**Государственное бюджетное общеобразовательное учреждение**

**г. Москвы Школа № 1434 «Раменки»**

**Программа генерирующая музыку**

**Автор работы:**

Ученик 10 класса Б ГБОУ Школы №1434 г. Москва

Фрик Евгений Владимирович.

**Руководитель работы:**

Учитель технологии ГБОУ Школы №1434 г. Москва

Чеушев Алексей Александрович.

**г. Москва, 2023 год.**

**Оглавление**

[Введение 3](#_Toc135250329)

[Анализ существующих подходов к реализации поставленной цели 4](#_Toc135250330)

[Описание теоретической части проекта 5](#_Toc135250331)

[Описание практической части проекта 6](#_Toc135250332)

[Основные результаты и выводы 7](#_Toc135250333)

[План дальнейших действий 8](#_Toc135250334)

[Список использованной литературы 9](#_Toc135250335)

# **Введение**

В школьном курсе информатики я изучил разные виды алгоритмов и язык программирования Python. Я узнал о работе языков программирования и их видах. Немного раньше я увлёкся написанием музыки и окончил музыкальную школу, тогда мне пришла идея создания программы, которая будет генерировать мелодии. Моя задача – разработать программу на языке Python, изучить результаты и оптимизировать её.

Таким образом, целью проекта на тему «Программа генерирующая музыку» является разработать компьютерную программу для генерации музыкальных мелодий.

Для достижения этой цели необходимо было решить следующие задачи:

1. Проведение анализа существующих решений.
2. Улучшение функционирования руки-манипулятора за счёт использования гироскопа.
3. Создание модели предлагаемого технического решения.
4. Проведение испытания модели.
5. Проанализировать полученные результаты.
6. Разработать предложения по дальнейшей деятельности.

# **Анализ существующих подходов к реализации поставленной цели**

Для исследования нам необходимо собрать модель руки-манипулятора

На мой взгляд существует несколько решений поставленной задачи:

|  |  |  |
| --- | --- | --- |
| Существующие решения | | Предлагаемое решение проекта |
| Генетический алгоритм | Нейронная сеть | Алгоритм случайного поиска |
| Ген. алгоритмы сложны и требуют сильно углублённые знания в области математики и информатики.  Их применение могло бы значительно увеличить эффективность программы, однако изучение этой темы заняло бы большое количество времени.  В результате данное решение не подходит для моего индивидуального проекта. | Нейросети требуют очень большую выборку уже готовых решений, которой, к сожалению, у меня нет.  Также нейросети требуют большое количество накопителей и высокопроизводительную технику.  По этим причинам данное решение также не подходит. | Это решение будет наилучшим благодаря относительно низкой сложности осуществления и отсутствия минусов нейросети.  С помощью данного алгоритма будет возможно в течение нескольких секунд создать готовый продукт даже на низкопроизводительных устройствах. |

# **Описание теоретической части проекта**

Для разработки программы генерирующей музыкальные мелодии требуется обладать некоторыми знаниями в музыкальной теории. При создании музыкальных мелодий стоит обратить внимание на такие факторы, как:

* Соответствие нот и тональностей.
* Соотношение длительностей нот.
* Длительность общей мелодии.
* Максимальная разница между высотой нот.
* Начальная нота композиции.

Для лучшего звучания генерируемых мелодий я остановился на том, что:

* Длинна каждой ноты будет одинаковая, так как при различных длительностях сгенерированную мелодию нельзя будет применить с любым аккомпанементом.
* Общая длительность мелодии будет задаваться в количестве нот пользователем.
* Разброс в высоте нот будет ограничен двумя октавами вверх от начальной ноты (то есть в сумме нота может быть сгенерирована либо в первой либо во второй октаве) .
* Начальная нота мелодии будет являться тоникой тональности, в которой генерируется мелодия.

Благодаря данным ограничениям генерируемая музыкальная мелодия должна звучать гармонично и приятно.

# **Описание практической части проекта**

Практическая часть проекта заключалась в разработке самой программы для генерации музыкальных мелодий.

Для её написания требовалось решить, какой язык программирования использовать, какие библиотеки могут помочь с выполнением задачи, как будет происходить взаимодействие пользователя с программой.

В качестве языка программирования я выбрал питон, благодаря наличию большого количества библиотек и простоте написания кода. Конечный файл с мелодией имеет формат .midi, так как это самый распространённый и удобный вариант, для этого я использовал библиотеку melodia.

Настройка параметров программы для генерации мелодии осуществляется через консоль при запуске файла программы.

Параметры для настройки включают в себя:

* Название итогового файла (name).
* Тональность (scale, есть поддержка всех 30 тональностей).
* Настройку длинны мелодии (length, измеряется в количестве нот).
* Использование паттернов (patterns, использование рисунков высот нот. В популярной музыке можно часто встретить повторяющиеся тона нот относительно друг друга, пользователь может попросить программу использовать их при генерации мелодии).

Также есть возможность использования стандартных параметров: название: «Melody by MelodyGenerator», тональность ля минор, длинна в восемь нот, и использование паттернов отключено.

Программа для генерации работает по принципу алгоритма случайного поиска. Для создания ноты сначала генерируется случайное число от нуля до семи и ещё одно число от нуля до одного (для этого потребовалась встроенная в питон библиотека random. Первое число – сама нота, второе же – октава, в которой нота будет находится. Затем с помощью таблицы тональностей определяется надо для заданной тональности поставить с нотой диез, бемоль или ничего. Таблица выглядит, как массив объектов состоящих из 9 полей. Первые два поля – названия тональностей, дальше идут семь полей, которые могут принимать значения от нуля до двух, где ноль – бемоль, единица – ничего (бекар), двойка – диез. После этого с помощью библиотеки melodia нота запаковывается в итоговый в midi файл. Этот процесс повторяется несколько раз, в зависимости от количества нот, которое попросил сгенерировать пользователь. В итоге получается файл с расширением .midi содержащий созданную программой мелодию.

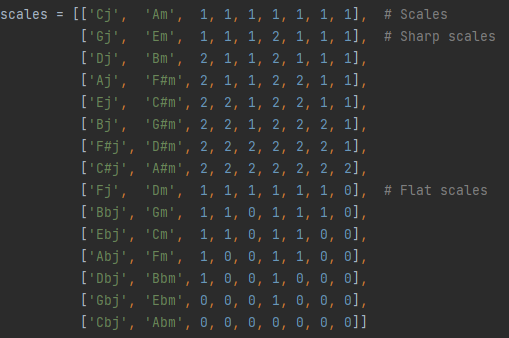


Рисунок 1: таблица тональностей в программе.

Вот код программы:

# imports  
  
from melodia.core import Track  
from melodia.core import Tone   
from melodia.io import midi  
import melodia  
import random as r  
  
  
# melodia

# melodia.io.midi.dump()

# name is in the end  
track = Track(signature=(4, 4))  
  
  
# variables  
  
cleanscales = ['Cj', 'Am', 0] # Clean C major and A minor  
# Cj means C major  
# Am means A minor  
# Small b means flat  
# # means sharp  
  
# 0 is flat  
# 1 is nothing  
# 2 is sharp  
  
scales = [['Cj', 'Am', 1, 1, 1, 1, 1, 1, 1], # Scales  
 ['Gj', 'Em', 1, 1, 1, 2, 1, 1, 1], # Sharp scales  
 ['Dj', 'Bm', 2, 1, 1, 2, 1, 1, 1],  
 ['Aj', 'F#m', 2, 1, 1, 2, 2, 1, 1],  
 ['Ej', 'C#m', 2, 2, 1, 2, 2, 1, 1],  
 ['Bj', 'G#m', 2, 2, 1, 2, 2, 2, 1],  
 ['F#j', 'D#m', 2, 2, 2, 2, 2, 2, 1],  
 ['C#j', 'A#m', 2, 2, 2, 2, 2, 2, 2],  
 ['Fj', 'Dm', 1, 1, 1, 1, 1, 1, 0], # Flat scales  
 ['Bbj', 'Gm', 1, 1, 0, 1, 1, 1, 0],  
 ['Ebj', 'Cm', 1, 1, 0, 1, 1, 0, 0],  
 ['Abj', 'Fm', 1, 0, 0, 1, 1, 0, 0],  
 ['Dbj', 'Bbm', 1, 0, 0, 1, 0, 0, 0],  
 ['Gbj', 'Ebm', 0, 0, 0, 1, 0, 0, 0],  
 ['Cbj', 'Abm', 0, 0, 0, 0, 0, 0, 0]]  
  
# до ре ми фа соль ля си  
keys = ['C', 'D', 'E', 'F', 'G', 'A', 'B']  
keysstr = 'CDEFGAB'  
  
  
# functions  
  
def genmelody(scale, length, usepatterns):  
 print('Starting generation')  
 key = Tone.from\_notation(scale[:-1] + '3').pitch  
 workingscale = scales[0]  
 for i in range(15):  
 if scale == scales[i][0] or scale == scales[i][1]:  
 workingscale = scales[i]  
 if (usepatterns == 'Yes' or usepatterns == 'yes') and length % 8 != 0:  
 print('Error! Length should be divisible by 8 if you want to use patterns.')  
 return 0  
  
 elif usepatterns == 'Yes' or usepatterns == 'yes':  
 print("patterns")  
 track.add(melodia.core.note.Note(key, (1, 4)))  
 track.add(melodia.core.note.Note(key-4, (1, 4)))  
 track.add(melodia.core.note.Note(key, (1, 4)))  
 track.add(melodia.core.note.Note(key-4, (1, 4)))  
 for i in range(int(length/8) - 1):  
 workingscale = scales[0]  
 ns = ''  
 for j in range(4):  
 n = r.randint(0, 6)  
 if workingscale[2 + n] == 0:  
 ns = chr(n + ord('A')) + 'b' + str(3)  
 elif workingscale[2 + n] == 1:  
 ns = chr(n + ord('A')) + str(3)  
 else:  
 ns = chr(n + ord('A')) + '#' + str(3)  
 track.add(melodia.core.note.Note(ns, (1, 4)))  
 track.add(melodia.core.note.Note(Tone.from\_notation(ns).pitch-4, (1, 4)))  
 track.add(melodia.core.note.Note(Tone.from\_notation(ns).pitch, (1, 4)))  
 track.add(melodia.core.note.Note(Tone.from\_notation(ns).pitch-4, (1, 4)))  
 track.add(melodia.core.note.Note(Tone.from\_notation(ns).pitch, (1, 4)))  
 for j in range(4):  
 n = r.randint(0, 6)  
 o = r.randint(0, 1) # Notes could be generated between 2 octaves so that the melody will sound better  
 if workingscale[2 + n] == 0:  
 ns = chr(n + ord('A')) + 'b' + str(3 + o)  
 elif workingscale[2 + n] == 1:  
 ns = chr(n + ord('A')) + str(3 + o)  
 else:  
 ns = chr(n + ord('A')) + '#' + str(3 + o)  
 track.add(melodia.core.note.Note(ns, (1, 4)))  
  
 else:  
 track.add(melodia.core.note.Note(key, (1, 4)))  
 for i in range(length-1):  
 n = r.randint(0, 6)  
 o = r.randint(0, 1) # Notes could be generated between 2 octaves so that the melody will sound better  
 if workingscale[2 + n] == 0:  
 ns = chr(n+ord('A')) + 'b' + str(3+o)  
 elif workingscale[2 + n] == 1:  
 ns = chr(n+ord('A')) + str(3+o)  
 else:  
 ns = chr(n+ord('A')) + '#' + str(3+o)  
 track.add(melodia.core.note.Note(ns, (1, 4)))  
 print('Melody is generated!')  
  
  
# main  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 print('Use standard parameters?')  
 t = input()  
 if t != 'Yes' and t != 'yes':  
 print('Enter parameters:')  
 print('Name is: ')  
 name = input()  
 print('Scale is: ') # Major is written as j  
 scale = input()  
 print('Length is: ')  
 length = int(input()) # Amount of notes in the melody  
 print('Use patterns or not: ') # Length should be divisible by 8 if you want to use patterns  
 usepatterns = input()  
 else:  
 name = 'Melody by MelodyGenerator'  
 scale = 'Am'  
 length = 8  
 usepatterns = 'No'  
  
 genmelody(scale, length, usepatterns)  
  
 with open(name + '.mid', 'wb') as f:  
 midi.dump(track, f)

# **Основные результаты и выводы**

В результате работы были проведены такие действия, как:

1. Разработана программа для генерации музыкальных мелодий на языке программирования Python.
2. Сгенерированы музыкальные мелодии для проверки работоспособности программы.
3. Отобраны лучшие из сгенерированных мелодий, в качестве примера работы программы.
4. Получены и проанализированы результаты опытов.
5. Разработаны предложения по дальнейшей деятельности.

В результате реализация данного проекта был наглядно показан процесс функционирования алгоритма для генерации музыки и на основе результатов испытаний можно предложить их использование начинающими музыкальными продюсерами, для более быстрого обучения и общего распространения музыки.

# **План дальнейших действий**

В дальнейшем предполагается разработать интерфейс для более простого взаимодействия пользователя с программой, добавить дополнительные параметры для более гибкой настройки генерации мелодии, улучшить механизм генерации, возможно разработать генетический алгоритм для этой цели.

# **Список использованной литературы**

1. Общая информация, основные термины: <https://ru.wikipedia.org/wiki/>
2. Музыкальная теория: <https://music-education.ru/>
3. Использованная библиотека melodia для создания файлов с расширением .midi: https://github.com/meownoid/melodia