Task 1 - Initial Constraint System and Melody String Generation

About the task …

This task sets up the minimum constraints – octaves and note durations – needed to generate the melodies for a music sample. A note object is set up to represent the different characteristics of a note: pitch, duration, octave, and its string representation for EasyABC. A global variable for the total number of beats is used to generate melodies of the same length. Durations are generated first, then pitches, and lastly the note objects are generated. More tweaking of constraints will need to be done at a later stage to create better sounding melodies.

Demo

```
[2]> ( demo--generate-melodies )
Melody 1: (A/2' B' B/2' G2' F/2' C/2' D2' A/2' G' A/2' G' B2'
F2' B/2' D2' E' B' C/2' A/2' B' A2' B' D2' G/2')
Melody 2: (C/2' B/2' D2' A/2' B' C/2' B/2' E' D2' B' B2' G2' B'
D2' A/2' B' F/2' F2' A/2' G' G' A/2' G/2' A2')
Melody 3: (A, A2, B, C2, E, B2, C4, G, E4, A, F2, G, C, A2, F,)

Melody 1: (A F2 G2 D/2 A D D F/2 A/2 E/2 F2 B/2 G G/2 B2 E/2 F/2
G2 E A E/2 C2 D/2 G G)
Melody 2: (G' B' E2' F/2' D2' G' E' E2' B2' E' C' F' F' E/2'
D/2' C' C/2' A2' D/2' G/2' B2' C' B/2' C/2')
Melody 3: (D2, D, B, D2, B2, C4, A2, D4, D, G4, C2, E,)

Melody 1: (G2' B/2' D/2' D/2' F' A2' A/2' C2' E2' F2' G' C/2'
D2' G2' G/2' D/2' E2' G/2' C' A/2' C2' D/2')
Melody 2: (D/2 C/2 E/2 D D2 D/2 D2 D G2 A/2 F/2 F2 F2 E2 F2 D2
B/2 F/2 A2 G/2 D C/2)
Melody 3: (E4, C4, G2, A, C, G4, C4, E2, B4,)
NIL
[3]> ( demo--generate-melodies )
Melody 1: (E' D' F' G/2' C2' A2' G2' G' C' E' E/2' E2' G' G2'
D/2' D2' E' E2' A' G/2' F')
Melody 2: (D' F/2' B2' C' D2' G' D2' F2' D/2' A' D2' C/2' E2'
D/2' C/2' F' G2' F2' F2' G/2')
Melody 3: (A, A, F, A, G, F4, D2, E, F2, A, G, C, A2, D4, G2,
G,)
```

```
Melody 1: (A2' B2' C' C/2' B2' F/2' E' B/2' C/2' F/2' F2' E'
C/2' E' G' C' F' F' E/2' D2' A' F2' E' E/2')
Melody 2: (F B2 F B/2 A2 A/2 E/2 A G/2 G/2 D2 D A D/2 B C/2 E2 C
F A2 D2 D2 E/2)
Melody 3: (G2, G4, G, D2, F4, E4, F2, G2, D2, E2, F,)

Melody 1: (E/2' D' F2' A/2' A' D2' B' F2' C/2' D/2' C2' G2' E2'
D' D2' C' D/2' B2' G2' C/2')
Melody 2: (E2' C' A' C2' A/2' B' D' C/2' E/2' D/2' D2' F2' D'
D/2' G2' B2' G2' F2' C/2' D2')
Melody 3: (G4, B, G4, G, G2, F4, B, G4, G, G4,)
NIL
[4]> ( demo--generate-melodies )
Melody 1: (A2 F2 F2 E/2 B/2 A2 D/2 C A B/2 A/2 A/2 A/2 E/2 B2 A
G2 C/2 F C E2 D E B/2)
Melody 2: (E2 A2 A2 G/2 F/2 E2 F/2 E E F/2 E/2 E/2 E/2 G/2 F2 E
B2 E/2 A E G2 F G F/2)
Melody 3: (C, D, A4, C4, F, E2, E4, E2, C4, E, D2,)

Melody 1: (A2' B/2' A2' G' G' C2' F/2' F' F/2' F' F/2' B' D' F2'
C2' B' G2' F/2' D' E' E2' E/2')
Melody 2: (E2' A2' B' A2' F/2' D' E/2' G' C2' D' C2' F/2' E'
B/2' F' F' G' F2' F/2' B' F/2' G2')
Melody 3: (E, A4, E2, C4, D2, C2, G2, A2, D, E2, A, G2, C,)

Melody 1: (E/2 E2 F D2 E/2 A E D D2 A2 G F2 F B B E2 F/2 C2 G2
G/2)
Melody 2: (G/2 G2 A F2 G/2 E G F F2 E2 B A2 A F F G2 A/2 E2 B2
B/2)
Melody 3: (E2, C2, G, C2, F, B2, F4, B4, D4, F2, G2,)
NIL
[5]>
```

Code for the demo

```
; Method to demo all three melody generation
( defmethod demo--generate-melodies ()
    ;Iteration 1
```

```
    ( setf melody1-notes ( generate-melody1 ) )
    ( setf melody2-notes ( generate-melody2 melody1-notes ) )
    ( setf melody3-notes ( generate-bassline ) )
    ( format t "Melody 1: ~A~%" ( display-notes-list melody1-notes ) )
    ( format t "Melody 2: ~A~%" ( display-notes-list melody2-notes ) )
    ( format t "Melody 3: ~A~%" ( display-notes-list melody3-notes ) )
    (terpri)
    ;Iteration 2
    ( setf melody1-notes ( generate-melody1 ) )
    ( setf melody2-notes ( generate-melody2 melody1-notes ) )
    ( setf melody3-notes ( generate-bassline ) )
    ( format t "Melody 1: ~A~%" ( display-notes-list melody1-notes ) )
    ( format t "Melody 2: ~A~%" ( display-notes-list melody2-notes ) )
    ( format t "Melody 3: ~A~%" ( display-notes-list melody3-notes ) )
    (terpri)
    ;Iteration 3
    ( setf melody1-notes ( generate-melody1 ) )
    ( setf melody2-notes ( generate-melody2 melody1-notes ) )
    ( setf melody3-notes ( generate-bassline ) )
    ( format t "Melody 1: ~A~%" ( display-notes-list melody1-notes ) )
    ( format t "Melody 2: ~A~%" ( display-notes-list melody2-notes ) )
    ( format t "Melody 3: ~A~%" ( display-notes-list melody3-notes ) )
)
```

Code for the Initial Constraint System and Melody String Generation

```
; File: acga.lisp
; Hybridized constraint system and genetic algorithm for
; user-interactive algorithmic composition.

( setf *beat-total* 26 )
; Global var for current constraint array we are woring with
( setf *constraint-arr* '() )


; Method to generate a random melody based on the constraints
; of the first melody in the proposal
; note: use format t for EasyABC display method :)
( defmethod generate-melody1 ()
    ( setf *constraint-arr* *melody-durations* )
    ( setf dlist ( generate-durations '() ) )
    ( setf plist ( generate-pitches dlist ) )
    ( generate-notes dlist plist '() ( select-random-arr-element *melody-octaves* ) )
)

; Method to generate melody2 as either a harmonization,
; permutation, or random melody
( defmethod generate-melody2 ( ( m1-notes list ) )
    ( setf choice ( random 3 ) )
    (cond
        (( = choice 0 )
            ( create-harmonization m1-notes )
        )
        (( = choice 1 )
            ; permutation
            ( list-permutation m1-notes )
        )
        (( = choice 2 )
            ;random melody
            ( generate-melody1 )
        )
    )

)

; Method to create a harmony given a list of notes
( defmethod create-harmonization ( ( notes-list list ) )
    (cond
        (( = 0 ( length notes-list ) )
            '()
```

```
; Method to create a harmony given a list of notes
( defmethod create-harmonization ( ( notes-list list ) )
    (cond
        (( = 0 ( length notes-list ) )
            '()
        )
        (t
            ( setf current-pitch ( note-pitch ( car notes-list ) ) )
            ( setf current-pitch-interval ( position current-pitch *CMAJOR* ) )
            (cond
                (( > ( + current-pitch-interval 2 ) ( - (length *CMAJOR*) 1 ) )
                    ( setf new-pitch ( nth ( - current-pitch-interval 3 ) *CMAJOR* ) )
                    ( setf new-note
                        ( make-instance 'note
                            :pitch new-pitch
                            :duration ( note-duration ( car notes-list ) )
                            :octave ( note-octave ( car notes-list ) )
                            :str-representation ( generate-str-representation new-pitch ( note-duration ( car notes-list ) ) ( note-octave ( car notes-list ) ) )
                        )
                    )
                    ( cons new-note ( create-harmonization ( cdr notes-list ) ) )
                )
                (t
                    ( setf new-pitch ( nth ( + current-pitch-interval 2 ) *CMAJOR* ) )
                    ( setf new-note
                        ( make-instance 'note
                            :pitch new-pitch
                            :duration ( note-duration ( car notes-list ) )
                            :octave ( note-octave ( car notes-list ) )
                            :str-representation ( generate-str-representation new-pitch ( note-duration ( car notes-list ) ) ( note-octave ( car notes-list ) ) )
                        )
                    )
                    ( cons new-note ( create-harmonization ( cdr notes-list ) ) )
                )

            )

        )
    )
)

```

```
; Method for rearranging the elements of an input list
( defmethod list-permutation ( ( li list ) )
    (cond
        (( = 0 ( length li ) )
            '()
        )
        (t
            ( setf removed-element ( select-random-arr-element li ) )
            ( cons removed-element ( list-permutation ( remove removed-element li ) ) )
        )
    )
)

; Method to generate a random bassline based on the constraints of the system
( defmethod generate-bassline ()
    ( setf *constraint-arr* *bassline-durations* )
    ( setf dlist ( generate-durations '() ) )
    ( setf plist ( generate-pitches dlist ) )
    ( generate-notes dlist plist '() ( select-random-arr-element *bassline-octave* ) )
)


; Method to take pitches/durations/octaves, instantiate as note objs, and throw into list of notes
( defmethod generate-notes ( ( duration-list list ) ( pitch-list list ) (note-list list) octave )
    (cond
        (( = 0 ( length duration-list ) )

            note-list
        )
        (t
            ( setf current-duration ( get-duration-representation ( car duration-list ) ) )
            ( setf current-pitch ( prin1-to-string ( car pitch-list ) ) )
            ( setf octave-rep ( get-octave-representation octave ) )
            ( setf str-rep ( concatenate 'string current-pitch current-duration octave-rep ))
            ( setf current-note
                ( make-instance 'note
                    :pitch ( car pitch-list )
                    :duration ( car duration-list )
                    :octave octave
                    :str-representation str-rep
                )
            )
            ( setf note-list ( append (list current-note) note-list  ) )
            ( generate-notes ( cdr duration-list ) ( cdr pitch-list ) note-list octave )
        )
    )
```

```
; Method that generates a list of durations based on
; the *beat-total*
( defmethod generate-durations ( ( duration-list list ) )
    ( cond
        (( > ( sum duration-list ) *beat-total* )
            ( generate-durations '() )
        )
        (( = ( sum duration-list ) *beat-total* )
            duration-list
        )
        (t
            ( setf duration-list ( append duration-list ( list ( select-random-arr-element *constraint-arr* ) ) ) )
            ( generate-durations duration-list )
        )
    )
)

; Help Method that sums the elements of a list together -- used for duration calculation
( defmethod sum ( ( li list ) ) )
    (cond
        (( = ( length li ) 0 )
            0
        )
        (t
            ( + ( car li ) ( sum ( cdr li ) ) )
        )
    )
)

; Helper Method that selects a random element from a list
( defmethod select-random-arr-element ( ( li list ) ) )
    ( nth ( random ( length li ) ) li )
)

; Class for note representation
( defclass note ()
    (
        ( pitch :accessor note-pitch :initarg :pitch )
        ( duration :accessor note-duration :initarg :duration )
        ( octave :accessor note-octave :initarg :octave )
        ( str-representation :accessor note-str-representation :initarg :str-representation )
    )
)
```

```
; Method to get the duration of a note as represented in EasyABC
( defmethod get-duration-representation ( duration )
    ( setf duration-assoc ( pairlis '(4 2 1 0.5 0.25 ) '("4" "2" "" "/2" "/4") ))
    ( cdr ( assoc duration duration-assoc ) )
)

; Method to get the octave of a note as represented in EasyABC
( defmethod get-octave-representation ( octave )
    (cond
        (( = octave 1 )
            "\,"
        )
        (( = octave 2 )
            ""
        )
        (( = octave 3 )
            "\'"
        )
    )
)

; Method that generates a list of pitches based on the
; length of duration-list
( defmethod generate-pitches ( ( duration-list list ) )
    ( cond
        (( = ( length duration-list) 1 )
            ( list ( select-random-arr-element *CMAJOR* ) )
        )
        (t
            ( cons ( select-random-arr-element *CMAJOR* ) ( generate-pitches ( cdr duration-list ) ) )
        )
    )

)
```

```
211   ; Method to generate the str-representation of a note
212   ; based on pitch, duration, and octave
213   ( defmethod generate-str-representation ( p d o )
214       ( setf current-duration ( get-duration-representation d ) )
215       ( setf current-pitch ( prin1-to-string p ) )
216       ( setf octave-rep ( get-octave-representation o ) )
217       ( concatenate 'string current-pitch current-duration octave-rep )
218
219   )
220
221   ; Method to display the list of notes nicely
222   ( defmethod display-notes-list (( notes-list list ))
223       ( mapcar #'note-str-representation notes-list )
224   )
225
226
227   ; Knowledge-Base for Constraint System
228   ( setf *CMAJOR* '( C D E F G A B ) )
229   ( setf *melody-durations* '( 2 1 0.5 ) )
230   ( setf *bassline-durations* '( 4 2 1 ) )
231   ( setf *melody-octaves* '( 2 3 ) )
232   ( setf *bassline-octave* '( 1 ) )
233
```