Apprentissage profond - 8INF887

Projet - Génération de Mélodies

Sprint 2 - 06/04/25

Héléna Barbillon - BARH30530200
Amandine Lapique - LAPA07570200

Travail réalisé

Amandine

Dans le dernier sprint, j'avais indiqué que j'avais commencé à travailler sur l'implémentation d'un auto-encodeur, mais que j'avais été mise en difficulté par le prétraitement des données. J'ai repris et poursuivi ce travail en me concentrant sur le prétraitement des données.

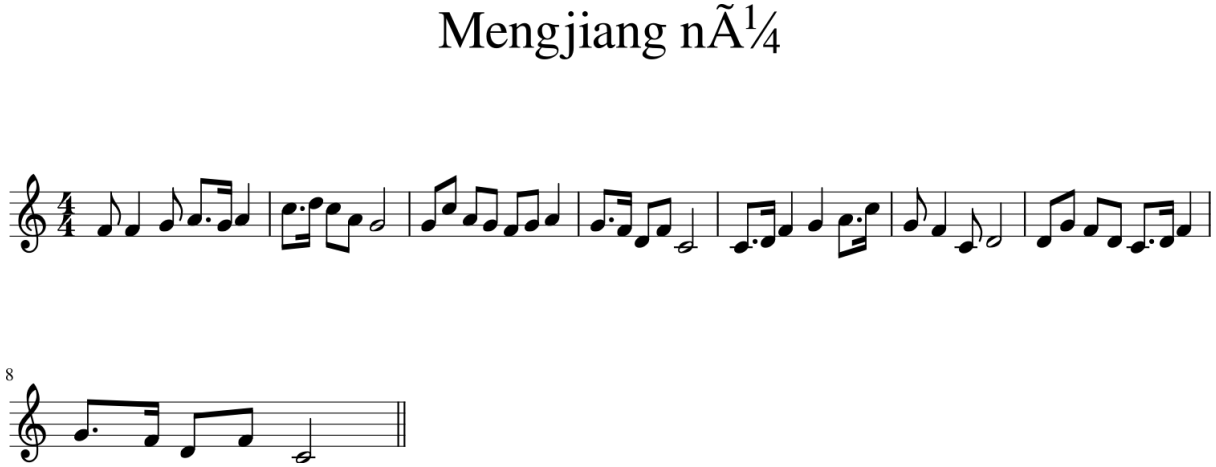
J'ai implémenté une première version d'auto-encodeur dans le notebook "VAE_gen_1" disponible en annexe. Pour cette première version, j'ai transformé les mélodies en une suite de nombres. Je me suis inspirée du travail d'Héléna et j'ai utilisé des 1 pour simuler le rythme des notes. Ainsi, une note tenue est représentée par sa valeur, puis d'une suite de 1 représentant sa durée. J'ai ensuite représenté les silences par des 0. Afin d'utiliser ces séquences dans un auto-encodeur, je les ai normalisées. Cette solution n'était clairement pas optimale, car elle m'a fait perdre beaucoup d'informations.

J'ai tout de même entraîné un modèle d'auto-encodeur basique avec mes données prétraitées. Je me suis inspirée du modèle de GeeksforGeeks disponible à l'adresse : https://www.geeksforgeeks.org/implementing-an-autoencoder-in-pytorch/.

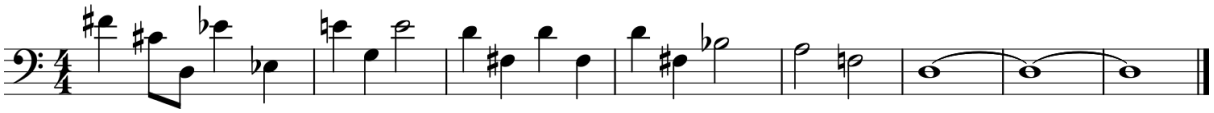
J'ai eu quelques résultats intéressants. Malheureusement, j'ai perdu beaucoup d'informations lors du retour aux notes à cause de la normalisation. Le modèle n'était pas capable d'apprendre que les notes tenues étaient représentées par des 1. J'avais donc des valeurs de notes très basses, autour de 1 ou 2, qui correspondaient certainement à des 1. Pour régler ce problème, j'ai considéré que toutes les valeurs en dessous de 50 étaient des 1. Cela n'était peut-être pas la meilleure solution, mais ça m'a permis d'avancer. J'ai ensuite mis en place un VAE à partir du modèle d'auto-encodeur classique. J'ai obtenu de nouveaux résultats.

Exemple de résultats :

Mélodie original



Mélodie générée



Dans l'ensemble, les résultats sont peu satisfaisants car l'encodage des mélodies n'est pas optimal. On peut observer que les notes sont beaucoup plus basses dans la mélodie générée que dans la mélodie originale. Selon moi, cela est dû au fait que le modèle apprend avec beaucoup de 1 et cela crée un déplacement des notes vers le bas. En parallèle, le fait de considérer que toutes les notes en dessous de 50 sont des 1 fait perdre le modèle en précision, notamment au niveau du rythme.

J'ai donc décidé de recommencer le processus en utilisant un autre encodage des données. L'idée est de créer deux encodeurs, un pour le rythme et un pour les notes. Pour l'instant, je n'ai pas mis en place les modèles, je n'ai effectué que le prétraitement des données, visible dans le notebook "VAE_gen_2". J'ai séparé chaque mélodie en 2 séquences, une représentant les notes et l'autre la longueur des notes. J'ai ensuite paddé les séquences pour qu'elles fassent toutes la même taille. Je pensais réutiliser la normalisation, mais en y réfléchissant, j'ai réalisé que l'utilisation de vecteurs one-hot était plus adaptée à ce problème. J'ai donc mis en place des tenseurs PyTorch. Cela a nécessité quelques modifications sur les séquences, mais aucune de ces modifications n'entraîne une perte d'information. Enfin, j'ai mis les séquences sous forme de one-hot.

La mise en place des encodeurs basiques m'a permis de bien comprendre le fonctionnement de ce type de modèle et d'obtenir des premiers résultats. Avec la nouvelle représentation des données, je suis confiante et je pense pouvoir avoir des résultats plus intéressants. Il sera certainement possible de regrouper des rythmes et des notes générés à partir de deux mélodies différentes, ce qui augmente le nombre de possibilités.

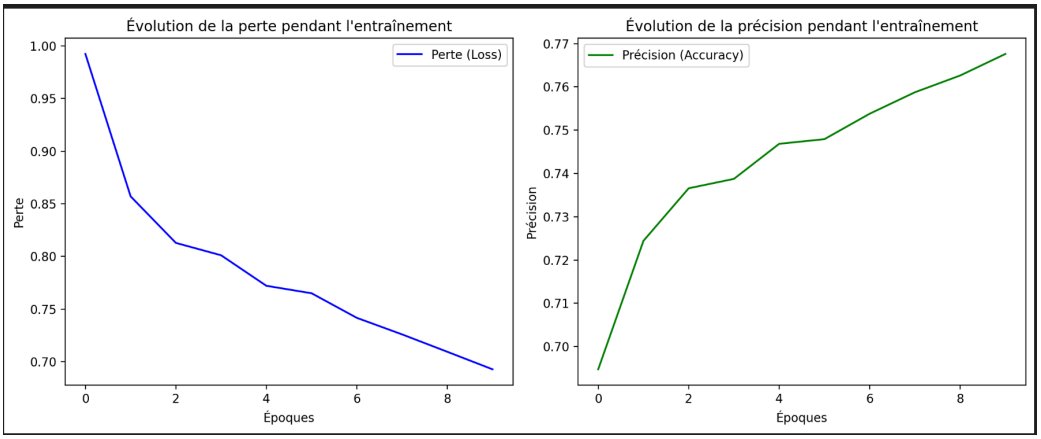
Héléna

J'ai finis de suivre le tutoriel pour générer de la musique avec un RNN-LSTM. L'entraînement du réseau se fait en lui faisant prédire une note qui vient après une séquence de notes. Pour l'entraînement on donne donc en input une partie du morceau, et en output la note suivante. Le modèle est composé d'une couche LSTM, d'une couche de dropout, et enfin une couche dense en output.

```
# create the model architecture
input = keras.layers.Input(shape=(None, output_units))
x = keras.layers.LSTM(num_units[0])(input)
x = keras.layers.Dropout(0.2)(x)

output = keras.layers.Dense(output_units, activation="softmax")(x)

model = keras.Model(input, output)
```

La prédiction de mélodies se fait en donnant le début du morceau, une "seed", et le modèle complète le reste.
J'ai testé plusieurs datasets, de différentes tailles, et différentes seed. Le dataset "han" est composé de 1091 musiques traditionnelles chinoises. Les résultats sont assez satisfaisants, avec seulement une note de départ on obtient des musiques qui paraissent cohérentes, dans le style chinois. (cf fichier *LSTM_han.midi*)

J’ai également fait des tests avec des musiques traditionnelles allemandes (1683 musiques), et les résultats sont aussi satisfaisants. (cf fichier LSTM_deutsch.midi).
Les entraînements ont été faits sur 10 époques, et les courbes de loss et accuracy semblent indiquer qu’on pourrait les lancer sur un nombre plus grand d’époques, ce que j’essaierai pour le prochain sprint.



J’ai essayé de donner en seed la première mesure de “A vous dirais-je maman”, pour voir si le modèle était capable de prédire une suite cohérente. Vous trouverez un résultat dans le fichier “ahvousdiraisjemaman_han.midi” de la suite de cette mélodie par le modèle entraîné sur le dataset chinois, et un autre dans le fichier “ahvousdiraisjemaman_deutsch” du modèle du dataset allemand. On reconnaît la différence entre les deux styles musicaux.

Suite du projet

- Implémentation des modèles d’auto-encodeur avec la nouvelle représentation des données
- Préparation de la présentation orale

Bilan

Nous commençons à avoir nos premiers résultats, comme prévu dans la planification de notre premier sprint. Les résultats du modèle RNN-LSTM sont plutôt bons et les résultats des modèles d’auto-encodeur sont à améliorer. Nous aurons des choses à présenter lors de l’exposé oral et nous sommes confiantes pour la suite.

Annexe : entrainement LSTM

```
import os
import music21 as m21
import json
import numpy as np
import tensorflow.keras as keras
import matplotlib.pyplot as plt



def convert_songs_to_int(songs, mappings):
      # Conversion des datas  74 _ _ _ 69 en integers suivant le mapping

      int_songs = []

      songs = songs.split() # 74 _ _ _ 69 --> ['74', '_', '_', '_',
'69']

      # map songs to int
      for symbol in songs:
      int_songs.append(mappings[symbol])

      return int_songs


#  ----------------------Générer séquences d'entrainement
def generate_training_sequences(sequence_length, songs, mappings):
      """Create input and output data samples for training. Each sample
is a sequence.

      :param sequence_length (int): Length of each sequence. With a
quantisation at 16th notes, 64 notes equates to 4 bars
      :param songs (string) : songs datas
      :param mappings (dict): mapping between

      :return inputs (ndarray): Training inputs
      :return targets (ndarray): Training targets
      """

      # map songs to int
      int_songs = convert_songs_to_int(songs, mappings)

      inputs = [] # fenetre qu'on décalle
      targets = [] # note qu'on cherche à prédire à partir de la
séquence d'input

      # generate the training sequences
```

```
      num_sequences = len(int_songs) - sequence_length
      for i in range(num_sequences):
      inputs.append(int_songs[i:i+sequence_length])
      targets.append(int_songs[i+sequence_length])

      # one-hot encode the sequences
      vocabulary_size = len(set(int_songs))
      # inputs size: (# of sequences, sequence length, vocabulary size)
      inputs = keras.utils.to_categorical(inputs,
num_classes=vocabulary_size)
      targets = np.array(targets)

      return inputs, targets


#  -------------------- Construction du modèle
def build_model(output_units, num_units, loss, learning_rate):
      """Builds and compiles model

      :param output_units (int): Num output units
      :param num_units (list of int): Num of units in hidden layers
      :param loss (str): Type of loss function to use
      :param learning_rate (float): Learning rate to apply

      :return model (tf model)
      """

      # create the model architecture
      input = keras.layers.Input(shape=(None, output_units))
      x = keras.layers.LSTM(num_units[0])(input)
      x = keras.layers.Dropout(0.2)(x)

      output = keras.layers.Dense(output_units, activation="softmax")(x)

      model = keras.Model(input, output)

      # compile model
      model.compile(loss=loss,

optimizer=keras.optimizers.Adam(learning_rate=learning_rate),
                  metrics=["accuracy"])

      model.summary()

      return model
```

```
# -------------------- Génération de mélodies

def sample_with_temperature(probabilites, temperature):
      """Samples an index from a probability array reapplying softmax
using temperature

      :param predictions (nd.array): Array containing probabilities for
each of the possible outputs.
      :param temperature (float): Float in interval [0, 1]. Numbers
closer to 0 make the model more deterministic.
      A number closer to 1 makes the generation more unpredictable.

      :return index (int): Selected output symbol
      """
      predictions = np.log(probabilites) / temperature
      probabilites = np.exp(predictions) / np.sum(np.exp(predictions))

      choices = range(len(probabilites)) # [0, 1, 2, 3]
      index = np.random.choice(choices, p=probabilites)

      return index


def generate_melody(model, start_symbols, mappings, seed, num_steps,
max_sequence_len, temperature):
      """Generates a melody using the DL model and returns a midi file.

      :param seed (str): Melody seed with the notation used to encode
the dataset
      :param num_steps (int): Number of steps to be generated
      :param max_sequence_len (int): Max number of steps in seed to be
considered for generation
      :param temperature (float): Float in interval [0, 1]. Numbers
closer to 0 make the model more deterministic.
            A number closer to 1 makes the generation more
unpredictable.

      :return melody (list of str): List with symbols representing a
melody
      """

      # create seed with start symbols
      seed = seed.split()
      melody = seed
```

```
        seed = start_symbols + seed

        # map seed to int
        seed = [mappings[symbol] for symbol in seed]

        for _ in range(num_steps):
        # limit the seed to max_sequence_length
        seed = seed[-max_sequence_len:]

        # one-hot encode the seed
        onehot_seed = keras.utils.to_categorical(seed,
num_classes=len(mappings))
        # (1, max_sequence_length, num of symbols in the vocabulary)
        onehot_seed = onehot_seed[np.newaxis, ...]

        # make a prediction
        probabilities = model.predict(onehot_seed)[0]
        # [0.1, 0.2, 0.1, 0.6] -> 1
        output_int = sample_with_temperature(probabilities, temperature)

        # update seed
        seed.append(output_int)

        # map int to our encoding
        output_symbol = [k for k, v in mappings.items() if v ==
output_int][0]

        # check whether we're at the end of a melody
        if output_symbol == "/":
                break

        # update melody
        melody.append(output_symbol)

        return melody
```

Annexe : notebooks jupyter

VAE_gen_1

Génération de mélodies avec un VAE vanille vanillé

Génération avec un autoencoder simple

J'ai récupéré le modèle d'autoencoder vanille pour mnist de Geeksforgeeks

https://www.geeksforgeeks.org/implementing-an-autoencoder-in-pytorch/

Préparation des données

Dans ce notebook le pré-traitement des données est basique et pas très réfléchit. Il consiste à transformer les notes en une liste de nombre. Si la note est tenue, la durée est représenté par un ensemble de 1. De plus, les silence sont représetné avec des 0.

La liste de nombre est ensuite normalisée et ramenée entre 0 et 1. Les modèles sont ensuite entrainées à partir de cette liste.

Cela pose un problème lors du décodage. En effet, des notes très basses aparaissent. Cela est dû au faite que le modèle n'apprend pas que le 0.01 est une valeur spécifique correspondant à une note tenue. Pour régler ce problème, lors du décodage j'ai indiqué que toutes les notes en dessous de 0.5 sont des 0.1. Cela est problématique car ça ne respecte pas vraiment la logique de l'encodage. En revanche, cela est une bonne solution en attendant de trouver une meilleure solution d'encodage.

Dans le notebook, VAE_gen_2 je propose une nouvelle version de l'encodage qui est plus adaptée à notre problématique.

In [1]:

```
## Lecture des fichiers de musique

import os
import music21 as m21

def load_songs(data_path, max_songs_nb):
    songs = []
    for path, subdirs, files in os.walk(data_path):
        for file in files:
            if file[-3:] == "krn":
                #print(os.path.join(path, file))
                song = m21.converter.parse(os.path.join(path, file))
                songs.append(song)
                max_songs_nb -= 1
                if max_songs_nb == 0 : return songs
    return songs

DATASET_PATH = "data/han"

print("loading data...")
songs = load_songs(DATASET_PATH, 1000)
print("songs loaded")
```

```
loading data...
songs loaded
```

In [2]:

```
## Prétraitement des données

# Durée des notes
ACCEPTABLE_DURATIONS = [
    0.25, # 16th note
    0.5, # 8th note
    0.75,
    1.0, # quarter note
    1.5,
    2, # half note
    3,
    4 # whole note
]

# Vérification de la durée des notes et suppression des éléments non conformes
def has_acceptable_durations(song, acceptable_durations):
    for note in song.flatten().notesAndRests:
        if note.duration.quarterLength not in acceptable_durations:
```

```
            return False
    return True

for song in songs:
    if not has_acceptable_durations(song, ACCEPTABLE_DURATIONS):
        # song.show()
        # song.show("midi")
        songs.remove(song)

# Transposition en do majeur
def transpose(song, print_enabled=False):
    # transpose song in Cmaj/Amin

    # get key signature
    parts = song.getElementsByClass(m21.stream.Part)
    measures_part0 = parts[0].getElementsByClass(m21.stream.Measure)
    key = measures_part0[0][4]
    if print_enabled : print("old key : ", key)

    # estimate key if not indicated
    if not isinstance(key, m21.key.Key):
        key = song.analyze("key")

    # get interval for transposition
    if key.mode == "major":
        interval = m21.interval.Interval(key.tonic, m21.pitch.Pitch("C"))
    elif key.mode == "minor":
        interval = m21.interval.Interval(key.tonic, m21.pitch.Pitch("A"))

    transposed_song = song.transpose(interval)

    return transposed_song

transposed_songs = []
for song in songs:
    transposed_songs.append(transpose(song))
```

In [3]:

```
## Encodage des données

"""
- Chaque note est représentée par un entier
- La durée d'une note est représentée par des 1 qui la suive
- Les silences sont représentées par l'entier 0
"""

def encode_song(song, time_step=0.25):
    encoded_song = []

    for event in song.flat.notesAndRests:
        if isinstance(event, m21.note.Note):
            symbol = event.pitch.midi
        elif isinstance(event, m21.note.Rest):
            symbol = 0

        steps = int(event.duration.quarterLength / time_step)
        for step in range(steps):
            if step == 0:
                encoded_song.append(symbol)
            else:
                encoded_song.append(1)

    return encoded_song

# Exemple
song = transposed_songs[0]
encoded_song = encode_song(song)
print(encoded_song)

## Création des séquences
encoded_songs = [encode_song(song) for song in transposed_songs]
```

```
/home/am/UQAC/AP/projet/projet-env/lib/python3.12/site-packages/music21/stream/base.py:3675: Music21Deprec
ationWarning: .flat is deprecated.  Call .flatten() instead
  return self.iter().getElementsByClass(classFilterList)
```

```
[69, 69, 1, 1, 72, 1, 69, 1, 67, 1, 64, 67, 69, 1, 0, 1, 74, 1, 74, 1, 72, 1, 69, 1, 67, 1, 64, 67, 69, 1,
0, 1, 69, 1, 69, 1, 72, 1, 67, 1, 69, 1, 0, 1, 74, 1, 74, 1, 72, 1, 67, 1, 69, 1, 0, 1, 69, 1, 69, 1, 72, 1
, 69, 67, 64, 1, 0, 1, 74, 1, 74, 1, 72, 1, 69, 67, 64, 1, 0, 1, 76, 1, 76, 1, 74, 1, 76, 1, 72, 1, 1, 1, 7
9, 1, 76, 1, 74, 1, 1, 1, 72, 1, 69, 67, 69, 1, 0, 1, 72, 1, 1, 1, 69, 1, 69, 1, 67, 1, 64, 67, 69, 1, 0, 1
, 72, 1, 69, 69, 67, 1, 64, 67, 69, 1, 1, 1]
```

In [4]:

```
## Padding
# Toutes les données doivent avoir la même longueur
# On ajoute des 0 à la fin des séquences

# affiche la longueur maximale des séquences
max_length = max([len(song) for song in encoded_songs])
print("max length : ", max_length)
mean_length = int(sum(map(len, encoded_songs)) / len(encoded_songs))
print("mean length : ", mean_length)
```

```
max length :  960
mean length :  168
```

In [20]:

```
# On fait des séquences de 256 éléments

def padding(song, sequence_length=128):
    if len(song) >= sequence_length:
        return song[:sequence_length]
    input_sequence = song.copy()
    for i in range(sequence_length - len(input_sequence)):
        input_sequence.append(0)
    return input_sequence

input_sequence = padding(encoded_songs[0])
print(input_sequence)

padded_songs = [padding(song) for song in encoded_songs]
```

```
[69, 69, 1, 1, 72, 1, 69, 1, 67, 1, 64, 67, 69, 1, 0, 1, 74, 1, 74, 1, 72, 1, 69, 1, 67, 1, 64, 67, 69, 1,
0, 1, 69, 1, 69, 1, 72, 1, 67, 1, 69, 1, 0, 1, 74, 1, 74, 1, 72, 1, 67, 1, 69, 1, 0, 1, 69, 1, 69, 1, 72, 1
, 69, 67, 64, 1, 0, 1, 74, 1, 74, 1, 72, 1, 69, 67, 64, 1, 0, 1, 76, 1, 76, 1, 74, 1, 76, 1, 72, 1, 1, 1, 7
9, 1, 76, 1, 74, 1, 1, 1, 72, 1, 69, 67, 69, 1, 0, 1, 72, 1, 1, 1, 69, 1, 69, 1, 67, 1, 64, 67, 69, 1, 0, 1
, 72, 1, 69, 69]
```

In [30]:

```
# Moyenne des sequence
mean_length = int(sum(map(len, padded_songs)) / len(padded_songs))
print("mean length : ", mean_length)
```

```
mean length :  128
```

In [31]:

```
# Valeur maximale des notes dans les sequences
max_note = max([max(song) for song in padded_songs])
print("max note : ", max_note)

# Valeur minimale des notes dans les sequences autre que 0 et 1
min_note = min([min(filter(lambda x: x > 1, song)) for song in padded_songs])
print("min note : ", min_note)
```

```
max note :  98
min note :  50
```

In [32]:

```
## Normalisation
# On normalise les données entre 0 et 1

def normalize(song):
    return [note / 100 for note in song]

normalized_songs = [normalize(song) for song in padded_songs]
print(normalized_songs[0])
```

```
[0.69, 0.69, 0.01, 0.01, 0.72, 0.01, 0.69, 0.01, 0.67, 0.01, 0.64, 0.67, 0.69, 0.01, 0.0, 0.01, 0.74, 0.0
1, 0.74, 0.01, 0.72, 0.01, 0.69, 0.01, 0.67, 0.01, 0.64, 0.67, 0.69, 0.01, 0.0, 0.01, 0.69, 0.01, 0.69, 0
.01, 0.72, 0.01, 0.67, 0.01, 0.69, 0.01, 0.0, 0.01, 0.74, 0.01, 0.74, 0.01, 0.72, 0.01, 0.67, 0.01, 0.69,
0.01, 0.0, 0.01, 0.69, 0.01, 0.69, 0.01, 0.72, 0.01, 0.69, 0.67, 0.64, 0.01, 0.0, 0.01, 0.74, 0.01, 0.74,
0.01, 0.72, 0.01, 0.69, 0.67, 0.64, 0.01, 0.0, 0.01, 0.76, 0.01, 0.76, 0.01, 0.74, 0.01, 0.76, 0.01, 0.72
, 0.01, 0.01, 0.01, 0.79, 0.01, 0.76, 0.01, 0.74, 0.01, 0.01, 0.01, 0.72, 0.01, 0.69, 0.67, 0.69, 0.01, 0
.0, 0.01, 0.72, 0.01, 0.01, 0.01, 0.69, 0.01, 0.69, 0.01, 0.67, 0.01, 0.64, 0.67, 0.69, 0.01, 0.0, 0.01,
0.72, 0.01, 0.69, 0.69]
```

In [33]:

```
## Création des données d'entraînement
```

```
def create_input_sequence(song, sequence_length=128):
    if has_acceptable_durations(song, ACCEPTABLE_DURATIONS):
            transposed_song = transpose(song)
            encoded_song = encode_song(transposed_song)
            padded_song = padding(encoded_song, sequence_length)
            input_sequence = normalize(padded_song)
            return input_sequence
    else :
        return None


def create_input_sequences(songs, sequence_length=128):
    input_sequences = []
    for song in songs:
        input_sequence = create_input_sequence(song, sequence_length)
        if input_sequence:
            input_sequences.append(input_sequence)
    return input_sequences

input_sequences = create_input_sequences(songs)
print(input_sequences[0])
```

```
/home/am/UQAC/AP/projet/projet-env/lib/python3.12/site-packages/music21/stream/base.py:3675: Music21Deprec
ationWarning: .flat is deprecated.  Call .flatten() instead
  return self.iter().getElementsByClass(classFilterList)
```

```
[0.69, 0.69, 0.01, 0.01, 0.72, 0.01, 0.69, 0.01, 0.67, 0.01, 0.64, 0.67, 0.69, 0.01, 0.0, 0.01, 0.74, 0.0
1, 0.74, 0.01, 0.72, 0.01, 0.69, 0.01, 0.67, 0.01, 0.64, 0.67, 0.69, 0.01, 0.0, 0.01, 0.69, 0.01, 0.69, 0
.01, 0.72, 0.01, 0.67, 0.01, 0.69, 0.01, 0.0, 0.01, 0.74, 0.01, 0.74, 0.01, 0.72, 0.01, 0.67, 0.01, 0.69,
0.01, 0.0, 0.01, 0.69, 0.01, 0.69, 0.01, 0.72, 0.01, 0.69, 0.67, 0.64, 0.01, 0.0, 0.01, 0.74, 0.01, 0.74,
0.01, 0.72, 0.01, 0.69, 0.67, 0.64, 0.01, 0.0, 0.01, 0.76, 0.01, 0.76, 0.01, 0.74, 0.01, 0.76, 0.01, 0.72
, 0.01, 0.01, 0.01, 0.79, 0.01, 0.76, 0.01, 0.74, 0.01, 0.01, 0.01, 0.72, 0.01, 0.69, 0.67, 0.69, 0.01, 0
.0, 0.01, 0.72, 0.01, 0.01, 0.01, 0.69, 0.01, 0.69, 0.01, 0.67, 0.01, 0.64, 0.67, 0.69, 0.01, 0.0, 0.01,
0.72, 0.01, 0.69, 0.69]
```

In [34]:

```
# Conversion en tenseurs
import torch
input_sequences = torch.tensor(input_sequences, dtype=torch.float32)
```

Mise en place d'un modèle d'autoencoder simple

Le modèle d'entoencoder est un modèle classique utilisé normalement pour mnist. Dans ce notebook je n'ai pas amélioré le modèle car le but était de comprendre le fonctionnement et d'analyser des premier résultats avec un autoencoder basique.

In [39]:

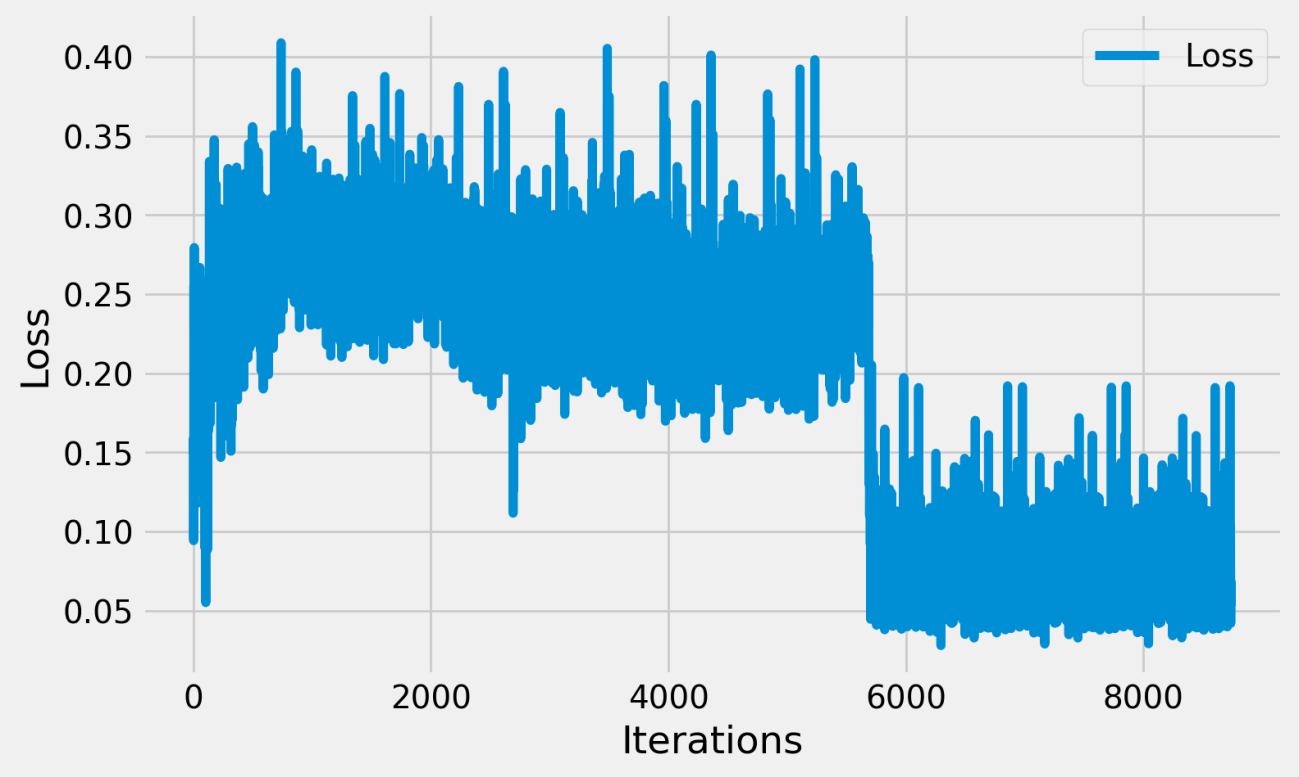
```
## Autoencoder simple
# On a simplement changé la taille de l'entrée qui correspond à la taille des séquences

import torch
from torch import nn, optim
import matplotlib.pyplot as plt

class AE(nn.Module):
    def __init__(self):
        super(AE, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, 36),
            nn.ReLU(),
            nn.Linear(36, 18),
            nn.ReLU(),
            nn.Linear(18, 9)
        )
        self.decoder = nn.Sequential(
            nn.Linear(9, 18),
            nn.ReLU(),
            nn.Linear(18, 36),
            nn.ReLU(),
            nn.Linear(36, 64),
            nn.ReLU(),
            nn.Linear(64, 128),
            nn.Sigmoid()
        )

    def forward(self, x):
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
```

```
        return decoded

model = AE()
loss_function = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.1, weight_decay=1e-5)
```

In [40]:

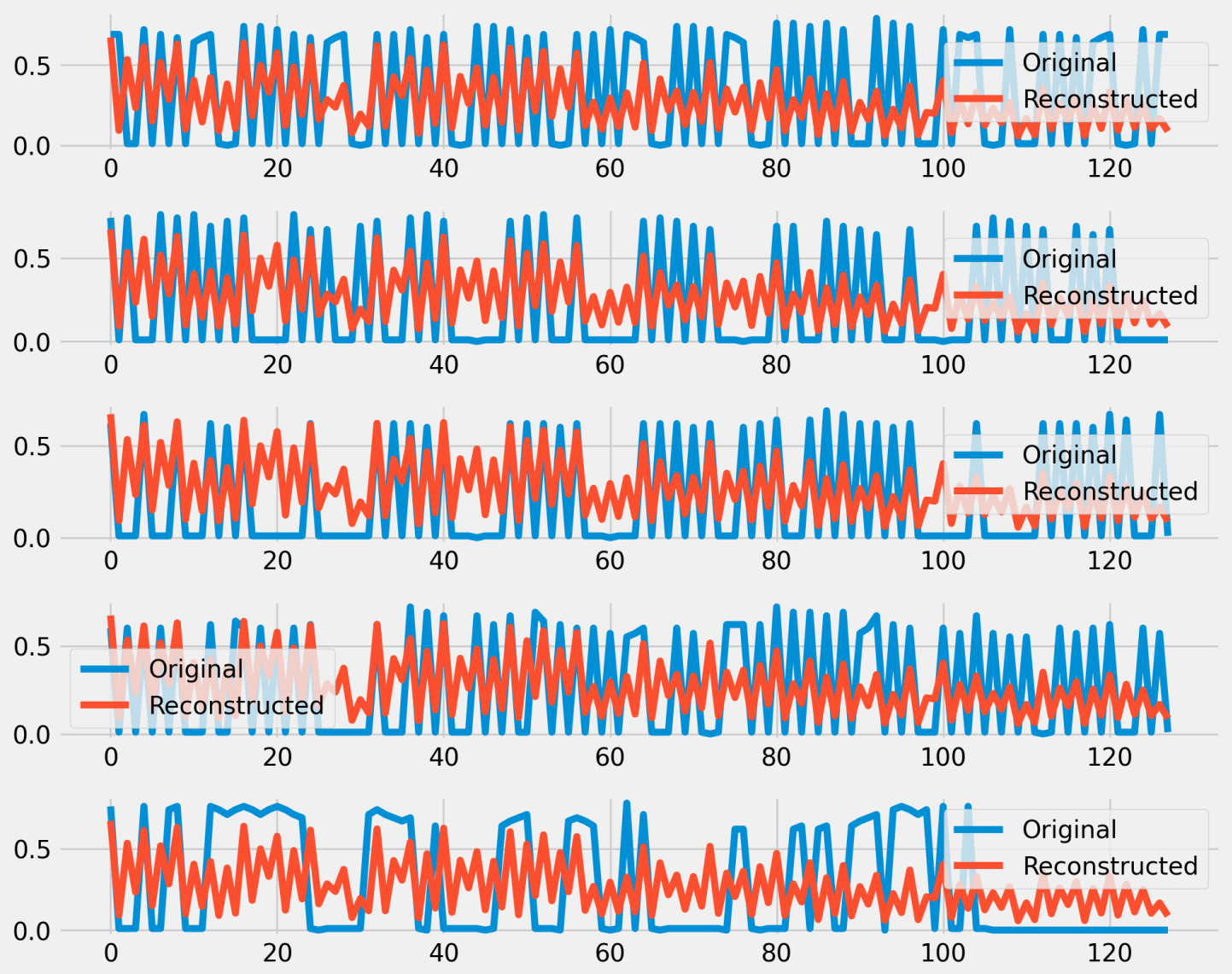
```
# Entraînement

epochs = 10
outputs = []
losses = []

device = torch.device("cpu")
model.to(device)

for epoch in range(epochs):
    for sequences in input_sequences:
        sequences = sequences.view(-1, 128).to(device)

        reconstructed = model(sequences)
        loss = loss_function(reconstructed, sequences)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        losses.append(loss.item())

    outputs.append((epoch, sequences, reconstructed))
    print(f"Epoch {epoch+1}/{epochs}, Loss: {loss.item():.6f}")

plt.style.use('fivethirtyeight')
plt.figure(figsize=(8, 5))
plt.plot(losses, label='Loss')
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
Epoch 1/10, Loss: 0.271937
Epoch 2/10, Loss: 0.237018
Epoch 3/10, Loss: 0.219362
Epoch 4/10, Loss: 0.224362
Epoch 5/10, Loss: 0.211705
Epoch 6/10, Loss: 0.209049
Epoch 7/10, Loss: 0.055269
Epoch 8/10, Loss: 0.053343
Epoch 9/10, Loss: 0.052826
Epoch 10/10, Loss: 0.052762
```



In [41]:

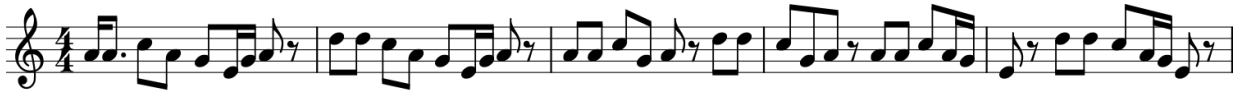
```
# Visualisation des résultats

def plot_reconstructed_sequence(model, sequences):
    reconstructed = model(sequences)
    sequences = sequences.view(-1, 128).detach().numpy()
    reconstructed = reconstructed.view(-1, 128).detach().numpy()

    plt.figure(figsize=(10, 8))
    for i in range(5):
        plt.subplot(5, 1, i+1)
        plt.plot(sequences[i], label='Original')
        plt.plot(reconstructed[i], label='Reconstructed')
        plt.legend()
    plt.tight_layout()
    plt.show()

plot_reconstructed_sequence(model, input_sequences[:5])
```



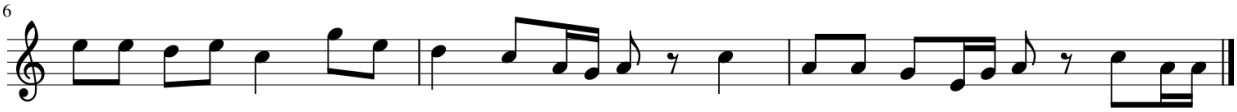
In [42]:

```
# Fonction pour afficher les sequences sous forme de notes

def decode_song(encoded_song, time_step=0.25):
    from music21 import stream, note

    decoded_song = stream.Stream()
    i = 0

    while i < len(encoded_song):
        symbol = int(encoded_song[i] * 100)  # Remultiplier et arrondir


        # Déterminer la durée de l'événement
        duration = time_step
        while i + 1 < len(encoded_song) and encoded_song[i + 1] <= 0.5 and encoded_song[i + 1] != 0 :
            duration += time_step
            i += 1  # Avancer tant qu'on a des "1" (sustain)
```

```
        # Ajouter une note ou un silence
        if symbol == 0:
            m21_event = note.Rest(quarterLength=duration)
        else:
            m21_event = note.Note(midi=symbol, quarterLength=duration)

        decoded_song.append(m21_event)
        i += 1  # Passer à l'événement suivant

    # Supprimer les silences en trop
    for i in range(len(decoded_song) - 1, 0, -1):
        if decoded_song[i].isRest and decoded_song[i - 1].isRest:
            decoded_song.pop(i)

    return decoded_song


# Exemple
decoded_song = decode_song(input_sequences[0])
decoded_song.show()
```



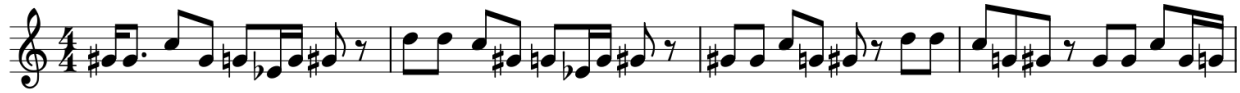
6



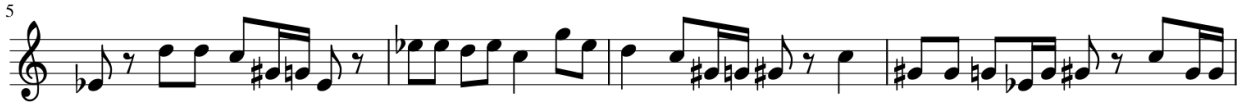
In [43]:

```
# Visualisation des résultats avec les notes

def plot_reconstructed_song(model, sequences, time_step=0.25, max_songs=2):
    reconstructed = model(sequences)
    sequences = sequences.view(-1, 256).detach().numpy()
    reconstructed = reconstructed.view(-1, 256).detach().numpy()

    for i in range(max_songs):
        original_song = decode_song(sequences[i], time_step)
        reconstructed_song = decode_song(reconstructed[i], time_step)

        print("Original")
        original_song.show()
        print("Reconstructed")
        reconstructed_song.show()

plot_reconstructed_song(model, input_sequences, max_songs=2)
```

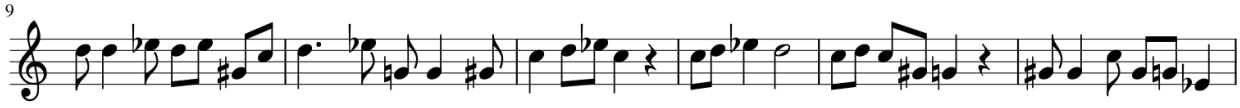
Original



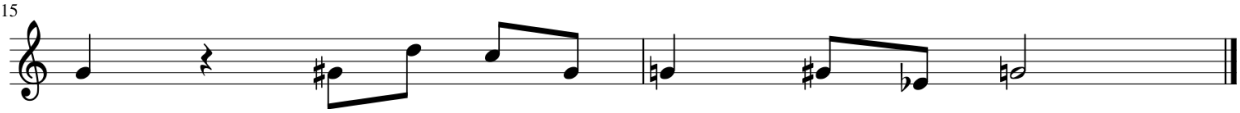
5



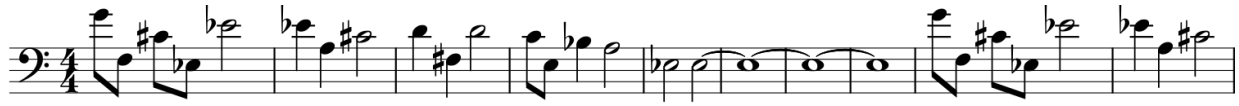
9

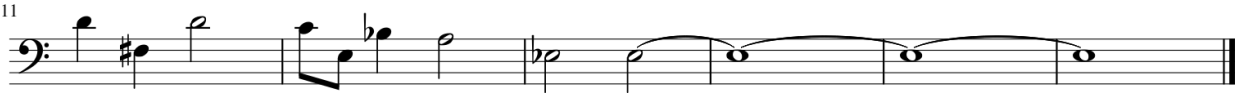


15

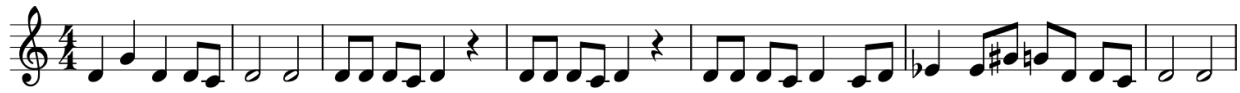


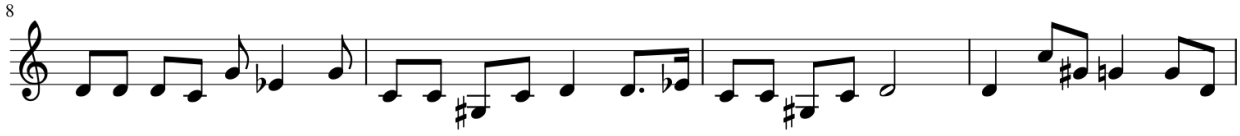
Reconstructed

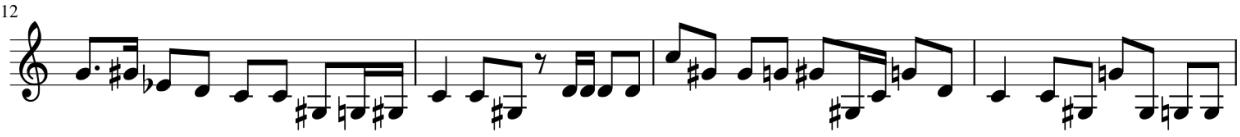


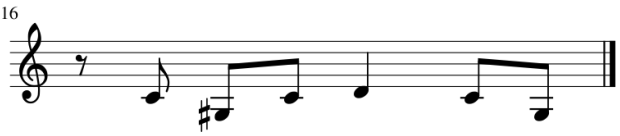


Original

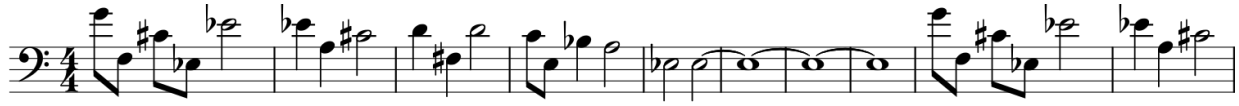


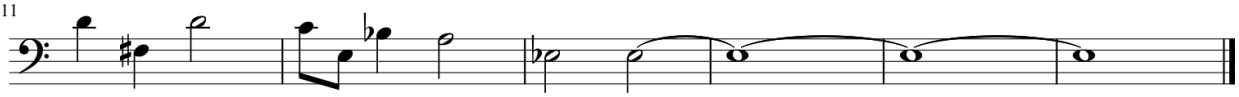






Reconstructed



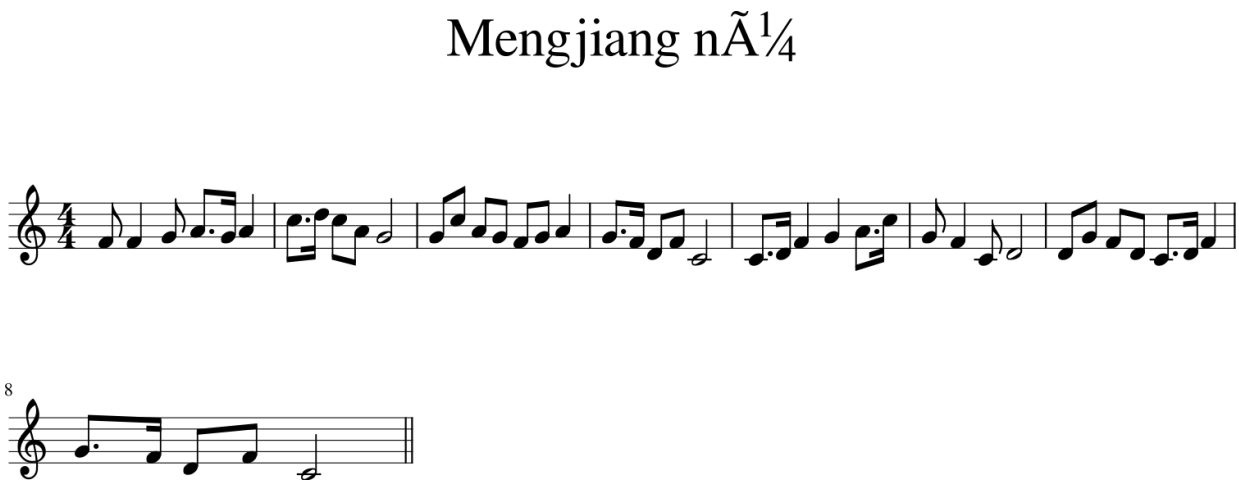


In [63]:

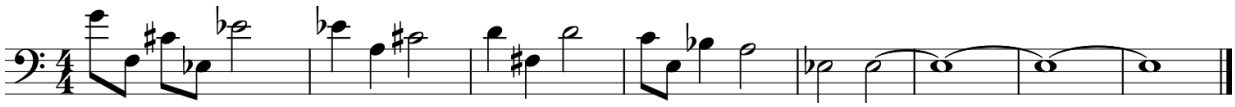
```
# Génération de musique

def generate_song(model, song, sequence_length=128, time_step=0.25):
    model.eval()
    input_song = create_input_sequence(song, sequence_length)
    if not input_song:
        print("Song has unacceptable duration. Can't generate song.")
        return None
    input_song = torch.tensor(input_song, dtype=torch.float32).view(1, -1)
    input_song = input_song.to(device)

    # generate song
    new_song = model(input_song)
    new_song = new_song.view(-1, 128).detach().numpy()
    new_song = new_song[0]

    # decode song
    new_song = decode_song(new_song, time_step)
    return new_song

# Exemple
```

```
song_path = "data/han/han0407.krn"
song = m21.converter.parse(song_path)
print("Original song")
song.show()
song.show("midi")

generated_song = generate_song(model, song)
print("Generated song")
generated_song.show()
generated_song.show("midi")
generated_song.write("midi", "ae1_0407.mid")
```

```
Original song
```




```
Generated song
```



```
Out[63]:
'ae1_0407.mid'
```

VAE vanille

Modèle créé à partir du modèle d'autoencoder précédent avec la même préparation des données

```
In [54]:
```

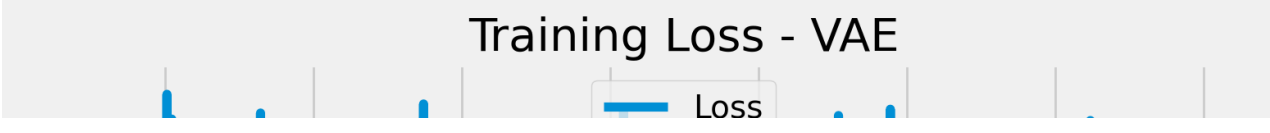
```
import torch
from torch import nn, optim

class VAE(nn.Module):
    def __init__(self):
        super(VAE, self).__init__()
        # Encoder
        self.encoder = nn.Sequential(
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, 36),
            nn.ReLU(),
            nn.Linear(36, 18),
            nn.ReLU()
        )
        self.fc_mu = nn.Linear(18, 9)
        self.fc_logvar = nn.Linear(18, 9)

        # Decoder
        self.decoder = nn.Sequential(
            nn.Linear(9, 18),
            nn.ReLU(),
            nn.Linear(18, 36),
            nn.ReLU(),
            nn.Linear(36, 64),
            nn.ReLU(),
            nn.Linear(64, 128),
            nn.Sigmoid()
        )

    def reparameterize(self, mu, logvar):
        std = torch.exp(0.5 * logvar)
```

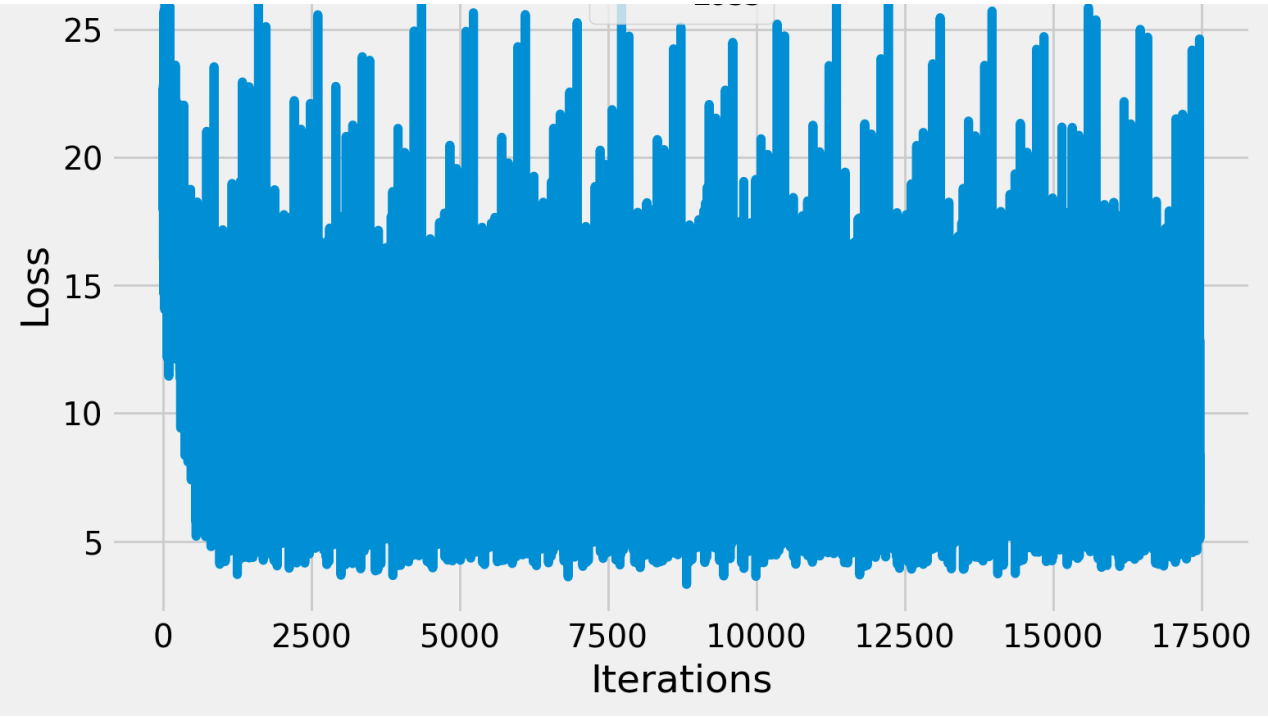
```
        eps = torch.randn_like(std)  # bruit gaussien
        return mu + eps * std

    def forward(self, x):
        x = self.encoder(x)
        mu = self.fc_mu(x)
        logvar = self.fc_logvar(x)
        z = self.reparameterize(mu, logvar)
        reconstructed = self.decoder(z)
        return reconstructed, mu, logvar

def vae_loss(reconstructed, original, mu, logvar):
    reconstruction_loss = nn.functional.mse_loss(reconstructed, original, reduction='sum')
    # KL divergence
    kl_div = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
    return reconstruction_loss + kl_div

# Initialisation du modèle et des hyperparamètres
model_vae = VAE()
optimizer = optim.Adam(model_vae.parameters(), lr=0.0001, weight_decay=1e-5)
```

In [55]:

```
epochs = 20
outputs = []
losses = []

device = torch.device("cpu")  # ou "cuda" si tu as un GPU
model_vae.to(device)

for epoch in range(epochs):
    for sequences in input_sequences:
        sequences = sequences.view(-1, 128).to(device)

        reconstructed, mu, logvar = model_vae(sequences)
        loss = vae_loss(reconstructed, sequences, mu, logvar)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        losses.append(loss.item())

    outputs.append((epoch, sequences.detach(), reconstructed.detach()))
    print(f"Epoch {epoch+1}/{epochs}, Loss: {loss.item():.6f}")

# Affichage de la courbe des pertes
import matplotlib.pyplot as plt

plt.style.use('fivethirtyeight')
plt.figure(figsize=(8, 5))
plt.plot(losses, label='Loss')
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.legend()
plt.title("Training Loss - VAE")
plt.show()
```

```
Epoch 1/20, Loss: 6.759548
Epoch 2/20, Loss: 6.679450
Epoch 3/20, Loss: 6.130944
Epoch 4/20, Loss: 6.152315
Epoch 5/20, Loss: 6.698268
Epoch 6/20, Loss: 6.271441
Epoch 7/20, Loss: 6.273901
Epoch 8/20, Loss: 6.257823
Epoch 9/20, Loss: 6.373527
Epoch 10/20, Loss: 6.522851
Epoch 11/20, Loss: 6.156287
Epoch 12/20, Loss: 6.244071
Epoch 13/20, Loss: 6.495395
Epoch 14/20, Loss: 6.314327
Epoch 15/20, Loss: 6.940214
Epoch 16/20, Loss: 6.205518
Epoch 17/20, Loss: 6.432604
Epoch 18/20, Loss: 6.389891
Epoch 19/20, Loss: 6.427300
Epoch 20/20, Loss: 6.907916
```

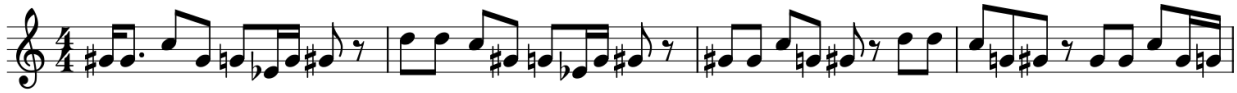




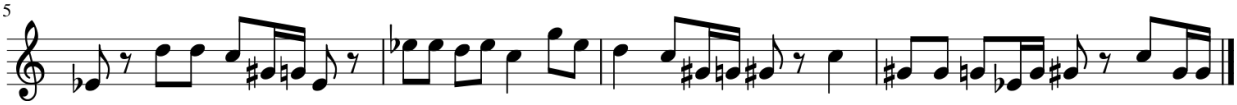
In [56]:

```
def plot_reconstructed_song(model, sequences, time_step=0.25, max_songs=2):
    model.eval()
    with torch.no_grad():
        reconstructed, _, _ = model(sequences)

    sequences = sequences.view(-1, 128).detach().numpy()
    reconstructed = reconstructed.view(-1, 128).detach().numpy()

    for i in range(min(max_songs, len(sequences))):
        original_song = decode_song(sequences[i], time_step)
        reconstructed_song = decode_song(reconstructed[i], time_step)
        print("Original")
        original_song.show()
        print("Reconstructed")
        reconstructed_song.show()

plot_reconstructed_song(model_vae, input_sequences)
```

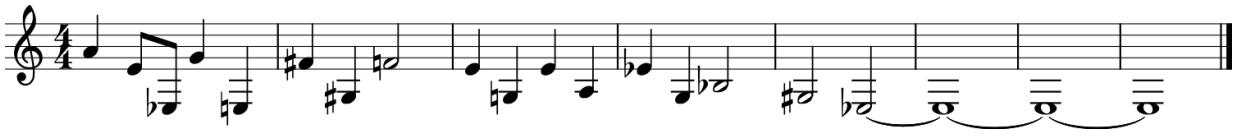
Original



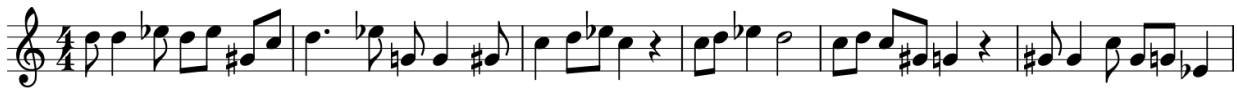
5



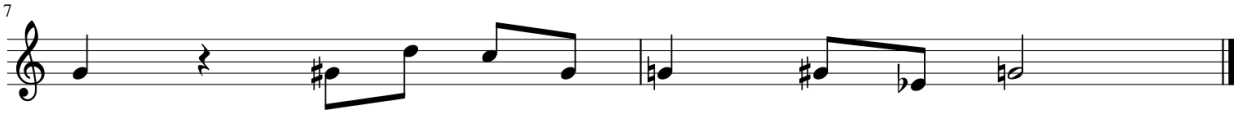
Reconstructed



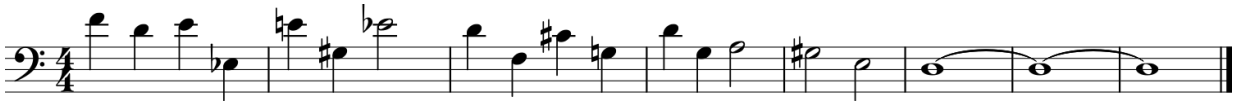
Original



7



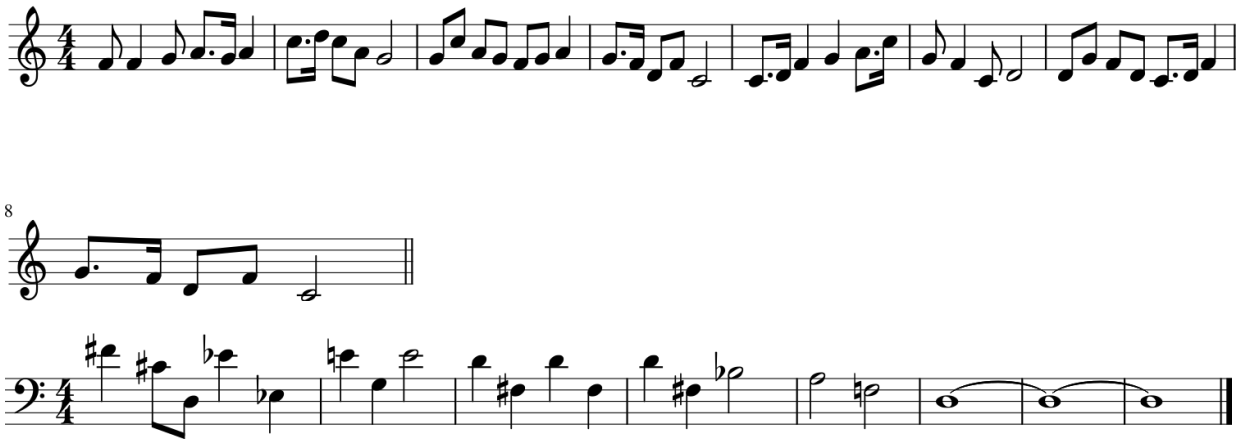
Reconstructed



In [62]:

```
# Génération de musique

def generate_song(model, song, sequence_length=128, time_step=0.25):
    model.eval()
    input_song = create_input_sequence(song, sequence_length)
    if not input_song:
        print("Song has unacceptable duration. Can't generate song.")
        return None

    input_song = torch.tensor(input_song, dtype=torch.float32).view(1, -1).to(device)

    # Reconstruction via VAE (on ignore mu et logvar ici)
    with torch.no_grad():
        reconstructed, _, _ = model(input_song)

    reconstructed = reconstructed.view(-1, 128).cpu().numpy()[0]

    # Decode the reconstructed song
    new_song = decode_song(reconstructed, time_step)
    return new_song


# Exemple
song_path = "data/han/han0407.krn"
song = m21.converter.parse(song_path)
song.show()
song.show("midi")

generated_song = generate_song(model_vae, song)
generated_song.show()
generated_song.show("midi")
generated_song.write("midi", "vae1_0407.mid")
```

Mengjiang nÃ¼



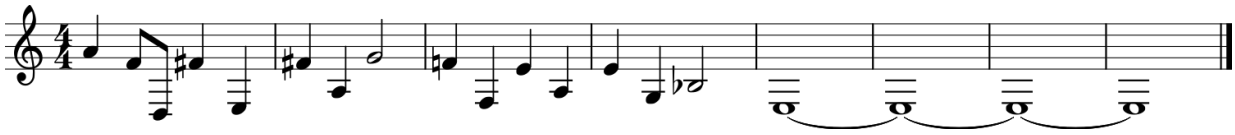
Out[62]:

```
'vae1_0407.mid'
```

In [60]:

```
def generate_random_song(model, time_step=0.25):
    model.eval()
    with torch.no_grad():
        # Tirage aléatoire dans l'espace latent (taille 9 dans ton cas)
        z = torch.randn(1, 9).to(device)
        reconstructed = model.decoder(z)
        reconstructed = reconstructed.view(-1, 128).cpu().numpy()[0]

        new_song = decode_song(reconstructed, time_step)
        return new_song

random_song = generate_random_song(model_vae)
random_song.show()
```

```
random_song.show("midi")
```



In [ ]:

```
```

VAE_gen_2

Génération de mélodies avec un VAE avec amélioration du traitement des données

Prétraitement des données

Dans ce notebook les données musicales vont être représentées par deux séquences de nombres. Une séquence représente les notes et l'autre le rythme. Ces séquences seront ensuite normalisées pour être utilisées pour entrainer deux modèles d'autoencodeurs différents. Ainsi, nous allons créé un modèle d'autoencodeur pour le rythme et un pour les notes.

Cette méthode nous permet de supprimer le problèmes que nous rencontrions précédemment avec les 1 pour former le rythme. De plus, étant donné que le rythme et le choix des notes sont des actions différentes nous espérons avoir de meilleurs résultats.

Initialement, nous avions pensé faire deux séquences et les concaténer pour entrainer un seul modèle mais nous avons changé d'avis. Nous pensons que faire deux modèles pour ces deux actions est plus intéressant. En fonction du temps que nous avons et des résultats obtenus nous allons peut être également entrainer un modèle de ce type.

In [1]:

```
## Lecture des fichiers de musique

import os
import music21 as m21

def load_songs(data_path, max_songs_nb):
    songs = []
    for path, subdirs, files in os.walk(data_path):
        for file in files:
            if file[-3:] == "krn":
                #print(os.path.join(path, file))
                song = m21.converter.parse(os.path.join(path, file))
                songs.append(song)
                max_songs_nb -= 1
                if max_songs_nb == 0 : return songs
    return songs

DATASET_PATH = "data/han"

print("loading data...")
songs = load_songs(DATASET_PATH, 1000)
print("songs loaded")
```

```
loading data...
songs loaded
```

In [62]:

```
## Prétraitement des données

# Durée des notes
ACCEPTABLE_DURATIONS = [
    0.25, # 16th note
    0.5, # 8th note
    0.75,
    1.0, # quarter note
    1.5,
    2, # half note
    3,
    4 # whole note
]

# Vérification de la durée des notes et suppression des éléments non conformes
def has_acceptable_durations(song, acceptable_durations):
    for note in song.flatten().notesAndRests:
        if note.duration.quarterLength not in acceptable_durations:
            return False
    return True

for song in songs:
    if not has_acceptable_durations(song, ACCEPTABLE_DURATIONS):
        # song.show()
```

```
        # song.show("midi")
        songs.remove(song)

# Transposition en do majeur
def transpose(song, print_enabled=False):
    # transpose song in Cmaj/Amin

    # get key signature
    parts = song.getElementsByClass(m21.stream.Part)
    measures_part0 = parts[0].getElementsByClass(m21.stream.Measure)
    key = measures_part0[0][4]
    if print_enabled : print("old key : ", key)

    # estimate key if not indicated
    if not isinstance(key, m21.key.Key):
        key = song.analyze("key")

    # get interval for transposition
    if key.mode == "major":
        interval = m21.interval.Interval(key.tonic, m21.pitch.Pitch("C"))
    elif key.mode == "minor":
        interval = m21.interval.Interval(key.tonic, m21.pitch.Pitch("A"))

    transposed_song = song.transpose(interval)

    return transposed_song

transposed_songs = []
for song in songs:
    transposed_songs.append(transpose(song))
```

In [63]:

```
# Information sur les notes
# Note maximale et minimal
max = 0
min = 1000
for song in transposed_songs:
    for note in song.flatten().notes:
        if note.pitch.midi > max: max = note.pitch.midi
        if note.pitch.midi < min: min = note.pitch.midi

print("max note : ", m21.pitch.Pitch(max).nameWithOctave)
print("min note : ", m21.pitch.Pitch(min).nameWithOctave)
# Affichage en nombre
print("max note : ", max)
print("min note : ", min)
```

```
max note :  D7
min note :  D3
max note :  98
min note :  50
```

Les notes sont comprises entre 50 et 98, ainsi on peut forcer les silences à être soit en dessous de 50 soit au dessus de 98.

J'ai décidé de représenté les silences par la valeurs 99. Ainsi, nous avons 50 valeurs de notes.

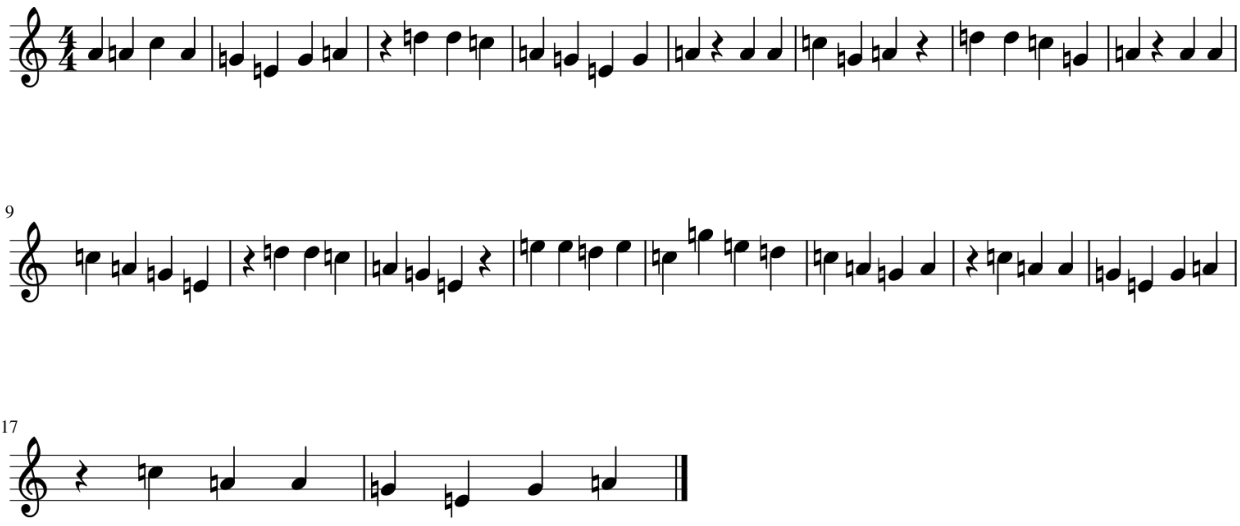
On pourrait pensé que cela n'est valable que pour ce dataset mais en réalité il est assez rare de monter au dessus de D7. Bien sur cela reste à vérifier si on veut réutiliser le modèle sur un autre dataset.

In [18]:

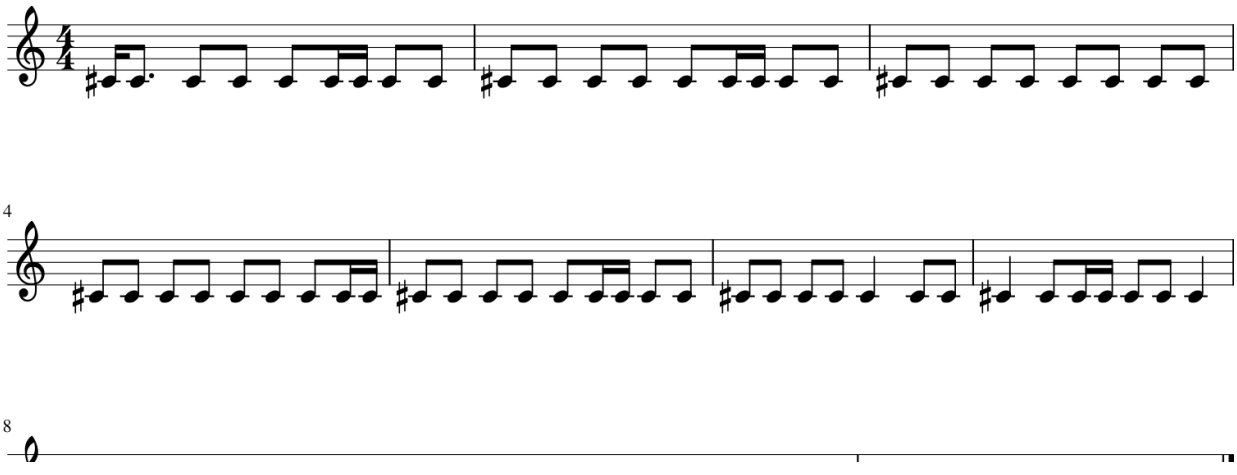
```
# Encodage des mélodies en 2 séquences (notes et durées)

def encode_song(song):
    notes = []
    durations = []
    for note in song.flatten().notesAndRests:
        if isinstance(note, m21.note.Note):
            notes.append(note.pitch.midi)
            durations.append(note.duration.quarterLength)
        elif isinstance(note, m21.note.Rest):
            notes.append(99) # 99 pour les silences
            durations.append(note.duration.quarterLength)
    return notes, durations

# Exemple d'encodage
encoded_song = encode_song(transposed_songs[0])
print("encoded song notes : ", encoded_song[0])
print("encoded song durations : ", encoded_song[1])
```

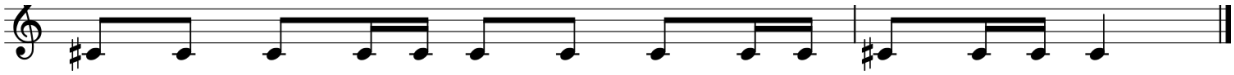
```python
# Affichage du rythme et des notes au format midi

def show_encoded_song(encoded_song):
    # Création de 2 streams
    notes_stream = m21.stream.Stream()
    durations_stream = m21.stream.Stream()

    # Notes
    notes = encoded_song[0]
    for note in notes:
        if note == 99:
            rest = m21.note.Rest()
            rest.quarterLength = 1
            notes_stream.append(rest)
        else:
            note = m21.note.Note(note)
            note.quarterLength = 1
            notes_stream.append(note)

    # Durées
    durations = encoded_song[1]
    for duration in durations:
        note = m21.note.Note(61) # DO
        note.quarterLength = duration
        durations_stream.append(note)

    # Affichage
    print("---Notes encodés---")
    notes_stream.show()
    print("---Rythmes encodés---")
    durations_stream.show()

    return notes_stream, durations_stream

encoded_song_stream = show_encoded_song(encoded_song)
```

```
encoded song notes :  [69, 69, 72, 69, 67, 64, 67, 69, 99, 74, 74, 72, 69, 67, 64, 67, 69, 99, 69, 69, 72,
67, 69, 99, 74, 74, 72, 67, 69, 99, 69, 69, 72, 69, 67, 64, 99, 74, 74, 72, 69, 67, 64, 99, 76, 76, 74, 76,
72, 79, 76, 74, 72, 69, 67, 69, 99, 72, 69, 69, 67, 64, 67, 69, 99, 72, 69, 69, 67, 64, 67, 69]
encoded song durations :  [0.25, 0.75, 0.5, 0.5, 0.5, 0.25, 0.25, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.25,
0.25, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.25, 0.25, 0.5
, 0.5, 0.5, 0.5, 0.5, 0.5, 0.25, 0.25, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 1.0, 0.5, 0.5, 1.0, 0.5, 0.25, 0.25, 0.5,
0.5, 1.0, 0.5, 0.5, 0.5, 0.25, 0.25, 0.5, 0.5, 0.5, 0.25, 0.25, 0.5, 0.25, 0.25, 1.0]
---Notes encodés---
```



```
---Rythmes encodés---
```

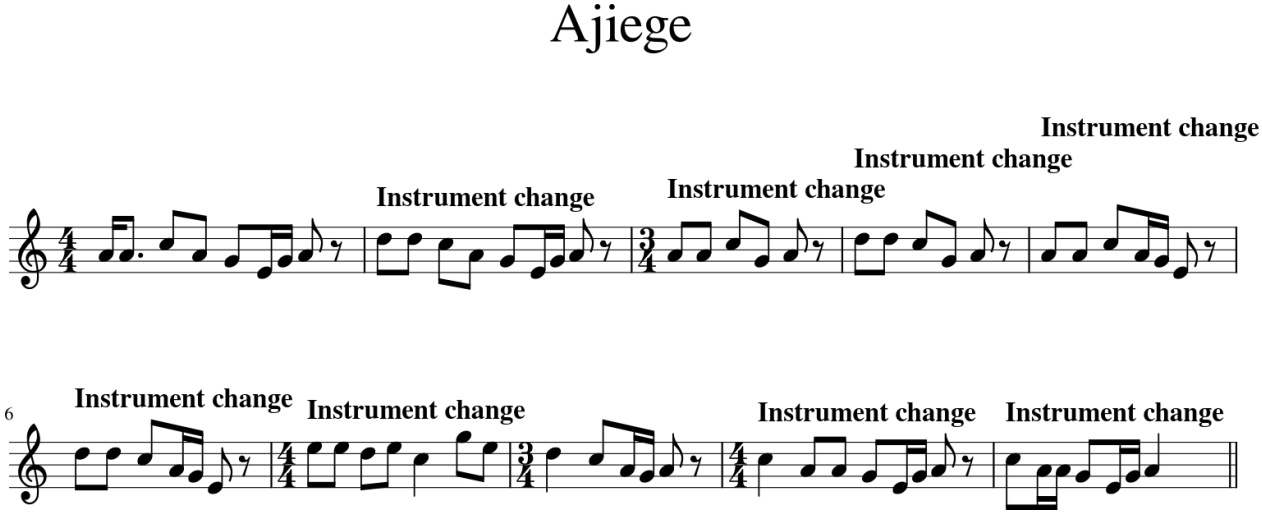




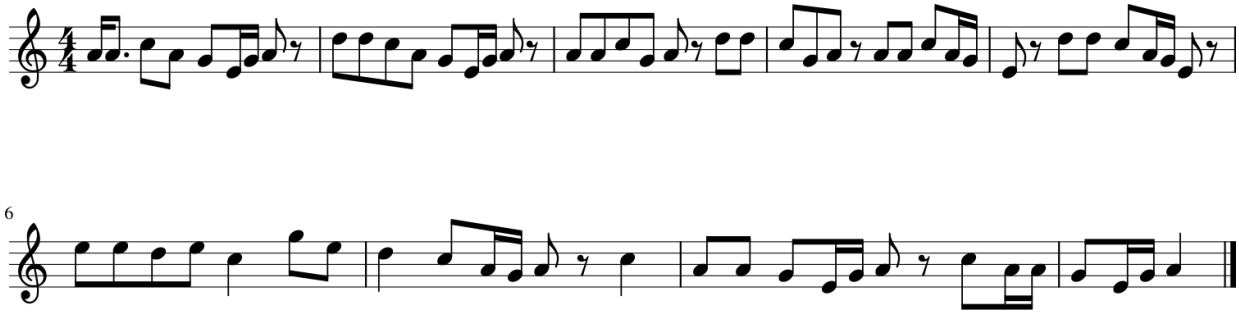
In [16]:

```
# Fonction de décodage

def decode_song(encoded_song):
    notes = encoded_song[0]
    durations = encoded_song[1]
    decoded_song = m21.stream.Score()
    for i in range(len(notes)):
        if notes[i] == 99:
            decoded_song.append(m21.note.Rest(durations[i]))
        else:
            note = m21.note.Note()
            note.pitch.midi = notes[i]
            note.duration.quarterLength = durations[i]
            decoded_song.append(note)
    return decoded_song

# Exemple de décodage
print("Son original : ")
transposed_songs[0].show()
print("Son décodé : ")
decoded_song = decode_song(encoded_song)
decoded_song.show()
```

```
Son original :
```




```
Son décodé :
/home/am/UQAC/AP/projet/projet-env/lib/python3.12/site-packages/music21/musicxml/m21ToXml.py:427: MusicXMLWarning: <music21.stream.Score 0x721d4ab31100> is not well-formed; see isWellFormedNotation()
  warnings.warn(f'{sc} is not well-formed; see isWellFormedNotation()',
```



In [64]:

```
# Encodage des sequences

encoded_songs = []
for song in transposed_songs:
    encoded_songs.append(encode_song(song))

# Affichage de la première séquence
print("encoded song 0 : ", encoded_songs[0][0])
```

```
encoded song 0 :  [69, 69, 72, 69, 67, 64, 67, 69, 99, 74, 74, 72, 69, 67, 64, 67, 69, 99, 69, 69, 72, 67,
9, 99, 74, 74, 72, 67, 69, 99, 69, 69, 72, 69, 67, 64, 99, 74, 74, 72, 69, 67, 64, 99, 76, 76, 74, 76, 72,
9, 76, 74, 72, 69, 67, 69, 99, 72, 69, 69, 67, 64, 67, 69, 99, 72, 69, 69, 67, 64, 67, 69]
```

In [21]:

```
# Padding des séquences
# Il faut que toutes les séquences aient la même longueur

# Information sur les séquences
max_sequence_length = 0
min_sequence_length = 1000
mean_sequence_length = 0
for song in encoded_songs:
    if len(song[0]) > max_sequence_length: max_sequence_length = len(song[0])
    if len(song[0]) < min_sequence_length: min_sequence_length = len(song[0])
    mean_sequence_length += len(song[0])
mean_sequence_length /= len(encoded_songs)
print("max sequence length : ", max_sequence_length)
print("min sequence length : ", min_sequence_length)
print("mean sequence length : ", mean_sequence_length)
```

```
max sequence length :  520
min sequence length :  17
mean sequence length :  73.49718151071026
```

In [68]:

```
# On va choisir une longueur de 100 pour les séquences

MAX_SEQUENCE_LENGTH = 100
# On va padder les séquences avec des silences (99) à la fin et des durées de 4
def pad_sequence(sequence, max_length):
    notes = sequence[0]
    durations = sequence[1]
    if len(notes) < max_length:
        for i in range(max_length - len(notes)):
            notes.append(99)
            durations.append(4.0)
    else :
        notes = notes[:max_length]
        durations = durations[:max_length]
    return notes, durations

# Padding des séquences
# Séparation des notes et des durées
notes_padded_sequences = []
durations_padded_sequences = []
padded_encoded_songs = []
for song in encoded_songs:
    padded_sequence = pad_sequence(song, MAX_SEQUENCE_LENGTH)
    notes_padded_sequences.append(padded_sequence[0])
    durations_padded_sequences.append(padded_sequence[1])
    padded_encoded_songs.append(padded_sequence)
# Affichage de la première séquence padée
print("padded encoded song 0 : ", padded_encoded_songs[0][0])
print("padded encoded song 0 : ", padded_encoded_songs[0][1])

# Affichage de la longueur moyenne des séquences
mean_sequence_length = 0
for song in padded_encoded_songs:
    mean_sequence_length += len(song[0])
mean_sequence_length /= len(padded_encoded_songs)
print("mean sequence length : ", mean_sequence_length)
```

```
padded encoded song 0 :  [69, 69, 72, 69, 67, 64, 67, 69, 99, 74, 74, 72, 69, 67, 64, 67, 69, 99, 69, 69, 7
2, 67, 69, 99, 74, 74, 72, 67, 69, 99, 69, 69, 72, 69, 67, 64, 99, 74, 74, 72, 69, 67, 64, 99, 76, 76, 74,
6, 72, 79, 76, 74, 72, 69, 67, 69, 99, 72, 69, 69, 67, 64, 67, 69, 99, 72, 69, 69, 67, 64, 67, 69, 99, 99,
9, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99]
padded encoded song 0 :  [0.25, 0.75, 0.5, 0.5, 0.5, 0.25, 0.25, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.25,
0.25, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.25, 0.25, 0.5
, 0.5, 0.5, 0.5, 0.5, 0.25, 0.25, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 1.0, 0.5, 0.5, 1.0, 0.5, 0.25, 0.25, 0.5,
0.5, 1.0, 0.5, 0.5, 0.5, 0.25, 0.25, 0.5, 0.5, 0.5, 0.25, 0.25, 0.5, 0.25, 0.25, 1.0, 4.0, 4.0, 4.0, 4.0,
4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4
.0, 4.0, 4.0]
mean sequence length :  100.0
```

Maintenant, on a des séquences divisées en 2 avec une partie notes et une parties rytme et elles font toutes la même taille.

Pour la suite, je pensais normaliser les séquences mais en y réfléchissant, l'utilisation d'un one-hot semble plus approprié. J'aurais déjà pu utiliser un one-hot dans le notebook précédent mais je n'y avais pas pensé.

Encodage des données en one-hot

In [69]:

```
# Pour faire des one hot il faut des entiers
# Encodage des durées en entier
durations_int = []
for duration in durations_padded_sequences:
    duration_int = []
    for d in duration:
        if d == 0.25:
            duration_int.append(0)
        elif d == 0.5:
            duration_int.append(1)
        elif d == 0.75:
            duration_int.append(2)
        elif d == 1.0:
            duration_int.append(3)
        elif d == 1.5:
            duration_int.append(4)
        elif d == 2.0:
            duration_int.append(5)
        elif d == 3.0:
            duration_int.append(6)
        elif d == 4.0:
            duration_int.append(7)
        else:
            print("Unknown duration : ", d)
            duration_int.append(8)
    durations_int.append(duration_int)

# Affichage de la première séquence de durées
print("durations padded song 0 : ", durations_padded_sequences[0])
print("durations padded song 0 int : ", durations_int[0])
```

```
durations padded song 0 :  [0.25, 0.75, 0.5, 0.5, 0.5, 0.25, 0.25, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.25
, 0.25, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.25, 0.25, 0
.5, 0.5, 0.5, 0.5, 0.25, 0.25, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 1.0, 0.5, 0.5, 1.0, 0.5, 0.25, 0.25, 0.5
, 0.5, 1.0, 0.5, 0.5, 0.5, 0.25, 0.25, 0.5, 0.5, 0.5, 0.25, 0.25, 0.5, 0.25, 0.25, 1.0, 4.0, 4.0, 4.0, 4.0
, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0,
4.0, 4.0, 4.0]
durations padded song 0 int :  [0, 2, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 3, 1, 1, 3, 1, 0, 0, 1, 1, 3, 1, 1,
1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 3, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,
7, 7, 7, 7, 7]
```

In [70]:

```
# Transformation des séquences en tenseurs pytorch
import torch
import torch.nn.functional as F

notes_tensor = torch.tensor(notes_padded_sequences, dtype=torch.long)
durations_tensor = torch.tensor(durations_int, dtype=torch.long)

print("notes tensor shape : ", notes_tensor.shape)
print("durations tensor shape : ", durations_tensor.shape)
```

```
notes tensor shape :  torch.Size([874, 100])
durations tensor shape :  torch.Size([874, 100])
```

In [71]:

```
# Pour faire du one hot sur 50 il faut que les notes soient entre 0 et 49
# Décalage des notes de 50 à 99 de 0 à 49
min_note = notes_tensor.min().item()
print("min note : ", min_note)
notes_tensor_shifted = notes_tensor - min_note
```

```
min note :  50
```

In [72]:

```
nb_notes = 50 # on a 50 notes différentes
nb_durations = 8 # on a 8 durées différentes

# one hot encoding
notes_one_hot = F.one_hot(notes_tensor_shifted, num_classes=nb_notes)
duration_one_hot = F.one_hot(durations_tensor, num_classes=nb_durations)
print("notes one hot shape : ", notes_one_hot.shape)
print("durations one hot shape : ", duration_one_hot.shape)
```

```
notes one hot shape :  torch.Size([874, 100, 50])
durations one hot shape :  torch.Size([874, 100, 8])
```

In [73]:

```
# Affichage de la première séquence
print("notes one hot : ", notes_one_hot[0])
print("durations one hot : ", duration_one_hot[0])
```

```
notes one hot :  tensor([[0, 0, 0,  ..., 0, 0, 0],
        [0, 0, 0,  ..., 0, 0, 0],
        [0, 0, 0,  ..., 0, 0, 0],
        ...,
        [0, 0, 0,  ..., 0, 0, 1],
        [0, 0, 0,  ..., 0, 0, 1],
        [0, 0, 0,  ..., 0, 0, 1]])
durations one hot :  tensor([[1, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 1, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 1, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 1, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 1, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0, 0, 0],
```

```
       [0, 0, 0, 1, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 1]])
```

In [ ]:

Génération de mélodies chinoises avec un LSTM (dataset "han")

In [17]:

```
import os
import music21 as m21
import json
from datetime import datetime
from preprocessing import *
from training import *
```

Paramètres

In [18]:

```
DATASET_PATH = "../data/han/original_songs"
SAVE_DIR_ENCODED_SONGS = "../data/han/encoded_songs"

SEQUENCE_LENGTH = 64
FILE_DATASET_PATH = "../data/han/file_dataset"
MAPPINGS_PATH = "../data/han/mapping.json"

NUM_UNITS = [256]
LOSS = "sparse_categorical_crossentropy"
LEARNING_RATE = 0.001
EPOCHS = 10
BATCH_SIZE = 64

MODEL_PATH = "../data/han/model_RNN_LSTM.keras"

MAPPING_PATH = "../data/han/mapping.json"
SAVE_GENERATED_MELODIES_DIR = "../generated_melodies/RNN-LSTM/han"
seed = "55"
```

Préparation des données

- **Chargement des musiques**
- **Enlever notes plus courtes que des double croches**
- **Enlever les musiques vides**
- **Transposer tout en Do majeur (La mineur)**
- **Encoder les musiques en string, tout mettre dans un fichier texte et sauvegarder l'équivalence (mappings)**

In [19]:

```
print("loading data...")
songs = load_songs(DATASET_PATH, 10000)
print(len(songs),"songs loaded")
```

```
loading data...
1223 songs loaded
```

Enlever durées non acceptables (notes plus courtes que double croche) et musiques vides

In [20]:

```
print("avant filtrage :", len(songs))
for song in songs:
    if not has_acceptable_durations(song, ACCEPTABLE_DURATIONS):
        songs.remove(song)
    if len(song.recurse().getElementsByClass(m21.note.Note)) == 0:
        songs.remove(song)
print("après filtrage :", len(songs))
```

```
avant filtrage : 1223
après filtrage : 1091
```

Transposer en do majeur

In [21]:

```
transposed_songs = []
```

```
for song in songs:
    transposed_songs.append(transpose(song))
```

Encoder les musiques en format fichier texte

In [22]:

```
encoded_songs = []
for song in transposed_songs:
    encoded_songs.append(encode_song(song))
```

sauvegarde dans un fichier texte

In [23]:

```
for i, encoded_song in enumerate(encoded_songs):
    save_path = os.path.join(SAVE_DIR_ENCODED_SONGS, str(i))
    with open(save_path, "w") as fp:
        fp.write(encoded_song)
```

Tout mettre dans un fichier

In [ ]:

```
create_single_file_dataset(dataset_path = SAVE_DIR_ENCODED_SONGS, file_dataset_path=FILE_DATASET_PATH, sequ
ence_length=SEQUENCE_LENGTH)
```

Mapping des symboles

In [25]:

```
songs=load(FILE_DATASET_PATH)
create_mapping(songs, MAPPINGS_PATH)
mappings, OUTPUT_UNITS = load_json(MAPPINGS_PATH)
print(OUTPUT_UNITS)
```

```
45
```

Entrainement

In [26]:

```
inputs, targets = generate_training_sequences(64, songs, mappings)
```

In [27]:

```
# build the model
model = build_model(OUTPUT_UNITS, NUM_UNITS, LOSS, LEARNING_RATE)
```

Model: "functional_1"

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, None, 45)	0
lstm_1 (LSTM)	(None, 256)	309,248
dropout_1 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 45)	11,565

Total params: 320,813 (1.22 MB)

Trainable params: 320,813 (1.22 MB)

Non-trainable params: 0 (0.00 B)

In [28]:

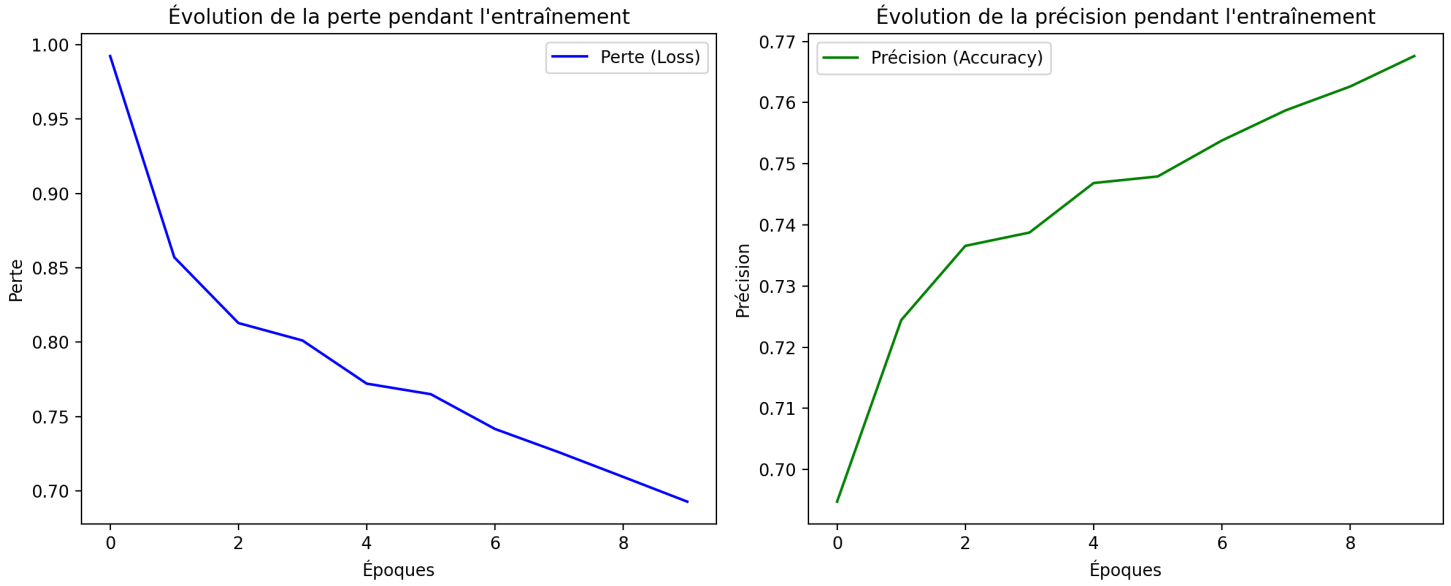
```
# train the model

history = model.fit(inputs, targets, epochs=EPOCHS, batch_size=BATCH_SIZE)
```

```
Epoch 1/10
3984/3984 ━━━━━━━━━━━━━━━━━━━━ 223s 56ms/step - accuracy: 0.6731 - loss: 1.1465
Epoch 2/10
```

```
3984/3984 ━━━━━━━━━━━━━━━━━━━━ 274s 69ms/step - accuracy: 0.7209 - loss: 0.8678
Epoch 3/10
3984/3984 ━━━━━━━━━━━━━━━━━━━━ 199s 50ms/step - accuracy: 0.7345 - loss: 0.8213
Epoch 4/10
3984/3984 ━━━━━━━━━━━━━━━━━━━━ 247s 62ms/step - accuracy: 0.7385 - loss: 0.8039
Epoch 5/10
3984/3984 ━━━━━━━━━━━━━━━━━━━━ 226s 57ms/step - accuracy: 0.7459 - loss: 0.7723
Epoch 6/10
3984/3984 ━━━━━━━━━━━━━━━━━━━━ 214s 54ms/step - accuracy: 0.7464 - loss: 0.7695
Epoch 7/10
3984/3984 ━━━━━━━━━━━━━━━━━━━━ 209s 52ms/step - accuracy: 0.7549 - loss: 0.7393
Epoch 8/10
3984/3984 ━━━━━━━━━━━━━━━━━━━━ 237s 59ms/step - accuracy: 0.7582 - loss: 0.7266
Epoch 9/10
3984/3984 ━━━━━━━━━━━━━━━━━━━━ 218s 55ms/step - accuracy: 0.7636 - loss: 0.7057
Epoch 10/10
3984/3984 ━━━━━━━━━━━━━━━━━━━━ 269s 68ms/step - accuracy: 0.7676 - loss: 0.6890
```

In [35]:

```
fig, axs = plt.subplots(1, 2, figsize=(12, 5))

# Subplot 1 : Loss
axs[0].plot(history.history['loss'], label='Perte (Loss)', color='blue')
axs[0].set_xlabel('Époques')
axs[0].set_ylabel('Perte')
axs[0].set_title("Évolution de la perte pendant l'entraînement")
axs[0].legend()

# Subplot 2 : Accuracy
axs[1].plot(history.history['accuracy'], label='Précision (Accuracy)', color='green')
axs[1].set_xlabel('Époques')
axs[1].set_ylabel('Précision')
axs[1].set_title("Évolution de la précision pendant l'entraînement")
axs[1].legend()

# Affichage
plt.tight_layout()
plt.show()
```



In [30]:

```
# save the model
model.save(MODEL_PATH)
```

Génération de mélodies

In [31]:

```
# load model
print(MODEL_PATH)
model = keras.models.load_model(MODEL_PATH)

# start symboles
start_symbols = ["/"] * SEQUENCE_LENGTH

# mappings
```

```
with open(MAPPING_PATH, "r") as fp:
    mappings = json.load(fp)
```

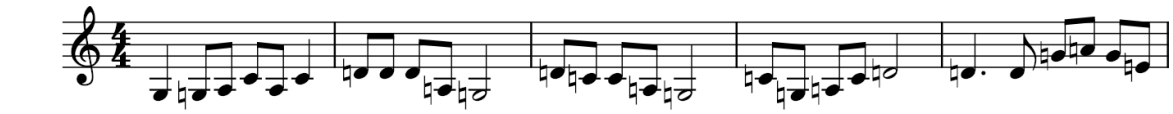
```
../data/han/model_RNN_LSTM.keras
```

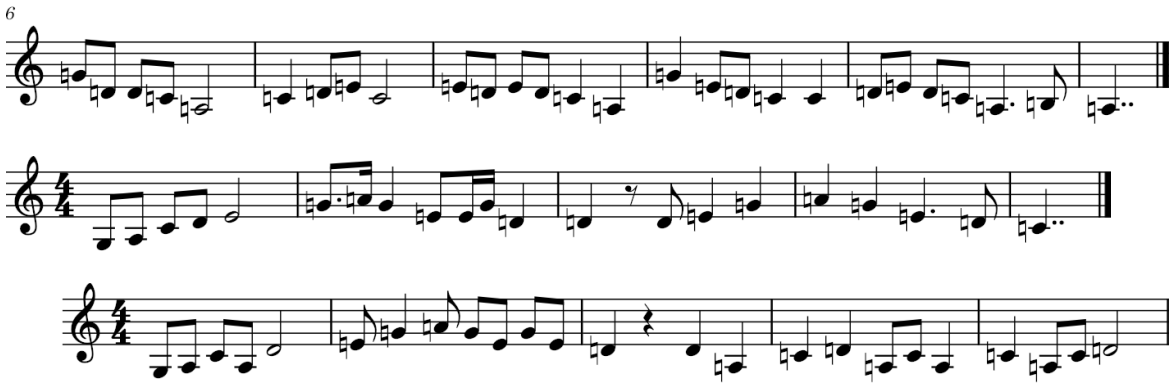
In [ ]:

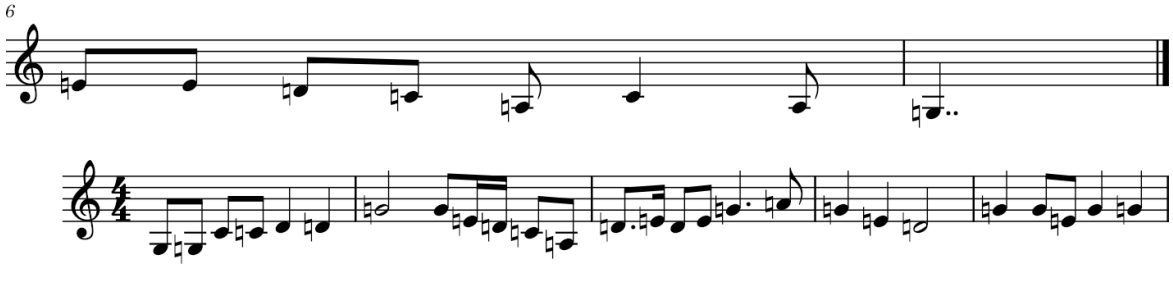
```
# Générer 10 mélodies
melodies = []
for _ in range (10):
    melodies.append(generate_melody(model, start_symbols, mappings, seed, 500, SEQUENCE_LENGTH, 0.7))


print(melodies[0])
```

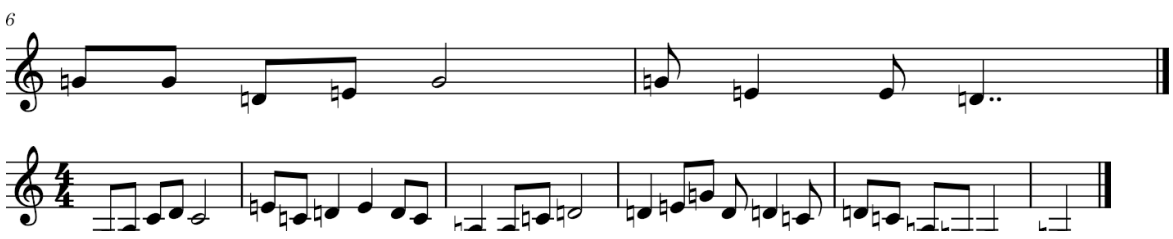
In [33]:

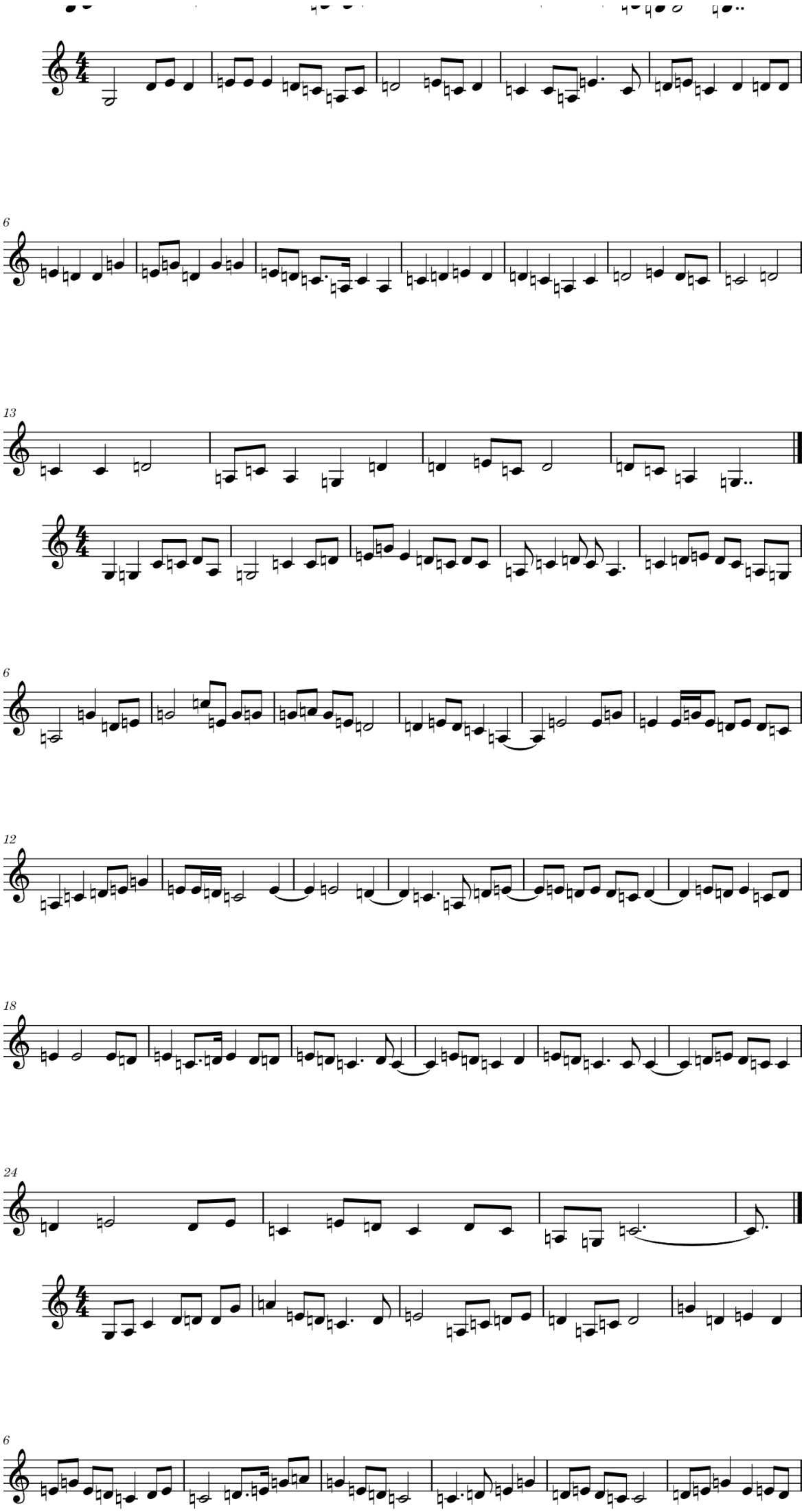
```
# Les mettre en format midi
songs = []
for i in range(10):
    songs.append(convert_to_midi(melodies[i]))

for song in songs:
    song.show("midi")
    song.show()
```

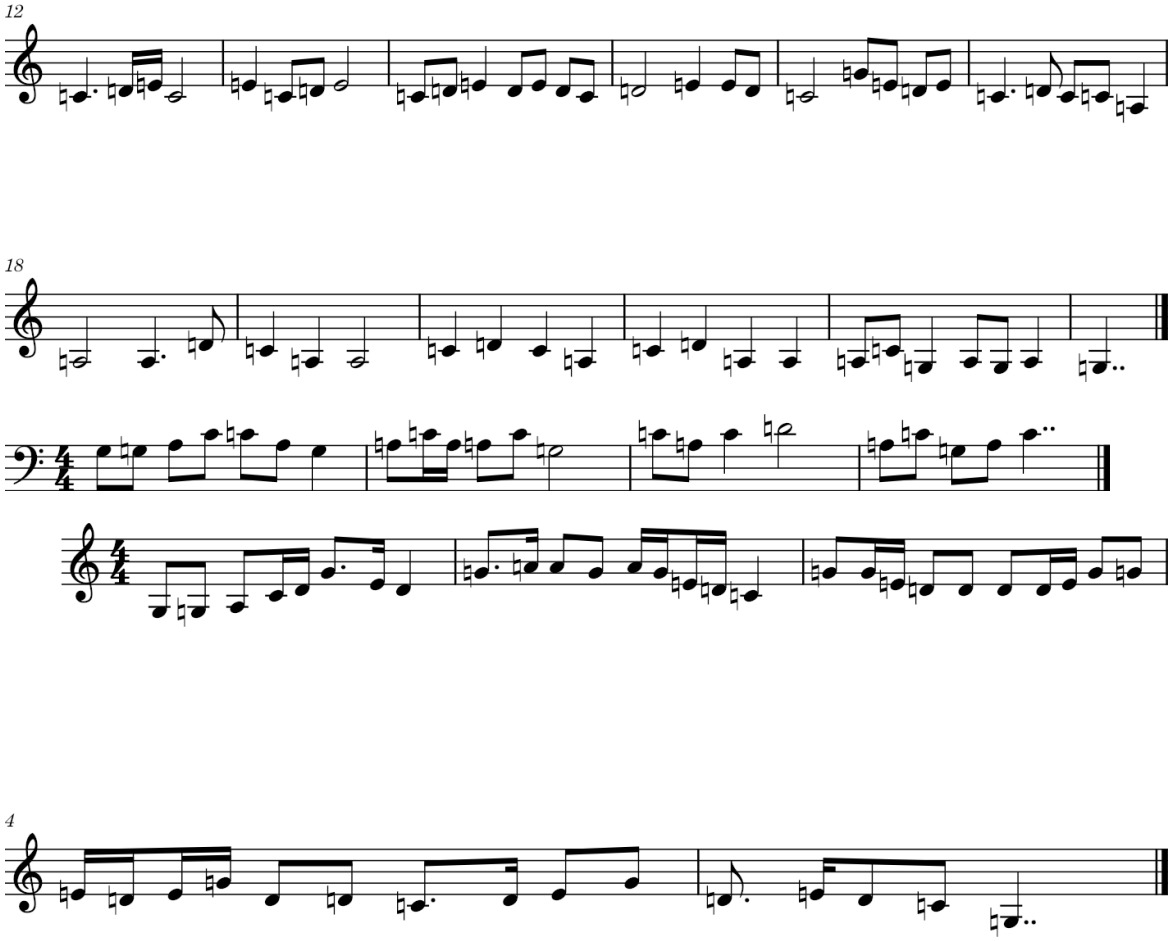












In [34]:

```
# save
timestamp = datetime.now().strftime("%d%m_%H%M%S")  # Format sauvegarde : JJMM_HHMMSS_i
for i in range(10):
    songs[i].write("midi", SAVE_GENERATED_MELODIES_DIR + f"/{timestamp}_{i}.mid")
```

Génération de mélodies allemandes avec un LSTM (dataset "deutsch")

In [1]:

```
import os
import music21 as m21
import json
from datetime import datetime
from preprocessing import *
from training import *
```

Paramètres

In [2]:

```
DATASET_PATH = "../data/deutsch/original_songs"
SAVE_DIR_ENCODED_SONGS = "../data/deutsch/encoded_songs"

SEQUENCE_LENGTH = 64
FILE_DATASET_PATH = "../data/deutsch/file_dataset"
MAPPINGS_PATH = "../data/deutsch/mapping.json"

NUM_UNITS = [256]
LOSS = "sparse_categorical_crossentropy"
LEARNING_RATE = 0.001
EPOCHS = 10
BATCH_SIZE = 64

MODEL_PATH = "../data/deutsch/model_RNN_LSTM.keras"

MAPPING_PATH = "../data/deutsch/mapping.json"
SAVE_GENERATED_MELODIES_DIR = "../generated_melodies/RNN-LSTM/deutsch"
seed = "55"
```

Préparation des données

In [3]:

```
print("loading data...")
songs = load_songs(DATASET_PATH, 10000)
print(len(songs),"songs loaded")
```

```
loading data...
1700 songs loaded
```

Enlever durées non acceptables (notes plus courtes que double croche) et musiques vides

In [4]:

```
print("avant filtrage :", len(songs))
for song in songs:
    if not has_acceptable_durations(song, ACCEPTABLE_DURATIONS):
        songs.remove(song)
    if len(song.recurse().getElementsByClass(m21.note.Note)) == 0:
        songs.remove(song)
print("après filtrage :", len(songs))
```

```
avant filtrage : 1700
après filtrage : 1683
```

Transposer en do majeur

In [5]:

```
transposed_songs = []
for song in songs:
    transposed_songs.append(transpose(song))
```

Encoder les musiques en format fichier texte

In [6]:

In [6]:

```
encoded_songs = []
for song in transposed_songs:
    encoded_songs.append(encode_song(song))
```

sauvegarde dans un fichier texte

In [7]:

```
for i, encoded_song in enumerate(encoded_songs):
    save_path = os.path.join(SAVE_DIR_ENCODED_SONGS, str(i))
    with open(save_path, "w") as fp:
        fp.write(encoded_song)
```

Tout mettre dans un fichier

In [ ]:

```
create_single_file_dataset(dataset_path = SAVE_DIR_ENCODED_SONGS, file_dataset_path=FILE_DATASET_PATH, sequ
ence_length=SEQUENCE_LENGTH)
```

Mapping des symboles

In [9]:

```
songs=load(FILE_DATASET_PATH)
create_mapping(songs, MAPPINGS_PATH)
mappings, OUTPUT_UNITS = load_json(MAPPINGS_PATH)
print(OUTPUT_UNITS)
```

```
38
```

Entrainement

In [10]:

```
inputs, targets = generate_training_sequences(64, songs, mappings)
```

In [11]:

```
# build the model
model = build_model(OUTPUT_UNITS, NUM_UNITS, LOSS, LEARNING_RATE)
```

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, None, 38)	0
lstm (LSTM)	(None, 256)	302,080
dropout (Dropout)	(None, 256)	0
dense (Dense)	(None, 38)	9,766

Total params: 311,846 (1.19 MB)

Trainable params: 311,846 (1.19 MB)

Non-trainable params: 0 (0.00 B)

In [12]:

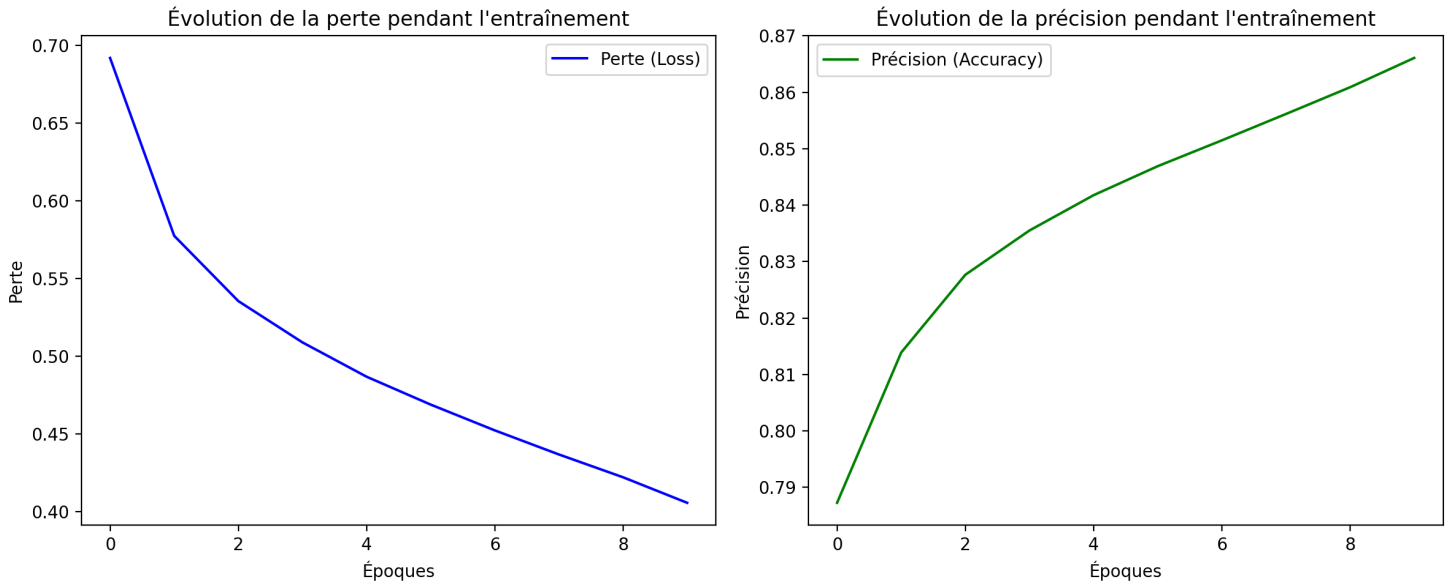
```
# train the model

history = model.fit(inputs, targets, epochs=EPOCHS, batch_size=BATCH_SIZE)
```

```
Epoch 1/10
5660/5660 ━━━━━━━━━━━━━━━━━━━━ 322s 57ms/step - accuracy: 0.7712 - loss: 0.8240
Epoch 2/10
5660/5660 ━━━━━━━━━━━━━━━━━━━━ 293s 52ms/step - accuracy: 0.8091 - loss: 0.5917
Epoch 3/10
5660/5660 ━━━━━━━━━━━━━━━━━━━━ 327s 58ms/step - accuracy: 0.8258 - loss: 0.5393
Epoch 4/10
5660/5660 ━━━━━━━━━━━━━━━━━━━━ 297s 52ms/step - accuracy: 0.8348 - loss: 0.5116
Epoch 5/10
5660/5660 ━━━━━━━━━━━━━━━━━━━━ 278s 49ms/step - accuracy: 0.8414 - loss: 0.4889
```

```
5660/5660 ━━━━━━━━━━━━━━━━━━━━ 378s 67ms/step - accuracy: 0.8414 - loss: 0.4869
Epoch 6/10
5660/5660 ━━━━━━━━━━━━━━━━━━━━ 367s 65ms/step - accuracy: 0.8467 - loss: 0.4689
Epoch 7/10
5660/5660 ━━━━━━━━━━━━━━━━━━━━ 330s 58ms/step - accuracy: 0.8516 - loss: 0.4499
Epoch 8/10
5660/5660 ━━━━━━━━━━━━━━━━━━━━ 297s 52ms/step - accuracy: 0.8574 - loss: 0.4340
Epoch 9/10
5660/5660 ━━━━━━━━━━━━━━━━━━━━ 301s 53ms/step - accuracy: 0.8620 - loss: 0.4181
Epoch 10/10
5660/5660 ━━━━━━━━━━━━━━━━━━━━ 301s 53ms/step - accuracy: 0.8677 - loss: 0.4005
```

In [20]:

```
fig, axs = plt.subplots(1, 2, figsize=(12, 5))

# Subplot 1 : Loss
axs[0].plot(history.history['loss'], label='Perte (Loss)', color='blue')
axs[0].set_xlabel('Époques')
axs[0].set_ylabel('Perte')
axs[0].set_title("Évolution de la perte pendant l'entraînement")
axs[0].legend()

# Subplot 2 : Accuracy
axs[1].plot(history.history['accuracy'], label='Précision (Accuracy)', color='green')
axs[1].set_xlabel('Époques')
axs[1].set_ylabel('Précision')
axs[1].set_title("Évolution de la précision pendant l'entraînement")
axs[1].legend()

# Affichage
plt.tight_layout()
plt.show()
```



In [14]:

```
# save the model
model.save(MODEL_PATH)
```

Génération de mélodies

In [15]:

```
# load model
print(MODEL_PATH)
model = keras.models.load_model(MODEL_PATH)

# start symboles
start_symbols = ["/"] * SEQUENCE_LENGTH

# mappings
with open(MAPPING_PATH, "r") as fp:
    mappings = json.load(fp)
```

```
../data/deutsch/model_RNN_LSTM.keras
```

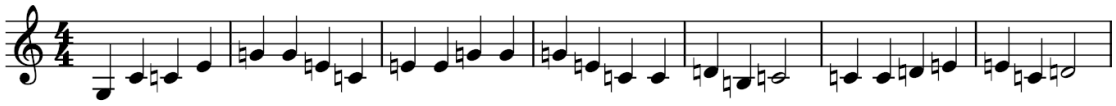
In [ ]:

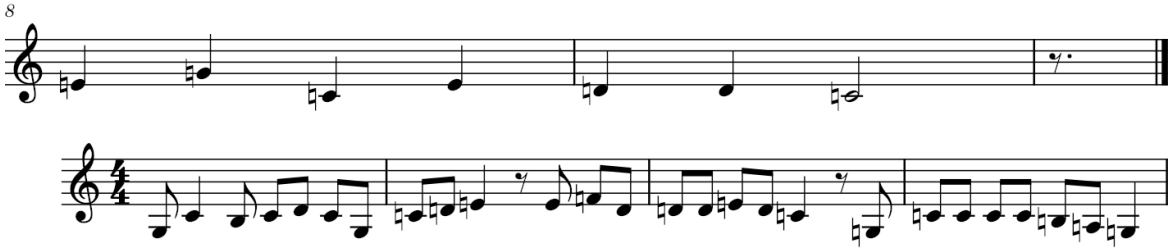
```
# Générer 10 mélodies
```

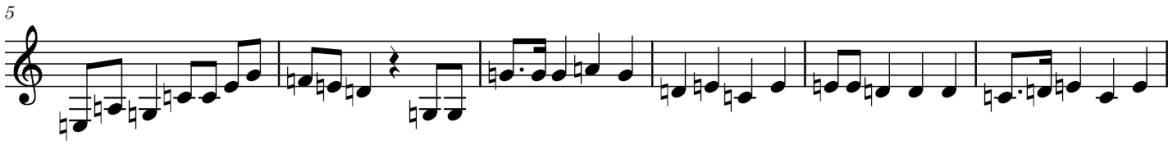
```
melodies = []
for _ in range (10):
    melodies.append(generate_melody(model, start_symbols, mappings, seed, 500, SEQUENCE_LENGTH, 0.7))


print(melodies[0])
```

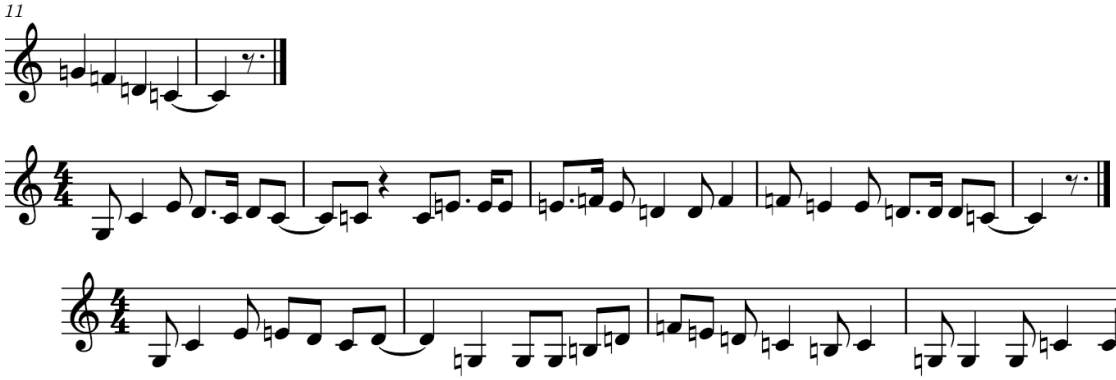
In [17]:

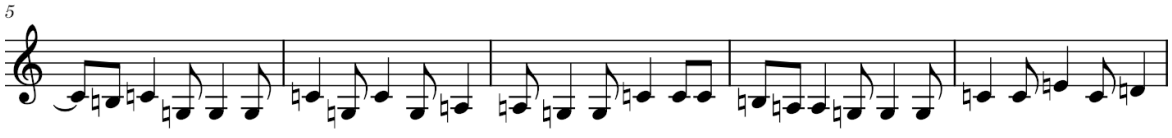
```
# Les mettre en format midi
songs = []
for i in range(10):
    songs.append(convert_to_midi(melodies[i]))

for song in songs:
    song.show("midi")
    song.show()
```

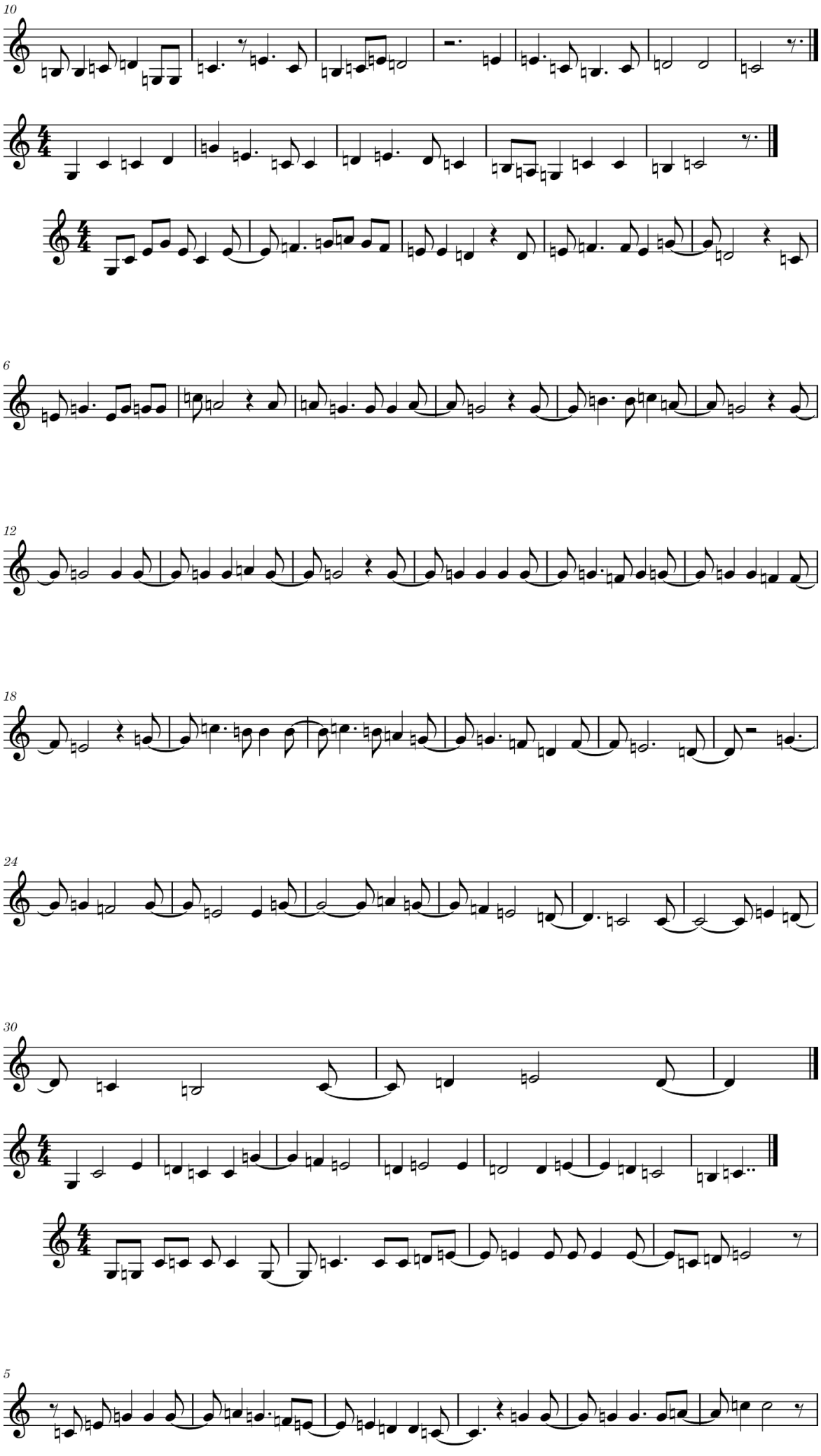


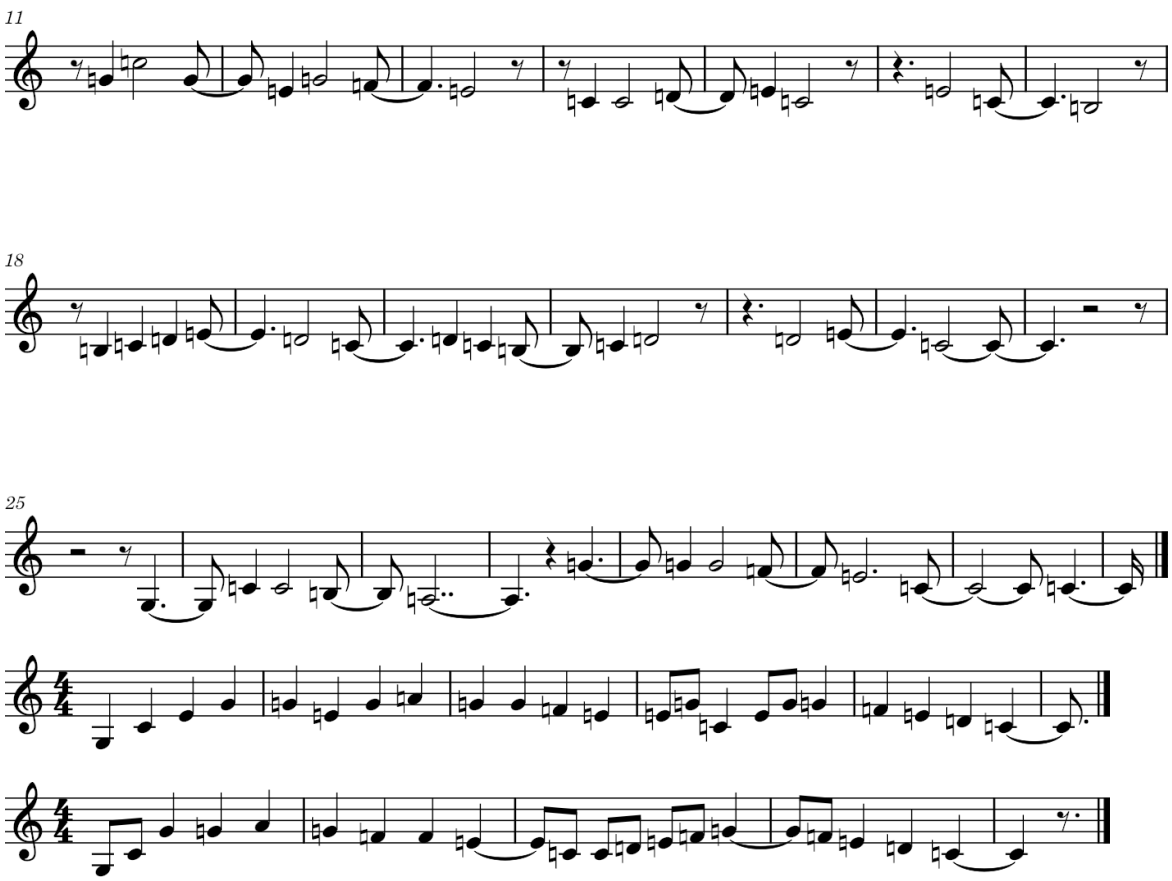












In [18]:

```
# save
timestamp = datetime.now().strftime("%d%m_%H%M%S")  # Format sauvegarde : JJMM_HHMMSS_i
for i in range(10):
    songs[i].write("midi", SAVE_GENERATED_MELODIES_DIR + f"/{timestamp}_{i}.mid")
```

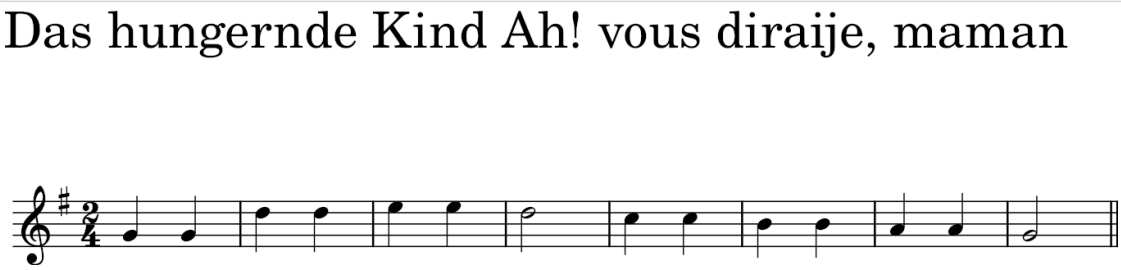
Génération de la suite d'une mélodie connue : "Ah vous dirais-je maman"

In [36]:

```
import os
import music21 as m21
import json
from datetime import datetime
from preprocessing import *
from training import *
```

In [37]:

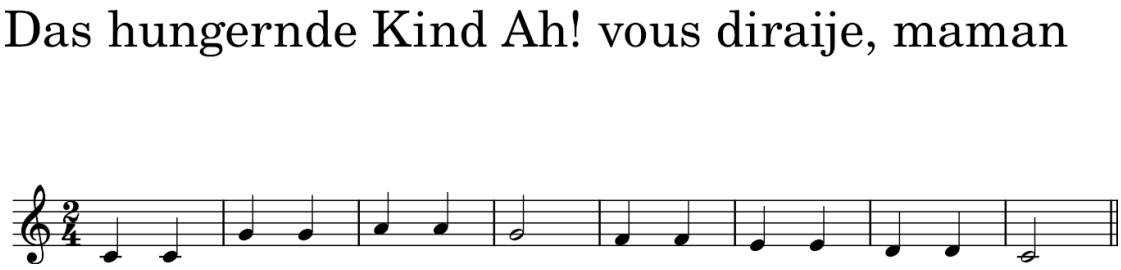
```
# Récupérer les notes de "Ah vous dirais-je maman"
french_dataset_path = "../data/france"
french_songs = load_songs(french_dataset_path, 100)
french_songs[11].show()
french_songs[11].show("midi")
```



In [38]:

```
test_song = transpose(french_songs[11], True)
test_song.show()
test_song.show("midi")
```

```
old key :  G major
```



In [39]:

```
test_song = encode_song(test_song)
```

In [ ]:

```
print(test_song)
print(test_song[:64]) # 4 premières mesures
```

```
60 _ _ _ 60 _ _ _ 67 _ _ _ 67 _ _ _ 69 _ _ _ 69 _ _ _ 67 _ _ _ _ _ _ _ 65 _ _ _ 65 _ _ _ 64 _ _ _ 64 _ _ _ 6
2 _ _ _ 62 _ _ _ 60 _ _ _ _ _ _ _ _
60 _ _ _ 60 _ _ _ 67 _ _ _ 67 _ _ _ 69 _ _ _ 69 _ _ _ 67 _ _ _ _
```

Modèle entraîné sur le dataset han (chinois)

In [41]:

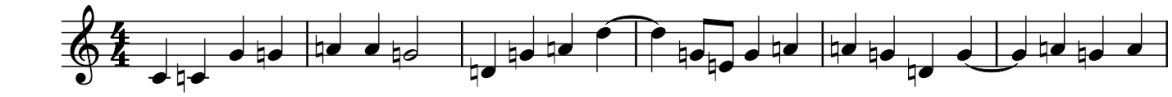
```
model = keras.models.load_model("../data/han/model_RNN_LSTM.keras")

# start symboles
start_symbols = ["/"] * 64

# mappings
with open("../data/han/mapping.json", "r") as fp:
    mappings = json.load(fp)
```

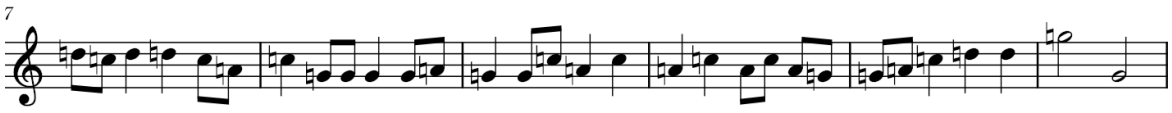
In [ ]:

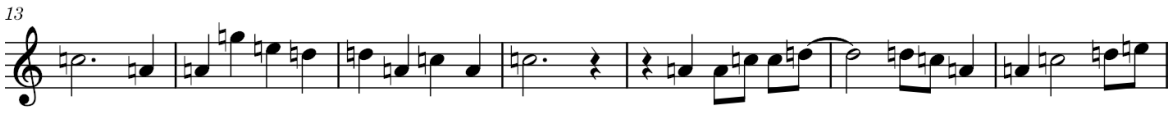
```
# Générer 10 mélodies à partir de ah vous dirais-je maman
melodies = []
for _ in range (10):
    melodies.append(generate_melody(model, start_symbols, mappings, test_song[:64], 500, 64, 0.7))
```

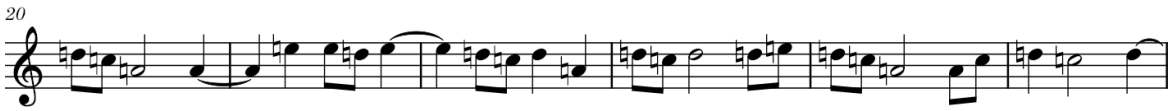
In [56]:

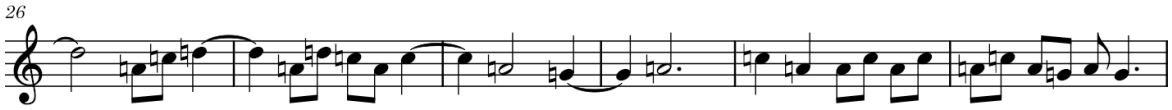
```
# Les mettre en format midi
songs = []
for i in range(10):
    songs.append(convert_to_midi(melodies[i]))

for song in songs:
    song.show("midi")
    song.show()

songs[5].write("midi", "../generated_melodies/goodexamples/ahvousidraisjemaman_han.midi")
```

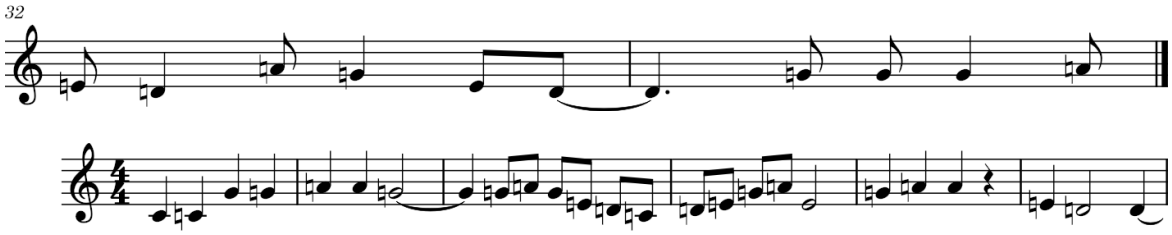


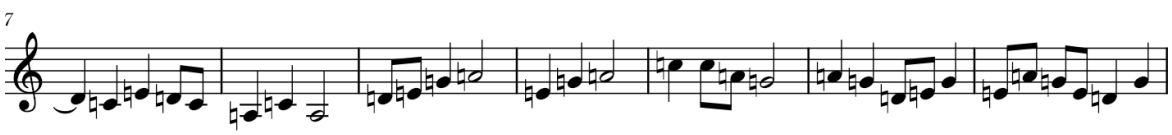


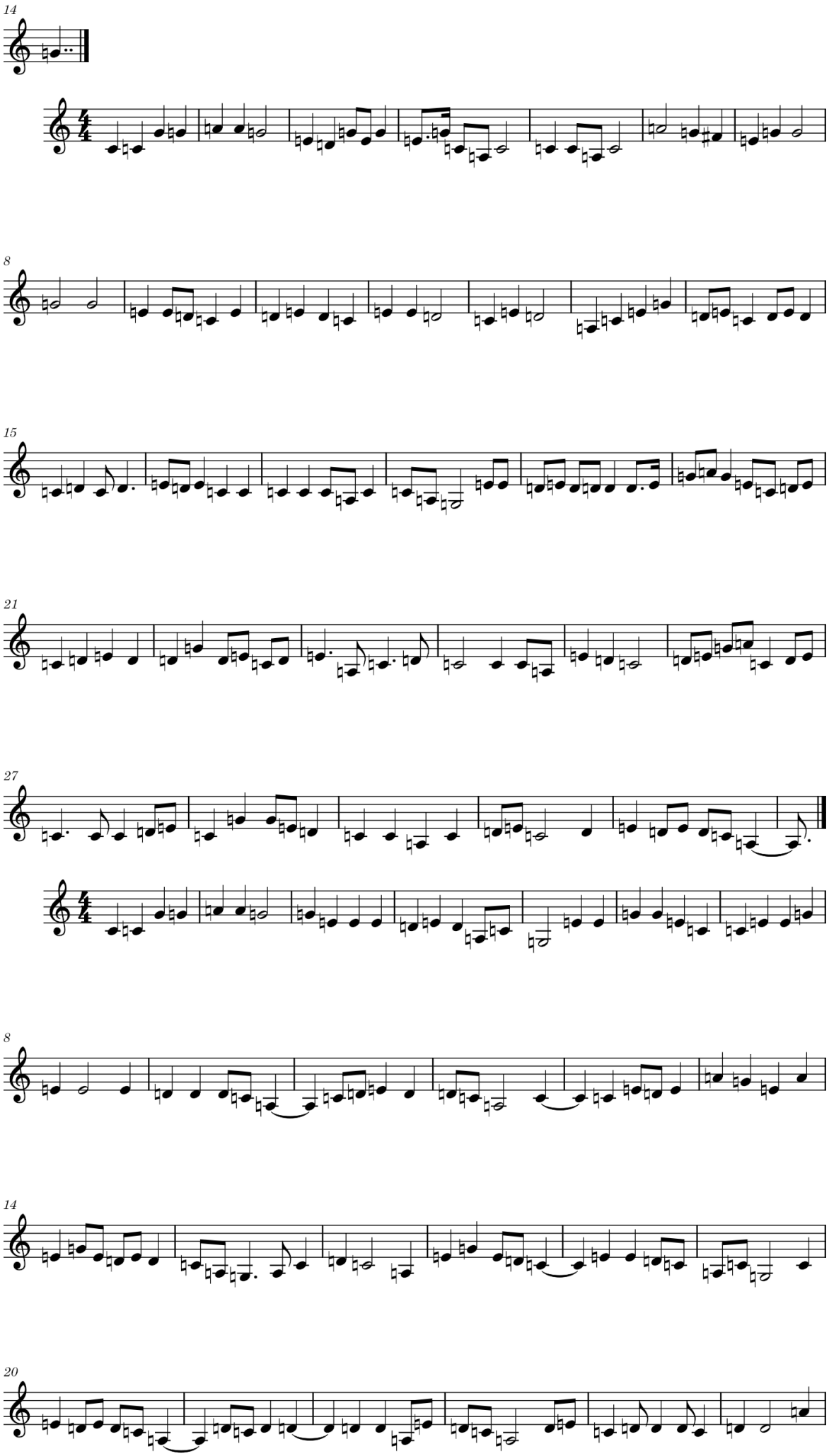


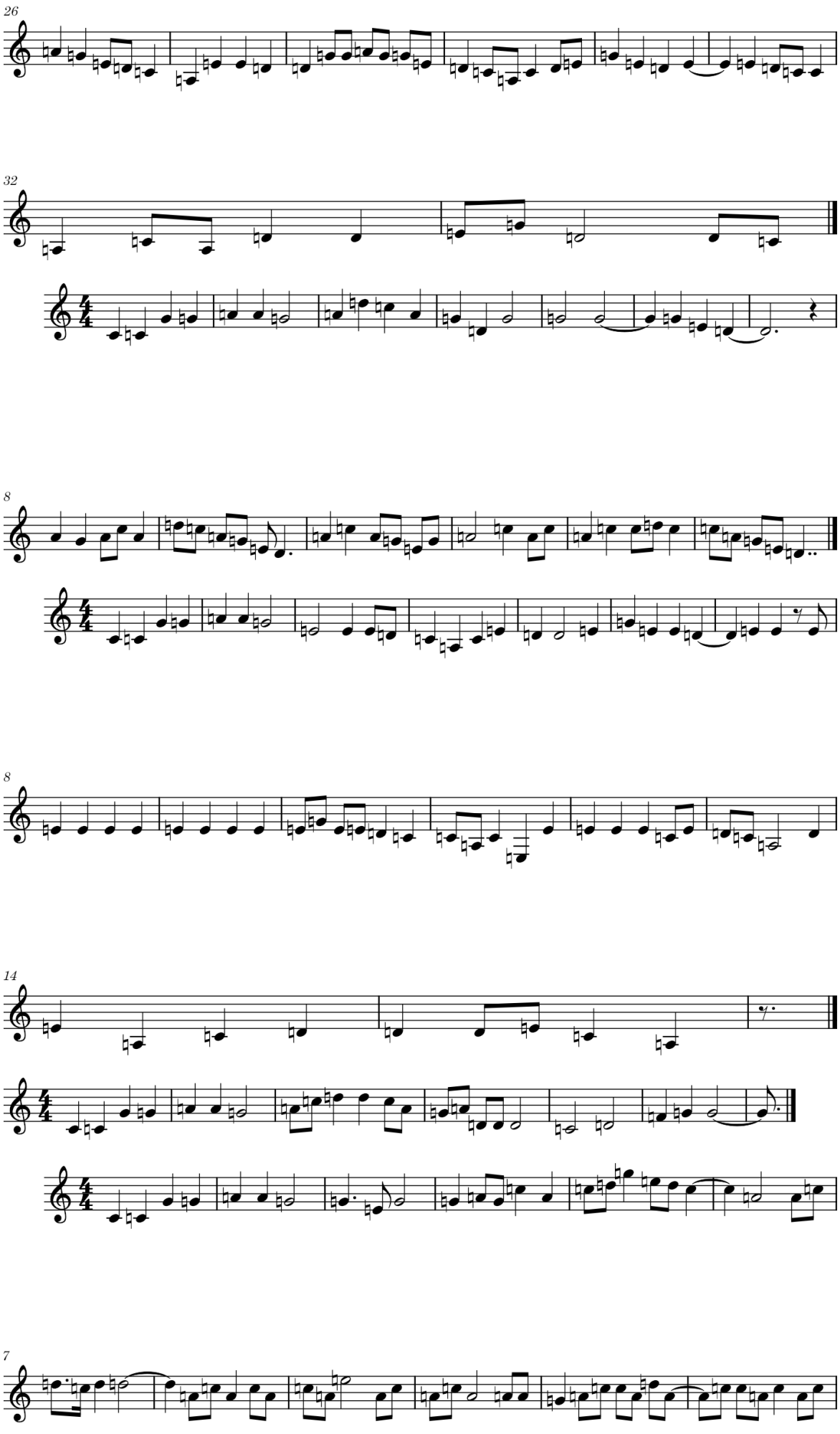


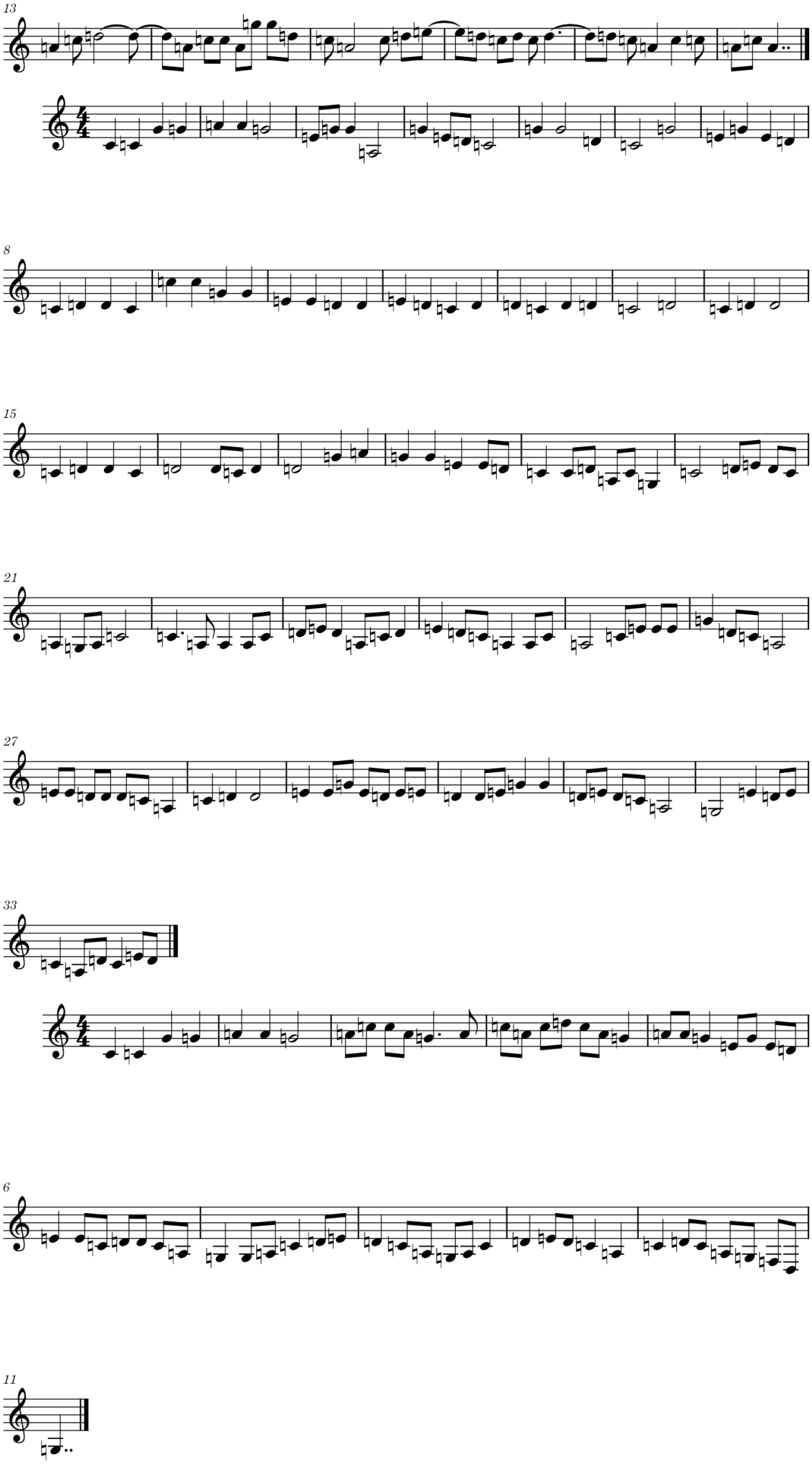












Out[56]:

'../generated_melodies/goodexamples/ahvousidraisjemaman_han.midi'

Modèle entraîné sur le dataset deutsch (allemand)

In [45]:

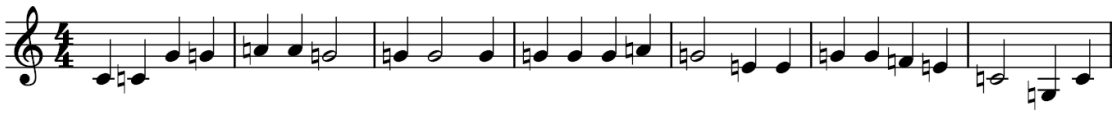
```
model_deutsch = keras.models.load_model("../data/deutsch/model_RNN_LSTM.keras")

# start symboles
start_symbols = ["/"] * 64

# mappings
with open("../data/deutsch/mapping.json", "r") as fp:
    mappings_deutsch = json.load(fp)
```

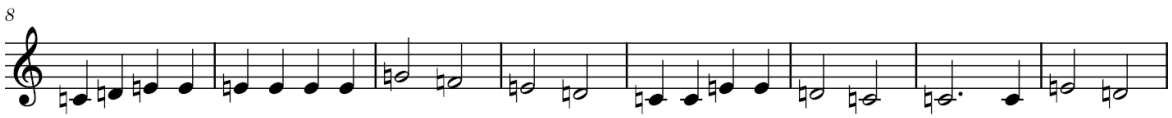
In [ ]:

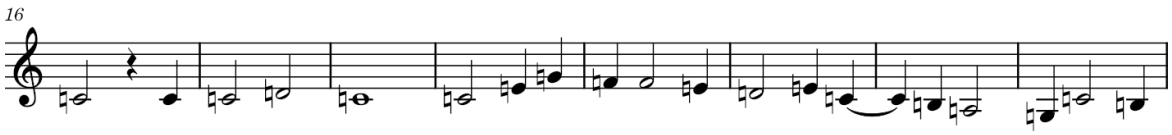
```
# Générer 10 mélodies à partir de ah vous dirais-je maman
melodies_deutsch = []
for _ in range (10):
    melodies_deutsch.append(generate_melody(model_deutsch, start_symbols, mappings_deutsch, test_song[:64]
, 500, 64, 0.7))
```

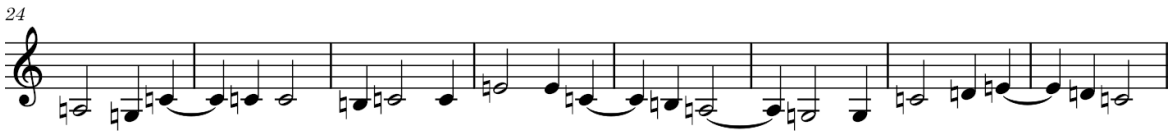
In [57]:

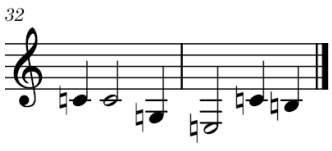
```
# Les mettre en format midi
songs_d = []
for i in range(10):
    songs_d.append(convert_to_midi(melodies_deutsch[i]))

for song in songs_d:
    song.show("midi")
    song.show()
songs_d[1].write("midi", "../generated_melodies/goodexamples/ahvousidraisjemaman_deutsch.midi")
```

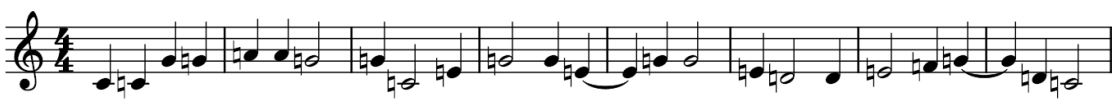




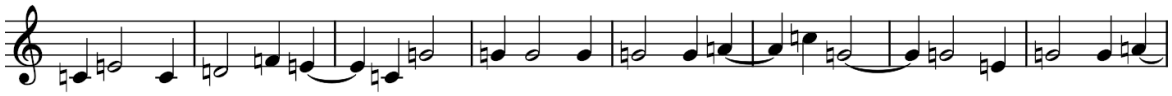




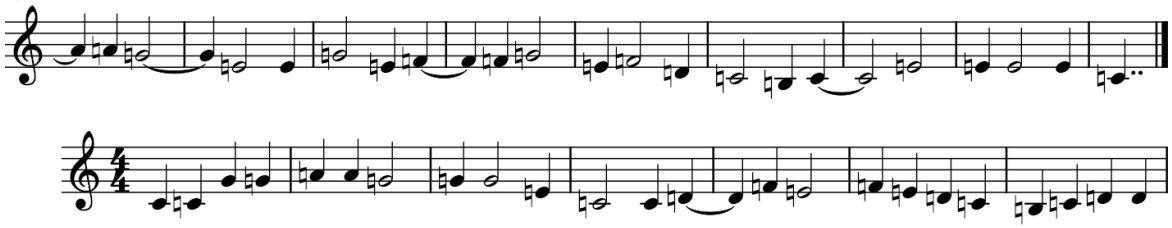




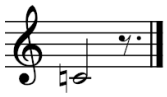
9

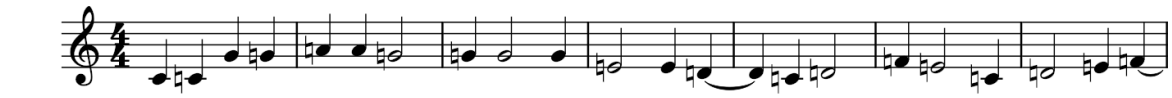


17

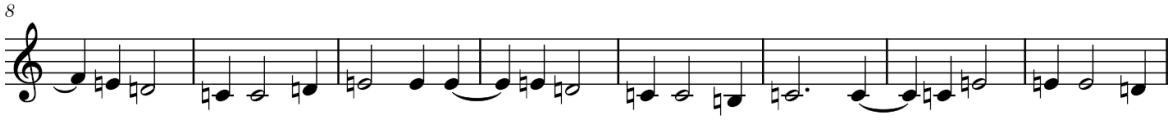


8

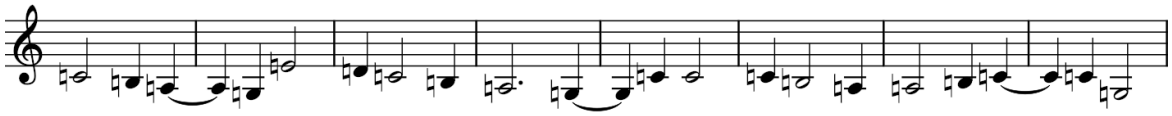




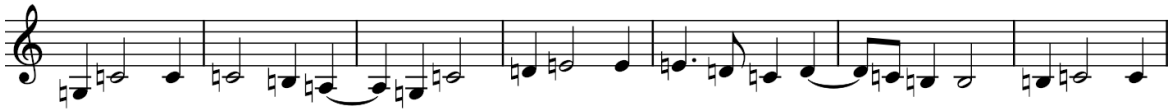
8



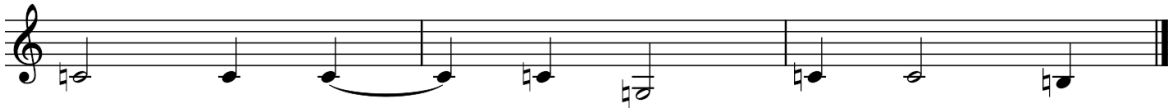
16

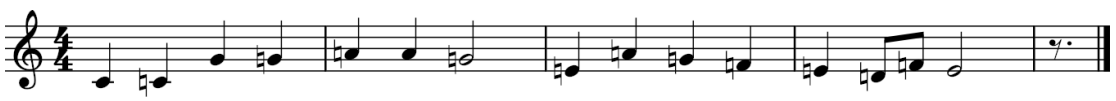


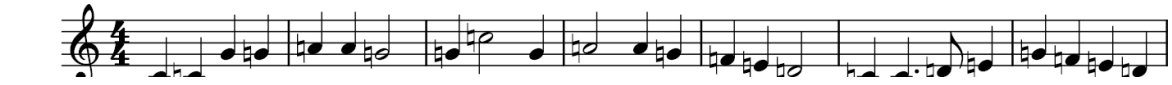
24



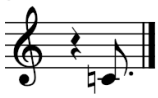
31

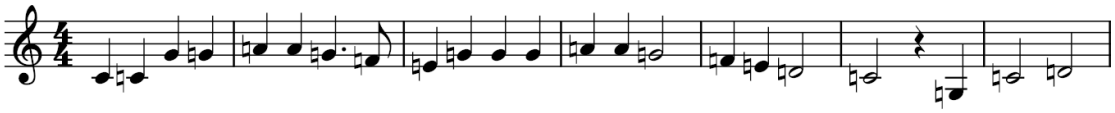




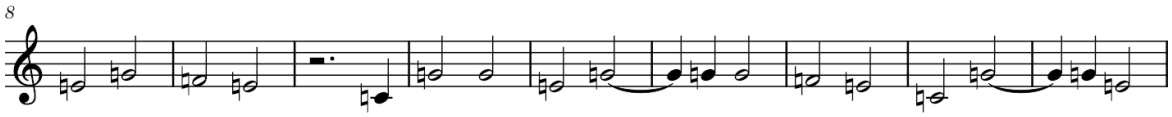


8

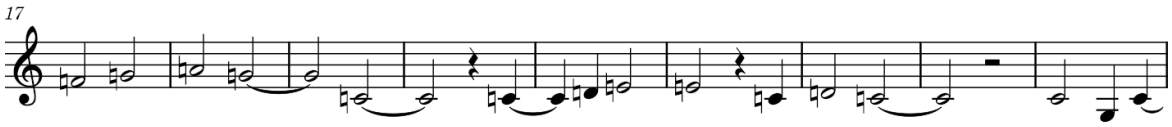




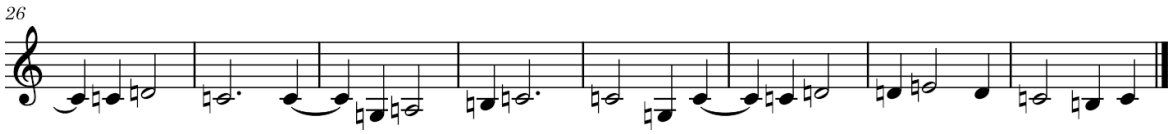
8

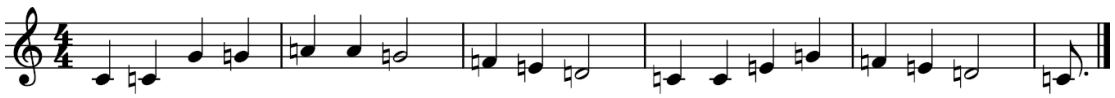


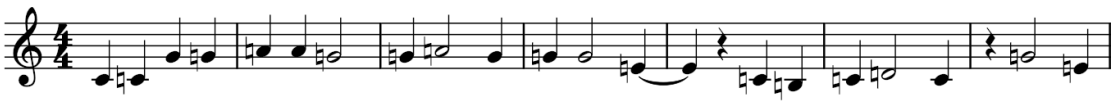
17



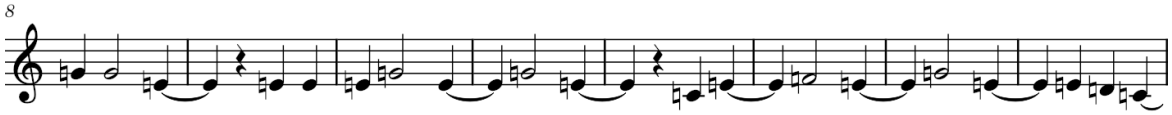
26



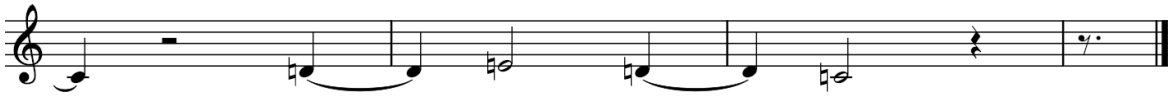


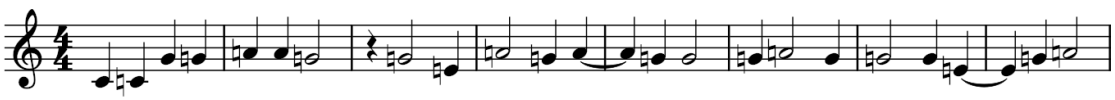


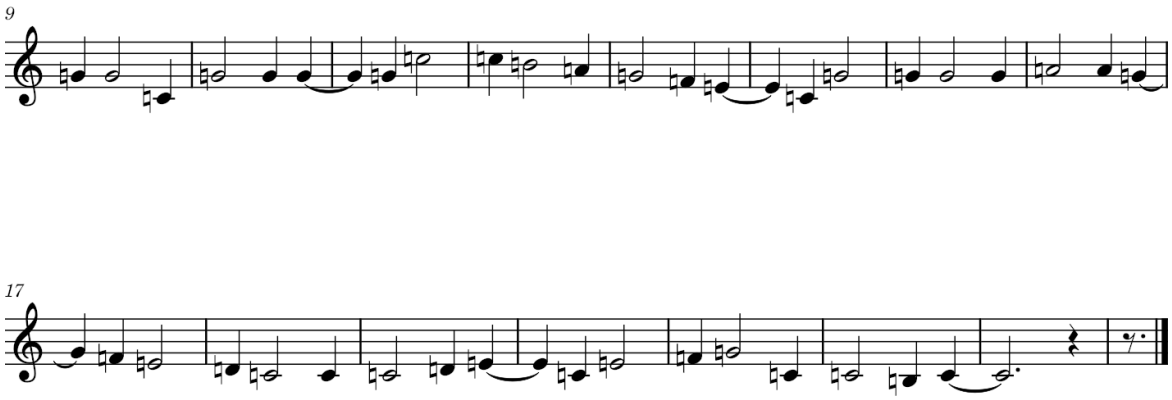
8



16







Out[57]:

'../generated_melodies/goodexamples/ahvousidraisjemaman_deutsch.midi'