

Pregunta2

Luego de leer la pregunta 2 completa decidí crear una plantilla para empezar las pruebas, es decir ,prepare una clase Board (tablero) y una clase Piece (casillas del tablero).En otras palabras hare el TicTacToe a mi manera. Siguiendo los requisitos.

Clase Tablero

```
public class Board {  
    2 usages  
    private final int tamaño = 3;  
    1 usage  
    private Piece[][] board;  
  
    1 usage new *  
    public Board(){  
        this.board = new Piece[tamaño][tamaño];  
    }  
}
```

Clase Piece

```
1 usage new *  
public class Piece {  
    1 usage  
    private int x;  
    1 usage  
    private int y;  
  
    no usages new *  
    public Piece(int x,int y){  
        this.x=x;  
        this.y=y;  
    }  
}
```

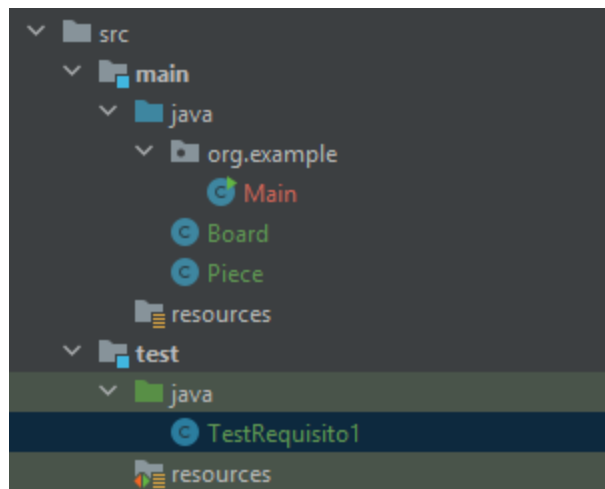
Requisito 1 : Se puede colocar una pieza en cualquier espacio vacío de un tablero de 3×3.

```
no usages new *  
public class TestRequisito1 {  
    |  
}  
}
```

Prueba 1 : Cuando una pieza se coloca en cualquier lugar **fuera del eje x**, se lanza → RuntimeException

Prueba 2 : Cuando una pieza se coloca en cualquier lugar **fuera del eje y**, se lanza → RuntimeException

Prueba 3 : Cuando una pieza se coloca en un espacio ocupado, se lanza → RuntimeException



WhenPiecelsoffAxisXThenReturnException(): prueba que verifica si una pieza lanzará una excepción RuntimeException si se coloca fuera del eje X.

```
@Test
public void WhenPieceIsOffAxisXThenReturnException()
{

}
}
```

WhenPieceIsOffAxisYThenReturnException(): prueba que verifica si una pieza lanzará una excepción RuntimeException si se coloca fuera del eje Y.

```
@Test
public void WhenPieceIsOffAxisYThenReturnException()
{

}
}
```

WhenPieceIsOnOccupiedSpaceThenReturnsException(): prueba que verifica si una pieza lanzará una excepción RuntimeException si se coloca en un espacio ocupado.

```
@Test
public void WhenPieceIsOnOccupiedSpaceThenReturnsException()
{

}
}
```

Prueba: límites del tablero I

- 1) Crear el método jugar
- 2) argumento (<1 o >3) o (<0 o >2) del eje X → **RuntimeException**

EJECUTANDO las pruebas RGR:

1 ejecución) Se crea la prueba y debería arrojar rojo porque el método jugar no existe

```

    public void WhenPieceIsOffAxisXThenReturnException()
    {
        Board tablero = new Board();
        tablero.jugar(0,2);
    }

```

```

C:\Users\Computer\Desktop\CC-3S2_repo\CC-3S2\PARCI
    tablero.jugar(0,2);
           ^
symbol:   method jugar(int,int)
location: variable tablero of type Board

```

2 ejecución) Debería fallar porque no se lanza **RuntimeException**

Definimos el método Jugar

```

public void jugar(int x, int y) {
    if( x < 0 || x >= tamaño-1)
    {
        System.out.println("Fuera de rango");
    }
}

```

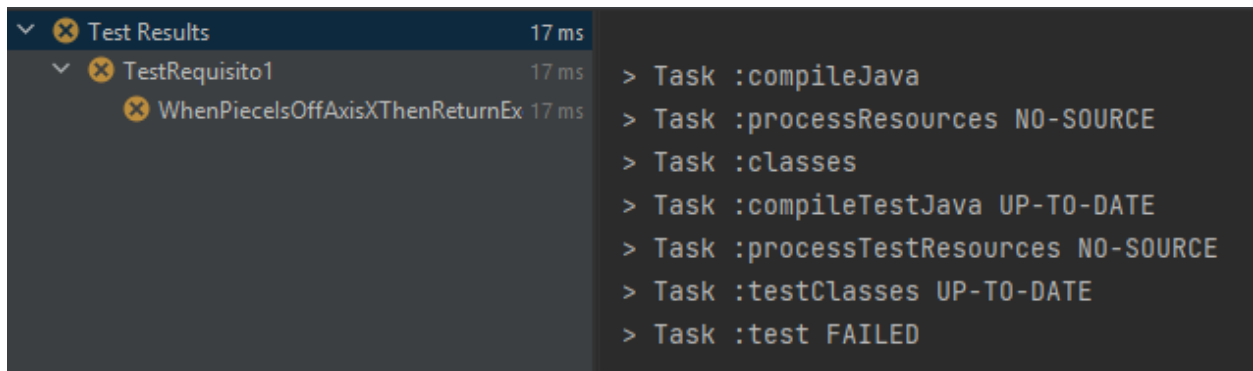
Y completamos el método de prueba con el **assert**

```

@Test
public void WhenPieceIsOffAxisXThenReturnException()
{
    Board tablero = new Board();
    assertThrows(RuntimeException.class, ()->tablero.jugar( x: -1, y: 2));
}

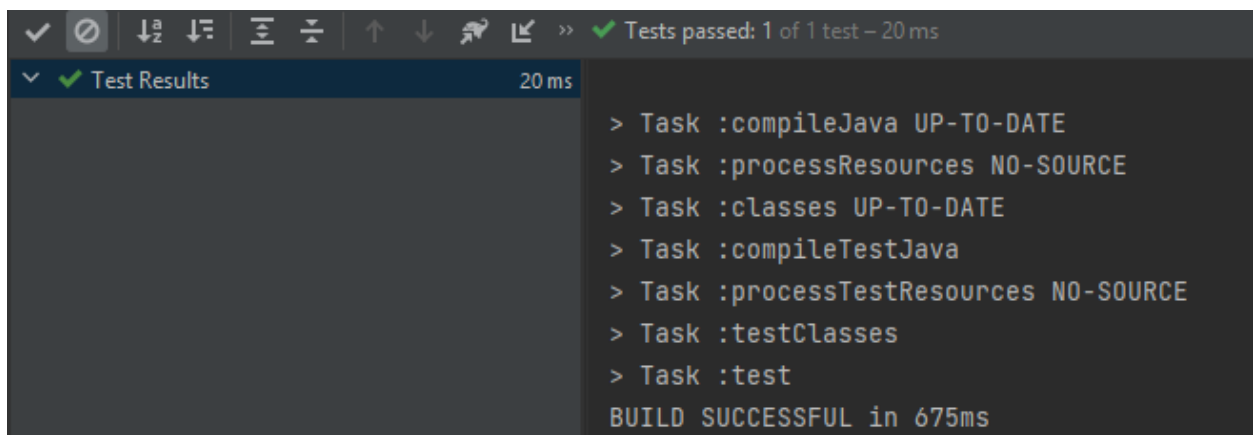
```

Ejecutamos y falla



3 ejecución) Debería tener éxito - Modificamos el método jugar

```
public void jugar(int x, int y) {  
    if( x < 0 || x >= tamaño-1)  
    {  
        throw new RuntimeException("Coordenada X fuera de rango");  
    }  
}
```



Prueba: límites del tablero II

- 1) Modificar el método jugar
- 2) argumento (<1 o >3) o (<0 o >2) del eje Y → **RuntimeException**

```

@Test
public void WhenPieceIsOffAxisYThenReturnException()
{
    Board tablero = new Board();
    assertThrows(RuntimeException.class, ()->tablero.jugar( x: 0, y: 3));
}

```

```

public void WhenPieceIsOffAxisYThenReturnException()
{
    Board tablero = new Board();
    assertThrows(RuntimeException.class, ()->tablero.jugar( x: 0, y: 3));
}

```

Tests failed: 1 of 1 test – 15 ms

Test Results	15 ms
TestRequisito1	15 ms
WhenPieclsOffAxisYThenReturnEx	15 ms

```

> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :compileTestJava
> Task :processTestResources NO-SOURCE
> Task :testClasses
> Task :test FAILED

```

Vemos que falla es decir necesitamos hacer la implementación:

```

public void jugar(int x, int y) {
    if( x < 0 || x >= tamaño-1)
    {
        throw new RuntimeException("Coordenada X fuera de rango");
    }

    if( y < 0 || y >= tamaño-1)
    {
        throw new RuntimeException("Coordenada Y fuera de rango");
    }
}

```

Ejecutamos y tiene éxito

```
@Test
public void WhenPieceIsOffAxisYThenReturnException()
{
    Board tablero = new Board();
    assertThrows(RuntimeException.class, ()->tablero.jugar(x: 0, y: 3));
}
```

Run: TestRequisito1.WhenPiecelOffAxisYThenReturnException X

✓ Tests passed: 1 of 1 test – 15 ms

✓ Test Results 15 ms

- > Task :compileJava
- > Task :processResources NO-SOURCE
- > Task :classes
- > Task :compileTestJava UP-TO-DATE
- > Task :processTestResources NO-SOURCE
- > Task :testClasses UP-TO-DATE
- > Task :test

BUILD SUCCESSFUL in 657ms

Prueba - lugar ocupado

- 1) Modificar el método jugar
- 2) Si la casilla esta en un espacio ocupado → **RuntimeException**

```
@Test
public void WhenPieceIsOnOccupiedSpaceThenReturnsException()
{
}
}
```

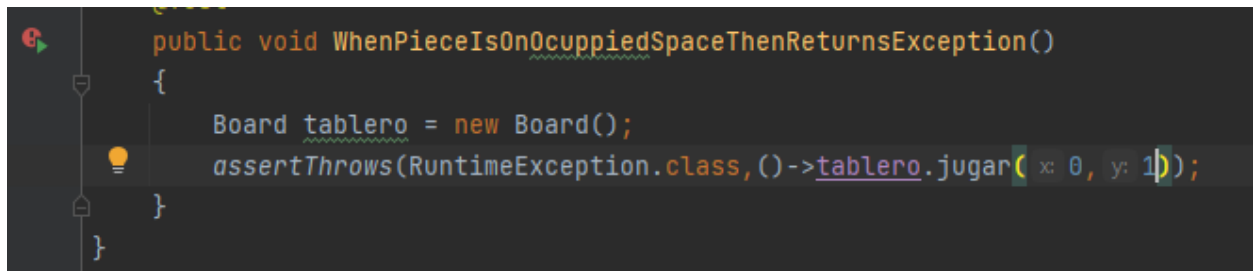
Escogemos cualquier casilla dentro del limite

```

@Test
public void WhenPieceIsOnOccupiedSpaceThenReturnsException()
{
    Board tablero = new Board();
    assertThrows(RuntimeException.class, ()->tablero.jugar(x: 0, y: 1));
}

```

Ejecutamos y nos falla , es decir requiere una **implementación**



```

public void WhenPieceIsOnOccupiedSpaceThenReturnsException()
{
    Board tablero = new Board();
    assertThrows(RuntimeException.class, ()->tablero.jugar(x: 0, y: 1));
}

```

Implementación:

almacenar la ubicación de las piezas colocadas en un arreglo.

vacío : "\0" — ocupado: "X"

Creamos una matriz "estadoCasilla" que almacena el char si esta ocupado o no. En el constructor lo inicializa con vacío.


```

public class Board {
    8 usages
    private final int tamaño = 3;
    1 usage
    private Piece[][] board;
    3 usages
    private char[][] estadoCasilla;
    3 usages new*
    public Board(){
        this.board = new Piece[tamaño][tamaño];
        this.estadoCasilla= new char[tamaño][tamaño];
        for(int i = 0;i < tamaño;i++)
        {
            for(int j= 0;j < tamaño;j++)
            {
                estadoCasilla[i][j]='\0';
            }
        }
    }
}

```

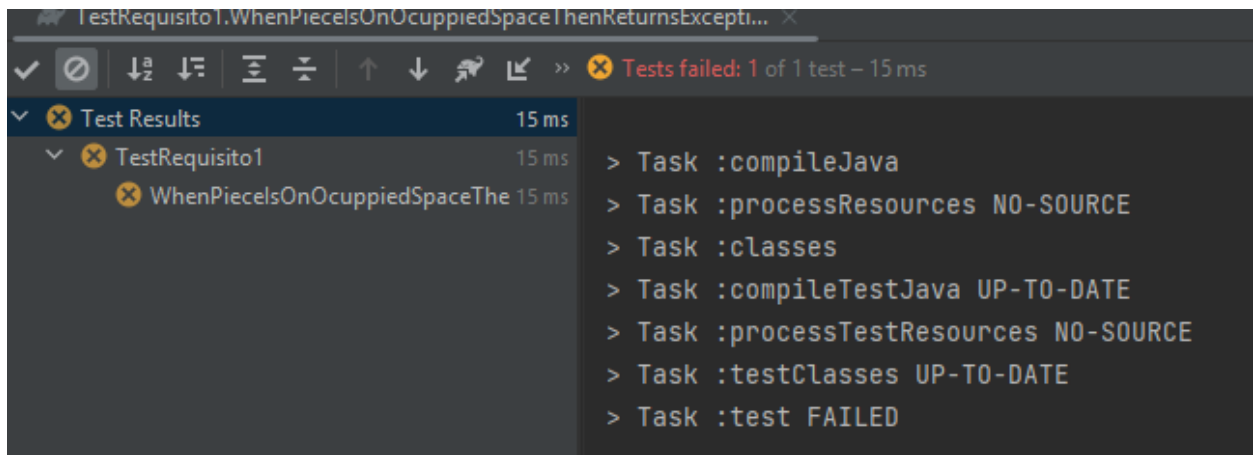
Cambiamos el test según la implementación :

```

@Test
public void WhenPieceIsOnOccupiedSpaceThenReturnsException()
{
    Board tablero = new Board();
    char[][] casillas =tablero.getEstadoCasilla();
    casillas[0][1]='X';
    assertEquals(RuntimeException.class,()->tablero.jugar(x: 0, y: 1));
}
}

```

Ejecutamos y falla



Necesita otra implementación pero en el método jugar .Agregamos el **if** donde verifica Coordenada ocupada o no.

```
3 usages new *
public void jugar(int x, int y) {
    if( x < 0 || x >= tamaño-1)
    {
        throw new RuntimeException("Coordenada X fuera de rango");
    }
    if( y < 0 || y >= tamaño-1)
    {
        throw new RuntimeException("Coordenada Y fuera de rango");
    }
    if(estadoCasilla[x][y]!='X'){
        throw new RuntimeException("Coordenada Ocupada");
    }
}
}
```

Ejecutamos el test y pasa

```

21 public void WhenPieceIsOnOccupiedSpaceThenReturnsException()
22 {
23     Board tablero = new Board();
24     char[][] casillas = tablero.getEstadoCasilla();
25     casillas[0][1] = 'X';
26     assertEquals(RuntimeException.class, () -> tablero.jugar(x: 0, y: 1));
27 }
28 }
29

```

```

> Task :compileJava
> Task :processResources NO-SOURCE
> Task :classes
> Task :compileTestJava UP-TO-DATE
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE
> Task :test
BUILD SUCCESSFUL in 655ms

```

Refactorización

En la refactorización podríamos hacer bastantes cosas. Una de estas si nos damos cuenta en el propio constructor tenemos un for de for. Esto se puede mover a un método mas separado con un nombre que se entienda que se esta haciendo.

```

public Board(){
    this.board = new Piece[tamano][tamano];
    this.estadoCasilla = new char[tamano][tamano];
    for(int i = 0; i < tamano; i++)
    {
        for(int j = 0; j < tamano; j++)
        {
            estadoCasilla[i][j] = '\0';
        }
    }
}

```

Lo llamaremos **rellenarCasillasAVacias** porque es lo que hace:

```
public Board(){
    this.board = new Piece[tamano][tamano];
    this.estadoCasilla= new char[tamano][tamano];
    rellenarCasillasAVacias();
}

1 usage new *
private void rellenarCasillasAVacias(){
    for(int i = 0; i < tamano; i++)
    {
        for(int j= 0; j < tamano; j++)
        {
            estadoCasilla[i][j]='\0';
        }
    }
}
```

Luego en cada uno de los if del método Jugar podríamos extraerlos a una función separada que tenga el nombre de lo que se esta evaluado. Entonces de lo que tenemos:

```
public void jugar(int x, int y) {
    if( x < 0 || x >= tamano-1 )
    {
        throw new RuntimeException("Coordenada X fuera de rango");
    }
    if( y < 0 || y >= tamano-1)
    {
        throw new RuntimeException("Coordenada Y fuera de rango");
    }
    if(estadoCasilla[x][y]=='X'){
        throw new RuntimeException("Coordenada Ocupada");
    }
}
```

Pasaremos a lo siguiente:

```

public void jugar(int x, int y) {

    if( CoordenadasXnoValida(x,y) )
    {
        throw new RuntimeException("Coordenada X fuera de rango");
    }

    if( CoordenadasYnoValida(x,y) )
    {
        throw new RuntimeException("Coordenada Y fuera de rango");
    }

    if(CasillaOcupada(x,y)){
        throw new RuntimeException("Coordenada Ocupada");
    }

}

1 usage new *
private boolean CoordenadasXnoValida(int x,int y){
    return x < 0 || x >= tamaño-1;
}

1 usage new *
private boolean CoordenadasYnoValida(int x,int y){
    return y < 0 || y >= tamaño-1;
}

1 usage new *
private boolean CasillaOcupada(int x,int y){
    return estadoCasilla[x][y]=='X';
}

```

Esos 3 if podrían estar en uno solo. Pero ya no indicaría el mensaje exacto de si el **RunTimeException** fue por X e Y o ocupada.

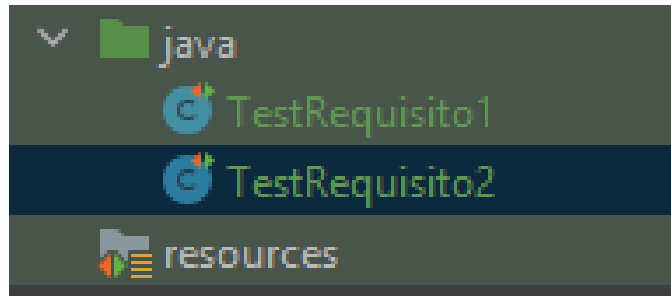
Requisito 2: **agregar soporte para dos jugadores**

Problema: que jugador continua

```

new *
public class TestRequisito2 {
    ...
}

```



De manera análoga vamos a hacer nuestras 3 pruebas

- El primer turno lo debe jugar el jugador X.

```
@Test
public void WhenXPlayFirstThenReturnsX(){
    |
}
```

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

```
@Test
public void WhenLastTurnWasXThenReturnsO(){
    |
}
```

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

```
@Test
public void WhenLastTurnWasOThenReturnsX(){
    |
}
```

Prueba – X juega primero

- 1) Crear el método ProximoJugador → retorna X

De lo anterior vamos refactorizando el test también :

```
2 usages
Board tablero;
new *
@BeforeEach
public void IniciarTablero() {
    tablero = new Board();
}
```

EJECUTANDO las pruebas RGR:

1 ejecución) Se crea la prueba y debería arrojar rojo porque el método ProximoJugador no existe.

```
@Test
public void WhenXPlayFirstThenReturnsX() {
    char jugadorActual='X';
    assertEquals('X', tablero.proximoJugador(jugadorActual));
}
```

TestRequisito2.WhenXPlayFirstThenRetur 419 ms	C:\Users\Computer\Desktop\CC-3S2_repo\CC-3S2\PARCIAL\Pregunta 2
:compileTestJava 1 error 76 ms	assertEquals('X', <u>tablero.proximoJugador(jugadorActual)</u>)
TestRequisito2.java src/test/java 1 error	^
cannot find symbol method proximoJuga	symbol: method proximoJugador(char)
	location: variable tablero of type Board

2 ejecución) Debería fallar porque no se lanza X

```
public char proximoJugador(char jugadorActual) {
    return ' ';
}
```

Ejecutamos y falla → necesita una implementación

```

@Test
public void WhenXPlayFirstThenReturnsX(){
    char jugadorActual='X';
    assertEquals( expected: 'X',tablero.proximoJugador(jugadorActual));
}

```

Run: TestRequisito2.WhenXPlayFirstThenReturnsX

Tests failed: 1 of 1 test – 17 ms

Test Results 17 ms

- TestRequisito2 17 ms
 - WhenXPlayFirstThenReturnsX() 17 ms

> Task :compileJava UP-TO-DATE
 > Task :processResources NO-SOURCE
 > Task :classes UP-TO-DATE
 > Task :compileTestJava
 > Task :processTestResources NO-SOURCE
 > Task :testClasses
 > Task :test FAILED

3 ejecución) Debería tener éxito

```

public char proximoJugador(char jugadorActual) {
    return 'X';
}

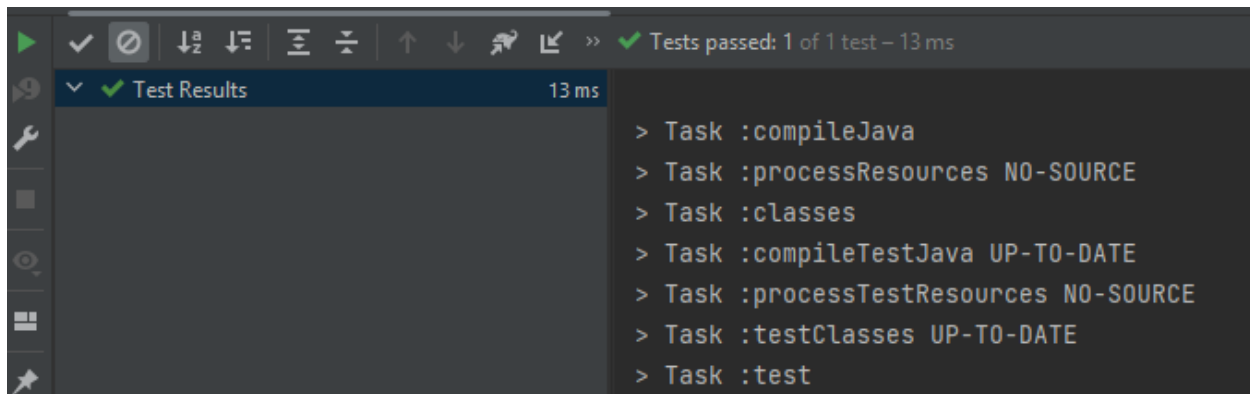
```

Luego de esta implementación , la prueba tiene éxito

```

@Test
public void WhenXPlayFirstThenReturnsX(){
    char jugadorActual='X';
    assertEquals( expected: 'X',tablero.proximoJugador(jugadorActual));
}

```

O juego justo después de X

Se necesita implementar una nueva variable que almacene el ultimo jugador que hizo una jugada. Además de su setter

```
public class Board {  
    8 usages  
    private final int tamaño = 3;  
    1 usage  
    private Piece[][] board;  
    4 usages  
    private char[][] estadoCasilla;  
    no usages  
    private char Ultimojugador='X';  
    4 usages new *  
    public Board(){  
        this.board = new Piece[tamaño][tamaño];  
        this.estadoCasilla= new char[tamaño][tamaño];  
        rellenarCasillasAVacias();  
    }  
}
```

```
public void setUltimojugador(char a) {  
    this.Ultimojugador=a;  
}
```

1) Modificar el método ProximoJugador : Hacer que se cambien de turno luego de la jugada.

Para que nuestro testAnteriorNoFalle necesitamos agregarle que nos de el próximo jugador Cuando ah acabado la jugada. como boolean

```
2 usages new *
public char proximoJugador(boolean AcaboLaJugada) {
    if(!AcaboLaJugada) return Ultimojugador;

    if (Ultimojugador == 'X') {
        return 'O';
    } else {
        return 'X';
    }
}
```

Luego las 2 pruebas salen exitosas con al implementacion .

```
@Test
public void WhenXPlayFirstThenReturnsX(){
    assertEquals( expected: 'X',tablero.proximoJugador( AcaboLaJugada: false));
}

new *
@Test
public void WhenLastTurnWasXThenReturns0(){
    tablero.setUltimojugador('X');
    assertEquals( expected: 'O',tablero.proximoJugador( AcaboLaJugada: true));
}
```

Prueba: X juega justo después de O

Esta prueba debe pasar sin escribir algún código de implementación. Por lo tanto se desecha.

Ejecutamos y efectivamente pasa ,por lo tanto se debe desechar

```

@Test
public void WhenLastTurnWas0ThenReturnsX(){
    tablero.setUltimoJugador('0');
    assertEquals( expected: 'X', tablero.proximoJugador( AcaboLaJugada: true));
}

```

Requisito 3: agregar condiciones ganadoras

Creamos nuestra clase para el requisito 3

```

public class TestRequisito3 {
    5 usages
    Board tablero;
    @BeforeEach
    public void IniciarTablero() { tablero = new Board(); }
}

```

Luego de morir en mi doble examen que tuve, en este punto ya no tenia noción de la realidad. Así que vi necesario usar una variable enum para asignar el resultado de la victoria. es decir que la clase jugar por cada jugada va verificar si se gano o no ,pero inicializando un atributo mas que almacene el resultado tipo enum.

```

public enum Resultado{GANA_0,GANA_X, NO_GANADOR}
no usages
private Resultado currentGameState;

```

Prueba: por defecto no hay ganador

```

public Resultado EstablecerGanador() {
    return Resultado.NO_GANADOR;
}

```

```

@Test
public void testHayGanadorDefault() {
    Board tablero = new Board();
    tablero.jugar(x: 0, y: 0);
    assertEquals(Board.Resultado.NO_GANADOR, tablero.currentGameState);
}

```

Prueba – condición ganadora I

Prueba no pasa por lo tanto necesitamos una **implementación**

```

@Test
public void WhenAllHorizontalLineisOccupiedThenPlayerWin() {
    tablero.jugar(x: 0, y: 0); // juega X
    tablero.jugar(x: 1, y: 0); // juega Y
    tablero.jugar(x: 0, y: 1); // juega X
    tablero.jugar(x: 1, y: 1); // juega Y
    tablero.jugar(x: 0, y: 2); // juega X
    assertEquals(Board.Resultado.GANA_X, tablero.currentGameState);
}

```

En este momento nos damos cuenta que también era necesario actualizar las variables del tablero cada que se hacer una jugada.

Y para no agregar una variable mas usaremos el char[][] estadoCasilla ;

que nos indicaba vacío: '\0' y ocupado: 'X' , ahora nos va indicar

vacío: '\0' y ocupado casilla X: 'X' ocupado casilla O: 'O'

IMPLEMENTANDO

Cambiamos de turno siempre y cuando no haiga ganador

```

public void jugar(int x, int y) {
    if( CoordinadasXnoValida(x,y) )
    {
        throw new RuntimeException("Coordinada X fuera de rango");
    }
    if( CoordinadasYnoValida(x,y))
    {
        throw new RuntimeException("Coordinada Y fuera de rango");
    }
    if(CasillaOcupada(x,y)){
        throw new RuntimeException("Coordinada Ocupada");
    }
    estadoCasilla[x][y]=Ultimojugador;
    currentGameState = EstablecerGanador();
    if(currentGameState==Resultado.NO_GANADOR)
    {
        Ultimojugador = proximoJugador( AcaboLaJugada: true);
    }
}
}

```

Recorriendo las filas establecemos un ganador si lo hay

```

1 usage
public Resultado EstablecerGanador() {
    //Recorremos las Filas
    for(int i=0;i<tamano;i++)
    {
        if(estadoCasilla[i][0]==Ultimojugador
            && estadoCasilla[i][1]==Ultimojugador
            && estadoCasilla[i][2]==Ultimojugador
        ){
            return (Ultimojugador=='X')?Resultado.GANA_X:Resultado.GANA_O;
        }
    }
    return Resultado.NO_GANADOR;
}

```

Pasando las pruebas

```
@Test
public void WhenAllHorizontalLineisOccupiedThenPlayerWin() {
    tablero.jugar(x: 0, y: 0); // juega X
    tablero.jugar(x: 1, y: 0); // juega Y
    tablero.jugar(x: 0, y: 1); // juega X
    tablero.jugar(x: 1, y: 1); // juega Y
    tablero.jugar(x: 0, y: 2); // juega X
    assertEquals(Board.Resultado.GANA_X, tablero.currentGameState);
}
```

Tests passed: 1 of 1 test – 13 ms

Test Results 13 ms

- > Task :compileJava
- > Task :processResources NO-SOURCE
- > Task :classes
- > Task :compileTestJava
- > Task :processTestResources NO-SOURCE
- > Task :testClasses
- > Task :test

BUILD SUCCESSFUL in 723ms

Prueba – condición ganadora II

Análogo pero para una línea vertical es decir ahora recorreremos cada columna

Pasamos la prueba falla porque falta implementar

```
@Test
public void WhenAllVerticalLineisOccupiedThenPlayerWin() {
    tablero.jugar(x: 0, y: 0); // juega X
    tablero.jugar(x: 1, y: 1); // juega Y
    tablero.jugar(x: 1, y: 0); // juega X
    tablero.jugar(x: 0, y: 1); // juega Y
    tablero.jugar(x: 2, y: 0); // juega X
    assertEquals(Board.Resultado.GANA_X, tablero.currentGameState);
}
```

IMPLEMENTANDO

```
//Recorremos las Columnas
for(int i=0;i<tamano;i++)
{
    if(estadoCasilla[0][i]==Ultimojugador
        && estadoCasilla[1][i]==Ultimojugador
        && estadoCasilla[2][i]==Ultimojugador
    ){
        return (Ultimojugador=='X')?Resultado.GANA_X:Resultado.GANA_O;
    }
}

return Resultado.NO_GANADOR;
```

Pasamos la prueba y es exitosa

```
@Test
public void WhenAllVerticalLineisOccupiedThenPlayerWin() {
    tablero.jugar(x: 0, y: 0); // juega X
    tablero.jugar(x: 1, y: 1); // juega Y
    tablero.jugar(x: 1, y: 0); // juega X
    tablero.jugar(x: 0, y: 1); // juega Y
    tablero.jugar(x: 2, y: 0); // juega X
    assertEquals(Board.Resultado.GANA_X,tablero.currentGameState);
}
```

Prueba – condición ganadora III

Análogo, pero ahora para las diagonales. Al principio no pasa porque falta implementar.

```

@Test
public void WhenAllDiagonalLineIsOccupiedThenPlayerWin() {
    tablero.jugar(x: 0, y: 0); // juega X
    tablero.jugar(x: 0, y: 1); // juega Y
    tablero.jugar(x: 1, y: 1); // juega X
    tablero.jugar(x: 0, y: 2); // juega Y
    tablero.jugar(x: 2, y: 2); // juega X
    assertEquals(Board.Resultado.GANA_X, tablero.currentGameState);
}

```

✓ Tests failed: 1 of 1 test – 18 ms

Test Results	18 ms
TestRequisito3	18 ms
WhenAllDiagonalLineIsOccupiedTh	18 ms

```

> Task :compileJava
> Task :processResources NO-SOURCE
> Task :classes
> Task :compileTestJava
> Task :processTestResources NO-SOURCE
> Task :testClasses
> Task :test FAILED

expected: <GANA_X> but was: <NO_GANADOR>
Expected :GANA_X
Actual   :NO_GANADOR

```

IMPLEMENTANDO

```

//Recorremos las Diagonal1
if( estadoCasilla[0][0]==Ultimojugador && estadoCasilla[1][1]==Ultimojugador &&
    return (Ultimojugador=='X')?Resultado.GANA_X:Resultado.GANA_0;
}

```

Volviendo a ejecutar pasa exitosamente


```

@Test
public void WhenAllDiagonalLine1isOccupiedThenPlayerWin() {
    tablero.jugar(x: 0, y: 0); // juega X
    tablero.jugar(x: 0, y: 1); // juega Y
    tablero.jugar(x: 1, y: 1); // juega X
    tablero.jugar(x: 0, y: 2); // juega Y
    tablero.jugar(x: 2, y: 2); // juega X
    assertEquals(Board.Resultado.GANA_X, tablero.currentGameState);
}

```

✓ Tests passed: 1 of 1 test – 14 ms

✓ Test Results 14 ms

```

> Task :compileJava
> Task :processResources NO-SOURCE
> Task :classes
> Task :compileTestJava
> Task :processTestResources NO-SOURCE
> Task :testClasses
> Task :test
BUILD SUCCESSFUL in 707ms

```

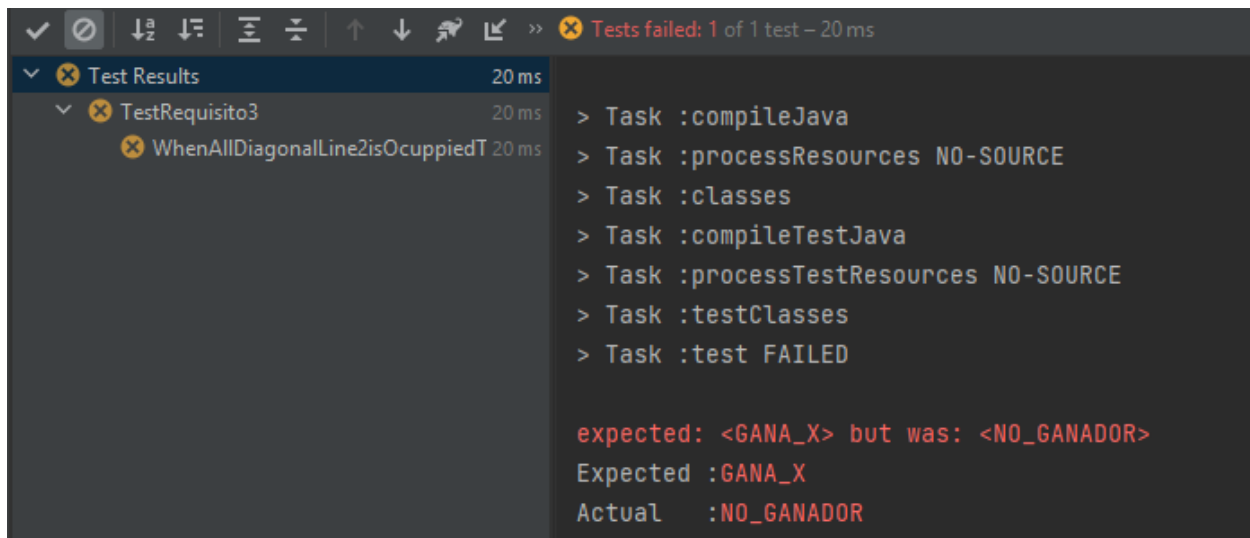
Prueba – condición ganadora IV

Análogo no pasa, falta implementar

```

@Test
public void WhenAllDiagonalLine2isOccupiedThenPlayerWin() {
    tablero.jugar(x: 0, y: 2); // juega X
    tablero.jugar(x: 0, y: 1); // juega Y
    tablero.jugar(x: 1, y: 1); // juega X
    tablero.jugar(x: 0, y: 0); // juega Y
    tablero.jugar(x: 2, y: 0); // juega X
    assertEquals(Board.Resultado.GANA_X, tablero.currentGameState);
}

```



IMPLEMENTACION

```
//Recorremos las Diagonal2
if(estadoCasilla[0][2]==Ultimojugador && estadoCasilla[1][1]==Ultimojugador &&
{
    return (Ultimojugador=='X')?Resultado.GANA_X:Resultado.GANA_0;
}

return Resultado.NO_GANADOR;
}
```

Requisito 4: condiciones de empate

El resultado es un empate cuando se llenan todas las casillas.

```
public class TestRequisito4 {
    6 usages
    Board tablero;
    @BeforeEach
    public void IniciarTablero() { tablero = new Board(); }
    @Test
```

Añadimos el empate :

```
public enum Resultado { GANA_O, GANA_X, NO_GANADOR, EMPATE }
8 usages
```

Ejecutamos el test y falta implementacion

```
13 public void WhenIsFullBoardThenReturnTie(){
14     tablero.jugar(x: 0, y: 0); // juega X
15     tablero.jugar(x: 1, y: 0); // juega O
16     tablero.jugar(x: 2, y: 0); // juega X
17     tablero.jugar(x: 0, y: 1); // juega O
18     tablero.jugar(x: 2, y: 1); // juega X
19     tablero.jugar(x: 1, y: 1); // juega O
20     tablero.jugar(x: 1, y: 2); // juega X
21     tablero.jugar(x: 2, y: 2); // juega O
22     tablero.jugar(x: 0, y: 2); // juega X
23     assertEquals(Board.Resultado.EMPATE, tablero.currentGameState);
24 }
25 }
```

IMPLEMENTACION

```
// Empate
int casillasOcupadas = 0;
for (int i = 0; i < tamaño; i++) {
    for (int j = 0; j < tamaño; j++) {
        if (estadoCasilla[i][j] != '\0') {
            casillasOcupadas++;
        }
    }
}
if (casillasOcupadas == tamaño * tamaño) {
    return Resultado.EMPATE;
}
return Resultado.NO_GANADOR;
```

Ejecutamos el test y pasa exitosamente:

```
public void WhenIsFullBoardThenReturnTie(){
    tablero.jugar(x: 0, y: 0); // juega X
    tablero.jugar(x: 1, y: 0); // juega 0
    tablero.jugar(x: 2, y: 0); // juega X
    tablero.jugar(x: 0, y: 1); // juega 0
    tablero.jugar(x: 2, y: 1); // juega X
    tablero.jugar(x: 1, y: 1); // juega 0
    tablero.jugar(x: 1, y: 2); // juega X
    tablero.jugar(x: 2, y: 2); // juega 0
    tablero.jugar(x: 0, y: 2); // juega X
    assertEquals(Board.Resultado.EMPATE, tablero.currentGameState);
}
```

Tests passed: 1 of 1 test – 15 ms

Test Results 15 ms

- > Task :compileJava
- > Task :processResources NO-SOURCE
- > Task :classes
- > Task :compileTestJava
- > Task :processTestResources NO-SOURCE
- > Task :testClasses
- > Task :test

Refactorización

Extraemos cada una de las evaluaciones de condición en un método booleano:

```

public Resultado EstablecerGanador() {
    if (HayGanadorEnFilas()) {
        return (Ultimojugador == 'X') ? Resultado.GANA_X : Resultado.GANA_O;
    }

    if (HayGanadorEnColumnas()) {
        return (Ultimojugador == 'X') ? Resultado.GANA_X : Resultado.GANA_O;
    }

    if (HayGanadorEnDiagonal1() || HayGanadorEnDiagonal2()) {
        return (Ultimojugador == 'X') ? Resultado.GANA_X : Resultado.GANA_O;
    }

    if (EsEmpate()) {
        return Resultado.EMPATE;
    }

    return Resultado.NO_GANADOR;
}

```

Esto vendrían a ser los métodos:

```

private boolean HayGanadorEnFilas() {
    for (int i = 0; i < tamaño; i++) {
        if (estadoCasilla[i][0] == Ultimojugador
            && estadoCasilla[i][1] == Ultimojugador
            && estadoCasilla[i][2] == Ultimojugador) {
            return true;
        }
    }
    return false;
}

```

1 usage

```

private boolean HayGanadorEnColumnas() {
    for (int i = 0; i < tamaño; i++) {
        if (estadoCasilla[0][i] == Ultimojugador
            && estadoCasilla[1][i] == Ultimojugador
            && estadoCasilla[2][i] == Ultimojugador) {
            return true;
        }
    }
    return false;
}

```

1 usage

```

private boolean HayGanadorEnDiagonal1() {
    return estadoCasilla[0][0] == Ultimojugador
        && estadoCasilla[1][1] == Ultimojugador
        && estadoCasilla[2][2] == Ultimojugador;
}

```

1 usage

```

private boolean HayGanadorEnDiagonal2() {
    return estadoCasilla[0][2] == Ultimojugador
        && estadoCasilla[1][1] == Ultimojugador
        && estadoCasilla[2][0] == Ultimojugador;
}

```

```
private boolean EsEmpate() {  
    int casillasOcupadas = 0;  
    for (int i = 0; i < tamaño; i++) {  
        for (int j = 0; j < tamaño; j++) {  
            if (estadoCasilla[i][j] != '\0') {  
                casillasOcupadas++;  
            }  
        }  
    }  
    return casillasOcupadas == tamaño * tamaño;  
}
```

Cobertura de código