PRÁCTICA METODOLOGÍA DE LA PROGRAMACIÓN

2022-23

DISEÑO

NOMBRE	APELLIDOS	DNI	GRADO
Lucía	Domínguez Rodrigo	49452469-P	GII+GIS
Ángel	Marqués García	34295108-S	GII+GIS
Marcos	Jiménez Pulido	04860135-M	GII+GIS
Sergio	De Oro Fernández	54712181-B	GII+GIS

LISTA DE DISTRIBUCIÓN DE ROLES

NOMBRE	ROL	ORGANIZAZIÓN
Sergio	Jefe de Proyecto	*URJC*
Lucía	Analista Funcional	*URJC*
Marcos	Analista Programador	*URJC*
Sergio	Ingeniero de Desarrollo	*URJC*
Ángel	Quality Assurance Tester	*URJC*

Contenido

OBJETIVOS DEL DOCUMENTO

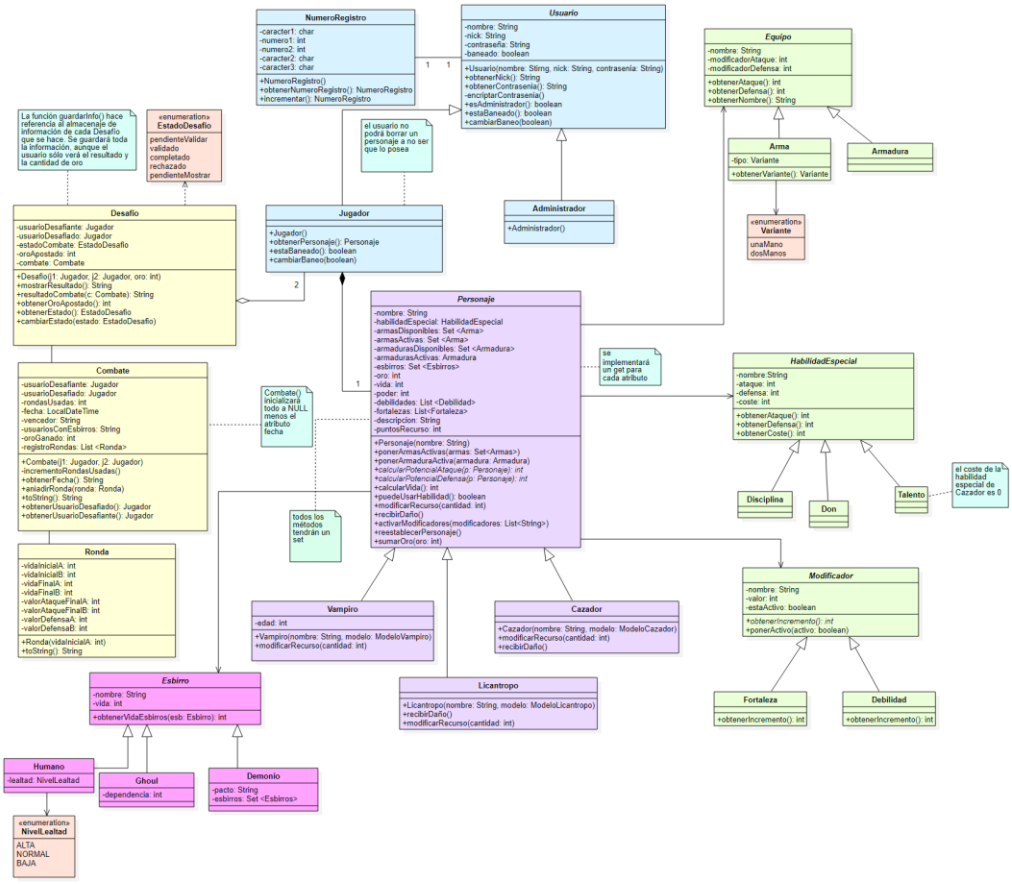
El siguiente documento tiene como objetivo describir los diseños necesarios para el desarrollo del software que la empresa *MetProg URJC S.L* ha pedido desarrollar. Los siguientes diseños servirán de guía para los desarrolladores involucrados en el proyecto, y permitirá la correcta implementación de las funcionalidades requeridas por el cliente. Dichos diseños se basan en los requisitos obtenidos y descritos en la fase anterior del proyecto, recogidos en el documento "*Documento de análisis*".

Es importante destacar el uso de distintos tipos de diagramas, que permitirán a los desarrolladores comprender mejor el funcionamiento del sistema en términos de interacciones entre sus diversas partes. Cada diagrama contendrá una explicación para mejor comprensión de este.

DIAGRAMAS DE CLASES

Diagrama de las clases del juego

El siguiente diagrama muestra las relaciones entre las distintas clases principales del juego. Cabe destacar que la gran mayoría de estas funciones sirven a modo de registro, es decir, almacenan u obtienen información y ofrecen operaciones básicas con ella, pero no se encargan de ejecutar el hilo principal del programa.



Estas clases han sido clasificadas en distintos grupos, que se muestran en el diagrama mediante colores. A continuación, iremos explicando cada uno de estos grupos.

El primer grupo está formado por la clase Usuario y sus derivadas, que están coloreadas de azul. La clase Usuario es abstracta, por lo que para instanciarla se deberá usar uno de sus hijos; las clases Jugador o Administrador. Estas clases aportan más información sobre el usuario, y le añaden, limitan, o describen métodos y funcionalidades. La clase usuario contiene una instancia de la clase NumeroRegistro, que también estará presente en sus hijos. Esta clase se guardará en un atributo, y se usará para distinguir entre distintos Usuarios. También servirá para saber el tipo de Usuario (si es Jugador o Administrador), ya que todos los Administradores compartirán un mismo número de registro (se utilizará su Nick para diferenciarlos).

El siguiente grupo es el de Personaje y sus derivadas, marcadas con un color morado claro. Este grupo es esencial, ya que, si bien no es demasiado grande, se relaciona con prácticamente todos los demás grupos.

La clase Personaje formará parte de la clase Jugador, es decir, una instancia de Personaje sólo podrá existir si forma parte de una instancia de Jugador. Si se borra el Jugador, se borrará su Personaje asociado. La clase tiene una serie de atributos y métodos que le permiten llevar a cabo sus responsabilidades, que consisten en lo siguiente:

1. Mantener un registro de las características del personaje
2. Calcular la vida, y los potenciales de ataque y defensa para ser usados en el combate
3. Llevar la cuenta de los recursos que se utilicen durante el combate (puntosRecurso)

Al igual que la clase Usuario, la clase Personaje es abstracta, ya que los personajes deben pertenecer necesariamente a una de las siguientes categorías: Vampiros, Cazadores, y Licántropos. Dado que cada uno de esos tipos de personaje tendrán funcionalidades ligeramente distintas (o implementarán las mismas funcionalidades de manera distinta), se han creado clases para cada uno de ellos.

En color rosa encontramos las clases derivadas de Esbirro. Los esbirros podrán ser de tres tipos distintos, Humano, Ghoul, y Demonio, y dado que cada tipo tiene características propias, se han creado clases para cada uno. Es importante destacar que las instancias de Personaje contendrán un conjunto de objetos de tipo Esbirro (se debe tener en cuenta que, si el personaje es un Vampiro, no puede tener esbirros Humanos).

El grupo de clases coloreadas en verde representan las clases que modifican los valores de ataque/defensa del personaje. El grupo está formado por Equipo, HabilidadEspecial, y Modificador, y aquellas clases que heredan de estas. Las tres clases principales de este grupo (las mencionadas) son abstractas, ya que todas ellas se subdividen en categorías diferentes. Entre estas categorías cabe destacar la categoría Arma, que añade un atributo "tipo" que guarda un valor del enumerado Variante que se usa para saber si es de una mano o de dos, y las categorías de Modificador. Estas: Fortaleza y Debilidad, se distinguen únicamente en la implementación de un método; obtenerIncremento. Las fortalezas lo implementarán para que devuelva valores positivos, y las debilidades para que devuelva valores negativos. Cabe destacar que estas clases tomarán valores fijos, leídos de un archivo.

Todas las clases de este grupo se podrán almacenar en los atributos de la clase Personaje. Se debe tener en cuenta que, el tipo de HabilidadEspecial que tenga el Personaje irá ligado al tipo de Personaje que sea (Vampiros con Disciplinas, Licántropos con Dones, y Cazadores con Talentos).

El último conjunto de clases vendrá indicado en amarillo, y se corresponderá a la sección relacionada con los desafíos y los combates. Como viene indicado en el diagrama, un desafío contiene a dos jugadores, mientras que un Jugador puede aparecer en cualquier número de Desafíos. La clase *Desafio* se ocupará de registrar cada desafío junto con su estado. También se ocupa de crear instancias de la clase Combate. En función del estado de un desafío, distintos tipos de usuarios podrán acceder a él.

La clase Combate se ocupa de registrar cada combate. Una instancia de Combate se corresponde con un combate entre dos jugadores, y guarda toda la información pertinente sobre lo ocurrido en ese combate. Para ayudarle en ese cometido existe la clase Ronda, que registra lo ocurrido en cada ronda individual de un combate.

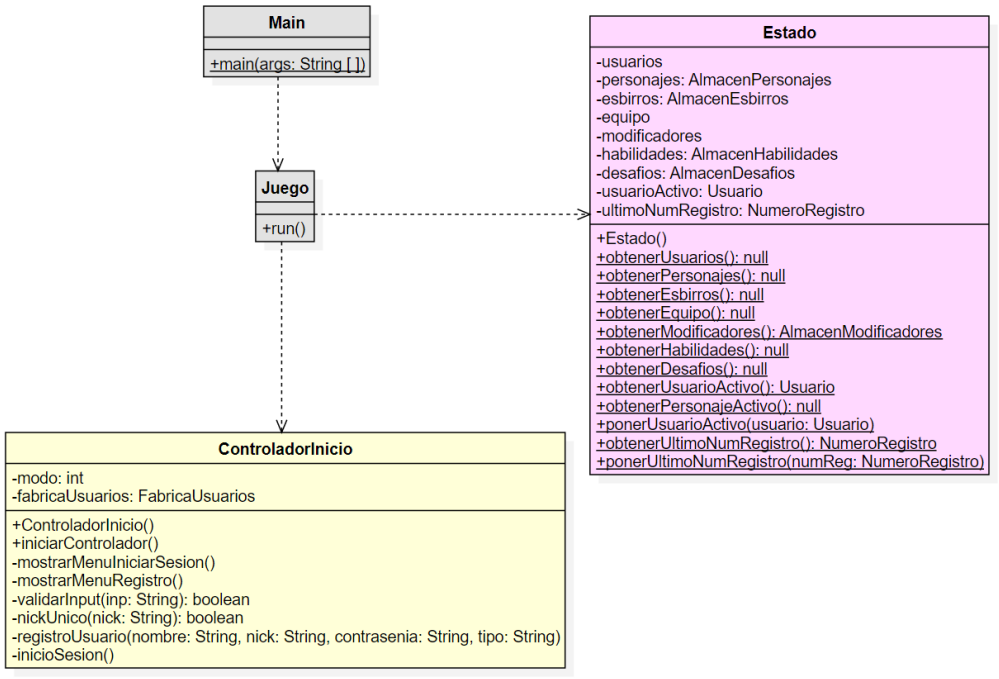
Cabe destacar que estas clases no son las encargadas de llevar a cabo los combates, ni de contener la lógica de estos. Simplemente sirven para registrar lo ocurrido en los combates. La clase encargada de llevar a cabo los combates es DirectorCombate, que se especificará más adelante.

Las clases estarán relacionadas con los controladores, que son los que llevan la gran mayoría de la ejecución del programa.

Las clases Personaje, Usuario, Desafío, Combate, y Ronda, junto con todas sus clases hijas, deben implementar la interfaz Serializable. El diagrama no refleja esto para preservar la claridad.

Diagrama de Clases de Inicio

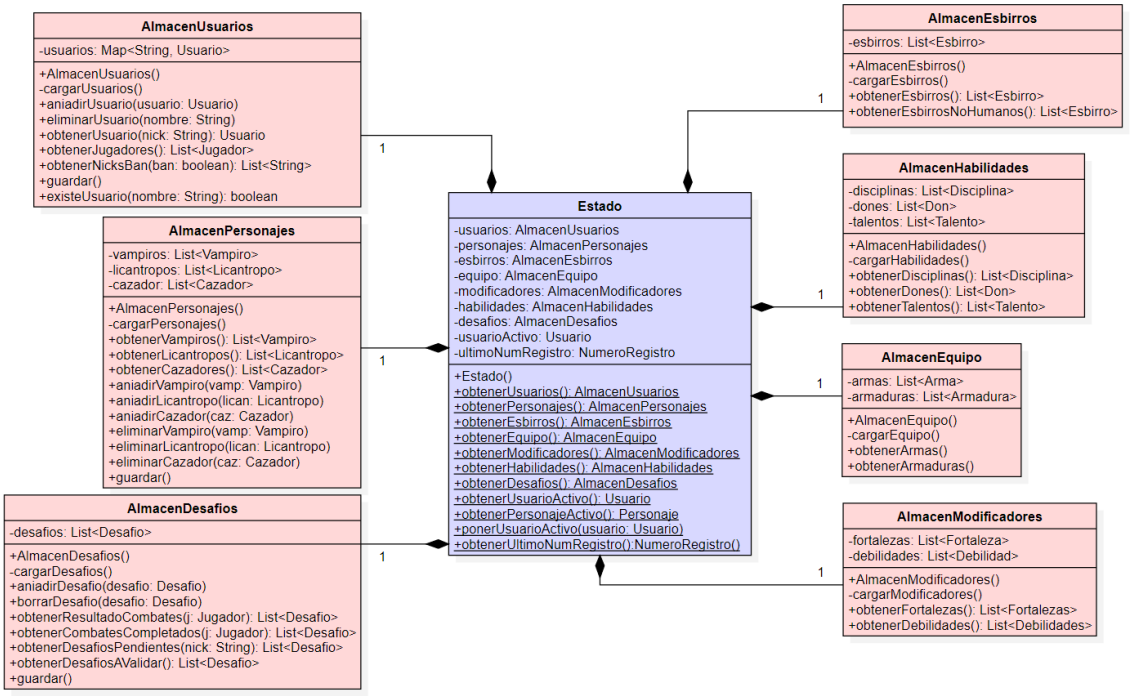
El siguiente diagrama muestra las clases involucradas en el inicio de la aplicación.



La clase Main creará una instancia de Juego, y llamará a su método run, que se ocupará de crear una instancia de Estado, y pedirle que cargue todos sus datos. Una vez hecho eso, crea una instancia de ControladorInicio, y le pasa el control llamando a su método iniciarControlador. El resto del programa ocurrirá dentro de esa función.

Diagrama de Clases de la Base de Datos

En el siguiente diagrama se muestran las clases relacionadas con la “base de datos” que usará el software.



La clase principal de este diagrama es Estado. Esta se ocupa de almacenar una copia de las clases que se han guardado en archivos. Para ello contiene una serie de clases almacén, que guarda cada una en un atributo. También es la responsabilidad de la clase Estado saber cuál es el usuarioActivo, y llevar la cuenta de los números de registro. Para ello, tiene un atributo, ultimoNumRegistro, que corresponde al último número de registro que se ha asociado a un Usuario.

Los métodos de esta clase son estáticos. Esto se ha decido así para que su información sea accesible desde cualquier parte del programa.

Las otras clases del diagrama son todas almacenes. Estas clases contienen al menos, un método para cargar su información y métodos para hacer distintas búsquedas.

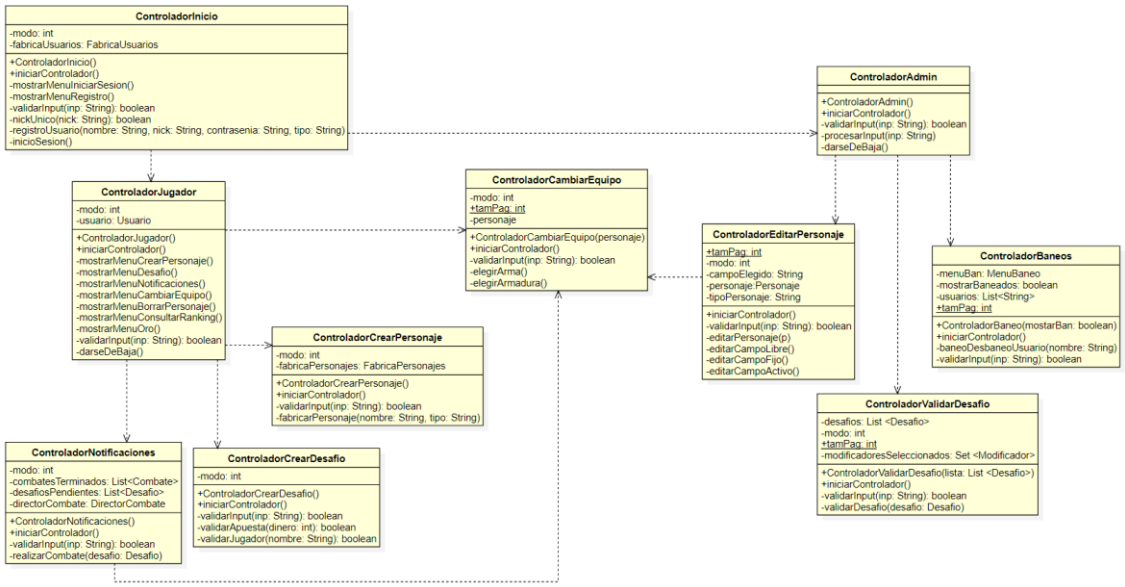
Los almacenes de la izquierda (usuarios, personajes y desafíos) hacen referencia a la información que se obtiene desde ficheros binarios. Por ello, tienen una serie de métodos que les permiten ser modificados (añadiendo o eliminando elementos), y un método para actualizar el fichero que los almacena.

Por otro lado, los almacenes de la derecha hacen referencia a información que se obtiene desde ficheros de texto plano, por lo que sólo tienen métodos para obtener información; estos ficheros no podrán ser modificados.

Diagrama de Clases de los Controladores

La ejecución del programa y la gran mayoría de las funcionalidades ocurrirá desde una serie de clases llamadas controladores. Cada controlador tendrá asociadas una serie de clases llamadas Menu, que harán la función de una vista; muestran información por pantalla y devuelven un input del usuario. En función del input del usuario el controlador realizará una función u otra. Además, los controladores usarán las clases que necesiten de los otros diagramas para cumplir con sus responsabilidades. Para este diseño se ha seguido la propuesta de arquitectura de software: “Modelo Vista Controlador (MVC)”.

El siguiente diagrama muestra todos los controladores del sistema, y las relaciones entre ellos. Este diagrama tiene similitud con el diagrama de navegación, pero tiene algunas diferencias notables.



El ControladorInicio usa los controladores ControladorAdmin y ControladorJugador, ya que desde el menú de inicio puedes navegar hasta cualquiera de esos dos.

De igual manera, el ControladorAdmin usa los controladores ControladorValidarDesafio, ControladorBaneos, y ControladorEditarPersonaje, ya que esos son los menús a los que puedes navegar desde el ControladorAdmin.

Desde el ControladorJugador se puede acceder a los controladores ControladorCrearDesafio, ControladorCrearPersonaje, ControladorNotificaciones, y ControladorCambiarEquipo.

Este último controlador (ControladorCambiarEquipo) es una excepción, ya que se puede acceder a él desde varios sitios. Lo usan ControladorEditarPersonaje, ControladorNotificaciones, y ControladorJugador. Esto es porque el servicio que presta es muy versátil, y necesita ser accedido desde distintos lugares (un administrador debe poder cambiar el equipo activo de un personaje, y un jugador debe poder cambiar su equipo activo a voluntad, o antes de un combate).

En cuanto a la navegación por la aplicación:

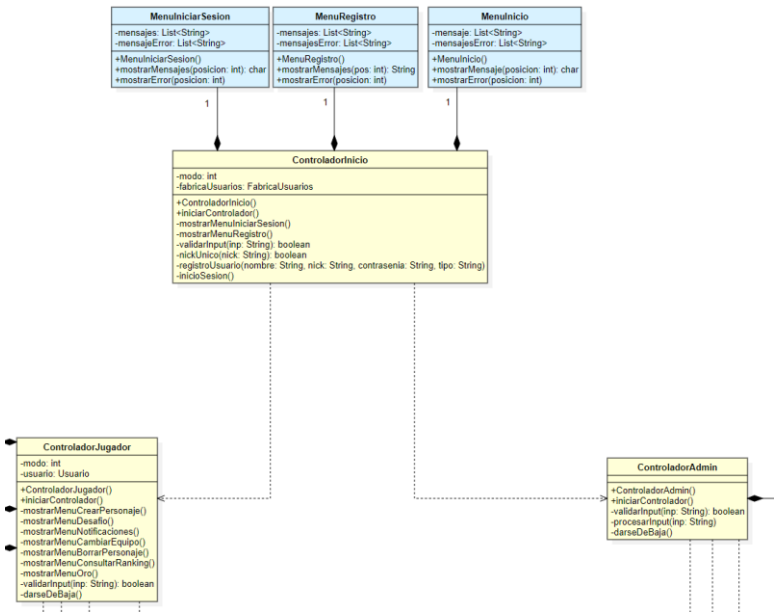
- El usuario podrá navegar libremente entre controladores en el sentido de las flechas
- Cuando el usuario decida salir de un controlador, navegará en la dirección opuesta de las flechas, desde el navegador en el que está, hasta el navegador que usó para llegar a este
- Si el usuario decide salir del ControladorInicio, se cerrará la aplicación

Los controladores, a pesar de no heredar de una clase abstracta (podrían hacerlo si facilitan la implementación, se ha decidido que no por ser muy distintos entre sí), contienen un método iniciarControlador público, desde el que se ejecutarán las acciones pertinentes. También tienen un método validarInput, que comprueba si los datos recibidos por el usuario son correctos. Este depende de una variable modo en cada uno de los controladores. En los controladores con

muchas opciones, podría ser pertinente añadir un método que procese la entrada del usuario cuando sea válida para evitar funciones complejas de leer.

Diagrama de Clases de Controladores y Menús

En los siguientes diagramas se muestra la relación de los controladores con sus correspondientes menús. Estos menús, por lo general, contendrán una lista de mensajes que se mostrarán según su posición (elegida en el controlador en función del modo y momento de la ejecución). Además, contendrán una lista de mensajes de error cuando los datos introducidos por el usuario no sean correctos.



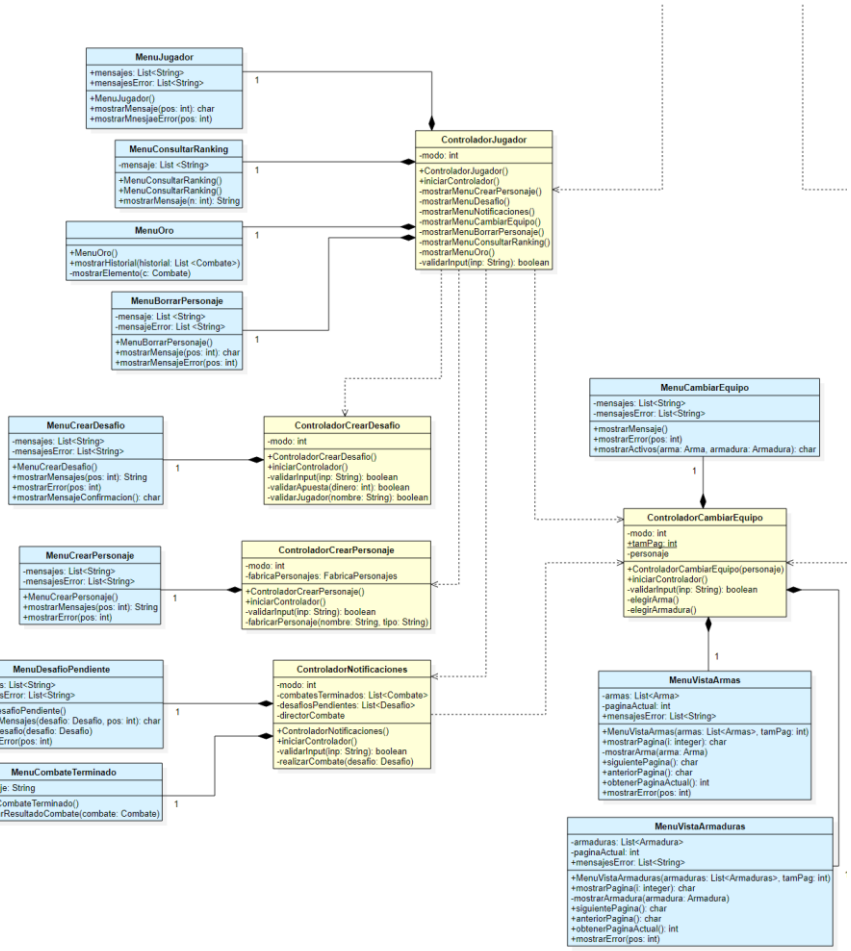
En este primer diagrama indicaremos las relaciones que tiene el controlador de inicio. Este tendrá tres menús. Al iniciar el programa, es decir, cuando se llama a iniciarControlador, se mostrará el MenuInicio. Este menú da la posibilidad al usuario de elegir entre darse de alta o iniciar sesión. En función de la elección del usuario, se mostrará el menú de registro o el menú de iniciar sesión.

El menú de registro solicita al usuario un nombre, un *nick* o apodo, y una contraseña. Si el *nick* introducido no es utilizado por ningún otro usuario, el controlador de inicio registrará al usuario en el almacén de usuarios. Si el *nick* no es válido, se mostrará un mensaje de error que indicará que el *nick* introducido ya está en uso. Por último, se preguntará al usuario si desea ser jugador o administrador. Una vez se han introducido datos válidos, el controlador inicio utiliza la fábrica de usuarios para crear una instancia de Usuario con los datos introducidos.

El menú de iniciar sesión solicita al usuario un *nick* y una contraseña. Si el *nick* introducido existe, se buscará ese usuario en el almacén, y se comprobará su contraseña con la introducida por el usuario. En caso de que la contraseña no coincida, se mostrará un mensaje de error indicando que la contraseña introducida no es correcta. En el caso de que el *nick* introducido no sea correcto, se mostrará un mensaje de error igual que el del menú anterior.

Una vez el inicio de sesión se lleve a cabo correctamente, se establece el usuario activo de la clase Estado. Con el método esAdminsitrador de la clase Usuario, se decidirá si se llama al controlador jugador, o al controlador administrador, creando una instancia de uno de estos.

Usuario Jugador



En el caso en el que la cuenta desde la que se inicie sesión sea de tipo Jugador, se ejecutará el controlador de Jugador, con 3 vistas asociadas. Al iniciar este controlador, se creará una instancia del controlador notificaciones, para mostrarle al jugador los posibles desafíos que tenga pendientes por aceptar (se muestran mediante el menú de desafíos pendientes). Por cada desafío, el jugador tendrá la opción de aceptarlo o rechazarlo. Si rechazase el desafío, el jugador será penalizado con un 10% del dinero apostado en el desafío y se devolverá la cantidad apostada al desafiante. Si lo acepta, se le restará al jugador la cantidad de la apuesta, y dará inicio el combate. El controlador de notificaciones utilizará el director de combate, que será el encargado de realizar el combate y establecer las recompensas. Si ha habido algún vencedor, al jugador vencedor se le sumará una victoria. El desafío pasará a tener como estado “porMostrar” y el controlador llamará al método guardar de los almacenes personaje, usuario y desafío, en ese orden.

Una vez mostrados todos los desafíos pendientes (o inmediatamente en caso de que no los hubiera), se muestran los resultados de los combates en los que haya participado el jugador. Por cada combate, se mostrará toda su información, así como su registro de rondas. Cuando el

usuario pulse algún botón, el desafío asociado a ese combate pasará a tener como estado 'completado' y el controlador llamará al método guardar del almacén de desafíos.

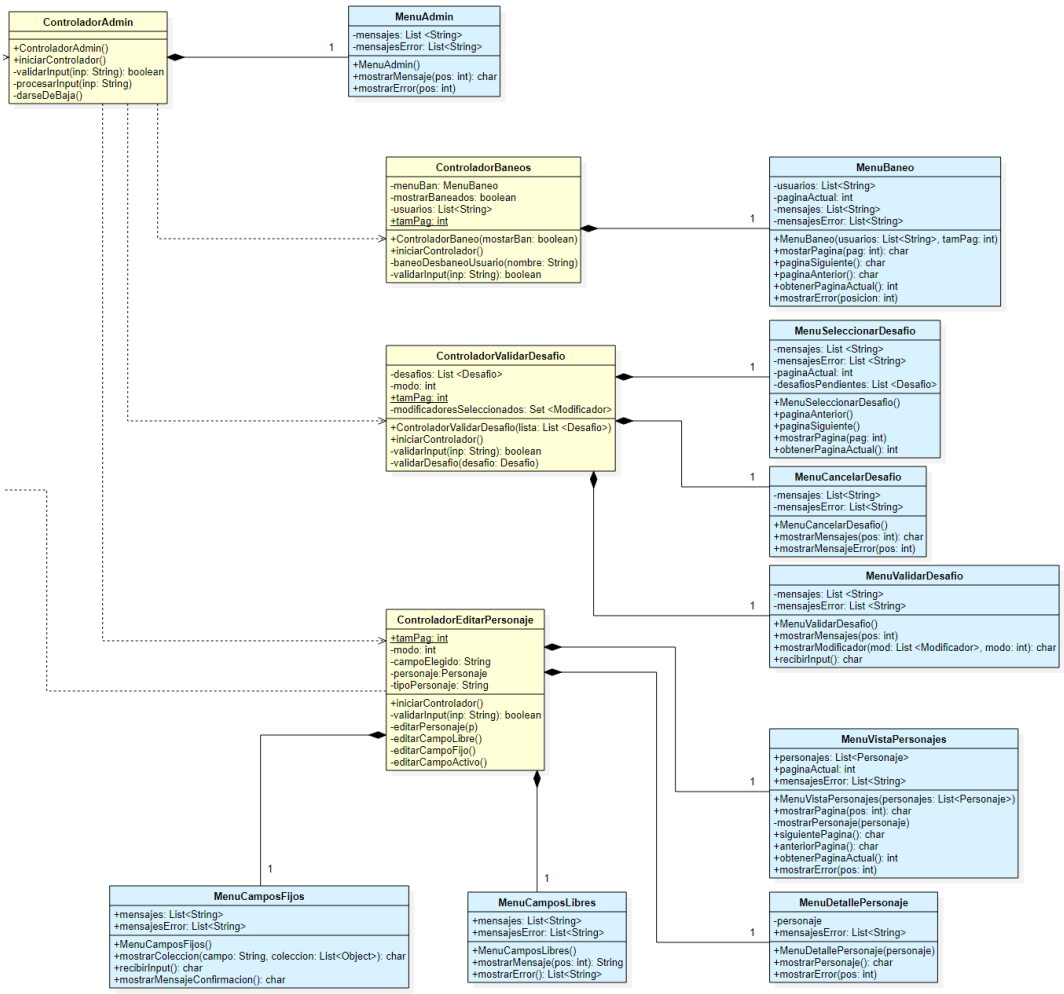
Una vez mostradas todas las notificaciones (en caso de que las hubiera), se mostrará el menú jugador, que tiene como opciones: crear personaje, crear desafío, cambiar equipo activo, consultar ranking, consultar historial de oro, borrar personaje, darse de baja y cerrar sesión. El jugador se mantendrá en este menú hasta que decida cerrar sesión o darse de baja. A continuación, se explicarán las opciones del menú:

- Crear personaje: El controlador jugador utilizará el controlador crear personaje, creando una instancia de este. Siempre que el jugador no tenga un personaje, el controlador crear personaje solicitará a través de su menú un nombre para el personaje y el tipo de personaje. Cuando los datos introducidos sean correctos, el controlador crear personaje utilizará la fábrica de personajes, que creará una instancia del tipo de personaje elegido y le asignará su nombre. Una vez creado el personaje, el controlador llamará al método añadirPersonaje del tipo escogido del almacén de personajes.
- Crear desafío: El controlador jugador utilizará el controlador crear desafío, creando una instancia de este. Siempre que el personaje del jugador tenga algún equipo activo, el controlador crear desafío, a través de su menú, solicitará al jugador el nombre el jugador que desea desafiar y la cantidad de oro que desea apostar (esta cantidad no podrá superar nunca la cantidad de oro del usuario). Una vez introducidos todos los datos, se mostrará un mensaje de confirmación al jugador. Si lo acepta, se le resta al jugador la cantidad apostada y se crea una instancia de la clase Desafío. También tendrá la opción de cambiar el equipo activo. Para esto, el controlador de crear desafío tendrá la opción de cambiar equipo activo. Se llamará al método guardar del almacén de usuarios y al método añadirDesafío del almacén de desafíos.
- Cambiar equipo activo: El controlador jugador utilizará el controlador cambiar equipo, creando una instancia de este. Siempre que el jugador tenga un personaje, el controlador dará al jugador dos opciones: cambiar armas activas o cambiar armadura activa (opciones mostradas desde el menú cambiar equipo). Si el jugador escoge la opción "cambiar arma", el controlador mostrará el menú vista armas. Este menú solicita al jugador un arma a escoger entre las disponibles en su personaje. Una vez elegida un arma, podrá elegir otra si esta tiene como variante "una mano". Una vez elegidas las armas, el controlador cambiará las armas activas anteriores a disponibles y las elegidas a activas. Por último, el controlador llamará al método guardar del almacén de personajes. Este proceso será análogo al elegir "cambiar armadura".
- Consultar ranking: El controlador jugador solicitará al almacén de usuarios una lista de jugadores, que se mostrará a través del menú consultar ranking. Esta lista podrá estar ordenada en función de la cantidad de oro o de la cantidad de victorias. Cuando el jugador pulse un botón, saldrá de este menú.
- Consultar historial de oro: El controlador jugador solicitará al almacén de desafíos los combates completados por el jugador activo, que se mostrarán el menú oro. En este menú solamente se mostrará el oro ganado o perdido por el jugador. Cuando el jugador pulse un botón, saldrá de este menú.
- Borrar personaje: Siempre que el jugador tenga un personaje, el controlador mostrará a través del menú borrar personaje un mensaje de confirmación. Si el jugador acepta,

entonces el controlador jugador llamará al método eliminarPersonaje correspondiente a su tipo de personaje.

- Darse de baja: El controlador jugador mostrará por pantalla un mensaje de confirmación. Si el jugador acepta, el controlador borrará el personaje del jugador y, después, borrará la cuenta de este. Al terminar este proceso se cierra sesión.

Usuario Administrador



En el caso en el que la cuenta desde la que se inicie sesión sea de tipo Administrador, se ejecutará el controlador de Administrador, con una única vista asociada. Este menú de Administrador se mostrará tras ejecutar el controlador, y ofrecerá las opciones de banear o desbanear jugadores, validar desafíos, editar personajes, darse de baja y cerrar sesión. El administrador se mantendrá en este menú hasta que decida cerrar sesión o darse de baja. A continuación, se explicarán las opciones del menú:

- Banear o desbanear jugadores: El controlador de administrador utilizará el controlador de baneos, creando una instancia de este. Este controlador ofrecerá la posibilidad de banear o desbanear jugadores. Si el administrador decide banear jugadores, en este menú se muestra una lista de los jugadores que no están baneados. Una vez elegido un jugador, el controlador de baneos solicitará al almacén de usuarios el jugador escogido,

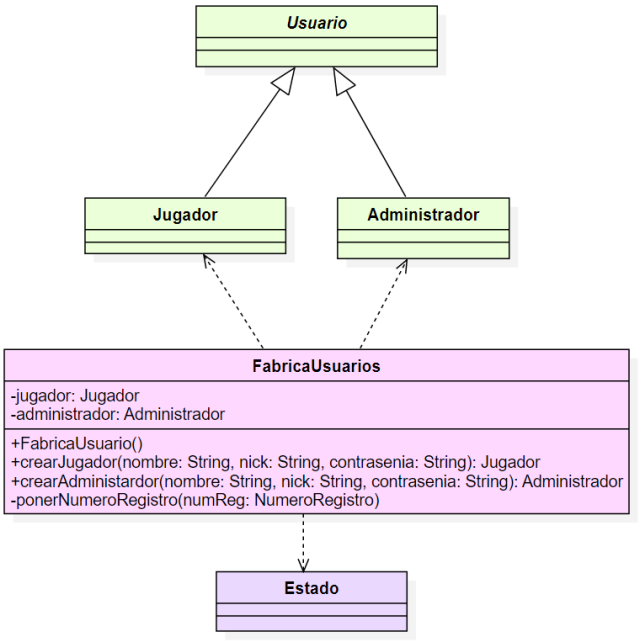
cambiará su campo baneado a “verdadero” y llamará al método guardar del almacén de usuarios.

- Validar desafíos: El controlador de administrador utilizará el controlador de validar desafíos, creando una instancia de este. Al iniciar este controlador se muestra el menú seleccionar desafío. Cuando el administrador selecciona un desafío, se le dará la opción de validarlo o rechazarlo. Si cancela el desafío, se mostrará el menú cancelar desafío que tendrá 2 opciones que representan el motivo de la cancelación del desafío (o bien al administrador no le parece justo, o bien el desafío incumple las normas). Si el desafío incumple las normas, el controlador solicitará al almacén de usuarios el jugador desafiante para banearlo (explicado anteriormente). En ambos casos se devolverá al desafiante el oro apostado. Por otro lado, si el administrador decide validar el desafío, se le mostrará en el menú validar desafío los modificadores del personaje desafiante (puede seleccionar tantos como quiera). Una vez seleccionados, se obtiene el personaje desafiante y se llama al método guardar del almacén de personajes. Este proceso se repite para los modificadores del personaje desafiado.
- Editar personaje: El controlador de administrador utilizará el controlador de editar personaje, creando una instancia de este. Se muestra en el menú de vista de personajes todos los personajes. Cuando el administrador elija uno, se mostrará en el menú detalle personaje todas las características del personaje, y podrá elegir un atributo para cambiar. Los campos se dividen en:
    - Campos libres: Nombre, vida, poder y descripción. Cuando se elija uno de estos campos, se mostrará el menú campos libres, que le pedirá al administrador un nuevo valor para este campo. Si el valor es correcto (vida y poder entre 1 y 5; nombre no vacío), cambiará el campo del personaje seleccionado y llamará al método guardar del almacén de personajes.
    - Campos fijos: Estos campos son: Habilidad especial, armas disponibles, armaduras disponibles, esbirros, fortalezas y debilidades. Si se elige uno de estos campos, se mostrará en el menú campos fijos una lista con los valores posibles, cumpliendo las restricciones de esbirros y habilidades. El administrador podrá elegir tantos elementos como quiera (excepto habilidades, que solamente puede elegir una). Cuando termine de seleccionar se cambiará el atributo seleccionado por los valores elegido por el administrador (con sus correspondientes solicitudes a los almacenes). Por último, se llama al método guardar del almacén de personajes.
    - Equipo activo: Se llama al controlador cambiar equipo y se inicia ese controlador.
- Darse de baja: Ocurre como en el controlador jugador.

Diagrama de Clases de Fábrica de Usuarios

En la sección “Diagrama de las clases del juego” se muestra la clase abstracta Usuario, de la que heredan otras dos clases; Jugador y Administrador. Para gestionar la creación de instancias de estas clases nos hemos guiado por el patrón de diseño FactoryMethod. Así, hemos creado una clase FabricaUsuarios que es la encargada de crear nuevas instancias de las clases Jugador y Administrador.

El siguiente diagrama muestra la clase Fabrica de Usuario. Esta clase será utilizada desde el ControladorInicio cuando el usuario quiera registrar una nueva cuenta.



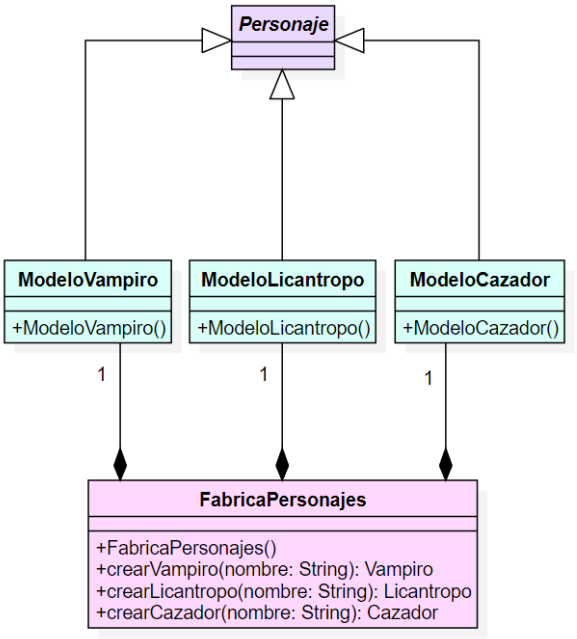
Esta clase se utilizará para crear usuarios de ambos tipos. Además, esta clase utilizará la clase Estado para obtener un número de registro no usado. Esto es necesario ya que todas las instancias de Jugador deberán tener asociado un NumeroRegistro único.

Es importante mencionar que NO utiliza Estado para guardar el Usuario creado. El responsable de que esta acción se lleve a cabo es el ControladorInicio.

Diagrama de Clases de Fábrica de Personajes

En la sección "Diagrama de las clases del juego" se muestra la clase abstracta Personaje, de la que heredan otras tres clases; Vampiro, Licántropo, y Cazador. Para gestionar la creación de instancias de estas clases nos hemos guiado por el patrón de diseño FactoryMethod. Así, hemos creado una clase FabricaPersonajes que es la encargada de crear nuevas instancias de las clases Vampiro, Licántropo, y Cazador.

En el siguiente diagrama se muestra la clase FabricaPersonajes. Esta clase será utilizada desde el controlador CrearPersonaje.

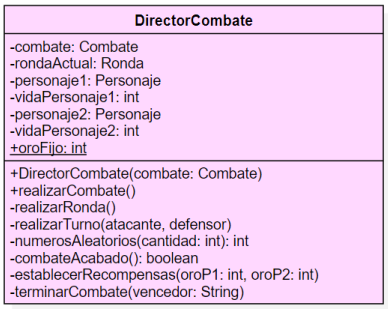


Esta clase se utilizará para crear cualquier tipo de personaje. Utiliza modelos de cada clase para establecer la información predeterminada de un personaje.

Cuando se cree un Personaje, el Usuario decidirá su nombre y su tipo, y todos los demás atributos los decidirá el sistema mediante estos modelos, poniéndole al personaje los mismos valores que estén almacenados en el modelo correspondiente. Cada modelo almacenará valores por defecto para una serie de atributos. Estos valores serán distintos para cada modelo. Los atributos mencionados son: armasDisponibles, armadurasDisponibles, habilidadEspecial, esbirros, debilidades, fortalezas, vida, poder, y descripción.

Diagrama de la clase DirectorCombate

El siguiente diagrama muestra la clase *DirectorCombate*, utilizada para la ejecución de los combates.



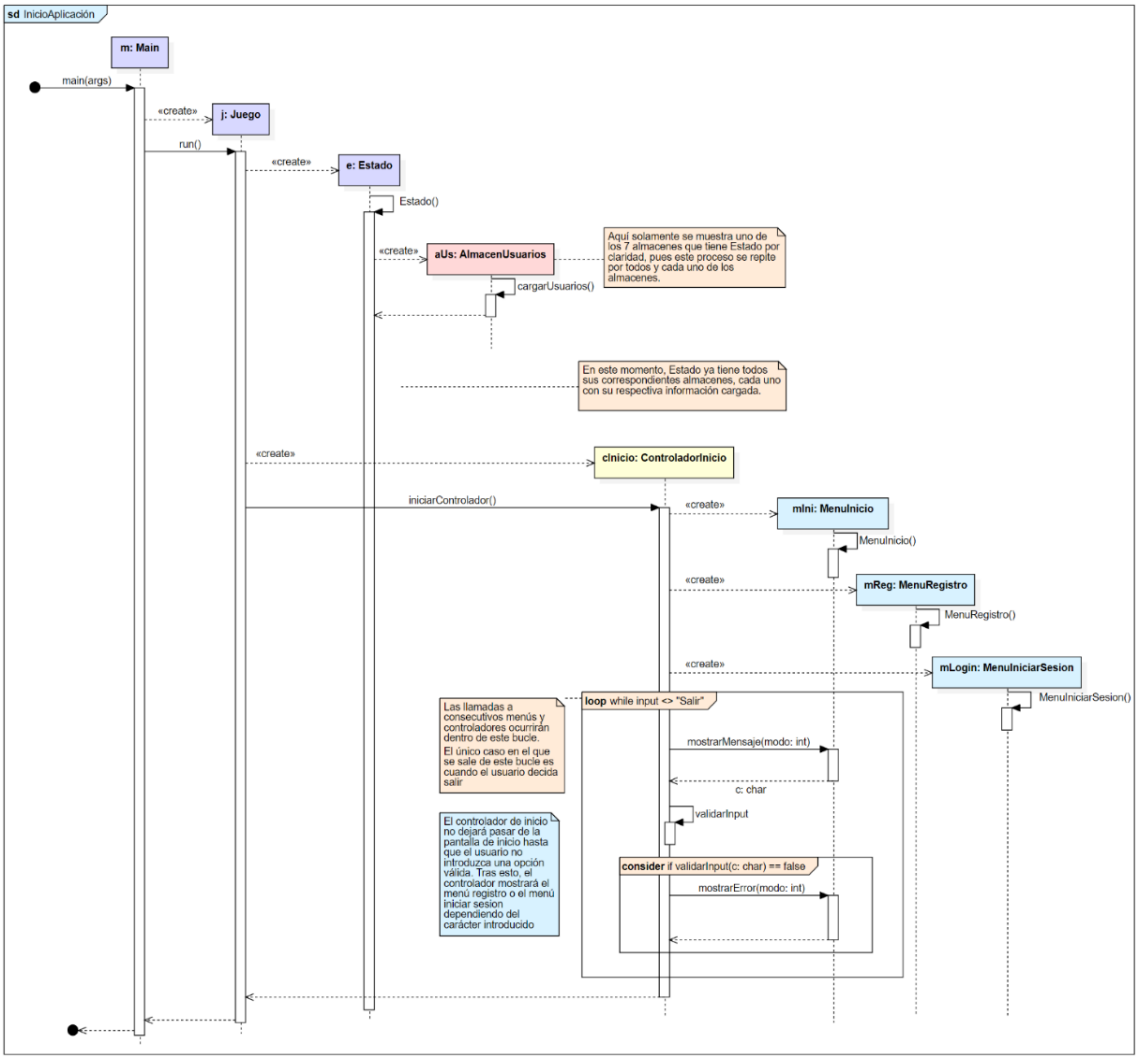
Esta clase contiene todas las funciones necesarias para ejecutar un combate entre dos personajes. Será utilizada por el ControladorNotificacion cuando se acepte un desafío.

Es la encargada de llevar a cabo cada combate, y se comunica con una instancia de la clase Combate para ir registrando lo que ocurre. El único método público que tiene (aparte del constructor) es realizarCombate, que, al ser llamado, utilizará los demás métodos. El funcionamiento en detalle de esta clase se explica en tres diagramas de secuencia, en la sección Combate.

DIAGRAMAS DE SECUENCIA

Inicio de la aplicación

El siguiente diagrama de secuencia muestra los métodos que se invocarán al iniciar el sistema, al igual que los eventos que ocurrirán al salir del mismo, pero omite los pasos intermedios (funcionamiento de los controladores).



Nada más empezar la aplicación, la clase Main creará una instancia de Juego, e invocará su método run. Desde este método, la clase Juego creará una instancia de Estado. Al ser creada esta instancia, el constructor de Estado inicializará sus atributos, creando una serie de clases almacén, y llamando al método equivalente a cargarDatos de cada uno de ellos. Una vez vuelva el control a Juego, creará una instancia de ControladorInicio, y llamará a su método iniciarControlador.

Este método se ocupa de crear las vistas necesarias para el ControladorInicio, y una vez creadas, muestra la primera, MenuInicio, a través del método mostrarMensaje. Este método devolverá el input del usuario. Si el input no formara parte de un rango de inputs válidos, se mostraría un mensaje de error, que le comunicaría este hecho al usuario, y se le pediría de nuevo su input.

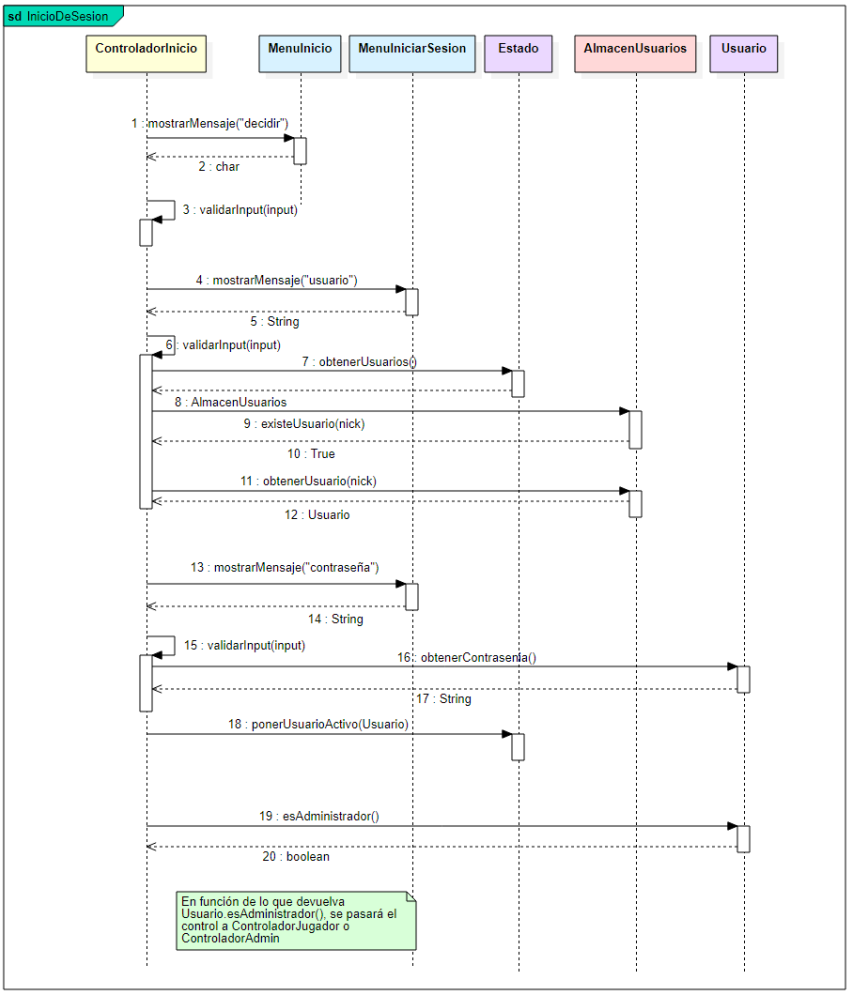
Suponiendo que el input fuera válido, se llevarían a cabo llamadas a los Controladores y Menús pertinentes.

Es importante recalcar que, si bien se puede acceder a otros Controladores y Menús, siempre será desde dentro de este; la única forma de salir de este controlador es que lo decida activamente el usuario.

Cuando el usuario decida salir de este Controlador, se cesará la ejecución del sistema sin llevar a cabo ninguna función adicional.

Inicio de Sesión

El siguiente diagrama de secuencia muestra los métodos que se invocarán durante la navegación del Menú de Inicio. Muestra el supuesto en el que el usuario decide iniciar sesión, y todos los inputs que introduce son válidos, desde el momento en el que se pide el primer input del usuario, hasta el momento en el que se cargaría el siguiente menú.



La secuencia en este caso será la siguiente:

1. Se pregunta al usuario si quiere registrarse o iniciar sesión
2. Pedimos el nombre de usuario
3. Comprobamos que el nombre de usuario sea válido. Para ello:

    - Obtenemos el AlmacenUsuarios de Estado, y a través de él comprobamos que existe un usuario con ese nombre. Si existe, lo cargamos en un atributo
4. Pedimos la contraseña
5. Comprobamos que la contraseña sea válida, comparándola con la contraseña del usuario guardado como atributo.

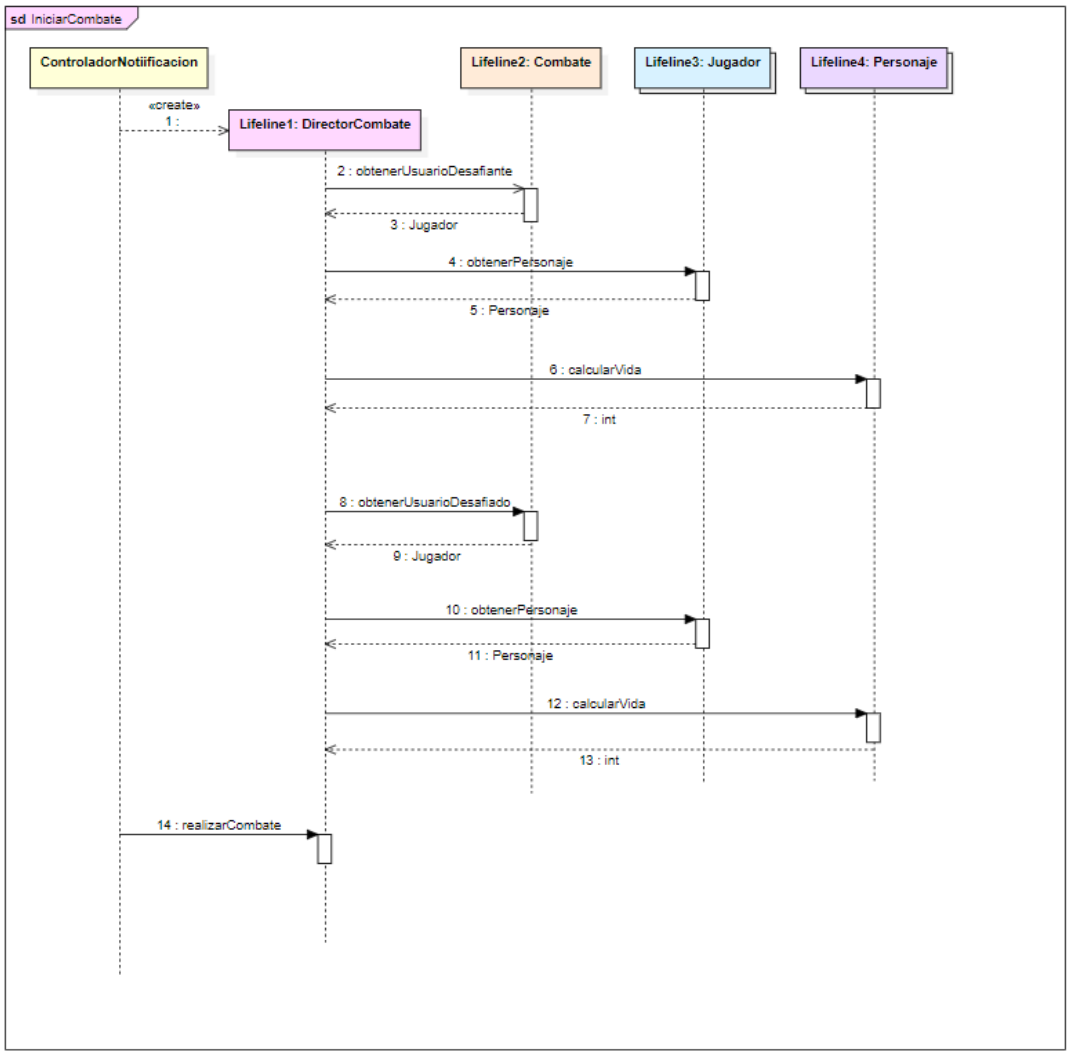
Es importante resaltar lo siguiente: en las llamadas al método mostrarMensaje, el diagrama muestra que se pasa como parámetro una String. Esto no es así en realidad, sino que se pasará un integer. En el diagrama se ha puesto una String para que sea más fácil saber qué mensaje se está mostrando en cada llamada. Así, la llamada mostrarMensaje("decidir") mostrará un mensaje como: "¿Prefiere (1) Iniciar sesión, (2) Registrarse, o (3) Salir?", la llamada mostrarMensaje("usuario") mostrará algo similar a "Introduzca su nombre de usuario", etc.

Combate

En los siguientes diagramas se muestran los métodos invocados durante la realización de un combate. Dada la complejidad y longitud de la secuencia, se ha dividido el combate en tres diagramas: iniciar combate, realizar combate, y finalizar combate.

Iniciar Combate

Cuando un Jugador inicia sesión, si alguien le ha desafiado, el controlador ControladorNotificacion le permitirá aceptarlo o rechazarlo. Si lo acepta, se le permite cambiar su equipo, y una vez satisfecho el usuario, empieza el combate. Lo primero que ocurre al iniciar un combate es que se crea una instancia de DirectorCombate, que se encargará de llevar a cabo el combate. Este diagrama muestra las acciones que lleva a cabo DirectorCombate en su constructor, desde que es llamado, hasta que acaba, y muestra, pero no detalla, el siguiente método llamado.

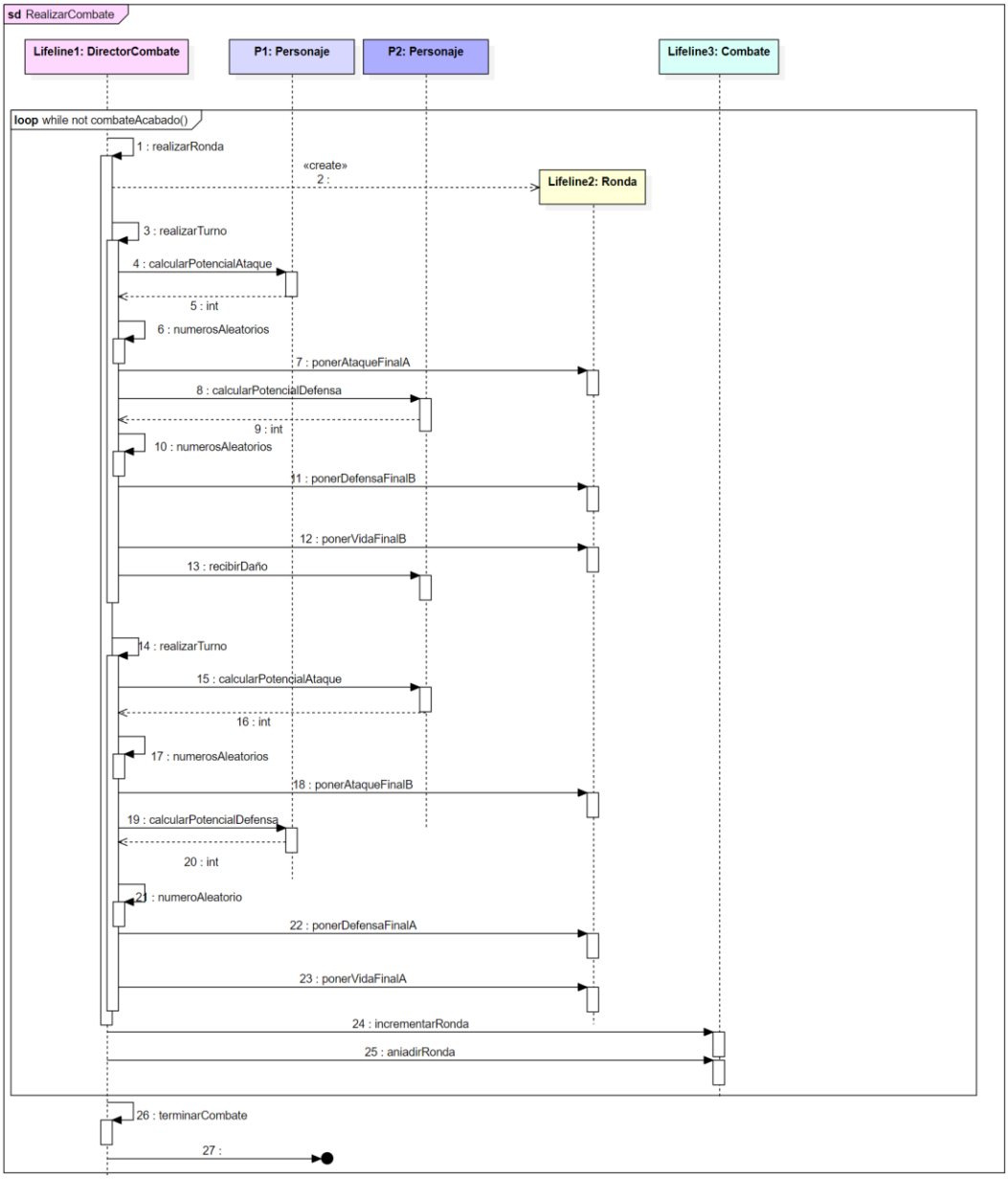


La secuencia de pasos será la siguiente:

1. El ControladorNotificacion crea una instancia de DirectorCombate, pasándole un Combate en el constructor
2. El director de combate pide al Combate el Usuario desafiante.
3. Una vez lo tiene, lo usa para obtener su personaje (con el método obtenerPersonaje).
4. Almacena el personaje en un atributo, y llama a su método calcularVida. Guarda el valor devuelto en otro atributo.
5. Repite los pasos 2, 3, y 4 para el Usuario desafiado
6. El ControladorNotificacion llama a DirectorCombate.realizarCombate().

Realizar Combate

El siguiente diagrama describe el proceso que sigue un combate una vez iniciado.



Un combate consiste en una serie de rondas en las cuales cada personaje ataca al otro, y puede o no hacerle daño. Cuando uno de los dos personajes quede sin puntos de vida, el combate acaba. El método realizarCombate por consiguiente contendrá un bucle cuya condición será la "muerte" de uno de los personajes, y, mientras no se cumpla esa condición, llamará al método realizarRonda.

Cada ronda a su vez consta de dos turnos. En un turno ataca el personaje A y defiende el B, y en el siguiente ataca el personaje B y defiende el A. Así, el método realizarRonda consistirá en dos llamadas al método realizarTurno. Sin embargo, la ronda también debe guardar la vida inicial de los personajes, por lo que antes de hacer las llamadas a realizarTurno, guardará en la Ronda las vidas iniciales de ambos personajes (esto no está reflejado en el diagrama).

El método realizarTurno sigue los siguientes pasos:

1. Calcula el potencial de ataque del personaje atacante

2. Calcula (y guarda) el ataque real del personaje atacante
3. Calcula el potencial de defensa del personaje defensor
4. Calcula (y guarda) la defensa real del personaje atacante
5. Compara el ataque y defensa reales. Si el ataque es mayor o igual que la defensa, llama al método recibirDaño del personaje defensor, y le resta uno a su vida. Si la defensa es mayor, no hace nada
6. Guarda la vida final del personaje defensor en la Ronda.

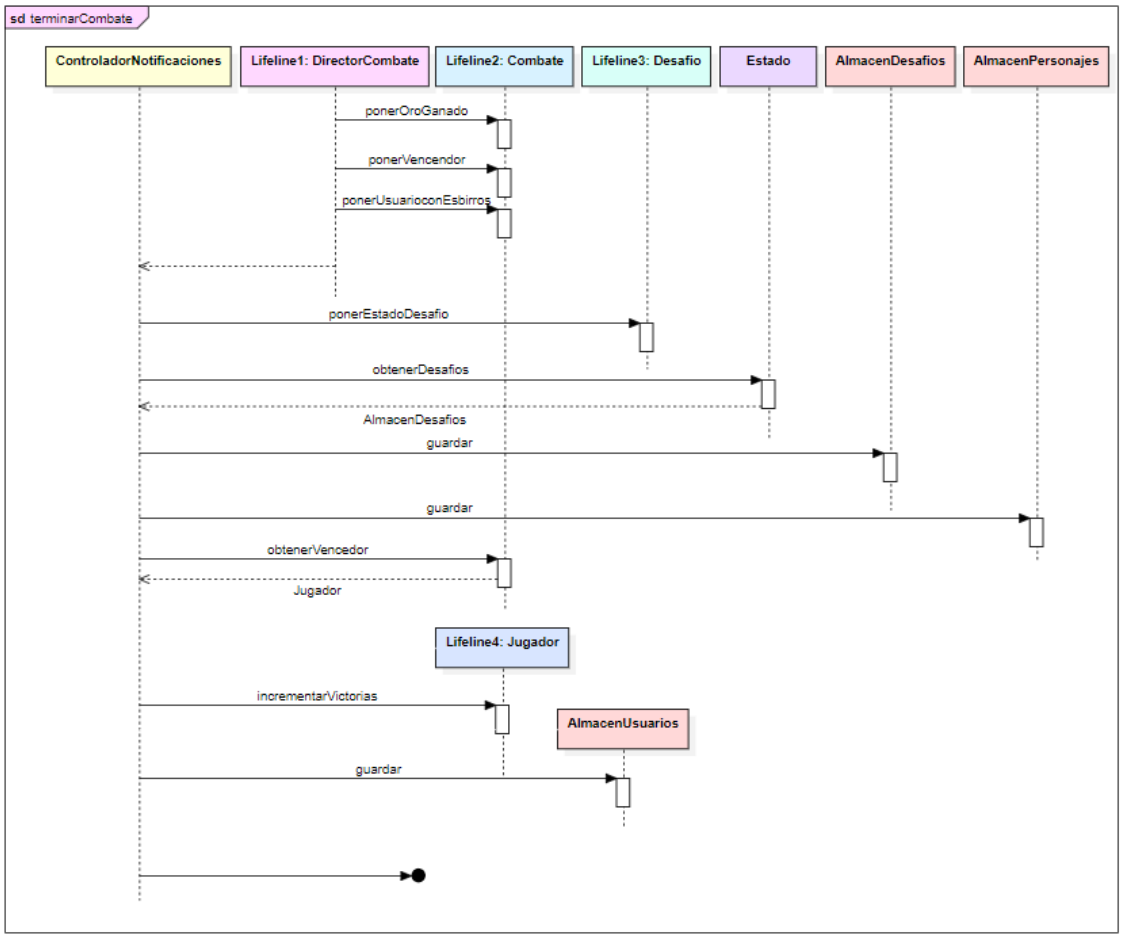
En los pasos 2 y 4 el ataque y defensa reales se calculan generando tantos números aleatorios entre 1 y 6 como indique el potencial de ataque/defensa. Luego, se cuenta el número de cincos y seises obtenidos, y ese será el valor de ataque/defensa real.

Al acabar ambos turnos, iniciarCombate llama a los métodos incrementarRonda y aniadirRonda, para que el combate guarde la ronda que acaba de llevarse a cabo. Luego, borra la ronda guardada en el atributo rondaActual, y lo repite todo si ambos personajes tienen al menos un punto de vida.

El diagrama muestra una instancia de este proceso en la cual el personaje A hiere al personaje B, pero el B no hiere al A.

Terminar combate

El último diagrama del combate recoge las acciones que se llevan a cabo una vez ha acabado el combate, en el método terminarCombate.



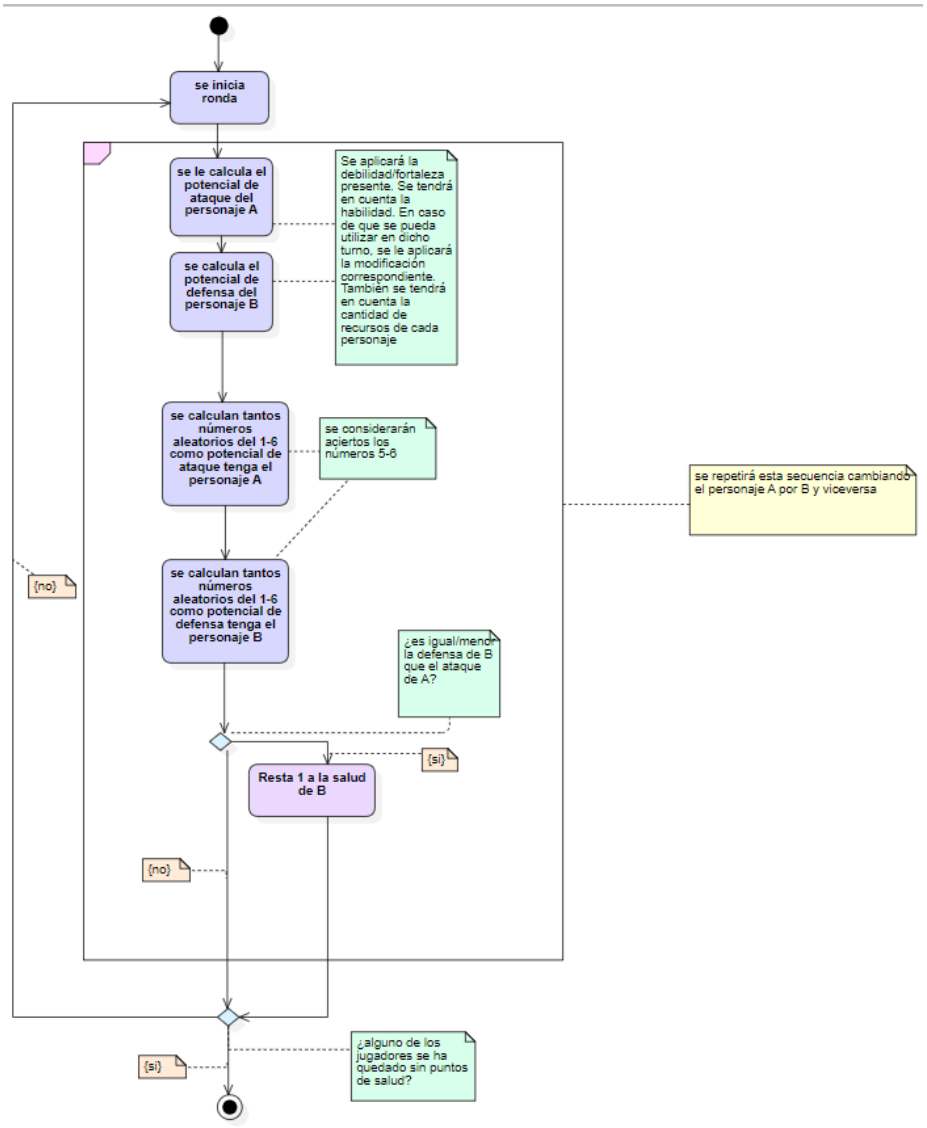
Al ser llamado este método, el DirectorCombate actualiza Combate, añadiéndole la información que depende del vencedor.

Luego, se devuelve el control a ControladorNotificaciones, que cambia el estado del desafío a completado, y actualiza el archivo que contiene los desafíos para que refleje el cambio. De igual manera, actualiza el archivo que contiene los personajes para que refleje el cambio en oro de los mismos, y por último, le incrementa el número de victorias al ganador, y actualiza el almacén de usuarios para que refleje ese cambio.

DIAGRAMAS DE ACTIVIDAD

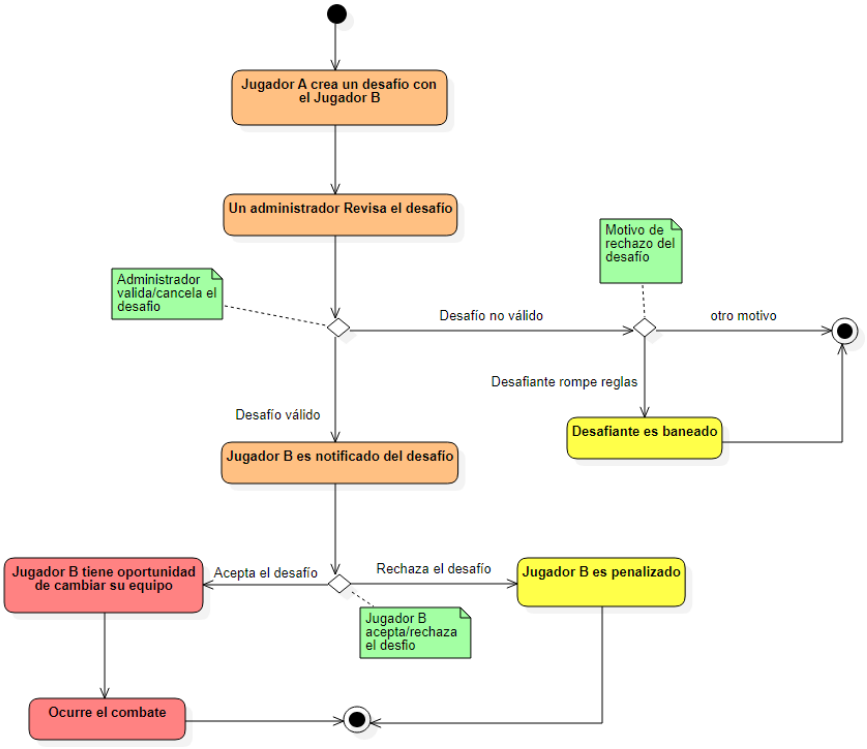
Combate

En el siguiente diagrama se explica el flujo que sigue la ejecución de un combate.



Desafíos

A continuación, se muestra un diagrama de actividad que recoge las acciones que ocurren a lo largo de la vida de un desafío.

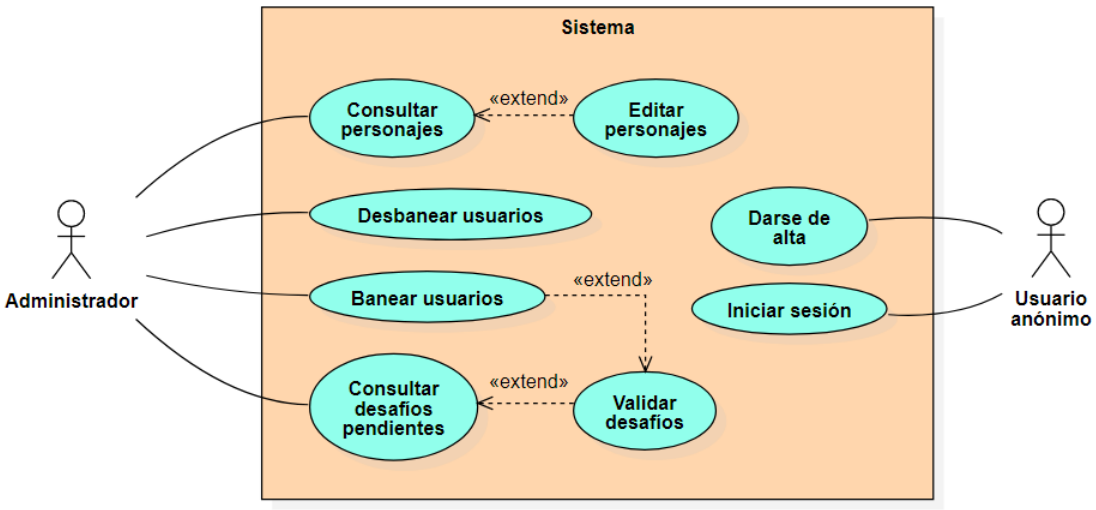


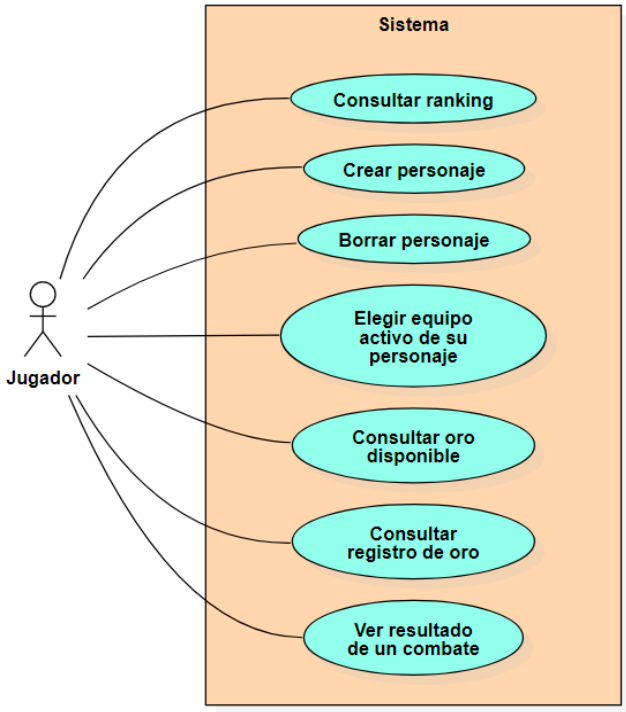
Si el desafío acaba en el nodo fin de abajo del diagrama, entonces el desafío se guarda. Si ha ocurrido el combate, se guarda con el combate asociado, mientras que, si ha sido rechazado, no hay combate que guardar, y el desafío se guardará como “rechazado”, sin ningún combate asociado.

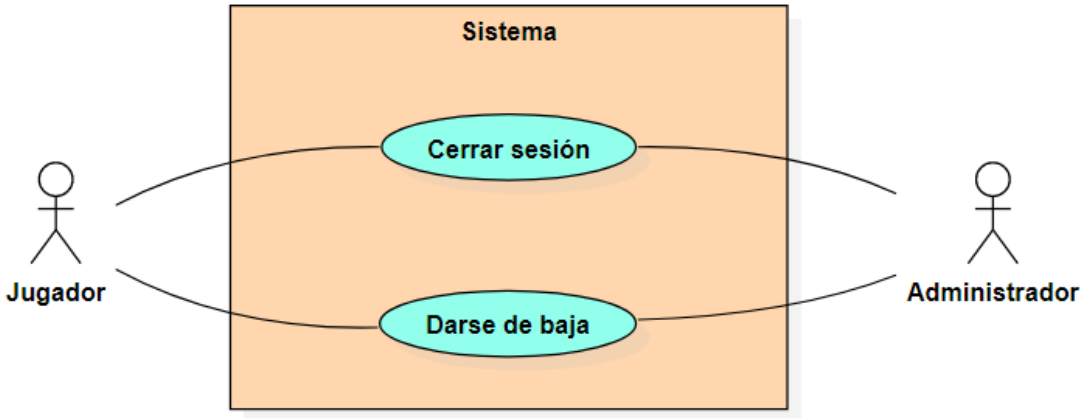
Por otro lado, si el desafío acaba en el nodo fin de la derecha del diagrama, entonces no se guardará, ya que no era un desafío válido, sino que se borrará, y se devolverá el dinero al usuario desafiante.

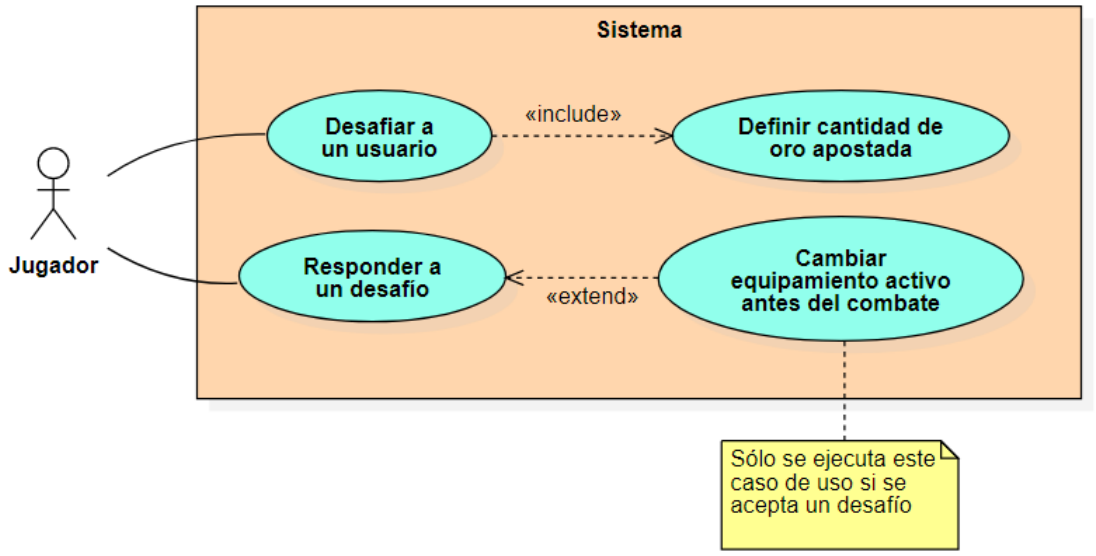
DIAGRAMAS DE CASOS DE USO

Los siguientes diagramas muestran los casos de uso del software, en función de los permisos que tenga cada usuario.


Sistema
Consultar personajes
«extend»
Editar personajes
Desbanear usuarios
Darse de alta
Banear usuarios
«extend»
Iniciar sesión
Consultar desafíos pendientes
«extend»
Validar desafíos
Administrador
Usuario anónimo

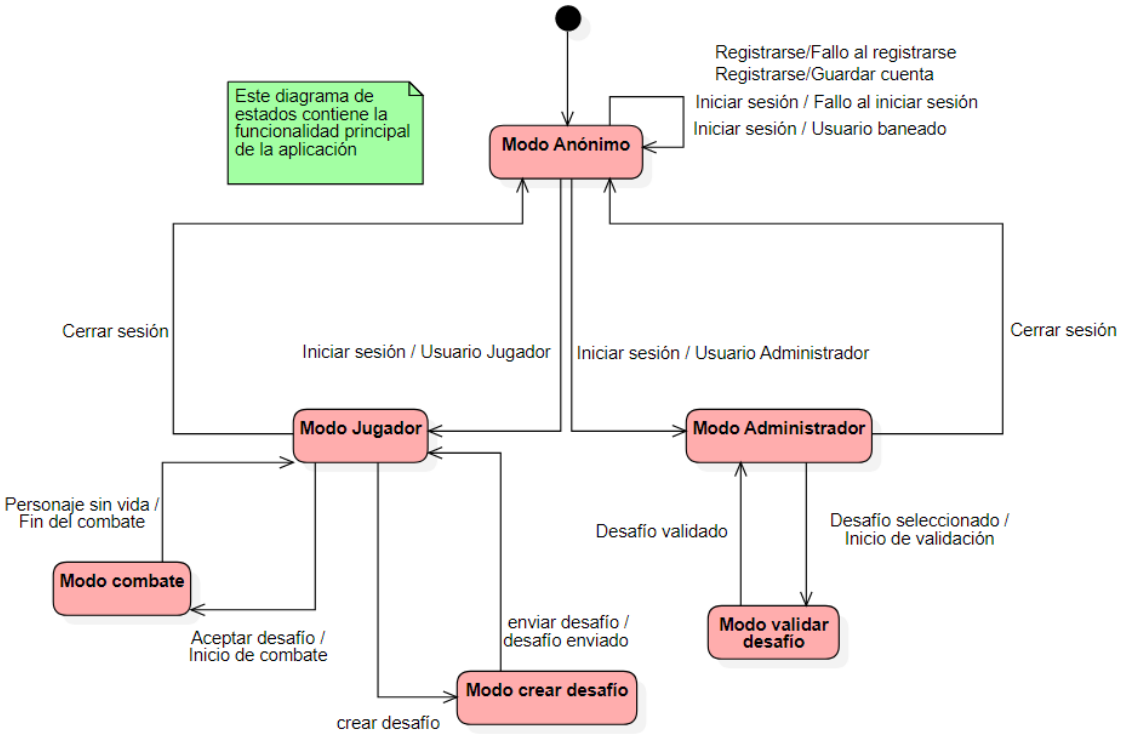

Sistema
Consultar ranking
Crear personaje
Borrar personaje
Elegir equipo activo de su personaje
Consultar oro disponible
Consultar registro de oro
Ver resultado de un combate
Jugador


Sistema
Cerrar sesión
Darse de baja
Jugador
Administrador



DIAGRAMAS DE ESTADOS

El siguiente diagrama muestra los estados principales por los que se moverá un usuario al llevar a cabo la funcionalidad principal de la aplicación (el sistema de desafíos y combates).



La aplicación empezará en modo anónimo, hasta que un usuario inicie sesión. Cuando esto ocurra, se comprobará qué tipo de usuario ha iniciado sesión, y se irá al estado pertinente.

Si ha iniciado sesión un administrador, pasaremos al Modo Administrador, desde el cual se podrán hacer una serie de acciones. Una de esas acciones consiste en seleccionar un desafío para validar. Cuando eso ocurra, se pasará al Modo Validar Desafío, desde donde se llevarán a

cabo todos los pasos necesarios para validar un desafío. Una vez el administrador de por completada la validación, volverá al Modo Administrador.

Si el usuario inicia sesión como jugador, pasamos al Modo Jugador. Desde este Modo, podremos crear desafíos y aceptarlos. Si queremos crear un desafío pasaremos al modo crear desafío, donde podremos decidir a quién desafiar y por cuanto oro. Una vez hayamos creado nuestro desafío, lo enviaremos, volviendo al modo jugador. Si decidiéramos aceptar un desafío, pasaríamos al modo combate, donde el combate ocurriría automáticamente. Al acabar el combate, volveríamos al modo jugador.

Cuando se cierra sesión desde los modos jugador y administrador, se vuelve al modo anónimo.

MAPA DE NAVEGACIÓN

En el siguiente mapa de navegación se representará los distintos menús a los que el usuario podrá acceder, dependiendo de si es jugador o administrador. A partir de los menús que el usuario desee seleccionar, se representará el flujo de menús que podrá escoger.