PROJ2

Rozpoznawanie akordów gitarowych

Sprawozdanie końcowe

Zespół Dżejsomat:

Jakub Grzechnik **325278**

Jakub Strzelczyk **325325**

Marcin Dudek **329066**

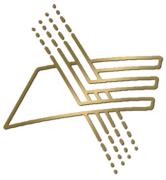
Kacper Karczmarek **325284**

Oliwia Borkowska **325254**

Opiekun projektu:

dr inż. Maciej Sypniewski

16 czerwca 2024



Wydział Elektroniki
i Technik Informacyjnych

POLITECHNIKA WARSZAWSKA

Spis treści

1. Zakres i cel projektu

Celem projektu jest:
— Stworzenie pliku z akordami i oznaczeniem „bicia" gitarowego (rozpoznanie standardów formatów plików) dla zagranego na gitarze prostego utworu muzycznego
— Poznanie/rozwinięcie umiejętności programowania aplikacji z wykorzystaniem wybranej technologii

2. Organizacja pracy

2.1. Uzgodnienie metody zarządzania projektem

W celu optymalnego zarządzania projektem oraz adaptacyjnego rozwiązywania problemów uzgodniliśmy zgodnie z prowadzącym, iż użyjemy metody Scrum z pewnymi uproszczeniami.

2.2. Role członków zespołu

Ustaliliśmy role poszczególnych członków Scrum Teamu:

— **Właściciel produktu** - Jakub Grzechnik
— **Scrum Master** - Oliwia Borkowska
— **Deweloperzy** - Jakub Strzelczyk, Marcin Dudek, Kacper Karczmarek

2.3. Narzędzia wspomagające prowadzenie projektu

Przystąpiliśmy do określenia narzędzi oraz technologii wykorzystywanych w pracy nad projektem. Do zaprogramowania aplikacji wykorzystamy język Python - jest stosunkowy prosty, oferuje bogaty zestaw wbudowanych bibliotek, z których możemy skorzystać, a także mamy już doświadczenie w jego użyciu z poprzednich semestrów.

— **Trello** [1] - utworzyliśmy tablicę na platformie Trello. Na tej tablicy można efektywnie zarządzać zadaniami i projektami. Możemy dodawać karty reprezentujące różne zadania, przypisywać je do członków zespołu, ustawiać terminy wykonania, a także śledzić postęp prac. Dzięki Trello z łatwością organizujemy nasze projekty, co przyczynia się do efektywnej współpracy i lepszego zarządzania czasem.

— **GitLab** [2] - utworzyliśmy repozytorium na platformie Gitlab w celu umożliwienia współpracy zespołowej nad kodem źródłowym oraz śledzenia zmian. GitLab to platforma do zarządzania projektami oparta na systemie kontroli wersji Git. Umożliwia ona współpracę zespołową nad kodem źródłowym, śledzenie zmian, zarządzanie zadaniami i wiele innych funkcji związanych z cyklem życia projektu.

— **Doodle** [3] - następnym wybranym przez nas narzędziem, którego użyliśmy do wspomagania prowadzonego przez nas projektu jest narzędzie Doodle. Zdecydowaliśmy, że to narzędzie będzie użyteczne i skuteczne, ponieważ umożliwia elastyczne planowanie spotkań i terminów, zarządzanie harmonogramem, konsultacje i dyskusje, przypomnienia i notyfikacje oraz dostępna jest opcja analizy dostępności czasowej, która w naszym przypadku jest kluczowa.

— **Microsoft Teams** - do ułatwienia komunikacji, korzystaliśmy z zespołu projektowego założonego na platformie Microsoft Teams. Umożliwiło nam to organizowanie spotkań online oraz dzielenie się plikami. Teams stał się centralnym miejscem dla naszej komunikacji i koordynacji działań. Dzięki funkcjom takim jak chat, wideokonferencje, udostępnianie plików, a także integracja z innymi narzędziami, skutecznie zarządzaliśmy komunikacją w zespole oraz śledziliśmy postęp prac.

2.4. Uzgodnienie rytmu i formy spotkań zespołu

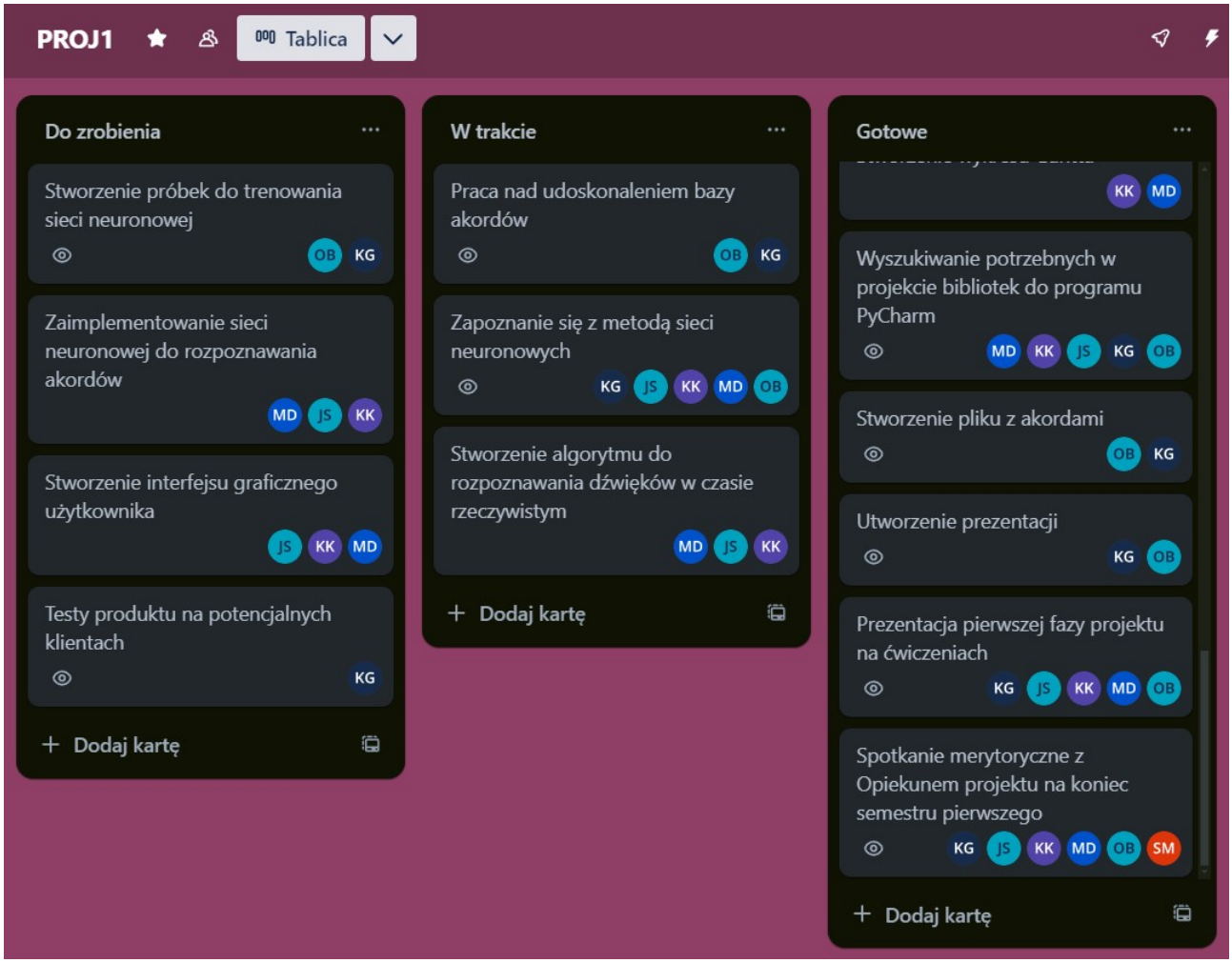
Spotkania bez udziału Opiekuna w procesie projektowym obejmują kluczowe etapy planowania i analizy Sprintów. Na początku każdego Sprintu odbywa się spotkanie Planowania Sprintu, podczas którego ustalamy zadania do realizacji. Następnie, spotkanie Daily Scrum pozwala na śledzenie postępu prac, dostosowując częstotliwość do potrzeb zespołu.

Po zakończeniu Sprintu przeprowadzamy Retrospektywę Sprintu, analizując przebieg i identyfikując obszary do poprawy, co umożliwia ciągłe doskonalenie procesów.

Spotkania z Opiekunem, zwane Przeglądem Sprintu, odbywają się co 2-4 tygodnie. Prezentujemy na nich efekty pracy w danym okresie, dostosowując formę do potrzeb zespołu i charakteru projektu. Ta struktura spotkań pozwala efektywnie zarządzać projektem, utrzymywać transparentność i dostosowywać działania do dynamicznych wymagań projektowych.

2.5. Monitorowanie pracy zespołu

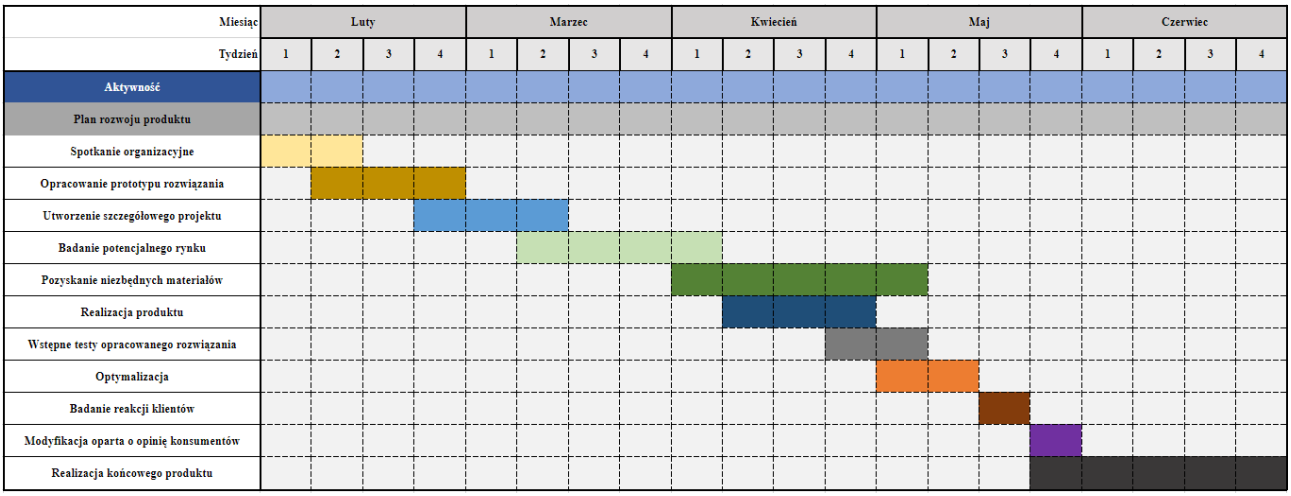
Tak jak już wcześniej wspomnieliśmy, do regularnego monitorowania postępu prac poszczególnych członków zespołu używamy aplikacji Trello. Link do naszej tablicy: Tablica_PROJ1. Aktywność naszego zespołu można obserwować również na kanale **"[T22] Rozpoznawanie akordów gitarowych"** w aplikacji Microsoft Teams.



Rysunek 1. Widok tablicy na Trello

2.6. Wykres Gantta

Zaprojektowaliśmy wykres Gantta, przedstawiający nasz plan realizacji projektu na drugi semestr.



Rysunek 2. Wykres Gannta

3. Podobne rozwiązania dostępne na rynku komercyjnym

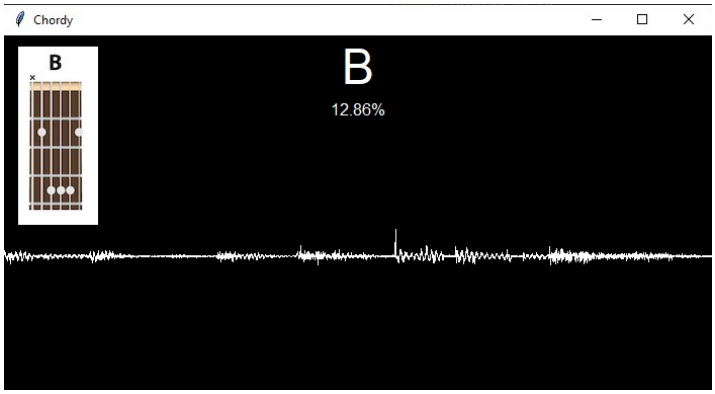
3.1. Opis aplikacji

Na platformie internetowej GitHub udało nam się znaleźć bardzo interesujące rozwiązanie. Aplikacja "Chordy" została zaprojektowana z wykorzystaniem trzech oddzielnych wątków, które skutecznie izolują procesy przesyłania strumieniowego audio, rozpoznawania akordów oraz renderowania interfejsu użytkownika (GUI). Każdy z tych wątków ma swoją specyficzną rolę, co umożliwia płynne działanie aplikacji. Dodatkowo, zastosowano kolejki do zarządzania danymi, co wprowadza dodatkową warstwę izolacji i koordynacji między wątkami.

Aby przekazać strumień audio z mikrofonu do kolejki fragmentów, "Chordy" wykorzystuje bibliotekę PyAudio. Ta biblioteka umożliwia płynne przesyłanie danych dźwiękowych do aplikacji, a następnie umieszczenie ich w kolejce do dalszego przetwarzania. Dodatkowo, aplikacja "Chordy" wykorzystuje bibliotekę SciPy do przeprowadzania ponownego próbkowania szeregów czasowych, co może wpłynąć na poprawę jakości analizy dźwięku.

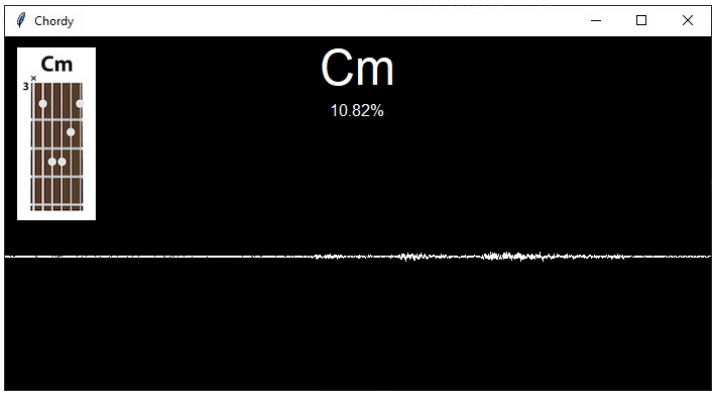
3.2. Testowanie

Przeprowadziliśmy kilka testów w celu sprawdzenia poprawności działania aplikacji.



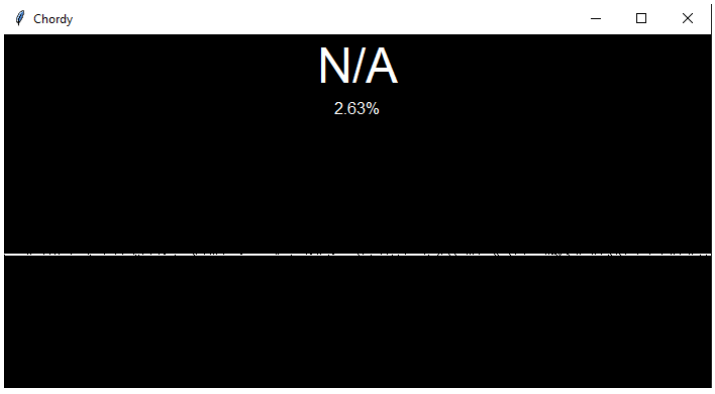
Rysunek 3. Akord B

W pierwszym teście odtworzyliśmy za pomocą telefonu akord "B". Aplikacja poprawnie zidentyfikowała zagrany akord. Wyświetlone zostało również natężenie rozpoznanego dźwięku, które zostało określone na "12,86%".



Rysunek 4. Akord Cm

Następnie odtworzyliśmy akord "Cm", aplikacja ponownie poprawnie zidentyfikowała zagrany akord, a natężenie dźwięku zostało określone na "10,82%". Niskie natężenie wynika z niskiej jakości używanego przez nas mikrofonu.



Rysunek 5. Akord NA - bezdźwięk

W ramach ostatniego testu nie odtworzyliśmy żadnego akordu. Aplikacja w czasie rzeczywistym poprawnie informuje, że nie jest odgrywany żaden akord. Widoczne, niskie natężenie dźwięku jest spowodowane szumem występującym w otoczeniu.

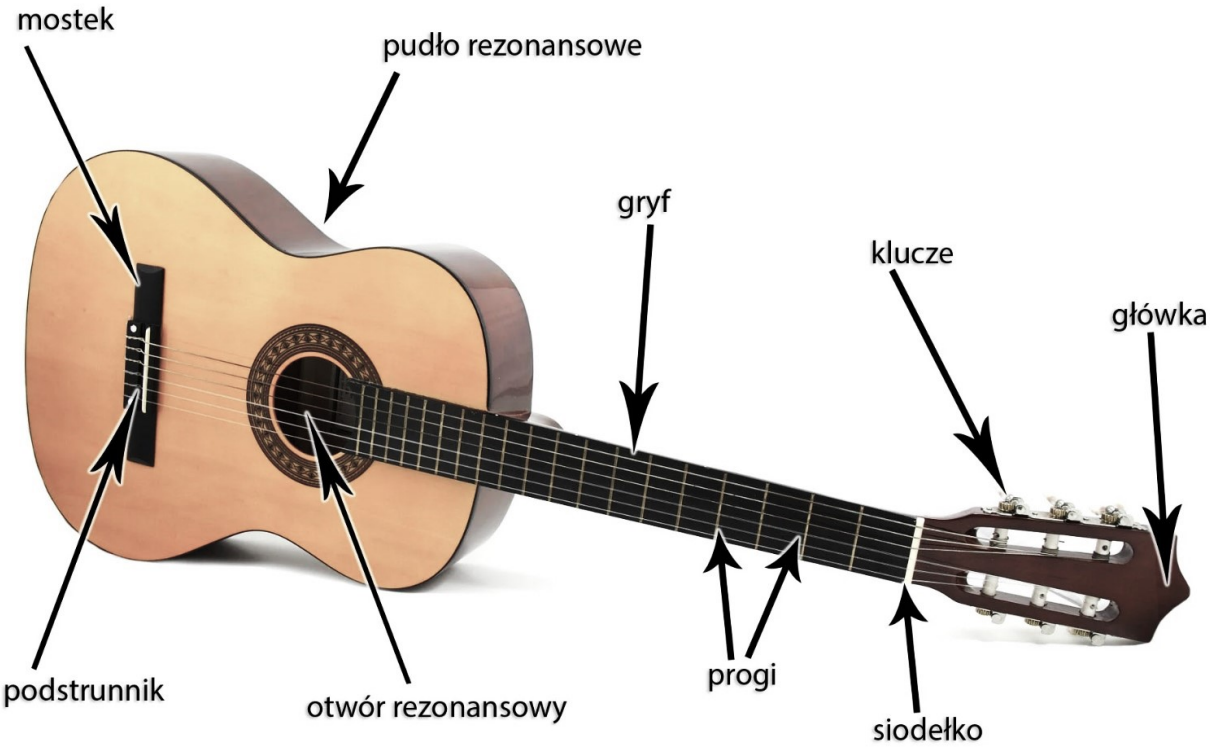
4. Zapoznanie z podstawami teorii muzyki

4.1. Podstawy teorii muzyki

Kluczową cechą sygnału dźwiękowego, istotną przy ocenie poprawności odtwarzanego dźwięku na gitarze, jest jego wysokość, wyrażona poprzez częstotliwość podstawową sygnału. Dźwięki składające się z czystych tonów, czyli sygnałów sinusoidalnych są połączone z harmonicznymmi wydźwiękami oraz mają konkretne częstotliwości drgań, wybrane spośród wszystkich możliwych odbieranych przez ludzkie ucho (przedział 20Hz - 20kHz).
Obecnie w muzyce używa się dźwięków od C0 o częstotliwości 16Hz do C8 o częstotliwości 4186Hz. Nuty w gamie muzycznej są zdefiniowane przez stałe proporcje, a standardową częstotliwością jest dźwięk A4 (440Hz), dzięki której możemy określić rodzaj tonu, czy strojenia.

4.2. Akustyka gitary

Aby prawidłowo zrealizować nasz projekt musieliśmy również zapoznać się ze schematem budowy gitary przedstawionym poniżej:



Rysunek 6. Budowa gitary [4]

Pudło rezonansowe gitary służy do wzmacniania dźwięku wywoływanego przez struny. Na gryfie umieszczone są progi prostopadle do strun. Ich główną funkcją jest skracanie efektywnej długości drgającej struny, co prowadzi do zmiany częstotliwości podstawowej wydobywanych dźwięków. Za pomocą kluczy możemy zmieniać napięcie strun tak, aby dostosować ich częstotliwość.

W naszym projekcie przyjmiemy najbardziej popularny strój gitary, w którym struny mają określone dźwięki. Zaczynając od najgrubszej struny o najniższej częstotliwości:
— Pierwsza struna - dźwięk E2 o częstotliwości 82.4Hz
— Druga struna - dźwięk A2 o częstotliwości 110.0Hz
— Trzecia struna - dźwięk D3 o częstotliwości 146.8Hz
— Czwarta struna - dźwięk G3 o częstotliwości 196.0Hz
— Piąta struna - dźwięk B3 o częstotliwości 246.9Hz
— Szósta struna - dźwięk E4 o częstotliwości 329.6Hz

5. Metoda analizy widmowej

5.1. Wykorzystywane biblioteki

Ustaliliśmy, że pomocne w realizacji projektu mogą być gotowe biblioteki dostępne w Python:

— **NumPy** [5] **:** NumPy może być używane w kontekście rozpoznawania akordów gitarowych, zwłaszcza do manipulacji i analizy danych dźwiękowych. Dodatkowo NumPy może okazać się przydatne do przetwarzania macierzowego, co jest ważne przy analizie danych dźwiękowych, na przykład w przypadku stosowania transformacji Fouriera. Ponadto zawiera funkcję numpy.hamming, która służy do generowania okna Hamminga. Okna Hamminga są często używane w przetwarzaniu sygnałów, analizie widma i innych dziedzinach inżynierii dźwięku oraz sygnałów.

— **Math** [6] **:** Biblioteka "math" może być używana w celu realizacji pewnych podstawowych operacji matematycznych, które są istotne przy analizie sygnałów dźwiękowych. Ta biblioteka umożliwia obliczanie modułu wartości liczby zespolonej, czy też modułu sygnałów, gdzie w przypadku analizy sygnałów dźwiękowych, próbki są reprezentowane w ten sposób. Po obliczeniu modułu sygnałów, można porównać te wartości z określonym progiem oraz zidentyfikować pożądane próbki. Zidentyfikowany szum może zostać wyeliminowany, a z sygnału można uzyskać najistotniejsze informacje.

— **Time** [7] **:** W kontekście rozpoznawania akordów gitarowych, biblioteka "time" skupia się na obszarze pomiaru czasu i zarządzaniu czasem wykonywania operacji. Biblioteka może zostać wykorzystana do pomiaru czasu trwania wykonania określonych fragmentów kodu. Ponadto funkcje tej biblioteki pozwalają na kontrolowanie opóźnienia między operacjami, co może być istotne w procesie analizy sygnałów dźwiękowych.

— **PyAudio** [8] **:** stanowi narzędzie do realizacji operacji związanych z dźwiękiem. Jest niezwykle przydatna w dziedzinie rozpoznawania akordów gitarowych, zapewniając narzędzia do manipulacji i analizy danych dźwiękowych. Biblioteka umożliwia inicjalizację i konfigurację strumieni audio do nagrywania i odtwarzania dźwięku. Dzięki temu można efektywnie zarządzać procesem obsługi dźwięku, dostosowując parametry, takie jak częstotliwość próbkowania, dla optymalnej kontroli nad jakością i charakterem dźwięku. Biblioteka ta jest niezbędnym narzędziem dla projektów związanych z przetwarzaniem dźwięku oraz eksploracją muzycznych aspektów danych audio.

— **Matplotlib** [9] **:** jest to biblioteka używana do wizualizacji danych. Umożliwia tworzenie wykresów, personalizację ich wyglądu oraz zaawansowaną wizualizację. Jest niezwykle użyteczna w dziedzinie analizy danych, umożliwiając efektywne przedstawienie infomracji w formie graficznej.

5.2. Planowane działanie i wygląd aplikacji

Rysunek 7. Planowana struktura interfejsu graficznego

Głównym zadaniem aplikacji jest rozpoznawanie akordów gitarowych w czasie rzeczywistym. Program będzie wyświetlał grany akord oraz weryfikował poprawność jego zagrania w procentach (0% - akord źle zagrany , 100% - akord zagrany idealnie). Aplikacja będzie pokazywała graficznie jak wygląda ułożenie palców na gryfie gitary dla danego akordu. Poza podstawową funkcjonalnością aplikacja będzie pokazywała kierunek bitego akordu, tzn. czy akord jest grany od góry czy od dołu. Będzie widać z jakim tempem jest grany akord. Dodatkowo będzie możliwość wyboru prostego utworu gitarowego do zagrania. Wtedy wyświetli się tekst utworu i akordy które trzeba zagrać. Aplikacja będzie informowała użytkownika o tym czy akord jest grany za szybko, za wolno czy z odpowiednim tempem.

5.3. Baza akordów

Pierwsza wesja naszej bazy danych pokazana poniżej zawiera 24 akordy. Każdy z nich zawiera sześć częstotliwości odpowiadających wysokościom dźwięków na odpowiednich strunach gitary. Aby zobrazować schemat naszej bazy, kolorami zostały oznaczone struny, na których dana częstotliwość występuje.

database.txt

```
A,329.60,277.14,220.00,164.78,110.00,82.40
A7,329.60,277.14,196.00,164.78,110.00,82.40
Am,329.60,261.58,220.00,164.78,110.00,82.40
Am7,329.60,261.58,196.00,164.78,110.00,82.40
Amaj7,329.60,277.14,207.65,164.78,110.00,82.40
Bb,349.20,293.61,233.08,130.81,110.00,82.40
B7,369.96,246.90,220.00,155.53,123.47,82.40
Bm,369.96,293.61,246.94,184.96,110.00,82.40
C,329.60,261.58,196.00,164.78,130.81,97.991
C7,329.60,261.58,233.08,164.78,130.81,82.40
Cmaj7,329.60,246.90,196.00,164.78,130.81,82.40
D,369.96,293.61,220.00,146.80,110.00,82.40
D7,369.96,261.58,220.00,146.80,110.00,82.40
Dm,349.20,293.61,220.00,146.80,110.00,82.40
Dm7,349.20,261.58,220.00,146.80,110.00,82.40
Dmaj7,369.96,277.14,220.00,146.80,110.00,82.40
E,329.60,246.90,207.65,164.78,123.47,82.40
E7,329.60,246.90,207.65,146.80,123.47,82.40
Em,329.60,246.90,196.00,164.78,123.47,82.40
Em7,329.60,293.61,196.00,164.78,123.47,82.40
F,349.20,261.58,220.00,130.81,110.00,82.40
Fmaj,329.60,261.58,220.00,130.81,110.00,82.40
G,369.96,246.90,196.00,146.80,123.47,97.991
G7,349.20,246.90,196.00,146.80,123.47,97.991
```

Rysunek 8. Baza akordów

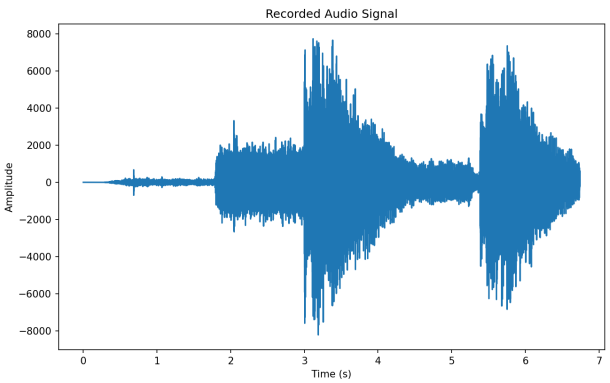
Rysunek 9. Widok strun gitary

Jednak po dyskusji w szerszym gronie odbiorców, poznaliśmy wiele innych perspektyw przechowywania informacji o akordach. W przyszłym rozwiązaniu naszego projektu z użyciem sieci neuronowych przewidujemy liczne modyfikacje struktury i treści naszej bazy.

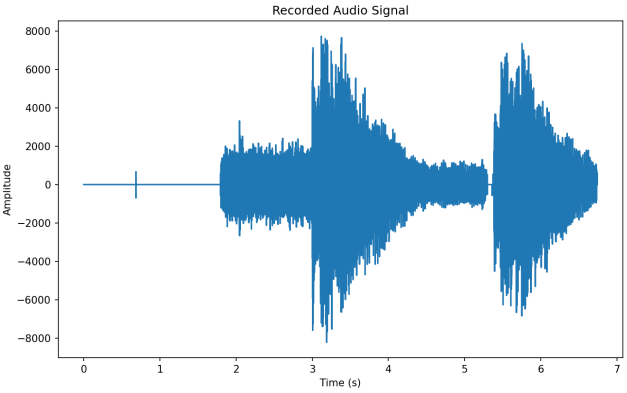
5.4. Działanie aplikacji

Działanie naszej aplikacji przed zakończeniem bieżącego semestru polega na możliwości nagrania pojedycznego akordu i jego rozpoznania.

Nasz program za pomocą biblioteki pyAudio rejestruje dźwięk, próbkuje z odpowiednią częstotliwością, która nie powoduje zjawiska aliasingu - my ustawiliśmy fs = 44kHz. Następnie zeruje on wartości próbek, których moduł amplitudy nie przekracza progu, który uznaliśmy jako szum otoczenia. Poniżej przedstawiamy wykresy prezentujące sygnał przed odszumieniem i po odszumieniu.



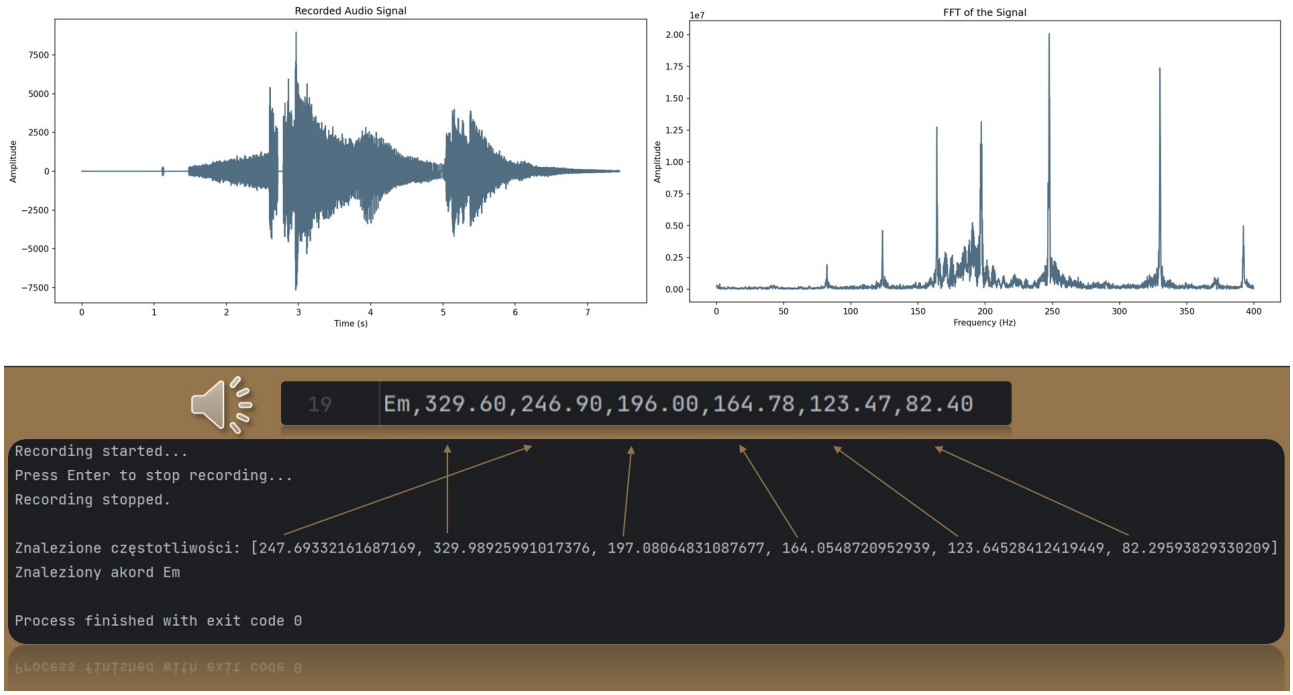
Rysunek 10. Sygnał przed odszumieniem



Rysunek 11. Sygnał po odszumieniu

Następnie za pomocą biblioteki numPy na nasz sygnał nakładamy okno Hamminga w celu uniknięcia wycieków widma i przekształcamy go w dziedzinę częstotliwości, korzystając z numpy.fft.

Napisaliśmy funkcję, która bada częstotliwości, dla których amplituda jest największa i porównuje je z tymi występującymi w bazie danych. Na podstawie najlepszego dopasowania na ekran zostaje wypisany znaleziony akord.



Rysunek 12. Wynik działania aplikacji

6. Metoda sieci neuronowych

Korzystając z sugestii opiekuna, a także prowadząc spotkania z członkami zespołów zdecydowaliśmy, iż lepszym rozwiązaniem od analizy widmowej sygnału będzie skorzystanie z sieci neuronowych do rozpoznania akordów. Takie podejście umożliwiłoby nam szybsze, efektywniejsze, a także dokładniejsze rozpoznanie akordu w czasie rzeczywistym. Wykorzystując odpowiednie dane treningowe wytrenujemy naszą sieć, tak aby aplikacja rozpoznawała akordy grane na różnych typach gitar tj. akustycznej, klasycznej oraz elektrycznej.

Rysunek 13. Grafika poglądowa wygenerowana przez AI

6.1. Definicja kamienia milowego

Kamień milowy, czyli poziom zaawansowania naszego projektu jaki chcielibyśmy osiągnąć do ustalonego terminu określamy na:

— utworzenie kompletnej bazy akordów zawierającej nagrania podstawowych chwytów gitarowych zagranych na wiele sposobów przy użyciu gitar akustycznej oraz klasycznej w celu zastosowania jej jako dane treningowe
— wytrenowanie sieci neuronowej na podstawie danych treningowych, umożliwiające uzyskanie jak najlepszej dokładności rozpoznawania danych testowych na podstawie wprowadzonych cech sygnału
— umożliwienie rozpoznawania akordów za pomocą wytrenowanej sieci neuronowej w czasie rzeczywistym

Termin kamienia milowego - **21 kwietnia 2024**

6.2. Realizacja kamienia milowego

6.2.1. Baza akordów

Stworzyliśmy bazę akordów, do której dodaliśmy próbki z repozytorium [10] zaproponowanego przez Opiekuna projektu. Zawiera ona 12 podstawowych akordów, jakimi są: A, am, C, D, dm, E, em, F, fm, G, H, hm. Każdy z nich jest reprezentowany przez 200 próbek, dzięki czemu każdy akord przedstawiony jest w różnych kontekstach brzmieniowych, grany różnymi technikami i w różnorodnych stylach.

Dodatkowo, wzbogadziliśmy naszą bazę akordów o dodatkowe 120 próbek, po 10 dla każdego akordu. Nagraliśmy je własnoręcznie, z użyciem róznorodnych technik, aby zwiększyć zróżnicowanie i pełniej oddać charakter każdego akordu. W rezultacie nasza baza składa się łącznie z 2520 próbek. Jest wszechstronnym zbiorem, który umożliwia dokładną analizę brzmienia każdego akordu w różnych kontekstach muzycznych, co zapewnia szerokie zastosowanie w praktycznych badaniach akustycznych.

6.2.2. Analiza próbek dźwiękowych

Zanim przystąpiliśmy do pracy zapoznaliśmy się z możliwymi metodami przetworzenia naszych próbek dźwiękowych, abyśmy mogli na ich podstawie wytrenować sieć neuronową.

— **MFCC (Mel-Frequency Cepstral Coefficients)** [11] to współczynniki, które reprezentują krótko-czasową moc widmową sygnału akustycznego na skali Mel. Skala Mel jest nieliniową skalą częstotliwości, która jest bardziej zgodna z ludzkim postrzeganiem wysokości dźwięku.
— **PCP (Pitch Class Profile)** [12] analizuje zawartość widmową dźwięku, aby zidentyfikować intensywność różnych klas wysokości dźwięku w czasie. Każda z 12 klas wysokości odpowiada jednej z dwunastu półtonów w skali muzycznej (od C do H).
— **Chroma CENS (Chroma Energy Normalized Statistics)** [13] to zmodyfikowana wersja chromagramu, czyli PCP, która jest bardziej odporna na zmiany tempa i dynamiki. Może być używana do analizy tonacji i harmonii.
— **Spectral Contrast** [14] mierzy różnice w poziomach energii między szczytami a dolinami w spektrum dźwiękowym. Jest używana do analizy barwy dźwięku.
— **Zero-Crossing Rate** [15] to miara liczby punktów, w których sygnał dźwiękowy zmienia znak (przechodzi przez zero). Jest używana do analizy rytmu i barwy dźwięku.

Sprawdziliśmy kilka z tych metod i najbardziej efektywna na obecny moment okazała się metoda PCP. Na podstawie utworzonej bazy akordów utworzyliśmy plik w formacie .json, który przechowuje informacje o zawartości próbek. W procesie tworzenia wspomogliśmy się sztuczną inteligencją ChatGPT [16].

Wypisz kroki potrzebne do stworzenia pliku w formacie .json na podstawie próbek dźwiękowych przy użyciu metody pcp.

Aby stworzyć plik w formacie .json na podstawie próbek dźwiękowych przy użyciu metody pcp (pitch class profile), wykonaj następujące kroki:

1. **Importowanie bibliotek:**
   - Upewnij się, że masz zainstalowane niezbędne biblioteki, takie jak `librosa`, `numpy`, `json` oraz `os`.
2. **Definiowanie ścieżek i parametrów:**
   - Określ ścieżkę do katalogu z próbkami dźwiękowymi oraz ścieżkę do pliku .json, w którym zapiszesz dane.
   - Zdefiniuj częstotliwość próbkowania (np. 22050 Hz).
3. **Wczytywanie sygnału dźwiękowego:**
   - Użyj `librosa.load(file_path)`, aby wczytać sygnał dźwiękowy z pliku.
4. **Ekstrakcja cech dźwiękowych:**
   - Użyj funkcji `librosa.feature.chroma_stft` do ekstrakcji chromagramu z sygnału dźwiękowego.
5. **Normalizacja cech:**
   - Znormalizuj wartości chromagramu do przedziału [0, 1] za pomocą `librosa.util.normalize`.
6. **Obliczenie profilu pcp:**
   - Oblicz średnią znormalizowanego chromagramu wzdłuż osi czasowej, aby uzyskać profil pcp.
7. **Tworzenie struktury danych:**
   - Stwórz słownik do przechowywania danych, który zawiera mapowanie etykiet, wartości pcp oraz etykiety próbek.
8. **Przechodzenie przez katalogi:**
   - Przejdź przez wszystkie katalogi i pliki w podanej ścieżce, używając `os.walk`.
9. **Przetwarzanie plików dźwiękowych:**
   - Dla każdego pliku dźwiękowego oblicz profil pcp i dodaj go do słownika danych razem z odpowiednią etykietą.
10. **Zapisanie danych do pliku .json:**
    - Użyj `json.dump` do zapisania słownika danych w formacie .json.

Rysunek 14. Screen rozmowy ze sztuczną inteligencją ChatGPT

6.2.3. Sieć neuronowa

Zaprojektowaliśmy sieć neuronową do klasyfikacji akordów. Proces rozpoczynamy od wczytania danych z pliku JSON, który zawiera informacje o cechach (pcp) i etykietach (labels). Następnie dzielimy dane na trzy zestawy: treningowy, walidacyjny i testowy, co pozwala nam efektywnie trenować model oraz oceniać jego wydajność na nieznanych danych.

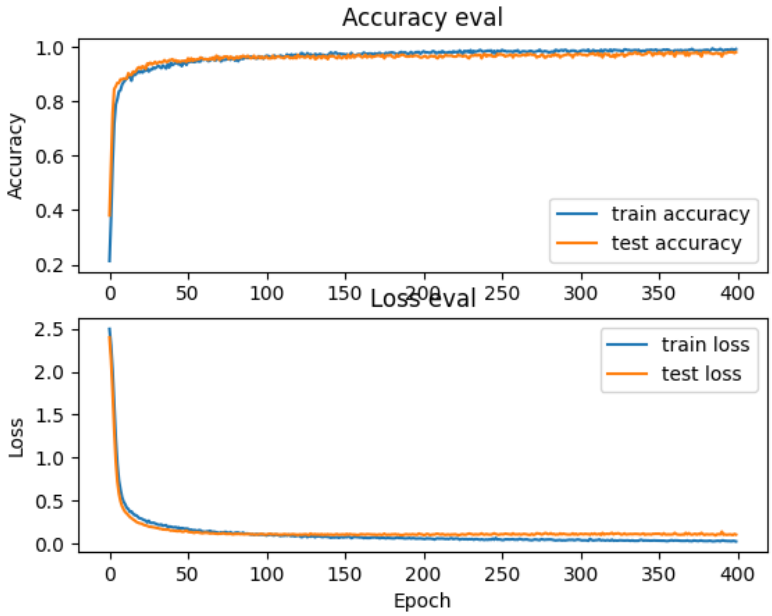
Model budujemy przy użyciu sekwencyjnej architektury Keras na podstawie dokumentacji [17]. W pierwszym kroku sieć przekształca wejściowe dane, spłaszczając je do jednowymiarowego wektora. Następnie dane przechodzą przez trzy gęste warstwy w pełni połączonych neuronów. Pierwsze dwie warstwy składają się z 512 neuronów każda i używają funkcji aktywacji ReLU, co pozwala modelowi na naukę nieliniowych zależności w danych. Po każdej z tych warstw stosujemy dropout w celu zapobiegania przeuczenia się modelu, co polega na losowym wyłączaniu pewnej liczby neuronów podczas treningu. Trzecia warstwa gęsta ma 64 neurony, również z aktywacją ReLU. Ostateczna warstwa wyjściowa składa się z 13 neuronów (ponieważ zakładamy, że mamy 13 klas) i używa funkcji aktywacji softmax, która przekształca wyniki do prawdopodobieństw przynależności do poszczególnych klas. Nasz model kompilujemy przy użyciu optymalizatora Adam z małą szybkością uczenia (0.0001), co pozwala na stopniową i stabilną aktualizację wag modelu.

Trenujemy model przez 400 epok na zbiorze treningowym, przy jednoczesnym monitorowaniu jego wydajności na zbiorze walidacyjnym. Po zakończeniu treningu wyniki wizualizujemy, przedstawiając zmiany w dokładności i stracie zarówno dla zbioru treningowego, jak i walidacyjnego w kolejnych epokach.

```
Epoch 395/400
39/39 [==============================] - 0s 5ms/step - loss: 0.0274 - accuracy: 0.9903 - val_loss: 0.1026 - val_accuracy: 0.9806
Epoch 396/400
39/39 [==============================] - 0s 5ms/step - loss: 0.0290 - accuracy: 0.9879 - val_loss: 0.1055 - val_accuracy: 0.9806
Epoch 397/400
39/39 [==============================] - 0s 5ms/step - loss: 0.0250 - accuracy: 0.9919 - val_loss: 0.1028 - val_accuracy: 0.9806
Epoch 398/400
39/39 [==============================] - 0s 5ms/step - loss: 0.0304 - accuracy: 0.9911 - val_loss: 0.0974 - val_accuracy: 0.9806
Epoch 399/400
39/39 [==============================] - 0s 5ms/step - loss: 0.0283 - accuracy: 0.9895 - val_loss: 0.1101 - val_accuracy: 0.9774
Epoch 400/400
39/39 [==============================] - 0s 5ms/step - loss: 0.0236 - accuracy: 0.9919 - val_loss: 0.1033 - val_accuracy: 0.9806
17/17 - 0s - loss: 0.1246 - accuracy: 0.9652 - 35ms/epoch - 2ms/step

Test accuracy: 0.9651837348937988
```

Rysunek 15. Widok z terminala na koniec trenowania sieci neuronowej



Rysunek 16. Wizualizacja wyników dokładności oraz strat na wykresach

Na koniec zapisujemy model w formacie HDF5, co umożliwia jego późniejsze użycie. Wydajność modelu oceniamy na zestawie testowym, a dokładność testowa jest wyświetlana, co pozwala nam ocenić, jak dobrze model radzi sobie z nowymi, nieznanymi danymi.

6.2.4. Rozpoznanie akordów w czasie rzeczywistym

Mając wszystkie niezbędne komponenty zaprojektowaliśmy wstępny program główny do testowania działania naszego programu. Rozpoczęliśmy testy przy użyciu naszej gitary elektroakustycznej. Po zagraniu po sobie trzech akordów (F, G, A) program prawidłowo je zidentyfikował i wyświetlił na ekranie co widać na rysunku poniżej.

```
Identified chord: F with confidence: 0.9998534917831421
1/1 [==============================] - 0s 15ms/step
Identified chord: F with confidence: 0.9995694756507874
1/1 [==============================] - 0s 15ms/step
1/1 [==============================] - 0s 13ms/step
Identified chord: G with confidence: 0.998322069644928
1/1 [==============================] - 0s 14ms/step
Identified chord: G with confidence: 0.9999995231628418
1/1 [==============================] - 0s 15ms/step
1/1 [==============================] - 0s 15ms/step
Identified chord: A with confidence: 0.9999995231628418
1/1 [==============================] - 0s 15ms/step
Identified chord: A with confidence: 0.9999997615814209
```

Rysunek 17. Widok z terminala podczas pracy programu do identyfikacji akordów

7. Efekt końcowy projektu

7.1. Zrealizowane zadania

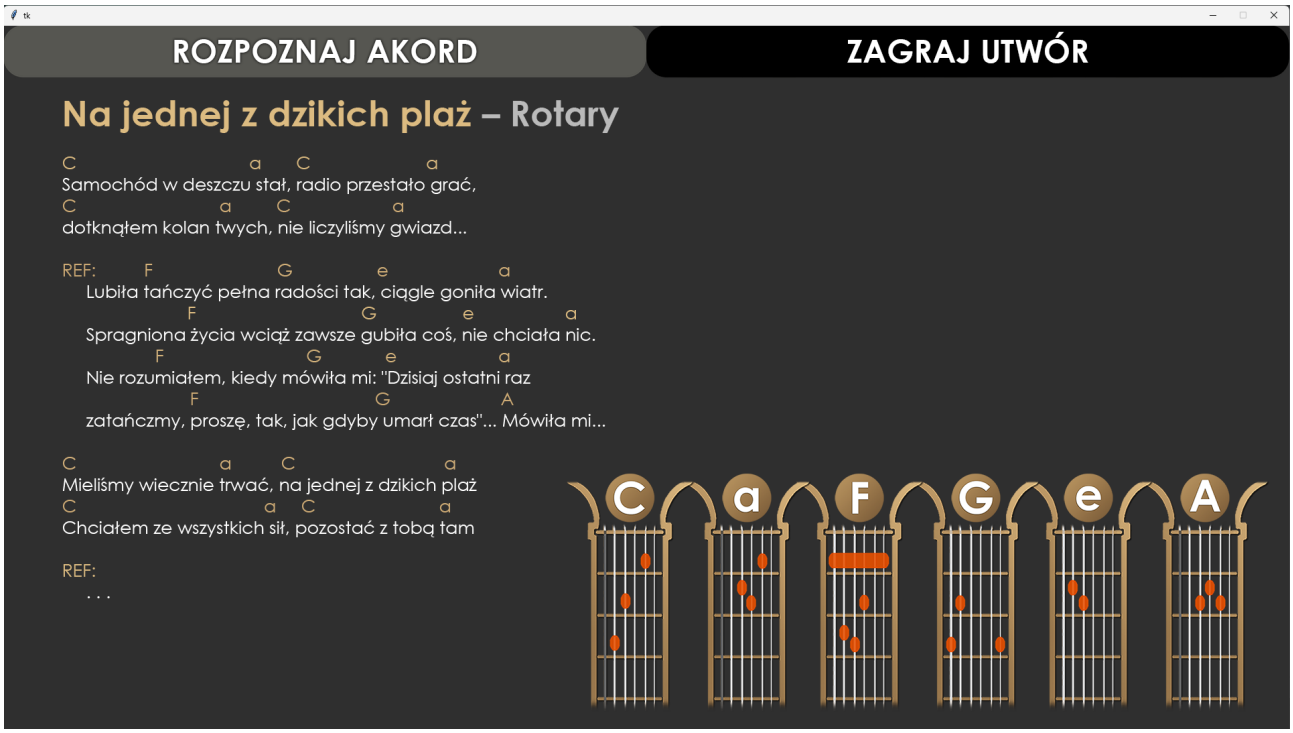
Ostatecznie udało nam się utworzyć program, który rozpoznaje akordy w czasie rzeczywistym. Posiada on interfejs graficzny, który jest bardzo intuicyjny i prosty dla użytkownika końcowego. Aplikacja podczas gry na gitarze pokazuje aktualny akord, a także wyświetla grafikę gitary wraz z ułożeniem palców na gryfie. Użytkownik może także wybrać jedną z kilku dostępnych w programie piosenek w celu sprawdzenia akordów gitarowych niezbędnych do jej zagrania. Niestety nie udało nam się zrealizować funkcjonalności pozwalających na rozpoznanie kierunku bicia gitarowego oraz określenie tempa gry. Zamiast realizacji tych funkcji, skupiliśmy się na udoskonaleniu interfejsu graficznego i dokładności rozpoznawania akordów.

7.2. Interfejs graficzny

W interfejsie umieściliśmy dwie zakładki: "Rozpoznaj akord" i "Zagraj utwór". Pierwsza z nich pokazuje akord jaki jest grany przez użytkownika oraz zawiera grafikę gitary z ułożeniem palców na poszczególnych strunach, druga zaś pozwala na wybranie utworu, wraz z tesktem oraz potrzebnymi akordami. Dzięki zastosowaniu takiego pomysłu, użytkownik może dowiedzieć się w jaki sposób grać dany akord i sprawdzić czy gra go poprawnie.

Rysunek 18. Widok strony głównej "Rozpoznaj Akord" interfejsu graficznego

Rysunek 19. Widok zakładki "Zagraj utwór" interfejsu graficznego ze spisem utworów



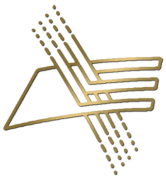
Rysunek 20. Widok zakładki "Zagraj utwór" interfejsu graficznego z przykładowym utworem

7.3. Kody programów

Wszystkie kody opracowanych programów wraz z instrukcją użytkownika znajdują się na naszym repozytorium w serwisie GitLab [2].

Literatura

[1] Trello - Tablica PROJ1
https://trello.com/invite/b/dRmpPSE4/ATTI646fa462b8879d3b2e67a1761f7df6d4932A2A58/proj1
[2] GitLab - Repozytorium
https://gitlab-stud.elka.pw.edu.pl/proj_rozpoznawanie_akordow/proj_rozpoznawanie_akordow
[3] Doodle
https://doodle.com/meeting/participate/id/erEJ1xWd
[4] Budowa gitary
https://kursgitary.pl/budowa-gitary/
[5] NumPy - Dokumentacja
https://numpy.org/doc/stable/
[6] Math - Dokumentacja
https://docs.python.org/3/library/math.html
[7] Time - Dokumentacja
https://docs.python.org/3/library/time.html
[8] PyAudio - Dokumentacja
https://people.csail.mit.edu/hubert/pyaudio/docs/#pyaudio-documentation
[9] MatPlotLib - Dokumentacja
https://python101.readthedocs.io/pl/latest/pylab/
[10] Baza akordów - Github
https://github.com/maoz-grossman/Chord-recognition?tab=readme-ov-file
[11] Mel-Frequency Cepstral Coefficients
https://medium.com/@MuhyEddin/feature-extraction-the-most-important-steps-in-developing-any-machine-learning
[12] Pitch Class Profile
https://ccrma.stanford.edu/ kglee/pubs/klee-icmc06.pdf
[13] Chroma Energy Normalized Statistics
https://www.audiolabs-erlangen.de/resources/MIR/FMP/C7/C7S2_CENS.html
[14] Spectral Contrast
https://arxiv.org/pdf/1811.01222
[15] Zero-Crossing Rate
https://speechprocessingbook.aalto.fi/Representations/Zero-crossing_rate.html
[16] Openai - ChatGPT
https://openai.com/index/chatgpt/
[17] Tensorflow - Model Keras
https://www.tensorflow.org/api_docs/python/tf/keras/Model



Wydział Elektroniki
i Technik Informacyjnych

POLITECHNIKA WARSZAWSKA