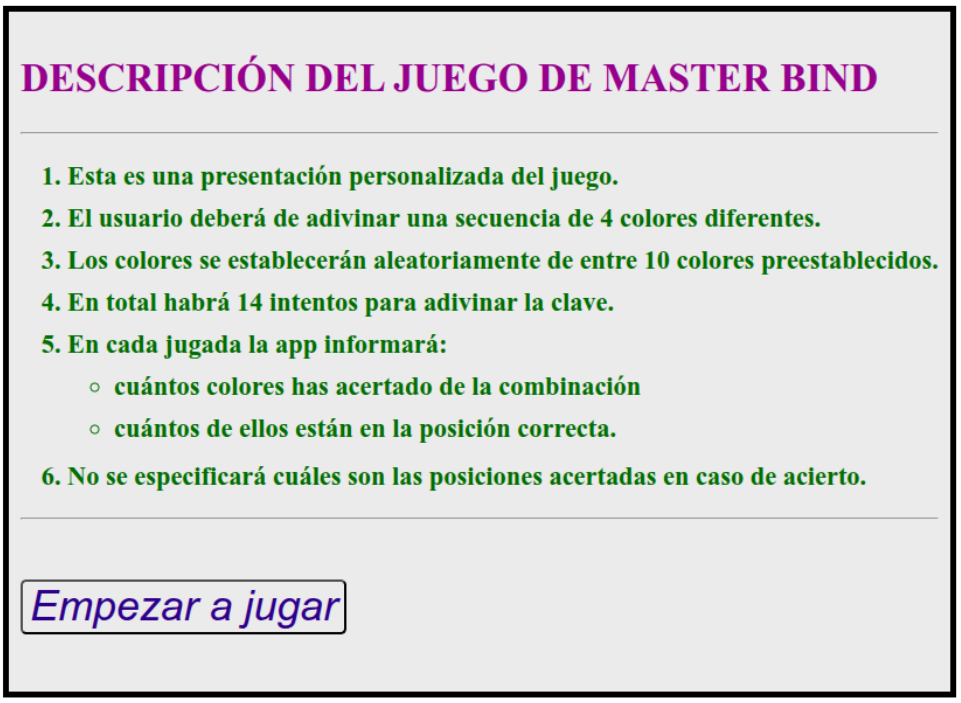
MasterMind - Examen DWES

El objetivo del juego es el siguiente:



Para esto vamos a tener que crear en principio 3 clases, 4 archivos + el archivo de Composer, paso a paso.

Lo 1º va a ser crear la carpeta “src” en la cual crearemos las clases necesarias, Clave.php, Plantilla.php y Jugada.php.

Después de eso vamos a crear el composer.json e incluir ahí lo que necesitamos para que reconozca las clases, el autoload y el psr-4:

```json
{
  "name": "usuario/mastermind-examen",
  "description": "Proyecto Mastermind para examen de entorno servidor.",
  "type": "project",
  "authors": [
    {
      "name": "Tu Nombre",
      "email": "tu.email@ejemplo.com"
    }
  ],
  "autoload": {
    "psr-4": {
      "MasterBind\\": "src/"
    }
  },
  "require": {}
}
```

Después de crearlo, hacemos un composer update en la terminal.

Ahora creamos nuestro archivo controlador.php, va a ser el archivo más importante de toda la aplicación pero ahora no lo vamos a codificar entero, primero requerimos el autoload, importamos las 3 clases que hemos mencionado antes e iniciamos una sesión que nos permitirá acceder a la clave y jugadas:

```php
<?php

    require 'vendor/autoload.php';

    use MasterBind\Clave;
    use MasterBind\Jugada;
    use MasterBind\Plantilla;

    session_start();
?>
```

Más tarde ya haremos el switch con toda la funcionalidad pero primero tenemos que programar estas clases que hemos indicado.

Clave.php

Esta clase tiene como objetivo clave gestionar la clave secreta, para ello vamos a necesitar lo siguiente:

1. Incluir el namespace MasterBind que hemos especificado en el composer.json
2. Los colores para generar la clave y una variable para guardar la clave:

```php
// Colores disponibles
public const COLORES = ['Azul', 'Rojo', 'Naranja', 'Verde', 'Violeta', 'Amarillo', 'Marron', 'Rosa'];

// Almacena la clave secreta actual
private static $clave = [];
```

$clave va a ser estática porque queremos que sea de la clase, no del objeto.

3. obtener_clave(), el método para asegurarnos de que la clave sea persistente y el único método que se va a llamar desde el controlador:

```php
public static function obtener_clave() {
    // Lógica de Persistencia
    if(isset($_SESSION['clave'])) {
        self::$clave = $_SESSION['clave']; // Si ya existe, la carga desde la sesión
    } else {
        // Lógica de Generación
        self::genera_clave();
        $_SESSION['clave'] = self::$clave;
    }
    return self::$clave;
}
```

Si ya está en la sesión, se recoge y si no está se llama al método genera_clave():

4. genera_clave(), poco que explicar:

```php
private static function genera_clave() {
    self::$clave = [];
    $colores = self::COLORES;

    // Función clave: array_rand selecciona 4 índices aleatorios y ÚNICOS
    $posiciones = array_rand($colores, num: 4);
    foreach ($posiciones as $posicion) {
        self::$clave[] = $colores[$posicion];
    }
}
```

5. get_clave_html(), devuelve el contenido HTML para la interfaz:

```php
public static function get_clave_html() {
    $clave_html = "";
    foreach (self::$clave as $color){
        $clave_html .= "<div style='background-color:".$color.";'></div>";
    }
    return $clave_html;
}
```

Plantilla.php

El único objetivo de esta clase es generar el formulario que se va a ver en jugar.php pero para externalizar la interfaz de la lógica lo hacemos en esta clase:

```php
static public function genera_formulario_juego() : string {
    $html_select = "";

    // 1. Obtener lista de colores
    $colores = Clave::COLORES;

    // 2. Bucle para generar 4 select
    for($n = 0; $n < 4; $n++) {
        // MUY IMPORTANTE: Se usa 'combinacion[]' para que PHP lo reciba como array
        $html_select .= "<select name='combinacion[]'>";

        // Opción por defecto (disabled selected)
        $html_select .= "<option value='' disabled selected>-- Color " . ($n + 1) . " --</option>";

        // 3. Loop secundario: Agregar todas las opciones de color
        foreach ($colores as $color) {
            // El 'value' es lo que se enviará en el POST, y la 'class' es para CSS
            $html_select .= "<option class='$color' value='$color'> $color </option> ";
        }

        // Cierre del <select>
        $html_select .= "</select>";
    }
    return $html_select;
}
```

Un método estático, para no tener que instanciar un objeto Plantilla, que genera los 4 select con los que se trata de adivinar la clave.

Ahora vamos a otra clase muy importante

Jugada.php

Jugada como su nombre indica va a evaluar la selección del usuario y compararla con la clave generada.

Vamos a necesitar 3 propiedades en esta clase: colores, posiciones_acertadas y colores_acertados. Estas 2 últimas las vamos a inicializar a 0 en el constructor mientras que colores va a ser el array de la jugada que vamos a recibir de los 4 select que ha realizado el usuario:

```php
class Jugada
{

    private $colores;
    private $posiciones_acertadas;
    private $colores_acertados;

    public function __construct(array $jugada)
    {
        $this->posiciones_acertadas = 0;
        $this->colores_acertados = 0;
        $this->colores = $jugada;
        $clave = Clave::obtener_clave();
        $this->evalua_jugada($clave);
    }
```

Ahora vamos a ver el método más importante de Jugada que es evalua_jugada(), aquí necesitamos dar valor a posiciones_acertadas y colores_acertados

```php
private function evalua_jugada(array $clave)
{
    $jugada = array_unique($this->colores);//!Quitamos los duplicados de la jugada

    foreach ($jugada as $color) {
        // Si el color se encuentra en la clave aunque no sea en la misma posición, sumamos
        if (in_array($color, $clave))
            $this->colores_acertados++;
    }
    foreach ($this->colores as $posicion => $color)
        // Si el color está en la misma posición que en la clave, sumamos
        if ($color == $clave[$posicion])
            $this->posiciones_acertadas++;
}
```

Importante que a diferencia del MasterMind clásico aquí no tenemos en cuenta que las fichas negras(posición + color) se resten a las fichas blancas (color sin posición).

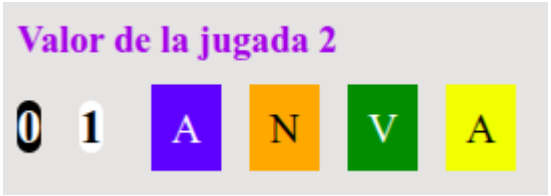
2 getters simples de posiciones y colores:

```php
public function get_posiciones_acertadas()
{
    return $this->posiciones_acertadas;
}

public function get_colores_acertados()
{
    return $this->colores_acertados;
}
```

Este toString() va a ser lo que vamos a ver después de cada jugada:

```php
public function __toString(): string
{
    $jugada_html = "";
    for ($n = 0; $n < $this->posiciones_acertadas; $n++)
        $jugada_html .= "<span class='negro'>$n</span>";
    for ($n = $this->posiciones_acertadas; $n < $this->colores_acertados; $n++)
        $jugada_html .= "<span class='blanco'>$n</span>";
    foreach ($this->colores as $color)
        $jugada_html .= "<span class='$color'>$color</span>";
    return $jugada_html;
}
```



Finalmente el obtener_historico_jugadas():

```php
public static function obtener_historico_jugadas(): string
{
    $html = "";
    $jugadas = $_SESSION['jugadas'];
    foreach ($jugadas as $pos => $jugada) {
        // El unserialize las convierte otra vez en objetos Jugada, para el toString()
        $jugada = unserialize($jugada);
        $html .= " Jugada $pos: $jugada<br />";
    }
    return $html;
}
```

Para ver todo el acumulado de jugadas, básicamente el acumulado de todos los toString() de las jugadas previas.

Hasta aquí la clase de Jugada y ya hemos programado todas las clases necesarias, ahora vamos a por la lógica de controlador, finJuego y jugar.

Controlador.php

Vamos a terminar con controlador.php:

Tenemos 2 objetivos en este archivo, una función que evalúe si el juego ha acabado y un switch para cada acción que puedes hacer (Mostrar Clave, Ocultar Clave, Resetear Clave y Jugar).

Para empezar con evaluar_fin_juego va a recibir un objeto Jugada que se crea en la opción de Jugar en el switch, tras recibir los select del usuario. Para el final del juego hay 2 opciones, o el jugador ha acertado los 4 colores en su posición, o ha consumido más de 10 jugadas:

```php
function evaluar_fin_juego(Jugada $jugada)
{
    // Si ha acertado todas las posiciones, redirigimos a finJuego con el
    // parámetro de win en true
    if($jugada->get_posiciones_acertadas() == 4) {
        $win = true;
        header( header: "location:finJuego.php?win=$win");
        exit;
    }
    // Si ya lleva más de 10 jugadas, redirigimos a finJuego con el
    // parámetro de win en false
    if(sizeof($_SESSION['jugadas']) >= 10) {
        $win = false;
        header( header: "location:finJuego.php?win=$win");
        exit;
    }
}
```

Ahora vamos a programar el motor del juego que es el switch:

```php
$opcion = $_POST['submit'] ?? "";
switch($opcion) {
    case "Mostrar Clave":
        $mostrar_ocultar_clave = "Ocultar Clave";
        $informacion = Clave::obtener_clave();
        break;
    case "Ocultar Clave":
        $mostrar_ocultar_clave = "Mostrar Clave";
        break;
    case "Resetear la Clave":
        session_destroy(); // Elimina todas las jugadas y clave previas
        session_start(); // Inicia una nueva sesión
        $clave = Clave::obtener_clave(); // Crea una nueva clave
        break;
    case "Jugar":
        $jugada = new Jugada($_POST['combinacion']);
        $_SESSION['jugadas'][] = serialize($jugada);
        evaluar_fin_juego($jugada);
        $informacion = Jugada::obtener_historico_jugadas();
        break;

}
```

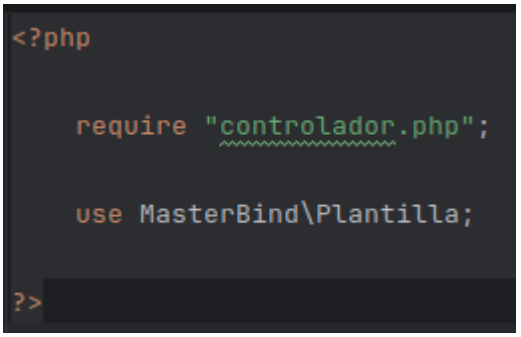
Cada opción se explica por sí sola, en el caso de jugar que es el más complejo, se crea un objeto Jugada con los select de los colores y tras eso se llama al método de evaluar_fin_juego que acabamos de programar.

Ahora ya tenemos toda la parte lógica programada por lo que sólo falta enlazarla con los archivos principales para el desarrollo del juego: jugar.php y finJuego.php

jugar.php

En este archivo vamos a dejar el HTML que tiene que ver con todo lo que es el juego en sí, que hemos generado en Plantilla, los toString de jugada…

El único código de PHP va a ser la llamada al controlador para que inicie la clave, la sesión y demás cuestiones que hemos visto en controlador.php:



Y el código HTML vamos a diferenciar entre los botones de control:

El formulario que hemos hecho en Plantilla para seleccionar los colores y nuestra jugada:

```
    <fieldset>
        <legend>Selecciona tu Jugada</legend>
        <form action="jugar.php" method="POST">
            <div class="grupo_select">
                <?= Plantilla::genera_formulario_juego()?>
            </div>
            <input type="submit" value="Jugar" name="submit" class="btn-jugar">
        </form>
    </fieldset>
</div>
```

Finalmente tenemos histórico de jugadas que mostrará todas nuestras jugadas con nuestros aciertos parciales y totales:

```
<fieldset class="informacion">
    <h2>HISTÓRICO DE JUGADAS</h2>
    <div class="informacion-contenido">
        <?= $informacion ?? "Sin información que mostrar" ?>
        <?php if (!isset($_SESSION['jugadas'])): ?>
            <p>¡Haz tu primera jugada!</p>
        <?php endif; ?>
    </div>
</fieldset>
```

Todo esto como se ve es pura interfaz, la lógica ya la hemos programado y cada vez que hagamos click en “Jugar” se ejecutarán todas las validaciones para ver si hemos ganado o no.

Por último vamos a ver el archivo de finJuego.php que es muy sencillo y sólo evalúa si hemos ganado o perdido.

finJuego.php

Ahora sí, último archivo, respeto al código lo único que va a verificar es si el parámetro win que le hemos pasado desde la redirección en el controlador.php es true o false, dependiendo de eso mostrará victoria o derrota.

```php
<?php
    require 'vendor/autoload.php';

    use ...

    session_start();

    $jugadas = $_SESSION['jugadas'];
    $win = $_GET['win'] ?? "";
    $intento = sizeof ($_SESSION['jugadas']);
    if ($win)
        $msj="<h1>¡FELICIDADES!<br>Has ganado en la jugada nº: $intento <br>";
    else
        $msj="<h1> HAS AGOTADO TUS JUGADAS!!!!!";
    $html_clave = Clave::get_clave_html();
    $informacion = $msj;
    $informacion .= Jugada::obtener_historico_jugadas();

?>
```

Y la interfaz:

```html
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <link rel="stylesheet" href="./css/estilo.css" type="text/css">
</head>
<body>

    <fieldset class="informacion">
        <h2>INFORMACIÓN</h2>
        <fieldset>
            <legend>Sección de información</legend>
            <?= $informacion ?? ""?>
        </fieldset>
    </fieldset>
    <form action="index.php">
        <input type="submit" value="Volver Al index">
    </form>
</div>
</body>

</body>
</html>
```

Tiene un botón para volver al index.php y empezar a jugar otra vez.

Mejoras

El juego finalmente ya funciona, ahora lo único que necesitamos es incorporar algunas mejoras de diseño que no influyen en la funcionalidad y la implementación del registro y login con bases de datos.

Link del código fuente:
https://github.com/espeletamarcos/DAW2/tree/main/DesarrolloServidor/php/app/mastermind-examen