Trabajo Práctico Especial
Autómatas, Teoría de Lenguajes y Compiladores
Grupo BRITOS
Simple Music Compiler

Integrantes:

Agustin Roca	Legajo: 59160
Guido Barbieri	Legajo: 59567
Nicolas Britos	Legajo: 59529

Índice:

Idea subyacente y objetivo del lenguaje:

La música es algo que todos tendemos a disfrutar, pero hacer música requiere saber tocar algún instrumento o mucho tiempo para aprender a sintetizar música. Investigando sobre librerías o lenguajes de programación notamos que ninguno es lo suficientemente simple aunque no tenga mucha funcionalidad. Algo para principiantes. Nuestra idea viene a resolver eso. Brindamos una forma fácil de poder programar música, que sea simple y fácil de aprender. Tratamos que el lenguaje sea muy fácil de leer y muy intuitivo. El lenguaje también ayuda a aquellas personas que tienen alguna idea para alguna melodía pero que la melodía requiere de cierta habilidad con el instrumento deseado. Por ejemplo, en un piano, esta habilidad podría ser la velocidad necesaria para cambiar de posiciones las manos. Con el lenguaje podría escuchar esa melodía sin necesidad de esa habilidad.

Consideraciones realizadas

Al encontrar que la manera de generar sonido para nosotros era a través de Python, y la librería Simpleaudio, decidimos que el compilador genere un archivo “out.py” que esta armado de forma tal que al ejecutarlo con el intérprete de Python suene la música programada.

Descripción del desarrollo del TP:

Para comenzar tuvimos que buscar una buena forma de reproducir sonidos generados programática. En un principio consideramos usar la librería Portaudio[1] de C pero esta parecía muy compleja y decidimos buscar otra alternativa más sencilla, de forma que podamos concentrarnos en nuestro lenguaje en vez de luchar contra librerías, especialmente siendo que generar sonido resulta una tarea difícil. Así es como nos encontramos con la librería Simpleaudio[2] para Python. Esto hace que sea más fácil ambos el manejo de audio como la portabilidad, algo que podría ser muy beneficioso a futuro.

Teniendo la librería de manejo de audio, investigamos sobre la generación programática de audio y, en especial, música. Desde notas hasta cómo generar programáticamente algo que se asemeje a un piano real.

Hubo también mucha investigación teórica de música ya que la librería Simpleaudio requiere de un arreglo de números en punto flotante que sean las imágenes de una función de onda. Entonces hubo que investigar cómo distintos tipos de ondas generaban distintos sonidos. Con esto encontramos la relación numérica entre las diferentes notas y también que lo único que cambia entre los

[1] Portaudio
[2] Simpleaudio

instrumentos es la forma de la onda. Por ejemplo, un A4 (LA en la octava número cuatro según el índice acústico científico), suena distinto en una violín que en un piano porque la forma de las ondas son distintas pero no cambia la frecuencia ni amplitud (Ver Figura 1).

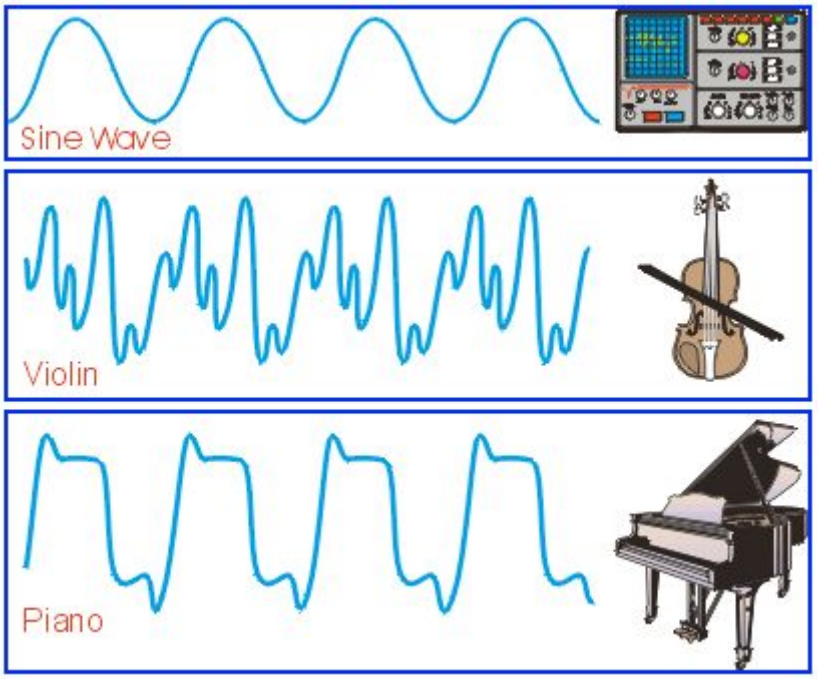

Figura 1. Distintas formas de ondas para un piano y un violín.

Una vez investigado todo esto empezamos a plantear cómo sería nuestro lenguaje y desarrollamos un programa de prueba y una idea de cómo debería verse en Python (este programa sería example.music que está en la carpeta de examples).

De ahí en más el resto del trabajo consistió en armar la gramática, testear y corregir los errores que fueron surgiendo hasta que estuvimos tranquilos de que el compilador funcionaba, agregando también cualquier funcionalidad no contemplada inicialmente que sume para el trabajo, como por ejemplo tener la posibilidad de tocar las notas en una guitarra. Además, tuvimos en cuenta la performance del compilador, para lo que revisamos código y decidimos implementar un hashmap para almacenar las variables que se están analizando, haciendo mucho más rápida la comprobación de asignaciones y operaciones válidas.

Descripción de la gramática:

La gramática del lenguaje del SimpleMusicCompiler está descrita de la siguiente forma:

$G = <V_T, V_N, P, S>$

V_T = símbolos terminales,
V_N = símbolos no terminales,
P = producciones y
S = símbolo distinguido

V_T = { ', ')', '*', '+', ',', '-', '/', ';', '=', '[', ']', '{', '}', error, BOOL_OP, BPM, INTEGER, DOUBLE, VOLUME, NOTE_T, NOTE, INT_T, DOUBLE_T, NEW_ID, WHILE, PLAY, DURING, LENGTH, INT_VAR, DOUBLE_VAR, NOTE_VAR, INT_ARRAY_VAR, DOUBLE_ARRAY_VAR, NOTE_ARRAY_VAR, AS, GUITAR, PIANO, IN, THREAD, IF }

V_N = { S, SET_BPM, VOL, NUMBER, PLAY_FUNC, LENGTH_FUNC, ARRAY, ARRAY_VAR, NUMBER_ARRAY_VAR, NOTE_ARRAY, NOTE_LIST, INT_ARRAY, INT_LIST, DOUBLE_ARRAY, DOUBLE_LIST, IF_BLOCK, WHILE_LOOP, OPEN_BRACKET, CLOSE_BRACKET, EXPRESION, INT_ASSIGNATION, DOUBLE_ASSIGNATION, NOTE_ASSIGNATION, INT_A_ASSIGNATION, DOUBLE_A_ASSIGNATION, NOTE_A_ASSIGNATION, ASSIGNATION, INT_VAL,INT_STRING, DOUBLE_VAL, DOUBLE_STRING, NOTE_VAL, BOOLEAN_VAL, LINE, LINE_P }

P = {
S -> SET_BPM VOL LINE,
SET_BPM -> BPM INT_STRING ';',
VOL -> VOLUME NUMBER ';',
NUMBER -> INT_STRING
    / DOUBLE_STRING,
PLAY_FUNC -> PLAY NOTE_VAL DURING NUMBER
    / PLAY NOTE_VAL DURING NUMBER AS PIANO
    / PLAY NOTE_VAL DURING NUMBER AS GUITAR
    / PLAY NOTE_VAL DURING NUMBER IN THREAD
    / PLAY NOTE_VAL DURING NUMBER AS PIANO IN THREAD
    / PLAY NOTE_VAL DURING NUMBER AS GUITAR IN THREAD,
LENGTH_FUNC -> LENGTH '(' ARRAY ')'
    / LENGTH '(' ARRAY_VAR ')',
ARRAY -> INT_ARRAY

```
        / DOUBLE_ARRAY
        / NOTE_ARRAY,
ARRAY_VAR  ->  NUMBER_ARRAY_VAR
        / NOTE_ARRAY_VAR,
NUMBER_ARRAY_VAR  ->  INT_ARRAY_VAR
        / DOUBLE_ARRAY_VAR,
NOTE_ARRAY  ->  '[' NOTE_LIST ']',
NOTE_LIST  ->  NOTE_VAL ',' NOTE_LIST
        / NOTE_VAL,
INT_ARRAY  ->  '[' INT_LIST ']',
INT_LIST  ->  INT_STRING ',' INT_LIST
        / INT_STRING,
DOUBLE_ARRAY  ->  '[' DOUBLE_LIST ']',
DOUBLE_LIST  ->  DOUBLE_STRING ',' DOUBLE_LIST
        / DOUBLE_STRING,
IF_BLOCK  ->  IF '(' BOOLEAN_VAL ')' OPEN_BRACKET LINE CLOSE_BRACKET,
WHILE_LOOP  ->  WHILE '(' BOOLEAN_VAL ')' OPEN_BRACKET LINE
CLOSE_BRACKET,
OPEN_BRACKET  ->  '{',
CLOSE_BRACKET  ->  '}',
EXPRESION  ->  ASSIGNATION
        / NOTE_VAL
        / NUMBER,
INT_ASSIGNATION  ->  INT_T NEW_ID '=' INT_STRING
        / INT_VAR '=' INT_STRING,
DOUBLE_ASSIGNATION  ->  DOUBLE_T NEW_ID '=' DOUBLE_STRING
        / DOUBLE_VAR '=' DOUBLE_STRING,
NOTE_ASSIGNATION  ->  NOTE_T NEW_ID '=' NOTE_VAL
        / NOTE_VAR '=' NOTE_VAL,
INT_A_ASSIGNATION  ->  INT_T '[' ']' NEW_ID '=' INT_ARRAY
        / INT_ARRAY_VAR '=' INT_ARRAY,
DOUBLE_A_ASSIGNATION  ->  DOUBLE_T '[' ']' NEW_ID '=' DOUBLE_ARRAY
        / DOUBLE_ARRAY_VAR '=' DOUBLE_ARRAY,
NOTE_A_ASSIGNATION  ->  NOTE_T '[' ']' NEW_ID '=' NOTE_ARRAY
        / NOTE_ARRAY_VAR '=' NOTE_ARRAY,
ASSIGNATION  ->  INT_VAR '+' '+'
        / INT_VAR '-' '-'
        / INT_VAR '*' '=' INT_STRING
        / INT_VAR '/' '=' INT_STRING
        / INT_VAR '+' '=' INT_STRING
        / INT_VAR '-' '=' INT_STRING
        / DOUBLE_VAR '+' '+'
```

```
        / DOUBLE_VAR '-' '-'
        / DOUBLE_VAR '*' '=' INT_STRING
        / DOUBLE_VAR '/' '=' INT_STRING
        / DOUBLE_VAR '+' '=' INT_STRING
        / DOUBLE_VAR '-' '=' INT_STRING
        / DOUBLE_VAR '*' '=' DOUBLE_STRING
        / DOUBLE_VAR '/' '=' DOUBLE_STRING
        / DOUBLE_VAR '+' '=' DOUBLE_STRING
        / DOUBLE_VAR '-' '=' DOUBLE_STRING
        / NOTE_A_ASSIGNATION
        / INT_A_ASSIGNATION
        / DOUBLE_A_ASSIGNATION
        / NOTE_ASSIGNATION
        / INT_ASSIGNATION
        / DOUBLE_ASSIGNATION,
INT_VAL  ->  INTEGER
        / INT_VAL '+' INT_VAL
        / INT_VAL '-' INT_VAL
        / INT_VAL '*' INT_VAL
        / INT_VAL '/' INT_VAL,
INT_STRING  ->  INT_STRING '+' INT_STRING
        / INT_STRING '-' INT_STRING
        / INT_STRING '*' INT_STRING
        / INT_STRING '/' INT_STRING
        / INT_ARRAY_VAR '[' INT_STRING ']'
        / INT_VAL
        / LENGTH_FUNC
        / INT_VAR
        / '(' INT_STRING ')',
DOUBLE_VAL  ->  DOUBLE
        / DOUBLE_VAL '+' DOUBLE_VAL
        / DOUBLE_VAL '-' DOUBLE_VAL
        / DOUBLE_VAL '*' DOUBLE_VAL
        / DOUBLE_VAL '/' DOUBLE_VAL
        / DOUBLE_VAL '+' INT_VAL
        / DOUBLE_VAL '-' INT_VAL
        / DOUBLE_VAL '*' INT_VAL
        / DOUBLE_VAL '/' INT_VAL
        / INT_VAL '+' DOUBLE_VAL
        / INT_VAL '-' DOUBLE_VAL
        / INT_VAL '*' DOUBLE_VAL
        / INT_VAL '/' DOUBLE_VAL,
```

```
DOUBLE_STRING  ->  DOUBLE_VAL
        / DOUBLE_VAR
        / DOUBLE_ARRAY_VAR '[' INT_STRING ']'
        / DOUBLE_STRING '+' INT_STRING
        / DOUBLE_STRING '-' INT_STRING
        / DOUBLE_STRING '*' INT_STRING
        / DOUBLE_STRING '/' INT_STRING
        / INT_STRING '+' DOUBLE_STRING
        / INT_STRING '-' DOUBLE_STRING
        / INT_STRING '*' DOUBLE_STRING
        / INT_STRING '/' DOUBLE_STRING
        / '(' DOUBLE_STRING ')',
NOTE_VAL  ->  NOTE
        / NOTE_ARRAY_VAR '[' INT_STRING ']'
        / NOTE_ARRAY_VAR '[' INT_ARRAY_VAR ']'
        / NOTE_VAL '+' INT_STRING
        / NOTE_VAL '+' NOTE_VAL
        / NOTE_VAL '-' INT_STRING
        / NOTE_VAR
        / '(' NOTE_VAL ')',
BOOLEAN_VAL  ->  INT_STRING BOOL_OP INT_STRING
        / DOUBLE_STRING BOOL_OP INT_STRING
        / INT_STRING BOOL_OP DOUBLE_STRING
        / DOUBLE_STRING BOOL_OP DOUBLE_STRING
        / '(' BOOLEAN_VAL ')',
LINE  ->  LINE_P LINE
        / LINE_P,
LINE_P  ->  EXPRESION ';'
        / WHILE_LOOP
        / IF_BLOCK
        / PLAY_FUNC ';'
        / SET_BPM
        / VOL
}

S = S
```

Teniendo en cuenta esta gramática, vemos que lo primero que se tiene que aclarar es el BPM (*Beats Per Minute*) y el volumen de la canción (un número entre 0 y 1). Se puede cambiar ambos valores en el medio del programa, pero obligatoriamente tienen que estar al principio (y en ese orden). Para aclarar esto simplemente hay que poner, `bpm 120;` y `volume 0.5;`, por ejemplo. Luego se programan las líneas del programa que se quieran ejecutar.

Para hacer sonar una nota hay que escribir el comando `play A4 during 1.5 as guitar in thread_1;` donde la sección `as guitar` puede ser `as piano` también, y es opcional. Si no se aclara se toma como default `as piano`. La parte de `in thread_1` es opcional también, en lugar de `thread_1` puede ir cualquier palabra del tipo `thread_(número)`. Si no se aclara, el default es `thread_0`. La sección `during 1.5` significa que se quiere tocar la nota por 1.5 beats, para pasar esto a segundos hay que tener en cuenta el bpm seteado. Por ejemplo si el bpm está seteado en 120, 1.5 beats se traducen como 0.75 segundos. Finalmente la palabra inmediatamente después del `play` debe ser una nota. Esta nota puede estar en una variable o ser desreferenciada desde un arreglo, pero debe ser una nota. Más adelante se hablará de operaciones que devuelven notas también.

También está la posibilidad de hacer un bloque `if` o un bloque `while`, estos tienen la siguiente sintaxis.

```
while(i<4){
     play A4 during 1.5 as guitar in thread_1;
     i++;
}
```

Las indentaciones no son obligatorias pero sí recomendadas. Dentro de los paréntesis obligatoriamente debe haber alguna operación booleana como `<` `>` `==` `<=` `>=` y también los bloques deben estar encerrados por llaves.

Existen seis tipos en este lenguaje, `int`, `double`, `note`, `int[]`, `double[]` y `note[]`. Las únicas operaciones que devuelven ints son `int + int`, `int - int`, `int * int`, `int / int`, desreferenciar un `int[]` y el operador `length` de un array de cualquier tipo.

Para `double` están las mismas primeras cuatro pero si alguno de los dos operandos (o ambos) es un `double` devuelve un `double`.

Finalmente para `note` existen la suma con otra `note` o con un `int`, y también la resta con un `int`. La operación `note + note` lo que hace es sumar las ondas devolviendo una nota simulando que se están tocando ambas notas al mismo tiempo. Por ejemplo, `play C4 + E4 during 1;` lo que haría es tocar ambas notas en el piano durante un beat. La suma y resta con ints cambia los semitonos. Por ejemplo, `A4 + 3` es lo mismo que `C5` porque en la escala musical el orden es, A4 - A#4 - B4 - C5. A esto le vimos especial funcionalidad para la guitarra, se recomienda ver el ejemplo de wonderwall.music para entender mejor esto.

Dificultades encontradas en el desarrollo del TP:

Una de las principales dificultades que tuvimos fueron las inconsistencias de Bison y FLEX entre macOS y Linux. Al final optamos por concentrarnos en Linux. De esto surge que, por algún motivo que no entendimos bien, en macOS el compilador presenta memory leaks al correrlo con Valgrind, los cuales no aparecen en Linux.

También nos encontramos que algunos de los conflictos del tipo shift/reduce eran muy difíciles de resolver. Si bien no es ideal, estos no generan problema.

Con respecto al sonido, el mayor problema fue encontrar alguna fórmula de onda que genere sonido de un piano y una guitarra ya que el sonido de la onda seno común sonaba muy poco real. Para generar ondas de piano utilizamos una aproximación que a pesar de no ser tan certera, suena lo suficientemente cerca a nuestro criterio. La aproximación es la siguiente:

$$Piano(t) = 1.05 * (sin^3(2\pi f(t + 0.188)) + sin(2\pi f(t + \tfrac{2}{3} + 0.188)) * e^{-0.004 * 2\pi * f * t} \quad (1)$$

siendo f la frecuencia de la nota que se desea tocar.

Al comparar el gráfico de esta aproximación con el gráfico de la onda de un piano de la Figura 1 podemos notar diferencias, sin embargo, el sonido generado por ella es lo suficientemente cercano como para ser considerado como el de un piano. Para llegar a esta fórmula nos basamos en las dos respuestas a una pregunta realizada en Stack Exchange[3].

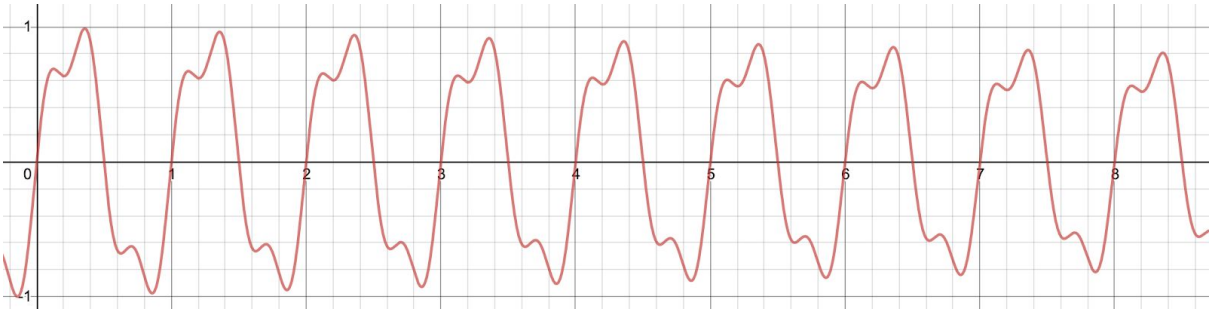

Figura 2. Gráfico de la ecuación 1 en función del tiempo t, tomando f = 1.

En cuanto a la guitarra, utilizamos el algoritmo de Karplus-Strong[4] para simular el sonido que produce una cuerda llegando a resultados muy positivos. Sin embargo, el algoritmo resulta bastante lento en comparación a la ecuación 1.

Uno de los primeros problemas que surgieron fue como saber que frecuencia corresponde a cada nota. Pudimos encontrar una sencilla fórmula para resolver esto:

[3] Mathematical equation for the sound wave that a piano makes
[4] Understanding the Karplus-Strong with Python

$$Freq(Nota) = 440\ Hz * 2^{\frac{s}{12}+(o-4)} \quad (2)$$

siendo s a cuantos semitonos de distancia hay hasta la nota A (La), teniendo en cuenta que el orden es C-C#-D-D#-E-F-F#-G-G#-A-A#-B. Por ejemplo, si la nota es un C (Do), *s* sería -9. Y siendo o el número de la octava de la nota. Usando esta fórmula podemos verificar que la frecuencia del Do central (C4), es $Freq(C4) = 440\ Hz * 2^{\frac{-9}{12}+(4-4)} = 440\ Hz * 2^{\frac{-9}{12}} \approx 261.63\ Hz$.

Otro problema encontrado en cuanto al sonido, fue la opción de tocar varias notas e incluso instrumentos al mismo tiempo, esto se pudo solucionar agregando los "*threads*" en el comando "*play*". Cabe aclarar, que realmente no se arman distintos hilos de ejecución sino que se arman distintas listas de notas por cada thread que luego se suman, aprovechando que para hacer sonar dos sonidos simultáneamente simplemente hay que sumar sus ondas. Decidimos tomar el nombre de *threads* porque nos pareció lo más intuitivo.

Por último, tuvimos que resolver cómo concatenar distintos sonidos sin depender de la velocidad de procesamiento de la computadora para generar nuevos sonidos. Es decir, si teníamos un buffer con las ondas y que se vayan escuchando a medida que estén disponibles, existía la posibilidad de que en algún momento se genere silencio indeseado porque no se pudo procesar cual era el siguiente sonido que se deseaba tocar antes de que terminara el anterior. Esto lo solucionamos procesando todos los sonidos primero y una vez hecho eso tocarlo. Aun así esto podría traer problemas de memoria si la canción es lo suficientemente grande como para superar el límite de memoria que Python puede usar en ejecución.

Futuras extensiones:

La principal extensión que nos gustaría ver a futuro es asincronismo. Cuando discutimos la idea original decidimos que era más importante hacer una forma sincrónica dado que este tipo de programación es más fácil y, como sugiere el nombre del trabajo, queremos que sea fácil. Dicho esto, el agregar asincronismo le daría más poder al lenguaje. Por otro lado, esta tarea no es muy fácil, especialmente por el lado de la planificación.

También sería una adición importante implementar funciones, las cuales podrían ser de gran utilidad, especialmente para la reutilización de código, como podría ser el estribillo de una canción que se repite varias veces a lo largo de ella. Las mismas no deberían ser difíciles de implementar

Se podría agregar también más instrumentos además de guitarra y piano o incluso la posibilidad de que algún usuario más experto ingrese su propia forma de onda para reproducirla en el parlante. La dificultad de esto depende de dos factores. El primero es encontrar una onda que suene como el instrumento, y segundo si se pueden representar con notas también. Esta última nos afectó por ejemplo para percusión. Encontramos un algoritmo que haga ondas similares a distintos tambores pero no nos pareció muy intuitivo representar estos sonidos con notas como A4.

Por último, nuestra implementación tiene un límite, dado que para que suenen bien las notas requerimos que estas estén en una misma lista (probamos múltiples veces el ejecutarlas en distintas listas pero simplemente no pudimos hacer que suene como queríamos). Esto hace que la duración de la música está limitada por la memoria como se mencionó anteriormente. Estaría bueno, a futuro, encontrar una forma de, sin perder calidad, poder tocar las notas individualmente, de esta forma sacando la restricción. La dificultad de esta tarea es desconocida ya que primero se debería investigar como hacerlo y la complejidad de la implementación depende de esto.

Referencias:

- https://dsp.stackexchange.com/questions/46598/mathematical-equation-for-the-sound-wave-that-a-piano-makes
- https://flothesof.github.io/Karplus-Strong-algorithm-Python.html
- https://en.wikipedia.org/wiki/Piano_key_frequencies
- https://en.wikipedia.org/wiki/Scientific_pitch_notation
- http://www.techlib.com/reference/musical_note_frequencies.htm
- https://simpleaudio.readthedocs.io/en/latest/tutorial.html
- https://stackoverflow.com/questions/7666509/hash-function-for-string/7666577#7666577

Canciones utilizadas en los ejemplos provistos:

- River Flows in You - Yurima, (이루마) (hasta el minuto 1:12)
- Lost Boy - Ruth B
- Wonderwall - Oasis (desde el 0:16 hasta el 0:53)