Descripción de las pruebas realizadas para comprobar el correcto funcionamiento del programa.

José Victor García Llorente

Alejandro Martín Carrera

Jorge Padilla Rodríguez

Eloy de Sande de las Heras

Olga Somalo Serrano

Fase Pruebas

Contenido

[Casos de prueba 2](#_Toc134028176)

[Caso de prueba para caso de uso de “Registrar Usuario” 2](#_Toc134028177)

[Casos de prueba para caso de uso de “Login” 2](#_Toc134028178)

[Caso de prueba para caso de uso de 3](#_Toc134028179)

# Casos de prueba

## Caso de prueba para caso de uso de “Registrar Usuario”

* ENTRADA:

Nombre de usuario: Hacker.

* CONDICIONES DE EJECUCIÓN:

El nombre de usuario “Hacker” ya está registrado.

* RESULTADO ESPERADO:

Lanza un mensaje indicando que ya está registrado y no deja entrar al sistema.

* OBJETIVO DEL CASO DE PRUEBA:

Comprobar que no deja registrarse a un usuario que ya existe.

## Casos de prueba para caso de uso de “Login”

CASO 1:

* ENTRADA:

Nombre de usuario: Hacker.

* CONDICIONES DE EJECUCIÓN:

El nombre de usuario “Hacker” no está registrado.

* RESULTADO ESPERADO:

Lanza un mensaje indicando que no está registrado y no deja entrar al sistema.

* OBJETIVO DEL CASO DE PRUEBA:

Comprobar que no deja entrar al sistema a un usuario que no está registrado.

CASO 2:

* ENTRADA:

Nombre de usuario: Hacker.

Contraseña: 0000.

* CONDICIONES DE EJECUCIÓN:

El nombre de usuario “Hacker” está registrado, pero la contraseña no es correcta.

* RESULTADO ESPERADO:

Lanza un mensaje indicando que la contraseña es incorrecta y no deja entrar al sistema.

* OBJETIVO DEL CASO DE PRUEBA:

Comprobar que no deja entrar al sistema a un usuario que sí existe pero que introduce una contraseña equivocada.

Caso de prueba para caso de uso de “Banear Usuario”

* ENTRADA:

Nombre de usuario: Hacker.

* CONDICIONES DE EJECUCIÓN:

Intenta iniciar sesión el usuario “Hacker”, que está baneado.

* RESULTADO ESPERADO:

Lanza un mensaje indicando que está baneado y no deja entrar al sistema.

* OBJETIVO DEL CASO DE PRUEBA:

Comprobar que no deja entrar al sistema a un usuario que está baneado.

Caso de prueba para caso de uso de “Desbanear Usuario”

* ENTRADA:

Nombre de usuario: Hacker.

* CONDICIONES DE EJECUCIÓN:

Intenta iniciar sesión el usuario “Hacker”, que está desbaneado.

* RESULTADO ESPERADO:

Le deja entrar al sistema.

* OBJETIVO DEL CASO DE PRUEBA:

Comprobar que deja entrar al sistema a un usuario que está desbaneado.

Caso de prueba para caso de uso de “Validación de combate”

* ENTRADA:

Nombre de usuario: Hacker.

Contraseña: 0000.

* CONDICIONES DE EJECUCIÓN:

Inicia sesión el usuario “Hacker”, que ha sido desafiado por otro usuario y un operador del sistema ha validado el desafío.

* RESULTADO ESPERADO:

Al entrar al sistema le aparece el desafío pendiente y le da la opción de aceptarlo o rechazarlo.

* OBJETIVO DEL CASO DE PRUEBA:

Comprobar que la validación de combate se ha realizado correctamente y el usuario desafiado puede aceptarla o rechazarla.

Caso de pruebas 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.

Test método “loadRanking”

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

Test método “SaveRanking”

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

Test método “LoadEquipos”

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

Test 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.

Test método “LoadGenerador”

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

Test 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.

Test método “LoadHabilidades”

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

Test 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.

Casos de prueba de la clase “Cazador”

Test método “GetPotencialAtaque”

Comprobamos que el método nos devuelve el valor que se corresponde a la realidad.

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.

Test método “GetPotencialDefensa”

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

Test método “PrepararCombate”

Comprobamos que el método nos devuelve el valor que se corresponde a la realidad.

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.

Casos de prueba de la clase “Cliente”

Test método “CambiarPersonaje”

Este test comprobará que el método “CambiarPersonaje” funciona correctamente.

Para ello creamos un nuevo personaje, con habilidades por defecto y se las modificamos.

Comprobamos que las habilidades se han modificado correctamente.

Test método “ElegirEquipo”

Este test comprobará que el método “ElegirEquipo” funciona correctamente.

Para ello creamos un nuevo cliente, y le añadimos habilidades, arma, armadura y modificamos su arma activa.

Comprobamos que las propiedades se hayan modificado correctamente.

Test método “Desafiar”

Este test comprobará que el método “Desafiar” funciona correctamente.

Para ello, creará instancias de las clases implicadas (Menu, GeneradorIDs, Cliente, Ranking) y desafiará a los dos clientes creados para comprobar que el método funciona correctamente.

Test método “ResponderDesafios”

Este test comprobará que el método “ResponderDesafios” funciona correctamente.

Para ello, creará instancias de las clases implicadas y desafiará a los dos clientes creados.

Finalmente, comprobará que el método “ResponderDesafios” se activará correctamente.

Test método “HayDesafios”

Este test comprobará que el método “HayDesafios” funciona correctamente.

Para ello, se comprobará con un boolean si el método no se activa cuando no hay desafios pendientes.

Test método “AñadirCombate”

Este test comprobará que el método “AñadirCombate” funciona correctamente.

El test creará un nuevo combate (con valores por defecto), lo desarrollará y comprobará que el combate se añade correctamente al cliente.

Test método “ConsultaCombates”

Este test comprobará que el método “ConsultaCombates” funciona correctamente.

Para ello, creará un nuevo combate que añadirá a un nuevo usuario y comproborá que la lista de combates de un cliente se muestre correctamente.

Test método “ConsultaRanking”

Test método “EnviarDesafio”

Test método “ComprarItem”