOcupado 14 ms

Process finished with exit code 0

Ahora nos pide Refactorizar el método Jugar, esto para que se tenga un mejor entendimiento del código

```
public void jugar(int row, int column) {
    validarLimitesTablero(row, column);
    validarLugarOcupado(row, column);
    realizarMovimiento(row, column);
    cambiarTurno();
}

1 usage new *
private void validarLimitesTablero(int row, int column) {
    if (row < 0 || row >= TOTALROWS || column < 0 || column >= TOTALCOLUMNS) {
        throw new RuntimeException("Posición fuera de los límites del tablero.");
    }
}

1 usage new *
private void validarLugarOcupado(int row, int column) {
    if (grid[row][column] != Cell.EMPTY) {
        throw new RuntimeException("Lugar ocupado.");
    }
}

1 usage new *
private void realizarMovimiento(int row, int column) {
    grid[row][column] = (turn == 'X') ? Cell.CROSS : Cell.NOUGHT;
}

1 usage new *
private void cambiarTurno() {
    turn = (turn == 'X') ? 'O' : 'X';
}
```

Requisito 2: agregar soporte para dos jugadores

El siguiente requisito se dividió en estas 3 pruebas:

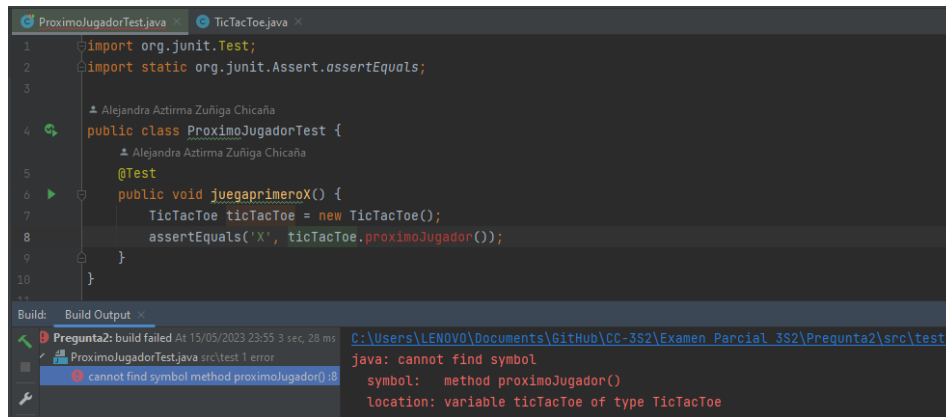
- El primer turno lo debe jugar el jugador X.
- Si el último turno fue jugado por X, entonces el próximo turno debe ser jugado por O

- Si el último turno fue jugado por O, entonces el próximo turno debe ser jugado por X

Procederemos a escribir las pruebas, antes que la implementación del método, en este caso el método `getTurn()`;

Prueba – X juega primero

Se creó la prueba para que el método `proximoJugador` devuelva X. Pero al pasar ejecutarlo se ve que no se compila puesto que aún no se creó dicho método.



```
1 import org.junit.Test;
2 import static org.junit.Assert.assertEquals;
3
4 public class ProximoJugadorTest {
5     @Test
6     public void juegaprimeroX() {
7         TicTacToe ticTacToe = new TicTacToe();
8         assertEquals('X', ticTacToe.proximoJugador());
9     }
10 }
```

Build: Build Output

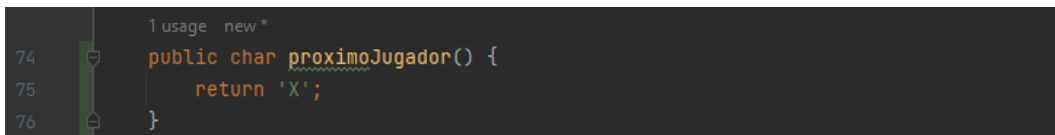
Pregunta2: build failed At 15/05/2023 23:55 3 sec, 28 ms

ProximoJugadorTest.java src/test 1 error

cannot find symbol method proximoJugador()

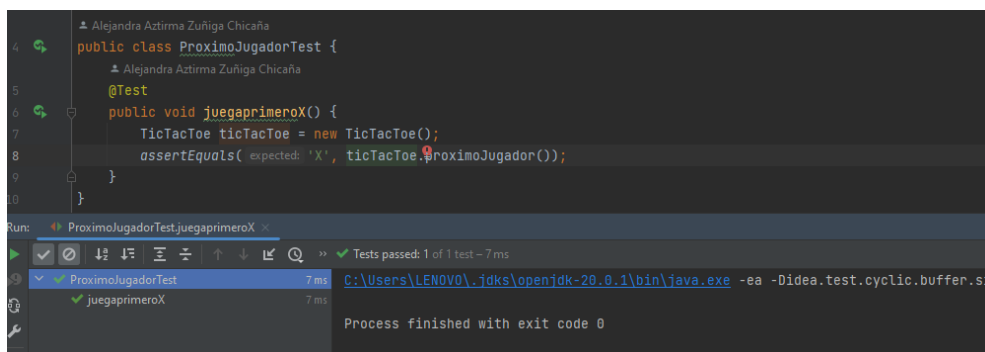
java: cannot find symbol
symbol: method proximoJugador()
location: variable ticTacToe of type TicTacToe

Se implementó el método `ProximoJugador`, el cual devuelve el turno inicial, en este caso X. Esta implementación inicial solo cumple con la primera prueba. En las pruebas posteriores, refinaremos este código para determinar correctamente qué jugador debe jugar a continuación.



```
1 usage new *
74 public char proximoJugador() {
75     return 'X';
76 }
```

Ahora si una vez creado nuestro método nuestra prueba pasa.



```
4 public class ProximoJugadorTest {
5     @Test
6     public void juegaprimeroX() {
7         TicTacToe ticTacToe = new TicTacToe();
8         assertEquals( expected: 'X', ticTacToe.proximoJugador());
9     }
10 }
```

Run: ProximoJugadorTest.juegaprimeroX

Tests passed: 1 of 1 test - 7 ms

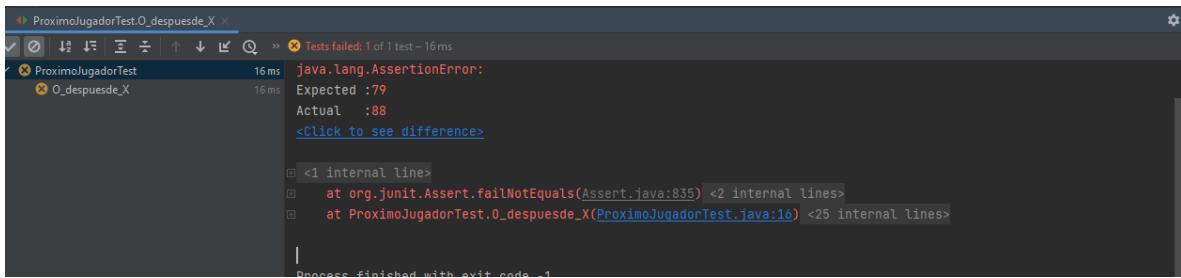
ProximoJugadorTest 7 ms

juegaprimeroX 7 ms

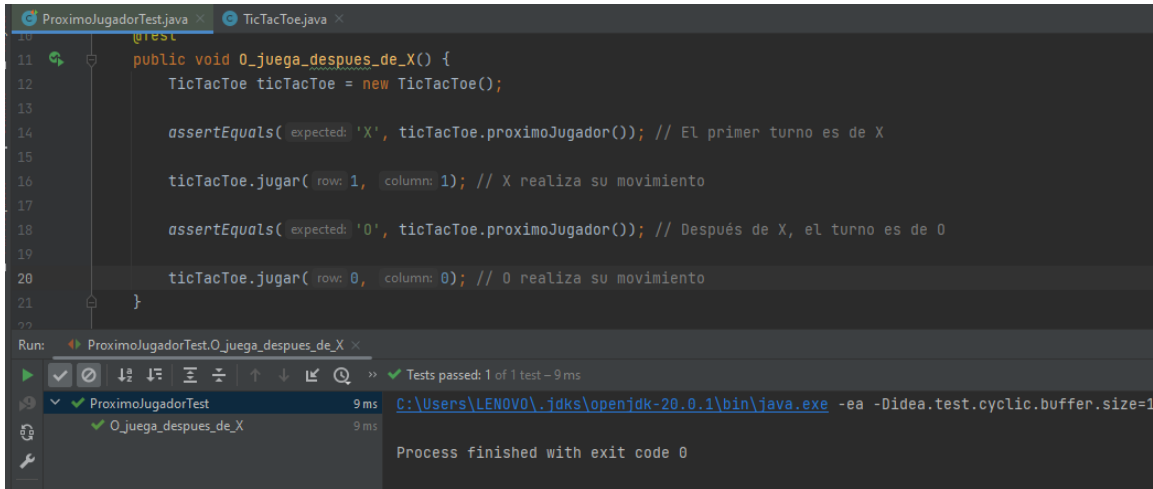
Process finished with exit code 0

Prueba: O juego justo después de X

Como hemos estado haciendo, primero se implemento la prueba, por ende, la prueba falló ya que no se implemento el método `jugar`

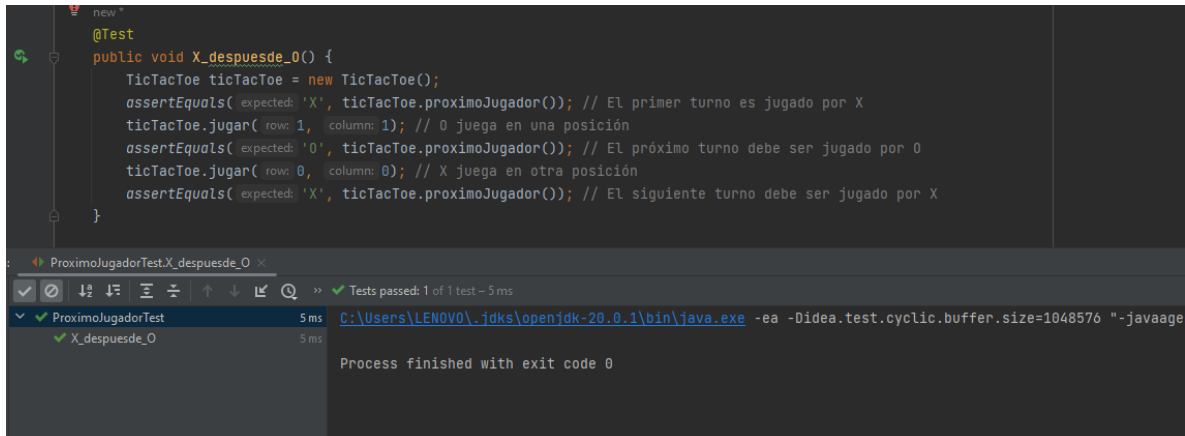


El método jugar ya existe, pero falta las condiciones necesarias para que pueda pasar la prueba



Prueba: X juega justo después de O

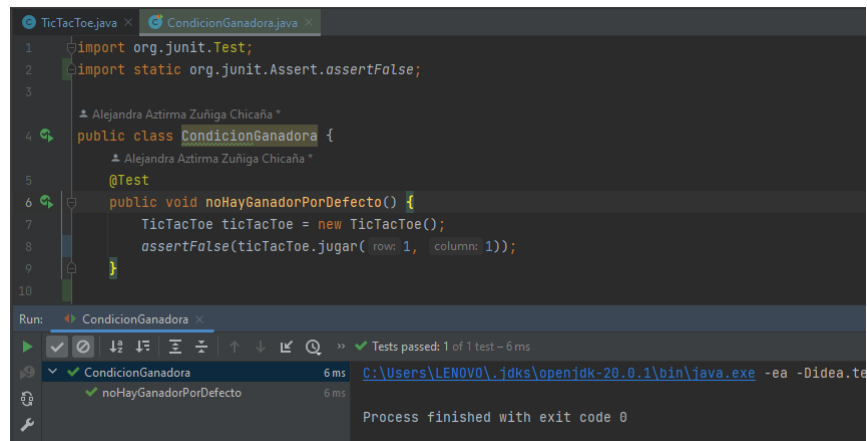
No se implementó nada al método ProximoJugador y aun así la prueba ha pasado, como se sugirió se desechará esta prueba, pero de todos modos aquí se deja una captura de la prueba.



Requisito 3: agregar condiciones ganadoras

Prueba: por defecto no hay ganador

En esta prueba, se crea una instancia de TicTacToe y se llama al método jugar() con una posición válida en el tablero (en este caso, la posición (1, 1)). Luego se verifica que el resultado del método jugar() sea false, lo cual indica que no hay ganador.



```
1 import org.junit.Test;
2 import static org.junit.Assert.assertFalse;
3
4 public class CondicionGanadora {
5     @Test
6     public void noHayGanadorPorDefecto() {
7         TicTacToe ticTacToe = new TicTacToe();
8         assertFalse(ticTacToe.jugar( row: 1, column: 1));
9     }
10 }
```

Run: CondicionGanadora

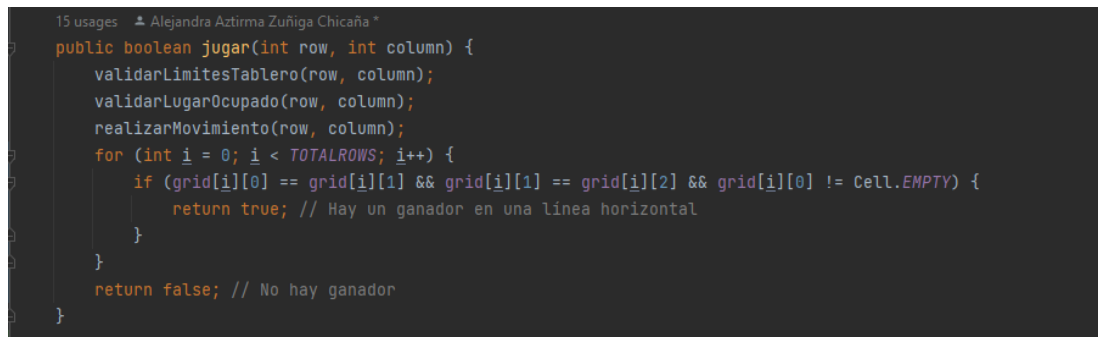
Test	Duration	Exit Code
CondicionGanadora	6 ms	0
noHayGanadorPorDefecto	6 ms	0

Tests passed: 1 of 1 test - 6 ms

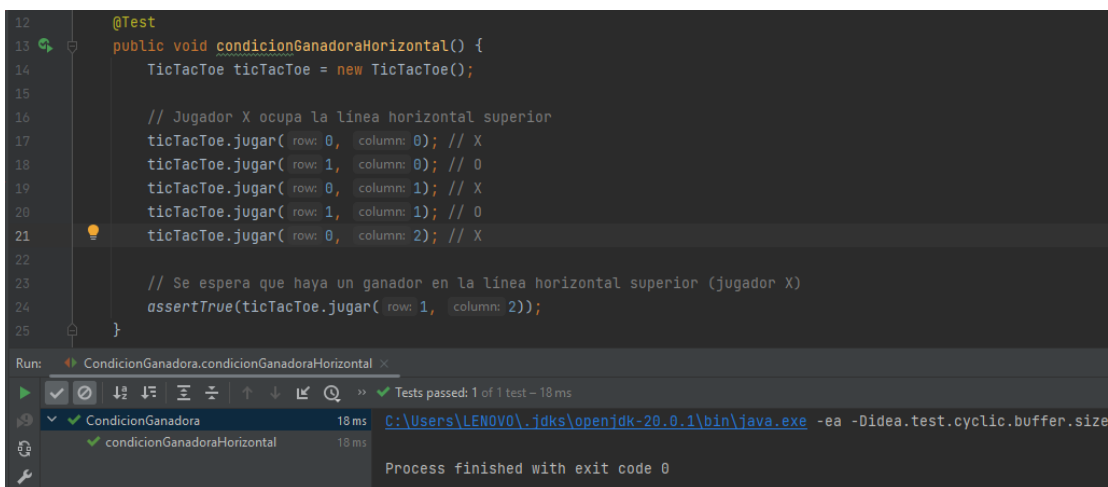
Process finished with exit code 0

Prueba – condición ganadora I

Se realizó la implementación del método jugar y de esta manera se procedió a hacer la prueba respectiva para verificar esta condición.



```
15 usages Alejandro Azirma Zuñiga Chicaña *
public boolean jugar(int row, int column) {
    validarLimitesTablero(row, column);
    validarLugarOcupado(row, column);
    realizarMovimiento(row, column);
    for (int i = 0; i < TOTALROWS; i++) {
        if (grid[i][0] == grid[i][1] && grid[i][1] == grid[i][2] && grid[i][0] != Cell.EMPTY) {
            return true; // Hay un ganador en una línea horizontal
        }
    }
    return false; // No hay ganador
}
```



```
12 @Test
13 public void condicionGanadoraHorizontal() {
14     TicTacToe ticTacToe = new TicTacToe();
15
16     // Jugador X ocupa la línea horizontal superior
17     ticTacToe.jugar( row: 0, column: 0); // X
18     ticTacToe.jugar( row: 1, column: 0); // 0
19     ticTacToe.jugar( row: 0, column: 1); // X
20     ticTacToe.jugar( row: 1, column: 1); // 0
21     ticTacToe.jugar( row: 0, column: 2); // X
22
23     // Se espera que haya un ganador en la línea horizontal superior (jugador X)
24     assertTrue(ticTacToe.jugar( row: 1, column: 2));
25 }
```

Run: CondicionGanadora.condicionGanadoraHorizontal

Test	Duration	Exit Code
CondicionGanadora	18 ms	0
condicionGanadoraHorizontal	18 ms	0

Tests passed: 1 of 1 test - 18 ms

Process finished with exit code 0

Nuevamente se procedió a refactorizar el método Jugar, para poder realizar los siguientes casos sobre las condiciones ganadoras.

```

15 usages  Alejandra Aztirma Zuñiga Chicaña *
public boolean jugar(int row, int column) {
    validarLimitesTablero(row, column);
    validarLugarOcupado(row, column);
    realizarMovimiento(row, column);
    return hayGanador();
}

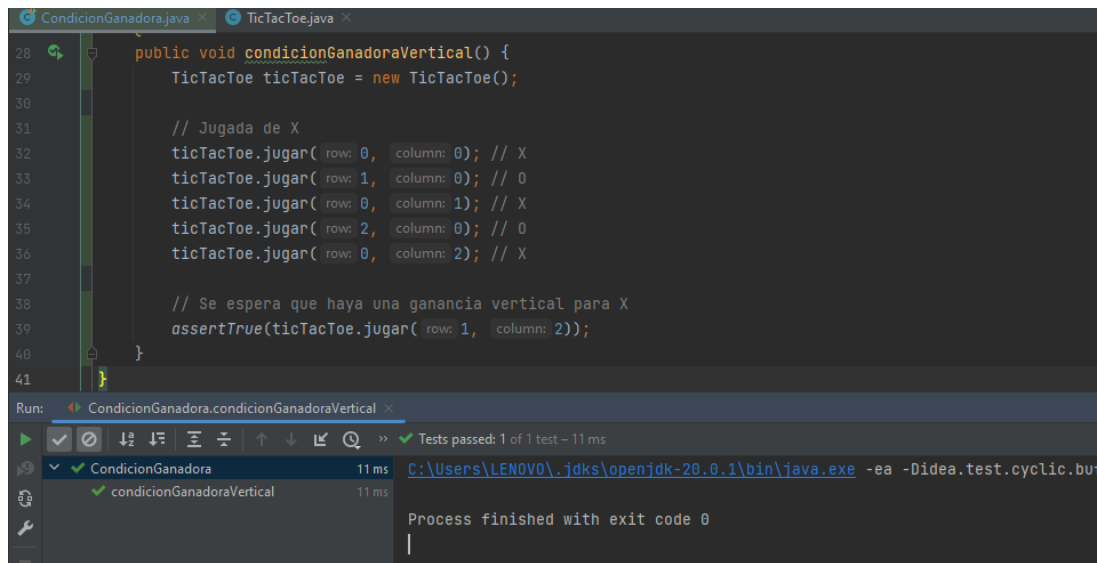
1 usage new *
private boolean hayGanador() {
    for (int i = 0; i < TOTALROWS; i++) {
        if (grid[i][0] == grid[i][1] && grid[i][1] == grid[i][2] && grid[i][0] != Cell.EMPTY) {
            return true; // Hay un ganador en una línea horizontal
        }
    }
    return false; // No hay ganador
}

```

En esta refactorización, se creó un método separado llamado hayGanador() que se encarga de verificar si hay un ganador en las líneas horizontales. El método jugar() ahora simplemente llama a hayGanador() y retorna el resultado.

Prueba – condición ganadora II

Esta implementación es similar a la anterior



The screenshot shows an IDE with two tabs: 'CondicionGanadora.java' and 'TicTacToe.java'. The 'CondicionGanadora.java' tab is active, showing the following code:

```

28 public void condicionGanadoraVertical() {
29     TicTacToe ticTacToe = new TicTacToe();
30
31     // Jugada de X
32     ticTacToe.jugar( row: 0, column: 0); // X
33     ticTacToe.jugar( row: 1, column: 0); // 0
34     ticTacToe.jugar( row: 0, column: 1); // X
35     ticTacToe.jugar( row: 2, column: 0); // 0
36     ticTacToe.jugar( row: 0, column: 2); // X
37
38     // Se espera que haya una ganancia vertical para X
39     assertTrue(ticTacToe.jugar( row: 1, column: 2));
40 }
41

```

Below the code, the 'Run' tab shows the execution results. The test 'CondicionGanadora.condicionGanadoraVertical' passed successfully. The output window shows 'Process finished with exit code 0'.

También se implementó el método hayGanador() para verificar la ganancia vertical:

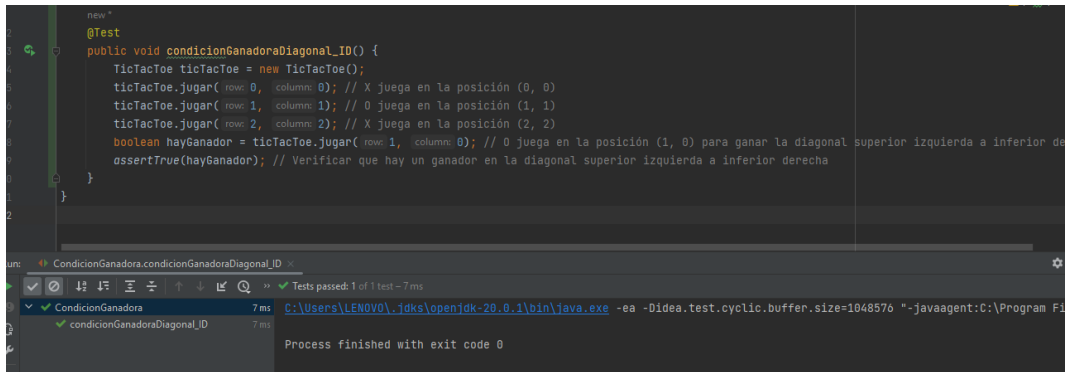
```

private boolean hayGanador() {
    for (int i = 0; i < TOTALROWS; i++) {
        if (grid[i][0] == grid[i][1] && grid[i][1] == grid[i][2] && grid[i][0] != Cell.EMPTY) {
            return true; // Hay un ganador en una línea horizontal
        }
    }
    // Verificación de ganancia vertical
    for (int j = 0; j < TOTALCOLUMNS; j++) {
        if (grid[0][j] == grid[1][j] && grid[1][j] == grid[2][j] && grid[0][j] != Cell.EMPTY) {
            return true; // Hay una ganancia vertical
        }
    }
    return false; // No hay ganador
}

```


Prueba – condición ganador III

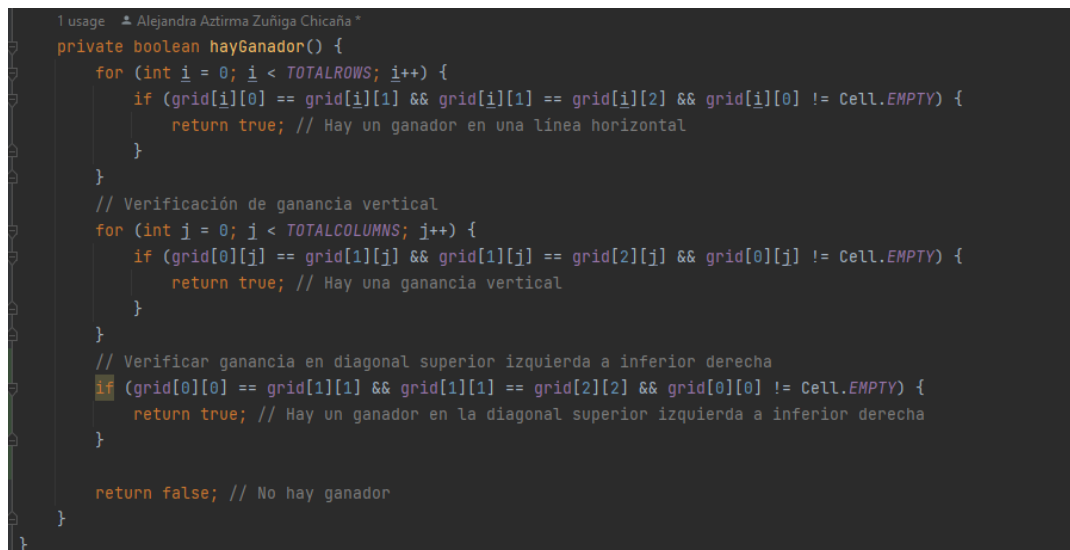
La prueba "ganarDiagonalSuperiorIzquierdaInferiorDerecha" verifica si el juego detecta correctamente una ganancia en la línea diagonal superior izquierda a inferior derecha.



```
new *
@Test
public void condicionGanadoraDiagonal_ID() {
    TicTacToe ticTacToe = new TicTacToe();
    ticTacToe.jugar( row: 0, column: 0); // X juega en la posición (0, 0)
    ticTacToe.jugar( row: 1, column: 1); // 0 juega en la posición (1, 1)
    ticTacToe.jugar( row: 2, column: 2); // X juega en la posición (2, 2)
    boolean hayGanador = ticTacToe.jugar( row: 1, column: 0); // 0 juega en la posición (1, 0) para ganar la diagonal superior izquierda a inferior derecha
    assertTrue(hayGanador); // Verificar que hay un ganador en la diagonal superior izquierda a inferior derecha
}

JUnit
ConditionGanadora.condicionGanadoraDiagonal_ID
7 ms
C:\Users\LENOVO\jdk\openjdk-20.0.1\bin\java.exe -ea -Didea.test.cyclic.buffer.size=1048576 "-javaagent:C:\Program F...
Process finished with exit code 0
```

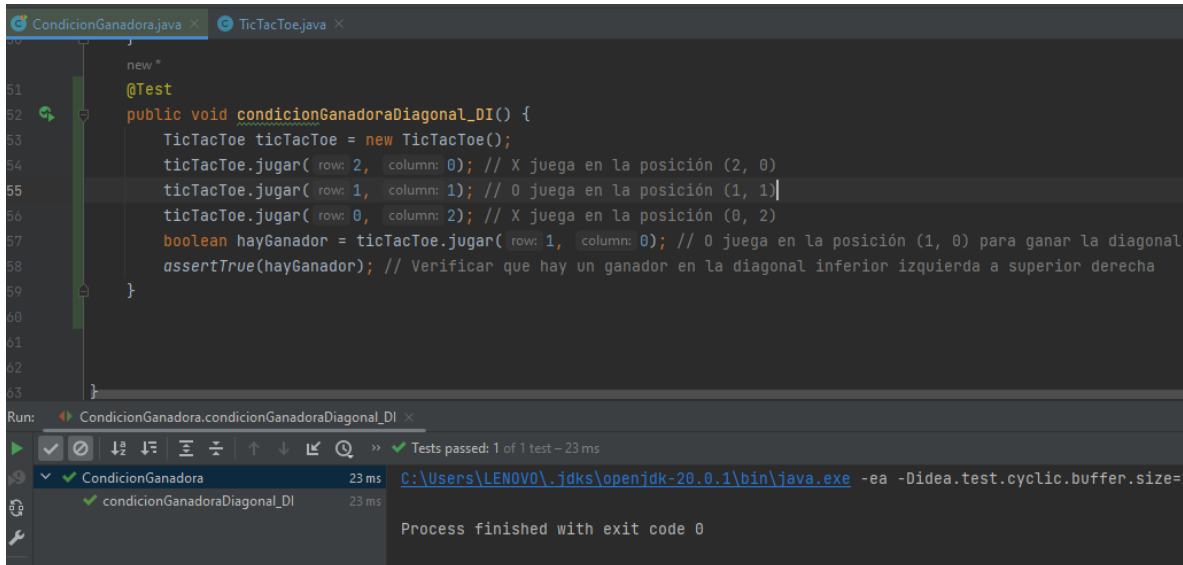
Por otro lado se tuvo que implementar nuevamente el método hayGanador(), se verifica tanto la ganancia en líneas horizontales, verticales y diagonales, asegurando que todas las condiciones ganadoras estén cubiertas.



```
1 usage  Alejandra Aztirma Zuñiga Chicaña *
private boolean hayGanador() {
    for (int i = 0; i < TOTALROWS; i++) {
        if (grid[i][0] == grid[i][1] && grid[i][1] == grid[i][2] && grid[i][0] != Cell.EMPTY) {
            return true; // Hay un ganador en una línea horizontal
        }
    }
    // Verificación de ganancia vertical
    for (int j = 0; j < TOTALCOLUMNS; j++) {
        if (grid[0][j] == grid[1][j] && grid[1][j] == grid[2][j] && grid[0][j] != Cell.EMPTY) {
            return true; // Hay una ganancia vertical
        }
    }
    // Verificar ganancia en diagonal superior izquierda a inferior derecha
    if (grid[0][0] == grid[1][1] && grid[1][1] == grid[2][2] && grid[0][0] != Cell.EMPTY) {
        return true; // Hay un ganador en la diagonal superior izquierda a inferior derecha
    }
    return false; // No hay ganador
}
```

Prueba – condición ganadora IV

Finalmente, existe la última condición ganadora posible para abordar: El jugador gana cuando se forma la diagonal desde la parte inferior izquierda hasta la parte superior derecha



```
new *
@Test
public void condicionGanadoraDiagonal_DI() {
    TicTacToe ticTacToe = new TicTacToe();
    ticTacToe.jugar( row: 2, column: 0); // X juega en la posición (2, 0)
    ticTacToe.jugar( row: 1, column: 1); // 0 juega en la posición (1, 1)
    ticTacToe.jugar( row: 0, column: 2); // X juega en la posición (0, 2)
    boolean hayGanador = ticTacToe.jugar( row: 1, column: 0); // 0 juega en la posición (1, 0) para ganar la diagonal
    assertTrue(hayGanador); // Verificar que hay un ganador en la diagonal inferior izquierda a superior derecha
}
```

Run: CondicionGanadora.condicionGanadoraDiagonal_DI

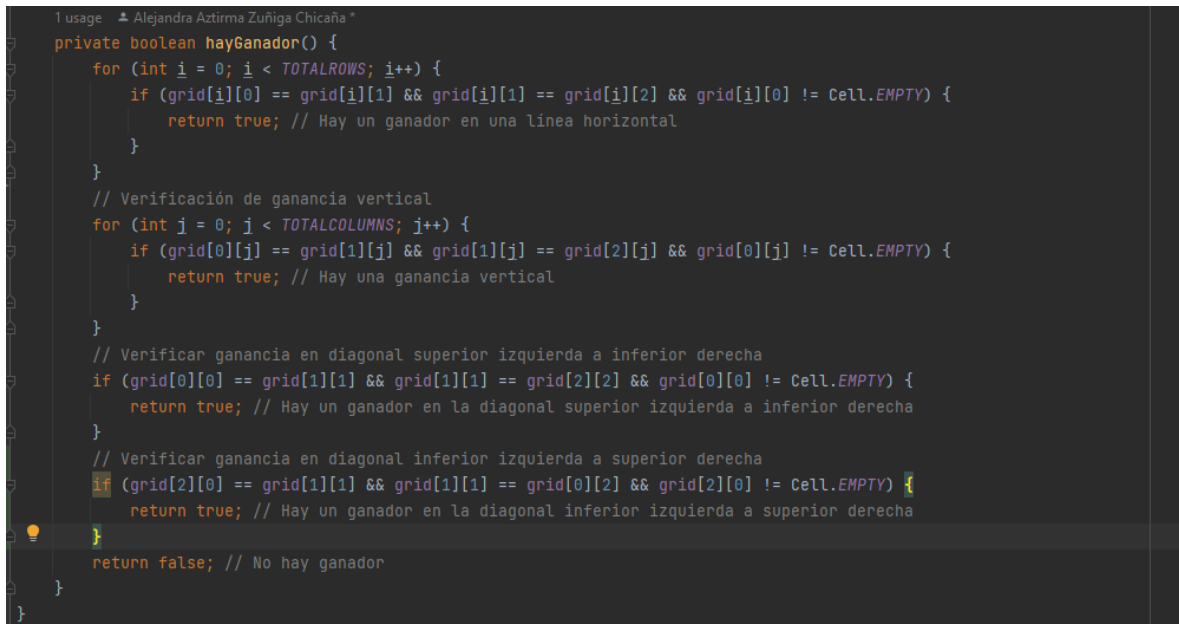
Tests passed: 1 of 1 test - 23 ms

CondicionGanadora 23 ms C:\Users\LENOVO\jdk\openjdk-20.0.1\bin\java.exe -ea -Didea.test.cyclic.buffer.size=

condicionGanadoraDiagonal_DI 23 ms

Process finished with exit code 0

Con esta implementación, se verifican todas las condiciones ganadoras posibles, incluyendo las líneas diagonales desde la parte inferior izquierda hasta la parte superior derecha.



```
1 usage  Alejandra Azirma Zuñiga Chicaña *
private boolean hayGanador() {
    for (int i = 0; i < TOTALROWS; i++) {
        if (grid[i][0] == grid[i][1] && grid[i][1] == grid[i][2] && grid[i][0] != Cell.EMPTY) {
            return true; // Hay un ganador en una línea horizontal
        }
    }
    // Verificación de ganancia vertical
    for (int j = 0; j < TOTALCOLUMNS; j++) {
        if (grid[0][j] == grid[1][j] && grid[1][j] == grid[2][j] && grid[0][j] != Cell.EMPTY) {
            return true; // Hay una ganancia vertical
        }
    }
    // Verificar ganancia en diagonal superior izquierda a inferior derecha
    if (grid[0][0] == grid[1][1] && grid[1][1] == grid[2][2] && grid[0][0] != Cell.EMPTY) {
        return true; // Hay un ganador en la diagonal superior izquierda a inferior derecha
    }
    // Verificar ganancia en diagonal inferior izquierda a superior derecha
    if (grid[2][0] == grid[1][1] && grid[1][1] == grid[0][2] && grid[2][0] != Cell.EMPTY) {
        return true; // Hay un ganador en la diagonal inferior izquierda a superior derecha
    }
    return false; // No hay ganador
}
```

Refactorización

```

1 usage  Alejandra Azirma Zuñiga Chicaña
private boolean hayGanador() {
    for (int i = 0; i < TOTALROWS; i++) {
        if (grid[i][0] == grid[i][1] && grid[i][1] == grid[i][2] && grid[i][0] != Cell.EMPTY) {
            return true; // Hay un ganador en una línea horizontal
        }
    }
    // Verificación de ganancia vertical
    for (int j = 0; j < TOTALCOLUMNS; j++) {
        if (grid[0][j] == grid[1][j] && grid[1][j] == grid[2][j] && grid[0][j] != Cell.EMPTY) {
            return true; // Hay una ganancia vertical
        }
    }
    // Verificar ganancia en diagonales
    if ((grid[0][0] == grid[1][1] && grid[1][1] == grid[2][2] && grid[0][0] != Cell.EMPTY) ||
        (grid[0][2] == grid[1][1] && grid[1][1] == grid[2][0] && grid[0][2] != Cell.EMPTY)) {
        return true; // Hay un ganador en una línea diagonal
    }
    return false; // No hay ganador
}
}

```

En esta refactorización, el bucle existente condiciones adicionales para verificar las dos diagonales. Esto reutiliza el bucle existente y evita la duplicación de código.

Requisito 4: condiciones de empate

Se creó por ultima la prueba para el caso de empate, en este caso la prueba no funciona debido a que no se implementó aún el método hayEmpate()

```

3 CondicionGanadora.java  TicTacToe.java
@Test
public void manejoEmpate() {
    TicTacToe ticTacToe = new TicTacToe();

    // Llenar todas las casillas del tablero
    ticTacToe.jugar( row: 0, column: 0);
    ticTacToe.jugar( row: 0, column: 1);
    ticTacToe.jugar( row: 0, column: 2);
    ticTacToe.jugar( row: 1, column: 0);
    ticTacToe.jugar( row: 1, column: 2);
    ticTacToe.jugar( row: 1, column: 1);
    ticTacToe.jugar( row: 2, column: 1);
    ticTacToe.jugar( row: 2, column: 0);
    ticTacToe.jugar( row: 2, column: 2);

    // Verificar que no haya ganador y sea un empate
    assertTrue(ticTacToe.hayEmpate());
}

```

Build: Build Output

Pregunta2: build failed At 16/05/2022 2 sec, 576 ms

CondicionGanadora.java src/test 1 error

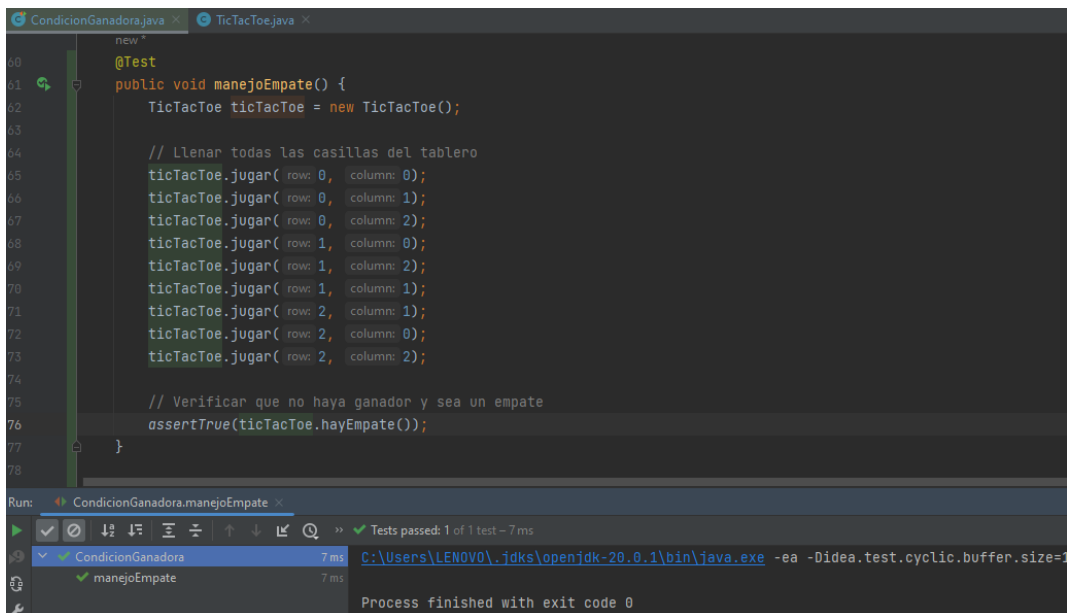
cannot find symbol method hayEmpate():?/

java: cannot find symbol
symbol: method hayEmpate()
location: variable ticTacToe of type TicTacToe

Se implementó el método hayEmpate para que así la prueba anteriormente mencionada pueda pasar.

```
1 usage new *
public boolean hayEmpate() {
    for (int row = 0; row < TOTALROWS; row++) {
        for (int column = 0; column < TOTALCOLUMNS; column++) {
            if (grid[row][column] == Cell.EMPTY) {
                return false; // Todavía hay casillas vacías, no es un empate
            }
        }
    }
    return true; // Todas las casillas están ocupadas, es un empate
}
```

Una vez implementado, la prueba paso.



The screenshot shows an IDE with two tabs: 'CondicionGanadora.java' and 'TicTacToe.java'. The 'CondicionGanadora.java' tab is active, showing a test method named 'manejoEmpate()' with the following code:

```
60 new *
61 @Test
62 public void manejoEmpate() {
63     TicTacToe ticTacToe = new TicTacToe();
64
65     // Llenar todas las casillas del tablero
66     ticTacToe.jugar( row: 0, column: 0);
67     ticTacToe.jugar( row: 0, column: 1);
68     ticTacToe.jugar( row: 0, column: 2);
69     ticTacToe.jugar( row: 1, column: 0);
70     ticTacToe.jugar( row: 1, column: 2);
71     ticTacToe.jugar( row: 1, column: 1);
72     ticTacToe.jugar( row: 2, column: 1);
73     ticTacToe.jugar( row: 2, column: 0);
74     ticTacToe.jugar( row: 2, column: 2);
75
76     // Verificar que no haya ganador y sea un empate
77     assertTrue(ticTacToe.hayEmpate());
78 }
```

Below the code editor, the 'Run' window shows the execution of 'CondicionGanadora.manejoEmpate'. The test passed, with the message 'Tests passed: 1 of 1 test - 7 ms'. The command line shows the execution of 'C:\Users\LENOVO\jdk\openjdk-20.0.1\bin\java.exe -ea -Didea.test.cyclic.buffer.size=1'. The process finished with exit code 0.

Por último, nos piden nuevamente un refactorización usando un método llamado esGanador() a pesar de que no esté al alcance la última prueba.

```

1 usage: = Arreglo Arreglo Longo Cincena
2
3 public boolean esGanador(int row, int column) {
4     Cell player = grid[row][column];
5     // Verificar linea horizontal
6     boolean horizontalWin = true;
7     for (int col = 0; col < TOTALCOLUMNS; col++) {
8         if (grid[row][col] != player) {
9             horizontalWin = false;
10            break;
11        }
12    }
13    if (horizontalWin) {
14        return true;
15    }
16    // Verificar linea vertical
17    boolean verticalWin = true;
18    for (int r = 0; r < TOTALROWS; r++) {
19        if (grid[r][column] != player) {
20            verticalWin = false;
21            break;
22        }
23    }
24    if (verticalWin) {
25        return true;
26    }
27    // Verificar linea vertical
28    boolean verticalWin = true;
29    for (int r = 0; r < TOTALROWS; r++) {
30        if (grid[r][column] != player) {
31            verticalWin = false;
32            break;
33        }
34    }
35    if (verticalWin) {
36        return true;
37    }
38    // Verificar diagonales
39    if (row == column || row + column == TOTALROWS - 1) {
40        boolean diagonalWin = true;
41        for (int i = 0; i < TOTALROWS; i++) {
42            if (grid[i][i] != player && grid[i][TOTALROWS - 1 - i] != player) {
43                diagonalWin = false;
44                break;
45            }
46        }
47        if (diagonalWin) {
48            return true;
49        }
50    }
51    return false;
52 }
53
54 1 usage: = Arreglo Arreglo Longo Cincena
55
56 public boolean hayEmpate() {
57     for (int row = 0; row < TOTALROWS; row++) {
58         for (int col = 0; col < TOTALCOLUMNS; col++) {
59             if (grid[row][col] == Cell.EMPTY || esGanador(row, col)) {
60                 return false; // Todavía hay casillas vacías o hay un ganador, no hay empate
61             }
62         }
63     }
64     return true; // Todas las casillas están ocupadas y no hay ganador, hay empate
65 }
66
67 1 usage: new *
68
69 private boolean hayGanador() {
70     return esGanador(int row, int column);
71 }
72
73 }

```

Con esta refactorización, se mejora la claridad y legibilidad del código al separar las responsabilidades de verificación de empate y ganador en métodos individuales. Esto facilita el mantenimiento y comprensión del código,