**1. Análisis del problema y requisitos del sistema**

**Descripción general**

El sistema que se va a desarrollar es una aplicación de gestión de torneos de eSports. Su finalidad es facilitar la organización completa de un torneo: desde el registro de los equipos y jugadores, hasta la creación del torneo, la inscripción de los participantes, la gestión de resultados de las partidas y la asignación de premios.

Este sistema debe ser utilizado principalmente por un **Administrador**, quien se encarga de la configuración, seguimiento y gestión de toda la información. Por otro lado, los **Jugadores** no gestionan datos, pero sí pueden consultar información relevante como equipos, jugadores, resultados y premios.

El modelado del sistema se realiza utilizando **diagramas de casos de uso UML**, los cuales ayudan a visualizar las principales funcionalidades desde la perspectiva de los actores que interactúan con él.

**¿Quiénes son los actores que interactúan con el sistema?**

Según el diagrama de casos de uso diseñado, se identifican dos actores principales:

**1. Administrador**

* Es el actor responsable de todas las operaciones administrativas del sistema.
* Tiene acceso completo a todas las funcionalidades.
* Se encarga de registrar entidades, gestionar el desarrollo del torneo y asignar premios.

**2. Jugador**

* Es un actor secundario, sin capacidad de modificación de datos.
* Su papel es únicamente **consultivo**: accede al sistema para visualizar información sobre su equipo, resultados y premios.
* No ejecuta acciones de gestión.

**¿Cuáles son las acciones que cada actor puede realizar?**

**Acciones del Administrador:**

* **Registrar equipo:** crea un nuevo equipo en el sistema.
* **Añadir jugadores:** vincula jugadores a equipos.
* **Registrar jugadores:** introduce los datos individuales de cada jugador.
* **Consultar equipos:** visualiza los equipos registrados.
* **Consultar jugadores:** visualiza los jugadores asociados a cada equipo.
* **Crear torneo:** da de alta un nuevo torneo.
* **Inscribir equipos:** añade equipos al torneo activo.
* **Registrar resultados:** guarda los resultados de las partidas jugadas.
* **Consultar resultados:** revisa los resultados ya registrados.
* **Asignar premios:** otorga premios a los equipos ganadores.
* **Consultar premios:** revisa los premios ya asignados.

**Acciones del Jugador:**

* **Consultar equipos:** puede ver los equipos registrados en el torneo.
* **Consultar jugadores:** puede ver los miembros de cada equipo.
* **Consultar resultados:** puede ver los resultados de las partidas jugadas.
* **Consultar premios:** puede ver los premios asignados por el administrador.

**¿Cómo se relacionan entre sí las entidades del sistema?**

A partir de los casos de uso, se infieren las siguientes relaciones entre entidades:

* Un **equipo** está compuesto por **uno o más jugadores**. Esta relación es de tipo **composición**, ya que los jugadores no existen en el sistema si no están vinculados a un equipo.
* Un **torneo** contiene **varios equipos inscritos**. Esta relación es de tipo **agregación**, ya que los equipos pueden existir fuera del torneo.
* Un **torneo** incluye **varias partidas**, cada una enfrentando a dos equipos. Las partidas solo existen en el contexto del torneo.
* Cada **partida** genera un **resultado**, que registra qué equipo ganó y con qué puntuación.
* Un **premio** se asigna a un **equipo ganador** una vez finalizado el torneo.

Además, se identifican relaciones funcionales entre los casos de uso:

* La acción **"Registrar equipo"** incluye **"Añadir jugadores"**, que a su vez incluye **"Registrar jugadores"**.
* La consulta de equipos puede **extenderse** con la consulta de jugadores.
* La acción **"Crear torneo"** incluye la inscripción de equipos como paso obligatorio.

Estas relaciones han sido representadas en el diagrama mediante las relaciones <<include>> y <<extend>>.

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

**Objetivo**

El propósito de este paso es identificar las **interacciones clave entre los actores y el sistema** en función de las funcionalidades que ofrece. Se han agrupado los **casos de uso por áreas funcionales**, facilitando así el diseño del sistema en módulos y permitiendo una implementación más clara y ordenada en futuras fases del desarrollo.

Para ello se ha elaborado un **diagrama de casos de uso UML**, el cual representa visualmente los actores implicados y las acciones que cada uno puede ejecutar sobre el sistema, utilizando relaciones <<include>> y <<extend>> cuando es necesario.

**Actores del sistema**

* **Administrador**: Usuario principal. Gestiona toda la actividad del sistema: equipos, jugadores, torneos, partidas y premios.
* **Jugador**: Usuario pasivo. No realiza acciones de gestión, pero puede consultar información pública del sistema.

**Casos de uso del sistema**

Los casos de uso se han clasificado en cuatro grandes áreas funcionales:

**Gestión de equipos y jugadores:**

En esta área se agrupan los casos de uso relacionados con el registro y consulta de los equipos participantes y sus miembros.

* **Registrar equipo**  
  Acción realizada por el administrador para crear un nuevo equipo en el sistema.
* **Añadir jugadores a un equipo**  
  Acción que permite asociar jugadores a un equipo previamente creado.
* **Registrar jugadores**  
  Caso de uso incluido dentro de "Añadir jugadores", ya que cada jugador debe ser registrado en el sistema antes de asignarlo a un equipo.  
  Relación utilizada: <<include>>.
* **Consultar lista de equipos y jugadores**  
  Esta funcionalidad se modela con dos casos de uso:
  + **Consultar equipos**: visualiza los equipos registrados.
  + **Consultar jugadores**: se activa opcionalmente tras consultar un equipo específico.  
    Relación utilizada: <<extend>>.

**Gestión de torneos:**

Aquí se encuentran los casos necesarios para la creación de torneos y la inscripción de los equipos participantes.

* **Crear torneo**  
  Permite al administrador dar de alta un nuevo torneo en el sistema.
* **Inscribir equipos en el torneo**  
  Una vez creado el torneo, se debe registrar qué equipos participarán.  
  Relación utilizada: Crear torneo <<include>> Inscribir equipos.

**Gestión de partidas y resultados:**

Estas funcionalidades permiten registrar el desarrollo del torneo y mantener actualizada su clasificación.

* **Registrar resultado de una partida**  
  Permite guardar el resultado de una partida jugada entre dos equipos.
* **Consultar resultados**  
  Disponible tanto para el administrador como para los jugadores. Este caso de uso **no depende directamente** de registrar un resultado, por tanto, se accede de forma independiente (sin <<extend>>).

**Gestión de premios:**

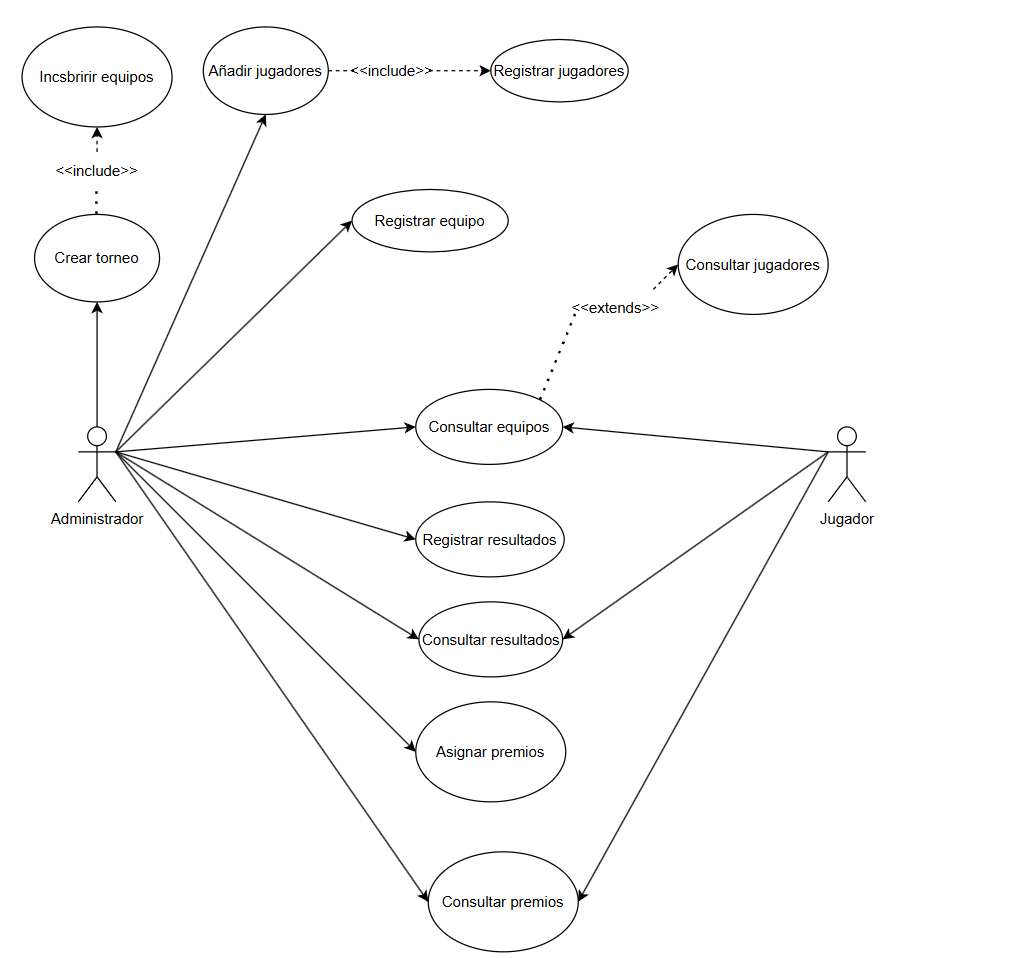
El cierre del torneo incluye la asignación de recompensas a los equipos ganadores.

* **Asignar premios a los ganadores**  
  Solo disponible para el administrador. Se otorgan premios tras registrar los resultados finales del torneo.
* **Consultar premios**  
  Permite a jugadores y al administrador consultar los premios asignados a cada equipo. No depende directamente de "Asignar premios", por lo que se accede por separado.

Relaciones entre casos de uso

En el diagrama de casos de uso se han representado varias relaciones especiales:

* Registrar equipo incluye Añadir jugadores, y este a su vez incluye Registrar jugadores.
* Consultar equipos puede extenderse opcionalmente a Consultar jugadores.
* Crear torneo incluye Inscribir equipos.



**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 las 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:

* **Jugador**: Representa a cada jugador con atributos como nombre, alias (nickname) y edad.
* **Equipo**: Agrupa a varios jugadores y tiene un nombre identificativo.
* **Torneo**: Contiene los datos del torneo, como los equipos inscritos y las partidas generadas.
* **Partida**: Representa un enfrentamiento entre dos equipos y almacena su resultado.
* **Premio**: Define el tipo de premio y el equipo al que se le asigna como ganador.
* **Resultado**: Recogera el resultado de las partidas y el equipo ganador.

**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**: Gestiona el registro de equipos, la incorporación de jugadores y la consulta de la información asociada.
* **GestorTorneos**: Permite crear torneos, inscribir equipos y generar los emparejamientos entre ellos.
* **GestorResultados**: Administra el registro de los resultados de las partidas y actualiza la clasificación general.
* **GestorPremios**: Se ocupa de asignar los premios correspondientes a los equipos ganadores al finalizar el torneo.

**Clases de interfaz (Vista)**

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

* **VistaPrincipal**: Controla el menú principal y da acceso a las distintas funcionalidades del sistema.
* **VistaEquipos, VistaTorneos, VistaResultados**: Son vistas especializadas que permiten mostrar o introducir información sobre equipos, torneos o resultados.

**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 fuera del torneo.
* **Composición**: Las partidas forman parte inseparable del torneo. Si se elimina el torneo, se eliminan también sus partidas.
* **Dependencia**: Las vistas dependen de los gestores de control para ejecutar las acciones del sistema.

**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**, que representan los datos principales del sistema.
* En la zona intermedia se sitúan las **clases de control**, que gestionan la lógica de negocio.
* En la parte inferior se ubican las **clases de interfaz**, que permiten la interacción entre el usuario y el sistema.

**Contenido de las clases**

Cada clase muestra de forma explícita:

* **Atributos**, con su nombre, tipo de dato y visibilidad (- para privado).
* **Métodos**, con su nombre, parámetros, tipo de retorno y visibilidad (+ para público).
* **Relaciones**, utilizando las notaciones estándar UML:
  + Asociaciones simples, como entre equipo y jugador.
  + Agregaciones, como entre torneo y equipos.
  + Composiciones, como entre torneo y partidas.
  + Dependencias, como entre las vistas y los controladores.

