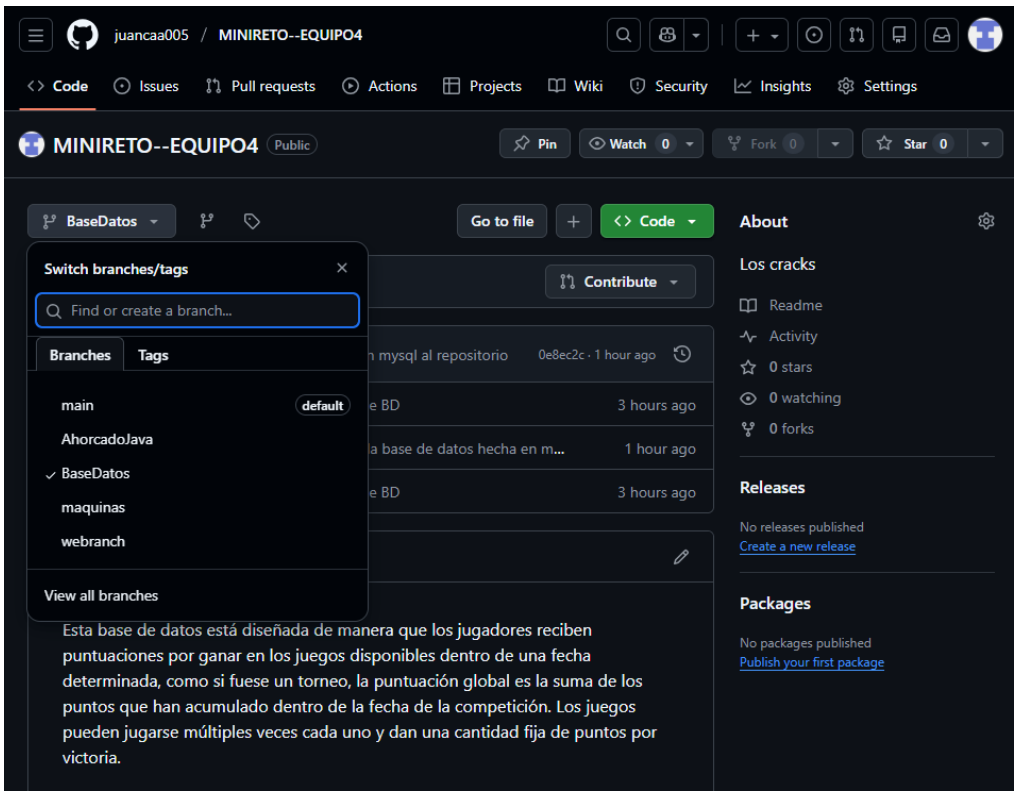
SISTEMAS INFORMATICOS

Para el apartado de sistemas informáticos hemos realizado un diario con una guía de instalación de las máquinas virtuales, así como la explicación de como ejecutarlo, todo esto lo tenemos en la rama main del repositorio de GitHub.

Enlace a nuestro repositorio

ENTORNOS DE DESARROLLO

Refiriéndose al módulo de entornos de desarrollo hemos utilizado GitHub para compartir el contenido entre nosotros. A su vez hemos usado GitHub desktop y gitBash para poder subir esos contenidos y poder también descargarlos desde la nube a local. Hemos creado varias ramas/branches para poder organizar mejor el trabajo y luego realizado un merge para finalmente tener todo el contenido unificado. Hemos editado el README para incluir información clave que se deba saber.



PROGRAMACION

En programación hemos usado todas las dotes y conocimientos que el profesor nos ha brindado y hemos creado los juegos hundir la flota y el ahorcado. Hemos usado estructuras de control y arrays para crearlos.

Primeramente para el juego hundir la flota:

No es solo disparar, sino que controla bien los turnos, los barcos, los impactos y el final de la partida.

Lo primero que se hace es definir las reglas base del juego. Por ejemplo, el tamaño del tablero se fija con esta línea:

```
private static final int TAMANIO_TABLERO = 10;
```

Gracias a esto, todo el juego sabe que el tablero es de 10x10 y si algún día se quisiera cambiar, solo habría que modificar ese número.

Después se definen los **estados de cada casilla del tablero**, usando constantes para que el código sea fácil de leer. Por ejemplo:

```
private static final int AGUA_OCULTA = 0;

private static final int BARCO_OCULTO = 1;

private static final int BARCO_TOCADO = 3;
```

Esto sirve para saber en todo momento si una casilla tiene agua, un barco sin tocar, un barco tocado o uno ya hundido, sin usar números raros que no significan nada.

También se definen los **tipos de barcos**, su tamaño y cuántos hay de cada uno. Esto se ve claro en líneas como:

```
private static final int[] TAMANIO_BARCOS = {5, 4, 3, 1};

private static final int[] CANTIDAD_BARCOS = {1, 2, 3, 4};
```

Esto hace que el juego sea muy flexible, porque si mañana quisieras cambiar la flota, no tendrías que tocar la lógica del juego, solo estos arrays.

El programa usa dos tableros principales, uno para el jugador y otro para la máquina, creados así:

```
private int[][] tableroJugador = new int[TAMANIO_TABLERO][TAMANIO_TABLERO];

private int[][] tableroMaquina = new int[TAMANIO_TABLERO][TAMANIO_TABLERO];
```

En estos tableros se guarda la información real de dónde hay barcos y agua.

Además, hay otros dos tableros que sirven solo para controlar los disparos:

```
private boolean[][] disparosJugador = new
boolean[TAMANIO_TABLERO][TAMANIO_TABLERO];
```

Esto permite saber si ya se ha disparado a una casilla y evita que el jugador o la máquina repitan disparos en el mismo sitio.

Una de las partes más importantes del programa es cómo se guardan los barcos. En lugar de guardar solo "hay barco aquí", se guardan todas las coordenadas de cada barco usando esta estructura:

```
private int[][][] barcosJugador;
```

Así, cada barco sabe exactamente qué casillas ocupa. Esto es clave para poder comprobar si un barco está completamente hundido sin tener que recorrer todo el tablero.

Cuando empieza el juego, el constructor llama a este método:

```
inicializarJuego();
```

Aquí se dejan los tableros llenos de agua, se reinician los disparos y se preparan los arrays de barcos, asegurando que cada partida empieza limpia.

Luego viene la **colocación de barcos del jugador**, que es manual. El programa va pidiendo fila, columna y orientación, mostrando el tablero mientras colocas. Esto se controla con el método:

```
private void colocarBarcosJugador()
```

Además, solo pregunta la orientación cuando el barco tiene más de una casilla, lo cual es un detalle muy bien pensado para evitar preguntas innecesarias.

Antes de colocar un barco, siempre se comprueba que la posición sea válida usando:

```
if (esColocacionValida(tableroJugador, fila, columna, tamaño, horizontal))
```

Aquí se verifica que el barco no se salga del tablero y que no se solape con otro, lo que evita errores y trampas.

La máquina coloca sus barcos de forma automática usando números aleatorios, con líneas como:

```
int fila = random.nextInt(TAMANIO_TABLERO);
```

Esto hace que cada partida sea distinta y que el jugador no pueda memorizar posiciones.

Durante el juego, los turnos se gestionan con una variable muy simple pero efectiva:

```
boolean turnoJugador = true;
```

Si el jugador acierta, vuelve a disparar. Si falla, pasa el turno. Esto se controla devolviendo true o false desde métodos como:

```
private boolean realizarDisparoJugador()
```

Cuando se dispara, el programa comprueba si hay barco con esta condición:

`if (tableroMaquina[fila][columna] == BARCO_OCULTO)`

Si hay barco, lo marca como tocado y además llama a un método que revisa si el barco entero ha sido hundido.

Ese método es una de las partes más potentes del programa:

`verificarYMarcarHundido(...)`

Aquí se recorren las coordenadas del barco y se comprueba si todas están tocadas. Si lo están, se cambian todas a estado hundido y se marca el barco como destruido.

El final del juego se controla revisando si todos los barcos de un jugador están hundidos, usando contadores y arrays como:

`if (hundidosJugador[i] == 0)`

Cuando uno de los dos pierde todos sus barcos, el juego se detiene y se muestra el mensaje de victoria o derrota.

Por último, el método que muestra los tableros imprime ambos **lado a lado**, ocultando correctamente la información del enemigo y mostrando solo lo que el jugador ha descubierto. Esto se ve claramente en el método:

`mostrarTablerosLadoALado();`

Gracias a esto, el juego es fácil de seguir visualmente y no revela información que no debería verse.

Por ultimo tenemos el metodo main que ejecuta todo y lo lanza para jugar en la consola

```java

package com.mycompany.mavenproject1;

import java.util.Random;
import java.util.Scanner;

public class HundirLaFlota2Main {


    public static void main(String[] args) {
        // Crear una instancia del juego
        HundirFlota2 juego = new HundirFlota2();

        // Iniciar el juego
        juego.iniciarJuego();
    }
}
```

POR OTRO LADO EL AHORCADO:

Esta clase forma parte del juego del **Ahorcado** y su función principal es **ayudar al programa principal** con tareas concretas, como pedir confirmaciones al usuario, separar palabras en letras y mostrar la palabra oculta por pantalla. No es la clase que controla todo el juego, sino una clase de apoyo, y eso ya es una buena decisión de diseño.

MINIRETO EQUIPO 4

La clase se llama `AhorcadoClase`, lo que deja claro que aquí no está el `main`, sino métodos que se reutilizan. Esto se ve directamente en la declaración:

```
public class AhorcadoClase {
```

Además, todos los métodos son `static`, lo que permite usarlos sin necesidad de crear un objeto de esta clase, algo muy cómodo para este tipo de utilidades.

El primer método sirve para **preguntar al usuario si quiere empezar o repetir una partida**. Esto se ve en la cabecera:

```
public static char pregunta(String men, Scanner teclado)
```

El método recibe un mensaje y el teclado, lo que permite reutilizarlo para distintas preguntas. Internamente, muestra el mensaje por pantalla con esta línea:

```
System.out.println(men + " (s/n)");
```

De esta forma, el usuario sabe que solo puede responder con s o n.

Luego se recoge la respuesta del usuario y se transforma a minúscula para evitar errores si escribe una S mayúscula:

```
resp = teclado.next().toLowerCase().charAt(0);
```

Esto es un detalle importante, porque hace el programa más tolerante con la entrada del usuario.

Después, el método entra en un bucle que **obliga al usuario a escribir una respuesta válida**. Mientras no escriba s o n, el programa sigue insistiendo:

```
while (resp != 's' && resp != 'n') {
```

Si se equivoca, se muestra un mensaje de error claro y vuelve a pedir la respuesta. Cuando por fin es correcta, el método devuelve el carácter, lo que permite al programa principal decidir qué hacer.

El segundo método se encarga de **separar una palabra en letras**, algo fundamental en el ahorcado. Este método empieza aquí:

```
public static char[] separa(String palAzar)
```

Recibe la palabra que hay que adivinar y devuelve un array de caracteres, es decir, letra por letra.

Primero se crea el array con el mismo tamaño que la palabra:

```
letras = new char[palAzar.length()];
```

Después, con un bucle, se va copiando cada letra del string al array:

```
letras[i] = palAzar.charAt(i);
```

Esto permite que el juego pueda comparar fácilmente las letras que va diciendo el jugador con las letras reales de la palabra.

Este método es clave porque transforma algo complejo (un string) en algo más manejable para el juego, que es un array de caracteres.

El último método se encarga de **mostrar por pantalla la palabra oculta o parcialmente descubierta**. Su cabecera es:

```
public static void imprimeOculta(char[] tusRespuestas)
```

Recibe un array de caracteres que normalmente contiene guiones bajos o letras acertadas.

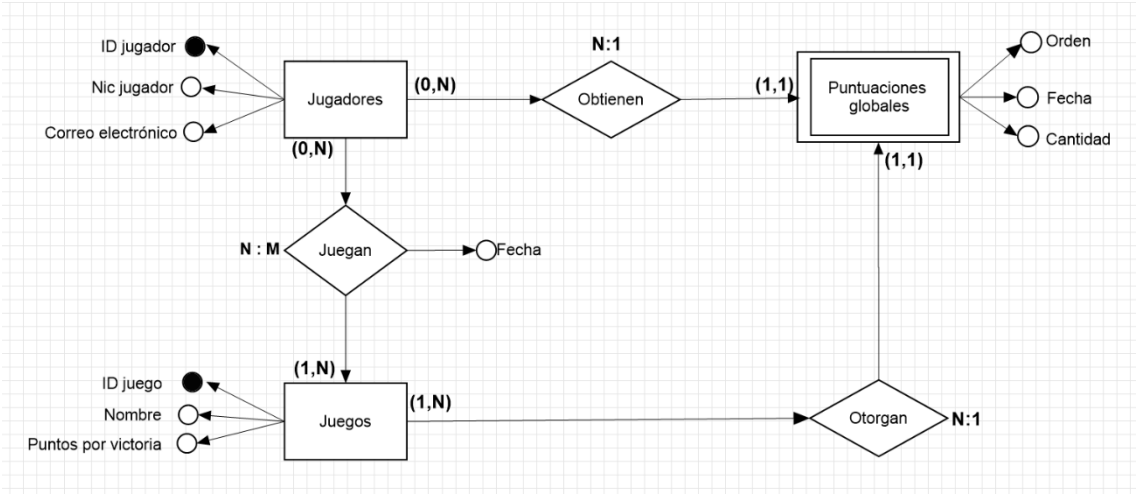
Dentro del método se recorre el array y se imprime cada carácter con un espacio al lado:

```
System.out.print(tusRespuestas[i] + " ");
```

Gracias a esto, la palabra se ve separada por espacios, lo que hace que el jugador pueda distinguir mejor cuántas letras tiene y cuáles ha acertado.

BASES DE DATOS

En bases de datos hemos hecho lo pedido por el profesorado, primero hemos hecho el diagrama E/R:



Este diagrama representa un sistema de **jugadores que juegan a juegos y van consiguiendo puntos**. La idea general es bastante sencilla y se entiende rápido.

Por un lado está la entidad **Jugadores**, que guarda la información básica de cada jugador: un **ID** para identificarlo, el **nick** y el **correo electrónico**. Es lo típico y tiene sentido porque cada jugador es único dentro del sistema.

Luego está la entidad **Juegos**, donde se guardan los distintos juegos disponibles. Cada juego tiene su **ID**, su **nombre** y los **puntos que da por ganar**. Esto sirve para que no todos los juegos puntúen igual y se pueda ajustar la dificultad o la importancia de cada uno.

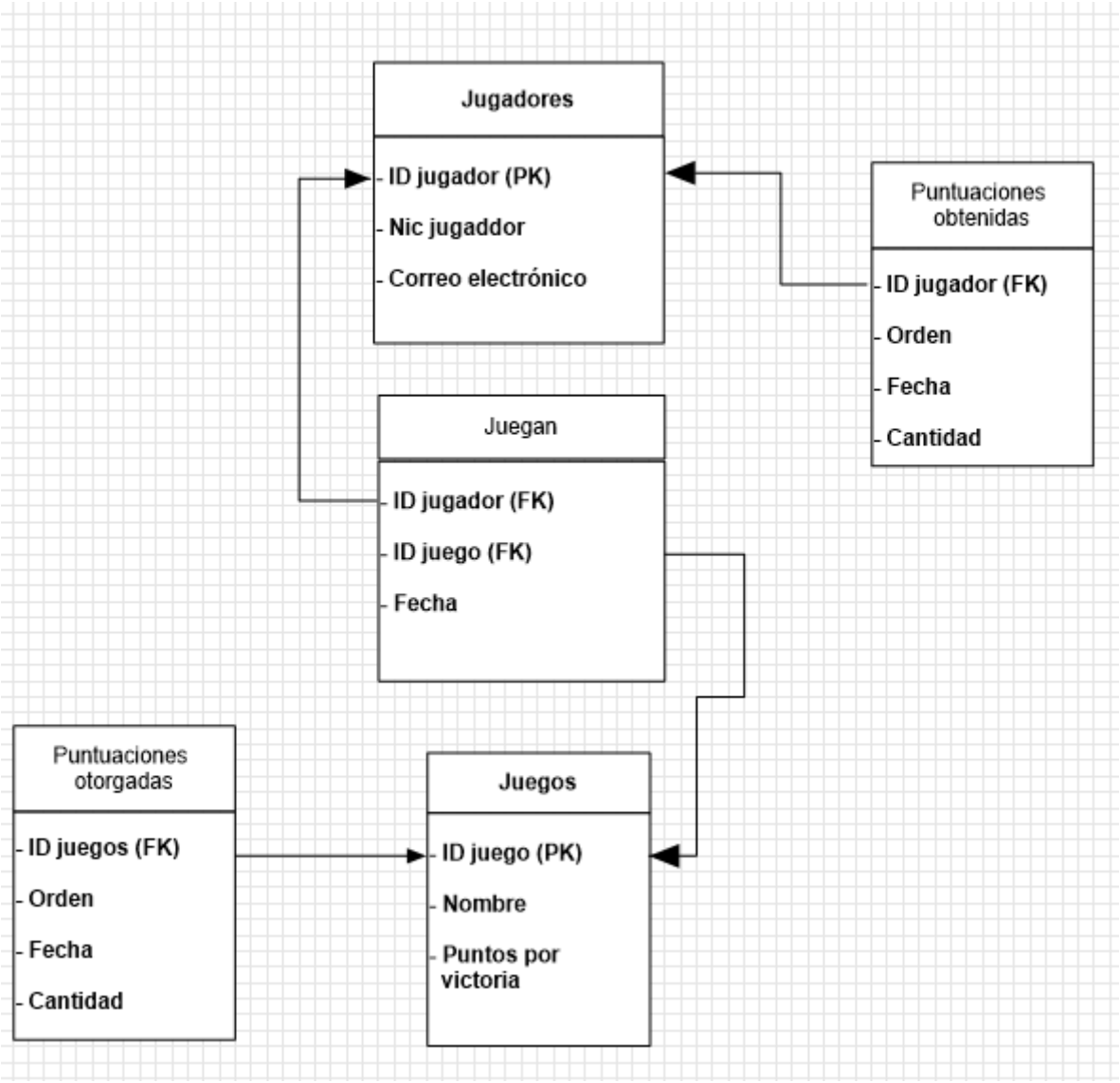
Entre jugadores y juegos está la relación **Juegan**, que es **muchos a muchos**. Esto significa que un jugador puede jugar a varios juegos y un juego puede ser jugado por muchos jugadores distintos. Además, esta relación tiene la **fecha**, que viene muy bien para saber cuándo se jugó cada partida.

Después aparece la entidad **Puntuaciones globales**, que representa el resultado final o el ranking. Aquí se guarda el **orden**, la **fecha** y la **cantidad de puntos**. Es como el resumen de lo que ha conseguido el jugador.

La relación **Obtienen** une a los jugadores con las puntuaciones globales y viene a decir que **un jugador puede tener varias puntuaciones a lo largo del tiempo**, pero cada puntuación pertenece solo a un jugador.

Por último, la relación **Otorgan** conecta los juegos con las puntuaciones globales. La idea es que los **juegos son los que dan los puntos**, y esos puntos acaban reflejados en la puntuación global del jugador.

LUEGO HEMOS HECHO EL DIAGRAMA RELACIONAL:



Este diagrama es básicamente el **paso del modelo entidad–relación a tablas de base de datos**, o sea, cómo quedaría todo montado en SQL.

La tabla **Jugadores** guarda la info básica de cada jugador. Tiene el **ID jugador** como clave primaria, que identifica a cada uno, y luego el **nick** y el **correo electrónico**. Es la tabla principal de los jugadores y de aquí cuelga casi todo lo demás.

Luego está la tabla **Juegos**, que hace lo mismo pero con los juegos. Tiene su **ID juego** como clave primaria, el **nombre del juego** y los **puntos por victoria**. Esto sirve para saber cuánto puntúa cada juego cuando se gana.

La relación de muchos a muchos entre jugadores y juegos se resuelve con la tabla **Juegan**. Esta tabla es clave porque une a un jugador con un juego concreto. Tiene como campos el **ID jugador** y el **ID**

juego, que son claves foráneas, y además guarda la **fecha** en la que se jugó. Así se sabe quién jugó a qué y cuándo, sin duplicar información.

Después aparece la tabla **Puntuaciones obtenidas**, que está relacionada con los jugadores. Aquí se guardan las puntuaciones que va consiguiendo cada jugador a lo largo del tiempo. El **ID jugador** actúa como clave foránea y permite saber a quién pertenece esa puntuación. El **orden**, la **fecha** y la **cantidad** sirven para montar rankings o historiales de puntos.

Por otro lado está la tabla **Puntuaciones otorgadas**, que está relacionada con los juegos. Esta tabla guarda los puntos que un juego otorga en un momento concreto. El **ID juego** es clave foránea y conecta directamente con la tabla Juegos. El resto de campos sirven para saber cuándo se dieron esos puntos y cuántos fueron.

POR ULTIMO HICIMOS LA BASE DE DATOS REAL PARA MYSQL:

```
DROP TABLE IF EXISTS `juegan`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `juegan` (
  `IDpartida` int NOT NULL,
  `IDjugador` int NOT NULL,
  `IDjuego` int NOT NULL,
  `Fecha` date NOT NULL,
  PRIMARY KEY (`IDpartida`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `juegan`
--

LOCK TABLES `juegan` WRITE;
/*!40000 ALTER TABLE `juegan` DISABLE KEYS */;
/*!40000 ALTER TABLE `juegan` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `juegos`
--

DROP TABLE IF EXISTS `juegos`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `juegos` (
  `idjuego` int NOT NULL AUTO_INCREMENT,
  `nombre` varchar(50) NOT NULL,
  `puntos_por_victoria` int NOT NULL,
  PRIMARY KEY (`idjuego`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `juegos`
--

LOCK TABLES `juegos` WRITE;
/*!40000 ALTER TABLE `juegos` DISABLE KEYS */;
/*!40000 ALTER TABLE `juegos` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `jugadores`
--

DROP TABLE IF EXISTS `jugadores`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `jugadores` (
  `idJugador` int NOT NULL,
  `Nic_jugador` varchar(30) NOT NULL,
  `correo_electronico` varchar(100) NOT NULL,
  PRIMARY KEY (`idJugador`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `jugadores`
--

LOCK TABLES `jugadores` WRITE;
/*!40000 ALTER TABLE `jugadores` DISABLE KEYS */;
/*!40000 ALTER TABLE `jugadores` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `puntuaciones obtenidas`
--

DROP TABLE IF EXISTS `puntuaciones obtenidas`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `puntuaciones obtenidas` (
  `Idjugador` int NOT NULL,
  `orden` int NOT NULL,
  `Fecha` date NOT NULL,
  `Cantidad` int NOT NULL,
  PRIMARY KEY (`Idjugador`)
```

Empiezo por la tabla **jugadores**, que es una de las principales. Aquí se define con esta parte:

`CREATE TABLE jugadores (`

Tiene el `idJugador` como **clave primaria**, lo cual está bien porque identifica a cada jugador de forma única. Luego están el `Nic_jugador` y el `correo_electronico`, que son los datos básicos del usuario. Es una tabla limpia, sin cosas raras, justo lo que se espera.

Después está la tabla **juegos**, creada con:

`CREATE TABLE juegos (`

Aquí el `idjuego` es clave primaria y además es `AUTO_INCREMENT`, lo cual es buena idea porque los juegos se pueden ir añadiendo solos sin preocuparse del ID. También guarda el nombre del juego y los puntos por victoria, que encaja perfecto con el modelo que habías planteado.

La tabla **juegan** representa claramente la relación entre jugadores y juegos. Se nota porque tiene estos campos:
`IDjugador`, `IDjuego` y `Fecha`
Esto sirve para saber **qué jugador jugó a qué juego y cuándo**. Además, has añadido `IDpartida` como clave primaria, lo cual está bastante bien porque identifica cada partida de forma única y evita problemas si un mismo jugador juega muchas veces al mismo juego.

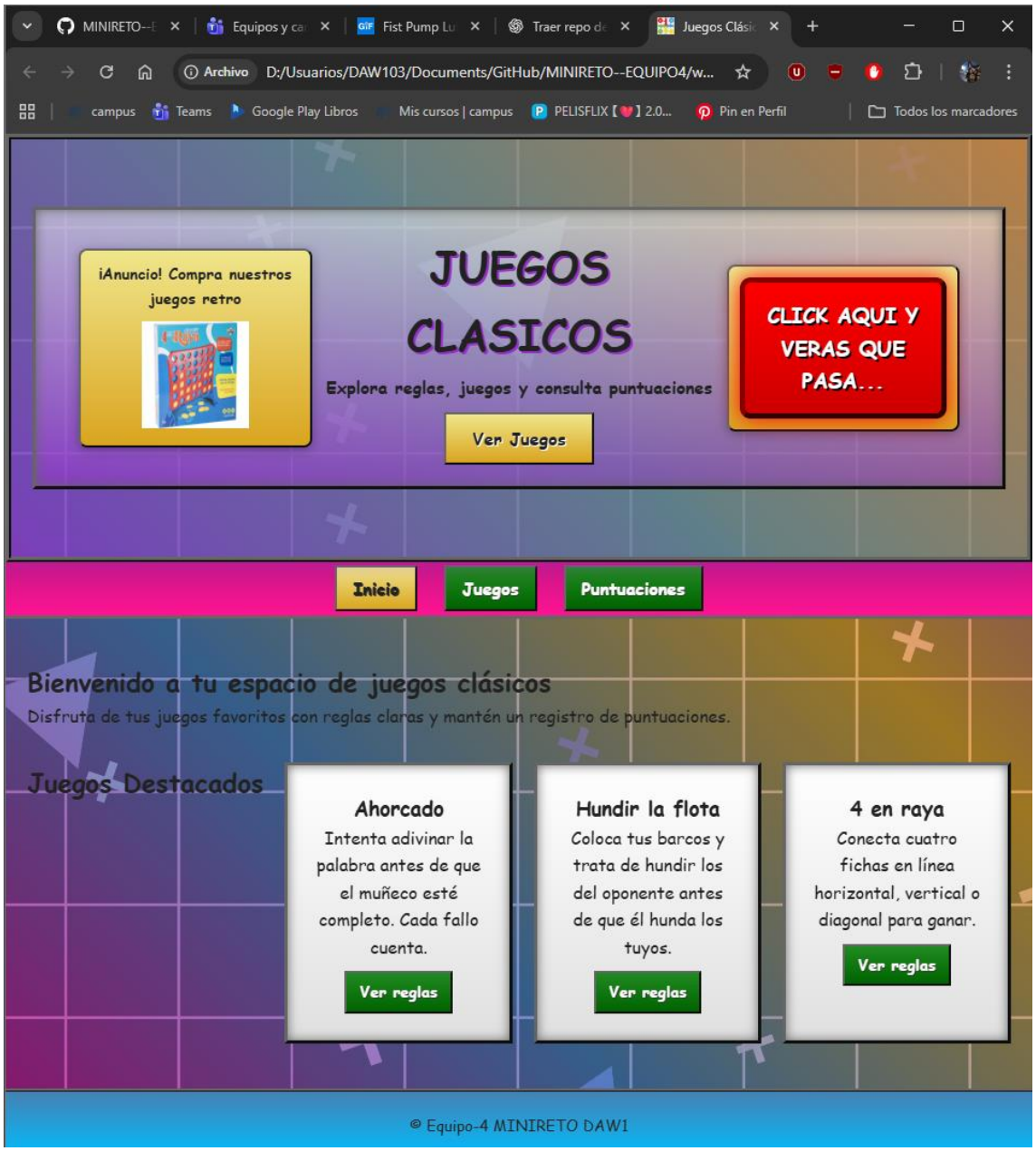
Luego está la tabla **puntuaciones obtenidas**, que guarda las puntuaciones que consigue cada jugador. Aquí aparece el campo:
`Idjugador`
que identifica a qué jugador pertenece esa puntuación, junto con el `orden`, la `fecha` y la `cantidad` de puntos. La idea se entiende bien: es como el historial o ranking del jugador.

Y por último está **puntuaciones otorgadas**, que funciona de forma parecida pero asociada a los juegos. Se nota porque usa:
`Idjuegos`
para indicar qué juego es el que otorga esos puntos. El resto de campos (orden, fecha y cantidad) sirven para guardar cuándo y cuántos puntos se dieron.

LENGUAJE DE MARCAS

Refiriendose al modulo de lenguaje de marcas hemos diseñado una web en la que creemos que se han aplicado casi todos los conocimientos usados en clase e incluso añadidos un par extras.

Primeramente hemos optado por crear la pagina index:



¡Empecé el código con <!DOCTYPE html> y <html lang="es">, que es básicamente decirle al navegador “oye, esto es HTML5 y está en español”. Es algo que siempre hay que poner para que todo funcione bien.

En el <head> puse cosas importantes: el título de la página, que se vea correctamente en la pestaña del navegador, y el enlace al **CSS** que hace que todo se vea bonito. También añadí un icono para la pestaña (favicon) y algunas cosas que ayudan a que la web se vea bien en móviles.

Después añadí un **vídeo de fondo**, que se reproduce solo, sin sonido y en bucle. Esto lo puse para que la página se vea más moderna y llamativa. Para que el contenido se lea encima del vídeo, añadí una capa con .overlay, que es como una especie de filtro semitransparente.

En la **cabecera** puse varias cosas: a la izquierda un pequeño anuncio con imagen, en el centro el título principal y un botón para ir a la sección de juegos, y a la derecha otro botón más grande que hace una broma, que abre otra página (sí, un Rickroll). Todo esto lo hice usando para separar las zonas y luego poder darles estilo con CSS.

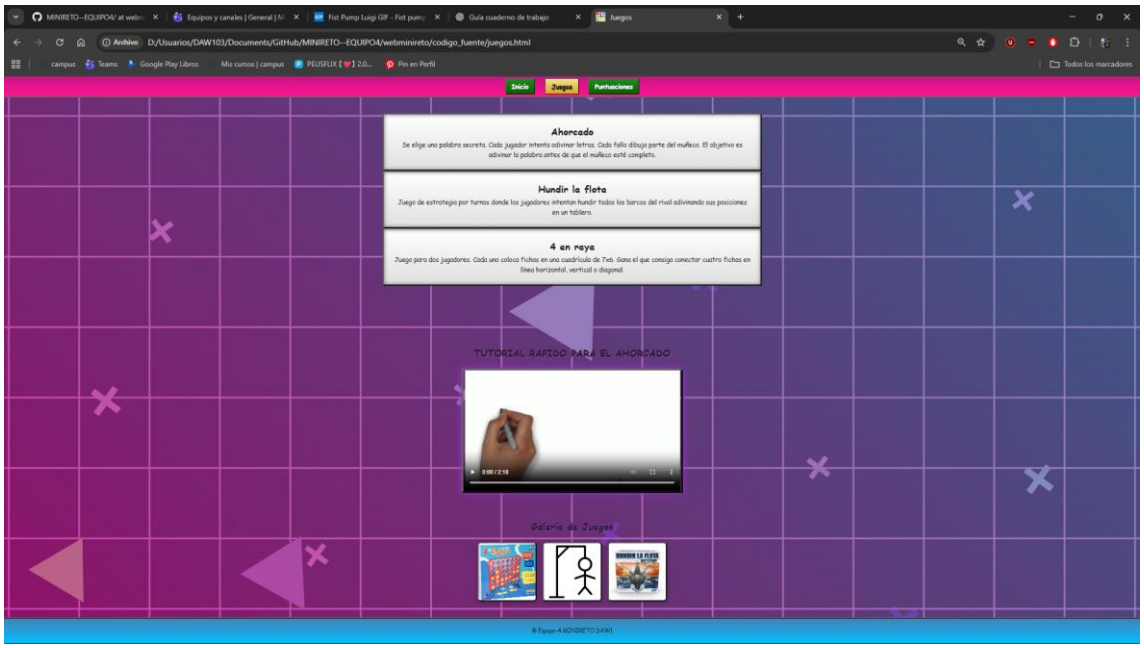
La **navegación** es un menú simple con tres enlaces: Inicio, Juegos y Puntuaciones. Esto permite moverse por la web sin perderse.

En el **main**, hice dos secciones: una de introducción con un pequeño mensaje de bienvenida y otra con los **juegos destacados**. Cada juego tiene su nombre, una pequeña descripción y un botón para ver las reglas. Esto lo hice con tarjetas (con clase card-juego) para que sea más visual y fácil de entender.

Finalmente, añadí un **footer** con un mensaje de copyright, y un **banner de cookies** que aparece para que el usuario acepte el uso de cookies. También conecté un archivo script.js para que el banner funcione al darle a “Aceptar”.

En resumen, todo el código está pensado para que la página sea clara, fácil de navegar y atractiva visualmente. Usé etiquetas semánticas como <header>, <nav>, <main> y <footer> para que tenga sentido, y clases en los para aplicar estilos con CSS.

PAGINA JUEGOS



Empecé el código igual que en la página principal, diciendo que era HTML5 y que estaba en español para que el navegador lo interprete bien. En el head puse el título de la página como “Juegos”, añadí un favicon para que se vea un iconito en la pestaña y conecté el CSS para que todo se vea igual que en el resto del proyecto. También añadí la metaetiqueta para que la web se vea bien en móviles, así no se descuadra ni en tablets ni en teléfonos.

Manteniendo la estética de toda la web, puse un vídeo de fondo que se reproduce solo, sin sonido y en bucle. Esto hace que la página sea más moderna y atractiva. Para que el contenido se lea encima del vídeo, añadí una especie de filtro semitransparente que mejora la visibilidad del texto y los elementos sobre el fondo en movimiento.

MINIRETO EQUIPO 4

La navegación la hice igual que en la página principal, con tres enlaces: Inicio, Juegos y Puntuaciones. Marqué el enlace de Juegos como activo para que el usuario sepa en qué página está. Esto hace que moverse por la web sea muy fácil y claro.

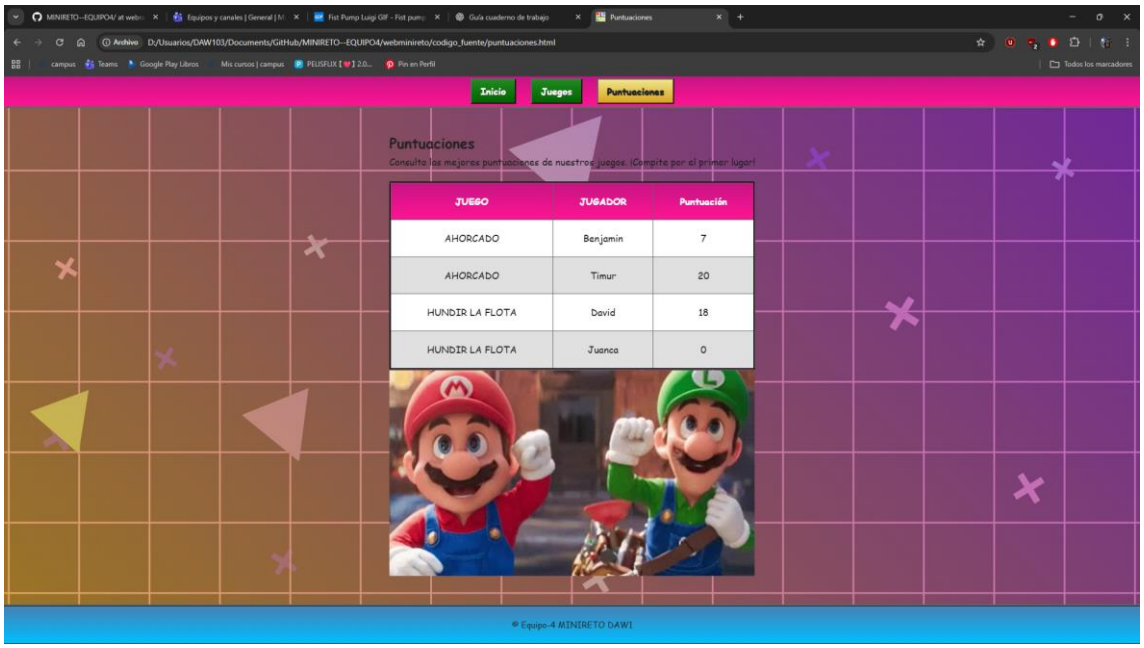
En el main puse las secciones de los tres juegos: Ahorcado, Hundir la Flota y 4 en Raya. Cada sección tiene su título y un párrafo explicando cómo se juega de forma sencilla, para que cualquiera lo entienda. También puse un identificador único para cada sección, así se pueden enlazar directamente desde otras partes de la web, como los botones de "Ver reglas" de la página principal.

Luego añadí una sección de vídeo para el Ahorcado, con controles para reproducirlo, pausarlo o ajustar el volumen. Esto sirve como tutorial rápido para los usuarios que prefieren aprender viendo cómo se juega en vez de leerlo.

Después añadí una galería de imágenes con fotos de los tres juegos. Las organicé en una cuadrícula para que se vea ordenado y no ocupe demasiado espacio. Esto ayuda a que el usuario identifique visualmente los juegos y hace la página más atractiva.

Por último, puse un footer con un mensaje de copyright igual que en la página principal y así todo queda consistente.

PAGINA PUNTUACIONES



Empecé la página igual que las anteriores, diciendo que era HTML5 y que estaba en español, y en el head puse el título "Puntuaciones", añadí el favicon para que se vea en la pestaña y conecté el CSS externo para que los estilos sean los mismos que en el resto de la web. También incluí la metaetiqueta para que se vea bien en móviles y tablets, así todo se mantiene consistente y no se descuadra en ningún dispositivo.

Mantuve el vídeo de fondo igual que en las otras páginas, que se reproduce solo, sin sonido y en bucle, para que todo el proyecto tenga la misma identidad visual y se vea moderno. También añadí el mensaje que aparece si el navegador no soporta vídeo, para que no quede un hueco vacío.

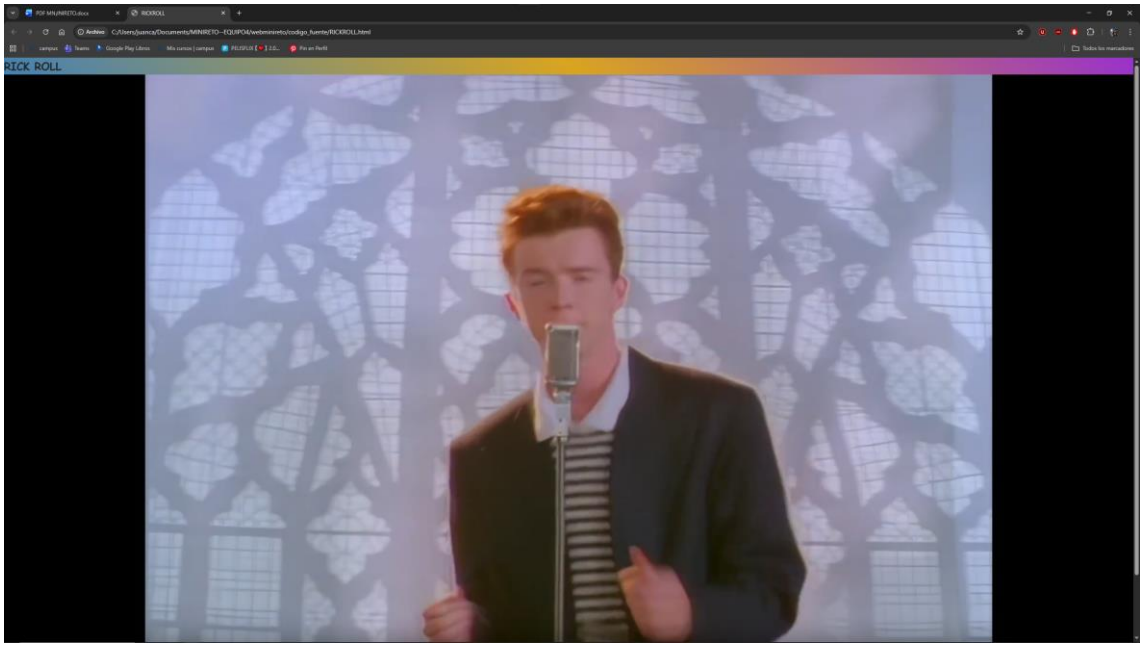
La navegación sigue la misma estructura, con tres enlaces: Inicio, Juegos y Puntuaciones. Marqué Puntuaciones como activo para que el usuario sepa en qué página está y pueda moverse fácilmente sin perderse.

En el main puse un título que indica que es la sección de puntuaciones y un pequeño párrafo explicando que ahí se pueden consultar las mejores puntuaciones y competir por el primer lugar. Debajo añadí una tabla con las puntuaciones de los jugadores en los distintos juegos. La tabla tiene encabezados para el nombre del juego, el jugador y la puntuación, y varias filas con datos reales de ejemplo, incluyendo mi propia puntuación en Hundir la Flota. Esto permite que el usuario vea claramente quién va primero y cómo se compara con otros.

Para hacer la página más divertida y dinámica añadí un GIF de Luigi celebrando, que se carga desde Tenor. Esto le da un toque más visual y entretenido a la página, haciendo que no sea solo texto y tablas.

Finalmente, puse un footer igual que en las otras páginas, con un mensaje de copyright para que todo quede consistente y profesional.

PAGINA RICKROLL



Esta página la hice como una broma dentro del proyecto, para que cuando alguien haga clic en el botón de la página principal, se abra este “Rickroll”. Empecé igual que siempre, definiendo que era HTML5 y que el idioma era español. En el head puse el título “RICKROLL”, añadí el favicon para que se vea en la pestaña y conecté el CSS para mantener la apariencia del proyecto aunque sea una página sencilla.

En el body puse un título grande que dice “RICK ROLL” para que se vea claro de qué va la página, y debajo añadí el vídeo. El vídeo se reproduce automáticamente, está sin sonido y tiene controles para que el usuario pueda pausarlo o reproducirlo cuando quiera. También puse el mensaje que aparece si el navegador no soporta vídeo, para que no quede un espacio vacío.

MINIRETO EQUIPO 4

Esta página es muy simple, porque su único objetivo es sorprender al usuario con el Rickroll. No hace falta navegación ni secciones complejas, solo mostrar el vídeo de forma directa y asegurarse de que funcione en cualquier navegador. La estética se mantiene sencilla, y todo está pensado para que cumpla su función sin complicaciones.