Task 9 – “Double Crossover”

This task features the implementation of three different functionalities: the calculation of “most fit” individuals based on highest melody1 ranking and highest melody1 ranking, crossover of a single melody, and crossover of four individuals into one based on *melody1-rank* and *melody2-rank*. Essentially, the parents for melody1 are the individuals with top rated *melody1-rank* and the second-best *melody1-rank* from a selection. The parents for melody2 follow a similar suit, just based on *melody2-rank* instead of *melody1-rank*. All of the melodies go through crossover and get placed into a new individual. Crossover of a single melody involves taking the first half of notes (based on duration) from one parent and appending it with the latter half of notes from another parent.

Most Fit Individual Demo

```
[2]> ( demo--most-fit-individual )
Selected melody1-rank...(1 4 6)
Most fit melody1-rank: 6

Selected melody2-rank...(8 9 6)
Most fit melody2-rank: 9
NIL
[3]> ( demo--most-fit-individual )
Selected melody1-rank...(2 4 8)
Most fit melody1-rank: 8

Selected melody2-rank...(9 6 7)
Most fit melody2-rank: 9
NIL

[4]>
```

Most Fit Individual Demo Code

```
; Demo that showcases the most-fit-bassline and most-fit-melodies methods.
( defmethod demo--most-fit-individual ()
    ( setf popu ( initial-population ) )
    ( setf selection ( select-individuals popu ) )
    ( assign-random-ranks selection )
```

```
    ( format t "Selected melody1-rank...~A~%" ( mapcar
#'music-melody1-rank selection ))

    ( format t "Most fit melody1-rank: ~A~%~%" ( music-melody1-rank (
most-fit-melody1 selection ) ) )

    ( format t "Selected melody2-rank...~A~%" ( mapcar
#'music-melody2-rank selection ) )

    ( format t "Most fit melody2-rank: ~A~%" ( music-melody2-rank (
most-fit-melody2 selection ) ) )

)
```

Most Fit Individual Code

```
; Method to calculate and display the highest-ranked bassline rank in a
list of
; music individuals.
( defmethod most-fit-melody2 ( ( selection list ) &aux max-value
max-individual )
    ( setf max-individual ( max-val selection 0 #'music-melody2-rank ) )
    max-individual
)

; Method to calculate and display the highest-ranked melodies rank in a
list of
; music individuals.
( defmethod most-fit-melody1 ( ( selection list ) &aux max-value
max-individual )
    ( setf max-individual ( max-val selection 0 #'music-melody1-rank ) )
    max-individual
)

; Method to calculate the maximum value given a list.
( defmethod max-val ( ( l list ) current-max func )
```

```
    (cond
        (( null l )
            current-max
        )
        (( or ( equal current-max 0 ) ( > ( funcall func ( car l ) ) ( 
funcall func current-max ) ) )
            ( max-val ( cdr l ) ( car l ) func )
        )
        (t
            ( max-val ( cdr l ) current-max func )
        )
    )
)
```

Melody Crossover Demo

```
[4]> ( demo--crossover )
------------------MELODY CROSSOVER TEST------------------
Mother: (E2' A/2' D2' D' F2' G/2' C' B2' F2' A/2' G2' A2' B/2'
F2' C2' B/2' C/2' C/2' D/2')
Father: (E' A' A/2' G2' G/2' D' D/2' D2' A/2' A2' F2' E' B/2' G'
B/2' G2' G' F' B2' A2')

 PERFORMING CROSSOVER...
Child: (E2' A/2' D2' D' F2' G/2' C' B2' F2' E' B/2' G' B/2' G2'
G' F' B2' A2')
NIL
[5]> ( demo--crossover )
------------------MELODY CROSSOVER TEST------------------
Mother: (A/2 F2 G/2 F/2 D A F/2 E2 E F F/2 G/2 B/2 F/2 B/2 B D2
C G/2 F2 B B2 D2)
Father: (D2 F/2 B G2 G A/2 C/2 G2 A2 E E2 A B2 D D/2 F2 F/2 A/2
C/2 A/2 F)

 PERFORMING CROSSOVER...
Child: (A/2 F2 G/2 F/2 D A F/2 E2 E F F/2 G/2 B/2 F/2 B/2 B D2
B2 D D/2 F2 F/2 A/2 C/2 A/2 F)
NIL
```

Melody Crossover Demo Code

```
; Demo method to show the crossover performed on a single melody.
( defmethod demo--crossover ()
    ( setf m ( generate-melody1 ) )
    ( setf f ( generate-melody1 ) )

    ( format t "------------------MELODY CROSSOVER
TEST------------------~%")
    ( format t "Mother: ~A~%" ( display-notes-list m ) )
    ( format t "Father: ~A~%" ( display-notes-list f ) )

    ( format t "~% PERFORMING CROSSOVER... ~%" )
    ( setf new-melody ( melody-crossover m f ) )

    ( format t "Child: ~A~%" ( display-notes-list new-melody ) )
)
```

Melody Crossover Code

```
; Method that performs the crossover of a single melody by
; taking the first half of one melody notes list and putting
; it together with the latter half of the melody notes list based
; on a randomly generated duration within *beat-total*.
( defmethod melody-crossover ( ( m list ) ( f list ) &aux dpos )
    ( setf dpos ( + 1 ( random *beat-total* ) ) )

    (setf result ( append ( first-n m dpos ) ( rest-n f dpos ) ))

    (cond
        (( = ( sum ( mapcar #'note-duration result ) ) *beat-total* )
            result
        )
        (t
```

```
            ( melody-crossover m f )
        )
    )


)

; Method that returns the first half of a given list
; based on duration.
( defmethod first-n ( ( m list ) dpos )
    (cond
        (( <= dpos 0 )
            '()
        )
        (t
            ( cons ( car m ) ( first-n ( cdr m ) ( - dpos ( note-duration
(car m ) ) ) ) ) )
        )
    )

)

; Method that returns the second half of a given list
; based on input duration.
( defmethod rest-n ( ( f list ) dpos )
    (cond
        (( <= dpos 0 )
            f
        )
        (t
            ( rest-n ( cdr f ) ( - dpos ( note-duration ( car f ) ) ) ) )
        )
    )
)
```

Double Crossover Demo

```
[2]> ( demo--double-crossover )
```

```
------------------MUSIC CROSSOVER TEST------------------
Mother:
E E/2 B/2 E A2 D D/2 B A/2 G/2 A2 G B/2 G A/2 B2 C2 E2 D B E/2
A2
Father:
D C/2 C E2 G2 D2 E/2 C G/2 A/2 C2 A2 B B2 D E2 D2 C
Child:
E E/2 B/2 E A2 D D/2 B A/2 G/2 A2 G B/2 G A2 B B2 D E2 D2 C
Mother:
F E/2 E G2 B2 F2 G/2 E B/2 E/2 E2 E2 F F2 F G2 F2 E
Father:
G4, F2, C4, E, C4, D, C2, A4, G2,
Child:
F E/2 E G2 B2 F2 G/2 E E, C4, D, C2, A4, G2,
----------------Individual 0----------------
X:1
T:Individual 0
C:Dystopian Tuesday
M:4/4
L:1/4
Q:1/4=120
V:S clef=treble name=S
V:A clef=treble name=A
%%score [ ( S ) ( A ) ]
K:C
V:S
%%MIDI program 0
E E/2 B/2 E A2 D D/2 B A/2 G/2 A2 G B/2 G A2 B B2 D E2 D2 C
V:A
F E/2 E G2 B2 F2 G/2 E E, C4, D, C2, A4, G2,
NIL
[3]>
```

Double Crossover Demo Code

```
; Method for demoing double crossover.
( defmethod demo--double-crossover ()
```

```
    ( setf popu ( initial-population ) )
    ( setf selection ( select-individuals popu ) )
    ( assign-random-ranks selection )
    ( setf melody1-m ( most-fit-melody1 selection ) )
    ( setf melody1-f ( most-fit-melody2 ( remove melody1-m selection ) ) )
    ( format t "------------------MUSIC CROSSOVER
TEST------------------~%")
    ( format t "Mother: ~%" )
    ( display-melody1 melody1-m )
    ( format t "Father: ~%" )
    ( display-melody1 melody1-f )



    ( setf melody2-m ( most-fit-melody2 selection ) )
    ( setf melody2-f ( most-fit-melody2 ( remove melody2-m selection ) ) )

    ( setf new-sample ( double-crossover melody1-m melody1-f melody2-m
melody2-f ) )

    ( format t "Child: ~%" )
    ( display-melody1 new-sample )

    ( format t "Mother: ~%" )
    ( display-melody2 melody2-m )
    ( format t "Father: ~%" )
    ( display-melody2 melody2-f )
    ( format t "Child: ~%" )
    ( display-melody2 new-sample )

    ( easyabc-display new-sample )

)


; Demo helper method that randomly assigns ranks to a list of music
individuals
; Meant to simulate interactive selection without me having to interact :)
( defmethod assign-random-ranks ( ( selection list ) &aux current-m )
    ( cond
```

```
        (( > ( length selection ) 0 )
            ( setf current-m ( car selection ) )
            ( setf ( music-melody2-rank current-m ) ( random 11 ) )
            ( setf ( music-melody1-rank current-m ) ( random 11 ) )
            ( setf ( music-rank current-m )
                ( + ( music-melody2-rank current-m ) ( music-melody1-rank
current-m ) )
            )

            ( assign-random-ranks ( cdr selection ) )
        )
    )
)
```

Double Crossover Code

```
; Double-crossover method - takes four music individuals as input:
;    -m1 -> top melody1-rank music individual
;    -f1 -> second best melody1-rank music individual
;    -m2 -> top melody2-rank music individual
;    -f2 -> second best melody2-rank music individual
; Creates a new individual with a crossover of the two top melody1-rank
music
; individuals and a crossover ot the two top melody2-rank music
individuals
( defmethod double-crossover ( ( m1 music ) ( f1 music ) ( m2 music ) ( f2
music )
    &aux melody1-m1 melody2-m2
         melody1-f1 melody2-f2 )

    ( setf melody1-m1 ( music-melody1 m1 ) )
    ( setf melody2-m2 ( music-melody2 m2 ) )

    ( setf melody1-f1 ( music-melody1 f1 ) )
    ( setf melody2-f2 ( music-melody2 f2 ) )
```

```
    ( setf m1-crossover ( melody-crossover melody1-m1 melody1-f1 ) )
    ( setf m2-crossover ( melody-crossover melody2-m2 melody2-f2 ) )

    ( setf new-melody2-rank ( + ( music-melody2-rank m2 ) (
music-melody2-rank f2 ) ) )
    ( setf new-melody1-rank ( + ( music-melody1-rank m1 ) (
music-melody1-rank f1 ) ) )
    ( setf new-music-rank ( + new-melody2-rank new-melody1-rank ) )

    ( make-instance 'music
        :melody1 m1-crossover
        :melody2 m2-crossover
        ;:melody3 m3-crossover
        :rank new-music-rank
        :melody2-rank new-melody2-rank
        :melody1-rank new-melody1-rank
        :num 0
    )


)
```