Task 10 – Population-Based Mutation Methods

This task adds methods to help perform mutation in preparation for dealing with populations for the genetic algorithm. Task 3 already implemented mutation at the music individual level, so all that needed to be added in this task were the constant for percent mutation and the *maybe-mutate* method. Now, mutation can occur based on a percent probability.

Demo

**Red highlighting denotes mutated notes

```
[5]> ( demo--maybe-mutate )
F/2 C2 F2 C2 D G/2

F/2 C2 F2 C2 D G/2

F/2 C2 F2 C2 D G/2

F/2 C2 F2 C2 D G/2

F/2 C2 F2 C2 D G/2
 *
F/2 C2 G2 C2 D G/2

F/2 C2 G2 C2 D G/2

F/2 C2 G2 C2 D G/2

F/2 C2 G2 C2 D G/2
 *
F/2 F G2 C2 D G/2

F/2 F G2 C2 D G/2
 *
F/2 F G2 C2 C/2 G/2
 *
F/2 F G2 C2 G/2 G/2

F/2 F G2 C2 G/2 G/2
```

```
F/2 F G2 C2 G/2 G/2

F/2 F G2 C2 G/2 G/2
 *
F/2 F G2 C/2 G/2 G/2

F/2 F G2 C/2 G/2 G/2

F/2 F G2 C/2 G/2 G/2
 *
F/2 F G2 G/2 G/2 G/2
 *
F/2 F B/2 G/2 G/2 G/2
```

Demo Code

```
; Demo method for maybe-mutate. Shows whether a music individual
; mutates or not by denoting melody1 with an * for mutation.
; Only Melody1 is shown because mutation affects both melodies,
; so for demo purposes we can show one.
( defmethod demo--maybe-mutate ()
    ( setf m ( generate-music-sample ( random 100 ) ) )
    ( display-melody1 m )
    ( terpri )
    ( dotimes ( x 20 )
        ( display-melody1 m )
        ( if ( maybe-mutate m ) ( princ " *" ) )
        ( terpri )

    )
)
```

Code

```
; ----------Population Mutation Methods----------

; percentage of mutation (set at 50%)
( defconstant *pc-m* 50 )

; Mutates music samples based on *pc-m* probability
( defmethod maybe-mutate ( ( m music ) )

    ( cond
        (( <= ( + 1 ( random 100 ) ) *pc-m* )
            ( mutation m )
            t
        )
        (t
            nil
        )
    )
)
```