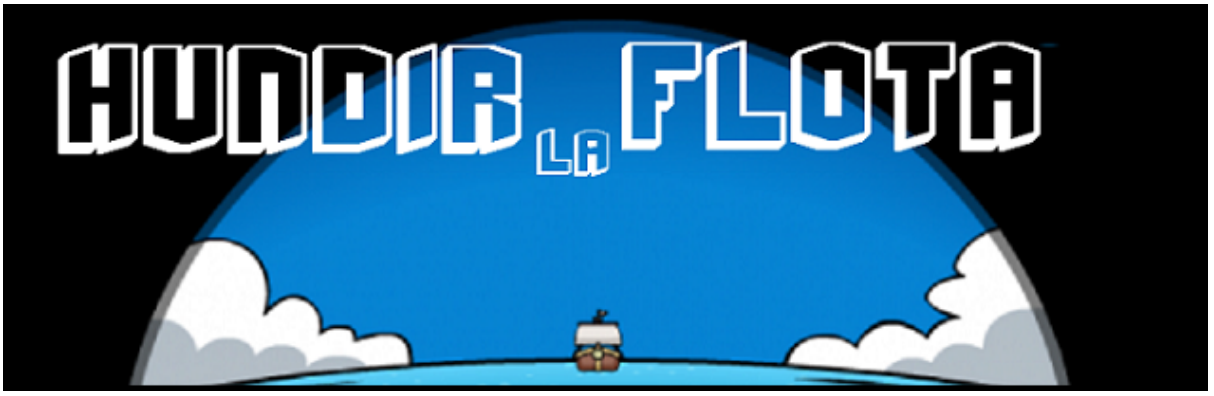
CFGS DESARROLLO DE APLICACIONES WEB

Módulo: DESARROLLO DE APLICACIONES EN SERVIDOR

Proyecto 2ªEvaluación

Hundir la flota: Juego de tablero por turnos basado en protocolo http y comunicación asíncrona .



Alumna: Elisabet Martin Muriel

Fecha 20 de Marzo 2022

GitHub https://github.com/emmuriel/hundirflota

1. Introducción

1.1. Contexto del proyecto

En este proyecto he desarrollado una aplicación web que permite jugar en línea.

El juego es una recreación del mítico juego BattleShip o Hundir la Flota. Un juego de tablero bidimensional, en el que se juega por turnos.

El objetivo del juego es bombardear la flota contraria con el mínimo número de intentos y antes de que la flota propia sea bombardeada.

En este proyecto solo se implementa una modalidad de juego:

> **Jugador-Servidor.** El jugador solicita la partida y juega contra la máquina.

¿Por qué elegí este proyecto?

Primero porque es un proyecto completo en el que el servidor ataca a la base de datos haciendo inserciones, modificaciones, consultas y borrados. Utiliza estructuras de datos como arrays y clases. Hace uso de variables de sesión y cookies.

Al ser un juego por turnos es posible implementarlo usando la arquitectura cliente-servidor.

La segunda razón es porque es un proyecto completo que se ajusta a lo pedido en la práctica y que además desarrolla e integra lo aprendido en otros módulos que componen el ciclo formativo.

La tercera razón es íntimamente personal. Ya que este proyecto parte de un proyecto personal antigüo desarrollado en 2010, pretendiendo adaptar y migrar a tecnologías más actuales y de software libre. Este proyecto también pretende ampliar funcionalidades y aumentar la seguridad de la aplicación resultante. Por lo tanto es además un ejercicio de reflexión, superación y reconstrucción personal.

1.2. Funcionamiento del juego

El juego consiste en 2 tableros, matrices de 10x10 escogidos al azar. Un tablero pertenece al jugador otro a la máquina o adversario.

Cada una de estas matrices contiene una flota de barcos que serán el objetivo a bombardear. El resto de casillas

La flota se compone de:

- un barco de 5 casillas
- un barco de 4 casilla
- dos barcos de 3 casillas cada uno.
- dos barcos de 2 casillas cada uno.

En la aplicación el turno siempre empieza el jugador. Dispara contra el tablero del Boot.

Si el disparo alcanza un objetivo, el jugador que acierta repite turno, en cambio si falla, el turno pasa al jugador contrario.

Las casillas sólo pueden ser disparadas una vez.

Gana el jugador que bombardee antes toda la flota del adversario.

2. Objetivos

Analizar, definir y adaptar las reglas del juego a un proyecto realizable en tiempo y forma ajustado al protocolo cliente-servidor.

Analizar los requisitos para crear la BBDD.

Modelar e implementar una BBDD relacional.

Insertar datos en la BBDD.

Modificar registros en la BBDD.

Eliminar registros en la BBDD.

Consultar registros en la BBDD.

Diseñar un sistema de autenticación de usuarios.

Implementar uso de sesiones para mantener la persistencia de datos.

Definir e implementar el código usando el paradigma POO en php.

Diseñar e implementar la interfaz de usuario

Implementar JSON para transferencia de información

Implementar AJAX (asincronía) para realizar las peticiones al servidor en el transcurso de la partida.

Diseñar clases y métodos. POO

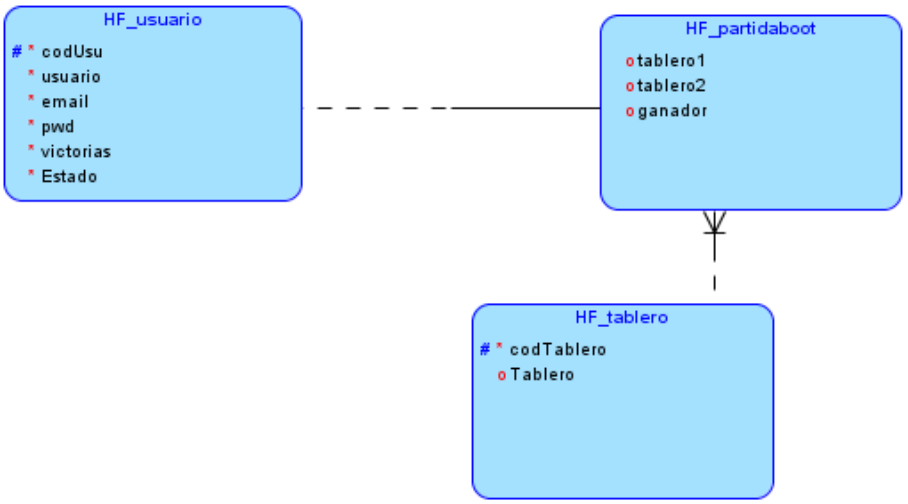
Implementar un sistema monojugador contra la máquina.

Implementar un algoritmo inteligente del lado del servidor que simule cierta lógica de juego contra un usuario.

Implementar una lógica de turnos.

3. Diseño.

Modelo Entidad-Relacion



Modelo Relacional

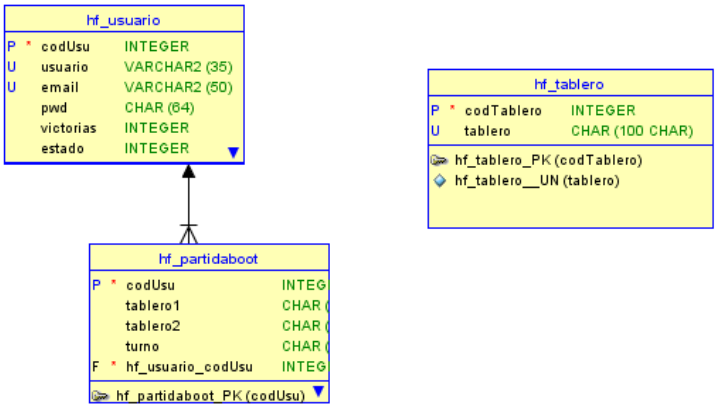
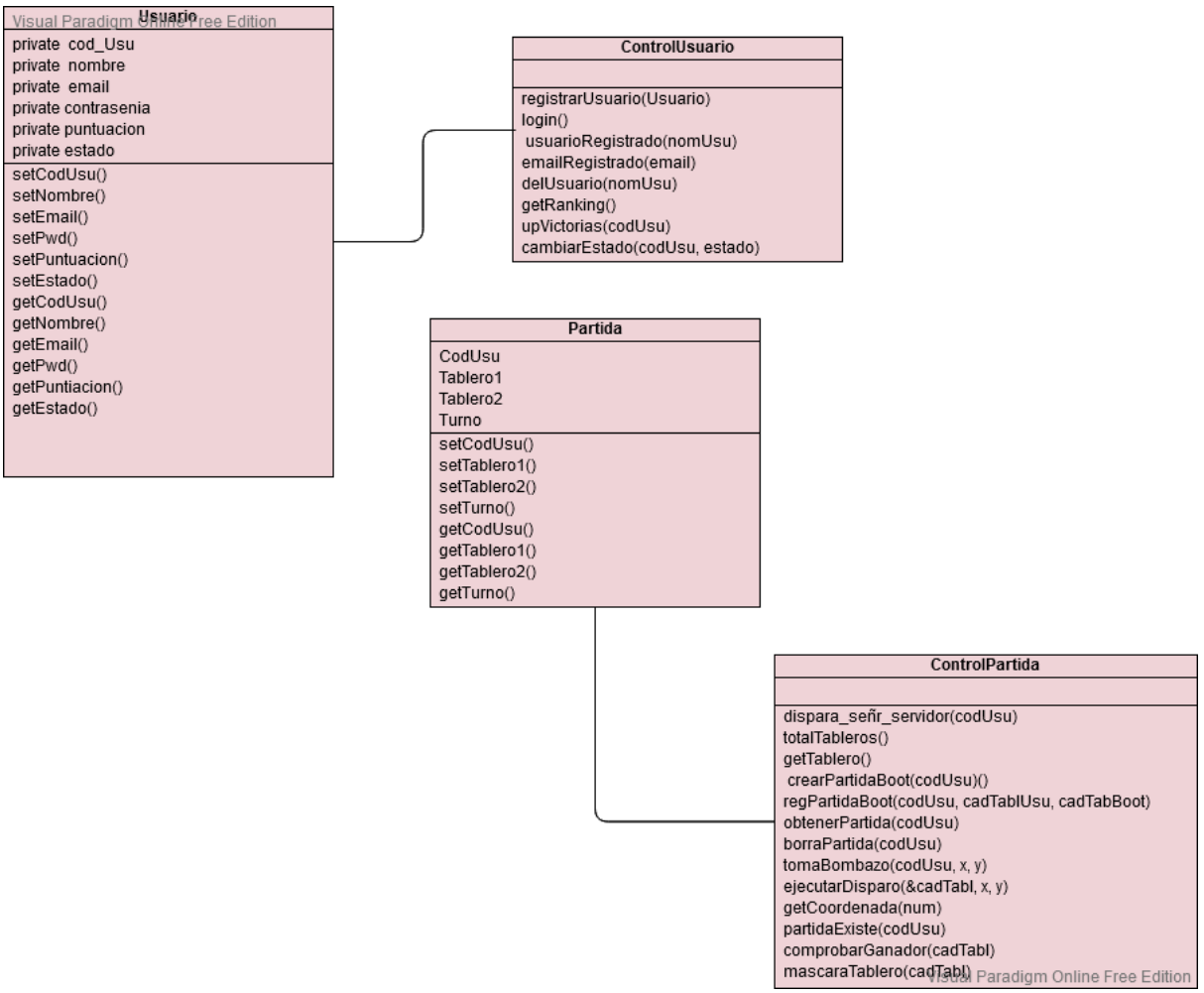


Diagrama de clases.

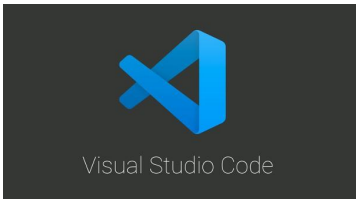


4. Codificación

1. Entorno de programación

IDE: Visual Studio Code

Editor de texto y depurador



.

Lo he escogido por ser con el que hemos estado trabajando durante el curso y por ser un editor ligero, con buenos plugins, sin coste y además integra funcionalidad intellisense que hace que editar código sea muy cómodo.

2. Lenguajes y herramientas

Sistema gestor de BBDD libre

propiedad de Oracle.

Servidor Apache

PHP. Lenguaje de programación en servidor

He utilizado el paquete XAMP para Windows que me permite instalar un servidor Apache con PHP y el SGBD MySql para poder correr la aplicación y efectuar pruebas. Son las herramientas con las que actualmente estamos aprendiendo a trabajar y además son libres.

Lenguajes de programación en cliente

Javascript.

Necesitaba de un lenguaje de script en cliente para realizar peticiones de manera asíncronas al servidor y realizar las funciones necesarias para procesar las respuestas modificando el DOM y los estilos.

HTML5. Lenguaje de hipertexto

Necesario para poder representar la información pedida al servidor.

CSS3. Hojas de estilo en cascada.

Para poder dar formato vistoso y agradable a esa información recogida por el cliente

JSON para intercambio de datos

No es un lenguaje en sí, sino un formato de documento. Es un fichero de texto plano y sirve para el intercambio de información entre servidor y cliente. En un principio este intercambio de datos se iba a hacer con XML, pero la versión más actual de php ha dejado obsoletas las funciones que permitían las respuestas asíncronas al objeto XMLhttpRequest de Javascript, siendo en la actualidad el formato más usado.

Visual Paradigm Online

Visual Paradigm

Herramienta online para

digramas de clases

Oracle Data Modeler

Git. Control de Versiones:

Repositorios:

3. Códigos fuentes relevantes

Manejo del array asociativo $_SESSION

Index.php

```
index.php > ...
1  <?php
2  session_name("HundirFlota");
3  session_start(); ?>
4  <!DOCTYPE html>
5  <html lang="es">
```

```php
<?php
/************************************************************************************************************** */
require_once("modelo/clases.php");
require_once("modelo/controlador_usuario.php");
require_once("modelo/moduloConexion.php");


    if (isset($_POST['entrar'])) {  #Login

        if (empty($_POST['txt_usuario']) || empty($_POST['txt_password'])) { //Validación de datos de entrada
            echo "<div id='error'><span class='error'>Error ** Todos los campos son obligatorios</span><div>";
        } else {
            extract($_POST, EXTR_PREFIX_ALL, 'S');
            $ctrlUsu = new ControlUsuario();
            //Sanitizacion de las variables pasada por POST
            $nombreOk= htmlspecialchars($S_txt_usuario); #elimina caracteres especiales HTML --Evita ataques XSS
            $nombreOk=htmlentities($nombreOk, ENT_QUOTES); #elimina comillas dobles y simples --evita SQLInjection
            $passOK= htmlspecialchars($S_txt_password);
            $passOK=htmlentities($passOK, ENT_QUOTES);
            $obUsu = $ctrlUsu->login($nombreOk, $passOK);  //devuelve un usuario
            if ($obUsu->getCodusu() <= 0) { //el usuario / contraseña incorrectos

                echo "<div id='error'><span class='error'>Error ** El usuario o la contraseña son incorrectos</span>/div>";
            } else {
                #AKI DEBERIAMOS MACHACAR LA SESION ANTIGUA DE LA BBD CON LA NUEVA SESION PARA EVITAR DOBLES LOGEOS
                #Esto se implementará más adelante, modificacion de la BBDD

                #Cargar usuario en Session serializando y redireccionar a partida
                $_SESSION['usuario'] = serialize($obUsu);
                header("Location: https://localhost/HF/Partida.html");
            }
        }
    }

    if (isset($_POST['registrar'])) { #Registro
        header("Location: https://localhost/HF/Registro.php");
    }
```

Aunque esté un poco antiguo, he usado la función serialize() para guardar una vez hecho el login el valor de un objeto Usuario inicializado con los valores devueltos por MySql a la hora de hacer la autenticación.

Este objeto guardado en $_SESSION me permite guardar datos como el código de usuario y obtenerlo en cualquiera de los archivos php que componen la aplicación.

También podemos ver la llamada a dos funciones muy importantes a la hora de añadir seguridad a las entradas de formulario:

htmlspecialchars() – Elimina los caracteres especiales como <>

htmlentities(*string*, ENT_QUOTES) - No permite la entrada de comillas dobles ni simples

Partida.php valida si $_SESSION guarda el valor [‘usuario’]

```php
require_once("modelo/clases.php");
require_once("modelo/controlador_partida.php");
require_once("modelo/controlador_usuario.php");
require_once("modelo/moduloConexion.php");
require_once("control/funciones.php");

//Controlar que el usuario esté logeado
if ($_SESSION['usuario']) {
  $obUsu = unserialize($_SESSION['usuario']); # Deserializacion del objeto Usuario.
  $json= json_decode(file_get_contents('php://input'),true);

  if (isset($json['peticion'])) {      #PROCESAR JSON
    $ctrlPartida = new controlPartida();
    $peticion=$json['peticion'];

    switch ($peticion) {
```

Sólo si el usuario se ha logeado correctamente existirá $_SESSION[‘usuario’] y el código de partida.php se ejecutará.

De lo contrario la cookie de session se machaca y el usuario solo podrá ver el codigo html de la aplicación, es decir Partida.html

```php
} else {
  session_destroy();
  setcookie('HundirFlota','',time()-100);
  header("Location: https://Error.php");
}


?>
```

Manejo de peticiones asíncronas Partida.php

```php
<?php
session_name("HundirFlota");
session_start();
header('Content-Type: application/json');

/************************************************************************************************************ */
require_once("modelo/clases.php");
require_once("modelo/controlador_partida.php");
require_once("modelo/controlador_usuario.php");
require_once("modelo/moduloConexion.php");
require_once("control/funciones.php");

//Controlar que el usuario esté logeado
if ($_SESSION['usuario']) {
  $obUsu = unserialize($_SESSION['usuario']); # Deserializacion del objeto Usuario.
  $json= json_decode(file_get_contents('php://input'),true);

  if (isset($json['peticion'])) {      #PROCESAR JSON
    $ctrlPartida = new controlPartida();
    $peticion=$json['peticion'];
```

Partida.php no envía HTML ante las peticiones, solo envia archivos JSON y recibe las peticiones a través de archivos JSON

El flujo de control de este archivo php es un switch con 6 casos, en los que cada uno de los casos responde a una de las peticiones hechas por el cliente.

```php
switch ($peticion) {
  case 1:  #CARGAR DATOS USUARIO
    $datosUsu = utf8_encode('BIENVENID@ ' . $obUsu->getNombre() . '!!      Total Partidas Ganadas:  ' . $obUsu->getPuntuacion());
    $response = array ("datosUsu"=>$datosUsu);
    echo json_encode($response,JSON_FORCE_OBJECT,512); // al array hay que forzarlo a ser objeto
    break;

  case "2": #CREAR NUEVA PARTIDA

    //Comprueba que ya no haya una partida jugandose

    $existePartida = $ctrlPartida->partidaExiste($obUsu->getCodUsu());
    if (!$existePartida) {
      //Empieza partida
      $ctrlPartida->crearPartidaBoot($obUsu->getCodUsu());
      $partida = $ctrlPartida->obtenerPartida($obUsu->getCodUsu());
      #Enmascarar tablero boot
      $tablero2=$ctrlPartida->mascaraTablero($partida->getTablero2());
      $partida->setTablero2($tablero2);

      responseJson($obUsu, $partida, "0");
    }else{
      #Enmascarar tablero boot
      $partida = $ctrlPartida->obtenerPartida($obUsu->getCodUsu());
      $tablero2=$ctrlPartida->mascaraTablero($partida->getTablero2());
      $partida->setTablero2($tablero2);

      responseJson($obUsu, $partida, "0");
    }
    break;

  case "3": #PROCESAR DISPARO USUARIO

    $x=$json['x'];   //Recoge las coodenadas mandadas x json
    $y=$json['y'];
    procesarComprobacionDisparo($obUsu,$x,$y);
    break;
```

```php
case "4": #PROCESAR RECARGA... dispara señor servidor...

  $ganador;
  $car_gan;
  $ctrlPartida->dispara_señr_servidor($obUsu->getCodUsu());

  $partidaActualizada = $ctrlPartida->obtenerPartida($obUsu->getCodUsu());

  #Comprobar ganador
  $ganador = $ctrlPartida->comprobarGanador($partidaActualizada->getTablero1());

  if ($ganador == true) {
    $car_gan = "2";  //Gana señor servidor
  } else {
    $car_gan = "0";  //No hay ganador
  }
  #Enmascarar tablero boot
  $tablero2=$ctrlPartida->mascaraTablero($partidaActualizada->getTablero2());
  $partidaActualizada->setTablero2($tablero2);
  responseJson($obUsu, $partidaActualizada, $car_gan);  #Response server

  break;

case "5": #ABANDONAR PARTIDA
  $ctrlPartida->borraPartida($obUsu->getCodUsu());
  $ok=1;
  $oka= array ("ok"=>$ok);
    echo json_encode($oka, JSON_FORCE_OBJECT,3);  #Response server
  break;

case "6": #ABANDONAR PARTIDA  Y CERRAR SESION
  $ctrlPartida->borraPartida($obUsu->getCodUsu());
  $ctrlUsu = new ControlUsuario();
  $erro = $ctrlUsu->cambiarEstado($obUsu->getCodUsu(), 0);
  logout();
```

Estas peticiones se producen de manera asíncrona desde el cliente siendo los diferentes eventos desencadenantes los que las ejecutan.

El resultado de estas peticiones es un JSON con la información necesaria para que el cliente procese la respuesta, generalmente son actualizaciones de la partida. Este fichero php hace uso de las clases **Partida, Usuario, controlUsuario y controlPartida**

La mayoría de las respuestas JSON son generadas con la función *responseJson()* del archivo funciones.php

```php
/*'-----------------------------------------------------------------------------------
    ' Nombre: responseJson
    ' Proceso:  Concatena los datos de usuario y de la partida para mandar una respuesta http
    ' Entradas: un objeto Usuario, Un objeto partida, y un entero que indica quien es el ganador.
    ' Salidas: no tiene, envia la cadena por http al cliente en background
    '-----------------------------------------------------------------------------------*/
function responseJson($usu, $partida, $ganador)
{
  $respuesta= array("usuario"=>$usu->getCodUsu(),
                    "partida"=>$partida->getTablero1() . "|" . $partida->getTablero2(). "|",
                    "turno"=>$partida->getTurno(),
                    "ganador"=>$ganador);
  echo json_encode($respuesta,JSON_FORCE_OBJECT,512);
}
```

El método que mantiene el control del turno de disparo es:

ControlPartida::tomaBombazo

```php
 * Salidas:  Cambia la cadena tablero de la BBDD y actualiza turno segun se acierte o no y el
             jugador que esté jugando.
 *------------------------------------------------------------------------------------------ */
public function tomaBombazo($codUsu, $x, $y)
{
    $cadena_tablero = "";
    #Comprobar turno

    #abrir conexion
    $conexion = conexionBBDD();
    $resultado = $conexion->query("SELECT turno, tablero1, tablero2 FROM hf_partidaBoot WHERE codUsu=$codUsu");
    $resultado->data_seek(0);
    while ($fila = $resultado->fetch_assoc()) { //Si el usuario tiene partida. Carga tableros y turno
        $turno  = intval($fila['turno'],10);
        $tableroUsu   = $fila['tablero1'];
        $tableroBoot   = $fila['tablero2'];
    }
    $conexion->close();

    #SI EL TURNO ERA DEL JUGADOR comprobar coordenadas en tablero2
    if ($turno == 1) {

        $acierto = self::ejecutarDisparo($tableroBoot, $x, $y);
        if ($acierto == 1) {
            $nuevo_turno = 1;  //Turno sigue en jugador
        } else {
            $nuevo_turno = 0; //Turno pasa a señor_servidor
        }
    } else { # turno==0 TURNO DE SERVIDOR: comprobar coodenadas en $tableroUsu
        $acierto = self::ejecutarDisparo($tableroUsu, $x, $y);
        if ($acierto == 1) {
            $nuevo_turno = 0;  //Turno sigue en señor_servidor
        } else {
            $nuevo_turno = 1; //Turno pasa a jugador
        }
    }

    //echo "el turno es". $nuevo_turno;

    #abrir conexion, actualizar la partida
    $conexion = conexionBBDD();
    $resultado = $conexion->query("UPDATE hf_partidaBoot SET tablero1='" . $tableroUsu . "' , tablero2 = '" . $tableroBoot . "', turno='" . $nuevo_turno . "' WHERE codUsu=$codUsu");
    $conexion->close();
}
```

Este método recibe como parámetro el código del usuario al que se está disparando (sea del jugador o del boot) llama al método de la propia clase: *ejecutarDisparo()* que según la posición de la coordenada comparará el contenido de la cadena y retorna un valor booleano verdadero si se ha acertado en el disparo o falso de los contrario.

controlPartida::ejecutarDisparo()

modelo > controlador_partida.php > PHP Intelephense > controlPartida > ejecutarDisparo

```php
public function ejecutarDisparo(&$cadTabl, $x, $y)
{
    $diana = 0;
    $cadena = ""; //String
    #Pasar la cadena a array de caracteres
    $cpCadTabl=$cadTabl;
    $arr_cad = str_split($cpCadTabl);

    $posicion= $arr_cad[$x * 10 + $y];
    switch ($posicion){
         #COMPROBAR DISPARO
        case "0":
            $arr_cad[$x * 10 + $y] = "#";  //AGUA
            $diana = 0;
            break;

        case "1":
            $arr_cad[$x * 10 + $y] = "x"; //TOCADO'
            $diana = 1;
            break;
        case "2":
            $arr_cad[$x * 10 + $y] = "x"; //TOCADO'
            $diana = 1;
            break;
        case "N":
            $arr_cad[$x * 10 + $y] = "x"; //TOCADO'
            $diana = 1;
            break;

        case "S":
            $arr_cad[$x * 10 + $y] = "x"; //TOCADO'
            $diana = 1;
            break;
        case "W":
            $arr_cad[$x * 10 + $y] = "x"; //TOCADO'
            $diana = 1;
            break;
        case "E":
            $arr_cad[$x * 10 + $y] = "x"; //TOCADO'
            $diana = 1;
            break;
    }

         #VOLVER A PASAR MATRIZ A CADENA contatenando caracteres
         for ($i = 0; $i <100; $i++) {

                $cadena .= $arr_cad[$i];
        }
    $cadTabl = $cadena;  #SOBRESCRIBIR LA CADENA DE ENTRADA/SALIDA (* pasada por referencia)
```

Recibe la cadena del tablero por referencia y las coordenadas, que en el caso del servidor se generan aleatoriamente y el el caso del usuario son recogidas a través de la petición número 3 del usuario

En el proceso. La cadena pasa a ser un array unidimensional en el que la posición será el índice [$x*10+$y]. Según su contenido determinaremos si había un barco y ha sido tocado o si había agua y el disparo ha sido fallido. También se modifica el valor del array para [x*10+y] y se vuelve a pasar a cadena.

En conclusión este método devuelve un booleano que permite determinar el turno de juego y la cadena modificada con el disparo que permite actualizar la partida.

Siguiente con la lógica de la partida, la partida solo termina por 3 motivos:

1- que el usuario abandone la partida: #Recogido en la petición 5

2- Que el usuario cierre el navegador: #Recogido en la petición 6

3- Que alguien gane la partida. Existe un método que se encarga de controlar este supuesto:

ControlPartida::Comprobar ganador

```php
/* '-----------------------------------------------------------------------------
' Nombre: comprobarGanador
' Proceso: Recorre un array generado con la cadena del tablero, determinando si aun
'           existe una casilla de barco sin explotar (caracter "1")
' Entradas: una cadena
' Salidas: Un booleano, True si no existen casillas con el caracter "1", False de lo
'           contrario.
'----------------------------------------------------------------------------- */
public function comprobarGanador($cadTabl)
{
    $ganador = true;
    $arr_cad = array(100);

    #Pasar la cadena a array de caracteres
    $arr_cad = str_split($cadTabl);
    for ($i = 0; $i <= 99; $i++) {
        if ($arr_cad[$i] == "1" || $arr_cad[$i] == "2" || $arr_cad[$i] == "N" || $arr_cad[$i] == "S" || $arr_cad[$i] == "W" || $arr_cad[$i] == "E") {
            $ganador = false;
        }
    }
    return $ganador;
}
```

Sin este método sería imposible determinar si la partida ha llegado a su fin, así que cada vez que se actualiza la partida por un disparo. Hay que comprobar si la partida terminó. Esta comprobación se hace tanto en el servidor, como en el cliente.

Recorre un array generado con la cadena de la partida actualizada comprobando si queda alguna posicion con un contenido explotable.

Cabe destacar que los contenidos explotables son los caracteres que representan a la flota y son : N-S-E-W-1-2.

Si no queda ninguno de estos caracteres en el tablero del enemigo después de ejecutar el último disparo, el artífice del disparo es el ganador,.

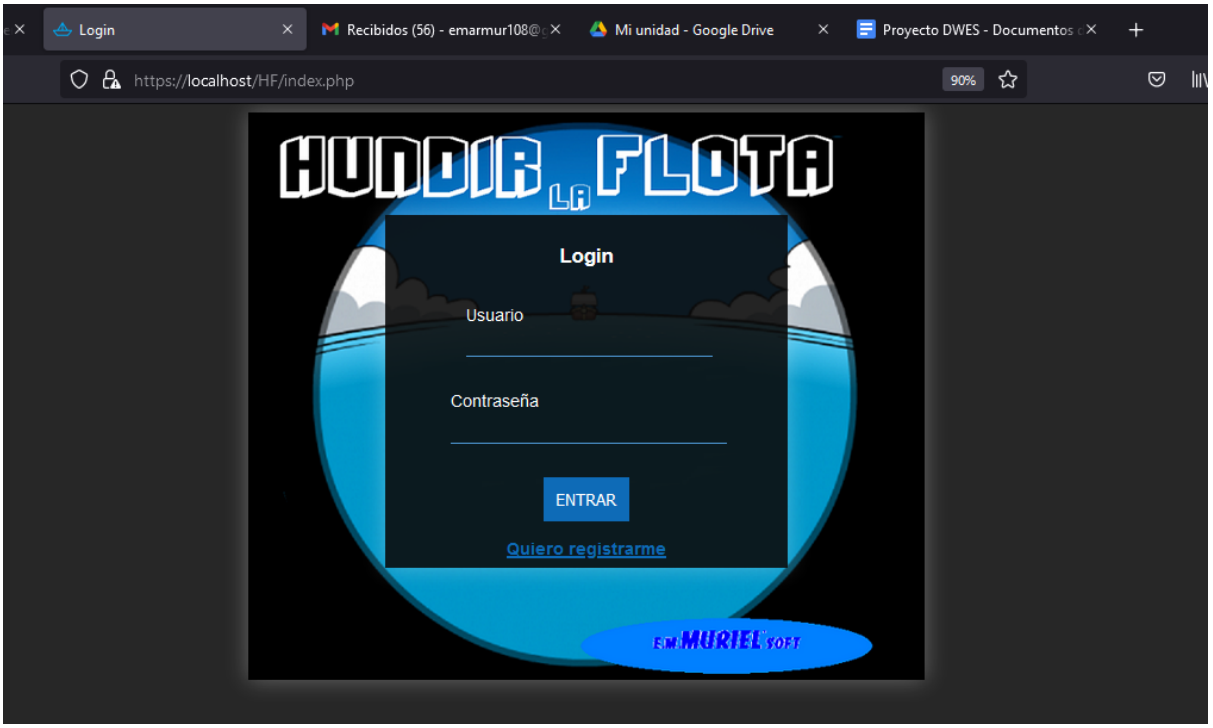
5. Pruebas de ejecución

Al autenticarnos correctamente nos redirige a HF/Partida.html

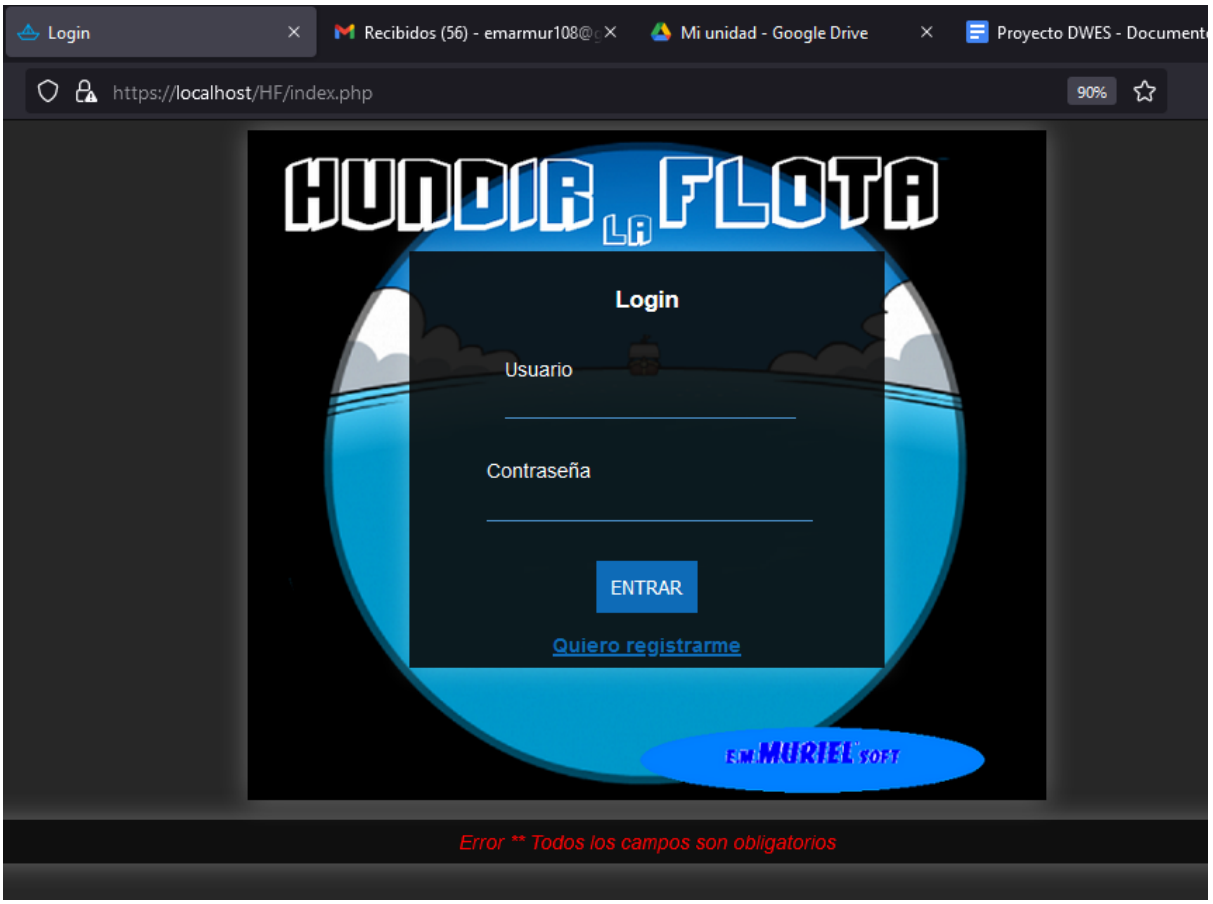
Partida.html nos muestra un mensaje de bienvenida con el total de partidas ganadas.

Se ha producido una petición asíncrona (peticion=1) desde javascript y el servidor contesta con la cadena

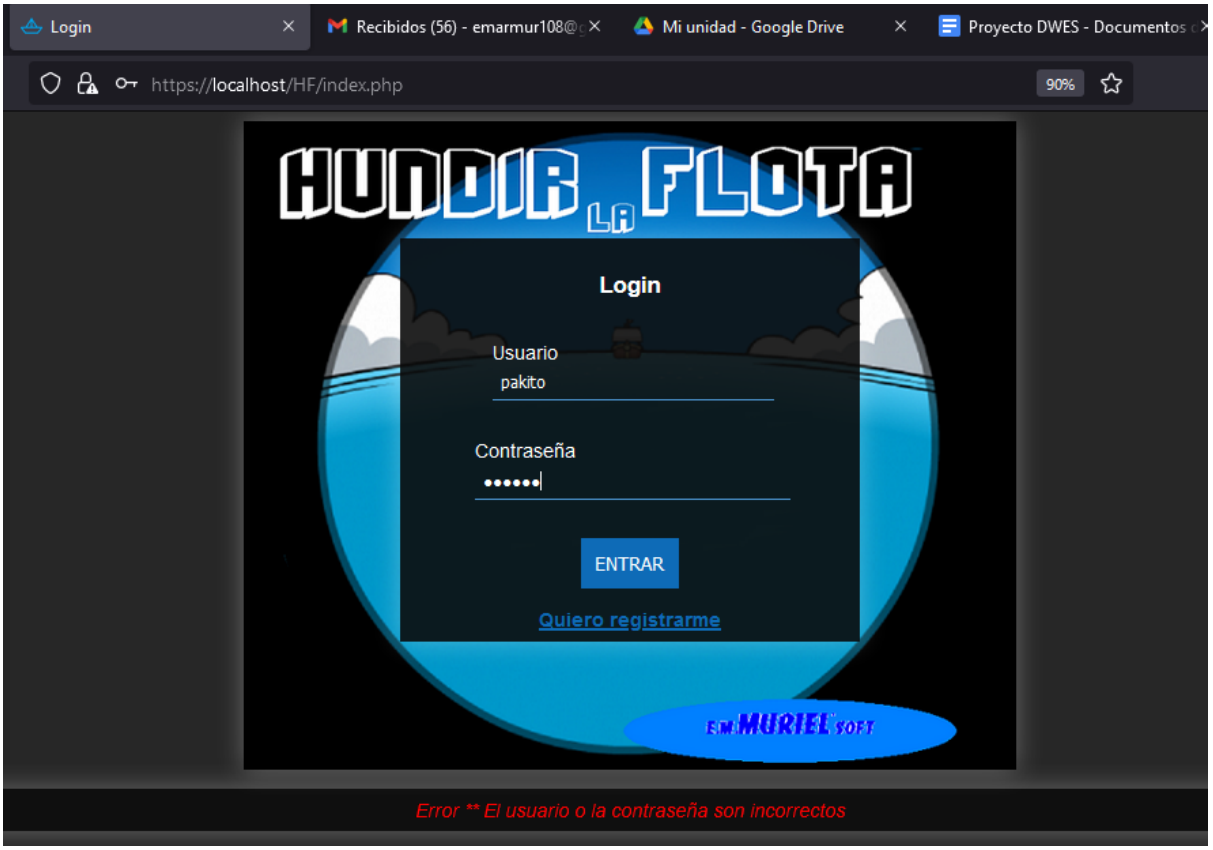
Login . Index.php



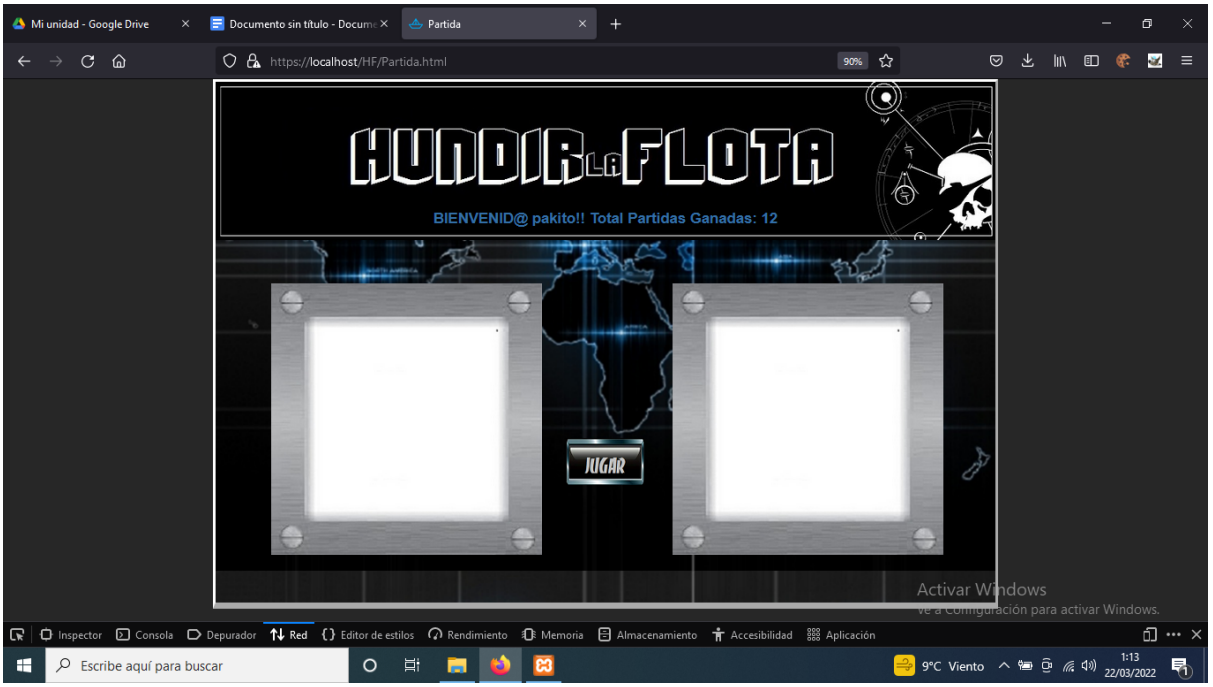
Errores de Login. Campos vacíos.



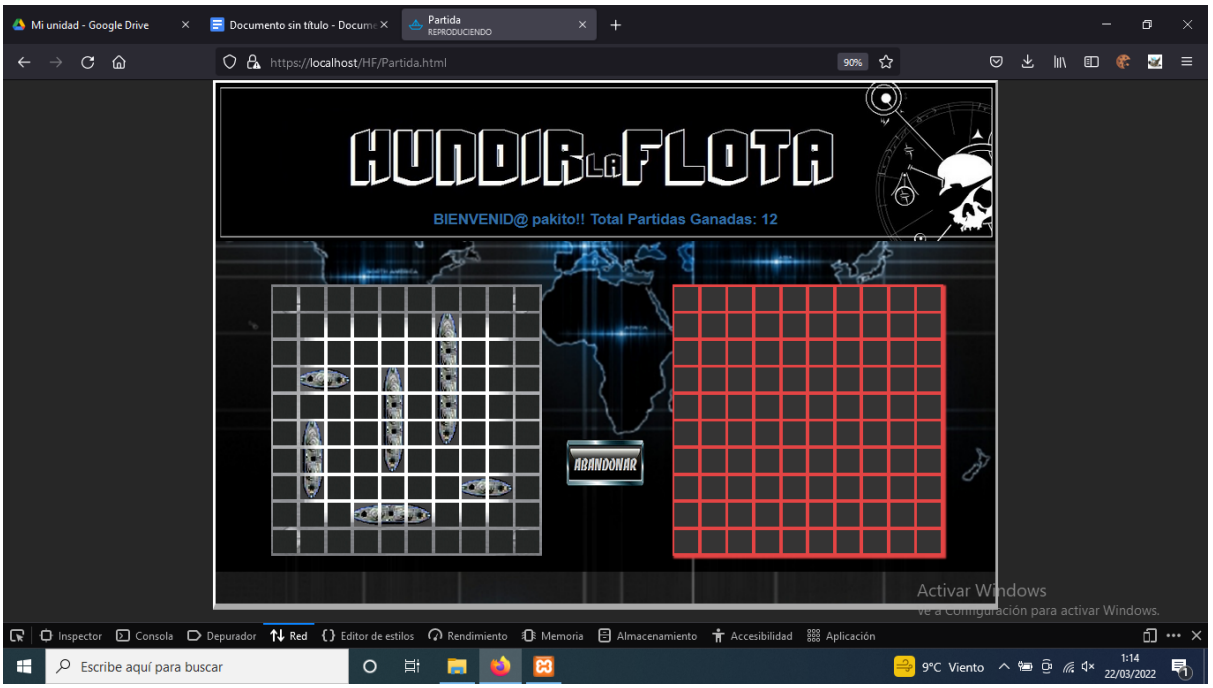
Error de login: Usuario o contraseña no válido:



Login correcto. Carga datos de usuario . Partida.html



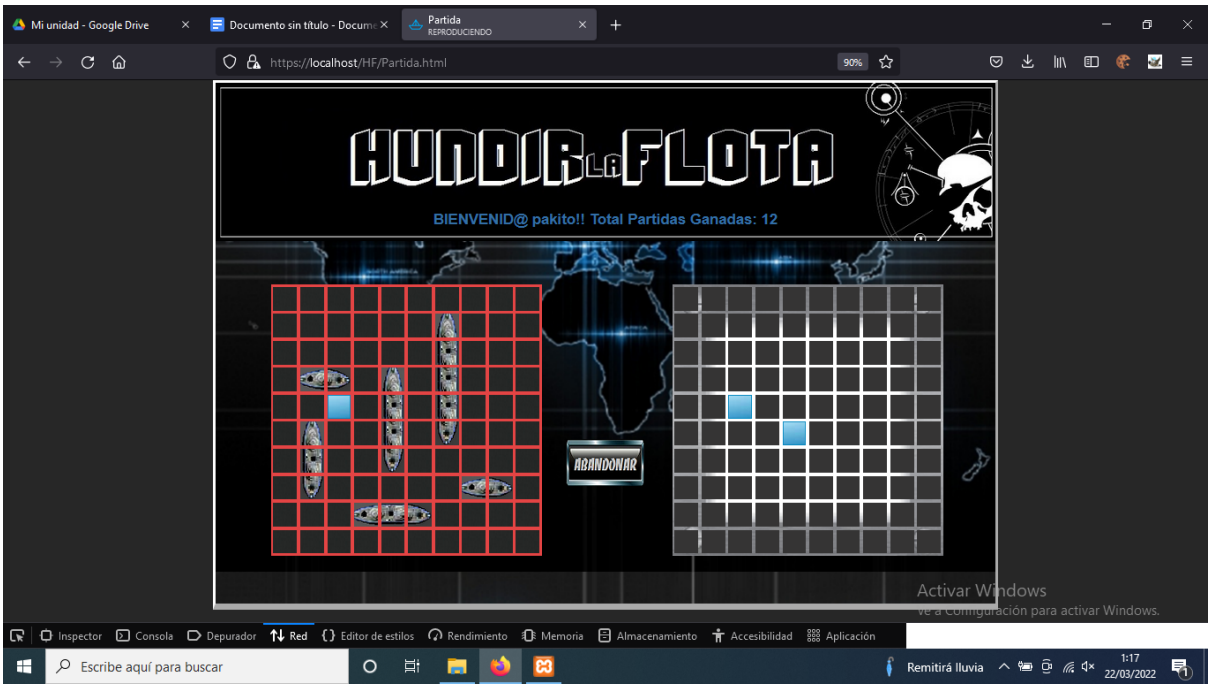
Cuando hacemos click en el boton “jugar” . El evento disparado realiza una petición de tipo 2. Crear una nueva partida. El servidor crea la partida envia un jason con los datos. Javascritp crea los tableros y añade los estilos.

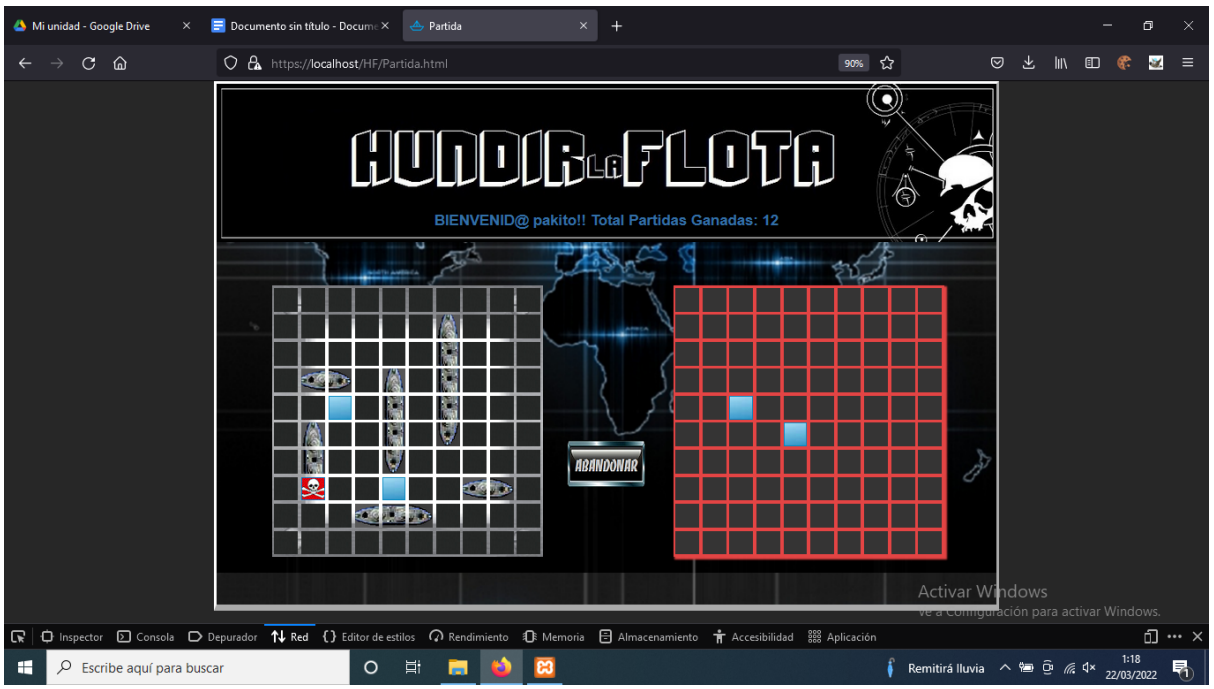


Siempre empieza el Usuario. Es decir el tablero en rojo.

Probar disparos y manejo del turno:

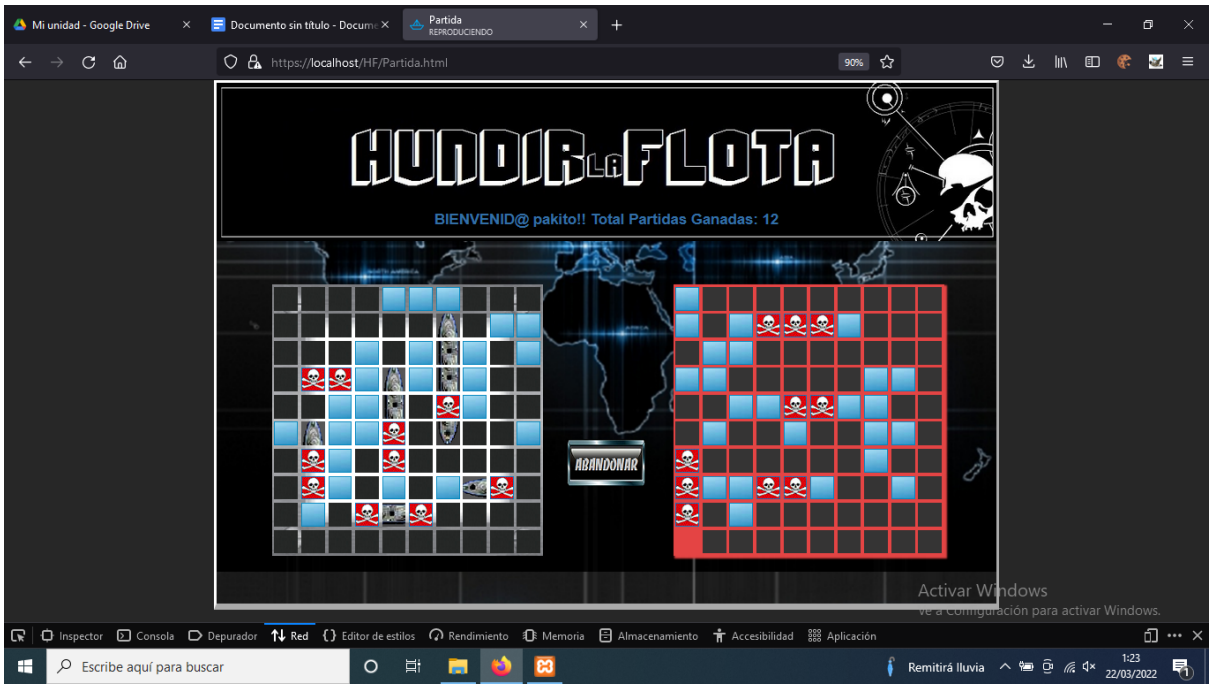
Tras un disparo fallido el turno pasa al servidor. Podemos ver el color rojo en nuestro tablero.



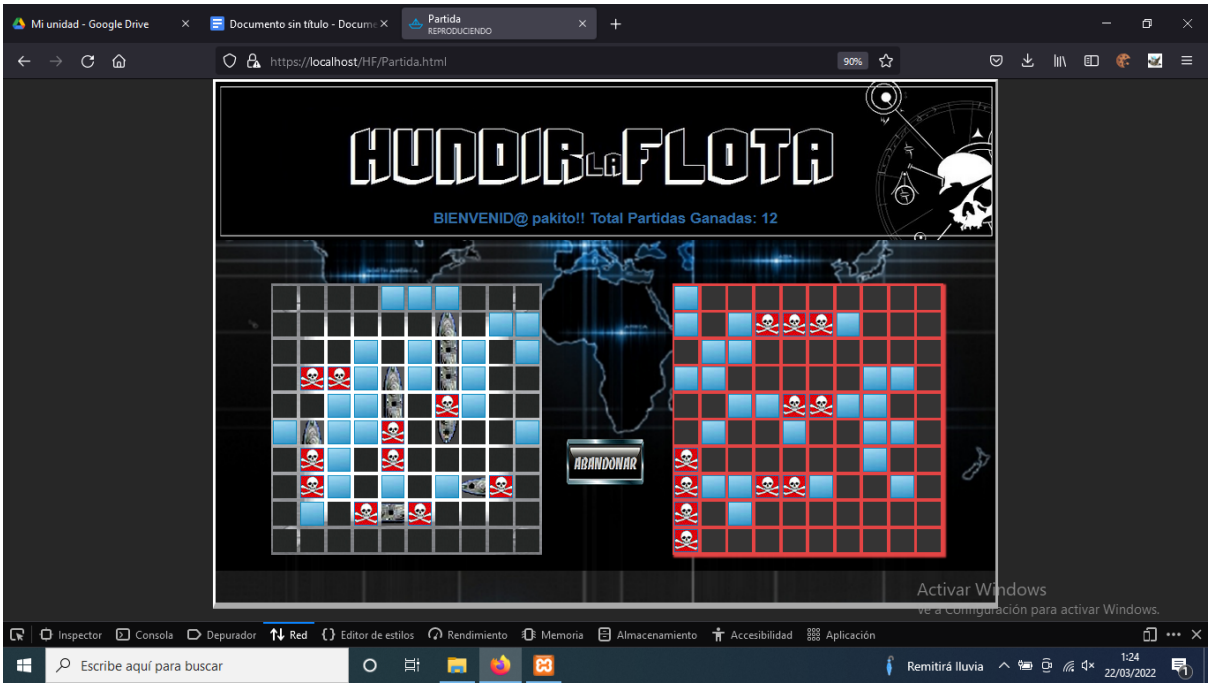


El servidor nos ha hecho un tocado y ha vuelto a tirar. Esta vez dando en agua ya que no hay implementada aún un algoritmo inteligente que rastree los caminos a seguir y las coordenadas a las que dispara son aleatorias.

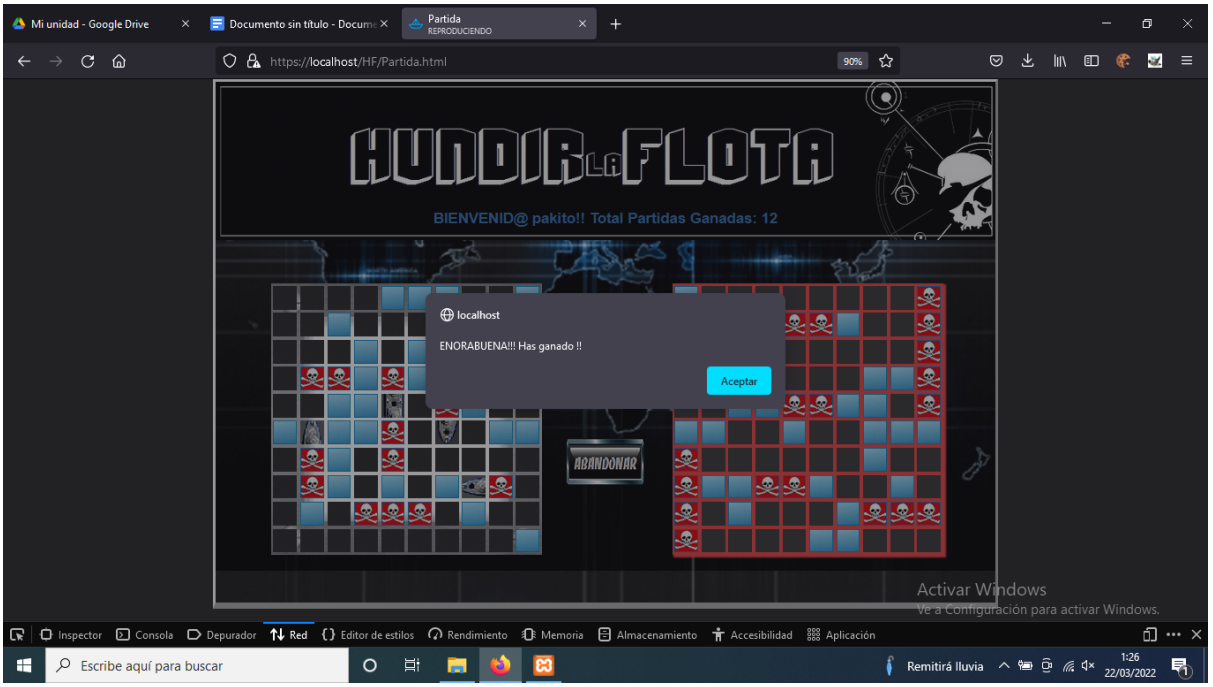
Después de un disparo fallido turno de jugador.



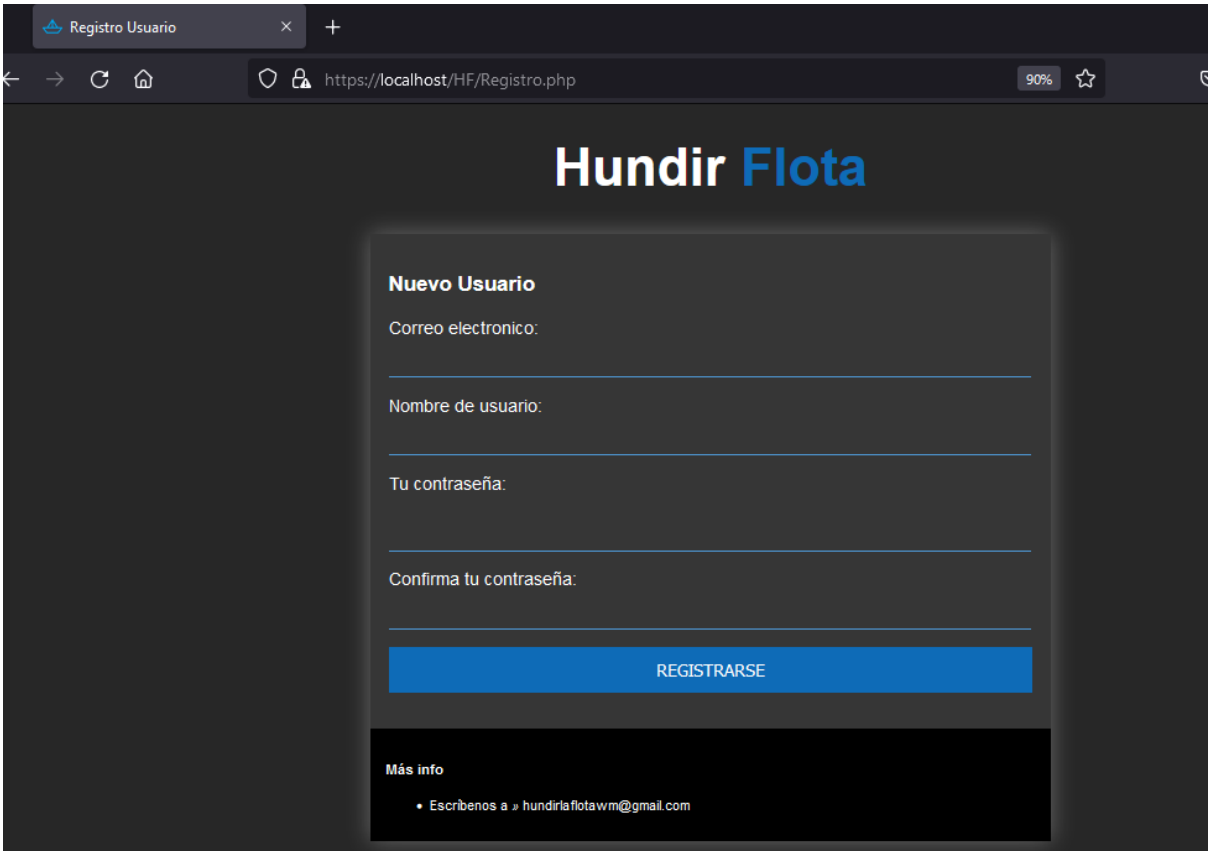
Podemos observar como despues de acertar el disparo seguimos teniendo el turno



Ganador partida usuario:

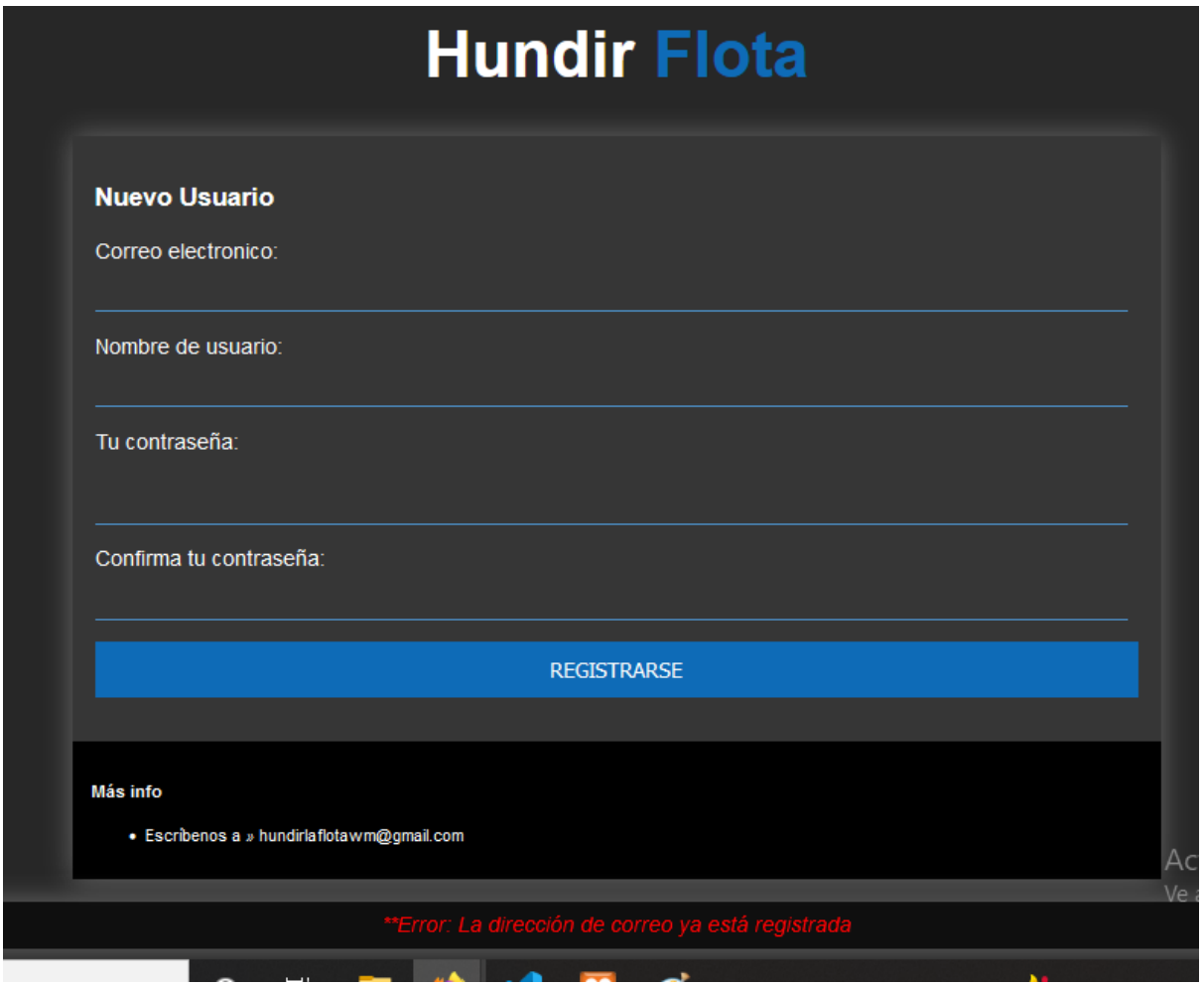


Formulario de Registro. Registro.php



Errores de formulario. Campo vacio

Correo/ usuario ya registrado. (recordar que son campos con restricción Unique en la BBDD)



También valida contraseña inseguras y que las contraseñas coincidan.

6. Conclusiones

Json es un formato muy cómodo y ampliamente usado en la actualidad para transferencia de datos.

Serialize() pese aser una función más antigua es muy cómoda a la hora de poder guardar objetos en sesión.

Implementar un algoritmo que permita al servidor una jugada “lógica” es una tarea compleja que necesita de un análisis más profundo del que he podido darle en el tiempo que he tenido, no obstante será el siguiente paso para mejorar la aplicación.

Me pregunto si una base de datos no relacional sería más adecuada para esta aplicación, ya que la tabla partidadBoot no sigue los principios relacionales. Realmente esta tabla podría haberse suprimido de la BBDD y guardar los datos de la partida en $_SESSION,

pero en parte la práctica consiste en atacar la BBDD desde el servidor y es lo que se ha pretendido con ese modelo.

Es interesante ver la combinación las tecnologías aprendidas en los diferentes módulos y ver cómo se integran entre ellas para conseguir una proyecto más o menos completo. La integración entre los componentes de la aplicación le añade complejidad pero sin duda con las herramientas aprendidas y un poco de imaginación pueden hacerse aplicaciones muy vistosas.

7. Bibliografía

Webgrafía

https://www.php.net

Imitar la sobrecarga de constructores en php.

https://desarrolloweb.com/articulos/sobrecarga-constructores-php.html

Multitud de consultas (errores, Json, etc)

https://stackoverflow.com

https://es.stackoverflow.com

https://www.anerbarrena.com/php-array-tipos-ejemplos-3876/

https://coderwall.com/p/p2kumg/json_encode-vs-serialize-with-php-arrays

https://www.w3schools.com/php/func_string_str_split.asp

https://stackoverflow.com/questions/804045/preferred-method-to-store-php-arrays-json-encode-vs-serialize

https://tursos.com/como-definir-el-tiempo-de-vida-de-una-variable-de-sesion-en-php/

https://mixkit.co/free-sound-effects/