Descripción breve

Documento de explicación de las pruebas realizadas al software desarrollado.

PRUEBAS

Videojuego de Combate

Contenido

[2. Introducción 3](#_Toc1642306988)

[3. App 4](#_Toc1558429872)

[3.1. Método “loadRanking” 5](#_Toc1615652773)

[3.2. Método “saveRanking” 5](#_Toc933077558)

[3.3. Método “loadEquipos” 5](#_Toc118789831)

[3.4. Método “saveEquipos” 6](#_Toc2118942672)

[3.5. Método “loadGenerador” 6](#_Toc471732809)

[3.6. Método “saveGenerador” 6](#_Toc1436403594)

[3.7. Método “loadHabilidades” 6](#_Toc1766643188)

[3.8. Método “saveHabilidades” 7](#_Toc989658883)

[3.9. Método “run” 7](#_Toc1902147474)

[3.9.1. Pruebas no exitosas 8](#_Toc969617079)

[4. Personaje 8](#_Toc980675663)

[4.1. Método AnadirItem 9](#_Toc2015211888)

[4.1.1. Pruebas no exitosas 9](#_Toc1524325588)

[4.2. Método ModificarOro 9](#_Toc1069732323)

[4.3. Método PonerArmaActiva 10](#_Toc770656354)

[4.4. Método OroSuficiente 10](#_Toc1023730922)

[4.5. Método AñadirModificador 11](#_Toc764532796)

[4.6. Método AñadirEsbirro 11](#_Toc495219558)

[4.7. Método CalcularSaludEsbirros 12](#_Toc1832143815)

[4.8. Método CalcularAtaqueEquipo 12](#_Toc1053046368)

[4.9. Método CalcularModificadoresAtaque 13](#_Toc1326850226)

[4.10. Método CalcularModificadoresDefensa 13](#_Toc1232702537)

[4.10.1. Pruebas no exitosas 14](#_Toc1427597206)

[5. NombreClase2 14](#_Toc1507214631)

[5.1. Método nombreMétodo1 15](#_Toc527926805)

[5.1.1. Pruebas no exitosas 15](#_Toc2021058917)

[5.2. Método nombreMétodo2 15](#_Toc1410497119)

[5.2.1. Pruebas no exitosas 16](#_Toc318506840)

Tabla de ilustraciones

Ilustración 1 Diagrama de clases NombreClase1 4

Ilustración 2 Diagrama de clases NombreClase2 6

# Introducción

En este documento realizaremos una revisión de los casos de prueba realizados en la práctica de la creación de un videojuego para la empresa *MetProg Urjc S.L.*

Para el análisis, lo realizaremos clase por clase, centrándonos en los Test que hemos realizado en cada clase.

De cada Test, explicaremos en qué consiste en si mismo, así como los resultados obtenidos.

# App

Tabla

Descripción generada automáticamente

Ilustración 1 Diagrama de la clase App

Para la clase “App” hemos diseñado diferentes test, cada uno de ellos focalizado en probar/chequear el funcionamiento de cada método.

Como todos ellos necesitan la instantación de app, lo primero que haremos será crear un objeto de la clase app, mediante el comando @Before.

## Método “loadRanking”

Comprobamos que el archivo “ranking.ser” existe y se puede acceder a él.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Entrada | Condiciones de ejecución | Respuesta Esperada | Respuesta Obtenida | Comentarios |
| 1 | - | La clase App está instanciada | No devuelve excepción (el ranking existe o se ha creado correctamente) | No devuelve excepción | - |

## Método “saveRanking”

Verificamos que se guarda correctamente la estructura ranking en el archivo.

Para ello creamos un ranking, lo guardamos en el fichero y lo leemos para verificar que coinciden.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Entrada | Condiciones de ejecución | Respuesta Esperada | Respuesta Obtenida | Comentarios |
| 1 | Ranking | La clase App está instanciada y el archivo “ranking.ser” existe | NotNull | NotNull | - |

## Método “loadEquipos”

Comprobamos que el archivo “equipos.ser” existe y se puede acceder a él.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Entrada | Condiciones de ejecución | Respuesta Esperada | Respuesta Obtenida | Comentarios |
| 1 | - | La clase App está instanciada | NotNull | NotNull | - |

## Método “saveEquipos”

Comprobamos que se puede acceder con permiso de escritura al archivo “equipos.ser”.

Para ello, cargamos en una variable los equipos almacenados en el fichero, añadimos un nuevo equipo a la lista y guardamos los cambios en el fichero.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Entrada | Condiciones de ejecución | Respuesta Esperada | Respuesta Obtenida | Comentarios |
| 1 | Equipo | La clase App está instanciada y el archivo “equipos.ser” existe | No se produce excepción | No se produce excepción | En caso de que no se guardase correctamente, saltaría una excepción |

## Método “loadGenerador”

Comprobamos que el archivo “generador.ser” existe y se puede acceder a él.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Entrada | Condiciones de ejecución | Respuesta Esperada | Respuesta Obtenida | Comentarios |
| 1 | - | La clase App está instanciada | NotNull | NotNull | - |

## Método “saveGenerador”

Comprobamos que se puede acceder con permiso de escritura al archivo “generador.ser”.

Para ello, cargamos en una variable los generadores almacenados en el fichero (lectura), añadimos nuevos generadores a la lista y guardamos los cambios en el archivo (escritura).

A continuación leemos de nuevo el fichero (actualizado con los nuevos generadores) y los guardamos en otra variable. Verificamos que los cambios se han realizado correctamente, comprobando la extensión de los generadores actualizos en la variable (sin guardarse en el archivo) y los cargados del fichero. También comprobamos que los generadores añadimos son los mismos.

Para concluir, eliminamos los generadores de prueba añadidos, restaurando el generador.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Entrada | Condiciones de ejecución | Respuesta Esperada | Respuesta Obtenida | Comentarios |
| 1 | Generador | La clase App está instanciada y el archivo “generador.ser” existe | Equals | Equals | Si el generador actualizado en memoria y el generador creado fueran distintos, su longitud no coincidiría |

## Método “loadHabilidades”

Comprobamos que el archivo “habilidades.ser” existe y se puede acceder a él.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Entrada | Condiciones de ejecución | Respuesta Esperada | Respuesta Obtenida | Comentarios |
| 1 | - | La clase App está instanciada | NotNull | NotNull | Si el archivo “habilidades.ser” no se encontrase o no se pudiese crear, devolvería Null |

## Método “saveHabilidades”

Comprobamos que se puede acceder con permiso de escritura al archivo “habilidades.ser”.

Para ello, cargamos en una variable las habilidades almacenadas en el fichero (lectura), añadimos nuevas habilidades a la lista y guardamos los cambios en el archivo (escritura).

A continuación leemos de nuevo el fichero (actualizado con las nuevas habilidades) y los guardamos en otra variable. Verificamos que los cambios se han realizado correctamente, comprobando la extensión de la lista de habilidades actualiza en la variable (sin guardarse en el archivo) y las cargadas del fichero. También comprobamos que las habilidades que añadimos son las mismas.

Para concluir, eliminamos las habilidades de prueba añadidas, restaurando la lista original.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Entrada | Condiciones de ejecución | Respuesta Esperada | Respuesta Obtenida | Comentarios |
| 1 | Habilidades | La clase App está instanciada y el archivo “habilidades.ser” existe | True | True | - |
| 2 | Talento | La clase App está instanciada y el archivo “habilidades.ser” existe | True | True | - |
| 3 | Disciplina | La clase App está instanciada y el archivo “habilidades.ser” existe | True | True | - |
| 4 | Don | La clase App está instanciada y el archivo “habilidades.ser” existe | True | True | - |

## Método “run”

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Entrada | Condiciones de ejecución | Respuesta Esperada | Respuesta Obtenida | Comentarios |
| 1 | -1 | Lorem ipsum | 5 | 5 | Lorem ipsum |
| 2 | 0 | Lorem ipsum | “hola” | “hola” | Lorem ipsum |
| 3 | 1 | Lorem ipsum | true | false | Lorem ipsum |
| 4 | null | Lorem ipsum | NullPointerException | NullPointerException | Lorem ipsum |

@Test

public void testRun() throws Exception {

System.out.println("run");

app.run();

// TODO review the generated test code and remove the default call to fail.

fail("The test case is a prototype.");

}

### Pruebas no exitosas

# Personaje

Diagrama

Descripción generada automáticamente

Ilustración 2 Diagrama de la clase Personaje

## Método AnadirItem

Comprobamos que se puede añadir un Equipo a un personaje. Para ello creamos una instancia de Equipo, en este caso un Arma, y utilizando el método anadirItem se la añadimos a la instancia nueva de Personaje. Luego comprobamos que el nuevo personaje tenga ese arma guardada.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Entrada | Condiciones de ejecución | Respuesta Esperada | Respuesta Obtenida | Comentarios |
| 1 | Equipo | Las clases Equipo y Personaje se pueden instanciar | True | True | - |

## Método ModificarOro

Para comprobar que el método ModificarOro funciona correctmamente instanciamos la clase Personaje (cabe recordar que el oro inicial es 50), le restamos una cantidad de oro a través del método ModificarOro y comprobamos que el valor modificado se ajusta a la realidad

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Entrada | Condiciones de ejecución | Respuesta Esperada | Respuesta Obtenida | Comentarios |
| 1 | 5 | La clase Personaje se puede instanciar | True (45) | True (45) |  |

## Método PonerArmaActiva

Instanciamos la clase Arma y Personaje, y añadimos al personaje el arma nueva como arma activa. A continuación, comprobamos que el arma activa del personaje corresponde a la que acabamos de modificar.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Entrada | Condiciones de ejecución | Respuesta Esperada | Respuesta Obtenida | Comentarios |
| 1 | Arma | Las clases Arma y Personaje se pueden instanciar | True | True | La respuesta es True si el arma activa coincide con la esperada |

## Método OroSuficiente

Instanciamos la clase Personaje, que por defecto tendrá 50 de oro. Llamamos al método OroSuficiente, pasándole difentes valores. Comprobamos que si los valores que le pasamos son menores o iguales al oro que tiene el personaje el método nos debe devolver true. En caso contrario, false

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Entrada | Condiciones de ejecución | Respuesta Esperada | Respuesta Obtenida | Comentarios |
| 1 | 5 | La clase Personaje se puede instanciar | true | true | - |
| 2 | 50 | La clase Personaje se puede instanciar | true | true |  |
| 3 | 51 | La clase Personaje se puede instanciar | false | false | El valor de entrada es mayor al oro del personaje |

## Método AñadirModificador

Instanciamos las clases Modificador y Personaje. Añadimos al personaje el nuevo modificador con valor de ataque 2. Comprobamos que el potencial de ataque del modificador del personaje coincide con el valor de ataque dado.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Entrada | Condiciones de ejecución | Respuesta Esperada | Respuesta Obtenida | Comentarios |
| 1 | 2 | Las clases Modificador y Personaje se pueden instanciar | True | True | La respuesta obtendia es true si el valor es 2 |

## Método AñadirEsbirro

Instanciamos las clases Esbirro y Personaje. Añadimos al personaje un nuevo esbirro. Comprobamos que se ha añadido correctamente

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Entrada | Condiciones de ejecución | Respuesta Esperada | Respuesta Obtenida | Comentarios |
| 1 | Esbirro | Las clases Esbirro y Personaje se pueden instanciar | True | True | La respuesta obtenida es true si el esbirro se ha podido añadir correctamente |

## Método CalcularSaludEsbirros

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Entrada | Condiciones de ejecución | Respuesta Esperada | Respuesta Obtenida | Comentarios |
| 1 | -1 | Lorem ipsum | 5 | 5 | Lorem ipsum |
| 2 | 0 | Lorem ipsum | “hola” | “hola” | Lorem ipsum |
| 3 | 1 | Lorem ipsum | true | false | Lorem ipsum |
| 4 | null | Lorem ipsum | NullPointerException | NullPointerException | Lorem ipsum |

public void testCalcularSaludEsbirros() {

System.out.println("calcularSaludEsbirros");

int salud=1;

Esbirro es = new Humano("humano", salud, Lealtad.ALTA);

Personaje instance = new Licantropo("lican", new Don("don", 1,1,1));

instance.añadirEsbirro(es);

instance.calcularSaludEsbirros();

assertTrue(instance.getSalud\_esbirros()==salud);

// TODO review the generated test code and remove the default call to fail.

fail("The test case is a prototype.");

}

## Método CalcularAtaqueEquipo

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Entrada | Condiciones de ejecución | Respuesta Esperada | Respuesta Obtenida | Comentarios |
| 1 | -1 | Lorem ipsum | 5 | 5 | Lorem ipsum |
| 2 | 0 | Lorem ipsum | “hola” | “hola” | Lorem ipsum |
| 3 | 1 | Lorem ipsum | true | false | Lorem ipsum |
| 4 | null | Lorem ipsum | NullPointerException | NullPointerException | Lorem ipsum |

public void testCalcularAtaqueEquipo() {

System.out.println("calcularAtaqueEquipo");

Personaje instance = new Licantropo("lican", new Don("don", 1,1,1));

int ataque=2;

Arma arma = new Arma("arma", ataque, 0, 1, 2);

instance.ponerArmaActiva(arma);

int expResult = ataque;

int result = instance.calcularAtaqueEquipo();

assertEquals(expResult, result);

// TODO review the generated test code and remove the default call to fail.

fail("The test case is a prototype.");

}

## Método CalcularModificadoresAtaque

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Entrada | Condiciones de ejecución | Respuesta Esperada | Respuesta Obtenida | Comentarios |
| 1 | -1 | Lorem ipsum | 5 | 5 | Lorem ipsum |
| 2 | 0 | Lorem ipsum | “hola” | “hola” | Lorem ipsum |
| 3 | 1 | Lorem ipsum | true | false | Lorem ipsum |
| 4 | null | Lorem ipsum | NullPointerException | NullPointerException | Lorem ipsum |

public void testCalcularModificadoresAtaque() {

System.out.println("calcularModificadoresAtaque");

Personaje instance = new Licantropo("lican", new Don("don", 1,1,1));

int ataque=2;

instance.añadirModificador(new Modificador("mod", ataque, true));

int expResult = ataque;

int result = instance.calcularModificadoresAtaque();

assertEquals(expResult, result);

// TODO review the generated test code and remove the default call to fail.

fail("The test case is a prototype.");

}

## Método CalcularModificadoresDefensa

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Entrada | Condiciones de ejecución | Respuesta Esperada | Respuesta Obtenida | Comentarios |
| 1 | -1 | Lorem ipsum | 5 | 5 | Lorem ipsum |
| 2 | 0 | Lorem ipsum | “hola” | “hola” | Lorem ipsum |
| 3 | 1 | Lorem ipsum | true | false | Lorem ipsum |
| 4 | null | Lorem ipsum | NullPointerException | NullPointerException | Lorem ipsum |

public void testCalcularModificadoresDefensa() {

System.out.println("calcularModificadoresDefensa");

Personaje instance = new Licantropo("lican", new Don("don", 1,1,1));

int defensa=2;

instance.añadirModificador(new Modificador("mod", defensa, false));

int expResult = defensa;

int result = instance.calcularModificadoresDefensa();

assertEquals(expResult, result);

// TODO review the generated test code and remove the default call to fail.

fail("The test case is a prototype.");

}

### Pruebas no exitosas

# Vampiro

Diagrama

Descripción generada automáticamente

Ilustración 2 Diagrama de la clase Vampiro

## Método getPotencialAtaque()

Comprobamos que el método nos devuelve el valor correspondiente. Para ello, creamos una disciplina y con ella un vampiro, y llamamos al método “GetPotencialAtaque”. Finalmente, comprueba que el valor obtenido coincide con el valor esperado.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Entrada | Condiciones de ejecución | Respuesta Esperada | Respuesta Obtenida | Comentarios |
| 1 | Vampiro | Creamos una disciplina y con ella un vampiro. | 6 | 6 | - |

## Método getPotencialDefensa()

Análogamente al test de “GetPotencialAtaque”, comprobamos que el método funciona correctamente.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Entrada | Condiciones de ejecución | Respuesta Esperada | Respuesta Obtenida | Comentarios |
| 1 | Vampiro | Creamos una disciplina y con ella un vampiro. | 6 | 6 | - |

## Método prepararCombate()

Comprobamos que el método nos devuelve los valor correspondientes al preparar el combate. Para ello, creamos una disciplina y con ella un vampiro, y llamamos al método “PrepararCombate”. Finalmente, comprueba que los valores guardados (salud, sangre y salud de los esbirros) coinciden con los valores esperados.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Entrada | Condiciones de ejecución | Respuesta Esperada | Respuesta Obtenida | Comentarios |
| 1 | Vampiro | Creamos una disciplina y con ella un vampiro. | Salud: 5, sangre:0,  salud esbirros:0 | Salud: 5, sangre:0,  salud esbirros:0 | - |

# Cazador

Diagrama

Descripción generada automáticamente

Ilustración Diagrama de la clase Cazador

## Método getPotencialAtaque()

Comprobamos que el método nos devuelve el valor correspondiente. Para ello, creamos e inicialazamos las clases requeridas (talento, personaje, arma, armadura) con valores conocidos y llamamos al método “GetPotencialAtaque”. Comprobamos que ambos valores coinciden.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Entrada | Condiciones de ejecución | Respuesta Esperada | Respuesta Obtenida | Comentarios |
| 1 | Cazador | Creamos un talento y con él un cazador. Además, le añadimos un arma y una armadura. | 8 | 8 | - |

## Método getPotencialDefensa()

Análogamente al test de “GetPotencialAtaque”, comprobamos que el método funciona correctamente.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Entrada | Condiciones de ejecución | Respuesta Esperada | Respuesta Obtenida | Comentarios |
| 1 | Cazador | Creamos un talento y con él un cazador. Además, le añadimos un arma y una armadura. | 10 | 10 | - |

## Método prepararCombate()

Comprobamos que el método nos devuelve los valor correspondientes al preparar el combate. Para ello, creamos e inicialazamos las clases requeridas (talento, cazador) con valores conocidos y llamamos al método “PrepararCombate”. Comprobamos que los parámetros de entrada coinciden con los que se ejecutan.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Entrada | Condiciones de ejecución | Respuesta Esperada | Respuesta Obtenida | Comentarios |
| 1 | Cazador | Creamos un talento y con él un cazador. | Salud: 5  Voluntad: 3  Salud esbirros: 0 | Salud: 5  Voluntad: 3  Salud esbirros: 0 | - |

# Licántropo

Diagrama

Descripción generada automáticamente

Ilustración Diagrama de la clase Licantropo

## Método getPotencialAtaque()

Comprobamos que el método nos devuelve el valor correspondiente. Para ello, creamos un don y con él un licántropo, al que le damos un arma y una amadura. Finalmente, llamamos al método “GetPotencialAtaque” y comprobamos que el valor obtenido coincide con el valor esperado.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Entrada | Condiciones de ejecución | Respuesta Esperada | Respuesta Obtenida | Comentarios |
| 1 | Licántropo | Creamos un don y con él un licántropo. Además, le añadimos un arma y una armadura. | 4 | 4 | - |

## Método getPotencialDefensa()

Análogamente al test de “GetPotencialAtaque”, comprobamos que el método funciona correctamente.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Entrada | Condiciones de ejecución | Respuesta Esperada | Respuesta Obtenida | Comentarios |
| 1 | Licántropo | Creamos un don y con él un licántropo. Además, le añadimos un arma y una armadura. | 4 | 4 | - |

## Método prepararCombate()

Comprobamos que el método nos devuelve los valor correspondientes al preparar el combate. Para ello, creamos u n don y con él un licántropo, y llamamos al método “PrepararCombate”. Finalmente, comprueba que los valores guardados (salud, rabia y salud de los esbirros) coinciden con los valores esperados.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Entrada | Condiciones de ejecución | Respuesta Esperada | Respuesta Obtenida | Comentarios |
| 1 | Vampiro | Creamos una disciplina y con ella un vampiro. | Salud: 5  Rabia:0  Salud esbirros:0 | Salud: 5  Rabia: 0  Salud esbirros: 0 | - |