Trabajo Final de Grado

Desarrollo de sitio Web

2º Desarrollo de Aplicaciones Web

Nicolás Lerible García
IES Campanillas - Curso 19/20

ÍNDICE

Introducción

Presentación del proyecto

Esta es una memoria práctica la cual describe el trabajo realizado en el proyecto de final de grado del ciclo de grado superior de Desarrollo de Aplicaciones Web.

Mi proyecto consiste en el desarrollo de un videojuego tipo “Space Shooter” para el centro de ciencia ***Principia***. Dicho videojuego debe estar alojado en un sitio web que será accesible desde cualquier navegador.

Con respecto a los usuarios, existen tres tipos distintos. Para empezar están los usuarios no registrados, los cuales únicamente podrán acceder a la información general. En cambio, los usuarios registrados tienen una mayor amplitud de funcionalidades dentro de la aplicación. Dentro de este último grupo se encuentran los usuarios y los administradores, quienes tienen funcionalidades distintas acordes a las necesidades de cada uno.

Planteamiento general

Las principales funcionalidades de esta aplicación, a grandes rasgos, son:

- Mostrar el juego interactivo.
- Permitir el registro de usuarios.
- Acceso de los usuarios a una página con sus datos de juego.
- Mostrar un ranking con las puntuaciones de los usuarios.
- Permitir a los administradores gestionar la base de datos.

Análisis

Herramientas y tecnologías utilizadas

Tecnologías

Para el desarrollo del backend y la conexión con la base de datos he decidido utilizar el lenguaje ***PHP*** , en concreto el framework ***Laravel*** (v7.1) con adición de composer y artisan, ya que utiliza la arquitectura de Modelo Vista Controlador (MVC) y por su compatibilidad con las bases de datos ***SQL***.

En lo relativo al desarrollo del videojuego he utilizado el lenguaje ***javaScript*** (ES6) con ayuda de las librerías de ***Phaser*** (v3.2).

Para la programación de los estilos he utilizado el lenguaje ***SASS*** con la herramienta sass-loader para su debido traspaso a ***CSS***

Herramientas

El entorno de desarrollo integrado (IDE) que he utilizado durante el desarrollo de este proyecto ha sido principalmente ***VisualStudioCode***. Para el despliegue en local he utilizado ***XAMPP***, con su respectiva herramienta para el desarrollo de la base de datos, phpMyAdmin.

Funcionalidades implementadas

Clasifico las funcionalidades en función de los distintos tipos de usuarios que forman parte de la aplicación.

Funcionalidades de los usuarios sin registrar

- Tienen acceso al juego.
- Su puntuación se guarda en localStorage
- Pueden ver el ranking global.
- Pueden registrarse como usuario en la base de datos.

- Pueden iniciar sesión.

Funcionalidades de los usuarios registrados

- Tienen acceso al juego.
- Su puntuación se guarda en localStorage.
- Pueden guardar su puntuación en la base de datos.
- Tienen acceso al ranking global.
- Sus puntuaciones en la base de datos se reflejan en el ranking.
- Tienen acceso a una página con sus mejores puntuaciones.
- Pueden cerrar sesión.

Funcionalidades de los administradores

- Tienen acceso a un listado de los usuarios registrados en la base de datos.
- Pueden ver, crear, modificar, y borrar usuarios y su información.
- Tienen acceso a todas las puntuaciones de los usuarios registrados.
- Pueden borrar las puntuaciones de la base de datos.
- Pueden cerrar sesión

Estructura de la base de datos

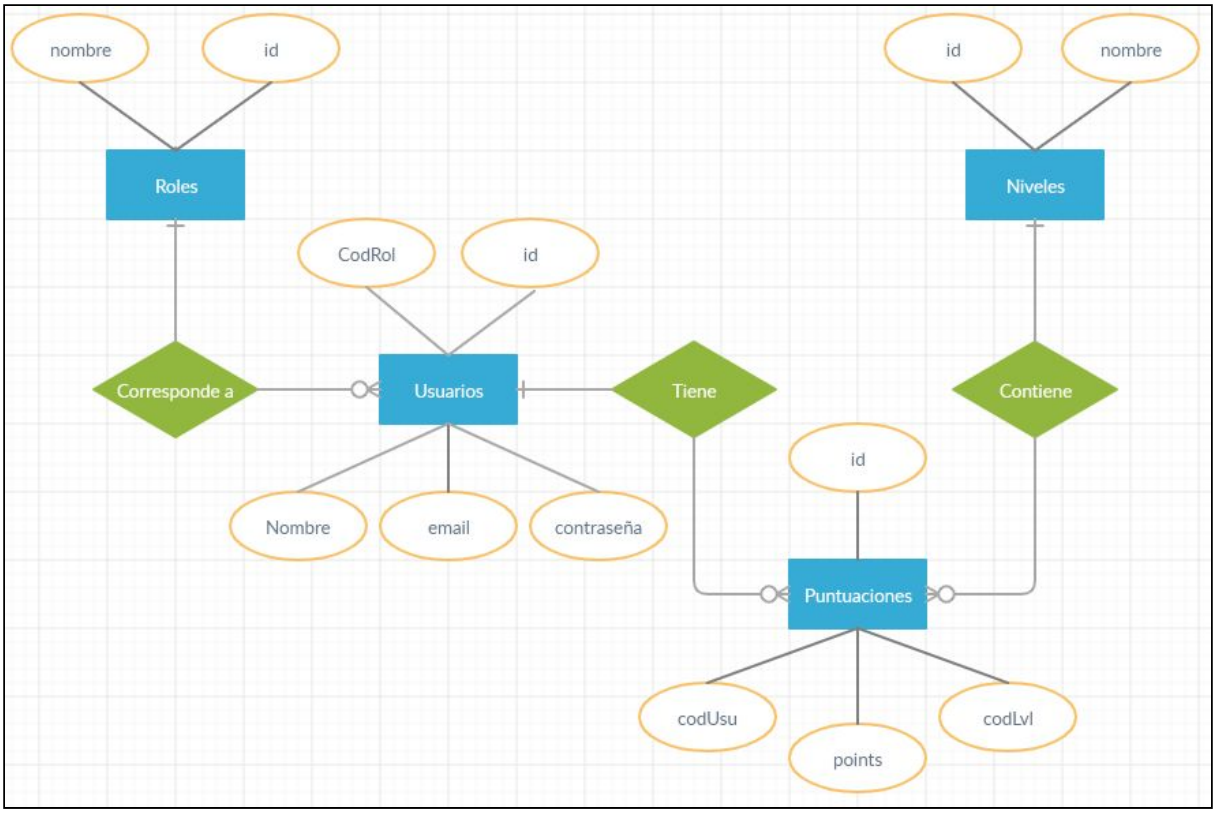
El siguiente diagrama muestra el sistema de relaciones entre los distintos elementos que conforman la aplicación. Las distintas tablas son:

Roles: Representa a los distintos roles que tienen los usuarios.

Usuarios: Representa a los usuarios registrados. Pueden registrarse a través de un login con su nombre de usuario y contraseña.

Niveles: Representa los distintos niveles del videojuego.

Puntuaciones: Representa las puntuaciones de los usuarios en un nivel en concreto.



Sistema de Navegación

Tanto usuarios como administradores tendrán un sistema de navegación acorde a las funciones disponibles para cada rol.

Los ***administradores*** podrán gestionar tanto los usuarios como los niveles y puntuaciones en la base de datos.

Los ***usuarios*** podrán jugar al videojuego, acceder a su página de usuario con información relativa al juego y ver el ranking global con las puntuaciones.

Diseño

Colores

Ya que el color principal del logo de principia es el naranja he escogido este color como el color principal de la aplicación. He decidido escoger una gama bicromatica entre tonos de naranja y azul ya que son colores complementarios.

#4d0000 #fe6625 #fb9334 #7bb5ae #167070 #414a4f

Diseño de navegación

He decidido usar un navegador vertical para la vista desde ***ordenador*** y una navegación horizontal para ***móviles*** ya que aprovecha al máximo la distribución de espacio en distintas interfaces.

Desarrollo

Creación de Roles

Para esta aplicación he definido tres tipos distintos de usuarios:

- Administrador: Puede modificar y borrar tanto usuarios como partidas
- Usuario: Sus estadísticas se almacenan en la base de datos y afectan al ranking global, se pueden ver en su página de usuario.
- General: Usuario sin registrar, puede ver la información sobre el juego y jugar, pero su información no se almacena en la base de datos.

Distribución de Roles

Para esto he decidido crear por medio de migraciones una tabla “roles” la cual contiene la id y nombre de cada rol, así mismo he añadido a la tabla “user” la clave foránea “codRol” para indicar a qué rol está asociado cada usuario.

Administración de permisos de acceso

El paquete laravel/ui que he utilizado para la instalación ya proporciona un middleware para controlar el acceso de usuarios si no están registrados. Por eso mismo voy a utilizar el mismo sistema para las páginas cuyo acceso esté permitido solo a los administradores.

Voy a crear un middleware llamado “`EsAdmin.php`” usando php artisan y registrarlo en “kernel.php” con la clave “checkAdmin”.

AdmUserListController.php - function **_construct**

```php
//Comprueba si el usuario esta conectado
$this->middleware('auth');
//Comprueba si el usuario es administrador
$this->middleware('checkAdmin');
```

En el modelo “`User.php`” añado una nueva función llamada “esAdmin” que comprueba si el usuario es administrador. Esta función la utilizaré en el middleware

"`EsAdmin.php`" para que dependiendo de que lo sea o no le permita permanecer en la página o sea redireccionado a la página de bienvenida.

EsAdmin.php - function **handle**

```php
//Información del usuario registrado
$user = Auth::user();
//Si $user es admin el usuario permanece en la página
if ($user->esAdmin()){
    return $response;
}
//Si no lo es se le redirecciona al inicio
return redirect('/');
```

Lista de Usuarios

El administrador tendrá acceso a una lista con la información de los usuarios registrados, los cuales podrán ser creados, modificados y eliminados. Después de cada operación el administrador recibe un mensaje informando de que la operación se ha realizado con éxito.

Listado de usuarios

Este listado se mostrará en "/admin/users" así que creo la ruta para que esta dirección sea controlada por el controlador "`AdmUserListController.php`"

Una vez hecho esto, puedo pedirle al controlador que cuando acceda a esta dirección, la vista reciba un array con los datos de todos los usuarios.

AdmUsersListController.php - function **index**

```php
//Array de usuarios registrados
$usuarios =User::all();

return view('admin.users.index', compact('usuarios'));
```

Creación de usuarios

El archivo "`create.blade.php`" con el formulario de creación de usuarios se encuentra en "/admin/users/create".

Este formulario utiliza el método ***POST*** y su acción me lleva a la función "store" del controlador "`AdmUsersListController.php`" donde añado el registro a la base de datos.
Pero antes, para encriptar la contraseña de los usuarios, guardo la petición en una variable ($info) para poder modificar la contraseña por medio de un bcrypt antes de enviarla a la base de datos:

AdmUsersListController.php - function **store**

```php
//Recoge la infomación
$info = $request->all();
//Encripta la contraseña
$info['password']=bcrypt($request->password);
//Crea el usuario
User::create($info);

return redirect('/admin/users');
```

Edición de usuarios

El archivo “edit.blade.php” con el formulario de edición de usuarios se encuentra en “/admin/users/**X**/edit”, siendo “**X**” el id del usuario cuya información queremos editar.

Terminada la creación de usuarios, para poder modificarlos he añadido un botón al final de cada una de las entradas de la tabla. Este botón me redireccionará a “edit.blade.php” por medio del comando `route('users.edit', $usuario->id)`, donde tendré el formulario de edición del usuario específico al que he seleccionado.

Entonces modifico la función edit dentro de “`AdmUsersListController.php`” para que busuque y envíe la información del usuario antes de redirigirme a edit.blade.php.

AdmUsersListController.php - function **edit**

```
//Busca al usuario cuya id corresponda con $id
$user=User::findOrFail($id);

return view('admin.users.edit', compact('user'));
```

Una vez hecho esto,he decidido eliminar el parámetro de contraseña ya que se trata de información crítica que sólo debería saber el usuario.

Además, como mi intención es modificar los datos del usuario, debo utilizar el método ***PATCH*** en el formulario y cambiar su acción de forma que al pulsar el botón “Modificar” del formulario se ejecute la función update de “`AdmUsersListController.php`”.

Ahora lo que quiero es que, después de que se actualice la información del usuario, se me redireccione a la lista de usuarios para comprobar que todo ha funcionado correctamente.

*AdmUsersListController.php - function **update***

```
//Recoge la infomación actualizada
$info = $request->all();
//Busca al usuario cuya id corresponda con $id
$user=User::findOrFail($id);
//Actualiza al usuario con la nueva información
$user->update($info);

return redirect('/admin/users');
```

Borrado de usuarios

Para la eliminación de los usuarios voy a añadir un formulario con el botón de borrado junto al de edición en la tabla de usuarios. Este formulario utiliza el método ***DELETE*** y su acción me lleva a la función "destroy" de "`AdmUsersListController.php`".

*AdmUsersListController.php - function **destroy***

```
//Busca al usuario cuya id corresponda con $id
$user=User::findOrFail($id);
//Elimino el registro de la base de datos
$user->delete();
```

Feedback

Por último, quiero recibir un mensaje que me informe de que las operaciones que he realizado se hayan completado con éxito. Yo he utilizado el método "flash" por medio de sesiones ya que esta información sólo me servirá momentáneamente.

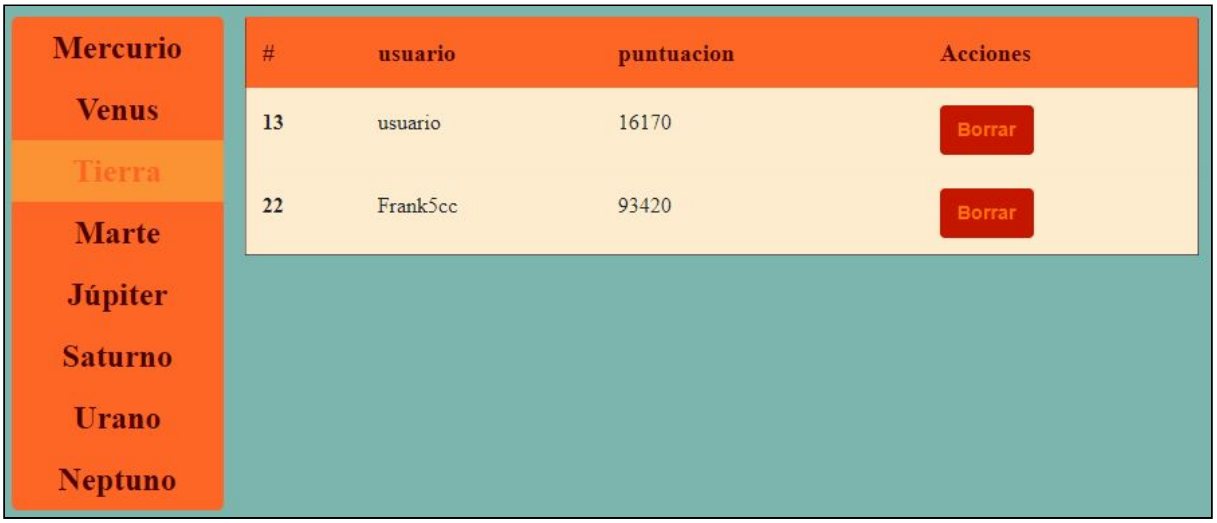
Utilizo el parámetro general "`feedbackUsu`" para que maneje los mensajes que quiera enviar con respecto a los usuarios, ya sea por creación edición o eliminado.

*AdmUsersListController.php - function **destroy***

```
//Envia un feedback al administrador
Session::flash ('feedbackUsu', 'Usuario eliminado con exito');
```

Lista de Puntuaciones

El administrador tendrá acceso a una lista con las puntuaciones de todos los usuarios registrados en para el nivel seleccionado. Además, los administradores podrán eliminar registros pero no podrán ni crearlos ni editarlos.



Listado de puntuaciones

Este listado se mostrará en "/admin/levels/[codigo del nivel]" por lo que creo una ruta de forma de que se acceda a ella por medio de una petición ***GET*** esta dirección sea controlada por la función "index" del controlador "`AdmScoreController.php`". No utilizo resource ya que no voy a necesitar muchas de las rutas que se generarían de esta forma.

Ahora, configuro la función index del controlador para que cuando acceda a la vista, esta reciba un array con todas las puntuaciones, así como información sobre el nivel seleccionado en ese momento.

*AdmScoreController.php - function **index***

```
//Array de puntuaciones cuyo codigo de nivel sea $id
$scores = Score::where('codLvl', $id)->get();
//Nivel seleccionado
$currentLvl = $id;
//Informacion de los niveles
$levels = Level::all();

return view('admin.levels.score',  compact(['scores','currentLvl', 'levels']));
```

Borrado de puntuaciones

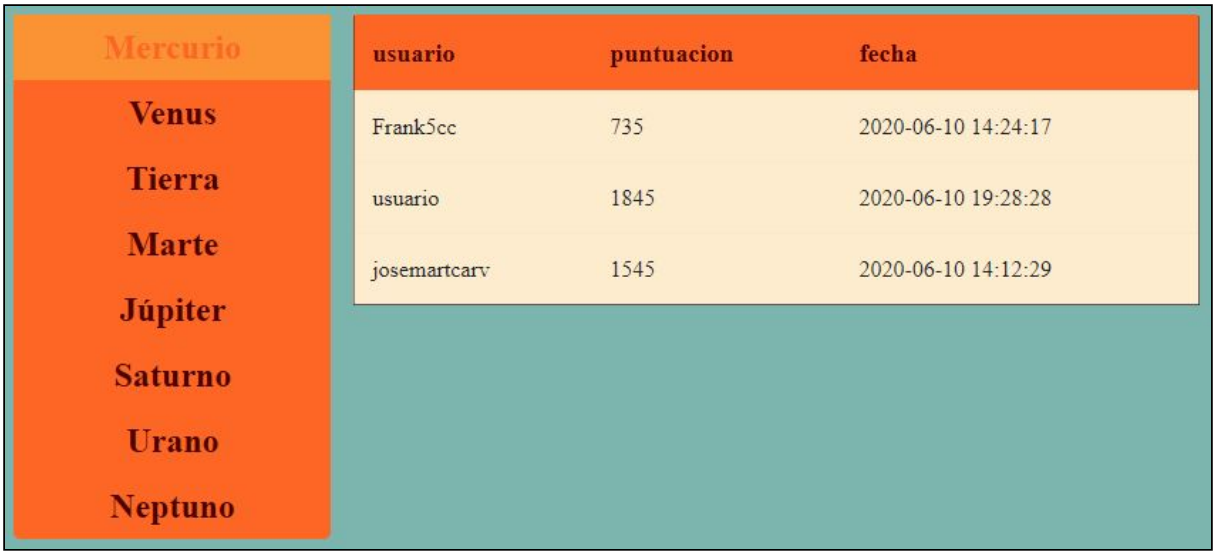
Para la eliminación de los niveles voy a añadir un formulario con el botón de borrado en la tabla de puntuaciones. Este formulario utiliza el método ***DELETE*** y su acción me lleva a la función “destroy” de “`AdmScoreController.php`”.

*AdmScoreController.php - function **destroy***

```
//Puntuacion cuya id coincida con $id
$score=Score::findOrFail($id);
//Borra el registro de la base de datos
$score->delete();
```

Ranking de niveles

Todos los usuarios tendrán acceso a una Ranking con las puntuaciones de todos los usuarios registrados para el nivel seleccionado ordenados desde la puntuacion mas alta a la más baja. Con la diferencia con el listado de puntuaciones de los administradores de que no tendra opcion de modificar ningun registro.



Ranking de puntuaciones

Este listado se mostrará en "/ranking/[codigo del nivel]" por lo que debo crear una ruta de forma de que al acceder a ella por medio de una petición ***GET***, esta dirección sea controlada por la función "ranking" del controlador "`UsrGameController.php`".

Ahora, configuro la función index del controlador para que cuando acceda a la vista, esta reciba un array con todas las puntuaciones, así como información sobre el nivel seleccionado en ese momento de la misma forma que se hizo en el listado de puntuaciones de la administración.

UsrGameController.php - function **ranking**

```
$scores = Score::where('codLvl', $id)->orderBy('points', 'desc')->get();
$currentLvl = $id;
$levels = Level::all();

return view('user.game.ranking',  compact(['scores','currentLvl', 'levels']));
```

Información del usuario

La página principal del usuario muestra una lista con la puntuación máxima del usuario en los distintos niveles del juego. Además, haciendo click en los links de

cada nivel se redirecciona al ranking correspondiente para comprobar tu posición global.

#	usuario	Nivel	puntuacion
10	usuario	Mercurio	1845
9	usuario	Venus	3780
13	usuario	Tierra	16170
14	usuario	Marte	405
12	usuario	Júpiter	1245
11	usuario	Saturno	2100
28	usuario	Urano	2430

Listado de niveles

Este listado se mostrará en "/user/home" así que creo una ruta la cual estará controlada por el controlador "`UsrUserController.php`"

Una vez hecho esto, el controlador automáticamente hará que cuando acceda a esta dirección, la vista reciba un array con todas las puntuaciones máximas del usuario que se encuentre conectado.

UsrUserController.php - function **index**

```
$user = Auth::user()->getId();
$info=User::findOrFail($user);
$scores=Score::where("codUsu", $user)->orderBy('codLvl')->get();
```

Videogame

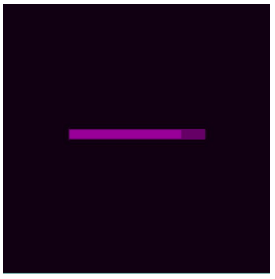
Main Configuration

The main configuration of the videogame is hold in the “game.js” file. In which is specified the width and height of the game window (512px x 512px), the differents scenes that take part in game such as the GameOver scene; and some other general parameters like the player’s speed.

This file makes the fundamentals of how the game will work so it must be referenced the last in order to put everything together.

Scenes

Preload



This is the first scene of the videogame, in here every asset, sound and animation that the game needs is created. So it has to be pre-loaded before any other file as most of the scenes require the use of those assets.

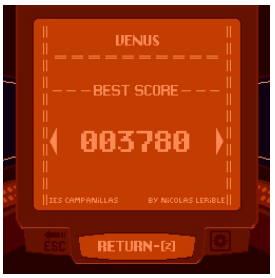
GameMenu

Once everything is loaded, the menu scene is shown to make the player know that now the game is ready to be player it only has the logo and a button that starts the game.

ChooseLevel

This is the principal scene of the game. Here you can choose the game you want to play and press a button to play it. Also you are able to move to the scene with your best scores in every level and to return to the Main scene.

OptionsMenu



Here you can see your scores on every level. If you haven't completed a certain level your score will be shown as "000000" until you complete it. Once seen that you can return to the ChoosLevel scene by simply pressing "Z".

Level

In the level scene, the player can control a spaceship and shoot to the differents enemies. This scene calls a class depending on the chosen level that holds how the enemies are going to appear in the game screen.

Victory

Here the score you reached during the game is shown. This score is saved through local storage only if your score is higher than your best score, but in order to save the score at the database you must press the "Save changes button".

GameOver

Here the score you reached during the game is also shown. But this score is not saved through local storage in any case.

Mechanics

This scenes are a little bit different from the others because they are runned over the rest at the same time. This scenes manage information that are important in between scenes.

HUD

This scene manages the scores and lives of the player in every level

GameController

This scene mainly manages the audio and the music in game

Classes

Player

It is the principal class of the game and represents the spaceship the player will control once playing the game. This class manages it owns score, how many lives the player has, the movement and shoots of the spaceship, and the “power”.

The “power” in the player class means that depending on your level, the spaceship will shoot in 3 differents ways. Shoots are managed in a different class that receives the movement direction.



Enemies

Every enemy has it own class which manages different movement patterns, spritesheets and other specific parameters. All this classes extends a main class called Enemy with the general parameters and functions of every enemy in game such as the speed, management of lives, etc.

Planets

Depending on the level the player chossed, the level scene will call one of this classes. Inside this class is specified how enemies will appear in screen during the

game. Here, the enemies classes are called to appear in a certain moment so every level has a different gameplay.

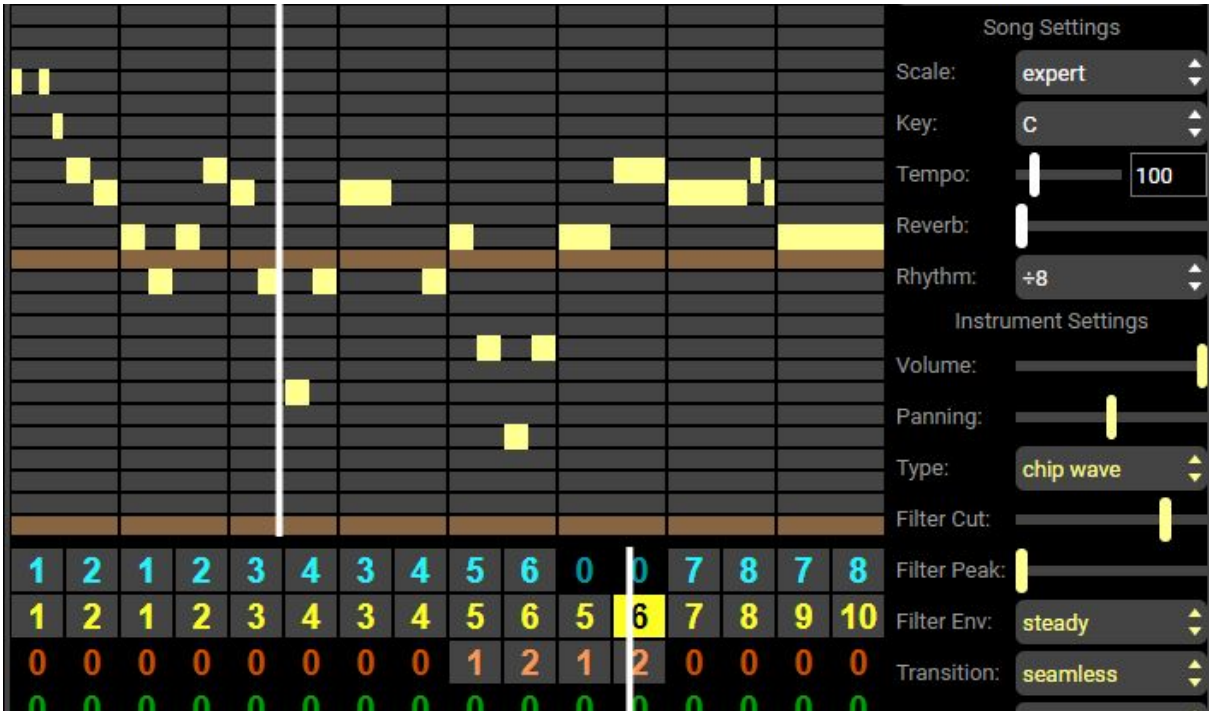
*Mercury.js - function **horde***

```
//Part 1
this.scene.time.addEvent({ delay: 3000, callback: () => { this.makeU(); }, callbackScop
this.scene.time.addEvent({ delay: 6000, callback: () => { this.makeSplit(); }, callback
//Part 2
this.scene.time.addEvent({ delay: 25000, callback: () => { this.makeU(); }, callbackSc
this.scene.time.addEvent({ delay: 30000, callback: () => { this.makeSplit(); }, callbac
this.scene.time.addEvent({ delay: 30000, callback: () => { this.makeTyphon(); }, callba
this.scene.time.addEvent({ delay: 35000, callback: () => { this.makeU(); }, callbackSc
//Mini Boss
this.scene.time.addEvent({ delay: 54000, callback: () => { this.makeMiniBoss(); }, cal
```

Assets and audios

music

For the development of the music all songs have been created using the online tool BeepBox. This tool offers a simple interface with a very complete variety of functions that simplifies the creation of songs for any style.



Sounds

On the other hand, for the creation of shorter sounds I used another online tool called Bfxr. This tool Generates different sounds for anyone to use them and modify with any problem.

Sprite Sheets

For the creation of the spritesheets I used the online tool Piskel, with this tool you can create pixel arts of almost any dimensions. Also, it brings lots of facilities to help the user in the creation of animated sprites and offers a wide variety of ways to save,export and import files depending on your necessities.

Despliegue

Para el despliegue de la aplicación he necesitado hacerme con un hosting online en el que poder alojar tanto mi página web como mi base de datos. En mi caso he decidido utilizar “hostinger” con las siguientes características:

- Número de sitios ilimitado
- Cuentas de correo ilimitadas
- Ancho de banda ilimitado
- 4X Recursos asignados
- Certificado SSL gratis
- Caché LiteSpeed
- Servidores de nombres protegidos con Cloudflare
- Integración con Github
- Gestor de DNS
- Administrador de acceso
- Bases de datos MySQL ilimitadas
- 100 Subdominios
- Cuentas FTP Ilimitadas
- Dominio Gratis
- Acceso SSH

Entre otras cosas he elegido principalmente este host por proporcionarme un dominio gratuito así como un certificado ***SSL*** y por el acceso a bases de datos ***MySQL*** ilimitadas.

Bibliografía

- **Docuemtación PHP:** https://www.php.net/
- **Documentación Laravel:** https://laravel.com/docs/7.x
- **Documentacion Phaser:** https://phaser.io/phaser3
- **Información general:** https://www.wikipedia.org/
- **Manuales Desarrollo web:** https://desarrolloweb.com/manuales/