IDA DE OLLA V2

# Aplicando las sugerencias

En el trimestre pasado nuestro equipo de testers aportó una serie de sugerencias que comenzaremos a incluir las que se puedan adaptar rápidamente.

## Mensaje de guardado de datos:

“Este es un problema mucho menor que tengo con el aviso inicial de guardado de puntuación con treedude, y es solamente un detalle con el formato del texto de la ventana emergente que lo haría cambiar tal que así”

Añadiendo una línea en blanco adicional entre párrafos, además de puntos al final, y cambiando el “permites” por un “permite” (al estar tratando al usuario de “usted”. Todo esto con el fin de mejorar la legibilidad.

Se realizan los cambios sugeridos por el tester

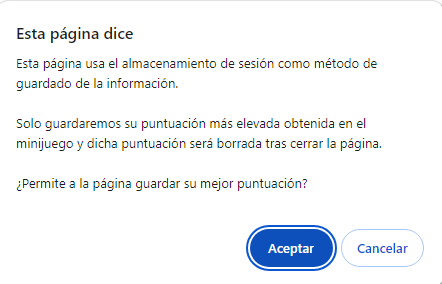


Ilustración 1: Nuevo mensaje

## Efectos de sonido adicionales:

“Para hacer la navegación del proyecto más amena no estaría mal incorporar efectos de sonido más sutiles al navegar por el menú del proyecto”

Nos toca volver a revisar el recurso original ya que efectivamente cuenta con un audio al navegar entre las opciones, la idea será al igual que con el golpe replicarlo de la manera más similar posible

Lo que trataremos de replicar **NO** es el sonido de fondo, si no el sonido que es reproducido al cambiar de opción

### Filtro

Lo primero que haremos será buscar el filtro de sonido más similar o el posible que hayan utilizado, para ello reproduciremos los sonidos por defecto de todos los filtros. Temporalmente usamos el action.js de settings para pruebas. Preparamos el terreno adjuntando en el index.html de settings sounds.js y sounds.js creamos la función: optCh() que reproducirá el sonido y será llamada desde el action.

Para ahorrar en líneas de código importaremos la clase de Sound.js en sounds.js, esto lo realizaremos de la siguiente manera.

#### Importación y exportación…. Hardcodeado

En Sound.js se introducirá “export” delante de class y en sounds.js será añadido al principio del script el siguiente bloque de código:

import {

<elemento importado: en este caso, la clase>  
 …

} from ‘<ruta del js que importa>’

Para comprobar su correcto funcionamiento volveremos al talador, y quitaremos la referencia del index.html hacia Sound.js y tendrá que continuar funcionando.

Al ser ejecutado saltarán entre otros el siguiente error:



Ilustración 2: ERROR MÓDULO

Para solucionarlo en los index.html que hagan referencia a sounds.js habrá que agregarles el atributo type=”module”. Al realizarlo podemos observar un error diferente:

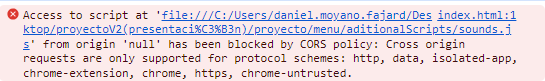
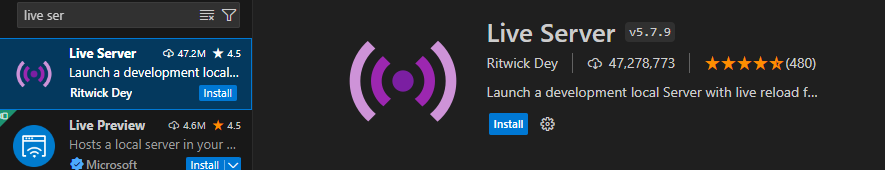


Ilustración 3: Error de servidor

Esto ocurre por las directivas de seguridad del navegador, las cuales impiden el acceso de archivos locales a las páginas. Para poder solventar el problema será necesario instalar la siguiente extensión: 

Tras múltiples intentos de diferentes maneras que acabaron en fracaso, se procede a realizar cierta trampa mediante el uso de jquery, se crea el script adicional de soundInitializer.js, el cual adjuntará al body los scripts de la clase de Sound y sounds en el orden correcto para poder evitar repetir el código, actualmente se encuentra hardcodeado pero se trabajará para que se haga mediante la obtención de rutas relativas de los archivos.

#### DIRECCIONAMIENTO RELATIVO

Para crear la función y que funcione independiente de la localización del HTML donde sea anexado, será necesario diferenciar el nivel donde se encuentra y en función de ello retroceder o avanzar.

Para obtener la ubicación actual se usará la línea:

window.location.pathname+window.location.search

Buscando en internet, descubrimos que en Node.js existe el método path.relative() para sacar rutas relativas de elementos, no obstante trataremos de evitar usar frameworks.

Usando solo javascript encontramos una página que nos indica cómo saber si la carpeta indicada existe donde es marcada, pero dicha manera ya no es soportada en navegadores actuales por lo que tendremos que realizarlo de otra manera.

Calcularemos la ruta en la que se encuentra el html mediante contar las barras “/” de la URL, a ese número se le restará 2 ya que es en el apartado de menú y no en el de la raíz donde se encuentra el directorio de aditionalScripts, en caso de que el resultado sea -1, eso implicará que se encuentra en el html de la raíz (la portada) por lo que para ese caso en concreto prepararemos la variable que llevará la ruta negativa para acceder a /menú/.

De lo contrario solo quedará una opción: el resultado de contar las barras y restarle 2 es igual o mayor que 0. Es ahora cuando entraríamos en un bucle que se repetiría menguando el valor de la variable que llevaba el resultado de la operación de las barras – 2 y cada vez que menguara su valor agregaríamos un: “../” subiendo un directorio en el src, estas acciones serían repetidas hasta que el valor de la variable sea 0.

Finalmente, independientemente del caso que se diera le agregaremos a la variable que llevaría la dirección relativa de los scripts el directorio “aditionalScripts/”. Y por último ejecutaríamos las líneas para adjuntar al body los scripts de la clase Sonido y el script sound:

$("body").append("<script src = “+ <variable de la ruta> +”class/Sound.js defer></script>");

$("body").append("<script src = “+ <variable de la ruta> +”sounds.js defer></script>");

### Ampliando la clase

Ahora nos encontramos en la tesitura en que no queremos que todos los sonidos tengan el mismo volumen, como es en este caso donde queremos que los sonidos del menú sean más bajo. Es por ello que necesitamos añadir la posibilidad de invocar sonidos con diferentes volúmenes. El método sonar actualmente tiene los parámetros f: filtro, n: valor/ frecuencia, c: canal. Creamos el método de setVolumen en la clase Sonido.js, al cual se le pasará el parámetro numérico vol: donde se juzgará en un try catch. Si el valor es válido se asignará dicho valor al volumen, de lo contrario se usará el valor por defecto: 1.

# 2º Round de Galletas

Al ahora realizar la ejecución desde un servidor probamos a realizar una cookie de prueba para ver si ahora sí son creadas, el resultado es positivo.

Por lo que se procede a comentar la parte de comparePoints y sustituirla por su equivalente usando cookies, las galletas al funcionar de manera diferente es obligatorio crear una función específica para la eliminación de las cookies guardadas.

Se comienza con la función de checkCookie, a la cual se la pasará como parámetro el nombre de la cookie que queremos consultar, sin embargo, en caso de que no se le pase un valor, por defecto tendrá en cuenta el valor “puntuacion", valor usado en checkSession.

Dentro de la función será donde construiremos la cookie a buscar, se creará la variable nombre la cual llevará el valor del nombre de la cookie concatenado con “=”, por otro lado, tendremos migas que cogerá la cookie del documento y la dividirá tomando como referencia el carácter “;”.

Es en este punto donde se accede en un bucle que recorrerá el contenido guardado en migas, en cada iteración tendremos la variable caracter que pillará el valor de la posición de i relativa a migas. Mientras que el primer carácter de dentro de la variable caracter sea “ “, se cogerá la subcadena de la variable caracter, comenzando desde el siguiente carácter hasta el final.

Si la posición de la variable nombre coincide con la 1ª posición de la cadena de la variable ya preparada de caracter se devolverá la subcadena resultante de coger los caracteres desde la longitud de la variable de nombre y la longitud de carácter.

De lo contrario si i recorre toda la longitud de migas y no se cumple la condición de anteriormente nombrada la función devolverá null, indicando que no se encontró la cookie.

Una vez que ya tenemos checkCookie() la incorporamos al código: en un if comprobaremos si el valor devuelto por checkCookie es nulo/falso, para ejecutarlo y precisar menos líneas de código invertiremos el valor obtenido y así si devuelve un valor no realizará nada, pero si devuelve nulo se invertirá y se realizarán las acciones pertinentes.

Proseguimos avanzando adapatando la función usada para borrar las cookies como valor por defecto use el de puntuación

Como siguiente paso recordamos que la función de checkSeession era llamada desde el action.js y desde la función ya se preparaba todo. En vez de borrarlo todo se opta a adaptar la función de checkSession para usar checkCookie y su resultado devuelto para la evaluación de la condición. Como siguiente paso se altera el mensaje del alert para reflejar el método de guardado de datos actual.

Tras esto se modifica la función de doCookie para que como parámetros por defecto tenga: valorCookie/puntuación: 0, nombreCookie: puntuacion y Dias: 7.

Por último, toca modificar la función de comparePoints para que use checkCookie y doCookie, recordemos que la función de doCookie sirve tanto para crear una nueva galleta como para rescribir una ya existente si pasamos como nombre de la cookie el nombre de la cookie ya creada.

Una vez terminado de adaptar el código es hora de probarlo

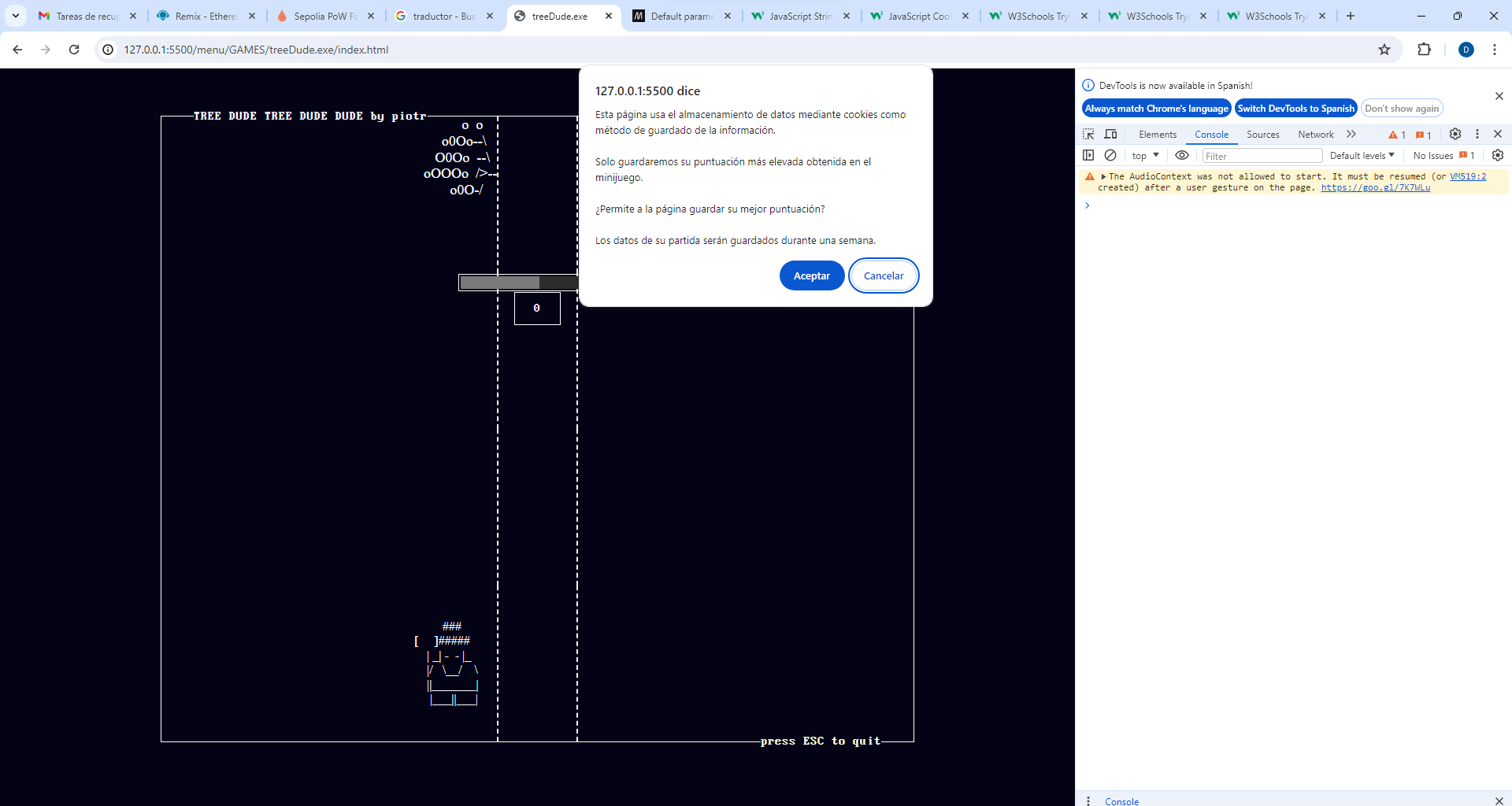


Ilustración 4: Prueba 1 no guardamos datos

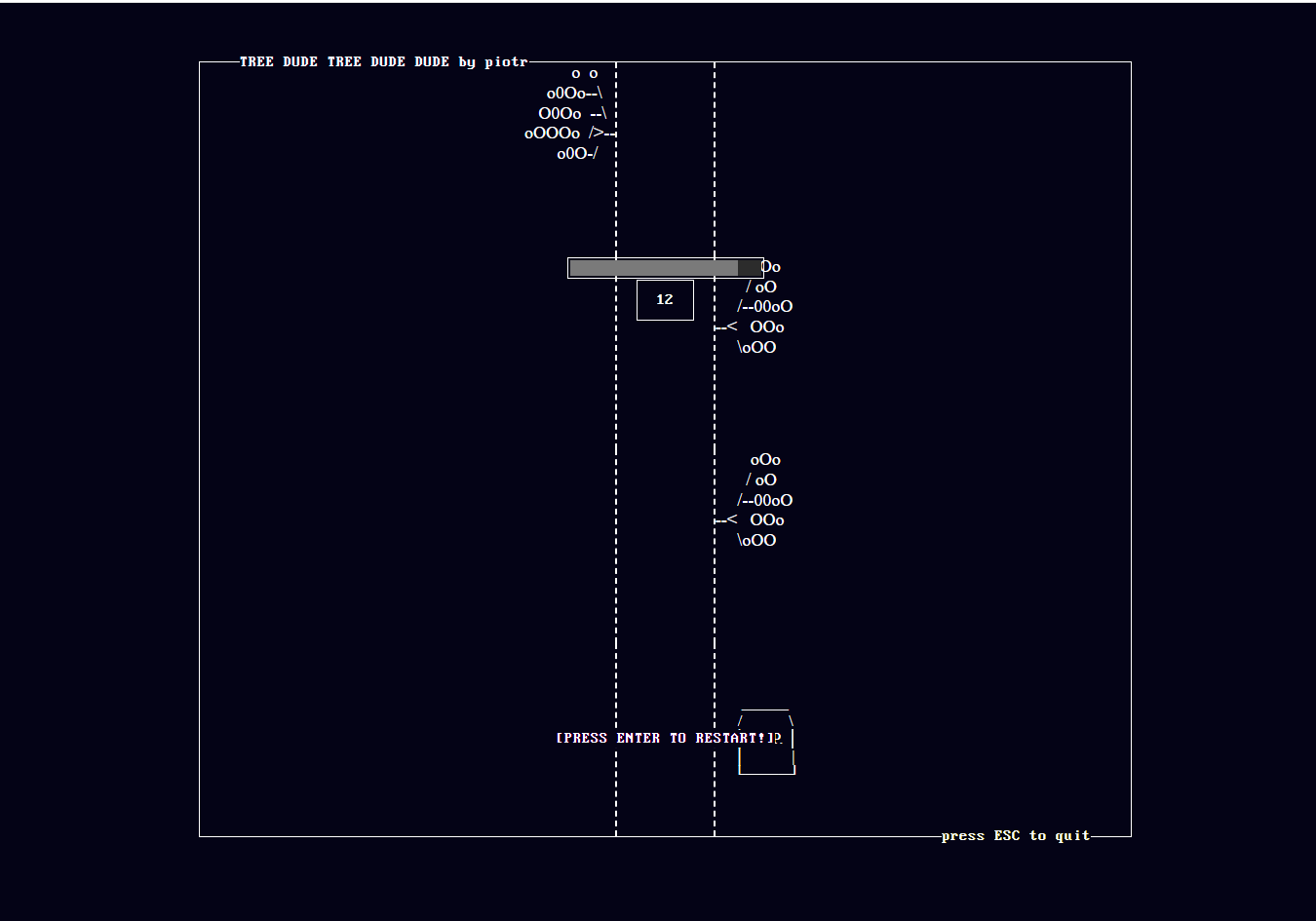


Ilustración 5: Sacamos 12 puntos



Como podemos observar no se guardó la puntuación



Ilustración 6: Guardamos puntuación

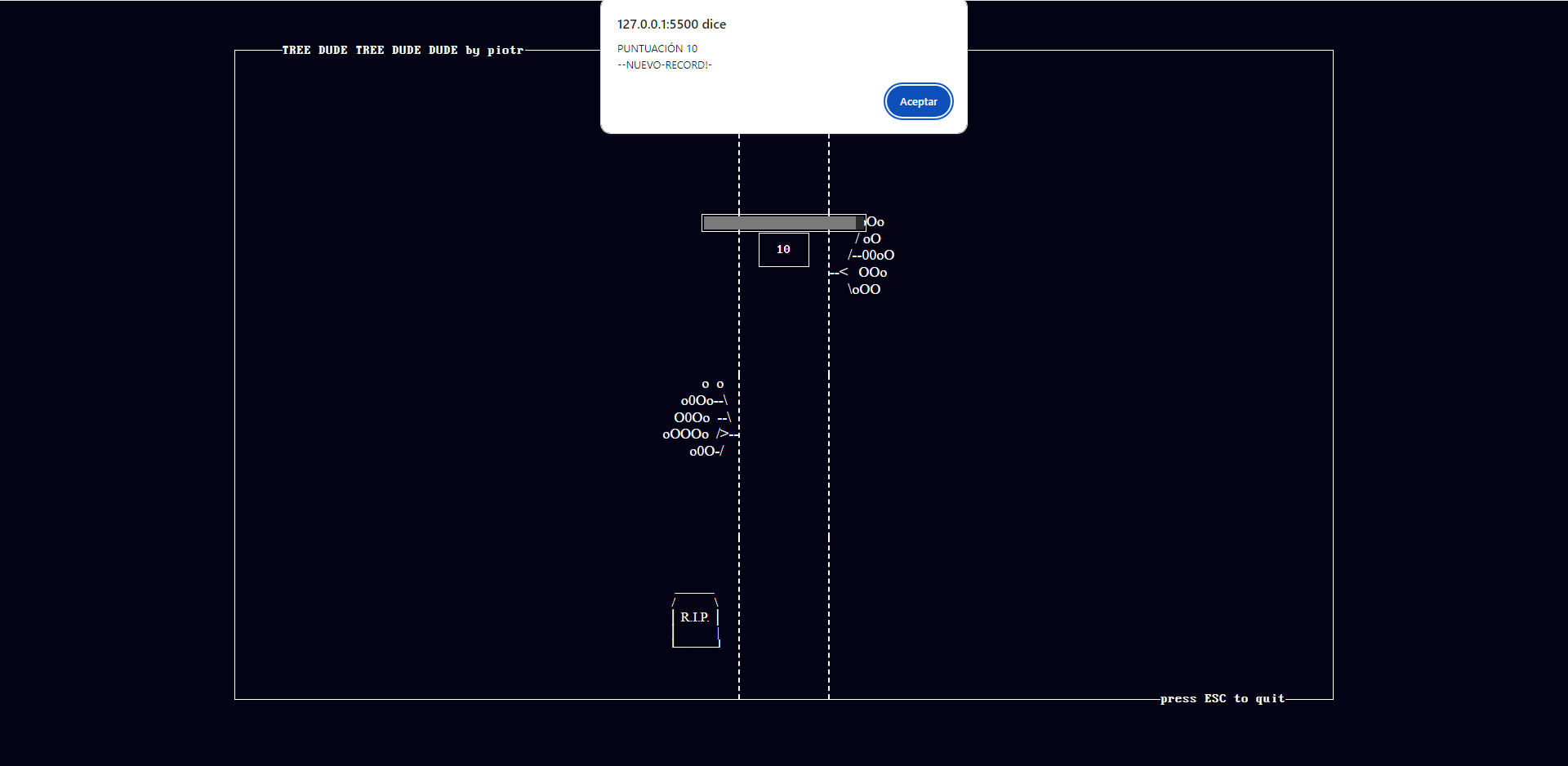


Ilustración 7: Puntuación actual + record

# Ideando

Cogiendo de los ejercicios de clase cogemos el uso de una promesa: (Tema 3)

La cual consistía en tirar 6 veces un dado imaginario y si salía un 6 daba error. Comprobamos su correcta funcionalidad, pero además testeamos si ejecuta código simultáneamente, al poner justo de debajo un log. El resultado resulta ser el esperado siendo un sí.

# Caminos

Debido a la falta de tiempo se debe decidir rápido por que camino tirar, por un lado, tenemos la posibilidad de complicar más el juego e ir aumentando la velocidad en la que la vida va disminuyendo según se va aumentando el nivel, así como alterar las posibilidades de creación de ramas o mientras se juega simultáneamente reproducir música

## Música

Para comenzar veremos como se escucha las “acciones” del talador al mismo tiempo que reproducimos un sonido que luego irá evolucionando a la música

Primero cogemos el código que ya teníamos de ejecución de código simultáneo en el action.js de settings y por el momento lo movemos a songs.js por el momento. Luego en el html de treedude se hará referencia al songs.js.

Tras comprobar la ejecución correcta del código antiguo ahora queremos que se repita indefinidamente, esto se consigue encapsulando el código en una función que será llamada en un intervalo definido en action.js, como siguiente paso volveremos a comentar el código simultáneo y se adaptará el “esqueleto” a nuestro gusto, en un principio no queremos que se repita varias veces por lo que la parte de iteraciones la comentamos, comentaremos la constante que llevaba los números: “const numbers = []” y en su lugar crearemos la constante timbre = new Sonido. Como no queremos repetir el código varias veces comentamos la parte del bucle y el if también se comenta y lo sustituimos por un try catch.

Es en este momento donde aprovechamos para retocar la clase Sound, en concreto el método sonar para que todos los valores tengan un valor por defecto y así facilitar la ejecución de sonidos de prueba no teniendo que poner ningún parámetro. Es así como por defecto ponemos el filtro a sine, el valor de la frecuencia 440.0, el canal 0 (ambos canales) y el volumen 1.

Volviendo a songs si salta un error en el catch realizaremos un reject donde el error será guardado en fail y igualaremos error a true y “message” a fail, si todo se ejecuta correctamente llegaríamos a la parte del resolve donde igualaríamos error a false y como no tenemos ningún valor que queramos saber su valor se comentará la parte de value. Por último comentamos la parte interior de la invocación a doTask (esto debido al no repetir la invocación ninguna vez), también se comentará la parte del then para no llenar la consola de mensajes pero la parte del catch la dejaremos tal cual ya que si ocurre algún error queremos saber que es lo que ocurrió, tras esto es la hora de volver a probar la ejecución. Con esto nuestra prueba finaliza con éxito.

Ahora necesitamos generar la melodía, es decir una sucesión de notas por lo que deberemos descomentar la parte de las iteraciones, pero no queremos tocar todo el rato la misma nota, queremos tocar la nota que corresponda. Por lo que tendríamos que buscar la frecuencia de cada nota si no fuera por en Reddit parece que alguien ya nos ha hecho parte del trabajo, cuando podamos acceder a la página revisaremos si alguien pudo resolverla al completo, pero por lo menos ya tenemos un comienzo:

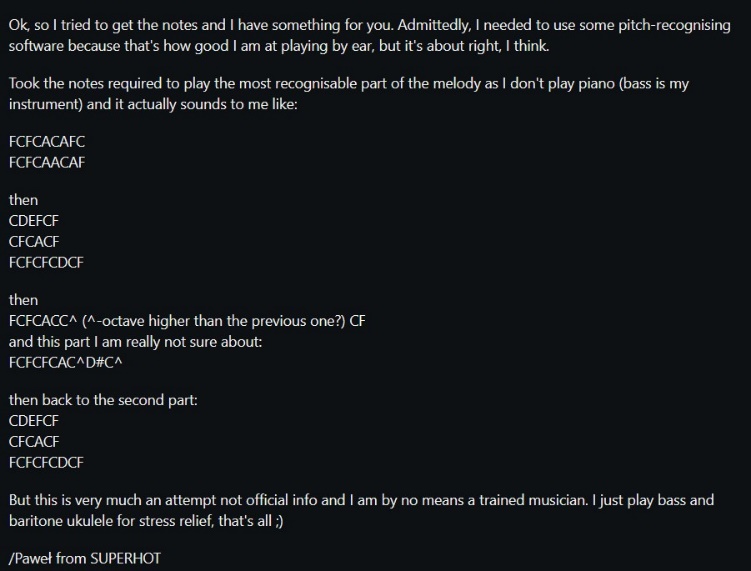


Ilustración 8: Acordes treedude V2

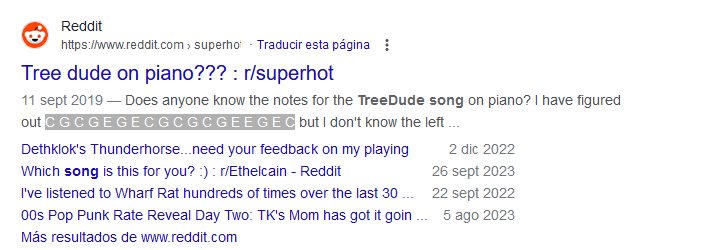
Para aquel que no tenga conocimientos musicales habremos incluido la explicación de lo que es un acorde. Esto más la página que nos ayudó en un principio a la creación de sonidos nos será más que suficiente para sacar las frecuencias necesarias ya que la misma cuenta con una tabla de frecuencias de los distintos acordes según su tono. De momento para la música usaremos el mismo filtro usado que para los efectos de sonido, véase sawtooth. Necesitamos que la reproducción de sonido se repita 18, (aunque con diferentes valores dependiendo de la iteración), es en este momento donde iremos a la clase Sound de nuevo para preparar el método setSound que usaremos para establecer el valor de la frecuencia, de momento ya tenemos un tono… Aceptable, pero la duración de las notas en la canción original es más corta que en la nuestra por lo que crearemos un nuevo método y en la construcción de la clase Sound se añadirá otro parámetro nuevo: t de tiempo o time, por defecto le pondremos el valor que teníamos al principio 0.5. Luego el método será setDuration el cual validará el tiempo pasado como parámetro. Tras esto se adaptará al nuevo tiempo de espera para volver a “ejecutar” la canción. Con esto se crea el intervalo para reproducir la música, cuando el jugador muere, se finaliza el intervalo de la música para inicializar otro con la música que se reproducirá al morir, ahora bien, el problema que puede surgir es que el jugador muera antes que se termine un “bucle” de la canción que se reproduce al estar vivo por ello se deberá chequear el valor de la variable ded antes de reproducir cada nota, si la variable esta en false se reproducirá la nota que toque en dicha iteración, de lo contrario para finalizar la ejecución de la canción se lanzará un error. Que recogeremos en el catch, con esto retocaremos un poco más las diferentes canciones que se reproducirán y se dará por finalizado el apartado.

Ilustración 9: Acordes de treedude V1

## Mute

Ahora que reproducimos sonidos tendremos que dar la opción de poder quitar el sonido al usuario si así lo desea. Primero gracias a Bootstrap obtendremos los iconos de volumen:



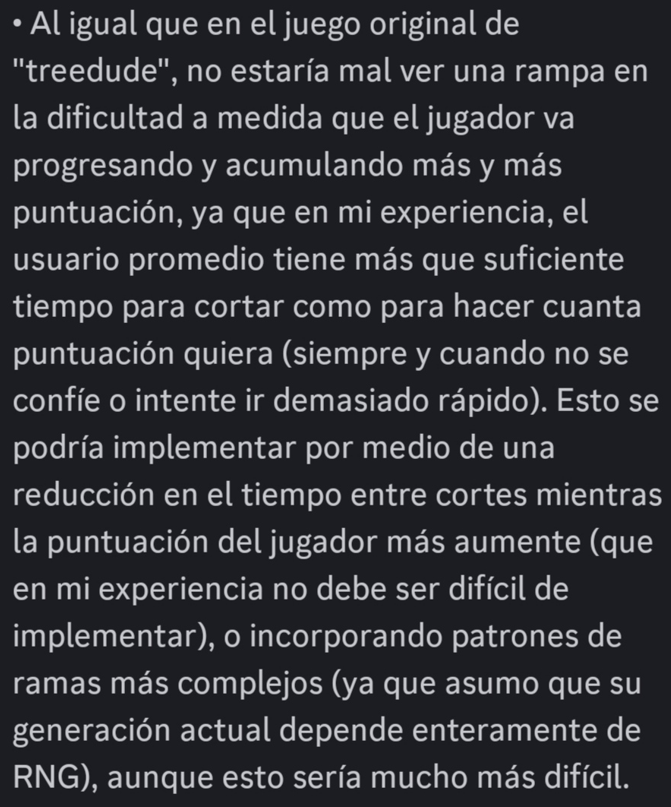
Ilustración 10:Iconos

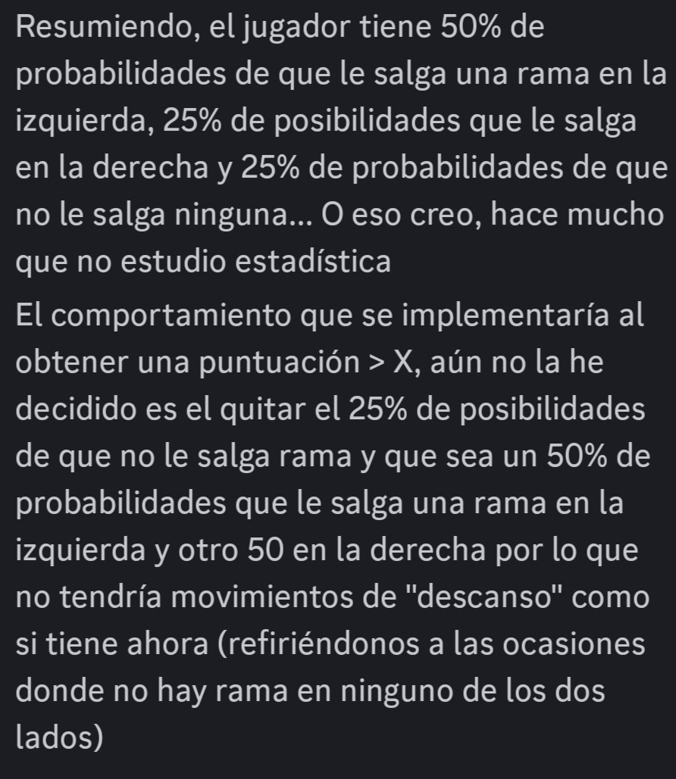
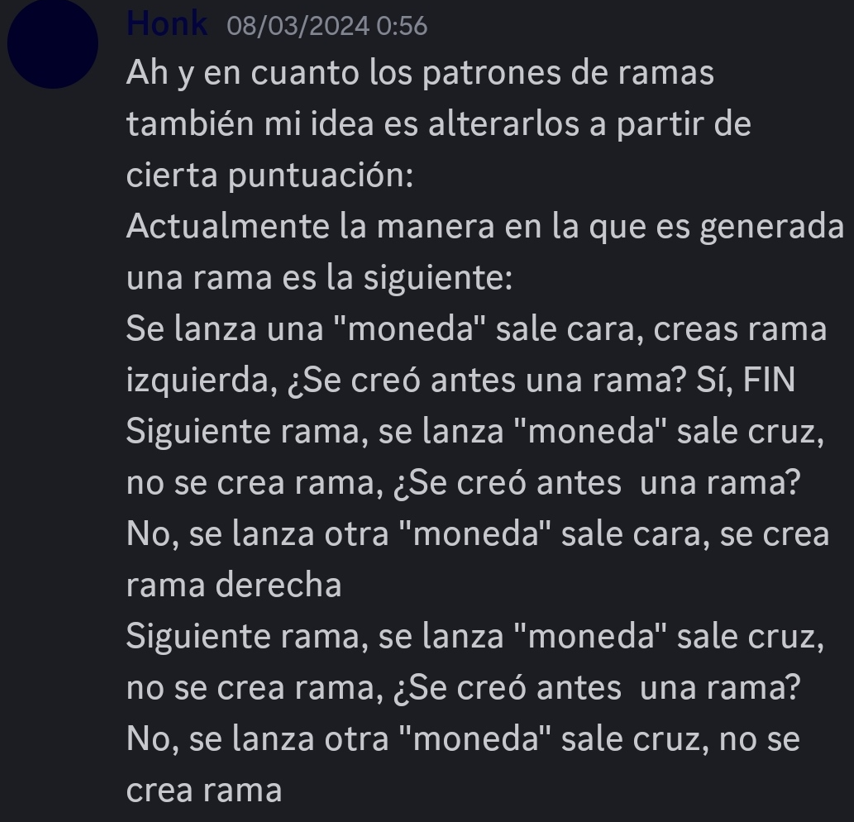
Por el momento comentaremos las líneas del icono que usaremos para cuando el sonido esté desactivado, como siguiente paso les aumentaremos el tamaño a 32 y colocaremos en el lado derecho de la pantalla dejando un margen de 20px, como siguiente paso se registrará cuando se pulse la “M” desde action y crearemos la **var**iable mute con valor false por defecto, una vez que ya registramos la pulsación se creará en treedude.js la función chgVol() donde si el valor de mute es verdadero se cambiará a falso y viceversa. Una vez se comprueba el correcto funcionamiento del cambio del valor, queda sustituir el icono en función del nuevo valor de mute.

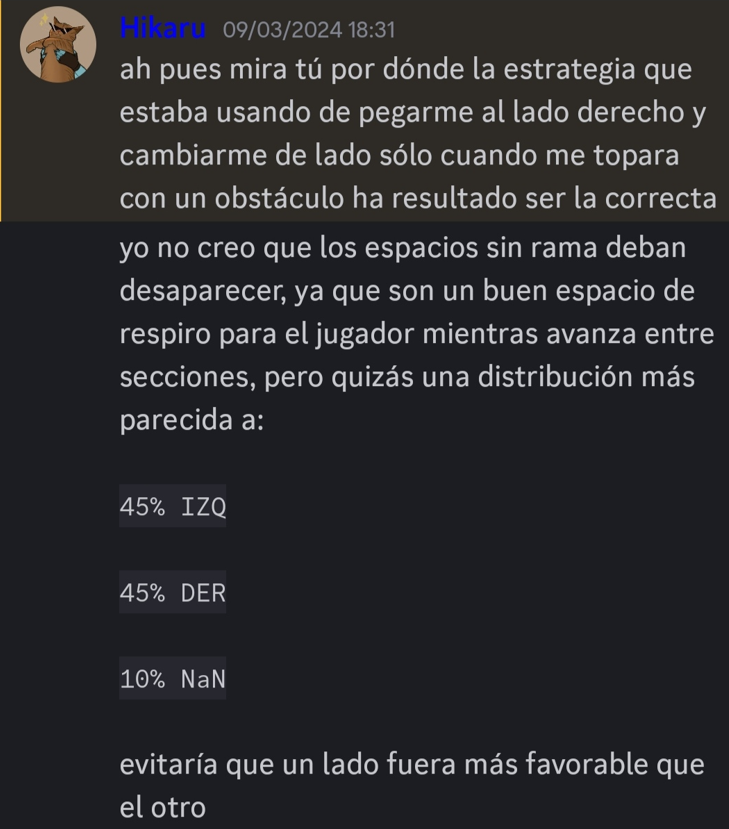
Tras esto en esta versión lo que haremos será continuar con la ejecución de las melodías solo que con el volumen 0. Para ello modificaremos la función de hit para que reciba el valor de mute si es verdadero se reproducirá el golpe con volumen 0, de lo contrario no se pasará el valor del volumen reproduciéndolo por defecto a valor 1.

## Aumentando la dificultad generación de ramas

En parte porque ya lo teníamos planeado y además fue sugerido por uno de los testers pasamos a aumentar la dificultad del juego:







Para crear la nueva generación de ramas se usará como base la generación de ramas original. En el código original generábamos 2 números aleatorios, uno para la izquierda y otro para la derecha, si el valor de la izquierda generaba una rama ya se dejaba el valor de la derecha a 0, por el contrario se generaba otro número aleatorio entre 0 y 1 dando como posibilidades que se generase una rama en la derecha o no, de esta manera como se explicó en las capturas previas la manera de hacer “trampa” en el juego es quedándose en la derecha ya que la posibilidad que te salga una rama en menor a que te salga en la izda.

Para solucionar esto generaremos a partir de ahora un único número aleatorio del 1 al 100, si el número es menor de 45 generaremos una rama en la izquierda, por otro lado, si el número es mayor a 45 y menor a 91 se generará a la derecha, por último, si es mayor de 90, no se generará rama. Ahora que ya tenemos el enunciado toca codificarlo, para empezar se eliminarán las líneas de código que nos sobran como el interruptor SW y el bucle. Ahora se modificará la propiedad de Math.floor(Math.random()…) de \*2 a \*100) +1:

let rama = Math.floor(Math.random() \* 100 + 1);

Y por último toca modificar el condicional a continuación se muestra el código del condicional con cada condición a una línea, en este caso por lo tanto el poner las llaves quedaría como voluntario, no obstante es preferible siempre usar las llaves para macar de manera más clara el fin de los bloques de código:

\*  
if (rama < 45) cadena = [1,0];

else if (rama < 91) cadena = [0,1];

else cadena = [0,0];

Se comprueba la correcta ejecución del código

## Aumentando la dificultad subiendo de nivel

Como actualmente no podemos contar con los medios de revisar el patrón que sigue treedude para subir de nivel vamos a ir avanzando en los aspectos más básicos que sí podemos hacer siendo estos: el gráfico y el sonoro, ya que cuando el jugador sube de nivel salta un indicativo sonoro que siempre es el mismo y un letrero aleatorio en la parte superior de la pantalla ocultando brevemente los puntos:

### GRÁFICOS

Los posibles letreros que le pueden salir al jugador son los siguientes:

* FASTER
* LEVEL UP
* TICK TOCK
* CHOP CHOP
* SPEED UP

Y nosotros adicionalmente añadiremos uno extra: HONK, primero: mediante ASCII trataremos de reproducir manualmente el resultado final de uno de los letreros: como letrero de ejemplo usaremos FASTER.

████ ███████████████████████████  
█████████████████████████████████  
████████████████████████████████  
███████████████████████████████  
█████████████████████████████████

Nota: Los espacios en blanco para no descentrar los caracteres se realizaron mediante el mismo caracter pero con fuente blanca.

El siguiente paso es conseguir que se vea igual en el navegador, primero lo haremos con fondo blanco y luego en negro para esto se volverá a usar de forma temporal SETTINGS, para esto en draw crearemos una nueva función: para la primera prueba craremos invokeSquare que creará un array bidimensional 5\*5 poniendo todos los valores a true, después crearemos la función writeChar a la que se le pasará el valor devuelto por invokeSquare, la cual irá recorriendo el array bidimensional, si el valor es verdadero agregará a una cadena <a class=”char”>█</a>, de lo contrario agregará <a class=”spc”>█</a>, una vez que termine con la primera agrupación de subcadenas además agregará un: “<br>”. Para finalmente devolver la cadena total, después esa cadena la agregaremos al body. Una vez que se compruebe que funciona correctamente dando como resultado, para conseguirlo antes se accederá al css de settings para agregar las diferentes clases, en este caso al tener el fondo del documento por defecto blanco y la fuente negra solo deberemos crear el estilo de spc que se le dará una opacidad de 0 para que sea transparente.

█████

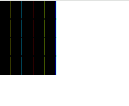
█████

█████

█████

█████

Resultado:



Para corregir las líneas será necesario asignarlas un margin de -1px

Tras conseguir que funcione correctamente crearemos invokeLetter la cual recibirá por parámetro la letra que queremos invocar y en función de la letra que introduzcamos se creará su array bidimensional correspondiente.

Una vez ya tenemos el esquema correcto de todas las letras de la palabra de prueba (faster) movemos las funciones de invokeLetter y writeChar a letters.js

A continuación gracias al siguiente vídeo conseguimos la invocación a una función con N parámetros.

<https://www.youtube.com/watch?v=1Q0npMK0Odw>

Primero crearemos una cadena con los esqueletos de cada una de las letras necesarias.

Después para juntarlas en una palabra primero deberemos sacar la longitud de la palabra en sí, así mismo cuantos “cuadrados” ocupa cada letra, para conseguir realizarlo a tiempo y facilitarnos la tarea haremos que todas las letras ocupen los mismos cuadrados máximos

Y tras mucho tiempo conseguimos que se muestre la palabra faster, con ciertas trampas pero al no quedar demasiado tiempo se da por correcto burro como animal de compañía. Por algún motivo al escribir el contenido mediante javascript siempre parece recortar parte del final del mensaje de manera provisional para solucionarlo lo que se ha hecho es introducir espacios al final del mensaje y así que los caracteres que se “coma” sean los espacios.

Una vez que contemos ya con todas las letras necesarias para los mensajes tocará el seleccionar el mensaje una vez llegada a la subida de nivel de manera aleatoria, mostrarlo un segundo, hacerlo desaparecer, volver a hacer que aparezca para por fin ocultarlo definitivamente.

Para esto haremos la prueba cuando se alcancen los 10 puntos, al llegar a dicha puntuación llamaremos a la función levelup que generará un número aleatorio del 0 al 5, mediante un switch case se pintará la palabra equivalente.

Es en esta página donde encontramos el código js para hacer que el texto parpadee:

<https://codedamn.com/news/html/how-to-create-flashing-blinking-text-using-html>

let <variable> = document.getElementById(<idElemento');

setInterval(function() {

blinking\_text.style.display = (blinking\_text.style.display == 'none' ? '' : 'none');

}, <tiempo>);

Explicado brevemente el intervalo nuevo revisa el display del elemento marcado, si es none, lo quita dejando que el display por defecto se aplique, de lo contrario lo iguala a none y así todo el rato. Con esto vamos al script de draw para crear la función blinking usando la anterior marcada como base. A la función la pasaremos un mínimo de un valor y máximo de tres valores: el id del elemento que queremos que parpadee, el tiempo de cada intervalo y cuantas veces queremos que lo haga. Dentro de la función crearemos la variable: metronomo (para llevar el intervalo) y contador igualado a 0, mientras que los parámetros serán: identifcador que siempre deberá pasársele el id del elemento que queremos que parpadee, time con valor por defecto 250 y repeticiones que será recibida por parámetro con valor por defecto 8. Una vez creada la variable de metronomo y entendiendo el funcionamiento de los parámetros pasaremos a crear la variable objeto la cual igualaremos al valor del documento de dentro del html con el id pasado por el parámetro identificador.

Tras esto estableceremos el intervalo explicado previamente añadiendo dentro del mismo la condición de si contador==repeticiones finalizaremos el intervalo. Ya preparada la función en draw.js ahora proseguimos a rellenar msg con el mensaje que toque cada vez para esta versión y hacerlo más sencillo estableceremos el valor del letrero desde treedude.js justo antes de llamar a blinking, una vez que todo está implementado solo queda probar el código.

### SONORO

Al ir justos de tiempo no nos vamos a enrollar con tratar de obtener el mismo sonido, aprovechamos para modificar el método de setCanal para además de aceptar left y right como valores válidos también aceptar valores numéricos para poder jugar con las distancias, esto se implementa mediante la propiedad <variable>.isNan: devuelve false en caso de ser un número el valor de la variable por lo que al ponerlo en un if hará que se salte la igualación a 0 en caso de ser un número. Luego realizamos la comprobación de si está o no en mute la página, para que si está en mute lance un “error” y no suene nada. De lo contrario creamos una promesa asíncrona que se repetirá desde el -4 al 6 con una pasa de 500 milisegundos entre cada iteración y cada iteración irá intercalando un timbre más agudo o más grabe en función de la iteración

### FUNCIONALIDAD

El mensaje ya se encuentra implementado también el sonido solo falta implementar que de verdad aumente la velocidad, lo que se codificará es: (TEMP) en el if puntos==10 finalizar el intervalo de bajada de energía y volverlo a crear pero con un intervalo menor. Para facilitarnos a futuro haremos que el tiempo del intervalo de energía se guarde en una variable y a esa misma variable irla restando su valor hasta llegar a un mínimo X.

https://alvarotrigo.com/blog/wait-1-second-javascript/

# (EXTRA) Recuperando web 3

Pese a no entrar en el proyecto final también se comentará brevemente el desarrollo de las tareas de recuperación de web 3

Para la recuperación de la asignatura de web 3 como primer ejercicio se nos asignó lo siguiente:

## Enunciado

Un jugador debe iniciar el juego llamando a un método "iniciarPartida". Este método espera recibir 1 ETH (por lo tanto, tiene que ser una función payable). Después podrá llamar al método "jugar" donde tendrá que pasarle su jugada (piedra, papel o tijera) mediante una transacción. Cuando lo haya hecho, el Contrato Inteligente hará de contrincante y mediante un número aleatorio (tendrás que hacer una pequeña investigación para saber cómo generar números aleatorios) le dará su jugada. El contrato tendrá que evaluar las jugadas y determinar un ganador. Si el jugador gana, se llevará su ETH + 1 ETH que le dará el propio contrato inteligente.

Consideraciones:

1. El desarrollo lo puedes hacer directamente en Remix, no hace falta que uses hardhat.
2. Para la generación de números aleatorios NO es necesario que uses Chainlink VRF (lo comento porque encontrarás referencias a este servicio). Puedes hacerlo de manera más simple (aunque sea menos segura).
3. Es necesario que me entregues el código fuente y que despliegues el contrato en la red de Sepolia. Tendrás que indicarme el address del contrato desplegado.

## Obtención de Ether

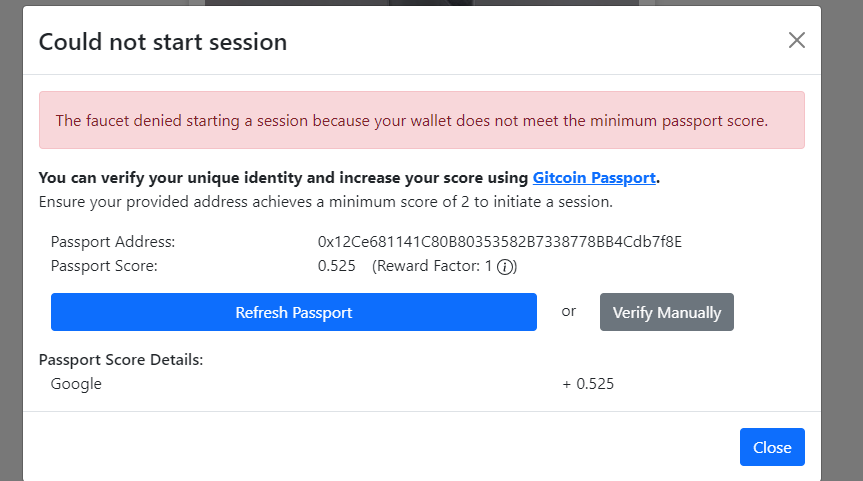
Para comenzar necesitábamos ether para hacer las pruebas y finalmente publicar el contrato y que el mismo funcionase, por lo que nos pusimos con la búsqueda de faucets de Ether, la mayoría de faucets necesitaban que la cartera tuviera un mínimo de ether, hasta que encontramos:

<https://sepolia-faucet.pk910.de/>

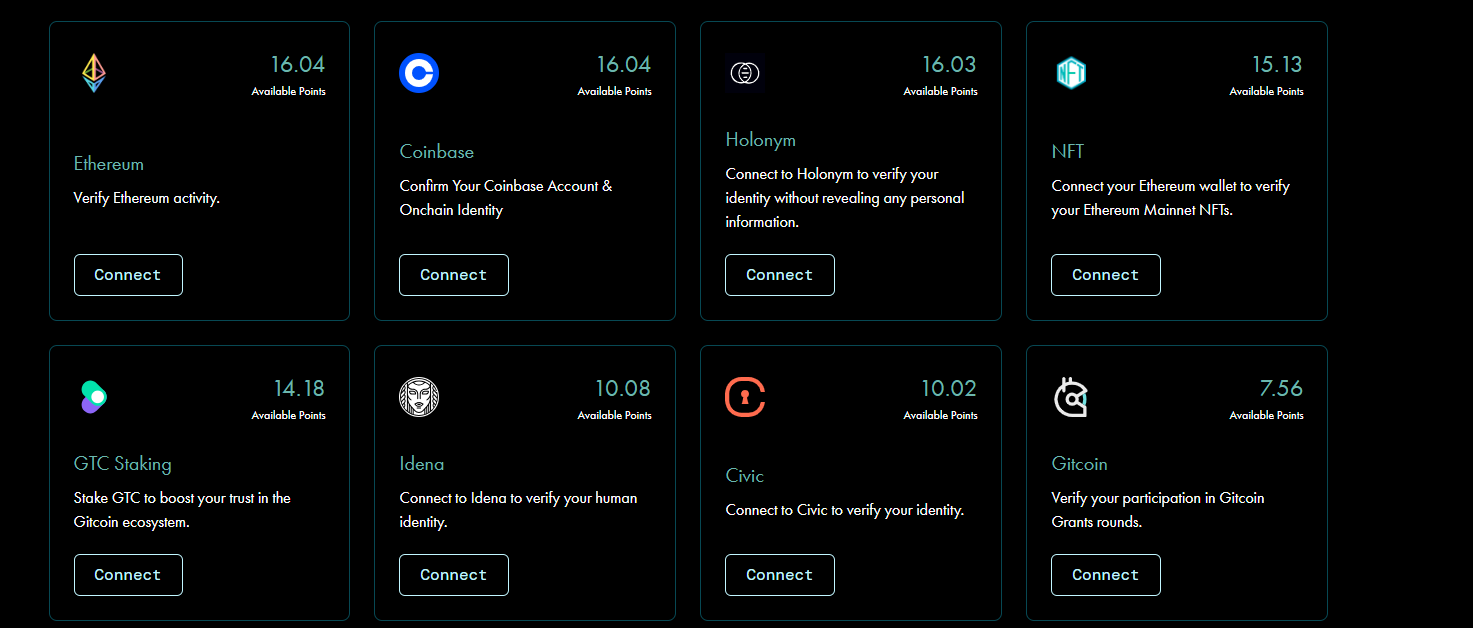
Ahí conseguimos minar en las carteras:

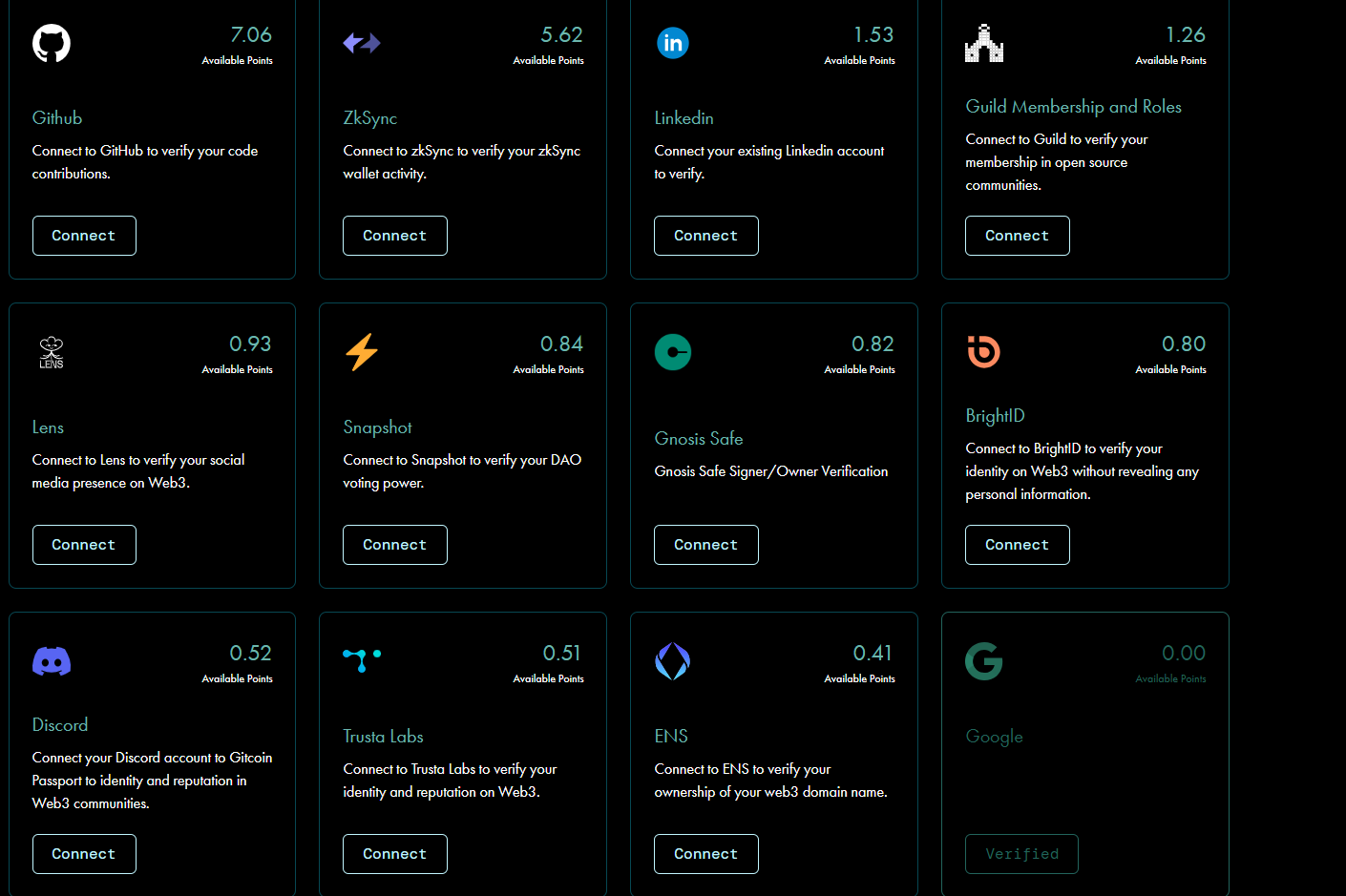
0x0e0d73CE40a90Efd4A526Ec6fc75052c547815C7 y 0x12Ce681141C80B80353582B7338778BB4Cdb7f8E

Alrededor de 8/9 ethers. Tras esto la página nos pedía “aumentar la puntuación de nuestra cartera”



Por lo que nos pusimos a ello intentando dar la menor cantidad de datos posibles, primero nos solicitaba nos solicitaba acceder con Ethereum en <https://passport.gitcoin.co/#/>, y ahí nos daba diferentes opciones de aumentar nuestra puntuación





La 1ª que vimos medio aceptable era la de verificar la cuenta de Google ([Daniel.moyano.fajardo@alumnojoyfe.iepgroup.es](mailto:Daniel.moyano.fajardo@alumnojoyfe.iepgroup.es)), esto nos daba 0.56, desgraciadamente todas las demás nos solicitaban algún tipo de vinculación con cuentas externas o revisión de nuestra inexistente actividad en sus diferentes plataformas, hasta que probamos con Idena la cual en principio solo nos solicitaba verificar que no fuéramos bots, todo bien: tras un par de tests, generación de una clave privada, etc… Llegamos al punto de activar la cuenta para ello teníamos que darnos de alta en una de las siguientes plataformas:

* Telegram
* Discord
* Twitter
* Reddit

Para después enviar nuestros resultados de el o los tests realizados para que uno de los usuarios ya registrados te pase una invitación y activar la cuenta, el problema es que todas las plataformas se encuentran capadas por el proxy de la escuela.

Por lo que se optó a probar en otro navegador a crear una cartera nueva, en este caso Firefox y conseguir más ethers de prueba y trabajar también con ellos

No se consiguió, por lo que tuvimos que ir tirando con el ether que conseguimos acumular entre las dos carteras. Una vez ya tenemos el funcionamiento de las transacciones del contrato nos disponemos a tratar de sacar la manera de obtener nº aleatorios en solidity.

Gracias a esta página obtenemos una manera:

<https://codedamn.com/news/solidity/how-to-generate-random-numbers-in-solidity>

uint (keccak256(abi.encodePacked (msg.sender, block.timestamp, randNo)));

Con esto como base iremos sustrayendo cifras mediante el módulo de 10x hasta quedarnos solo con la 1ª cifra

Gracias a esta

# BIBLIOGRAFÍA:

<https://www.youtube.com/watch?v=NSnckA3LyGw>

Para exportación e importación de archivos js en archivos js.