

Informe Sprint 1

Ingeniería del Software



Mikel Abad

Andima Freire

Julen Mendiguren

Jon Ander González

26 de marzo de 2017

GRUPO 10 Abad y cia

Índice:

**Introducción2**

**Decisiones del juego3**

**Actas de reunión de grupo5**

Acta reunión 23 / 02 / 17. 5

Acta reunión 02 / 03 / 17. 6

Acta reunión 09 / 02 / 17. 7

Acta reunión 16 / 02 / 17. 9

Acta reunión 23 / 02 / 17. 9

**Reparto de tareas10**

**Diagrama de Clases11**

**Diagramas de Secuencia 12**

Diagrama de secuencia “colocarBarco” 12

Diagrama de secuencia “ponerEscudo” 13

**Casos de Prueba 14**

Usuario Test 14

Jugador Test 15

Lista Coordenadas Test 16

Coordenada Test 18

Battleship Test 19

Barco Test 20

Almacén Test 21

# Introducción**.**

En la realización del proyecto, el juego de hundir la flota, utilizaremos la metodología SCRUM. Como parte de esta metodología el proyecto estará dividido en “Historias de Usuario y Sprints.

En este primer Sprint abordaremos las siguientes historias de usuario:

Decisiones del juego:

El jugador una vez inicie el juego dispondrá de un tablero con 100 casillas, organizadas en 10 filas y 10 columnas donde situar cada barco. Tendrá que colocar un total de 10 barcos y podrá controlar su orientación con el botón de orientación. Los barcos serán:

* 4 fragatas de 1 casilla.
* 3 destructores de 2 casillas.
* 2 submarinos de 3 casillas.
* 1 portaviones de 4 casillas.

\*Para colocar los barcos tiene que respetar la distancia de una casilla entre los barcos.



Una vez colocados todos los barcos el usuario dispondrá de dos tableros, uno con sus barcos colocados y otro que mostrará el tablero del ordenador. También dispondrá de una tienda donde comprar armamento.

El dinero inicial que dispone para comprar es de 150 y podrá comprar el siguiente armamento en la tienda:

* Bomba. Gratuita y de uso ilimitado, que golpeará una única posición.
* Misil. De coste 10 y un máximo de 10 usos. El misil al golpear un barco sin escudo, destruirá el barco por completo.
* Misil Norte-Sur. Con un coste de 45 y un único uso en la partida. Destruirá todos los barcos sin escudo situados en la línea Norte-Sur, en caso de disponer de escudo se lo quitará.
* Misil Este-Oeste. Con un coste de 45 y un único uso en la partida. Destruirá todos los barcos sin escudo situados en la línea Este-Oeste, en caso de disponer de escudo se lo quitará.
* Misil BOOM. Con un coste de 90 y un único uso en la partida. Destruirá todos los barcos sin escudo situados en la línea Norte-Sur y Este-Oeste, en caso de disponer de escudo se lo quitará.
* Escudo de coste 5 y un máximo de 5 usos. Otorgará un escudo al barco entero. Si es alcanzado por un misil, el barco perderá su escudo, pero seguirá con vida.
* Reparación. Coste de 10 y un máximo de 5 usos. Reparará el barco entero siempre que el barco esté tocado. Si el barco es destruido el barco no podrá ser reparado.

|  |  |  |
| --- | --- | --- |
| **Armas** | **Cantidad** | **Precio** |
| Bomba | Ilimitada | 0 |
| Misil | 10 | 10 |
| Misil Norte-Sur | 1 | 45 |
| Misil Este-Oeste | 1 | 45 |
| Misil BOOM | 1 | 90 |
| Escudo | 5 | 25 |
| Reparación | 5 | 10 |

Actas de reunión del grupo:

## Acta reunión 23 / 02 / 17.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| blanco_medianoEUITI  INDUSTRIA INGENIARITZA TEKNIKOKO UNIBERTSITATE ESKOLA  ESCUELA UNIVERSITARIA DE INGENIERIA TECNICA INDUSTRIAL    ACTA DE REUNIÓN 1   |  |  |  |  | | --- | --- | --- | --- | | Fecha: *23 / 02 /2017* | Hora: *12:00* | Lugar: *Aula 3* | Duración: *3h* | | Trabajo:  *Ingeniería del software. Proyecto Hundir la flota.* | | | | | Personas Asistentes:  Mikel Abad, Andima Freire, Julen Mendiguren, Jon Ander González | | | | | **ASUNTOS TRATADOS**:  *Tareas de las HU 1 del primer sprint*  *Identificar clases necesarias* | | | | | **PRINCIPALES ACUERDOS ALCANZADOS**:  *Acordada forma de juego y estructuración de las pantallas.* | | | | | Fecha de la próxima reunión: *02 / 03 / 2017* | | | |   **FIRMAS** |

## Acta reunión 02 / 03 / 2017.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| blanco_medianoEUITI  INDUSTRIA INGENIARITZA TEKNIKOKO UNIBERTSITATE ESKOLA  ESCUELA UNIVERSITARIA DE INGENIERIA TECNICA INDUSTRIAL    ACTA DE REUNIÓN 2   |  |  |  |  | | --- | --- | --- | --- | | Fecha: *02 / 02 /2017* | Hora: *12:00* | Lugar: *Aula 3* | Duración: *3h* | | Trabajo:  *Ingeniería del software. Proyecto Hundir la flota.* | | | | | Personas Asistentes:  Mikel Abad, Andima Freire, Julen Mendiguren, Jon Ander González | | | | | **ASUNTOS TRATADOS**:  Diseñar clases necesarias | | | | | **PRINCIPALES ACUERDOS ALCANZADOS**:  Establecidos precios de armamento y numero de usos. | | | | | Fecha de la próxima reunión: *09 / 03 / 2017* | | | |   **FIRMAS** |

## Acta de reunión 09 / 03 / 2017

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| blanco_medianoEUITI  INDUSTRIA INGENIARITZA TEKNIKOKO UNIBERTSITATE ESKOLA  ESCUELA UNIVERSITARIA DE INGENIERIA TECNICA INDUSTRIAL    ACTA DE REUNIÓN 3   |  |  |  |  | | --- | --- | --- | --- | | Fecha: *09 / 03 /2017* | Hora: *12:00* | Lugar: *Aula 3* | Duración: *3h* | | Trabajo:  *Ingeniería del software. Proyecto Hundir la flota.* | | | | | Personas Asistentes:  Mikel Abad, Andima Freire, Julen Mendiguren, Jon Ander González | | | | | **ASUNTOS TRATADOS**:  Implementar y concretar clases | | | | | Fecha de la próxima reunión: *16 / 03 / 2017* | | | |   **FIRMAS** |

## Acta de reunión 16 / 03 / 2017.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| blanco_medianoEUITI  INDUSTRIA INGENIARITZA TEKNIKOKO UNIBERTSITATE ESKOLA  ESCUELA UNIVERSITARIA DE INGENIERIA TECNICA INDUSTRIAL    ACTA DE REUNIÓN 4   |  |  |  |  | | --- | --- | --- | --- | | Fecha: *16 / 03 /2017* | Hora: *12:00* | Lugar: *Aula 3* | Duración: *3h* | | Trabajo:  *Ingeniería del software. Proyecto Hundir la flota.* | | | | | Personas Asistentes:  Mikel Abad, Andima Freire, Julen Mendiguren, Jon Ander González | | | | | **ASUNTOS TRATADOS**:  *Implementar clases*  *Pruebas unitarias clases.* | | | | | Fecha de la próxima reunión: *23 / 03 / 2017* | | | |   **FIRMAS** |

## Acta de reunión 23 / 03 / 2017

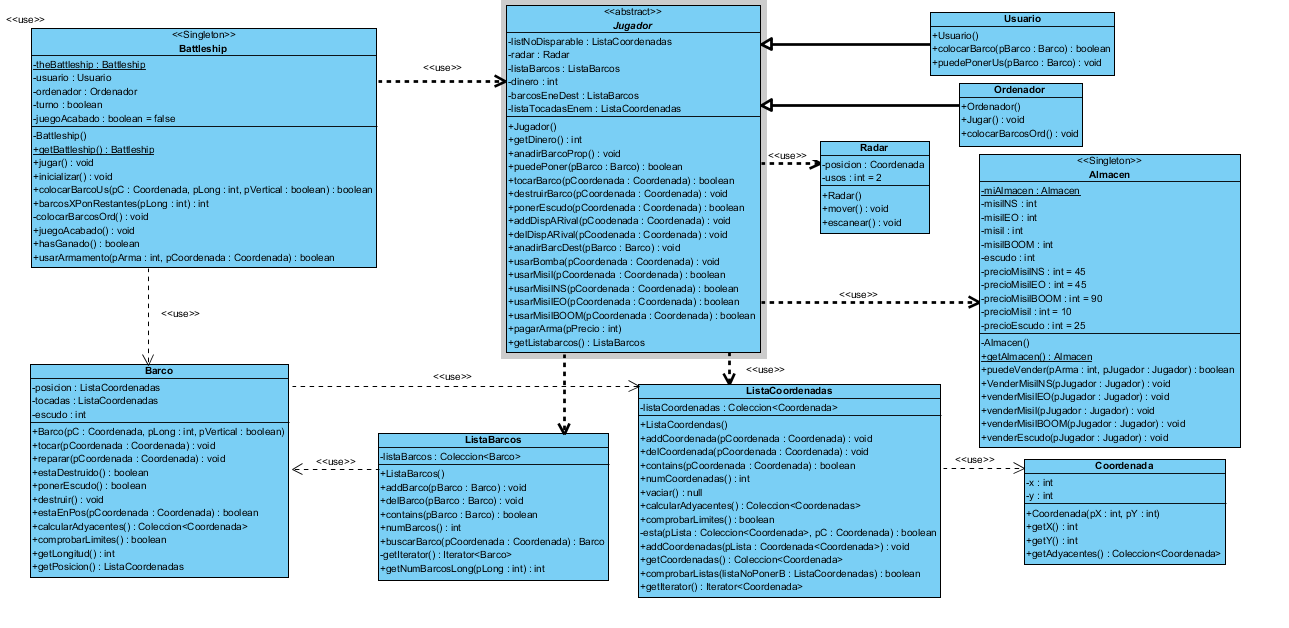
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| blanco_medianoEUITI  INDUSTRIA INGENIARITZA TEKNIKOKO UNIBERTSITATE ESKOLA  ESCUELA UNIVERSITARIA DE INGENIERIA TECNICA INDUSTRIAL    ACTA DE REUNIÓN 5   |  |  |  |  | | --- | --- | --- | --- | | Fecha: *23 / 03 /2017* | Hora: *12:00* | Lugar: *Aula 3* | Duración: *3h* | | Trabajo:  *Ingeniería del software. Proyecto Hundir la flota.* | | | | | Personas Asistentes:  Mikel Abad, Andima Freire, Julen Mendiguren, Jon Ander González | | | | | **ASUNTOS TRATADOS**:  *Solucionar conflictos implementación.* | | | | | Fecha de la próxima reunión: *30 / 03 / 2017* | | | |   **FIRMAS** |

Reparto de tareas:

* Documento con el reparto de tareas entre los miembros del grupo.

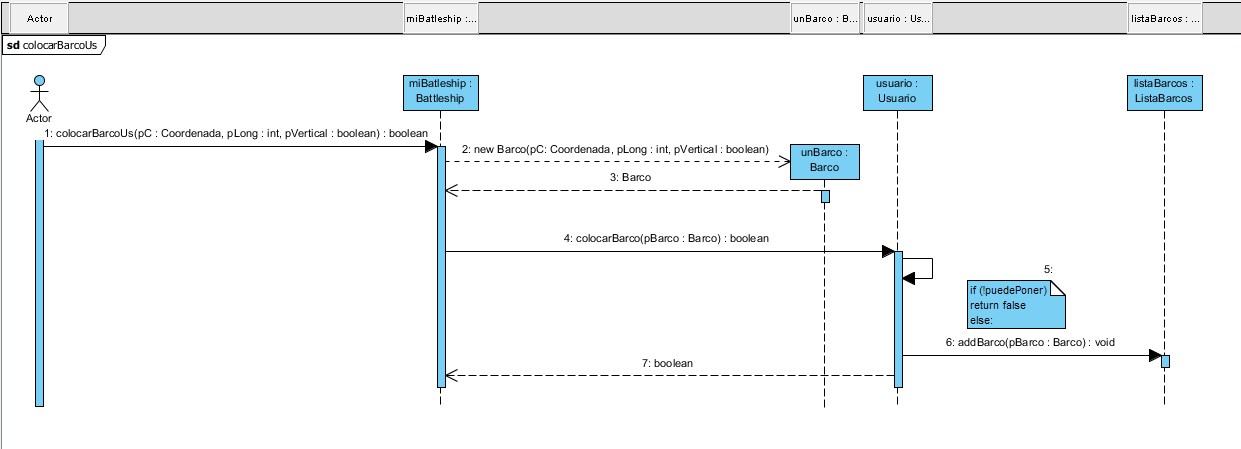
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sprint** | **Tarea** | **Responsable** | **Planif.** **Inicial** | **Planif. Real** | **Comentarios** |
| 1 | Identificar clases necesarias | Grupo | 1h | 1h |  |
| 1 | Diseñar clases necesarias | Grupo | 2h | 5h | Clases relacionadas con: |
| 1 | Implementar y concretar clases | Mikel Abad | 1.5h | 4h | Jugador/Ordenador |
| 1 | Implementar y concretar clases | Andima Freire | 1.5h | 4h | Barcos/Coordenadas |
| 1 | Implementar y concretar clases | Jon Ander González | 1.5h | 4h | Principal/Listas |
| 1 | Implementar y concretar clases | Julen Mendiguren | 1.5h | 4h | Armamento/Almacén |
| 1 | Solucionar conflictos implementación. | Grupo | 0.5h | 8h |  |
| 1 | Pruebas unitarias clases: | Mikel Abad | 2h | 2h | Barcos/Coordenadas |
| 1 | Pruebas unitarias clases: | Andima Freire | 2h | 2h | Jugador/Ordenador |
| 1 | Pruebas unitarias clases: | Jon Ander González | 2h | 2h | Armamento/Almacén |
| 1 | Pruebas unitarias clases: | Julen Mendiguren | 2h | 2h | Principal/Listas |
| 1 | Documentación del Sprint | Grupo | 5h | 5h |  |

Diagrama de clases:

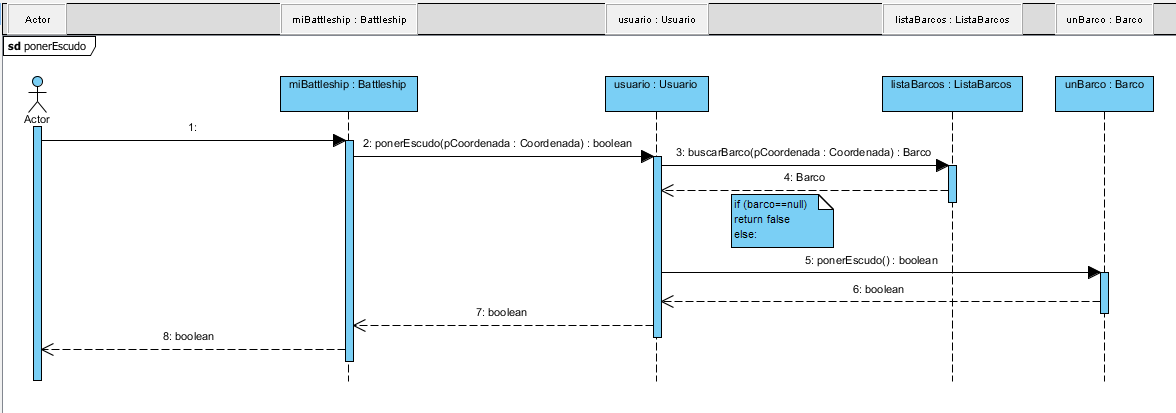


Diagramas de secuencias:

## Diagrama de secuencia “colocarBarco”.



## Diagrama de secuencia “ponerEscudo”:



Casos de prueba:

## UsuarioTest

|  |
| --- |
| **public** **class** UsuarioTest {    @Before  **public** **void** setUp() **throws** Exception {  Battleship.*getBattleship*().inicializar();  }  @Test  **public** **void** testPuedePonerUs() {  Usuario u = **new** Usuario();  Barco unBarco;    // Puede poner un barco de longitud 4 sin tener más barcos de 4  unBarco = **new** Portaaviones(**new** Coordenada(0,0), **true**);  *assertTrue*(u.puedePonerUs(unBarco));  u.colocarBarco(unBarco);  // No puede poner otro barco de longitud 4  unBarco = **new** Portaaviones(**new** Coordenada(2,0), **false**);  *assertFalse*(u.puedePonerUs(unBarco));    u = **new** Usuario();    // Puede poner dos barcos de longitud 3  unBarco = **new** Submarino(**new** Coordenada(4,2), **true**);  *assertTrue*(u.puedePonerUs(unBarco));  u.colocarBarco(unBarco);  unBarco = **new** Submarino(**new** Coordenada(6,2), **true**);  *assertTrue*(u.puedePonerUs(unBarco));  u.colocarBarco(unBarco);  // No puede poner otro de tamaño 3  unBarco = **new** Submarino(**new** Coordenada(0,0), **false**);  *assertFalse*(u.puedePonerUs(unBarco));    u = **new** Usuario();    // Puede poner tres barcos de longitud 2  unBarco = **new** Destructor(**new** Coordenada(0,0), **true**);  *assertTrue*(u.puedePonerUs(unBarco));  u.colocarBarco(unBarco);  unBarco = **new** Destructor(**new** Coordenada(2,0), **true**);  *assertTrue*(u.puedePonerUs(unBarco));  u.colocarBarco(unBarco);  unBarco = **new** Destructor(**new** Coordenada(4,0), **true**);  *assertTrue*(u.puedePonerUs(unBarco));  u.colocarBarco(unBarco);  // No puede poner otro de tamaño 2  unBarco = **new** Destructor(**new** Coordenada(6,0), **true**);  *assertFalse*(u.puedePonerUs(unBarco));    u = **new** Usuario();    // Puede poner cuatro barcos de longitud 1  unBarco = **new** Fragata(**new** Coordenada(0,0));  *assertTrue*(u.puedePonerUs(unBarco));  u.colocarBarco(unBarco);  unBarco = **new** Fragata(**new** Coordenada(2,0));  *assertTrue*(u.puedePonerUs(unBarco));  u.colocarBarco(unBarco);  unBarco = **new** Fragata(**new** Coordenada(4,0));  *assertTrue*(u.puedePonerUs(unBarco));  u.colocarBarco(unBarco);  unBarco = **new** Fragata(**new** Coordenada(6,0));  *assertTrue*(u.puedePonerUs(unBarco));  u.colocarBarco(unBarco);  // No puede poner otro de tamaño 1  unBarco = **new** Fragata(**new** Coordenada(8,0));  *assertFalse*(u.puedePonerUs(unBarco));  }  } |

## Test Jugador

|  |
| --- |
| **public** **class** TestJugador {  @BeforeClass  **public** **static** **void** setUpBeforeClass() **throws** Exception {  Battleship.*getBattleship*().inicializar();  }  @AfterClass  **public** **static** **void** tearDownAfterClass() **throws** Exception {  }  @Test  **public** **void** testPuedePoner() {  Usuario u = **new** Usuario();  Barco b = **new** Portaaviones(**new** Coordenada(0,0), **false**);  u.puedePoner(b);  u.colocarBarco(b);  Barco b2 = **new** Submarino(**new** Coordenada(0,0), **true**);  *assertFalse*(u.puedePoner(b2));  b2=**new** Submarino (**new** Coordenada(1,1),**true**);  *assertFalse*(u.puedePoner(b2));  b2=**new** Fragata(**new** Coordenada(9,9));  *assertTrue*(u.puedePoner(b2));  b2=**new** Submarino (**new** Coordenada(8,8),**true**);  *assertFalse*(u.puedePoner(b2));    Barco b4=**new** Submarino (**new** Coordenada(5,5), **true**);  *assertTrue*(u.puedePoner(b4));  u.puedePoner(b4);  u.colocarBarco(b4);    Barco b3 = **new** Submarino(**new** Coordenada(3,6), **false**);  *assertFalse*(u.puedePoner(b3));      }  } |

## Lista Coordenadas Test

|  |
| --- |
| **public** **class** ListaCoordenadasTest {  @Before  **public** **void** setUp() **throws** Exception {  }  @After  **public** **void** tearDown() **throws** Exception {  }    @Test  **public** **void** testEstaEnLista() {  Coordenada c=**new** Coordenada(3,3);  ListaCoordenadas l=**new** ListaCoordenadas();  *assertFalse*(l.getCoordenadas().contains(c));  l.addCoordenada(c);  *assertTrue*(l.getCoordenadas().contains(c));  }    @Test  **public** **void** testCalcularAdyacentes() {  ListaCoordenadas l=**new** ListaCoordenadas();  ArrayList<Coordenada> adyacentes=**new** ArrayList<Coordenada>();  ArrayList<Coordenada> misAdyacentes=**new** ArrayList<Coordenada>();  adyacentes=l.calcularAdyacentes();  *assertEquals*(adyacentes.size(),0);    Coordenada c=**new** Coordenada(5,5);  l.addCoordenada(c);  adyacentes=l.calcularAdyacentes();    misAdyacentes.add(c);  misAdyacentes.add(**new** Coordenada(5,4));  misAdyacentes.add(**new** Coordenada(5,6));  misAdyacentes.add(**new** Coordenada(4,5));  misAdyacentes.add(**new** Coordenada(6,5));  misAdyacentes.add(**new** Coordenada(4,4));  misAdyacentes.add(**new** Coordenada(4,6));  misAdyacentes.add(**new** Coordenada(6,4));  misAdyacentes.add(**new** Coordenada(6,6));    **int** cont = 0;  // Miramos que ambos arraylist contienen las mismas coordenadas  **for** (**int** i = 0; i < adyacentes.size(); i++) {  **for** (**int** j = 0; j < adyacentes.size(); j++) {  **if** (( adyacentes.get(i).getX()== misAdyacentes.get(j).getX()) && adyacentes.get(i).getY()==misAdyacentes.get(j).getY()) {  cont++;  }  }  }  *assertEquals*(cont, adyacentes.size());  Coordenada c2=**new** Coordenada(5,7);  l.addCoordenada(c2);  adyacentes=l.calcularAdyacentes();  misAdyacentes.add(c2);  misAdyacentes.add(**new** Coordenada (5,7));  misAdyacentes.add(**new** Coordenada (4,7));  misAdyacentes.add(**new** Coordenada (6,7));  misAdyacentes.add(**new** Coordenada (4,8));  misAdyacentes.add(**new** Coordenada (5,8));  misAdyacentes.add(**new** Coordenada (6,8));    cont = 0;  // Miramos que ambos arraylist contienen las mismas coordenadas  **for** (**int** i = 0; i < adyacentes.size(); i++) {  **for** (**int** j = 0; j < adyacentes.size(); j++) {  **if** ((adyacentes.get(i).getX() == misAdyacentes.get(j).getX())&& adyacentes.get(i).getY() == misAdyacentes.get(j).getY()) {  cont++;  }  }  }  *assertEquals*(cont, adyacentes.size());    Coordenada c3=**new** Coordenada(5,6);  l.addCoordenada(c3);  adyacentes=l.calcularAdyacentes();    cont = 0;  // Miramos que ambos arraylist contienen las mismas coordenadas  **for** (**int** i = 0; i < adyacentes.size(); i++) {  **for** (**int** j = 0; j < adyacentes.size(); j++) {  **if** ((adyacentes.get(i).getX() == misAdyacentes.get(j).getX()) && adyacentes.get(i).getY() == misAdyacentes.get(j).getY()) {  cont++;  }  }  }  *assertEquals*(cont, adyacentes.size());    }  @Test  **public** **void** testFueraDeLimites() {  ListaCoordenadas l=**new** ListaCoordenadas();  *assertFalse*(l.fueraDeLimites());  Coordenada c=**new** Coordenada(3,3);  l.addCoordenada(c);  *assertFalse*(l.fueraDeLimites());  Coordenada c2=**new** Coordenada(9,0);  l.addCoordenada(c2);  *assertFalse*(l.fueraDeLimites());  Coordenada c3=**new** Coordenada(32,-3);  l.addCoordenada(c3);  *assertTrue*(l.fueraDeLimites());  Coordenada c4=**new** Coordenada(-1,-13);  l.addCoordenada(c4);  *assertTrue*(l.fueraDeLimites());  }  @Test  **public** **void** testComprobarListas() {  ListaCoordenadas l=**new** ListaCoordenadas();  ListaCoordenadas l2=**new** ListaCoordenadas();  Coordenada c=**new** Coordenada(3,3);  Coordenada c2=**new** Coordenada(9,0);  Coordenada c3=**new** Coordenada(4,5);  Coordenada c4=**new** Coordenada(2,7);  *assertFalse*(l.comprobarListas(l2));  l.addCoordenada(c);  *assertFalse*(l.comprobarListas(l2));  l.addCoordenada(c2);  l2.addCoordenada(c3);  l2.addCoordenada(c4);  *assertFalse*(l.comprobarListas(l2));  l.addCoordenada(c3);  *assertTrue*(l.comprobarListas(l2));  *assertTrue*(l2.comprobarListas(l));    }  } |

## Coordenada Test

|  |
| --- |
| **public** **class** CoordenadaTest {  @Before  **public** **void** setUp() **throws** Exception {  }  @After  **public** **void** tearDown() **throws** Exception {  }  @Test  **public** **void** testGetAdyacentes() {  Coordenada c=**new** Coordenada(3,3);  ArrayList<Coordenada> adyacentes=c.getAdyacentes();  ArrayList<Coordenada> adyacentes2=**new** ArrayList<Coordenada>();  adyacentes2.add(**new** Coordenada(3,3));  adyacentes2.add(**new** Coordenada(2,3));  adyacentes2.add(**new** Coordenada(4,3));  adyacentes2.add(**new** Coordenada(2,2));  adyacentes2.add(**new** Coordenada(3,2));  adyacentes2.add(**new** Coordenada(4,2));  adyacentes2.add(**new** Coordenada(2,4));  adyacentes2.add(**new** Coordenada(3,4));  adyacentes2.add(**new** Coordenada(4,4));    *assertEquals*(adyacentes.get(0).getX(),adyacentes2.get(0).getX());  *assertEquals*(adyacentes.get(0).getY(),adyacentes2.get(0).getY());  *assertEquals*(adyacentes.get(1).getX(),adyacentes2.get(1).getX());  *assertEquals*(adyacentes.get(1).getY(),adyacentes2.get(1).getY());  *assertEquals*(adyacentes.get(2).getX(),adyacentes2.get(2).getX());  *assertEquals*(adyacentes.get(2).getY(),adyacentes2.get(2).getY());  *assertEquals*(adyacentes.get(3).getX(),adyacentes2.get(3).getX());  *assertEquals*(adyacentes.get(3).getY(),adyacentes2.get(3).getY());  *assertEquals*(adyacentes.get(4).getX(),adyacentes2.get(4).getX());  *assertEquals*(adyacentes.get(4).getY(),adyacentes2.get(4).getY());  *assertEquals*(adyacentes.get(5).getX(),adyacentes2.get(5).getX());  *assertEquals*(adyacentes.get(5).getY(),adyacentes2.get(5).getY());  *assertEquals*(adyacentes.get(6).getX(),adyacentes2.get(6).getX());  *assertEquals*(adyacentes.get(6).getY(),adyacentes2.get(6).getY());  *assertEquals*(adyacentes.get(7).getX(),adyacentes2.get(7).getX());  *assertEquals*(adyacentes.get(7).getY(),adyacentes2.get(7).getY());  *assertEquals*(adyacentes.get(8).getX(),adyacentes2.get(8).getX());  *assertEquals*(adyacentes.get(8).getY(),adyacentes2.get(8).getY());    } |

## Battleship Test

|  |
| --- |
| **public** **class** BattleshipTest {  @Before  **public** **void** setUp() **throws** Exception {  Battleship.*getBattleship*().inicializar();  }  @After  **public** **void** tearDown() **throws** Exception {  }    @Test  **public** **void** testColocarBarcoUs() {    Battleship.*getBattleship*().colocarBarcoUs("Portaaviones", **new** Coordenada(0,0), **true**);  *assertTrue*(Battleship.*getBattleship*().hayBarcoUsu(**new** Coordenada(0,0)));  *assertTrue*(Battleship.*getBattleship*().hayBarcoUsu(**new** Coordenada(0,1)));  *assertTrue*(Battleship.*getBattleship*().hayBarcoUsu(**new** Coordenada(0,2)));  *assertTrue*(Battleship.*getBattleship*().hayBarcoUsu(**new** Coordenada(0,3)));  *assertFalse*(Battleship.*getBattleship*().hayBarcoUsu(**new** Coordenada(0,4)));    Battleship.*getBattleship*().colocarBarcoUs("Submarino", **new** Coordenada(4,4), **false**);  *assertTrue*(Battleship.*getBattleship*().hayBarcoUsu(**new** Coordenada(4,4)));  *assertTrue*(Battleship.*getBattleship*().hayBarcoUsu(**new** Coordenada(5,4)));  *assertTrue*(Battleship.*getBattleship*().hayBarcoUsu(**new** Coordenada(6,4)));  *assertFalse*(Battleship.*getBattleship*().hayBarcoUsu(**new** Coordenada(3,4)));  *assertFalse*(Battleship.*getBattleship*().hayBarcoUsu(**new** Coordenada(7,4)));    Battleship.*getBattleship*().colocarBarcoUs("Destructor", **new** Coordenada(6,6), **false**);  *assertTrue*(Battleship.*getBattleship*().hayBarcoUsu(**new** Coordenada(6,6)));  *assertTrue*(Battleship.*getBattleship*().hayBarcoUsu(**new** Coordenada(7,6)));  *assertFalse*(Battleship.*getBattleship*().hayBarcoUsu(**new** Coordenada(5,6)));  *assertFalse*(Battleship.*getBattleship*().hayBarcoUsu(**new** Coordenada(8,6)));    Battleship.*getBattleship*().colocarBarcoUs("Fragata", **new** Coordenada(9,9), **true**);  *assertTrue*(Battleship.*getBattleship*().hayBarcoUsu(**new** Coordenada(9,9)));  }  @Test  **public** **void** testPuedeColocar() {  *assertTrue*(Battleship.*getBattleship*().puedeColocar("Portaaviones", **new** Coordenada(0,0), **true**));  Battleship.*getBattleship*().colocarBarcoUs("Portaaviones", **new** Coordenada(0,0), **true**);  *assertFalse*(Battleship.*getBattleship*().puedeColocar("Fragata", **new** Coordenada(0,2),**false**));    *assertTrue*(Battleship.*getBattleship*().puedeColocar("Submarino", **new** Coordenada(4,4), **false**));  Battleship.*getBattleship*().colocarBarcoUs("Submarino", **new** Coordenada(4,4), **false**);    *assertTrue*(Battleship.*getBattleship*().puedeColocar("Destructor", **new** Coordenada(6,6), **false**));  Battleship.*getBattleship*().colocarBarcoUs("Destructor", **new** Coordenada(6,6), **false**);    *assertFalse*(Battleship.*getBattleship*().puedeColocar("Portaaviones", **new** Coordenada(5,0),**false**));    *assertTrue*(Battleship.*getBattleship*().puedeColocar("Fragata", **new** Coordenada(9,9), **true**));  Battleship.*getBattleship*().colocarBarcoUs("Fragata", **new** Coordenada(9,9), **true**);    }  @Test  **public** **void** testBarcosXPonRestantes() {  *assertEquals*(Battleship.*getBattleship*().barcosXPonRestantes("Portaaviones"),1);  Battleship.*getBattleship*().colocarBarcoUs("Portaaviones", **new** Coordenada(0,0), **true**);  *assertEquals*(Battleship.*getBattleship*().barcosXPonRestantes("Portaaviones"),0);    *assertEquals*(Battleship.*getBattleship*().barcosXPonRestantes("Submarino"),2);  Battleship.*getBattleship*().colocarBarcoUs("Submarino", **new** Coordenada(4,4), **false**);  *assertEquals*(Battleship.*getBattleship*().barcosXPonRestantes("Submarino"),1);    *assertEquals*(Battleship.*getBattleship*().barcosXPonRestantes("Destructor"),3);  Battleship.*getBattleship*().colocarBarcoUs("Destructor", **new** Coordenada(6,6), **false**);  *assertEquals*(Battleship.*getBattleship*().barcosXPonRestantes("Destructor"),2);    *assertEquals*(Battleship.*getBattleship*().barcosXPonRestantes("Fragata"),4);  Battleship.*getBattleship*().colocarBarcoUs("Fragata", **new** Coordenada(9,9), **true**);  *assertEquals*(Battleship.*getBattleship*().barcosXPonRestantes("Fragata"),3);  }  } |

## Barco Test

|  |
| --- |
| **public** **class** BarcoTest {  @BeforeClass  **public** **static** **void** setUpBeforeClass() **throws** Exception {  }  @AfterClass  **public** **static** **void** tearDownAfterClass() **throws** Exception {  }  @Test  **public** **void** testEstaEnPos() {  Barco b = **new** Portaaviones(**new** Coordenada(0, 0), **true**);  *assertTrue*(b.estaEnPos(**new** Coordenada(0, 0)));  *assertTrue*(b.estaEnPos(**new** Coordenada(0, 1)));  *assertTrue*(b.estaEnPos(**new** Coordenada(0, 2)));  *assertTrue*(b.estaEnPos(**new** Coordenada(0, 3)));  *assertFalse*(b.estaEnPos(**new** Coordenada(0, 4)));  *assertFalse*(b.estaEnPos(**new** Coordenada(0, -1)));  Barco b2 = **new** Submarino(**new** Coordenada(5, 7), **false**);  *assertTrue*(b2.estaEnPos(**new** Coordenada(5, 7)));  *assertTrue*(b2.estaEnPos(**new** Coordenada(6, 7)));  *assertTrue*(b2.estaEnPos(**new** Coordenada(7, 7)));  *assertFalse*(b2.estaEnPos(**new** Coordenada(8, 7)));  *assertFalse*(b2.estaEnPos(**new** Coordenada(4, 7)));  *assertFalse*(b2.estaEnPos(**new** Coordenada(5, 6)));  }  @Test  **public** **void** testFueraDeLimites() {  Barco b = **new** Portaaviones(**new** Coordenada(0, 0), **true**);  *assertFalse*(b.fueraDeLimites());  Barco b2 = **new** Portaaviones(**new** Coordenada(7, 0), **true**);  *assertFalse*(b2.fueraDeLimites());  Barco b3 = **new** Fragata(**new** Coordenada(10, 0));  *assertTrue*(b3.fueraDeLimites());  Barco b4 = **new** Submarino(**new** Coordenada(5, 9), **false**);  *assertFalse*(b4.fueraDeLimites());  Barco b5 = **new** Submarino(**new** Coordenada(5, 5), **false**);  *assertFalse*(b.fueraDeLimites());  Barco b6 = **new** Submarino(**new** Coordenada(-1, 5), **false**);  *assertTrue*(b6.fueraDeLimites());  }  @Test  **public** **void** testCalcularAdyacentes() {  Barco b = **new** Destructor(**new** Coordenada(0, 0), **true**);  ArrayList<Coordenada> adyacentes = b.calcularAdyacentes();  ArrayList<Coordenada> misAdyacentes = **new** ArrayList<Coordenada>();      // Barco  misAdyacentes.add(**new** Coordenada(0, 0));  misAdyacentes.add(**new** Coordenada(0, 1));  // Arriba y abajo  misAdyacentes.add(**new** Coordenada(0, -1));  misAdyacentes.add(**new** Coordenada(0, 2));  // Derecha  misAdyacentes.add(**new** Coordenada(1, -1));  misAdyacentes.add(**new** Coordenada(1, 0));  misAdyacentes.add(**new** Coordenada(1, 1));  misAdyacentes.add(**new** Coordenada(1, 2));  // Izquierda  misAdyacentes.add(**new** Coordenada(-1, -1));  misAdyacentes.add(**new** Coordenada(-1, 0));  misAdyacentes.add(**new** Coordenada(-1, 1));  misAdyacentes.add(**new** Coordenada(-1, 2));  *assertEquals*(misAdyacentes.size(), adyacentes.size());  **int** cont = 0;  // Miramos que ambos arraylist contienen las mismas coordenadas  **for** (**int** i = 0; i < adyacentes.size(); i++) {  **for** (**int** j = 0; j < adyacentes.size(); j++) {  **if** ((adyacentes.get(i).getX() == misAdyacentes.get(j).getX())  &&adyacentes.get(i).getY()misAdyacentes.get(j).getY()) {  cont++;  }  }  }  *assertEquals*(cont, adyacentes.size());  }  } |

## Almacén Test

|  |
| --- |
| **public** **class** AlmacenTest {  @Before  **public** **void** setUp() **throws** Exception {  }  @After  **public** **void** tearDown() **throws** Exception {  }  @Test  **public** **void** testPuedeVender() {  Almacen a = Almacen.*getAlmacen*();  Usuario u = **new** Usuario();  Usuario u2 = **new** Usuario();  *assertTrue*(a.puedeVender(0, u));  a.venderEscudo(u);  *assertTrue*(a.puedeVender(0, u2));  a.venderEscudo(u2);  *assertTrue*(a.puedeVender(0, u));  a.venderEscudo(u);  *assertTrue*(a.puedeVender(0, u));  a.venderEscudo(u);  // Hasta aquÃ­ u deberia tener 25, u2 75, y 1 escudo en el almacen  *assertTrue*(a.puedeVender(0, u));  *assertTrue*(a.puedeVender(0, u2));  a.venderEscudo(u);  *assertFalse*(a.puedeVender(0, u));// Ni quedan escudos ni u tiene dinero  *assertFalse*(a.puedeVender(0, u2));//No quedan escudos  *assertFalse*(a.puedeVender(1, u));// u no tiene dinero  *assertTrue*(a.puedeVender(1, u2));//quedan misiles y u2 tiene 75 de dinero  *assertEquals*(u2.getDinero(),75);  *assertEquals*(u.getDinero(),0);  }  @Test  **public** **void** testVenderEscudo() {  Almacen a = Almacen.*getAlmacen*();  Usuario u = **new** Usuario();  a.venderEscudo(u);  *assertEquals*(u.getDinero(), 75);  }  @Test  **public** **void** testVenderMisil() {  Almacen a = Almacen.*getAlmacen*();  Usuario u = **new** Usuario();  a.venderMisil(u);  *assertEquals*(u.getDinero(), 90);  a.venderMisil(u);  *assertEquals*(u.getDinero(), 80);  }  @Test  **public** **void** testVenderMisilNS() {  Almacen a = Almacen.*getAlmacen*();  Usuario u = **new** Usuario();  *assertEquals*(u.getDinero(), 100);  *assertTrue*(a.puedeVender(2, u));  a.venderMisilNS(u);  *assertEquals*(u.getDinero(), 55);  *assertFalse*(a.puedeVender(2, u));  }  @Test  **public** **void** testVenderMisilEO() {  Almacen a = Almacen.*getAlmacen*();  Usuario u = **new** Usuario();  *assertEquals*(u.getDinero(), 100);  *assertTrue*(a.puedeVender(3, u));  a.venderMisilEO(u);  *assertEquals*(u.getDinero(), 55);  *assertFalse*(a.puedeVender(3, u));  }  @Test  **public** **void** testVenderMisilBOOM() {  Almacen a = Almacen.*getAlmacen*();  Usuario u = **new** Usuario();  *assertEquals*(u.getDinero(), 100);  *assertTrue*(a.puedeVender(4, u));  a.venderMisilBOOM(u);  *assertEquals*(u.getDinero(), 10);  *assertFalse*(a.puedeVender(4, u));  *assertTrue*(a.puedeVender(1, u));  }  } |