**Bloque 1: NIVEL 1 CAJA NEGRA: IWT42 Grupo 6**

***Nombres:*  
Adrián Infante Pérez**

**Manuel Guerra Solís**

INDICE

[Nivel 1: Caja Negra 3](#_Toc182865195)

[1. Métodos probados, estrategia de prueba y justificación. 3](#_Toc182865196)

[Shot 3](#_Toc182865197)

[Alien 3](#_Toc182865198)

[Bomb 4](#_Toc182865199)

[Player 4](#_Toc182865200)

[Board 4](#_Toc182865201)

[2. Conjunto de casos de prueba diseñados, entradas a probar salidas esperadas 6](#_Toc182865202)

[Shot 6](#_Toc182865203)

[Alien 6](#_Toc182865204)

[Bomb 7](#_Toc182865205)

[Player 7](#_Toc182865206)

[Board 8](#_Toc182865207)

[3. Las implementaciones en Junit 5 de los casos de prueba definidos 9](#_Toc182865208)

[Shot 9](#_Toc182865209)

[Alien 9](#_Toc182865210)

[Bomb 10](#_Toc182865211)

[Player 11](#_Toc182865212)

[Board 13](#_Toc182865213)

[4. Los resultados obtenidos, casos de prueba NO han sido superados casos de prueba han sido superados. 15](#_Toc182865214)

[Shot 15](#_Toc182865215)

[Alien 15](#_Toc182865216)

[Bomb 15](#_Toc182865217)

[Player 16](#_Toc182865218)

[Board 16](#_Toc182865219)

# Nivel 1: Caja Negra

## Métodos probados, estrategia de prueba y justificación.

Las estrategias de prueba que hemos tenido en cuenta y han sido usadas durante la práctica se pueden clasificar de la siguiente manera:

* **Prueba de valor límite:** Valores que se encuentran en los límites de los valores de entrada. Se aplica cuando los valores no válidos son un problema
* **Prueba de valores especiales:** Valores que el desarrollador tiene razones para creer que podrían producir un error. Se aplica cuando hay un conocimiento del dominio y se conocen puntos típicos de fallo.
* **Clases de equivalencia:** Se utiliza información sobre el mapeo funcional (dominio - rango) para identificar casos de prueba. Se aplica cuando hay un conocimiento sobre el dominio de las variables de entrada
* **Tablas de decisión:** Se utiliza para identificar las posibles condiciones de entrada y las posibles acciones de salida. Se aplica cuando hay reglas de negocio complejas.

Ahora vamos a enumerar los diferentes métodos probados dividiéndolo por las diferentes clases:

### Shot

* private void initShot(int x, int y)
  + Prueba de valor límite, aquí hemos decidido probar por parámetro que pasa si ingresaban valores al límite de lo permitido para ver el error.
  + Prueba de valor especial, aquí hemos probado a ingresar valores negativos para ver el manejo de errores.

### Alien

* public Alien(int x, int y)
  + Prueba de valor límite, aquí hemos decidido probar por parámetro que pasa si ingresaban valores al límite de lo permitido para ver el error.
  + Prueba de valor especial, aquí hemos probado a ingresar valores negativos para ver el manejo de errores.

### Bomb

* public Bomb(int x, int y)
  + Prueba de valor límite, aquí hemos decidido probar por parámetro que pasa si ingresaban valores al límite de lo permitido para ver el error.
  + Prueba de valor especial, aquí hemos probado a ingresar valores negativos para ver el manejo de errores.

### Player

* public void act()
  + Clases de equivalencia, evalúa un caso donde el jugador se mueve dentro de los límites normales del tablero, probando un movimiento estándar
* public void keyPressed(KeyEvent e)///public void keyReleased(KeyEvent e)
  + Prueba de valor límite, aquí se verifica el comportamiento del jugador cuando se encuentra en los límites de movimiento permitidos

### Board

* private void gameInit()
  + Prueba de valor límite, aquí se verifica que el número inicial de alienígenas esté en el valor exacto requerido.
  + Clases de equivalencia, verifica una clase de casos válidos
* private void update()
  + Prueba de valor límite, aquí se verifica el comportamiento en el valor límite de las muertes de los alienígenas necesarias para ganar el juego.
  + Clases de equivalencia, este caso comprueba que las condiciones del juego son normales y no ha habido muertes aún.
* private void update\_shots()
  + Clases de equivalencia, aquí se comprueba la situación donde un disparo elimina un alienígena con un comportamiento esperado.
  + Prueba de valores especiales, La posición exacta del disparo es crítica y se tiene que tener en cuenta para detectar las colisiones con los alienígenas
* private void update\_bomb()
  + Prueba de valores especiales, Las coordenadas exactas de la bomba coincidiendo con las del jugador se deben mirar con cuidado para saber si le da o no al jugador.
  + Clases de equivalencia, se comprueba la colisión entre un jugador y un ataque del enemigo produce la muerte.
* private void update\_aliens()
  + Clases de equivalencia, se comprueba la condición en la que los alienígenas invaden.

## Conjunto de casos de prueba diseñados, entradas a probar salidas esperadas

Los casos de prueba diseñados vamos a explicarlos por cada una de las clases que hemos testeado:

### Shot

* private void initShot(int x, int y)
  + Se prueba el correcto funcionamiento de crear un disparo, se le asignen las correspondientes coordenadas, se introduce x=100 e y=200 y se espera que la salida sea x=100+6 e y=200-1 para comprobar que la altura y anchura estén correspondidas
  + Se prueba el que pasa cuando se ingresan coordenadas out of bounds, esperando que se reasigne este valor se introduce el valor x=anchura máxima del tablero +100 e y=altura máxima del tablero + 200, se espera salida de x= anchura máxima del tablero e y=altura máxima del tablero
  + Se prueba que pasa si se asignan valores negativos a las coordenadas, se introduce x=-100 e y=-200 y se espera que se reasignen las coordenadas esperándose los valores x=-100+6 e y=-200-1

### Alien

* public Alien(int x, int y)
  + Se prueba el correcto funcionamiento de crear un disparo, se le asignen las correspondientes coordenadas, se introduce x=50 e y=100 y se espera que la salida sea x=50 e y=100para comprobar que la altura y anchura estén correspondidas
  + Se prueba el que pasa cuando se ingresan coordenadas out of bounds, esperando que se reasigne este valor se introduce el valor x=anchura máxima del tablero +10 e y=altura máxima del tablero + 10, se espera salida de x= anchura máxima del tablero e y=altura máxima del tablero
  + Se prueba que pasa si se asignan valores negativos a las coordenadas, se introduce x=-20 e y=-50 y se espera que se reasignen las coordenadas esperándose los valores x=0 e y=0

### Bomb

* public Bomb(int x, int y)
  + Se prueba el correcto funcionamiento de crear un disparo, se le asignen las correspondientes coordenadas, se introduce x=50+10 e y=290-10 y se espera que la salida sea x=50+10 e y=290-10 para comprobar que la altura y anchura estén correspondidas
  + Se prueba el que pasa cuando se ingresan coordenadas out of bounds, esperando que se reasigne este valor, se introduce el valor x=anchura máxima del tablero +10 e y=altura máxima del tablero - 10, se espera salida de x= anchura máxima del tablero e y=altura máxima del tablero-10
  + Se prueba que pasa si se asignan valores negativos a las coordenadas, se introduce x=-10 e y=-20 y se espera que se reasignen las coordenadas esperándose los valores x=5 e y=0

### Player

* public void act()
  + Se prueba el correcto movimiento del jugador dentro del tablero, impidiendo moverse si alcanza el final de este mismo. Para ello se introduce la tecla presionada tanto para la derecha como para la izquierda para ver como cambian sus coordenadas al llegar al límite, se espera que al llegar al límite esas coordenadas no sobrepasen el límite
  + Se comprueba el correcto funcionamiento del cambio de coordenadas para moverse por el tablero, para esto se presiona la tecla derecha e izquierda para comprobar si estas coordenadas cambian, se espera que, si se ha tocado alguna tecla, las coordenadas del jugador hayan cambiado.
* public void keyPressed(KeyEvent e)///public void keyReleased(KeyEvent e)
  + Estos métodos han sido probados a la vez junto con el método act(); ya que de manera aislada no creemos correcto que sea factible probar ya que usan librerías internas de java.

### Board

* private void gameInit()
  + Se comprueba que el juego se ha iniciado correctamente, se llama al constructor de board, que inicializa el juego y se espera que todos los valores estén correctos, isIngame=true, que exista alienes, disparos y jugadores y que el número de aliens sea el mismo que el número de aliens a destruir.
* private void update()
  + Se comprueba que la condición de ganar sea correcta, se eliminan todos los alienígenas igualando el número de muertes al número total de alienígenas, si esto pasa se espera que la condición de ganar ocurra.
  + También se comprueba que, en caso de que no se elimine a ningún alienígena, el valor isInGame=true.
* private void update\_shots()
  + Se comprueba si un alien y un disparo coinciden, se introduce un alien con posición x=a e y=b así como un disparo con coordenadas x=a e y=b, se espera que el alien se destruya y muera.
* private void update\_bomb()
  + Se comprueba que pasa cuando una bomba alcanza al jugador, en este caso se introduce una bomba con posición x=a e y=b, además de poner al jugador en las coodernada x=a e y=b, se espera que el jugador cambie su estado de isDying=true.
* private void update\_aliens()

## Las implementaciones en Junit 5 de los casos de prueba definidos

### Shot

package main;

import org.junit.jupiter.api.Test;

import space\_invaders.sprites.Shot;

import static org.junit.jupiter.api.Assertions.\*;

class ShotTest {

  @Test

  void testInitShotWithinBounds() {

    Shot shot = new Shot(100, 200);

    assertEquals(100 + 6, shot.getX()); // 100 + H\_SPACE

    assertEquals(200 - 1, shot.getY()); // 200 - V\_SPACE

  }

  @Test

  void testInitShotOutOfBounds() {

    Shot shot = new Shot(Commons.BOARD\_WIDTH + 100, Commons.BOARD\_HEIGHT + 200);

    assertEquals(Commons.BOARD\_WIDTH + 100 + 6, shot.getX());

    assertEquals(Commons.BOARD\_HEIGHT + 200 - 1, shot.getY());

  }

  @Test

  void testInitShotNegativeCoordinates() {

    Shot shot = new Shot(-100, -200);

    assertEquals(-100 + 6, shot.getX());

    assertEquals(-200 - 1, shot.getY());

  }

}

### Alien

package main;

import org.junit.jupiter.api.Test;

import space\_invaders.sprites.Alien;

import static org.junit.jupiter.api.Assertions.\*;

class AlienTest {

  @Test

  void testInitAlienWithinBounds() {

    Alien alien = new Alien(50, 100);

    assertEquals(50, alien.getX());

    assertEquals(100, alien.getY());

  }

  @Test

  void testInitAlienOutOfBounds() {

    Alien alien = new Alien(Commons.BOARD\_WIDTH + 10, Commons.BOARD\_HEIGHT + 10);

    assertEquals(Commons.BOARD\_WIDTH, alien.getX());

    assertEquals(Commons.BOARD\_HEIGHT, alien.getY());

  }

  @Test

  void testInitAlienNegativeCoordinates() {

    Alien alien = new Alien(-20, -50);

    assertEquals(0, alien.getX());

    assertEquals(0, alien.getY());

  }

}

### Bomb

package main;

import org.junit.jupiter.api.Test;

import space\_invaders.sprites.Alien;

import space\_invaders.sprites.Alien.Bomb;

import static org.junit.jupiter.api.Assertions.\*;

class BombTest {

  @Test

  void testBombInitializationWithinBounds() {

    Alien alien = new Alien(Commons.BORDER\_LEFT + 50, Commons.GROUND - 10);

    Bomb bomb = alien.getBomb();

    assertEquals(Commons.BORDER\_LEFT + 50, bomb.getX());

    assertEquals(Commons.GROUND - 10, bomb.getY());

    assertTrue(bomb.isDestroyed());

  }

  @Test

  void testBombInitializationOutOfBoundsRight() {

    Alien alien = new Alien(Commons.BOARD\_WIDTH + 10, Commons.GROUND - 10);

    Bomb bomb = alien.getBomb();

    assertEquals(Commons.BOARD\_WIDTH, bomb.getX());

    assertEquals(Commons.GROUND - 10, bomb.getY());

  }

  @Test

  void testBombInitializationNegativeCoordinates() {

    Alien alien = new Alien(-10, -20);

    Bomb bomb = alien.getBomb();

    assertEquals(Commons.BORDER\_LEFT, bomb.getX());

    assertEquals(0, bomb.getY());

  }

}

### Player

package main;

import org.junit.jupiter.api.Test;

import space\_invaders.sprites.Player;

import java.awt.event.KeyEvent;

import javax.swing.JPanel;

import static org.junit.jupiter.api.Assertions.\*;

class PlayerTest {

    @Test

    void testPlayerMoveRightWithinBounds() {

        var mockPanel = new JPanel();

        Player player = new Player();

        player.setX(Commons.BORDER\_LEFT + 50);

        KeyEvent keyEventRight = new KeyEvent(mockPanel, KeyEvent.KEY\_PRESSED, System.currentTimeMillis(), 0,

                KeyEvent.VK\_RIGHT, KeyEvent.CHAR\_UNDEFINED);

        player.keyPressed(keyEventRight);

        player.act();

        player.keyReleased(keyEventRight);

        assertEquals(Commons.BORDER\_LEFT + 52, player.getX());

    }

    @Test

    void testPlayerMoveLeftWithinBounds() {

        var mockPanel = new JPanel();

        Player player = new Player();

        player.setX(Commons.BORDER\_RIGHT - 50);

        KeyEvent keyEventLeft = new KeyEvent(mockPanel, KeyEvent.KEY\_PRESSED, System.currentTimeMillis(), 0,

                KeyEvent.VK\_LEFT, KeyEvent.CHAR\_UNDEFINED);

        player.keyPressed(keyEventLeft);

        player.act();

        player.keyReleased(keyEventLeft);

        assertEquals(Commons.BORDER\_RIGHT - 48, player.getX());

    }

    @Test

    void testPlayerMoveLeftBoundary() {

        var mockPanel = new JPanel();

        Player player = new Player();

        player.setX(Commons.BORDER\_LEFT);

        KeyEvent keyEventLeft = new KeyEvent(mockPanel, KeyEvent.KEY\_PRESSED, System.currentTimeMillis(), 0,

                KeyEvent.VK\_LEFT, KeyEvent.CHAR\_UNDEFINED);

        player.keyPressed(keyEventLeft);

        player.act();

        player.keyReleased(keyEventLeft);

        assertEquals(Commons.BORDER\_LEFT, player.getX());

    }

    @Test

    void testPlayerMoveRightBoundary() {

        var mockPanel = new JPanel();

        Player player = new Player();

        player.setX(Commons.BOARD\_WIDTH - Commons.BORDER\_RIGHT - Commons.PLAYER\_WIDTH);

        KeyEvent keyEventRight = new KeyEvent(mockPanel, KeyEvent.KEY\_PRESSED, System.currentTimeMillis(), 0,

                KeyEvent.VK\_RIGHT, KeyEvent.CHAR\_UNDEFINED);

        player.keyPressed(keyEventRight);

        player.act();

        player.keyReleased(keyEventRight);

        assertEquals(Commons.BOARD\_WIDTH - Commons.BORDER\_RIGHT - Commons.PLAYER\_WIDTH, player.getX());

    }

}

### Board

package main;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import space\_invaders.sprites.\*;

import static org.junit.jupiter.api.Assertions.\*;

class BoardTest {

  private Board board;

  @BeforeEach

  void setUp() {

    board = new Board();

  }

  @Test

  void testGameInit() {

    assertNotNull(board);

    assertTrue(board.isInGame());

    assertNotNull(board.getAliens());

    assertNotNull(board.getPlayer());

    assertNotNull(board.getShot());

    assertEquals(board.getAliens().size(), Commons.NUMBER\_OF\_ALIENS\_TO\_DESTROY);

  }

  @Test

  void testGameWon() throws InterruptedException {

    board.setDeaths(Commons.NUMBER\_OF\_ALIENS\_TO\_DESTROY);

    Thread.sleep(Commons.DELAY);

    assertFalse(board.isInGame());

    assertEquals("Game won!", board.getMessage());

  }

  @Test

  void testGameContinues() {

    board.setDeaths(0);

    assertTrue(board.isInGame());

  }

  @Test

  void testShotDestroysAlien() throws InterruptedException {

    board.getTimer().stop();

    var alien = board.getAliens().get(0);

    var shot = new Shot(alien.getX() - 6, alien.getY() + 1);

    board.setShot(shot);

    board.getTimer().start();

    Thread.sleep(Commons.DELAY);

    assertEquals(1, board.getDeaths());

  }

  @Test

  void testBombKillsPlayer() throws InterruptedException {

    board.getTimer().stop();

    Alien alien = board.getAliens().get(0);

    var bomb = alien.getBomb();

    var player = board.getPlayer();

    bomb.setX(player.getX());

    bomb.setY(player.getY());

    board.getTimer().start();

    Thread.sleep(Commons.DELAY);

    assertTrue(player.isDying());

  }

}

void testGameLostByInvasion() throws InterruptedException {

    board.getTimer().stop();

    var alien = board.getAliens().get(0);

    alien.setY(Commons.GROUND);

    board.getTimer().start();

    Thread.sleep(Commons.DELAY);

    assertFalse(board.isInGame());

    assertEquals("Invasion!", board.getMessage());

  }

## Los resultados obtenidos, casos de prueba NO han sido superados casos de prueba han sido superados.

qwñkehjfqñkwjbfñkqwbfñkeqjbfñkjeqnfñkqjernfñlj

### Shot

#### Casos han pasado exitosamente el test

* void testInitShotWithinBounds()
* void testInitShotOutOfBounds()
* testInitShotNegativeCoordinates()

#### Casos no han pasado el test

* Todos los test de esta clase han sido pasados con éxito

### Alien

#### Casos han pasado exitosamente el test

* void testInitAlienWithinBounds()
* void testInitAlienNegativeCoordinates()

#### Casos no han pasado el test

* void testInitAlienOutOfBounds()
  + Este test se esperaba obtener 358 y se ha obtenido 368

### Bomb

#### Casos han pasado exitosamente el test

* void testBombInitializationWithinBounds()

#### Casos no han pasado el test

* void testBombInitializationOutOfBoundsRight()
  + Este test se espera 5 y se ha obtenido -10
* void testBombInitializationNegativeCoordinates()
  + Este test se esperaba 280 pero se obtiene 350

### Player

#### Casos han pasado exitosamente el test

* void testPlayerMoveLeftWithinBounds()

#### Casos no han pasado el test

* void testPlayerMoveRightWithinBounds()
  + Este test se esperaba 57 pero es 2
* void testPlayerMoveLeftBoundary()
  + Este teste se esperaba 5 pero es 2
* void testPlayerMoveRightBoundary()
  + Este test se esperaba 313 pero es 2

### Board

#### Casos han pasado exitosamente el test

* void testGameInit()
* void testGameContinues()

#### Casos no han pasado el test

* void testGameWon()
  + Este test se esperaba Game Won pero es Invasion
* void testShotDestroysAlien()
  + Este test se esperaba 1 pero es 0
* void testBombKillsPlayer()
  + Este test se esperaba true, pero es false
* void testGameLostByInvasion()
  + Este test se esperaba un false, pero es true