Mission 8: Premorseal Communications

Start date: 9 September 2017
Due: 15 September 2017, 23:59

The Sound Library

We are distinguishing two types here, `sound` and `sourcesound`. You can think of a `sourcesound` as a Source version of an inefficient analog sound, whereas a `sound` is a digital sound that can be efficiently handled using your brower's built-in sound processing capabilities.

To describe an analog sound, you need an analog wave function, as described in Lecture 4B, and the duration of the sound. The wave function takes a time (in seconds) as argument and returns an amplitude (a number between -1 and 1). We say a wave function has the type

```
wave : (number) -> number
```

where the first number represents the argument of the function, the time in seconds, and the second number represents the result, the amplitude of the sound. The following constructor and accessor functions are given:

```
function make_sourcesound(wave, duration) {
    return pair(wave, duration);
}

function get_wave(sourcesound) {
    return head(sourcesound);
}

function get_duration(sourcesound) {
    return tail(sourcesound);
}
```

As usual, make sure you do not break the abstraction of a `sourcesound` and always use these functions to make and access `sourcesound`s.

To try things out, you are given a function `noise`, which has the type

```
noise : (number) -> sourcesound
```

where `number` is the duration of the noisy `sourcesound` to be created.

In order to convert between the inefficient analog `sourcesound`s and the efficient digital `sound`s, you are provided the functions `sourcesound_to_sound` and `sound_to_sourcesound`:

```
sourcesound_to_sound : (sourcesound) -> sound
```

```
sound_to_sourcesound : (sound) -> sourcesound
```

Note that the function `sound_to_sourcesound` carries out the digital sampling described in Lecture 4B.

The `play` function, accepts digital `sounds`; it has the type:

```
play : (sound) -> undefined
```

Note that `sourcesound` is not a very efficient representation of sounds. Make sure that your `sourcesounds` are not longer than two seconds.

You can try `play` as follows:

```
play(sourcesound_to_sound(noise(0.5)));
```

after which you should hear half a second of noise. (If you don't, your browser does not support sound; use a different one or ask your Avenger for advice).

Warning: The sound produced might be very loud! Turn down the volume before you attempt, especially in a public place or if you have headphones on.

The `sourcesound` Discipline

We require that all `sourcesounds` have the following property:

```
(get_wave(sourcesound))(get_duration(sourcesound) + t) === 0
```

for any number `t > 0` . Thus the wave must return 0 when the duration is up. This `sourcesound` discipline will make your tasks a lot easier.

This mission has **four** tasks.

Task 1:

The `play` function in the library only allows you to play `sounds` but not `sourcesounds`. Since you can only create your own custom `sourcesounds`, it would be useful if you had a function that can play `sourcesounds`.

Write a function `play_sourcesound` that takes in a `sourcesound` as input and plays it.

Task 2:

Write a function `cut_sourcesound` that takes a `sourcesound` and a new duration (in seconds) as arguments. It returns a new `sourcesound` that ends at the new duration. You can assume that the new duration is shorter than the duration of the `sourcesound`.

Note: Make sure your `sourcesound` follows the `sourcesound` discipline or marks will be deducted.

Task 3:

A sine wave is defined by

$$\texttt{sine\_wave(t)} = \sin(2\pi f t)$$

where `t` is the time since the begining of the sound (in seconds), `f` is the frequency (in hertz) and `sin` is the trigonometric sine function whose argument is considered given in radians.

Write a function `sine_sound` that takes a frequency (in hertz) and a duration (in seconds) as arguments. It returns a sound of the given duration with the wave function `sine_wave` defined above.

You will find the `math_sin()` function useful for this task.

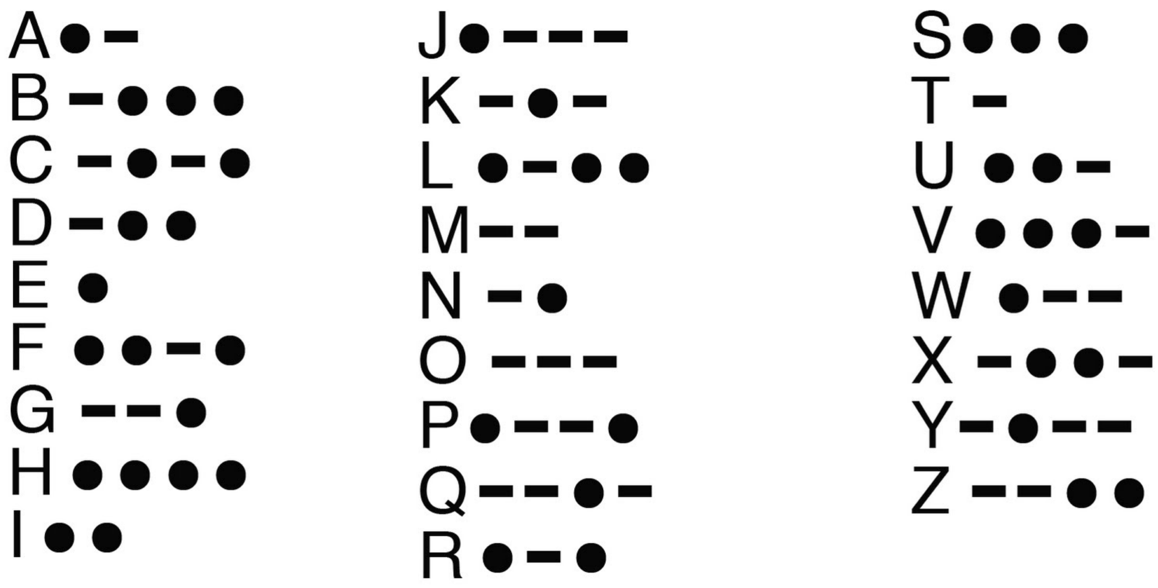
Note: The return value is a `sound`, not a `sourcesound`.

Task 4:

Write a function `consecutively` that takes a list of sounds as argument. It returns a new sound composed of the sounds in the list in sequential order, considering their respective durations.

Now, we need to set up the transmitter and call for help. Use this function to play "SOS" in morse code. The dots are 0.1s long, the dashes are 0.2s long and the pauses are 0.1s long. Use a sine wave of 500Hz for the tone.

It takes some time (about 10-20 seconds) to generate the sound. Please be patient.



A	●▬	J	●▬▬▬	S	●●●
B	▬●●●	K	▬●▬	T	▬
C	▬●▬●	L	●▬●●	U	●●▬
D	▬●●	M	▬▬	V	●●●▬
E	●	N	▬●	W	●▬▬
F	●●▬●	O	▬▬▬	X	▬●●▬
G	▬▬●	P	●▬▬●	Y	▬●▬▬
H	●●●●	Q	▬▬●▬	Z	▬▬●●
I	●●	R	●▬●		

Submission

To submit your work to the Source Academy, place your program in the “Source” tab of the online editor within the mission page, save the program by clicking the “Save” button, and click the “Submit” button. Please ensure the required function from each Task is included in your submission. Note that submission is final and that any mistakes in submission requires extra effort from a tutor or the lecturer himself to fix.

IMPORTANT: Make sure you’ve saved the latest version of your work by clicking the “Save” button before finalizing your submission!