

Requisito: Contar juegos ganados en juego simple

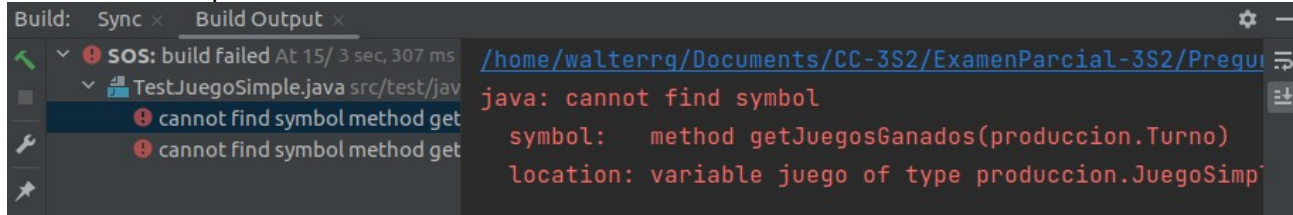
Se quiere saber cuantos juegos ha ganado el jugador azul y rojo. Se trabaja sobre la clase JuegoSimple y TestJuegoSimple del sprint 4.

Prueba: ningún jugador tiene alguna victoria

Prueba:

```
@Test
public void testJuegoSinVictoria() {
    assertEquals(0, juego.getJuegosGanados(Turno.AZUL));
    assertEquals(0, juego.getJuegosGanados(Turno.ROJO));
}
```

Resultados de la prueba:



Falla porque aún no se ha creado el método juegosGanados.

Implementación:

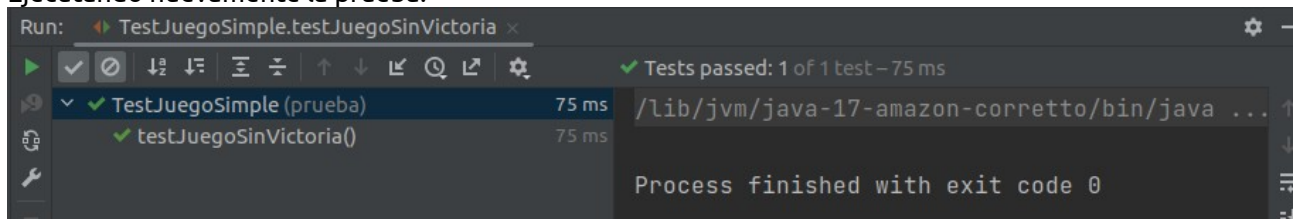
Se definen los campos:

```
private int juegosGanadosAzul = 0;
private int juegosGanadosRojo = 0;
```

y el método:

```
public int getJuegosGanados(Turno color) {
    if (color == Turno.AZUL) {
        return juegosGanadosAzul;
    } else {
        return juegosGanadosRojo;
    }
}
```

Ejecutando nuevamente la prueba:



La prueba pasa

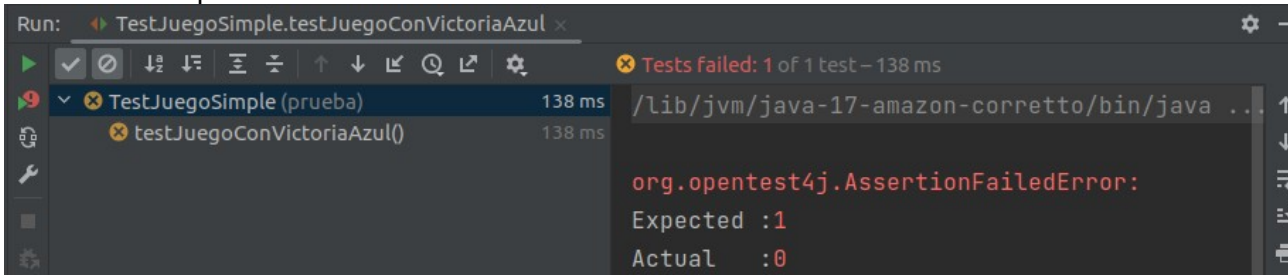
Prueba: el jugador azul tiene una victoria y el rojo ninguna

Prueba:

```
@Test
public void testJuegoConVictoriaAzul() {
    juego.realizarMovimiento(0, 0, Celda.S);
    juego.realizarMovimiento(1, 1, Celda.O);
    juego.realizarMovimiento(2, 2, Celda.S);

    assertEquals(1, juego.getJuegosGanados(Turno.AZUL));
    assertEquals(0, juego.getJuegosGanados(Turno.ROJO));
}
```

Resultado de la prueba:



```
Run: TestJuegoSimple.testJuegoConVictoriaAzul x
Tests failed: 1 of 1 test - 138 ms
TestJuegoSimple (prueba) 138 ms
testJuegoConVictoriaAzul() 138 ms
org.opentest4j.AssertionFailedError:
Expected :1
Actual   :0
```

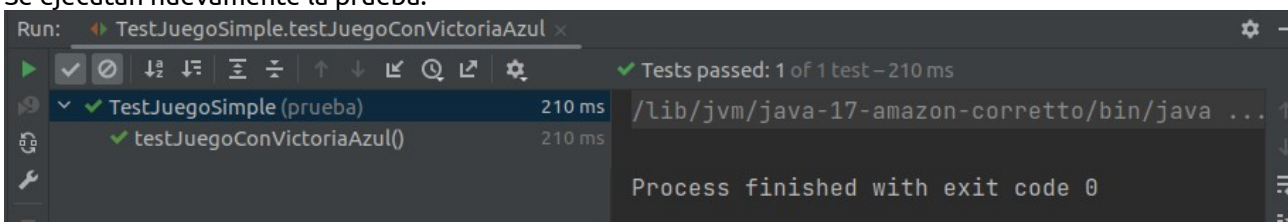
La prueba falla

Implementación:

Para la implementación se modifica el método actualizarEstadoJuego:

```
public void actualizarEstadoJuego(int fila, int columna) {
    if (hizoSos(fila, columna)) {
        ganador = getTurno();
        estadoJuegoActual = (turno == Turno.ROJO) ? EstadoJuego.GANO_ROJO :
EstadoJuego.GANO_AZUL;
        if(estadoJuegoActual == EstadoJuego.GANO_AZUL) {
            juegosGanadosAzul++;
        }
    } else if (esEmpate()) {
        estadoJuegoActual = EstadoJuego.EMPATE;
    }
}
```

Se ejecutan nuevamente la prueba:



```
Run: TestJuegoSimple.testJuegoConVictoriaAzul x
Tests passed: 1 of 1 test - 210 ms
TestJuegoSimple (prueba) 210 ms
testJuegoConVictoriaAzul() 210 ms
Process finished with exit code 0
```

La prueba pasa

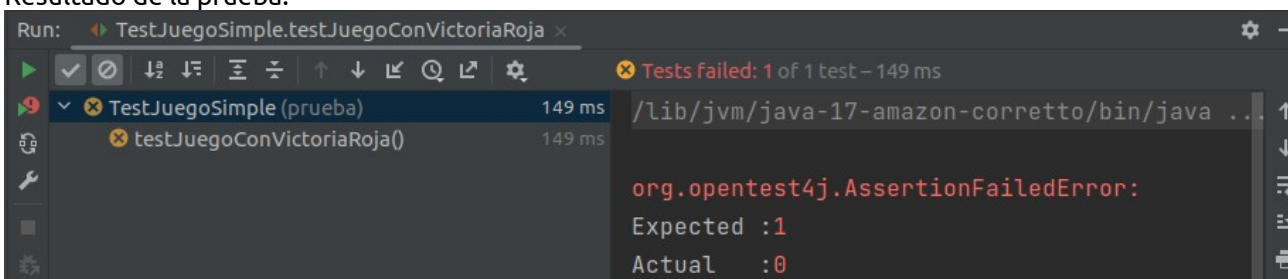
Prueba: el jugador rojo tiene una victoria y el azul ninguna

Prueba:

```
@Test
public void testJuegoConVictoriaRoja() {
    juego.realizarMovimiento(0, 2, Celda.S);
    juego.realizarMovimiento(0, 0, Celda.S);
    juego.realizarMovimiento(1, 1, Celda.O);
    juego.realizarMovimiento(2, 2, Celda.S);

    assertEquals(0, juego.getJuegosGanados(Turno.AZUL));
    assertEquals(1, juego.getJuegosGanados(Turno.ROJO));
}
```

Resultado de la prueba:



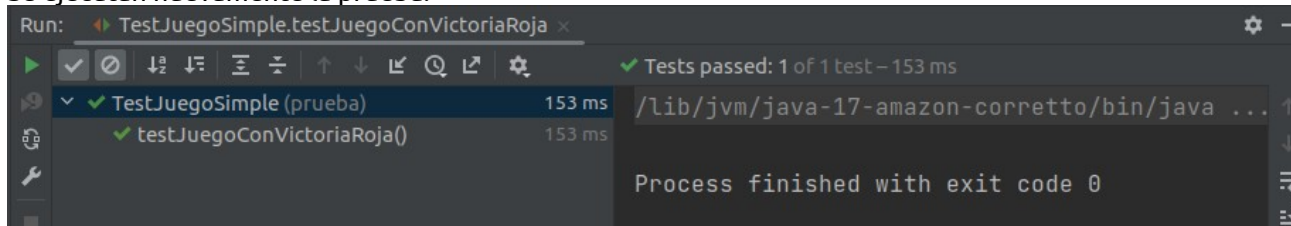
```
Run: TestJuegoSimple.testJuegoConVictoriaRoja x
Tests failed: 1 of 1 test - 149 ms
TestJuegoSimple (prueba) 149 ms
testJuegoConVictoriaRoja() 149 ms
org.opentest4j.AssertionFailedError:
Expected :1
Actual   :0
```

La prueba falla

Implementación:

```
public void actualizarEstadoJuego(int fila, int columna) {
    if (hizoSos(fila, columna)) {
        ganador = getTurno();
        estadoJuegoActual = (turno == Turno.ROJO) ? EstadoJuego.GANO_ROJO :
EstadoJuego.GANO_AZUL;
        if(estadoJuegoActual == EstadoJuego.GANO_AZUL) {
            juegosGanadosAzul++;
        } else {
            juegosGanadosRojo++;
        }
    } else if (esEmpate()) {
        estadoJuegoActual = EstadoJuego.EMPATE;
    }
}
```

Se ejecutan nuevamente la prueba:



La prueba pasa

Refactorización:

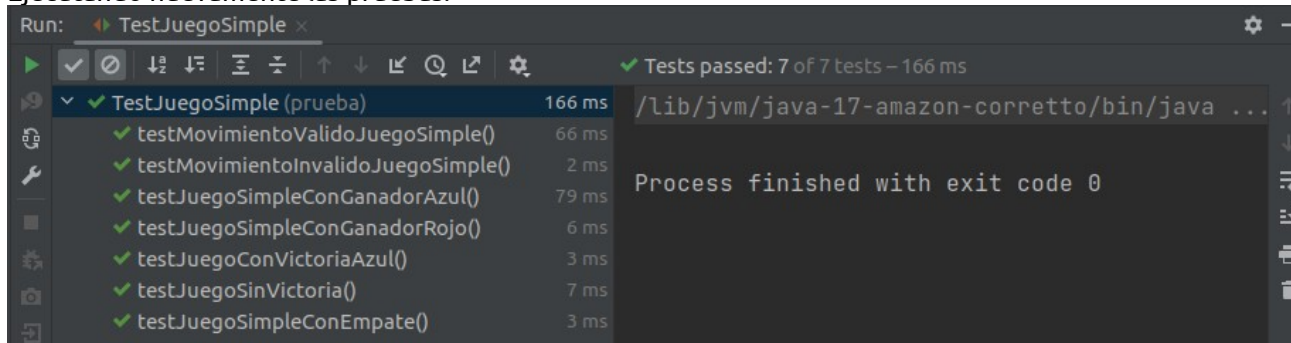
Se mueve la funcionalidad de actualizar el número de juegos ganados a su propio método y se llama desde el método realizarMovimiento luego de llamar al método actualizarEstadoJuego.

```
public void realizarMovimiento(int fila, int columna, Celda valorCelda) {
    if (fila >= 0 && fila < totalFilas && columna >= 0 && columna <
totalColumnas
        && getCelda(fila, columna) == Celda.VACIA) {
        setCelda(fila, columna, valorCelda);
        actualizarEstadoJuego(fila, columna);
        actualizarJuegosGanados();
        if (getEstadoJuego() == EstadoJuego.JUGANDO) {
            setTurno((getTurno() == Turno.ROJO) ? Turno.AZUL : Turno.ROJO);
        }
    }
}
```

```
public void actualizarEstadoJuego(int fila, int columna) {
    if (hizoSos(fila, columna)) {
        ganador = getTurno();
        estadoJuegoActual = (turno == Turno.ROJO) ? EstadoJuego.GANO_ROJO :
EstadoJuego.GANO_AZUL;
    } else if (esEmpate()) {
        estadoJuegoActual = EstadoJuego.EMPATE;
    }
}
```

```
public void actualizarJuegosGanados() {
    if(estadoJuegoActual == EstadoJuego.GANO_ROJO) {
        juegosGanadosRojo++;
    } else if(estadoJuegoActual == EstadoJuego.GANO_AZUL){
        juegosGanadosAzul++;
    }
}
```

Ejecutando nuevamente las pruebas:



Las pruebas pasan

Prueba: El jugador azul tiene varias victorias y el rojo ninguna

Prueba donde el jugador azul tiene dos victorias y el jugador rojo cero:

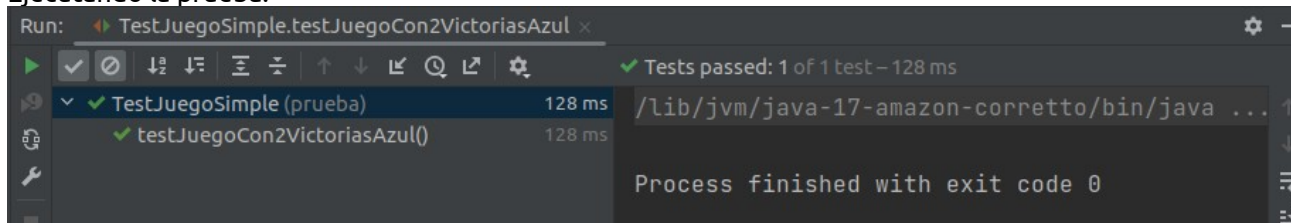
```
@Test
public void testJuegoCon2VictoriasAzul() {
    juego.realizarMovimiento(0, 0, Celda.S);
    juego.realizarMovimiento(1, 1, Celda.O);
    juego.realizarMovimiento(2, 2, Celda.S);

    juego.resetearJuego(3);
    juego.setEstadoJuego(EstadoJuego.JUGANDO);

    juego.realizarMovimiento(0, 0, Celda.S);
    juego.realizarMovimiento(1, 1, Celda.O);
    juego.realizarMovimiento(2, 2, Celda.S);

    assertEquals(2, juego.getJuegosGanados(Turno.AZUL));
    assertEquals(0, juego.getJuegosGanados(Turno.ROJO));
}
```

Ejecutando la prueba:



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

Prueba: El jugador rojo tiene varias victorias y el azul ninguna

Prueba:

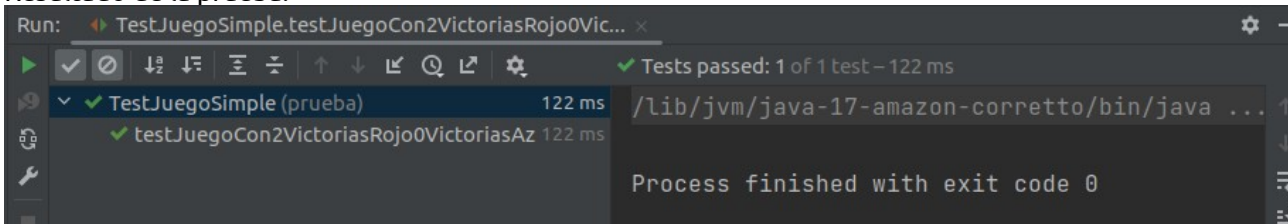
```
@Test
public void testJuegoCon2VictoriasRojo0VictoriasAzul() {
    juego.realizarMovimiento(0, 2, Celda.S);
    juego.realizarMovimiento(0, 0, Celda.S);
    juego.realizarMovimiento(1, 1, Celda.O);
    juego.realizarMovimiento(2, 2, Celda.S);

    juego.resetearJuego(3);

    juego.realizarMovimiento(0, 2, Celda.S);
    juego.realizarMovimiento(0, 0, Celda.S);
    juego.realizarMovimiento(1, 0, Celda.O);
    juego.realizarMovimiento(2, 0, Celda.S);

    assertEquals(0, juego.getJuegosGanados(Turno.AZUL));
    assertEquals(2, juego.getJuegosGanados(Turno.ROJO));
}
```

Resultado de la prueba:



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

Prueba: Ambos jugadores tienen varias victorias

Prueba:

```
@Test
public void testJuegoCon2VictoriasAzul2VictoriasRojo() {
    juego.realizarMovimiento(0, 0, Celda.S);
    juego.realizarMovimiento(1, 1, Celda.O);
    juego.realizarMovimiento(2, 2, Celda.S);

    juego.resetearJuego(3);
    juego.setEstadoJuego(EstadoJuego.JUGANDO);

    juego.realizarMovimiento(0, 0, Celda.S);
    juego.realizarMovimiento(1, 1, Celda.O);
    juego.realizarMovimiento(2, 2, Celda.S);

    juego.resetearJuego(3);
    juego.setEstadoJuego(EstadoJuego.JUGANDO);

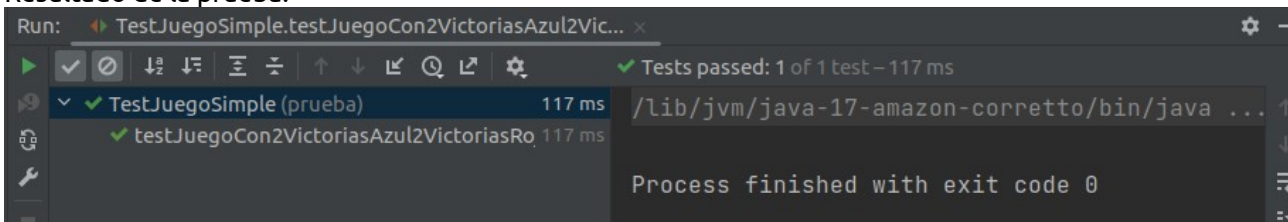
    juego.realizarMovimiento(0, 2, Celda.S);
    juego.realizarMovimiento(0, 0, Celda.S);
    juego.realizarMovimiento(1, 1, Celda.O);
    juego.realizarMovimiento(2, 2, Celda.S);

    juego.resetearJuego(3);
    juego.setEstadoJuego(EstadoJuego.JUGANDO);

    juego.realizarMovimiento(0, 2, Celda.S);
    juego.realizarMovimiento(0, 0, Celda.S);
    juego.realizarMovimiento(1, 0, Celda.O);
    juego.realizarMovimiento(2, 0, Celda.S);

    assertEquals(2, juego.getJuegosGanados(Turno.AZUL));
    assertEquals(2, juego.getJuegosGanados(Turno.ROJO));
}
```

Resultado de la prueba:



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