POLITECNICO
MILANO 1863

Fuity – a game of chants

Project for:

- Advanced Coding Tools and Methodologies (F. Bruschi, V. Rana)
- Computer Music – Representations and Models (A. Sarti)

Tommaso Sciotto, 914844

Feb 2019

Overview

Fuity is a musical game, intended to improve one's ability to create melodies.

The game revolves around two parrots: one, controlled by the user, sings a melody; the other, controlled by the program, responds with an imitation. The resulting musical form is a **canon**, with the user as leader and the program as follower.

Game dynamics resemble those of **courting**: based on the resulting melodies and their interaction, the follower may move towards or away from the leader. The goal of the game is to make the two parrots kiss.

Name meaning

The canon is a contrapuntal technique closely related to the **fugue** (from the latin *fuga*, “escape”). The common mechanism involves a *subject* melody and one or more *imitations*: in the fugue, melodies usually grow apart from one another, while in the canon they remain closely coupled throughout the entire piece.

Another word derived from *fuga* is the sicilian **fuitina**, indicating the consensual, to some extent theatrical runaway of two lovers.

From *fugue* and *fuitina* derives **Fuity**, a parrot-friendly name, in that even those who can’t speak may pronounce it by whistling.

Canon variables

A canon is generally defined by the **interval** at which the imitation is transposed (e.g. *canon at the III*).

The imitation motion may be **parallel**, preserving the original intervals, or **similar**, scaling them by a certain factor.

The canon may be **inverse**, if intervals in the imitation are the vertical mirror image of those in the subject; it may be **backwards**, if mirroring is performed horizontally in time.

Although more options are traditionally available (*free* vs. *fixed* canon, *mensural* canon), they are not implemented in the current version of the game.

Canons may also be defined by their number of voices: in *Fuity*, the form is by definition a *canon in two voices*.

Tunings

The user may choose between three tuning systems: **Pythagorean tuning** (frequency ratios between notes are composed of powers of 2 and 3), **5-limit just intonation** (ratios composed of powers of 2, 3, 5) and **equal temperament** (ratios composed of powers of the 12^{th} root of 2).

	Major scale ratios
Pythagorean tuning	$1 \quad 9/8 \quad 81/64 \quad 4/3 \quad 3/2 \quad 27/16 \quad 243/128$
5-limit just intonation	$1 \quad 9/8 \quad 5/4 \quad 4/3 \quad 3/2 \quad 5/3 \quad 15/8$
Equal temperament	$1 \quad 2^{2/12} \quad 2^{4/12} \quad 2^{5/12} \quad 2^{7/12} \quad 2^{9/12} \quad 2^{11/12}$

Musical options

The user may select a *tonal reference* (by means of **key**, **octave** and **alteration**) and a **modal scale**; they may also intervene directly on the **signature**.

Imitation notes are transformed by means of an offset **interval**, a scaling **module** and a **sign**. The sign accounts for the *inverse* option.

A separate *reverse* option is available, performing a backwards canon within the scope of a measure.

The **tempo** scan is dynamic and may be changed in real time. A **metronome** option is available.

Keyboard

The user may play notes from a diatonic scale, spanning across two octaves, invariantly mapped over the **computer keyboard** or a parallel **MIDI device**.

Game dynamics

At the beginning of the game, once the leader starts singing, the follower appears and responds after a **4/4 measure**.

At every measure, the resulting performance is **evaluated**, and the follower may move forwards or backwards, reducing or extending the measure by a quarter note.

When the measure is reduced to zero, the user **wins** and the two parrots walk away together, kissing.

When the follower moves further back from the initial position, the user **loses**, and the leader is left alone.

Introduction

Fuity

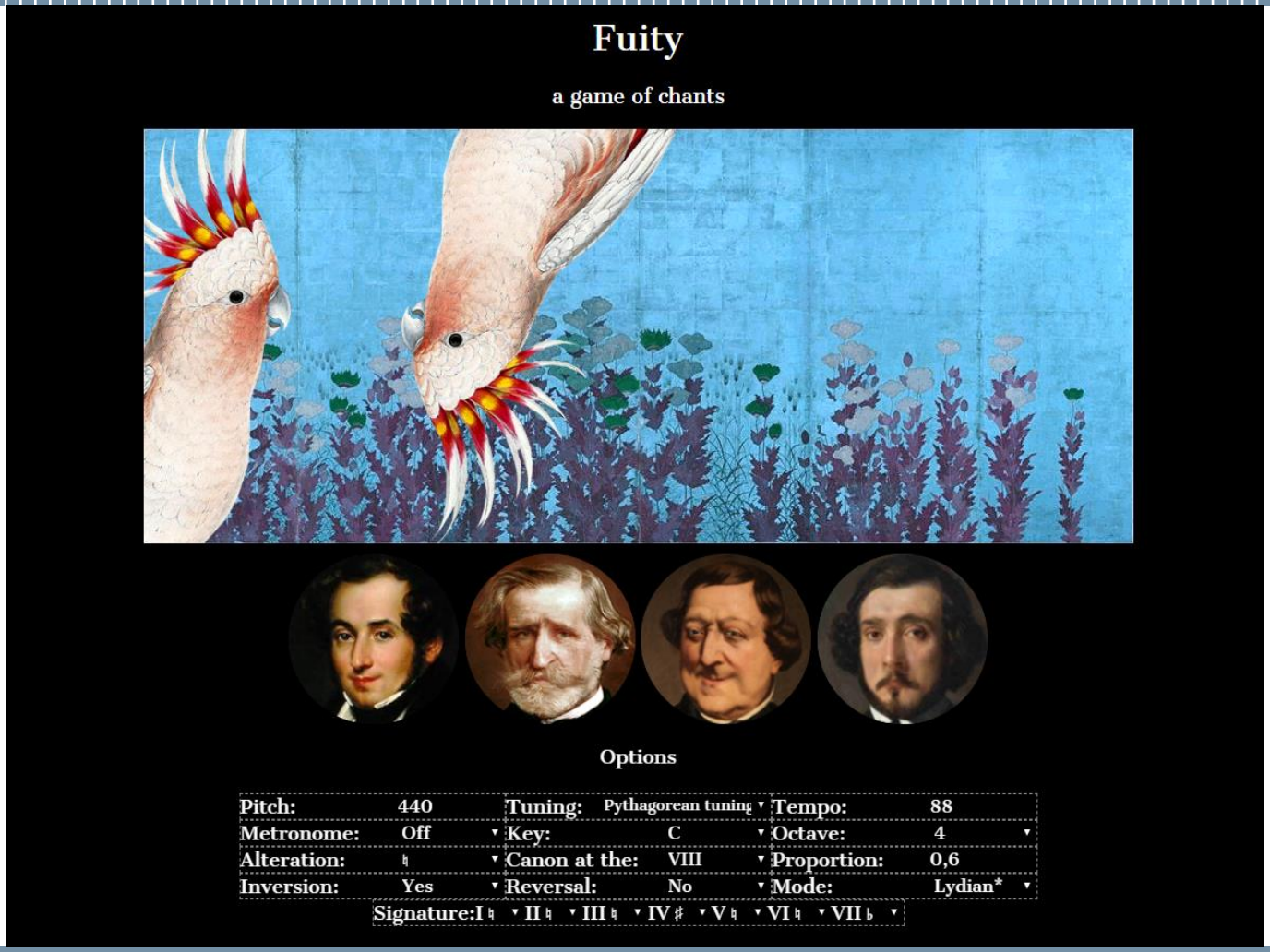
a game of chants

You are a parrot, all by yourself in some sort of peaceful, eerie wilderness. Silence is thick and you burst into singing, although it seems you've never tried before. Such careless attitude attracts an individual of your kind.
How should they respond? As any parrot would: by imitating. Perhaps they mock you, perhaps they show some interest. The only way to figure out is you keep singing.
Just take the right steps, learn to listen, be consistent, not boring. Whatever game it is, you may want to make this a game for two.

In the first page, a brief introduction cues the user on the nature of the game.

In the second page, the user may select his character.

Game interface



Main stage

The two parrots are initially a 4/4 measure apart, the leader standing on the left, and the follower on the right.

The four avatars

Evaluation is performed based on four parameters: **melodic consonance**, **harmonic consonance**, **contour quality** and **variety**.

Each parameter is assigned to an *avatar* of composition, from left to right: Vincenzo **Bellini**, Giuseppe **Verdi**, Gioacchino **Rossini** and Gaetano **Donizetti**.

Melodic consonance

Melodic consonance is evaluated by Bellini, limited to the leader melody, at every new note (if there is more than one note in the melody).

Bellini calculates the difference between the last and previous note in semitones: if the interval is a minor or major 2^{nd} or 3^{rd}, a perfect 4^{th} or 5^{th}, or an ascending minor 6^{th}, he smiles.

In all other cases, including repeated notes, he frowns.

Harmonic consonance

Harmonic consonance is evaluated by Verdi, at every tatum (i.e. 1/32), if both parrots are singing.

Verdi calculates the difference between the follower and leader note in semitones, in their smallest inversion: if the interval is a 3^{rd}, a perfect 5^{th} or a 6^{th}, he smiles.

In all other cases, including unison, he frowns.

Contour quality

Contour quality is evaluated by Rossini, limited to the leader melody, at every new note (if there are more than two notes in the melody).

Rossini observes the pattern in the last three notes: if there's no melodic dissonance, a leap is followed or preceded by an opposite step, or it is a legal double leap (V-IV, III-III minor vs. major, minor III-IV), he smiles.

In all other cases, he frowns.

Variety

Variety is evaluated by Donizetti, at every measure.

Donizetti writes down the melodic pattern of the last and previous measure: while rolling one measure over the other (and wrapping it), he looks for the largest match (i.e. the translation for which most notes are played simultaneousl), and calculates the interval in semitones (*delta*) at every tatum. From the deltas, he then computes the variance of the distribution: if it falls above a certain threshold, variety is sufficient, therefore he smiles.

Otherwise, he frowns.

Variety algorithm - grids

```
//conform grid length to the shortest (due to fluctuations)
if (yourGrid.length < yourPreviousGrid.length)
  yourPreviousGrid = yourPreviousGrid.slice(0, yourGrid.length);
else if (yourPreviousGrid.length < yourGrid.length)
  yourGrid = yourGrid.slice(0, yourPreviousGrid.length);

//subtract their mean value from both grids
if(yourGrid.length > 0) {
  for (var k = 0; k < yourGrid.length; k++) {
    if (!isNaN(yourGrid[k])) {
      yourGridSum += yourGrid[k];
      yourGridValidLength++;
    }
    if (!isNaN(yourPreviousGrid[k])) {
      yourPreviousGridSum += yourPreviousGrid[k];
      yourPreviousGridValidLength++;
    }
  }
  yourGridMean = Math.round(yourGridSum/yourGridValidLength);
  yourPreviousGridMean = Math.round(yourPreviousGridSum/yourPreviousGridValidLength);
  for (var k = 0; k < yourGrid.length; k++)
    yourGrid[k] -= yourGridMean;
  for (var k = 0; k < yourPreviousGrid.length; k++)
    yourPreviousGrid[k] -= yourPreviousGridMean;
}
```

Grids are arrays of note values in semitons, whose index corresponds to the time at which the note is observed.

Current and previous grids are conformed to the shortest length (due to clock fluctuations).

Both grids are then subtracted their mean value.

In this way, melodies are represented by patterns of values distributed in time.

If, at a given time, no note is being played, a NaN value is stored.

Variety algorithm - variance

```
validDeltas = 0;
maxValidDeltas = 0;
delta2 = 0;
delta2sum = 0;
minDelta2sum = 0;
for (var k = 0; k < yourGrid.length; k++) {
  for (var i = 0; i < yourGrid.length; i++) {
    delta2 = (yourGrid[i]-yourPreviousGrid[(i+k)%yourGrid.length])**2;
    if (!isNaN(delta2)) {
      validDeltas++;
      delta2sum += delta2;
    }
  }
  if(validDeltas > maxValidDeltas) {
    maxValidDeltas = validDeltas;
    minDelta2sum = delta2sum;
  }
  else if (validDeltas == maxValidDeltas && delta2sum < minDelta2sum) {
    minDelta2sum = delta2sum;
  }
}

variance = minDelta2sum/Math.abs(maxValidDeltas-1);
```

Deltas express the difference between simultaneous notes.

They are valid if both notes have an integer value (i.e. they are not *NaN*).

As the previous measure (grid) is rolled over the current (for each *k*), it must be wrapped around its length (by the modulo operator *%*).

maxValidDeltas expresses the largest number of valid deltas, the sum of whose squares is stored in *delta2sum.*

maxValidDeltas governs the minimum *delta2sum*, expressed as *minDelta2sum*.

Upon evaluating all possible translations, the two values are used for direct calculation of the distribution variance.

Variety – simplified representation

No translation (k = 0)

i (t)	Previous	Prev - μP	Current	Curr - μC	Δ²
0	2	0	1	-1	1
1	2	0	1	-1	1
2	1	-1	2	0	1
3	1	-1	2	0	1
4	2	0	NaN	NaN	NaN
5	2	0	NaN	NaN	NaN
6	4	2	1	-1	9
7	4	2	1	-1	9
8	5	3	4	2	1
9	NaN	NaN	3	1	NaN
10	2	0	2	0	0
11	1	-1	NaN	NaN	NaN
12	1	-1	3	1	4
13	4	2	3	1	1
14	3	1	1	-1	4
15	NaN	NaN	1	-1	NaN

μP	μC
2,42857	1,92308

∑Δ²	=	32
n	=	11
σ² = ∑Δ² / (n-1) =		3,2

Optimal translation (k = 12)

i (t)	Previous	Prev - μP	Current	Curr - μC	Δ²
0	2	0	1	-1	1
1	2	0	1	-1	1
2	4	2	2	0	4
3	4	2	2	0	4
4	5	3	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN
6	2	0	1	-1	1
7	1	-1	1	-1	0
8	1	-1	4	2	9
9	4	2	3	1	1
10	3	1	2	0	1
11	NaN	NaN	NaN	NaN	NaN
12	2	0	3	1	1
13	2	0	3	1	1
14	1	-1	1	-1	0
15	1	-1	1	-1	0

μP	μC
2,42857	1,92308

∑Δ²	=	24
n	=	13
σ² = ∑Δ² / (n-1) =		2

For k = 0, as for most other translations, NaN values interfere, resulting in a smaller valid sample (11 values vs. 16).

For k = 12, NaN values overlap, giving the largest available sample.

Upon subtracting their mean, values in grids are rounded to the closest integer.

Maximum sample length is prioritized over minimal sum of delta squared.

The first original value in the previous grid is highlighted in red.

Implementation

The game is implemented as an **HTML5** page, relying on a **CSS** stylesheet and **JavaScript** files.

Notes are objects characterized by a diatonic and a chromatic value, a start time, a duration and an audio object.

Audio objects rely on the **Web Audio API**, and comprise an oscillator node, a gain node and scheduling parameters and functions.

Melodies are stored as **arrays** of notes. The imitation is first stored in a measure-long **buffer**; at every measure, the score is evaluated, the buffer is reformatted (shortened, extended, reversed) and added to the follower melody.

Stability issues

JavaScript events rely on an imprecise clock (integer time, expressed in milliseconds) for performing and scheduling tasks, so that invoking a recursive function systematically skews its own scan.

The Web Audio API, on the other hand, relies on a highly precise clock (floating point time, expressed in seconds) for scheduling audio events.

In the current implementation of the game, all time based tasks are performed, under conditions, by a single function, invoked at the animation rate of the browser. Since this scan is unstable, the function is provided with a lookahead (set to a tatum), within which it performes all forecoming events, so as not to skip them as the two clocks drift out of sync.

This solution, though, it stable only for tempos up to 128 BPM, and only if few options are modified.

This issue may be resolved by implementing a worker-based structure, so as to distribute conflicting functions to as many separate threads as possible.

Portability issues

The current version of Fuity was developed and tested in **Chrome 72**, Windows 10 (64-bit) environment.

The program does not work properly in Firefox, Microsoft Edge and Chrome for iPad. More debugging and testing will be needed in order to extend portability, at least among most popular browsers and operative systems.

Evolution

Upon fixing known issues, future updates may include:

- more contrapuntal options (*free/fixed canon, mensural canon, second species counterpoint, difference canon*);
- real time accidents and keyboard shortcuts for most options;
- improved animation and user interface;
- more complex game dynamics, possibly involving more than two voices and more than one player;
- a save option for encoding the performance as a .midi or an .mp3 file.

The ultimate goal would be to make the game playable with voice, including vocoding options and speech generation.

Project links

Playable game:

http://fuity.surge.sh/

Source files repository:

https://github.com/tommasosciotto/fuity/

Trailer:

https://www.youtube.com/watch?v=KSIWT4Mb80o