**CAJA BLANCA DE SHOT**

Para las pruebas de caja blanca referentes a la clase Shot, el único método que se prueba es Shot(),  
que a su vez lo único que hace es llamar al metodo initShot.  
  
Este método tiene una lógica sencilla: asigna una imagen al disparo y ajusta sus coordenadas x e y de acuerdo  
con valores constantes (H\_SPACE y V\_SPACE).  
  
public Shot(int x, int y) {  
 initShot(x, y);  
}  
  
 private void initShot(int x, int y) {  
  
1) var shotImg = "src/main/resources/images/shot.png";  
2) var ii = new ImageIcon(shotImg);  
3) setImage(ii.getImage());  
  
4) int H\_SPACE = 6;  
5) setX(x + H\_SPACE);  
  
6) int V\_SPACE = 1;  
7) setY(y - V\_SPACE);  
 }

1. Grafo:  
   (I) → (1) → (2) → (3) → (F)

1. Complejidad Ciclomática

* El número de regiones del grafo es igual a la complejidad ciclomática = 1
* V(G) = A – N + 2 => 4 – 5 + 2 = 1

Donde A es el número de aristas y N es el número de nodos contenidos en el grafo V

* V(G) = P + 1 => 0 + 1 = 1

Donde P es el número de nodos predicado contenidos en el grafo.

La complejidad ciclomática de initShot es 1, lo cual indica que solo hay un camino posible en el método (es una estructura secuencial sin bifurcaciones)

1. Conjunto de caminos independientes  
   Dado que la complejidad ciclomática es 1, solo hay un camino independiente en initShot:  
     
   Camino único: (I → 1→ 2 → 3 → F)
2. Conjunto de casos de prueba:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Camino | X | Y | Salida esperada |
| 1 | I → 1→ 2 → 3 → F | 100 | 50 | (106, 49) |

1. Implementación:

@Test  
public void testInitShot() {  
 // Verifica que la imagen se haya cargado correctamente  
 assertNotNull(shot.getImage(), "La imagen del disparo debe estar inicializada.");  
  
 // Verifica la coordenada X ajustada con H\_SPACE  
 assertEquals(106, shot.getX(), "La coordenada X debería ser 106 (100 + 6).");  
  
 // Verifica la coordenada Y ajustada con V\_SPACE  
 assertEquals(49, shot.getY(), "La coordenada Y debería ser 49 (50 - 1).");  
}

1. Resultados obtenidos:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Camino | X | Y | Salida |
| 1 | I → 1→ 2 → 3 → F | 100 | 50 | (106, 49) |

En este caso la prueba para la clase shot fueron superada correctamente, no se encontraron problemas de lógica en este método, y no se requiere depuración.

**CAJA BLANCA DE GAMEINIT**  
  
 private void gameInit() {  
  
 (1) this.aliens = new ArrayList<>();  
  
 (2) for (int i = 0; i < 4; i++) {  
 (3) for (int j = 0; j < 6; j++) {  
  
 (4) var alien = new Alien(Commons.ALIEN\_INIT\_Y + 18 \* j,  
 (5) Commons.ALIEN\_INIT\_Y + 18 \* i);  
 (6) this.aliens.add(alien);  
 }  
 }  
  
 (7) this.player = new Player();  
 (8) this.shot = new Shot();  
 }

1. Grafo:

v

v

F

F

2. Complejidad Ciclomática  
 El número de regiones del grafo es igual a la complejidad  
 ciclomática = 3  
  
 V(G) = A – N + 2 => 8 – 7 + 2 = 3  
 Donde A es el número de aristas y N es el número de nodos  
 contenidos en el grafoV  
  
 V(G) = P + 1 => 2 + 1 = 3  
 Donde P es el número de nodos predicado contenidos en el grafo.  
  
 La complejidad ciclomática de gameInit() es 3, lo cual indica que solo hay 3 camino posible  
 en el método.  
  
 3. Conjunto de caminos independientes  
 Dado que la complejidad ciclomática es 3 en initShot:  
  
 Caminos:  
 1) I → 1 → 2 → [3 → 4 → 3 → 2] → 5 → F  
 2) I → 1 → 2 → 5 → F  
 3) I → 1 → 2 → [3 → 2]→ 5 → F

**CAJA BLANCA DE UPDATE**

private void update() {  
  
(1) if (deaths == Commons.CHANCE) {  
 inGame = false;  
(2) timer.stop();  
 message = "Game won!";  
 }  
  
 this.player.act();  
 update\_shots();  
(3) update\_aliens();  
 update\_bomb();  
 }

1. Grafo:

v

1. Complejidad Ciclomática  
   * El número de regiones del grafo es igual a la complejidad  
     ciclomática = 2
   * V(G) = A – N + 2 => 5 – 5 + 2 = 2  
     Donde A es el número de aristas y N es el número de nodos  
     contenidos en el grafo V
   * V(G) = P + 1 => 1 + 1 = 2  
     Donde P es el número de nodos predicado contenidos en el grafo.
   * La complejidad ciclomática de update() es 2, lo cual indica que solo hay 2 camino posible en el método.
2. Conjunto de caminos independientes

Dado que la complejidad ciclomática es 2 en update:  
  
 Caminos:  
 1) { I → 1 → 2 → 3→ F }  
 2) { I → 1 → 3→ F }

El método update tiene un flujo relativamente simple, con una única condición (if), y el flujo es secuencial en el resto de las instrucciones. La única bifurcación ocurre en la condición deaths == Commons.CHANCE, lo que da lugar a los dos caminos independientes.

**CAJA BLANCA DE UPDATE\_SHOT**

private void update\_shots() {  
 if (this.shot.isVisible()) {  
  
 int shotX = this.shot.getX();  
 int shotY = this.shot.getY();  
  
 for (Alien alien : this.aliens) {  
  
 int alienX = alien.getX();  
 int alienY = alien.getY();  
  
 if (alien.isVisible() && this.shot.isVisible()) {  
 if (shotX >= (alienX)  
 && shotX <= (alienX + Commons.ALIEN\_WIDTH)  
 && shotY >= (alienY)  
 && shotY <= (alienY + Commons.ALIEN\_HEIGHT)) {  
  
 var ii = new ImageIcon(explImg);  
 alien.setImage(ii.getImage());  
 alien.setDying(true);  
 deaths--;  
 this.shot.die();  
 }  
 }  
 }  
  
 int y = this.shot.getY();  
 y -= 4;  
  
 if (y < 0) {  
 this.shot.die();  
 } else {  
 this.shot.setY(y);  
 }  
 }  
 }

1. Grafo:

v

v

F

F

v

F

F

v

v

F

11

10

1. Complejidad Ciclomática  
   * El número de regiones del grafo es igual a la complejidad  
     ciclomática = 6
   * V(G) = A – N + 2 => 17 – 13 + 2 = 6  
     Donde A es el número de aristas y N es el número de nodos  
     contenidos en el grafo V
   * V(G) = P + 1 => 5 + 1 = 6  
     Donde P es el número de nodos predicado contenidos en el grafo.
   * La complejidad ciclomática de update\_shots () es 6, lo cual indica que solo hay 6 camino posible en el método.
2. Conjunto de caminos independientes

Dado que la complejidad ciclomática es 6 en update\_shots:

Caminos:

1. I → 1 → F
2. I → 1 → 2 → 3 → 4 → 5 → 6 → 7 → 3 → 8 → 9 → 10 → F
3. I → 1 → 2 → 3 → 8 → 9 → 10 → F
4. I → 1 → 2 → 3 → 8 → 9 → 11 → F
5. I → 1 → 2 → 3 → 4 → 5 → 3 → 8 → 9 → 10 → F
6. I → 1 → 2 → 3 → 4 → 5 → 6 → 3 → 8 → 9 → 10 → F

**CAJA BLANCA DE UPDATE\_ALIENS**

private void update\_aliens(){  
 for (Alien alien : this.aliens) {  
  
 int x = alien.getX();  
  
 if (x <= Commons.*BOARD\_WIDTH* - Commons.*BORDER\_RIGHT* && direction != -1) {  
  
 direction = 0;  
  
 Iterator<Alien> i1 = this.aliens.iterator();  
  
 while (i1.hasNext()) {  
  
 Alien a2 = i1.next();  
 a2.setY(a2.getY() + Commons.*GO\_DOWN*);  
 }  
 }  
  
 if (x <= Commons.*BORDER\_LEFT* && direction != 1) {  
  
 direction = 1;  
  
 Iterator<Alien> i2 = this.aliens.iterator();  
  
 while (i2.hasNext()) {  
  
 Alien a = i2.next();  
 a.setX(a.getY() + Commons.*GO\_DOWN*);  
 }  
 }  
 }  
  
 Iterator<Alien> it = this.aliens.iterator();  
  
 while (it.hasNext()) {  
  
 Alien alien = it.next();  
  
 if (alien.isVisible()) {  
  
 int y = alien.getY();  
  
 if (y > Commons.*GROUND* - Commons.*ALIEN\_HEIGHT*) {  
 inGame = false;  
 message = "Invasion!";  
 }  
  
 alien.act(direction);   
 }  
 }  
}

17

16

13

14

15

12

11

10

9

88

7

6

5

3

4

2

1

18

1. Grafo:

F

v

F

F

F

v

v

F

v

F

F

F

F

v

v

v

v

18

17

16

15

14

133

12

11

10

1. Complejidad Ciclomática  
   * El número de regiones del grafo es igual a la complejidad  
     ciclomática = 9
   * V(G) = A – N + 2 => 27 – 20 + 2 = 9  
     Donde A es el número de aristas y N es el número de nodos  
     contenidos en el grafo V
   * V(G) = P + 1 => 8+ 1 = 9  
     Donde P es el número de nodos predicado contenidos en el grafo.
   * La complejidad ciclomática de update\_aliens () es 9, lo cual indica que solo hay 9 camino posible en el método.
2. Conjunto de caminos independientes

Dado que la complejidad ciclomática es 9 en update\_aliens:  
  
 Caminos:

1. I → 1 → 11 → 12 → F
2. I → 1 → 2 → 3 → 4 → 5 → 7 → 1 → 11 → 12
3. I → 1 → 2 → 3 → 4 → 5 → 6 → 5 → 7 → 1 → 11 →12→ F
4. I → 1 → 2 → 3 → 7 → 1 → 11 → 12 → F
5. I → 1 → 2 → 3 → 7 → 8 → 9 → 1 → 11 → 12 → F
6. I → 1 → 2 → 3 → 7 → 8 → 9 → 10 → 9 → 1 → 11 → 12 → F
7. I → 1 → 11 → 12 → 13 →14 →12 →F
8. I → 1 → 11 → 12 → 13 → 14 → 15 → 16 → 17 → 18 → 12 → F
9. I → 1 → 11 → 12 → 13 → 14 → 15 → 16 → 18 → 12 → F

**CAJA BLANCA DE UPDATE\_BOMB**

private void update\_bomb(){  
 var generator = new Random();  
  
 for (Alien alien : this.aliens) {  
  
 int shot = generator.nextInt(15);  
 Alien.Bomb bomb = alien.getBomb();  
  
 if (shot == Commons.*CHANCE* && alien.isVisible() && bomb.isDestroyed()) {  
  
 bomb.setDestroyed(false);   
 bomb.setX(alien.getX());  
 bomb.setY(alien.getY());  
 }  
  
 int bombX = bomb.getX();  
 int bombY = bomb.getY();  
 int playerX = this.player.getX();  
 int playerY = this.player.getY();  
  
 if (this.player.isVisible() && !bomb.isDestroyed()) {  
  
 if (bombX >= (playerX)  
 && bombX <= (playerX + Commons.*PLAYER\_WIDTH*)  
 && bombY >= (playerY)  
 && bombY <= (playerY + Commons.*PLAYER\_HEIGHT*)) {  
  
 var ii = new ImageIcon(explImg);  
 this.player.setImage(ii.getImage());  
 this.player.setDying(false);  
 bomb.setDestroyed(true);  
 }  
 }  
  
 if (!bomb.isDestroyed()) {  
  
 bomb.setY(bomb.getY() + 1);  
  
 if (bomb.getY() >= Commons.*GROUND* - Commons.*BOMB\_HEIGHT*) {  
  
 bomb.setDestroyed(false);  
 }  
 }  
 }  
  
}

13

12

11

10

4

9

81

7

6

5

3

2

1

1. Grafo:

F

v

v

F

F

F

F

F

v

v

13

12

11

10

1. Complejidad Ciclomática  
   * El número de regiones del grafo es igual a la complejidad  
     ciclomática = 7
   * V(G) = 20 – 15 + 2 => 5 + 2 = 7  
     Donde A es el número de aristas y N es el número de nodos  
     contenidos en el grafo V
   * V(G) = P + 1 => 6 + 1 = 7  
     Donde P es el número de nodos predicado contenidos en el grafo.
   * La complejidad ciclomática de update\_bomb() es 7, lo cual indica que solo hay 7 camino posible en el método.
2. Conjunto de caminos independientes

Dado que la complejidad ciclomática es 7 en update\_bomb:  
  
 Caminos:

1. I → 1 → 2 → F
2. I → 1 → 2 → [ 3 → 4 → 5 → 6 → 7 → 8 → 9 → 10 → 2] → F
3. I → 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9 → 10 → 11 → 12 → F
4. I → 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9 → 10 → 11 → 12 → 13 → F
5. I → 1 → 2 → [ 3→ 4 → 6→ 7 → 8 → 9 → 10 → 2] → F
6. I → 1 → 2 → [ 3→ 4 → 6→ 7 → 10 → 2] → F
7. I → 1 → 2 → [ 3→ 4 → 6→ 8 → 10→ 2] → F