



Sistema de Gestión de Torneos

Departamento Ingeniería Civil Informática

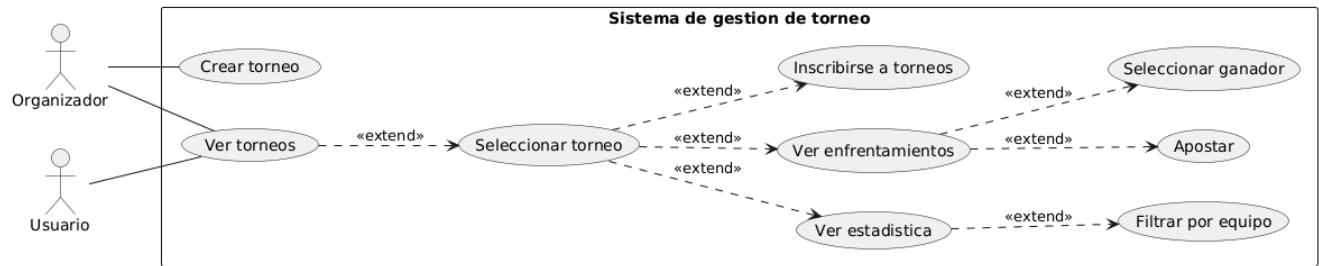
Grupo 17: Agustín Salgado, Ignacio Silva, Juan Pablo Núñez

Profesor Geoffrey Hecht

Repositorio: <https://github.com/elamql/Proyecto-DOO.git>



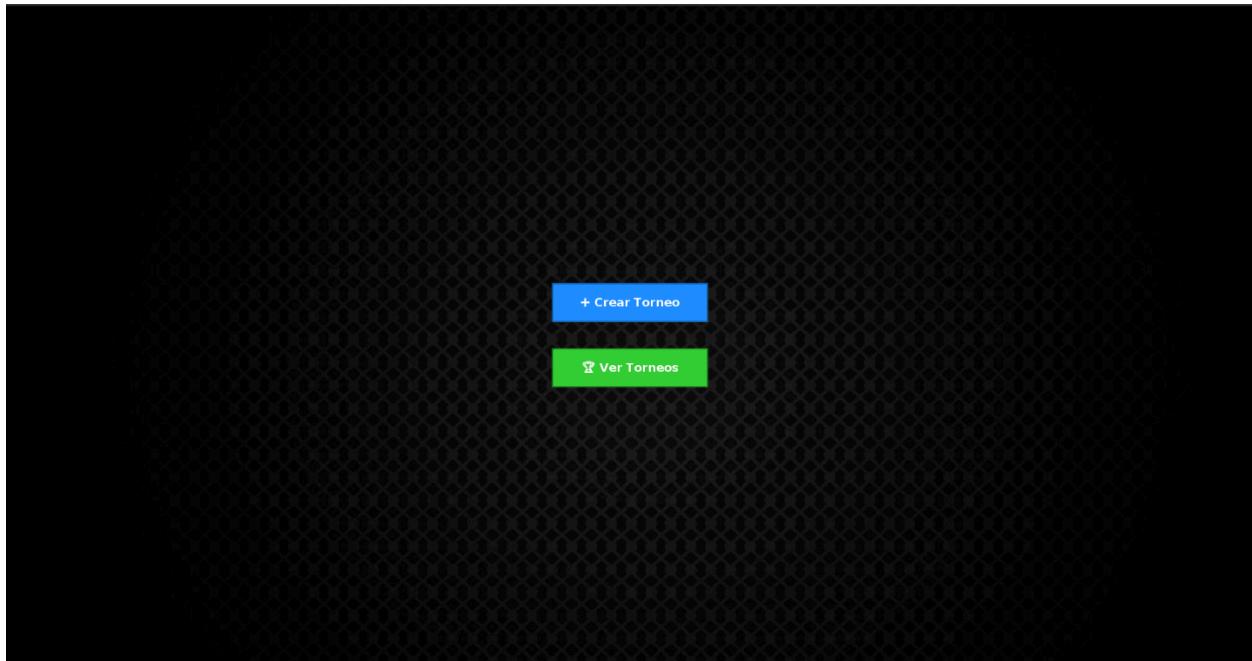
Diagrama de casos de uso



El diagrama de casos de uso muestra las funcionalidades del sistema de gestión de torneo, las interacciones que tienen el usuario y el organizador.

Interfaz

- Menú principal





- Vista de enfrentamientos

Estado: Todos ▾ Disciplina: Todas las disciplinas ▾ Formato: Todos los formatos ▾

Torneo	Disciplina	Formato	Estado	Ver
Torneo Fútbol	FUTBOL	ELIMINATORIA	Activo	Ver
Champions FIFA	FIFA	ELIMINATORIA	Activo	Ver
Roland Garros	TENIS	LIGA	Activo	Ver
Ping Pong Liga	TENIS_DE_MESA	LIGA	Activo	Ver
Copa Ajedrez	AJEDREZ	ELIMINATORIA	Activo	Ver
Worlds Lol	LOL	LIGA	Activo	Ver
LoL	LOL	ELIMINATORIA	Por empezar	Ver

<- Volver

- Torneo ejemplo

TORNEO FÚTBOL

ELIMINATORIA de FUTBOL

Fecha:	01-06-2025 10:00
Participantes:	4
Estado:	Cerrado (Torneo en curso)

<- Volver Ver Enfrentamientos Ver Estadísticas



- Enfrentamientos de torneo ejemplo con 4 participantes

Enfrentamientos del Torneo

Puntos: 1000

Partido 1: Equipo A vs Equipo B

Seleccionar Ganador | Apostar

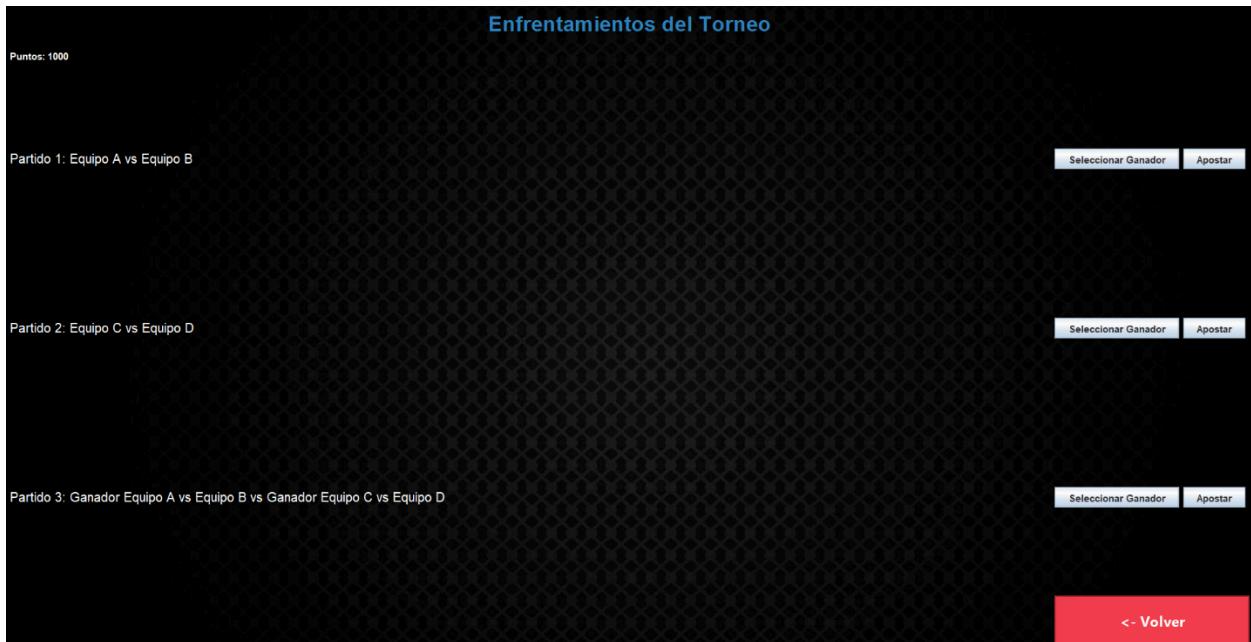
Partido 2: Equipo C vs Equipo D

Seleccionar Ganador | Apostar

Partido 3: Ganador Equipo A vs Equipo B vs Ganador Equipo C vs Equipo D

Seleccionar Ganador | Apostar

<- Volver



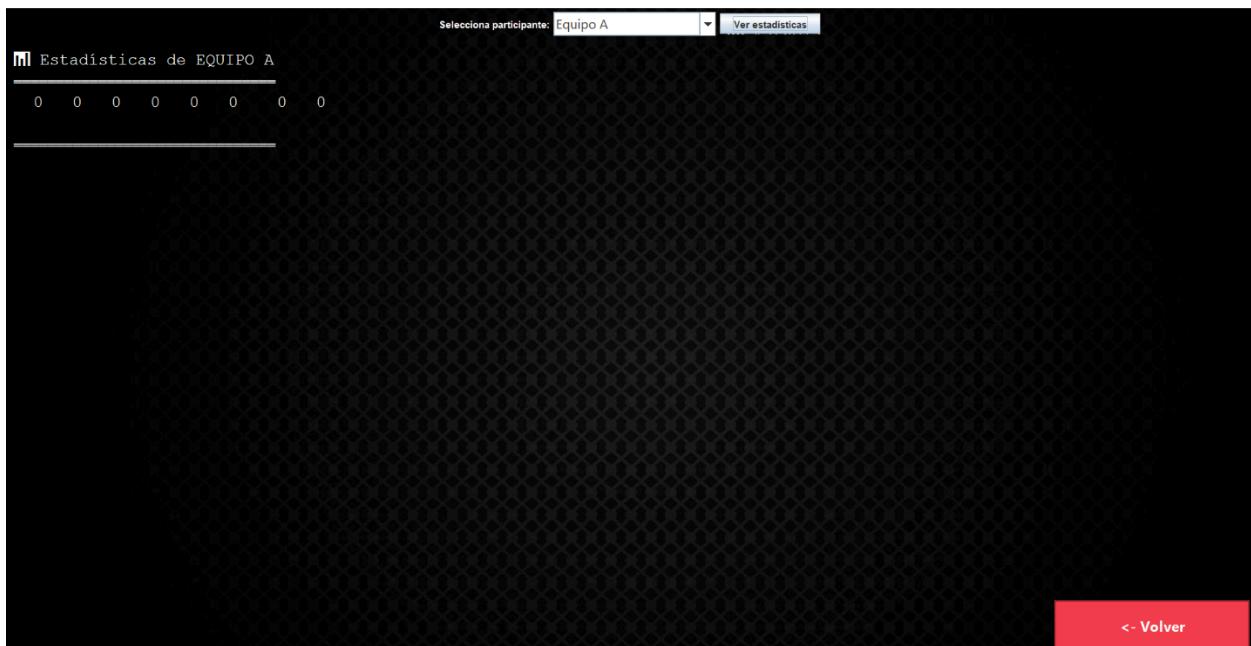
- Vista de estadísticas

Selecciona participante: Equipo A | Ver estadísticas

Estadísticas de EQUIPO A

Parámetro	Valor
Ganados	0
Empatados	0
Perdidos	0
Goles Anotados	0
Goles Recibidos	0
Títulos Ganados	0

<- Volver





- Vista estadísticas Liga

Tabla de Posiciones - Liga

Pos	Jugador	PJ	PG	PE	PP	PA	PC	DP	P
1	Dana	6	4	0	2	17	10	7	12
2	Charlie	6	3	1	2	17	18	-1	10
3	Alice	6	2	1	3	15	16	-1	7
4	Bob	6	2	0	4	13	18	-5	6

[-> Volver](#)

- Creación de torneo

Crear Nuevo Torneo

Nombre:

Fecha (dd-MM-yyyy HH:mm):

Disciplina:

Formato:

Tipo de Torneo: Individual Por equipos

Contraseña:

[-> Volver](#) [Confirmar](#)



- Inscribirse a torneo

LOL
ELIMINATORIA de LOL

Fecha: 10-06-2026 20:00
Participantes: 0
Estado: Activo (Inscripciones abiertas)

[Inscribirse](#) | [-> Volver](#)

Input X

? ¿Cuántos integrantes tiene tu equipo?

OK **Cancel**

Integrante 1 X

? Nombre del Jugador 1: Juan
Número de contacto del Jugador 1: 86242348

OK **Cancel**

Integrante 2 X

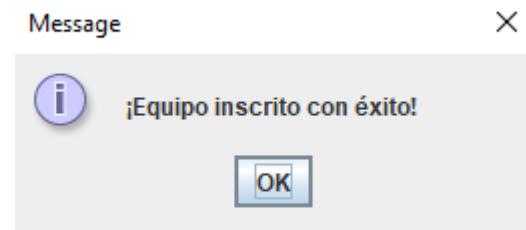
? Nombre del Jugador 2: Alberto
Número de contacto del Jugador 2: 47381295

OK **Cancel**

Datos del equipo X

? Nombre del equipo: Club Fc
Número de contacto del equipo: 83424689

OK **Cancel**



- Diagrama de clases UML:

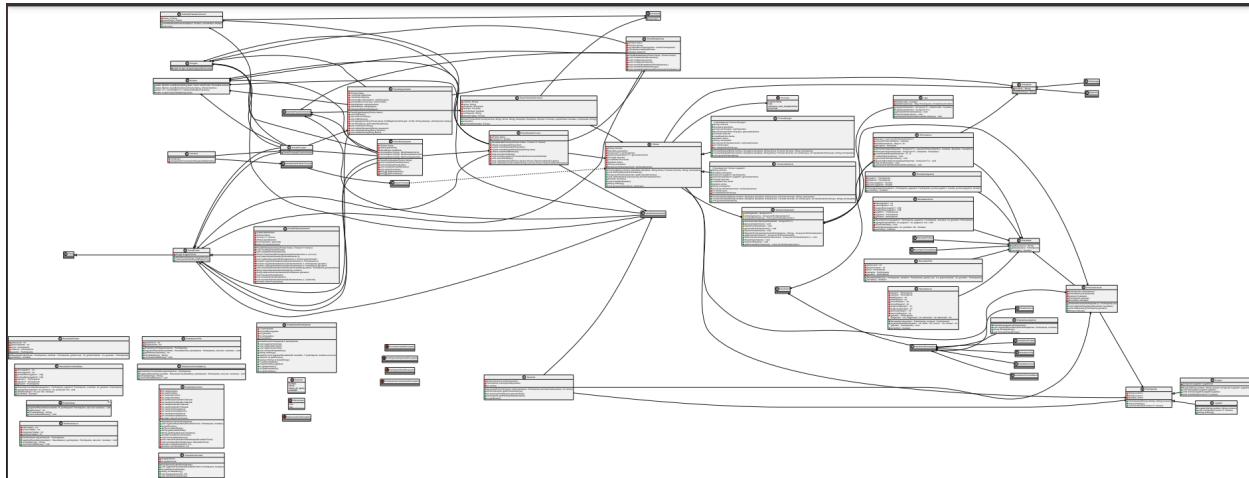
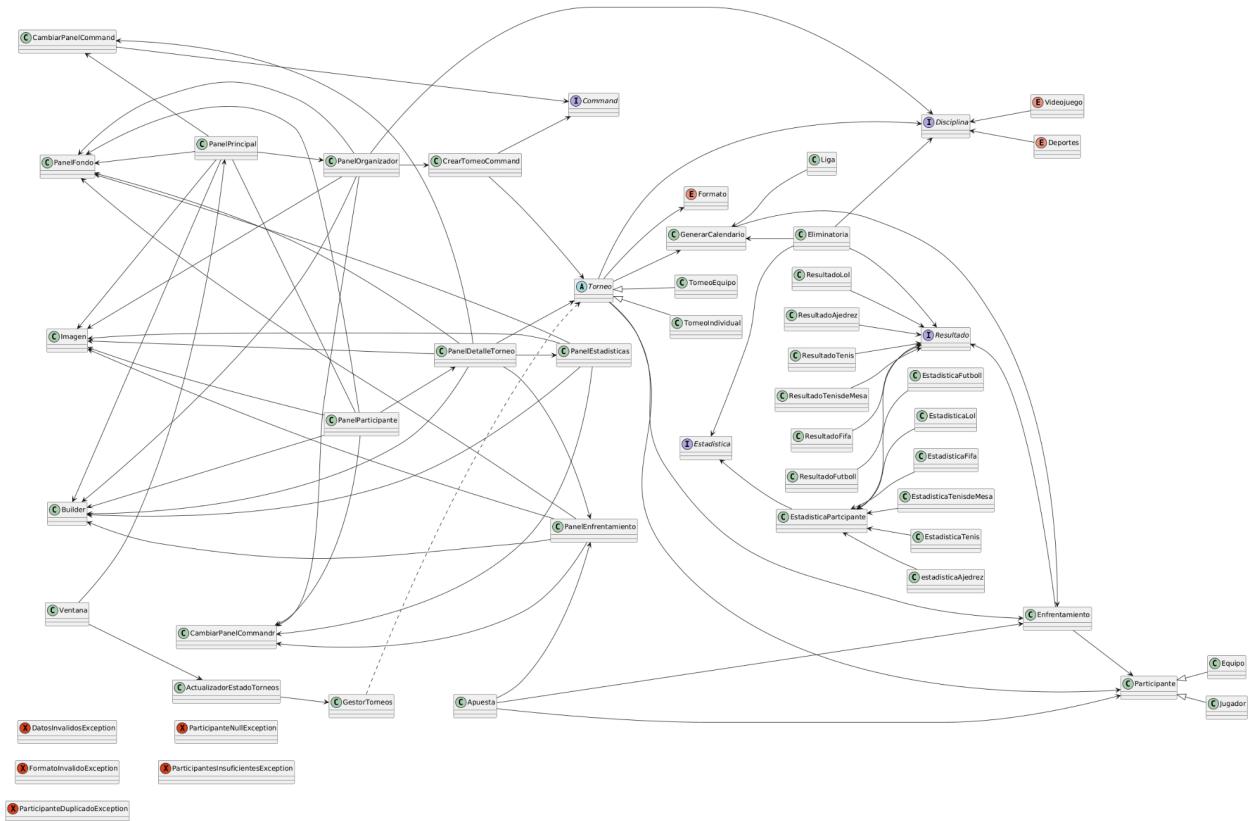




Diagrama de clases UML (sin propiedades ni métodos)



Se tuvo que hacer 2 versiones del uml ya que al añadir los métodos y propiedades desaparecen las conexiones de ciertas clases



Patrones de diseño

- Patrón Builder:

El patrón Builder se ha utilizado para encapsular la lógica de la construcción de componentes gráficos personalizados (principalmente Button y ComboBox) que requieren múltiples pasos de configuración visual y funcional.

Este patrón permite separar la creación compleja de objetos de su representación final, facilitando la reutilización, el mantenimiento y la coherencia visual de la interfaz de usuario.

Este patrón soluciona los siguientes problemas:

- Evita la duplicación de código al crear botones o JComboBox con estilos repetitivos
- Mejora la legibilidad del código al encapsular la configuración interna
- Facilita futuras modificaciones al comportamiento o apariencia de las componentes sin afectar otras partes del sistema

- Patrón Command:

Se utilizó el patrón Command para encapsular las acciones o solicitudes como objetos independientes, separando la invocación de la ejecución de una operación. Este patrón permite manejar acciones complejas de manera desacoplada.

Se definió la interfaz Command, que declara el método execute(), implementada por comandos concretos que representan acciones específicas, por ejemplo, CambiarPanelCommand y CrearTorneoCommand.

Se optó por este patrón debido a:

- Desacopla el código que solicita una acción del que la ejecuta, mejorando la modularidad.
- Facilita la extensión del sistema agregando nuevas acciones sin modificar el código existente.
- Mejora la legibilidad y mantenibilidad al encapsular la lógica de cada acción en clases específicas.



- Patrón Strategy:

Se usó el patrón Strategy con el objetivo de encapsular las lógicas de estadísticas y resultados en distintos tipos de deportes o videojuegos.

Cada disciplina presenta las reglas propias para determinar victorias, derrotas, empates, puntos, criterios de desempate, forma para representar resultados, por lo que buscamos una forma flexible y extensible para desacoplar las lógicas por deporte.

Se definió la interfaz Resultado, que representa el común para modelar el resultado de un enfrentamiento, esto permite que un enfrentamiento trabaje de forma polimórfica con resultado de distintos tipos de deporte.

Se definió la interfaz Estadística, la cual fue implementada por la clase abstracta EstadisticasParticipante y sus subclases, cada subclase encapsula como interpretar un Resultado y actualizar las métricas correspondientes.

Se optó por este patrón debido a los siguientes beneficios:

- Evitar condicionales muy extensos
- Permite extender fácilmente el sistema con nuevos tipos de resultados y estadísticas sin modificar las clases que existen
- Favorece el principio abierto/cerrado
- Respeta el principio de responsabilidad única al aislar los resultados y estadísticas del resto de la lógica del torneo
- Mejora legibilidad, mantenimiento y testeo del código

- Patrón Template Method:

En el diseño del sistema se aplicó el patrón Template Method en dos contextos clave:

1. Manejo de estadísticas participante:

Se utilizo el patrón Template Method para definir la estructura general del manejo de estadísticas de participantes en diferentes tipos de competencias deportivas o videojuegos, dejando los pasos específicos de la actualización de estadísticas y calculo de puntos a subclases concretas.



La clase abstracta EstadisticasParticipante implementa los métodos comunes, como registrar victorias, empates y derrotas, reiniciar estadísticas y mantener el conteo básico de partidos jugados, ganados, empatados y perdidos. Sin embargo, delega a las subclases la implementación de los métodos más específicos como registrarResultado, getPuntos, etc.

Se optó por este patrón porque:

- Permite definir un esqueleto común para el manejo de las estadísticas, asegurando consistencia y evitando duplicación de lógica común.
- Facilita la extensión del sistema para nuevos deportes o modalidades simplemente creando nuevas subclases que implementan los pasos específicos.
- Respeta el principio de responsabilidad única y favorece la reutilización del código.
- Mejora la legibilidad y el mantenimiento al separar claramente las partes fijas y las variables del algoritmo de estadísticas.

2. Generación de calendario de enfrentamientos:

La clase abstracta GenerarCalendario aplica también el patron template method para definir la escritura general del proceso de generación de un calendario de enfrentamientos en distintos formatos de torneo. Así la clase base fija el flujo general del algoritmo, mientras que las subclases concretas aportan las variaciones específicas para cada formato de calendario.

Este diseño permite:

- Garantizar que siempre se realice validación previa antes de generar los enfrentamientos
- Evitar duplicación de código.
- Facilita la extensión para nuevos formatos de torneo creando nuevas subclases que definen únicamente la generación de enfrentamientos particulares.
- Mejorar la claridad y mantenimiento del código al separar la estructura fija y los detalles variables del algoritmo.



Decisiones tomadas

En el desarrollo del sistema de gestión de torneos, decidimos utilizar patrones de diseño como Template Method para definir tanto de la generación de calendarios como en el manejo de estadísticas en EstadisticasParticipante, facilitando la extensión del código para nuevas clases formato. También se decidió aplicar el patrón Builder para la construcción de componentes gráficos en la interfaz (como botones y ComboBox), dado que los códigos se pueden duplicar y haciendo una clase simple que contenga lo necesario se puede reducir bastante la complejidad del código. Se empleó el patrón Strategy para encapsular la lógica de resultados y estadísticas a través de interfaces, lo cual permite que el sistema sea fácilmente extensible, en lugar de modificar las clases existentes, basta con crear nuevas clases que implementen las interfaces Resultado o Estadística.

Por otro lado, se decidió mantener separada la lógica de generación de datos y la presentación en la interfaz gráfica, con el fin de mejorar la modularidad, facilitar las pruebas, y permitir futuras ampliaciones sin afectar otras partes del sistema.

Problemas en el desarrollo

Uno de los problemas que tuvimos fue la extensibilidad del código, ya que intentamos complejizar la generación de torneos, creando torneos de grupos con eliminatoria que finalmente por temas de tiempo tuvimos que dejar, otro de los problemas que tuvimos fue acordar patrones de diseño para implementar, la idea principal era dejar un código potencialmente escalable para agregar más deportes, disciplinas y formatos, lo cual lo conseguimos, aunque de todas formas quedo un código extenso.

También otro problema encontrado fue la integración de las estadísticas en la GUI, dado que se solapaban al presionar el botón, cosa que claramente es un problema en el código.

Encontramos también problemas con la clase Liga, ya que esta acepta parámetros genéricos para obtener polimorfismo y poder hacer ligas que acepten jugadores y equipos, así como distintos tipos de resultados sin la necesidad de crear variantes de liga o métodos extensos para cada caso, lo cual fue un problema ya que nos costó lograr hacer que funcionara con todo.



Autocritica

Aunque nuestro proyecto logró cumplir las características principales que queríamos implementar, hay aspectos que pudimos abordar con mayor/mejor planificación y enfoque técnico desde el inicio.

- **Falta de una arquitectura clara desde el inicio:** en las primeras etapas del código no teníamos mucha idea de cómo empezar a hacerlo, lo que generó un problema que deberíamos solucionar más tarde, que es refactorizarlo y mantenerlo, si hubiésemos diseñado mejor el código desde el principio podríamos haber ahorrado tiempo e incluso integrar más deportes, etc.
- **Tardamos en crear testing:** al principio las pruebas que hacíamos eran solo por mains múltiples y buscábamos errores solo depurando por consola, lo que claramente nos retrasó bastante, ya que nos costaba y demorábamos mucho en encontrar errores de lógica que estaban medio escondidos.
- **Demasiada complejidad de algunas clases:** hay clases que manejan demasiada lógica, lo que hace que el código parezca “poco orientado a objetos”, ya que es un código demasiado extenso que maneja muchos casos, también esto nos dejó con el hecho de dejar inutilizada una clase que genera grupos con eliminatoria, ya que esa clase manejaba mucha complejidad de casos, con casos concretos, por lo que también sumado a la falta de tiempo tuvimos que dejar de lado.



Propuestas de mejora para futuros proyectos

Lo principal para mejorar es tratar de hacer un diagrama antes de empezar a hacer código, algo simple para tener una idea de como empezar a hacer el proyecto, como un diagrama de casos de uso más complejo para crear la base del proyecto, hacer esa base funcional y luego agregar todos los detalles.

Una mejora para este proyecto seria hacer una mantención al código, eliminar casteos innecesarios, tratar de hacer el código lo más extensible que se pueda, conseguir la separación de la GUI con la lógica, también añadir GruposEliminatoria con todo lo que teníamos pensado añadir, también podríamos reforzar la GUI, crear código mas robusto y evitar la duplicación de este. también crear más test para cubrir todos los casos bordes y casos más generales.