Supplementary material for "Comparing First-Order and Second-Order Markov Chains for Algorithmic Melody Generation"

Section A – Programs

1. Order 1 Melody Generator

```
# order1_melody generator (first-order Markov)
# Builds empirical first-order Markov transition probabilities from a single melody sequence (count-and-normalize).
# Pitch and rhythm are modeled separately using independent Markov chains and combined afterward.
Thus, pitch–rhythm dependency is not modeled.
# Generates 75 melodies of 16 note-events each using probabilistic sampling (Python random library).
# Stochastic sampling is used with a random seed to enable reproducibility.
# Pitch encoding: {letter}{optional #}{octave} (e.g., G4, F#5). Durations are in beats.

import random
import os
random.seed(50)
import json
script_dir = os.path.dirname(os.path.abspath(__file__))

# Training melody pitches
pitches = [
    "D5","G4","A4","B4","C5","D5","G4","G4","E5","C5","D5","E5","F#5","G5","G4","G4",
    "C5","D5","C5","B4","A4","B4","C5","B4","A4","G4","F#4","G4","A4","B4","G4","A4",
    "D5","G4","A4","B4","C5","D5","G4","G4","E5","C5","D5","E5","F#5","G5","G4","G4",
    "C5","D5","C5","B4","A4","B4","C5","B4","A4","G4","A4","B4","A4","G4","F#4","G4",
    "B4","G4","A4","B4","G4","A4",
    "D5","E5","F#5","D5",
    "G5","E5","F#5","G5","D5",
    "C#5","B4","C#5","A4",
    "A4","B4","C#5","D5","E5","F#5",
    "G5","F#5","E5","F#5",
    "A4","C#5","D5",
    "D5","G4","F#4","G4",
    "E5","G4","F#4","G4",
    "D5","C5","B4",
    "A4","G4","F#4","G4","A4",
    "D4","E4","F#4","G4","A4","B4",
    "C5","B4","A4",
    "B4","D5","G4","F#4","G4"
]

# Corresponding rhythmic durations
rhythm = [
    1,0.5,0.5,0.5,0.5,1,1,1,1,0.5,0.5,0.5,0.5,1,1,1,
    1,0.5,0.5,0.5,0.5,1,0.5,0.5,0.5,0.5,1,0.5,0.5,0.5,0.5,3,
    1,0.5,0.5,0.5,0.5,1,1,1,1,0.5,0.5,0.5,0.5,1,1,1,
    1,0.5,0.5,0.5,0.5,1,0.5,0.5,0.5,0.5,1,0.5,0.5,0.5,0.5,3,
    1,0.5,0.5,0.5,0.5,1,
    0.5,0.5,0.5,0.5,0.5,
    1,0.5,0.5,0.5,0.5,0.5,
    1,0.5,0.5,1,
    0.5,0.5,0.5,0.5,0.5,0.5,0.5,
    1,1,1,1,
    1,1,3,
    1,0.5,0.5,1,
    1,0.5,0.5,1,
    1,1,1,
    0.5,0.5,0.5,0.5,0.5,1,
    0.5,0.5,0.5,0.5,0.5,0.5,0.5,
    1,1,1,
    0.5,0.5,1,1,
    3
]
assert len(pitches) == len(rhythm), "Training pitch and rhythm streams must be aligned."


# First-order Markov model functions
```

```
def build_first_order_chain(seq):
    """
    Construct a first-order Markov chain from a sequence.
    Returns a dictionary mapping each element to a list of (next_element, probability).
    """
    trans = {}
    for i in range(len(seq)-1):
        a, b = seq[i], seq[i+1]
        trans.setdefault(a, []).append(b)
    chain = {}
    for a, lst in trans.items():
        freq = {}
        for x in lst:
            freq[x] = freq.get(x, 0) + 1
        s = sum(freq.values())
        # Normalize counts to probabilities
        chain[a] = [(k, freq[k]/s) for k in freq]
    return chain

def generate_sequence(chain, length, start_state=None):
    """
    Generate a sequence of given length from a first-order Markov chain.
    If starting state is a valid state in the chain, generation starts from it; otherwise a random
valid start state is used.
    Restarts are counted.
    """
    keys = list(chain.keys())
    curr = start_state if (start_state in chain) else random.choice(keys)
    out = [curr]
    restart_count = 0
    for _ in range(length-1):
        nxts = chain.get(curr)
        if not nxts:
            # fallback: if current state has no outgoing transitions, restart from a random valid
state
            curr = random.choice(keys)
            restart_count += 1
        else:
            choices, probs = zip(*nxts)
            curr = random.choices(choices, probs)[0]
        out.append(curr)
    return out, restart_count

def zip_notes_rhythms(notes, rhythms):
    """
    Combine generated pitches and durations into a sequence of [note, duration] pairs.
    """
    return [[n, r] for n, r in zip(notes, rhythms)]

# Build Markov chains for pitches and rhythms
note_chain = build_first_order_chain(pitches)
rhythm_chain = build_first_order_chain(rhythm)

# Generate 75 synthetic melodies (each 16 notes long)
# Track total and per-melody restart counts during melody generation.
melodies = []
total_restarts = 0
melody_restart_data = []
for i in range(1, 76):
    n_start = random.choice(pitches)
    r_start = random.choice(rhythm)
    n_seq, n_restart = generate_sequence(note_chain, 16, n_start)
    r_seq, r_restart = generate_sequence(rhythm_chain, 16, r_start)
    total = n_restart + r_restart
    total_restarts += total
    melody_restart_data.append({
        "melody_no": i,
        "pitch_restarts": n_restart,
        "rhythm_restarts": r_restart,
        "total_restarts": total
    })
    melodies.append({"melody_no": i, "sequence": zip_notes_rhythms(n_seq, r_seq)})

# Export generated melodies to JSON
```

```
with open(os.path.join(script_dir, "Order1_melodies.json"), "w") as f:
    json.dump(melodies, f, indent=2)

with open(os.path.join(script_dir, "Order1_restart_stats.json"), "w") as f:
    json.dump(melody_restart_data, f, indent=2)

# Compute average restarts per melody and print total restarts and average restarts per melody
avg_restarts = total_restarts/len(melodies)
print("Total restarts:", total_restarts)
print("Average restarts per melody", avg_restarts)
```

2. Order 2 Melody Generator

```
# order2_melody generator (second-order Markov)
# Builds empirical second-order Markov transition probabilities from a single melody sequence (count-
and-normalize).
# Pitch and rhythm are modeled separately using independent Markov chains and combined afterward.
Thus, pitch–rhythm dependency is not modeled.
# Generates 75 melodies of 16 note-events each using probabilistic sampling (Python random library).
# Stochastic sampling is used with a random seed to enable reproducibility
# Pitch encoding: {letter}{optional #}{octave} (e.g., G4, F#5). Durations are in beats.

import random
import os
random.seed(50)
import json
script_dir = os.path.dirname(os.path.abspath(__file__))

# Training melody pitches
pitches = [
    "D5","G4","A4","B4","C5","D5","G4","G4","E5","C5","D5","E5","F#5","G5","G4","G4",
    "C5","D5","C5","B4","A4","B4","C5","B4","A4","G4","F#4","G4","A4","B4","G4","A4",
    "D5","G4","A4","B4","C5","D5","G4","G4","E5","C5","D5","E5","F#5","G5","G4","G4",
    "C5","D5","C5","B4","A4","B4","C5","B4","A4","G4","A4","B4","A4","G4","F#4","G4",
    "B4","G4","A4","B4","G4","A4",
    "D5","E5","F#5","D5",
    "G5","E5","F#5","G5","D5",
    "C#5","B4","C#5","A4",
    "A4","B4","C#5","D5","E5","F#5",
    "G5","F#5","E5","F#5",
    "A4","C#5","D5",
    "D5","G4","F#4","G4",
    "E5","G4","F#4","G4",
    "D5","C5","B4",
    "A4","G4","F#4","G4","A4",
    "D4","E4","F#4","G4","A4","B4",
    "C5","B4","A4",
    "B4","D5","G4","F#4","G4"
]

# Corresponding rhythmic durations (in beats)
rhythm = [
    1,0.5,0.5,0.5,0.5,1,1,1,1,0.5,0.5,0.5,0.5,1,1,1,
    1,0.5,0.5,0.5,0.5,1,0.5,0.5,0.5,0.5,1,0.5,0.5,0.5,0.5,3,
    1,0.5,0.5,0.5,0.5,1,1,1,1,0.5,0.5,0.5,0.5,1,1,1,
    1,0.5,0.5,0.5,0.5,1,0.5,0.5,0.5,0.5,1,0.5,0.5,0.5,0.5,3,
    1,0.5,0.5,0.5,0.5,1,
    0.5,0.5,0.5,0.5,
    1,0.5,0.5,0.5,0.5,
    1,0.5,0.5,1,
    0.5,0.5,0.5,0.5,0.5,0.5,
    1,1,1,1,
    1,1,3,
    1,0.5,0.5,1,
    1,0.5,0.5,1,
    1,1,1,
    0.5,0.5,0.5,0.5,1,
    0.5,0.5,0.5,0.5,0.5,0.5,
    1,1,1,
    0.5,0.5,1,1,
    3
]
assert len(pitches) == len(rhythm), "Training pitch and rhythm streams must be aligned."
```

```
# Second-order Markov model functions

def build_second_order_chain(seq):
    """
    Construct a second-order Markov chain from a sequence.
    Returns a dictionary mapping each pair of consecutive elements to a list of (next_element,
probability).
    """
    trans = {}
    for i in range(len(seq)-2):
        pair = (seq[i], seq[i+1])
        nxt = seq[i+2]
        trans.setdefault(pair, []).append(nxt)
    chain = {}
    for pair, lst in trans.items():
        freq = {}
        for x in lst:
            freq[x] = freq.get(x, 0) + 1
        s = sum(freq.values())
        # Normalize counts to probabilities
        chain[pair] = [(k, freq[k]/s) for k in freq]
    return chain

def generate_sequence(chain, length, starting_pair=None):
    """
    Generate a sequence from a second-order Markov chain.
    If the starting pair exists in the chain, generation starts from it; otherwise a random valid pair
is used.
    Returns a list of length `length`.
    """
    keys = list(chain.keys())
    curr = starting_pair if (starting_pair in chain) else random.choice(keys)
    out = [curr[0], curr[1]]
    restart_count = 0
    for _ in range(length-2):
        nxts = chain.get(curr)
        if not nxts:
            # fallback: if current state pair has no outgoing transitions, restart from a random valid
state pair
            curr = random.choice(keys)
            out.append(curr[1])
            restart_count += 1
        else:
            choices, probs = zip(*nxts)
            nxt = random.choices(choices, probs)[0]
            out.append(nxt)
            # shift pair window for next iteration
            curr = (curr[1], nxt)
    return out, restart_count

def zip_notes_rhythms(notes, rhythms):
    """Combine generated pitches and durations into a sequence of [note, duration] pairs."""
    return [[n, r] for n, r in zip(notes, rhythms)]

# Build second-order Markov chains for pitches and rhythms
note_chain = build_second_order_chain(pitches)
rhythm_chain = build_second_order_chain(rhythm)

# Generate 75 synthetic melodies (each 16 notes long)
# Track total and per-melody restart counts during melody generation.
melodies = []
total_restarts = 0
melody_restart_data = []
for i in range(1, 76):
    # Pick random starting indices for 2-note / 2-duration seeds
    iN = random.randint(0, len(pitches)-2)
    iR = random.randint(0, len(rhythm)-2)
    n_start = (pitches[iN], pitches[iN+1])
    r_start = (rhythm[iR], rhythm[iR+1])
    # Generate sequences using second-order Markov chain
    n_seq, n_restart = generate_sequence(note_chain, 16, n_start)
    r_seq, r_restart = generate_sequence(rhythm_chain, 16, r_start)
    total = n_restart + r_restart
```

```
    total_restarts += total
    melody_restart_data.append({
        "melody_no": i,
        "pitch_restarts": n_restart,
        "rhythm_restarts": r_restart,
        "total_restarts": total
    })
    melodies.append({"melody_no": i, "sequence": zip_notes_rhythms(n_seq, r_seq)})

# Export generated melodies to JSON
with open(os.path.join(script_dir, "Order2_melodies.json"), "w") as f:
    json.dump(melodies, f, indent=2)

with open(os.path.join(script_dir, "Order2_restart_stats.json"), "w") as f:
    json.dump(melody_restart_data, f, indent=2)

# Compute average restarts per melody and print total restarts and average restarts per melody
avg_restarts = total_restarts / len(melodies)
print("Total restarts:", total_restarts)
print("Average restarts per melody:", avg_restarts)
```

3. Random Baseline Generator

```
import random
import json
import os

random.seed(50)

# Training melody pitches
pitches = [
    "D5","G4","A4","B4","C5","D5","G4","G4","E5","C5","D5","E5","F#5","G5","G4","G4",
    "C5","D5","C5","B4","A4","B4","C5","B4","A4","G4","F#4","G4","A4","B4","G4","A4",
    "D5","G4","A4","B4","C5","D5","G4","G4","E5","C5","D5","E5","F#5","G5","G4","G4",
    "C5","D5","C5","B4","A4","B4","C5","B4","A4","G4","A4","B4","A4","G4","F#4","G4",
    "B4","G4","A4","B4","G4","A4",
    "D5","E5","F#5","D5",
    "G5","E5","F#5","G5","D5",
    "C#5","B4","C#5","A4",
    "A4","B4","C#5","D5","E5","F#5",
    "G5","F#5","E5","F#5",
    "A4","C#5","D5",
    "D5","G4","F#4","G4",
    "E5","G4","F#4","G4",
    "D5","C5","B4",
    "A4","G4","F#4","G4","A4",
    "D4","E4","F#4","G4","A4","B4",
    "C5","B4","A4",
    "B4","D5","G4","F#4","G4"
]

# Corresponding rhythmic durations
rhythm = [
    1,0.5,0.5,0.5,0.5,1,1,1,1,0.5,0.5,0.5,0.5,1,1,1,
    1,0.5,0.5,0.5,0.5,1,0.5,0.5,0.5,0.5,1,0.5,0.5,0.5,0.5,3,
    1,0.5,0.5,0.5,0.5,1,1,1,1,0.5,0.5,0.5,0.5,1,1,1,
    1,0.5,0.5,0.5,0.5,1,0.5,0.5,0.5,0.5,1,0.5,0.5,0.5,0.5,3,
    1,0.5,0.5,0.5,0.5,1,
    0.5,0.5,0.5,0.5,
    1,0.5,0.5,0.5,0.5,
    1,0.5,0.5,1,
    0.5,0.5,0.5,0.5,0.5,0.5,0.5,
    1,1,1,1,
    1,1,3,
    1,0.5,0.5,1,
    1,0.5,0.5,1,
    1,1,1,
    0.5,0.5,0.5,0.5,0.5,1,
    0.5,0.5,0.5,0.5,0.5,0.5,0.5,
    1,1,1,
    0.5,0.5,1,1,
    3
]
```

```
# Use unique pitch and rhythm values so the baseline is uniformly random
pitch_pool = sorted(set(pitches))
rhythm_pool = sorted(set(rhythm))

def generate_random_sequence(pool, length):
    return [random.choice(pool) for _ in range(length)]

def zip_notes_rhythms(notes, rhythms):
    return [[n, r] for n, r in zip(notes, rhythms)]

random_melodies = []
for i in range(1, 76):   # 75 random melodies to match FO and SO
    n_seq = generate_random_sequence(pitch_pool, 16)
    r_seq = generate_random_sequence(rhythm_pool, 16)
    random_melodies.append({
        "melody_no": i,
        "sequence": zip_notes_rhythms(n_seq, r_seq)
    })

# Get the directory where this script is located
script_dir = os.path.dirname(os.path.abspath(__file__))
output_path = os.path.join(script_dir, "random_melodies.json")

with open(output_path, "w") as f:
    json.dump(random_melodies, f, indent=2)
```

Section B – Results Data

1. Objective Evaluation Results

Order 1						
Melody No	Ending On Tonic	Interval Smoothness	Stepwise Ratio	Motif Repetition	Rhythmic Variety	Final Score
1	0	1	0.6	0.623333	0.579328	0.560532
2	0	1	0.8	0.292917	0.353838	0.489351
3	0	1	0.733333	0.324583	0.419631	0.495509
4	0	0.866667	0.533333	0.08125	0.397756	0.375801
5	0	0.933333	0.533333	0.302292	0.54968	0.463728
6	0	0.933333	0.8	0.1625	0.219631	0.423093
7	0	1	0.666667	0.292917	0.579328	0.507782
8	0	0.933333	0.866667	0.395208	0.563213	0.551684
9	0	0.933333	0.733333	0.14375	0.652935	0.49267
10	0	1	1	0.871042	0.800632	0.734335
11	0	1	0.8	0.205417	0.677965	0.536676
12	0	0.933333	0.733333	0.383333	0.673294	0.544659
13	1	1	0.866667	0.286458	0.531963	0.737018
14	0	0.733333	0.533333	0.378333	0.106403	0.350281
15	0	0.933333	0.933333	0.51	0.496555	0.574644
16	1	0.866667	0.733333	0.368958	0.521685	0.698129
17	1	0.933333	0.8	0.302292	0.571685	0.721462
18	0	0.933333	0.933333	0.327083	0.460794	0.530909
19	1	0.866667	0.466667	0.355417	0.679215	0.673593
20	0	0.933333	0.666667	0.342292	0.55606	0.49967
21	0	1	0.8	0.249167	0.347756	0.479384
22	0	1	0.666667	0.397083	0.641715	0.541093
23	0	1	0.8	0.52875	0.25593	0.516936
24	0	0.933333	0.933333	0.392917	0.353838	0.522684

25	1	1	0.8	0.390833	0.594463	0.757059
26	0	0.933333	0.733333	0.249167	0.70109	0.523385
27	1	0.866667	0.733333	0.173333	0.577965	0.67026
28	0	0.866667	0.8	0.563958	0.591338	0.564393
29	0	0.933333	0.666667	0.234583	0.73106	0.513129
30	1	1	0.733333	0.311667	0.76843	0.762686
31	1	1	0.8	0.249167	0.648078	0.739449
32	0	1	0.866667	0.360833	0.588919	0.563284
33	0	0.933333	0.666667	0.08125	0.521555	0.440561
34	0	0.933333	0.733333	0.549792	0.52468	0.548228
35	0	1	0.733333	0.323958	0.47481	0.50642
36	1	0.933333	0.733333	0.41375	0.638919	0.743867
37	1	1	0.6	0.377292	0.506113	0.696681
38	0	1	0.933333	0.504583	0.460794	0.579742
39	0	1	0.8	0.311667	0.613213	0.544976
40	1	0.933333	0.666667	0.08125	0.407669	0.617784
41	0	1	0.666667	0.448333	0.640025	0.551005
42	1	0.933333	0.666667	0.473125	0.3496	0.684545
43	0	0.866667	0.733333	0.167917	0.53859	0.461301
44	0	0.933333	0.666667	0.167917	0.688703	0.491324
45	0	1	0.866667	0.224583	0.647965	0.547843
46	0	1	0.933333	0.346042	0.470169	0.549909
47	0	1	0.733333	0.391667	0.704328	0.565866
48	0	0.866667	0.6	0.347083	0.601203	0.482991
49	0	1	0.666667	0.307708	0.66843	0.528561
50	0	0.933333	0.6	0.2	0.71359	0.489385
51	0	0.866667	0.733333	0.249167	0.61234	0.492301
52	0	0.933333	0.733333	0.230417	0.60606	0.500629
53	0	0.933333	0.8	0.167917	0.552525	0.490755
54	1	1	0.866667	0.384583	0.615305	0.773311
55	1	0.866667	0.733333	0.173333	0.397756	0.634218
56	0	0.866667	0.6	0.238333	0.579544	0.456909
57	0	0.866667	0.8	0.167917	0.744953	0.515907
58	0	1	0.933333	0.495833	0.685465	0.622926
59	0	0.866667	0.666667	0.249167	0.646685	0.485837
60	1	0.933333	0.8	0.364583	0.616715	0.742926
61	0	0.8	0.733333	0.234375	0.537274	0.460997
62	0	1	0.466667	0.230417	0.51856	0.443129
63	0	1	0.733333	0.1625	0.698078	0.518782
64	1	0.933333	0.733333	0.340833	0.670169	0.735534
65	0	1	0.866667	0.5925	0.529328	0.597699
66	0	1	0.866667	0.420833	0.716828	0.600866
67	0	0.866667	0.466667	0.08125	0.6994	0.422797
68	0	0.866667	0.866667	0.572917	0.588919	0.579034
69	0	0.933333	0.8	0.302292	0.551419	0.517409
70	0	1	0.866667	0.3875	0.604544	0.571742
71	0	0.866667	0.733333	0.230417	0.694953	0.505074

72	1	1	0.933333	0.4325	0.106403	0.694447
73	1	0.933333	0.666667	0.221042	0.647965	0.693801
74	0	0.933333	0.8	0.167917	0.621275	0.504505
75	0	0.933333	0.8	0.134375	0.641715	0.501885
Averages	0.24	0.943111	0.747556	0.320022	0.563289	0.562796
Order 2						
Melody No	Ending On Tonic	Interval Smoothness	Stepwise Ratio	Motif Repetition	Rhythmic Variety	Final Score
1	0	1	0.8	0.432083	0.579287	0.562274
2	0	1	0.666667	0.5275	0.377988	0.514431
3	0	0.866667	0.6	0.258542	0.563919	0.457825
4	0	1	0.933333	0.47125	0.397756	0.560468
5	0	1	0.8	0.287708	0.737363	0.565014
6	0	1	0.8	0.167917	0.62481	0.518545
7	0	0.933333	0.733333	0.610833	0.410794	0.537659
8	0	1	0.866667	0.403333	0.601162	0.574232
9	0	1	0.733333	0.08125	0.715435	0.506004
10	0	1	0.8	0.411667	0.4746	0.537253
11	0	0.866667	0.733333	0.08125	0.588213	0.453893
12	1	1	0.666667	0.307708	0.481963	0.691268
13	0	1	0.533333	0.325208	0.679215	0.507551
14	1	0.8	0.666667	0.679792	0.619013	0.753094
15	0	1	0.8	0.307708	0.440025	0.509547
16	1	0.933333	0.666667	0.355208	0.315465	0.654135
17	0	1	0.733333	0.134375	0.402072	0.453956
18	0	1	0.933333	0.446042	0.669953	0.609866
19	1	0.8	0.8	0.311042	0.577805	0.697769
20	1	1	0.8	0.298542	0.563919	0.732492
21	0	0.933333	0.666667	0.271042	0.520169	0.478242
22	0	1	0.933333	0.258542	0.5119	0.540755
23	1	0.866667	0.733333	0.08125	0.638662	0.663982
24	1	0.866667	0.733333	0.357708	0.47481	0.686504
25	0	1	0.866667	0.374167	0.579328	0.564032
26	1	1	0.866667	0.304583	0.556113	0.745473
27	1	1	0.8	0.333542	0.579328	0.742574
28	0	0.866667	0.6	0.516458	0.646555	0.525936
29	0	1	0.733333	0.167917	0.596715	0.499593
30	0	0.866667	0.533333	0.214792	0.654215	0.453801
31	0	1	1	0.726458	0.452072	0.635706
32	1	1	0.866667	0.41	0.653838	0.786101
33	0	1	0.866667	0.481875	0.541506	0.578009
34	1	0.933333	0.866667	0.431667	0.694463	0.785226
35	0	1	0.8	0.221042	0.644953	0.533199
36	0	0.933333	0.8	0.311667	0.397756	0.488551
37	1	1	0.8	0.167917	0.69359	0.732301

38	0	1	1	0.49	0.53718	0.605436
39	0	0.866667	0.733333	0.790208	0.5744	0.592922
40	1	1	0.733333	0.399167	0.670169	0.760534
41	0	1	0.866667	0.5325	0.673294	0.614492
42	0	1	0.933333	0.589375	0.415225	0.587587
43	0	0.8	0.666667	0.307708	0.673294	0.489534
44	0	0.933333	0.733333	0.134375	0.654544	0.491117
45	0	1	0.8	0.437083	0.579328	0.563282
46	0	1	0.733333	0.336667	0.646555	0.543311
47	1	0.866667	0.4	0.221042	0.556113	0.608764
48	0	1	0.666667	0.264792	0.673294	0.52095
49	0	0.933333	0.733333	0.47	0.596555	0.546644
50	0	0.933333	0.733333	0.287708	0.5744	0.505755
51	0	0.866667	0.6	0.589167	0.615488	0.534264
52	0	0.866667	0.733333	0.234583	0.657453	0.498407
53	1	1	0.933333	0.427708	0.641338	0.800476
54	1	0.866667	0.666667	0	0.573078	0.621282
55	0	0.933333	0.733333	0.283542	0.716828	0.533407
56	1	1	0.866667	0.458333	0.781024	0.821205
57	0	0.666667	0.6	0.472708	0.500881	0.448051
58	0	0.8	0.666667	0.343333	0.666715	0.495343
59	0	0.933333	0.8	0.336667	0.746649	0.56333
60	0	1	0.933333	0.395208	0.509055	0.567519
61	0	0.933333	0.466667	0.61125	0.397756	0.481801
62	0	1	0.6	0.557708	0.68859	0.56926
63	0	1	0.866667	0.613333	0.3119	0.55838
64	0	1	0.8	0.45375	0.660465	0.582843
65	0	1	0.8	0.720833	0.591506	0.622468
66	1	1	0.866667	0.607917	0.56984	0.808885
67	1	1	0.8	0.400208	0.642044	0.76845
68	1	0.733333	0.466667	0.1625	0.604215	0.593343
69	1	0.866667	0.533333	0.14375	0.591338	0.627018
70	0	1	0.666667	0.249167	0.670169	0.5172
71	1	1	0.8	0.350208	0.6994	0.769922
72	0	1	0.8	0.317083	0.6994	0.563297
73	0	1	0.733333	0.49	0.594463	0.563559
74	1	1	0.733333	0.48625	0.551419	0.7542
75	0	1	0.8	0.368958	0.66231	0.566254
Averages	0.306667	0.950222	0.753778	0.371525	0.584325	0.593303
Random						
Melody No	Ending On Tonic	Interval Smoothness	Stepwise Ratio	Motif Repetition	Rhythmic Variety	Final Score
1	0	0.8	0.266667	0	0.704506	0.354235
2	0	0.6	0.266667	0	0.810697	0.335473
3	0	0.8	0.266667	0	0.748256	0.362985

4	0	0.8	0.4	0.08125	0.736278	0.403506
5	0	0.666667	0.333333	0	0.784226	0.356845
6	0	0.6	0.133333	0	0.745653	0.295797
7	0	0.6	0.266667	0.08125	0.785697	0.346723
8	0	0.733333	0.133333	0	0.767528	0.326839
9	1	0.8	0.333333	0.08125	0.844382	0.611793
10	0	0.533333	0.133333	0	0.79726	0.292785
11	0	0.733333	0.2	0.08125	0.646726	0.332262
12	0	0.666667	0.266667	0.08125	0.827976	0.368512
13	0	0.4	0.133333	0	0.698197	0.246306
14	0	0.6	0.2	0.086667	0.862351	0.349803
15	1	0.6	0.133333	0	0.779506	0.502568
16	0	0.733333	0.333333	0.339792	0.826002	0.446492
17	0	0.4	0.2	0.08125	0.713822	0.279014
18	1	0.6	0.2	0.08125	0.796726	0.535595
19	0	0.666667	0.133333	0.08125	0.862351	0.34872
20	0	0.6	0.533333	0.292917	0.744135	0.434077
21	0	0.6	0.2	0	0.727805	0.305561
22	1	0.733333	0.466667	0.20625	0.779506	0.637151
23	0	0.466667	0.066667	0.08125	0.756101	0.274137
24	0	0.666667	0.533333	0	0.779506	0.395901
25	0	0.466667	0.133333	0	0.771726	0.274345
26	0	0.8	0.333333	0.08125	0.767006	0.396318
27	0	0.6	0.2	0	0.791947	0.318389
28	0	0.533333	0.4	0.08125	0.74089	0.351095
29	0	0.466667	0.133333	0.1625	0.856101	0.32372
30	0	0.666667	0.133333	0.08125	0.662351	0.30872
31	0	0.733333	0.066667	0	0.792528	0.318506
32	0	0.733333	0.266667	0	0.76601	0.353202
33	0	0.533333	0.266667	0	0.825385	0.325077
34	0	0.6	0.333333	0	0.718601	0.330387
35	0	0.8	0.4	0.287708	0.734226	0.444387
36	0	0.533333	0.333333	0	0.76218	0.325769
37	0	0.866667	0.333333	0	0.738502	0.3877
38	0	0.6	0.4	0	0.773256	0.354651
39	0	0.733333	0.333333	0	0.70339	0.354011
40	1	0.6	0.2	0	0.841947	0.528389
41	1	0.466667	0.133333	0.08125	0.810756	0.498401
42	0	0.6	0.133333	0	0.787765	0.30422
43	0	0.6	0.266667	0	0.744135	0.32216
44	0	0.8	0.4	0	0.645072	0.369014
45	0	0.6	0.333333	0	0.59101	0.304869
46	0	0.533333	0.333333	0	0.81601	0.336535
47	0	0.733333	0.2	0	0.812765	0.34922
48	0	0.533333	0.066667	0	0.579447	0.235889
49	0	0.8	0.466667	0.08125	0.683153	0.406214
50	0	0.6	0.2	0.08125	0.752899	0.32683

51	0	0.533333	0.333333	0	0.836278	0.340589
52	0	0.466667	0.266667	0.14375	0.77226	0.329869
53	0	0.6	0.2	0.221042	0.715476	0.347303
54	0	0.933333	0.4	0	0.778757	0.422418
55	0	0.6	0.4	0.086667	0.826903	0.382714
56	1	0.333333	0.266667	0.08125	0.635377	0.463325
57	0	0.6	0.133333	0.221042	0.74089	0.339053
58	0	0.733333	0.066667	0	0.784226	0.316845
59	0	0.533333	0.2	0	0.740435	0.294754
60	0	0.4	0.133333	0.08125	0.83464	0.289845
61	0	0.466667	0.266667	0.08125	0.801381	0.323193
62	0	0.6	0.266667	0	0.751381	0.32361
63	1	0.733333	0.266667	0	0.82214	0.564428
64	1	0.4	0.133333	0	0.801381	0.466943
65	0	0.733333	0.466667	0.08125	0.769752	0.4102
66	0	0.533333	0.333333	0	0.80976	0.335285
67	0	0.466667	0.266667	0	0.771649	0.300997
68	0	0.4	0.133333	0.08125	0.719953	0.266907
69	0	0.466667	0.266667	0.08125	0.809226	0.324762
70	0	0.8	0.133333	0.08125	0.72226	0.347369
71	0	0.8	0.2	0	0.529127	0.305825
72	0	0.6	0.466667	0.086667	0.767528	0.384172
73	0	0.8	0.4	0	0.787765	0.397553
74	0	0.733333	0.266667	0	0.656113	0.331223
75	0	0.733333	0.266667	0	0.681024	0.336205
Averages	0.12	0.625778	0.259556	0.0523	0.758132	0.363153
order1_avg						0.562796
order2_avg						0.593303
random_avg						0.363153

2. Subjective Evaluation Results

i) Participant Background

Listener Evaluation Responses		
Participant ID	Music training	More musical group
B0hV9bmgn8yGY8AuN4t2U	Advanced	Group A
NgIj92UkkwLrSfPCNnFhD	Advanced	Group A
5MxW9w5FYFv6YYkuGrvyc	None	Group B
lmKNOwEEhuftT5UJynZfd	Advanced	Group A
EXdbZQ4iJ8PwWFAe6R4yZ	Beginner	Group A
Dv0EVsye0nYtK2QVlpgxi	Beginner	No noticeable difference
kcBkxUuyJRBiN6WOmai0J	Intermediate	Group B
Rad5wuOSdDIxFmYaTVl1h	None	Group A

YBt4SFbLIM6lL0f3YRLlp	Beginner	Group A
w3arWjykth4V7ddQ5pbx8	None	Group A
TJ0onZi0eZWREXHfaQcJ2	Intermediate	Group A
s3yVIuE5iNdQGV1hHATKF	None	No noticeable difference
K02RzyYWiJipC9hPCw1ah	None	**Group A**
G2R2m7fxvkoo9zPh5s4M0	None	**Group A**
NyyqJolRD4qtc32HpA2RL	Advanced	**Group A**
		Mean rating

ii) Group A Melodies Ratings

(Melody A1)

A1			
Coherent & structured	Random / disjointed	Pleasant to listen to	Overall rating
3	4	4	3
2	4	2	3
5	1	5	4
4	2	3	4
3	2	5	4
2	2	3	3
2	4	4	3
4	2	5	4
2	1	4	5
4	3	3	4
4	2	4	4
3	4	4	3
4	3	5	3
4	2	4	4
3	4	3	3
3.266666667	2.666666667	3.866666667	3.6

(Melody A2)

A2			
Coherent & structured	**Random / disjointed**	**Pleasant to listen to**	**Overall rating**
4	3	3	4
2	3	3	2
4	2	4	3
5	1	5	5
2	3	4	2
2	4	2	1
2	1	5	5
2	2	3	3
2	1	5	4
2	3	4	2
2	2	3	4
2	2	4	3
2	3	4	4
2	1	5	4
2	3	4	4
2.466666667	2.266666667	3.866666667	3.333333333

(Melody A3)

A3			
Coherent & structured	**Random / disjointed**	**Pleasant to listen to**	**Overall rating**
3	2	4	3
3	3	4	3
5	1	5	5
4	2	4	4
2	2	2	3
3	1	4	4
4	2	4	4
4	2	2	3
4	1	5	4
3	4	3	4
3	3	4	3
4	3	3	3
4	3	3	4
4	3	4	3
3	2	4	4
3.533333333	2.266666667	3.666666667	3.6

iii) Group B Melodies Ratings

(Melody B1)

B1			
Coherent & structured	**Random / disjointed**	**Pleasant to listen to**	**Overall rating**
1	4	2	2
1	4	2	2
3	2	3	3
2	4	1	1
3	1	3	3
1	5	1	1
1	5	3	3
4	2	3	3
5	2	5	5
3	3	4	2
2	4	2	2
2	4	3	2
3	3	3	3
3	5	3	3
2	3	2	2
2.4	3.4	2.666666667	2.466666667

(Melody B2)

B2			
Coherent & structured	**Random / disjointed**	**Pleasant to listen to**	**Overall rating**
1	2	2	3
5	4	2	2
5	2	5	4
4	1	4	4
2	2	3	3

2	3	2	3
5	1	5	5
2	4	3	3
4	1	4	5
5	1	4	3
3	4	3	3
3	4	2	1
4	3	3	4
2	4	2	2
2	3	2	2
3.266666667	2.6	3.066666667	3.133333333

(Melody B3)

B3			
Coherent & structured	**Random / disjointed**	**Pleasant to listen to**	**Overall rating**
1	5	2	2
2	3	3	3
5	1	5	5
3	3	3	3
3	2	3	4
2	2	4	2
4	2	5	4
3	2	3	3
4	1	5	5
1	4	2	4
3	2	3	3
1	4	2	1
4	3	4	3
2	4	3	3
1	5	2	1
2.6	2.866666667	3.266666667	3.066666667