The purpose of the researchers in the past year was to find an easy way to generate all the possible rhythms present in music, in order to extend the dictionary of Automatic Music Composition[?]. Before proceeding with the implementation of the drum machine, it's necessary to dwelve into the algorithm of Euclid and to explain what it has to do with rhythm generation.
In order to compute the greatest common divisor between two integers, the greek mathematician proposed this solution: the smaller number is repeatedly subtracted from the greater until the greater is zero or becomes smaller than the smaller, in which case it is called the remainder. This remainder is then repeatedly subtracted from the smaller number to obtain a new remainder. This process is continued until the remainder is zero[?]. The same procedure can be done more efficiently with divisions.

Algorithm 1 Euclid's Algorithm

```
1: procedure EUCLID(m, k)
2:     if k == 0 then
3:         return m
4:     else
5:         return EUCLID(k, m mod k)
```

The derivation of G. Touissant[?] comes from an analyisis over Bjorklund's studies on SNS accelerators[?] in nuclear physics. In this case, time is divided into intervals and during some of these intervals an onset is to be enabled by a timing system that generates pulses that accomplish this task. The problem for a given number n of time intervals, and another given number $k < n$ of pulses, is to distribute the pulses as evenly as possible among these intervals.
In our case the algorithm to develop take as input a tuple (k, n), where n represents the length of the played sequence and k the number of onsets inside the sequence. The output is a sequence of length n, where the k onsets are equally spaced inside the sequence. For example, if we consider the tuple $(4, 16)$, the result has to be a sequence like:

[1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0]

As Touissant explains in his work[?], with this mechanism we can produce all the possible rhythmic sequences, linking them to a specific musical genre(Figure 1). For instance:

- $E(2, 3) = [1, 0, 1]$ is a common Afro-Cuban drum pattern. For example, it is the conga rhythm of the (6/8)-time Swing Tumbao[?].
- $E(2, 5) = [1, 0, 1, 0, 0]$, when it is started on the second onset ($[1, 0, 0, 1, 0]$) is the metric pattern of Dave Brubeck's *Take Five* as well as *Mars* from *The Planets* by Gustav Holst[?].
- $E(5, 16) = [1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0]$ is the Bossa-Nova rhythm necklace of Brazil, started on the second onset[?].

The application uses as reference the algorithm explained by Bjorklund[?]. This algorithm adopts the Euclidean's one(Algorithm 1), to calculate at each step a remainder, needed to equally divide onsets inside the sequence of 0's and 1's.
Each cycle is composed by two steps: the first consists in creating the base string and

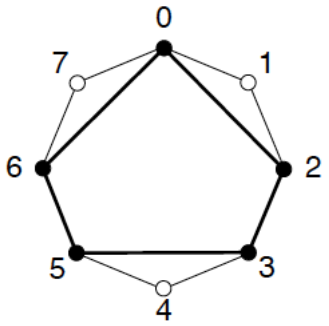

Figure 1: The Euclidean rhythm $E(5, 8)$ of the Cuban cinquillo

the second in spacing the onsets inside the base string.
Considering m the length of sequence and n the number of onsets, and supposing that $n < \frac{m}{2}$, we need to create a sequence of $m - n$ zeros and one another of n ones. In the case of five pulses of thirteen cycles, we start from seqeuences [00000000] and [11111].
We start by simply dividing the five 1's into the eight 0's. This will put a 1 after every 0, with three 0's left over. It is visible that the three 0's correspond to the rest of division between 8 and 5: [0101010101] and [000].
Once again, we can think of this as two optimally distributed strings. One string is ten bits long and contains five "01" pairs (an optimal distribution of five pulses over ten slots). The second string contains three 0's. If we now distribute the three 0's over the five "01" pairs, we get three "010" triples with two "01" pairs left over: [010][010][010][01][01].
We still have two optimally distributed strings. "010010010" is an optimal distribution of three pulses over nine slots and "0101" is an optimal distribution of two pulses over four slots. We now repeat the process, dividing the two "01" strings into the three "010" strings. This gives us two five-bit strings ("01001") with one three-bit string ("010") as a remainder: [01001][01001][010][01].
We can stop the process when the remainder reaches one or zero. Our final pattern, therefore, is [0100101001010] , which is as evenly as we can distribute five pulses over thirteen slots.