**Clase Shot**

**Mecánica de disparos**: La nave puede disparar únicamente hacia arriba. Sólo se puede tener un disparo por pantalla cada vez, es decir, hasta que el disparo actual no desaparezca no se podrán disparar más proyectiles.

El disparo desaparece si:

1) desaparece por la parte superior de la pantalla, o

2) alcanza a alguno de los alienígenas o de sus bombas.

Pruebas de caja negra

**1) ¿Qué se debe probar y por qué?**

Dado que estamos probando una mecánica de disparo con condiciones específicas, deberíamos centrarnos en:

* **Disparo único en pantalla**: Asegurar que no haya más de un disparo simultáneo.
* **Dirección de disparo**: Confirmar que el disparo se mueve solo hacia arriba.
* **Condiciones de desaparición del disparo**:
  + **Salida de la pantalla por la parte superior**.
  + **Colisión con alienígenas o sus bombas**.

Además, es crucial implementar y probar los límites de las coordenadas del disparo. **Si alguna de las coordenadas indicadas es mayor al máximo permitido, se le asignará el valor máximo permitido.** Esto asegura que el disparo no se desplace fuera de los límites definidos del juego.

**Por otro lado, si se introducen valores negativos de coordenada, se asignará el mínimo permitido (0).** Esto previene que el disparo se posicione fuera de la pantalla en el lado negativo del eje Y.

**2) ¿Cuál de las 4 pruebas de caja negra sería mejor y por qué?**

* **Clases de Equivalencia** es la mejor opción, ya que esta técnica permite dividir los posibles resultados en clases representativas (por ejemplo, colisión/no colisión y disparo en pantalla/sin disparo en pantalla). Esta estrategia permitirá cubrir las combinaciones relevantes sin tener que probar todos los escenarios posibles.
* **Robust Boundary Value Testing** para la parte de límites de las coordenadas del disparo. Esta técnica es particularmente útil para verificar que los disparos se comporten correctamente en los límites definidos, como asegurar que el disparo no supere el límite superior de la pantalla o que no tenga coordenadas negativas.

**3) Tipos de Pruebas de Equivalencia**

Dado que la mecánica de disparo en tu videojuego tiene condiciones específicas sobre cómo los disparos deben ser manejados, **la mejor opción sería usar Strong Normal Equivalence Class Testing**. Esto permite:

* Probar todos los caminos válidos y asegurar que el sistema se comporta correctamente bajo condiciones normales.

**4) Paso 1: Definir las Clases de Equivalencia**

|  |  |  |  |
| --- | --- | --- | --- |
| **Clases de Entrada** | | | |
| Clases Validas | Intento de disparar (A) | CV1 | Disparo exitoso (sin disparo activo) |
| Dirección del disparo (B) | CV2 | Disparo hacia arriba |
| Condiciones de Desaparición (C) | CV3 | Disparo que se mueve hacia arriba y sale de la pantalla |
| CV4 | Disparo que colisiona con un alienígena o bomba |
| Clases No Validas | Intento de disparar (A) | CNV1 | Intentar disparar cuando ya hay un disparo en pantalla |
| Dirección del disparo (B) | CNV2 | Intentar disparar hacia abajo, izquierda o derecha |
| Condiciones de Desaparición (C) | CNV3 | Disparo que no colisiona y no sale de la pantalla |

**5) Definir los Casos de pruebas:**

1CV(A) \*1CV(B) \* 2CV(C) = 2 Casos de Prueba

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CASO DE PRUEBA | A | B | C | SALIDA ESPERADA |
| 1 | CV1 | CV2 | CV3 | Disparo sale de la pantalla |
| 2 | CV1 | CV2 | CV4 | El disparo colisiona y se destruye |

**6) Test:**

*@Test  
public void testShotInitialization() {  
 assertNotNull(shot);  
 assertTrue(shot.isVisible(), "El disparo debe ser visible al inicializarse.");  
 assertEquals(0, shot.getX(), "La coordenada X debe ser 0.");  
 assertEquals(0, shot.getY(), "La coordenada Y debe ser 0.");  
}  
  
@Test  
public void testShotPositioning() {  
 shot.setX(100);  
 shot.setY(200);  
 assertEquals(100, shot.getX(), "La coordenada X debe ser 100.");  
 assertEquals(200, shot.getY(), "La coordenada Y debe ser 200.");  
}  
@Test  
public void testShotVisibility() {  
 shot.setVisible(false);  
 assertFalse(shot.isVisible(), "El disparo debe ser invisible.");  
 shot.setVisible(true);  
 assertTrue(shot.isVisible(), "El disparo debe ser visible.");  
}  
  
@Test  
public void testShotMovement() {  
 shot.setX(150);  
 shot.setY(150);  
 shot.setY(shot.getY() - 10); // Simular disparo hacia arriba  
 assertEquals(140, shot.getY(), "La coordenada Y debe ser 140 tras disparar.");  
}  
  
@Test  
public void testShotOutOfBounds() {  
 shot.setY(Commons.GROUND + 10); // Más allá del límite inferior  
 assertEquals(Commons.GROUND, shot.getY(), "El disparo no debe superar el límite de la pantalla.");  
}*

**7) Detalles sobre las pruebas**

1. **testShotInitialization:** Esta prueba verifica que, al crear un nuevo disparo, se inicialice correctamente. La expectativa de que las coordenadas sean (0, 0) y que el disparo sea visible es clave para asegurarte de que el disparo comienza en la posición correcta. Esto está relacionado con el **CASO DE PRUEBA 1**, ya que al comprobar que el disparo sale de la pantalla, estamos asegurando que comience en un estado válido.
2. **testShotPositioning:** Esta prueba se enfoca en establecer nuevas coordenadas para el disparo. Esto es fundamental para el **CASO DE PRUEBA 2**, ya que una posición incorrecta podría afectar las colisiones y la lógica del juego.
3. **testShotVisibility:** La visibilidad del disparo es importante en el contexto de colisiones, ya que un disparo no visible no puede afectar a otros objetos en la pantalla. Esta prueba se relaciona con el **CASO DE PRUEBA 2** porque la visibilidad puede influir en la lógica de colisión.
4. **testShotMovement:** Esta prueba asegura que el disparo se mueva hacia arriba correctamente. Dado que el movimiento del disparo es esencial para su funcionalidad en el juego, se relaciona con el **CASO DE PRUEBA 2**, donde un disparo exitoso debería resultar en una colisión y destrucción de los enemigos.
5. **testShotOutOfBounds:** Esta prueba se asegura de que el disparo no salga del límite inferior de la pantalla. Esto también se relaciona con el **CASO DE PRUEBA 1**, ya que el disparo no debe superar el límite de la pantalla.

**8) Resultados**

*[ERROR] Failures:*

*[ERROR] ShotTest.testShotInitialization:26 La coordenada X debe ser 0. ==> expected: <0> but was: <6>*

*[ERROR] ShotTest.testShotOutOfBounds:56 El disparo no debe superar el límite de la pantalla. ==> expected: <290> but was: <300>*

*[INFO]*

*[ERROR] Tests run: 5, Failures: 2, Errors: 0, Skipped: 0*

**9) Estado de los Casos de Prueba**

|  |  |  |
| --- | --- | --- |
| **Caso de Prueba** | **Resultado** | **Fallas** |
| 1 | Fallido | - Fallo en **testShotInitialization:** La coordenada X no es 0 (actualmente es 6).  - Fallo en **testShotOutOfBounds**: El disparo supera el límite de la pantalla. |
| 2 | Exitoso | - Todas las pruebas relacionadas con este caso (**testShotPositioning, testShotVisibility, testShotMovement**) se ejecutan sin fallos. |

**10) Prueba de valores límite:**

* X o Y > Max (Valor max. Permitido)

***Prueba:*** *Si x o y se le asigna una coordenada valida esta funciona.*

***Prueba:*** *Si x se le asigna un valor max al permitido, se le asigne el max permitido.*

***Prueba:*** *Si y se le asigna un valor max al permitido, se le asigne el max permitido****.***

* X o Y = - nº (Valor mín. permitido = 0)

***Prueba:*** *Si x se le asigna un valor negativo, se le asigne el min permitido.*

***Prueba:*** *Si y se le asigna un valor negativo, se le asigne el min permitido****.***

**11) Robust Boundary Value Testing**

* n = 2
* 6n + 1

6(2) +1 = 13 casos de pruebas.

**12) Casos de Prueba para Robust Boundary Value Testing**

**Valores a considerar:**

* Para X:
  + Justo debajo del límite inferior: -1
  + Límite inferior: 0
  + Justo por encima del límite inferior: 1
  + Valor nominal: **179** (la mitad de 0 y 358).
  + Justo por debajo del límite superior: 357
  + Límite superior: 358
  + Justo por encima del límite superior: 359
* Para Y:
  + Justo debajo del límite inferior: -1
  + Límite inferior: 0
  + Justo por encima del límite inferior: 1
  + Valor nominal: **175** (la mitad de 0 y 350).
  + Justo por debajo del límite superior: 349
  + Límite superior: 350
  + Justo por encima del límite superior: 351

|  |  |  |  |
| --- | --- | --- | --- |
| **Caso** | **X** | **Y** | **Salida Esperada (X, Y)** |
| 1 | -1 | 175 | (0,175) |
| 2 | 0 | 175 | (0,175) |
| 3 | 1 | 175 | (1,175) |
| 4 | 179 | 175 | (179,175) |
| 5 | 357 | 175 | (357,175) |
| 6 | 358 | 175 | (358,175) |
| 7 | 359 | 175 | (358,175) |
| 8 | 175 | -1 | (175,0) |
| 9 | 175 | 0 | (175,0) |
| 10 | 175 | 1 | (175,1) |
| 11 | 175 | 349 | (175,349) |
| 12 | 175 | 350 | (175,350) |
| 13 | 175 | 351 | (175,350) |

**13) TEST:**

*// VALORES LIMITES  
 @Test  
 public void testShotInitializationWithNegativeXAndValidY() {  
 Shot shot = new Shot(-1, 175);  
 assertEquals(0, shot.getX(), "(C1) La coordenada X debería ser 0 para un valor negativo.");  
 assertEquals(175, shot.getY(), "(C1) La coordenada Y debería ser 175.");  
 }  
  
 @Test  
 public void testShotInitializationWithZeroXAndValidY() {  
 Shot shot = new Shot(0, 175);  
 assertEquals(0, shot.getX(), "(C2) La coordenada X debería ser 0.");  
 assertEquals(175, shot.getY(), "(C2) La coordenada Y debería ser 175.");  
 }  
  
 @Test  
 public void testShotInitializationWithValidXAndY() {  
 Shot shot = new Shot(1, 175);  
 assertEquals(1, shot.getX(), "(C3) La coordenada X debería ser 1.");  
 assertEquals(175, shot.getY(), "(C3) La coordenada Y debería ser 175.");  
 }  
  
 @Test  
 public void testShotInitializationWithValidXAndInvalidY() {  
 Shot shot = new Shot(179, 175);  
 assertEquals(179, shot.getX(), "(C4) La coordenada X debería ser 179.");  
 assertEquals(175, shot.getY(), "(C4) La coordenada Y debería ser 175.");  
 }  
  
 @Test  
 public void testShotInitializationWithZeroXAndY() {  
 Shot shot = new Shot(Commons.BOARD\_WIDTH - 1, 175);  
 assertEquals(Commons.BOARD\_WIDTH - 1, shot.getX(), "(C5) La coordenada X debería ser 357.");  
 assertEquals(175, shot.getY(), "(C5) La coordenada Y debería ser 175.");  
 }  
  
  
 @Test  
 public void testShotInitializationWithZeroXAndYAboveMin() {  
 Shot shot = new Shot(Commons.BOARD\_WIDTH, 175);  
 assertEquals(Commons.BOARD\_WIDTH, shot.getX(), "(C6) La coordenada X debería ser 358.");  
 assertEquals(175, shot.getY(), "(C6) La coordenada Y debería ser 175.");  
 }  
  
 @Test  
 public void testShotInitializationWithXAboveMinAndInvalidY() {  
 Shot shot = new Shot(Commons.BOARD\_WIDTH + 1, 175);  
 assertEquals(Commons.BOARD\_WIDTH, shot.getX(), "(C7) La coordenada X debería ser 358.");  
 assertEquals(175, shot.getY(), "(C7) La coordenada Y debería ser 175.");  
 }  
  
  
 @Test  
 public void testShotInitializationWithXAboveMinAndZeroY() {  
 Shot shot = new Shot(175, -1);  
 assertEquals(175, shot.getX(), "(C8) La coordenada X debería ser 175.");  
 assertEquals(0, shot.getY(), "(C8) La coordenada Y debería ser 0.");  
 }  
  
  
 @Test  
 public void testShotInitializationWithMaxXAndMaxY() {  
 Shot shot = new Shot(175, 0);  
 assertEquals(175, shot.getX(), "(C9) La coordenada X debería ser 175.");  
 assertEquals(0, shot.getY(), "(C9) La coordenada Y debería ser 0.");  
 }  
  
  
 @Test  
 public void testShotInitializationWithXAboveMaxAndYAboveMax() {  
 Shot shot = new Shot(175, 1);  
 assertEquals(175, shot.getX(), "(C10) La coordenada X debería ser el valor 175.");  
 assertEquals(1, shot.getY(), "(C10) La coordenada Y debería ser el valor 1.");  
 }  
  
 @Test  
 public void testShotInitializationWithXAboveMaxAndMaxY() {  
 Shot shot = new Shot(175, Commons.BOARD\_HEIGHT -1);  
 assertEquals(175, shot.getX(), "(C11) La coordenada X debería ser el valor 175.");  
 assertEquals(Commons.BOARD\_HEIGHT -1, shot.getY(), "(C11) La coordenada Y debería ser 349.");  
 }  
  
 @Test  
 public void testShotInitializationWithMaxXAndYAboveMax() {  
 Shot shot = new Shot(175, Commons.BOARD\_HEIGHT);  
 assertEquals(175, shot.getX(), "(C12) La coordenada X debería ser 175.");  
 assertEquals(Commons.BOARD\_HEIGHT, shot.getY(), "(C12)La coordenada Y debería ser el valor 350.");  
 }  
  
  
 @Test  
 public void testShotInitializationWithMaxXAndY() {  
 Shot shot = new Shot(175, Commons.BOARD\_HEIGHT + 1);  
 assertEquals(175, shot.getX(), "(C13) La coordenada X debería ser 175.");  
 assertEquals(Commons.BOARD\_HEIGHT, shot.getY(), "(C13) La coordenada Y debería ser el valor max 350.");  
 }*

**14) Resultados:**

*[ERROR] Failures:*

*[ERROR] ShotTest.testShotInitializationWithMaxXAndMaxY:121 (C9) La coordenada X debería ser 175. ==> expected: <175> but was: <181>*

*[ERROR] ShotTest.testShotInitializationWithMaxXAndY:151 (C13) La coordenada X debería ser 175. ==> expected: <175> but was: <181>*

*[ERROR] ShotTest.testShotInitializationWithMaxXAndYAboveMax:143 (C12) La coordenada X debería ser 175. ==> expected: <175> but was: <181>*

*[ERROR] ShotTest.testShotInitializationWithNegativeXAndValidY:62 (C1) La coordenada X debería ser 0 para un valor negativo. ==> expected: <0> but was: <5>*

*[ERROR] ShotTest.testShotInitializationWithValidXAndInvalidY:83 (C4) La coordenada X debería ser 179. ==> expected: <179> but was: <185>*

*[ERROR] ShotTest.testShotInitializationWithValidXAndY:76 (C3) La coordenada X debería ser 1. ==> expected: <1> but was: <7>*

*[ERROR] ShotTest.testShotInitializationWithXAboveMaxAndMaxY:136 (C11) La coordenada X debería ser el valor 175. ==> expected: <175> but was: <181>*

*[ERROR] ShotTest.testShotInitializationWithXAboveMaxAndYAboveMax:129 (C10) La coordenada X debería ser el valor 175. ==> expected: <175> but was: <181>*

*[ERROR] ShotTest.testShotInitializationWithXAboveMinAndInvalidY:105 (C7) La coordenada X debería ser 358. ==> expected: <358> but was: <365>*

*[ERROR] ShotTest.testShotInitializationWithXAboveMinAndZeroY:113 (C8) La coordenada X debería ser 175. ==> expected: <175> but was: <181>*

*[ERROR] ShotTest.testShotInitializationWithZeroXAndValidY:69 (C2) La coordenada X debería ser 0. ==> expected: <0> but was: <6>*

*[ERROR] ShotTest.testShotInitializationWithZeroXAndY:90 (C5) La coordenada X debería ser 357. ==> expected: <357> but was: <363>*

*[ERROR] ShotTest.testShotInitializationWithZeroXAndYAboveMin:98 (C6) La coordenada X debería ser 358. ==> expected: <358> but was: <364>*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Caso** | **X** | **Y** | **Salida Esperada (X, Y)** | **Resultado** | **Descripción del Error** |
| 1 | -1 | 175 | (0,175) | (5,175) | La coordenada X debe ser 0 pero para un valor negativo, pero es 5 |
| 2 | 0 | 175 | (0,175) | (6,175) | La coordenada X debe ser 0 pero es 6 |
| 3 | 1 | 175 | (1,175) | (7,175) | La cordenada X debe ser 1 pero es 7 |
| 4 | 179 | 175 | (179,175) | (185,175) | La coordenada X debe ser 179 pero es 185 |
| 5 | 357 | 175 | (357,175) | (363,175) | La coordenada X debe ser 357 pero es 363 |
| 6 | 358 | 175 | (358,175) | (364,175) | La coordenada X debe ser 358 pero es 364 |
| 7 | 359 | 175 | (358,175) | (365,175) | La coordenada X debe ser 358 pero es 365 |
| 8 | 175 | -1 | (175,0) | (181,0) | La coordenada X debe ser 175 pero es 181 |
| 9 | 175 | 0 | (175,0) | (181,0) | La coordenada X debe ser 175 pero es 181 |
| 10 | 175 | 1 | (175,1) | (181,1) | La coordenada X debe ser 175 pero es 181 |
| 11 | 175 | 349 | (175,349) | (181,349) | La coordenada X debe ser 175 pero es 181 |
| 12 | 175 | 350 | (175,350) | (181,350) | La coordenada X debe ser 175 pero es 181 |
| 13 | 175 | 351 | (175,350) | (181,350) | La coordenada X debe ser 175 pero es 181 |

**14) Resumen del comportamiento observado:**

1. **Coordenada Y:**
   * Si se excede el valor máximo (350), se asigna correctamente al máximo permitido (350).
   * Si es negativo, se ajusta correctamente a 0.
2. **Coordenada X:**
   * Siempre muestra un incremento de +6 respecto al valor esperado, independientemente de si el valor está en el rango permitido o no.

**15) Resumen de Resultados**

* **Total de Casos de Prueba Ejecutados:** 13
* **Total de Casos de Prueba Superados:** 0
* **Total de Casos de Prueba NO Superados:** 13

**Fase de Depuración**

Todos los casos de prueba han fallado, indicando que la implementación actual tiene problemas significativos que necesitan ser abordados:

1. **Constructor de la Clase Shot:** Es fundamental revisar cómo se están inicializando las coordenadas X e Y.
2. **Lógica de Ajuste:** Asegurarse de que los valores se ajusten correctamente según las condiciones (valores negativos, máximos y mínimos).
3. **Revisar Errores Comunes:** Es necesario analizar cualquier ajuste accidental en los valores de X que pueda estar ocurriendo durante la inicialización.