Sonic Pi

PCAD 2021-2022

Sonic Pi Tutorial: https://sonic-pi.net/tutorial.html#section-12

1 Warm-up exercises

Warm-up exercise 1.

- Copy the following code to Sonic Pi and run:

```
sample :loop_amen
sample :loop_amen
```

- Add between the two instructions `sleep 0.877` then run.
- Loop the code exactly three times (a kind of C ++ for-loop)
- Turn it into an infinite loop.
- Add the following code in the same buffer:

```
loop do
   if one_in(2)
     sample :drum_heavy_kick
   else
     sample :drum_cymbal_closed
   end
   sleep 0.877
 end
```

  How can we play the loops simultaneously? (hint: use `in_thread`)

- Modify the code so that the probability of executing `sample: drum_heavy_kick` is lower than the current one.
- Insert a `sleep (0.3)` between the two threads. What happen? Synchronize the two threads (hints: use `cue / sync`).
- Given the following code (to be copied into a new Sonic Pi buffer):

```
live_loop :b do
 play :e4, release: 0.5
  sleep 0.5
end
```

```
live_loop :c do
 sample :bd_haus
  sleep 1
end
```

Synchronize the two live loops (hints: whenever a loop loops, it generates an event of type `cue`).

- In general we can also synchronize more than two threads. Run the following code: the master represents the orchestra director and the slaves the two musicians who must be synchronized with the director:

```
#master
in_thread do
  loop do
    cue :tick
    sleep 2
    cue :tock
    sleep 2
  end
end

#slave1
in_thread do
  loop do
    sync :tick
    sample :drum_cowbell
  end
end

#slave2
in_thread do
  loop do
    sync :tock
    sample :drum_splash_soft
  end
end
```

Warm-up exercise 2. Copy the following code into a new Sonic Pi buffer:

```
a = [6, 5, 4, 3, 2, 1]

live_loop :shuffled do
  a = a.shuffle
  sleep 0.5
end

live_loop :sorted do
  a = a.sort
  sleep 0.5
  puts "sorted: ", a
end
```

Where `a` is a Sonic Pi list, `shuffle` is the function that shuffles the Sonic Pi list, `sort` is the function that sorts the list in ascending order. Run the program. What do you observe? Do the two loops access the shared resource in mutual exclusion? Eliminate the race condition by using the Sonic Pi Time State (hint: use the `set` and `get` methods).

Warm-up exercise 3.
Given the following code:

```
loop do
 sample :perc_bell, rate: (rrand 0.125, 1)
 sleep 2
end
```

Modify the program so that the `rate` time varies between 0.125 and 1.5, and the `sleep` time between 0.2 and 2 (hint: randomization). Run the program several times. Is the melody always the same? Why?