

1. Análisis del problema y requisitos del sistema

El ejercicio plantea crear un diseño con el objetivo de gestionar torneos de eSports, contemplando todas las fases del proceso: desde la creación de equipos hasta la asignación de premios. Para ello, debemos empezar identificando a los actores implicados, las acciones que realizan y las entidades fundamentales que componen el sistema.

Actores principales

- **Administrador:** Es el actor principal y encargado de operar el sistema. Tiene acceso a todas las funcionalidades relacionadas con la gestión de equipos, jugadores, torneos, resultados y premios.
- **Jugador:** Forma parte de un equipo y puede ser inscrito en torneos, aunque no interactúa directamente con el sistema.
- **Sistema:** Ejecuta procesos automáticos como la generación de emparejamientos y la actualización de la clasificación tras cada partida.

Funcionalidades del sistema

El sistema ofrece al administrador un conjunto completo de acciones para controlar y supervisar los torneos:

- Registrar nuevos equipos.
- Añadir jugadores a los equipos registrados.
- Consultar listas actualizadas de equipos y jugadores.
- Crear torneos.
- Inscribir equipos en torneos concretos.
- Generar emparejamientos de partidas automáticamente.
- Registrar los resultados de las partidas jugadas.
- Asignar premios a los ganadores según la clasificación.

Entidades clave y sus relaciones

El sistema se estructura en torno a una serie de entidades que se relacionan de forma lógica entre sí:

- Un **Equipo** está compuesto por uno o varios **Jugadores**.
- Un **Torneo** agrupa varios **Equipos** que compiten entre sí.
- Cada **Torneo** contiene varias **Partidas**, donde se enfrentan los equipos.
- Las **Partidas** tienen un resultado que influye directamente en la **Clasificación** del torneo.
- Finalmente, un **Premio** se otorga al equipo (o jugador) vencedor en función de su desempeño.

2. Identificación de los casos de uso y elaboración del diagrama

Para comprender cómo interactúan los actores con el sistema, necesitamos identificar los **casos de uso** más relevantes. Esto permite representar gráficamente las funcionalidades clave mediante un **diagrama de casos de uso UML**, facilitando así el análisis y diseño del sistema.

Interacciones clave del Administrador

El administrador es el actor principal, con acceso a las siguientes funcionalidades:

- **Registrar equipo:** Permite crear un nuevo equipo introduciendo los datos básicos.
- **Añadir jugadores a un equipo:** Asocia uno o varios jugadores a un equipo existente.
- **Consultar lista de equipos y jugadores:** Visualiza los equipos registrados junto a sus respectivos miembros.
- **Crear torneo:** Define un nuevo torneo, estableciendo su nombre y configuración básica.
- **Inscribir equipo en un torneo:** Añade un equipo ya registrado a un torneo activo.
- **Generar emparejamientos de partidas:** Crea automáticamente las partidas entre los equipos inscritos.
- **Registrar resultado de una partida:** Introduce el resultado de una partida jugada.
- **Asignar premios a los ganadores:** Establece qué premios corresponden a los equipos mejor clasificados.

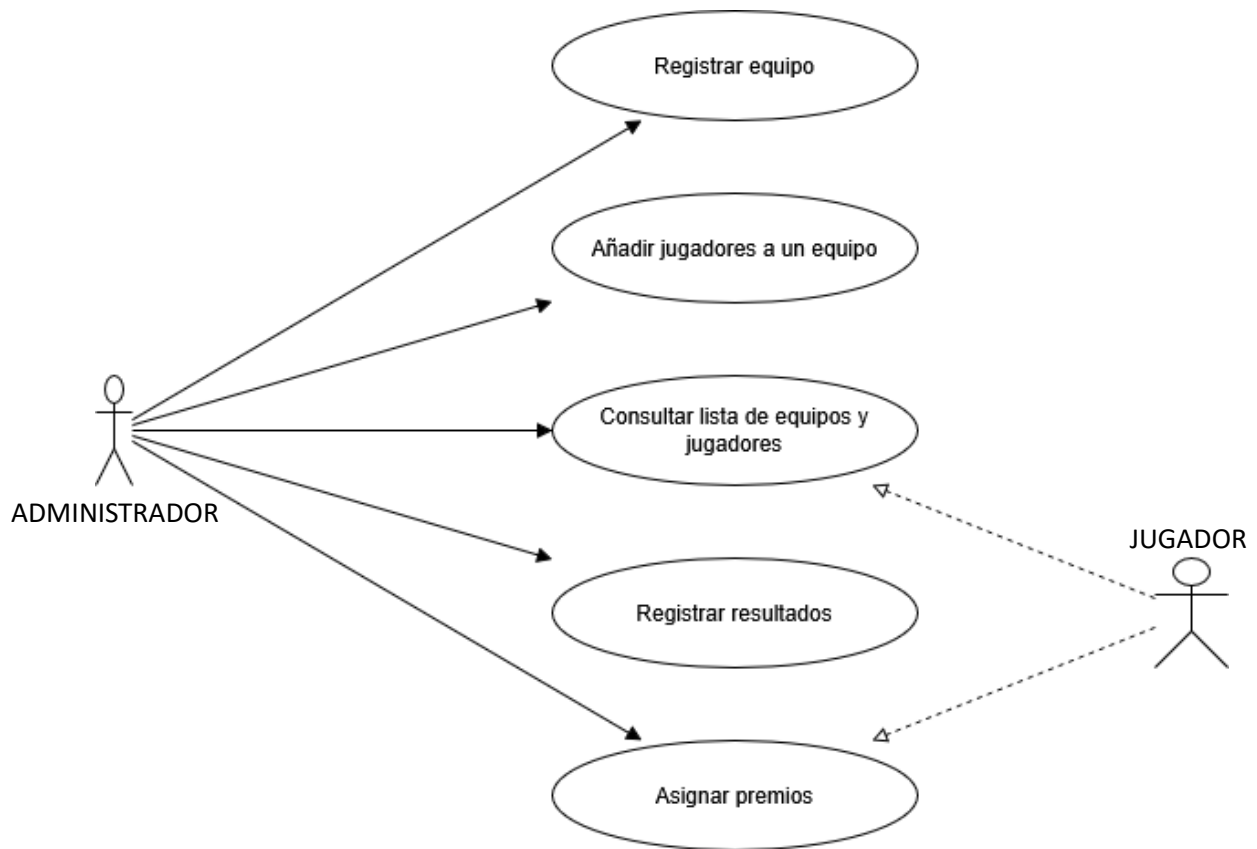
Participación del Jugador

Aunque el jugador no interactúa directamente con el sistema, está implicado de forma indirecta en los siguientes casos de uso:

- **Consultar lista de equipos y jugadores** (*relación «include»*): Puede ser accedido por el jugador a través de medios de visualización externos, como una interfaz pública.

Relaciones entre casos de uso

Se han establecido relaciones «include» para representar aquellos casos de uso que forman parte de otros. Por ejemplo, la consulta de equipos puede estar incluida tanto en la gestión de equipos como en la de torneos, ya que es una funcionalidad de apoyo común.



3. Identificación de clases y relaciones

Una vez definidos los casos de uso, vamos a estructurar el sistema desde el punto de vista de la programación orientada a objetos. Para ello, empezamos identificando las clases principales del sistema y clasificamos según su rol dentro de una arquitectura basada en el patrón **Modelo-Vista-Controlador (MVC)**.

Clases de entidad (Modelo)

Estas clases representan los elementos fundamentales del dominio del problema, es decir, los objetos reales que participan en los torneos de eSports:

- **Equipo:** Representa un equipo participante. Contiene atributos como el nombre del equipo y una lista de jugadores.
- **Jugador:** Cada jugador posee información como nombre, alias y edad.
- **Torneo:** Gestiona los datos del torneo, incluyendo los equipos inscritos y las partidas programadas.
- **Partida:** Representa un enfrentamiento entre dos equipos y almacena su resultado.
- **Premio:** Describe el tipo de premio y el equipo ganador al que se asigna.

Clases de control

Estas clases encapsulan la lógica del sistema. Se encargan de coordinar el flujo de información entre el modelo y la vista:

- **GestorEquipos:** Se encarga de registrar equipos, añadir jugadores y consultar la lista de equipos.
- **GestorTorneos:** Permite crear torneos, inscribir equipos y generar los emparejamientos.
- **GestorResultados:** Gestiona la entrada de resultados de partidas y la actualización de la clasificación del torneo.
- **GestorPremios:** Se encarga de asignar premios a los ganadores.

Clases de interfaz (Vista)

Estas clases están orientadas a la interacción con el usuario. Pueden implementarse como interfaces gráficas o de consola:

- **VistaPrincipal:** Controla el menú principal del sistema y da acceso a las demás vistas.
- **VistaEquipos, VistaTorneos, VistaResultados:** Especializadas en la gestión visual de sus respectivas funcionalidades.

Relaciones entre clases

Se han identificado diferentes tipos de relaciones entre las clases del sistema:

- **Asociación:** Un equipo está asociado a varios jugadores; una partida está asociada a dos equipos.
- **Agregación:** Un torneo contiene una colección de equipos, pero estos pueden existir de forma independiente.
- **Composición:** Las partidas forman parte inseparable del torneo, por lo que la relación es de composición.
- **Dependencia:** Las vistas dependen de los gestores de control para ejecutar la lógica.

4. Creación del diagrama de clases UML

Una vez identificadas las clases principales del sistema y sus relaciones, podemos crear el **diagrama de clases UML**, que representa gráficamente la estructura interna del sistema. Este diagrama permite visualizar de manera clara cómo se organizan las clases, qué atributos y métodos contiene cada una, y cómo se conectan entre ellas.

Estructura general del diagrama

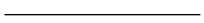
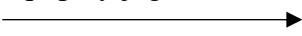
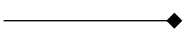
El diagrama se ha diseñado siguiendo el patrón **Modelo-Vista-Controlador (MVC)**, lo que facilita una organización modular y escalable del sistema:

- En la parte superior del diagrama se encuentran las **clases de entidad**, representando los datos esenciales del sistema.
- En la zona intermedia se sitúan las **clases de control**, responsables de gestionar la lógica de negocio.
- En la parte inferior se ubican las **clases de interfaz**, que permiten la interacción entre el sistema y el usuario.

Contenido de las clases

Cada clase muestra de forma explícita:

- **Atributos:** con su nombre, tipo de dato y visibilidad (- para privado, + para público).
- **Métodos:** con su nombre, parámetros, tipo de retorno y visibilidad.
- **Relaciones:** usando las notaciones estándar UML (líneas, flechas, diamantes...) para representar:

- Asociaciones simples (por ejemplo, entre equipo y jugador). 
- Agregaciones (entre torneo y equipos). 
- Composiciones (entre torneo y partidas). 
- Dependencias (entre vistas y controladores). 